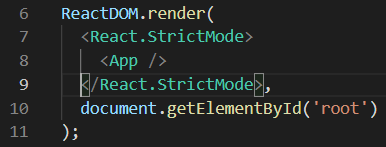
Component는 html을 반환하는 함수이다.

Index.js에서



<App />은 App.js에서의 App을 실행한다.

이런 javascript와 HTML의 조합을 jsx라고 부른다. 🡪 react에서의 유일한 특성

React application은 하나의 component만을 rendering해야한다.



위 코드는

Players component에 name이라는 property를 다른 4개의 value로 준 것이다.

Players function의 argument로 name이 들어가기 때문에



H1태그로 4개의 문장이 출력된다.

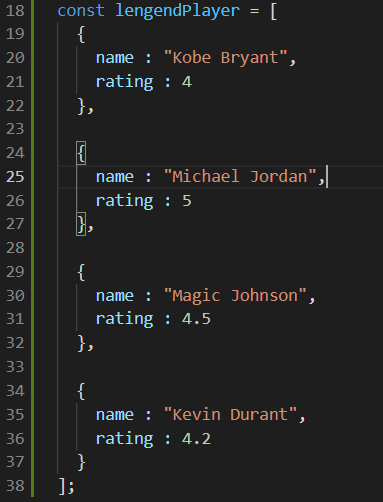
+ Props로 들어가는 value의 자료형은 여러 가지가 될 수 있다.

+ Component는 대문자로 시작해야 한다.

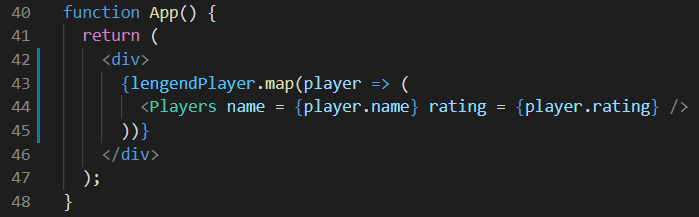
🡪 근데 위와 같은 방법도 효율적이지 못함!

결국엔 데이터를 추가할 때마다 코드를 복붙해야하기 때문에..

그래서 JS의 map을 이용할 것이다.



Object의 list 형태로 출력 대상을 다시 선언해주고,

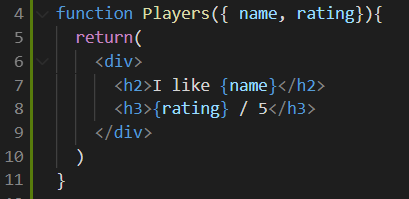


App()에서도 map을 이용하여 player라는 이름으로 legendplayer의 속성을 동일하게 받는다.

그리고 동시에 Players()의 인자로 넣어준다.

🡪 결국 실행되는 것은 App()이기 때문이다.!

+



Players의 구조는 위와 같다.

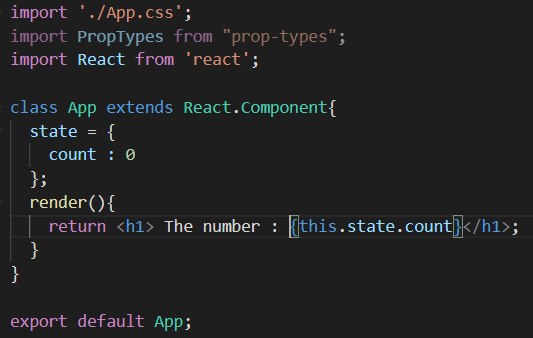


당연히 위와 같이 출력될 것이다.

STATE

React의 state를 사용하기 위해서는 class component를 사용한다.

State는 객체이다. Component의 data를 넣을 공간이 있고, 이 data는 동적이다.

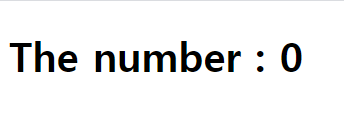


위와 같이 React.Component를 상속받는 class인 App을 생성한다.

내부에 state를 선언하고 state는 count라는 data를 가지고 있다.

Render()함수를 실행하고, count를 출력하기 위해서는

class이므로, this.state.count로 작성해준다.



요로코롬 나온다.

우리가 관심가질 부분은 당연히

저 data(예시에서는 count)를 어떻게 변경할 것인가 이다.

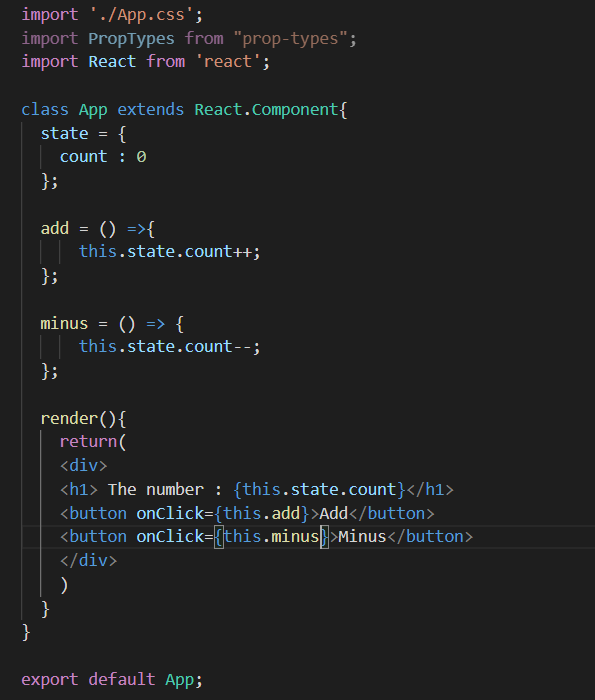
참고로 react의 button에는 onClick이 기본적으로 탑재되어있다.

js에서는 addEventListener()를 사용해야 했다. React가 더 편리한 부분이라고 할 수 있겠다.

그렇다면,

숫자를 add해주는 button과 minus해주는 button을 이용해서

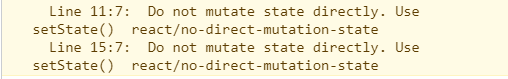
누를 때마다, 다시 말해 onClick이 발생할 때마다 add, minus해주는 함수를 달아주면 되지 않을까?



이런 식으로 말이다!

하지만 애석하게도…(어쩌면 당연하게도…) 버튼을 눌러도 아무런 변화가 없다.

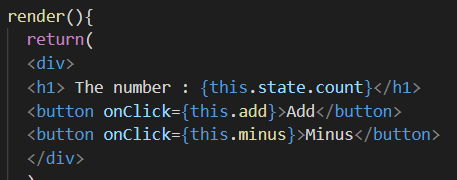
그리고 console창을 확인해보면,



대충 이런 경고창이 나온다.

해석하자면, state를 직접적으로 수정하지 말고, setState()를 사용하라는 경고이다.

다시 우리가 썼던 코드를 보자.



이런 식으로 함수를 쓰면,

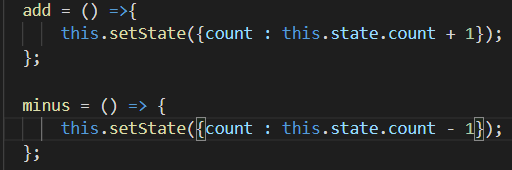
우리가 버튼을 누를 때마다 render를 다시 실행해서 refresh를 해줘야 값이 update된다.

