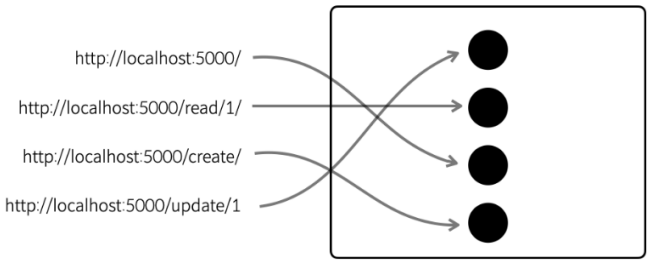
Web Programming

React programming

Router

라우팅

- 웹 애플리케이션에서 라우팅이라는 개념은 사용자가 요청한 URL에 따라 알맞는 페이지를 보여주는 것을 의미한다. 웹 애플리케이션을 만들 때 프로젝트를 하나의 페이지로 구성할 수도 있고, 여러 페이지를 구성할 수도 있다.
- 예를 들어, 일정 관리 애플리케이션에서는 하나의 페이지로 충분할 수 있겠지만, 블로그를 만든다고 가정하면 여러 페이지로 구성이 되어있다.
  - 글쓰기 페이지: 새로운 포스트를 작성하는 페이지.
  - 포스트 목록 페이지: 블로그에 작성된 여러 포스트들의 목록을 보여주는 페이지.
  - 포스트 읽기 페이지: 하나의 포스트를 보여주는 페이지.
- 이렇게 여러 페이지로 구성된 웹 애플리케이션을 만들 때 페이지 별로 컴포넌트들을 분리해가면서 프로젝트를 관리하기 위해 필요한 것이 바로 라우팅 시스템이다.



Routing

Router

SPA

- 싱글 페이지 애플리케이션이란, 한 개의 페이지로 이루어진 애플리케이션
- MPA는 사용자 인터랙션이 별로 없는 정적인 페이지들은 기존의 방식이 적합하지만, 사용자 인터랙션이 많고 다양한 정보를 제공하는 모던 웹 애플리케이션은 이 방식이 적합하지 않다
- HTML을 한번만 받아와서 웹 애플리케이션을 실행시킨 후에 그 이후에는 필요한 데이터만 받아와서 화면에 업데이트 해주는 것이 싱글 페이지 애플리케이션이다.
- 싱글 페이지 애플리케이션은 기술적으로는 한 페이지만 존재하는 것이지만, 사용자가 경험하기에는 여러 페이지가 존재하는 것 처럼 느낄 수 있다. 리액트 라우터와 같은 라우팅 시스템은 사용자의 브라우저 주소창의 경로에 따라 알맞는 페이지를 보여주는데, 이후 링크를 눌러서 다른 페이지로 이동하게 될 때 서버에 다른 페이지의 html을 새로 요청하는 것이 아니라, 브라우저의 History API를 사용하여 브라우저의 주소창의 값만 변경하고 기존에 페이지에 띄웠던 웹 애플리케이션을 그대로 유지하면서 라우팅 설정에 따라 또 다른 페이지를 보여주게 된다.

Router

리액트 라우터 적용 및 기본 사용법

프로젝트 생성 및 라이브러리 설치

↓

페이지를 만들고 이동해보기

↓

URL 파라미터와 쿼리스트링 사용해보기

↓

중첩된 라우트 구현하기

↓

리액트 라우터의 부가기능 사용해보기

Router

프로젝트 생성 및 라이브러리 설치

- npx create-react-app router-tutorial or yarn create react-app router-tutorial
- cd router-tutorial
- yarn add react-router-dom or npm i react-router-dom

프로젝트에 라우터 적용

- src/index.js 파일에서 react-router-dom에 내장되어 있는 BrowserRouter라는 컴포넌트를 사용하여 감싸면 된다. 이 컴포넌트는 웹 애플리케이션에 HTML5의 History API를 사용하여 페이지를 새로 불러오지 않고도 주소를 변경하고 현재 주소의 경로에 관련된 정보를 리액트 컴포넌트에서 사용할 수 있도록 해 준다.

