09-movie-rank

영화진흥위원회 박스오피스 OpenAPI 연동 예제

#01. 프로젝트 생성

```
yarn create react-app 09-movie-rank
```

1) 추가 패키지 설치

프로젝트를 VSCode로 열고, `Ctrl` + `~`를 눌러 터미널 실행

지금까지 살펴본 기본 패키지

```
yarn add react-router-dom qs react-helmet node-sass styled-components axios dayjs
yarn add redux react-redux redux-actions redux-devtools-extension redux-logger redux-thunk @reduxjs/toolkit
```

이번 단원에서 다룰 새로운 패키지

```
yarn add chart.js react-chartjs-2 react-loader-spinner react-helmet-async
```

#02. 프로젝트 초기화

1) 불필요한 파일 삭제

1. **src폴더** 하위에서 App.css와 index.css, logo.svg 삭제
2. **App.js** 파일에서 App.css와 logo.svg에 대한 참조(import) 구문 제거
3. **index.js** 파일에서 index.css에 대한 참조(import) 구문 제거

2) `/src/store.js`

폴더와 파일을 직접 생성해야 함.

```
import { configureStore, getDefaultMiddleware } from '@reduxjs/toolkit';
import { createLogger } from 'redux-logger';

// Slice 오브젝트 참조 구문 명시 위치

const logger = createLogger();

const store = configureStore({
    // 개발자가 직접 작성한  Slice 오브젝트들이 명시되어야 한다.
    reducer: {
        name: object,
        name: object
        ...
    },
    // 미들웨어를 사용하지 않을 경우 이 라인 생략 가능
    middleware: [...getDefaultMiddleware({serializableCheck: false}), logger],
    // redux-devtools-extension을 사용하지 않을 경우 false 혹은 이 라인 명시 안함
    devTools: true
```

```
});

export default store;
```

3) `/src/index.js`

패키지 참조

```
import { BrowserRouter } from 'react-router-dom';
import { Provider } from 'react-redux';
import store from './store';
```

랜더링 처리

```
ReactDOM.render(
    <React.StrictMode>
        <Provider store={store}>
            <BrowserRouter>
                <App />
            </BrowserRouter>
        </Provider>
    </React.StrictMode>,
    document.getElementById('root')
);
```

4) `/src/app.js`

패키지 참조

```
import { NavLink, Routes, Route } from "react-router-dom";
```

혹은

```
import { Link, Routes, Route } from "react-router-dom";
```

Route 처리

```
<Routes>
    <Route path='url정의' element={<컴포넌트명/>} />
</Routes>
```

링크걸기

```
<NavLink|NavLink to="url정의">링크텍스트</NavLink|Nav>
```

5) Slice 모듈 작성

a) 동기처리를 수행하는 경우

```
import { createSlice } from '@reduxjs/toolkit'

const slice이름 = createSlice({
    name: 'slice별칭',
```

```
    initialState: {
        // 이 모듈이 관리하고자하는 상태값들을 명시
        변수1: 100,
        변수2: 200
    },

    reducers: {
        // 상태값을 갱신하기 위한 함수들을 구현
        // 컴포넌트에서 이 함수들을 호출할 때 전달되는 파라미터는 action.payload로 전달된다.
        // initialState와 동일한 구조의 JSON을 리턴한다.
        액션함수1: (state, action) => {...state},
        액션함수2: (state, action) => {...state}
    },
});

// 액션함수들 내보내기
export const { 액션함수1, 액션함수2 } = slice이름.actions;

// 리듀서 객체 내보내기
export default slice이름.reducer;
```

b) 비동기처리를 수행하는 경우

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit'
import axios from 'axios';

/** 비동기 처리 함수 구현 */
// payload는 이 함수를 호출할 때 전달되는 파라미터.
export const 액션함수 = createAsyncThunk("액션함수별칭", async (payload, { rejectWithValue }) => {
    let result = null;

    try {
        result = await axios.get(URL및 파라미터);
    } catch (err) {
        // 에러 발생시 `rejectWithValue()`함수에 에러 데이터를 전달하면 extraReducer의 rejected 함수가 호출된다.
        result = rejectWithValue(err.response);
    }

    return result;
});

