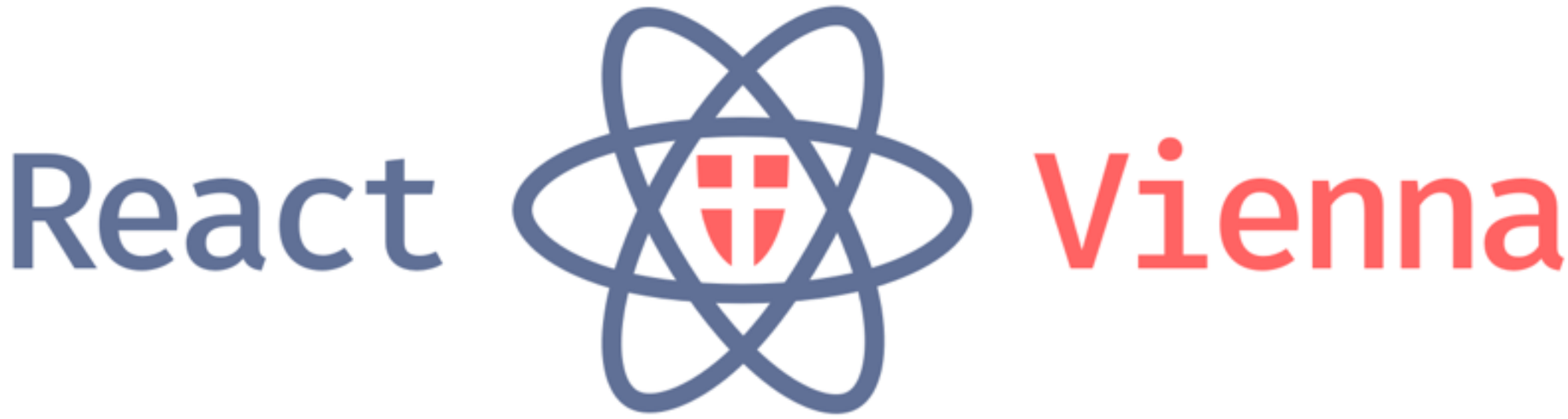
Modular CSS

with @okonetchnikov

Andrey Okonetchnikov

@okonetchnikov


React
Vienna

ColorSnapper

http://colorsnapper.com

kaffemik

Zollergasse 5,
1070 Wien

I ❤ Open Source

github.com/okonet/react-dropzone
github.com/okonet/lint-staged

and many more...