Router

프로젝트에 라우터 적용

- src/index.js 파일에서 react-router-dom에 내장되어 있는 BrowserRouter라는 컴포넌트를 사용하여 감싸면 된다. 이 컴포넌트는 웹 애플리케이션에 HTML5의 History API를 사용하여 페이지를 새로 불러오지 않고도 주소를 변경하고 현재 주소의 경로에 관련된 정보를 리액트 컴포넌트에서 사용할 수 있도록 해 준다.

src/index.js

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

const container = document.getElementById("root");
const root = createRoot(container);
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

Router

페이지 컴포넌트 만들기

- 사용자가 웹 사이트에 들어오게 됐을 때 가장 먼저 보여지게 될 Home 페이지 컴포넌트와 웹 사이트를 소개하는 About 페이지 컴포넌트를 만들기.
- src 디렉터리에 pages 디렉토리를 만들고, 그 안에 다음 파일들을 생성.

src/pages/Home.js

```
import React from 'react';

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

export default Home;
```

src/pages/About.js

```
import React from 'react';

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

export default About;
```

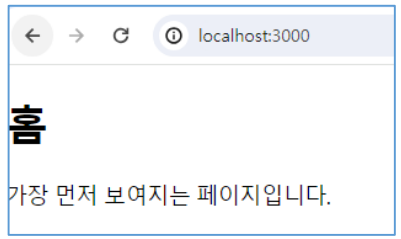
Router

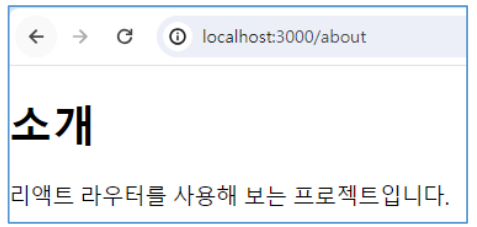
Route 컴포넌트로 특정 경로에 원하는 컴포넌트 보여주기

- 사용자의 브라우저 주소 경로에 따라 원하는 컴포넌트를 보여주기 위해서 Route라는 컴포넌트를 통해 라우트 설정을 해주어야 한다.
- Route 컴포넌트는 다음과 같이 사용한다.
  - ➢ <Route path="주소규칙" element={보여 줄 컴포넌트 JSX} />
- 그리고, Route 컴포넌트는 Routes 컴포넌트 내부에서 사용.
- App 컴포넌트를 다음과 같이 Route 컴포넌트를 사용하여 라우트 설정.

App.js

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

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

export default App;
```





Router

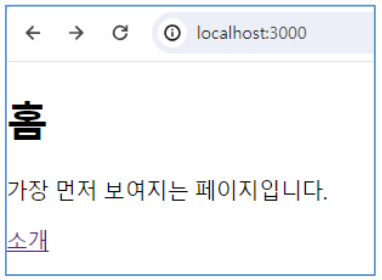
Link 컴포넌트를 사용하여 다른 페이지로 이동하는 링크 보여주기

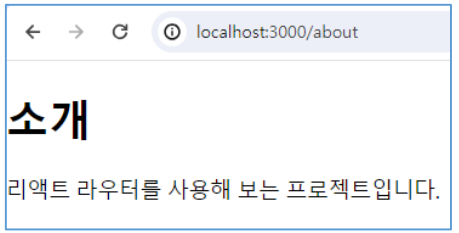
- Link 컴포넌트는 a 태그를 사용하긴 하지만, 페이지를 새로 불러오는 것을 막고 History API를 통해 브라우저 주소의 경로만 바꾸는 기능이 내장되어 있다.
- Link 컴포넌트는 다음과 같이 사용한다.
  - ➢ <Link to="경로">링크 이름</Link>
- Link 컴포넌트를 Home 페이지 컴포넌트에서 사용.

웹 페이지에서는 원래 링크를 보여줄 때 a 태그를 사용하는데, 리액트 라우터를 사용하는 프로젝트에서 a 태그를 바로 사용하면 안된다. 왜냐하면, a 태그를 클릭하여 페이지를 이동할 때 브라우저에서는 페이지를 새로 불러오게 되기 때문.

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>
      <Link to="/about">소개</Link>
    </div>
  );
};

export default Home;
```





