



영화 앱에 여러 기능 추가하기

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

- ❖ 가장 처음으로 만들 기능은 내비게이션 기능이다.
- ❖ 내비게이션 기능을 추가하면 어떤 효과를 기대할 수 있을까?
- ❖ 다음은 내비게이션 기능으로 Home, About 메뉴를 추가한 모습이다.



1 react-router-dom 설치하고 프로젝트 폴더 정리하기

- ❖ Home은 영화 앱 화면으로 이동시켜주고, About은 개발자 자기 소개 화면으로 이동시켜준다.
- ❖ 이때 '화면 이동'을 시켜주려면 '화면 이동을 시켜주는 장치'가 필요하다.
- ❖ 그게 라우터이다.
- ❖ 앞으로 라우터 라는 용어를 자주 사용할 건데, 라우터는 그냥 화면 이동을 시켜주는 장치라고 이해하면 된다.
- ❖ 라우터는 react-router-dom 패키지를 이용하면 쉽게 도입할 수 있다.

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ react-router-dom 설치하기

- 우선 **react-router-dom** 패키지를 설치해 보자.

>npm install react-router-dom@5.2.0

```
C:\movie_app_2021>npm install react-router-dom
added 3 packages, and audited 1411 packages in 2s

163 packages are looking for funding
  run `npm fund` for details

12 vulnerabilities (4 low, 8 moderate)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
C:\movie_app_2021>_
```

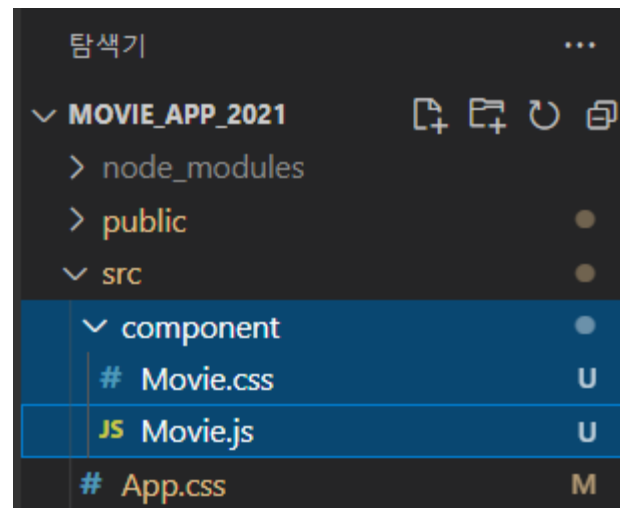
1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ components 폴더에 Movie 컴포넌트 옮기기

- 우선 react-router-dom 패키지를 설치해 보자.

➤ `npm install react-router-dom`

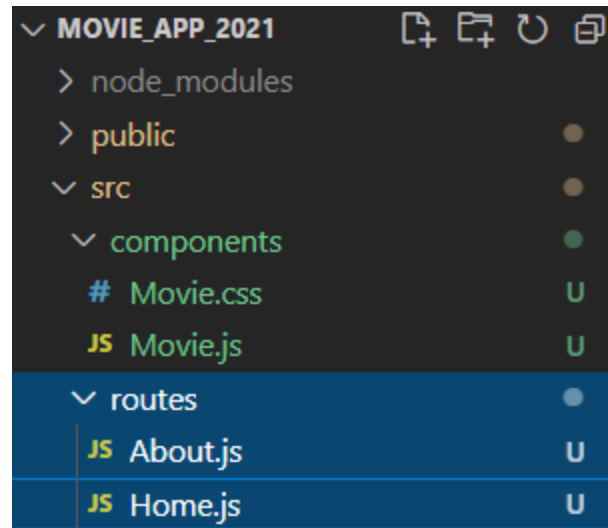
- 이제 컴포넌트를 역할에 맞게 분리해서 폴더에 담아 관리하자.
- 우선 **Movie** 컴포넌트부터 자리를 마련해 주자.
- **src** 폴더 안에 **components** 폴더를 만들자.
- 그리고 **Movie.js**와 **Movie.css**를 **components**에 옮기자.
- 만약 VSCode에서 무언가를 업데이트 하겠냐고 물어보는 창을 띄워주면 <No>를 누르면 된다.



1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ routes 폴더에 라우터가 보여줄 화면 만들기

- src 폴더 안에 routes 폴더를 생성한다.
- 여기에 라우터가 보여줄 화면(컴포넌트)을 만들 예정이다.
- 우리는 내비게이션에 Home, About 메뉴를 만들 거니까 Home.js, About.js 파일을 만들면 된다.
- Home.js 파일은 영화 앱 화면, About.js 파일은 개발자 소개 화면이다.
- Home.js 파일에 작성할 코드는 App.js 파일의 코드를 그대로 복사하면 되니까 Home.js 파일부터 수정해 보자.



1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ Home.js 수정하기

- App.js의 코드를 Home.js로 복사하자.
- 클래스 이름은 App에서 Home 으로 바꿔야 한다.
- Class Home, export default Home 모두 바꿔야 한다.
- Movie 컴포넌트를 임포트하고, 스타일링을 위한 Home.css도 임포트 하자.

1	import React from "react";
2	import axios from "axios";
3	import Movie from "../components/Movie";
4	import './Home.css';
5	
6	class Home extends React.Component{
	...
51	}
52	
53	export default Home;

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ Home.css 만들기

- **routes** 폴더에 **Home.css**를 만든 다음 아래와 같이 입력하자.
- 이 과정을 통해 영화 카드가 브라우저 폭에 맞게 1줄 또는 2줄로 출력 될 것이다.

1	.container{
2	height: 100%;
3	display: flex;
4	justify-content: center;
5	}
6	
7	.loader{
8	width: 100%;
9	height: 100vh;
10	display: flex;
11	justify-content: center;
12	align-items: center;
13	font-weight: 300;
14	}
15	

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ Home.css 만들기

- **routes** 폴더에 **Home.css**를 만든 다음 아래와 같이 입력하자.
- 이 과정을 통해 영화 카드가 브라우저 폭에 맞게 1줄 또는 2줄로 출력 될 것이다.

16	.movies{
17	display: grid;
18	grid-template-columns: repeat(2, minmax(400px, 1fr));
19	grid-gap: 100px;
20	padding: 50px;
21	width: 80%;
22	padding-top: 70px;
23	}
24	
25	@media screen and (max-width: 1090px){
26	.movies{
27	grid-template-columns: 1fr;
28	width: 100%;
29	}
30	}

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ App.js 수정하기

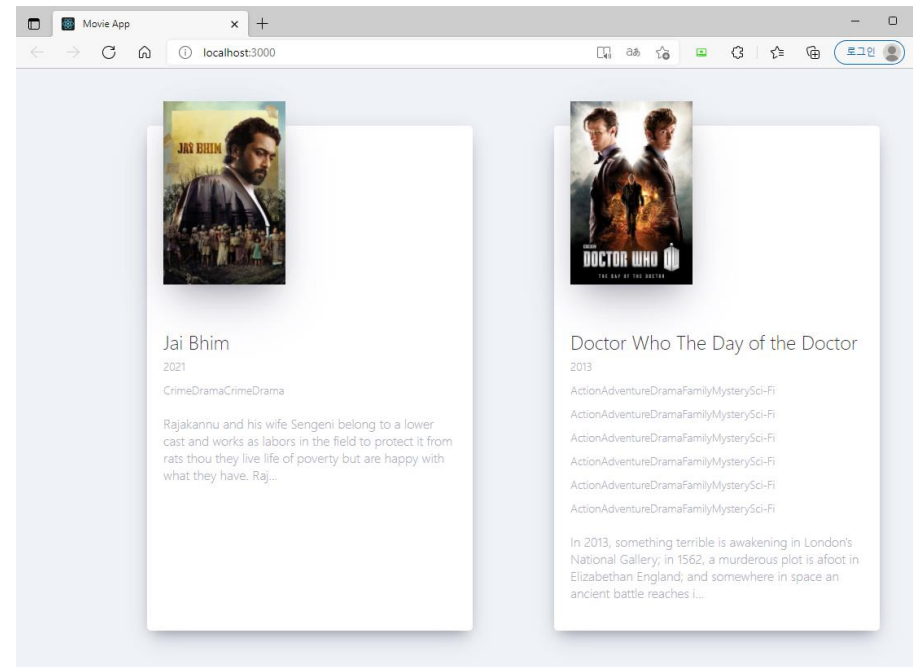
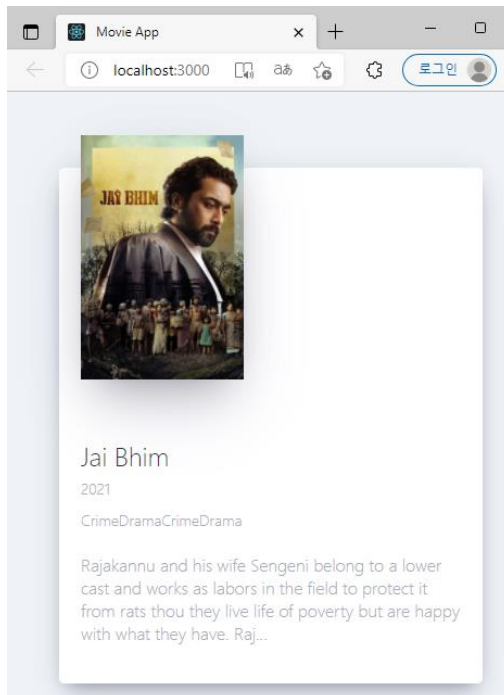
- 정말 그런지 확인해 보자.
- **App.js**를 다음과 같이 수정한 다음 영화 앱을 실행해 보자.

1	<code>import React from "react";</code>
2	<code>import Home from "../routes/Home";</code>
3	<code>import './App.css';</code>
4	
5	<code>function App(){</code>
6	<code> return <Home />;</code>
7	<code>}</code>
8	
9	<code>export default App;</code>

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ App.js 수정하기

- 코드를 저장하고 영화 앱을 실행하면, 브라우저 폭에 맞게 영화카드가 1줄 또는 2줄로 출력되고, 또 맨 위에 있는 영화 포스터 이미지가 잘 리지도 않게 바뀐 것을 확인할 수 있다.
- 이제 App.js가 2개의 라우터 (Home.js, About.js)를 보여줄 수 있도록 만들면 된다.



1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ 라우터 만들어 보기

- 이제 **App.js**를 수정해서 라우터를 만들어 보자.
- 라우터는 어떤 일을 할까?
- 라우터는 사용자가 입력한 **URL**을 통해 특정 컴포넌트를 불러준다.
- 예를 들어 사용자가 **localhost : 3000/home**이라고 입력하면 **Home** 컴포넌트를, **localhost : 3000/about**이라고 입력하면 **About** 컴포넌트를 불러주는 것!! 이게 라우터의 역할이다.
- **React-router-dom**은 여러 종류의 라우터를 제공하는데 우리는 **HashRouter**와 **Route** 컴포넌트를 사용할 것이다.

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ HashRouter와 Route 컴포넌트 사용하기

- HashRouter와 Route 컴포넌트를 임포트한 다음, HashRouter 컴포넌트가 Route 컴포넌트를 감싸 반환하도록 App.js를 수정해 보자.
- 그리고 컴포넌트를 임포트하는 코드와 사용한 코드는 잠시 지워두자.

1	import React from "react";
2	
3	import './App.css';
4	import {HashRouter, Route} from 'react-router-dom';
5	function App(){
6	return (
7	<HashRouter>
8	<Route/>
9	</HashRouter>
10);
11	}
12	
13	export default App;

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ HashRouter와 Route 컴포넌트 사용하기

- Route에는 2가지 props를 전달할 수 있는데 하나는 URL을 위한 path props이고, 하나는 URL에 맞는 컴포넌트를 불러 주기 위한 component props이다.
- path, component props를 통해 사용자가 접속한 URL을 보고, 그에 맞는 컴포넌트를 화면에 그릴 수 있게 되는 것이다.

