컴포넌트 스타일링

- 일반 css 컴포넌트를 스타일링하는 가장 기본적인 방식
- sass 자주 사용되는 css전처리기중 하나로 css를 문법을 사용하여 css코드를 더욱 쉽게 작성
- CSS Module 스타일을 작성할때 css클래스가 다른 css클래스의 이쁨과 절대 충돌하지 않도록 파일마다 고유한 이름을 자동으로 생성해주는 옵션
- styled-components 스타일을 자바스크립트 파일에 내장시키는 방식으로 스타일을 작성함과 동시
    - => 해당 스타일이 적동된 컴포넌트를 만드수 있음

9.1 가장 흔한 방식 일반 css

9.1.1 이름짓는 규칙

> 자신만의 네이밍규칙을 만들거나 메이저 업테의 코딩규칙 문서를 참고 ex)네이버 코딩 컨번션등

9.2 Sass 사용하기

스타일 코드의 재활용성을 높여

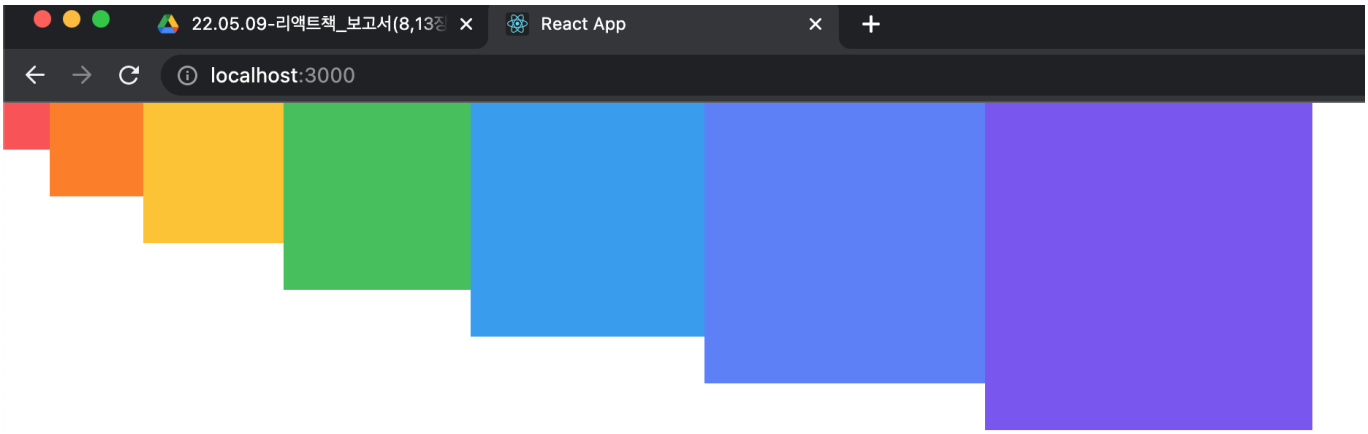
> $ yarn add node-sass

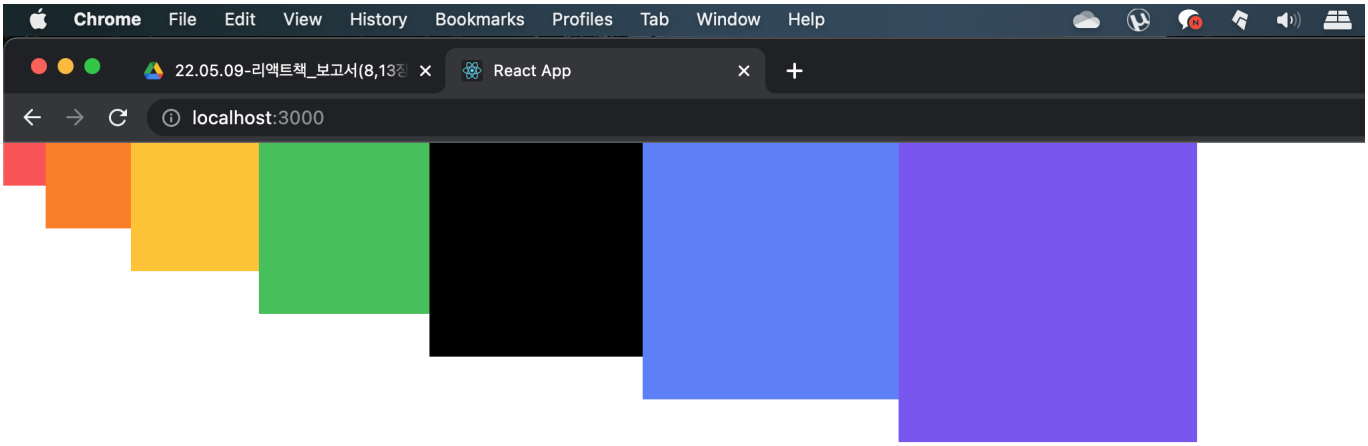
믹스인 만들기 재사용되는 스타일 블록을 함수처럼 사용 할 수 있음 이 믹스인(함수)를 호출한 위치에 {}안의 스타일 코드가 적용됨

```
.SassComponent {
  display: flex;
  background: $oc-gray-2;
  @include media("<768px") {
    background: $oc-gray-9;
  }
  .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;
    }
  }
}
```

