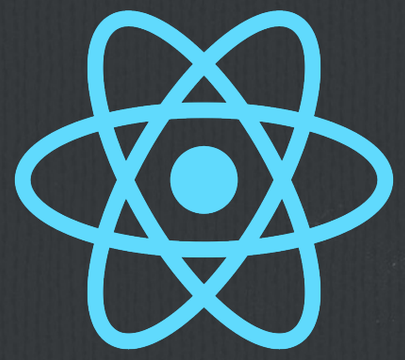


React Server Side Rendering

A Software Presentation From

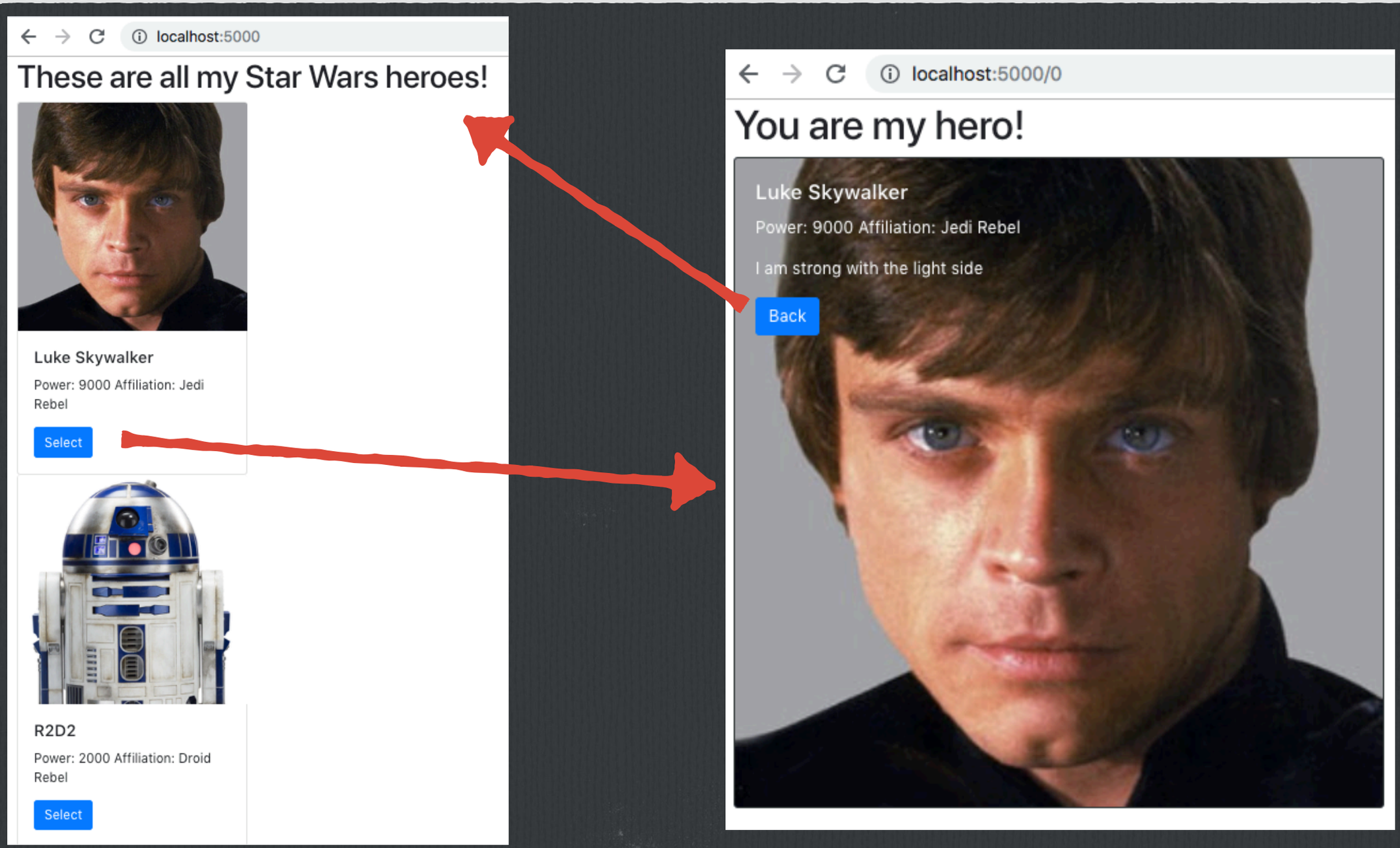
Jeremy Zerr

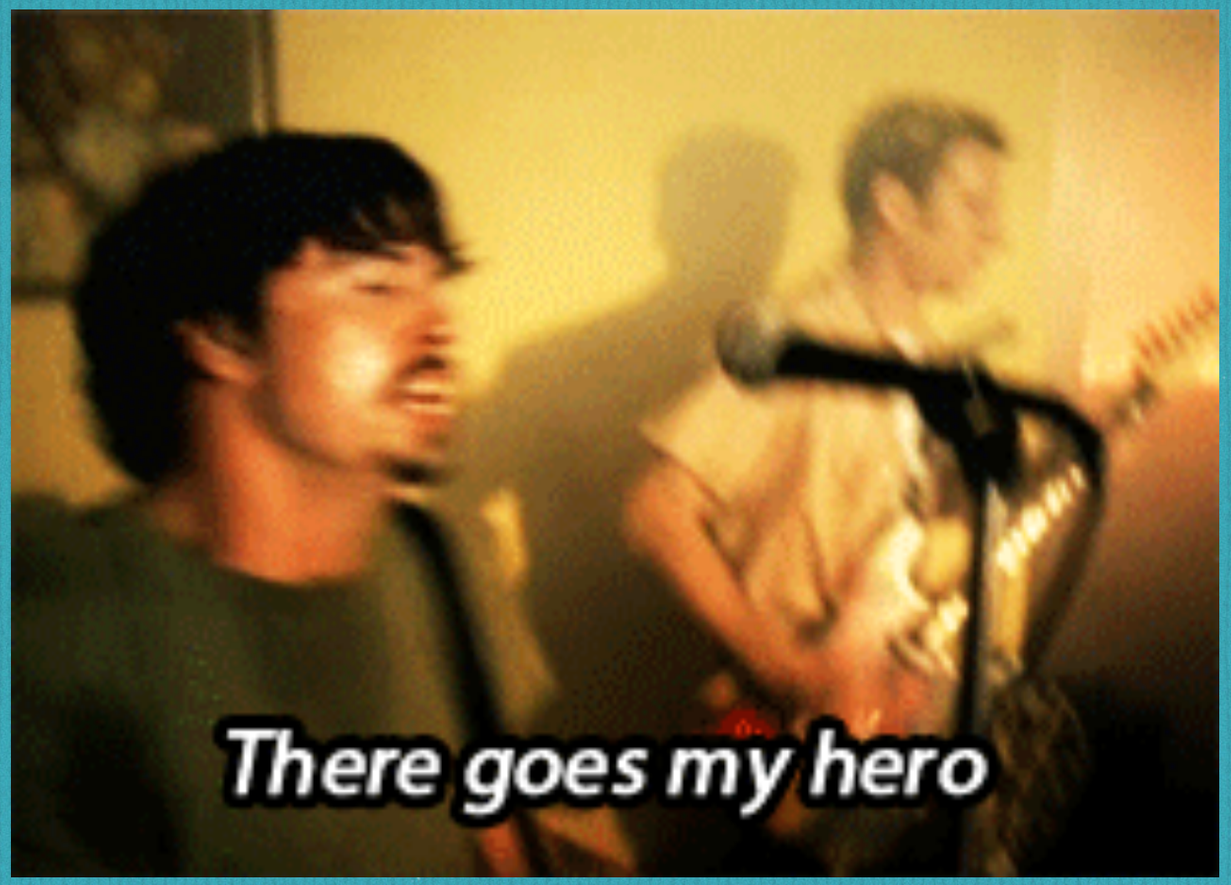
Site: https://www.zerrtech.com
https://www.linkedin.com/in/jrzerr
https://twitter.com/jrzerr

