

SEO IN REACT

3 ESSENTIAL TOOLS

TO BOOST YOUR APP'S SEO

PLUS, ISOMORPHIC APPS

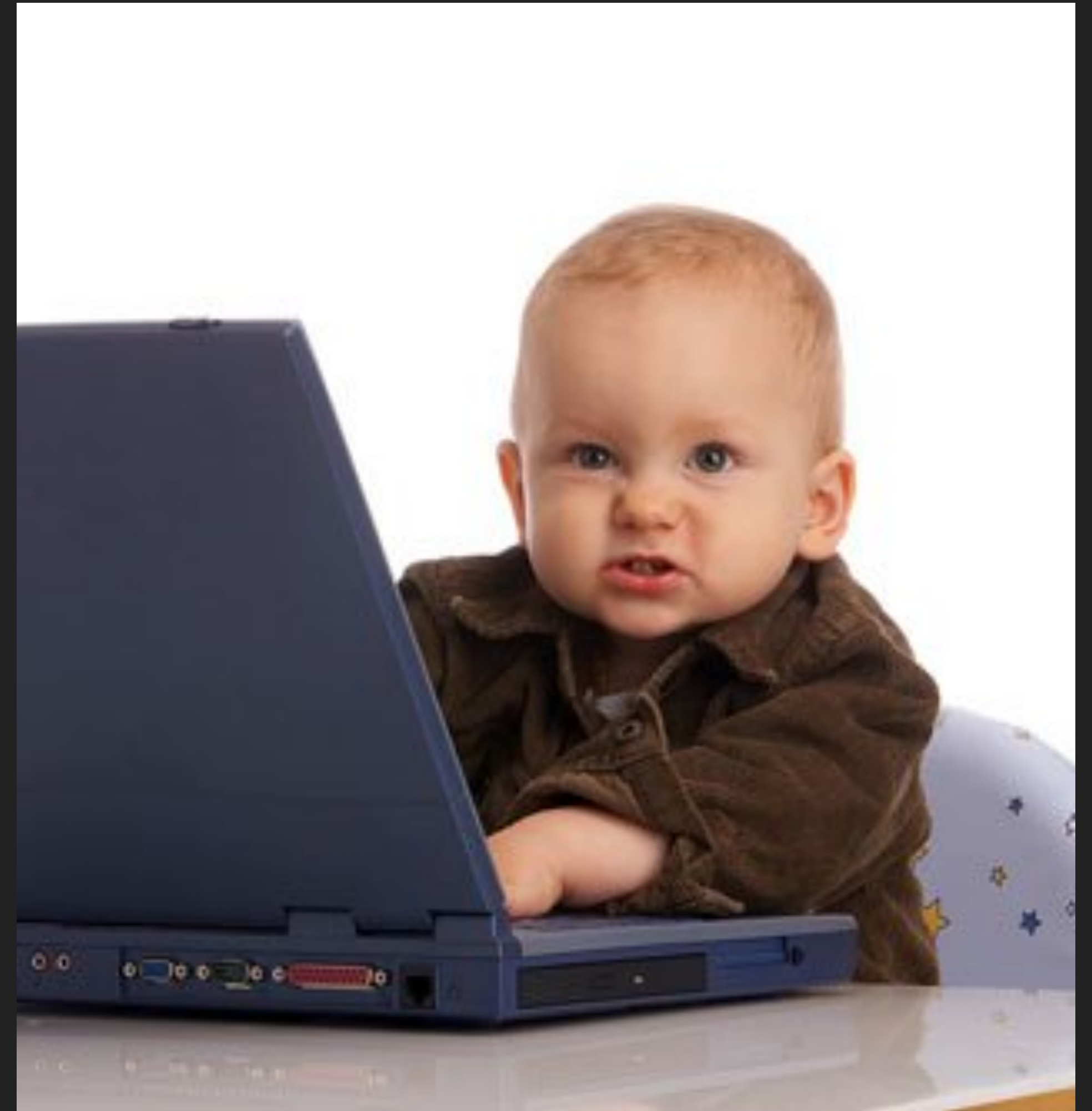
PRESTON WALLACE

**“GOOGLE RESULTS
AREN'T IMPORTANT”**

No One, ever.