Modular CSS

I design & develop

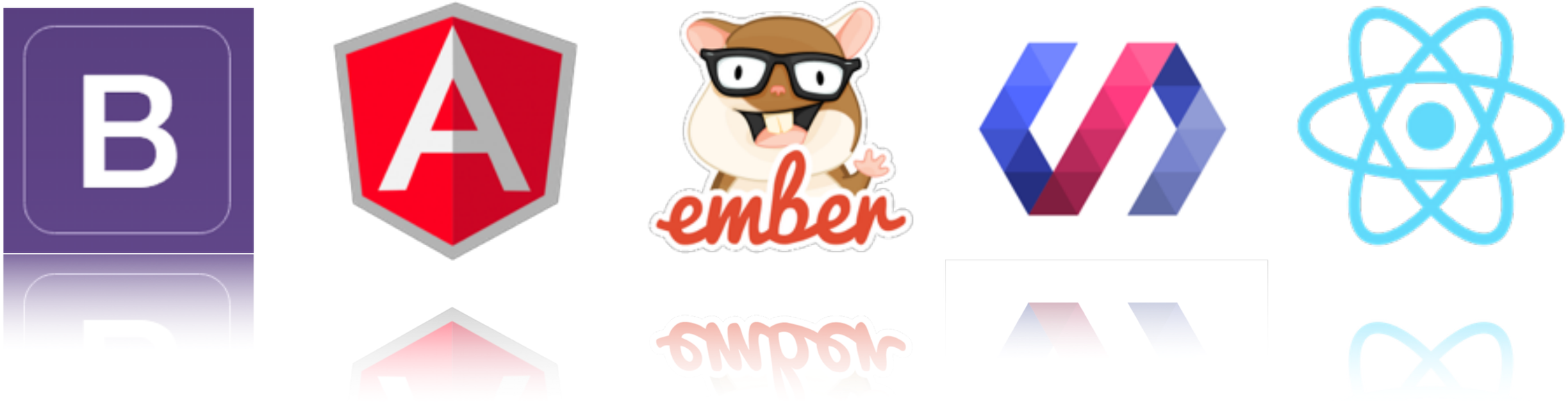
User Interfaces

Building scalable

User Interfaces

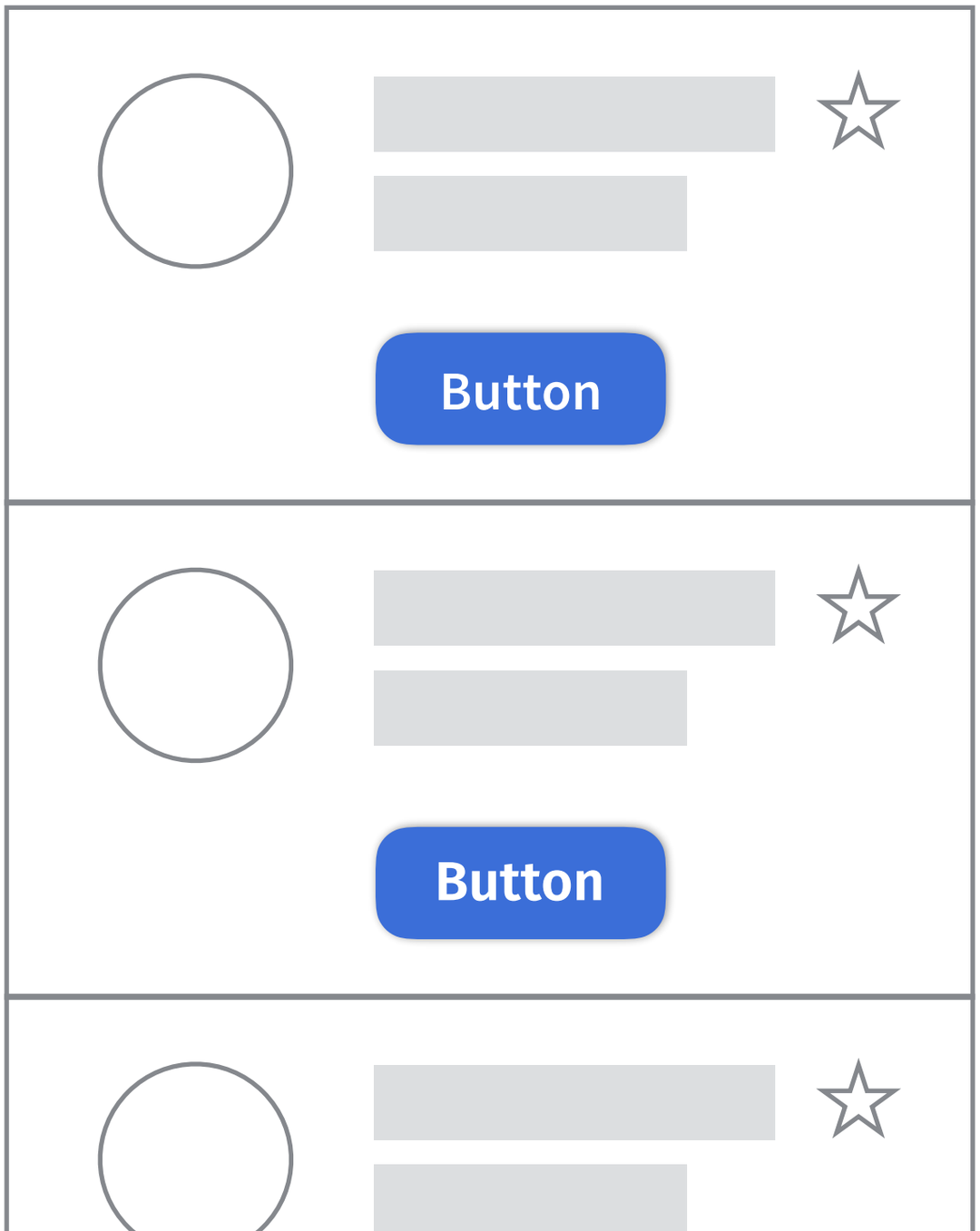
User Interface ∋ Components

Libraries & Frameworks using UI components

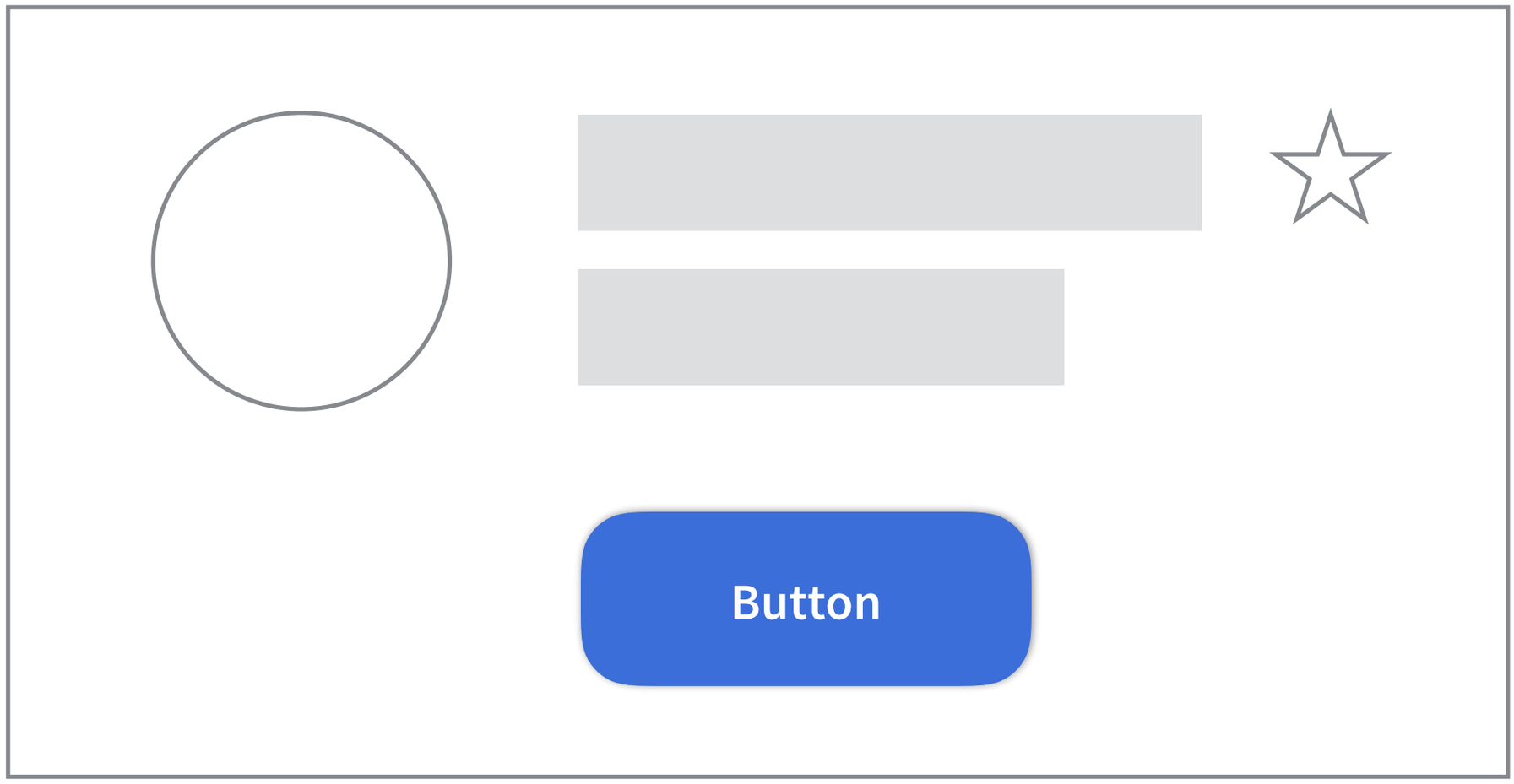


What are UI Components?

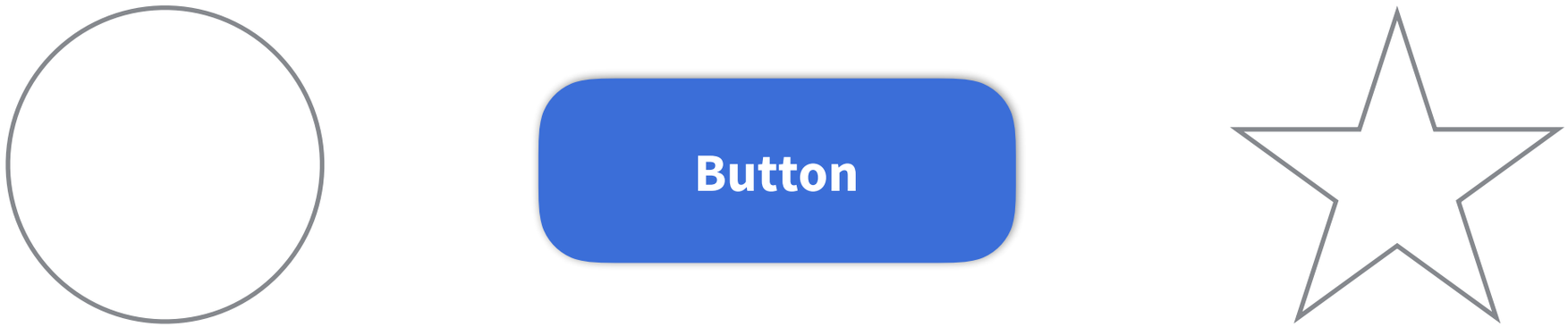
List component



List Item Component



Atomic components



Let’s build a Button!

Button component

Default

Disabled

Primary

In React.js everything is a component!

Button.js

```
const Button = ({ children }) => {
  return (
    <button
      type="button"
    >
      { children }
    </button>
  )
}
```

Button.js

```
const Button = ({ children }) => {
  return (
    <button
      type="button"
      className="btn"
    >
      <span className="label">
        { children }
      </span>
    </button>
  )
}
```

Button.js

```
const Button = ({ children, disabled }) => {
  return (
    <button
      type="button"
      disabled={disabled}
      className="btn"
    >
      <span className="label">
       { children }
      </span>
    </button>
  )
}
```

Button.js

```
1  const Button = ({ children, disabled }) => {
2    let classNames = 'btn'
3    if (disabled) classNames += ' btn--disabled'
4    return (
5      <button
6        type="button"
7        disabled={disabled}
8        className={classNames}
9      >
10       <span className="label">
11        { children }
12       </span>
13     </button>
14   )
15 }
```

Button.js

```
const Button = ({ children, disabled, primary }) => {
  let classNames = 'btn'
  if (disabled) classNames += ' btn--disabled'
  if (primary) classNames += ' btn--primary’
  return (
    <button
      type="button"
      disabled={disabled}
      className={classNames}
    >
      <span className="label">
       { children }
      </span>
    </button>
  )
}
```

Button component

Default

```
<Button>Default</Button>
```

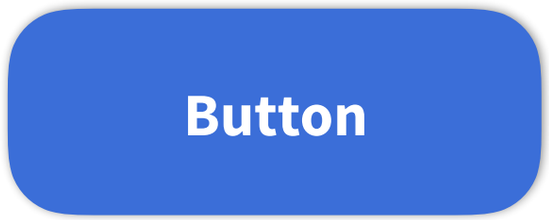
Disabled

```
<Button disabled>Disabled</Button>
```

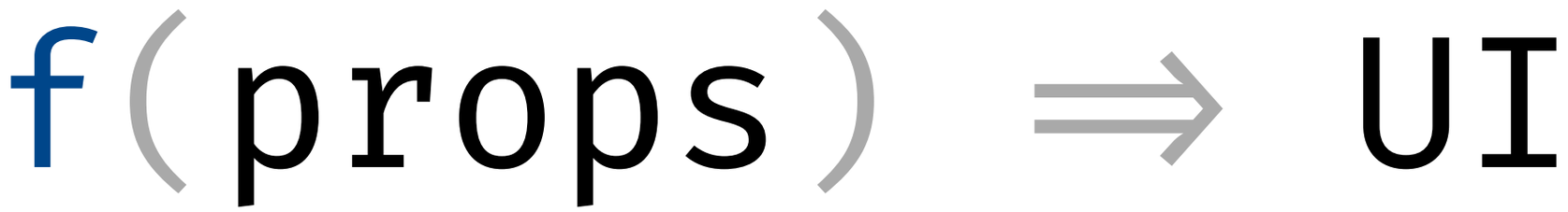
Primary

```
<Button primary>Primary</Button>
```