State를 직접적으로 변경하는 방식으로는 react가 우리의 의도를 바로 알 수 없기 때문에 위와 같은 경고가 나온다.

하지만 setState()를 사용하면, react가 우리의 의도를 파악해서 state를 알아서 refresh해준다.

(그 과정에서 render를 다시 호출하는 것)

따라서, add()와 minus()를 다음과 같이 작성해야 한다.

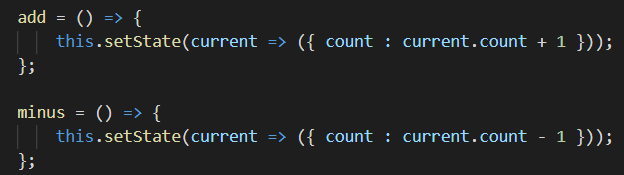


🡪 setState()를 사용하고, {}를 이용해 jsx의 prop을 표현해주고, class이기 때문에 this를 사용한 방식이다! 정말 객체지향적이다.

그런데 사실 위와 같은 방식이 neat한 방식은 아니다…

외부의 영향을 최대한 받지 않고 “현재 값”에 기반하여 data를 변경하고 싶을 때는,

react에서 제공하는 current를 이용하면 좋다.



위와 같은 방식이 가장 트렌디하다고 한다.

React component에서 사용하는 유일한 함수는 render()이다.

Render()가 실행되기 전과 후에 실행되는 다른 함수들이 있다.

즉, component가 update될 때, 호출되는 다른 함수들이 있다.

1. mounting = component가 태어나는 것과 같다.

1-1. constructor() : 생성자

🡪 component가 모니터에 표시될 때, component가 website에 갈 때, constructor를 호출한다.

1-2. static getDerivedStateFromProps() : 거의 사용하지 않는다.

1-3. render()

1-4. componentDidMount() : 해당 component가 최초로 render됐음을 알림

2. updating 🡪 개발자가 component의 값을 update하는 것

2-1. static getDerivedStateFromProps() : 거의 사용 안함

2-2. shouldComponentUpdate() : 거의 사용 안함🡪update를 할 지 말지 결정하는 것에 대한 함수

2-3. render()

2-4. getSnapshotBeforeUpdate() : 거의 사용 안함

2-5. componentDidUpdate() : component가 update됐는지를 확인하는 함수

🡪 update가 일어나면, 특히 setState()를 호출하면 위와 같은 순서로 함수들이 호출된다.

3. unmounting : component가 죽을 때, 즉 page를 바꿀 때

- componentWillUnmount : 변경사항을 저장하면 서버가 새로고침 되며 컴포넌트도 같이 사라지기 때문에 이를 통해 작동을 확인할 수 있다.

일반적으로 JS에서 data를 fetch하는 방법은 fetch를 사용하는 것이다.

🡪 하지만 더 나은 방법이 있는데, axios가 그것이다.

Axios는 fetch 위에 있는 작은 layer이다.

이 프로젝트에서는 yts라는 api를 사용할 것이다.

Yts는 영화를 불법 토렌트 형식으로 다운받는 사이트라서 url이 지속적으로 바뀐다.

그래서 니꼬쌤이 고정시킨 url을 사용할 것이다.

<https://yts-proxy.nomadcoders1.now.sh/list_movies.json>

이 url에 들어가면



스크롤을 보면 알 수 있겠지만

방대한 양의 api를 얻을 수 있다.

하지만 이 api를 직접 다루기 위해서는 data를 잡아야 한다(강의에서는 grab the data라고 표현했다.)

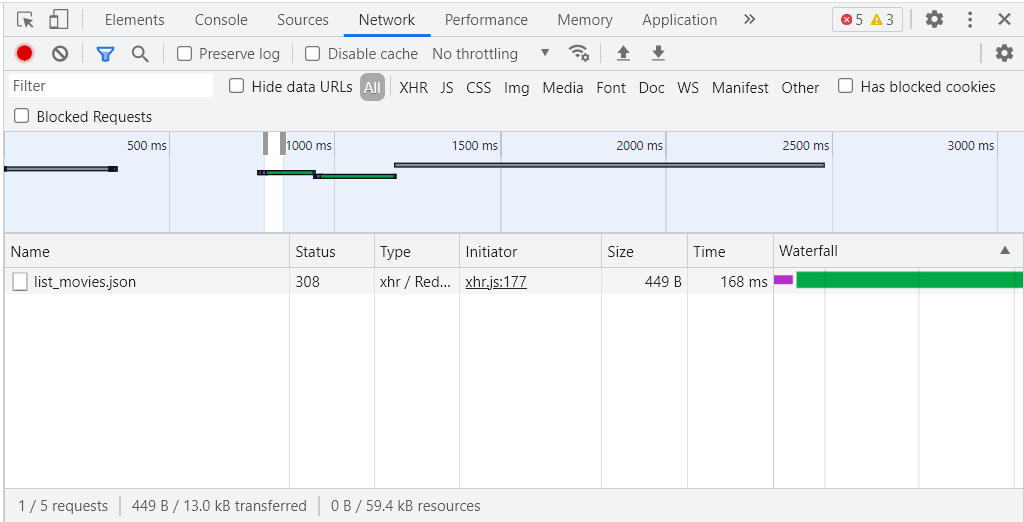
이 과정에서 axios를 사용할 것이다.

(나중에 axios에 대해 다시 조사해봐야겠다.



일단 코드의 내용을 보기 보다는 axios를 이용해 원하는 api를 get한 것만 확인해보자.

위를 실행하고 개발자 도구를 통해 네트워크를 확인하면 아래와 같다.



Axios가 무언가를 요청하고 있음을 볼 수 있다.

그렇다면 이제 코드를 자세히 보자



강의 내용에 따르면,

Axios.get은 항상 빠른 함수는 아니다.

심지어는 조금 느리다고까지 표현했다.

따라서, async한 함수(비동기적인 함수)라고 선언해주고 getMovies()가 실행완료되기까지 시간이 필요하다고 명시해주어야 한다.

Async()와 await을 이용해 표현한 것을 확인할 수 있다.

우선 async()를 이용해 해당 함수가 어떤 task를 기다려야 한다고 알려주고,

Axios 앞에 await을 추가해 axios를 기다려야 한다고 알려준다.

그렇다면 위 코드의 전체적인 흐름을 파악해보자.

우선 application이 render하면

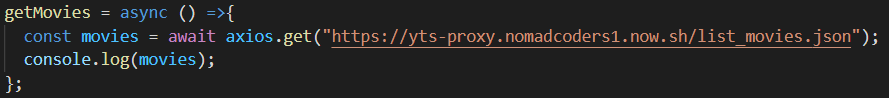
1. state가 정의된다. isLoading = true가 되고 movies라는 빈 배열이 생성된다.

2. render()에서 isLoading이 참이기 때문에 화면에 “Now Loading”이 출력된다.

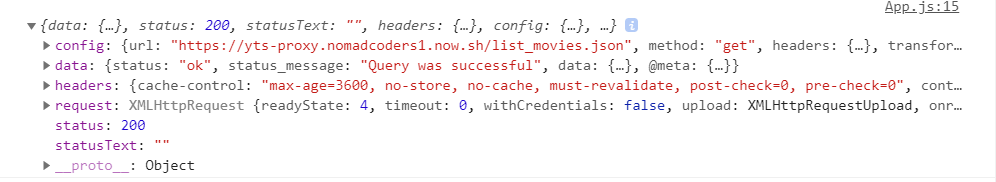
3. application이 성공적으로 mount됐기 때문에 getMovies 함수가 호출된다.