SEO STATS

- ▶ 95% of web traffic goes to sites on Page 1 of Google. (brafton.com)
- ▶ Over 65% of all web traffic from a Search Engine Results Page (SERP) go to the top 5 ranked listings (study by Advanced Web Rankings)
- ▶ Literally everyone uses google >>





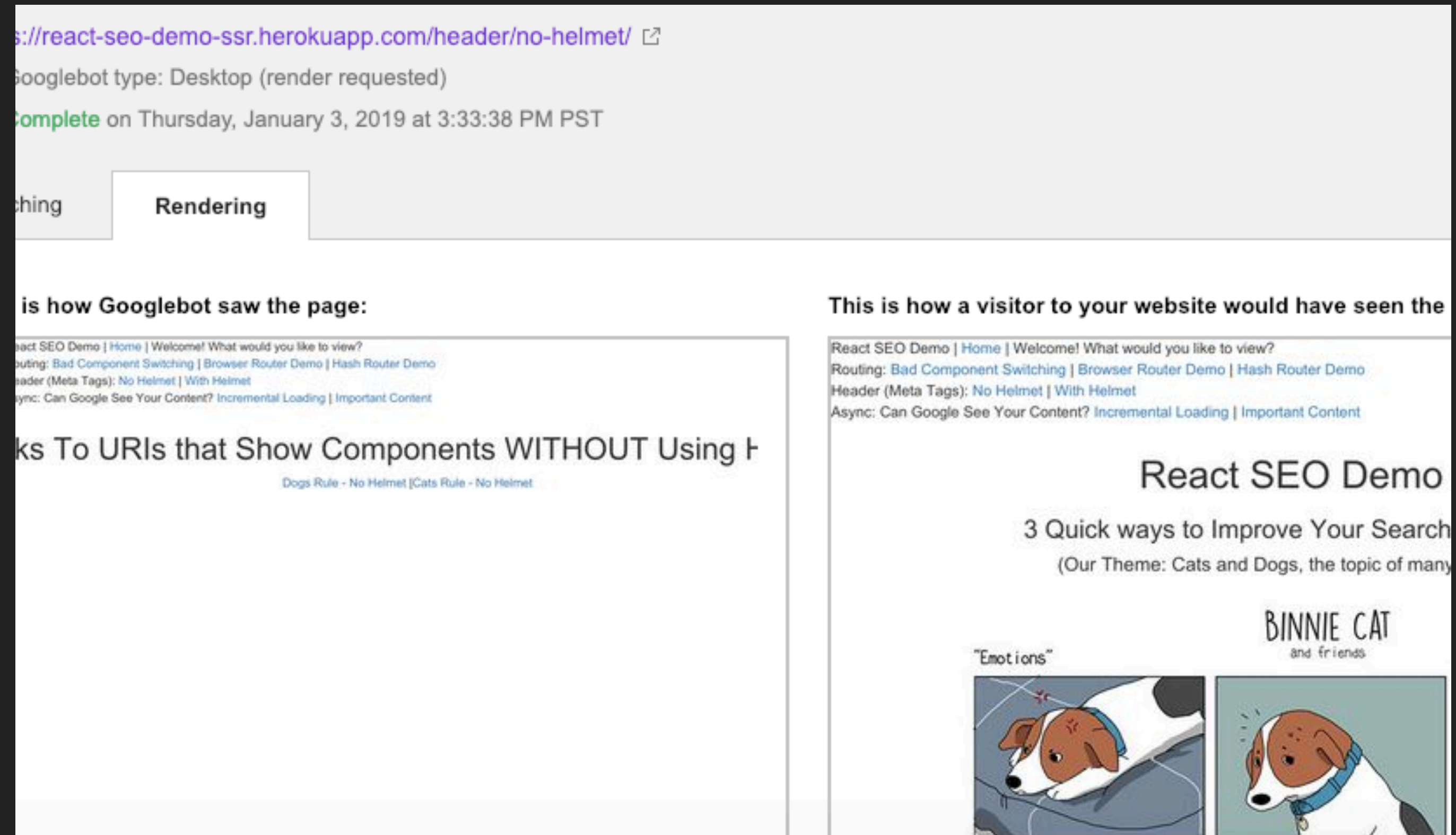
SO HOW DO I GET ON THE 1ST PAGE OF GOOGLE?

