React.js: reduce your javascript bundle with code splitting

Pierre Cavalet Follow
May 24, 2019 · 5 min read ★

Photo by Tanalee Youngblood on Unsplash

I'm technical expert at Kaliop and today I speak about react.js. If you want to know more about this subject, click here.

Users care more and more about performance. We are building rich applications, and as an application grows, **so does the javascript bundle**.

When developing a single page application, we create a lot of components. Each of these components has its own dependencies, util librairies and so on. We keep developing features and at some point, we realise that our website is slow 🙀. While investigating, we realise that our javascript bundle is too big. There are different ways to find this out:

- It takes too much time to load when using a slow network. We can test this easily using the chrome dev tools and the network tab.
- Webpack itself tells us that our bundle is too big:

```
WARNING in entrypoint size limit: The following entrypoint(s)
combined asset size exceeds the recommended limit (244 KiB). This can
impact web performance.
```

- Tools to analyze your website (such as lighthouse) tell us to reduce your bundle size:

```
Consider reducing the time spent parsing, compiling and executing JS.
You may find delivering smaller JS payloads helps with this.
```

If you came to one of these situations, the next question you may ask yourself is:

> *Fine, I have a performance problem because of my bundle size. But how do I deal with it?*

Mesure. Analyze. Take action.

We will use a simple react application with a *Home* and *Heavy* routes, with react router. The Heavy component will use lodash and moment libraries just to make it as large as possible (in terms of bundle size).

```
import React from 'react'

export default function Home() {
  return <div>Home</div>
}
```

Home.js hosted with ❤ by **GitHub** **view raw**

```
import React from 'react'
import _ from 'lodash'
import _ from 'lodash'
import moment from 'moment'

export default function Heavy() {
  const now = moment().format('YYYY-MM-DD')
  const textArray = ['The', 'date', 'is']
  const text = _.join(textArray, '')

  return (
    <div>
      {text} - {now}
    </div>
  )
}
```

```
import React from 'react'
import { BrowserRouter as Router, Route, Link } from 'react-router-dom'

import Home from './Home'
import Heavy from './Heavy'

function AppRouter() {
  return (
    <Router>
      <div>
        <nav>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/heavy/">Heavy</Link>
            </li>
          </ul>
        </nav>

        <Route path="/" exact component={Home} />
        <Route path="/heavy/" component={Heavy} />
      </div>
    </Router>
  )
}

export default AppRouter
```

Mesure

You should always mesure the impact of a problem **before** diving into some premature optimisations. We can use source-map-explorer to inspect our javascript bundles and then take actions accordingly.

Note: *I am using source-map-explorer because it is the recommended tool for react and we don't need to eject the configuration. For those who find it difficult to read (like myself), you can use webpack-bundle-analyzer but you might be forced to eject.*

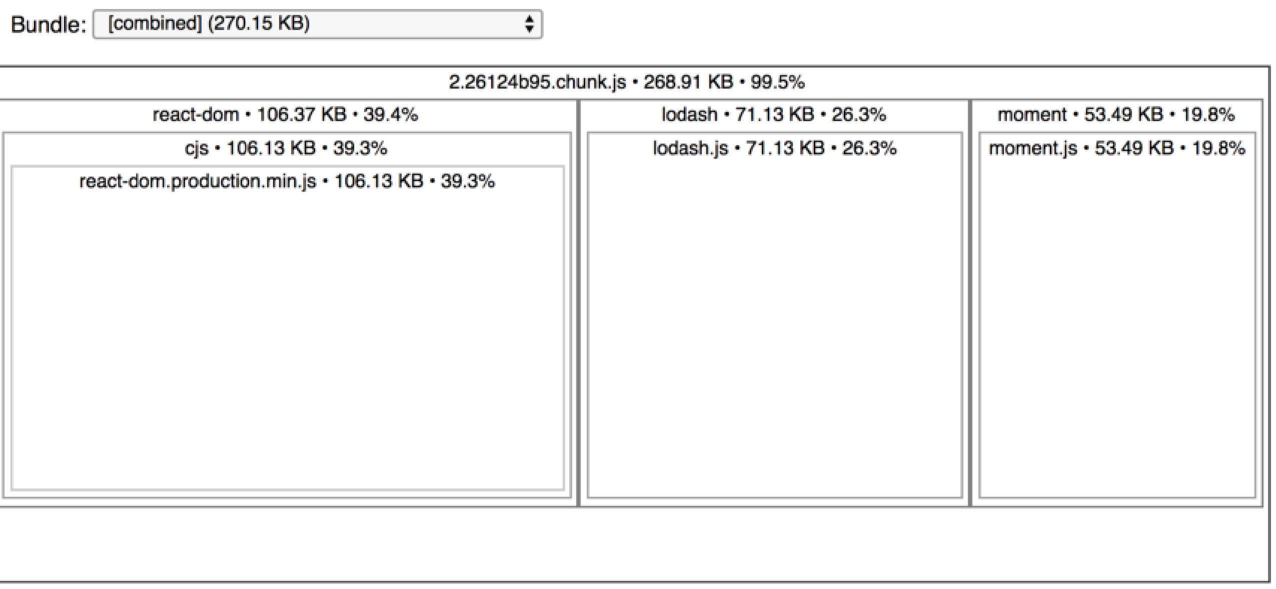
```
npm install --save source-map-explorer
```

