9. 컴퍼넌트 스타일링

- React에서 컴퍼넌트를 스타일링할 때는 다양한 방식을 사용할 수 있다.
  1. 일발 CSS
  2. `sass` : CSS 전처리기(pre processor)중 하나로 확장된 CSS 문법을 사용하여 보다 `쉽게 작성`을 도와 준다.
  3. CSS Module : 스타일을 작성할 때 CSS클래스가 `다른 CSS클래스와 충돌하지 않도록 고유명을 자동으로 생성해 주는 옵션`
  4. styled-components : `스타일을 JavaScript파일에 내장시키는 방식`으로 `스타일을 작성함과 동시에 스타일을 적용`
- CSS를 작성할 때 가장 중요한 점은 CSS클래스를 중복되지 않게 작성하는 것이다.
- CSS중복을 막기 위해 `특별한 규칙을 사용`하는 것과 `CSS selector를 활용`하는 것이다.

9.1 일반 CSS

9.1.1 Naming Rule

- `BEM Naming`방식은 CSS방법론 중 하나로 이름을 지을 때 일종의 규칙을 준수하는 것이다.

9.1.2 CSS Selector

- CSS Selector을 사용하면 CSS 클래슥 특정 클래스 내부에 있는 경우에 스타일을 적요할 수 있다.

9.2 Sass사용하기

- `Sass(Syntactically Awesome Style Sheets)는 CSS 전처리기로 복잡한 작업을 쉽게`해 주고
- `스타일코ㅡ의 재활용성`을 `높여 주고` `코드의 가독성을 높여서 유지보수를 더욱 쉽게`해 준다.
- create-react-app 구버전에서는 별도 추가작업을 필요했지만 `V2버전부터는 별도의 추가없이 바로 사용`할 수 있다.
- `Sass는 두 가지 확장자 .scss와 .sass를 지원` 차이점은
  - .sass는 중괄호와 세미콜론을 사용하지 않는다.
  - .scss는 기존 문법과 크게 다르지 않다., 보통 scss 문법을 더 많이 사용
- 라이브러리를 설치 `yarn add sass`

src/React0901Sass.scss

```scss
// 변수선언
$red: #fa5252;
$orange: #fd7e14;
$yellow: #fcc419;
```

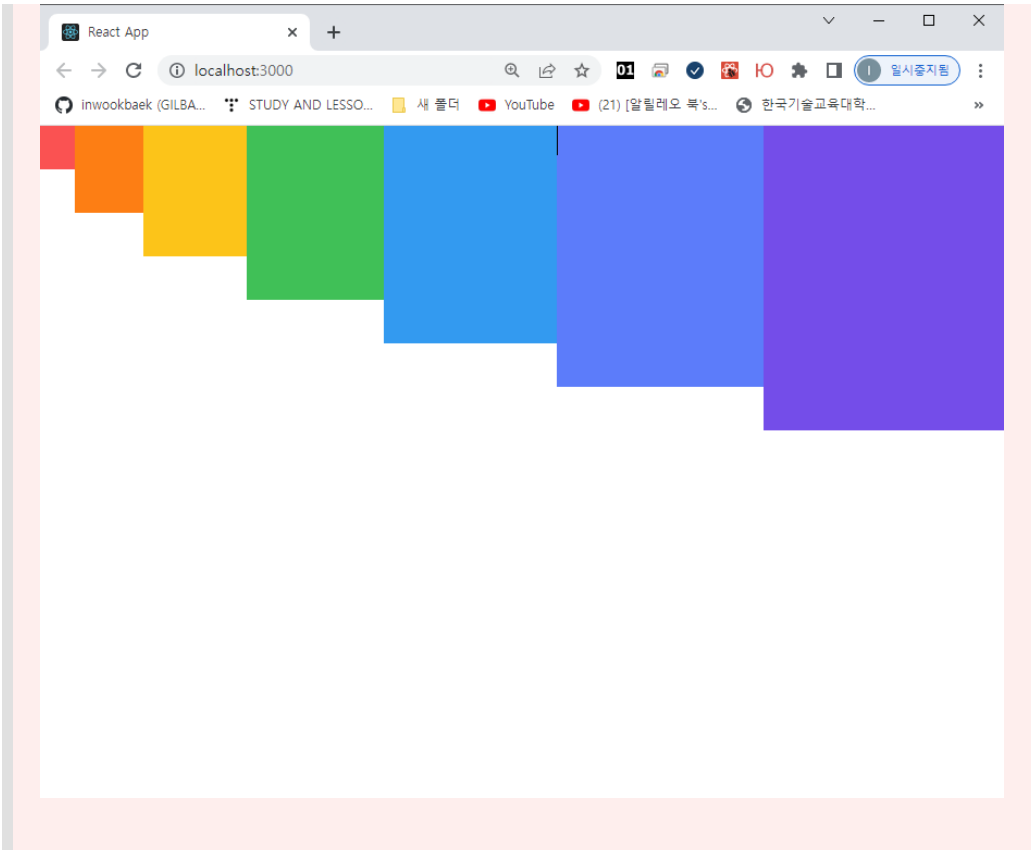
```scss
$green: #40c057;
$blue: #339af0;
$indigo: #5c7cfa;
$violet: #7950f2;

//믹스인 만들기(재사용되는 스타일 블럭을 함수처럼 사용가능
@mixin square($size) {
  $calculated: 32px * $size;
  width: $calculated;
  height: $calculated;
}

.React0901Sass {
  display: flex;
  .box {
    background: red; // 일반 CSS 에선 .SassComponent .box 와 마찬가지
    cursor: pointer;
    transition: all 0.3s ease-in;
    &.red {
      // .red 클래스가 .box 와 함께 사용 됐을 때
      background: $red;
      @include square(1);
    }
    &.orange {
      background: $orange;
      @include square(2);
    }
    &.yellow {
      background: $yellow;
      @include square(3);
    }
    &.green {
      background: $green;
      @include square(4);
    }
    &.blue {
      background: $blue;
      @include square(5);
    }
    &.indigo {
      background: $indigo;
      @include square(6);
    }
    &.violet {
      background: $violet;
      @include square(7);
    }
    &:hover {
      // .box 에 마우스 올렸을 때
      background: black;
    }
  }
}
```

mysrc/React0901Sass.js

```
import './React0901Sass.scss'

const React0901Sass = () => {
  return (
    <div className="React0901Sass">
      <div className="box red" />
      <div className="box orange" />
      <div className="box yellow" />
      <div className="box green" />
      <div className="box blue" />
      <div className="box indigo" />
      <div className="box violet" />
    </div>
  );
};

export default React0901Sass;
```

App.js

```
import React, { Component } from 'react'
import React0901Sass from './mysrc/React0901Sass'

class App extends Component {
  render() {
    return (
      <div>
        <React0901Sass />
      </div>
    )
  }
}

export default App;
```



> 작업한 뒤에 개발서버를 재시작해야 Sass가 성공적으로 적용된다.

