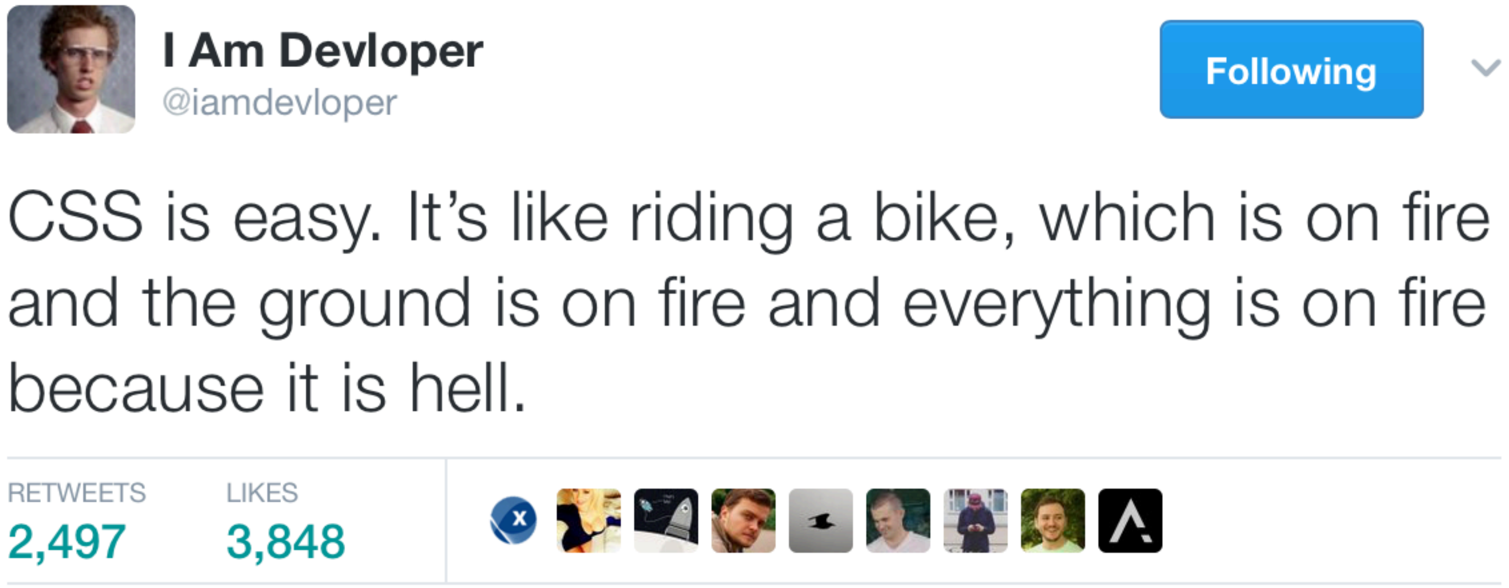
Styling

Cory House

@housecor reactjsconsulting.com



I Am Devloper
@iamdevloper

CSS is easy. It's like riding a bike, which is on fire and the ground is on fire and everything is on fire because it is hell.

RETWEETS 2,497

LIKES 3,848

The Plan

React styling approaches

- Compiled CSS
- Inline Styles
- CSS Modules
- CSS in JS

Styling decisions

1. Pick a styling approach
2. Theming

Compiled CSS and Naming Schemes

Why Compile CSS?

PostCSS

cssnext

Sass

{less}

Variables

Custom functions

Automatic vendor prefixes

Nested selectors

Sass Variables

```
$font-stack: Helvetica, sans-serif
$primary-color: #333 body
font: 100% $font-stack
color: $primary-color
```

Sass Nesting

```
nav
  ul
    margin: 0
    list-style: none
  li
    display: inline-block
```

```
nav ul {
  margin: 0;
  list-style: none;
}

nav li {
  display: inline-block;
}
```

PostCSS

Transpile your CSS

- Autoprefix
- CSS4
- Polyfill Flexbox
- Style linting
- Accessibility
- Much more!

CSS Naming Schemes

“I’m too scared to delete any styles.”

One Solution to Global CSS:

Naming Scheme: BEM, OOCSS, SMACSS

46% have used a naming scheme

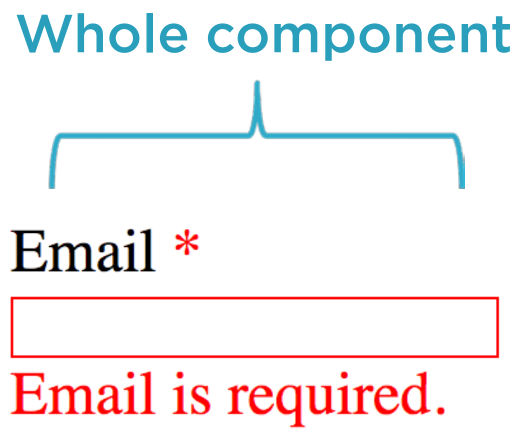
40% BEM

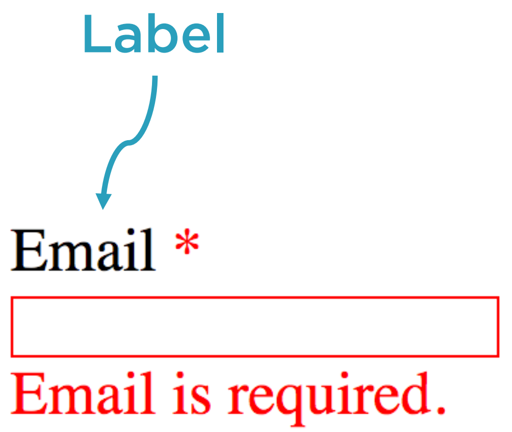
15% OOCSS

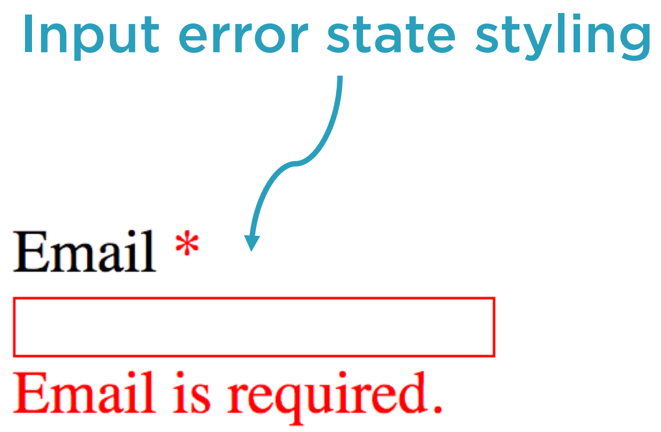
13% SMACSS

Source: sitepoint.com/front-end-tooling-trends-2017

BEM







Block
The component

Element
Part of the component

Modifier
A variant/extension

BEM Example

```
// Block
.registrationform{ }


// Element
.registrationform__submitbutton { }


// Modifier
.registrationform--error { }
```

Demo

CSS with BEM

Inline Styles

style

The `style` attribute accepts a JavaScript object with camelCased properties rather than a CSS string. This is consistent with the DOM `style` JavaScript property, is more efficient, and prevents XSS security holes. For example:

Code

```
const divStyle = {
  color: 'blue',
  backgroundImage: 'url(' + imgUrl + ')',
};

function HelloWorldComponent() {
  return <div style={divStyle}>Hello World!</div>;
}
```

Note that styles are not autoprefixed. To support older browsers, you need to supply corresponding style properties:

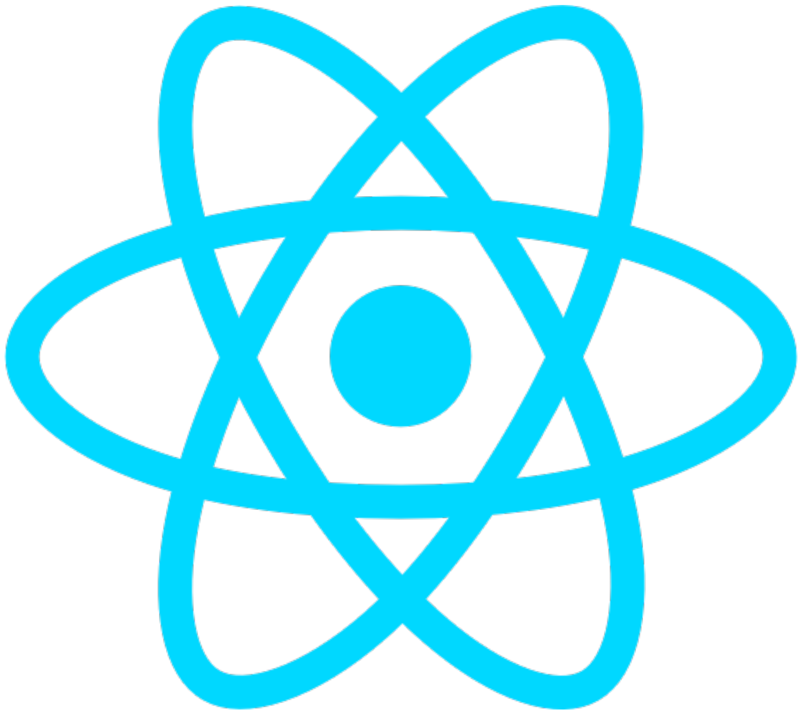
Code

```
const divStyle = {
  WebkitTransition: 'all', // note the capital 'W' here
  msTransition: 'all' // 'ms' is the only lowercase vendor prefix
};

function ComponentWithTransition() {
  return <div style={divStyle}>This should work cross-browser</div>;
}
```

Style keys are camelCased in order to be consistent with accessing the properties on DOM nodes from JS (e.g. `node.style.backgroundImage`). Vendor prefixes other than `ms` should begin with a capital letter. This is why `WebkitTransition` has an uppercase "W".

React Inline Styles – Differences from CSS



Keys are camelCased

Values are strings

Commas instead of semicolons

Capitalized vendor prefixes

Why React Inline Styles?

Advantages	Disadvantages
Easy	Can't use:
Encapsulated	• Pseudo classes
Explicit	• Pseudo elements
No mental mapping overhead	• Media queries
Deterministic	• Style fallbacks
Dynamic styles	• Animations
	Must use !important to override
	Messier markup

How opinionated should my styling be?

- Inline styles = strong enforcement
- External styles = more flexibility

Consider mixing React’s inline styles with plain CSS

Demo

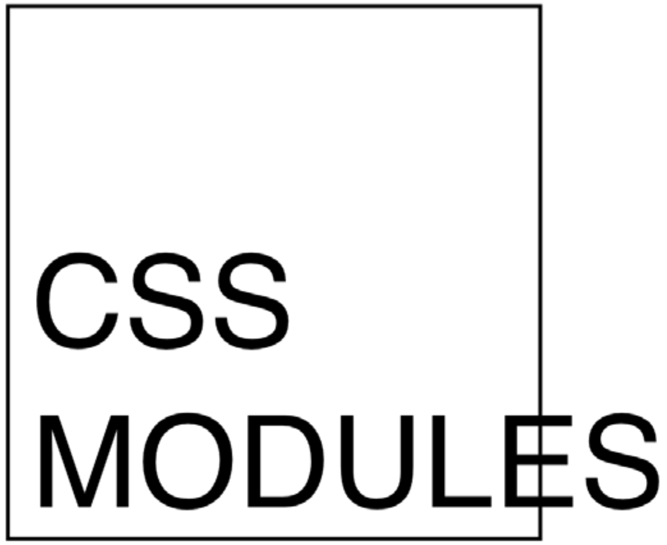
You’ve already seen the demo.☺

CSS Modules

CSS Modules

Class names are scoped locally by default.