The Plan

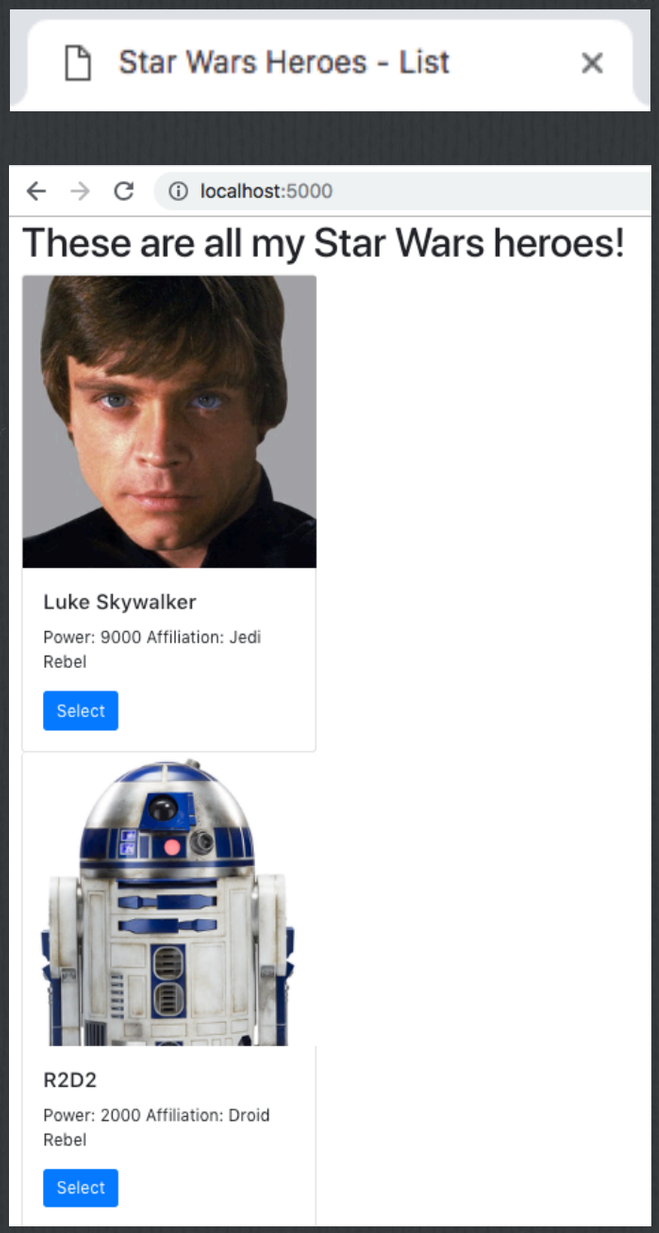
- [ ] Start with a React project started from Create React App that has some API calls, defines title tag, and a couple routes
- [ ] Show what problem we are trying to solve and how we would like to solve it
- [ ] Evolve the app into a React Server Side Rendering, showing the challenges encountered along the way

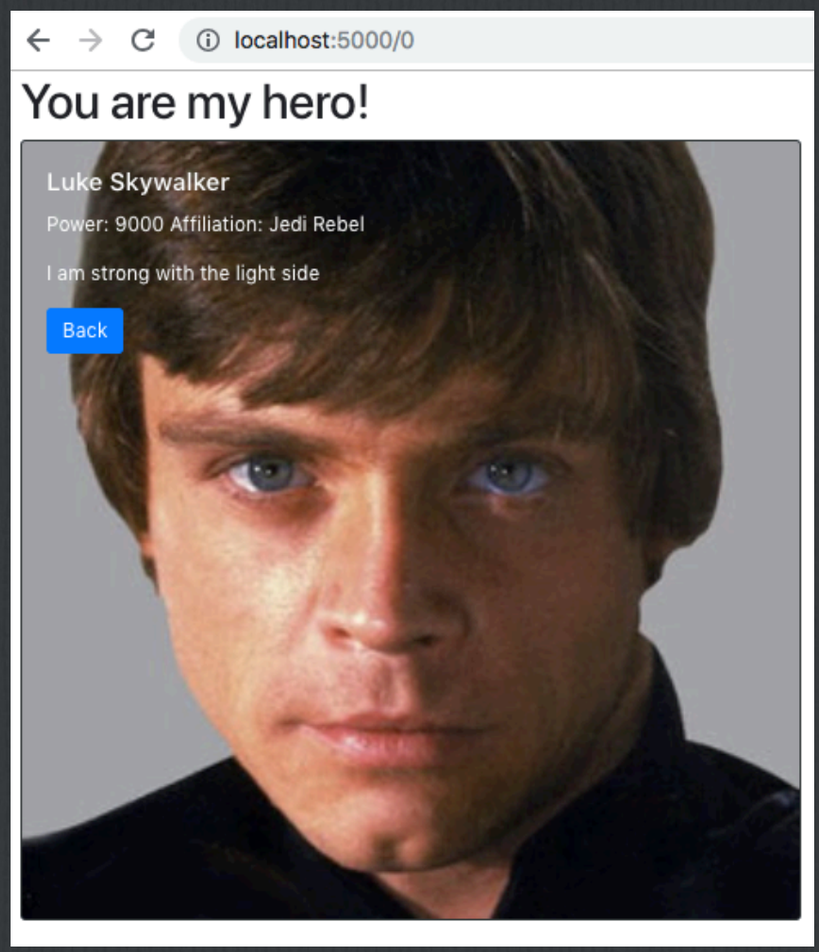
The App


localhost:5000
These are all my Star Wars heroes!
Luke Skywalker
Power: 9000 Affiliation: Jedi Rebel
Select
R2D2
Power: 2000 Affiliation: Droid Rebel
Select
localhost:5000/0
You are my hero!
Luke Skywalker
Power: 9000 Affiliation: Jedi Rebel
I am strong with the light side
Back