9.2.1 utils함수 분리하기

- Sass변수 및 믹스인은 다른 파일로 분리작성후 공유할 수 있다.

mysrc/styles/utils.scss

```scss
// 변수선언
$red: #fa5252;
$orange: #fd7e14;
$yellow: #fcc419;
$green: #40c057;
$blue: #339af0;
$indigo: #5c7cfa;
$violet: #7950f2;

//믹스인 만들기(재사용되는 스타일 블럭을 함수처럼 사용가능
@mixin square($size) {
  $calculated: 32px * $size;
  width: $calculated;
  height: $calculated;
}
```

mysrc/styles/React0901Sass.scss

```scss
// 변수와 mixin을 utils.scss로 분리

.React0901Sass {
  display: flex;
  .box {
    background: red; // 일반 CSS 에선 .SassComponent .box 와 마찬가지
    cursor: pointer;
    transition: all 0.3s ease-in;
    &.red {
      // .red 클래스가 .box 와 함께 사용 됐을 때
      background: $red;
      @include square(1);
    }
    &.orange {
      background: $orange;
      @include square(2);
    }
    &.yellow {
      background: $yellow;
      @include square(3);
    }
    &.green {
      background: $green;
      @include square(4);
    }
    &.blue {
      background: $blue;
      @include square(5);
    }
    &.indigo {
      background: $indigo;
```

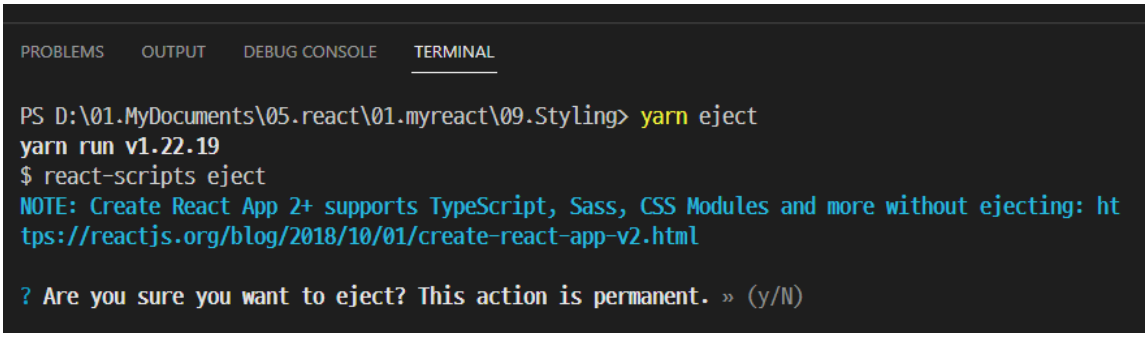
```
      @include square(6);
    }
    &.violet {
      background: $violet;
      @include square(7);
    }
    &:hover {
      // .box 에 마우스 올렸을 때
      background: black;
    }
  }
}
```