Router

URL 파라미터와 쿼리스트링

- 페이지 주소를 정의할 때 가끔은 유동적인 값을 사용해야 할 때도 있다.
  - URL 파라미터 예시: /profile/velopert
  - 쿼리스트링 예시: /articles?**page=1&keyword=react
- URL 파라미터는 주소의 경로에 유동적인 값을 넣는 형태고, 쿼리 스트링은 주소의 뒷부분에 ? 문자열 이후에 key=value 로 값을 정의하며 & 로 구분을 하는 형태다.
- 주로 URL 파라미터는 ID 또는 이름을 사용하여 특정 데이터를 조회할 때 사용하고, 쿼리스트링(Querystring)은 키워드 검색, 페이지네이션, 정렬 방식 등 데이터 조회에 필요한 옵션을 전달할 때 사용.

Router

URL 파라미터

- Profile 컴포넌트를 pages 디렉토리에 다음과 같이 작성

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

const data = {
tealighting: {
    name: '김다빈',
    description: '보안에 관심있는 학생',
  },
  dew: {
    name: '이이슬',
    description: 'DB에 관심있는 학생',
  },
  yellow: {
    name:  ' 조윤정',
    description:  ' 컴퓨터에 관심있는 학생',
  },
  frontend: {
    name: '최예진',
    description: 'front-end에 관심있는 학생',
  },
};
```

```
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;
```

Router

URL 파라미터

- URL 파라미터는 useParams라는 Hook을 사용하여 객체 형태로 조회할 수 있다. URL 파라미터의 이름은 라우트 설정을 할 때 Route 컴포넌트의 path props를 통하여 설정.
- 앞의 코드에서는 data 객체에 예시 프로필 정보들을 key-value 형태로 담아두었다. 그리고, Profile 컴포넌트에서는 username URL 파라미터를 통하여 프로필을 조회한 뒤에 프로필이 존재하지 않으면 '존재하지 않는 프로필입니다.' 라는 문구를 보여주고 존재한다면 프로필 정보를 보여주도록 로직을 작성했다.
- App 컴포넌트 파일을 열어서 새로운 라우트를 다음과 같이 설정.

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

export default App;
```

Router

URL 파라미터

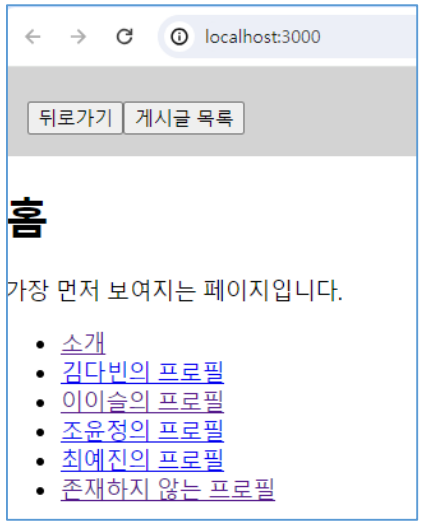
- URL 파라미터는 /profiles/:username과 같이 경로에 : 를 사용하여 설정.
- 만약 URL 파라미터가 여러 개인 경우엔 /profiles/:username/:field 와 같은 형태로 설정.
- Profile 페이지로 이동을 할 수 있도록 Home 페이지에 Link를 생성.
- 링크가 여러 개이기 때문에, ul 태그를 사용하여 리스트 형태로 보여줌.

