1. 프로젝트 준비 및 기본적인 사용법

우선 리액트 라우터를 사용할 프로젝트를 준비해주겠습니다.

```
$ npx create-react-app router-tutorial
```

그 다음엔 해당 프로젝트 디렉토리로 이동하여 라우터 관련 라이브러리를 설치하겠습니다.

```
$ cd router-tutorial
$ npm install react-router-dom
```

프로젝트에 라우터 적용

라우터 적용은 index.js 에서 `BrowserRouter` 라는 컴포넌트를 사용하여 구현하시면 됩니다.

src/index.js

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

// * App 을 BrowserRouter 로 감싸기
ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById('root')
);

serviceWorker.unregister();
```

페이지 만들기

이제 라우트로 사용 할 페이지 컴포넌트를 만들 차례입니다. 웹사이트에 가장 처음 들어왔을 때 보여줄 Home 컴포넌트와, 웹사이트의 소개를 보여주는 About 페이지를 만들어보겠습니다.

src/Home.js

```
import React from 'react';

const Home = () => {
  return (
    <div>
      <h1>홈</h1>
```

```
      <p>이곳은 홈이에요. 가장 먼저 보여지는 페이지죠.</p>
    </div>
  );
};

export default Home;
```

src/About.js

```
import React from 'react';

const About = () => {
  return (
    <div>
      <h1>소개</h1>
      <p>이 프로젝트는 리액트 라우터 기초를 실습해보는 예제 프로젝트랍니다.</p>
    </div>
  );
};

export default About;
```

이제 페이지로 사용 할 모든 컴포넌트가 완료되었습니다.

Route: 특정 주소에 컴포넌트 연결하기

사용자가 요청하는 주소에 따라 다른 컴포넌트를 보여줘보겠습니다. 이 작업을 할 때에는 `Route` 라는 컴포넌트를 사용합니다.
사용 방식은 다음과 같습니다:

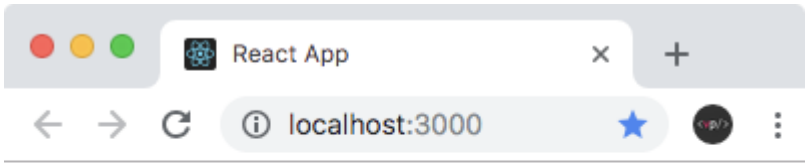
```
<Route path="주소규칙" component={보여주고싶은 컴포넌트}>
```

한번 App.js 에서 기존 코드들을 날리고, Route 들을 렌더링해주겠습니다.

src/App.js

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

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

export default App;
```

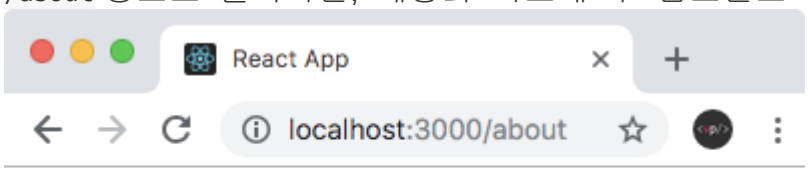
여기까지 하고 한번 `yarn start` 를 하여 개발서버를 시작해보세요.
이렇게 `/` 경로로 들어가면 홈 컴포넌트가 뜨고,



홈

이곳은 홈이에요. 가장 먼저 보여지는 페이지죠.

`/about` 경로로 들어가면, 예상과 다르게 두 컴포넌트가 모두 보여집니다!



홈

이곳은 홈이에요. 가장 먼저 보여지는 페이지죠.

소개

이 프로젝트는 리액트 라우터 기초를 실습해보는 예제 프로젝트
랍니다.

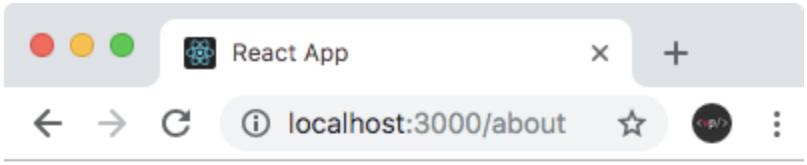
이는 `/about` 경로가 `/` 규칙과도 일치하기 때문에 발생한 현상인데요, 이를 고치기 위해선 Home 을 위한 라우트에 exact 라는 props 를 true 로 설정하시면 됩니다.

src/App.js

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

const App = () => {
```

```
  return (
    <div>
      <Route path="/" exact={true} component={Home} />
      <Route path="/about" component={About} />
    </div>
  );
};

export default App;
```

이렇게 하면 경로가 완벽히 똑같을때만 컴포넌트를 보여주게 되어 이슈가 해결됩니다.



소개

이 프로젝트는 리액트 라우터 기초를 실습해보는 예제 프로젝트
랍니다.

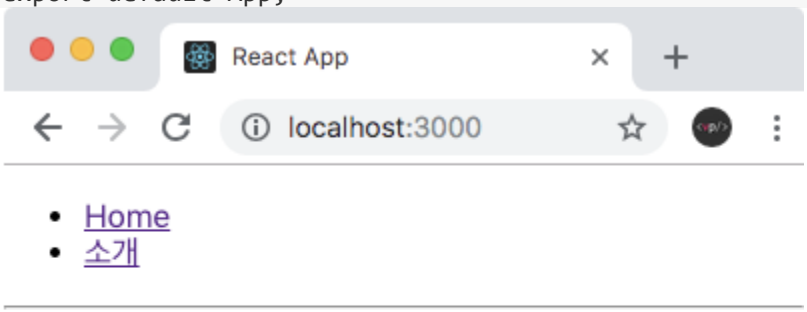
Link: 누르면 다른 주소로 이동시키기

Link 컴포넌트는 클릭하면 다른 주소로 이동시키는 컴포넌트입니다. 리액트 라우터를 사용할땐 일반 `<a href="...">...</a>` 태그를 사용하시면 안됩니다. 만약에 하신다면 onClick 에 `e.preventDefault()` 를 호출하고 따로 자바스크립트로 주소를 변환시켜주어야 합니다.
그 대신에 Link 라는 컴포넌트를 사용해야하는데요, 그 이유는 a 태그의 기본적인 속성은 페이지를 이동시키면서, 페이지를 아예 새로 불러오게됩니다. 그렇게 되면서 우리 리액트 앱이 지니고있는 상태들도 초기화되고, 렌더링된 컴포넌트도 모두 사라지고 새로 렌더링을 하게됩니다. 그렇기 때문에 a 태그 대신에 Link 컴포넌트를 사용하는데요, 이 컴포넌트는 HTML5 History API 를 사용하여 브라우저의 주소만 바꿀뿐 페이지를 새로 불러오지는 않습니다.

그럼 어디 한번 사용해볼까요?

src/App.js

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

const App = () => {
  return (
    <div>
      <ul>
        <li>
          <Link to="/">홈</Link>
        </li>
        <li>
          <Link to="/about">소개</Link>
        </li>
      </ul>
      <hr />
      <Route path="/" exact={true} component={Home} />
      <Route path="/about" component={About} />
    </div>
  );
};

export default App;
```



