9. 컴포넌트 스타일링

작성일시: 2022년 5월 10일 오후 2:50 참고 도서: 리액트를 다루는 기술

9.0 컴포넌트 스타일링 방식

- 여러가지 방식 중에 딱히 정해진 방식이 없습니다.

일반 CSS	컴포넌트를 스타일링하는 가장 기본적인 방식입니다.
SCSS	자주 사용되는 CSS 전처리기(pre-processor)중 하나로 확장된 CSS 문법을 사용하여 CSS코드를 더욱 쉽게 작성할 수 있도록 해줍니다.
CSS Module	스타일을 작성할 때 CSS 클래스가 다른 CSS 클래스의 이름과 절대 충돌하지 않도록 파일마다 고유한 이름을 자동으로 생성해주는 옵션입니다.
Styled-components	스타일을 자바스크립트 파일에 내장시키는 방식으로 스타일을 작성함과 동시에 해당 스타일이 적용된 컴포넌트를 만들 수 있게 해 줍니다.

9.1 가장 흔한 방식, 일반 CSS

기본 파일

App.js

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

App.css

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

9.1.1 이름 짓는 규칙

BEM 네이밍은 CSS 방법론 중 하나임으로, 본인만의 네이밍 규칙 혹은 메이저 업체의 네이밍 코딩 규칙 문서를 통해 습득하는 것이 좋다.

9.2 Sass사용하기

Sass는 스타일 코드의 재활용성을 높여준다. Sass 에성는 .sass .scss 를 지원한다 하지만, 요즘은 Sass 보다 Scss 사용을 권장한다.

.sass

```
$font-stack:Helvwtica, sans-serif
$primary-colo: #333

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

.scss

```
$font-stack:Helvwtica, sans-serif;
$primary-colo: #333;

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

> .sass는 중괄호 ({ })와 세미콜론(;)을 사용하지 않는다. 반면 .scss는 기존 css와 작성하는 방식이 크게 다르지 않다.

Sass 설치 방법

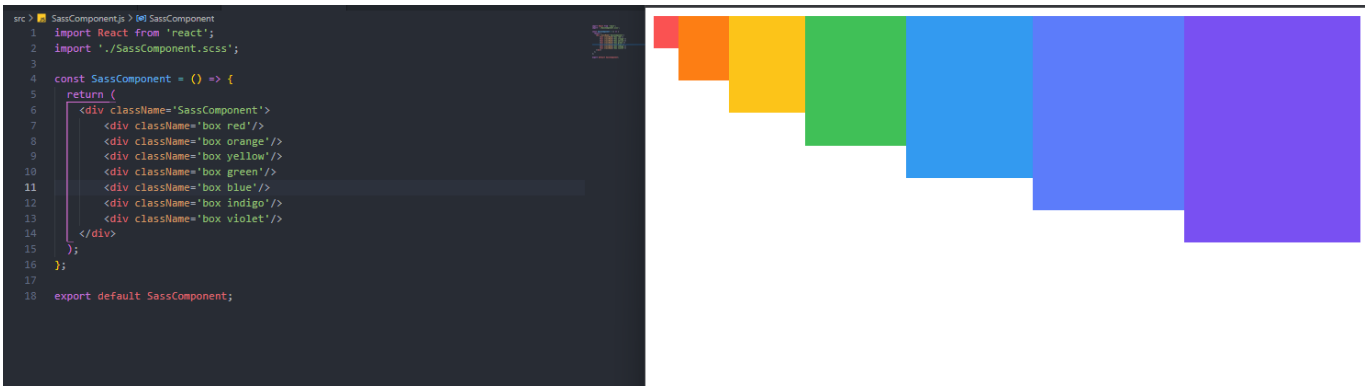
터미널에서 사용하는 디렉토리 위치에 아래 코드를 작성해주면 된다.

$ yarn add sass

SassComponent.scss

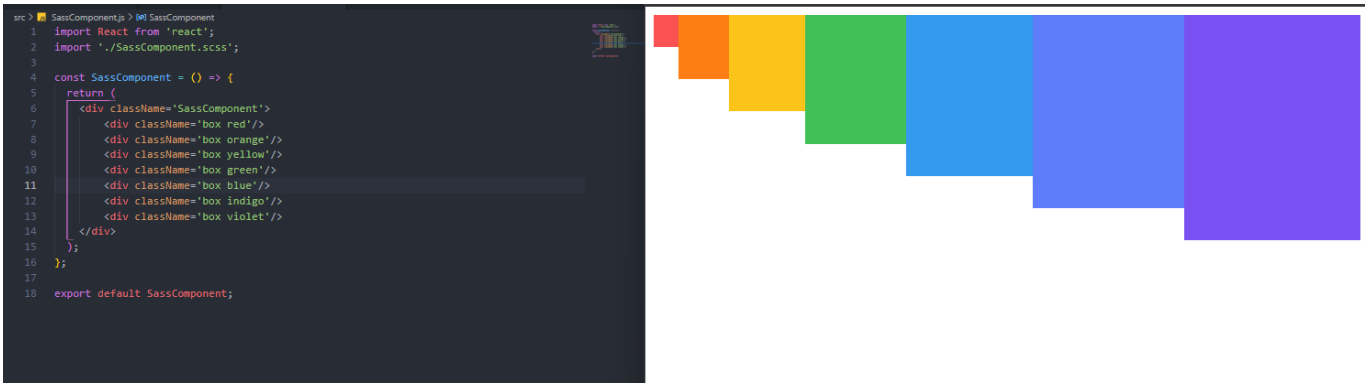
```
//변수 사용하기
$red: #fa5252;
$orange: #fd7e14;
$yellow: #fcc419;
$green: #40c057;
$blue: #339af0;
$indigo: #5c7cfa;
$violet: #7950f2;

//믹스인 말들기(재사용되는 스타일 블록을 함수처럼 사용할 수 있음)
@mixin square($size) {
    $calculated: 32px * $size;
    width: $calculated;
    height: $calculated;
}
```

```scss
.SassComponent{
    display: flex;
    .box{//일반 CSS에서는 .SassComponent .box와 마찬가지로
        background:red;
        cursor: pointer;
        transition: all 0.3s ease-in;
        &.red{
            //.red 클래스가  .box와 함께 사용되었을 때
            background: $red; //미리 정의 해놓은 변수 사용
            @include square(1); // 함수호출
        }
        &.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{
            //.box에 마우스를 올렸을 때
            background: black;
        }
    }
}
```



SassComponenet.js

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

const SassComponent = () => {
  return (
    <div className='SassComponent'>
        <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 SassComponent;
```



App.js

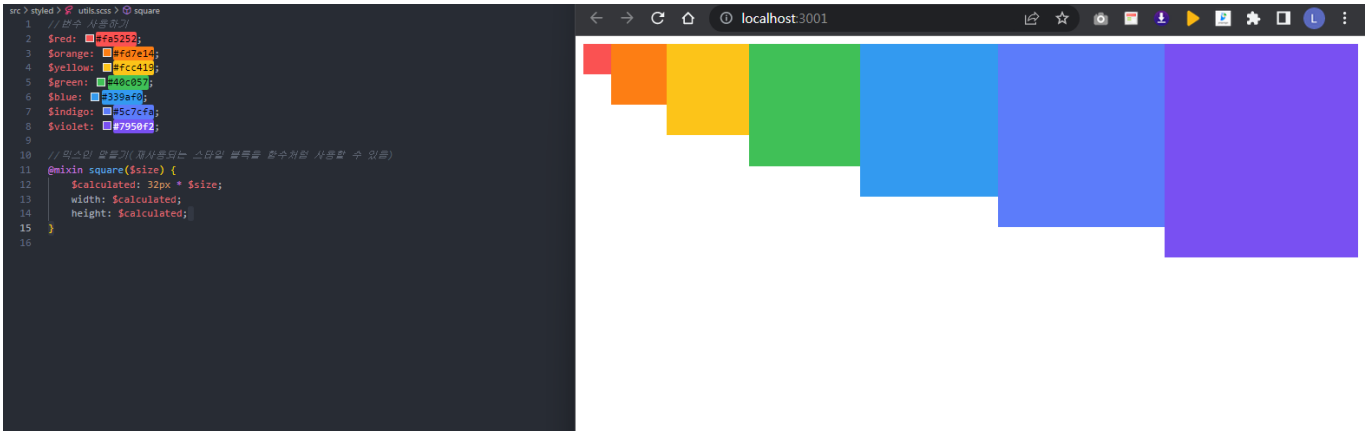
```
import React from 'react';
import SassComponent from './SassComponent';

const App = () => {
  return (
```

```
      <div>
        <SassComponent/>
      </div>
    )
}

export default App
```

9.2.1 utils 함수 분리하기

ultils.scss

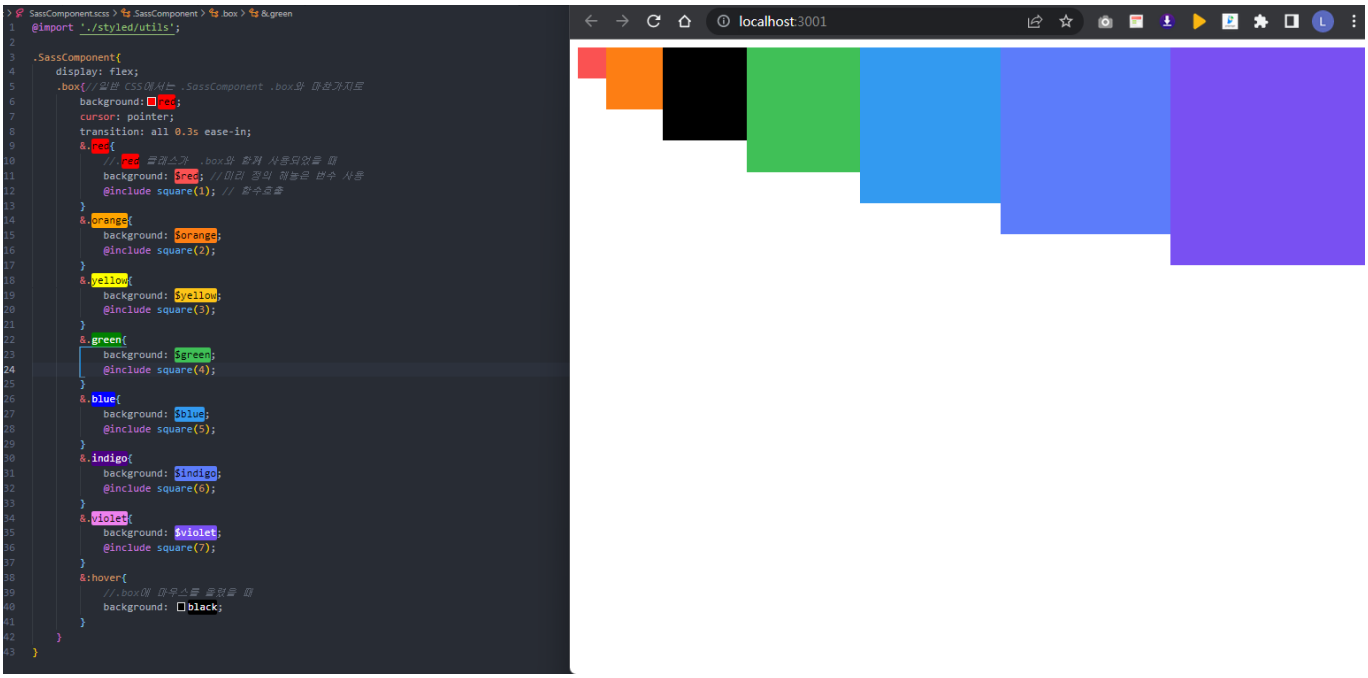
```
//변수 사용하기
$red: #fa5252;
$orange: #fd7e14;
$yellow: #fcc419;
$green: #40c057;
$blue: #339af0;
$indigo: #5c7cfa;
$violet: #7950f2;

//믹스인 말들기(재사용되는 스타일 블록을 함수처럼 사용할 수 있음)
@mixin square($size) {
    $calculated: 32px * $size;
    width: $calculated;
    height: $calculated;
}
```



SassComponent.scss

```
@import './styled/utils';