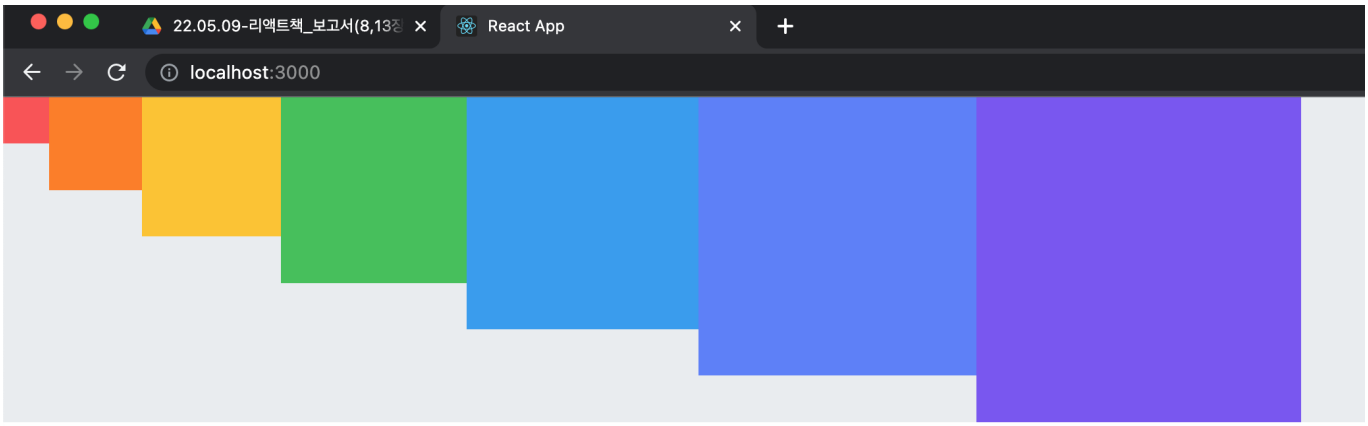
```
import React from 'react';
import './SassComponent.scss';

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;
```

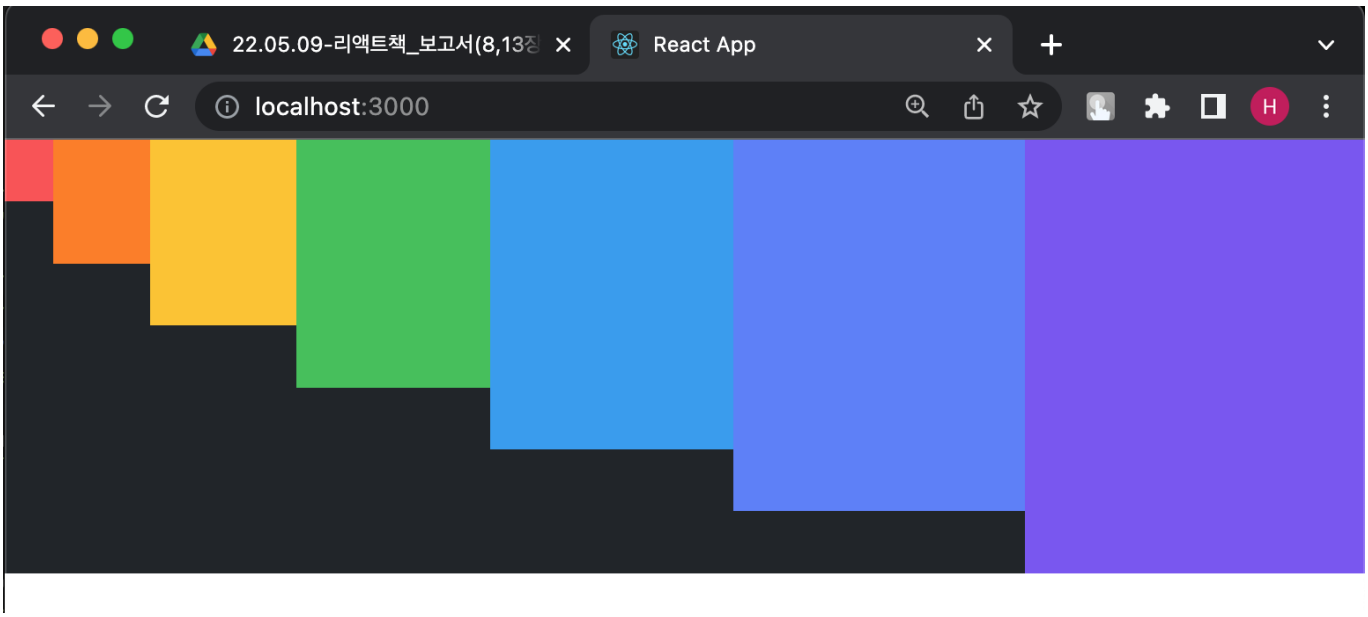





```
@import '~include-media/dist/include-media';
@import '~open-color/open-color';

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

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

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

.wrapper {
  background: black;
  padding: 1rem;
  color: white;
  font-size: 2rem;
  &.inverted {
    // inverted 가 .wrapper 와 함께 사용 됐을 때만 적용
    color: black;
    background: white;
    border: 1px solid black;
  }
}

/* 글로벌 CSS 를 작성하고 싶다면 */
:global {
  // :global {} 로 감싸기
  .something {
    font-weight: 800;
    color: aqua;
  }
  // 여기에 다른 클래스를 만들 수도 있겠죠?
}
```





