API 연동 실습2

Contents

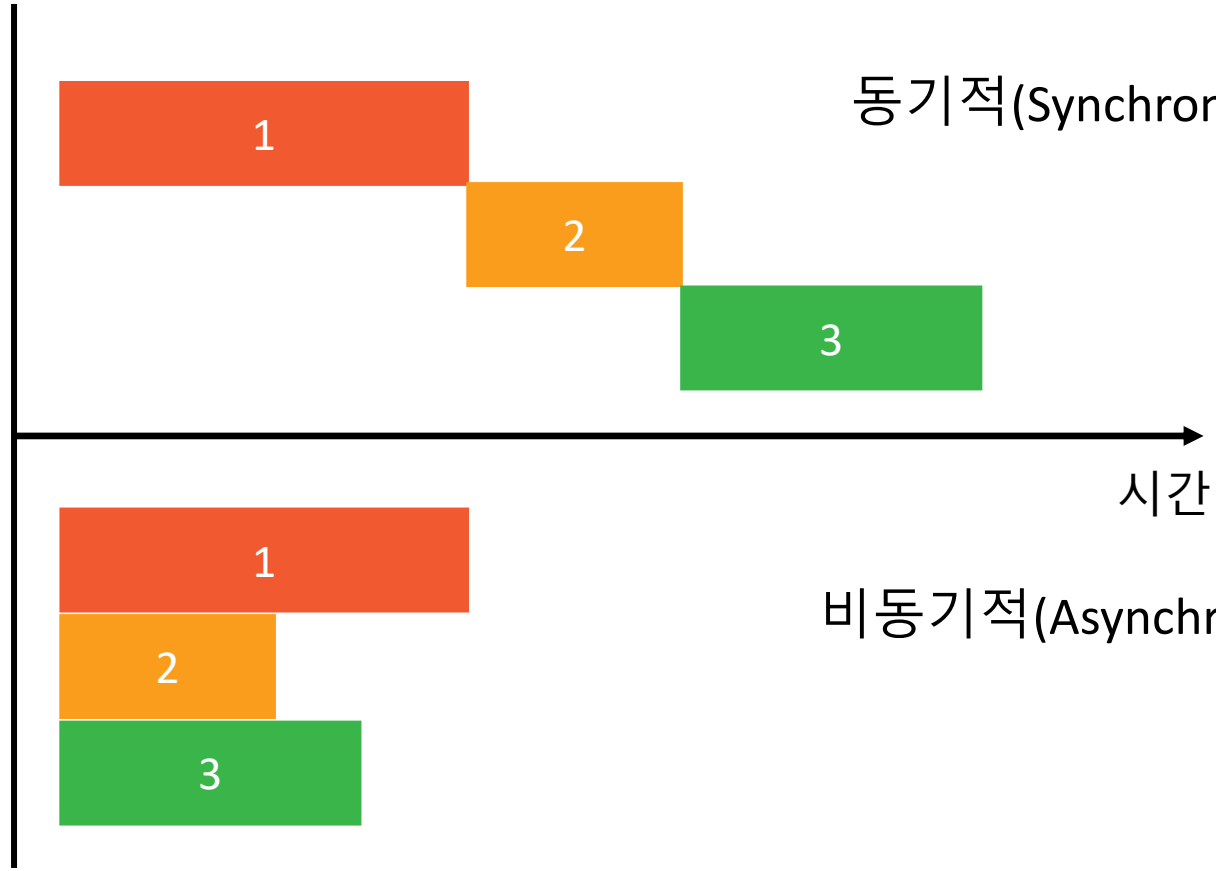
- API 연동 실습2 – movie list 조회

1. 비동기 처리
2. 스타일링 - CSS Module
3. react-query
4. ant design - UI 프레임웍
5. API – TMDB

(실습). 영화 앱 만들기

SECTION 1. 비동기(Asynchronous) 처리

◦ 동기적 / 비동기적 처리



동기적(Synchronous)

- JavaScript의 기본 동작 방식
  - 단일 스레드(싱글 스레드)
- 한 번에 하나의 작업을 수행

비동기적(Asynchronous)

- 어떠한 요청이 끝날 때까지 기다리지 않고, 응답에 관계없이 다음 작업이 실행되는 방식

SECTION 1. 비동기(Asynchronous) 처리

◦ 동기적 / 비동기적 처리

- 동기적 처리

```
function work() {
  const start = Date.now();
  for (let i = 0; i < 10000000; i++) {}
  const end = Date.now();
  console.log(end - start + 'ms');
}

console.log('작업 시작');
work();
console.log('다음 작업');
```