2. 파라미터와 쿼리

페이지 주소를 정의 할 때, 우리는 유동적인 값을 전달해야 할 때도 있습니다. 이는 파라미터와 쿼리로 나뉘어질 수 있는데요:

```
파라미터: /profiles/velopert
쿼리: /about?details=true
```

이것을 사용하는것에 대하여 무조건 따라야 하는 규칙은 없지만, 일반적으로는 파라미터는 특정 id 나 이름을 가지고 조회를 할 때 사용하고, 쿼리의 경우엔 어떤 키워드를 검색하거나, 요청을 할 때 필요한 옵션을 전달 할 때 사용됩니다.

URL Params

Profile 페이지에서 파라미터를 사용해봅시다.

`/profile/velopert` 이런식으로 뒷부분에 username 을 넣어줄 때 해당 값을 파라미터로 받아와보겠습니다.
Profile 이라는 컴포넌트를 만들어서 파라미터를 받아오는 예제 코드를 작성해보겠습니다.

src/Profile.js

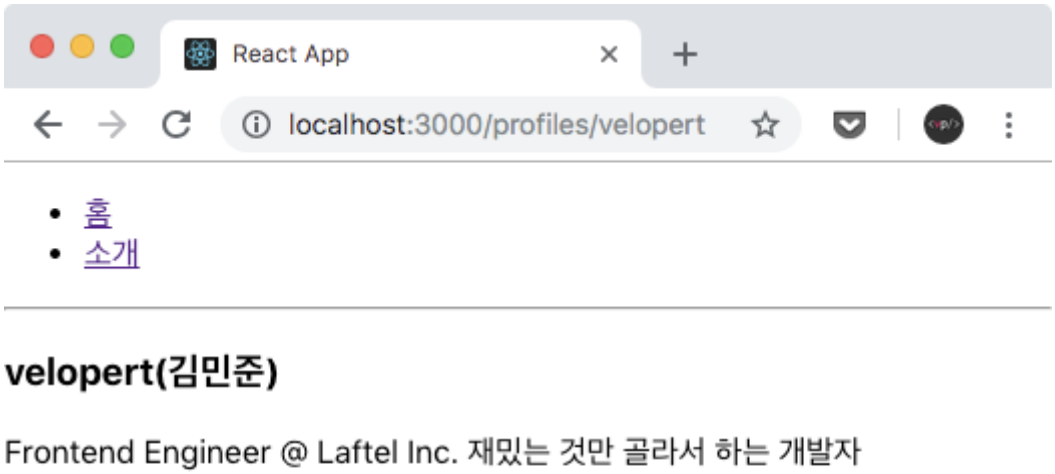
```
import React from 'react';

// 프로필에서 사용 할 데이터
const profileData = {
  velopert: {
    name: '김민준',
    description:
      'Frontend Engineer @ Laftel Inc. 재밌는 것만 골라서 하는 개발자'
  },
  gildong: {
    name: '홍길동',
    description: '전래동화의 주인공'
  }
};

const Profile = ({ match }) => {
  // 파라미터를 받아올 땐 match 안에 들어있는 params 값을 참조합니다.
  const { username } = match.params;
  const profile = profileData[username];
  if (!profile) {
    return <div>존재하지 않는 유저입니다.</div>;
  }
  return (
    <div>
      <h3>
        {username}({profile.name})
      </h3>
      <p>{profile.description}</p>
    </div>
  );
};

export default Profile;
```

파라미터를 받아올 땐 match 안에 들어있는 params 값을 참조합니다. match 객체안에는 현재의 주소가 `Route` 컴포넌트에서 정한 규칙과 어떻게 일치하는지에 대한 정보가 들어있습니다.
이제 Profile 을 App 에서 적용해볼건데요, path 규칙에는 `/profiles/:username` 이라고 넣어주면 username 에 해당하는 값을 파라미터로 넣어주어서 Profile 컴포넌트에서 match props 를 통하여 전달받을 수 있게 됩니다.

src/App.js

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

const App = () => {
  return (
    <div>
      <ul>
        <li>
          <Link to="/">홈</Link>
        </li>
        <li>
          <Link to="/about">소개</Link>
        </li>
      </ul>
      <hr />
      <Route path="/" exact={true} component={Home} />
      <Route path="/about" component={About} />
      <Route path="/profiles/:username" component={Profile} />
    </div>
  );
};

export default App;
```

`/profiles/velopert` 경로로 직접 입력하여 들어가보세요. 우리가 사전에 입력한 데이터들이 잘 나타나나요?



`/profiles/gildong` 경로도 확인해보시길 바랍니다.

Query

이번엔 About 페이지에서 쿼리를 받아오겠습니다. 쿼리는 라우트 컴포넌트에게 props 전달되는 location 객체에 있는 search 값에서 읽어올 수 있습니다. location 객체는 현재 앱이 갖고있는 주소에 대한 정보를 지니고있습니다.

이런식으로 말이죠:

```
{
  key: 'ac3df4', // not with HashHistory!
  pathname: '/somewhere'
  search: '?some=search-string',
  hash: '#howdy',
  state: {
    [userDefined]: true
  }
}
```

여기서 search 값을 확인해야하는데, 이 값은 문자열 형태로 되어있습니다. 객체 형태로 변환하는건 우리가 직접 해주어야 합니다.

이 작업은 qs 라는 라이브러리를 사용하여 쉽게 할 수 있습니다.

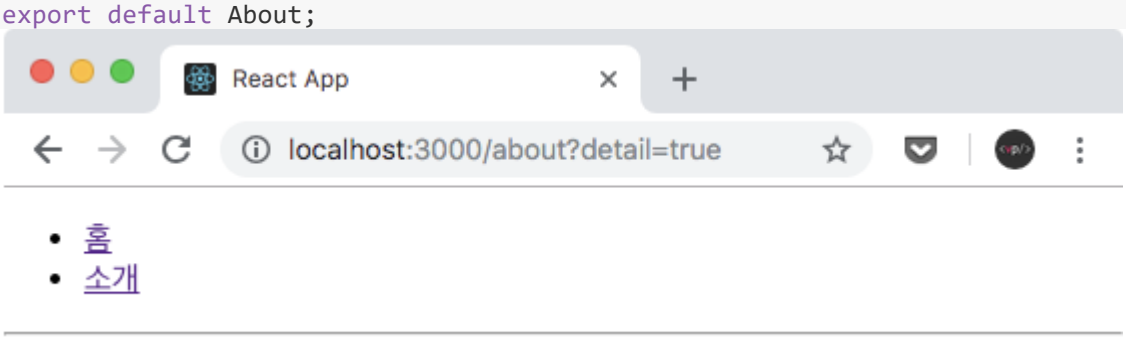
이 라이브러리를 설치해주세요:

```
$ yarn add qs
```

이제, About 컴포넌트에서 search 값에있는 detail 값을 받아와서, 해당 값이 true 일때 추가정보를 보여주도록 구현을 해보겠습니다.

src/About.js

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

const About = ({ location }) => {
  const query = qs.parse(location.search, {
    ignoreQueryPrefix: true
  });
  const detail = query.detail === 'true'; // 쿼리의 파싱결과값은 문자열입니다.

  return (
    <div>
      <h1>소개</h1>
      <p>이 프로젝트는 리액트 라우터 기초를 실습해보는 예제 프로젝트랍니다.</p>
      {detail && <p>추가적인 정보가 어쩌고 저쩌고..</p>}
    </div>
  );
};

export default About;
```



`/about?detail=true` 경로에 한번 들어가보세요. 추가정보가 잘 나타나나요?

3. 서브라우트

서브 라우트는, 라우트 내부의 라우트를 만드는것을 의미합니다. 이 작업은 그렇게 복잡하지 않습니다. 그냥 컴포넌트를 만들어서 그 안에 또 Route 컴포넌트를 렌더링하시면 됩니다.

서브 라우트 만들어보기

한번 Profiles 라는 컴포넌트를 만들어서, 그 안에 각 유저들의 프로필 링크들과 프로필 라우트를 함께 렌더링해보겠습니다.

