Lesson 3 - Styling React Components

Inline CSS

```
const ownStyles = {
  display: 'inline-block',
  verticalAlign: 'middle',
  margin: 0,
  padding: '10px 20px',
  border: 0,
  borderRadius: 2,
  color: '#ffffff',
  fontSize: 14,
  fontFamily: 'inherit',
  textTransform: 'uppercase',
  backgroundColor: '#2196f3',
};

const Button = ({ text, disabled }) => (
  <button
    style={{
      ...ownStyles,
      backgroundColor: disabled ? '#0000001f' : '#2196f3',
      color: disabled ? '#00000042' : '#ffffff',
    }}>
    {text}
  </button>
);
```

- Плохая производительность
- Не масштабируется
- Ограниченные возможности (псевдоэлементы, медиа и т.п.)
- Сложно дебажить в DevTools
- Практически нет поддержки в IDE
- Нет инструментов

External CSS

```
/* Button.css */
.Button {
  display: inline-block;
  vertical-align: middle;
  margin: 0;
  padding: 10px 20px;
  border: 0;
  border-radius: 2px;
  color: #ffffff;
  font-size: 14px;
  font-family: inherit;
  text-transform: uppercase;
  background-color: #2196f3;
```

```
  transition: background-color 250ms cubic-bezier(0.4, 0, 0.2, 1) 0ms;
}

.Button:hover,
.Button:focus {
  background-color: #1976d2;
}

.Button--disabled {
  background-color: #0000001f;
  color: #00000042;
}
```

```
// Button.jsx
import './Button.css';

const Button = ({ text, disabled }) => {
  const btnClass = `Button ${disabled && 'Button--disabled'}`;

  return <button className={btnClass}>{text}</button>;
};
```

- Не масштабируется
- Глобальное пространство имен
- Глубокая вложенность селекторов (без BEM)
- Отсутсвует удаление мертвого кода

CSS Modules

Проблему глобального пространсва имен решают CSS-модули. Для того чтобы добавить их поддержку в `create-react-app`, необходимо распаковать сборку.

В консоли выполним следующую команду. После чего сборка `create-react-app` будет распакована в корневую директорию.

```
npm run eject
```

или

```
yarn eject
```

Далее заходим в папку `config`.

В файле `webpack.config.dev.js` находим `css-loader` и заменяем оригинальные настройки

```
{
  loader: require.resolve('css-loader'),
```

```
  options: {
    importLoaders: 1,
  },
},
```

на

```
{
  loader: require.resolve('css-loader'),
  options: {
    importLoaders: 1,
      modules: true,
      localIdentName: '[name]__[local]__[hash:base64:5]',
  },
},
```

В файле `webpack.config.prod.js` находим `css-loader` и заменяем оригинальные настройки

```
 {
  loader: require.resolve('css-loader'),
  options: {
    importLoaders: 1,
    minimize: true,
    sourceMap: shouldUseSourceMap,
  },
},
```

на

```
{
  loader: require.resolve('css-loader'),
  options: {
    importLoaders: 1,
    modules: true,
    localIdentName: '[name]__[local]__[hash:base64:5]',
    minimize: true,
    sourceMap: shouldUseSourceMap,
  },
},
```

Использование очень напоминает обычный внешний CSS, за тем исключением, что все имена классов инкапсулируются в компоненте. Таким образом в результирующем CSS получаем уникальное имя класса.

CSS-модули производят мапинг классов из CSS-файла в объект с ключами по имени класса.

```
/* Button.css */
.button {
```

```
  display: inline-block;
  vertical-align: middle;
  margin: 0;
  padding: 10px 20px;
  border: 0;
  border-radius: 2px;
  color: #ffffff;
  font-size: 14px;
  font-family: inherit;
  text-transform: uppercase;
  background-color: #2196f3;
  transition: background-color 250ms cubic-bezier(0.4, 0, 0.2, 1) 0ms;
}

.button:hover,
.button:focus {
  background-color: #1976d2;
}

.disabled {
  background-color: #0000001f;
  color: #00000042;
}
```

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

const Button = ({ text, disabled }) => {
  const btnClass = `${styles.button} ${disabled && styles.disabled}`;

  return <button className={cls}>{text}</button>;
};
```

composes

В CSS-модулях есть интереная фича, можно делать композицию стилей прямо в CSS-файле, как `@extends` в `SASS`.

```
/* Button.css */
.button {
  display: inline-block;
  vertical-align: middle;
  margin: 0;
  padding: 10px 20px;
  border: 0;
  border-radius: 2px;
  color: #ffffff;
  font-size: 14px;
  font-family: inherit;
  text-transform: uppercase;
  background-color: #2196f3;
  transition: background-color 250ms cubic-bezier(0.4, 0, 0.2, 1) 0ms;
}
```

```
.button:hover,
.button:focus {
  background-color: #1976d2;
}

.disabled {
  composes: button;
  background-color: #0000001f;
  color: #00000042;
}
```

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

const Button = ({ text, disabled }) => {
  const btnClass = disabled ? styles.disabled : styles.button;

  return <button className={btnClass}>{text}</button>;
};
```

Можно делать композицию со стилями из других файлов.

```
/* colors.css */
.primary {
  background-color: #2196f3;
}

/* Button.css */
.button {
  composes: primary from 'path/to/colors.css';
}
```

А еще в CSS-модулях есть переменные.

```
/* variables.css */
@value primary-font-stack: Roboto, sans-serif;

/* Button.css */
@value primary-font-stack from 'path/to/variables.css';

.button {
  font-family: primary-font-stack;
}
```

classnames package

Для слияния имен CSS-классов можно написать собсвенную функцию или использовать готовое решение.

Пакет `classnames` это простая утилита для создания результирующего имени класса по

условию.

```
npm i classnames
```

или

```
yarn add classnames
```

Если не используются CSS-модули:

```
// Button.jsx
import classNames from 'classnames';
import './Button.css';

const Button = ({ text, disabled }) => {
  const btnClass = classNames({
    Button: true,
    'Button--disabled': disabled,
  });

  return <button className={btnClass}>{text}</button>;
};
```

В случае с CSS-модулями:

```
// Button.jsx
import classNames from 'classnames/bind';
import styles from './Button.css';

const cx = classNames.bind(styles);

const Button = ({ text, disabled }) => {
  const btnClass = cx({
    button: true,
    disabled: disabled,
  });

  return <button className={btnClass}>{text}</button>;
};
```

classnames on npm

Styled components

styled-components.com

Materials

- Sarrah Vesselov: How to style in React and not lose friends
- Maintainable CSS in React
- Get That CSS Out Of My JavaScript
- Maintainable CSS in React
- CSS Modules — Solving the challenges of CSS at scale
- CSS-in-JS Roundup: Styling React Components
- 5 UI Libraries to Use With React