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ Route 컴포넌트에 path, component props 추가하기

- About 컴포넌트를 임포트하고 path, component props에 URL과 About 컴포넌트를 전달하자.

1	import React from "react";
2	import './App.css';
3	import {HashRouter, Route} from 'react-router-dom';
4	import About from './routes/About';
5	
6	function App(){
7	return (
8	<HashRouter>
9	<Route path="/about" component={About} />
10	</HashRouter>
11);
12	}
13	
14	export default App;

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ About.js 수정하기

- 아직 **About.js**에는 아무것도 입력한 적이 없으니까 **About.js**도 간단히 작성해 보자.

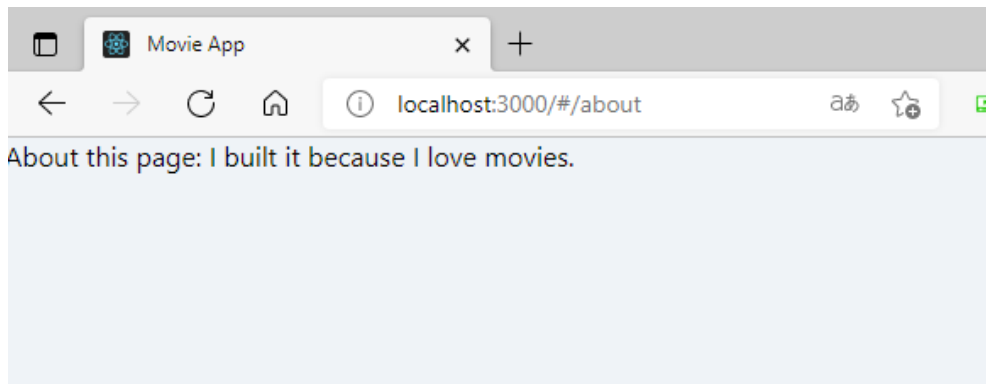
1	<code>import React from "react";</code>
2	
3	<code>function About(){</code>
4	<code> return About this page: I built it because I love movies.;</code>
5	<code>}</code>
6	
7	<code>export default About;</code>

- **HashRouter**에 **Route**를 넣었고, **Route**가 **About** 컴포넌트를 불러오도록 만들었다.
- 이제 우리의 라우터를 테스트해 보자.

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ 라우터 테스트해 보기

- **Localhost:3000/#**에 **path props**로 지정했던 값 **/about**을 붙여서 다시 접속해 보자.



- URL은 **localhost : 3000/#/about**이고, **About** 컴포넌트에 작성했던 내용이 나왔다!!!
- **Route** 컴포넌트에 전달한 **path props**를 보고 **component props**에 지정한 **About** 컴포넌트를 그려준 것이다.
- 이처럼 라우터는 엄청 쉬운 개념이다.
- 이제 **Home** 컴포넌트도 보여줄 수 있도록 **App.js**를 수정해 보자.

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ Home 컴포넌트를 위한 Route 컴포넌트 추가하기

- App 컴포넌트에 Home 컴포넌트를 임포트하고, 또 다른 Route 컴포넌트를 추가하자.

1	import React from "react";
2	import './App.css';
3	import {HashRouter, Route} from 'react-router-dom';
4	import About from './routes/About';
5	
6	function App(){
7	return (
8	<HashRouter>
9	<Route path="/" component={Home} />
10	<Route path="/about" component={About} />
11	</HashRouter>
12);
13	}
14	
15	export default App;

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

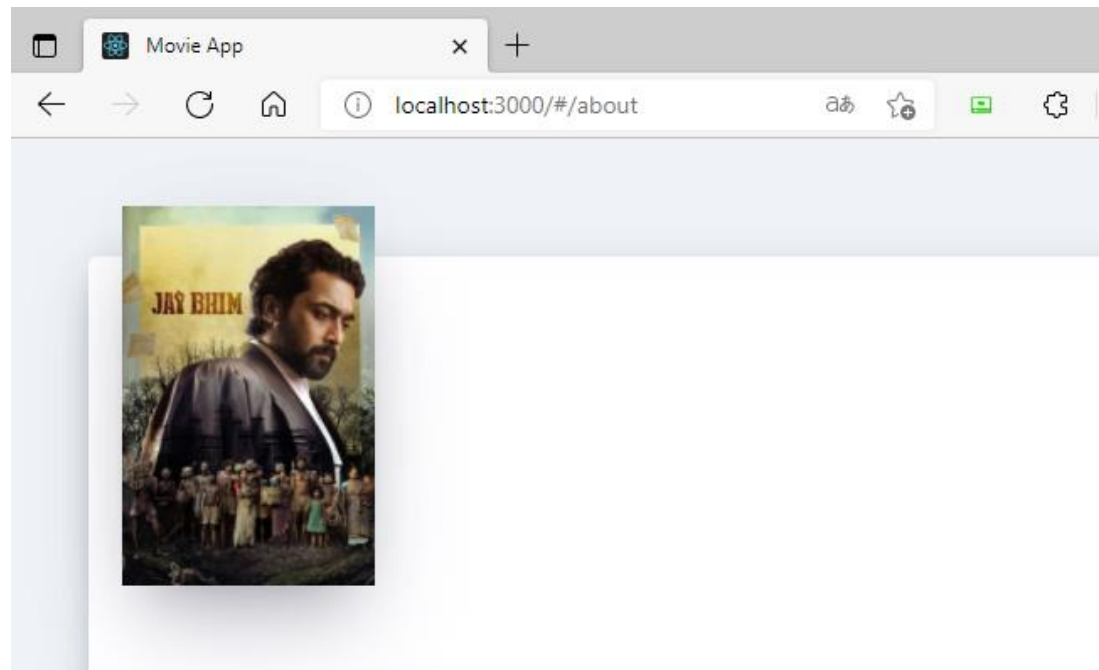
❖ Home 컴포넌트를 위한 Route 컴포넌트 추가하기

- `path props`를 `"/`으로 입력한 이유는 `localhost : 3000`에 접속하면 기본으로 보여줄 컴포넌트가 `Home` 컴포넌트라서 그렇다.

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ 라우터 테스트하고 문제 찾아보기

- 이제 다시 라우터를 테스트해 보자.
- `localhost : 3000`에 접속하면 주소 뒤에 자동으로 `/#/`가 붙으면서 영화 앱 화면이 나타날 것이다.
- 이어서 `/about`에도 접속해 보자.
- 그러면 이상하게도 **About** 컴포넌트와 함께 **Movie** 컴포넌트가 출력될 것이다.



1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ 라우터 테스트하고 문제 찾아보기

- `/about`에 접속하면 **About** 컴포넌트만 보여야 하는데 **Movie** 컴포넌트가 같이 보이는 문제가 있다.
- 리액트 라우터를 아직 제대로 활용하지 못하고 있기 때문에 그렇다.
- 리액트 라우터가 어떻게 동작하는지 자세히 알아보자.

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ 라우터 자세히 살펴보기

- 라우터의 동작을 자세히 알아보기 위해 **Home**, **About** 컴포넌트는 잠시 잊고, 아래와 같이 라우터를 구성하자.

	...
6	function App(){
7	return (
8	<HashRouter>
9	<Route path="/home" >
10	<h1>Home</h1>
11	</Route>
12	<Route path="/home/introduction">
13	<h1>Introduction</h1>
14	</Route>
15	<Route path="/about">
12	<h1>About</h1>
13	</Route>
14	</HashRouter>
15)

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

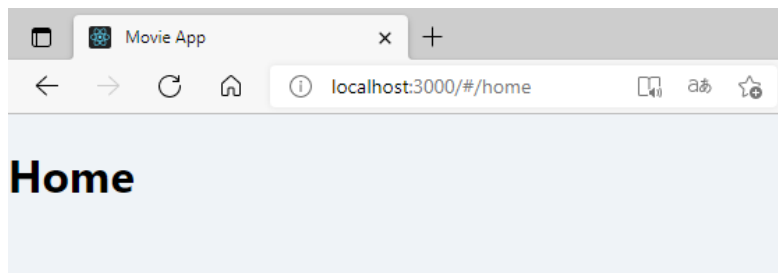
❖ 라우터 자세히 살펴보기

- 라우터가 `h1` 엘리먼트를 보여주고 있다.
- 여기서 주목해야 할 부분은 `path props`이다.
- 지금까지 공부한 대로면 아래의 상황이 예상될 것이다.
 - `localhost : 3000/#/home` → `<h1> Home </h1>` 출력
 - `localhost : 3000/#/home/introduction` → `<h1 >Introduction</h1 >` 출력
 - `localhost : 3000/#/about` → `<h1>About</h1>` 출력

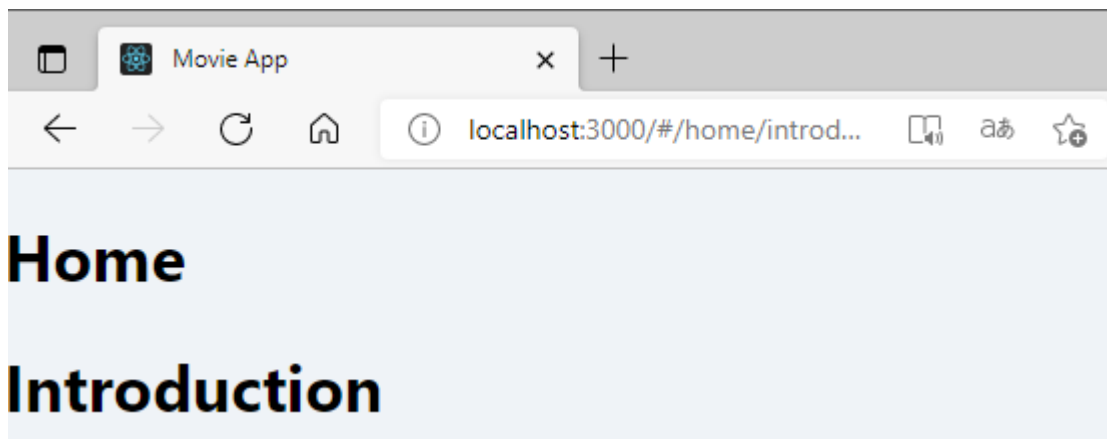
1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ 라우터 다시 테스트해 보기

- 정말 그럴까?
- 라우터를 테스트해 보자.
- 우선 `/home`에 접속해 보자.



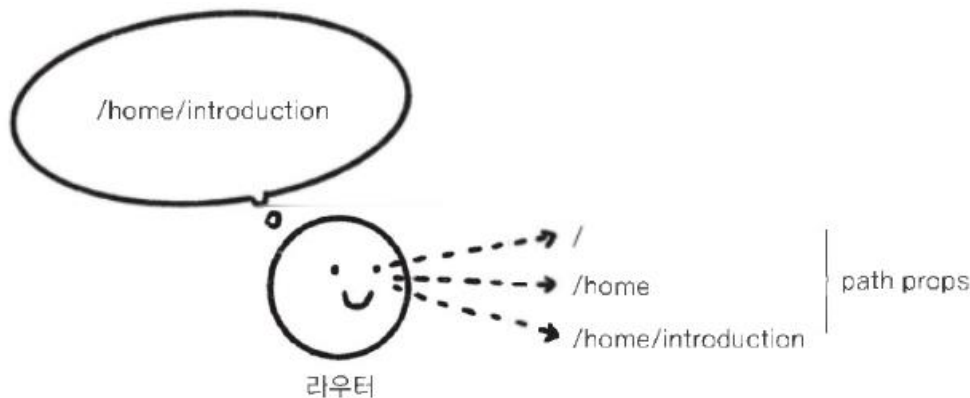
- 별 문제가 없어 보인다.
- `/home/introduction`에 접속하면 어떨까?



