Server Side Rendering

What are we going to cover?

Why Server Side Rendering?

Server Side Rendering with React and Next.js

Prerendering static content

Server Side Rendering

Why Server Side Rendering?

There are two main reasons:

- Search Engine Optimization
- Perceived performance when loading the application

Server Side Rendering and SEO

Most React applications serve up an almost **empty index.html** page

- They construct the UI only on the client using JavaScript

Most search engine **spiders don’t execute JavaScript** and will only see an almost empty page

- The Google spiders is the one exception here and executes some JavaScript

The default index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="theme-color" content="#000000">
    <link rel="manifest" href="/manifest.json">
    <link rel="shortcut icon" href="/favicon.ico">
    <title>React App</title>
    <link href="/static/css/main.c17080f1.css" rel="stylesheet">
</head>
<body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <script type="text/javascript“
            src="/static/js/main.96ccb336.js"></script>
</body>
</html>
```

Perceived performance

As the initial page is **almost empty** there is nothing for the browser to display

- Only when the JavaScript has executed is there a visible user interface

Loading and executing takes some time

- Specially on a mobile device with a slow network connections

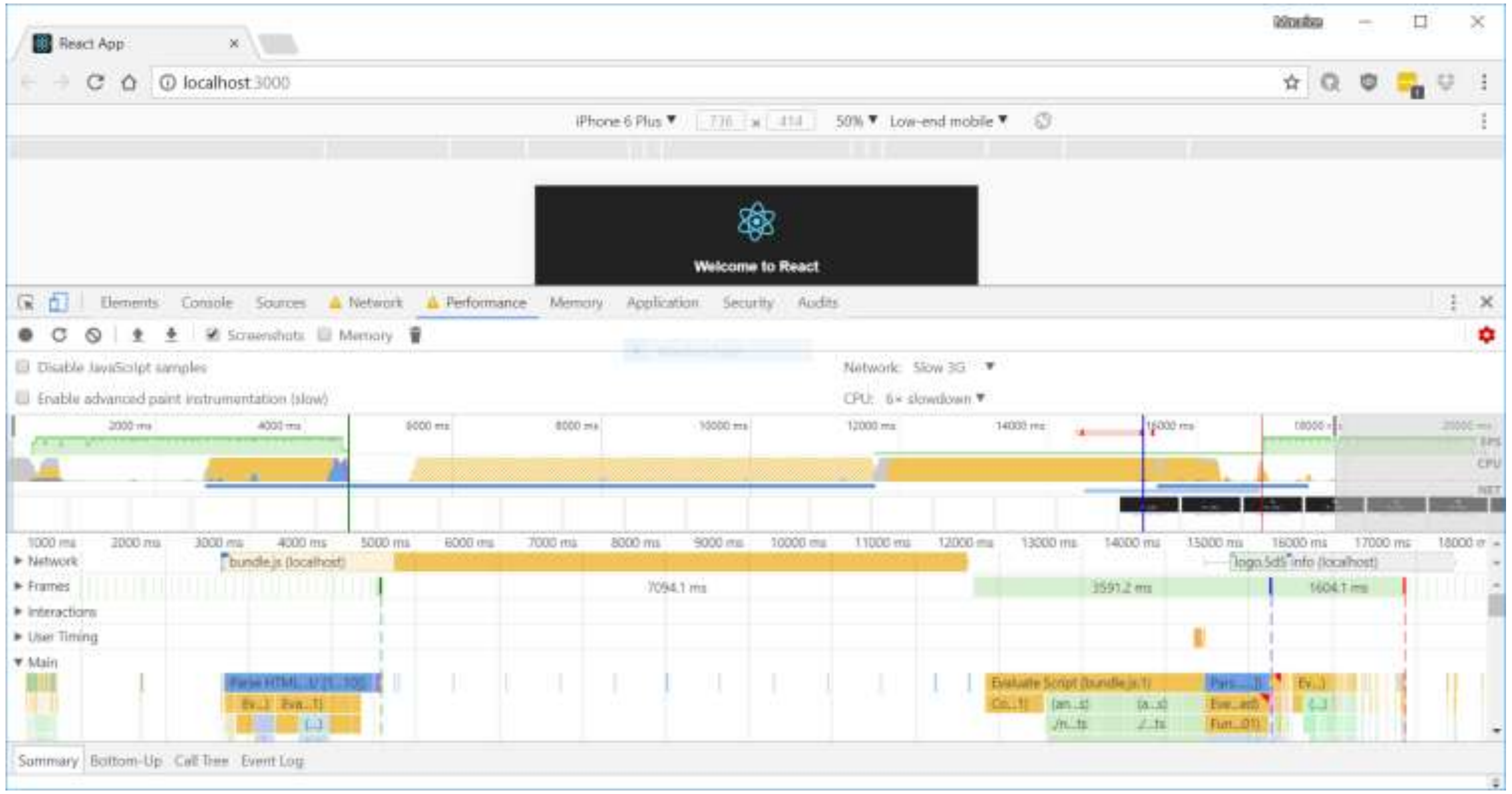
Service workers can help with the lack of network speed

- But are not supported on all browsers
- They are not active until the second time a user loads the application

Low-end device

Index.html loaded in 2.5 sec

Visible after 15 sec



Server Side Rendering

With server side rendering the React application is **rendered on the server**

- The browser receives complete markup
- This normally includes the complete page including any data that would be loaded asynchronously

The application UI is **visible** as soon as the initial page renders

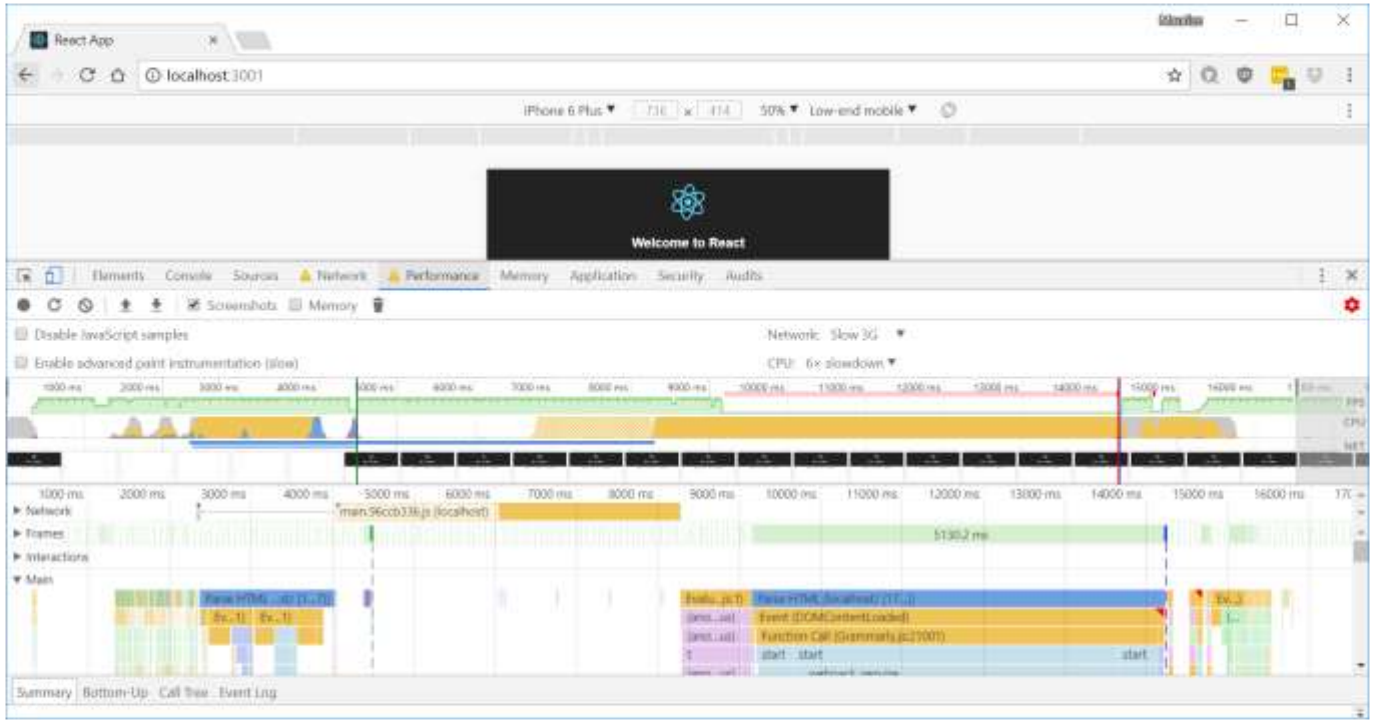
- But not fully functional until the JavaScript executes

Low-end device

Index.html loaded in 2.5 sec

Visible after 5 sec

Functional after 15 sec



iPhone 6 Plus ▾ 736 x 414 50% ▾ Low-end mobile ▾

☐ Disable JavaScript samples

☐ Enable advanced paint instrumentation (slow)

Network: Slow 3G ▾

CPU: 6x slowdown ▾

Network: main.96ccb336.js (localhost)

Frames: 5130.2 ms

Main:

Parse HTML (localhost) [17...]

Event (DOMContentLoaded)

Function Call (Grammarly.js:21001)

start start start

Requirements

The application needs to be rendered on the server using **ReactDOMServer**

- Either **renderToString()** or **renderToNodeStream()**

Activate the SSR application on the client using **ReactDOM.hydrate()**

- Instead of the usual ReactDOM.render()

Rehydrate a SSR application