4. getMovies()는 async하게 실행된다. 즉, axios.get이 완료될 때까지 기다린다.(await)

그렇다면 우리가 얻어온 api에 대해 알아볼 필요가 생긴다.

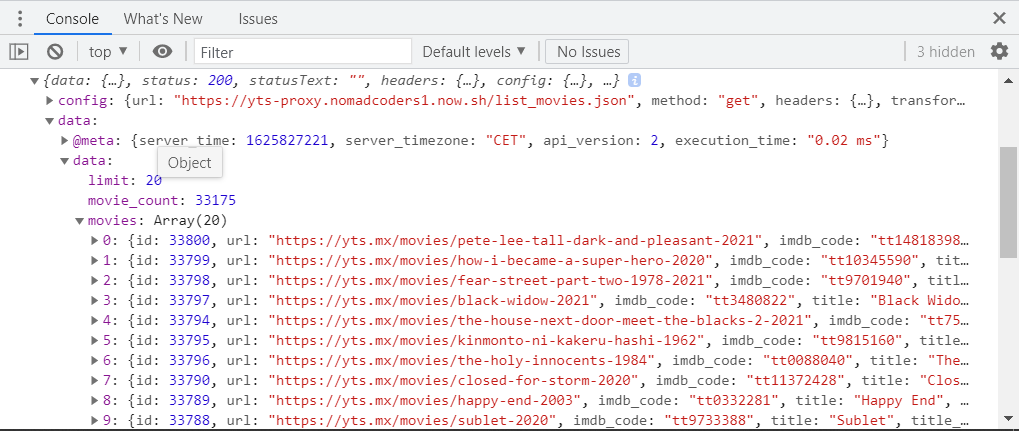


위와 같이 console.log를 이용하여 쌩으로 출력해보자.



뭐 대충 저렇게 나온다.

우리가 우선적으로 사용할 data는 movie의 목록이므로,



위와 같이 data – data – data – movies에 해당한다.

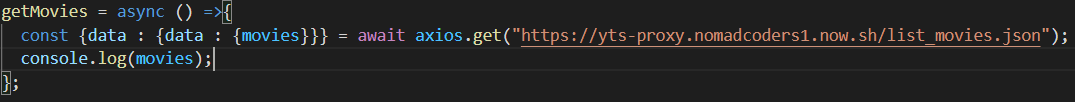
따라서,

Movies.data.data.movies를 출력하면(console.log(movies.data.data.movies;),



이렇게 20개의 영화 목록을 뽑아낼 수 있다.

근데 저렇게 무식하게 써놓으면 neat하지 못하므로,



해당 api의 data – data – movies로 표현하면 더 깔끔하다.

(근데 솔직히 난 여기까지는 생각하지 못했을 듯 ..)

그렇다면,

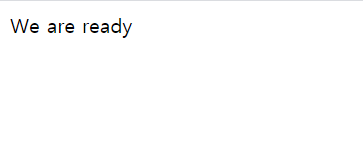
이제 우리가 할 것은 getmovies에서의 movies를, 즉 api에서 axios로 get한 movie의 list를,

state에서의 movies[]에 넣어주는 것이다. 또한 movie list를 모두 가져오면 loading이 끝나기 때문에 이 부분도 수정해야 한다.

This.setState({movies, isLoading : false});

로 작성하면 된다.

성공적으로 api에서 movies를 가져온다면



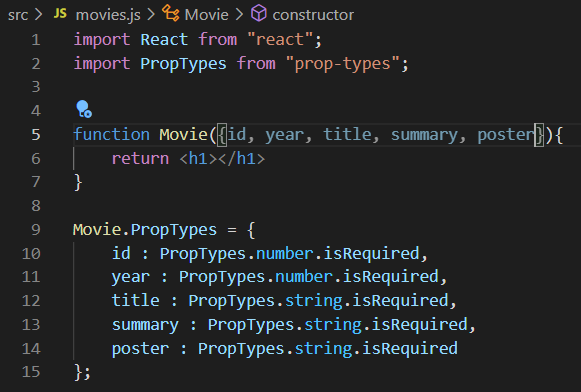
위와 같은 창을 볼 수 있을 것이다!

하지만 우리가 보고 싶은 것은 고작 “We are ready”와 같은 문장이 아니다.

영화의 정보에 대해 알고 싶은 것이므로,

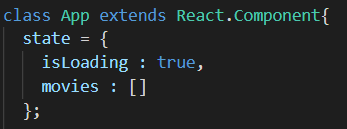
추가적인 js파일(movies.js)을 하나 생성해서 이어가보자.

다음과 같이 PropTypes를 이용하여 React 컴포넌트에 prop으로 넘어오는 입력값들이 어떤 타입을 가져야하는지 정의할 수 있다.



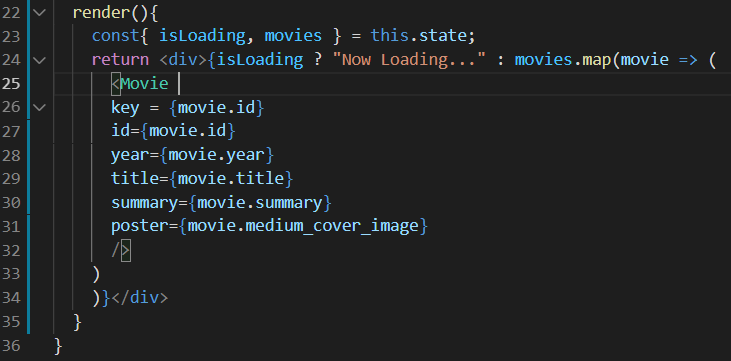
\*\*\*Function component의 인자로 props의 특정 값을 가져올 때는 중괄호로 감싸줘야한다! \*\*\*

이제 마지막으로 App.js에서 해당 내용을 render해주면 된다.



Render할 내용은 App class의 state 내부의 movies[]라는 data이다.

따라서



위와 같이 작성하면 된다.

Map을 이용하여 깔끔하게 작성해주었다.

이제 추가할 내용은

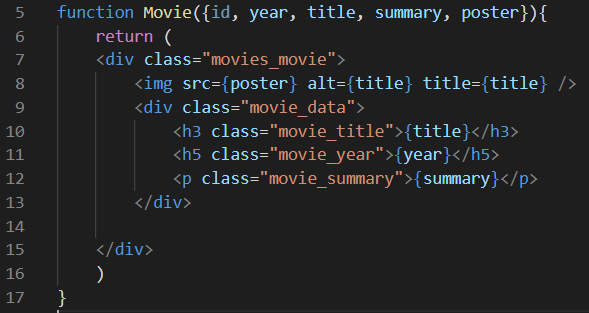
영화의 여러 정보들을 출력하게끔 하는 것이다.

강의를 듣다 보니 이상적인 코드는

“웹으로 표현되는 모든 조각들을 <div>와 같은 태그로 작성해준 코드”라는 생각이 들었다.