CSS Modules



Modular and reusable CSS

1. Write plain CSS
2. Import stylesheet
3. Reference style like an object

1. Declare stylesheet

```
/* TextInput.css */
.error {
  color: 🟥red;
}
```

2. Configure webpack

3. Import stylesheet & apply styles

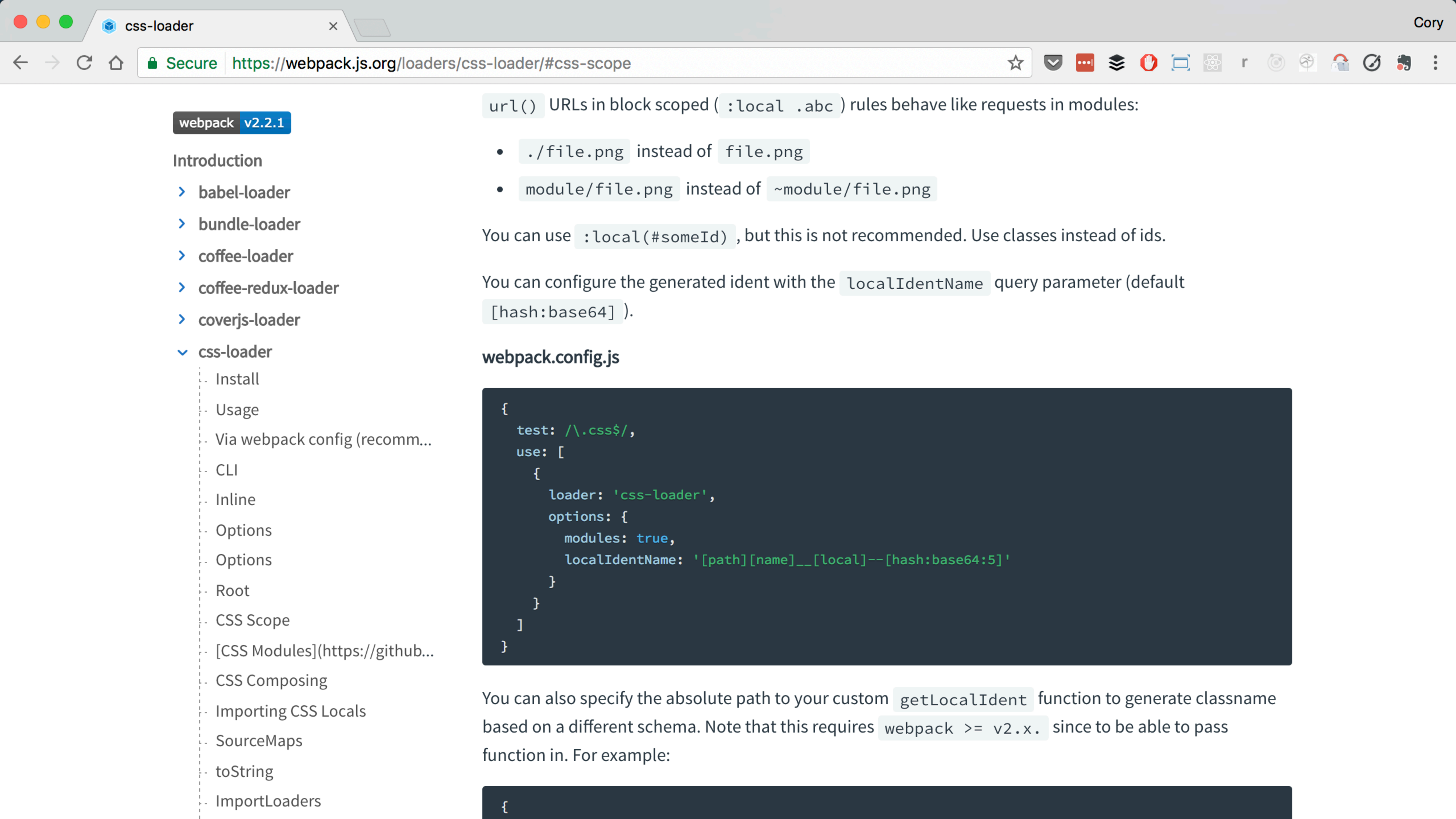
```
import React from 'react';
import PropTypes from 'prop-types';
import Label from '../Label';
import styles from './textInput.css';

/** TextInput using CSS Modules */
function TextInputCssModules ({htmlId, name, label, type = "text",
  return (
    <div className={styles.fieldset}>
      <Label htmlFor={htmlId} label={label} required={required} />
      <input
        id={htmlId}
        type={type}
        name={name}
        className={error && styles.inputError}
        placeholder={placeholder}
        value={value}
        onChange={onChange}
        {...props}/>
        {children}
      {error && <div className={styles.error}>{error}</div>}
    </div>
  );
};
```

```
<div className="textInput__error___16c-r">{error}</div>}
```

```
// Webpack 1
'css-loader?modules&importLoaders=1&localIdentName=[name]_[local]_[hash:base64:5]'


// Webpack 2
loader: 'css-loader',
options: {
  modules: true,
  localIdentName: '[name]_[local]_[hash:base64:5]'
},
```



webpack v2.2.1

Introduction
- babel-loader
- bundle-loader
- coffee-loader
- coffee-redux-loader
- coverjs-loader
- css-loader
  - Install
  - Usage
  - Via webpack config (recomm...
  - CLI
  - Inline
  - Options
  - Options
  - Root
  - CSS Scope
  - [CSS Modules](https://github...
  - CSS Composing
  - Importing CSS Locals
  - SourceMaps
  - toString
  - ImportLoaders

`url()` URLs in block scoped (`:local .abc`) rules behave like requests in modules:

- `./file.png` instead of `file.png`
- `module/file.png` instead of `~module/file.png`

You can use `:local(#someId)` , but this is not recommended. Use classes instead of ids.

