Web Programming

React programming

리액트 컴포넌트 스타일링하기

1. Sass
2. CSS Module
3. styled-components

Sass

Sass(Syntactically Awesome Style Sheets)

- Sass는 세계에서 가장 성숙하고 안정적이며 강력한 전문가급 CSS 확장 언어다.
- CSS의 단점을 보완하기 위해 만든 CSS 전처리기이다. 보통 CSS를 사용하다보면 단순 반복되는 부분이 많은 등, 불편함이 느껴지기 마련인데, SASS는 이 부분을 거의 완전히 해소시켜주는 스타일시트 언어다. SASS에는 Sass와 SCSS가 있다.
- 또한 CSS의 확장팩 같은 언어이기 때문에 CSS 파일 그 자체를 SCSS로 확장자만 바꾸어주어도 정상적으로 작동한다. 다만 SASS 자체는 개발자용 언어이기 때문에 웹브라우저가 읽을 수 없다. 따라서 중간에 컴파일러를 거쳐서 CSS로 변환하여 HTML에 연결한다.
- 이 형식을 사용한 파일의 확장자는 .sass, .scss이다. Sass는 SCSS에서 중괄호를 없애서 용량을 줄일 수 있지만 실수로 인해 컴파일 에러가 뜰 확률이 꽤 크다.
- 비슷한 CSS 전처리기 언어로는 LESS, Stylus 등이 있다.
- CSS pre-processor 로서, 복잡한 작업을 쉽게 할 수 있게 해주고, 코드의 재활용성을 높여줄 뿐 만 아니라, 코드의 가독성을 높여주어 유지보수를 쉽게 해준다.

Sass & scss

sass

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

body
  font: 100% $font-stack
  color: $primary-color
```

scss

```
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

SCSS Sass

```scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

CSS

```css
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

Button 프로젝트(SCSS)

시작하기

1. npx create-react-app styling-with-sass
2. cd styling-with-sass
3. yarn add node-sass

- 새로운 리액트 프로젝트
- 해당 프로젝트 디렉터리에 node-sass 라이브러리를 설치
- 이 라이브러리는 Sass 를 CSS 로 변환해주는 역할을 한다.

Button 프로젝트(SCSS)

Button 컴포넌트 만들기

- Button 이라는 컴포넌트를 만들고, Sass 를 사용해서 스타일링

components/Button.js

```
import React from 'react';
import './Button.scss';

function Button({ children }) {
  return <button className="Button">{children}</button>;
}

export default Button;
```

```
$blue: #228be6; // 주석 사용

.Button {
  display: inline-flex;
  color: white;
  font-weight: bold;
  outline: none;
  border-radius: 4px;
  border: none;
  cursor: pointer;
  height: 2.25rem;
  padding-left: 1rem;
  padding-right: 1rem;
  font-size: 1rem;

  background: $blue; // 주석 사용
  &:hover {
    background: lighten($blue, 10%); // 색상 10% 밝게
  }

  &:active {
    background: darken($blue, 10%); // 색상 10% 어둡게
  }
}
```

components/Button.scss

- ✓ 기존 css 에서는 사용하지 못하던 문법들을 사용했다.
- ✓ $blue: #228be6; 이런 식으로 스타일 파일에서 사용 할 수 있는 변수를 선언 할 수도 있고 lighten() 또는 darken() 과 같이 색상을 더 밝게하거나 어둡게 해주는 함수도 사용 할 수 있다.

Button 프로젝트(SCSS)

Button 컴포넌트 만들기

- Button 이라는 컴포넌트를 만들고, Sass 를 사용해서 스타일링

App.js

```
import React from 'react';
import './App.scss';
import Button from './components/Button';

function App() {
  return (
    <div className="App">
      <div className="buttons">
        <Button>BUTTON</Button>
      </div>
    </div>
  );
}

export default App;
```

App.scss

```
.App {
  width: 512px;
  margin: 0 auto;
  margin-top: 4rem;
  border: 1px solid black;
  padding: 1rem;
}
```

기존의 App.css 를 App.scss 로 파일 이름을 수정한 뒤, 내용

Button 프로젝트(SCSS)

Button 크기 조정하기

- 우선 버튼 크기에 large, medium, small 를 설정해줄 수 있도록 구현
- className에 CSS 클래스 이름을 동적으로 넣어주고 싶으면
  - ➢ className={['Button', size].join(' ')} 또는 className={`Button ${size}`}
- yarn add classnames
- classNames 를 사용하면 다음과 같이 조건부 스타일링을 할 때 함수의 인자에 문자열, 배열, 객체 등을 전달하여 손쉽게 문자열을 조합 할 수 있다.

```
classNames('foo', 'bar'); // => 'foo bar'
classNames('foo', { bar: true }); // => 'foo bar'
classNames({ 'foo-bar': true }); // => 'foo-bar'
classNames({ 'foo-bar': false }); // => ''
classNames({ foo: true }, { bar: true }); // => 'foo bar'
classNames({ foo: true, bar: true }); // => 'foo bar'
classNames(['foo', 'bar']); // => 'foo bar'

// 동시에 여러개의 타입으로 받아올 수 도 있습니다.
classNames('foo', { bar: true, duck: false }, 'baz', { quux: true }); // => 'foo bar baz quux'

// false, null, 0, undefined 는 무시됩니다.
classNames(null, false, 'bar', undefined, 0, 1, { baz: null }, ''); // => 'bar 1'
```

Button 프로젝트(SCSS)

Button 크기 조정하기

- props 값이 button 태그의 className 으로 전달된다.

```
import React from 'react';
import './Button.scss';
import classNames from 'classnames';

function Button({ children, size }) {
  return <button className={classNames('Button', size)}>{children}</button>;
}

Button.defaultProps = {
  size: 'medium'
};

export default Button;
```

Button 프로젝트(SCSS)

Button 크기 조정하기

- Button.scss에서 다른 크기를 지정

```
$blue: #228be6;

.Button {
  display: inline-flex;
  color: white;
  font-weight: bold;
  outline: none;
  border-radius: 4px;
  border: none;
  cursor: pointer;

  // 사이즈 관리
  &.large {
    height: 3rem;
    padding-left: 1rem;
    padding-right: 1rem;
    font-size: 1.25rem;
  }
```

```
  &.medium {
    height: 2.25rem;
    padding-left: 1rem;
    padding-right: 1rem;
    font-size: 1rem;
  }

  &.small {
    height: 1.75rem;
    font-size: 0.875rem;
    padding-left: 1rem;
    padding-right: 1rem;
  }

  background: $blue;
  &:hover {
    background: lighten($blue, 10%);
  }
```

```
  &:active {
    background: darken($blue, 10%);
  }
}
```

```
.Button {
  &.large {

  }
}
```

```
.Button.large {

}
```

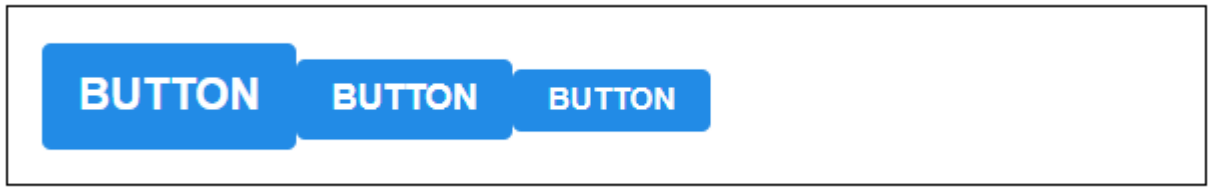
Button과 large CSS 클래스가 함께 적용되어 있으면 우리가 원하는 스타일을 적용하겠다는 것을 의미

Button 프로젝트(SCSS)

Button 크기 조정하기

- App.js에서 버튼들을 2개 더 렌더링 하고, size 값도 설정

```
import React from 'react';
import './App.scss';
import Button from './components/Button';

function App() {
  return (
    <div className="App">
      <div className="buttons">
        <Button size="large">BUTTON</Button>
        <Button>BUTTON</Button>
        <Button size="small">BUTTON</Button>
      </div>
    </div>
  );
}

export default App;
```



Button 프로젝트(SCSS)

Button 크기 조정하기

- 버튼끼리 함께 있을 때에는 여백이 있도록 Button.scss 를 다음과 같이 수정

```scss
$blue: #228be6;

.Button {
  display: inline-flex;
  color: white;
  font-weight: bold;
  outline: none;
  border-radius: 4px;
  border: none;
  cursor: pointer;

  // 사이즈 관리
  &.large {
    height: 3rem;
    padding-left: 1rem;
    padding-right: 1rem;
    font-size: 1.25rem;
  }
```

```scss
  &.medium {
    height: 2.25rem;
    padding-left: 1rem;
    padding-right: 1rem;
    font-size: 1rem;
  }

  &.small {
    height: 1.75rem;
    font-size: 0.875rem;
    padding-left: 1rem;
    padding-right: 1rem;
  }

  background: $blue;
  &:hover {
    background: lighten($blue, 10%);
  }
```

```scss
  &:active {
    background: darken($blue, 10%);
  }

  & + & {
    margin-left: 1rem;
  }
}
```

& + & 가 의미 하는 것은 .Button + .Button 이다. 만약 함께 있다면 우측에 있는 버튼에 여백을 설정 한 것이다.