Console

```
"작업 시작"
"101ms"
"다음 작업"
```

```
function work() {
  const start = Date.now();
  for (let i = 0; i < 10000000; i++) {}
  const end = Date.now();
  console.log(end - start + 'ms');
}

console.log('작업 시작');
work();
console.log('다음 작업');
```

SECTION 1. 비동기(Asynchronous) 처리

◦ 동기적 / 비동기적 처리

- 비동기적 처리 - setTimeout()

```
function work() {
  setTimeout(() => {
    const start = Date.now();
    for (let i = 0; i < 10000000; i++) {}
    const end = Date.now();
    console.log(end - start + 'ms');
  }, 0);
}

console.log('작업 시작');
work();
console.log('다음 작업');
```

Console

"작업 시작"

"다음 작업"

"101ms"

```
function work() {
  setTimeout(() => {
    const start = Date.now();
    for (let i = 0; i < 10000000; i++) {}
    const end = Date.now();
    console.log(end - start + 'ms');
  }, 0);
}

console.log('작업 시작');
work();
console.log('다음 작업');
```

SECTION 1. 비동기(Asynchronous) 처리

◦ 비동기적 처리 - setTimeout()

- 비동기적 처리 - setTimeout() & callback

```
function work(callback) {
  setTimeout(() => {
    const start = Date.now();
    for (let i = 0; i < 1000000000; i++) {}
    const end = Date.now();
    console.log(end - start + 'ms');
    callback();
  }, 0);
}

console.log('작업 시작');
work(() => {
  console.log('작업 종료')
});
console.log('다음 작업');
```

Console

```
"작업 시작"
"다음 작업"
"101ms"
"작업 종료"
```

```
function work(callback) {
 setTimeout(() => {
   const start = Date.now();
   for (let i = 0; i < 1000000000; i++) {}
   const end = Date.now();
   console.log(end - start + 'ms');
   callback();
 }, 0);
}

console.log('작업 시작');
work(() => {
 console.log('작업 종료')
});
console.log('다음 작업');
```

SECTION 1. 비동기(Asynchronous) 처리

◦ 비동기적 처리가 필요한 작업

- Ajax Web API 요청 - 서버에서 데이터를 받아서 처리하는 경우(서버에서 응답을 받을 때까지 대기해야 하기 때문에 비동기 처리)
- 파일 읽기 - 서버 쪽에서 파일을 읽어야 하는 상황일 때
- 암호화/복호화 - 암/복호화에 시간이 어느 정도 걸리는 경우
- 작업 예약 - 어떤 작업을 몇초 후에 실행해야 하는 상황

SECTION 1. 비동기(Asynchronous) 처리

◦ Callback Hell (콜백 지옥)

- 비동기적으로 처리해야 하는 일이 많아질수록 코드의 깊아기 계속 깊어지는 현상

```
function increaseAndPrint(n, callback) {
  setTimeout(() => {
    const increased = n + 1;
    console.log(increased);
    if (callback) {
      callback(increased);
    }
  }, 1000);
}
```

```
increaseAndPrint(0, n => {
  increaseAndPrint(n, n => {
    increaseAndPrint(n, n => {
      increaseAndPrint(n, n => {
        increaseAndPrint(n, n => {
          console.log('끝!');
        });
      });
    });
  });
});
```

SECTION 1. 비동기(Asynchronous) 처리

◦ Promise

- 비동기 작업을 조금 더 편하게 처리할 수 있도록 ES6에 도입된 기능
- 성공, 실패를 처리하는 내장 함수를 제공 - resolve, reject

```
const myPromise = new Promise((resolve, reject) => {
  // 구현
})
```

- 성공 처리

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve(1);
  }, 1000);
});

myPromise.then(n => {
  console.log(n);
});
```

Console

1

SECTION 1. 비동기(Asynchronous) 처리

- **Promise**
  - 실패 처리

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject(new Error());
  }, 1000);
});

myPromise
  .then(n => {
    console.log(n);
  })
  .catch(error => {
    console.log(error);
  });
```

Console

```
[object Error] { ... }
```

SECTION 1. 비동기(Asynchronous) 처리

- **Promise**
  - Callback 대체

```
function increaseAndPrint(n) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const value = n + 1;
      if (value === 5) {
        const error = new Error();
        error.name = 'ValueIsFiveError';
        reject(error);
        return;
      }
      console.log(value);
      resolve(value);
    }, 1000);
  });
}
```

```
increaseAndPrint(0).then((n) => {
  console.log('result: ', n);
})
```

Console

```
1
"result: "
1
```

SECTION 1. 비동기(Asynchronous) 처리

◦ Promise

• Promise Chanining

```
function increaseAndPrint(n) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const value = n + 1;
      if (value === 5) {
        const error = new Error();
        error.name = 'ValueIsFiveError';
        reject(error);
        return;
      }
      console.log(value);
      resolve(value);
    }, 1000);
  });
}
```

```
increaseAndPrint(0)
  .then(n => {
    return increaseAndPrint(n);
  })
  .then(n => {
    return increaseAndPrint(n);
  })
  .then(n => {
    return increaseAndPrint(n);
  })
  .then(n => {
    return increaseAndPrint(n);
  })
  .catch(e => {
    console.error(e);
  });
```