이에 맞게끔 loading 안내 문구도 태그로 감싸주고

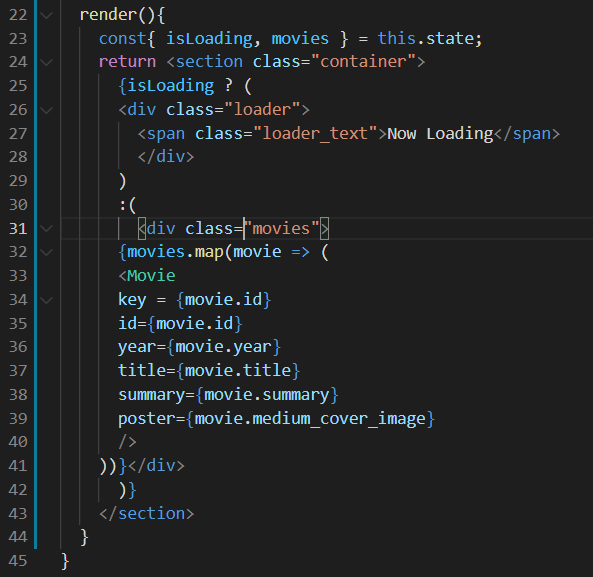
출력되는 영화의 정보(제목, 제작년도, 줄거리, 포스터)도 태그로 감싸주었다.



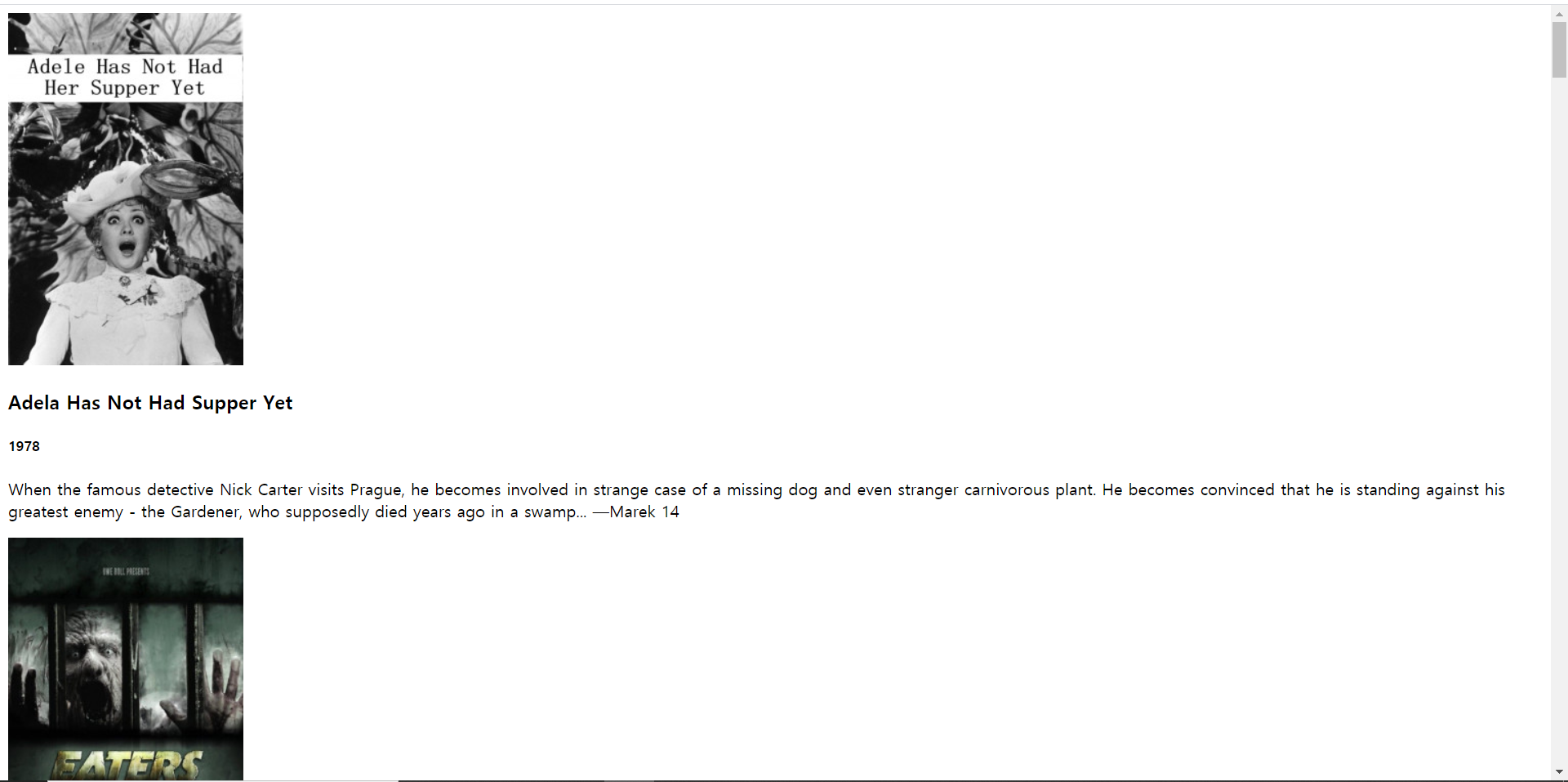
이렇게 말이다.

-> <img>에서의 alt는 마우스를 포스터에 올려놓을 때 영화의 제목이 나오게끔 해주는 역할을 한다.

(위 코드는 movies.js의 일부분이다.)



App.js의 코드도 위와 같이 적절한 태그로 작성해주었다.



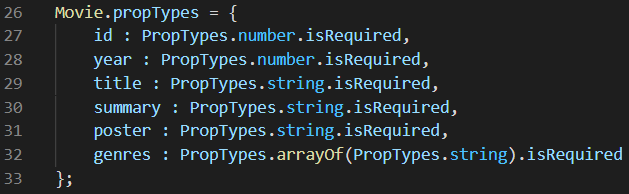
대충 이런 식으로 전체적인 골격이 생겼다!

추가적으로 각 영화의 장르를 추가할 것이다.

우선적으로 할 작업은

Movie.js파일에서 propTypes에 장르 항목을 추가해서

React 컴포넌트에 prop으로 넘어오는 입력값들이 어떤 타입을 가져야하는지 정의하는 것이다.

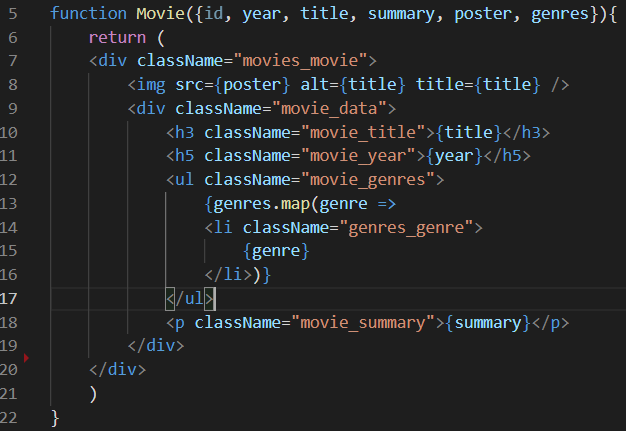


Api를 본 결과 genre는 문자열의 배열 형태였다.