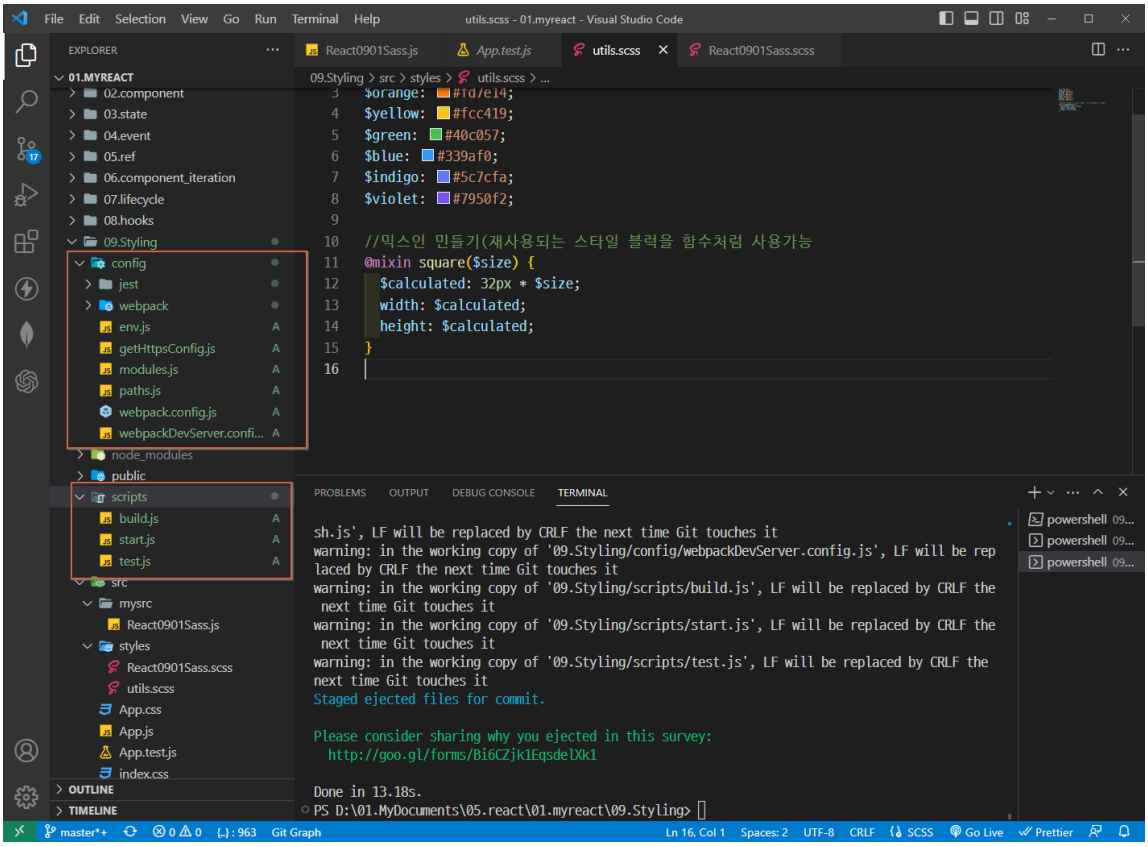
9.2.2 sass-loader 설정 커스터마이징

- 이 작업은 반드시 해야 되는 것은 아니지만 설정 해두면 유용하다.
- 디렉토리 구조가 깊어졌다면 상위폴더로 한참 거슬러 올라가야 하는 단점이 있다
- 이 경우에 `웹펙에서 sass를 처리하는 sass-loader로 해결할 수 있다.`
- `create-react-app은 구조의 복잡도를 낮추기 위해 세부설정이 모두 숨겨져 있다.`
- 이를 커스터마이징을 하려면 `프로젝트 디렉토리에서 yarn eject 명령어를 통해 새부설정을 밖으로 꺼내 주어야 한다.
- create-react-app은 기본적으로 git 설정이 되어 있는데
- `yarn eject는 commit되지 않은 내용이 있다면 진행되지 않으니 먼저 commit해야한다.`

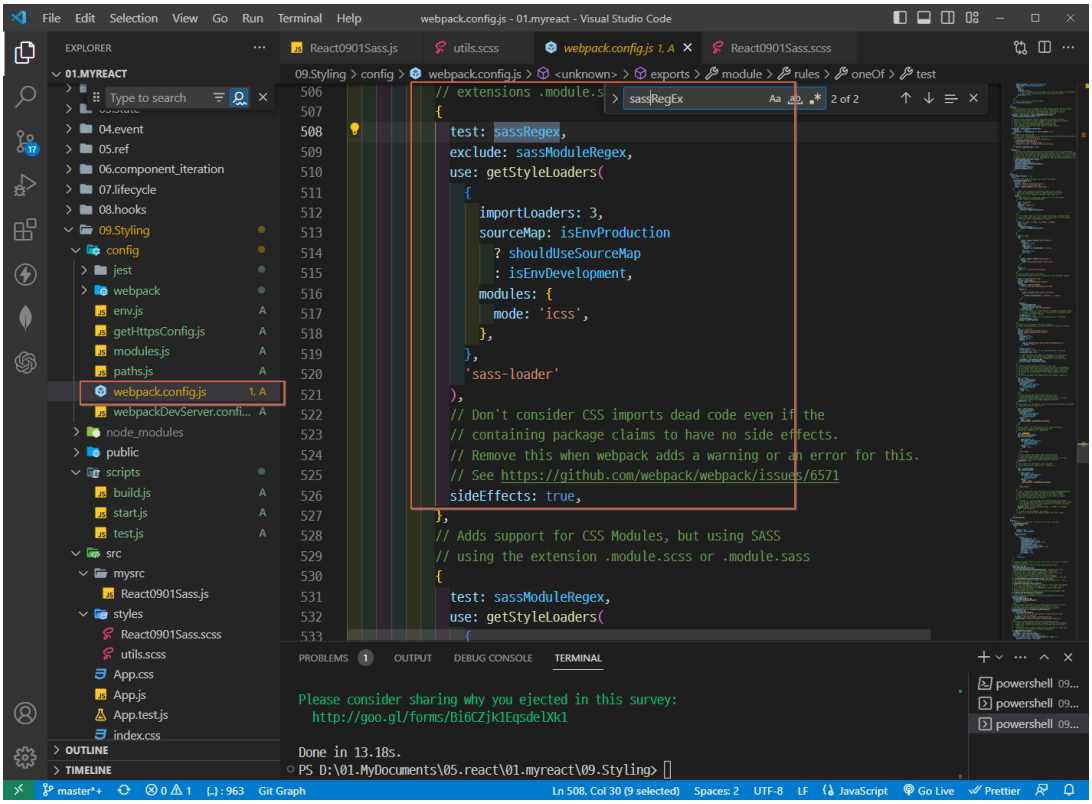
  > - git add.
  > - git commit -m 'Commit before yarn eject'

- commit후 `yarn eject` 명령어 실행





- 프로젝트에 `config폴더 생성확인` 후 `webpack.config.js`를 open



- `sass-loader부분을 아래와 같이 수정`

config/webpack.config.js 변경전

```
{
              test: sassRegex,
              exclude: sassModuleRegex,
              use: getStyleLoaders(
                {
                  importLoaders: 3,
                  sourceMap: isEnvProduction
                    ? shouldUseSourceMap
                    : isEnvDevelopment,
                  modules: {
                    mode: 'icss',
                  },
                },
                'sass-loader'
              ),
              sideEffects: true,
            },
```

config/webpack.config.js 변경후

```
{
              test: sassRegex,
              exclude: sassModuleRegex,
              use: getStyleLoaders({
                  importLoaders: 3,
                  sourceMap: isEnvProduction
                    ? shouldUseSourceMap
                    : isEnvDevelopment,
                  modules: {
                    mode: 'icss',
                  }
              }).concat({
                loader: require.resolve('sass-loader'),
                options: {
                  sassOptions: {
                    includePaths: [paths.appSrc + '/styles'],
                  },
                  additionalData: "@import 'utils';",
                },
              }),
              sideEffects: true
            },
```

```
// extensions .module.scss or .module.sass
{
  test: sassRegex,
  exclude: sassModuleRegex,
  use: getStyleLoaders({
    importLoaders: 2,
    sourceMap: isEnvProduction && shouldUseSourceMap
  }).concat({
    loader: require.resolve('sass-loader'),
    options: {
      includePaths: [paths.appSrc + '/styles'],
      sourceMap: isEnvProduction && shouldUseSourceMap,
      data: `@import 'utils';`
    }
  }),
  // Don't consider CSS imports dead code even if the
  // containing package claims to have no side effects.
  // Remove this when webpack adds a warning or an error for this.
  // See https://github.com/webpack/webpack/issues/6571
  sideEffects: true
},
```