There goes my hero

The App - Title tags

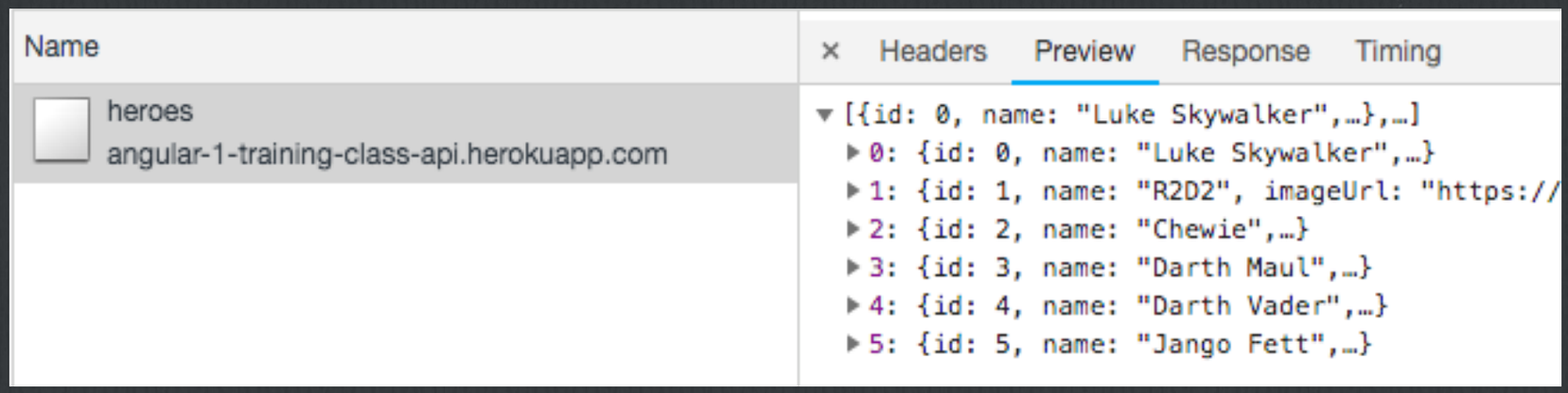




The App - API Calls

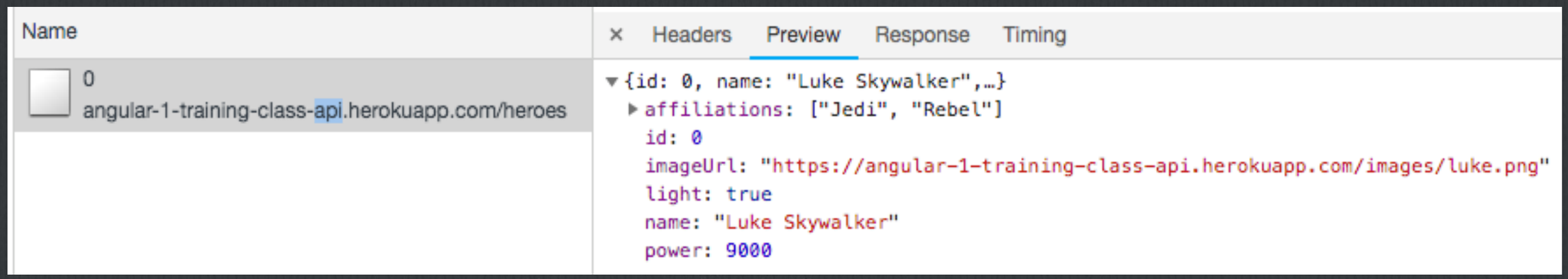
Star Wars Heroes - List

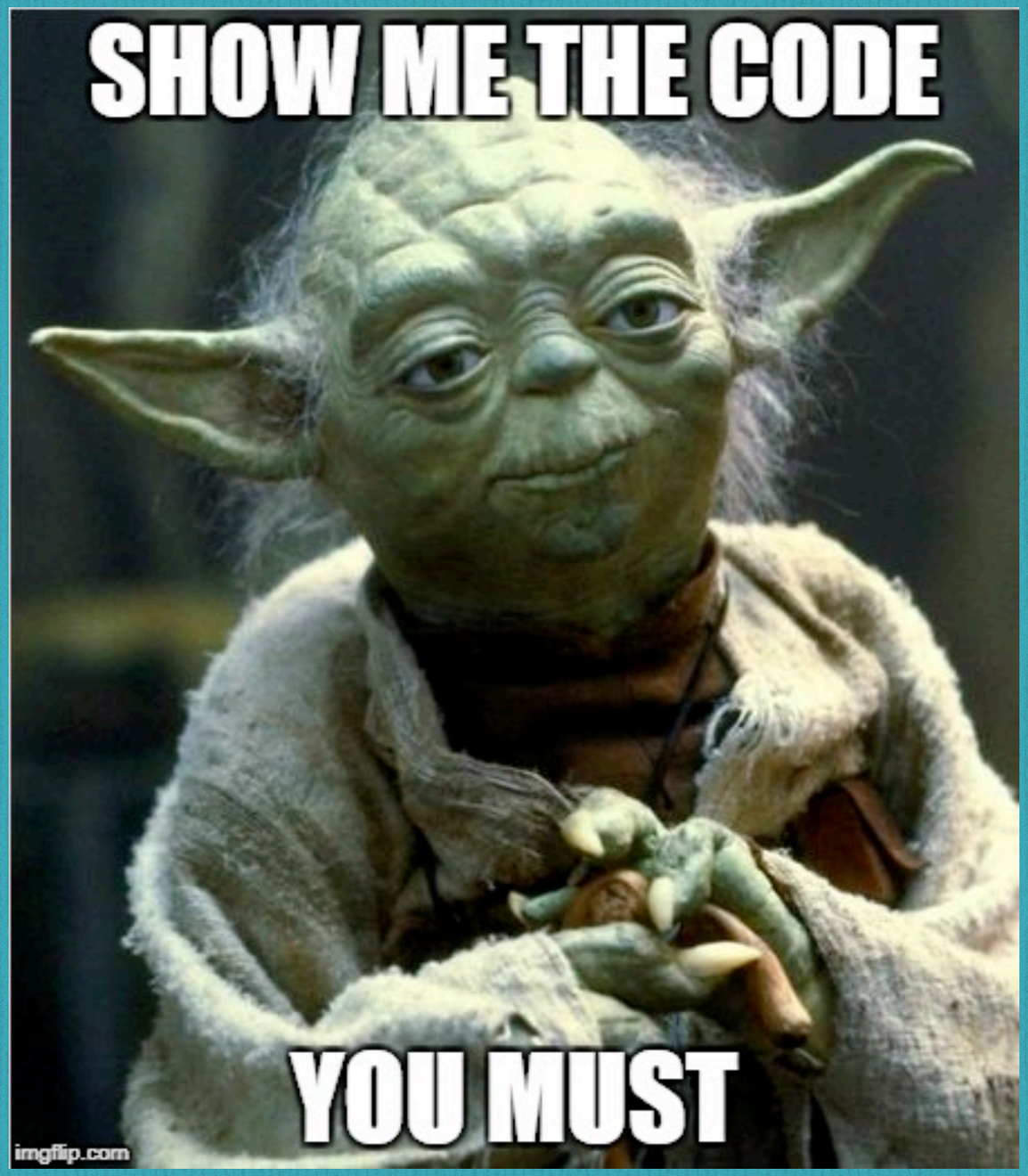
/heroes



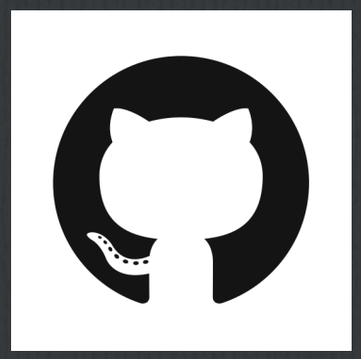
Star Wars Heroes - Luke Skyw

/heroes/0




SHOW ME THE CODE
YOU MUST
imgflip.com

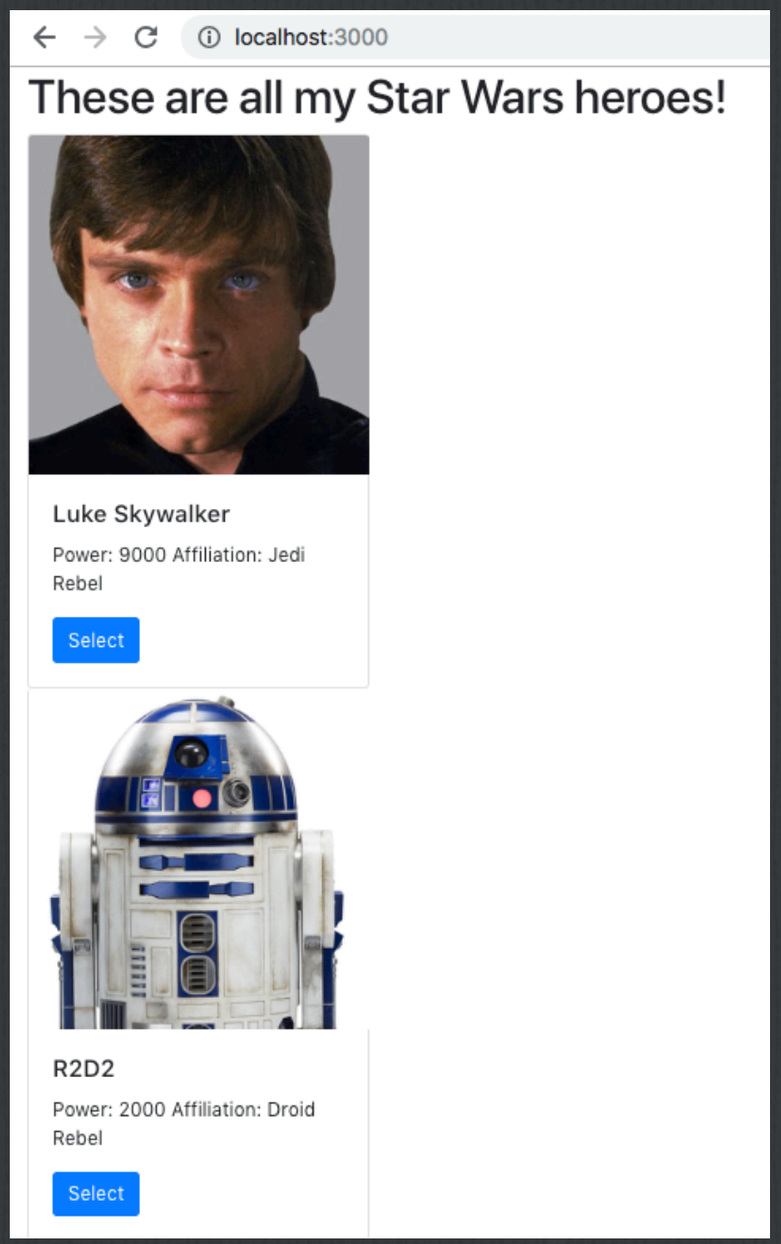
The App - Code



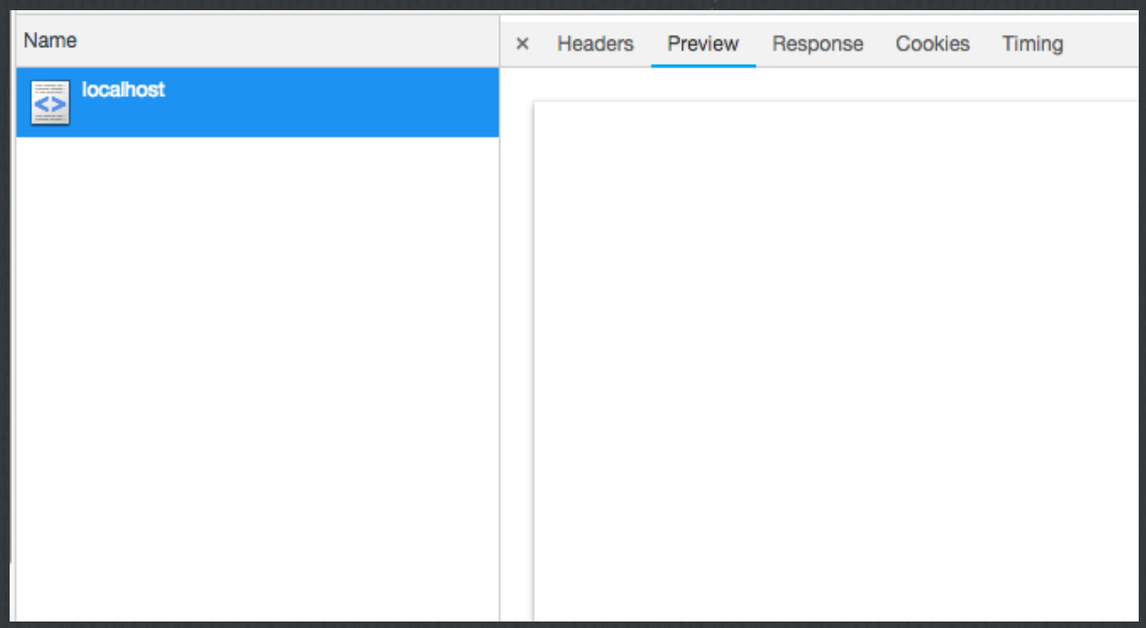
Zerrtech/zerrtech-react-ssr-demo

- [ ] 4 branches, 1 for each iteration
  - [ ] step-1-normal-app
  - [ ] step-2-ssr-initial
  - [ ] step-3-ssr-server-seed
  - [ ] step-4-ssr-api

Initial App Demo
Branch: step-1-normal-app



Initial page load has nothing!



Page fetches heroes from API then renders

Initial App Demo
Empty HTML

This empty initial page load is by design.
It's a Single Page App (SPA) after all.

```
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  <script src="/static/js/bundle.js"></script><script src="/static/js/1.ch
```

React renders into the div id="root"

What's wrong with that?

- ☐ Google can crawl Javascript on a page, but it takes extra steps and can delay full indexing since they have to wait for rendering by doing two passes.
- ☐ Performance wise, one trip to the server is quicker than N trips to the server to generate a page, especially in mobile where # parallel requests is smaller and connections are slower.
- ☐ Combined with SSR caching, this can mean that each page is only rendered once and used by many users. Loading indicators added while waiting for API calls wouldn't really show up, better user experience.