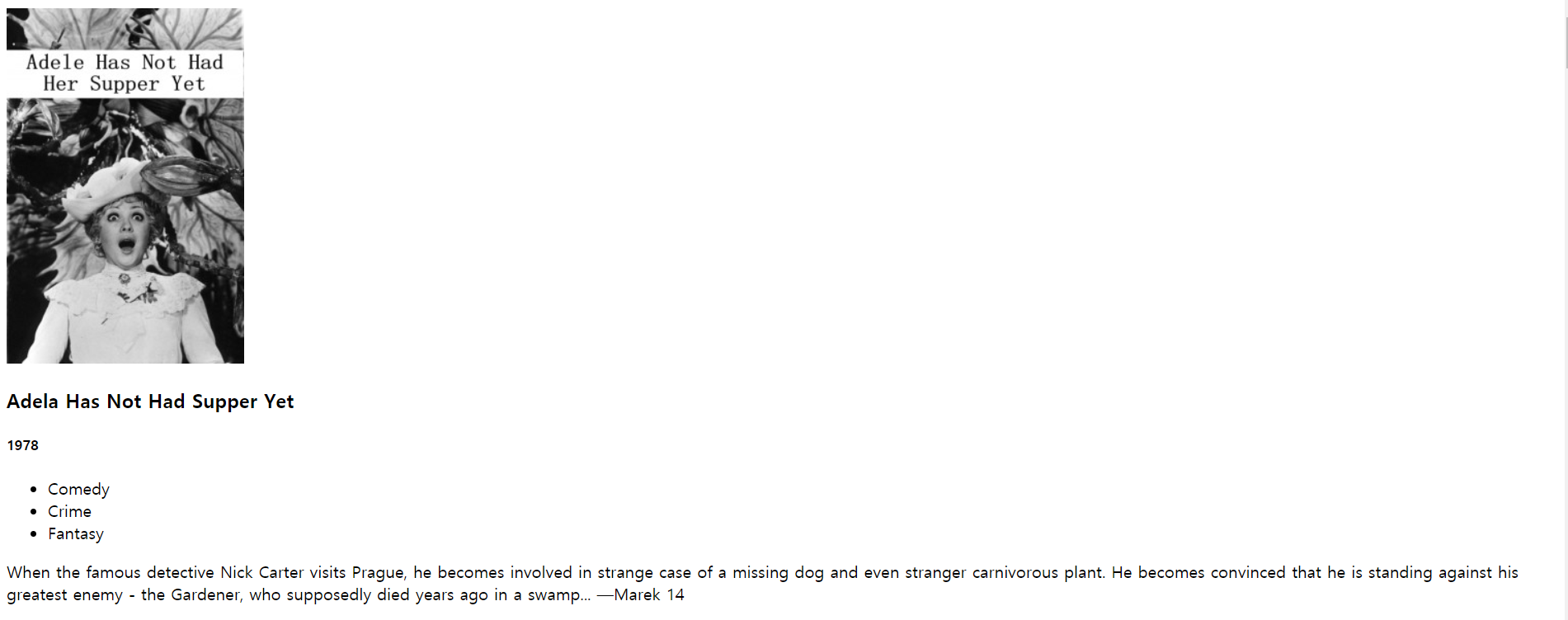
PropTypes이 배열인 경우에는 위처럼

PropTypes.arrayOf(PropTypes.문자형).isRequired 로 작성하면 된다.

그리고 App.js가 render하는 내용은 movie.js가 넘겨주는 Movie()이므로 Movie에도 genres를 추가한다.



이렇게 추가해준다.(js의 map을 자주 이용하는 듯하다.)



이렇게 장르까지 성공적으로 추가했다.

그런데 아직 해결해야할 오류가 있다.

console창을 확인해보면,



이런 에러 메시지가 있다.

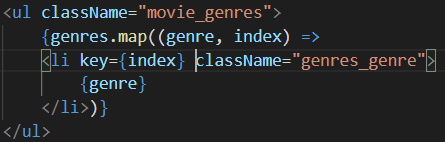
Map에 있는 각각의 item은 key가 필요한데 우리가 key를 부여하지 않았다는 것이다.

이전에는 movie 전체를 map했기 때문에 api에 기본적으로 주어진 id를 이용해서 key를 할당할 수 있었다. 하지만 이 경우 id가 따로 없는 genres prop을 map한 것이기 때문에 key를 줄 수 없다.

이 경우 다른 방법을 쓰면 된다.

사실 map()은 사용자에게 item에 하나씩 대응되는 0부터 시작하는 index 값도 함께 제공한다.(item number)

이 값을 id로 사용해주면 된다.



하지만 앞으로 우리가 만들 웹페이지는 하나의 창으로 구성할 것이 아니다.

다른 많은 웹페이지가 그렇듯

사용자의 클릭에 따라 다른 창을 제공하는 것이 이상적일 것이다.

이를 위해 react-router-dom을 이용할 것이다.

전체적인 구조는

기존의 영화 목록을 담고 있던 App.js를 home.js로 바꾸고

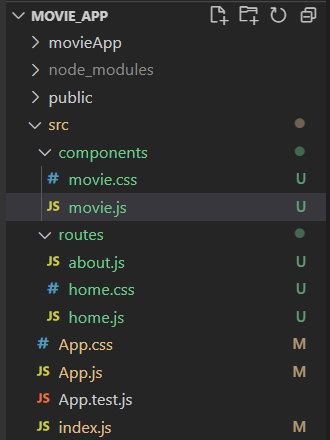
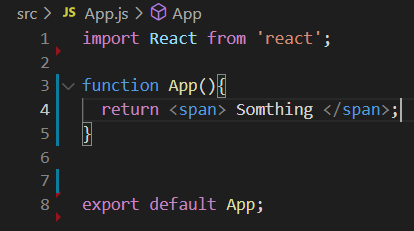
영화의 정보를 담는 페이지인 about.js를 따로 만들어 준다.

App.js는 사용자의 선택에 따라 home.js나 about.js를 보여주게 되는 것이다.

따라서

App.js의 코드를 그대로 home.js에 넣어주고 App.js는 우선 비워놓는다.

또한 아래와 같이 src밑에 추가적인 폴더를 만들어준다.



그리고 App.js에 router를 넣어주면 된다.

Router는 간단한 react componet이다.

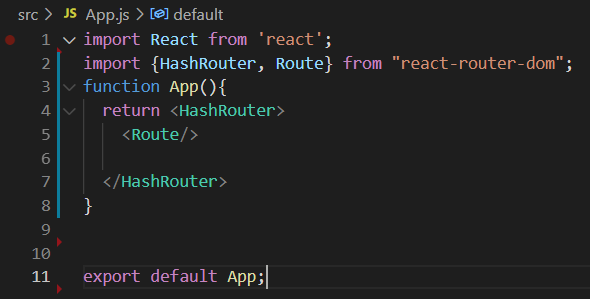
Router가 하는 역할은 매우 간단하다.

기본적으로 url을 가져와서 사용자의 명령에 따라 해당하는 component를 불러온다.

Router는 여러 종류가 있지만, 여기서는 hashrouter를 사용할 것이다.