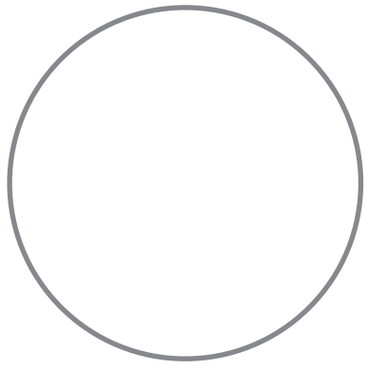
Anatomy of the component



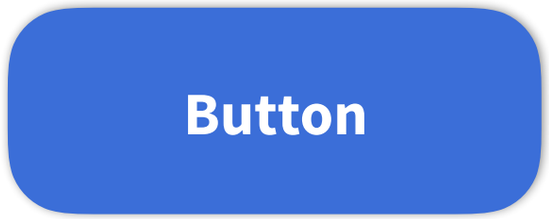
- Input
- State
- Behavior
- Styles


f(props) ⇒ UI

UI components should be *pure, self-contained & isolated*

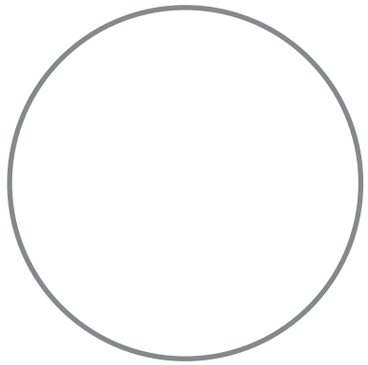


- Input
- State
- Behavior
- Styles



- Input
- State
- Behavior
- Styles

- Input
- State
- Behavior
- Styles

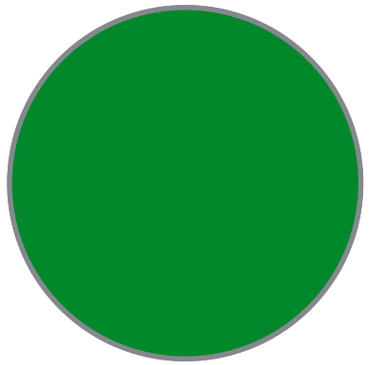


- Input
- State
- Behavior
- Styles

- Input
- State
- Behavior
- Styles

- Input
- State
- Behavior
- Styles

and yet...



- Input
- State
- Behavior

Button

- Input
- State
- Behavior

- Input
- State
- Behavior

Global styles

CSS is to blame?


CSS

Problems with CSS on scale

- ⛔ Disconnected from component’s code
- ⛔ Implicit dependencies
- ⛔ No styles isolation
- ⛔ Constants sharing
- ⛔ Minification

What can *we* do about it?

Where is my CSS?

```jsx
const Button = ({ children }) => {
  return (
    <button
      type="button"
      className="btn"
    >
      <span className="label">
       { children }
      </span>
    </button>
  )
}
```

Where is my CSS?

```
1  const Button = ({ children }) ⇒ {
2    return (
3      <button
4        type="button"
5        className="btn"
6      >
7        <span className="label">
8         { children }
9        </span>
10     </button>
11   )
12 }
```

Where is this class coming from?

application.css

application.css

```
21    clear: both;
22  }
23  .hide-text {
24    font: 0/0 a;
25    color: transparent;
26    text-shadow: none;
27    background-color: transparent;
28    border: 0;
29  }
30  .input-block-level {
31    display: block;
32    width: 100%;
33    min-height: 30px;
34    -webkit-box-sizing: border-box;
35    -moz-box-sizing: border-box;
36    box-sizing: border-box;
37  }
38  /*------------------------------------------------------------
39
40  [Typography Styles]
41
42  ------------------------------------------------------------*/
43  h1,
44  h2,
45  h3,
46  h4,
47  h5,
48  h6 {
49    margin-top: 0;
50    font-weight: 600;
51  }
```

Let’s search quickly...

Git branch: HEAD, index: 10?, working: 10?, Comparing against : HEAD, Line 1, Column 1 | Spaces: 2 | CSS

CSS Monolith!

application.css

1. Split code

Components > Files

```
public
├── images
│   └── icon.svg
├── javascripts
│   └── application.js
└── stylesheets
    └── application.css
```



```
public
├── images
│   └── icon.svg
├── javascripts
│   ├── Button.js
│   └── Dropdown.js
└── stylesheets
    ├── Buttons.css
    └── Dropdown.css
```

1. Split & co-locate

Separation of concerns?

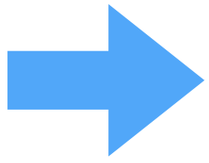
```
public
├── images
│   └── icon.svg
├── javascripts
│   ├── Button.js
│   └── Dropdown.js
└── stylesheets
    ├── Buttons.css
    └── Dropdown.css
```

Separation of ~~concerns~~ technologies

```
public
├── images
│   └── icon.svg
├── javascripts
│   ├── Button.js
│   └── Dropdown.js
└── stylesheets
    ├── Buttons.css
    └── Dropdown.css
```

Separation of concerns

```
public
├── images
│   └── icon.svg
├── javascripts
│   ├── Button.js
│   └── Dropdown.js
└── stylesheets
    ├── Buttons.css
    └── Dropdown.css
```



```
components
├── Button
│   ├── Button.css
│   ├── Button.js
│   ├── Button.spec.js
│   ├── icon.svg
│   └── index.js
└── Dropdown
```

2. Dependencies

Explicit dependencies

```
1  import './Button.css'
2
3  const Button = ({ children, disabled, primary }) => {
4    let classNames = 'btn'
5    if (disabled) classNames += ' btn-disabled'
6    if (primary) classNames += ' btn-primary'
7    return (
8      <button
9        type="button"
10       disabled={disabled}
11       className={classNames}
12     >
13       <span className="btn_label">
14        { children }
15       </span>
16     </button>
17   )
18 }
19 export default Button
```

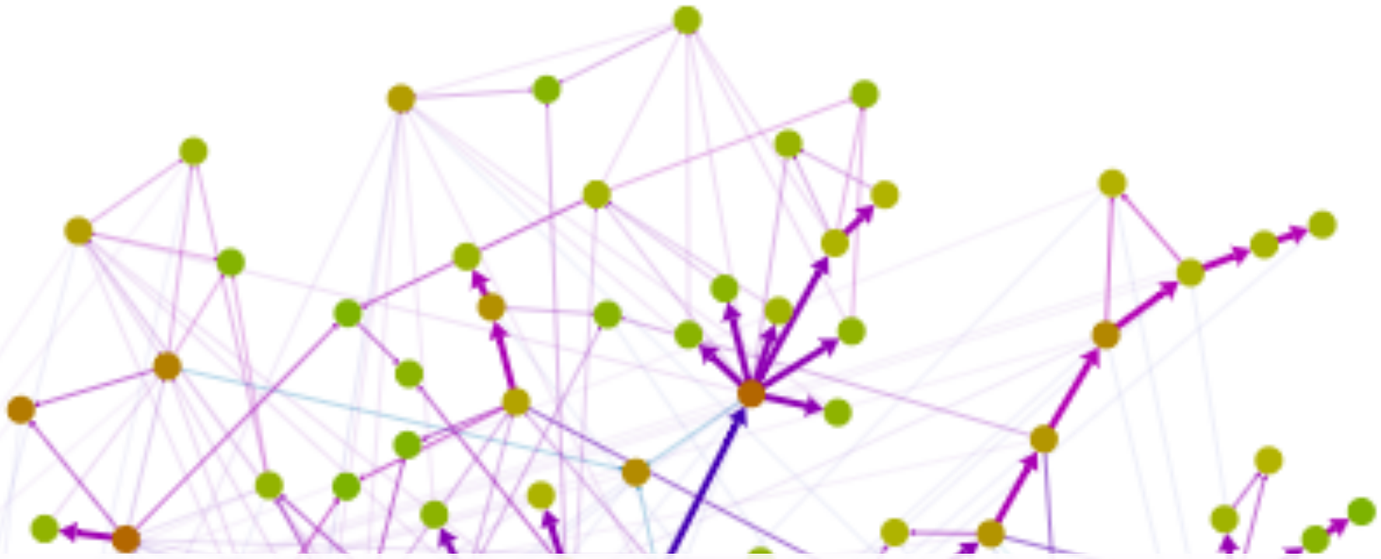

AH, NOT SO FAST.