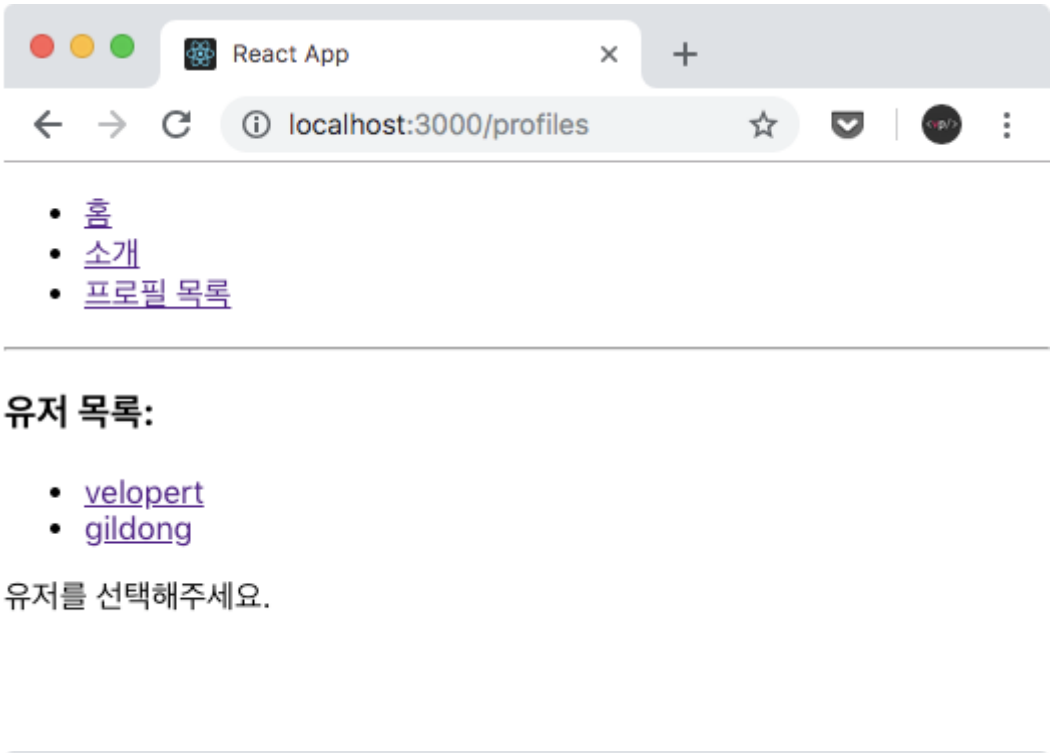
src/Profiles.js

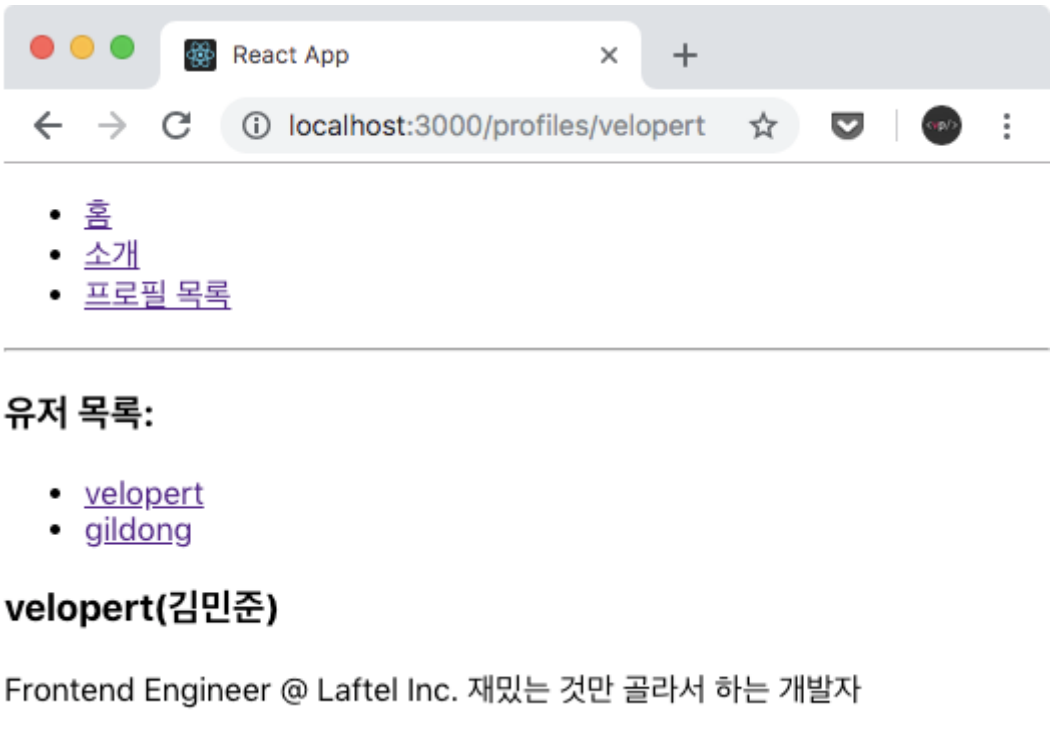
```
import React from 'react';
import { Link, Route } from 'react-router-dom';
import Profile from './Profile';

const Profiles = () => {
  return (
    <div>
      <h3>유저 목록:</h3>
      <ul>
        <li>
          <Link to="/profiles/velopert">velopert</Link>
        </li>
        <li>
          <Link to="/profiles/gildong">gildong</Link>
        </li>
      </ul>

      <Route
        path="/profiles"
        exact
        render={() => <div>유저를 선택해주세요.</div>}
      />
      <Route path="/profiles/:username" component={Profile} />
    </div>
  );
};

export default Profiles;
```

위 코드에서 첫번째 Route 컴포넌트에서는 `component` 대신에 `render` 가 사용되었는데요, 여기서는 컴포넌트가 아니라, JSX 자체를 렌더링 할 수 있습니다. JSX 를 렌더링하는 것이기에, 상위 영역에서 props 나 기타 값들을 필요하면 전달 해 줄 수있습니다.
그 다음, App 에서 Profiles 를 위한 링크와 라우트를 생성해주겠습니다 (기존 Profiles 라우트는 제거합니다.)

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

```
import Home from './Home';
import Profiles from './Profiles';

const App = () => {
  return (
    <div>
      <ul>
        <li>
          <Link to="/">홈</Link>
        </li>
        <li>
          <Link to="/about">소개</Link>
        </li>
        <li>
          <Link to="/profiles">프로필 목록</Link>
        </li>
      </ul>
      <hr />
      <Route path="/" exact={true} component={Home} />
      <Route path="/about" component={About} />
      <Route path="/profiles" component={Profiles} />
    </div>
  );
};

export default App;
```





서브라우트가 잘 구현 됐나요?

만약에 여러분이 만들게되는 서비스에서 특정 라우트 내에 탭 같은것을 만들게 된다면, 단순히 state 로 관리하는 것 보다 서브 라우트로 관리를 하는 것을 권장드립니다. 그 이유는, setState 같은것을 할 필요도 없고, 링크를 통하여 다른 곳에서 쉽게 진입 시킬 수도 있고, 나중에 검색엔진 크롤링 까지

고려한다면, 검색엔진 봇이 더욱 다양한 데이터를 수집해 갈 수 있기 때문입니다.

4. 리액트 라우터 부가기능

이번엔 알아두면 유용한 리액트 라우터의 부가기능들을 알아보겠습니다.

history 객체

history 객체는 라우트로 사용된 컴포넌트에게 match, location 과 함께 전달되는 props 중 하나입니다. 이 객체를 통하여, 우리가 컴포넌트 내에 구현하는 메소드에서 라우터에 직접 접근을 할 수 있습니다 - 뒤로가기, 특정 경로로 이동, 이탈 방지 등..

한번, 이 객체를 사용하는 예제 페이지를 작성해보겠습니다.

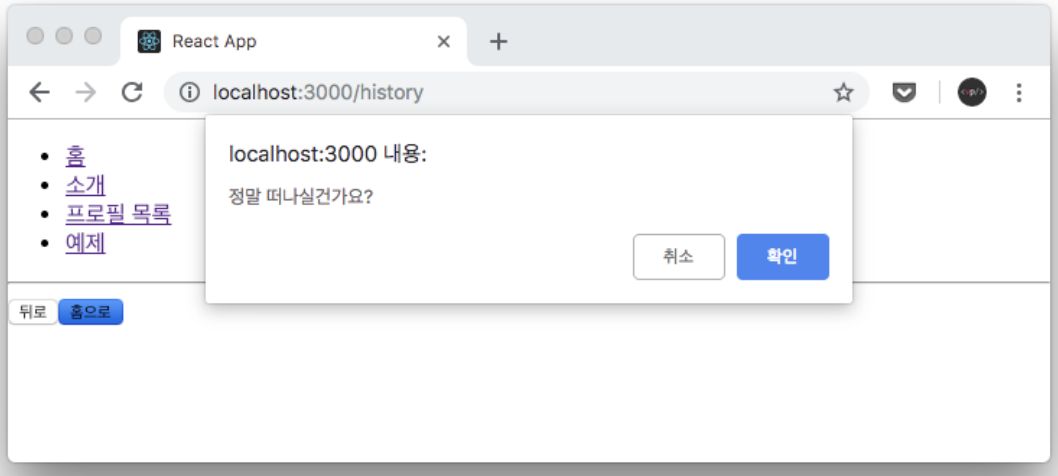
src/HistorySample.js

```
import React, { useEffect } from 'react';

function HistorySample({ history }) {
  const goBack = () => {
    history.goBack();
  };

  const goHome = () => {
    history.push('/');
  };

  useEffect(() => {
    console.log(history);
    const unblock = history.block('정말 떠나실건가요?');
    return () => {
      unblock();
    };
  }, [history]);

  return (
    <div>
      <button onClick={goBack}>뒤로가기</button>
      <button onClick={goHome}>홈으로</button>
    </div>
  );
}

export default HistorySample;
```