1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ 라우터 다시 테스트해 보기

- Home과 Introduction을 함께 출력하고 있다.
- 아까 본 문제와 같다.
- 이 문제가 발생한 이유는 라우터가 URL을 찾는 방식이 아래와 같기 때문이다.

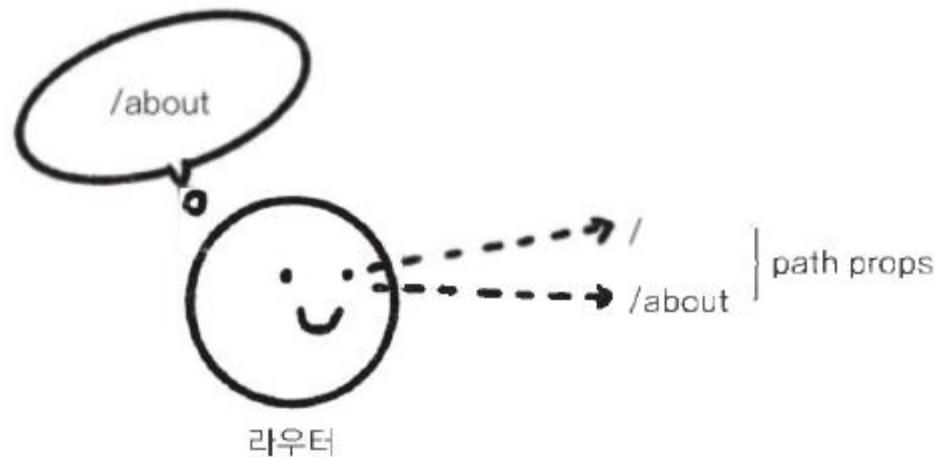


- 라우터는 사용자가 `/home/introduction`에 접속하면 `/`, `/home`, `/home/introduction` 순서로 `path props`가 있는지 찾는다.
- 그런데 `path props`에는 `/home`과 `/home/introduction`이 모두 있다.
- 그런 이유로 `/home/introduction`으로 접속한 경우 Home, Introduction 컴포넌트가 모두 그려지게 된다.

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ 라우터 다시 테스트해 보기

- 같은 원리로 사용자가 /about에 접속하면 /, /about 순서로 path props를 찾으므로 Home, About 컴포넌트가 모두 그려지게 된다.



1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ App.js 다시 원래대로 돌리기

- 이제 다시 **Home**, **About** 컴포넌트를 그려주었던 상태로 코드를 돌려 놓자.
- 그런 다음 **/about**에 다시 접속해 보자.

1	import React from "react";
2	import './App.css';
3	import {HashRouter, Route} from 'react-router-dom';
4	import About from './routes/About';
5	
6	function App(){
7	return (
8	<HashRouter>
9	<Route path="/" component={Home} />
10	<Route path="/about" component={About} />
11	</HashRouter>
12);
13	}
14	
15	export default App;

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ App.js 다시 원래대로 돌리기

- 그러면 여전히 **Home**, **About** 컴포넌트가 모두 그려진다.
- 이 현상을 고치려면 어떻게 해야 할까?
- 바로 **Route** 컴포넌트에 **exact props**를 추가하면 된다.
- **exact props**는 **Route** 컴포넌트가 **path props**와 정확하게 일치하는 URL에만 반응하도록 만들어 준다.
- 정말 그런지 확인해 보자.



David Attenborough: A Life on Our Planet
2020

ActionBiographyDocumentaryActionBiographyDocumentaryActionBiog

One man has seen more of the natural world than any other. His statement. In his 93 years, David Attenborough has visited ever

About this page: I built it because I love movies.

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ Route 컴포넌트에 exact props 추가해 보기

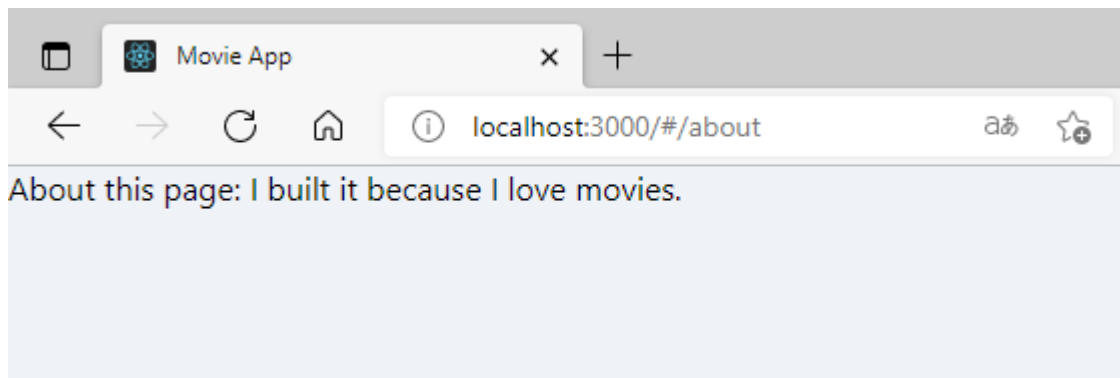
- path props가 "/"인 Route 컴포넌트에 `exact={true}`를 추가해 보자.

1	<code>import React from "react";</code>
2	<code>import './App.css';</code>
3	<code>import {HashRouter, Route} from 'react-router-dom';</code>
4	<code>import About from './routes/About';</code>
5	
6	<code>function App(){</code>
7	<code> return (</code>
8	<code> <HashRouter></code>
9	<code> <Route path="/" exact={true} component={Home} /></code>
10	<code> <Route path="/about" component={About} /></code>
11	<code> </HashRouter></code>
12	<code>);</code>
13	<code>}</code>
14	
15	<code>export default App;</code>

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ Route 컴포넌트에 `exact props` 추가해 보기

- 실행 결과를 보니 이제는 `/about`에 접속하면 `About` 컴포넌트만 출력된다.
- `path props`가 정확히 `/`인 경우에만 `Home` 컴포넌트만 출력되도록 설정되었기 때문이다.
- 이제 라우터를 제대로 활용할 수 있게 되었다.
- 마지막으로 `About` 컴포넌트의 모양을 다듬기 위해 스타일을 적용하자.



1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ About.css 작성하기

- **routes** 폴더에 **About.css** 파일을 생성한 다음 아래와 같이 입력하고 저장한다.

1	.about_container{
2	 box-shadow: 0 13px 27px -5px rgba(50, 50, 93, 0.25), 0 8px 16px -8px rgba(0, 0, 0, 0.3), 0 -6px 16px -6px rgba(0, 0, 0, 0.025);
3	 padding: 20px;
4	 border-radius: 5px;
5	 background-color: white;
6	 margin: 0 auto;
7	 margin-top: 100px;
8	 width: 100%;
9	 max-width: 400px;
10	 font-weight: 300;
11	}
12	
13	.about_container span:first-child{
14	 font-size: 20px;
15	}

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ About.css 작성하기

- **routes** 폴더에 **About.css** 파일을 생성한 다음 아래와 같이 입력하고 저장한다.

16	
17	<code>.about__container span:last-child{</code>
18	<code>display: block;</code>
19	<code>margin-top: 10px;</code>
20	<code>}</code>

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ About.css 작성하기

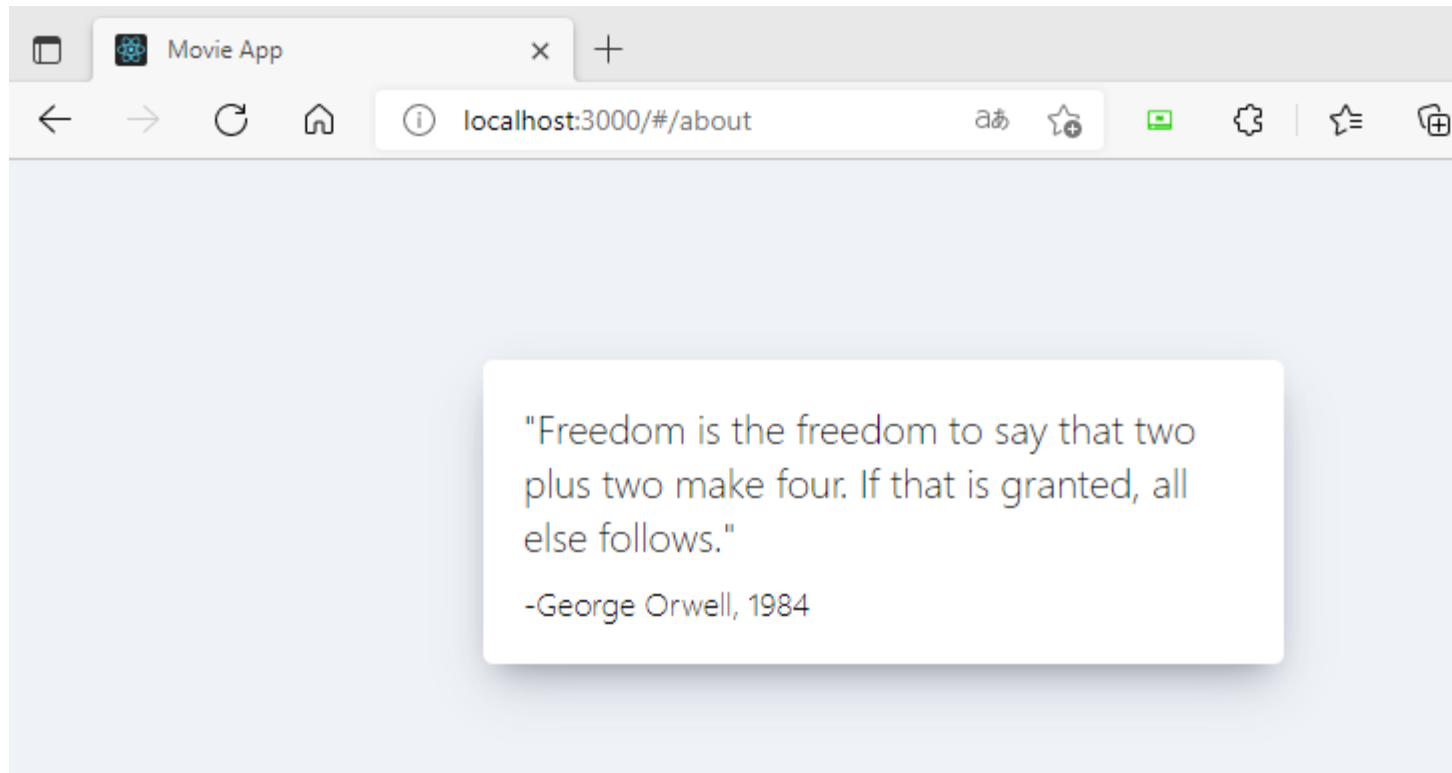
- 그런 다음 About.js에 About.css를 임포트하고 임포트한 About.css를 적용할 수 있도록 JSX를 수정하자.

1	import React from "react";
2	import './About.css';
3	
4	function About(){
5	return(
6	<div className="about__container">
7	
8	"Freedom is the freedom to say that two plus two make four. If that is granted, all else follows."
9	
10	-George Orwell, 1984
11	</div>
12);
13	}
14	
15	export default About;

1 react-router-dom 설치하고 프로젝트 폴더 정리하기

❖ About.css 작성하기

- 앱을 실행하여 /about으로 접속하면 위와 같은 화면이 나타날 것이다.
- 이제 내비게이션 메뉴를 만들면 된다.
- 내비게이션 메뉴를 만들어 보자.