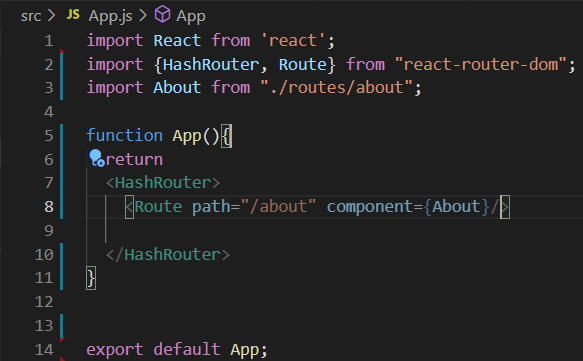
그리고

App function의 return으로 HashRouteer를 써주고 <Route/>를 HashRouter안에 넣어준다.



<Route/>안에는 중요한 props이 들어간다.

하나는 rendering할 화면이고 다른 하나는 어떤 task를 할지 정해주는 것이다.



Route를 위와 같이 작성했다면,

우리가 import한 about.js로 가서 About이라는 component를 rendering하라는 의미이다.

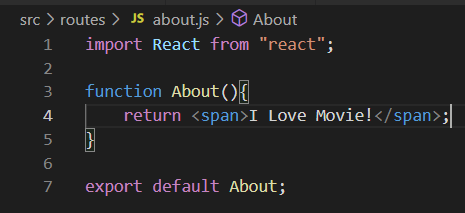
위처럼 작성하고 다시 창을 보면 url이 바뀌어 있다.



이렇게 말이다.

#은 그냥 hash값이라고 한다. 우리가 보고 싶은 창은 about.js의 About()을 실행한 창이다.

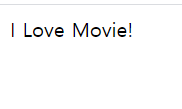
아래와 같은 about.js의 About()을 component로 받기 위해서는



url을 아래와 같이 작성해준다.

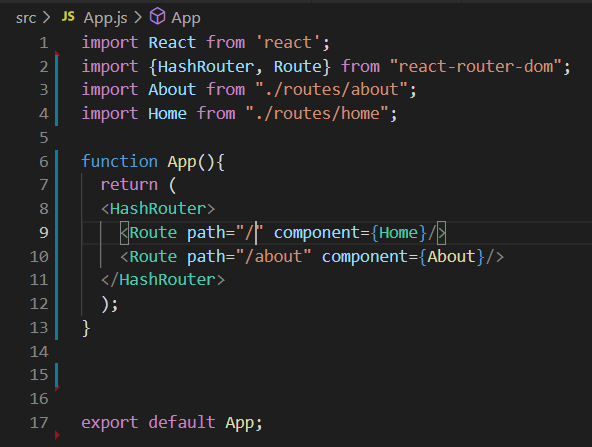


그러면 아래와 같이 원하는 결과를 볼 수 있다.



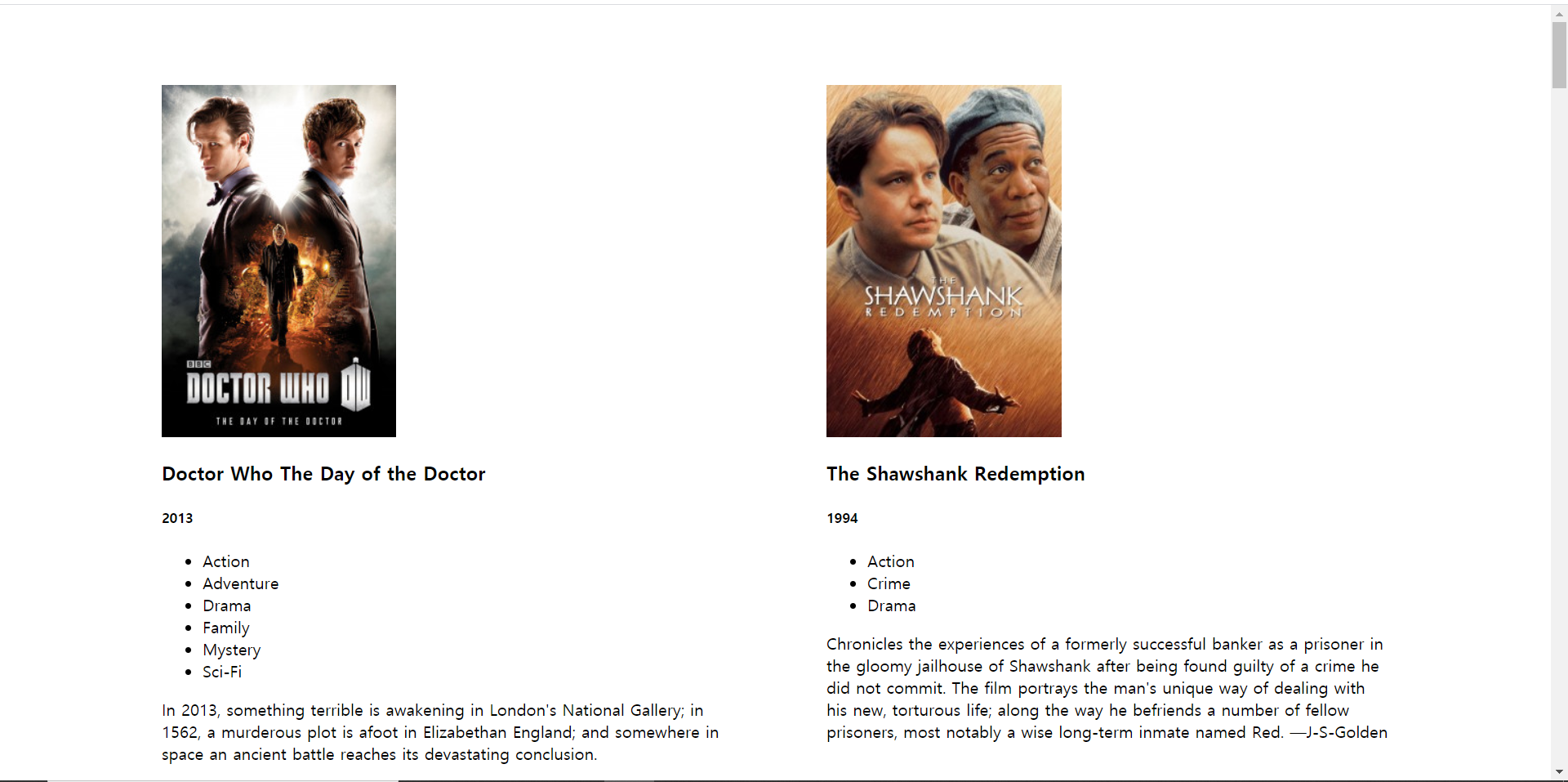
그렇다면 우리는 home으로 작성한 영화 목록 화면을 기본 값으로 두고 싶다.

따라서 추가적인 route를 하나 더 작성해준다.



이렇게 말이다.