Console

```
1
2
3
4
✖ [object Error] {
    name: "ValueIsFiveError"
  }
```

SECTION 1. 비동기(Asynchronous) 처리

- **Promise 장단점**
  - 작업의 개수가 많아져도 코드의 깊이가 깊어지지 않음
  - 에러가 몇 번째에서 발생했는지 찾기 어려움
  - 특정 조건에 따라 분기 처리에 어려움
  - 특정 값을 공유하면서 작업을 처리하기 어려움

SECTION 1. 비동기(Asynchronous) 처리

◦ async/await

- ES8에 추가된 기능으로 Promise를 쉽게 사용하게 해줌
- 구문과 구조가 표준 동기 함수를 사용하는 것과 비슷함
- Promise를 결과로 반환

```
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function process() {
  console.log('작업 시작');
  await sleep(1000);
  console.log('작업 종료');
}

process();
```

Console

"작업 시작"

"작업 종료"

- Promise의 앞부분에 await을 넣으면 promise가 끝날 때까지 기다렸다가 다음 작업 수행

SECTION 1. 비동기(Asynchronous) 처리

- **async/await**
  - async 함수는 promise를 반환

```
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function process() {
  console.log('작업 시작');
  await sleep(1000);
  console.log('작업 종료');
}

process().then(() => {
  console.log('process end');
});
```

Console

```
"작업 시작"
"작업 종료"
"process end"
```

SECTION 1. 비동기(Asynchronous) 처리

- **async/await**
  - 에러 처리 - try / catch

```
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function makeError() {
  await sleep(1000);
  const error = new Error();
  throw error;
}

async function process() {
  try {
    await makeError();
  } catch (e) {
    console.error(e);
  }
}

process();
```

```
Console

✖ [object Error] { ... }
```

SECTION 1. 비동기(Asynchronous) 처리

- **async/await**
  - 순차 처리

```
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

const getDog = async () => {
  await sleep(1000);
  return '강아지';
};

const getCat = async () => {
  await sleep(500);
  return '고양이';
};

const getRabbit = async () => {
  await sleep(2000);
  return '토끼';
};
```

```
async function process() {
  const dog = await getDog();
  console.log(dog);
  const cat = await getCat();
  console.log(cat);
  const rabbit = await getRabbit();
  console.log(rabbit);
}

process();
```

```
Console
  "강아지"
  "고양이"
  "토끼"
```

약 3.5초 소요

SECTION 1. 비동기(Asynchronous) 처리

- **async/await**
  - 동시 처리 - Promise.all

```
function sleep(ms) {
  return new Promise(resolve =>
    setTimeout(resolve, ms));
}

const getDog = async () => {
  await sleep(1000);
  console.log('강아지');
  return '강아지';
};

const getCat = async () => {
  await sleep(500);
  console.log('고양이');
  return '고양이';
};
```

```
const getRabbit = async () => {
  await sleep(2000);
  console.log('토끼');
  return '토끼';
};

async function process() {
  const results = await Promise.all([
    getDog(), getCat(), getRabbit()
  ]);
  console.log(results);
}

process();
```

Console

```
"고양이"
"강아지"
"토끼"
["강아지", "고양이", "토끼"]
```

약 2초 소요

SECTION 1. 비동기(Asynchronous) 처리

- **async/await**
  - 가장 빨리 끝난 작업 - Promise.race

```
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

const getDog = async () => {
  await sleep(1000);
  return '강아지';
};

const getCat = async () => {
  await sleep(500);
  return '고양이';
};

const getRabbit = async () => {
  await sleep(2000);
  return '토끼';
};
```

```
async function process() {
  const first = await Promise.race([
    getDog(), getCat(), getRabbit()
  ]);
  console.log(first);
}

process();
```

Console

"고양이"

SECTION 2. 스타일링

- **CSS Module**
  - 리액트 컴포넌트 파일에서 해당 CSS 파일을 불러올 때 클래스 이름이 모두 고유해짐(파일 경로, 파일 이름, 클래스 이름, 해쉬값 등이 사용됨)
  - 고유한 클래스 이름이 만들어지기 때문에, 실수로 다른 CSS 클래스 이름과 중복되는 것을 방지할 수 있음
  - CSS 파일의 확장자를 .module.css 로 저장

Box.js

```
import styles from './Box.module.css';

function Box() {
  return <div className={styles.box}>{styles.box}</div>
}

export default Box;
```