1. We can’t require CSS in JS

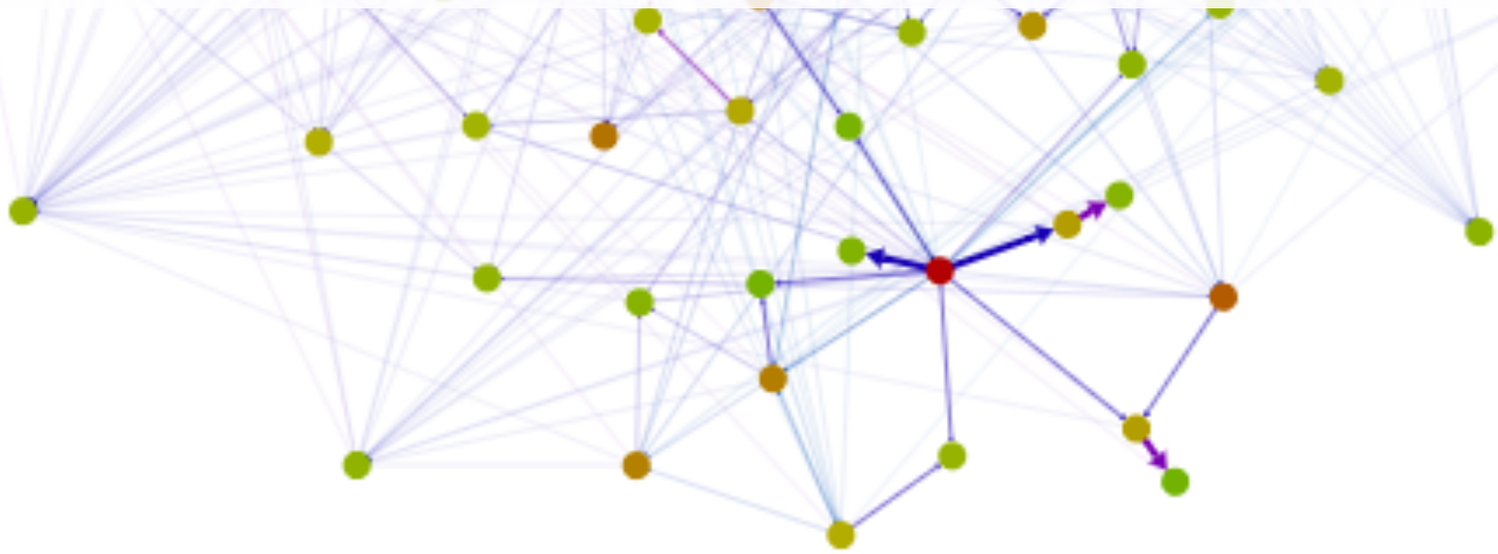
2. Browsers expect `<link>` and `<script>` tags

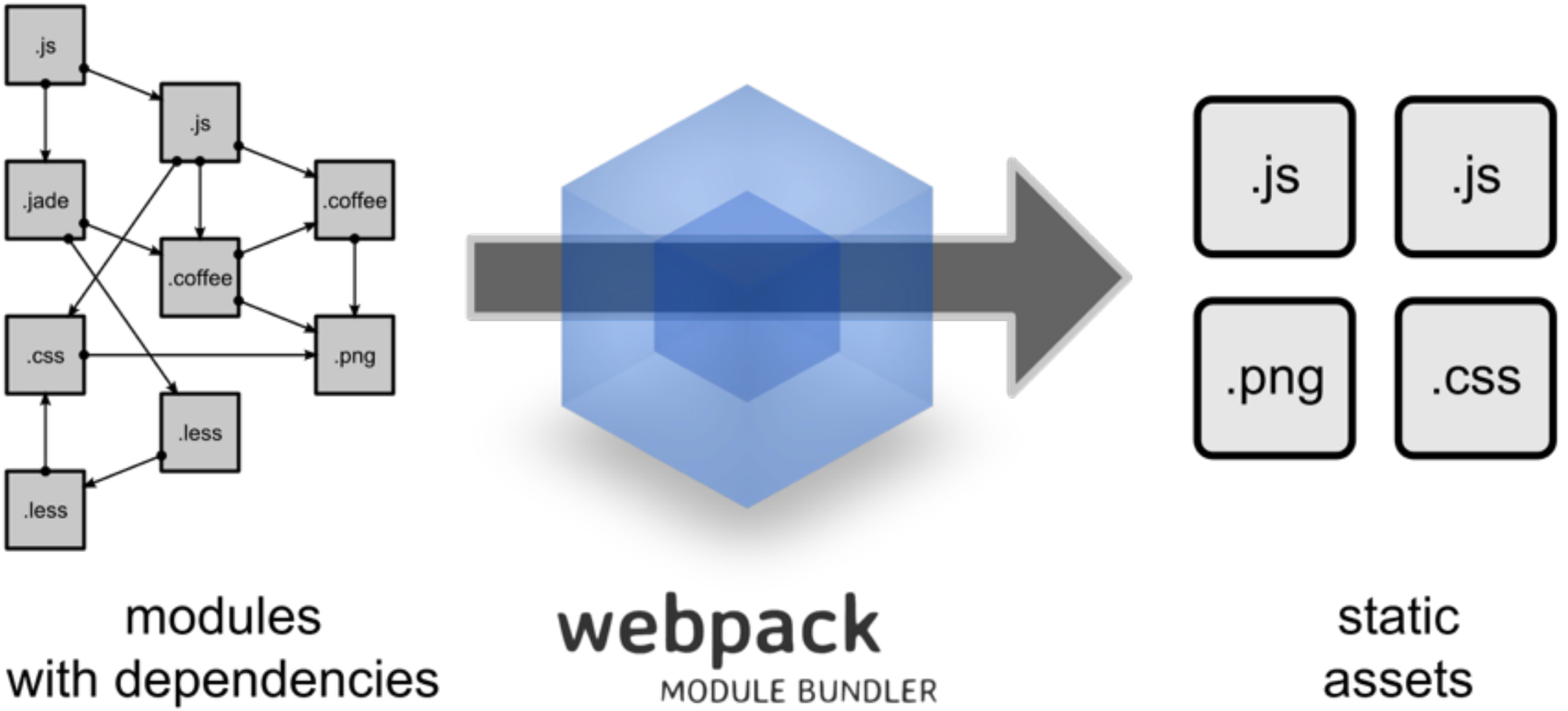
3. We should minimize the number of `*.css` and `*.js` requests

Common build tools operate on *file trees*



Dependency graph FTW!





http://webpack.github.io/

Allow requiring CSS with Webpack

```
module.exports = {
  module: {
    loaders: [
      { test: /\.css$/, loader: "style-loader!css-loader" },
      { test: /\.png$/, loader: "url-loader?limit=10000" },
      { test: /\.js$/, loader: "babel-loader" }
    ]
  }
};
```

- Load CSS files,
- Embed PNG images under 10Kb as Data URIs
- Transpile JS files with Babel

Now we can just import CSS!

```
1  import './Button.css'
2
3  const Button = ({ children, disabled, primary }) => {
4    let classNames = 'btn'
5    if (disabled) classNames += ' btn-disabled'
6    if (primary) classNames += ' btn-primary'
7    return (
8      <button
9        type="button"
10       disabled={disabled}
11       className={classNames}
12     >
13       <span className="btn_label">
14        { children }
15       </span>
16     </button>
17   )
18 }
19 export default Button
```

Tips & Tricks

✅ Split your code by components

✅ One directory per UI component

✅ One CSS file per UI component

✅ Explicit dependencies

✅ Co-locate JS, CSS and assets

How are we doing so far?