.SassComponent{
    display: flex;
    .box{//일반 CSS에서는 .SassComponent .box와 마찬가지로
        background:red;
        cursor: pointer;
```

```
        transition: all 0.3s ease-in;
        &.red{
            //.red 클래스가  .box와 함께 사용되었을 때
            background: $red; //미리 정의 해놓은 변수 사용
            @include square(1); // 함수호출
        }
        &.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{
            //.box에 마우스를 올렸을 때
            background: black;
        }
    }
}
```



9.3 CSS Module

css module은 css를 불러와서 사용할 때 클래스 이름을 고유한 값, 즉 [파일이름]*[클래스 이름]*[해시값]형태로 자동으로 만들어서 컴포넌트 스타일 클래스 이름이 중첩되는 현상을 방지해주는 기술이다. .module.css 확장자로 파일을 저장하기만 하면 CSS Module이 적용된다.

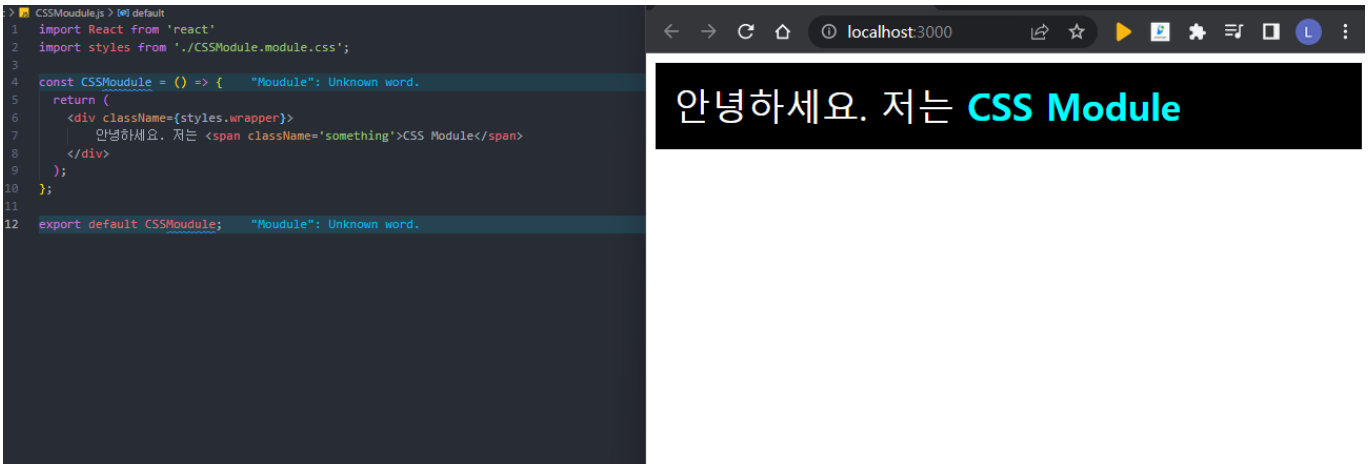
CSSModule.module.css

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

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

/* 글로벌 CSS를 작성하고 싶다면 */
:global .something{   /* :global을 사용하면 클래스 이름이 난독화 되지 않음 */
    font-weight: 800;
    color: aqua;
}
```

CSSModule.js

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

const CSSMoudule = () => {
  return (
    <div className={styles.wrapper}>
```

```
        안녕하세요. 저는 <span className='something'>CSS Module</span>
      </div>
    );
};

export default CSSMoudule;
```



App.js

```
import React from 'react';
import CSSMoudule from './CSSMoudule';

const App = () => {
  return (
    <div>
      <CSSMoudule/>
    </div>
  )
}

export default App
```

CSS Module을 사용한 클래스 이름 2개 이상 적용할 때

CSSModule.module.css

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

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

```
    background: white;
    border: 1px solid black;
}

/* 글로벌 CSS를 작성하고 싶다면 */
:global .something{   /* :global을 사용하면 클래스 이름이 난독화 되지 않음 */
    font-weight: 800;
    color: aqua;
}
```

CSSModule.js

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

const CSSMoudule = () => {
  return (
    <div className={`${styles.wrapper} ${styles.inverted}`}>
    {/*또는 className={[styles.wrapper, styled.inverted].join(' ')}도 사용가능하다
*/
        안녕하세요. 저는 <span className='something'>CSS Module</span>
    </div>
  );
};

export default CSSMoudule;
```

9.3.1 classnames

css 클래스를 조건부로 설정할 때 매우 유용한 라이브러리 이다.

설치 방법은 사용할 디렉토리 위치에서 터미널을 열어 아래 코드를 작성한다.

- $yarn add classnames

classnames 간략 사용법

```
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'; // 클래스 이름의 변수화
classNames('one', myClass, { myCondition: true} ); //= 'one hello myCondition'
```

예시코드

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

라이브러리 없이 같은 문제를 처리하기 위한 예시

예시코드

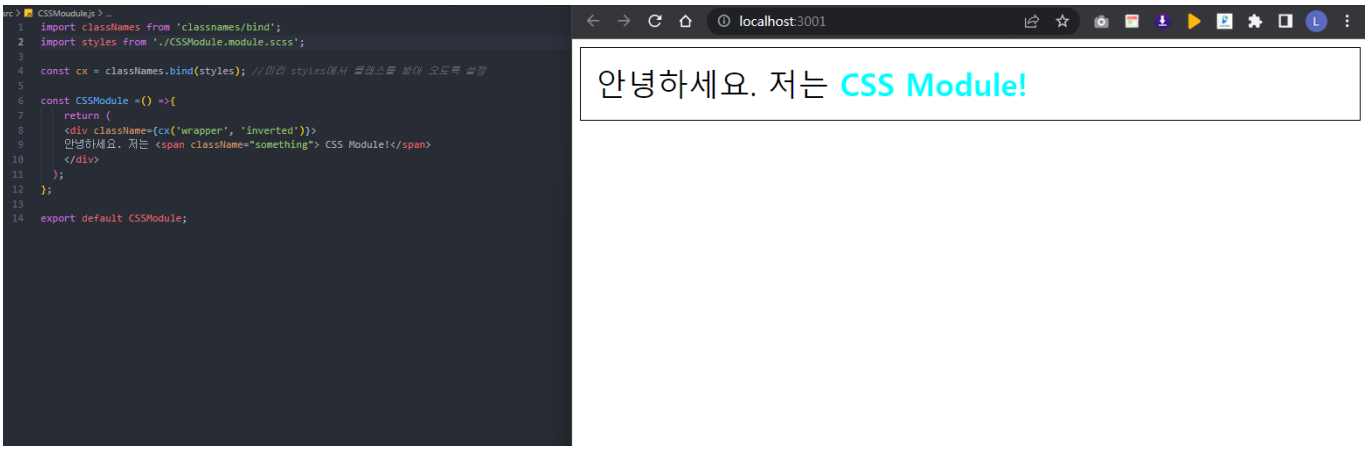
```
const MyComponent =({highlighted, theme})=>(
<div className={`MyComponent ${theme} ${highlighted? 'highlighted' : ''}`}>
</div>
);
```

classnames의 bind 함수 사용

classnames의 bind 함수 사용를 사용하면 클래스를 넣어 줄 때마다 styles.[클래스 이름] 형태를 사용할 필요가 없다. 사전에 미리 styles를 받아 온 후 사용할 수 있게 설정 후, cx('클래스이름', '클래스 이름2') 형태로 사용할 수 있다

예시코드

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

const cx = classNames.bind(styles); //미리 styles에서 클래스를 받아 오도록 설정

const CSSModule =() =>{
        return (
        <div className={cx('wrapper', 'inverted')}>
        안녕하세요. 저는 <span className="something"> CSS Module!</span>
        </div>
    );
};

export default CSSModule;
```

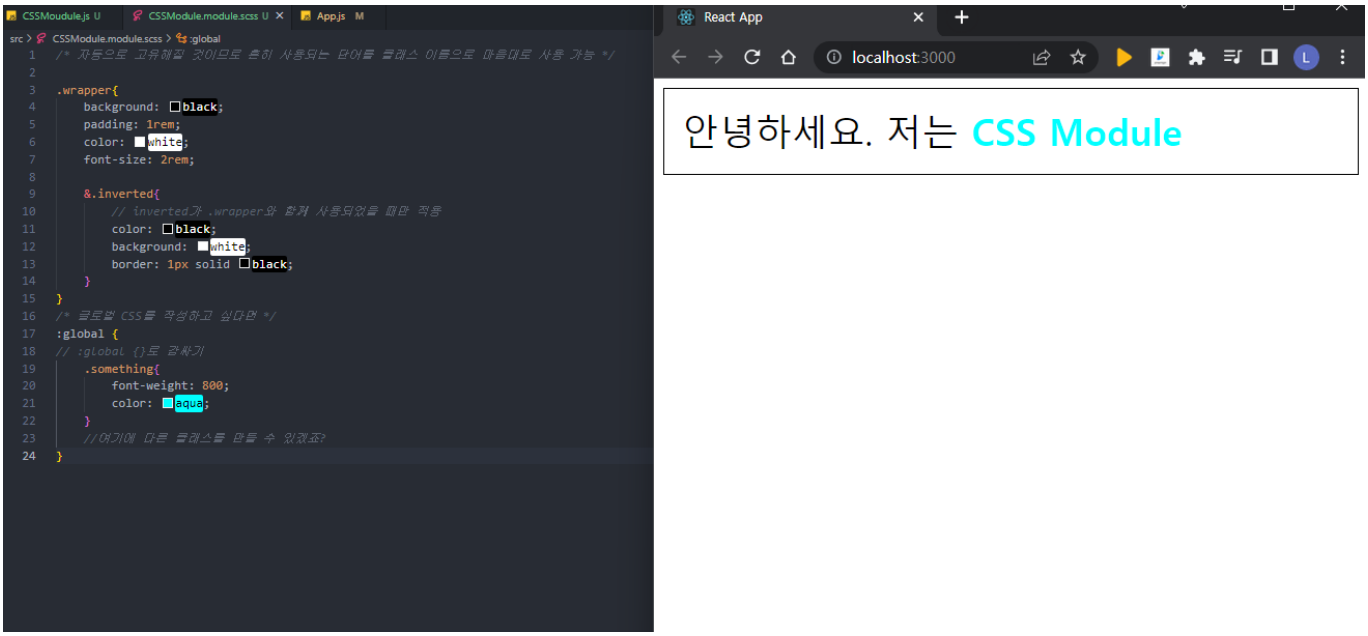


9.3.2 Sass와 함께 사용하기

Sass를 사용할 때도 파일 이름 뒤에 .module.scss확장자를 사용해주면 CSS Module로 사용할 수 있다.
CSSModule.module.css 파일이름을 CSSModule.scss로 변경할 수 있다.

CSCCModule.moudle.scss

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

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

    &.inverted{
        // inverted가 .wrapper와 함께 사용되었을 때만 적용
        color: black;
        background: white;
        border: 1px solid black;
    }
}
/* 글로벌 CSS를 작성하고 싶다면 */
:global {
// :global {}로 감싸기
    .something{
        font-weight: 800;
        color: aqua;
    }
    //여기에 다른 클래스를 만들 수 있겠죠?
}
```



9.4 styled-components

컴포넌트 스타일링은의 또 다른 패러다임은 자바스크립트 파일 안에 스타일을 선언 하는 방식이다. 이 방식을 'CSS-in-JS'라고 부른다.

설치 방법은 사용할 디렉토리 위치에서 터미널을 열어 아래 코드를 작성한다.