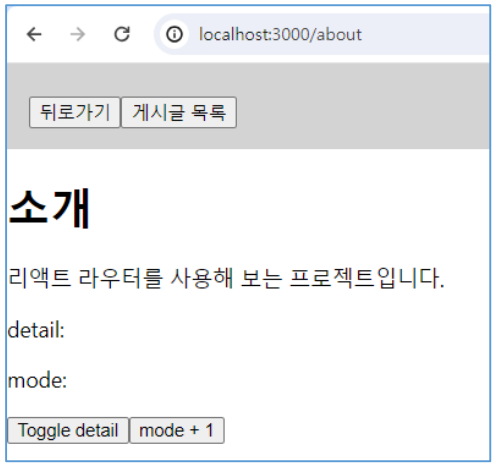
```
import React from 'react';
import { Link } from 'react-router-dom';

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/tealighting">김다빈의 프로필</Link>
        </li>
        <li>
          <Link to="/profiles/dew">이이슬의 프로필</Link>
        </li>
        <li>
          <Link to="/profiles/yellow">조윤정의 프로필</Link>
        </li>
        <li>
          <Link to="/profiles/frontend">최예진의 프로필</Link>
        </li>
        <li>
          <Link to="/profiles/void">존재하지 않는 프로필</Link>
        </li>
      </ul>
    </div>
  );
};
export default Home;
```