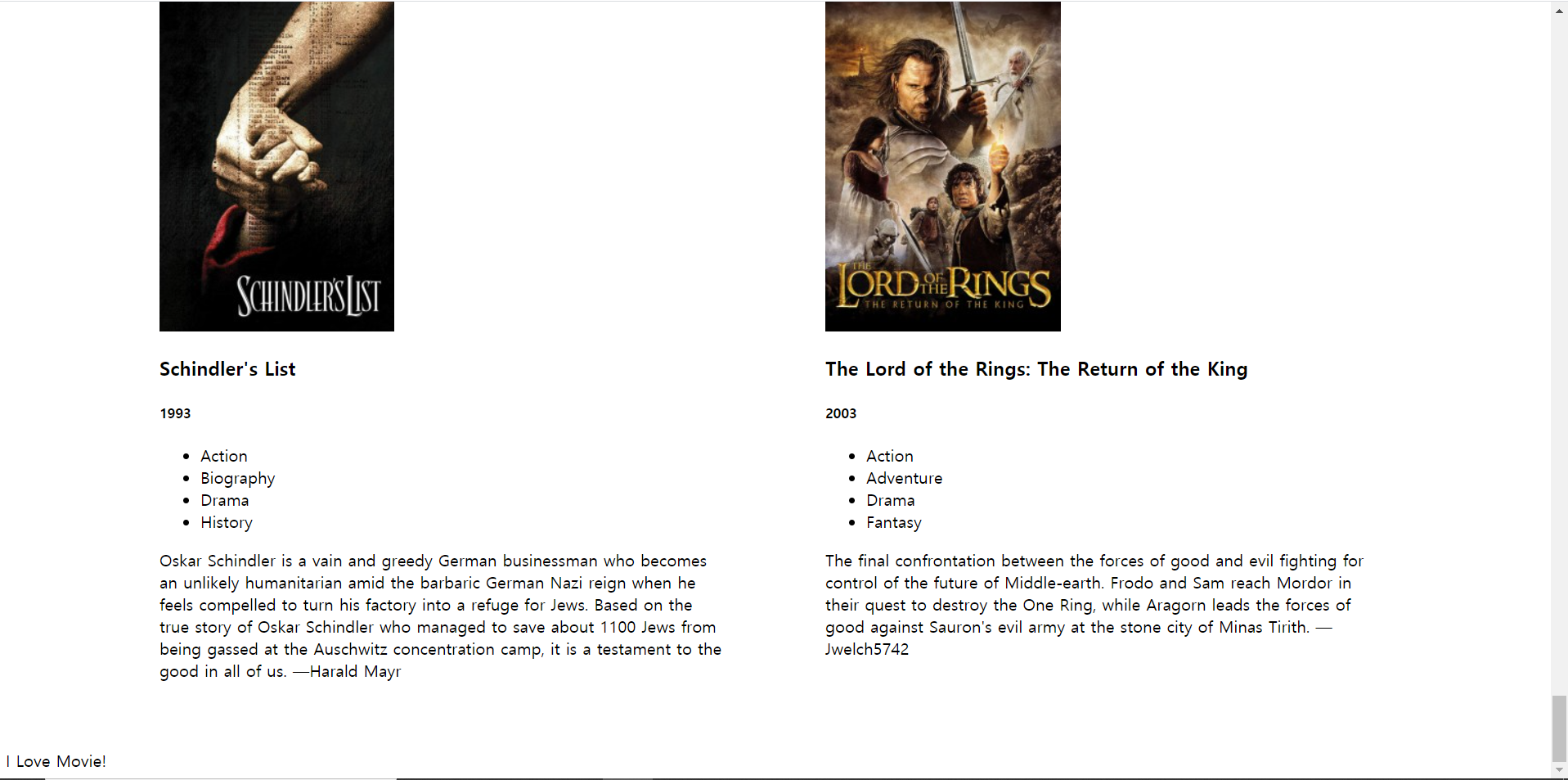
저장하고 다시 <http://localhost:3000/#/>로 들어가보면



원하는 창이 나온다

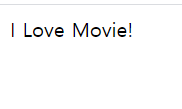
하지만 문제점이 하나 있다.

우리가 만든 about.js에 들어가면



이렇게 창이 나온다.

About은



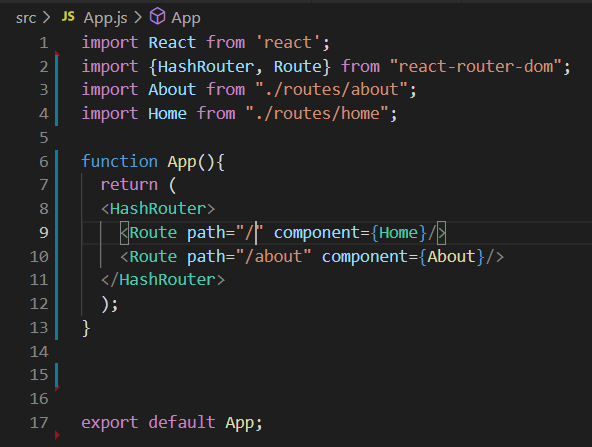
이렇게만 나와야 하는데 home이랑 겹쳐서 나온다는 것이다.

이것은 react의 url이 작동하는 방식과 연결되는 오류이다.

React는 기본적으로 url을 가져온다.

그 후 router에 있는 모든 url과 비교한다. 그리고 match되는 모든 url을 렌더링 한다.

다시 코드를 보자.



위에서 /about을 path로 설정한 것을 보면

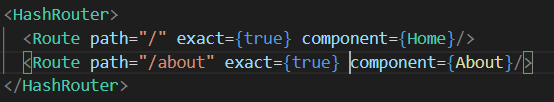
첫 번째 route에서 /와도 match되고 두 번째 route와도 일치한다.

따라서 두 개의 다른 component를 모두 화면에 띄우는 것이다.

이 오류를 방지하기 위해

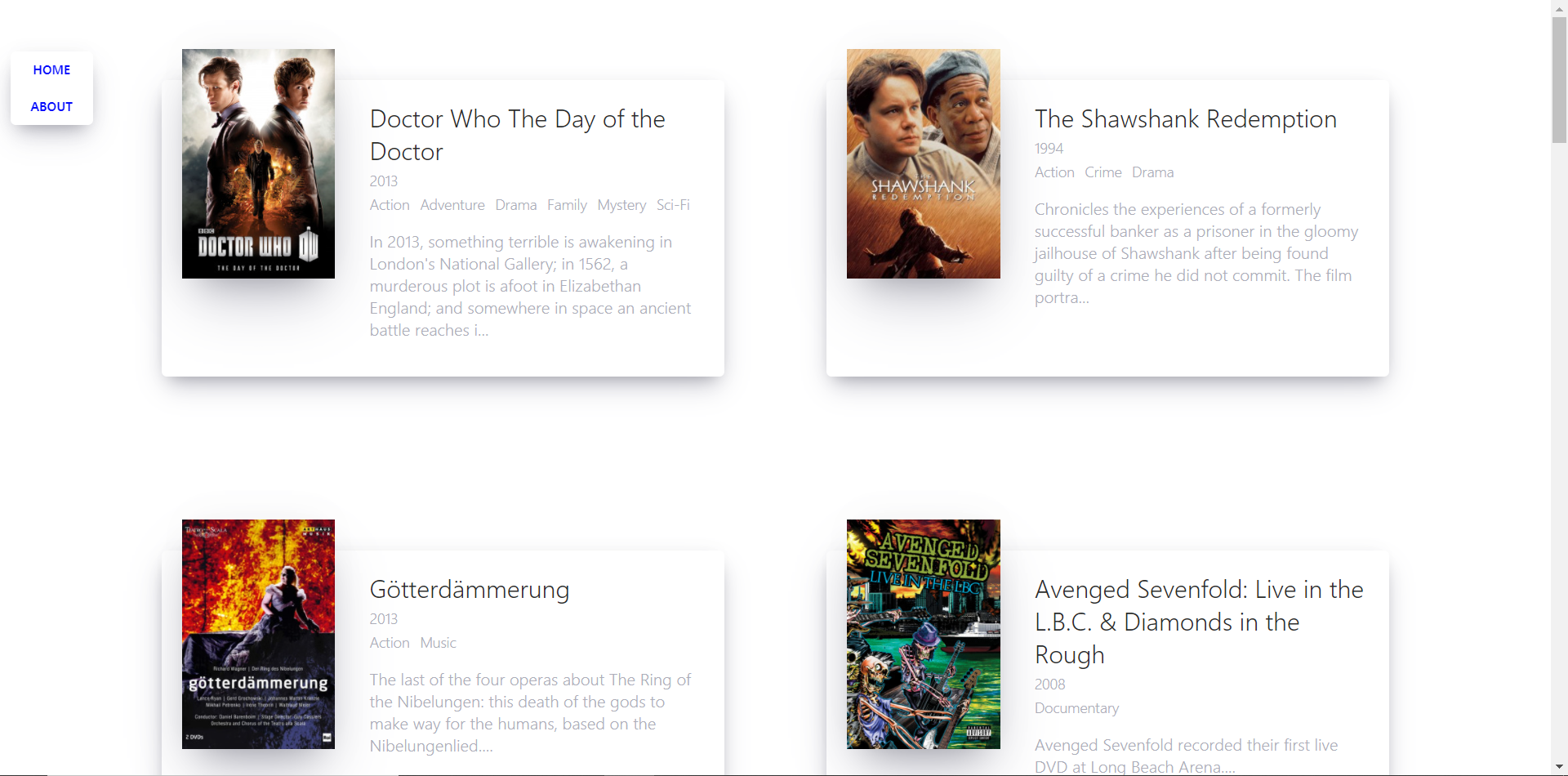
“exact true”를 추가해준다.