다음, App 에서 렌더링하고 버튼들을 눌러보세요.

src/App.js

```
import React from 'react';
import { Route, Link } from 'react-router-dom';
import About from './About';
import Home from './Home';
import Profiles from './Profiles';
import HistorySample from './HistorySample';

const App = () => {
  return (
    <div>
      <ul>
        <li>
          <Link to="/">홈</Link>
        </li>
        <li>
          <Link to="/about">소개</Link>
        </li>
        <li>
          <Link to="/profiles">프로필 목록</Link>
        </li>
        <li>
          <Link to="/history">예제</Link>
        </li>
      </ul>
      <hr />
      <Route path="/" exact={true} component={Home} />
      <Route path="/about" component={About} />
      <Route path="/profiles" component={Profiles} />
      <Route path="/history" component={HistorySample} />
    </div>
  );
};

export default App;
```



이렇게 history 객체를 사용하면 조건부로 다른 곳으로 이동도 가능하고, 이탈을 메시지박스를 통하여 막을 수 도 있습니다.

withRouter HoC

withRouter HoC 는 라우트 컴포넌트가 아닌곳에서 match / location / history 를 사용해야 할 때 쓰면 됩니다.

src/WithRouterSample.js

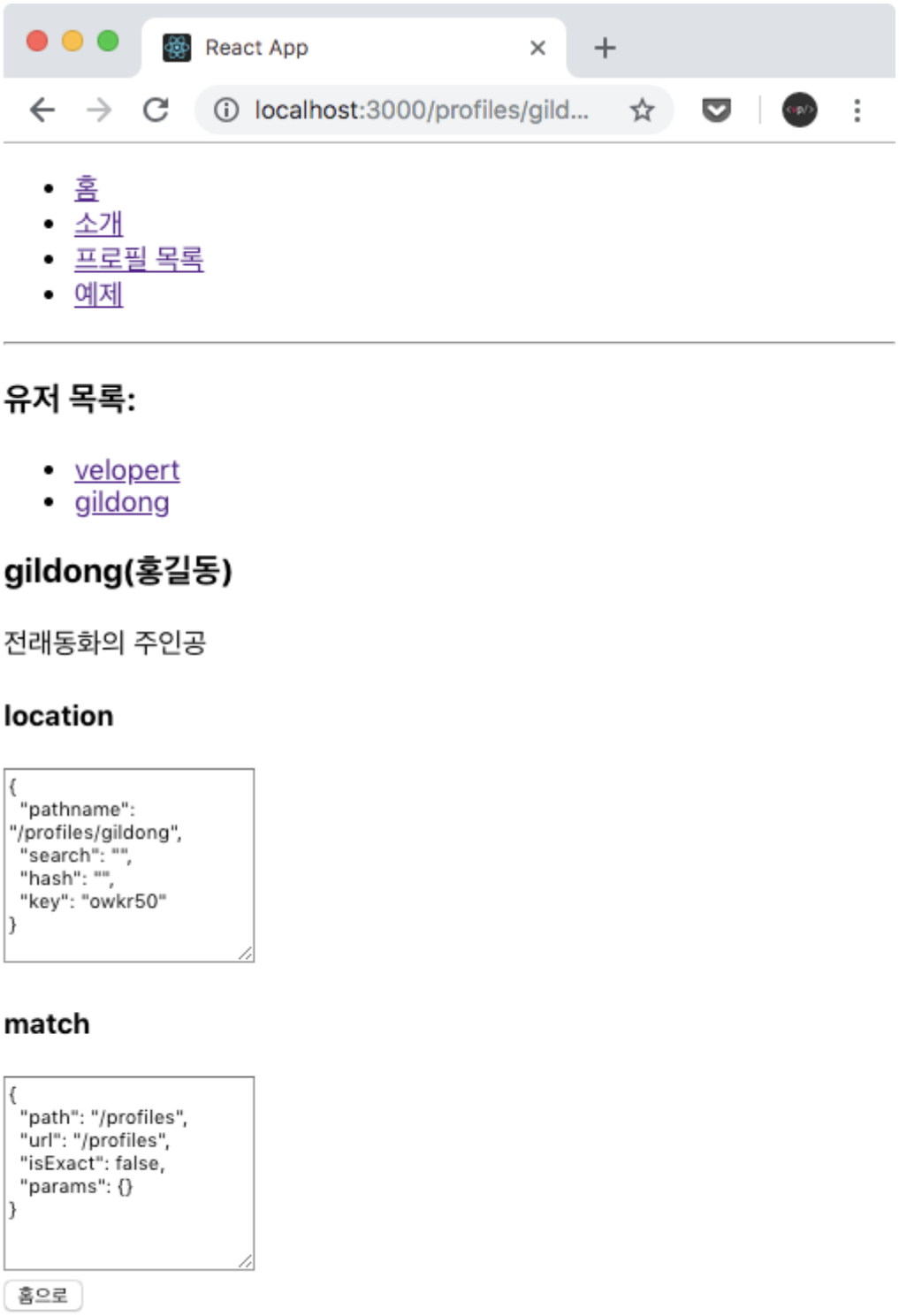
```
import React from 'react';
import { withRouter } from 'react-router-dom';
const WithRouterSample = ({ location, match, history }) => {
  return (
    <div>
      <h4>location</h4>
      <textarea value={JSON.stringify(location, null, 2)} readOnly />
      <h4>match</h4>
      <textarea value={JSON.stringify(match, null, 2)} readOnly />
      <button onClick={() => history.push('/')}>홈으로</button>
    </div>
  );
};

export default withRouter(WithRouterSample);
```