3 내비게이션 만들어보기

- ❖ 이제 라우터가 준비되었으므로 내비게이션을 통해 다른 화면으로 이동하면 된다.
- ❖ 여기서는 라우터를 이용하여 간단한 내비게이션을 만들어 볼 것이다.
- ❖ <Home>과 <About>이라는 2개의 버튼을 만들고, 각각의 버튼을 눌렀을 때 적절한 화면을 보여 주도록 클론 코딩할 것이다.

3 내비게이션 만들어보기

❖ Navigation 컴포넌트 만들기

- **components** 폴더에 **Navigation.js** 파일을 만들고 2개의 **a** 엘리먼트를 반환하도록 **JSX** 를 작성하자.

1	<code>import React from "react";</code>
2	
3	<code>function Navigation(){</code>
4	<code> return(</code>
5	<code> <div></code>
6	<code> Home</code>
7	<code> About</code>
8	<code> </div></code>
9	<code>);</code>
10	<code>}</code>
12	
13	<code>export default Navigation;</code>

- 2개의 **a** 엘리먼트는 각각 **URL**을 **/**와 **/about**으로 이동시켜줄 것 같다.
- 하지만 아니다.
- **Navigation** 컴포넌트를 **App** 컴포넌트에 포함시키면 어떤 문제가 생길까?

3 내비게이션 만들어보기

❖ Navigation 컴포넌트 App 컴포넌트에 포함시키기

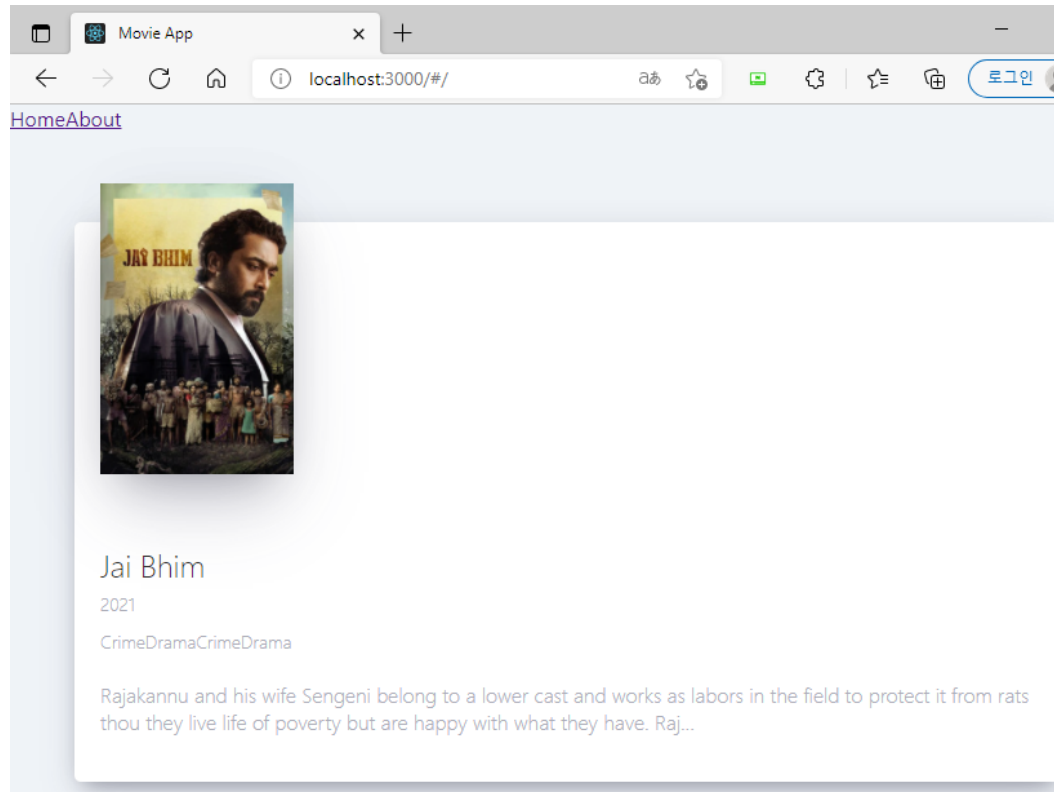
- App 컴포넌트에 Navigation 컴포넌트를 포함시켜보자.
- Navigation.js를 임포트하고, <HashRouter> </HashRouter> 사이에 포함시키면 된다.

1	import React from "react";
2	import './App.css';
3	import {HashRouter, Route} from 'react-router-dom';
4	import About from './routes/About';
5	import Navigation from './components/Navigation';
6	
7	function App(){
8	return (
9	<HashRouter>
10	<Navigation />
11	<Route path="/" exact={true} component={Home} />
12	<Route path="/about" component={About} />
13	</HashRouter>
14);
	...

3 네비게이션 만들어보기

❖ Navigation 컴포넌트 App 컴포넌트에 포함시키기

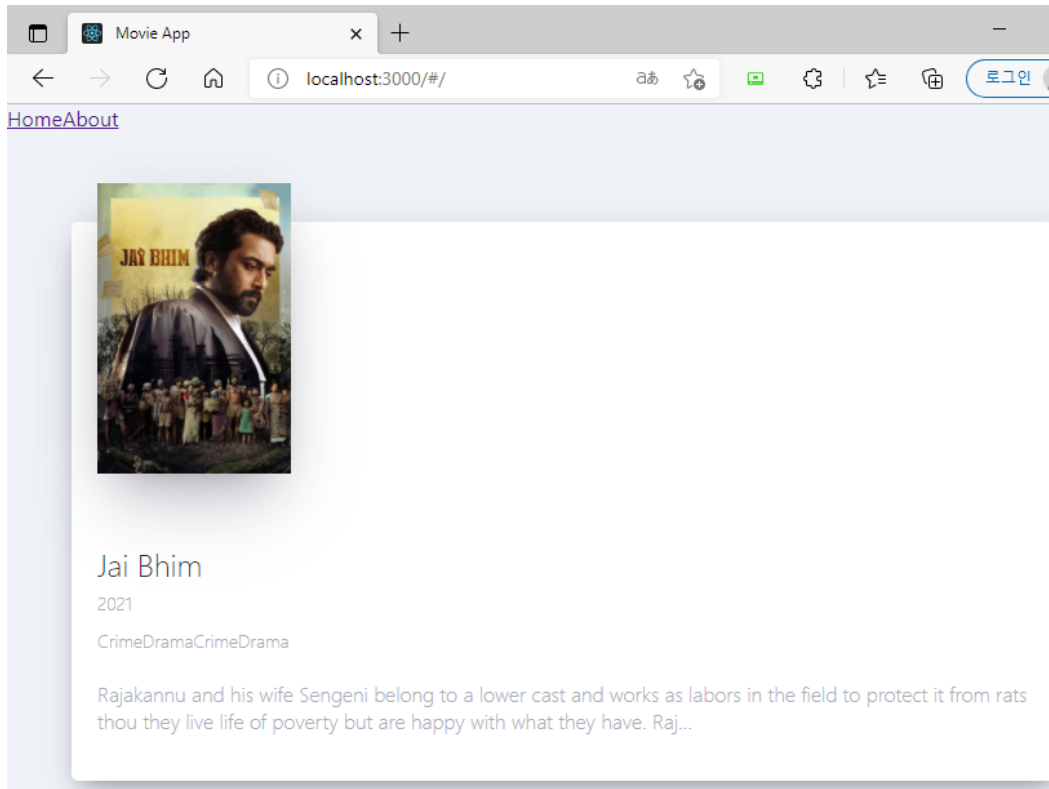
- 영화 앱을 실행하면 왼쪽 위에 Navigation 컴포넌트가 출력하는 Home, About 링크 (a 엘리먼트) 를 확인할 수 있다.
- 이걸 잘 동작할까?



3 네비게이션 만들어보기

❖ Home 링크 눌러 보기

- Home 링크를 눌러보면 겉으로 보기에는 잘 동작하는 것 같다.



3 내비게이션 만들어보기

❖ Home 링크 눌러 보기

- 하지만 현재는 링크를 누를 때마다 리액트가 죽고, 새 페이지가 열리는 문제가 있다(화면 전체가 새로 고침되는 문제).
- `a` 엘리먼트 특징 때문이다.
- `A` 엘리먼트의 `href` 속성은 페이지 전체를 다시 그리기 때문이다.
- 이 상태라면 필요한 부분만 다시 그려주는 리액트의 장점을 활용하기 힘들다.
- 이 문제를 해결하려면 어떻게 해야 할까?
- 바로 `react-router-dom`의 `Link` 컴포넌트를 사용하면 된다.

3 내비게이션 만들어보기

❖ a 엘리먼트 Link 컴포넌트로 바꾸기

- Navigation 컴포넌트에 Link 컴포넌트를 임포트한 다음 a 엘리먼트를 Link 컴포넌트로 바꾸자.
- 그리고 href 속성은 to로 바꾼다.
- 그런 다음 영화 앱을 다시 실행하고 Home과 About 링크를 한 번씩 눌러 보자.

1	import React from "react";
2	import {Link} from 'react-router-dom';
3	
4	function Navigation(){
5	return(
6	<div>
7	<Link to="/">Home</Link>
8	<Link to="/about">About</Link>
9	</div>
10);
12	}
13	
14	export default Navigation;

3 내비게이션 만들어보기

❖ a 엘리먼트 `Link` 컴포넌트로 바꾸기

- 자! 이제 페이지 전체가 다시 새로 고침되지 않는다.
- 내비게이션이 제대로 만들어졌다.
- 여기서 반드시 기억해야 할 내용은 `Link`, `Router` 컴포넌트는 반드시 `HashRouter` 안에 포함되어야 한다는 점이다.

3 네비게이션 만들어보기

❖ Navigation 컴포넌트 스타일링하기

- 마지막으로 네비게이션을 스타일링해 보자.
- **components** 폴더에 **Navigation.css** 파일을 만들고 아래와 같이 작성한 다음 **Navigation** 컴포넌트에 임포트시키자.
- 또 **Navigation.css** 파일을 **Navigation** 컴포넌트에 적용시키기 위해 **Navigation** 컴포넌트의 **JSX**를 수정하자.

1	<code>.nav{</code>
2	<code> z-index: 1;</code>
3	<code> position: fixed;</code>
4	<code> top: 50px;</code>
5	<code> left: 10px;</code>
6	<code> display: flex;</code>
7	<code> flex-direction: column;</code>
8	<code> background-color: white;</code>
9	<code> padding: 10px 20px;</code>
10	<code> box-shadow: 0 13px 27px -5px rgba(50, 50, 93, 0.25),</code> <code> 0 8px 16px -8px rgba(0, 0, 0, 0.3), 0 -6px 16px -6px rgba(0, 0, 0, 0.25);</code>
11	<code> border-radius: 5px;</code>
12	<code>}</code>

3 내비게이션 만들어보기

❖ Navigation 컴포넌트 스타일링하기

- 마지막으로 내비게이션을 스타일링해 보자.
- **components** 폴더에 **Navigation.css** 파일을 만들고 아래와 같이 작성한 다음 **Navigation** 컴포넌트에 임포트시키자.
- 또 **Navigation.css** 파일을 **Navigation** 컴포넌트에 적용시키기 위해 **Navigation** 컴포넌트의 **JSX**를 수정하자.