Button 프로젝트(SCSS)

Button 색상 설정하기

- 버튼의 색상에 blue, gray, pink 색을 설정 할 수 있도록 구현
- Button 에서 color 라는 props 를 받아올 수 있도록 해주고, 기본 값을 blue로 설정
- size 와 마찬가지로 color 값을 className에 포함

```
import React from 'react';
import classNames from 'classnames';
import './Button.scss';

function Button({ children, size, color }) {
  return (
    <button className={classNames('Button', size, color)}>{children}</button>
  );
}

Button.defaultProps = {
  size: 'medium',
  color: 'blue'
};

export default Button;
```

open-color

Button 프로젝트(SCSS)

Button 색상 설정하기

- Button.scss 파일 수정

```
$blue: #228be6;
$gray: #495057;
$pink: #f06595;

@mixin button-color($color) {
  background: $color;
  &:hover {
    background: lighten($color, 10%);
  }
  &:active {
    background: darken($color, 10%);
  }
}

.Button {
  display: inline-flex;
  color: white;
  font-weight: bold;
  outline: none;
  border-radius: 4px;
  border: none;
  cursor: pointer;
```

```
  // 사이즈 관리
  &.large {
    height: 3rem;
    padding-left: 1rem;
    padding-right: 1rem;
    font-size: 1.25rem;
  }

  &.medium {
    height: 2.25rem;
    padding-left: 1rem;
    padding-right: 1rem;
    font-size: 1rem;
  }

  &.small {
    height: 1.75rem;
    font-size: 0.875rem;
    padding-left: 1rem;
    padding-right: 1rem;
  }
```

```
  // 색상 관리
  &.blue {
    @include button-color($blue);
  }

  &.gray {
    @include button-color($gray);
  }

  &.pink {
    @include button-color($pink);
  }

  & + & {
    margin-left: 1rem;
  }
}
```

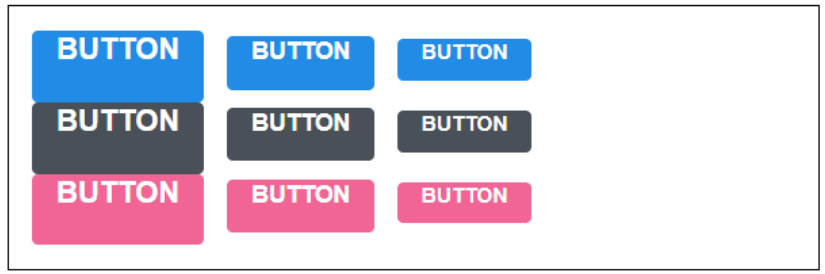
Sass의 mixin

Button 프로젝트(SCSS)

Button 색상 설정하기

- App 컴포넌트에서 다음과 같이 다른 색상을 가진 버튼들 렌더링

```
import React from 'react';
import './App.scss';
import Button from './components/Button';

function App() {
  return (
    <div className="App">
      <div className="buttons">
        <Button size="large">BUTTON</Button>
        <Button>BUTTON</Button>
        <Button size="small">BUTTON</Button>
      </div>
      <div className="buttons">
        <Button size="large" color="gray">
          BUTTON
        </Button>
        <Button color="gray">BUTTON</Button>
        <Button size="small" color="gray">
          BUTTON
        </Button>
      </div>
      <div className="buttons">
        <Button size="large" color="pink">
          BUTTON
        </Button>
        <Button color="pink">BUTTON</Button>
        <Button size="small" color="pink">
          BUTTON
        </Button>
      </div>
    </div>
  );
}

export default App;
```

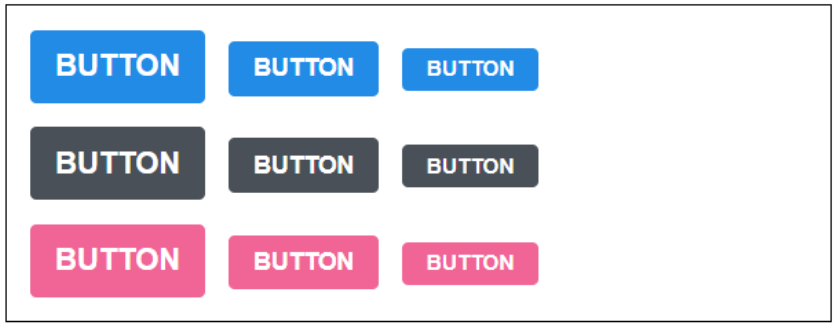


Button 프로젝트(SCSS)

Button 색상 설정하기

- App.scss를 다음과 같이 수정

```
.App {
  width: 512px;
  margin: 0 auto;
  margin-top: 4rem;
  border: 1px solid black;
  padding: 1rem;
  .buttons + .buttons {
    margin-top: 1rem;
  }
}
```



Button 프로젝트(SCSS)

outline 옵션 만들기

- outline 이라는 옵션을 주면 버튼에서 테두리만 보여지도록 설정
- Button.js 를 다음과 같이 수정
- outline 값을 props 로 받아와서 객체 안에 집어 넣은 다음에 classNames() 에 포함

```
import React from 'react';
import classNames from 'classnames';
import './Button.scss';

function Button({ children, size, color, outline }) {
  return (
    <button className={classNames('Button', size, color, { outline })}>
      {children}
    </button>
  );
}

Button.defaultProps = {
  size: 'medium',
  color: 'blue'
};

export default Button;
```

Button 프로젝트(SCSS)

outline 옵션 만들기

- outline 값이 true 일 때에만 button 에 outline CSS 클래스가 적용
- 만약 outline CSS 클래스가 있다면, 테두리만 보여지도록 scss 수정 시도 바람

```scss
$blue: #228be6;
$gray: #495057;
$pink: #f06595;

@mixin button-color($color) {
  background: $color;
  &:hover {
    background: lighten($color, 10%);
  }
  &:active {
    background: darken($color, 10%);
  }
  &.outline {
    color: $color;
    background: none;
    border: 1px solid $color;
    &:hover {
      background: $color;
      color: white;
    }
  }
}
```

```scss
.Button {
  display: inline-flex;
  color: white;
  font-weight: bold;
  outline: none;
  border-radius: 4px;
  border: none;
  cursor: pointer;

  // 사이즈 관리
  &.large {
    height: 3rem;
    padding-left: 1rem;
    padding-right: 1rem;
    font-size: 1.25rem;
  }
```

```scss
  &.medium {
    height: 2.25rem;
    padding-left: 1rem;
    padding-right: 1rem;
    font-size: 1rem;
  }

  &.small {
    height: 1.75rem;
    font-size: 0.875rem;
    padding-left: 1rem;
    padding-right: 1rem;
  }
```

```scss
  // 색상 관리
  &.blue {
    @include button-color($blue);
  }

  &.gray {
    @include button-color($gray);
  }

  &.pink {
    @include button-color($pink);
  }

  & + & {
    margin-left: 1rem;
  }
}
```

Button 프로젝트(SCSS)

outline 옵션 만들기

- App에서 사용

```
import React from 'react';
import './App.scss';
import Button from './components/Button';

function App() {
  return (
    <div className="App">
      <div className="buttons">
        <Button size="large">BUTTON</Button>
        <Button>BUTTON</Button>
        <Button size="small">BUTTON</Button>
      </div>
      <div className="buttons">
        <Button size="large" color="gray">
          BUTTON
        </Button>
        <Button color="gray">BUTTON</Button>
        <Button size="small" color="gray">
          BUTTON
        </Button>
      </div>
```

```
      <div className="buttons">
        <Button size="large" color="pink">
          BUTTON
        </Button>
        <Button color="pink">BUTTON</Button>
        <Button size="small" color="pink">
          BUTTON
        </Button>
      </div>
      <div className="buttons">
        <Button size="large" color="blue" outline>
          BUTTON
        </Button>
        <Button color="gray" outline>
          BUTTON
        </Button>
        <Button size="small" color="pink" outline>
          BUTTON
        </Button>
      </div>
    </div>
  );
}
```

Button 프로젝트(SCSS)

전체 너비 차지하는 옵션

- fullWidth라는 옵션이 있으면 버튼이 전체 너비를 차지하도록 구현
- 구현 방식은 방금 했었던 outline과 굉장히 유사

```
import React from 'react';
import classNames from 'classnames';
import './Button.scss';

function Button({ children, size, color, outline, fullWidth }) {
  return (
    <button
      className={classNames('Button', size, color, { outline, fullWidth })}
    >
      {children}
    </button>
  );
}

Button.defaultProps = {
  size: 'medium',
  color: 'blue'
};

export default Button;
```

Button 프로젝트(SCSS)

전체 너비 차지하는 옵션

- 스타일 수정

```
$blue: #228be6;
$gray: #495057;
$pink: #f06595;

@mixin button-color($color) {
  background: $color;
  &:hover {
    background: lighten($color, 10%);
  }
  &:active {
    background: darken($color, 10%);
  }
  &.outline {
    color: $color;
    background: none;
    border: 1px solid $color;
    &:hover {
      background: $color;
      color: white;
    }
  }
}
```

```
  &.pink {
    @include button-color($pink);
  }

  & + & {
    margin-left: 1rem;
  }

  &.fullWidth {
    width: 100%;
    justify-content: center;
    & + & {
      margin-left: 0;
      margin-top: 1rem;
    }
  }
}
```

