# 새로운 스타일링 방법

React 환경에 맞게 컴포넌트를 스타일링하는 방법으로 CSS Modules와 Sass를 사용합니다.

## 공통 및 컴포넌트 스타일

|  |  |  |  |
| --- | --- | --- | --- |
|  | 공통 global 스타일 | 공통 기능성 스타일 | 컴포넌트 스타일 |
| 폴더 | src/scss | src/scss | src/components |
| 파일명 | global.scss | \_utils.scss | 컴포넌트명.scss |
| @import (scss) | common, utils | function, variables, mixin, sprite… | utils |
| 스타일 범위 | 전역 | 지역 | 지역 |

스타일은 크게 공통과 컴포넌트 스타일로 나눌 수 있습니다. 공통 스타일은 우리가 기존에 사용했던 common.scss 이고, 컴포넌트 스타일은 UI 각 부분을 나눈 조각(e.g. header)의 스타일입니다.

#### scss 파일 기본 구조

#### 예시 )

src /  
├─── components/  
| ├─── TodoList.jsx  
| ├─── TodoList.scss // 컴포넌트 scss 파일  
| ├─── ...  
|   
└─── scss/  
 ├─── common/   
 | ├─── \_base.scss  
 | ├─── \_common.scss  
 | └─── \_font.scss  
 |  
 ├─── utils/  
 | ├─── \_function.scss  
 | ├─── \_variables.scss  
 | └─── \_mixin.scss  
 |  
 ├─── \_common.scss // 공통 스타일 scss. common 폴더의 scss 파일  
 ├─── \_utils.scss // 공통 기능성 scss. utils 폴더의 scss 파일  
 └─── global.scss // 전체 페이지에 적용되는 scss 파일

### 공통 스타일

scss 폴더에서 관리하는 공통 스타일입니다.

* global 스타일
  + **모든 페이지에 공통적으로 적용**되는 scss 파일 모음입니다. (e.g. common, base, font 등)
  + scss에 선언한 클래스명이 변환되지 않고 그대로 사용됩니다.
  + 최상위 루트 컴포넌트에 import하여 모든 페이지에 적용되는 환경을 구성할 수 있도록 개발에 요청합니다.
* 기능성 스타일
  + **재사용**을 위한 기능성 scss 파일 모음입니다. (e.g. variables, mixin, function 등)
  + global 스타일이나 개별 컴포넌트 스타일에서 필요한 경우 import하여 사용합니다.

### 컴포넌트 스타일

* components 폴더에서 관리하는 각 **컴포넌트의 스타일**입니다.
* scss에 선언한 클래스명이 CSS Modules로 인해 변환되어 고유한 클래스명을 가집니다. (e.g. .className\_\_hashcode)
* 1:1로 매칭되는 컴포넌트와 동일한 파일명을 가집니다.
* 매칭되는 컴포넌트에 import하여 사용합니다.

## 컴포넌트와 스타일 매핑

### 기본 스타일 매핑

import React from 'react';  
import '../scss/global.scss'  
  
...

* 기존의 css 스타일링처럼 클래스 변환 없이 사용하는 경우 매핑하는 방법입니다.
* CSS Modules를 사용하지 않고, css 혹은 scss로 스타일링을 할 때 사용합니다.
* 공통 global 스타일을 매핑할 때 사용합니다. > 실습에서는 global 스타일 매핑 작업을 직접 하지만 실무에서는 개발에 요청해주세요. 혹시라도 개발에서 사용하는 외부 라이브러리에 스타일이 포함된 경우 global 스타일이 적용되지 않을 수도 있습니다. 적용되는 여러 스타일 중 global 스타일이 최상단에 올 수 있도록 개발에 요청해주세요.

### CSS Modules 스타일 매핑

import React from 'react';  
import style from './ComponentName.scss';  
  
class ComponentName extends React.Component {  
 render(){  
 return (  
 <div className={style.container}>  
 <h1 className={style.title}>title</h1>  
 ...  
 </div>  
 )  
 }  
}  
  
export default ComponentName;

* CSS Modules를 사용하고 컴포넌트에 매칭되는 scss 파일을 매핑하는 방법입니다.

## CSS Modules

* CSS를 모듈화하여 사용하는 방식으로 스타일 범위를 지역 단위로 제한합니다.
* 자동적으로 클래스명에 해시값을 추가하여 고유한 클래스명을 만듭니다. (e.g. .className\_\_hashcode)
* 다른 컴포넌트에서 같은 클래스명을 사용하더라도 클래스명이 겹치지 않습니다.

### 범위 클래스

* 전역 또는 css 파일 범위 외부의 클래스에 스타일링이 필요할 경우 사용 가능합니다.
* 범위 클래스에는 전역과 지역이 있으며 3가지 방법으로 작성할 수 있습니다.
* **전역 범위 클래스** (주로 많이 사용)
  + :global , :global(.className), :global .className
* 지역 범위 클래스
  + :local , :local(.className), :local .className

### 범위에 따른 스타일링 방법

:global 과 :local을 비교해봅시다.

#### 실습. 하나의 컴포넌트 안에서 범위 클래스를 사용하기

* JSX

render() {

return (

<div className={style.local\_A}>

<div className="global\_A">A</div>

<div className="global\_B">

<div className={style.local\_B}>B</div>

</div>

</div>

);

}

* SCSS

.local\_A {  
 background-color: skyblue;  
  
 :global(.global\_A) {  
 color: red;  
 }  
}  
  
:global {  
 .global\_B {  
 background-color: pink;  
  
 :local(.local\_B) {  
 color: blue;  
 }  
 }  
}

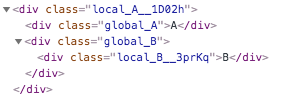
참고 사항.

1. 전역 범위 클래스는 레이아웃 스타일링 시 많이 사용합니다.

2. 지역 범위 스타일링 시 하나의 컴포넌트 내에 동일한 클래스명을 사용하지 않는 이상 클래스명이 겹치지 않기 때문에 부득이한 경우를 제외하고 보통은 2뎁스 중첩까지 사용합니다.

// 전역 범위  
:global {  
 .wrap {...}  
 .container {...}  
 .content {...}  
}

// 지역 범위  
.title {  
 font-size: 15px;  
 font-weight: 600;  
}  
  
.desc {  
 font-size: 13px  
}  
  
.box {  
 padding: 12px;  
 ...  
  
 &::before {  
 content: '';  
 ...  
 }  
}

* OUTPUT
* image
* 

## TodoList 실습

1. 공통 scss 파일 추가하기 ([소스 코드](https://oss.navercorp.com/UIT/markup-guide-react/tree/dev/todo/src/scss))
2. 공통 스타일 매핑하기
   * index.jsx
3. 컴포넌트 scss 파일 추가하기 ([소스 코드](https://oss.navercorp.com/UIT/markup-guide-react/tree/e872f25b936611453417fa686596a295b18a60f7/src/components))
4. 컴포넌트 스타일 매핑하기
   * TodoContainer.jsx
   * TodoInput.jsx
   * TodoList.jsx
   * TodoItem.jsx