13	
14	@media screen and (max-width: 1090px){
15	.nav{
16	left: initial;
17	top: initial;
18	bottom: 0px;
19	width: 100%;
20	}
21	}
22	

3 네비게이션 만들어보기

❖ Navigation 컴포넌트 스타일링하기

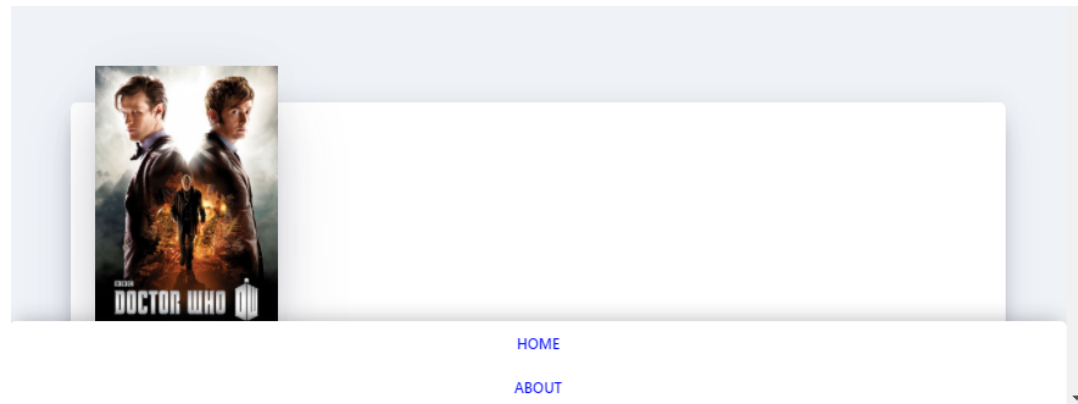
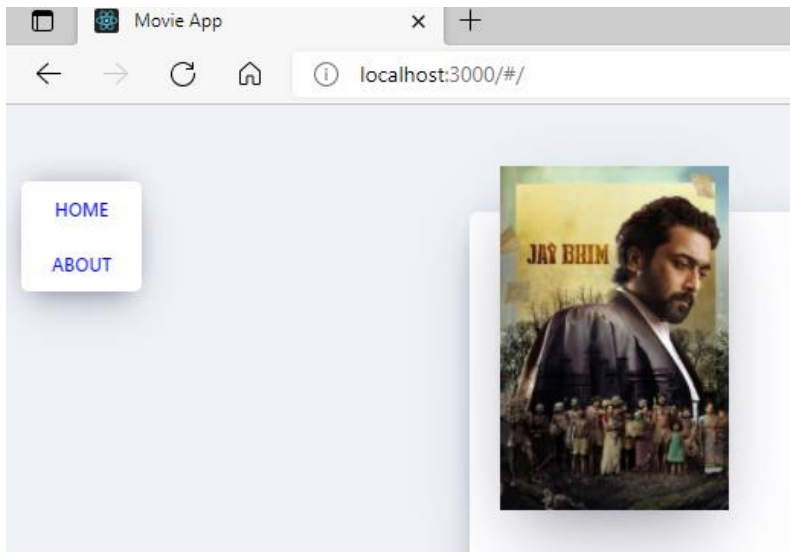
- 마지막으로 네비게이션을 스타일링해 보자.
- **components** 폴더에 **Navigation.css** 파일을 만들고 아래와 같이 작성한 다음 **Navigation** 컴포넌트에 임포트시키자.
- 또 **Navigation.css** 파일을 **Navigation** 컴포넌트에 적용시키기 위해 **Navigation** 컴포넌트의 **JSX**를 수정하자.

23	<code>.nav a{</code>
24	<code>text-decoration: none;</code>
25	<code>color: #0008fc;</code>
26	<code>text-transform: uppercase;</code>
27	<code>font-size: 12px;</code>
28	<code>text-align: center;</code>
29	<code>font-weight: 600;</code>
30	<code>}</code>
31	
32	<code>.nav a:not(:last-child){</code>
33	<code>margin-bottom: 20px;</code>
34	<code>}</code>

3 네비게이션 만들어보기

❖ Navigation 컴포넌트 스타일링하기

- 영화 앱을 실행하면 화면 폭이 넓어지면 왼쪽에, 화면 폭이 좁아지면 아래에 네비게이션이 위치할 것이다.



4 영화 상세정보 기능 만들어 보기

- ❖ Home에서 볼 수 있는 영화 정보는 아주 일부분이다.
- ❖ 영화 API를 통해 더 많은 정보를 받고 있으니까 이것 활용해 보자.
- ❖ 여기서는 영화 카드를 누르면 상세 정보를 보여 주는 기능을 만들 것이다.
- ❖ 그런데 여기서 잠깐!
- ❖ 이 기능을 만들기 위해서는 `route props`를 반드시 이해해야 한다.
- ❖ `Route props`란 라우팅 대상이 되는 컴포넌트에 넘겨주는 기본 `props`를 말한다.
- ❖ 다시 말하자면 우리가 직접 넘겨주지 않아도 기본으로 넘어가는 `route props`라는 것이 있고, 이것을 이용해야 영화 데이터를 상세 정보 컴포넌트에 전달할 수 있다.

4 영화 상세정보 기능 만들어 보기

❖ route props 살펴보기

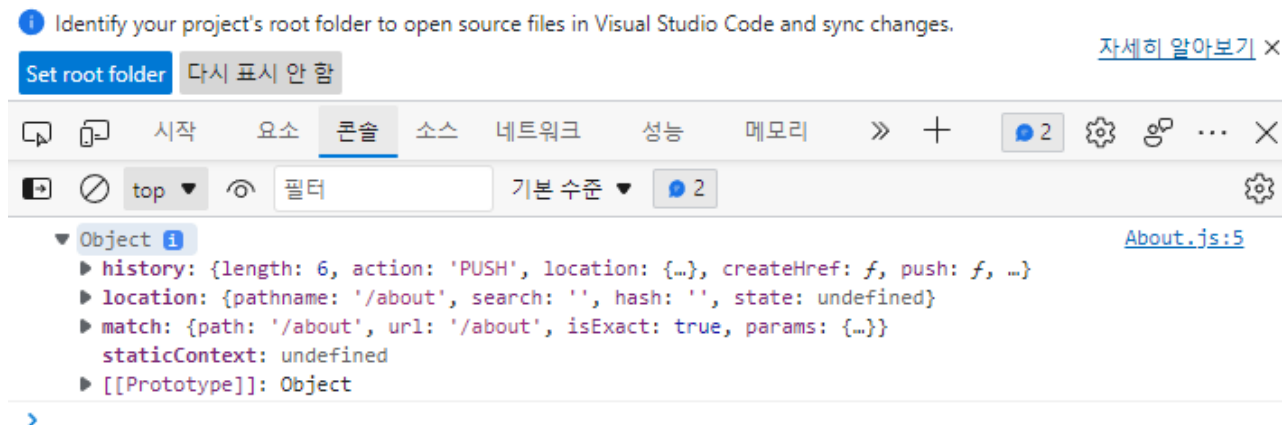
- 우선 `console.log()`를 통해 `About`으로 어떤 `props`가 넘어오는지 살펴보자.

```
1 import React from "react";
2 import './About.css';
3
4 function About(props){
5   console.log(props);
6   return(
7     <div className="about__container">
8       <span>
9         "Freedom is the freedom to say that two plus two make four. If that is
          granted, all else follows."
10      </span>
11      <span>-George Orwell, 1984</span>
12    </div>
13  );
14 }
15
16 export default About;
```


4 영화 상세정보 기능 만들어 보기

❖ route props 살펴보기

- About으로 이동하면 [콘솔] 탭에 `{history: {...}, Location: {...}, match: {...}, staticContext: undefined}`가 출력된다.
- 이게 바로 `react-router-dom`에서 `Route` 컴포넌트가 그려줄 컴포넌트에 전달한 props이다.
- 값이 좀 많아서 걱정될 수 있지만 이 값들을 다 알 필요는 없다.
- 우리가 주목해야 할 점은 `Route` 컴포넌트가 그려줄 컴포넌트에는 항상 이props가 전달되고, 이 props에 우리 마음대로 데이터를 담아 보내줄 수 있다는 사실이다.
- 그러면 `route props`에 데이터를 담아 보내보자.



4 영화 상세정보 기능 만들어 보기

❖ route props에 데이터 담아 보내기

- route props에 데이터를 담아 보내려면 Navigation 컴포넌트에 있는 Link 컴포넌트의 to props의 구조를 조금 바꿔야 한다.
- 아래와 같이 Navigation 컴포넌트 /about으로 보내주는 Link 컴포넌트의 to props를 수정해 보자.

1	import React from "react";
2	import {Link} from 'react-router-dom';
3	
4	function Navigation(){
5	return(
6	<div>
7	<Link to="/">Home</Link>
8	<Link to={{pathname: '/about', state: {fromNavigation: true}}} >About</Link>
9	</div>
10);
12	}
13	
14	export default Navigation;

4 영화 상세정보 기능 만들어 보기

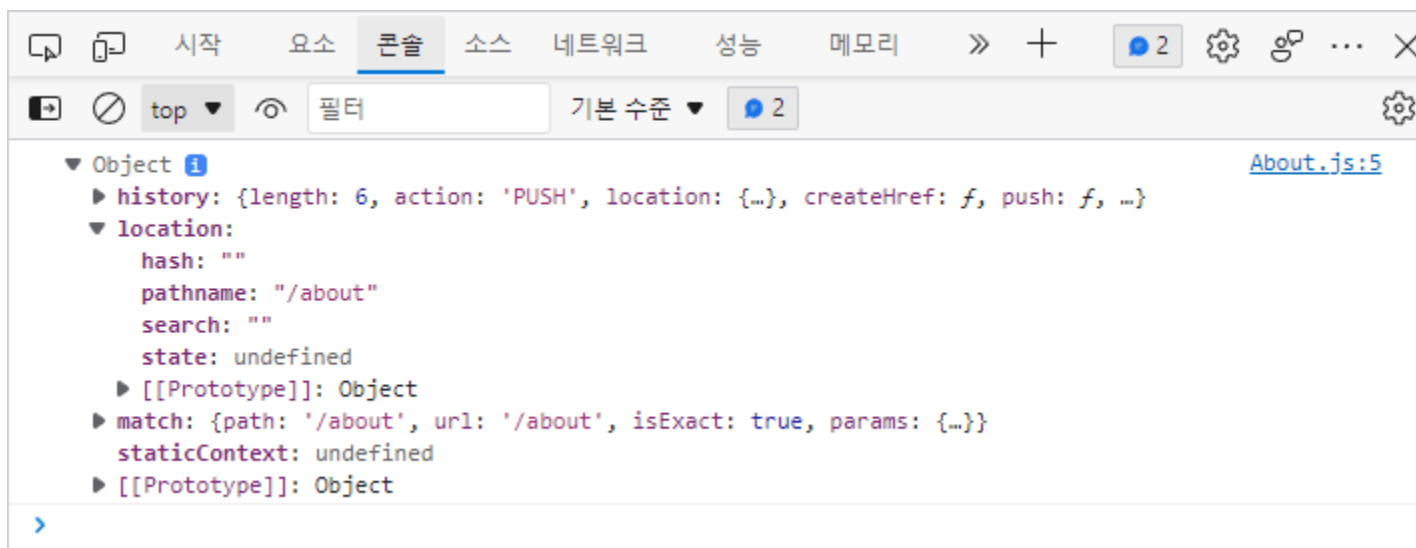
❖ `route props`에 데이터 담아 보내기

- 코드에서 보듯 `to props`에 객체를 전달했다.
- `pathname`은 URL을 의미하고, `state`는 우리가 `route props`에 보내 줄 데이터를 의미한다.
- `state`에 담아 보낸 객체가 정말 `About` 컴포넌트로 넘어갔을까?
- 확인해 보자.