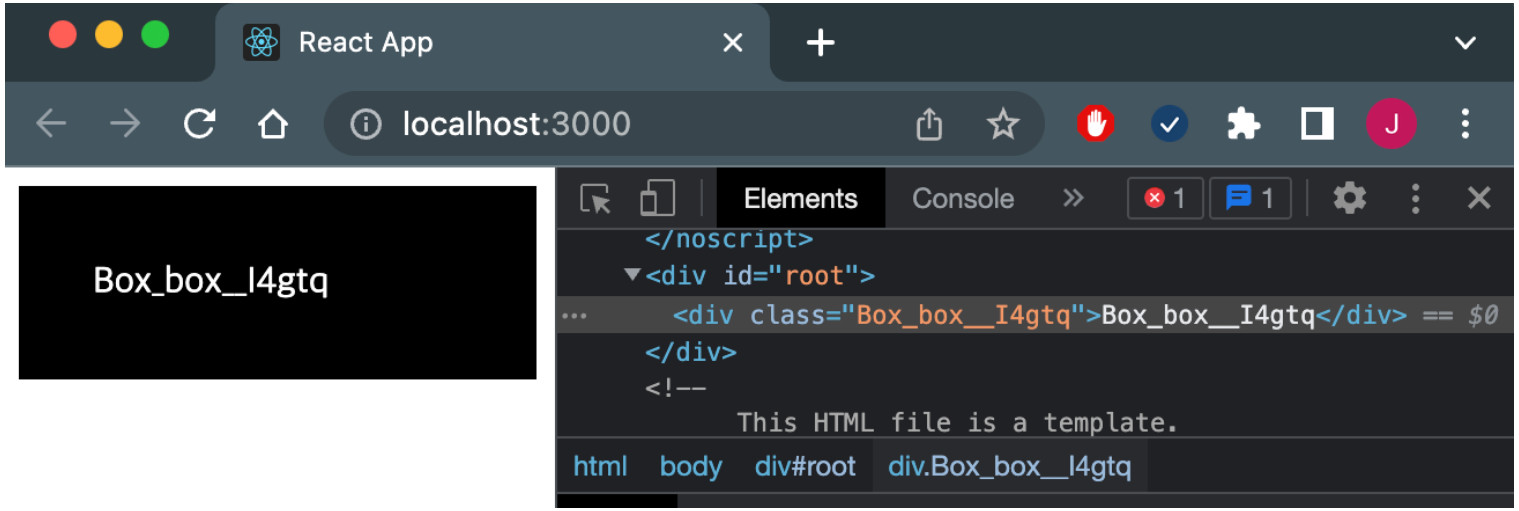
Box.module.css

```
.box {
  background: black;
  color: white;
  padding: 2rem;
}
```

SECTION 2. 스타일링

◦ CSS Module

* 실행 결과



- 레거시 프로젝트에 리액트를 도입할 때 (기존 CSS 클래스와 이름이 중복되지 않게 해줌)
- CSS 클래스 네이밍 규칙을 정하기 힘든 상황이거나 번거로울 때

SECTION 3. react-query

react-query

- 리액트 애플리케이션에서 서버 상태 가져오기, 캐싱, 동기화 및 업데이트를 보다 쉽게 다룰 수 있도록 도와주는 라이브러리

- 캐싱
- 백그라운드에서 오래된 데이터 업데이트 (데이터가 오래되었다고 판단되면 다시 get)
- 동일한 데이터에 대한 중복 요청을 단일 요청으로 통합 (동일 데이터를 여러 번 요청하면 한번만 요청)
- 데이터 업데이트를 가능한 빠르게 반영
- React hook과 사용 방법 유사

SECTION 3. react-query

- **react-query**
  - 사용법

```
import { QueryClient, QueryClientProvider } from '@tanstack/react-query'

const queryClient = new QueryClient();

function App() {
  return (
   <QueryClientProvider client={queryClient}>
      <div>블라블라</div>
   </QueryClientProvider>;
  );
}
```

- QueryClientProvider를 최상단에서 감싸줌
- QueryClient 인스턴스를 client props로 넣어 애플리케이션에 연결

SECTION 3. react-query

- **react-query api**
  - useQuery
    - 데이터를 get 하기 위한 api

```
const result = useQuery({
  queryKey,
  queryFn,
  // ...options ex) enabled, staleTime, ...
});
```

```
const getAllSuperHero = async () => {
  return await axios.get('http://localhost:4000/superheroes');
};

const { data, isLoading } = useQuery(['super-heroes'], getAllSuperHero);
```