- 설정파일을 저장후 시스템 재시작하면
- `utils.scss파일을 불러올 때 별도 정의할 필요 없이 @import 'utils.scss'와 같이 바로 불러올 수 있다.`
- React0901Sass.scss.scss에서 `@import 'utils.scss;` 로 수정후 정상작동여부를 확인*
- 새 파일을 생성할 때마다 utils.scss를 매번 포함시키는 것이 귀찮을 경우 `additionalData: "@import 'utils';",` 옵션을 설정
- 이렇게 설정하면 `sass파일을 불러올 때마다 코드의 맨 윗 부분에 특정 코드를 포함` 시켜 준다.

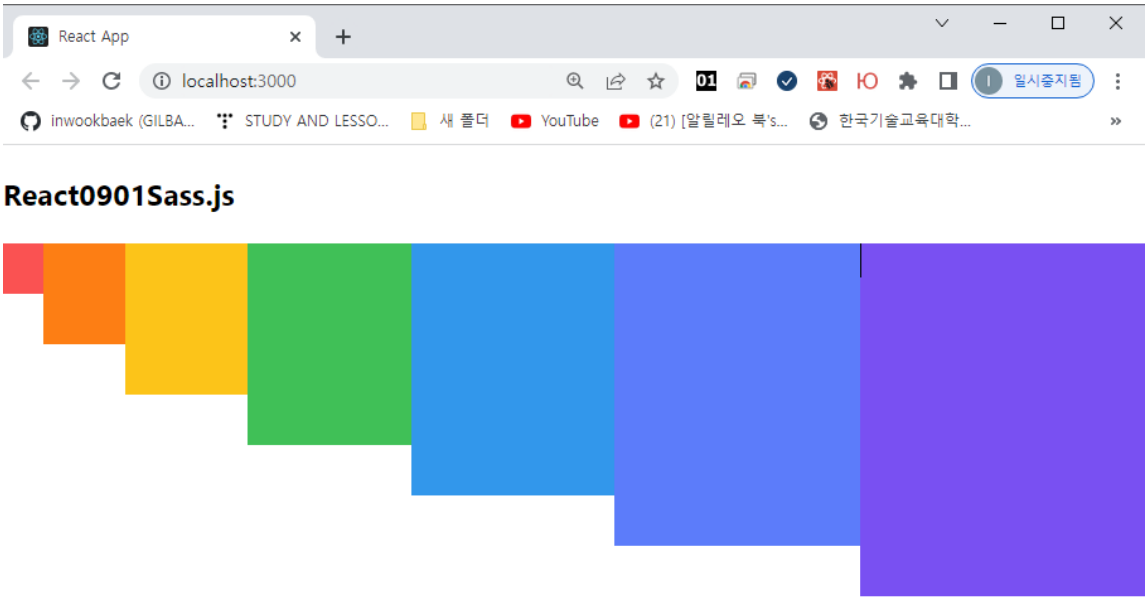
`sass 설정시 테스트 순서에 주의할 것`

> 1. yarn eject 적용전에는 React0901Sass.scss에 `@import '../styles/utils';` 를 추가후에 테스트
> 2. webpack.config.js을 additionalData 옵션 미설정시에는 React0901Sass.scss에 `@import 'utils';` 를 추가후 테스트(폴더설정없음)
> 3. webpack.config.js을 additionalData 옵션 미설정시에는 React0901Sass.scss에 `utils 설정없이 테스트` 한 후
>
> - 서버를 재시작하면 정상적으로 실행된다.

React0901Sass.js

```
import './React0901Sass.scss'

const React0901Sass = () => {
  return (
    <div>
      <h3>React0901Sass.js</h3>
```

```
        <div className="React0901Sass">
          <div className="box red" />
          <div className="box orange" />
          <div className="box yellow" />
          <div className="box green" />
          <div className="box blue" />
          <div className="box indigo" />
          <div className="box violet" />
        </div>
      </div>
    );
};

export default React0901Sass;
```



9.2.3 node_modules에서 라이브러리 불러오기

- sass장점 중 하나는 라이브러리를 쉽게 불러와서 사용할 수 있다는 점이다.
- 디렉토리 구조가 복잡할 경우 `@import '~library/styles';` 처럼 물결문자(~)를 사용 한다.
- 물결문자를 사용하면 `자동으로 node_modules에서 라이브러리 디렉토리를 탑지` 하여 불러올 수 있다.
- 연습삼아 유용한 라이브러리를 설치 후 테스트
  - 반응형 디자인을 쉽게 만들어 주는 `include-media(https://include-media.com)`

- 편리한 색상 팔레트인 `open-color(https://www.npmjs.com/package/open-color)`
- `yarn add open-color include-media`
- utils.scss 코드 상단에
  - `@import '~include-media/dist/include-media';`
  - `@import '~open-color/open-color';`

styles/utils.scss

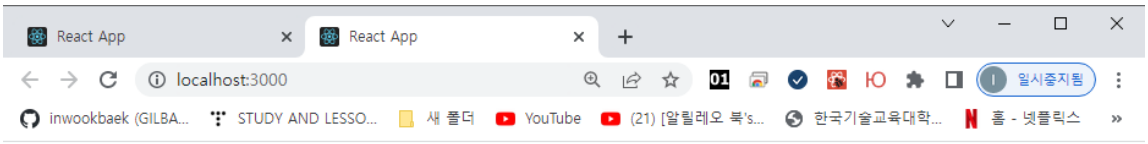
```scss
@import '~include-media/dist/include-media';
@import '~open-color/open-color';

// 변수선언
$red: #fa5252;
$orange: #fd7e14;
$yellow: #fcc419;
$green: #40c057;
$blue: #339af0;
$indigo: #5c7cfa;
$violet: #7950f2;

//믹스인 만들기(재사용되는 스타일 블럭을 함수처럼 사용가능
@mixin square($size) {
  $calculated: 32px * $size;
  width: $calculated;
  height: $calculated;
}
```