4 영화 상세정보 기능 만들어 보기

❖ route props 다시 살펴보기

- /about으로 이동한 다음 [콘솔] 탭에서 [location]을 펼쳐보자.
- 그러면 **state** 키에 우리가 보내준 데이터를 확인할 수 있다.



4 영화 상세정보 기능 만들어 보기

❖ Navigation 컴포넌트 정리하기

- 앞에서 작성한 코드는 사용하지 않을 것이므로 Navigation 컴포넌트를 원래대로 돌려 놓자.

1	<code>import React from "react";</code>
2	<code>import {Link} from 'react-router-dom';</code>
3	
4	<code>function Navigation(){</code>
5	<code> return(</code>
6	<code> <div></code>
7	<code> <Link to="/">Home</Link></code>
8	<code> <Link to="/about">About</Link></code>
9	<code> </div></code>
10	<code>);</code>
12	<code>}</code>
13	
14	<code>export default Navigation;</code>

4 영화 상세정보 기능 만들어 보기

❖ Navigation 컴포넌트 정리하기

- 이제 본격적으로 영화 상세 정보 기능을 만들어 보자.
- **Movie** 컴포넌트를 누르면 영화 상세 정보 페이지로 이동해야 하니까 **Movie** 컴포넌트에 **Link** 컴포넌트를 추가하면 된다.

4 영화 상세정보 기능 만들어 보기

❖ Movie 컴포넌트에 Link 컴포넌트 추가하기

- **Movie** 컴포넌트에 **Link** 컴포넌트를 임포트하고, **Link** 컴포넌트에 **to props**를 작성하면 된다.
- 이때 **Link** 컴포넌트의 위치에 주의해야 한다.
- 안 그러면 영화 카드 모양이 이상해질 것!!!

1	import React from "react";
2	import propTypes from "prop-types";
3	import './Movie.css';
4	import {Link} from 'react-router-dom';
5	
6	function Movie({ title, year, summary, poster, genres}){
7	return (
8	<div className="movie">
9	<Link to={{
10	pathname: '/movie-detail',
12	state: {year, title, summary, poster, genres},
13	}}>
14	</div>

4 영화 상세정보 기능 만들어 보기

❖ Movie 컴포넌트에 Link 컴포넌트 추가하기

- **Movie** 컴포넌트에 **Link** 컴포넌트를 임포트하고, **Link** 컴포넌트에 **to props**를 작성하면 된다.
- 이때 **Link** 컴포넌트의 위치에 주의해야 한다.
- 안 그러면 영화 카드 모양이 이상해질 것!!!

15	<code></code>
16	<code><div className="movie__data"></code>
17	<code> <h3 className="movie__title">{title}</h3></code>
18	<code> <h5 className="movie__year">{year}</h5></code>
19	<code> <ul className="movie__genres"></code>
20	<code> {genres.map((genre, index) => {</code>
21	<code> return (</code>
22	<code> <li key={index} className="movie__genres"></code>
23	<code> {genres}</code>
24	<code> </code>
25	<code>);</code>
26	<code> })}</code>
27	<code></code>
28	<code><p className="movie__summary">{summary.slice(0, 180)}...</p></code>

4 영화 상세정보 기능 만들어 보기

❖ Movie 컴포넌트에 Link 컴포넌트 추가하기

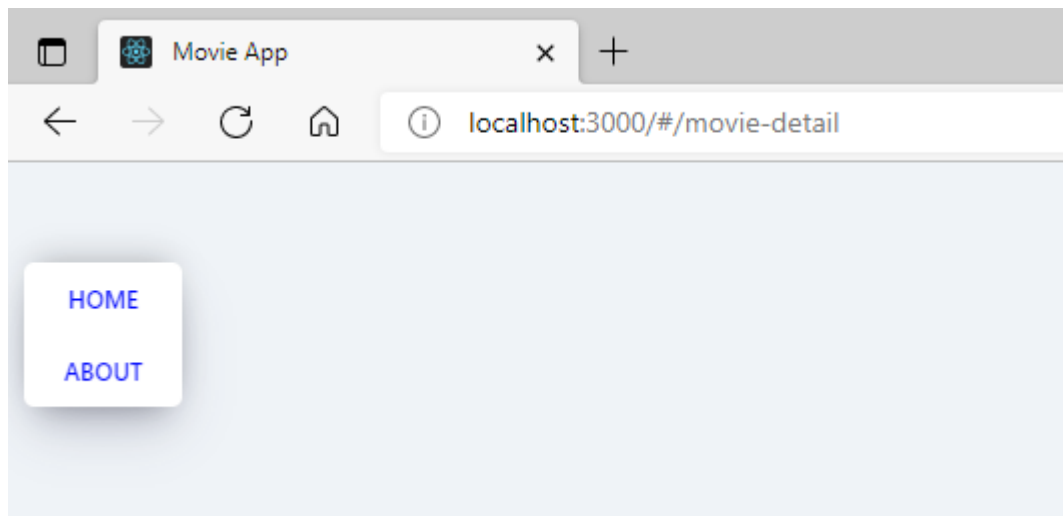
- **Movie** 컴포넌트에 **Link** 컴포넌트를 임포트하고, **Link** 컴포넌트에 **to** props를 작성하면 된다.
- 이때 **Link** 컴포넌트의 위치에 주의해야 한다.
- 안 그러면 영화 카드 모양이 이상해질 것!!!

29	<code></div></code>
30	<code></Link></code>
31	<code></div></code>
32	<code>);</code>
33	<code>}</code>
34	
35	<code>Movie.propTypes = {</code>
36	<code> year: propTypes.number.isRequired,</code>
37	<code> title: propTypes.string.isRequired,</code>
38	<code> summary: propTypes.string.isRequired,</code>
39	<code> poster: propTypes.string.isRequired,</code>
40	<code> genres: propTypes.arrayOf(propTypes.string).isRequired,</code>
41	<code>};</code>
42	<code>export default Movie;</code>

4 영화 상세정보 기능 만들어 보기

❖ Movie 컴포넌트에 Link 컴포넌트 추가하기

- 이제 영화 카드를 누르면 `/movie-detail`로 이동하게 된다.
- 그러면 `/movie-detail`로 이동했을 때 보여줄 화면이 필요하겠지?



4 영화 상세정보 기능 만들어 보기

❖ Detail 컴포넌트 만들기

- Detail 컴포넌트를 **routes** 폴더에 추가하자.
- 그리고 Detail 컴포넌트에서 **Movie** 컴포넌트의 **Link** 컴포넌트가 보내준 영화 데이터 (**state** : {**year**, **title**, **summary**, **poster**, **genres**})를 확인해 볼 수 있도록 **console.log()**도 작성하자.

1	<code>import React from "react";</code>
2	
3	<code>function Detail(props){</code>
4	<code> console.log(props);</code>
5	<code> return hello;</code>
6	<code>}</code>
7	
8	<code>export default Detail;</code>

4 영화 상세정보 기능 만들어 보기

❖ Detail 컴포넌트 만들기

- 아직 Detail을 출력해 주는 Route 컴포넌트를 추가하지 않았으므로 `console.log(props)`의 실행은 확인할 수 없다.
- App.js에서 Route 컴포넌트를 마저 추가해 주자.

4 영화 상세정보 기능 만들어 보기

❖ Route 컴포넌트 추가하기

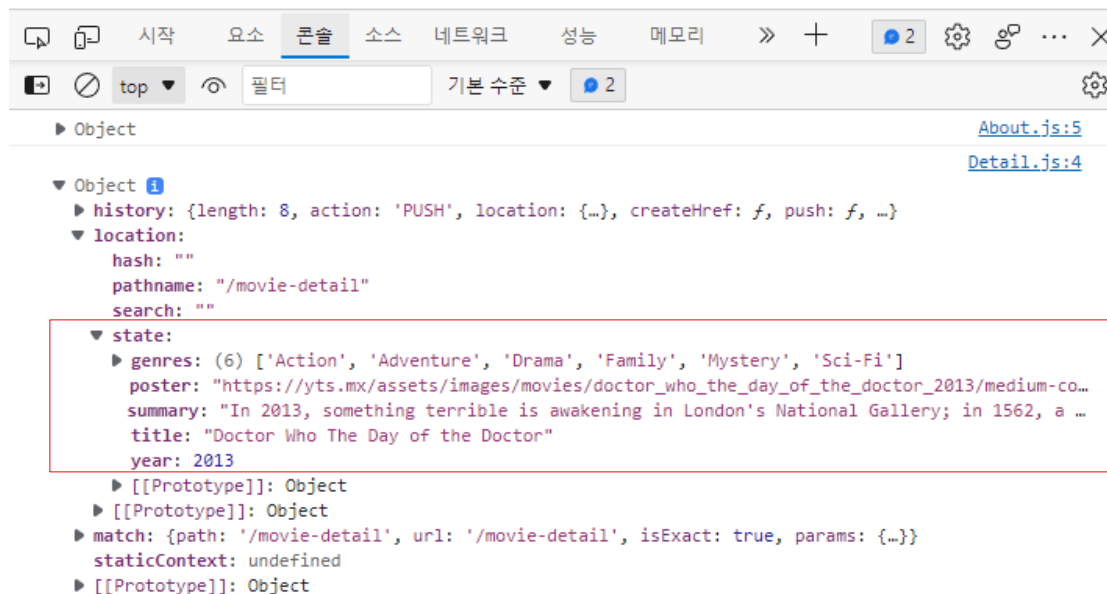
- App.js를 연 다음 Detail 컴포넌트를 임포트하고 Route 컴포넌트에서 Detail 컴포넌트를 그려 주도록 코드를 작성하자.

1	import React from "react";
2	import './App.css';
3	import {HashRouter, Route} from 'react-router-dom';
4	import About from './routes/About';
5	import Navigation from './components/Navigation';
6	import Detail from './routes/Detail';
7	
8	function App(){
9	return (
10	<HashRouter>
11	<Navigation />
12	<Route path="/" exact={true} component={Home} />
13	<Route path="/about" component={About} />
14	<Route path="/movie-detail" component={Detail} />
15	</HashRouter>
16);
	...

4 영화 상세정보 기능 만들어 보기

❖ 영화 카드를 눌러 /movie-detail로 이동한 다음 영화 데이터 확인하기

- 영화 카드를 눌러 /movie-detail로 이동해 보자.
- 화면을 보면 Detail 컴포넌트가 출력하고 있는 hello라는 문장이 보일 거야.
- 그리고 [콘솔] 탭을 보면 [location - state]에 Movie 컴포넌트에서 Link 컴포넌트를 통해 보내준 데이터가 들어있다.
- 직접 눈으로 확인해 보자.



4 영화 상세정보 기능 만들어 보기

❖ 영화 카드를 눌러 /movie-detail로 이동한 다음 영화 데이터 확인하기

- 자! 이 내용을 Detail 컴포넌트에서 출력하기만 하면 된다.
- 그런데 영화 카드를 눌러서 이동하지 않아도 영화 데이터는 Detail 컴포넌트에 잘 넘어갈까?
- 다시 말해 /movie-detail을 주소창에 직접 입력해서 이동하면 어떻게 될까?
- 한번 확인해 보자.

4 영화 상세정보 기능 만들어 보기

❖ /movie-detail로 바로 이동하기

- URL에 /movie-detail을 입력해서 바로 이동해 보자.
- 그런 다음 [콘솔] 탭에 영화 데이터가 있는지 확인해 보자.

http://localhost:3000/#/movie-detail



4 영화 상세정보 기능 만들어 보기

❖ /movie-detail로 바로 이동하기

- Detail 컴포넌트의 `hello`는 잘 출력하고 있지만 [콘솔] 탭에는 영화 데이터가 없다(`state`가 `undefined`).
- Detail 컴포넌트로 영화 데이터가 넘어오지 못한다.
- 이런 경우 사용자를 강제로 `Home`으로 돌려보내야 한다(다시 영화 카드를 눌러서 이동할 수 있도록).
- 바로 그 기능을 리다이렉트 기능이라 한다.
- 이제 리다이렉트 기능을 추가하기 위해 `Detail` 컴포넌트를 수정해 보자.