1. Content, Optimized for your Keywords
2. Domain Name and Age
3. Page Speed and Mobile Friendliness
4. Inbound Links
- 5. An Accessible URL**
- 6. Technical SEO (Title, Meta Tags, Image Alt Tags)**
- 7. Content Viewable By Search Engines (i.e. Google can crawl your pages)**

WE'LL FOCUS ON THE LAST 3

3 REACT CHALLENGES IN SEO

- ▶ One URL to Rule them all...
- ▶ Meta Tags
- ▶ JS Loads Differently by Google's bot (or it doesn't load at all)



THE REAL PROBLEM

THIS IS WHAT WE GET BACK WHEN REQUESTING '/'
(OR VIRTUALLY ANY OTHER OF OUR URLS FOR THAT MATTER)

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <!-- Responsive design? Check. This tag will make mobile browsers sc
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7     <title>React SEO Demo App: Preston Wallace's App</title>
8     <script defer src='/bundle.js'></script>
9   </head>
10  <body>
11    <div id='app'></div>
12  </body>
13 </html>
```

3 REACT CHALLENGES IN SEO... ...AND HOW WE ADDRESS THEM

1. One URL to Rule them all...

2. No Access to Meta Tags

3. JS Loads Differently by
Google's bot (or it doesn't
load at all

1. React Router

2. React Helmet

3. Fetch as Google

THE BEST WAY TO EXPLAIN

A CODE EXAMPLE!

CONSIDER THIS CODE:

Deployed Page: <https://react-seo-demo-app.herokuapp.com/>

GitHub Repo: <https://github.com/wallacepreston/react-seo-demo>

TIP #1: CREATE INDEXABLE URLS

**“YOUR SINGLE-PAGE-APPLICATION
SHOULDN'T BE A
SINGLE-URL-APPLICATION”**

Max Prin, Head of Technical SEO

TIP #1: CREATE INDEXABLE URLs

USE REACT ROUTER TO SWITCH VIEWS

- ▶ Deployed Page: <https://react-seo-demo-app.herokuapp.com/routing/browser-router>
- ▶ DON'T just switch views based on a variable in state, (i.e. set by clicking tabs).
- ▶ DO use REACT ROUTER (BrowserRouter), creating an indexable URL with each view.
- ▶ DON'T use HashRouter, since Google may not index a URL with the hash (#)

```
1  import React, { Component } from 'react';
2  import { Switch, Route, Link } from 'react-router-dom'
3  import Dogs from './dogs'
4  import Cats from './cats'
5
6  class BrowserRouterLinks extends Component {
7    render() {
8      return (
9        <div>
10         <Link to="/routing/browser-router/dogs/">View Dogs</Link>
11         <Link to="/routing/browser-router/cats/">View Cats</Link>
12         <Switch>
13           <Route path="/routing/browser-router/dogs/" component={Dogs} />
14           <Route path="/routing/browser-router/cats/" component={Cats} />
15         </Switch>
16       </div>
17     );
18   }
19 }
20
21 export default BrowserRouterLinks;
```

TIP #1: CREATE INDEXABLE URLs

Don't forget to wrap the

USE REACT ROUTER TO SWITCH VIEWS

component in BrowserRouter!

- ▶ Deployed Page: <https://react-seo-demo-app.netlify.com/routing/browser-router/>

- ▶ DON'T just switch on a variable in clicking tabs).

- ▶ DO use REACT (BrowserRouter indexable URL

- ▶ DON'T use Hash Google may not with the hash (e

```
1 import React, { Component } from 'react';

import React, { Component } from 'react';
import {BrowserRouter} from 'react-router-dom'
import BrowserRouterLinks from './browser-router-links'

class UsingBrowserRouter extends Component {
  render() {
    return (
      <div>
        <h1>Browser Router</h1>
        <BrowserRouter>
          <BrowserRouterLinks />
        </BrowserRouter>
      </div>
    );
  }
}
```

er-dom'