9.2.1 utils함수 분리하기

다른 scss파일을 불러올때는 @import구문을 사용합니다

9.3 CSSMODULE

- CSS를 불러와서 사용할때 클래스 이름을 고유한 값 [파일이름]_[클래스이름]__[해시값(랜덤값)] 형태로 자동으로 만들어서 컴포넌트 스타일 클래스 이름이 중첩되는 현상을 방지해주는기술
- 설정할 필요없이 파일을 .module.css 확장자로 저장하지만하면 됨
- CSS Module을 사용하면 클래스이름을 지을때 그 고유성에 대해 고민하지 않아도 됨
- 특정 클래스가 웹페이지에서 전역적으로 사용되는 경우 :global을 앞에 입력

> 서로 다른 css파일에 같은 클래스 이름이 있어도 난독화 과정에서 서로다른 값으로 구분된다 css클래스 이름이 멤버변로 인식되므로 클래스 이름을 카멜표기법으로 지정해야한다 스네이크 표기법을 사용한 경우 styles["클래스이름"] 형식으로 적용해야한다 클래스를 배열로 배치하여 join()함수를 사용하는것도 가능함 {[styles.wrapper, styles.inverted].join(" ")}

9.3.1 classnames

CSS클래스를 조건부로 설정할때 매우 유용한 라이브러리

- 여러가지 종류의파라미터를 조합해 css클래스를 설정할수 있기때문에 조건부로 클래스를 설정할때 편함

- props값이 존재할 때만 props에 저장된 값이 class이름으로 사용됨
- 내장된 bind함수를 사용하면 클래스를 넣어줄 때마나 styles.[클래스이름] 형태를 사용안해도 됨
- 사전에 styles에서 받아 온후 사용하게끔 설정해두고 cx('클래스이름', '클래스이름2')형태로 사용