- include-media와 open-color 적용, 화면 크기에 따라 반응형

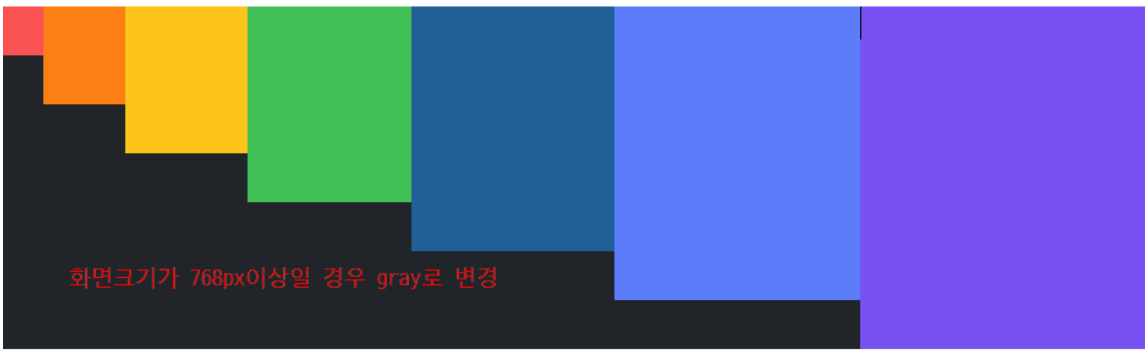
React0901Sass.scss

```scss
// @import '../styles/utils';
// @import 'utils';

.React0901Sass {
  display: flex;
//================================================
  background: $oc-gray-2;
  @include media('<768px') {
    background: $oc-gray-;
  }
//================================================
  .box {
    background: red; // 일반 CSS 에선 .SassComponent .box 와 마찬가지
    cursor: pointer;
    transition: all 0.3s ease-in;
    &.red {
      // .red 클래스가 .box 와 함께 사용 됐을 때
      background: $red;
      @include square(1);
    }
    &.orange {
      background: $orange;
      @include square(2);
    }
    &.yellow {
```

```
        background: $yellow;
        @include square(3);
      }
      &.green {
        background: $green;
        @include square(4);
      }
      &.blue {
        background: $blue;
        @include square(5);
      }
      &.indigo {
        background: $indigo;
        @include square(6);
      }
      &.violet {
        background: $violet;
        @include square(7);
      }
      &:hover {
        // .box 에 마우스 올렸을 때
        background: black;
      }
    }
  }
```



React0901Sass.js



9.3 CSS Module

- CSS module은 CSS를 불러와서 사용할 때 `클래스 이름을 고유한 값, 즉 [파일이름]_[클래스이름]__[해시값] 형태로 자옹 생성`
- 자동 생성되어 `컴퍼넌트 스트일 클래스 이름이 중복되는 현상을 방지해 주는 기술`
- CSS module을 사용하기 위해 구버전에서는 별도로 설정해야 했지만 v2부터는 create-react-app에 포함
- 별도 설정 없이 `.module.css확장자로 파일을 저장하기만 하면 CSS Module이 적용 된다.`

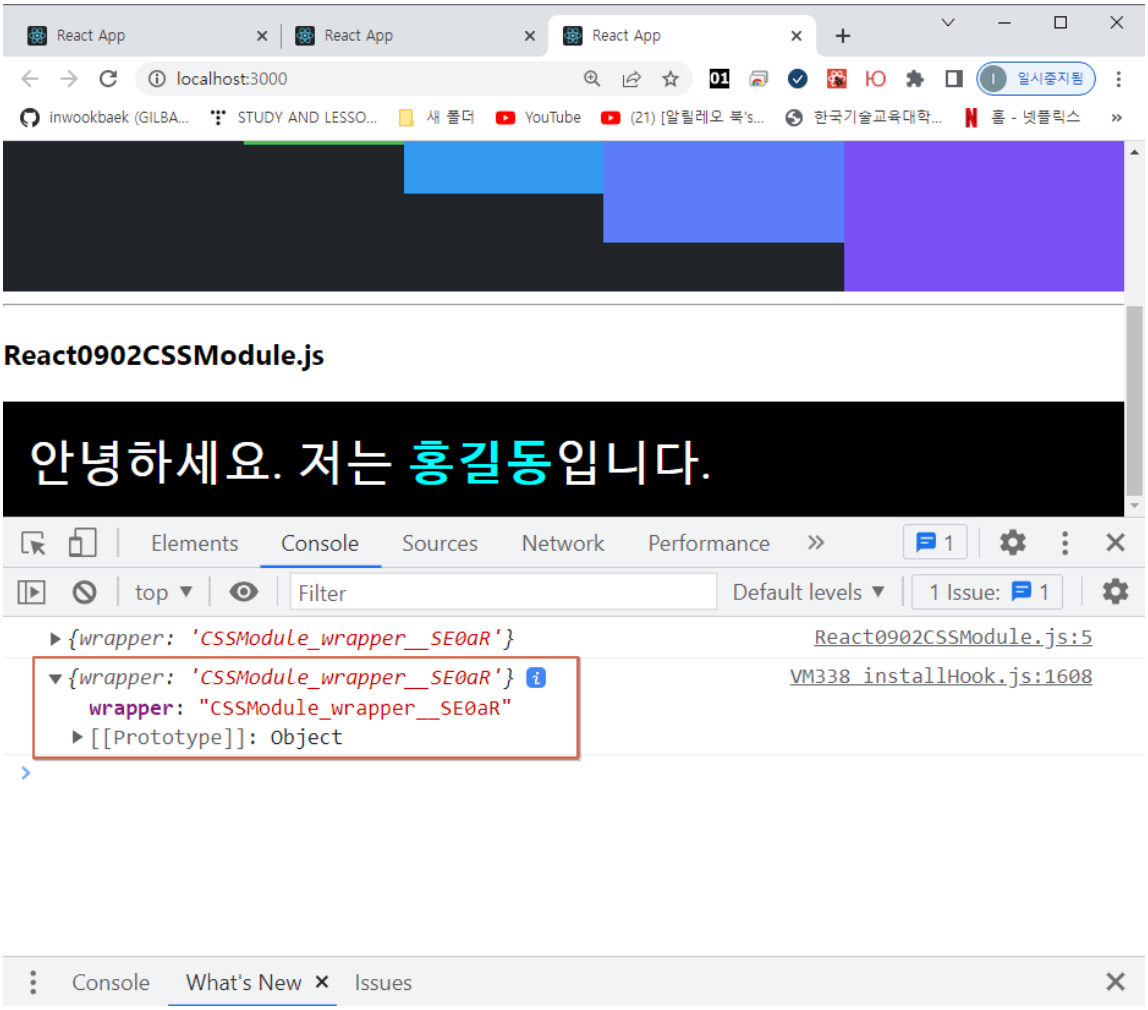
styles/React0902CSSModule.module.css

```css
/* 자동으로 고유해질 것이므로 흔히 사용되는 단어를 클래스 이름으로 마음대로 사용가능*/

.wrapper {
  background: black;
  padding: 1rem;
  color: white;
  font-size: 2rem;
}

/* 글로벌 CSS 를 작성하고 싶다면 */
:global .something {
    font-weight: 800;
    color: aqua;
}
```