SECTION 4. antd

- **Ant Design (https://ant.design/)**
  - 완성도 높은 UI 프레임웍. 다양한 테마와 components 제공
    - Pagination

```
import { Pagination } from 'antd';

<div>
  <Pagination
    defaultCurrent={page}
    total={total}
    defaultPageSize={10}
    onChange={(page) => setPage(page)}
  />
</div>
```

SECTION 5. API

◦ TMDB API (https://developer.themoviedb.org/)

- 영화, TV 프로그램 정보를 제공해주는 API. 무료로 운영
- 회원 가입 후 api key를 발급받아 사용

• API Reference : MOVIE LISTS > Now Playing

```
https://api.themoviedb.org/3/movie/now_playing?language=ko&page=1&region=KR
```

• api token -

eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiI0NjE3YjU4NGFjYWM5MWMzYmFlMzUxNzMwZjg0MTUz
MiIsInN1YiI6IjY0NmRkNjkyYzM1MTRjMmIwNzQyYjIwOSIsInNjb3BlcyI6WyJhcGlfcmVhZCJdLC
J2ZXJzaW9uIjoxfQ.ZF0rIJ-Wijjy_ka_c7zw3zjUhXgEmyxAsVZufRk2r4Y

SECTION 5. API

- **Now Playing API Test**

RESPONSE 200 TRY IT

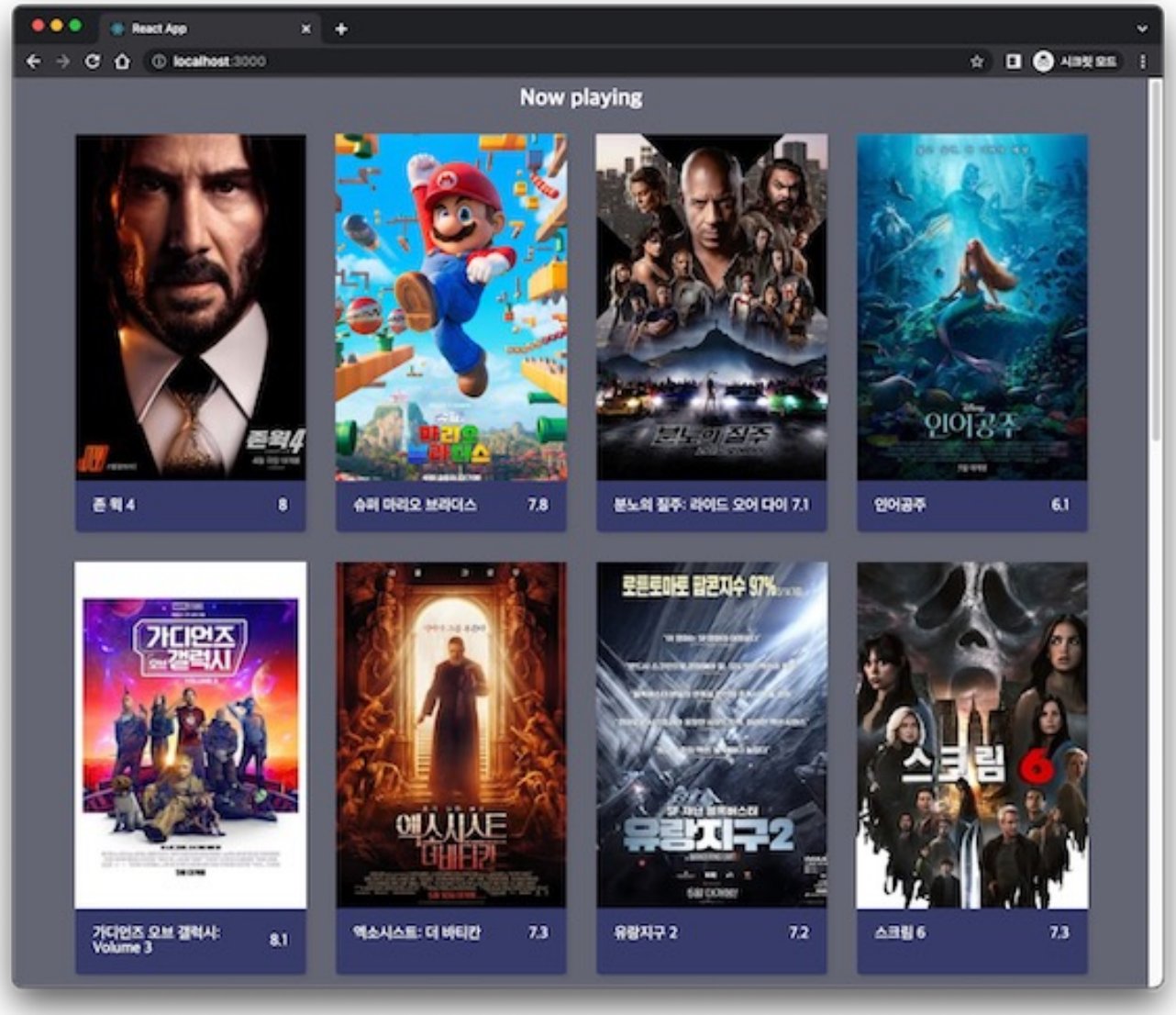

```
"page": 1,
"results": [
  {
    "adult": false,
    "backdrop_path": "/h8gHn0OzBoaefsYseUByqsmEDMY
    "genre_ids": [
      28,
      53,
      80
    ],
    "id": 603692,
    "original_language": "en",
    "original_title": "John Wick: Chapter 4",
    "overview": "죽을 위기에서 살아난 존 윅은 최고 회의를 쓰
    "popularity": 6448.501,
    "poster_path": "/9WF6TxCYwdiZw51NM92ConaQz1w.j
    "release_date": "2023-04-12",
    "title": "존 윅 4",
    "video": false,
    "vote_average": 8,
    "vote_count": 2322
  },
```