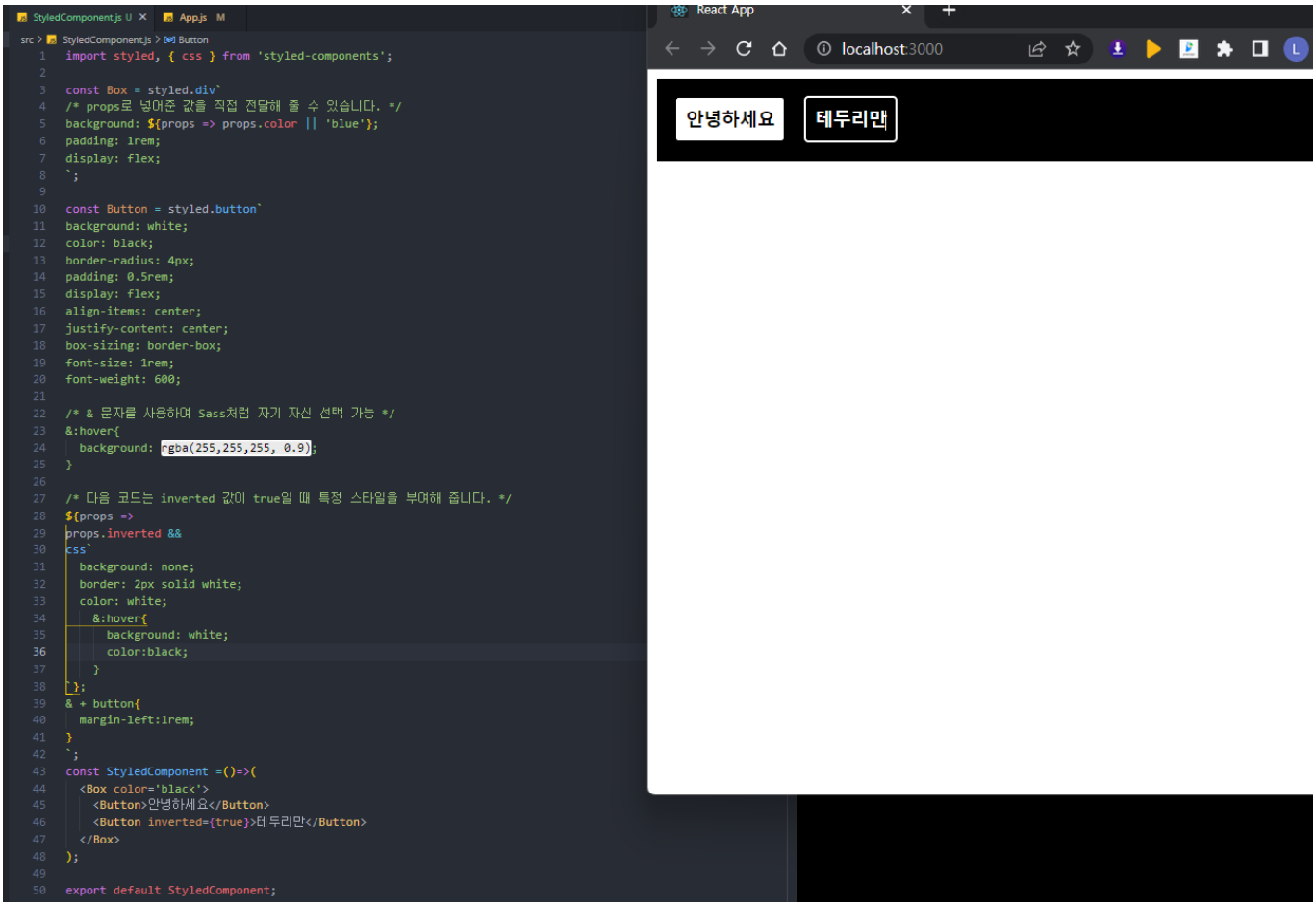
- $ yarn add styled-components

styled-components을 사용하면 자바스크립트 파일 하나에 스타일까지 같이 작성할 수 있기에 .css. 또는 scss 확장자를 파일을 따로 만들지 않아도 된다.

StyledComponent.js

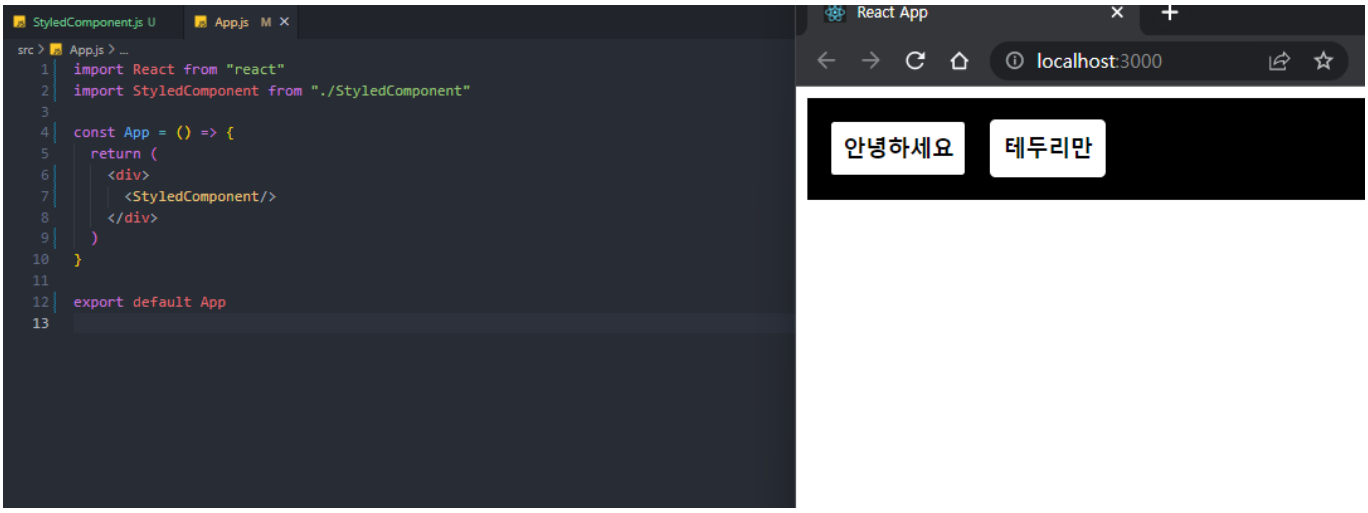
```
import styled, { css } from 'styled-components';

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

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 Sass처럼 자기 자신 선택 가능 */
&:hover{
  background: rgba(255,255,255, 0.9);
}

/* 다음 코드는 inverted 값이 true일 때 특정 스타일을 부여해 줍니다. */
${props =>
props.inverted && //true일 경우 css 부분이 실행됨
css`
  background: none;
  border: 2px solid white;
  color: white;
    &:hover{
      background: white;
      color:black;
    }
`};
& + button{
  margin-left:1rem;
}
`;
const StyledComponent =()=>(
  <Box color='black'>
    <Button>안녕하세요</Button>
    <Button inverted={true}>테두리만</Button>
  </Box>
);

export default StyledComponent;
```



App.js

```
import React from "react"
import StyledComponent from "./StyledComponent"

const App = () => {
  return (
    <div>
      <StyledComponent/>
    </div>
  )
}

export default App
```



9.4.1 Tagged 템플릿 리터럴

스타일을 작성할 때 `을 사용하여 만든 문자열에 스타일 정보를 넣어주는데 여기서 사용한 문법을 Tagged 템플릿 리터럴이라고 부른다. 일반 템플릿하고 다른 점은 템플릿 안에 자바스크립트 객체나 함수를 전달할 때 온전히 추출 할 수 있다.

9.4.2 스타일링 된 엘리먼트 만들기

styled-components를 사용하여 스타일링된 엘리먼트를 만들 때는 컴포넌트 파일의 상단에서 styeld를 불러오고, styled.태그명을 사용하여 구현합니다.

예시코드

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

const MyComponent = styled.div`
    font-size: 2rem;
`;
```

특정 컴포넌트 스타일링

예시코드

```
// 태그의 타입을 styled 함수의 인자로 전달
const MyInput = styled('input')`
    background: gray;
`
// 아예 컴포넌트 형식의 값을 넣어줌
const StyledLink = styled(Link)` //style을 함수처럼 적용
    color: blue;
`
```

9.4.3 스타일에서 props 조회하기

styled-components를 사용하면 스타일 쪽에서 컴포넌트에게 전달된 props 값을 참조할 수 있다.

StyledComponents.js-Box 컴포넌트

```
const Box = styled.div`
/* props로 넣어준 값을 직접 전달해 줄 수 있습니다. */
background: ${props => props.color || 'blue'}; //'blue'를 기본값으로 설정
padding: 1rem;
display: flex;
`;
```

9.4.4 props에 따른 조건부 스타일링

StyledComponent.js-Button

```
import styled, { css } from 'styled-components';
/*
단순 변수의 형태가 아니라 여러 줄의 스타일 구문을 조건부로 설정해야하는 경우에는 css를
불러와야 합니다.
*/

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 Sass처럼 자기 자신 선택 가능 */
&:hover{
  background: rgba(255,255,255, 0.9);
}

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

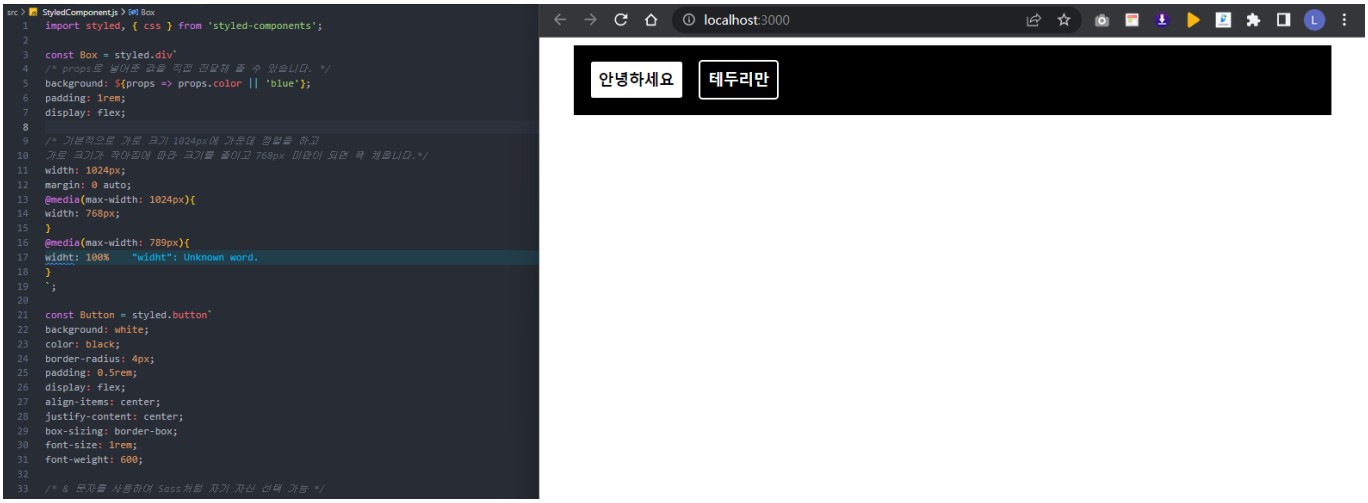
```
& + button{
  margin-left:1rem;
}
`;
// 이렇게 만든 컴포넌트는 props를 사용하여 서로 다른 스타일을 적용할 수 있다.
<Button>안녕하세요</Button>
<Button inverted={true}>테두리만</Button> //true일때 스타일이 적용됨.
```

9.4.5 반응형 디자인

styled-components를 사용할 때 반응형 디자인을 어떻게 사용하는지 알아보기. 일반 CSS 를 사용할 때와 똑같이 media 쿼리(query)를 사용하면 된다.