- CSS Module을 사용하면 클래스이름을 명명할 때 중복을 고민하지 않아도 된다.
- `웹페이지에 전역적으로 사용되는 경우라면 :global을 팦에 입력 하면 글로벌 css 임을 명시해 줄 수 있다.`

React0902CSSModule.js

```jsx
import styles from '../styles/CSSModule.module.css';

const React0902CSSModule = () => {

  console.log(styles);
  return (
    <div>
      <h3>React0902CSSModule.js</h3>
        <div className={styles.wrapper}>
          안녕하세요. 저는 <span className="something">홍길동</span>입니다.
        </div>
    </div>
  );
};
export default React0902CSSModule;
```



- CSS Module에서 사용한 클래스 이름과 해당 이름을 고유한 값이 key-value형태로 저장되어 있다.
- 클래스이름을 2개 이상 적용할 때는 다음과 같이 코드를 작성하면 된다.

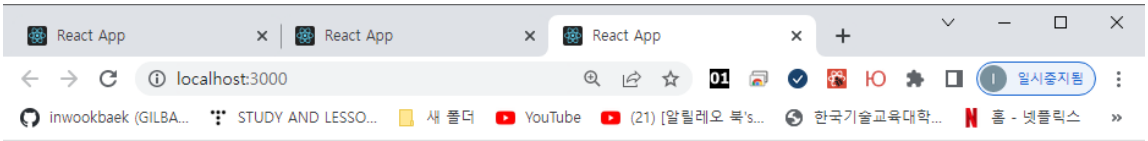
styles/React0902CSSModule.module.css

```
/* 자동으로 고유해질 것이므로 흔히 사용되는 단어를 클래스 이름으로 마음대로 사용가능*/

.wrapper {
  background: black;
  padding: 1rem;
  color: white;
  font-size: 2rem;
}

.inverted {
  color: black;
  background: white;
  border: 1px solid black;
}

/* 글로벌 CSS 를 작성하고 싶다면 */
:global .something {
    font-weight: 800;
    color: aqua;
}
```

React0902CSSModule.js

- 탬플릿 리터럴(백틱)을 사용하지 않을 경우 배열로 `{ [styles.wrapper, styles.inverted].join(' ') }` 정의하면 된다.

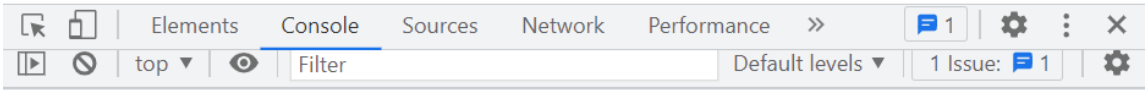
```
import styles from '../styles/CSSModule.module.css';

const React0902CSSModule = () => {

  console.log(styles);
  return (
    <div>
      <h3>React0902CSSModule.js</h3>
        <div className={`${styles.wrapper} ${styles.inverted}`}>
          안녕하세요. 저는 <span className="something">홍길동</span>입니다.
        </div>
    </div>
  );
};
export default React0902CSSModule;
```



React0901Sass.js

React0902CSSModule.js

안녕하세요. 저는 홍길동입니다.



9.3.1 classnames

- `classnames는 CSS 클래스를 조건부로 설정할 때 유용한 라이브러리`이다.
  - 라이브러리 설치 : `yarn add classnames`

classnames 간략 사용법

- 다양한 종류의 파라미터를 조합해서 css클래스를 설정

```
import classNames from 'classnames';

classNames('one' 'two;); // = 'one two'
classNames('one' {'two: true}); // = 'one two'
classNames('one' {'two: false}); // = 'one'
classNames('one' ['two', 'three']); // = 'one two three'

const myClass = 'hello';
classNames('one', myClass, { myCondigion: true }); //= 'one hello myCondition'
```

- 조건부로 클래스를 설정할 때 편리

```
const MyComponent = ({highlighted, theme}) => {
    <div className={classNames('MyComponent', { highlighted}, theme)
}>Hello</div>
}
```

- CSS Module과 함께 사용하면 CSS Module이 훨씬 사용이 쉬워진다
- `classnames에 내장되어 있는 bind함수를 사용`하면 클래스를 넣어 줄 때 마다 styles.[클래스이름]형태로 사용할 필요가 없다.
- 사전에 styles를 받아 온 후 사용하게 설정해 두고 cs('클래스이름1', '클래스이름2',...)형태로 사용할 수 있다.