You can configure the generated ident with the `localIdentName` query parameter (default `[hash:base64]`).

webpack.config.js

```
{
  test: /\.css$/,
  use: [
    {
      loader: 'css-loader',
      options: {
        modules: true,
        localIdentName: '[path][name]__[local]--[hash:base64:5]'
      }
    }
  ]
}
```

You can also specify the absolute path to your custom `getLocalIdent` function to generate classname based on a different schema. Note that this requires `webpack >= v2.x.` since to be able to pass function in. For example:

```
{
```

1. Declare stylesheet

```css
/* TextInput.css */
.error {
  color: 🟥 red;
}
```

2. Configure webpack

3. Import stylesheet & apply styles

```jsx
import React from 'react';
import PropTypes from 'prop-types';
import Label from '../Label';
import styles from './textInput.css';

/** TextInput using CSS Modules */
function TextInputCssModules ({htmlId, name, label, type = "text",
  return (
    <div className={styles.fieldset}>
      <Label htmlFor={htmlId} label={label} required={required} />
      <input
        id={htmlId}
        type={type}
        name={name}
        className={error && styles.inputError}
        placeholder={placeholder}
        value={value}
        onChange={onChange}
        {...props}/>
        {children}
      {error && <div className={styles.error}>{error}</div>}
    </div>
  );
};
```

```html
<div className="textInput__error___16c-r">{error}</div>}
```

Why CSS Modules?

Write plain CSS, or Sass/Less

Explicit

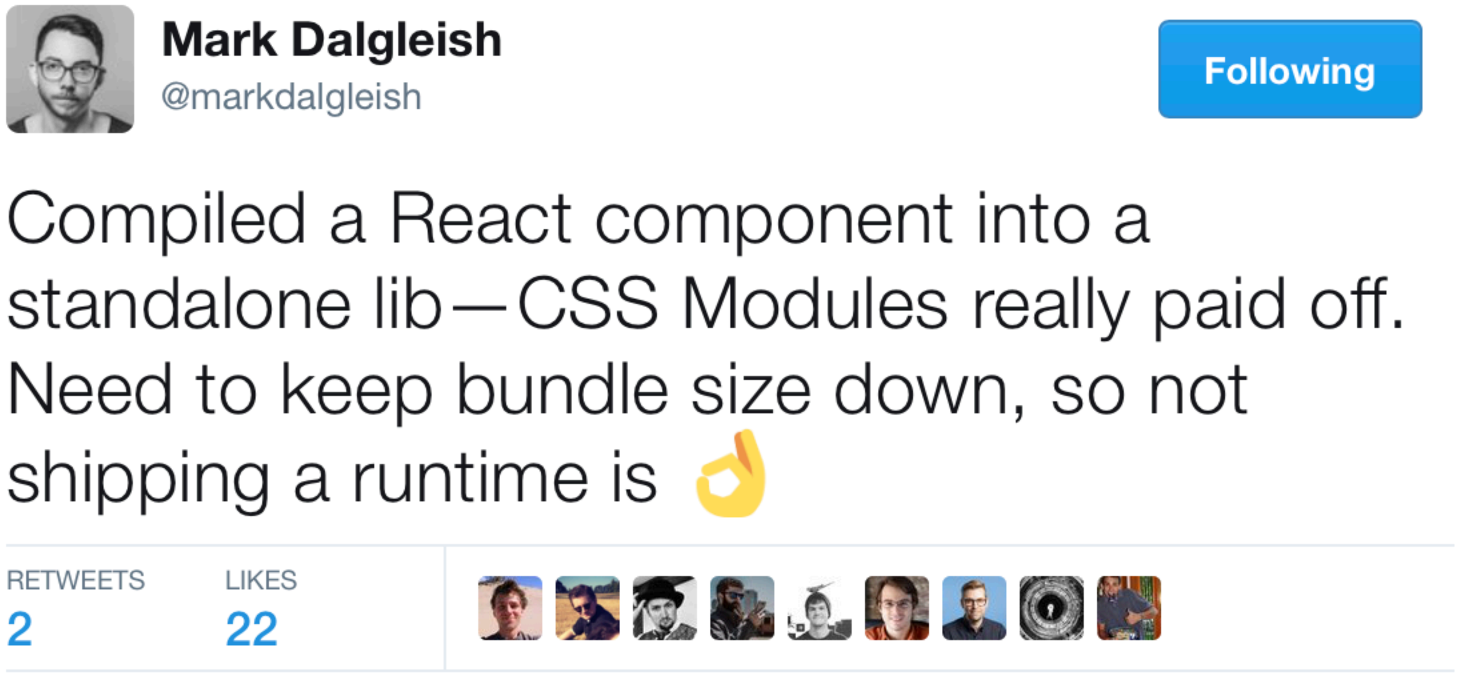
Composable

Automatic encapsulation

No risk of conflicts

Write short CSS selectors

Supports global CSS



Mark Dalgleish
@markdalgleish

Compiled a React component into a standalone lib—CSS Modules really paid off. Need to keep bundle size down, so not shipping a runtime is 👌