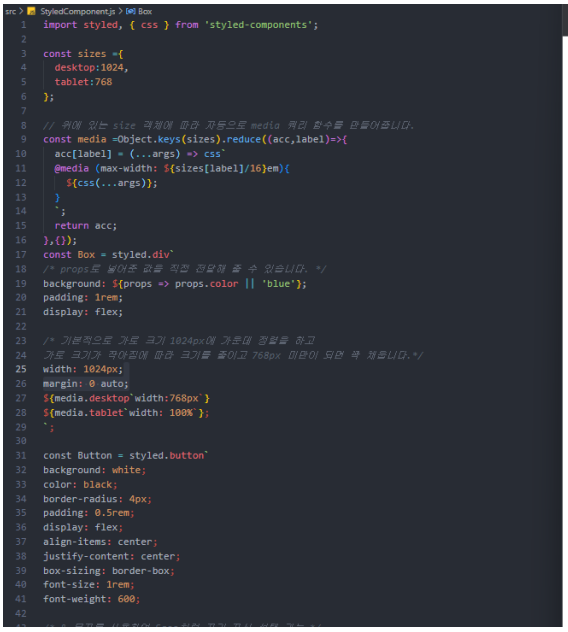
StyledComponent.js-Box

```
const Box = styled.div`
/* props로 넣어준 값을 직접 전달해 줄 수 있습니다. */
background: ${props => props.color || 'blue'};
padding: 1rem;
display: flex;
/* 기본적으로 가로 크기 1024px에 가운데 정렬을 하고
가로 크기가 작아짐에 따라 크기를 줄이고 768px 미만이 되면 꽉 채웁니다.*/
width: 1024px;
margin: 0 auto;
@media(max-width: 1024px){
width: 768px;
}
@media(max-width: 789px){
widht: 100%
}
`;
```



styled-components 메뉴얼에서 제동하는 유틸함수 적용

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

const sizes ={
  desktop:1024,
  tablet:768
};

// 위에 있는 size 객체에 따라 자동으로 media 쿼리 함수를 만들어줍니다.
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`
/* props로 넣어준 값을 직접 전달해 줄 수 있습니다. */
background: ${props => props.color || 'blue'};
padding: 1rem;
display: flex;

/* 기본적으로 가로 크기 1024px에 가운데 정렬을 하고
가로 크기가 작아짐에 따라 크기를 줄이고 768px 미만이 되면 꽉 채웁니다.*/
width: 1024px;
margin: 0 auto;
${media.desktop`width:768px`}
${media.tablet`width: 100%`};
`;
```



13. 리액트 라우터로 SPA 개발하기

작성일시: 2022년 5월 10일 오후 7:57 참고 도서: 리액트를 다루는 기술

13.1 라우팅이란?

웹 애플리케이션에서 라우팅은 사용자가 요청한 URL에 따라 알맞은 페이지를 보여주는 것을 의미함.

웹 애플리케이션을 만들 때 프로젝트를 하나의 페이지 혹은 여러 페이지로 구성할 수 있다. 이렇게 여러 개의 페이지로 구성된 웹 애플리케이션을 만들 때 페이지 별로 분리된 컴포넌트들을 하나의 프로젝트로 관리하기 위한 것이 라우팅 시스템이다.

리액트에서 라우트 시스템 구축을 위해 사용할 수 있는 것은 크게 두가지 이다.

- 리액트 라우터(Reaxt Router)

  리액트 관련 라이브러리 중 가장 오래됐고, 가장 많이 사용되고 있습니다. 컴포넌트 기반으로 라우팅 시스템을 설정 할 수 있습니다.

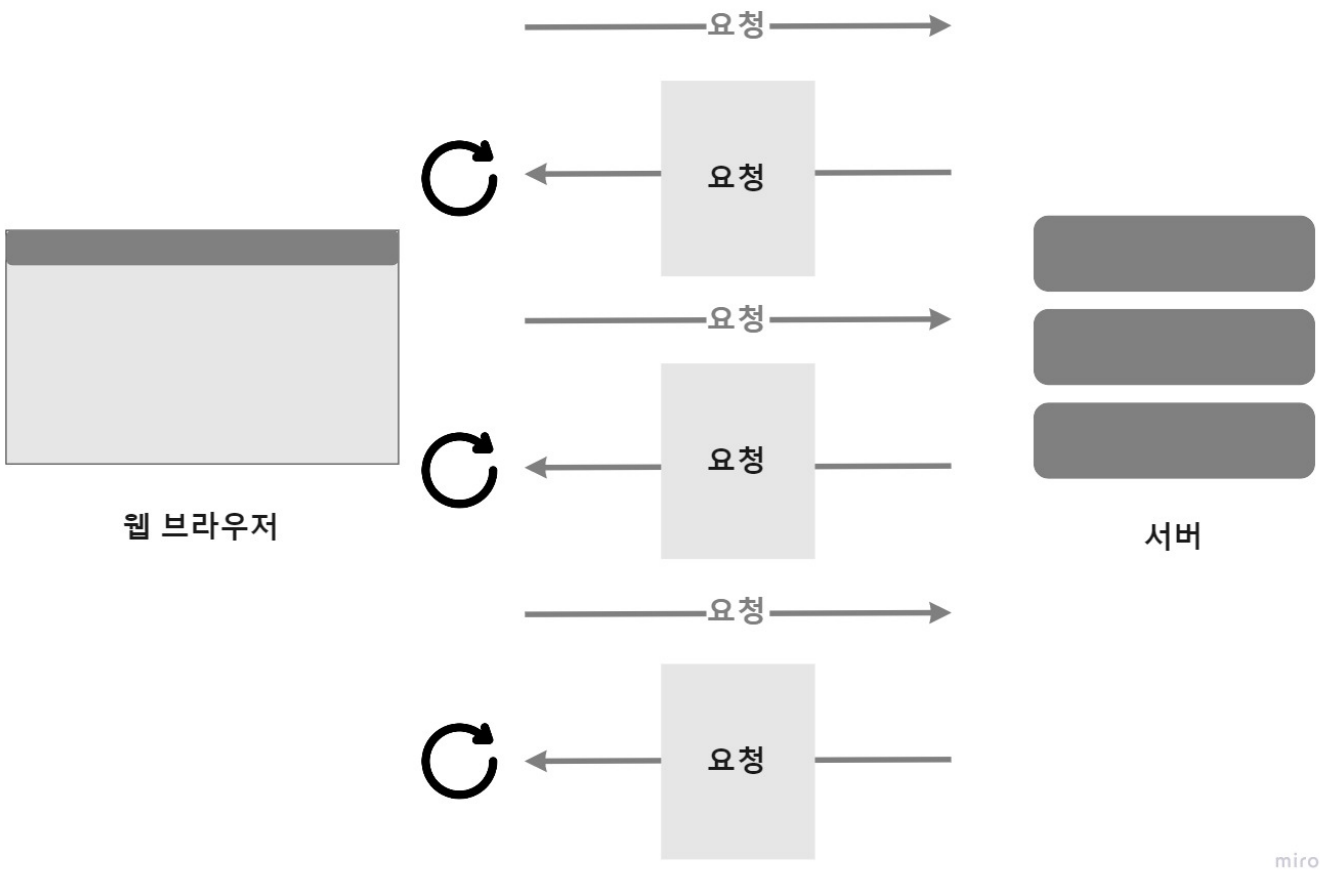
- Next.js

  리액트 프로젝트의 프레임워크이다. Create React App 처럼 리액트 프로젝트를 설정하는 기능, 라우팅 시스템, 최적화, 다국어 시스템 지원, 서버 사이트 렌더링 등 다양한 기능을 제공합니다. 이 프레임워크의 라우팅 시스템은 파일 경로 기반으로 작동해서 리액트 라우터의 대안으로 많이 사용되고 있습니다.

13.2 싱글 페이지(SAP) 애플리케이션이란?

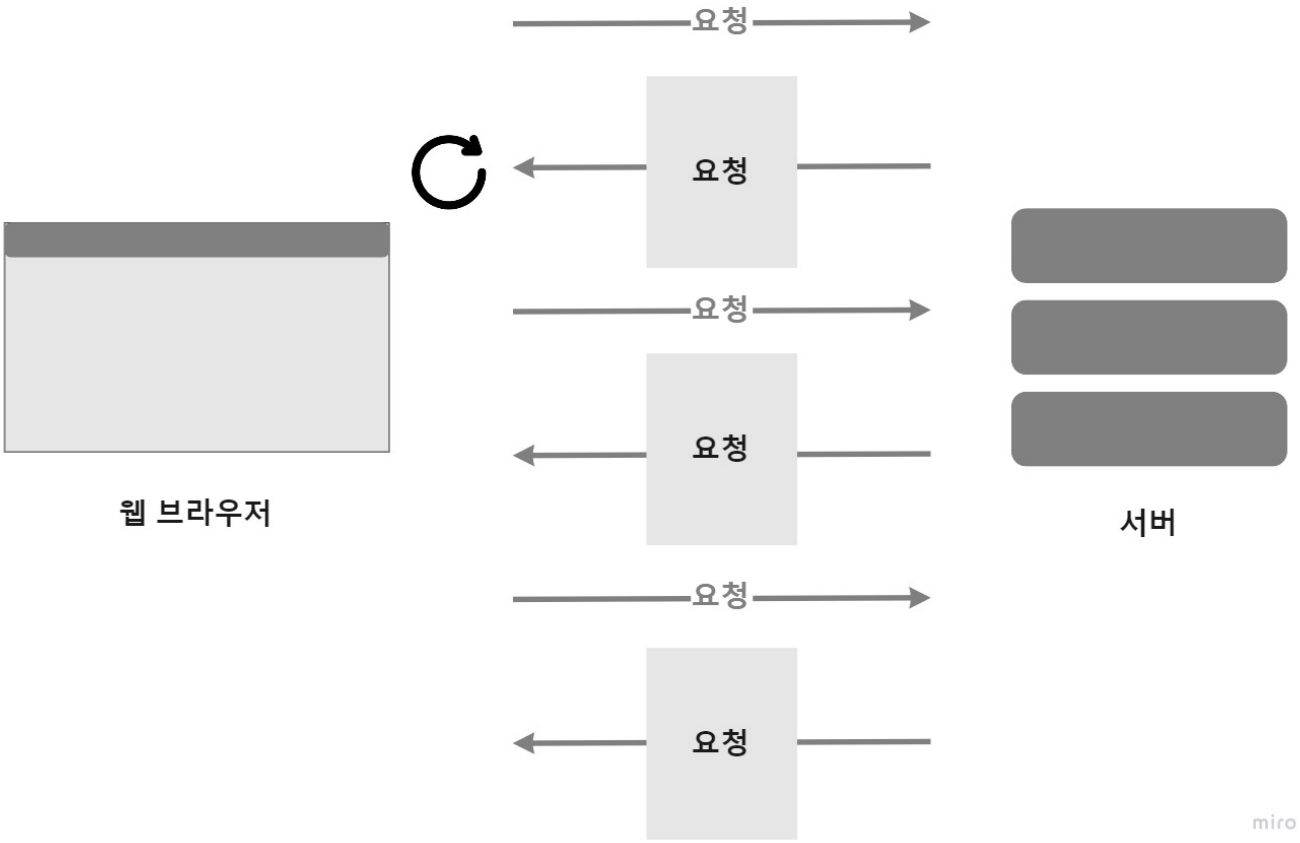
싱글 페이지 애플리케이션이란 하나의 페이지로 이루어진 애플리케이션이라는 의미이다.

다중페이지 원리 및 문제점



웹 서버라는 소프트웨어에 웹 브라우저가 접속한다. 이때, 전달되는 URL은 웹서버가 관리하는 폴더에 저장되어 있는 HTML파일 경로를 의미한다. 즉, 웹 브라우저가 웹 서버에 저장되어 있는 웹 페이지를 열람. 페이지 이동시마다 접속, 해제가 이뤄진다. 데이터에 따라 유동적인 html을 생성해주는 템플릿 엔진을 사용하기도 했다. ⇒ 하지만 그만큼 서버 자원을 사용하는 것고, 트래픽도 더 많이 나올 수 있다.

싱글 페이지 원리



뷰 렌더링(Ajax)을 사용자가 담당하게 하여 웹 애플리케이션을 브라우저에 불러와 사용자와의 인터렉션이 발생하면 필요한 부분만 자바스크립트를 사용하여 업데이트 하는 방식으로 사용하게 되었다.

싱글 페이지 애플리케이션은 한 페이지만 존재하지만, 사용자가 경험하기에는 여러 페이지가 존재하는 것처럼 느낄 수 있다. → 브라우저의 History API를 사용하여 브라우저의 주소창의 값만 변경하고 기존 페이지에 띄웠던 화면은 그대로 유지하면서 라우팅 설정에 따라 또 다른 페이지를 보여준다.

싱글 페이지(SPA)의 단점

앱의 규모가 커지면 최초 접속시 모든 화면을 구성하는 스크립트를 한번에 모두 로딩하기 때문에 자바스크립트 파일이 너무 커진다.

13.3 리액트 라우터 적용 및 기본 사용법

13.3.1 프로젝트 생성 및 라이브러리 설치

$yarn create react-app 폴더명(숫자 및 소문자만 가능)

디렉토리 위치로 이동

$cd 폴더명

리액트 라우터를 위한 라이브러리 설치

$ yarn add react-router-dom

13.3.2 프로젝트에 라우터 적용

index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { BrowserRouter } from 'react-router-dom';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
);
```

13.3.3 페이지 컴포넌트 만들기

Home.js

```
import React from 'react'

const Home = () => {
  return (
    <div>
        <h1>홈</h1>
        <p>가장 먼저 보여지는 페이지입니다.</p>
    </div>
  );
};

export default Home;
```

About.js

```
import React from 'react'

const About = () => {
  return (
    <div>
        <h1>소개</h1>
        <p>리액트 라우터를 사용해 보는 프로젝트입니다.</p>
```

```
      </div>
    );
};

export default About;
```

13.3.4 Route 컴포넌트로 특정 경로에 원하는 컴포넌트 보여주기

```
import React from "react";
import { Route,Routes } from "react-router-dom";
import About from "./pages/About";
import Home from "./pages/Home";

const App = () => {
  return (
    <div>
      <Routes>
        <Route path="/" exact={true} element={<Home/>}/>
        <Route path="/" element={<About/>}/>
      </Routes>
    </div>
  )
}

export default App
```

12.3.5 Link 컴포넌트를 사용하여 다른 페이지로 이동하는 링크 보여주기

리액트 라우터를 사용하는 프로젝트에서는 a 태그를 바로 사용하면 안된다. a태그는 클릭할때 브라우저에서 새로운 페이지를 불러오기 때문이다. Link 컴포넌트는 새로 불러오는 것을 막고 History API를 통해 브라우저 주소의 경로만 바꾸는 기능이 내장되어 있다.