React0902CSSModule.js에 classnames.bind()함수 적용

```
import classNames from 'classnames/bind';
import styles from '../styles/CSSModule.module.css';

const cx = classNames.bind(styles); // 사전에 styles에서 클래스 받아 오도록 설정

const React0902CSSModule = () => {

  console.log(styles)
  return (
    <div>
      <h3>React0902CSSModule.js</h3>
        <div className={ cx('wrapper', 'inverted') }>
          안녕하세요. 저는 <span className="something">홍길동</span>입니다.
        </div>
    </div>
  );
};

export default React0902CSSModule;
```

9.3.2 Sass와 함께 사용하기

- CSSModule.module.css을 CSSModule.module.scss로 복사 후 수정

CSSModule.module.scss

```
.wrapper {
  background: black;
  padding: 1rem;
  color: white;
  font-size: 2rem;
```

```
  .inverted { // invertd가 wrapper아 같이 사용되었을 때만 적용
    color: black;
    background: white;
    border: 1px solid black;
  }
}



/* 글로벌 CSS 를 작성하고 싶다면 */
:global {
  // global로 감싸기
  .something {
    font-weight: 800;
    color: aqua;
  }
  // 다른 클래스 추가 가능
}
```

React0902CSSModule.js

```
import classNames from 'classnames/bind';
// ../styles/CSSModule.module.css -> ../styles/CSSModule.module.scss로 변경
// import styles from '../styles/CSSModule.module.css';
import styles from '../styles/CSSModule.module.scss';

const cx = classNames.bind(styles); // 사전에 styles에서 클래스 받아 오도록 설정

const React0902CSSModule = () => {

  console.log(styles)
  return (
    <div>
      <h3>React0902CSSModule.js</h3>
        <div className={ cx('wrapper', 'inverted') }>
          안녕하세요. 저는 <span className="something">홍길동</span>입니다.....
        </div>
    </div>
  );
};

export default React0902CSSModule;
```

9.3.3 CSS Module이 아닌 파일에서도 CSS Module 사용하기

- CSS Module이 아닌 일반 `.css/.scss`에서도 `:local`을 사용하여 CSS Module을 사용할 수 있다.

```
:local .wrapper { ... 스타일 ... }

or

:local {
   .wrapper {
       ... style ...
```

```
    }
}
```

9.4 styled-components

- styled-components는 `자바스크립트 파일 안에 스타일을 선언하는 방식`
- 이 방식을 `CSS-in-JS`라고 한다. 참고는 https://github.com/MicheleBertoli/css-in-js에서 확인 가능
  - styled-components를 대체할 라이브러리는 `emotion`이 있다.
- JavaScript파일 하나에 스타일까지 작성할 수 있기 때문에 `.css or .scss등 스타일 파일을 별도로 만들지 않아도 되는 이점`이 있다.

> - 설치 : `yarn add styled-components`

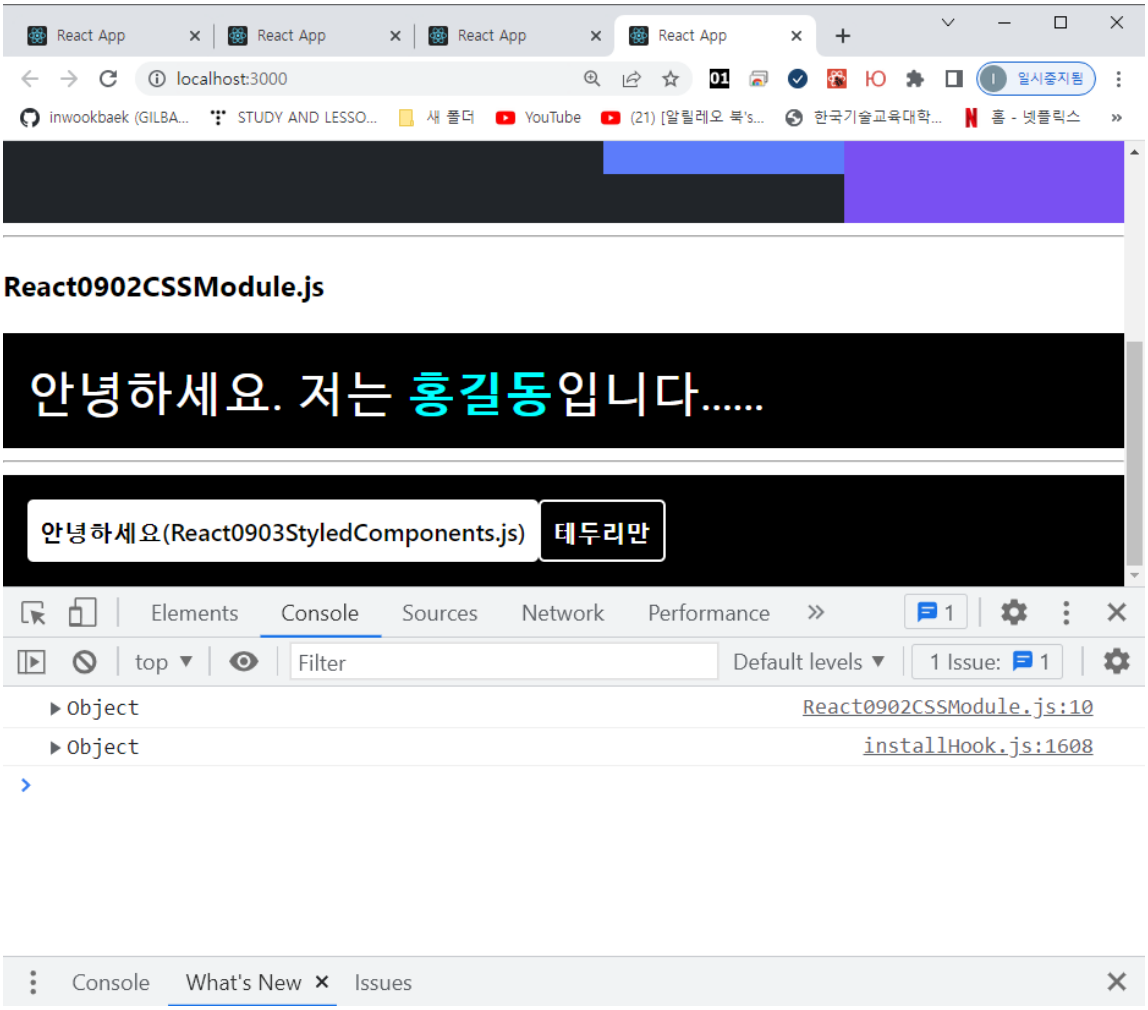
React0903StyledComponents.js

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

const Box = styled.div`
  /* props 로 넣어준 값을 직접 전달해줄 수 있습니다. */
  background: ${props => props.color || 'blue'};
  padding: 1rem;
  display: flex;
`;

const Button = styled.div`
  background: white;
  color: black;
  border-radius: 4px;
  padding: 0.5rem;
  display: flex;
  align-items: center;
  justify-content: center;
  box-sizing: border-box;
  font-size: 1rem;
  font-weight: 600;