RETWEETS 2 LIKES 22

Why Not CSS Modules

CSS
MODULES

No dynamic state-based styles

Theming is tricky

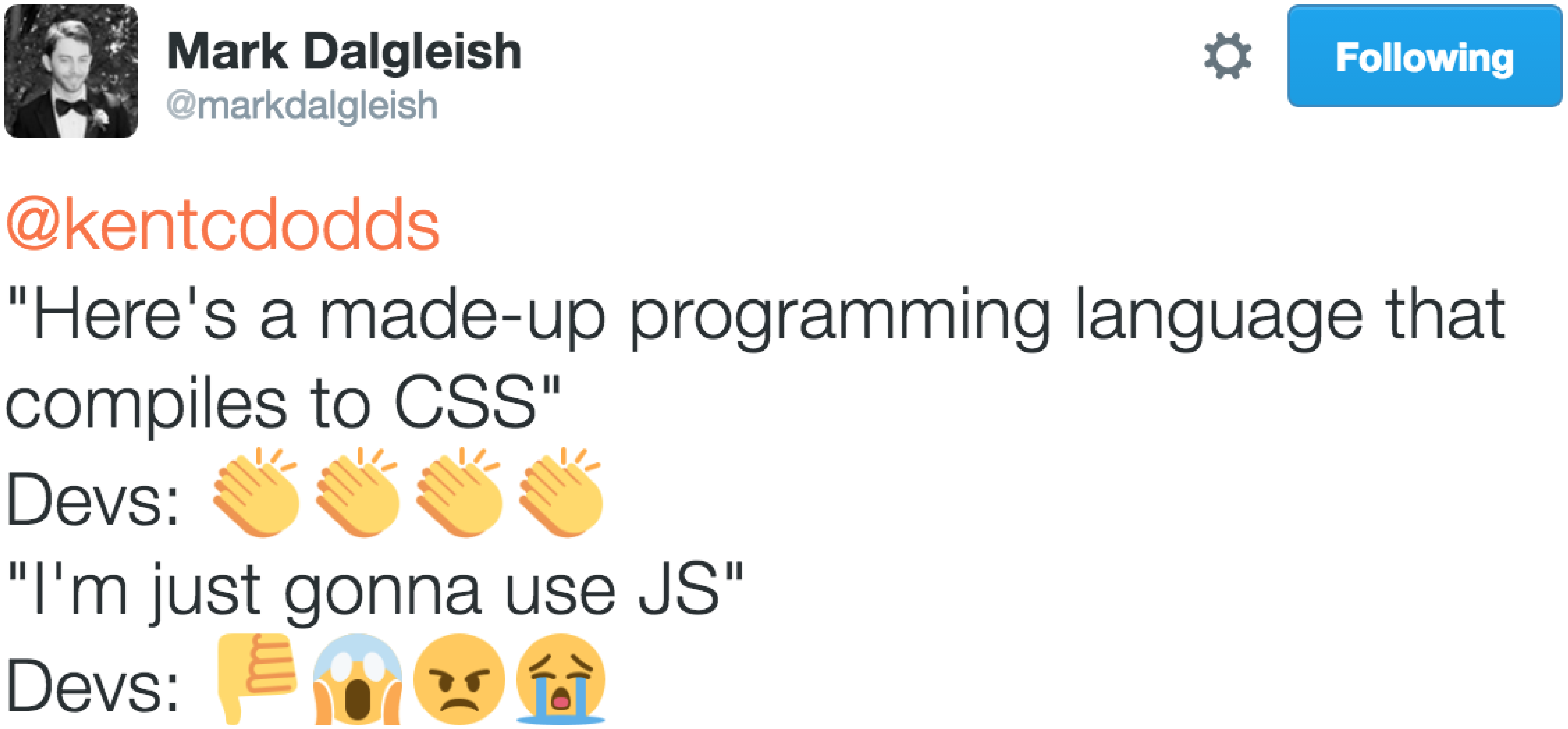
Demo

CSS Modules

CSS in JS

~50 CSS in JS options?!

O_o



Mark Dalgleish
@markdalgleish

@kentcdodds
"Here's a made-up programming language that compiles to CSS"
Devs: 👏👏👏👏
"I'm just gonna use JS"
Devs: 👎😱😠😭

Why JS Over Sass, Less, PostCSS?