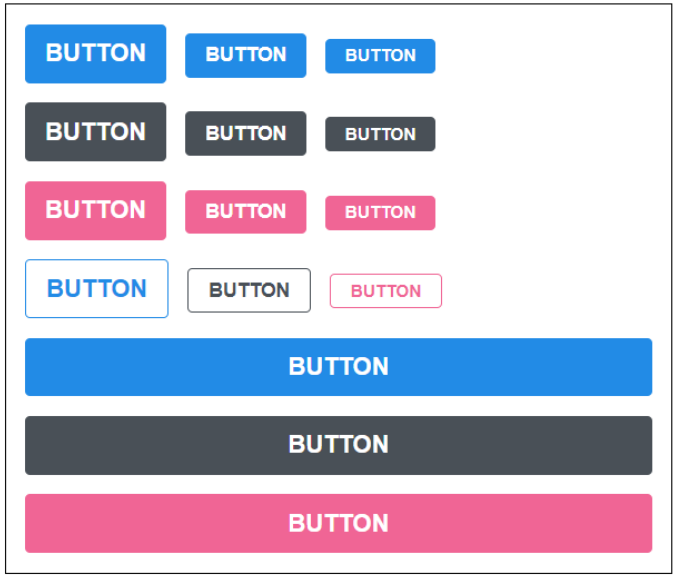
```
  &.small {
    height: 1.75rem;
    font-size: 0.875rem;
    padding-left: 1rem;
    padding-right: 1rem;
  }

  // 색상 관리
  &.blue {
    @include button-color($blue);
  }

  &.gray {
    @include button-color($gray);
  }
```

```
.Button {
  display: inline-flex;
  color: white;
  font-weight: bold;
  outline: none;
  border-radius: 4px;
  border: none;
  cursor: pointer;

  // 사이즈 관리
  &.large {
    height: 3rem;
    padding-left: 1rem;
    padding-right: 1rem;
    font-size: 1.25rem;
  }

  &.medium {
    height: 2.25rem;
    padding-left: 1rem;
    padding-right: 1rem;
    font-size: 1rem;
  }
```

Button 프로젝트(SCSS)

전체 너비 차지하는 옵션

- App에서 사용




```
import React from 'react';
import './App.scss';
import Button from './components/Button';

function App() {
  return (
    <div className="App">
      <div className="buttons">
        <Button size="large">BUTTON</Button>
        <Button>BUTTON</Button>
        <Button size="small">BUTTON</Button>
      </div>
      <div className="buttons">
        <Button size="large" color="gray">
          BUTTON
        </Button>
        <Button color="gray">BUTTON</Button>
        <Button size="small" color="gray">
          BUTTON
        </Button>
      </div>
```

```
      <div className="buttons">
        <Button size="large" color="pink">
          BUTTON
        </Button>
        <Button color="pink">BUTTON</Button>
        <Button size="small" color="pink">
          BUTTON
        </Button>
      </div>
      <div className="buttons">
        <Button size="large" color="blue"
outline>
          BUTTON
        </Button>
        <Button color="gray" outline>
          BUTTON
        </Button>
        <Button size="small" color="pink"
outline>
          BUTTON
        </Button>
      </div>
```

```
      <div className="buttons">
        <Button size="large" fullWidth>
          BUTTON
        </Button>
        <Button size="large" fullWidth color="gray">
          BUTTON
        </Button>
        <Button size="large" fullWidth color="pink">
          BUTTON
        </Button>
      </div>
    </div>
  );
}
```

Button 프로젝트(SCSS)

...rest props 전달하기

- 컴포넌트에 onClick 설정

```
import React from 'react';
import classNames from 'classnames';
import './Button.scss';

function Button({ children, size, color, outline, fullWidth, onClick }) {
  return (
    <button
      className={classNames('Button', size, color, { outline, fullWidth })}
      onClick={onClick}
    >
      {children}
    </button>
  );
}

Button.defaultProps = {
  size: 'medium',
  color: 'blue'
};

export default Button;
```

- 컴포넌트에 onMouseMove 설정

```
import React from 'react';
import classNames from 'classnames';
import './Button.scss';

function Button({ children, size, color, outline, fullWidth, onClick,
onMouseMove }) {
  return (
    <button
      className={classNames('Button', size, color, { outline, fullWidth })}
      onClick={onClick}
      onMouseMove={onMouseMove}
    >
      {children}
    </button>
  );
}

Button.defaultProps = {
  size: 'medium',
  color: 'blue'
};

export default Button;
```

Button 프로젝트(SCSS)

spread 와 rest

...rest props 전달하기

- 필요한 이벤트가 있을 때 마다 매번 이렇게 넣어주는 건 귀찮다. 이러한 문제를 해결 해줄 수 있는 문법이 바로 spread 와 rest 이다.
- 이 문법은 주로 배열과 객체, 함수의 파라미터, 인자를 다룰 때 사용하는데, 컴포넌트에서도 사용할 수 있다.
- ...rest를 사용해서 우리가 지정한 props 를 제외한 값들을 rest 라는 객체에 모아주고, <button> 태그에 {...rest} 를 해주면, rest 안에 있는 객체안에 있는 값들을 모두 <button> 태그에 설정을 해준다.

```
import React from 'react';
import classNames from 'classnames';
import './Button.scss';

function Button({ children, size, color, outline, fullWidth, ...rest }) {
  return (
    <button
      className={classNames('Button', size, color, { outline, fullWidth })}
      {...rest}
    >
```

```
 {children}
    </button>
  );
}
Button.defaultProps = {
  size: 'medium',
  color: 'blue'
};

export default Button;
```

Button 프로젝트(SCSS)

...rest props 전달하기

- App.js 에서 사용한 가장 첫번째 버튼에 onClick 을 설정

```
import React from 'react';
import './App.scss';
import Button from './components/Button';

function App() {
  return (
    <div className="App">
      <div className="buttons">
        <Button size="large" onClick={() => console.log('클릭됐다!')}>
          BUTTON
        </Button>
        <Button>BUTTON</Button>
        <Button size="small">BUTTON</Button>
      </div>
      <div className="buttons">
        <Button size="large" color="gray">
          BUTTON
        </Button>
        <Button color="gray">BUTTON</Button>
        <Button size="small" color="gray">
          BUTTON
        </Button>
      </div>
```

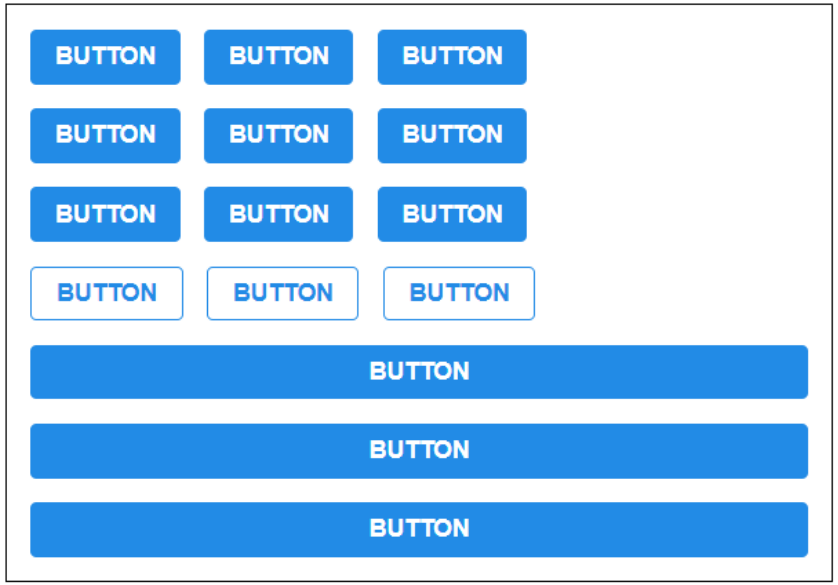
```
      <div className="buttons">
        <Button size="large" color="pink">
          BUTTON
        </Button>
        <Button color="pink">BUTTON</Button>
        <Button size="small" color="pink">
          BUTTON
        </Button>
      </div>
      <div className="buttons">
        <Button size="large" color="blue" outline>
          BUTTON
        </Button>
        <Button color="gray" outline>
          BUTTON
        </Button>
        <Button size="small" color="pink" outline>
          BUTTON
        </Button>
      </div>
```

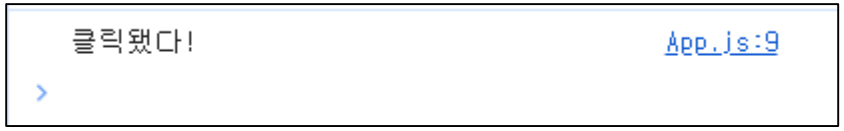
Button 프로젝트(SCSS)

...rest props 전달하기

- App.js 에서 사용한 가장 첫번째 버튼에 onClick 을 설정

```
 <div className="buttons">
        <Button size="large" fullWidth>
          BUTTON
        </Button>
        <Button size="large" color="gray" fullWidth>
          BUTTON
        </Button>
        <Button size="large" color="pink" fullWidth>
          BUTTON
        </Button>
      </div>
    </div>
  );
}

export default App;
```





defaultProps

react18.3 에서는 defaultProps 가 warning으로 표기

- deprecated

```
import React from 'react';
import classNames from 'classnames';
import './Button.scss';

function Button({ children, size, color, outline, fullWidth, ...rest })
{
  return (
    <button
      className={classNames('Button', size, color, { outline,
fullWidth })}
      {...rest}
    >
      {children}
    </button>
  );
}

Button.defaultProps = {
  size: 'medium',
  color: 'blue'
};

export default Button;
```

```
import React from 'react';
import classNames from 'classnames';
import './Button.scss';

function Button({ children, size, color, outline, fullWidth, ...rest }) {
  return (
    <button
      className={classNames('Button', size='medium',
color='blue', { outline, fullWidth })}
      {...rest}
    >
      {children}
    </button>
  );
}

export default Button;
```

```
▶Warning: Button:      react-dom.development.js:86
Support for defaultProps will be removed from
function components in a future major release. Use
JavaScript default parameters instead.
    at Button (
```

CSS module

CSS Module

- 리액트 프로젝트에서 컴포넌트를 스타일링 할 때 CSS Module 기술을 사용하면, CSS 클래스가 중첩되는 것을 완벽히 방지할 수 있다.
- CRA 로 만든 프로젝트에서 CSS Module 를 사용 할 때에는, CSS 파일의 확장자를 .module.css 로 하면 된다.
- 예를 들어서 다음과 같이 Box.module.css 라는 파일을 만들게 된다면

Box.module.css

```
.Box {
  background: black;
  color: white;
  padding: 2rem;
}
```

- 리액트 컴포넌트 파일에서 해당 CSS 파일을 불러올 때 CSS 파일에 선언한 클래스 이름들이 모두 고유해진다. 고유 CSS 클래스 이름이 만들어지는 과정에서는 파일 경로, 파일 이름, 클래스 이름, 해쉬값 등이 사용 될 수 있다.