  /* & 문자를 사용하여 Sass 처럼 자기 자신 선택 가능 */
  &:hover {
    background: rgba(255, 255, 255, 0.9);
  }

  /* 다음 코드는 inverted 값이 true 일 때 특정 스타일을 부여해줍니다. */
  ${props =>
    props.inverted &&
    css`
      background: none;
      border: 2px solid white;
      color: white;
      &:hover {
        background: white;
        color: black;
      }
    `};
```

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

const React0903StyledComponents = () => (
  <Box color="black">
    <Button>안녕하세요(React0903StyledComponents.js)</Button>
    <Button inverted={true}>테두리만</Button>
  </Box>
);

export default React0903StyledComponents;
```

App.js

```
import React, { Component } from 'react'
import React0901Sass from './mysrc/React0901Sass'
import React0902CSSModule from './mysrc/React0902CSSModule'
import React0903StyledComponents from './mysrc/React0903StyledComponents';

class App extends Component {
  render() {
    return (
      <div>
        <React0901Sass />
        <hr/>
        <React0902CSSModule />
        <hr/>
        <React0903StyledComponents />
      </div>
    )
  }
}

export default App;
```



- styled-components를 사용하는 가장 큰 장점은 props 값으로 전달해 주는 값을 쉽게 스타일에 적용할 수 있다는 점이다.
- styled-components을 syntex highlighting문법을 적용하기 위해서는 vscode-tyled-components 확장팩 설치

9.4.1 Tagged 템플릿 리터럴

- Tagged Template Literal을 사용하면 자바스크립트객체나 함수원본값을 그대로 추출할 수 있다.
- styled components는 이러한 속성을 사용하여 컴퍼넌트의 props를 쉽게 조회할 수 있도록 한다. ```js function tagged(...args) { consloe.log(args); }

tagged `hello ${{foo: 'bar}} ${() => 'world}`

9.4.2 스타일링된 엘리먼트 만들기

* styled components를 사용하여 스타일링된 엘리먼트를 만들 때는 컴퍼넌트파일의 상단에 styled를 로디하고
* `styled.태그명`을 사용하여 구현한다

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

```
const Box = styled.div`
  /* props 로 넣어준 값을 직접 전달해줄 수 있습니다. */
  background: ${props => props.color || 'blue'};
  padding: 1rem;
  display: flex;
`;
```

- 이렇게 styled.div뒤에 Tagged Template literal 문법을 통해 스타일을 넣어주면
- 해당 스타일이 적용된 div로 이러어진 react component가 생성된다.
- 그래서 후에 `Hello형태로 사용할 수 있다.
- 태그명이 유동적이거나 특정컴퍼넌트 자체에 스타일을 할 경우

```
const MyInput = styled('input')`
    backgroudL gray;
`
```

또는

```
const StyledLink = styled(Link) `
    color: blud;
`
```

- 여기에서 Link 컴퍼넌트는 react-router의 컴퍼넌트이다.

9.4.3 스타일에서 props 조회하기

```
//<Box color="black">(...)</Box>에서 전달된 props의 color값을 전달한다.

const Box = styled.div`
  /* props 로 넣어준 값을 직접 전달해줄 수 있습니다. */
  background: ${props => props.color || 'blue'};
  padding: 1rem;
  display: flex;
`;
```

- ${props => props.color || 'blue'};의미는 props를 조회해서 값이 있으면 color을 적용, 없으면 blue를 적용

9.4.5 props에 따른 조건부 스타일링

- props를 사용하여 서로 다른 스타일을 적용할 수 있다.

```
<Button>안녕하세요</Button>
<Button inverted={true}>테두리만</Button>
```

- Tagged Template를 사용하지 않는 다면 문자열로 간주가 되기 때문에 함수를 받아 사용하지 못한다는 것이다.

9.4.5 반응형디자인

React0903StyledComponents.js

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

const sizes = {
  desktop: 1024,
```

```
  tablet: 768
};

// 위에있는 size 객체에 따라 자동으로 media 쿼리 함수를 만들어줍니다.
// 참고: https://www.styled-components.com/docs/advanced#media-templates
const media = Object.keys(sizes).reduce((acc, label) => {
  acc[label] = (...args) => css`
    @media (max-width: ${sizes[label] / 16}em) {
      ${css(...args)};
    }
  `;

  return acc;
}, {});

const Box = styled.div`
  /* props 로 넣어준 값을 직접 전달해줄 수 있습니다. */
  background: ${props => props.color || 'blue'};
  padding: 1rem;
  display: flex;
  width: 1024px;
  margin: 0 auto;
  ${media.desktop`width: 768px;`}
  ${media.tablet`width: 100%;`};
`;

const Button = styled.button`
  background: white;
  color: black;
  border-radius: 4px;
  padding: 0.5rem;
  display: flex;
  align-items: center;
  justify-content: center;
  box-sizing: border-box;
  font-size: 1rem;
  font-weight: 600;

  /* & 문자를 사용하여 Sass 처럼 자기 자신 선택 가능 */
  &:hover {
    background: rgba(255, 255, 255, 0.9);
  }

  /* 다음 코드는 inverted 값이 true 일 때 특정 스타일을 부여해줍니다. */
  ${props =>
    props.inverted &&
    css`
      background: none;
      border: 2px solid white;
      color: white;
      &:hover {
        background: white;
        color: black;
      }
    `};
  & + button {
    margin-left: 1rem;
```

```
  }
`;

const React0903StyledComponents = () => (
  <Box color="black">
    <Button>안녕하세요(React0903StyledComponents.js)</Button>
    <Button inverted={true}>테두리만</Button>
  </Box>
);

export default React0903StyledComponents;
```