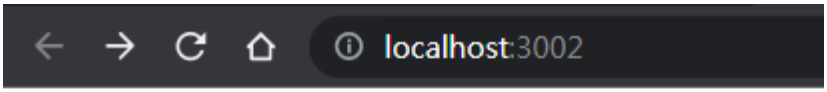
```
import React from 'react'
import { Link } from 'react-router-dom';
```

```
const Home = () => {
  return (
    <div>
        <h1>홈</h1>
        <p>가장 먼저 보여지는 페이지입니다.</p>
        <Link to="/about">소개</Link>
    </div>
  );
};

export default Home;
```

홈

가장 먼저 보여지는 페이지입니다.

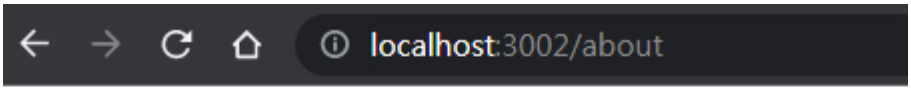
소개



홈

가장 먼저 보여지는 페이지입니다.

소개



소개

리액트 라우터를 사용해 보는 프로젝트입니다

13.4 URL 파라미터와 쿼리스트링

- URL 파라미터란

  주소 경로에 유동적인 값을 넣는 형태로, ID 또는 이름을 사용하여 특정 데이터를 조회할 때 사용된다. ex) /profile/velopart

- 쿼리스트링 파라미터란

  주소의 뒷부분에 ? 문자열 이후 key=value로 값을 정의하며 &로 구분하는 형태이다. 주로 키워드 검색, 페이지네이션, 정렬 방식 등 데이터 조회에 필요한 옵션을 전달할 때 사용된다. ex)/articles?page=1&keyword=react

13.4.1 URL 파라미터

Profile.js

```
import React from 'react'
import { useParams } from 'react-router-dom'

const data={
    velopert:{
        name:'김민준',
        description: '리액트를 좋아하는 개발자'
    },
    gildong: {
        name:'홍길동',
        description: '고전 소설 홍길동전의 주인공',
    },
};
const Profile = () => {
  const params = useParams();
  const profile =data[params.username];
    return (
    <div>
        <h1>사용자 프로필</h1>
        {profile ?(
            <div>
                <h2>{profile.name}</h2>
                <p>{profile.description}</p>
            </div>
        ) : (
            <p>존재하지 않는 프로필입니다. </p>
        )}
    </div>
  );
};

export default Profile;
```

App.js

```
import React from "react";
import { Route,Routes } from "react-router-dom";
import About from "./pages/About";
import Home from "./pages/Home";
import Profile from "./pages/Profile";

const App = () => {
  return (
    <div>
      <Routes>
        <Route path="/" exact={true} element={<Home/>}/>
        <Route path="/about" element={<About/>}/>
        <Route path="/profiles/:username" element={<Profile/>}/>
      </Routes>
    </div>
  );
};

export default App;
```

Home.js

```
import React from 'react'
import { Link } from 'react-router-dom';

const Home = () => {
  return (
    <div>
        <h1>홈</h1>
        <p>가장 먼저 보여지는 페이지입니다.</p>

        <ul>
            <li>
            <Link to="/about">소개</Link>
            </li>
            <li>
            <Link to="/profiles/velopert">velopert의 프로필입니다.</Link>
            </li>
            <li>
            <Link to="/profiles/gildong">gildong의 프로필입니다.</Link>
            </li>
            <li>
            <Link to="/profiles/void">존재하지 않는 프로필</Link>
            </li>
        </ul>
    </div>
  );
};

export default Home;
```

```jsx
import React from 'react'
import { Link } from 'react-router-dom';

const Home = () => {
  return (
    <div>
        <h1>홈</h1>
        <p>가장 먼저 보여지는 페이지입니다.</p>

        <ul>
            <li>
            <Link to="/about">소개</Link>
            </li>
            <li>
            <Link to="/profiles/velopert">velopert의 프로필입니다.</Link>    "velopert"
            </li>
            <li>
            <Link to="/profiles/gildong">gildong의 프로필입니다.</Link>    "gildong": U
            </li>
            <li>
            <Link to="/profiles/void">존재하지 않는 프로필</Link>
            </li>
        </ul>
    </div>
  );
};

export default Home;
```

홈

가장 먼저 보여지는 페이지입니다.

- 소개
- velopert의 프로필입니다.
- gildong의 프로필입니다.
- 존재하지 않는 프로필

localhost:3002/profiles/gildong

사용자 프로필

홍길동

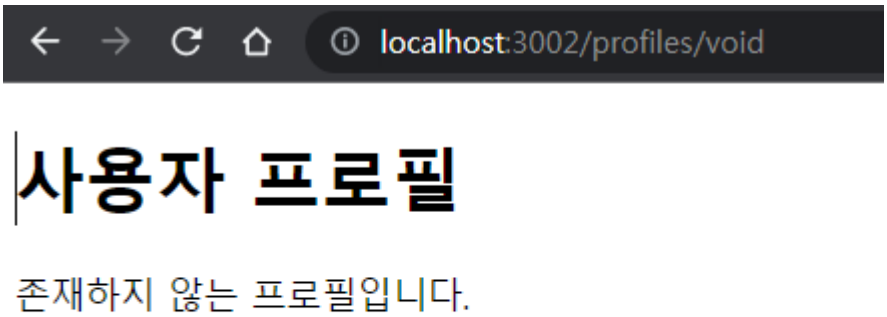
고전 소설 홍길동전의 주인공

localhost:3002/profiles/velopert

사용자 프로필

김민준

리액트를 좋아하는 개발자



13.4.2 쿼리스트링

About.js

```
import { useLocation } from "react-router-dom";

const About = () => {
    const location = useLocation();
  return (
    <div>
        <h1>소개</h1>
        <p>리액트 라우터를 사용해 보는 프로젝트입니다.</p>
        <p>쿼리스트링:{location.search}</p>
    </div>
  );
};

export default About;
```

쿼리스트링 파싱하는 코드

About.js

```
import {  useSearchParams } from "react-router-dom";

const About = () => {
    const [searchParams, setSearchParams] = useSearchParams();
    const detail = searchParams.get('detail');
    const mode = searchParams.get('mode');
    const onToggleDetail=()=>{
        setSearchParams({mode, detail:detail ==='true'? false: true});
    };
    const onIncreaseMode=()=>{
        const nextMode = mode === null ? 1 : parseInt(mode)+1;
        setSearchParams({mode:nextMode, detail});
    }
  return (
    <div>
        <h1>소개</h1>
        <p>리액트 라우터를 사용해 보는 프로젝트입니다.</p>
        <p>detail: {detail}</p>
        <p>mode: {mode}</p>
        <button onClick={onToggleDetail}>Toggle detail</button>
        <button onClick={onIncreaseMode}>mode+1</button>
    </div>
  );
};

export default About;
```

13.5 중첩된 라우트

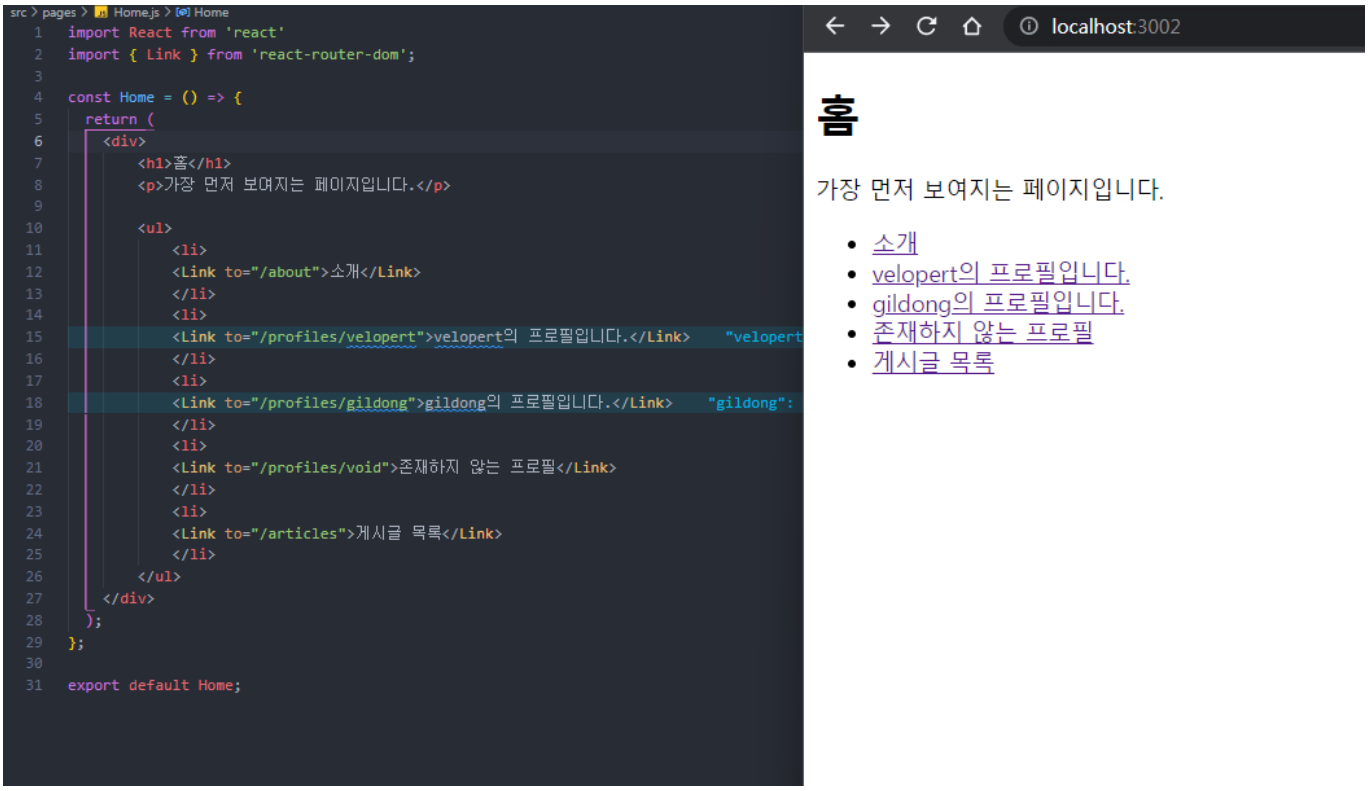
Articles.js

```
import React from 'react'
import { Link } from 'react-router-dom'
```

```
const Articles = () => {
  return (
    <ul>
        <li>
            <Link to="/articles/1">게시글 1</Link>
        </li>
        <li>
            <Link to="/articles/2">게시글 2</Link>
        </li>
        <li>
            <Link to="/articles/3">게시글 3</Link>
        </li>
    </ul>
  );
};

export default Articles;
```