✅ Disconnected from component's code

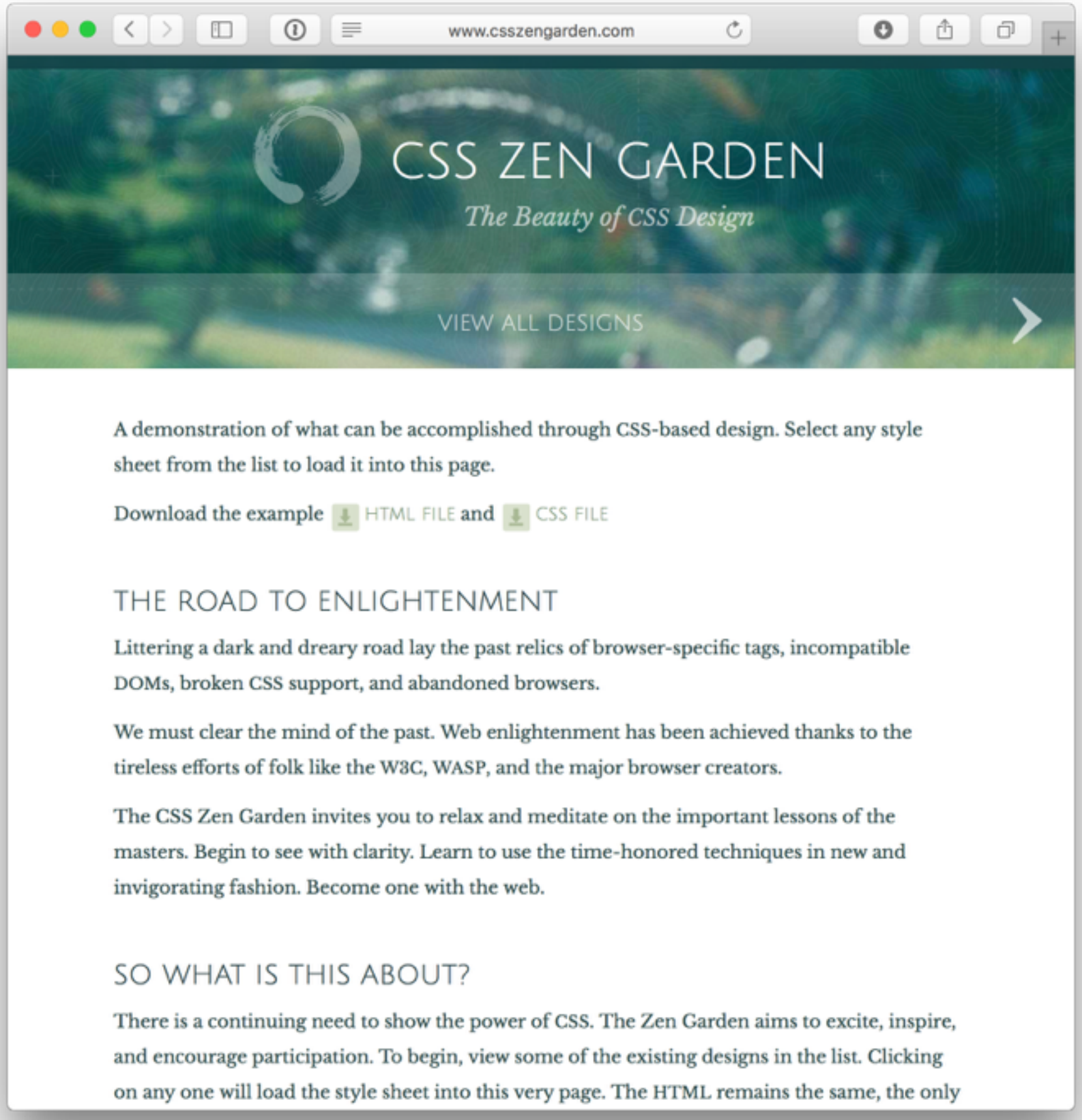
✅ Implicit dependencies

🚫 No styles isolation

🚫 Constants sharing

🚫 Minification

3. Styles isolation


www.csszengarden.com
CSS ZEN GARDEN
The Beauty of CSS Design
VIEW ALL DESIGNS
A demonstration of what can be accomplished through CSS-based design. Select any style sheet from the list to load it into this page.
Download the example HTML FILE and CSS FILE
THE ROAD TO ENLIGHTENMENT
Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, broken CSS support, and abandoned browsers.
We must clear the mind of the past. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WASP, and the major browser creators.
The CSS Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the time-honored techniques in new and invigorating fashion. Become one with the web.
SO WHAT IS THIS ABOUT?
There is a continuing need to show the power of CSS. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The HTML remains the same, the only

www.csszengarden.com/218/

no. 218

5.23 fl oz

DR. SHEA'S MIRACULOUS

CSS Zen Garden

THE BEAUTY OF CSS DESIGN

A demonstration of what can be accomplished through CSS-based design. Select any style sheet from the list to load it into

www.csszengarden.com/220/

CZG

MADE LOCALLY

CSS ZEN GARMENTS

IMPECCABLE QUALITY

A DEMONSTRATION OF WHAT CAN BE ACCOMPLISHED THROUGH CSS-BASED DESIGN. SELECT ANY STYLE SHEET FROM THE LIST TO LOAD IT INTO THIS PAGE.

www.csszengarden.com/221/

CSS Zen Garden

The Beauty of CSS Design

A demonstration of what can be accomplished through CSS-based design. Select any style sheet from the list to load it into this page.

Download the example html file and css file

“CSS allows complete and total control over the style of a hypertext document…”

– http://www.csszengarden.com/

CSS was designed for documents,

not web-applications

CSS = Cascading Style Sheets

“Three main sources of style information form a cascade. […] The user's style modifies the browser's default style. The document author's style then modifies the style some more.”

– MDN

Designing with cascade is like extending the DOM with JavaScript!

“DOM extension seemed so temptingly useful that few years ago, Prototype Javascript library made it an essential part of its architecture. But what hides behind seemingly innocuous practice is a huge load of trouble. […] DOM extension is one of the biggest mistakes Prototype.js has ever done.”

– by @kangax
http://perfectionkills.com/whats-wrong-with-extending-the-dom/

Debugging CSS

BEM

https://en.bem.info/

Naming convention

methodology

Working with BEM entities requires a knowledge of their naming rules.

The main idea of the naming convention is to make names of CSS selectors as informative and clear as possible. This will help make code development and debugging easier and also solve some of the problems faced by web developers.

Let's say we have a CSS selector named `menuitemvisible`. A quick view of such a notation doesn't allow us to identify types of BEM entities from the name of the selector.

Using BEM

```
import './Button.css'

const Button = ({ children, disabled, primary }) => {
  let classNames = 'Button'
  if (disabled) classNames += ' Button--disabled'
  if (primary) classNames += ' Button--primary'
  return (
    <button
      type="button"
      disabled={disabled}
      className={classNames}
    >
      <span className="Button__label">
       { children }
      </span>
    </button>
  )
}
export default Button
```

Using BEM

```
/* Button.css */

.Button { /* general rules */ }
.Button--disabled { /* disabled rules */ }
.Button--primary { /* primary rules */ }
.Button__label { /* label rules */ }
```