CSS → SCSS → BEM → CSS-Modules → Styled-Component

CSS module

CSS Module

- 예를 들어서 다음과 같은 코드로 Box 컴포넌트를 만든다면

Box.js

```
import React from "react";
import styles from "./Box.module.css";

function Box() {
  return <div className={styles.Box}>{styles.Box}</div>;
}
export default Box;
```

App.js

```
import React from 'react';
import './App.scss';
import Box from './Box';

function App() {
  return (
    <div>
      <Box />
    </div>
  );
}

export default App;
```

```
Box_Box__ek+rD
```

- className을 설정 할 때는 styles.Box 이렇게 import로 불러온 styles 객체 안에 있는 값을 참조해야 한다.

- CSS Module은 별도로 설치해야 할 라이브러리는 없다.
- 이 기능은 webpack에서 사용하는 css-loader에서 지원되는데, CRA 로 만든 프로젝트에는 이미 적용이 되어있으니 바로 사용하면 된다.

CSS module

CSS Module

- 클래스 이름에 대하여 고유한 이름들이 만들어지기 때문에, 실수로 CSS 클래스 이름이 다른 관계 없는 곳에서 사용한 CSS 클래스 이름과 중복되는 일에 대하여 걱정 할 필요가 없다.
- 이 기술은 다음과 같은 상황에 사용하면 유용하다.
  - 레거시 프로젝트에 리액트를 도입할 때
    (기존 프로젝트에 있던 CSS 클래스와 이름이 중복되어도 스타일이 꼬이지 않게 해준다.)
  - CSS 클래스를 중복되지 않게 작성하기 위하여 CSS 클래스 네이밍 규칙을 만들기 귀찮을 때
- 자주 사용하는 CSS module네이밍 규칙
  - 컴포넌트의 이름은 다른 컴포넌트와 중복되지 않게 한다.
  - 컴포넌트의 최상단 CSS 클래스는 컴포넌트의 이름과 일치시킨다. (예: .Button
  - 컴포넌트 내부에서 보여지는 CSS 클래스는 CSS Selector를 잘 활용한다. (예: .MyForm .my-input)
- 이런 규칙 외에도 BEM Convention 이란 것도 있다.
- 만약 CSS 클래스 네이밍 규칙을 만들고 따르기 싫다면, CSS Module 을 사용하면 된다.

CheckBox(CSS module)

CheckBox 컴포넌트

- CSS Module 기술을 사용하여 커스텀 체크박스 컴포넌트를 만드는 방법
- 스타일링도 하지 않고, 체크 아이콘도 사용하지 않고 그냥 이 컴포넌트에 필요한 HTML 태그들만 미리 선언.
- ...rest를 사용한 이유는, CheckBox 컴포넌트에게 전달하게 될 name, onChange 같은 값을 그대로 input 에게 넣어주기 위함.

components/CheckBox.js

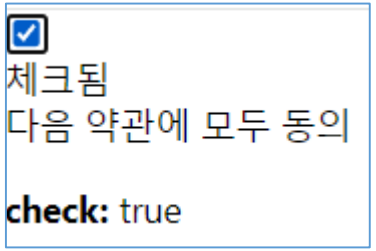
```
import React from 'react';

function CheckBox({ children, checked, ...rest }) {
  return (
    <div>
      <label>
        <input type="checkbox" checked={checked} {...rest} />
        <div>{checked ? '체크됨' : '체크 안됨'}</div>
      </label>
      <span>{children}</span>
    </div>
  );
}

export default CheckBox;
```

CheckBox(CSS module)

CheckBox 렌더링

- App 컴포넌트에서 렌더링

```
import React, { useState } from 'react';

import CheckBox from './components/CheckBox';

function App() {
  const [check, setCheck] = useState(false);
  const onChange = e => {
    setCheck(e.target.checked);
  };
  return (
    <div>
      <CheckBox onChange={onChange} checked={check}>
        다음 약관에 모두 동의
      </CheckBox>
      <p>
        <b>check: </b>
        {check ? 'true' : 'false'}
      </p>
    </div>
  );
}

export default App;
```



input이 아닌 텍스트 부분을 선택했는데도 값이 바뀌는 이유는 현재 우리가 해당 내용을 label 태그로 감싸줬기 때문.

CheckBox(CSS module)

스타일링

- yarn add react-icons
- 이 라이브러리를 사용하면 Font Awesome, Ionicons, Material Design Icons 등의 아이콘들을 컴포넌트 형태로 쉽게 사용 할 수 있다.
- 여기서는 Material Design Icons 의 MdCheckBox, MdCheckBoxOutline 을 사용.

```
import React from 'react';
import { MdCheckBox, MdCheckBoxOutlineBlank } from 'react-icons/md';

function CheckBox({ children, checked, ...rest }) {
  return (
    <div>
      <label>
        <input type="checkbox" checked={checked} {...rest} />
        <div>{checked ? <MdCheckBox /> : <MdCheckBoxOutlineBlank />}</div>
      </label>
      <span>{children}</span>
    </div>
  );
}

export default CheckBox;
```

CheckBox(CSS module)

components/CheckBox.module.css

- 컴포넌트 스타일링

```
.checkbox {
  display: flex;
  align-items: center;
}

.checkbox label {
  cursor: pointer;
}

/* 실제 input 을 숨기기 위한 코드 */
.checkbox input {
  width: 0;
  height: 0;
  position: absolute;
  opacity: 0;
}

.checkbox span {
  font-size: 1.125rem;
  font-weight: bold;
}
```

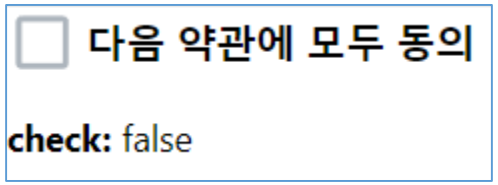
```
.icon {
  display: flex;
  align-items: center;
  /* 아이콘의 크기는 폰트 사이즈로 조정 가능 */
  font-size: 2rem;
  margin-right: 0.25rem;
  color: #adb5bd;
}

.checked {
  color: #339af0;
}
```

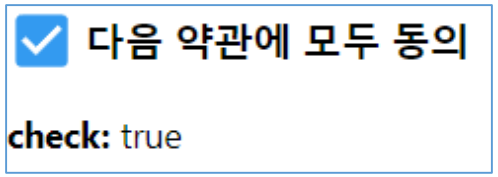
CheckBox(CSS module)

components/CheckBox.js

- 스타일 반영

```
import React from 'react';
import { MdCheckBox, MdCheckBoxOutlineBlank } from 'react-icons/md';
import styles from './CheckBox.module.css';

function CheckBox({ children, checked, ...rest }) {
  return (
    <div className={styles.checkbox}>
      <label>
        <input type="checkbox" checked={checked} {...rest} />
        <div className={styles.icon}>
          {checked ? (
            <MdCheckBox className={styles.checked} />
          ) : (
            <MdCheckBoxOutlineBlank />
          )}
        </div>
      </label>
      <span>{children}</span>
    </div>
  );
}

export default CheckBox;
```





개발자 도구로 엘리먼트를 선택해보면 고유한 클래스 이름이 만들어진 것을 확인 할 수 있다.

