The Issues of Using CSS

Global namespace - doesn't scale in complexity. Namespace conventions `:block_element__modifier` (submit_button__hover)

Dependency management - pathes has to be exact, can't manage the context of the dependencies (like node_modules require algorithm)

Dead code elimination - Hard to figure out what css classes are in use, even more so if your app does dynamic changes around your CSS (concat, etc..). You're forced to go over every HTML page to see what is actually being used.

Minification - Can't uglify class names, can't tell which classes you're using for semantics and which for design.

Sharing constants - Hard to achieve across files

```
:root{
   --myColor: red;
}

.button{
   color:var(myColor);
}
```

Css variables solves this a little, but there is no real way to share complex namespaces.

Non deterministic resolution

```
//style1.css
.blueish{
   color:blue
}

//style2.css
.reddish{
   color: red !important;
}

<div class='reddish bluish'>
      What color is this text?
</div>
```

Css overrides resolved by the order of declaration resolvement, `!important` declarations

Isolation
“Specificity battles” between component writer and style writer

Composition
```
.hoverRed{
    Color: #990000
}
```

# impossible: `.button:hover = .hoverRed`

SASS/LESS - help solve this issue, but can only be applied during build and not runtime
Can’t do something like:
```
.button{
      color:fn(timeOfDay)
}
```

CSS Modules provides some comfort in that it allows you to only compose classes that have been defined at that point in time, however once you revert to working with multiple files - you encounter the same issue again.

Using JS
Instead of :
```
//style.css
.reddish{
   color: red;
}

// button.js
<div className=’reddish’>
      What color is this text?
</div>
```

We can do:
```
// styles.js
export const reddish = {
   color: ‘red’;
}

// button.js
Import {reddish} from ‘yadayada/style/js’
<div style={reddish}>
```

```
      What color is this text?
</div>
```

The first 5 problems are solved automatically by moving to JS system.

Comparison of libs

- Webpack (using a CSS Loader)
- Glamor
- Styled Components
-

Webpack:

Using the following webpack config:

```
use: [
    { loader: "style-loader" },
    { loader: "css-loader" }
]
```

```
import '../site/styles.css';
```

Instead of

```
<link rel="stylesheet" href="styles.css">
```

Solves mostly the synchronous loading problem.
You can now simply call classes as though they were loaded using a regular `link` tag:

```
<div className=”angry”>
   What up homie?
</div>
```

Alternatively using a different config:
In webpack.config:
loader:
'style-loader!css-loader?modules&importLoaders=1&localIdentName=[name]__[local]___[hash:base64:5]'

```
import styles from '../site/styles.css';
```

Then you can use:

```
<div className={styles.angry}>
   What up homie?
</div>
```

Solves the global namespace and dependency problem

Glamor

```
// label is used for debugging purposes
let angry = css({
  label: 'helper name',
  color: 'red'
});

let otherRed = css({
  label: 'helper name',
  color: 'red',
});
```

Glamor merges identical classes, only one of the classes above will be loaded and used wherever either one was called. Classes generated are automatically scoped.

You could even use css states:

```
let angry = css({
  label: 'helper name',
  ':hover': {
    color: 'blue'
  },
  color: 'red'
});
```

You can also use nested rules:

```
let angry = css({
  ':hover': {
    color: 'blue'
  },
  color: 'red',
  '& span': {
    color: 'green',
    ':hover': {
      color: 'gold'
    }
  }
});

<div className={angry}>
  what up, homies! // this is red, blue on hover
  <span>what up, homies!</span> // this is green, gold on hover
```

```
</div>
```

You can also define your classes directly in the className definition, this causes the order of parameters given to determine the order of application:

```
// the blueish class will be the last applied
<div className={css(reddish, blueish)}>
  what up, homies!
</div>
```

There’s a simulate function in glamor that let's you force pseudo states such as `hover` and `active`. Amazing for debugging, testing etc.

```
import { css, simulate} from 'glamor'
...
<div className={angry} ...simulate(‘hover’)>
```

Classes that are duplicates of each other in terms of content, but have their properties applied in different order do not get merged (this is a bug?)

```
// These classes will not be merged
let reddish = css({
  color: 'red',
  backgroundColor: 'white'
});

let redder = css({
  backgroundColor: 'white',
  color: 'red'
});
```

Styled Components

Uses tagged template literals to take a set of rules and return a stateless functional component.

```
import styled from 'styled-components';
...
const StyledDiv = styled.div`
Color:red;
backgroundColor:${props => props.bg || 'white'};
&:hover:{
  color:blue
}
`;

<StyledDiv bg='grey'}>
   what up, homies!
</StyledDiv>
```

Note: missing a semicolon on a prop declaration in a rule, causes that prop to not be applied

```
const StyledDiv = styled.div`
   color: red // missing semi-colon, this will not be applied
`;
```

Styled-jsx

Add to .babelrc

```
"plugins": [
  "styled-jsx/babel"
]
```

Then without further imports use (transpilers huzzah!)

```
<div className="reddish">
<style jsx>{`
   .reddish{
      color: red;
   }
   .reddish:hover {
      color: blue;
    }
`}</style>
   what up, homies!
</div>
```

Note: Missing semicolons do not break properties applied in rules!

Other Options

There are many more available libraries depending your needs & preferences (performance, server-side rendering, what trends you prefer to follow, etc, each has it's own virtues and flaws...)
https://github.com/MicheleBertoli/css-in-js

Resources

https://github.com/JedWatson/classnames

Exercise repo:
https://github.com/threepointone/css-in-js-workshop

Glamor:
https://github.com/threepointone/glamor

Glam (new experimental version of Glamor that works not just in runtime):
https://github.com/threepointone/glam

How Glam Works:
https://gist.github.com/threepointone/0ef30b196682a69327c407124f33d69a

Benchmark:
https://github.com/A-gambit/CSS-IN-JS-Benchmarks/blob/master/RESULT.md

CSSStyleSheet.insertRule():
https://developer.mozilla.org/en-US/docs/Web/API/CSSStyleSheet/insertRule

Weak Maps usage in Glamor:
https://github.com/threepointone/glamor/blob/f4fce31163d0002331819fdb66a77e998ba6825d/docs/weakmaps.md

Polished - lightweight toolset for writing styles in JavaScript (by styled-components):
https://github.com/styled-components/polished

Tachyons -
http://tachyons.io/