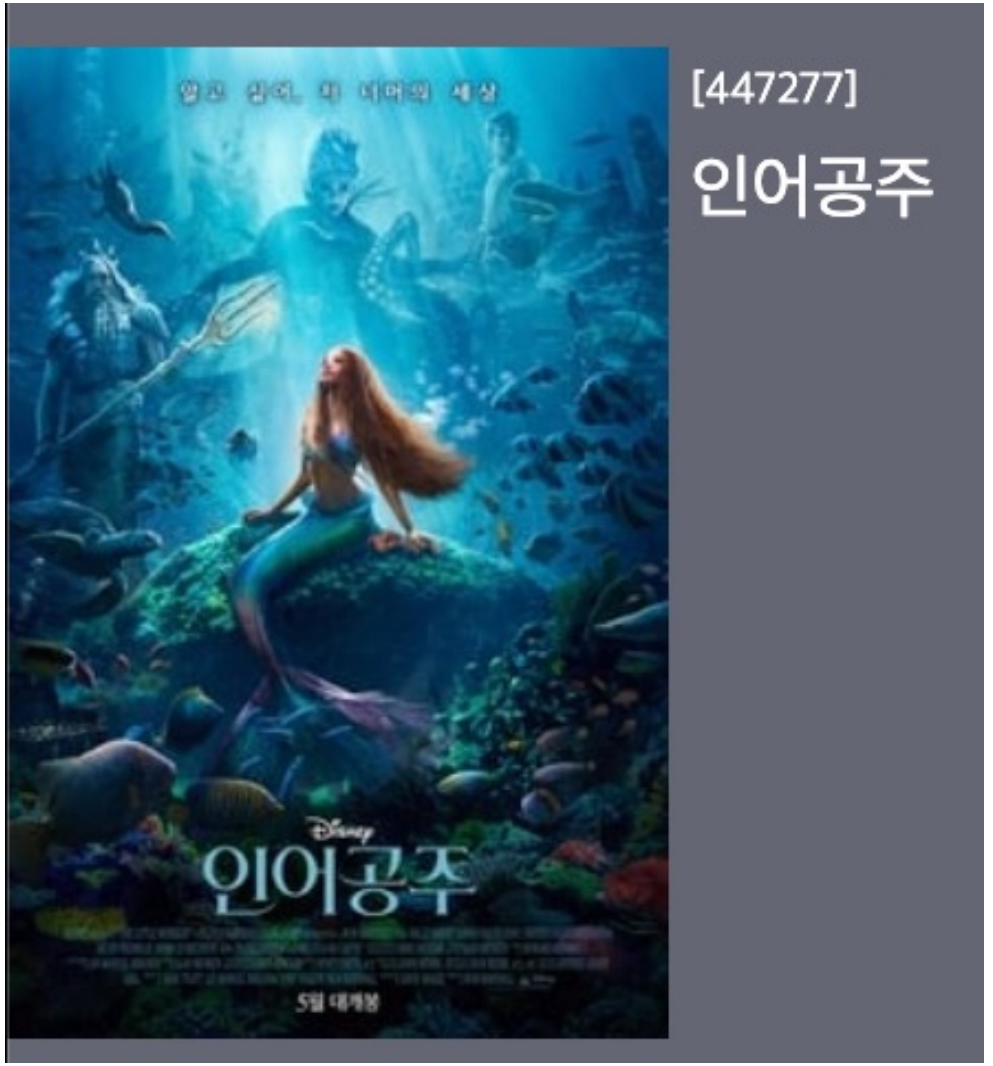

id: 영화 ID
title: 영화 제목
poster_path: 포스터 이미지 경로
vote_average: 평점

total_results: 전체 데이터 건수

PRACTICE. 영화 앱 만들기

◦ 실행 결과





PRACTICE. 영화 앱 만들기

◦ 프로젝트 생성 & 라이브러리 설치

- 프로젝트 생성

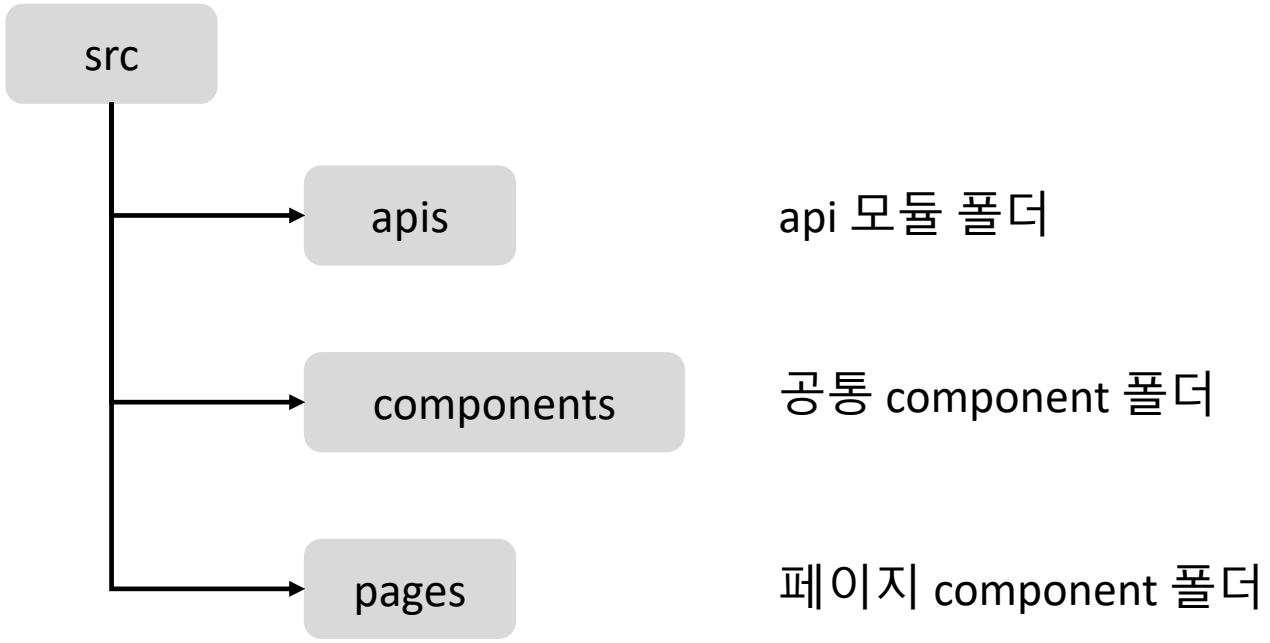
  npx create-react-app movie-app

- 라이브러리
  - npm install react-router-dom - page route
  - npm install @tanstack/react-query - caching
  - npm install axios - api call
  - npm install antd - pagination

```
"dependencies": {
  "@tanstack/react-query": "^4.29.12",
  "@testing-library/jest-dom": "^5.16.5",
  "@testing-library/react": "^13.4.0",
  "@testing-library/user-event": "^13.5.0",
  "antd": "^5.5.2",
  "axios": "^1.4.0",
  "react": "^18.2.0",
  "react-dom": "^18.2.0",
  "react-router-dom": "^6.11.2",
```

PRACTICE. 영화 앱 만들기

- **프로젝트 폴더 구성**



PRACTICE. 영화 앱 만들기

- apis/index.js – 영화 api call

```
import axios from 'axios';

const BASE_URL = 'https://api.themoviedb.org/3';
const API_KEY = '4617b584acac91c3bae351730f841532';

const getNowPlayingMovies = async (page) => {
  try {
    const {data} = await axios({
      method: 'get',
      url: BASE_URL + '/movie/now_playing',
      params: {
        api_key: API_KEY,
        language: 'ko',
        region: 'KR',
        page: page,
      },
    });
    return data;
  } catch (e) {
    throw Error(e);
  }
}

export { getNowPlayingMovies }
```