Rendering HTML only page

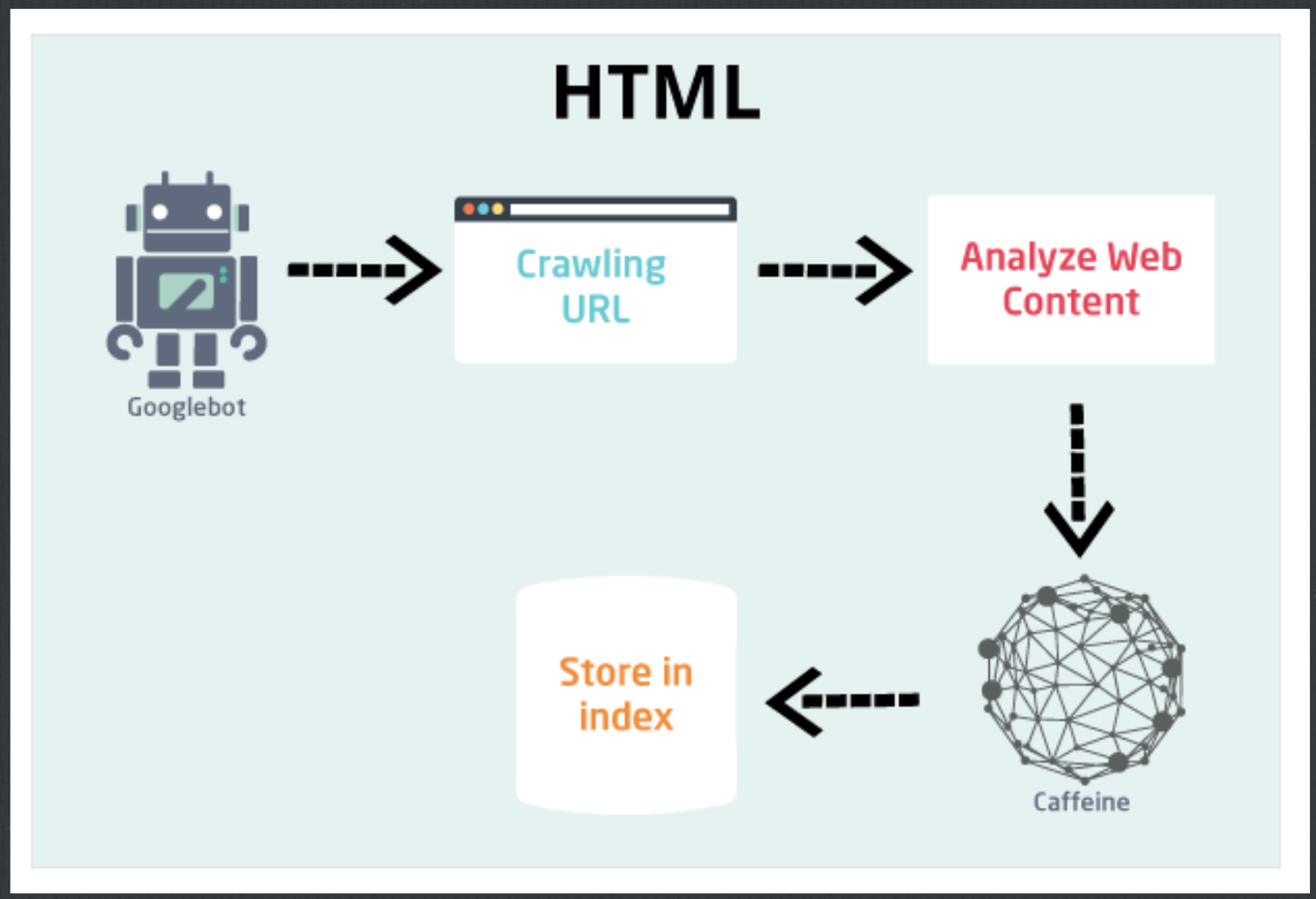


Image from SEOpressor

Rendering JS page

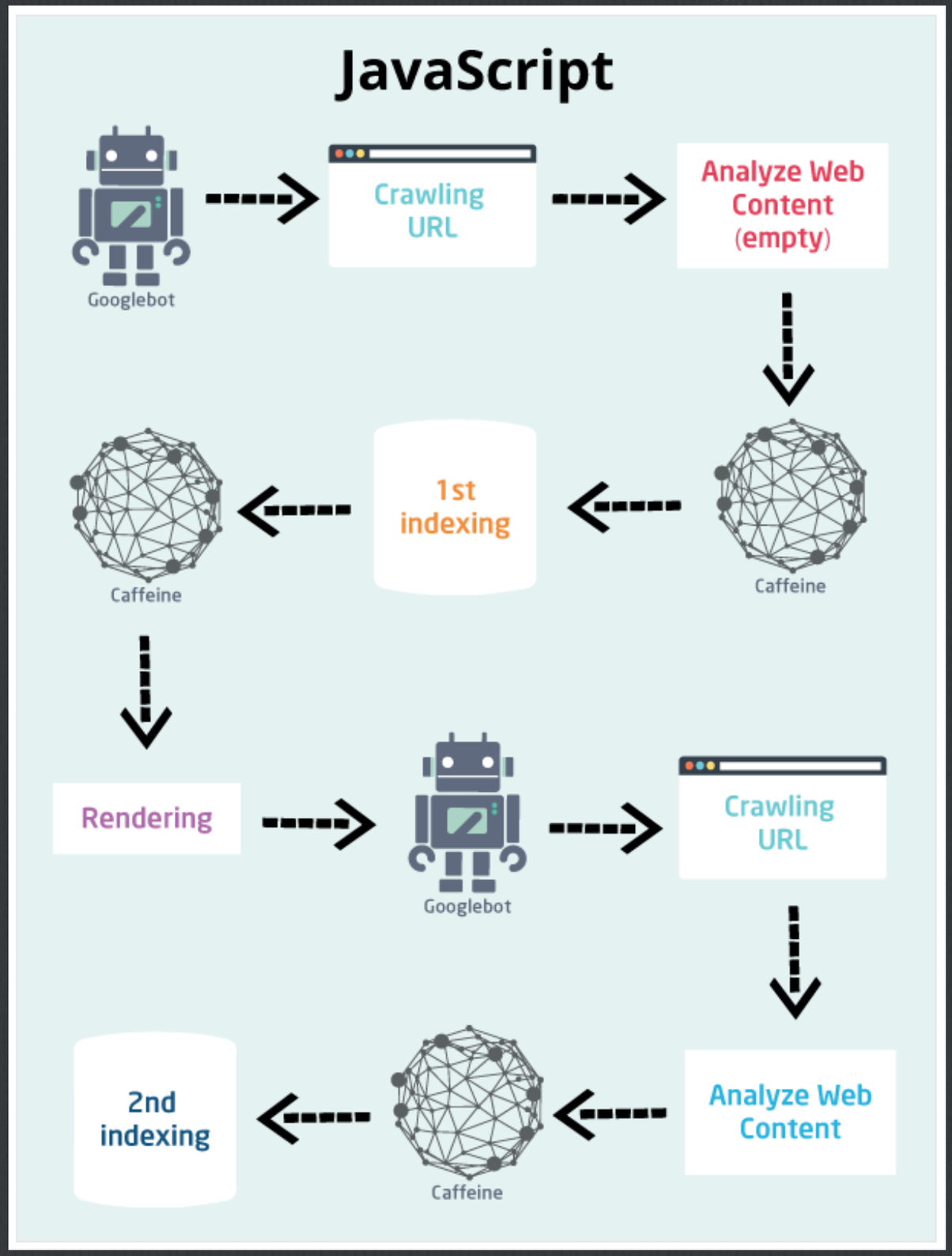


Image from SEOpressor

Google uses Chrome 41 to crawl

I have version 71

Can I Use - 41 vs. 71

Use the Google Search Console to Fetch as Google (or new Search Console URL Inspection)

Google Search Rendering Guide

Why don't all sites do SSR then?

- ☐ It's good for content-heavy, largely publicly accessible apps.
- ☐ If most of your content is behind a login, then SSR can't get at it either.
- ☐ A proper SSR implementation would use a layer of caching, so if you have data that absolutely needs to be as up to date as possible, probably not a good fit.
- ☐ Requires redesigning data fetching within components, ideal solution is specific to your app, no generic solution

Wait!!!
Weren’t all sites like this in 1999?

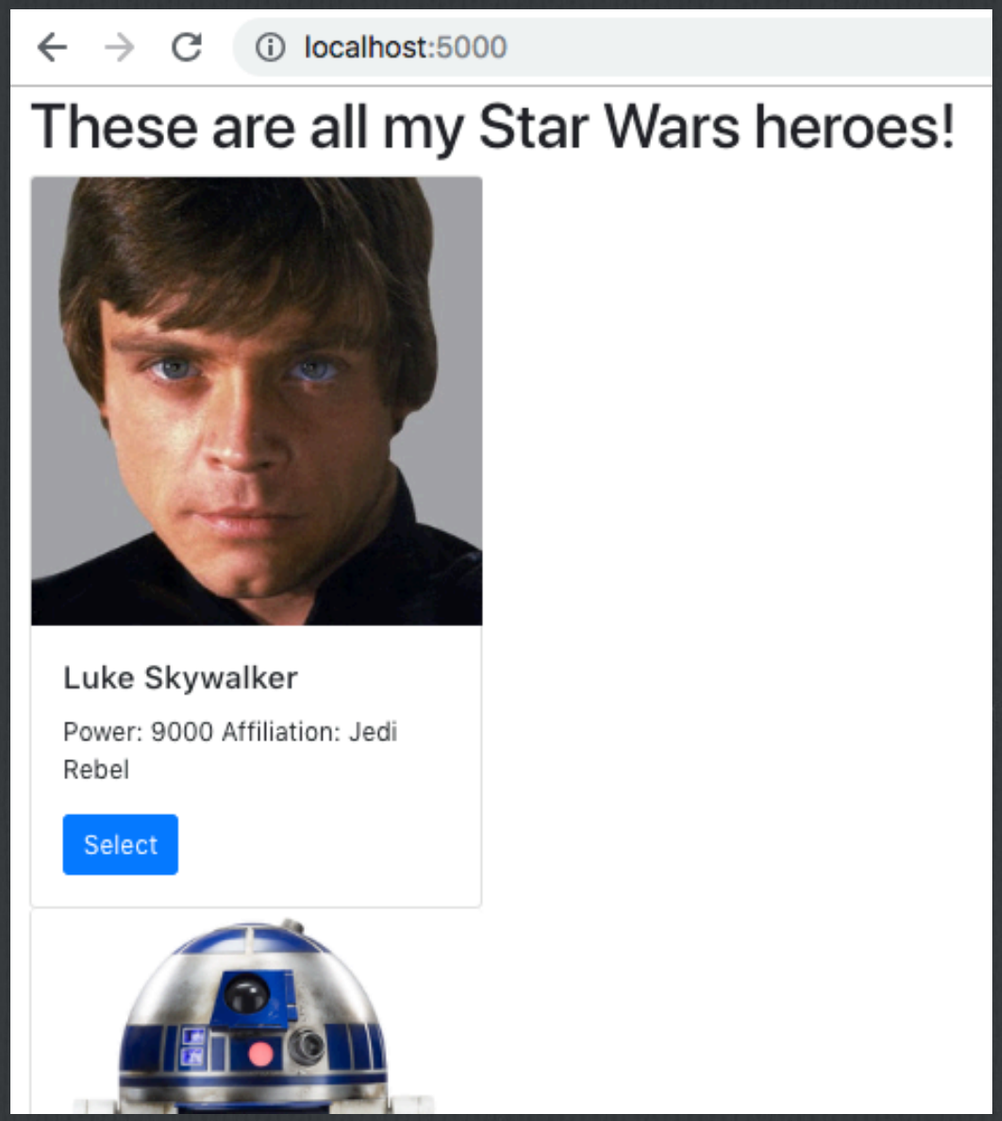
How does React rendering normally work?