```
s/">View Dogs</Link> |
s/">View Cats</Link>
```

```
r/dogs/' component={Dogs} />
r/cats/' component={Cats} />
```


TIP #2: WATCH YOUR HEAD!

HEADER TAGS WITH REACT-HELMET

- ▶ Deployed, NO header edited: <https://react-seo-demo-app.herokuapp.com/header/no-helmet/>

React SEO Demo App: Preston Wallace's App

<https://react-seo-demo-app.herokuapp.com/header/no-helmet/dogs-rule>

Links To URIs that Show Components WITHOUT Using Helmet Dogs Rule - No Helmet
|Cats Rule - No Helmet Dogs Rule Dog Though Translator Dogs Will Take Over ...

React SEO Demo App: Preston Wallace's App

<https://react-seo-demo-app.herokuapp.com/header/no-helmet/cats-rule>

Links To URIs that Show Components WITHOUT Using Helmet Dogs Rule - No Helmet
|Cats Rule - No Helmet Cats Rule Sinister Cats Everyone knows cats are ...

- ▶ Deployed WITH HELMET: <https://react-seo-demo-app.herokuapp.com/header/with-helmet/>

Dogs Can Use Tech Too!

<https://react-seo-demo-app.herokuapp.com/header/with-helmet/dogs-rule>

Dogs, with the aid of this technological device, can achieve human communication

Cats Are Very Sinister

<https://react-seo-demo-app.herokuapp.com/header/with-helmet/cats-rule>

Cats will truly take over the world. And of course, they show no mercy... Watch out, Dogs!

GOOGLE SERP (SEARCH ENGINE RESULTS PAGE) LISTINGS

TIP #2: WATCH YOUR HEAD!

HEADER TAGS WITH REACT-HELMET