9.3.2 sass와 함께 하용하기

CSSModule.module.css => CSSModule.module.scss

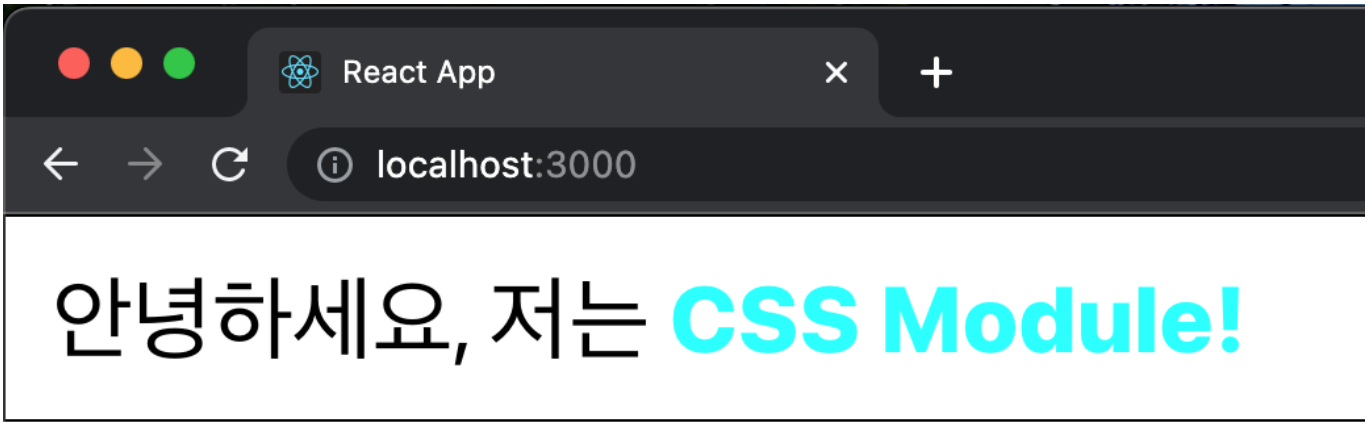
> CSSModule.module.scss

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

.wrapper {
  background: black;
  padding: 1rem;
  color: white;
  font-size: 2rem;
  &.inverted {
    // inverted 가 .wrapper 와 함께 사용 됐을 때만 적용
    color: black;
    background: white;
    border: 1px solid black;
  }
}

/* 글로벌 CSS 를 작성하고 싶다면 */
:global {
  // :global {} 로 감싸기
  .something {
    font-weight: 800;
    color: aqua;
  }
  // 여기에 다른 클래스를 만들 수도 있겠죠?
}
```

> CSSModule.js

```jsx
import React from 'react';
import classNames from 'classnames/bind';
import styles from './CSSModule.module.scss';

const cx = classNames.bind(styles); // 미리 styles 에서 클래스를 받아오도록 설정하고

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

```
};

export default CSSModule;
```



9.4 styled-components

- 자바스트립트 파일안에 스타일을 선언
- CSS in JS 라고 불림
- styled-components를 사용하면 자바스크립트 파일하나에 스타일까지 작성
- .css또는 .scss확장자를 가진 스타일 파일을 따로 만들지 않아도 된다

9.4.1 Tagged 탬플릿 리터럴

스타일 작성할때 `를 사용하여만든 문자열에 스타일 정보를 넣어주었습니다

9.4.2 스타일링된 엘리먼트 만들기

- styled-components를 사용하여 스타일링된 엘리먼트를 만들때는 컴포넌트 파일의 상단에서 styled를 불러오고 styled태그명을 사용하여 구현합니다
- styled.div뒤에 tagged탬플릿 리터럴 문법을 통해 스타일을 넣어주면, 해당 스타일이 적용된 div로 이루어진 리액트 컴포넌트가 생서됩니다
- 특정 컴포넌트 자체에 스타일링

9.4.3 스타일에서 props조회하기

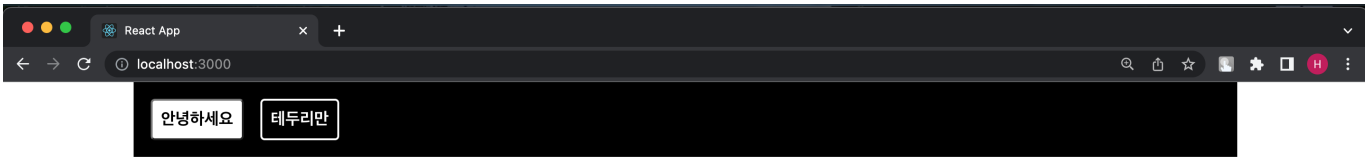
- 스타일 쪽에서 컴포넌트에게 전달ehls props값을 참조할

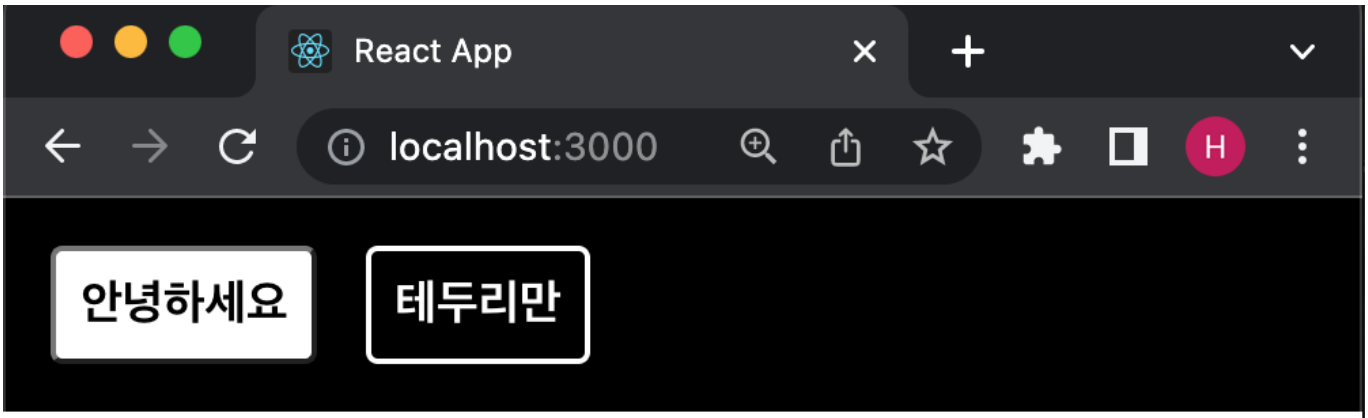
> StyledComponent.js

```
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.
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: 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 StyledComponent = () => (
  <Box color="black">
    <Button>안녕하세요</Button>
    <Button inverted={true}>테두리만</Button>
  </Box>
);

export default StyledComponent;
```