/** Slice 정의 (Action함수 + Reducer의 개념) */
const slice이름 = createSlice({
    name: 'slice별칭',
    initialState: {
        /** 상태값 구조 정의 (자유롭게 구성 가능함) */
        rt: null,           // HTTP 상태 코드(200,404,500 등)
        rtmsg: null,        // 에러메시지
        item: [],           // Ajax 처리를 통해 수신된 데이터
        loading: false      // 로딩 여부
    },
    // 내부 action 및 동기 action (Ajax처리시에는 사용하지 않음)
    reducers: {},
    // 외부 action 및 비동기 action
    extraReducers: {
        /** Ajax요청 준비 */
        [액션함수.pending]: (state, { payload }) => {
            // state값을 적절히 수정하여 리턴한다.
            return { ...state, loading: true }
        },
        /** Ajax 요청 성공 */
        [액션함수.fulfilled]: (state, { payload }) => {
            // state값을 적절히 수정하여 리턴한다.
            return {
                ...state,
```

```
                rt: payload.status,
                rtmsg: payload.statusText,
                item: payload.data,
                loading: false
            }
        },
        /** Ajax 요청 실패 */
        [액션함수.rejected]: (state, { payload }) => {
            // state값을 적절히 수정하여 리턴한다.
            return {
                ...state,
                rt: payload?.status ? payload.status : '500',
                rtmsg: payload?.statusText ? payload.statusText : 'Server Error',
                item: payload.data,
                loading: false,
            }
        }
    },
});

// 리듀서 객체 내보내기
export default slice이름.reducer;
```

`/src/store.js` 파일에 정의한 Slice 모듈 명시

```
import { configureStore, getDefaultMiddleware } from '@reduxjs/toolkit';
import { createLogger } from 'redux-logger';

import slice이름 from './slices/MySlice';

const logger = createLogger();

const store = configureStore({
    // 개발자가 직접 작성한  Slice 오브젝트들이 명시되어야 한다.
    reducer: {
        slice별칭: slice이름,
        ...
    },
    // 미들웨어를 사용하지 않을 경우 이 라인 생략 가능
    middleware: [...getDefaultMiddleware(), logger],
    // redux-devtools-extension을 사용하지 않을 경우 false 혹은 이 라인 명시 안함
    devTools: true
});

export default store;
```

6) 컴포넌트에서 사용하기

a) 필요한 기능 참조하기

```
// 상태값을 로드하기 위한 hook과 action함수를 dispatch할 hook 참조
import { useSelector, useDispatch } from 'react-redux'
// Slice에 정의된 액션함수들 참조
import { 함수1, 함수2 } from '../slices/MySlice';
```

b) 컴포넌트 내부에서 hook을 통해 필요한 Object 생성

```
// hook을 통해 slice가 관리하는 상태값 가져오기
const {변수1, 변수2} = useSelector((state) => state.slice별칭);
```

```
// dispatch 함수 생성
const dispatch = useDispatch();
```

c) 필요한 이벤트 핸들러 안에서 액션함수 디스패치하기

Slice에서 정의한 액션함수의 `action.payload` 파라미터로 전달된다.

다수의 파라미터가 필요한 경우 JSON객체로 묶어서 전달한다.

```
dispatch(액션함수1(파라미터));
dispatch(액션함수2(파라미터));
```

#03. 컴포넌트 활용

1) `Helmet` 컴포넌트 변경사항

비동기 처리가 적용될 경우 컴포넌트에서 에러가 발생하기 때문에 Promise가 적용되어야 한다.

react-helmet-async 패키지를 사용하여 이를 해결할 수 있다.

```
import React from 'react';
import { Helmet, HelmetProvider } from 'react-helmet-async';

const Meta = (props) => {
    return (
        <HelmetProvider>
            <Helmet>
                ... <title>, <meta> 태그 적용 ...
            </Helmet>
        </HelmetProvider>
    );
};
```

2) 그래프 그리기 - react-chartjs-2

chartjs 라이브러리를 리액트에서 사용할 수 있도록 wrappering 한 패키지

Github

https://github.com/reactchartjs/react-chartjs-2

활용예제 (공식문서)

https://react-chartjs-2.js.org/examples/

#04. 영화진흥위원회 Open API

> http://www.kobis.or.kr/kobisopenapi/

영화 박스 오피스 데이터를 조회할 수 있다.

사이트 회원 가입, 로그인 후 **키 발급받기** 메뉴를 통해 연동키를 발급받은 후 사용한다.

일별 박스 오피스 API

`OPEN API > 제공 서비스` 메뉴를 통해 **일별 박스오피스 API 서비스** API 스팩 확인

> http://www.kobis.or.kr/kobisopenapi/homepg/apiservice/searchServiceInfo.do

요청 주소

> http://www.kobis.or.kr/kobisopenapi/webservice/rest/boxoffice/searchDailyBoxOfficeList.json

요청 변수

변수명	데이터 타입	설명
key	문자열(필수)	발급받은키 값을 입력합니다.
targetDt	문자열(필수)	조회하고자 하는 날짜를 yyyymmdd 형식으로 입력합니다.