이를 추가하면, url이 일치할 때만 rendering을 해준다.



이렇게 말이다.

그러면 다른 두 개의 component가 동시에 rendering되는 현상을 방지할 수 있다.



현재상태이다. CSS를 고치는 부분이 참 어려웠다…

아무튼 이제 남은 것은 카드를 눌렀을 때 해당 영화에 관한 정보를 출력하도록 하는 것이다.

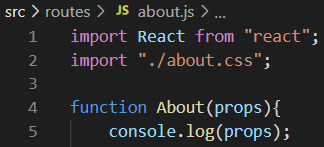
모든 component에는 props이 있다.

예를 들어, 우리가 작성했던 movies.js를 보면

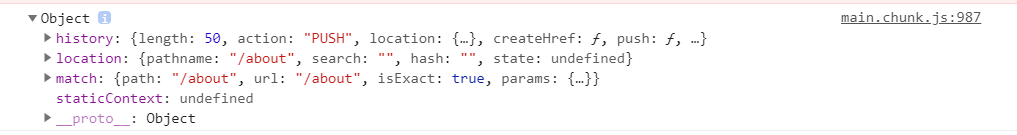


위 함수는 6개의 props를 매개변수로 사용하는 것이다.

About.js에도 prop이 있다.



위처럼 작성하고 페이지를 다시 보면,



를 볼 수 있는데, 위와 같은 사항들을 props로 가지고 있음을 뜻한다.

위 props들은 아직 about으로 전송된 것이 아니라,

React-router에 의해 넣어진 것이다.

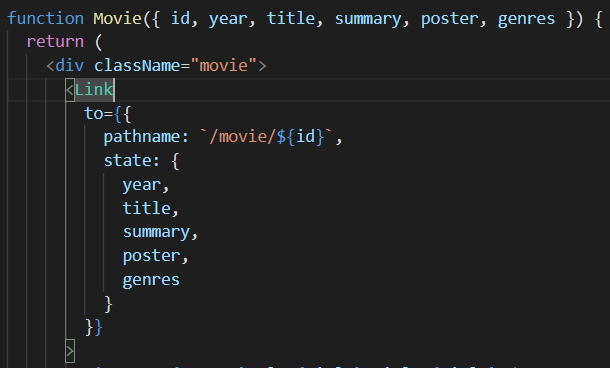
Router에 있는 모든 route는 기본값으로 props를 가진다.

Home page에서 movie 카드를 누르면 movie detail page로 넘어가게 할 것인데,

이때 해당 movie의 data를 모두 전송할 것이다.

그러기 위해서 movie.js를 수정해야 한다.

우선 코드부터 보자



Pathname(주소)는 각 영화의 id이고

클릭하게 되면 해당 주소에 모든 props를 전송하도록 했다.

이제 우리가 할 것은,

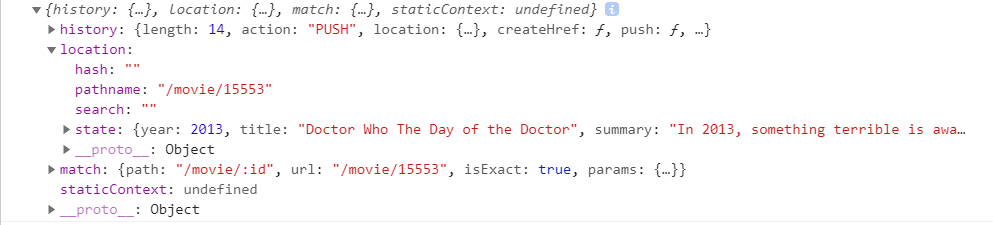
카드를 클릭하는 방법으로만 detail을 볼 수 있도록 하는 것이다.

그러기 위해서 route간의 props를 이용할 것이다.

Detail function을 만들어서

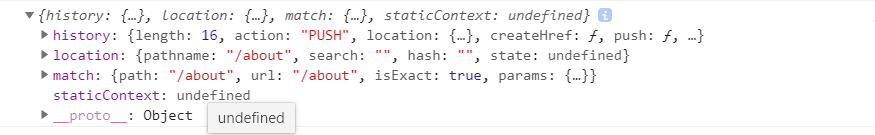
Console.log(props)

를 해보면 아래와 같이 출력된다.



즉, location-state에 영화에 대한 정보가 있다는 것이다.

하지만 영화카드를 클릭하는 것이 아닌, url을 통해서 movie detail에 접근하면



위와 같이 state가 undefined라고 나온다.

우리는 이처럼 클릭이 아닌 방법으로 접근하려고 했을 때 home page로 redirect하게 해주어야 한다.

이를 효과적으로 처리하기 위해 class component를 이용한다.

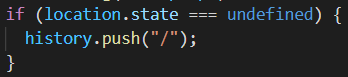
(이유는 자세히 모르겠다. 나중에 react hook 강의를 보면 알 수 있다고 한다.)



이전에 component life cycle 부분에서 componentDidMount()는 render()이전에 실행된다고 배웠다.

Render()하기 전에 props가 undefined이면, 즉 클릭이 아닌 다른 방법으로 접근하려고 하면 home으로 redirect하려는 것이 목적이다.

url을 변경해주어야 하는데, 이와 관련된 props가 “history”이다.



이렇게 history.push(“/”); 로 작성해주면 url을 home으로 변경할 수 있다.

만약 location.state가 undefined가 아니면 카드 클릭으로 detail에 접근하려는 것이다.

따라서 그 이후 render()가 실행된다.

나 같은 경우에는 영화 카드에 summary가 잘려 있어서 클릭하면 영화의 summary를 볼 수 있도록 작성했다.