13. 리액트 라우터로 SPA개발 하기

SPA : single page application 한개의 페이지로 이루어진 어플리케이션

- 웹 서버라는 소프트웨어에 웹브라우저가접속한다
- 이 때 전달되는 URL은 웹 서버가 관리하는 폴더에 저장되어있는 HTML파일을 경로를 의미

- 웹브라우저가 웹서버에 저장되어있는 웹페이지를 열람
- 페이지 이동시마다 접속 해제가 이루어짐
- 뷰 렌더링을 사용자의 브라우저가 담당하도록 하고 우선 어플리케이션을 브라우저에 불러와서 실행시틴후 사용자와의 인터렉션이 발생하면 필요한 부분만 자바스크립트를 사용하여 업데이트
- 새로운 데이터가 필요하다면 서버 API를 호출하여 필요한 데이터만 새로 불러와 어플리케이션에서 사용
- 다른주소에 다른화면을 보여주는 것을 라우팅이라함

13.1 라우팅이란?

13.1.1 SPA의 단점

- 앱의 규모가 커지면 자바스크립트 파일이 너무 커진다
- 최조 접속시 모든화면을 구성하는 스크립트를 한번에 모두 로딩, 사용자가 실제로 방문하지 않을수도있는 페이지까지 불러옴
- 리액트 라우터처럼 브라우저에서 자바스크립트를 사용하여 라우팅을 관리하는 것은 자바스크립트를 실행하지 않는 일반 크롤러에서는 페이지의 정보를 제대로 수집해 가지 못한다는 잠재적 단점
    - 검색결과에 페이지가 잘 나타나지 않음
        - 서버사이드 렌더링으로 해결

13.2 싱글페이지 어플리케이션

13.2.2 프렉젝트에 라우터 적용

프로젝트에 리액트 라우터를 적용시 src/index.js파일에서 react-router-dom에 내장된 BrowserRouter라는 컴포넌트를 사용하여 감싸면 된다

Link컴포넌트를 사용하여 다른 주소로 이동하기

- Link컴포넌트는 클릭하면 다른 주소로 이동시켜주는 컴포넌트
- Link 컴포넌트를 사용하여 페이지를 전환하면 페이지를 새로 불러오지 않고 애플리케이션을 그대로 유지한 상태에서 HTML5 History API를 사용하여 페이지의 주소만 변경
- 페이지 전환을 방지하는 기능이 내장

13.3 리액트 라우터 적용 및 기본 사용법

여러개의 path에 같은 컴포넌트

> indes.js

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

ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
```

```
  document.getElementById('root')
);
```

src/pages/Home.js

```
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>
      </ul>
    </div>
  );
};

export default Home;
```

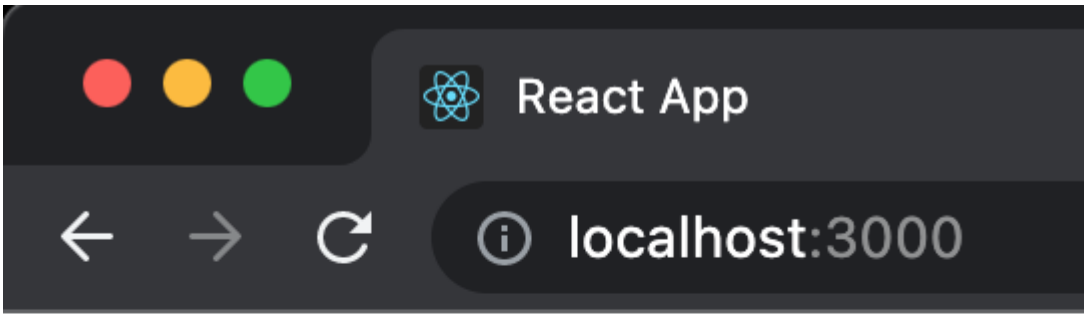
src/pages/about.js

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

    </div>
  );
};

export default About;
```