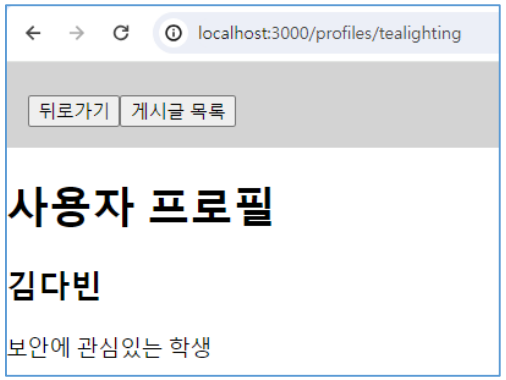
Router

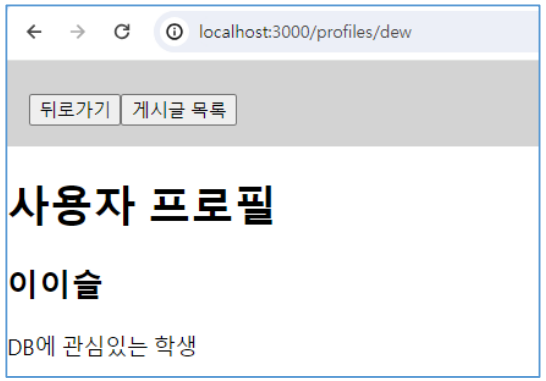
URL 파라미터

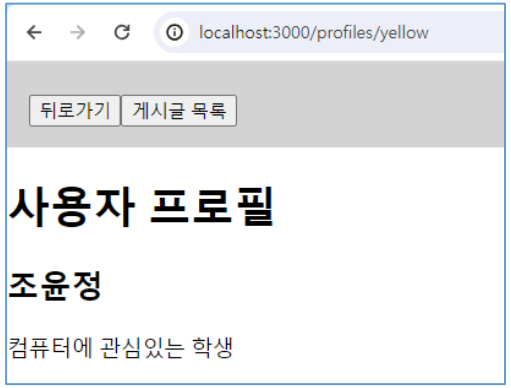
- URL 파라미터에 따라 다른 결과물이 잘 보임

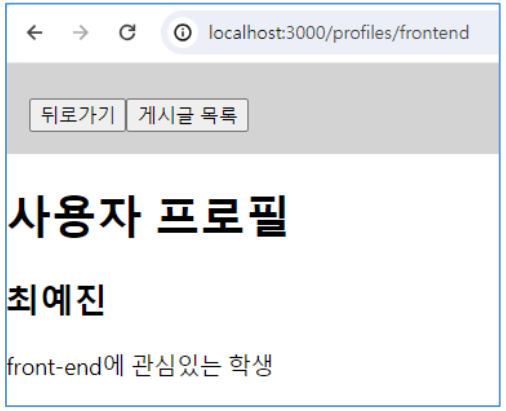


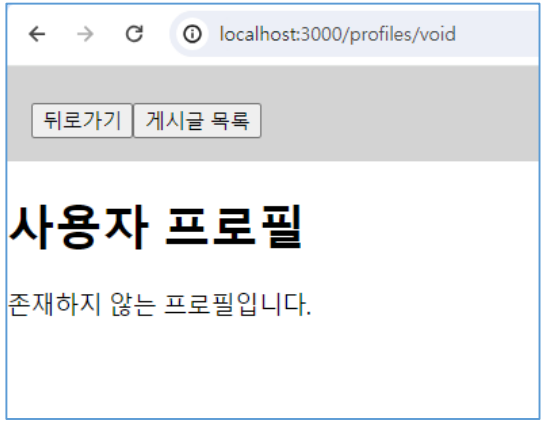












Router

쿼리스트링

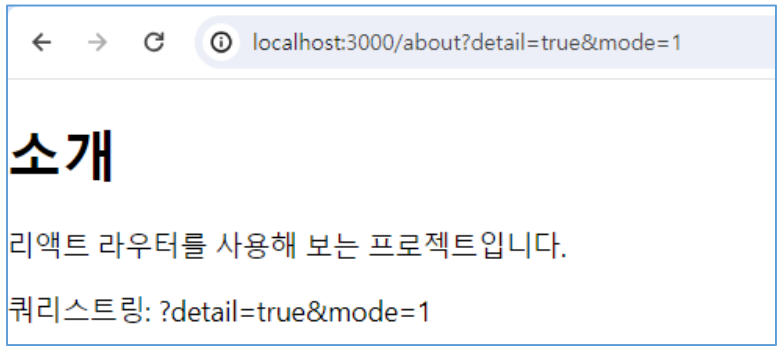
- 쿼리스트링을 사용할 때는 URL 파라미터와 달리 Route 컴포넌트를 사용할 때 별도로 설정해야되는 것은 없다.
- About 페이지 컴포넌트 수정

```
import React from 'react';
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;
```

Router

쿼리스트링

- useLocation이라는 Hook은 location 객체를 반환하며 이 객체는 현재 사용자가 보고있는 페이지의 정보를 지니고 있다. 이 객체에는 다음과 같은 값들이 있다.
  - ✓ pathname: 현재 주소의 경로 (쿼리스트링 제외)
  - ✓ search: 맨 앞의 ? 문자 포함한 쿼리스트링 값
  - ✓ hash: 주소의 # 문자열 뒤의 값 (주로 History API 가 지원되지 않는 구형 브라우저에서 클라이언트 라우팅을 사용할 때 쓰는 해시 라우터에서 사용.)
  - ✓ state: 페이지로 이동할때 임의로 넣을 수 있는 상태 값
  - ✓ key: location 객체의 고유 값, 초기에는 default 이며 페이지가 변경될 때마다 고유의 값이 생성됨
- 쿼리스트링은 location.search 값을 통해 조회를 할 수 있다.
- 주소창에 http://localhost:3000/about?detail=true&mode=1 라고 직접 입력



Router

쿼리스트링

- 리액트 라우터에서는 v6부터 useSearchParams라는 Hook을 통해서 쿼리스트링을 쉽게 다룰 수 있다.
- 다음은 이 Hook을 사용하여 쿼리스트링을 쉽게 파싱하여 사용하는 예시다.

```
import React from 'react';
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;
```

Router

쿼리스트링

- useSearchParams는 배열 타입의 값을 반환하며, 첫번째 원소는 쿼리 파라미터를 조회하거나 수정하는 메서드들이 담긴 객체를 반환한다.
- get 메서드를 통해 특정 쿼리 파라미터를 조회할 수 있고, set 메서드를 통해 특정 쿼리 파라미터를 업데이트 할 수 있다.
- 만약 조회 시에 쿼리 파라미터가 존재하지 않는다면 null 로 조회된다.
- 두번째 원소는 쿼리 파라미터를 객체 형태로 업데이트할 수 있는 함수를 반환한다.
- 쿼리 파라미터를 사용하실 때 주의하실점은 쿼리 파라미터를 조회할 때 값은 무조건 문자열 타입이라는 것이다.
- 즉, true 또는 false 값을 넣게 된다면 값을 비교할 때 꼭 'true' 와 같이 따옴표로 감싸서 비교를 해야 하고, 숫자를 다루게 된다면 parseInt를 사용하여 숫자 타입으로 변환을 해야 한다.