CheckBox(CSS module)

객체 안에 있는 값 조회

- CSS Module 을 사용 할 때에는 styles.icon 과 같이 객체안에 있는 값을 조회해야 하는데, 만약 클래스 이름에 - 가 들어가 있다면 다음과 같이 사용해야한다: styles['my-class']
- 그리고, 만약에 여러개가 있다면 다음과 같이 작성해한다: ${styles.one} ${styles.two}
- 조건부 스타일링을 해야 한다면 더더욱 번거롭다 ${styles.one} ${condition ? styles.two : ''}
- classnames 라이브러리에는 bind 기능을 사용하면 CSS Module 을 조금 더 편하게 사용 할 수 있다.
- yarn add classnames

- classnames 의 bind 기능을 사용하면, CSS 클래시 이름을 지정해 줄 때 cx('클래스이름')과 같은 형식으로 편하게 사용 할 수 있다.
- 여러 개의 CSS 클래스를 사용해야하거나, 조건부 스타일링을 해야 한다면 더욱 편리하다.

```
cx('one', 'two')
cx('my-component', {
  condition: true
})
cx('my-component', ['another', 'classnames'])
```

CheckBox(CSS module)

CheckBox

- CheckBox 수정

```
import React from 'react';
import { MdCheckBox, MdCheckBoxOutlineBlank } from 'react-icons/md';
import styles from './CheckBox.module.css';
import classNames from 'classnames/bind';

const cx = classNames.bind(styles);

function CheckBox({ children, checked, ...rest }) {
  return (
    <div className={cx('checkbox')}>
      <label>
        <input type="checkbox" checked={checked} {...rest} />
        <div className={cx('icon')}>
          {checked ? (
            <MdCheckBox className={cx('checked')} />
          ) : (
            <MdCheckBoxOutlineBlank />
          )}
        </div>
      </label>
      <span>{children}</span>
    </div>
  );
}

export default CheckBox;
```

CheckBox(CSS module)

기타 내용

- 참고로, CSS Module 은 Sass 에서도 사용 할 수 있다. 그냥 확장자를 .module.scss 로 바꿔주면 된다. 물론, 그 전에 node-sass 를 설치해야 한다.
- CSS Module 을 사용하고 있는 파일에서 클래스 이름을 고유화하지 않고 전역적 클래스이름을 사용하고 싶다면 다음과 작성하면 된다.

```
:global .my-global-name {

}
```

SASS

```
:global {
  .my-global-name {

  }
}
```

- 반대로, CSS Module을 사용하지 않는 곳에서 특정 클래스에서만 고유 이름을 만들어서 사용하고 싶다면 다음과 같이 할 수 있다.

```
:local .make-this-local {

}
```

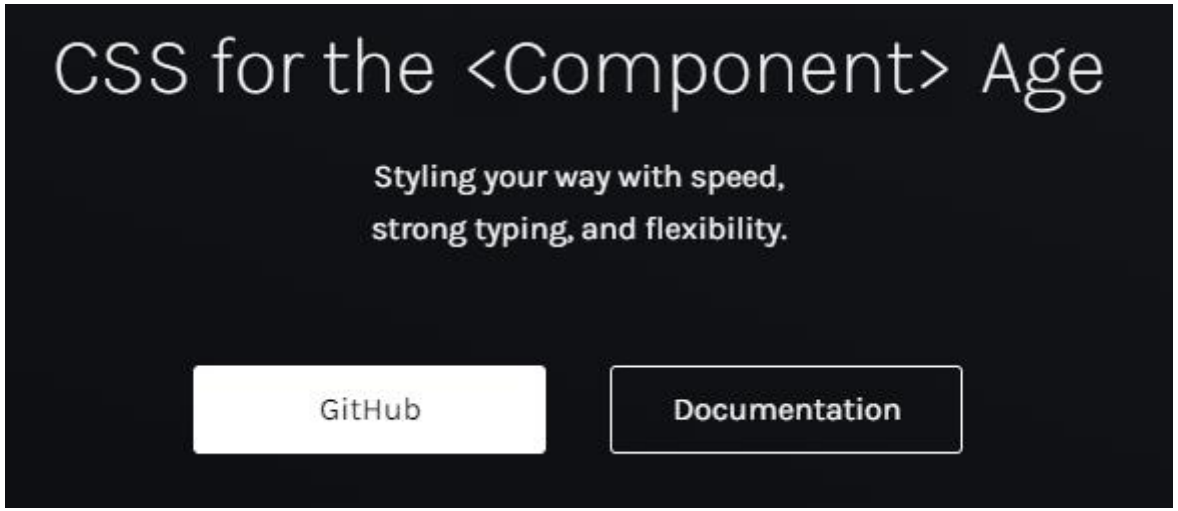
SASS

```
:local {
  .make-this-local {

  }
}
```

styled-components

styled-components

- CSS in JS 기술 : JS 안에 CSS 를 작성하는 것을 의미
- styled-components는 CSS in JS 관련 리액트 라이브러리 중에서 가장 인기 있는 라이브러리다.
- 이에 대한 대안으로는 emotion 과 styled-jsx가 있다.



styled-components

Template Literal

- Template Literal은 문자열 처리를 위한 리터럴로써 표현식을 포함할 수 있다.
- 기존에 우리가 문자열 사이에 동적으로 변경될 변수값을 넣어야 한다고 하면 ` '오늘의 날짜는' + today + 입니다. 이런식으로 + 연산으로 문자열 결합을 해야했다.
- 하지만, Template Literal 을 통해 표현식이 포함된 문자열처리를 한 번에 할 수 있다.
- 또한, 함수를 호출하며 Template Literal을 바로 파라미터로 넣어줄 수도 있는데 템플릿 안의 표현식이 있다면 평가가 된 결과가 파라미터로 넘어가게 된다.

```
console.log(`ABC`);
const one = 1, two = 2;
const result = `1 +2는 ${one + two}이 된다.`;
console.log(result);
console.log(`1 +2는₩n ${one + two}이 된다.`);
```

```
> console.log(`ABC`);
  const one = 1, two = 2;
  const result = `1 +2는 ${one + two}이 된다.`;
  console.log(result);
  console.log(`1 +2는₩n ${one + two}이 된다.`);
  ABC                                    instrument.ts:113
  1 +2는 3이 된다.                       instrument.ts:113
  1 +2는
   3이 된다.                             instrument.ts:113
```

styled-components

tagged Template

- 템플릿에 함수 이름을 작성한 형태를 tagged Template라 하는데 여기서 호출되는 함수를 태그함수(tag function)이라 부른다.

```
const one = 1, two = 2;
const show = (text, value) =>{
                console.log(`${text[0]}${value}`);
                console.log(text[1]);
};
show `1 + 2 = ${one + two}`;
```

1. Template에서 문자열과 표현식을 분리한다.
2. " 1 + 2 = " 가 문자열, ${one + two}가 표현식이며 평가결과는 3이다.
3. Show()함수를 호출한다.
4. 문자열을 배열로 넘겨준다.

   좌측 → 우측으로 배열 엘리먼트로 추가하며 마지막 빈 문자열을 엘리먼트로 추가한다.

   → 빈 문자열은 템플릿에서 자동으로 넘겨준 값이다.
5. 표현식의 평가 결과를 넘겨준다.

   → show(["1 + 2 =", ""], 3)과 같다.

```
> const one = 1, two = 2;
  const show = (text, value) =>{
      console.log(`${text[0]}${value}`);
      console.log(text[1]);
  };
  show `1 + 2 = ${one + two}`;
  1 + 2 = 3                                    VM45:3
                                               VM45:4
```

styled-components

tagged Template

- show 함수를 호출 할 때 표현식이 하나가 아니라 다수라면 태그 함수에 대응하는 파라미터 이름을 작성해야 한다.
- show `1 + 2 = ${one + two}이고 1-2는${one-two}입니다.`;
  - 위 템플릿은 템플릿을 기준으로 문자열은 표현식을 기준으로 split되어 배열이되고, 표현식은 각각이 하나씩 들어가게 된다. 해당 템플릿은 ["1 + 2 =", "이고 1-2는", "입니다"]이라는 배열 파라미터 하나와 ${one + two}의 평가 결과인 3과 ${one-two}의 평가결과인 -1이 전달되어 최종적으로 show(["1 + 2 =", "이고 1-2는", "입니다", ""], 3, -1)이 된다.
- 템플릿의 끝에 문자열이 있으면 빈 문자열을 설정하지 않는다.

```
const one = 1, two = 2;
const show = (text, plus, minus) =>{
                console.log(`${text[0]}${plus}`);      // 1 + 2 = 3
                console.log(`${text[1]}${minus}`);    //이고 1-2=-1
                console.log(`${text[2]}${text[3]}`);   //입니다. undefined
};
show `1 + 2 = ${one + two}이고 1-2는${one-two}입니다.`;
```

```
> const one = 1, two = 2;
  const show = (text, plus, minus) =>{
      console.log(`${text[0]}${plus}`);   // 1 + 2 = 3
      console.log(`${text[1]}${minus}`);  // 1-2=-1
      console.log(`${text[2]}${text[3]}`);    // undefined
  };
  show `1 + 2 = ${one + two}이고 1-2는${one-two}입니다.`;
1 + 2 = 3                                          VM48:3
이고 1-2는-1                                       VM48:4
입니다.undefined                                   VM48:5
```

styled-components

tagged Template

- Rest 파라미터를 사용하면 템플릿의 표현식을 좀 더 유연하게 받을 수 있다.
- const show = (text, ...rest)
  - rest 파라미터는 [3, -1]입니다.

```
const one = 1, two = 2;
const show = (text, ...rest) =>{
                console.log(`${text[0]}${rest[0]}`);                  // 1 + 2 = 3
                console.log(`${text[1]}${rest[1]}${text[2]}`);        //이고 1-2=-1입니다.
};
show `1 + 2 = ${one + two}이고 1-2는${one-two}입니다.`;
```

```
> const one = 1, two = 2;
const show = (text, ...rest) =>{
    console.log(`${text[0]}${rest[0]}`); // 1 + 2 = 3
    console.log(`${text[1]}${rest[1]}${text[2]}`);//이고 1-2=-1입니다.
};
show `1 + 2 = ${one + two}이고 1-2는${one-two}입니다.`;

1 + 2 = 3

이고 1-2는-1입니다.
```

styled-components

String.raw

- 많은 템플릿들을 사용할 때 줄바꿈(₩n)이나 유니코드(₩u{31}₩u{32})등을 사용할 때 평가결과로 출력 되기 때문에 문제가 생길 수 있다.
- 예를 들어 ₩n문자열 가이드 이라는 태그가 있다고 할 때 실제 노출되는 결과는 아래와 같다.

```
"<span>
문자열 가이드</span>"
```