App.js

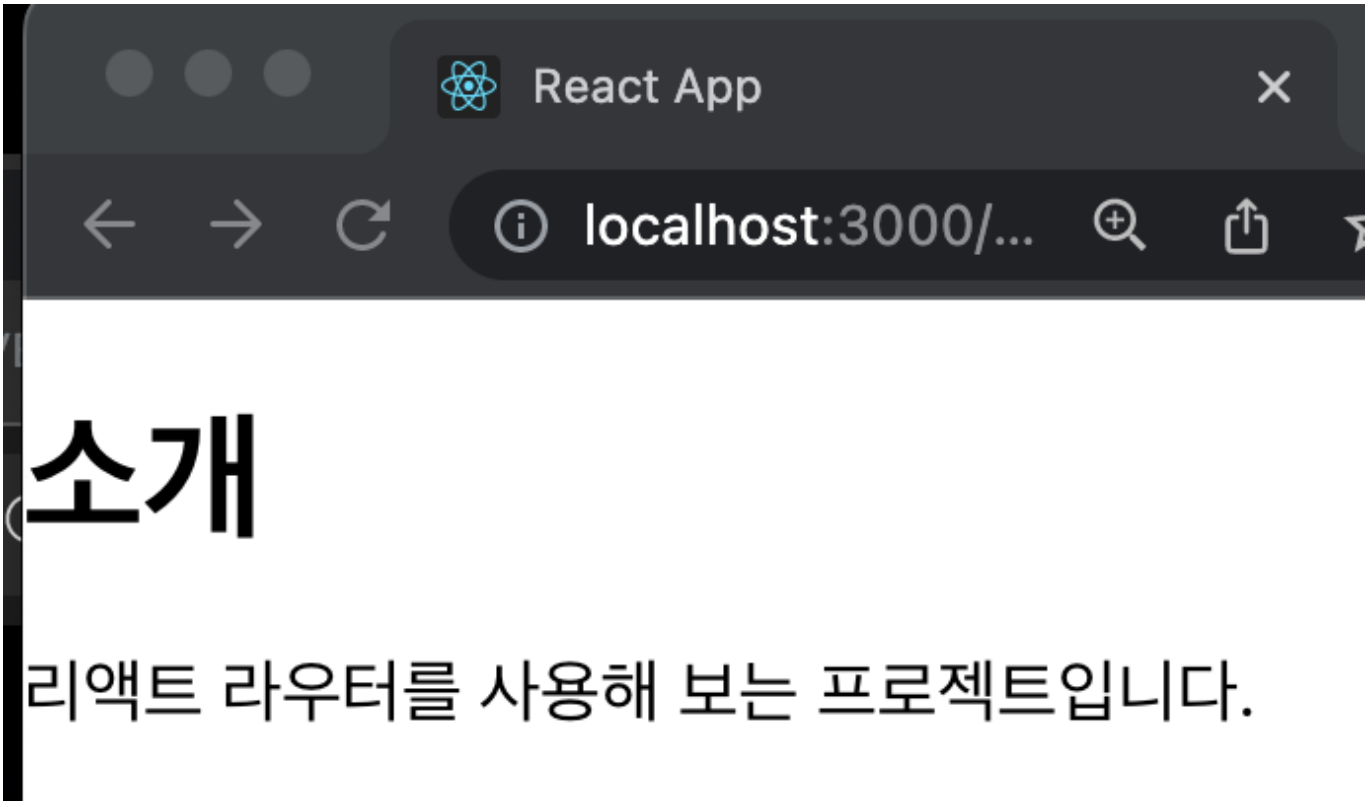
```
import { Route, Routes } from 'react-router-dom';
import About from './pages/About';
import Home from './pages/Home';


const App = () => {
```

```
    return (
      <Routes>

          <Route index element={<Home />} />
          <Route path="/about" element={<About />} />

      </Routes>
    );
  };

  export default App;
```



홈

가장 먼저 보여지는 페이지입니다.



13.4 URL파라미터와 쿼리스트링

react-router-dom패키지의 useParams()함수를 통해 URL파라미터가 저장되어 있는 객체를 리턴받을수 있다

```
import React from 'react';
import {useLocation} from 'react-router-dom';
const About = ()=>{
  const location = useLocation();
  const {search} = location;
```

```
  const query = new URLSearchParams(search);
}
const showDetail = Query.get('detail') === 'true';
```

- 쿼리를 사용할때는 쿼리 문자열을 객체로 피싱하는 과정에서 결과 값은 언제나 문자열
- 숫자를 받아 와야 하면 parseInt함수를 통해 꼭 숫자로 변환
- 논리 자료형 값을 사용해야 하는 경우에는 정확히 "true" 문자열이랑 일치하는지 비교

src/pages/Profile.js

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

```
import Profile from './pages/Profile';

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

export default App;
```

사용자 프로필

가나다

가나다라마바사

사용자 프로필

ABC

ABCDEFGHIJKLMNOPQRSTUVWXYZ

사용자 프로필

존재하지 않는 프로필입니다.

홈

가장 먼저 보여지는 페이지입니다.