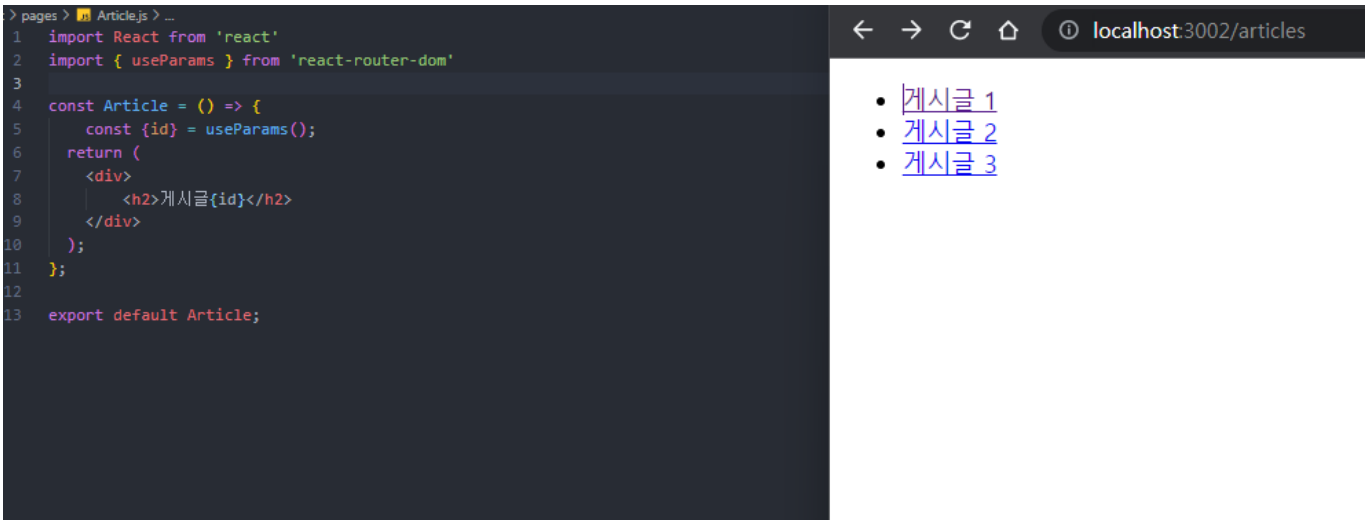
Article.js

```
import React from 'react'
import { useParams } from 'react-router-dom'

const Article = () => {
    const {id} = useParams();
  return (
    <div>
        <h2>게시글{id}</h2>
    </div>
  );
};

export default Article;
```



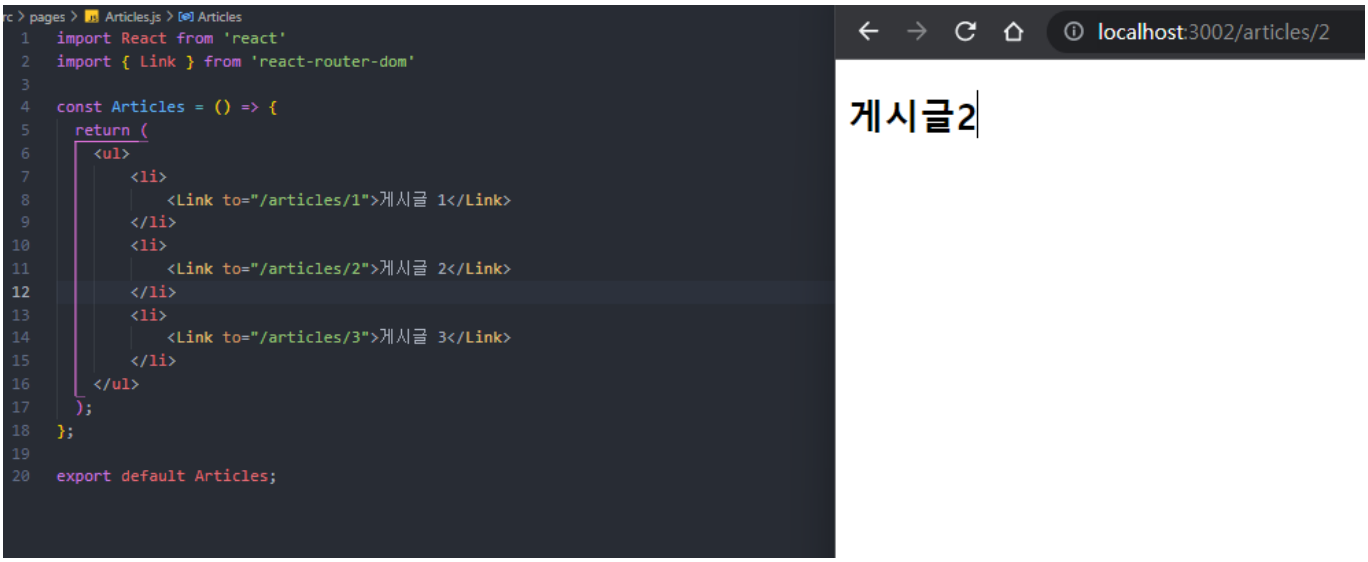
App.js

```
import React from "react";
import { Route,Routes } from "react-router-dom";
import About from "./pages/About";
import Article from "./pages/Article";
import Articles from "./pages/Articles";
import Home from "./pages/Home";
import Profile from "./pages/Profile";

const App = () => {
  return (
    <div>
      <Routes>
        <Route path="/" exact={true} element={<Home/>}/>
        <Route path="/about" element={<About/>}/>
        <Route path="/profiles/:username" element={<Profile/>}/>
        <Route path="/articles" element={<Articles/>}/>
        <Route path="/articles/:id" element={<Article/>}/>
      </Routes>
    </div>
  );
};

export default App;
```



Home.js

```
import React from 'react'
import { Link } from 'react-router-dom';

const Home = () => {
  return (
    <div>
        <h1>홈</h1>
        <p>가장 먼저 보여지는 페이지입니다.</p>

        <ul>
            <li>
            <Link to="/about">소개</Link>
            </li>
            <li>
            <Link to="/profiles/velopert">velopert의 프로필입니다.</Link>
            </li>
            <li>
            <Link to="/profiles/gildong">gildong의 프로필입니다.</Link>
            </li>
            <li>
            <Link to="/profiles/void">존재하지 않는 프로필</Link>
            </li>
            <li>
            <Link to="/articles">게시글 목록</Link>
            </li>
        </ul>
    </div>
  );
};

export default Home;
```



중첩된 라우트 형태로 설정

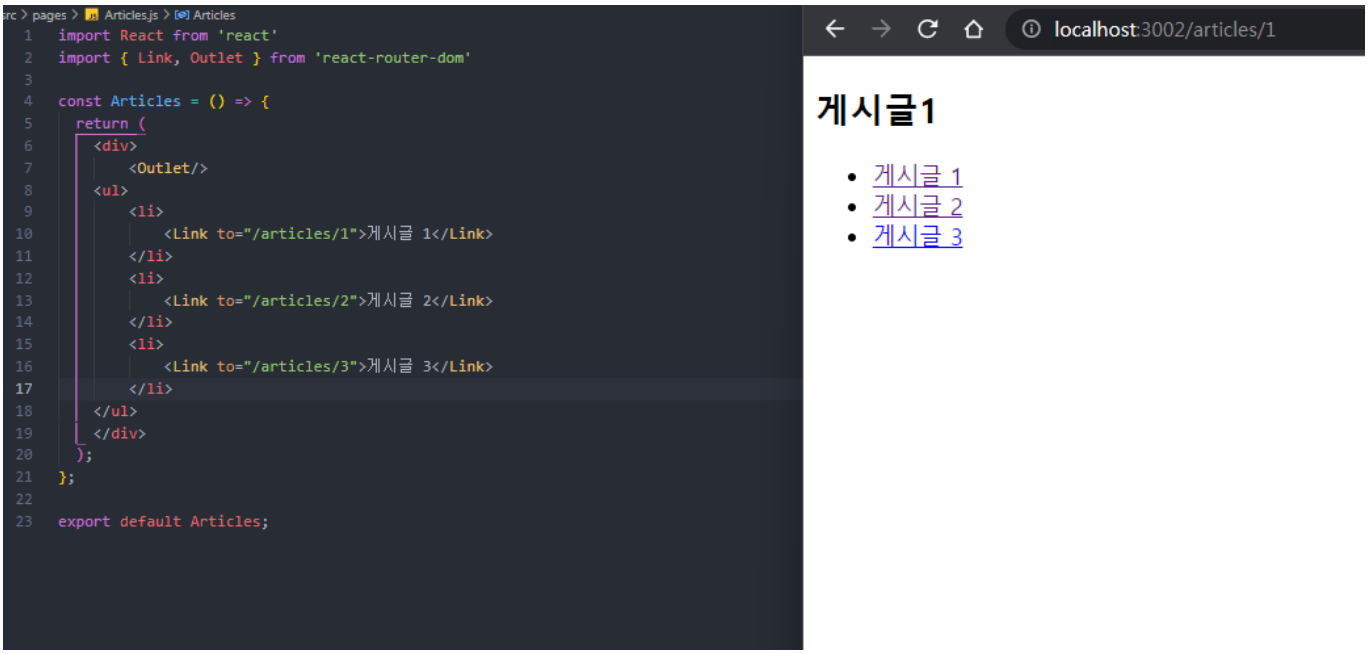
App.js

```
import React from "react";
import { Route,Routes } from "react-router-dom";
import About from "./pages/About";
import Article from "./pages/Article";
import Articles from "./pages/Articles";
import Home from "./pages/Home";
import Profile from "./pages/Profile";

const App = () => {
  return (
    <div>
      <Routes>
        <Route path="/" exact={true} element={<Home/>}/>
        <Route path="/about" element={<About/>}/>
        <Route path="/profiles/:username" element={<Profile/>}/>
        <Route path="/articles" element={<Articles/>}>
          <Route path=":id" element={<Article/>}/>
        </Route>
      </Routes>
    </div>
  );
};

export default App;
```

Articles.js

```
import React from 'react'
import { Link, Outlet } from 'react-router-dom'
```

```
const Articles = () => {
  return (
    <div>
        <Outlet/>
    <ul>
        <li>
            <Link to="/articles/1">게시글 1</Link>
        </li>
        <li>
            <Link to="/articles/2">게시글 2</Link>
        </li>
        <li>
            <Link to="/articles/3">게시글 3</Link>
        </li>
    </ul>
    </div>
  );
};

export default Articles;
```



13.5.1 공통 레이아웃 컴포넌트

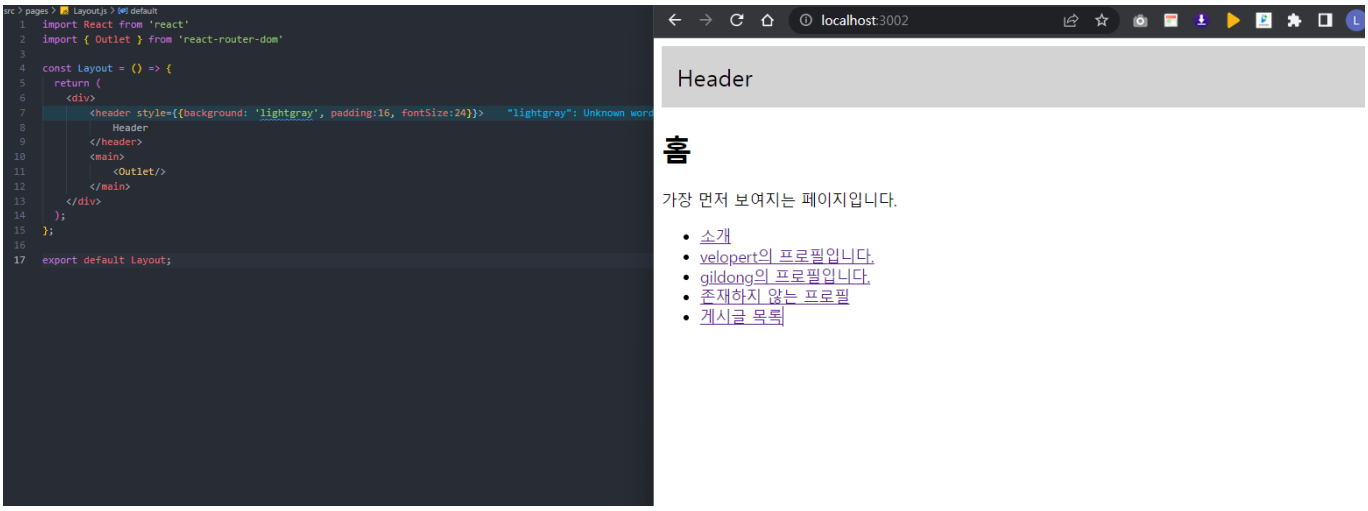
Layout.js

```
import React from 'react'
import { Outlet } from 'react-router-dom'

const Layout = () => {
  return (
    <div>
        <header style={{background: 'lightgray', padding:16, fontSize:24}}>
            Header
```

```
        </header>
        <main>
            <Outlet/>
        </main>
    </div>
  );
};

export default Layout;
```