- 그렇기에 이런 줄 바꿈 혹은 유니코드들을 단순 문자열로 취급하려 할 때 사용하는게 String.raw 이다.

```
> console.log(`₩u{31}₩u{32}`);
console.log(String.raw `₩u{31}₩u{32}`);
12
₩u{31}₩u{32}
```

styled-components

String.raw()

- String.raw()는 배열의 요소별, 혹은 문자열의 문자 하나씩을 각각 전개하며 조합하여 연결하는 함수다.

```
> const one = 1, two = 2;
  console.log(String.raw({raw:"ABCD"}, one , two, 3, 5));
  A1B2C3D
```

- 문자열

  ⇒ A를 반환 버퍼에 넣고 raw()의 두번째 파라미터 값을 버퍼에 첨부합니다. → A1

  ⇒ B를 반환 버퍼 끝에 첨부한 뒤 raw()의 3번째 파라미터 값을 버퍼에 첨부합니다 → A1B2

- 그 뒤도 동일하게 첨부하는데 이때 마지막 문자 D는 버퍼에 첨부 후에 파라미터가 더 있더라도 첨부하지 않는다.
- 그렇기 때문에 D뒤에 5가 첨부되 D5가 되야 하지만 첨부되지않기에 결과값은 A1B2C3D가 된다

```
> const rawValue = {raw : ["A", "B", "C"]};
  console.log(String.raw(rawValue, 1,2,3));
  A1B2C
```

- 배열

  ⇒ `A{1}B{2}C

- C뒤에는 표현식이 없는 것으로 처리해 3이 첨부되지 않는다.

styled-components

Tagged Template Literal

```
const name = 'react';
const message = `hello ${name}`;

console.log(message);
// "hello react"
```

```
const object = { a: 1 };
const text = `${object}`
console.log(text);
// "[object Object]"
```

```
const fn = () => true
const msg = `${fn}`;
console.log(msg);
// "() => true"
```

```
const red = '빨간색';
const blue = '파란색';
function favoriteColors(texts, ...values) {
  console.log(texts);
  console.log(values);
}
favoriteColors`제가 좋아하는 색은 ${red}과 ${blue}입니다.`
```

```
const red = '빨간색';
const blue = '파란색';
function favoriteColors(texts, ...values) {
   return texts.reduce((result, text, i) => `${result}${text}${values[i] ? `<b>${values[i]}</b>` : ''}`, '');
}
favoriteColors`제가 좋아하는 색은 ${red}과 ${blue}입니다.`
// 제가 좋아하는 색은 <b>빨간색</b>과 <b>파란색</b>입니다.
```

styled-components

styled-components 사용하기

- npx create-react-app styling-with-styled-components
- cd styling-with-styled-components
- yarn add styled-components

```
import React from 'react';
import styled from 'styled-components';

const Circle = styled.div`
  width: 5rem;
  height: 5rem;
  background: black;
  border-radius: 50%;
`;

function App() {
  return <Circle />;
}

export default App;
```

- styled-components 를 사용하면 이렇게 스타일을 입력함과 동시에 해당 스타일을 가진 컴포넌트를 만들 수 있다. 만약에 div를 스타일링 하고 싶으면 styled.div, input을 스타일링 하고 싶으면 styled.input 이런식으로 사용하면 된다.

styled-components

styled-components 사용하기

- Circle 컴포넌트에 color props 넣기
- Circle 컴포넌트에서는 color props 값을 설정해줬으면 해당 값을 배경색으로 설정하고, 그렇지 않으면 검정색을 배경색으로 사용하도록 설정하였다.

```
import React from 'react';
import styled from 'styled-components';

const Circle = styled.div`
  width: 5rem;
  height: 5rem;
  background: ${props => props.color || 'black'};
  border-radius: 50%;
`;

function App() {
  return <Circle color="blue" />;
}

export default App;
```

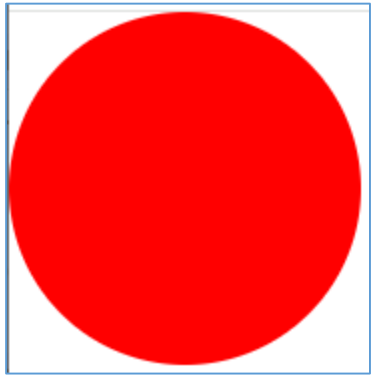
styled-components

styled-components 사용하기

- huge라는 props가 설정됐을 때 크기 변경
- 여러 줄의 CSS 코드를 조건부로 보여주고 싶다면 css 를 사용해야 한다.
- css를 불러와서 사용을 해야 그 스타일 내부에서도 다른 props 를 조회 할 수 있다.

```
import React from 'react';
import styled, { css } from 'styled-components';

const Circle = styled.div`
  width: 5rem;
  height: 5rem;
  background: ${props => props.color || 'black'};
  border-radius: 50%;
  ${props =>
    props.huge &&
    css`
      width: 10rem;
      height: 10rem;
    `}
`;
```

```
function App() {
  return <Circle color="red" huge />;
}

export default App;
```



Button(styled-components)

Button 만들기

- 이전 Sass 를 배울 때 만들었던 재 사용성 높은 Button 컴포넌트를 styled-components 로 구현

```
import React from 'react';
import styled from 'styled-components';

const StyledButton = styled.button`
  /* 공통 스타일 */
  display: inline-flex;
  outline: none;
  border: none;
  border-radius: 4px;
  color: white;
  font-weight: bold;
  cursor: pointer;
  padding-left: 1rem;
  padding-right: 1rem;

  /* 크기 */
  height: 2.25rem;
  font-size: 1rem;

  /* 색상 */
  background: #228be6;
  &:hover {
    background: #339af0;
  }
  &:active {
    background: #1c7ed6;
  }

  /* 기타 */
  & + & {
    margin-left: 1rem;
  }
`;

function Button({ children, ...rest }) {
  return <StyledButton {...rest}>{children}</StyledButton>;
}

export default Button;
```

Button(styled-components)

Button 만들기

- App.js에서 Button 사용

```
import React from 'react';
import styled from 'styled-components';
import Button from './components/Button';

const AppBlock = styled.div`
  width: 512px;
  margin: 0 auto;
  margin-top: 4rem;
  border: 1px solid black;
  padding: 1rem;
  align-items: center;
`;

function App() {
  return (
    <AppBlock>
      <Button>BUTTON</Button>
    </AppBlock>
  );
}

export default App;
```

Button(styled-components)

polished의 스타일 관련 유틸 함수 사용하기

- Sass 를 사용 할 때에는 lighten() 또는 darken() 과 같은 유틸 함수를 사용하여 색상에 변화를 줄 수 있었는데, CSS in JS 에서도 비슷한 유틸 함수를 사용하고 싶다면 polished 라는 라이브러리를 사용하면 된다.
- yarn add polished

Button(styled-components)

polished의 스타일 관련 유틸 함수 사용하기

- 버튼에 커서를 올렸을 때 색상 변경

```
import React from 'react';
import styled from 'styled-components';
import { darken, lighten } from 'polished';

const StyledButton = styled.button`
  /* 공통 스타일 */
  display: inline-flex;
  outline: none;
  border: none;
  border-radius: 4px;
  color: white;
  font-weight: bold;
  cursor: pointer;
  padding-left: 1rem;
  padding-right: 1rem;
  align-items: center;

  /* 크기 */
  height: 2.25rem;
  font-size: 1rem;

  /* 색상 */
  background: #228be6;
  &:hover {
    background: ${lighten(0.1, '#228be6')};
  }
  &:active {
    background: ${darken(0.1, '#228be6')};
  }

  /* 기타 */
  & + & {
    margin-left: 1rem;
  }
`;

function Button({ children, ...rest }) {
  return <StyledButton {...rest}>{children}</StyledButton>;
}

export default Button;
```

Button(styled-components)

ThemeProvider

- 색상 코드를 지닌 변수를 Button.js에서 선언을 하는 대신에 ThemeProvider라는 기능을 사용하여 styled-components로 만드는 모든 컴포넌트에서 조회하여 사용 할 수 있는 전역적인 값 설정
- theme을 설정하면 ThemeProvider 내부에 렌더링된 styled-components 로 만든 컴포넌트에서 palette를 조회하여 사용 할 수 있다.

```
import React from 'react';
import styled, { ThemeProvider } from 'styled-components';
import Button from './components/Button';

const AppBlock = styled.div`
  width: 512px;
  margin: 0 auto;
  margin-top: 4rem;
  border: 1px solid black;
  padding: 1rem;
`;
```

```
function App() {
  return (
    <ThemeProvider
      theme={{
        palette: {
          blue: '#228be6',
          gray: '#495057',
          pink: '#f06595'
        }
      }}
    >
      <AppBlock>
        <Button>BUTTON</Button>
      </AppBlock>
    </ThemeProvider>
  );
}

export default App;
```

ThemeProvider

Button(styled-components)