- [ ] A user types your home page URL into the browser
- [ ] Browser sends a request to the server for your home page
- [ ] HTML for home page is returned to Browser, but has a blank content body and links to scripts/CSS
- [ ] React boots up in Browser, renders the home page
- [ ] Browser React component may fire off API requests that when they return, causes another React render

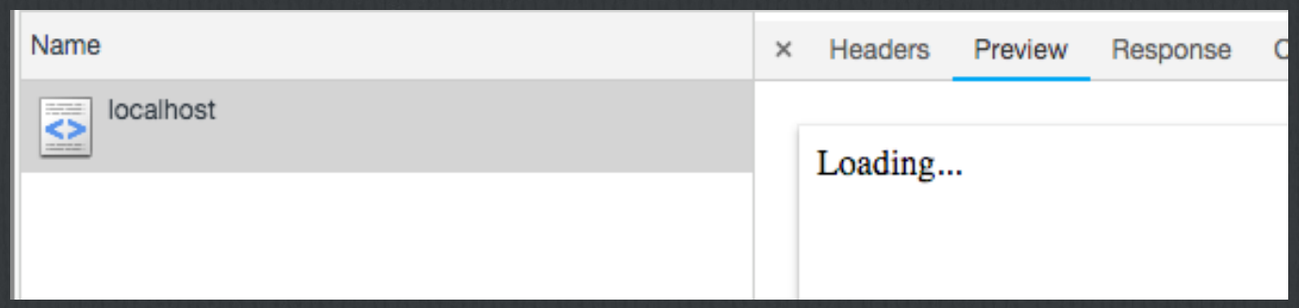
How does React SSR work?

- [ ] A user types your home page URL into the browser
- [ ] Browser sends a request to the server for your home page
- [ ] Server fetches data needed for the home page, seeds your component state, renders your React components, gets the HTML created by them, stuffs it into your React root in index.html and returns it to the browser
- [ ] React boots up in Browser, realizes everything was already rendered so has no changes (virtual DOM)
- [ ] Browser React component does not need to fire off API calls because they were already done on the server

Initial SSR
Branch: step-2-ssr-initial



Initial page load has our loading indicator



Client fetches heroes from API then renders

Initial SSR
Code changes

- ☐ Added a server script
  - ☐ Babel compiled
  - ☐ Express
  - ☐ React DOM Server renderToString and StaticRouter
- ☐ Build prod with yarn build, then start SSR with yarn start:ssr

Code diff - step 1 vs step 2

Static files served out of build dir

We load our Routes under StaticRouter instead of BrowserRouter

We grab our index.html and stuff our HTML in

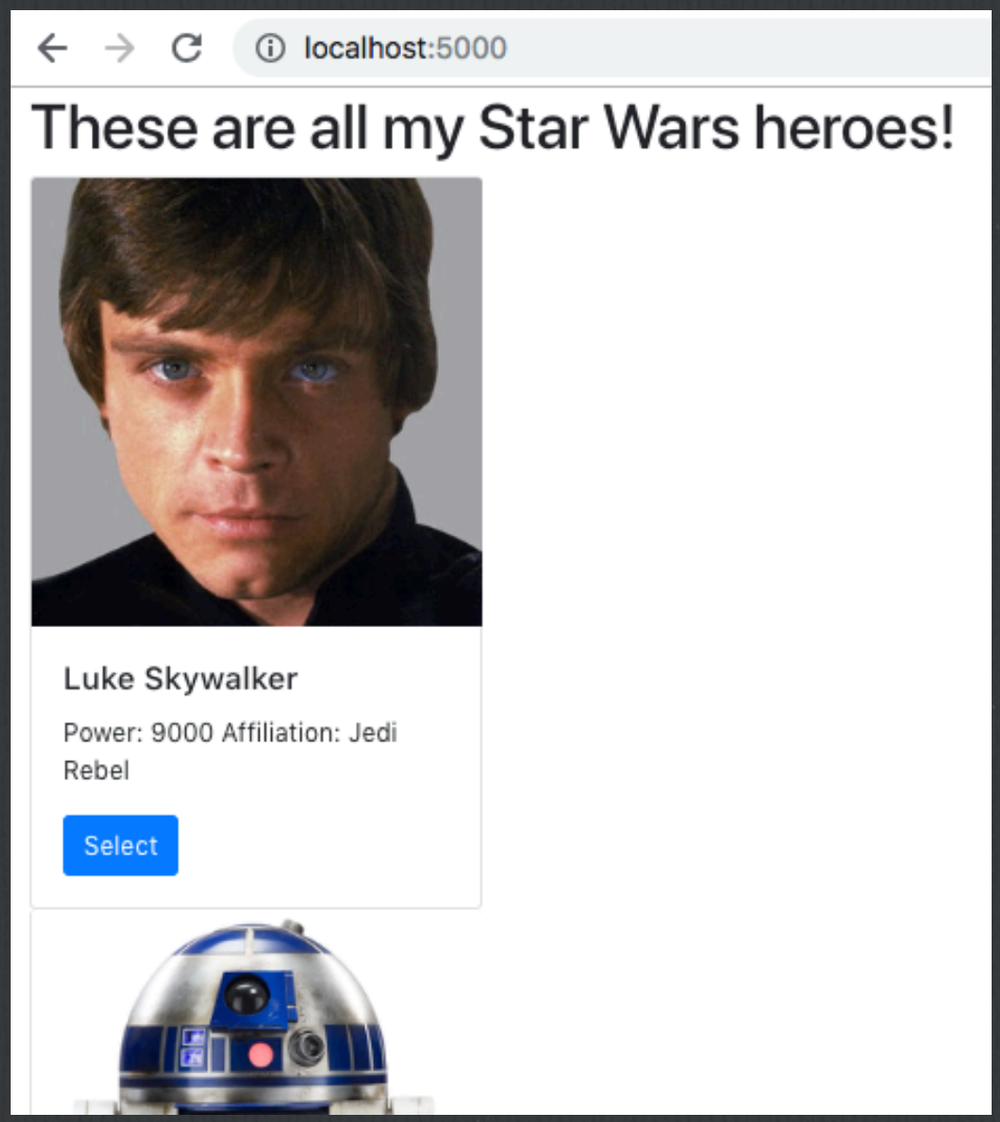
Step 2 - server.js

```
import express from "express";
import path from "path";
import fs from "fs";
import React from "react";
import { renderToString } from "react-dom/server";
import { StaticRouter } from "react-router-dom";
import Routes from "../src/Routes";
const app = express();

app.use("/static", express.static(path.resolve("./build/static")));

const distPath = path.resolve("./build");
const indexPath = `${distPath}/index.html`;

app.get("*", (req, res) => {
  const context = {};

  const ssr = (
    <StaticRouter context={context} location={req.url}>
      <Routes />
    </StaticRouter>
  );
  const dom = renderToString(ssr);

  fs.readFile(indexPath, "utf8", (err, data) => {
    if (err) {
      return res.status(500).send("Index file not found!");
    }
    res.writeHead(200, { "Content-Type": "text/html" });
    res.end(htmlTemplate(data, dom));
  });
});

app.listen(5000);

function htmlTemplate(data, dom) {
  return data.replace('<div id="root"></div>', `<div id="root">${dom}</div>`);
}
```