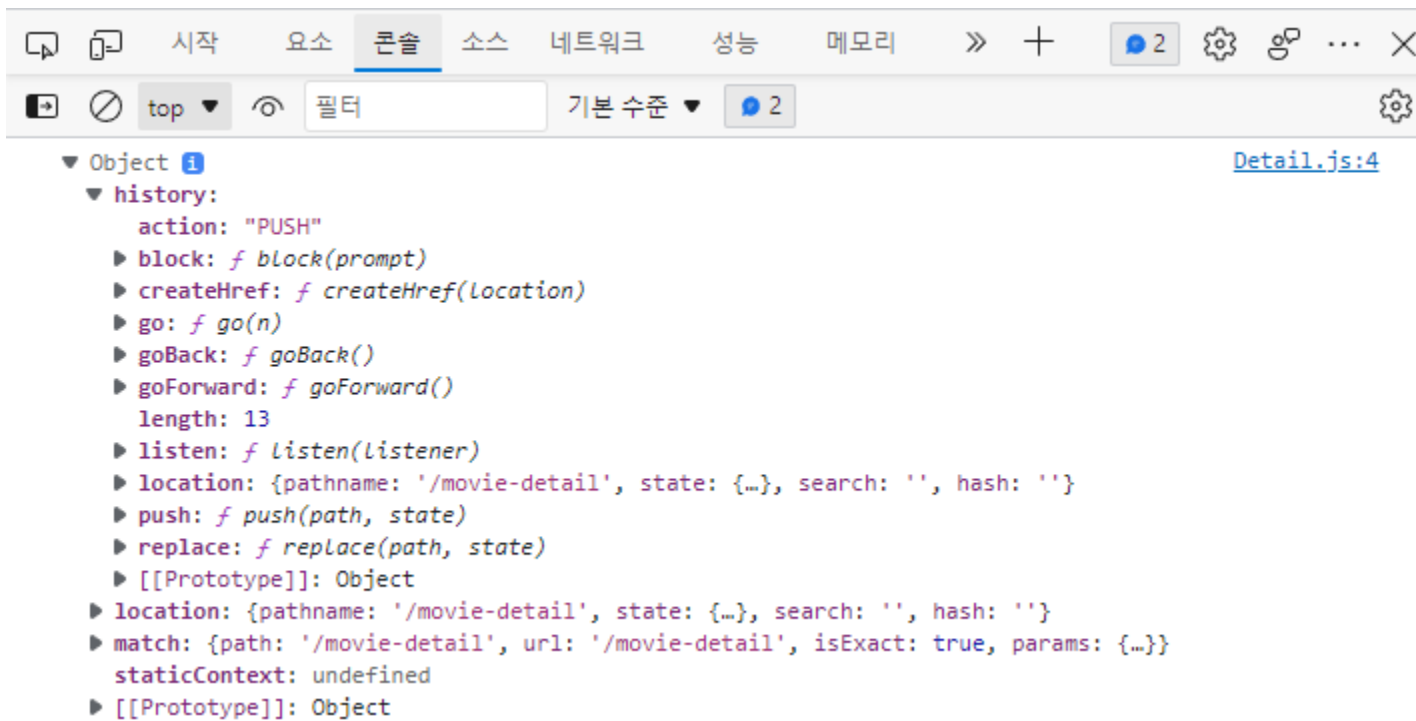
5 리다이렉트 기능 만들어 보기

- ❖ 앞에서 설명했듯 리다이렉트 기능을 추가해 보자.
- ❖ 리다이렉트 기능을 위해서는 `route props`의 `history` 키를 활용해야 한다.
- ❖ `History` 키는 언급한 적이 없으니 잘 모를 것이다.
- ❖ `History` 키에는 `push`, `go`, `goBack`, `goForward`와 같은 키가 있는데 그 키에는 URL을 변경해 주는 함수들이 들어 있다.
- ❖ 리다이렉트 기능은 이 함수들을 이용해 만든다.
- ❖ 우선 `history` 키에 정말 그런 값들이 들어 있는지 살펴보자.

5 리다이렉트 기능 만들어 보기

❖ history 키 살펴보기

- 주소창에 localhost : 3000를 입력해서 이동한 다음 아무 영화 카드나 눌러 이동해 보자.
- 그런 다음 [콘솔] 탭에서 [history]에 출력된 값을 펼쳐서 살펴보자.



5 리다이렉트 기능 만들어 보기

❖ history 키 살펴보기

- `push`, `go`, `goBack`, `goForward` 키가 보이는가?
- 이게 다 URL을 변경해 주는 함수이다.
- 이 중 지정한 URL로 보내주는 `push()` 함수를 사용해 보자.
- 그전에 `Detail` 컴포넌트를 클래스형 컴포넌트로 변경하자.
- 그래야 `componentDidMount()` 생명주기 함수를 사용해 `Detail` 컴포넌트가 마운트될 때 `push()` 함수를 실행할 수 있기 때문이다.

5 리다이렉트 기능 만들어 보기

❖ Detail 컴포넌트 클래스형 컴포넌트로 변경하기

- Detail 컴포넌트를 함수형에서 클래스형 컴포넌트로 변경한 다음 location, history 키를 구조 분해 할당하자.

1	import React from "react";
2	
3	class Detail extends React.Component{
4	componentDidMount(){
5	const {location, history} = this.props;
6	}
7	
8	render(){
9	return hello;
10	}
11	}
12	
13	export default Detail;

5 리다이렉트 기능 만들어 보기

❖ Detail 컴포넌트 클래스형 컴포넌트로 변경하기

- 여기까지 코드를 작성하고 다시 생각해 보자.
- 사용자가 URL을 직접 입력해서 `/movie-detail`로 이동하면 `location` 키의 `state` 키가 비어 있었다.
- 그런 경우에만 `history` 키의 `push()` 함수를 사용하도록 하자.

5 리다이렉트 기능 만들어 보기

❖ push() 함수 사용하기

- `location.state`가 `undefined`인 경우 `history.push("/")`를 실행하도록 코드를 작성하자.

```
1  import React from "react";
2
3  class Detail extends React.Component{
4    componentDidMount(){
5      const {location, history} = this.props;
6      if(location.state === undefined){
7        history.push('/');
8      }
9    }
10
11    render(){
12      return <span>hello</span>;
13    }
14  }
15
16  export default Detail;
```

5 리다이렉트 기능 만들어 보기

❖ 리다이렉트 기능 확인해 보기

- 영화 앱을 실행한 다음 직접 주소를 입력해서 `/movie-detail`으로 이동해 보자.
- 그러면 다시 `Home`으로 돌아오게 된다.
- 이제 영화 카드를 눌러 영화 상세 페이지로 이동하지 않으면 다시 첫 페이지로 이동하게 된다.
- 하지만 아직 영화 상세 정보 페이지로 이동하면 `hello`만 출력될 뿐이고 영화 상세 정보가 출력되지는 않는 상태이다.
- 이제 영화 상세 정보 페이지를 만들어 보자.

5 리다이렉트 기능 만들어 보기

❖ 영화 제목 출력하기

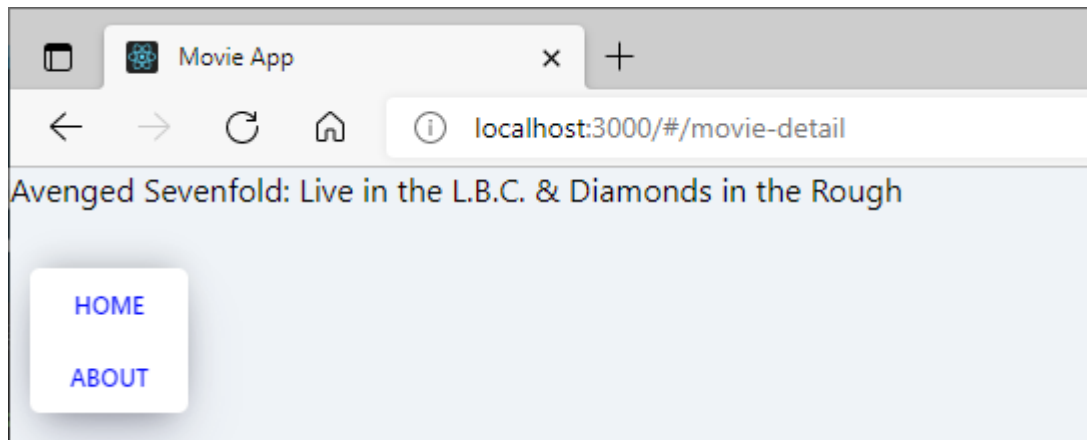
- 우선 영화 제목부터 출력해 보자.
- **Movie** 컴포넌트로부터 전달받은 영화 데이터는 `location.state`에 들어 있었다.
- 이제 `hello`가 아닌 `location.state.title`을 출력해 보자.

	...
11	render(){
12	const {location} = this.props;
13	return {location.state.title} ;
14	}
15	
16	export default Detail;

5 리다이렉트 기능 만들어 보기

❖ 영화 제목 출력하기

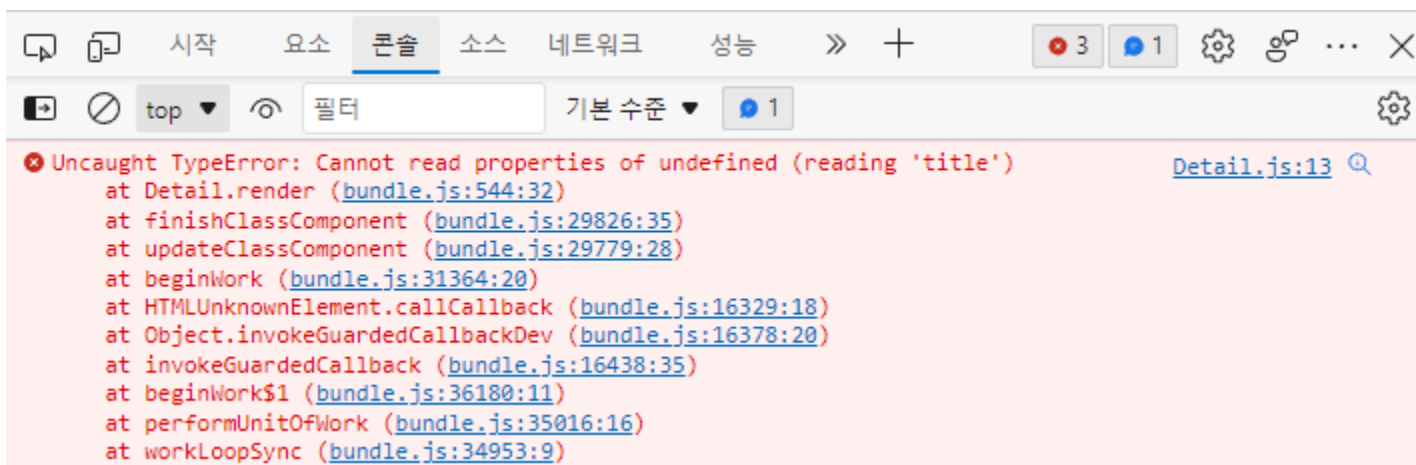
- 이제 첫 화면에서 영화 카드를 누르면 영화 제목이 나타난다.