App.js

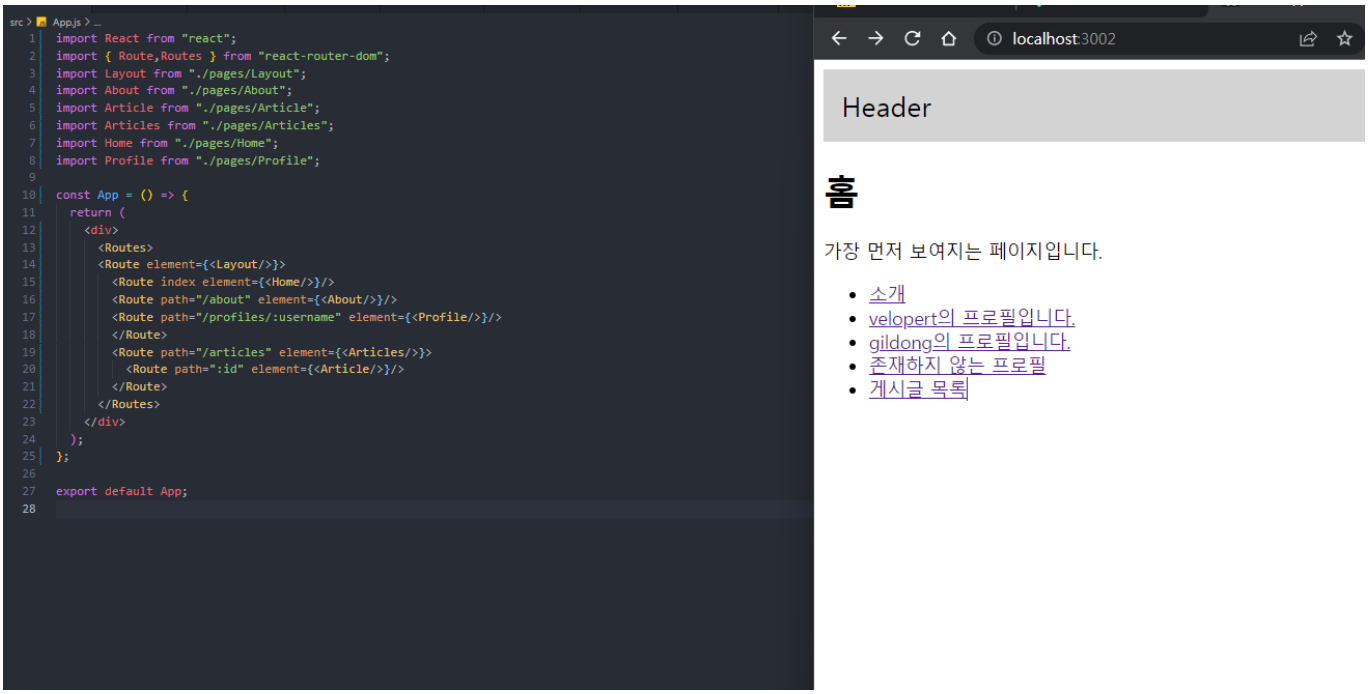
```
import React from "react";
import { Route,Routes } from "react-router-dom";
import Layout from "./pages/Layout";
import About from "./pages/About";
import Article from "./pages/Article";
import Articles from "./pages/Articles";
import Home from "./pages/Home";
import Profile from "./pages/Profile";

const App = () => {
  return (
    <div>
      <Routes>
      <Route element={<Layout/>}>
        <Route path="/" exact={true} element={<Home/>}/>
        <Route path="/about" element={<About/>}/>
        <Route path="/profiles/:username" element={<Profile/>}/>
        </Route>
        <Route path="/articles" element={<Articles/>}>
          <Route path=":id" element={<Article/>}/>
        </Route>
      </Routes>
    </div>
  );
};

export default App;
```

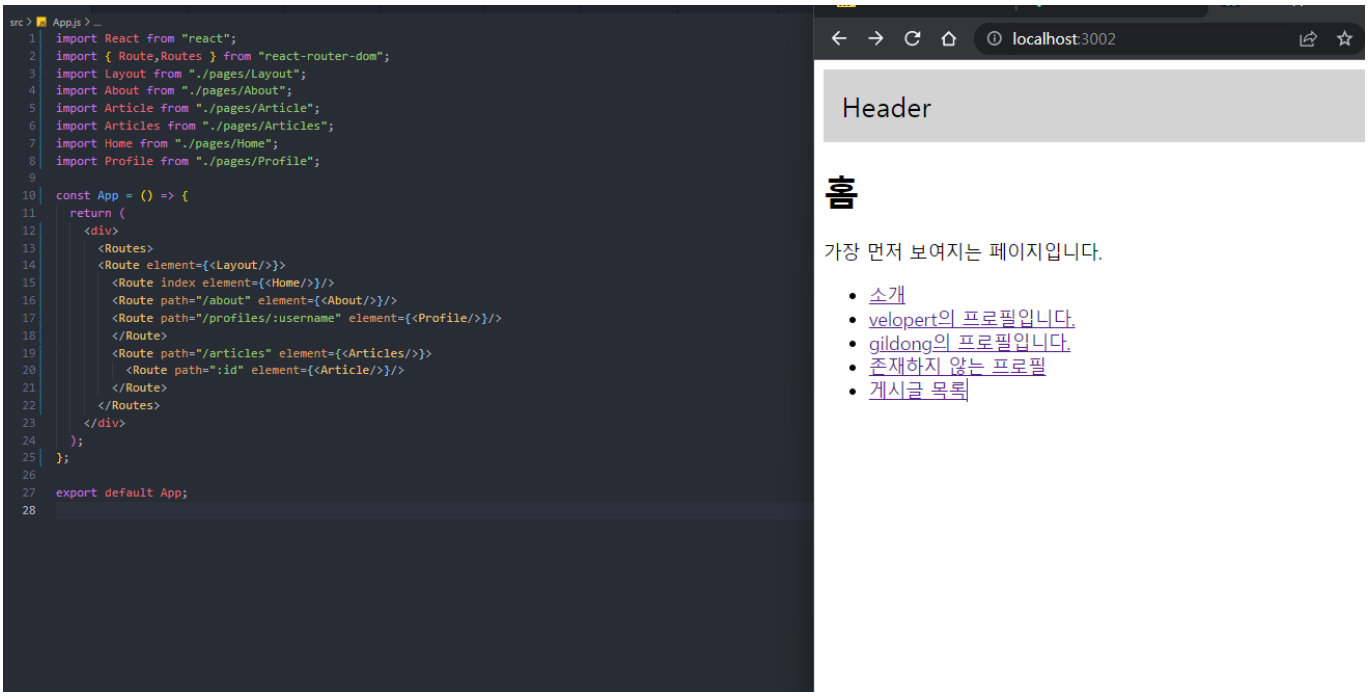


13.5.2 index props

App.js

```
import React from "react";
import { Route,Routes } from "react-router-dom";
import Layout from "./pages/Layout";
import About from "./pages/About";
import Article from "./pages/Article";
import Articles from "./pages/Articles";
import Home from "./pages/Home";
import Profile from "./pages/Profile";

const App = () => {
  return (
    <div>
      <Routes>
      <Route element={<Layout/>}>
        <Route index element={<Home/>}/>
        <Route path="/about" element={<About/>}/>
        <Route path="/profiles/:username" element={<Profile/>}/>
        </Route>
        <Route path="/articles" element={<Articles/>}>
          <Route path=":id" element={<Article/>}/>
        </Route>
      </Routes>
    </div>
  );
};

export default App;
```





13.6 리액트 라우터 부가 기능

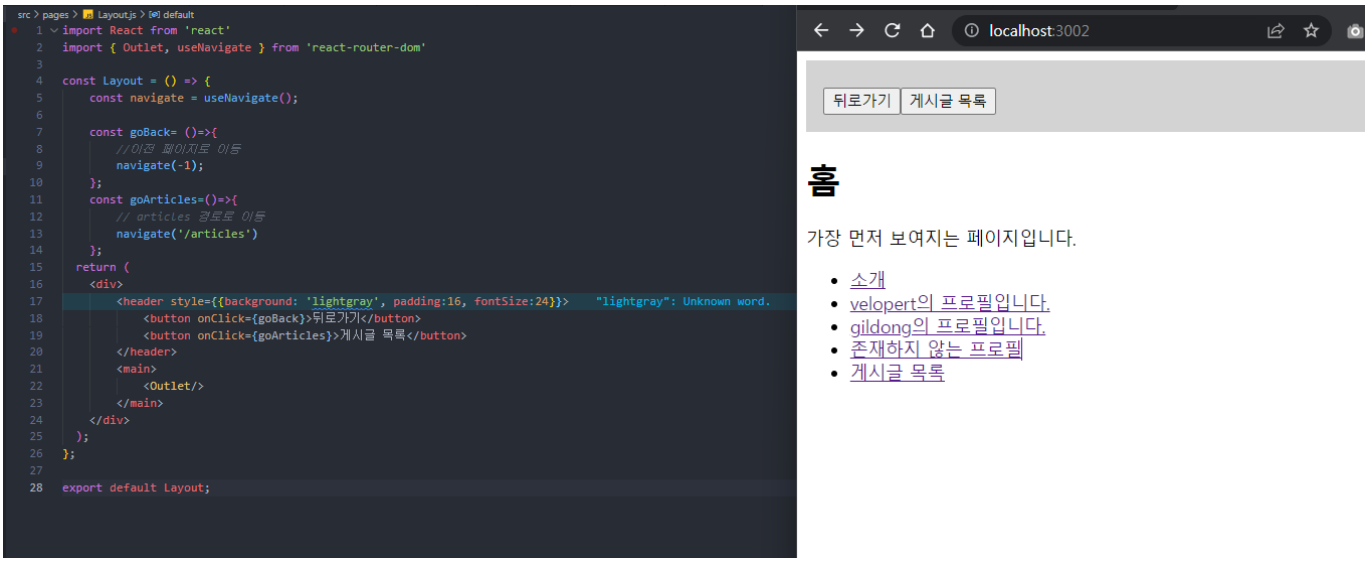
13.6.1 useNavigate

useNavigate는 Link 컴포넌트를 사용하지 않고 다른 페이지로 이동해야 하는 상황에 사용하는 Hook입니다.