```
1  import React from 'react';
2  import {Helmet} from 'react-helmet'
3  import DogsRuleNoHelmet from './dogs-rule-no-helmet';
4
5  const DogsRuleWithHelmet = () => (
6    <div>
7      <Helmet>
8        <title>Dogs Can Use Tech Too!</title>
9        <meta name="description" content="Dogs, with the aid of this technological device, can
10         achieve human communication" />
11      </Helmet>
12      <DogsRuleNoHelmet />
13    </div>
14  )
15  export default DogsRuleWithHelmet;
```


TIP #3: FETCH AS GOOGLE

WHAT HAPPENS IF GOOGLE WON'T FETCH?

- ▶ Check out the 'Fetch as Google' tool: <https://www.google.com/webmasters/tools/googlebot-fetch>
- ▶ Tip: Render SEO-Relevant Data First. If it takes too long to load, Google won't see it/index it.



Fetch as Google

<https://btcnova.com/about> 

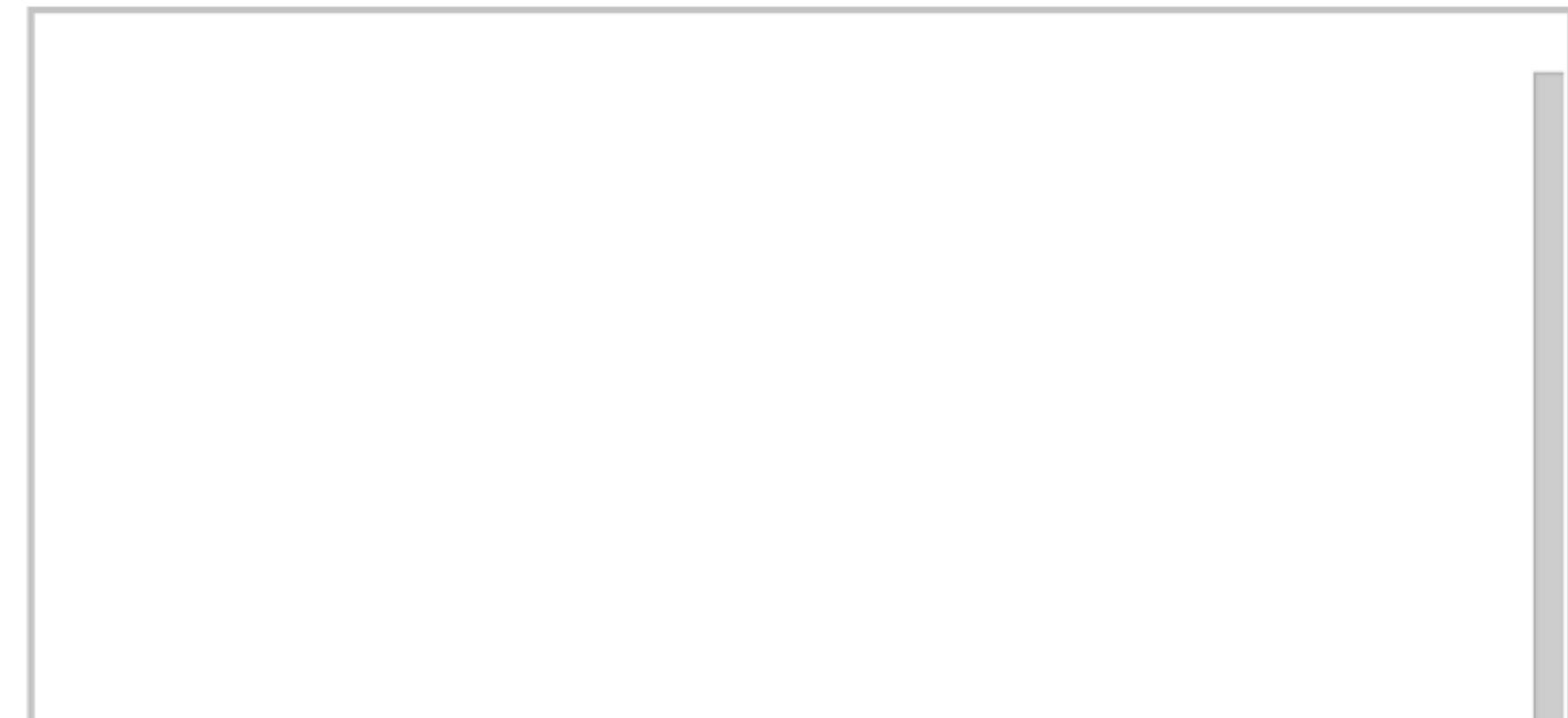
Googlebot type: Desktop (render requested)

✔ **Complete** on Tuesday, May 22, 2018 at 6:55:20 PM PDT

Fetching

Rendering

This is how Googlebot saw the page:



**YOU DON'T
HIDE FROM**

**WANT TO
GOOGLE!**



SSR: THE FINAL FRONTIER

(UNTIL WE FIND ANOTHER FRONTIER)

ISOMORPHIC APPS

CONSIDER THIS CODE:

Deployed Page: <https://react-seo-demo-ssr.herokuapp.com/>

GitHub Repo: <https://github.com/wallacepreston/react-seo-demo-ssr>

REVIEW: THE REAL PROBLEM

THIS IS WHAT WE GET BACK WHEN REQUESTING '/'
(OR VIRTUALLY ANY OTHER OF OUR URLS)

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <!-- Responsive design? Check. This tag will make mobile browsers sc
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7     <title>React SEO Demo App: Preston Wallace's App</title>
8     <script defer src='/bundle.js'></script>
9   </head>
10  <body>
11    <div id='app'></div>
12  </body>
13 </html>
```

<https://react-seo-demo-ssr.herokuapp.com/header/no-helmet/> 

Googlebot type: Desktop (render requested)

✔ **Complete** on Thursday, January 3, 2019 at 3:16:29 PM PST

Fetching

Rendering

This is how Googlebot saw the page:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charsets=
5     <!-- Respon
6     <meta name=
7     <title>Reac
8     <script def
9   </head>
10  <body>
11    <div id='ap
12  </body>
13 </html>
```

```
bile browsers sc
nitial-scale=1">
tle>
```


WHAT SSR ACHIEVES

THIS IS WHAT WE GET BACK WHEN REQUESTING '/'