이제 이걸 Profiles.js 에서 렌더링해보겠습니다.

src/Profiles.js

```
import React from 'react';
import { Link, Route } from 'react-router-dom';
import Profile from './Profile';
import WithRouterSample from './WithRouterSample';

const Profiles = () => {
  return (
    <div>
      <h3>유저 목록:</h3>
      <ul>
        <li>
          <Link to="/profiles/velopert">velopert</Link>
        </li>
        <li>
          <Link to="/profiles/gildong">gildong</Link>
        </li>
      </ul>

      <Route
        path="/profiles"
        exact
        render={() => <div>유저를 선택해주세요.</div>}
```

```
      />
      <Route path="/profiles/:username" component={Profile} />
      <WithRouterSample />
    </div>
  );
};

export default Profiles;
```



withRouter 를 사용하면, 자신의 부모 컴포넌트 기준의 match 값이 전달됩니다. 보시면, 현재 gildong 이라는 URL Params 가 있는 상황임에도 불구하고 params 쪽이 `{}` 이렇게 비어있죠? WithRouterSample 은 Profiles 에서 렌더링 되었고, 해당 컴포넌트는 `/profile` 규칙에 일치하기 때문에 이러한 결과가 나타났습니다.

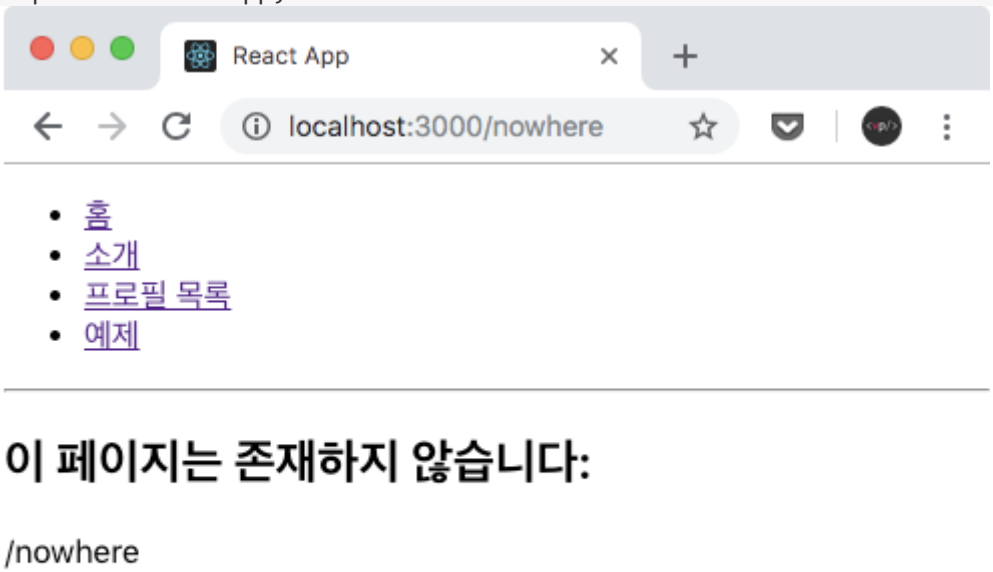
Switch

Switch 는 여러 Route 들을 감싸서 그 중 규칙이 일치하는 라우트 단 하나만을 렌더링시켜줍니다. Switch 를 사용하면, 아무것도 일치하지 않았을때 보여줄 Not Found 페이지를 구현 할 수도 있습니다.

한번 App.js 를 다음과 같이 수정해보세요:

```
import React from 'react';
import { Switch, Route, Link } from 'react-router-dom';
import About from './About';
import Home from './Home';
import Profiles from './Profiles';
import HistorySample from './HistorySample';

const App = () => {
  return (
    <div>
      <ul>
        <li>
          <Link to="/">홈</Link>
        </li>
        <li>
          <Link to="/about">소개</Link>
        </li>
        <li>
          <Link to="/profiles">프로필 목록</Link>
        </li>
        <li>
          <Link to="/history">예제</Link>
        </li>
      </ul>
      <hr />
      <Switch>
        <Route path="/" exact={true} component={Home} />
        <Route path="/about" component={About} />
        <Route path="/profiles" component={Profiles} />
        <Route path="/history" component={HistorySample} />
        <Route
          // path 를 따로 정의하지 않으면 모든 상황에 렌더링됨
          render={({ location }) => (
            <div>
              <h2>이 페이지는 존재하지 않습니다:</h2>
              <p>{location.pathname}</p>
            </div>
          )}
        />
      </Switch>
    </div>
  );
};
```

```
export default App;
```



NavLink

NavLink 는 Link 랑 비슷한데, 만약 현재 경로와 Link 에서 사용하는 경로가 일치하는 경우 특정 스타일 혹은 클래스를 적용 할 수 있는 컴포넌트입니다.