You already know JS

- Variables
- Functions
- Looping

JS is a better language

Leverage the same tooling

- Linter
- Minification
- Dead code elimination

CSS in JS Issues

No sourcemaps

Library lock-in

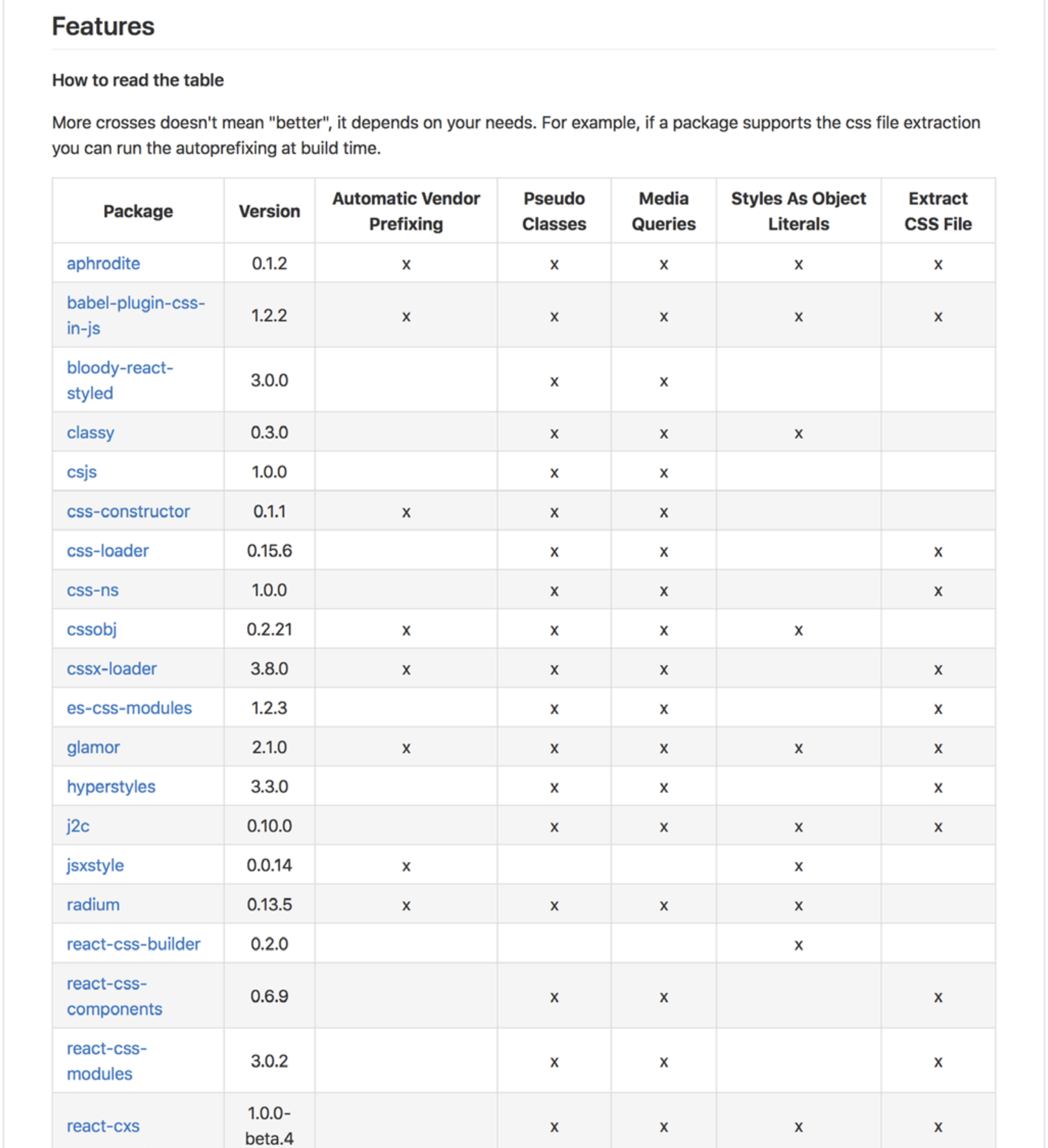
May lack full CSS support

Confusing when mixed with CSS

Forces consumers to accept your decision

JS styles

- Aphrodite
- Radium
- JSS
- jsxstyle
- Styled-components
- Glamour
- Over 40 more!



Features

How to read the table

More crosses doesn't mean "better", it depends on your needs. For example, if a package supports the css file extraction you can run the autoprefixing at build time.

Package	Version	Automatic Vendor Prefixing	Pseudo Classes	Media Queries	Styles As Object Literals	Extract CSS File
aphrodite	0.1.2	x	x	x	x	x
babel-plugin-css-in-js	1.2.2	x	x	x	x	x
bloody-react-styled	3.0.0		x	x		
classy	0.3.0		x	x	x	
csjs	1.0.0		x	x		
css-constructor	0.1.1	x	x	x		
css-loader	0.15.6		x	x		x
css-ns	1.0.0		x	x		x
cssobj	0.2.21	x	x	x	x	
cssx-loader	3.8.0	x	x	x		x
es-css-modules	1.2.3		x	x		x
glamor	2.1.0	x	x	x	x	x
hyperstyles	3.3.0		x	x		x
j2c	0.10.0		x	x	x	x
jsxstyle	0.0.14	x			x	
radium	0.13.5	x	x	x	x	
react-css-builder	0.2.0				x	
react-css-components	0.6.9		x	x		x
react-css-modules	3.0.2		x	x		x
react-cxs	1.0.0-beta.4		x	x	x	x

	styled-components	Radium	Aphrodite	JSS	Glamor	jsx-style
GitHub Stars	5160	5036	2627	1940	1895	1557
Automatic Vendor prefixing	x	x	x	x	x	x
Pseudo classes	x	x	x	x	x	x
Media queries	x	x	x	x	x	No
Write plain CSS	Yes	Object literals	Object literals	Object literals	Object literals	Object literals
Extract CSS file	No	No	x	x	x	x

Demo

CSS in JS via styled-components

Decision:
Styling approach

React Styling Approaches

Plain CSS

Compiled CSS

Inline Styles

CSS Modules

CSS in JS

	CSS	Sass/Less	CSS Modules	Inline styles	CSS in JS
Deterministic	No	No	Yes	Yes	Yes
Explicit application	No	No	Yes	Yes	Yes
Dead code elimination	No	No	Yes	Yes	Yes
Encapsulated	No, but BEM helps	No, but BEM helps	Yes	Yes	Yes
Collocated	No	No	No	Yes	Yes
Themeable	Yes	Yes, change variables	Yes, but no standard	Not easily	Varies
Full CSS Support	Yes	Yes	Yes	No	Not typically
Write standardized CSS	Yes	No	Yes, or Sass/Less/PostCSS	No	Varies
Generate plain CSS during build	Yes	Yes	Yes, but dynamic class names	No	Varies
Syntax highlighting	Yes	Yes	Yes	It's JS	Varies
Autocomplete when writing	Yes	Yes	Yes	No	No
Autocomplete when using	No	No	No	Yes	Yes
Library lock-in	No	Yes	CSS, no. In React, yes	Yes	Varies
Requires wrapper component	No	No	No	No	Varies
Server side rendering support	Yes	Yes	Must run webpack on server	Yes	Yes
Post-processing support	Yes	Yes	Yes	No	Varies
Lintable	Yes	Yes	Yes	No	Not typically
Build Setup	None	webpack loader	css-loader, or PostCSS	None	None
Style testing setup	Test via screenshots	Test via screenshots	Config tests to ignore	None	None

Oh great, they're all lousy.