API_KEY : 4617b584acac91c3bae351730f841532

PRACTICE. 영화 앱 만들기

- **components/Movie.module.css – 영화 정보 CSS**

```css
.container {
  display: flex;
  flex-direction: column;
  width: 250px;
  margin: 16px;
  background-color: #373b69;
  color: white;
  border-radius: 5px;
  box-shadow: 3px 3px 5px rgba(0, 0, 0, 0.1);
  font-size: 16px;
}

.container img {
  max-width: 100%;
}

.info {
  display: flex;
  padding: 20px;
  justify-content: space-between;
  align-items: center;
}

.info span:last-child {
  margin-left: 3px;
}
```

```css
.container {
  display: flex;
  flex-direction: column;
  width: 250px;
  margin: 16px;
  background-color: #373b69;
  color: white;
  border-radius: 5px;
  box-shadow: 3px 3px 5px rgba(0, 0, 0, 0.1);
  font-size: 16px;
}

.container img {
  max-width: 100%;
}

.info {
  display: flex;
  padding: 20px;
  justify-content: space-between;
  align-items: center;
}

.info span:last-child {
  margin-left: 3px;
}
```

PRACTICE. 영화 앱 만들기

- **components/Movie.jsx – 영화 정보**

```jsx
import React from 'react'
import { useNavigate } from 'react-router-dom';
import styles from './Movie.module.css';

export const IMG_BASE_URL = 'https://image.tmdb.org/t/p/w220_and_h330_face';

export default function Movie(props) {
    const navigate = useNavigate();

    const onClickMovieItem = () => {
        navigate(`/movie/${props.id}`, {
            state: props,
        });
    };

    return (
        <div className={styles.container} onClick={onClickMovieItem}>
            <img src={IMG_BASE_URL + props.poster_path} alt="영화포스터" />
            <div className={styles.info}>
            <h4>{props.title}</h4>
            <span>{props.vote_average}</span>
            </div>
        </div>
    );
}
```

이동할 페이지에 데이터를 전달(두번째 인자)

페이지 이동:

```
const location = useLocation();
console.log(location.state.poster_path);
```

PRACTICE. 영화 앱 만들기

- pages/Movies.jsx – 영화 목록

```jsx
import { useQuery } from '@tanstack/react-query';
import { useState } from 'react';
import Movie from '../components/Movie';
import { getNowPlayingMovies } from '../apis';
import { Pagination } from 'antd';

function Movies() {
  const [page, setPage] = useState(1);
  const [total, setTotal] = useState(1);

  const { isLoading, error, data } = useQuery(
    ['nowPlayingMovies', page],
    () => getNowPlayingMovies(page),
    {
      onSuccess: (res) => {
        setTotal(res.total_results);
      },
    }
  );

  if (isLoading) {
    return <span>Loading...</span>;
  }
  if (error instanceof Error) {
    return <span>An error has occurred : {error.message}</span>;
  }
  return (
    <div>
      <div className='app-container'>
        {data.results.map((item) => {
          return (
            <Movie
              key={item.id}
              id={item.id}
              title={item.title}
              poster_path={item.poster_path}
              vote_average={item.vote_average}
            />
          );
        })}
      </div>
      <div className='app-pagination-wrap'>
        <Pagination
          defaultCurrent={page}
          total={total}
          defaultPageSize={20}
          onChange={(page) => setPage(page)}
        />
      </div>
    </div>
  );
}

export default Movies;
```

PRACTICE. 영화 앱 만들기

- pages/MovieDetail.jsx – 영화 상세 정보

```jsx
import React, { useEffect } from 'react'
import { useLocation, useParams } from 'react-router-dom'
import { IMG_BASE_URL } from '../components/Movie';

export default function MovieDetail() {
  const { id } = useParams();
  const { state } = useLocation();

  useEffect(() => {
    console.log('state', state);
  }, [state]);
```

```jsx
  return (
    <div className="page-container">
      <div style={{display: 'flex'}}>
        <img style={{width: '300px', height: '450px'}}
          src={IMG_BASE_URL + state.poster_path} alt='영화 포스터 이미지' />
        <div>
          <div style={{fontSize:'20px', color:'white', padding:'10px'}}>
            [{id}]
          </div>
          <div style={{fontSize:'32px', color:'white', padding:'10px'}}>
            {state.title}
          </div>
        </div>
      </div>
    </div>
  );
}
```

PRACTICE. 영화 앱 만들기

- **App.js – route 구성**

```
import { BrowserRouter, Route, Routes } from 'react-router-dom';
import MovieDetail from './pages/MovieDetail';
import Movies from './pages/Movies';

function App() {
  return (
    <div className="root-wrap">
      <BrowserRouter>
        <div className="header-title">Now playing</div>
        <Routes>
          <Route path='/' element={<Movies />} />
          <Route path='/movie/:id' element={<MovieDetail />} />
        </Routes>
      </BrowserRouter>
    </div>
  );
}

export default App;
```

index.css
reset.css
- src 폴더에 추가

PRACTICE. 영화 앱 만들기

- **index.js – react-query 사용**

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";

import './index.css';
import App from './App';

const queryClient = new QueryClient();

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <QueryClientProvider client={queryClient}>
    <React.StrictMode>
      <App />
    </React.StrictMode>
  </QueryClientProvider>
);
```