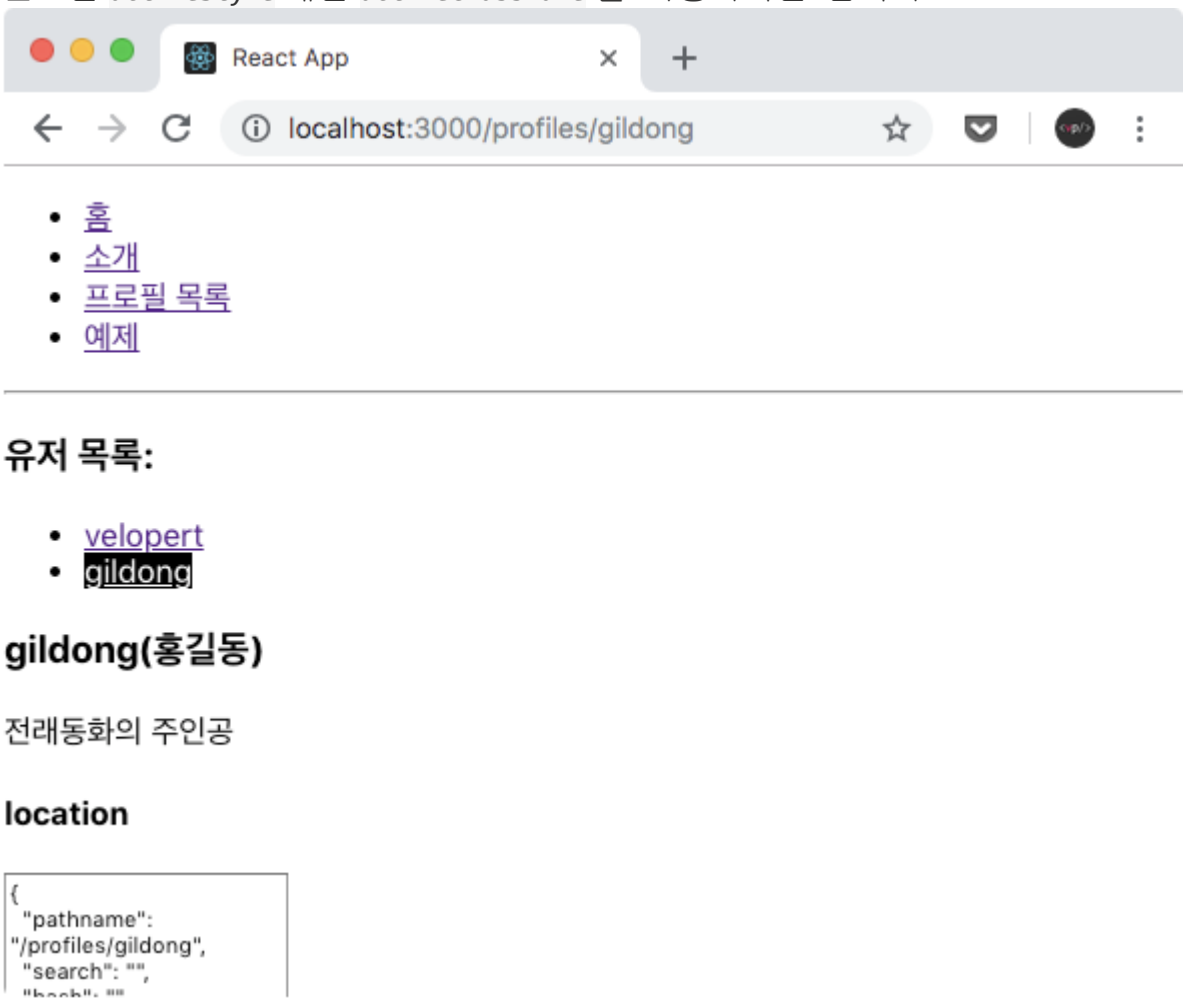
한번 Profiles 에서 사용하는 컴포넌트에서 Link 대신 NavLink 를 사용해보겠습니다.

src/Profiles.js

```
import React from 'react';
import { Route, NavLink } from 'react-router-dom';
import Profile from './Profile';
import WithRouterSample from './WithRouterSample';

const Profiles = () => {
  return (
    <div>
      <h3>유저 목록:</h3>
      <ul>
        <li>
          <NavLink
            to="/profiles/velopert"
            activeStyle={{ background: 'black', color: 'white' }}
          >
            velopert
          </NavLink>
        </li>
        <li>
```

```
          <NavLink
            to="/profiles/gildong"
            activeStyle={{ background: 'black', color: 'white' }}
          >
            gildong
          </NavLink>
        </li>
      </ul>

      <Route
        path="/profiles"
        exact
        render={() => <div>유저를 선택해주세요.</div>}
      />
      <Route path="/profiles/:username" component={Profile} />
      <WithRouterSample />
    </div>
  );
};

export default Profiles;
```

만약에 스타일이 아니라 CSS 클래스를 적용하시고 싶으면 `activeStyle` 대신 `activeClassName` 을 사용하시면 됩니다.



기타

지금까지 다룬 것들을 알아두시면 앞으로 리액트 라우터를 사용 할 때 구현하고싶은 것들을 충분히 만들 수 있을 것입니다.

이 외에도 다른 기능들이 있는데요..

- **Redirect**: 페이지를 리디렉트 하는 컴포넌트
- **Prompt**: 이전에 사용했던 history.block 의 컴포넌트 버전
- **Route Config**: JSX 형태로 라우트를 선언하는 것이 아닌 Angular 나 Vue 처럼 배열/객체를 사용하여 라우트 정의하기
- **Memory Router** 실제로 주소는 존재하지는 않는 라우터. 리액트 네이티브나, 임베디드 웹앱에서 사용하면 유용하다.

그 외의 것들은 공식 매뉴얼 을 참고하시길 바랍니다.

5. useReactRouter Hook 사용하기

지난 섹션에서 `withRouter` 라는 함수를 사용해서 라우트로 사용되고 있지 않은 컴포넌트에서도 라우트 관련 props 인 `match`, `history`, `location` 을 조회하는 방법을 확인해보았습니다.
`withRouter` 를 사용하는 대신에 Hook 을 사용해서 구현을 할 수도 있는데요, 아직은 리액트 라우터에서 공식적인 Hook 지원은 되고 있지 않습니다 (될 예정이긴 합니다).
그 전까지는, 다른 라이브러리를 설치해서 Hook 을 사용하여 구현을 할 수 있습니다. 이번 튜토리얼에서는 라이브러리를 설치해서 라우터에 관련된 값들을 Hook 으로 사용하는 방법을 알아보도록 하겠습니다.

우리가 사용 할 라이브러리의 이름은 use-react-router 입니다. yarn 을 사용하여 설치해주세요.

```
$ yarn add use-react-router
```

그 다음에, RouterHookSample.js 라는 파일을 생성 후 다음 코드를 작성해보세요.