```
1 <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="utf-8">
5       <title> Server Rendered Page </title>
6       <link rel="stylesheet" href="/index.css">
7     </head>
8     <body>
9       <div class="content">
10        <div id="app" class="wrap-inner">
11          <!-- magic happens here --> <div class="App"><nav><div class="container">React SEO Demo | <a
href="/">Home</a> | Welcome! What would you like to view?<p>Routing: <a href="/routing/bad-component-switching">Bad Component
Switching</a> | <a href="/routing/browser-router">Browser Router Demo</a> | <a href="/routing/hash-router">Hash Router Demo</a>
<br/>Header (Meta Tags): <a href="/header/no-helmet/">No Helmet</a> | <a href="/header/with-helmet/">With Helmet</a><br/>Async:
Can Google See Your Content? <a href="/async-stuff/incremental-loading/">Incremental Loading</a> | <a href="/async-
stuff/important-content">Important Content</a></p></div></nav><div class="justify-content-center"><div class="col text-center">
<div><h1>React SEO Demo App</h1><h3>3 Quick ways to Improve Your Search Engine Visibility</h3><h4>(Our Theme: Cats and Dogs, the
topic of many heated debates.)</h4></div></div></div></div>
12        </div>
13      </div>
14      <script>
15        window.__STATE__ = {}
16      </script>
17      <script src="/bundle.js"></script>
18    </body>
19  </html>
20
```


<https://react-seo-demo-ssr.herokuapp.com/header/no-helmet/> ↗

Googlebot type: Desktop (render requested)

✓ **Complete** on Thursday, January 3, 2019 at 3:16:29 PM PST

Fetching

Rendering

This is how Googlebot saw the page:

React SEO Demo | [Home](#) | Welcome! What would you like to view?
Routing: [Bad Component Switching](#) | [Browser Router Demo](#) | [Hash Router Demo](#)
Header (Meta Tags): [No Helmet](#) | [With Helmet](#)
Async: [Can Google See Your Content?](#) | [Incremental Loading](#) | [Important Content](#)

React SEO Demo App

3 Quick ways to Improve Your Search Engine Visibili

(Our Theme: Cats and Dogs, the topic of many heated debates.)

BINNIE CAT

and friends

"Emotions"



Moody



Anxious



Excited



React SEO Demo | [Bad Component Switching](#) | [Hash Router Demo](#) | [With Helmet](#) | [Async:](#)
| [Can Google See Your Content?](#) | [Incremental Loading](#) | [Important Content](#)

WHY RENDER SERVER-SIDE?

2 MAIN REASONS:

1. Faster initial Page Load/Render times
(only if your server is faster than the client!)
2. Fully-Indexable HTML (for SEO)

SO HOW IS SSR DONE?

The basic concept:

CLIENT-RENDERED

- ▶ Send `shell` html file
- ▶ Client loads `bundle.js`
- ▶ React's `virtual DOM` is injected into the html file.
- ▶ React smart-updates the view on subsequent renders.

SERVER-RENDERED

- ▶ Create html markup from React code, and send it back as a string response (essentially HTML document)
- ▶ Client loads `bundle.js` (nothing different here)
- ▶ React smart-updates the view on subsequent renders.

CLIENT-RENDERED



- ▶ Quick to pick which coffee/tea
- ▶ Modular
- ▶ Less overhead (only use the coffee you need)
- ▶ Not always ready (you have to wait a few seconds while it brews)

SERVER-RENDERED



- ▶ Always ready to serve
- ▶ Prepped ahead of time
- ▶ More complicated to maintain
- ▶ More overhead (use more coffee, even if you don't need it)

SO HOW IS SSR DONE?

When the server starts
(Before requests are made to the server)

CLIENT-RENDERED

Bundle.js is formed
(but not used yet)

SERVER-RENDERED

Bundle.js is formed, but not
used yet (nothing different yet)

WHERE SSR IS DIFFERENT:

React code (jsx) is compiled & rendered on the
server, before requests come in.

(note: Make sure to set up Babel correctly)

SO HOW IS SSR DONE?

Request comes in.

IN APP.JS

CLIENT-RENDERED

```
18 app.get('/*', (req, res, next) => {  
19   |   res.sendFile(path.join(__dirname, '..', 'index.html'))  
20   | })
```

Send back ... oh wait, let's go find index.html

SERVER-RENDERED