Router

중첩된 라우트

- 중첩된 라우트를 이해하기 위해, 게시글 목록을 보여주는 페이지와 게시글을 읽는 페이지 생성.
- pages 디렉터리에 다음 컴포넌트들 생성

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;
```

Router

중첩된 라우트

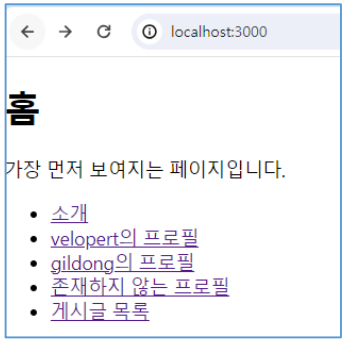
- 해당 페이지들의 라우트를 App 컴포넌트에서 설정

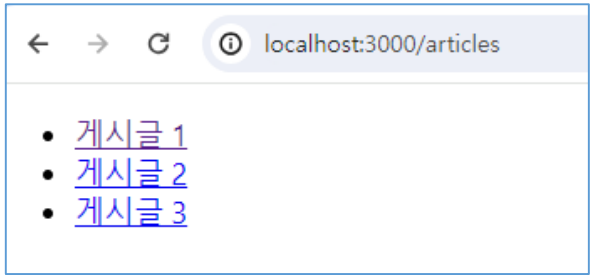
```
import React from 'react';
import { Route, Routes } from 'react-router-dom';
import About from './pages/About';
import Article from './pages/Article';
import Articles from './pages/Articles';
import Home from './pages/Home';
import Profile from './pages/Profile';

const App = () => {
  return (
    <Routes>
      <Route path="/" 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>
  );
};

export default App;
```

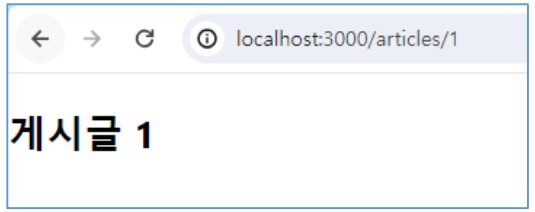
중첩된 라우트

- Home 컴포넌트에서 게시글 목록 페이지로 가는 링크 추가

```
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 프로필</Link>
        </li>
        <li>
          <Link to="/profiles/gildong">gildong의 프로필</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;
```







Router

중첩된 라우트

- 중첩된 라우트 형태로 라우트를 설정

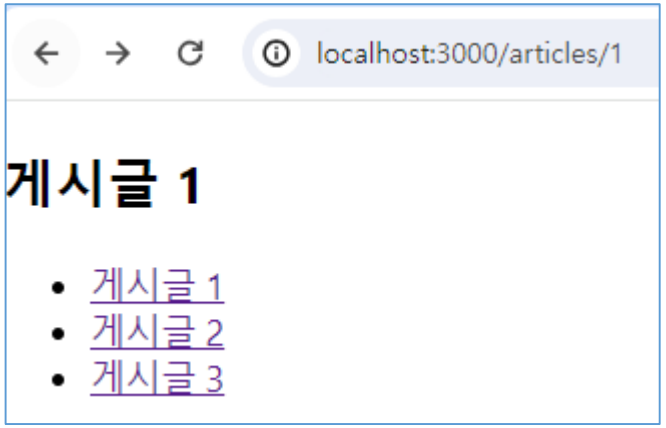
```
import React from 'react';
import { Route, Routes } from 'react-router-dom';
import About from './pages/About';
import Article from './pages/Article';
import Articles from './pages/Articles';
import Home from './pages/Home';
import Profile from './pages/Profile';

const App = () => {
  return (
    <Routes>
      <Route path="/" 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>
  );
};

export default App;
```

Router

중첩된 라우트

- Articles 컴포넌트에서 리액트 라우터에서 제공하는 Outlet 이라는 컴포넌트를 사용해주어야 한다.
- 이 컴포넌트는 Route의 children 으로 들어가는 JSX 엘리먼트를 보여주는 역할을 한다.