BEM

✅ Global namespace

✅ Cascade*

✅ No styles isolation

🚫 Not beginner friendly

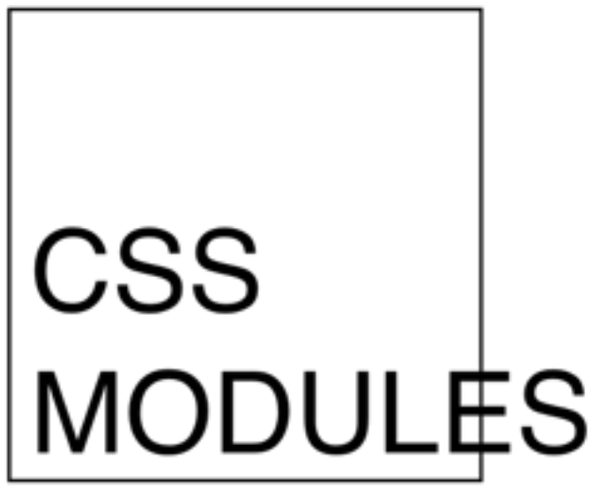
🚫 Very verbose

🚫 Requires discipline

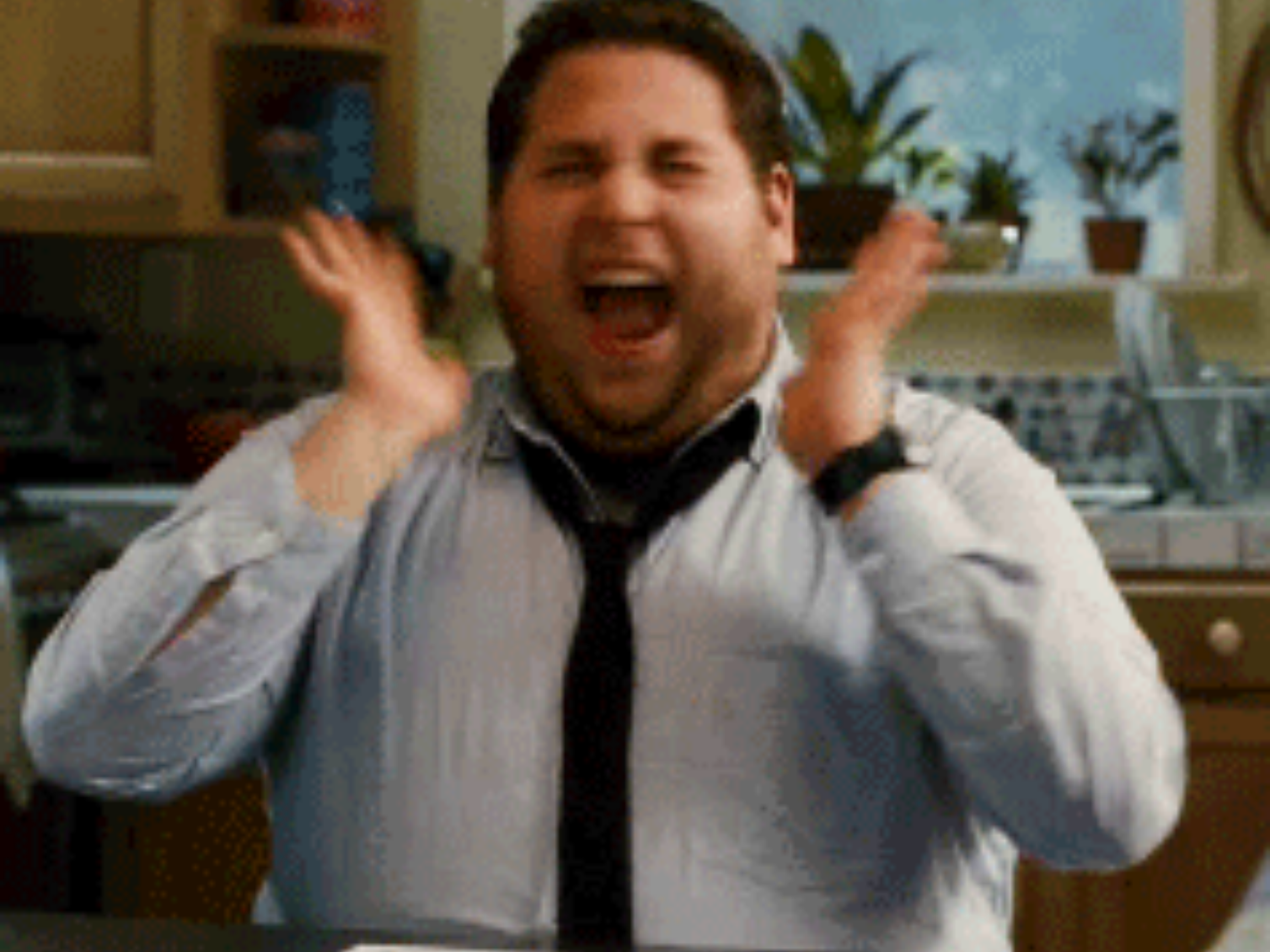
🚫 Minification

@markdalgleish

CSS-modules



https://github.com/css-modules/css-modules



How CSS-modules work

Enabling CSS-modules

```
// webpack.config.js

module.exports = {
  module: {
    loaders: [
      {
        test: /\.css$/,
        loader: "css"
      }
    ]
  }
};
```

Enabling CSS-modules

```
1  // webpack.config.js
2
3  module.exports = {
4    module: {
5      loaders: [
6        {
7          test: /\.css$/,
8          loader: "css?modules"
9        }
10     ]
11   }
12 };
```

Before: BEM-style

```
1 /* Button.css */
2
3 .Button { /* general rules */ }
4 .Button--disabled { /* disabled rules */ }
5 .Button--primary { /* primary rules */ }
6 .Button__label { /* label rules */ }
```

After: CSS-modules

```
1 /* Button.css */
2
3 .root { /* general rules */ }
4 .disabled { /* disabled rules */ }
5 .primary { /* primary rules */ }
6 .label { /* label rules */ }
```

Before: BEM-style

```
import './Button.css'

const Button = ({ children, disabled, primary }) => {
  let classNames = 'Button'
  if (disabled) classNames += ' Button--disabled'
  if (primary) classNames += ' Button--primary'
  return (
    <button
      type="button"
      disabled={disabled}
      className={classNames}
    >
      <span className="Button__label">
       { children }
      </span>
    </button>
  )
}
export default Button
```

After: CSS-modules

```
import styles from './Button.css'

const Button = ({ children, disabled, primary }) => {
  let className = {styles.root}
  if (disabled) className = {styles.disabled}
  if (primary) className = {styles.primary}
  return (
    <button
      type="button"
      disabled={disabled}
      className={className}
    >
      <span className={styles.label}>
       { children }
      </span>
    </button>
  )
}
export default Button
```

The result

```
<button class="Button">
  <span class="Button__label">
    Click me!
  </span>
</button>
```

```
<button class="Button_root_3fslE">
  <span class="Button_label_I8bKh">
    Click me!
  </span>
</button>
```

The result

```
/* Button.css */

.root { /* general rules */ }
.disabled { /* disabled rules */ }
.primary { /* primary rules */ }
.label { /* label rules */ }
```

```
styles: {
  root: "Button__root__abc5436",
  disabled: "Button__disabled__def6547",
  primary: "Button__primary__1638bcd",
  label: "Button__label__5dfg462"
}
```

The result

```
import styles from './Button.css'

const Button = ({ children, disabled, primary }) => {
  let className = {styles.root}
  if (disabled) className = {styles.disabled}
  if (primary) className = {styles.primary}
  return (
    <button
      type="button"
      disabled={disabled}
      className={className}
    >
      <span className={styles.label}>
       { children }
      </span>
    </button>
  )
}
export default Button
```

Prettier class names in dev

```
// webpack.config.js

module.exports = {
  module: {
    loaders: [
      {
        test: /\.css$/,
        loader: "css?modules
            &localIdentName=[name]__[local]__[hash:base64:5]"

      }
    ]
  }
};
```

Prettier class names in dev

```
import styles from './Button.css'

/*
styles = {
  button: ‘Button__root__3fslE’,
  label: ‘Button__label__I8bKh’
}
*/
```

name *local* *hash*

CSS
MODULES

BEM for free!