```
25 app.get('/*', (req, res) => {  
26   |   const { preloadedState, content } = ssr(initialState, req.url)  
27   |   const response = template('Server Rendered Page', preloadedState, content)  
28   |   res.setHeader('Cache-Control', 'assets, max-age=604800')  
29   |   res.send(response);  
30   | });
```

Send back ... oh wait, let's run `ssr` first.

SO HOW IS SSR DONE?

CSR

LOAD HTML DOCUMENT

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <!-- Responsive design? Check. This tag will make mobile browsers scale to device width -->
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7     <title>React SEO Demo App: Preston Wallace's App</title>
8     <script defer src='/bundle.js'></script>
9   </head>
10  <body>
11    <div id='app'></div>
12  </body>
13 </html>
```

Great. Here's index.html. Let's send this back.

SSR

GRAB REACT CODE (ON SERVER IN NODE)

```
6 import Root from '../views/components/root'
7 import configureStore from '../views/store'
8 module.exports = function (initialState, url = '/') {
9   // Model the initial state
10  const store = configureStore(initialState)
11  let content = renderToString(
12    <Provider store={store}>
13      <StaticRouter location={url} context={{}}>
14        <Root />
15      </StaticRouter>
16    </Provider>
17  );
18  const preloadedState = store.getState()
19  return {content, preloadedState};
20 }
```

Perfect. Grab the *already-transpiled* React code (the version in `views` that Node can understand), then give it back to app.js (this is the reason why it sends back a very performant **FIRST RENDER** COMPARED TO CSR)

SO HOW IS SSR DONE?

CSR

SSR

Nice. We've got `content` now. Let's send back... oh wait, what's this `template` thing?

```
25 app.get('/*', (req, res) => {  
26   const { preloadedState, content} = ssr(initialState, req.url)  
27   const response = template('Server Rendered Page', preloadedState, content)  
28   res.setHeader('Cache-Control', 'assets, max-age=604800')  
29   res.send(response);  
30 });
```

SO HOW IS SSR DONE?

CSR

SSR

CREATE HTML TO SEND

OK, render our `content` (our pre-transpiled React components, put all together), and the function returns our entire html document as a string.

```
1  function template(title, initialState = {}, content = ""){
2    let page = `<!DOCTYPE html>
3                <html lang="en">
4                  <head>
5                    <meta charset="utf-8">
6                    <title> ${title} </title>
7                    <link rel="stylesheet" href="/index.css">
8                  </head>
9                  <body>
10                   <div class="content">
11                     <div id="app" class="wrap-inner">
12                       <!-- magic happens here -->  ${content}
13                     </div>
14                   </div>
15                   <script>
16                     window.__STATE__ = ${JSON.stringify(initialState)}
17                   </script>
18                   <script src="/bundle.js"></script>
19                 </body>
20               </html>
21             `;
22
23    return page;
24  }
```


SO HOW IS SSR DONE?

CSR

We've got index.html, let's send it back!

```
18 app.get('/*', (req, res, next) => {  
19   res.sendFile(path.join(__dirname, '..', 'index.html'))  
20 })
```

SSR

Nice. Now that we have everything, let's send back ALL the html, no React needed on the client.

```
25 app.get('/*', (req, res) => {  
26   const { preloadedState, content } = ssr(initialState, req.url)  
27   const response = template('Server Rendered Page', preloadedState, content)  
28   res.setHeader('Cache-Control', 'assets, max-age=604800')  
29   res.send(response);  
30 });
```

THIS IS WHAT WE GET BACK WHEN REQUESTING '/'

CSR

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <!-- Responsive design? Check. This tag will make mobile browsers sc.
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7     <title>React SEO Demo App: Preston Wallace's App</title>
8     <script defer src='/bundle.js'></script>
9   </head>
10  <body>
11    <div id='app'></div>
12  </body>
13 </html>
```

The `shell` of an html file, into which we inject our 'virtual' DOM.

SSR

