Airbnb CSS-in-JavaScript Style Guide

A mostly reasonable approach to CSS-in-JavaScript

Table of Contents

Naming

- Use camelCase for object keys (i.e. "selectors").

  Why? We access these keys as properties on the `styles` object in the component, so it is most convenient to use camelCase.

```
// bad
{
  'bermuda-triangle': {
    display: 'none',
  },
}

// good
{
  bermudaTriangle: {
    display: 'none',
  },
}
```

- Use an underscore for modifiers to other styles.

  Why? Similar to BEM, this naming convention makes it clear that the styles are intended to modify the element preceded by the underscore. Underscores do not need to be quoted, so they are preferred over other characters, such as dashes.

```
// bad
{
  bruceBanner: {
    color: 'pink',
    transition: 'color 10s',
  },

  bruceBannerTheHulk: {
    color: 'green',
```

```
  },
}

// good
{
  bruceBanner: {
    color: 'pink',
    transition: 'color 10s',
  },

  bruceBanner_theHulk: {
    color: 'green',
  },
}
```

- Use `selectorName_fallback` for sets of fallback styles.

  Why? Similar to modifiers, keeping the naming consistent helps reveal the relationship of these styles to the styles that override them in more adequate browsers.

```
// bad
{
  muscles: {
    display: 'flex',
  },

  muscles_sadBears: {
    width: '100%',
  },
}

// good
{
  muscles: {
    display: 'flex',
  },

  muscles_fallback: {
    width: '100%',
  },
}
```

- Use a separate selector for sets of fallback styles.

  Why? Keeping fallback styles contained in a separate object clarifies their purpose, which improves readability.

```
// bad
```

```
{
  muscles: {
    display: 'flex',
  },

  left: {
    flexGrow: 1,
    display: 'inline-block',
  },

  right: {
    display: 'inline-block',
  },
}

// good
{
  muscles: {
    display: 'flex',
  },

  left: {
    flexGrow: 1,
  },

  left_fallback: {
    display: 'inline-block',
  },

  right_fallback: {
    display: 'inline-block',
  },
}
```

- Use device-agnostic names (e.g. “small”, “medium”, and “large”) to name media query breakpoints.

  > Why? Commonly used names like “phone”, “tablet”, and “desktop” do not match the characteristics of the devices in the real world. Using these names sets the wrong expectations.

```
// bad
const breakpoints = {
  mobile: '@media (max-width: 639px)',
  tablet: '@media (max-width: 1047px)',
  desktop: '@media (min-width: 1048px)',
};
```

```
// good
const breakpoints = {
  small: '@media (max-width: 639px)',
  medium: '@media (max-width: 1047px)',
  large: '@media (min-width: 1048px)',
};
```

Ordering

- Define styles after the component.

  > Why? We use a higher-order component to theme our styles, which is naturally used after the component definition. Passing the styles object directly to this function reduces indirection.

```
// bad
const styles = {
  container: {
    display: 'inline-block',
  },
};

function MyComponent({ styles }) {
  return (
    <div {...css(styles.container)}>
      Never doubt that a small group of thoughtful, committed citizens can
      change the world. Indeed, it's the only thing that ever has.
    </div>
  );
}

export default withStyles(() => styles)(MyComponent);

// good
function MyComponent({ styles }) {
  return (
    <div {...css(styles.container)}>
      Never doubt that a small group of thoughtful, committed citizens can
      change the world. Indeed, it's the only thing that ever has.
    </div>
  );
}

export default withStyles(() => ({
  container: {
    display: 'inline-block',
```

```
  },
}))(MyComponent);
```

Nesting

- Leave a blank line between adjacent blocks at the same indentation level.

  > Why? The whitespace improves readability and reduces the likelihood of merge conflicts.

```
// bad
{
  bigBang: {
    display: 'inline-block',
    '::before': {
      content: "''",
    },
  },
  universe: {
    border: 'none',
  },
}

// good
{
  bigBang: {
    display: 'inline-block',

    '::before': {
      content: "''",
    },
  },

  universe: {
    border: 'none',
  },
}
```

Inline

- Use inline styles for styles that have a high cardinality (e.g. uses the value of a prop) and not for styles that have a low cardinality.