ThemeProvider

- Button 컴포넌트에서 palette.blue 값 조회

```
import React from 'react';
import styled, { css } from 'styled-components';
import { darken, lighten } from 'polished';

const StyledButton = styled.button`
  /* 공통 스타일 */
  display: inline-flex;
  outline: none;
  border: none;
  border-radius: 4px;
  color: white;
  font-weight: bold;
  cursor: pointer;
  padding-left: 1rem;
  padding-right: 1rem;
  align-items: center;

  /* 크기 */
  height: 2.25rem;
  font-size: 1rem;
```

```
  /* 색상 */
  ${props => {
    const selected = props.theme.palette.blue;
    return css`
      background: ${selected};
      &:hover {
        background: ${lighten(0.1, selected)};
      }
      &:active {
        background: ${darken(0.1, selected)};
      }
    `;
  }}

  /* 기타 */
  & + & {
    margin-left: 1rem;
  }
`;
```

```
function Button({ children, ...rest }) {
  return <StyledButton {...rest}>{children}</StyledButton>;
}

export default Button;
```

Button(styled-components)

ThemeProvider

- ThemeProvider로 설정한 값은 styled-components에서 props.theme로 조회 할 수 있다. 지금은 selected 값을 무조건 blue 값을 가르키게 했는데, 이 부분을 Button 컴포넌트가 color props를 통하여 받아오게 될 색상을 사용하도록 수정. 지금은 기본 색상이 blue 가 되도록 설정.

```
import React from 'react';
import styled, { css } from 'styled-components';
import { darken, lighten } from 'polished';

const StyledButton = styled.button`
  /* 공통 스타일 */
  display: inline-flex;
  outline: none;
  border: none;
  border-radius: 4px;
  color: white;
  font-weight: bold;
  cursor: pointer;
  padding-left: 1rem;
  padding-right: 1rem;

  /* 크기 */
  height: 2.25rem;
  font-size: 1rem;
```

```
  /* 색상 */
  ${props => {
    const selected = props.theme.palette[props.color];
    return css`
      background: ${selected};
      &:hover {
        background: ${lighten(0.1, selected)};
      }
      &:active {
        background: ${darken(0.1, selected)};
      }
    `;
  }}

  /* 기타 */
  & + & {
    margin-left: 1rem;
  }
`;
```

```
function Button({ children, ...rest }) {
  return <StyledButton {...rest}>{children}</StyledButton>;
}

Button.defaultProps = {
  color: 'blue'
};

export default Button;
```

Button(styled-components)

ThemeProvider

- App 컴포넌트를 열어서 회색, 핑크색 버튼 렌더링

```
import React from 'react';
import styled, { ThemeProvider } from 'styled-components';
import Button from './components/Button';

const AppBlock = styled.div`
  width: 512px;
  margin: 0 auto;
  margin-top: 4rem;
  border: 1px solid black;
  padding: 1rem;
`;
```

```
function App() {
  return (
    <ThemeProvider
      theme={{
        palette: {
          blue: '#228be6',
          gray: '#495057',
          pink: '#f06595'
        }
      }}
    >
      <AppBlock>
        <Button>BUTTON</Button>
        <Button color="gray">BUTTON</Button>
        <Button color="pink">BUTTON</Button>
      </AppBlock>
    </ThemeProvider>
  );
}

export default App;
```

Button(styled-components)

리팩토링

- Button 컴포넌트의 코드 리팩토링

```
import React from 'react';
import styled, { css } from 'styled-components';
import { darken, lighten } from 'polished';

const StyledButton = styled.button`
  /* 공통 스타일 */
  display: inline-flex;
  outline: none;
  border: none;
  border-radius: 4px;
  color: white;
  font-weight: bold;
  cursor: pointer;
  padding-left: 1rem;
  padding-right: 1rem;

  /* 크기 */
  height: 2.25rem;
  font-size: 1rem;
 /* 색상 */
  ${({ theme, color }) => {
    const selected = theme.palette[color];
    return css`
      background: ${selected};
      &:hover {
        background: ${lighten(0.1, selected)};
      }
      &:active {
        background: ${darken(0.1, selected)};
      }
    `;
  }}

  /* 기타 */
  & + & {
    margin-left: 1rem;
  }
`;
```

props.theme.palette.blue와 같은 방법으로 값을 조회하는 대신에 비구조화 할당 문법을 사용하여 가독성을 높임

```
function Button({ children, color, ...rest }) {
  return <StyledButton color={color} {...rest}>{children}</StyledButton>;
}

Button.defaultProps = {
  color: 'blue'
};

export default Button;
```

Button(styled-components)

리팩토링

- 색상에 관련된 코드를 분리하여 사용할 수도 있다.

```
import React from 'react';
import styled, { css } from 'styled-components';
import { darken, lighten } from 'polished';

const colorStyles = css`
  ${({ theme, color }) => {
    const selected = theme.palette[color];
    return css`
      background: ${selected};
      &:hover {
        background: ${lighten(0.1, selected)};
      }
      &:active {
        background: ${darken(0.1, selected)};
      }
    `;
  }}
`;
```

```
const StyledButton =
styled.button`
  /* 공통 스타일 */
  display: inline-flex;
  outline: none;
  border: none;
  border-radius: 4px;
  color: white;
  font-weight: bold;
  cursor: pointer;
  padding-left: 1rem;
  padding-right: 1rem;
  align-items: center;
  /* 크기 */
  height: 2.25rem;
  font-size: 1rem;

  /* 색상 */
  ${colorStyles}
```

```
  /* 기타 */
  & + & {
    margin-left: 1rem;
  }
`;

function Button({ children, color, ...rest }) {
  return <StyledButton color={color}
{...rest}>{children}</StyledButton>;
}

Button.defaultProps = {
  color: 'blue'
};

export default Button;
```

Button(styled-components)

리팩토링

- size props 를 설정하여 버튼의 크기 조절

```
import React from 'react';
import styled, { css } from 'styled-components';
import { darken, lighten } from 'polished';

const colorStyles = css`
  ${({ theme, color }) => {
    const selected = theme.palette[color];
    return css`
      background: ${selected};
      &:hover {
        background: ${lighten(0.1, selected)};
      }
      &:active {
        background: ${darken(0.1, selected)};
      }
    `;
  }}
`;
```

```
const sizeStyles = css`
  ${props =>
    props.size === 'large' &&
    css`
      height: 3rem;
      font-size: 1.25rem;
    `}

  ${props =>
    props.size === 'medium' &&
    css`
      height: 2.25rem;
      font-size: 1rem;
    `}

    ${props =>
      props.size === 'small' &&
      css`
        height: 1.75rem;
        font-size: 0.875rem;
      `}
`;
```

Button(styled-components)

리팩토링

- size props 를 설정하여 버튼의 크기 조절

```
const StyledButton = styled.button`
  /* 공통 스타일 */
  display: inline-flex;
  outline: none;
  border: none;
  border-radius: 4px;
  color: white;
  font-weight: bold;
  cursor: pointer;
  padding-left: 1rem;
  padding-right: 1rem;
  align-items: center;

  /* 크기 */
  ${sizeStyles}

  /* 색상 */
  ${colorStyles}
```

```
  /* 기타 */
  & + & {
    margin-left: 1rem;
  }
`;

function Button({ children, color, size,  ...rest }) {
  return (
    <StyledButton color={color} size={size} {...rest}>
      {children}
    </StyledButton>
  );
}

Button.defaultProps = {
  color: 'blue'
};

export default Button;
```

Button(styled-components)

리팩토링

- size props 를 설정하여 버튼의 크기 조절

```
import React from 'react';
import styled, { ThemeProvider } from 'styled-components';
import Button from './components/Button';

const AppBlock = styled.div`
  width: 512px;
  margin: 0 auto;
  margin-top: 4rem;
  border: 1px solid black;
  padding: 1rem;
`;

const ButtonGroup = styled.div`
  & + & {
    margin-top: 1rem;
  }
`;
```

```
function App() {
  return (
    <ThemeProvider
      theme={{
        palette: {
          blue: '#228be6',
          gray: '#495057',
          pink: '#f06595'
        }
      }}
    >
      <AppBlock>
        <ButtonGroup>
          <Button size="large">BUTTON</Button>
          <Button>BUTTON</Button>
          <Button size="small">BUTTON</Button>
        </ButtonGroup>
```

Button(styled-components)

리팩토링

- size props 를 설정하여 버튼의 크기 조절

```
 <ButtonGroup>
          <Button color="gray" size="large">
            BUTTON
          </Button>
          <Button color="gray">BUTTON</Button>
          <Button color="gray" size="small">
            BUTTON
          </Button>
        </ButtonGroup>
        <ButtonGroup>
          <Button color="pink" size="large">
            BUTTON
          </Button>
          <Button color="pink">BUTTON</Button>
          <Button color="pink" size="small">
            BUTTON
          </Button>
        </ButtonGroup>
      </AppBlock>
    </ThemeProvider>
  );
}
```

```
export default App;
```

Button(styled-components)

리팩토링

- sizeStyles의 중복 코드 제거

```
import React from 'react';
import styled, { css } from 'styled-components';
import { darken, lighten } from 'polished';

const colorStyles = css`
  ${({ theme, color }) => {
    const selected = theme.palette[color];
    return css`
      background: ${selected};
      &:hover {
        background: ${lighten(0.1, selected)};
      }
      &:active {
        background: ${darken(0.1, selected)};
      }
    `;
  }}
`;
```

```
const sizes = {
  large: {
    height: '3rem',
    fontSize: '1.25rem'
  },
  medium: {
    height: '2.25rem',
    fontSize: '1rem'
  },
  small: {
    height: '1.75rem',
    fontSize: '0.875rem'
  }
};

const sizeStyles = css`
  ${({ size }) => css`
    height: ${sizes[size].height};
    font-size: ${sizes[size].fontSize};
  `}
`;
```