RouterHookSample.js

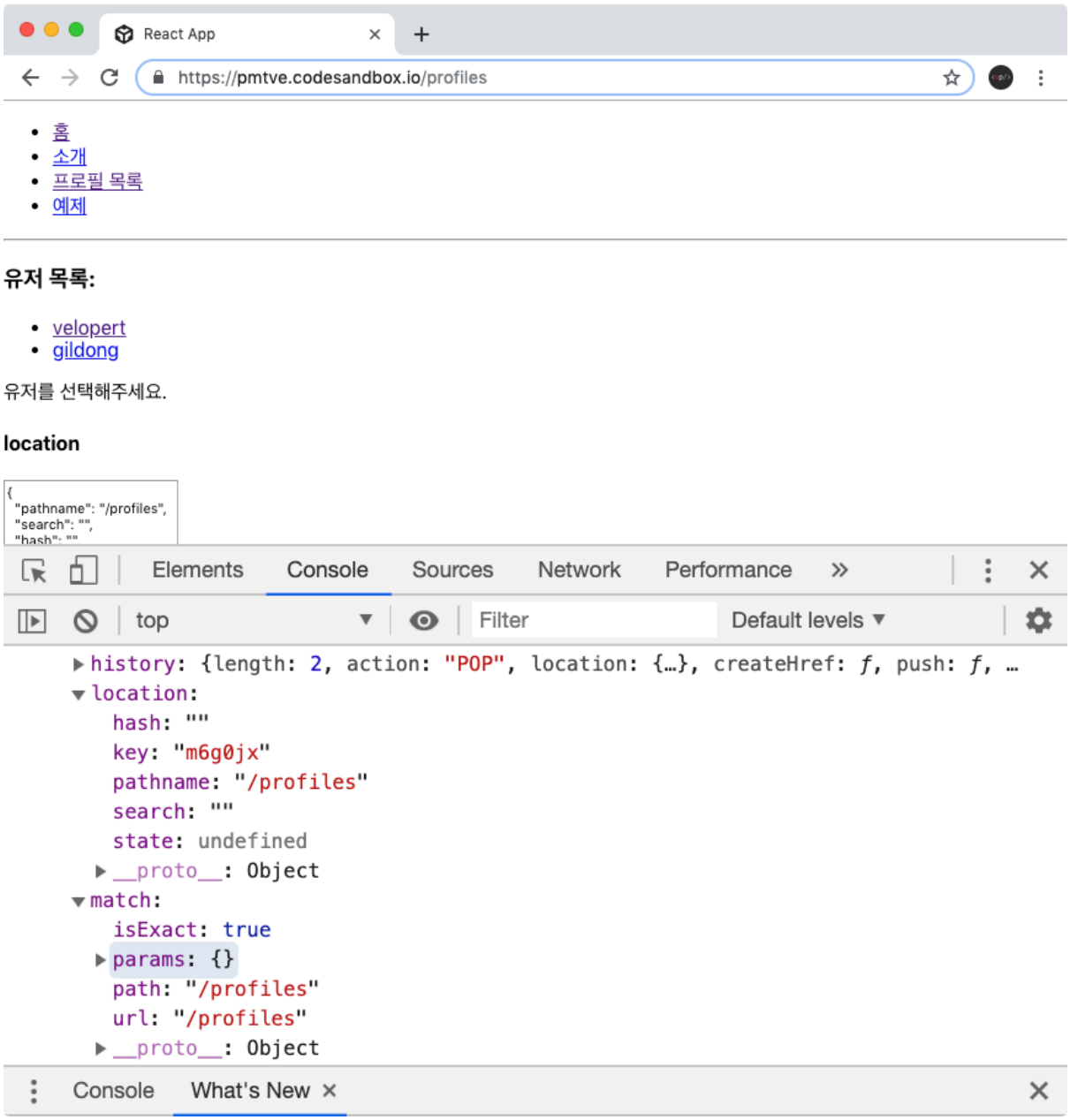
```
import useReactRouter from 'use-react-router';

function RouterHookSample() {
  const { history, location, match } = useReactRouter;
  console.log({ history, location, match });
  return null;
}
```

```
export default RouterHookSample;
```

이제 이 컴포넌트를 Profiles 컴포넌트에서 WithRouterSample 하단에 렌더링해보세요.

Profiles.js

```
import React from 'react';
import { Route, NavLink } from 'react-router-dom';
import Profile from './Profile';
import WithRouterSample from './WithRouterSample';
import RouterHookSample from './RouterHookSample';

const Profiles = () => {
  return (
    <div>
      <h3>유저 목록:</h3>
      <ul>
        <li>
          <NavLink
            to="/profiles/velopert"
            activeStyle={{ background: 'black', color: 'white' }}
          >
            velopert
          </NavLink>
        </li>
        <li>
          <NavLink
            to="/profiles/gildong"
            activeStyle={{ background: 'black', color: 'white' }}
          >
            gildong
          </NavLink>
        </li>
      </ul>

      <Route
        path="/profiles"
        exact
        render={() => <div>유저를 선택해주세요.</div>}
      />
      <Route path="/profiles/:username" component={Profile} />
      <WithRouterSample />
      <RouterHookSample />
    </div>
  );
};

export default Profiles;
```



콘솔을 확인해보세요. 위와 같이 프로필 목록 페이지를 열었을 때 `location`, `match`, `history` 객체들을 조회 할 수 있게 되었나요?
이 Hook 이 정식 릴리즈는 아니기 때문에 만약에 여러분들이 `withRouter` 가 너무 불편하다고 느낄 경우에만 사용하시는 것을 권장드립니다.
사용 한다고 해서 나쁠 것은 없지만, 나중에 정식 릴리즈가 나오게 되면 해당 라이브러리를 제거하고 코드를 수정해야 하는 일이 발생 할 것입니다.
적어도, `withRouter` 를 사용하셨다면, 레거시 코드로 유지해도 큰 문제는 없죠.
물론 추후 `useReactRouter` 를 사용하는 코드도 방치해도 될 지도 모르지만, 불필요한 라이브러리의 코드가 프로젝트에 포함된다는점, 그리고 정식 릴리즈가 되는 순간부터 `useReactRouter` 의 유지보수가 더 이상 이루어지지 않을 것 이라는 점을 생각하면, 중요한 프로젝트라면 사용을 하지 않는 편이 좋을 수도 있습니다.