```
1 <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="utf-8">
5       <title> Server Rendered Page </title>
6       <link rel="stylesheet" href="/index.css">
7     </head>
8     <body>
9       <div class="content">
10        <div id="app" class="wrap-inner">
11          <!-- magic happens here --> <div class="App"><nav><div class="container">React SEO Demo | <a
href="/">Home</a> | Welcome! What would you like to view?<p>Routing: <a href="/routing/bad-component-switching">Bad Component
Switching</a> | <a href="/routing/browser-router">Browser Router Demo</a> | <a href="/routing/hash-router">Hash Router Demo</a>
<br/>Header (Meta Tags): <a href="/header/no-helmet/">No Helmet</a> | <a href="/header/with-helmet/">With Helmet</a><br/>Async:
Can Google See Your Content? <a href="/async-stuff/incremental-loading/">Incremental Loading</a> | <a href="/async-
stuff/important-content">Important Content</a></p></div></nav><div class="justify-content-center"><div class="col text-center">
<div><h1>React SEO Demo App</h1><h3>3 Quick ways to Improve Your Search Engine Visibility</h3><h4>(Our Theme: Cats and Dogs, the
topic of many heated debates.)</h4></div></div></div></div>
12        </div>
13      </div>
14      <script>
15        window.__STATE__ = {}
16      </script>
17      <script src="/bundle.js"></script>
18    </body>
19  </html>
20
```

Our ENTIRE html file. Nothing needs to be loaded in JS on the first render.

SO HOW IS SSR DONE?

CSR

RUN REACT CODE (ON CLIENT)

(actually, we run `bundle.js` which is the transpiled version of all our components, in one single file.

SSR

`bundle.js` is run, but it doesn't render anything until we update something. For now, it just attaches event handlers to be ready!

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import {BrowserRouter} from 'react-router-dom';
4
5  import '../public/index.css';
6  import Root from '../components/root'
7
8  ReactDOM.render(
9    <BrowserRouter>
10     <Root />
11   </BrowserRouter>,
12   document.getElementById('app') // make sure this is the same as
13 );
14
```

SO HOW IS SSR DONE?

On Subsequent Renders

CSR

`render` will smart-update the ReactDOM, only changing that which has changed.

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import {BrowserRouter} from 'react-router-dom';
4
5  import '../public/index.css';
6  import Root from './components/root'
7
8  ReactDOM.render(
9    <BrowserRouter>
10     <Root />
11   </BrowserRouter>,
12   document.getElementById('app') // make sure this is the same as
13 );
14
```

SSR

`hydrate` is like `render`, but it will only attempt to attach event listeners to the existing markup. Basically, it `smart updates` what is displayed **after** all the html has been loaded.

```
2  import React from 'react'
3  import ReactDOM from 'react-dom'
4  import {BrowserRouter as Router} from 'react-router-dom';
5  import {Provider} from 'react-redux'
6
7  import configureStore from './store'
8  import Root from './components/root'
9
10 const state = window.__STATE__;
11 delete window.__STATE__;
12 const store = configureStore(state)
13
14 ReactDOM.hydrate(
15   <Provider store={store} >
16     <Router>
17       <Root />
18     </Router>
19   </Provider>,
20   document.getElementById('app')
21 )
```


TO REVIEW:

3 POWERFUL TOOLS:

(no SSR required)

1. React Router
2. React Helmet
3. Fetch as Google

4

TO REVIEW:

~~X~~ POWERFUL TOOLS:

(no SSR required)

1. React Router
2. React Helmet
3. Fetch as Google

4. ISOMORPHIC APP

Sends pre-rendered, fully-indexable HTML

Also has benefits of Single-Page Application

Potentially faster initial load time

RESOURCES

MY REACT SEO APP EXAMPLES:

ISOMORPHIC APP EXAMPLE

Deployed Page: <https://react-seo-demo-app.herokuapp.com/>

GitHub Repo: <https://github.com/wallacepreston/react-seo-demo>

ISOMORPHIC APP EXAMPLE

Deployed Page: <https://react-seo-demo-ssr.herokuapp.com/>

GitHub Repo: <https://github.com/wallacepreston/react-seo-demo-ssr>

MY REACT SEO TUTORIAL ON MEDIUM:

<https://medium.com/@wallace.preston/3-ways-improve-react-seo-without-isomorphic-app-a6354595e400>