4. Composition & Shared constants

Composition with Sass

```
/* Button.css */

.Button { /* common rules */ }
.Button--disabled { /* disabled rules */ }
.Button--primary { /* primary rules */ }
```

Composition with Sass

```
.Button--common { /* common rules */ }
.Button--disabled {
  @extends .Button--common;
  /* gray color, light background */
}
.Button--primary {
  @extends .Button--common;
  /* white color, blue background */
}
```

Composition with Sass

```
.Button--common, .Button--disabled, .Button--primary {
  /* common rules */
}
.Button--disabled {
  /* gray color, light background */
}
.Button--primary {
  /* white color, blue background */
}
```

Composition with CSS-modules

```
1 /* Button.css */
2
3 .root { /* general rules */ }
4 .disabled { /* disabled rules */ }
5 .primary { /* primary rules */ }
```

Composition with CSS-modules

```
/* Button.css */

.root { /* general rules */ }
.disabled {
  composes: root;
  /* disabled rules */
}
.primary {
  composes: root;
  /* primary rules */
}
```

Composition with CSS-modules

```
styles: {
  root: "Button__root__abc5436",
  disabled: "Button__root__abc5436 Button__disabled__def6547",
  primary: "Button__root__abc5436 Button__primary__1638bcd"
}
```

Use only one class name!

```
/* Don't do this */
`class=${[styles.normal, styles['primary']].join(" ")}`

/* Using a single name makes a big difference */
`class=${styles['primary']}`

/* camelCase makes it even better */
`class=${styles.isPrimary}`
```

The result

```
<button class=“Button Button--primary">
  <span class=“Button__label”>
    Click me!
  </span>
</button>
```

```
<button class=“Button_root_3fslE Button_primary_Hf415”>
  <span class="Button_label_I8bKh">
    Click me!
  </span>
</button>
```

Compose from other files!

```css
/* colors.css */
.primary {
  background-color: #4399fa;
}
.secondary {
  background-color: #999;
}
```

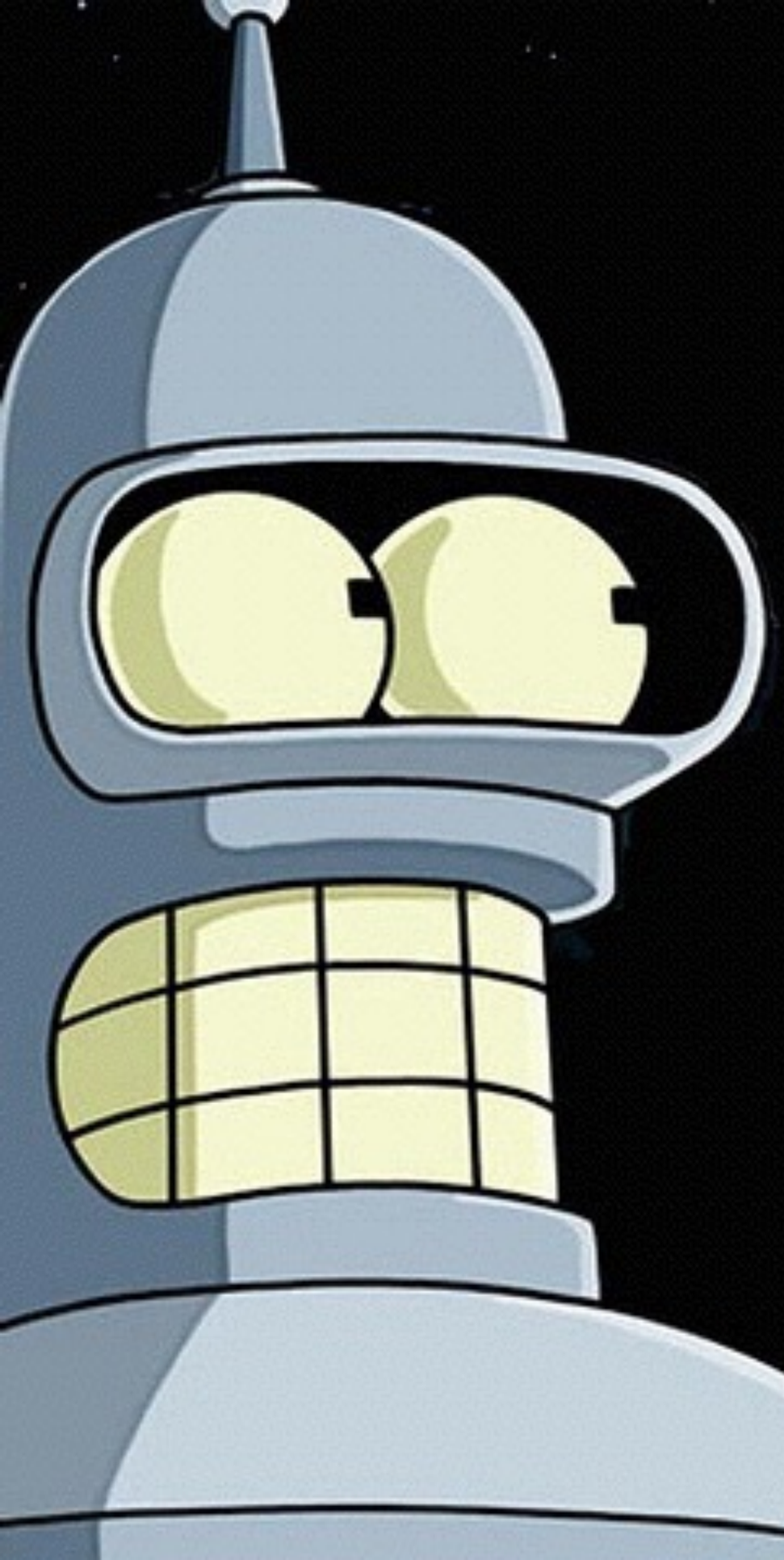
```css
/* Button.css */
.root { /* general rules */ }
.primary {
  composes: root;
  composes: primary from '../colors.css';
  /* primary rules */
}
```

Or use variables!

```
/* variables.css */
@value small: (max-width: 599px);

/* component.css */
@value small from "./variables.css";

.pageContent {
  background: green;
  @media small {
    background: red;
  }
}
```



Kill All Humans For A Better TOMORROW

Bender B. Rodríguez

5. Minification

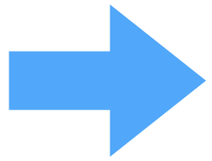
Minification

```
// webpack.config.js

module.exports = {
  module: {
    loaders: [
      {
        test: /\.css$/,
        loader: "css?modules"
      }
    ]
  }
};
```

Minification

```
<button class="button">
  <span class="label">
    Click me!
  </span>
</button>
```



```
<button class="1s6s23ca312">
  <span class="sdp9423cadg">
    Click me!
  </span>
</button>
```

Solved with CSS-modules

✅ Disconnected from component’s code

✅ Implicit dependencies

✅ No styles isolation

✅ Constants sharing

✅ Minification

CSS-modules overview

Local by default
but allow exceptions

```
.local-class {
  color: red;
}

:global(.prefix-modal-open) .local-class {
  color: green;
}
```

Composition

```
.common {
  /* all the common styles you want */
}
.normal {
  composes: common;
  /* anything that only applies to Normal */
}
.disabled {
  composes: common;
  /* anything that only applies to Disabled */
}
```

Explicit dependencies

```
.otherClassName {
  composes: className from "./style.css";
}
```

Import variables

```
/* variables.css */
@value small: (max-width: 599px);

/* component.css */
@value small from "./breakpoints.scss";

.pageContent {
  background: green;
  @media small {
    background: red;
  }
}
```

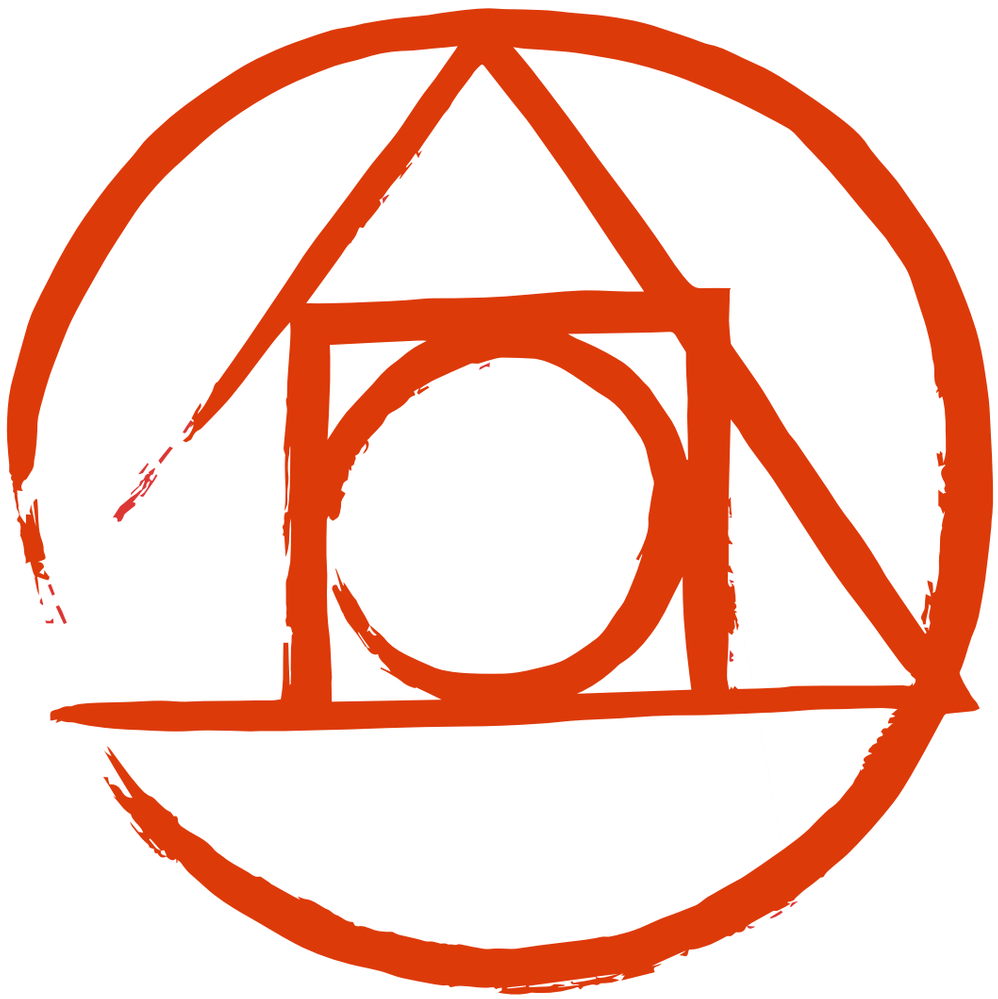
Use with pre-processors like Sass or Less

```
:global {
  .this-is-global {
    color: yellow;
  }
  .potential-collision-city {
    color: crap;
  }
}
```