Need theming?

- Prefer Sass/Less/cssnext, avoid inline

Can't trust people to follow conventions?

- Prefer inline or CSS in JS

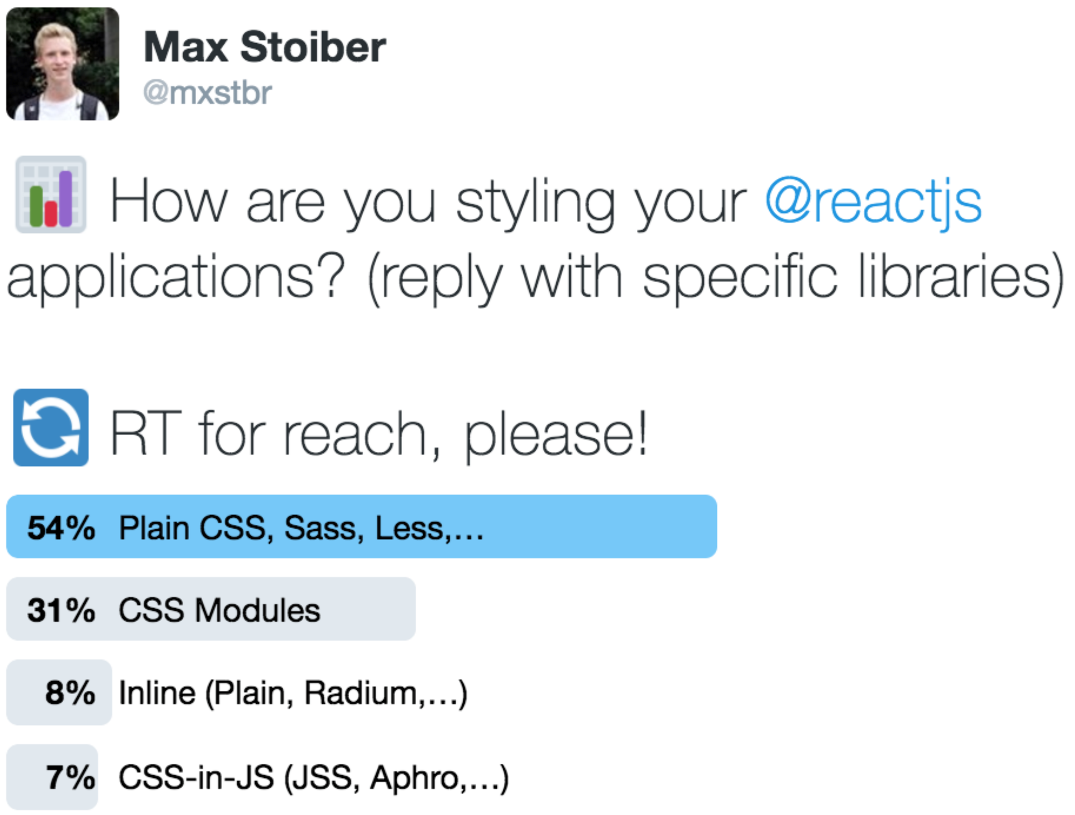
Concerned about lock-in?

- Prefer plain CSS

Automated style testing important?