  > Why? Generating themed stylesheets can be expensive, so they are best for discrete sets of styles.

```
// bad
export default function MyComponent({ spacing }) {
```

```
  return (
    <div style={{ display: 'table', margin: spacing }} />
  );
}

// good
function MyComponent({ styles, spacing }) {
  return (
    <div {...css(styles.periodic, { margin: spacing })} />
  );
}
export default withStyles(() => ({
  periodic: {
    display: 'table',
  },
}))(MyComponent);
```

Themes

- Use an abstraction layer such as react-with-styles that enables theming. *react-with-styles gives us things like `withStyles()`, `ThemedStyleSheet`, and `css()` which are used in some of the examples in this document.*

  Why? It is useful to have a set of shared variables for styling your components. Using an abstraction layer makes this more convenient. Additionally, this can help prevent your components from being tightly coupled to any particular underlying implementation, which gives you more freedom.

- Define colors only in themes.

```
// bad
export default withStyles(() => ({
  chuckNorris: {
    color: '#bada55',
  },
}))(MyComponent);

// good
export default withStyles(({ color }) => ({
  chuckNorris: {
    color: color.badass,
  },
}))(MyComponent);
```

- Define fonts only in themes.

```
// bad
```

```
export default withStyles(() => ({
  towerOfPisa: {
    fontStyle: 'italic',
  },
}))(MyComponent);

// good
export default withStyles(({ font }) => ({
  towerOfPisa: {
    fontStyle: font.italic,
  },
}))(MyComponent);
```

- Define fonts as sets of related styles.

```
// bad
export default withStyles(() => ({
  towerOfPisa: {
    fontFamily: 'Italiana, "Times New Roman", serif',
    fontSize: '2em',
    fontStyle: 'italic',
    lineHeight: 1.5,
  },
}))(MyComponent);

// good
export default withStyles(({ font }) => ({
  towerOfPisa: {
    ...font.italian,
  },
}))(MyComponent);
```

- Define base grid units in theme (either as a value or a function that takes a multiplier).

```
// bad
export default withStyles(() => ({
  rip: {
    bottom: '-6912px', // 6 feet
  },
}))(MyComponent);

// good
export default withStyles(({ units }) => ({
  rip: {
    bottom: units(864), // 6 feet, assuming our unit is 8px
  },
}))(MyComponent);
```

```
// good
export default withStyles(({ unit }) => ({
  rip: {
    bottom: 864 * unit, // 6 feet, assuming our unit is 8px
  },
}))(MyComponent);
```

- Define media queries only in themes.

```
// bad
export default withStyles(() => ({
  container: {
    width: '100%',

    '@media (max-width: 1047px)': {
      width: '50%',
    },
  },
}))(MyComponent);

// good
export default withStyles(({ breakpoint }) => ({
  container: {
    width: '100%',

    [breakpoint.medium]: {
      width: '50%',
    },
  },
}))(MyComponent);
```

- Define tricky fallback properties in themes.

  > Why? Many CSS-in-JavaScript implementations merge style objects together which makes specifying fallbacks for the same property (e.g. `display`) a little tricky. To keep the approach unified, put these fallbacks in the theme.

```
// bad
export default withStyles(() => ({
  .muscles {
    display: 'flex',
  },

  .muscles_fallback {
    'display ': 'table',
  },
```

```
}))(MyComponent);

// good
export default withStyles(({ fallbacks }) => ({
  .muscles {
    display: 'flex',
  },

  .muscles_fallback {
    [fallbacks.display]: 'table',
  },
}))(MyComponent);

// good
export default withStyles(({ fallback }) => ({
  .muscles {
    display: 'flex',
  },

  .muscles_fallback {
    [fallback('display')]: 'table',
  },
}))(MyComponent);
```

- Create as few custom themes as possible. Many applications may only have one theme.
- Namespace custom theme settings under a nested object with a unique and descriptive key.

```
// bad
ThemedStyleSheet.registerTheme('mySection', {
  mySectionPrimaryColor: 'green',
});

// good
ThemedStyleSheet.registerTheme('mySection', {
  mySection: {
    primaryColor: 'green',
  },
});
```

CSS puns adapted from Saijo George.