```
const rootElement = document.getElementById("root");
if (rootElement.childElementCount) {
  ReactDOM.hydrate(application, rootElement);
} else {
  ReactDOM.render(application, rootElement);
}
```

Webpack

Create a **server specific bundle** for the application

- The bundle created for on the client is not suitable for server side rendering

No need to bundle up everything into a single bundle

- Imports can be resolved quickly using CommonJS by Node

When using **Create-React-App** you can start with the default **webpack.config**

- Part of the **react-scripts** package

SSR Webpack.config

```
const config =
  require("react-scripts/config/webpack.config")
    ("production");


config.entry = "./src/index.ssr.js";


config.output.filename = "static/ssr/[name].js";
config.output.libraryTarget = "commonjs2";


config.target = "node";
config.externals = /^[a-z\-0-9]+$/;


module.exports = config;
```

Next.js

THE REACT FRAMEWORK FOR SERVER SIDE RENDERING

What is Next.js?

Next.js is a **lightweight framework** for static and **server-rendered applications**

- Using React as the UI library
- And Node on the server

Uses Webpack, Babel under the hood

There is no official project generator like **create-react-app**

- Not really needed as the defaults are very good
- There is a community driven **create-next-app**

Next.js application structure

Routing in Next.js is based on the **file system**

- Components in the **/pages** folder become routes

There is no **index.html** file on disk as this is generated by Next.js

- The content can be modified using a **_document.js** or **_app.js** on the **/pages** folder

Static files are served from the **/static** folder

- Exposed as **/static/**

Use the optional **next.config.js** to configure the application

- Add support for LESS, TypeScript etc.
- Many plugins available from Zeit or the community

Initial data population

The static **getInitialProps()** function can be use to load data

- Executed only on the server for the **initial page load**
- Executed on the client for any SPA navigation

The getInitialProps() is called with a **parameter** containing the **URL**, **query** and more

- Some parameters are only available on the server

Note: **fetch()** is not available on Node.js

- Use an NPM package like **isomorphic-fetch**

Initial data population

```
import { Component } from 'react';
import fetch from 'isomorphic-fetch';
const url = 'http://api.icndb.com/jokes/random/10/?escape=javascript’;

class Jokes extends Component {
  static async getInitialProps() {
    const rsp = await fetch(url);
    const data = await rsp.json();
    return { jokes: data.value };
  }
  render() {
    return (
      <ul>{this.props.jokes.map(joke => (<li>{joke.joke}</li>))}</ul>
    );
  }
}
export default Jokes;
```

Prerendering

Why prerendering

Rendering only the **static parts** of the page gives the same **perceived performance**
- But is a lot simpler
- No special runtime requirements

Server-side rendering is a much more **complex** solution
- Requires a NodeJS process on the server

Prerendering static content

An alternative to server side rendering is **prerendering the static content**
- This is done at build time and updates the index.html file

No need to have Node.JS on the server as **only static files** are needed
- Altogether a much simpler approach

Different tools to pre-render

There are **different tools** to use with prerendering a React application

- The README.md of a Create-React-App based application list two good alternatives
  - react-snapshot
  - react-snap

This is normally executed at **build time** by a developer or CI server

- With only a minimal change to the application

With react-snap you can **detect** a prerender using the UserAgent

- Can be required as the all lifecycle functions execute
- `const reactSnap = navigator.userAgent === "ReactSnap";`

Using react-snap package.json

```
"scripts": {
"start": "react-scripts start",
"build": "react-scripts build && react-snap",
"test": "react-scripts test --env=jsdom",
"eject": "react-scripts eject"
},
```

Conclusion

Search engine optimization can be an issue with React applications

- The initial page loaded by the search bot is often almost empty

Server Side Rendering can help with SEO

- The page also appears to load faster

Next.js is a great Server Side Rendering framework

- Based on Node and React

Often prerendering is a good alternative

- Only the static content but it executes much faster
- No runtime dependency on Node.js