```
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;
```



Router

공통 레이아웃 컴포넌트

- 중첩된 라우트와 Outlet은 페이지끼리 공통적으로 보여줘야 하는 레이아웃이 있을때도 유용하게 사용할 수 있다.
- 예를 들어, Home, About, Profile 페이지에서 상단에 헤더를 보여줘야 하는 상황을 가정해자.
- 첫 번째로 드는 생각은 아마 Header 컴포넌트를 따로 만들어두고 각 페이지 컴포넌트에서 재사용을 하는 방법일 것이다.
- 물론 이 방법이 틀린것은 아니지만, 방금 배운 중첩된 라우트와 Outlet을 활용하여 구현을 할 수도 있다.
- 중첩된 라우트를 사용하는 방식을 사용하면 컴포넌트를 한번만 사용해도 된다는 장점이 있다. 상황에 따라, 취향에 따라 구현을 하면 된다.

Router

공통 레이아웃 컴포넌트

- 공통 레이아웃을 위한 Layout 컴포넌트를 src 디렉터리에 생성

```
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;
```

Router

공통 레이아웃 컴포넌트

- App 컴포넌트를 다음과 같이 수정

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

const App = () => {
  return (
    <Routes>
      <Route element={<Layout />}>
        <Route path="/" 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>
  );
};

export default App;
```

Router

index props

- Route 컴포넌트에는 index라는 props가 있다.
- 이 props는 path="/"와 동일한 의미를 갖는다.

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

const App = () => {
  return (
    <Routes>
      <Route path="/" 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>
  );
};
```

Router

useNavigate

- useNavigate는 Link 컴포넌트를 사용하지 않고 다른 페이지로 이동을 해야 하는 상황에 사용하는 Hook.

```
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;
```

Router

useNavigate

- navigate 함수를 사용할 때 파라미터가 숫자 타입이라면 앞으로 가거나, 뒤로 간다. 예를 들어서 navigate(-1) 을 하면 한 번 뒤로 가고 navigate(-2) 를 하면 두 번 뒤로 간다. 반대로 navigate(1) 을 하면 앞으로 한 번 간다. 물론, 뒤로 가기를 한번 한 상태이여야 한다.
- 다른 페이지로 이동을 할 때 replace 라는 옵션이 있는데, 이 옵션을 사용하면 페이지를 이동할 때 현재 페이지를 페이지 기록에 남기지 않는다.
- 방금 작성했던 goArticles 함수를 다음과 같이 수정해보세요.

```
const goArticles = () => {
  navigate('/articles', { replace: true });
}
```

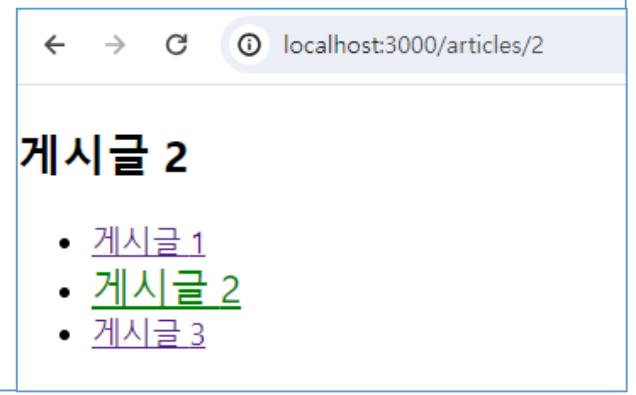
- 그 다음에 / 경로로 들어가서 Home 페이지를 띄운 뒤에, 소개 링크를 눌러 About 페이지로 이동하고, 상단의 게시글 목록 페이지를 누름.
- 그 상태에서 브라우저의 뒤로 가기 버튼을 눌러 이전 페이지로 이동을 하자. 만약 { replace: true } 설정이 없었더라면 직전에 봤던 페이지인 About 페이지가 나타나야 하지만, 이 옵션이 활성화되어있기 때문에, 그 전의 페이지인 Home 페이지가 나타나게 된다.

Router

useNavigate

- Articles 페이지 컴포넌트에서 이 컴포넌트를 사용

```
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;
```



Router

useNavigate

- 반복되는 코드가 여러 번 사용
- NavLink를 감싼 또 다른 컴포넌트를 만들어서 다음과 같이 리팩토링하여 사용하시는 것을 권장