Works on server, too!

https://github.com/css-modules/postcss-modules

FTW!

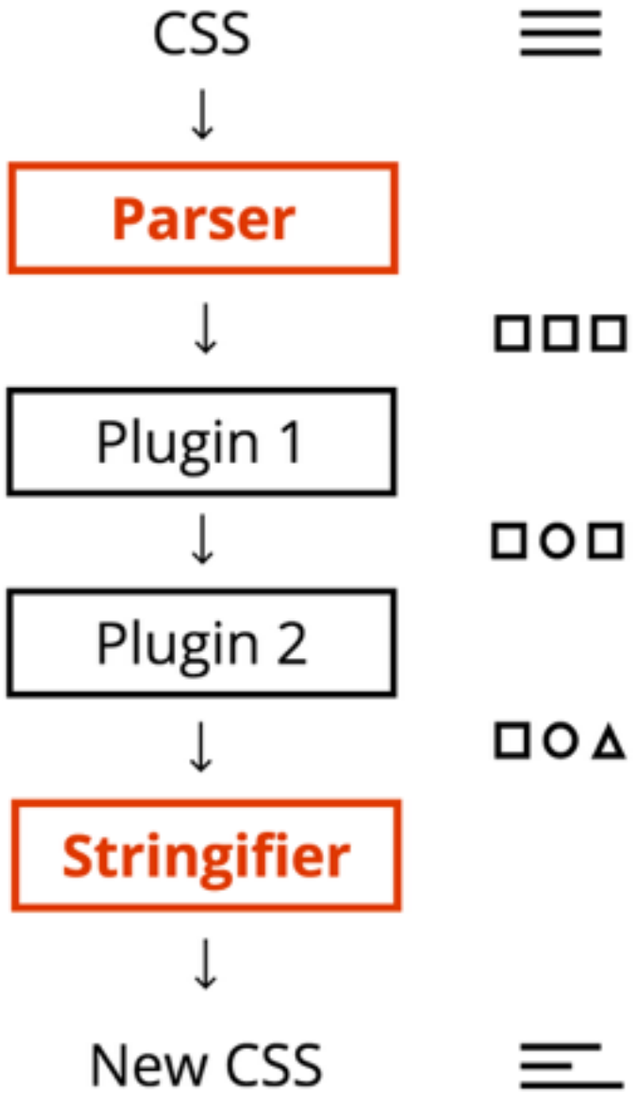


PostCSS

http://postcss.org

Like Babel, but for CSS

How PostCSS works



What’s possible?

Autoprefixer

```
:full-screen {
}
```

```
:-webkit-:full-screen {
}
:-moz-:full-screen {
}
:full-screen {
}
```

Sass/less syntax

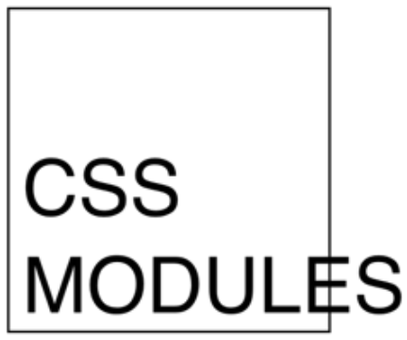
```
.block {
    &_title {
        font-size: 200%;
    }
}
```

```
.block_title {
    font-size: 200%;
}
```

Browser hacks

```
.foo {
  opacity: 0.8;
}
```

```
.foo {
  opacity: 0.8;
  filter: alpha(opacity=80)\9;
}
```



Local CSS

```
.name {
  color: gray;
}
```

```
.Logo__name__SVK0g {
  color: gray;
}
```

CSS next

```
:root {
  --red: #d33;
}
a {
  &:hover {
    color: color(var(--red) a(54%));
  }
}
```

```
a:hover {
  color: #dd3333;
  color: rgba(221, 51, 51, 0.54);
}
```

Stylelint

```
a {
  color: #4f;
}
```

```
visual.css
 2:12  ✖  Unexpected invalid hex color "#4f"
 4:1   ⚠  Expected ".foo.bar" to have a specificity no m
 6:13  ✖  Unexpected unit "px" for property "margin"
 7:17  ✖  Expected single space after "," in a single-li
```

PostCSS overview

1. More powerful than pre-processors thanks AST
2. ⚡Fast (a few times faster than Sass)
3. Lots of plugins and easy to write your own
4. Tools like linting or automatic properties sorting
5. IDE Support (WebStorm, Atom, Sublime)
6. Shared constants between CSS and JS
7. Works not only with SPAs or React
8. It’s just CSS*

How to setup

http://postcss.org

Example setup with Webpack

```
module.exports = {
  module: {
    loaders: [
      {
        test: /\.css$/,
        loader: ‘style!css&importLoaders=1!postcss-loader'
      }
    ]
  },
  postcss: function() {
    return [
      require('precss'),
      require('postcss-calc'),
      require('autoprefixer')({ browsers: ['last 2 version'] })
    ];
  }
};
```

Lint CSS as pre-commit hook

```
{
  "lint-staged": {
    "*.css": "stylelint"
  },
  "pre-commit": "lint-staged",
  "stylelint": {
    "extends": "stylelint-config-standard"
  }
}
```

https://github.com/okonet/lint-staged

http://postcss.parts/

by @mxstbr

Do I need it?

Do I need it?

✅ You work in a team

✅ You plan to update your CSS in the future

✅ You use third-party CSS and don’t want to break things

✅ You care about code readability and reusability

✅ You hate manual work

What about CSS-in-JS

Inline styles in JSX

- ✅ JS is more powerful
- ✅ Runtime re-calculation
- ✅ Dead code elimination
- 🚫 No pseudo-selectors / media queries
- 🚫 React.js-only
- 🚫 Harder to debug in DevTools
- 🚫 No IDE support
- 🚫 No tooling

CSS-in-JS

✅ JS is more powerful

✅ Shared variables

✅ Dead code elimination

🚫 No sourcemaps*

🚫 No IDE support*

🚫 No related tools*

Worth checking out

by @oleg008

by @mxstbr and @glennmaddern

“[…] I wouldn’t suggest using any CSS-in-JS tool over CSS Modules for decent-sized projects in the near term.”

– Glen Maddern
https://github.com/css-modules/css-modules/issues/187#issuecomment-257752790

Final notes

It’s not about writing code,
it’s about reading it.

Build isolated components.

Separate concerns,
not technologies.

Treat CSS seriously.

It’s here to stay.

Choose the right tool for the job.

Stay open-minded & keep experimenting!

Thank you!

Andrey Okonetchnikov

UI Engineer

@okonetchnikov

http://okonet.ru

https://github.com/okonet