Button(styled-components)

리팩토링

- sizeStyles의 중복 코드 제거

```
const StyledButton = styled.button`
  /* 공통 스타일 */
  display: inline-flex;
  outline: none;
  border: none;
  border-radius: 4px;
  color: white;
  font-weight: bold;
  cursor: pointer;
  padding-left: 1rem;
  padding-right: 1rem;

  /* 크기 */
  ${sizeStyles}

  /* 색상 */
  ${colorStyles}

  /* 기타 */
  & + & {
    margin-left: 1rem;
  }
`;

function Button({ children, color, size, ...rest }) {
  return (
    <StyledButton color={color} size={size} {...rest}>
      {children}
    </StyledButton>
  );
}

Button.defaultProps = {
  color: 'blue',
  size: 'medium'
};

export default Button;
```

Button(styled-components)

리팩토링

- Button 컴포넌트에 outline이라는 props를 설정하여 테두리 지정

```
import React from 'react';
import styled, { css } from 'styled-components';
import { darken, lighten } from 'polished';

const colorStyles = css`
  ${({ theme, color }) => {
    const selected = theme.palette[color];
    return css`
      background: ${selected};
      &:hover {
        background: ${lighten(0.1, selected)};
      }
      &:active {
        background: ${darken(0.1, selected)};
      }
      ${props =>
        props.outline &&
        css`
          color: ${selected};
          background: none;
          border: 1px solid ${selected};
```

```
        &:hover {
          background: ${selected};
          color: white;
        }
      `}
    `;
  }}
`;

const sizes = {
  large: {
    height: '3rem',
    fontSize: '1.25rem'
  },
  medium: {
    height: '2.25rem',
    fontSize: '1rem'
  },
  small: {
    height: '1.75rem',
    fontSize: '0.875rem'
  }
};
```

```
const sizeStyles = css`
  ${({ size }) => css`
    height: ${sizes[size].height};
    font-size: ${sizes[size].fontSize};
  `}
`;

const StyledButton = styled.button`
  /* 공통 스타일 */
  display: inline-flex;
  outline: none;
  border: none;
  border-radius: 4px;
  color: white;
  font-weight: bold;
  cursor: pointer;
  padding-left: 1rem;
  padding-right: 1rem;

  /* 크기 */
  ${sizeStyles}
```

Button(styled-components)

리팩토링

- Button 컴포넌트에 outline이라는 props를 설정하여 테두리 지정

```
  /* 색상 */
  ${colorStyles}

  /* 기타 */
  & + & {
    margin-left: 1rem;
  }
`;

function Button({ children, color, size, outline, ...rest }) {
  return (
    <StyledButton color={color} size={size} outline={outline} {...rest}>
      {children}
    </StyledButton>
  );
}

Button.defaultProps = {
  color: 'blue',
  size: 'medium'
};

export default Button;
```

Button(styled-components)

리팩토링

- Button 컴포넌트에 outline이라는 props를 설정하여 테두리 지정

```
import React from 'react';
import styled, { ThemeProvider } from 'styled-components';
import Button from './components/Button';

const AppBlock = styled.div`
  width: 512px;
  margin: 0 auto;
  margin-top: 4rem;
  border: 1px solid black;
  padding: 1rem;
`;

const ButtonGroup = styled.div`
  & + & {
    margin-top: 1rem;
  }
`;
```

```
function App() {
  return (
    <ThemeProvider
      theme={{
        palette: {
          blue: '#228be6',
          gray: '#495057',
          pink: '#f06595'
        }
      }}
    >
      <AppBlock>
        <ButtonGroup>
          <Button size="large">BUTTON</Button>
          <Button>BUTTON</Button>
          <Button size="small">BUTTON</Button>
        </ButtonGroup>
```

Button(styled-components)

리팩토링

- Button 컴포넌트에 outline이라는 props를 설정하여 테두리 지정

```
<ButtonGroup>
        <Button color="gray" size="large">
          BUTTON
        </Button>
        <Button color="gray">BUTTON</Button>
        <Button color="gray" size="small">
          BUTTON
        </Button>
      </ButtonGroup>
      <ButtonGroup>
        <Button color="pink" size="large">
          BUTTON
        </Button>
        <Button color="pink">BUTTON</Button>
        <Button color="pink" size="small">
          BUTTON
        </Button>
      </ButtonGroup>
```

```
 <ButtonGroup>
        <Button size="large" outline>
          BUTTON
        </Button>
        <Button color="gray" outline>
          BUTTON
        </Button>
        <Button color="pink" size="small" outline>
          BUTTON
        </Button>
      </ButtonGroup>
    </AppBlock>
  </ThemeProvider>
 );
}
```

Button(styled-components)

리팩토링

- fullWidth props로 버튼의 크기 100%

```
import React from 'react';
import styled, { css } from 'styled-components';
import { darken, lighten } from 'polished';

const colorStyles = css`
  ${({ theme, color }) => {
    const selected = theme.palette[color];
    return css`
      background: ${selected};
      &:hover {
        background: ${lighten(0.1, selected)};
      }
      &:active {
        background: ${darken(0.1, selected)};
      }
      ${props =>
        props.outline &&
        css`
          color: ${selected};
          background: none;
          border: 1px solid ${selected};
```

```
      &:hover {
            background: ${selected};
            color: white;
          }
        `}
    `;
  }}
`;

const sizes = {
  large: {
    height: '3rem',
    fontSize: '1.25rem'
  },
  medium: {
    height: '2.25rem',
    fontSize: '1rem'
  },
```

```
  small: {
    height: '1.75rem',
    fontSize: '0.875rem'
  }
};

const sizeStyles = css`
  ${({ size }) => css`
    height: ${sizes[size].height};
    font-size: ${sizes[size].fontSize};
  `}
`;
```

Button(styled-components)

리팩토링

- fullWidth props로 버튼의 크기 100%

```
const fullWidthStyle = css`
  ${props =>
    props.fullWidth &&
    css`
      width: 100%;
      justify-content: center;
      & + & {
        margin-left: 0;
        margin-top: 1rem;
      }
    `}
`;
const StyledButton = styled.button`
  /* 공통 스타일 */
  display: inline-flex;
  outline: none;
  border: none;
  border-radius: 4px;
  color: white;
  font-weight: bold;
  cursor: pointer;
  padding-left: 1rem;
  padding-right: 1rem;
```

```
  /* 크기 */
  ${sizeStyles}

  /* 색상 */
  ${colorStyles}

  /* 기타 */
  & + & {
    margin-left: 1rem;
  }

  ${fullWidthStyle}
`;
```

```
function Button({ children, color, size, outline, fullWidth, ...rest }) {
  return (
    <StyledButton
      color={color}
      size={size}
      outline={outline}
      fullWidth={fullWidth}
      {...rest}
    >
      {children}
    </StyledButton>
  );
}

Button.defaultProps = {
  color: 'blue',
  size: 'medium'
};

export default Button;
```

Button(styled-components)

리팩토링

- fullWidth props로 버튼의 크기 100%

```
import React from 'react';
import styled, { ThemeProvider } from 'styled-components';
import Button from './components/Button';

const AppBlock = styled.div`
  width: 512px;
  margin: 0 auto;
  margin-top: 4rem;
  border: 1px solid black;
  padding: 1rem;
`;

const ButtonGroup = styled.div`
  & + & {
    margin-top: 1rem;
  }
`;
```

```
function App() {
  return (
    <ThemeProvider
      theme={{
        palette: {
          blue: '#228be6',
          gray: '#495057',
          pink: '#f06595'
        }
      }}
    >
      <AppBlock>
        <ButtonGroup>
          <Button size="large">BUTTON</Button>
          <Button>BUTTON</Button>
          <Button size="small">BUTTON</Button>
        </ButtonGroup>
```

Button(styled-components)

리팩토링

- fullWidth props로 버튼의 크기 100%

```
<ButtonGroup>
        <Button color="gray" size="large">
          BUTTON
        </Button>
        <Button color="gray">BUTTON</Button>
        <Button color="gray" size="small">
          BUTTON
        </Button>
      </ButtonGroup>
      <ButtonGroup>
        <Button color="pink" size="large">
          BUTTON
        </Button>
        <Button color="pink">BUTTON</Button>
        <Button color="pink" size="small">
          BUTTON
        </Button>
      </ButtonGroup>
```

```
<ButtonGroup>
        <Button size="large" outline>
          BUTTON
        </Button>
        <Button color="gray" outline>
          BUTTON
        </Button>
        <Button color="pink" size="small" outline>
          BUTTON
        </Button>
      </ButtonGroup>
      <ButtonGroup>
        <Button size="large" fullWidth>
          BUTTON
        </Button>
        <Button size="large" color="gray" fullWidth>
          BUTTON
        </Button>
        <Button size="large" color="pink" fullWidth>
          BUTTON
        </Button>
      </ButtonGroup>
```

```
    </AppBlock>
    </ThemeProvider>
  );
}

export default App;
```

레포트(optional)

CSS 관련 주석 첨부
자신의 웹 페이지에 CSS 작업하기

발표주제

HOC & Hooks
React query
forceUpdate, flushSync
debouncing & throttling

Reference

- https://react.vlpt.us/styling/01-sass.html
- https://react.vlpt.us/mashup-todolist/01-create-components.html
- https://wikidocs.net/197754
- https://blog.toycrane.xyz/css%EC%9D%98-%EC%A7%84%ED%99%94-%EA%B3%BC%EC%A0%95-f7c9b4310ff7

TL;DR