```
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;
```

Router

NotFound 페이지 만들기

- 이 페이지는 사전에 정의되지 않는 경로에 사용자가 진입했을 때 보여주는 페이지.
- 즉, 페이지를 찾을 수 없을 때 나타나는 페이지.
- NotFound 컴포넌트를 pages 디렉터리에 만들어주세요.

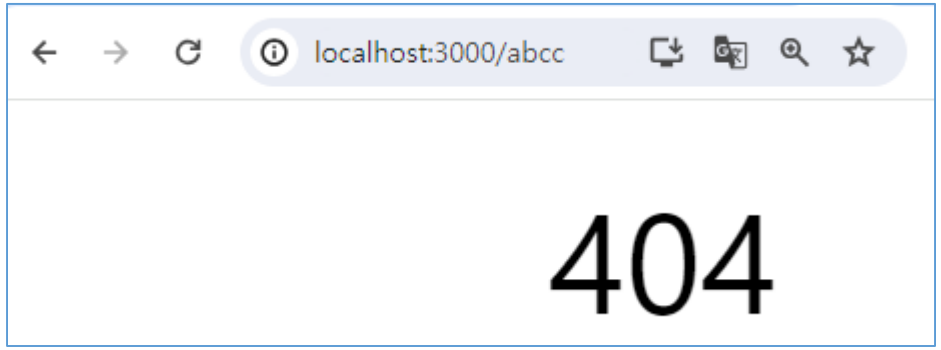
```
const NotFound = () => {
  return (
    <div
      style={{
        display: 'flex',
        alignItems: 'center',
        justifyContent: 'center',
        fontSize: 64,
        position: 'absolute',
        width: '100%',
        height: '100%',
      }}
    >
      404
    </div>
  );
};

export default NotFound;
```

Router

NotFound 페이지 만들기

- *는 wildcard 문자인, 이는 아무 텍스트나 매칭한다는 뜻.
- 이 라우트 엘리먼트의 상단에 위치하는 라우트들의 규칙을 모두 확인하고, 일치하는 라우트가 없다면 이 라우트가 화면에 나타나게 된다.

```
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';
import Layout from './pages/Layout';
import NotFound from './pages/NotFound';
```




```
const App = () => {
  return (
    <Routes>
      <Route path="/" 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>
  );
};

export default App;
```

Router

Navigate 컴포넌트

- Navigate 컴포넌트는 컴포넌트를 화면에 보여주는 순간 다른 페이지로 이동을 하고 싶을 때 사용하는 컴포넌트. 즉, 페이지를 리다이렉트 하고 싶을 때 사용.
- 예를 들어, 사용자의 로그인이 필요한 페이지인데 로그인을 안했다면 로그인 페이지를 보여줘야 되는 상황에 사용 할 수 있다.
- 두 페이지 컴포넌트를 pages 디렉터리에 생성.

```
import React from 'react';

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

export default Login;
```

```
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;
```

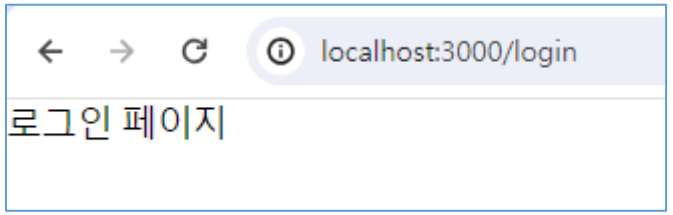
Router

Navigate 컴포넌트

- App 컴포넌트를 다음과 같이 수정

```
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';
import Layout from './pages/Layout';
import NotFound from './pages/NotFound';
import Login from './pages/Login';
import MyPage from './pages/MyPage';

const App = () => {
  return (
    <Routes>
      <Route path="/" 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>
  );
};

export default App;
```

브라우저에서 /mypage 경로로 이동



Mission

네비게이션 바 만들기

발표주제

톡방 공지

Reference

- https://reactrouter.com/en/6.23.1
- https://velog.io/@velopert/react-router-v6-tutorial
- https://www.opentutorials.org/course/4609/30207
- https://wikibook.github.io/react/router.html

TL;DR