5 리다이렉트 기능 만들어 보기

❖ /movie-detail로 바로 이동하기

- 그런데 또 다시 /movie-detail로 바로 이동하면 오류가 발생한다(새 로고침도 마찬가지이다).



The screenshot shows a web browser's developer console with the 'Console' tab selected. It displays an 'Uncaught TypeError: Cannot read properties of undefined (reading 'title')' at `Detail.js:13`. The stack trace includes the following frames:

```
Uncaught TypeError: Cannot read properties of undefined (reading 'title')    Detail.js:13
    at Detail.render (bundle.js:544:32)
    at finishClassComponent (bundle.js:29826:35)
    at updateClassComponent (bundle.js:29779:28)
    at beginWork (bundle.js:31364:20)
    at HTMLUnknownElement.callCallback (bundle.js:16329:18)
    at Object.invokeGuardedCallbackDev (bundle.js:16378:20)
    at invokeGuardedCallback (bundle.js:16438:35)
    at beginWork$1 (bundle.js:36180:11)
    at performUnitOfWork (bundle.js:35016:16)
    at workLoopSync (bundle.js:34953:9)
```

5 리다이렉트 기능 만들어 보기

❖ /movie-detail로 바로 이동하기

- `componentDidMount()` 생명주기 함수에 작성한 리다이렉트 기능이 동작하지 않는 것 같다.
- 왜 그럴까?
- 바로 그 이유는 Detail 컴포넌트는 [`render()` → `componentDidMount()`]의 순서로 함수를 실행하기 때문이다.
- `render()` 함수 내에서 `location.state.title`을 사용하려 하는데, `location.state`가 아까와 마찬가지로 `undefined`이기 때문이다.
- 그렇기 때문에 `render()` 함수에도 `componentDidMount()` 생명주기 함수에 작성한 리다이렉트 코드를 추가해 주어야 한다.

5 리다이렉트 기능 만들어 보기

❖ location.state 확인하기

- location.state가 없으면 render() 함수가 null을 반환하도록 수정하자.

	...
11	render(){
12	const {location} = this.props;
13	if(location.state){
14	return {location.state.title};
15	}else{
16	return null;
17	}
18	}
19	
20	export default Detail;

5 리다이렉트 기능 만들어 보기

❖ `location.state` 확인하기

- `location.state`가 없으면 `render()` 함수가 `null`을 반환하도록 만들어서 문제없이 실행되도록 만들었다.
- 그러면 이어서 `componentDidMount()` 생명주기 함수가 실행되면서 리다이렉트 기능이 동작한다.

6 영화 앱 깃허브에 배포하기

❖ package.json 파일 수정하기

- 이제 영화 앱을 깃허브에 배포해 보겠습니다.
- 가장 먼저 해야 할 일은 깃허브 주소를 영화 앱이 인식할 수 있도록 **package.json** 파일을 열어 수정하는 것입니다.
- **homepage** 키와 키값을 **browserslist** 키 아래에 추가하세요.
- 깃허브 계정과 저장소 이름에 주의하여 입력하세요.

1	{
2	"name": "movie_app_2021",
3	"version": "0.1.0",
4	"private": true,
5	"dependencies": {
6	"@testing-library/jest-dom": "^5.16.1",
7	"@testing-library/react": "^12.1.2",
8	"@testing-library/user-event": "^13.5.0",
9	"axios": "^0.24.0",
10	"prop-types": "^15.8.0",
11	"react": "^17.0.2",
12	"react-dom": "^17.0.2",
13	"react-router-dom": "^5.2.0",

6 영화 앱 깃허브에 배포하기

❖ package.json 파일 수정하기

- 이제 영화 앱을 깃허브에 배포해 보겠습니다.
- 가장 먼저 해야 할 일은 깃허브 주소를 영화 앱이 인식할 수 있도록 **package.json** 파일을 열어 수정하는 것입니다.
- **homepage** 키와 키값을 **browserslist** 키 아래에 추가하세요.
- 깃허브 계정과 저장소 이름에 주의하여 입력하세요.

14	"react-scripts": "5.0.0",
15	"web-vitals": "^2.1.2"
16	},
17	"scripts": {
18	"start": "react-scripts start",
19	"build": "react-scripts build"
20	},
21	"eslintConfig": {
22	"extends": [
23	"react-app",
24	"react-app/jest"
25]
26	},

6 영화 앱 깃허브에 배포하기

❖ package.json 파일 수정하기

- 이제 영화 앱을 깃허브에 배포해 보겠습니다.
- 가장 먼저 해야 할 일은 깃허브 주소를 영화 앱이 인식할 수 있도록 package.json 파일을 열어 수정하는 것입니다.
- homepage 키와 키값을 browserslist 키 아래에 추가하세요.

```
27  "browserslist": {  
28    "production": [  
29      ">0.2%",  
30      "not dead",  
31      "not op_mini all"  
32    ],  
33    "development": [  
34      "last 1 chrome version",  
35      "last 1 firefox version",  
36      "last 1 safari version"  
37    ],  
38  },  
39  "homepage": "https://jjin300.github.io/movie_app_2021"  
40 }
```

계정이름

저장소 이름

6 영화 앱 깃허브에 배포하기

❖ package.json 파일 수정하기

- 이어서 package.json 파일에 scripts 키값으로 명령어를 추가하세요.

	...
17	"scripts": {
18	"start": "react-scripts start",
19	"build": "react-scripts build",
20	"predeploy" : "npm run build",
21	"deploy" : "gh-pages -d build"
22	},
23	"eslintConfig": {
	...

6 영화 앱 깃허브에 배포하기

❖ 최종 코드 깃허브에 업로드하기

- 강의 초반에 영화 앱을 깃허브 저장소에 업로드 했었습니다.
- 같은 방법으로 최종 코드를 깃허브에 업로드하겠습니다.
- 아래의 명령어를 입력하여 최종 완성 코드를 깃허브 저장소에 업로드 하세요.

```
>git add .
```

```
>git commit -m " 최종 완성 코드 "
```

```
>git push origin master
```

6 영화 앱 깃허브에 배포하기

❖ `gh-pages` 설치하기

- 최종 완성 코드를 깃허브에 업로드했습니다.
- 이제 배포 서비스를 이용해서 다른 사람들이 영화 앱을 사용할 수 있도록 해야 합니다.
- 여기서는 깃허브에서 제공하는 *GitHub Pages* 서비스(무료)로 영화 앱을 배포합니다.
- 프로젝트 루트 폴더에서 다음 명령어를 입력하여 `gh-pages`를 설치하세요.

```
>npm install gh-pages
```

6 영화 앱 깃허브에 배포하기

❖ 깃허브 저장소 다시 확인하기

- 다음 명령어를 입력하면 여 분이 업로드한 깃허브 저장소의 주소를 확인할 수 있습니다.
- 저장소 주소에 있는 깃허브 계정 이름(jjin300)과 깃허브 저장소 이름(movie_app_2021)을 확인하세요.
- 이 이름들이 깃허브에 배포한 영화 앱 주소에 사용될 것입니다.

>git remote -v

```
C:\movie_app_2021>git remote -v
origin  https://github.com/jjin300/movie_app_2021.git (fetch)
origin  https://github.com/jjin300/movie_app_2021.git (push)
C:\movie_app_2021>
```

6 영화 앱 깃허브에 배포하기

❖ 영화 앱 깃허브에 배포하기

- 터미널에서 다음 명령어를 입력하여 영화앱을 배포하세요.

>npm run deploy

```
Creating an optimized production build...
Compiled with warnings.

src/routes/Detail.js
  Line 6:27: Expected '===' and instead saw '=='   egegeg

Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.

File sizes after gzip:

  59.81 kB  build/static/js/main.d55bcdbe.js
  895 B    build/static/css/main.4655aec4.css

The project was built assuming it is hosted at /movie_app_2021/.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.

Find out more about deployment here:

https://cra.link/deployment

> movie_app_2021@0.1.0 deploy
> gh-pages -d build

Published

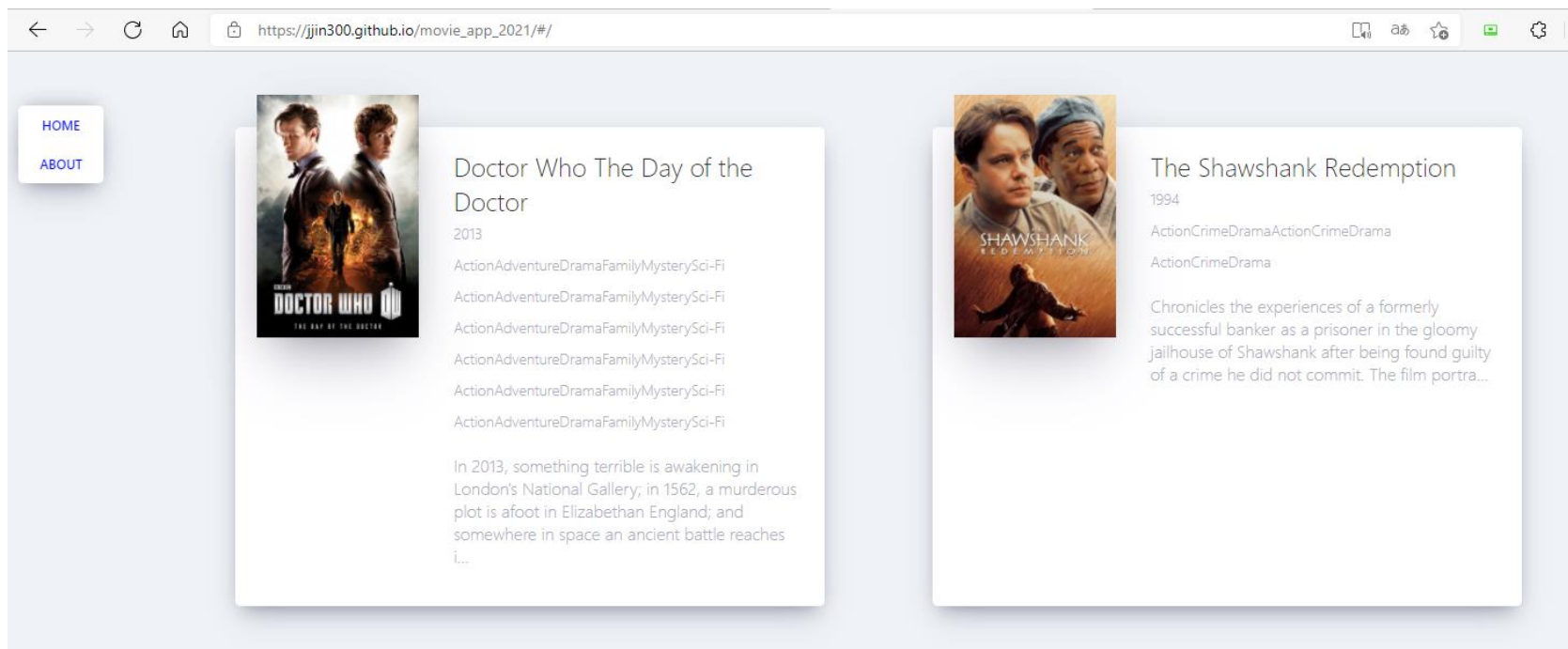
C:\movie_app_2021>
```

6 영화 앱 깃허브에 배포하기

❖ Github Pages에 접속하여 영화 앱 확인하기

- URL에 `https://[깃허브 계정].github.io/[저장소 이름]`을 입력해 보세요.
- 이 예에서는 아래와 같이 주소를 입력했습니다.
- 깃허브에 배포한 영화 앱을 확인해 보세요!

https://jjin300.github.io/movie_app_2021





Thank You !