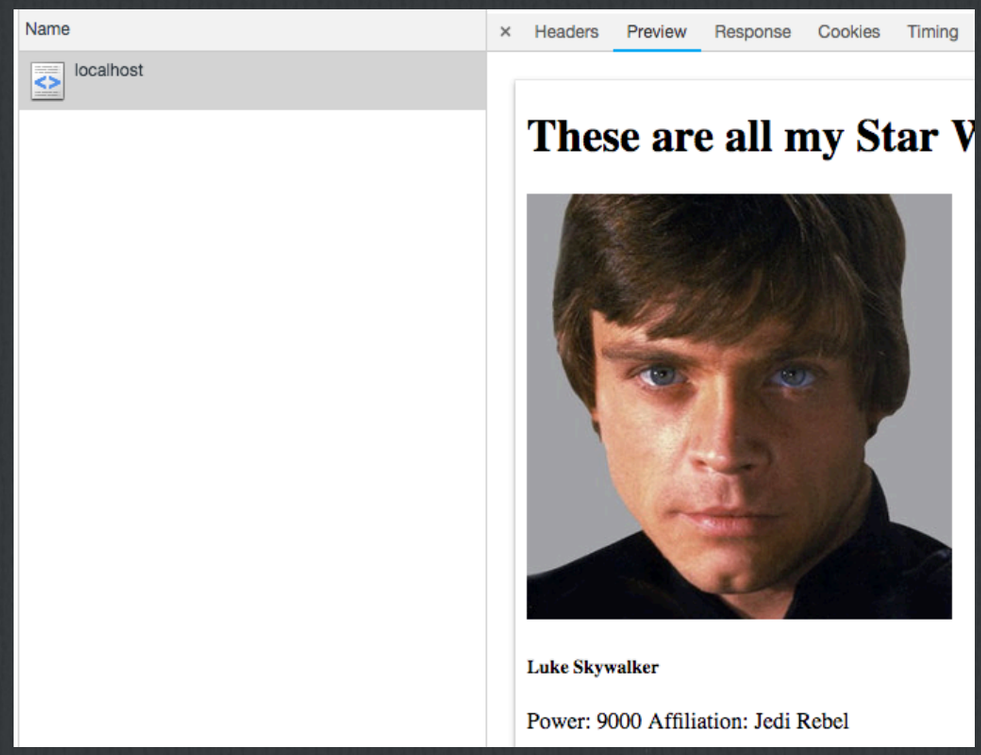
Initial SSR Review

- [ ] Notice no API calls were done within SSR
- [ ] The render is synchronous, makes one pass through then returns what it has
- [ ] Fact: when doing SSR, componentDidMount is not called
- [ ] We need to do our API calls on the server, then seed our state within our component so SSR can do the full page

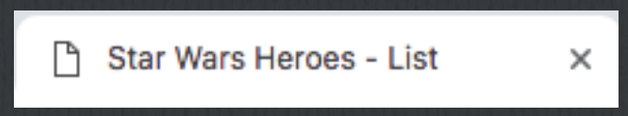
SSR Server Seed
Branch: step-3-ssr-server-seed



Data all there!



Title tag, check!



SSR Server Seed
Code changes

- [ ] Added fetching data into the server script by using a static method on each route component
- [ ] Pass the data into the component using StaticRouter context param
- [ ] Render Helmet and replace in HTML

Code diff - step 2 vs step 3

SSR Server Seed Data fetching

- Static method called getInitialState() added into Route component. Static so it can be called on the server easily.

```
20    static async getInitialState(matchParams) {
21      const heroes = await HeroList.getHeroes();
22      return {
23        heroes,
24        loading: false
25      };
26    }
```

- Server looks for a matching route and a getInitialState method, if so calls it

```
26    // find the route that matches
27    // then fire off any getInitialState function if it exists
28    const dataPromises = routeList
29      .map(route => {
30        const match = matchPath(req.url, route);
31        if (match && route.component.getInitialState) {
32          return route.component.getInitialState(match.params);
33        } else {
34          return null;
35        }
36      })
```

SSR Server Seed Data fetching

- We wait for API calls to be done then add data to context passed into StaticRouter
- Back in the Route component, we look for that data and merge into state in the constructor.
- staticContext is provided by React router withRouter()

```
39    // wait for all data to be fetched before we do ssr
40    Promise.all(dataPromises).then(initialStateList => {
41      context.initialState = initialStateList[0];
42      // we pass the data result into SSR on the StaticRouter context
43      const ssr = (
44        <StaticRouter context={context} location={req.url}>
45          <Routes />
46        </StaticRouter>
47      );
```

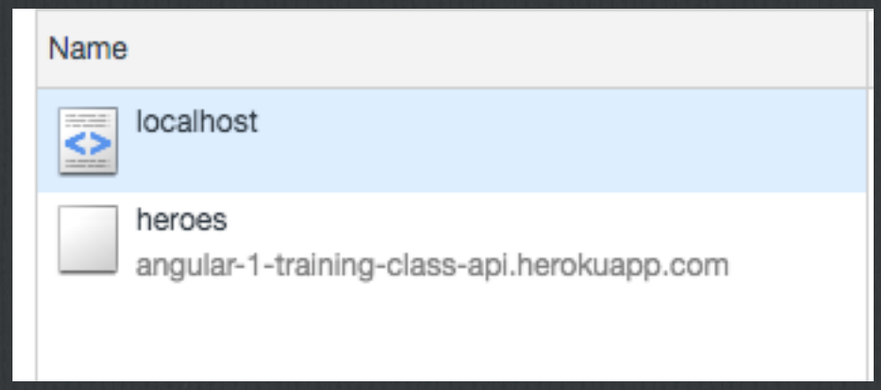
```
12      if (this.props.staticContext && this.props.staticContext.initialState) {
13        this.state = Object.assign(
14          this.state,
15          this.props.staticContext.initialState
16        );
17      }
```

```
72  export default withRouter(HeroList);
```