- 소개
- 가나다의 프로필
- ABC의 프로필
- 존재하지 않는 프로필
- 게시글 목록

13.5 중첩된 라우트

- 라우트 내부에 또 라우트를 정의하는것

- 라우트는 사용되고 있는 컴포넌트 내부에 Route컴포넌트 또 사용
- jsx에서 props를 설정할 때 값을 생략하면 자동으로 true로 설정
    - 예를들어 Route 에서 exact만 적어도 exact={true}라는 의미
- route 컴포넌트에는 보여주고 싶은 jsx를 넣어줄 수 있음

> src/pages/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 = () => {
    searchParams.set('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;
```

13.6 부가기능

> withRouter => useParams(), useLocation()으로 대체됨

> Switch => Routes로 변경됨 여러 route를 감싸서 그중 URL이 일치하는 단하나의 라우트만을 핸더링

> NavLink 링크가 활성되었을때의 스타일을 적용할때는 activeStyle값을 CSS클래스를 적용할때는 activeClassName값을 props로 넣어 주면 됩니다

> src/pages/Articles.js

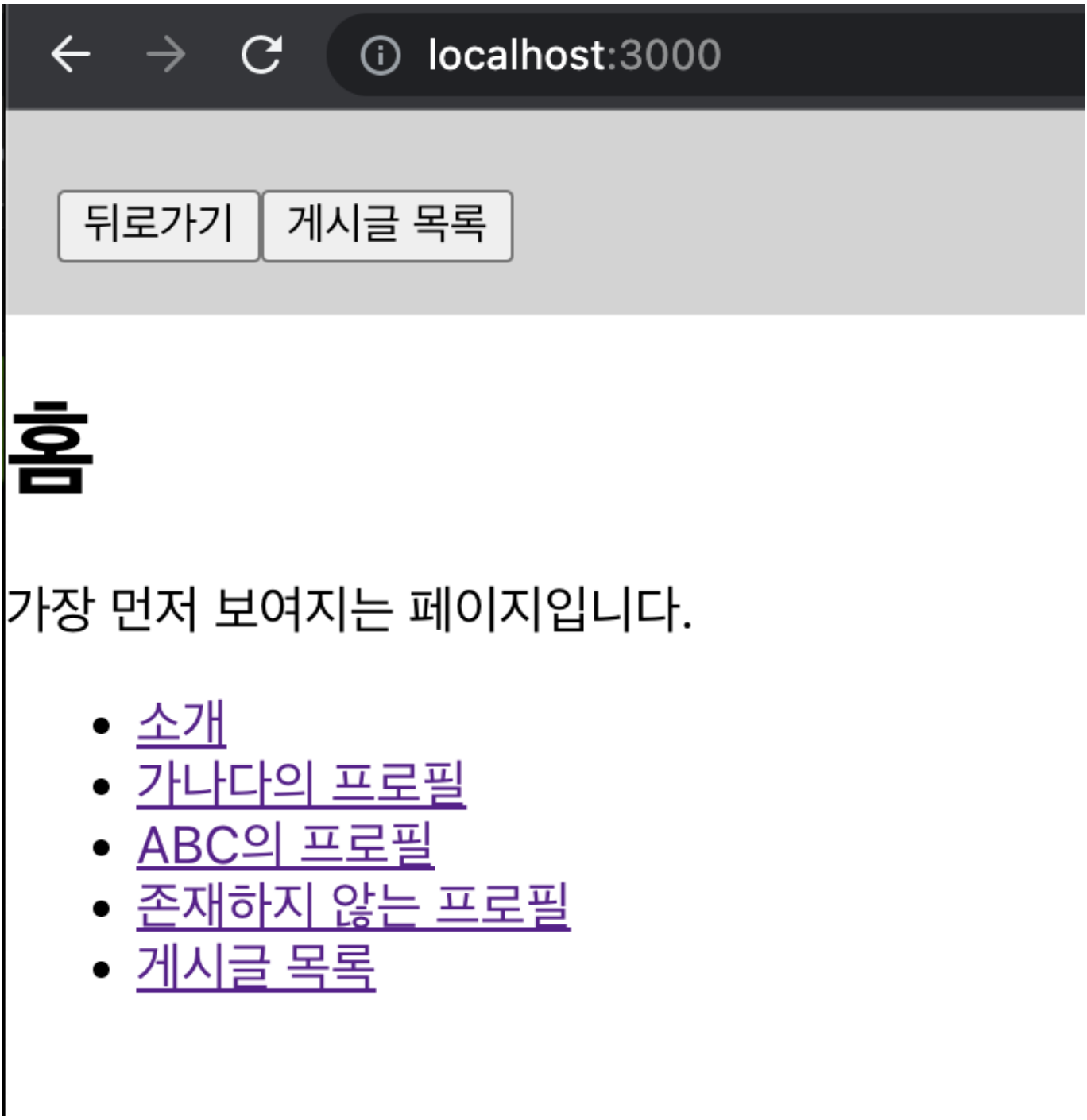
```
import { NavLink, Outlet } from 'react-router-dom';