Layout.js

```
import React from 'react'
import { Outlet, useNavigate } from 'react-router-dom'

const Layout = () => {
    const navigate = useNavigate();
```

```
    const goBack= ()=>{
        //이전 페이지로 이동
        navigate(-1);
    };
    const goArticles=()=>{
        // articles 경로로 이동
        navigate('/articles')
    };
  return (
    <div>
        <header style={{background: 'lightgray', padding:16, fontSize:24}}>
            <button onClick={goBack}>뒤로가기</button>
            <button onClick={goArticles}>게시글 목록</button>
        </header>
        <main>
            <Outlet/>
        </main>
    </div>
  );
};

export default Layout;
```



replace 옵션 사용

Layout.js-goArticles

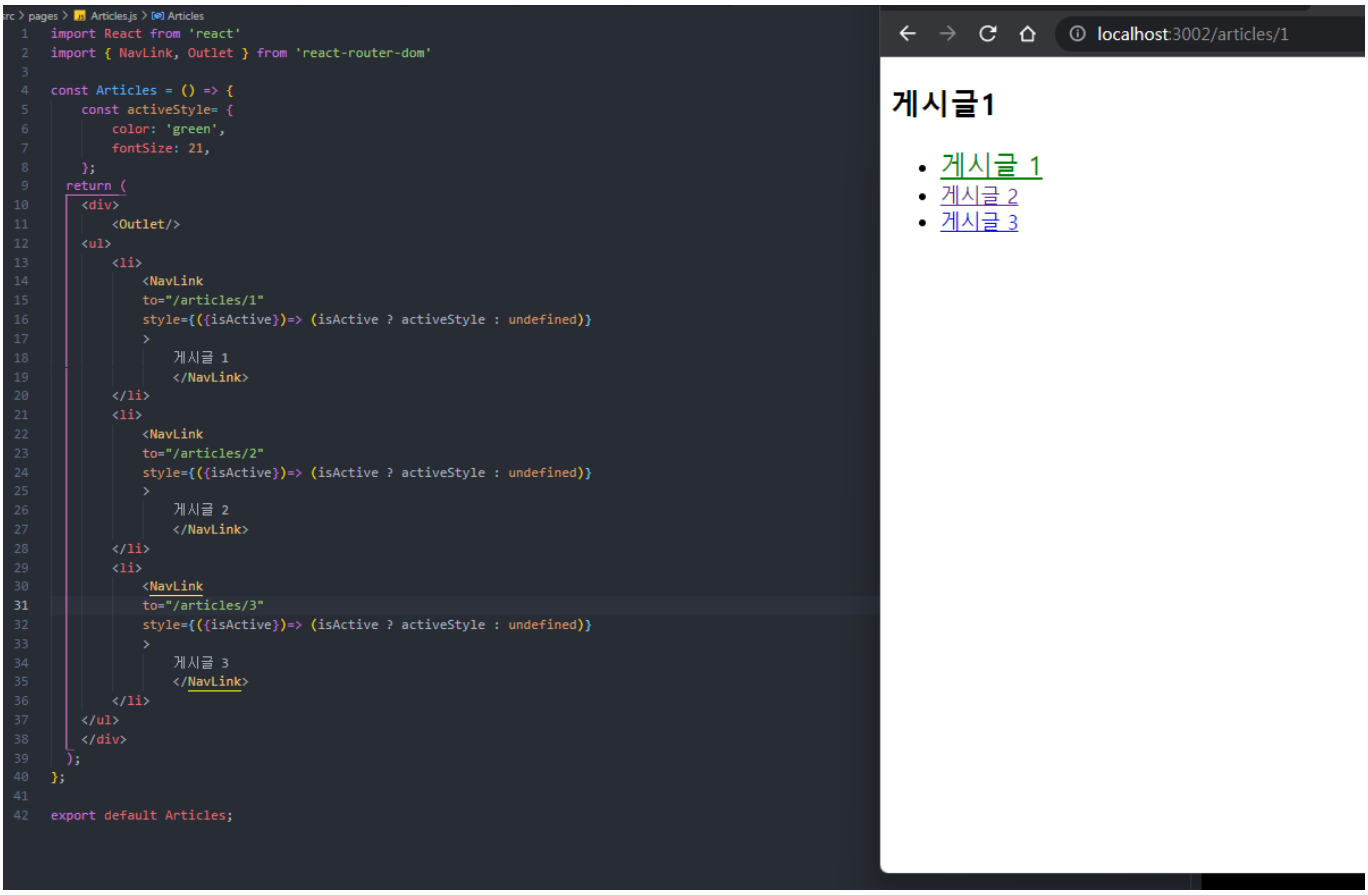
```
const goArticles=()=>{
        // articles 경로로 이동
        navigate('/articles', {replace: true});
    // 홈에서 다른 페이지 이동 후, 뒤로가기를 눌렀을 때 전 페이지가 아닌 홈으로 돌아간다.
 };
```

13.6.2 NavLink

NavLink 컴포넌트는 링크에서 사용하는 경로가 현재 라우트 경로와 일치하는 경우 특정 스타일 또는 CSS 클래스를 적용하는 컴포넌트 입니다.

Articles.js

```
import React from 'react'
import { NavLink, Outlet } from 'react-router-dom'

const Articles = () => {
    const activeStyle= {
        color: 'green',
        fontSize: 21,
    };
  return (
    <div>
        <Outlet/>
    <ul>
        <li>
            <NavLink
            to="/articles/1"
            style={({isActive})=> (isActive ? activeStyle : undefined)}
            >
                게시글 1
                </NavLink>
        </li>
        <li>
            <NavLink
            to="/articles/2"
            style={({isActive})=> (isActive ? activeStyle : undefined)}
            >
                게시글 2
                </NavLink>
        </li>
        <li>
            <NavLink
            to="/articles/3"
            style={({isActive})=> (isActive ? activeStyle : undefined)}
            >
                게시글 3
                </NavLink>
        </li>
    </ul>
    </div>
  );
};

export default Articles;
```

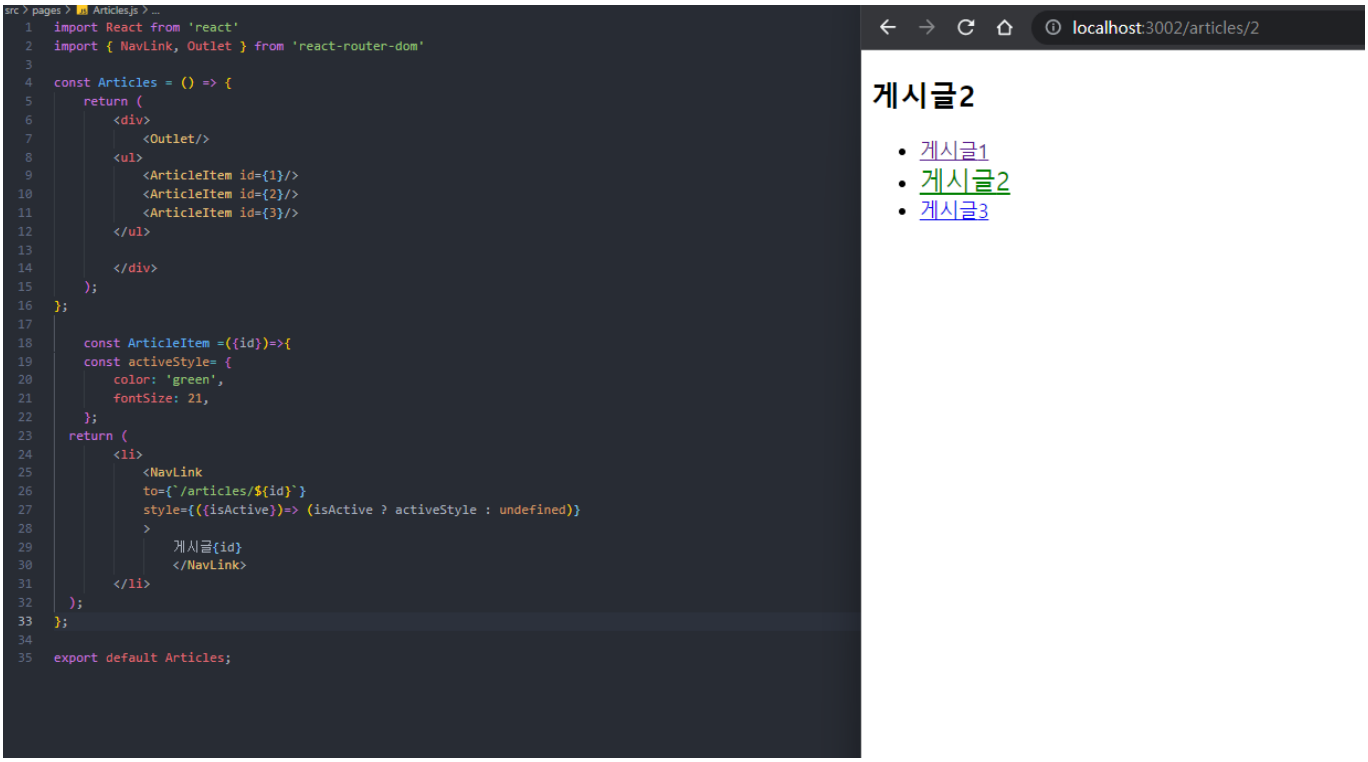


NavLink로 리팩터링 하기

Articles.js

```
import React from 'react'
import { NavLink, Outlet } from 'react-router-dom'

const Articles = () => {
    return (
        <div>
            <Outlet/>
        <ul>
            <ArticleItem id={1}/>
            <ArticleItem id={2}/>
            <ArticleItem id={3}/>
        </ul>

        </div>
    );
};

    const ArticleItem =({id})=>{
    const activeStyle= {
        color: 'green',
        fontSize: 21,
    };
  return (
        <li>
```

```
            <NavLink
            to={`/articles/${id}`}
            style={({isActive})=> (isActive ? activeStyle : undefined)}
            >
                게시글{id}
                </NavLink>
        </li>
  );
};

export default Articles;
```



13.6.3 NotFond 페이지 만들기

이 페이지는 사정에 정의되지 않은 경로에 사용자가 진입했을 때 보여는 페이지입니다. 즉 페이지를 찾을 수 없을 때 나타나는 페이지 입니다.

NotFond.js

```
import React from 'react'

const NotFound = () => {
  return (
    <div
    style={{
        display:'flex',
        alignItems:'center',
        fonstSize: 64,
        position: 'absolute',
        width: '100%',
```

```
            height: '100%',
        }}
        >
            404
            </div>
    );
};

export default NotFound;
```

App.js

```
import React from "react";
import { Route,Routes } from "react-router-dom";
import Layout from "./pages/Layout";
import About from "./pages/About";
import Article from "./pages/Article";
import Articles from "./pages/Articles";
import Home from "./pages/Home";
import Profile from "./pages/Profile";
import NotFound from "./pages/NotFound";

const App = () => {
  return (
    <div>
      <Routes>
      <Route element={<Layout/>}>
        <Route index element={<Home/>}/>
        <Route path="/about" element={<About/>}/>
        <Route path="/profiles/:username" element={<Profile/>}/>
        </Route>
        <Route path="/articles" element={<Articles/>}>
          <Route path=":id" element={<Article/>}/>
        </Route>
        <Route path="*" element={<NotFound/>}/>
      </Routes>
    </div>
  );
};

export default App;
```

13.6.4 Navigate 컴포넌트

Navigate 컴포넌트는 화면에 보여주는 순간 다른 페이지로 이동을 하고 싶을 때 사용하는 컴포넌트 입니다. 즉, 페이지를 리다이렉트 하고 싶을 때 사용합니다.

Login.js

```
import React from 'react'

const Login = () => {
  return (
    <div>로그인 페이지</div>
  );
};

export default Login;
```

MyPage.js

```
import React from 'react'
import { Navigate } from 'react-router-dom'

const MyPage = () => {
    const isLoggedIn = false;

    if(!isLoggedIn){
        return<Navigate to="/login" replace={true}/>
    }
  return <div>마이 페이지</div>
};

export default MyPage;
```

App.js

```
import React from "react";
import { Route,Routes } from "react-router-dom";
import Layout from "./pages/Layout";
import About from "./pages/About";
import Article from "./pages/Article";
import Articles from "./pages/Articles";
import Home from "./pages/Home";
import Profile from "./pages/Profile";
import NotFound from "./pages/NotFound";
import MyPage from "./pages/MyPage";
import Login from "./pages/Login";

const App = () => {
  return (
    <div>
      <Routes>
      <Route element={<Layout/>}>
        <Route index element={<Home/>}/>
        <Route path="/about" element={<About/>}/>
        <Route path="/profiles/:username" element={<Profile/>}/>
        </Route>
```

```
          <Route path="/articles" element={<Articles/>}>
            <Route path=":id" element={<Article/>}/>
          </Route>
          <Route path="/login" element={<Login/>}/>
          <Route path="/mypage" element={<MyPage/>}/>
          <Route path="*" element={<NotFound/>}/>
        </Routes>
      </div>
    );
  };

  export default App;
```