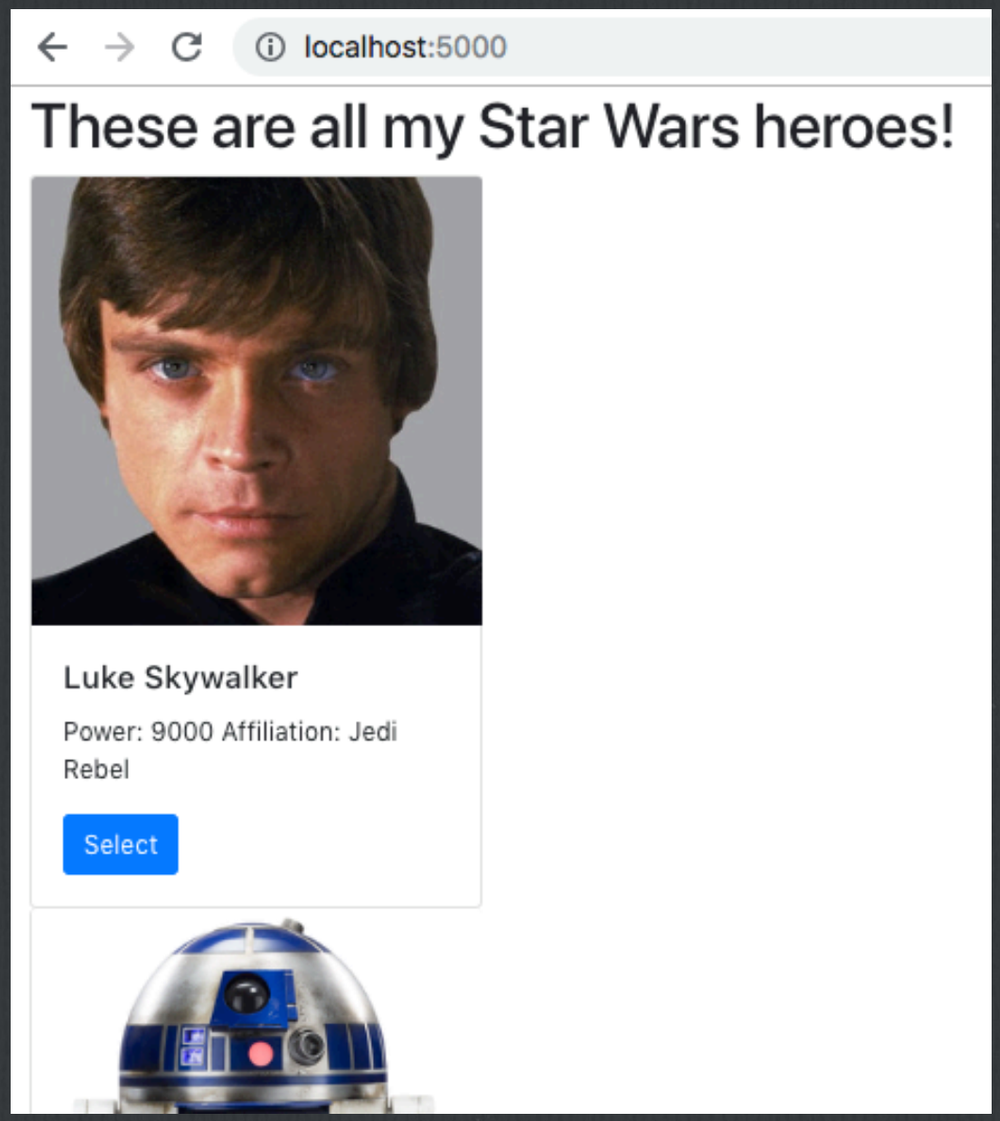
SSR Server Seed Review

- [ ] OK great, but now we are wasting our API call on the frontend, how do we prevent that?

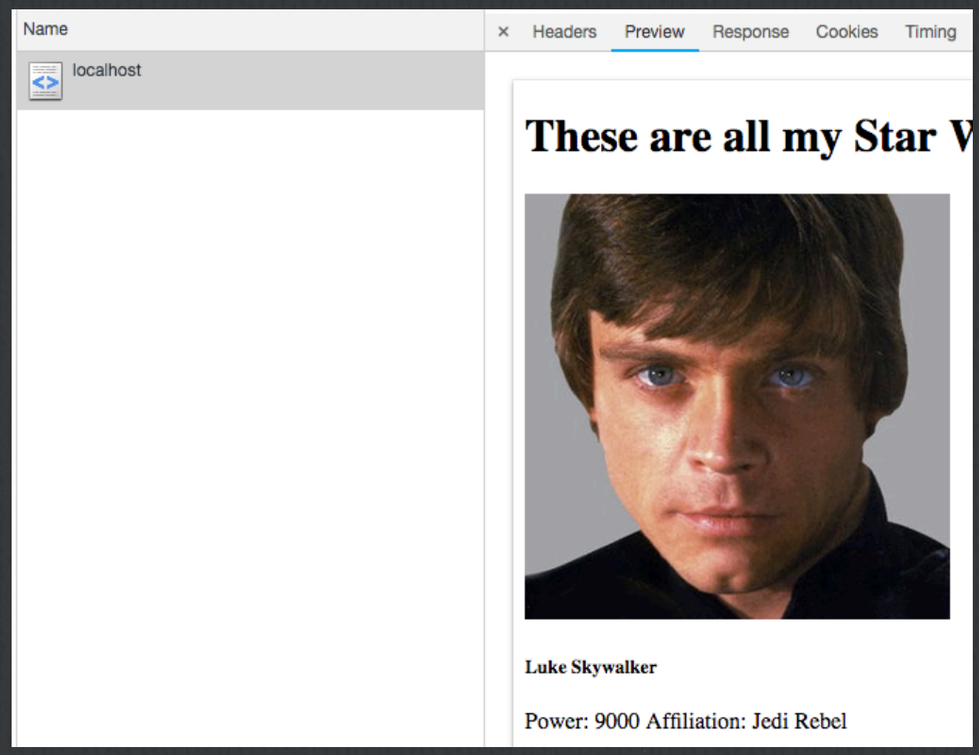
- [ ] We'll put that same initialState in our index.html so on the client side, we can also seed that data and avoid the API call



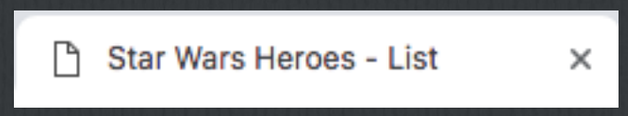
SSR API
Branch: step-4-api



Data all there! No extra API call



Title tag, check!



SSR API
Code changes

- [ ] Add a placeholder in index.html we can stuff our initial state into
- [ ] Check for this existing data in our static getInitialState() method
- [ ] Delete the data after we use it so other pages don't use it

Code diff - step 3 vs step 4

SSR API
API call saving

- [ ] Add a variable in index.html
- [ ] Server will stuff initialState into that var
- [ ] HeroList getInitialState checks for existing data

```
+    <script type="text/javascript">
+    window.__INITIAL_STATE__ = "__INITIAL_STATE__";
+    </script>
```

```
function htmlTemplate(data, dom, helmet, initialState) {
  return data
    .replace('<div id="root"></div>', `<div id="root">${dom}</div>`)
    .replace(/<title>.*?<\/title>/g, helmet.title.toString())
    .replace('"__INITIAL_STATE__"', JSON.stringify(initialState));
}
```

```
  static async getInitialState(matchParams) {
+   if (
+     typeof window !== "undefined" &&
+     window.__INITIAL_STATE__ &&
+     window.__INITIAL_STATE__ !== "__INITIAL_STATE__"
+   ) {
+     const initialState = window.__INITIAL_STATE__;
+     // need to delete the initial state so that subsequent
+     // navigations do not think they were server side rendered
+     // and pull up the wrong initial state
+     delete window.__INITIAL_STATE__;
+     return initialState;
+   }
    const hero = await HeroItem.getHero(matchParams.id);
```

React SSR Summary

- [ ] Demo of React SSR from create react app without ejecting webpack config
- [ ] Showed how to use SSR on an app that has async API calls and dynamic title tags
- [ ] Implemented where SSR is used and we also eliminate the client side API calls for max efficiency

As good as 1999? Definitely!

Next Steps

- [ ] Great opportunity to build a generic SSR React component for Route components to inherit from
  - [ ] Encapsulates all that window and staticContext stuff to make it easy on devs.
- [ ] Put caching in front of SSR so that you aren't running SSR on the page every time
- [ ] React Suspense API has high hopes to make it easier to identify those async API calls and make SSR easier

Thanks! Connect with us!
We would love to build your next app

A Software Presentation From

Jeremy Zerr

Site: https://www.zerrtech.com
https://www.linkedin.com/in/jrzerr
https://twitter.com/jrzerr