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;
```

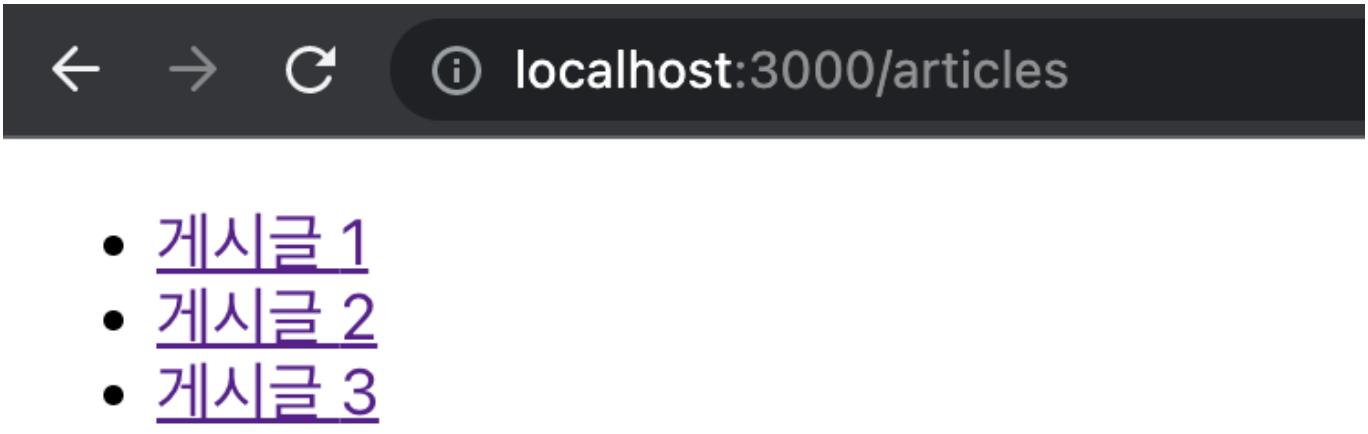
src/pages/Article.js

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

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

export default Article;
```

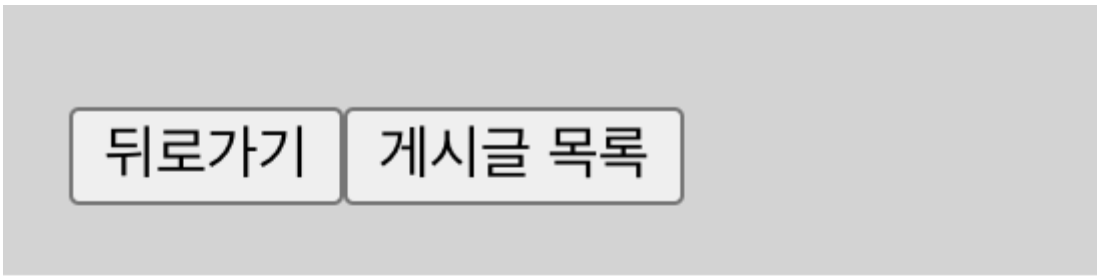
src/Layout.js

```
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', {
      replace: true,
    });
  };

  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;
```


localhost:3000
뒤로가기
게시글 목록
홈
가장 먼저 보여지는 페이지입니다.
소개
가나다의 프로필
ABC의 프로필
존재하지 않는 프로필
게시글 목록


localhost:3000/articles
게시글 1
게시글 2
게시글 3

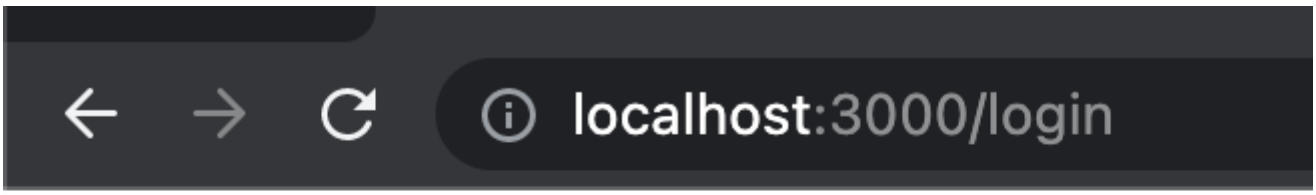

localhost:3000/articles/1
게시글 1
게시글 1
게시글 2
게시글 3



사용자 프로필

가나다

가나다라마바사



로그인 페이지