We need to add a new script in the *package.json*.

```
"scripts": {
  "analyze": "source-map-explorer 'build/static/js/*.js'",
  ...
}
```

We can then run:

```
npm run build
npm run analyze
```


Bundle: [combined] (270.15 KB)

2.26124b95.chunk.js • 268.91 KB • 99.5%
react-dom • 106.37 KB • 39.4%
cjs • 106.13 KB • 39.3%
react-dom.production.min.js • 106.13 KB • 39.3%
lodash • 71.13 KB • 26.3%
lodash.js • 71.13 KB • 26.3%
moment • 53.49 KB • 19.8%
moment.js • 53.49 KB • 19.8%

Analyze

By looking at the analyzer output, we can see that *lodash* and *moment* take a big place in our bundle. When we load the home page, we actually send lodash and moment unnecessarily. The home page component does not need those libraries.

Here, if we split the Heavy component from the main bundle, it would remove the lodash and moment libraries, reducing the bundle to just above half its size.

This is the kind of things that can drastically reduce your javascript bundles, but you have to keep in mind that it needs to require as little effort as possible, and it should have a significant effect on the bundle size. Do not optimise chunk that weigh 3KiB, aim for the big fish.

Take action

The goal here is to separate the Heavy component (and its dependencies) from the rest of the bundle. We will use the webpack code splitting feature. First let's see how the feature works **outside the react context**.

Note: *When using create-react-app, webpack is already configured to support code splitting with the dynamic import.*

Webpack supports the dynamic import syntax. It uses promises internally. Here is the difference between a static import and a dynamic import:

- The static import returns what is exported by the module as the **default export**.
- The dynamic import returns a promise that resolves an **object with a default property**. This property value is what is exported by the module as the **default export**. When webpack sees a dynamic import, it bundles what is not statically imported in the main bundle into its own chunk.

Back to our application context

First let's see what the network traffic looks like before splitting the component:

We can see that the chunk 2.xxxx weights 263kb (as we saw in the analyzer) and takes 7.36s to load in slow 3G. Now let's split the heavy component and see what happens.

We can use **lazy** and **Suspense**. Lazy allows you to render a dynamically imported component as a regular component and Suspense allows you to define a fallback while the component is loading.

Then we'll use this component in our router.

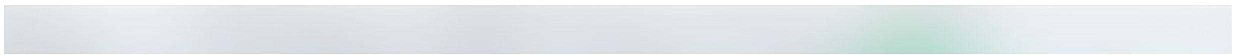
Now let's rebuild and analyze the bundle again.

```
npm run build
npm run analyze
```

This one is a little bit more detailed on the main chunk because the zoom did something different, but that's not important.

The most important thing to look at is that there are now two different chunks (2.xxxx and 3.xxxx). 3.xxxx has been created because we used a dynamic import, as we saw in the previous section. The 3.xxxx chunk contains the Heavy component and won't be loaded if we don't need it. Thanks to the split, our 2.xxxx chunk only weighs 148kb, which is a lot smaller than our previous 263kb.

This is what happens if we load the home page:



The page loads more than 2 seconds faster because the chunk is lighter. Now if we navigate to the heavy route, the chunk is automatically downloaded:

Recap

- Code splitting is useful to reduce your bundle size by splitting big parts and loading them only when you need it.
- **lazy** and **Suspense** helps you to split your components, but it is based on dynamic import which is a webpack feature. You can use this technique anywhere as long as you use webpack, even in non react application.
- Always analyze before trying to improve your performance and focus on the easiest tasks that have the biggest impact on performance.

Thanks to Enzo FABRE, Andréas Hanss, Clement “Snowji” D, and Samuel Bouic.

JavaScript React Code Splitting Web Performance Webpack