- Easier with inline or CSS in JS

July 2016



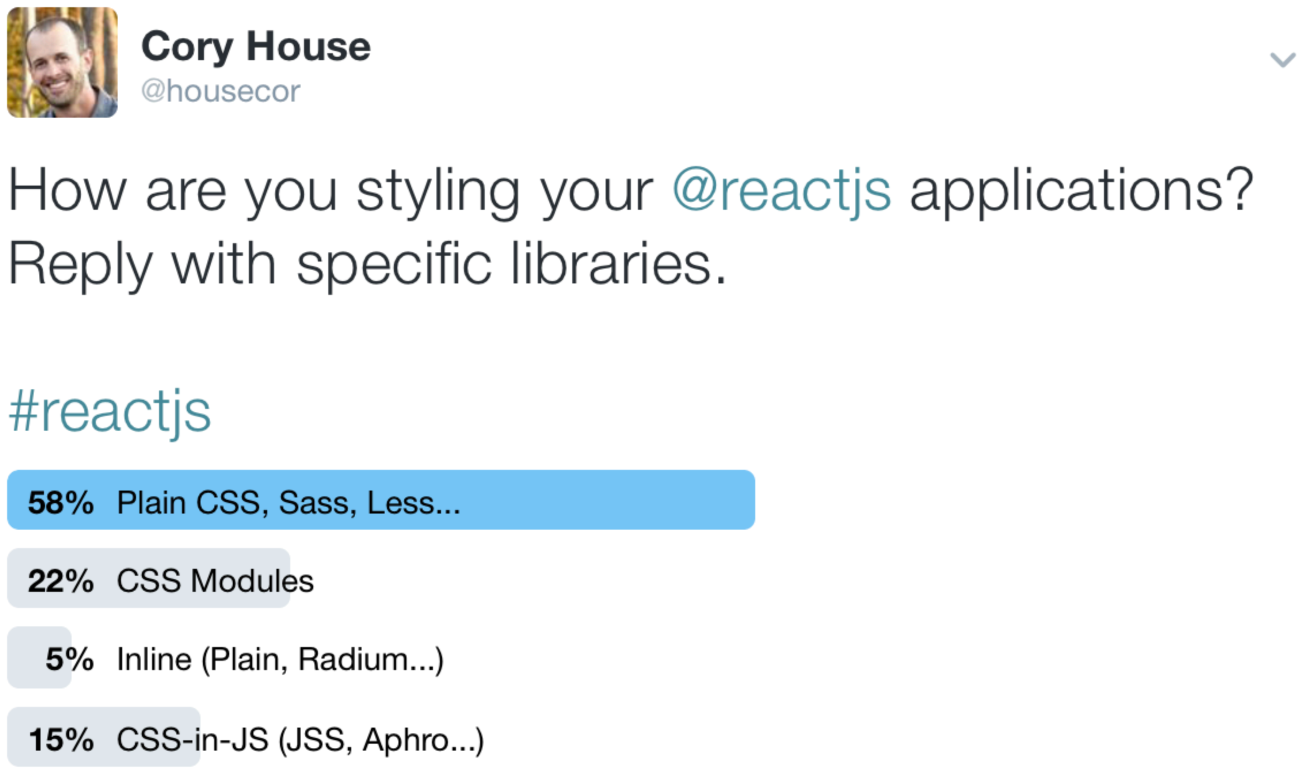
Max Stoiber
@mxstbr

📊 How are you styling your @reactjs applications? (reply with specific libraries)

🔄 RT for reach, please!

54%	Plain CSS, Sass, Less,...
31%	CSS Modules
8%	Inline (Plain, Radium,...)
7%	CSS-in-JS (JSS, Aphro,...)

Feb 2017



Cory House
@housecor

How are you styling your @reactjs applications? Reply with specific libraries.

#reactjs

58%	Plain CSS, Sass, Less...
22%	CSS Modules
5%	Inline (Plain, Radium...)
15%	CSS-in-JS (JSS, Aphro...)

Styling Approaches Used By Top Libraries

React Toolbox	CSS modules with cssNext via PostCSS
React Bootstrap	Less with bs prefix
Ant	Less with ant prefix
Blueprint	Sass
Grommet	Sass with BEM with grommetux prefix
Material-UI	Inline styles (also offer unstyled exports)

Decision:
Unstyled, enforced or themeable?

Styling Opinions

Font

Color

Size

Border

Positioning

Responsive Design

Unstyled Themeable Built-in styles

Opinion →

← Flexibility

Should a component be shipped *without* styles?

Public? Be themeable.

Private? Be opinionated.

Conway’s Law

Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations

Funnel designs through a centralized team.

Un-themed UI Libraries

material-ui-core

belle-core

elemental-core

react-toolbox-core

Un-themed UI Components

react-select

react-autocomplete

react-modal

If you're using external CSS, you can offer both themed and unthemed.

“Depending on whether you want the styles…you can import components in two different ways.”

React Toolbox

```
// Styled
import { AppBar } from 'react-toolbox/lib/app_bar';


// Unstyled
import AppBar from 'react-toolbox/lib/app_bar/AppBar.js';
```

Theming Approaches

There is no theming standard.

Theming Approaches By Tech

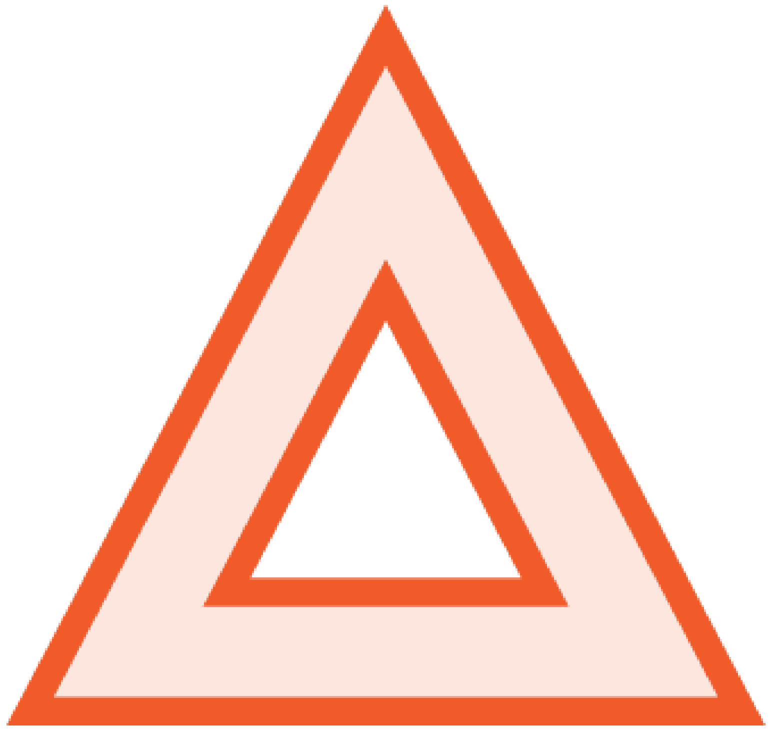
CSS, Sass, Less	Use a different stylesheet
Inline styles	Accept theme as object via prop
CSS modules	Accept theme as object via prop
CSS in JS	Varies

Careful With CSS Classes for Themes

May conflict with other markup

Consider a unique prefix

Namespace Your Theme Styles



Ant	ant
React-Bootstrap	bs
Grommet	grommetux

Wrap Up

React styling approaches

- Compiled CSS
- Inline Styles
- CSS Modules
- CSS in JS

Styling decisions

1. Use naming schemes for compiled CSS
2. Theming – Consider shipping without styles.

Next up: Testing