REACT SPA 기반

TMDB API를 활용한 고성능 SPA 영화 정보 제공 사이트

김찬영

INDEX
목 차

진행 배경

REACT Project : TMDB를 활용한 SPA 영화 서비스

접근방식의 직관성이 떨어짐
영화 정보와 관련된 방대한 데이터를 제공하지만,
그 정보에 접근하는 방식이 직관적이지 않음

불필요한 정보 과도화
원하는 카테고리에 접근시 필요없는 데이터들도 같이 나와
가독성이 떨어짐

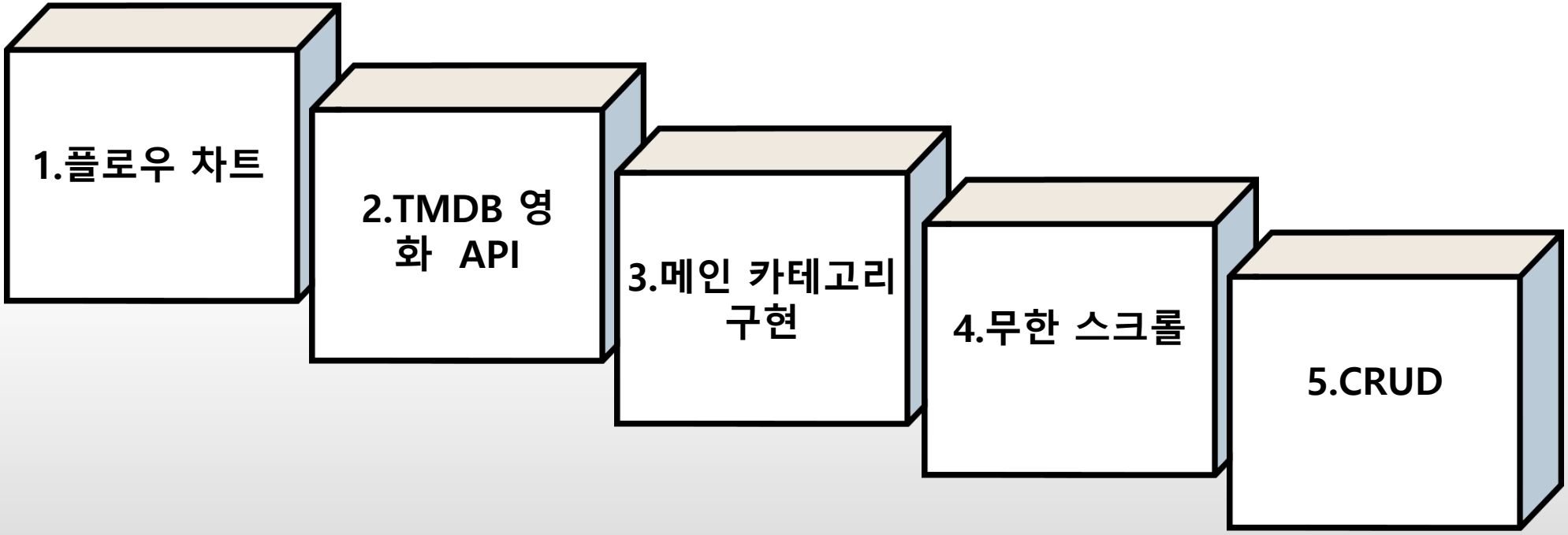
사용자 참여 유도 기능 필요
직관적인 평가기능이 없어 수치화하기 어려움

접근이 쉬운 카테고리 설정
접근이 쉬운 카테고리를 만들어
접근시 필요한 정보만 제공하는 페이지 생성

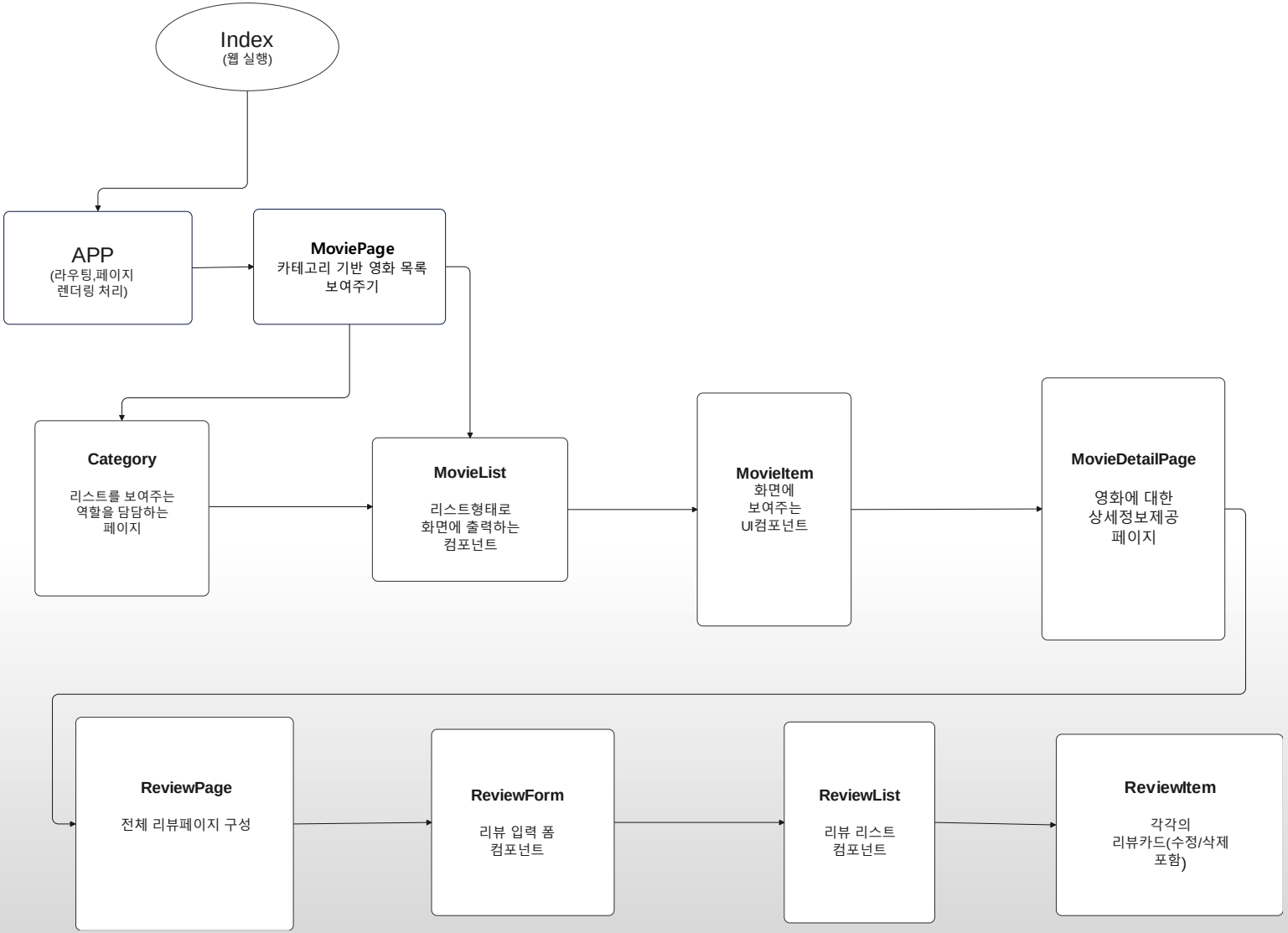
SPA를 활용한 새로운 페이지 구성 및 성능 최적화
리뷰게시판을 개선하고 동적 탐색이 필요한 페이지의
불편함을 무한스크롤을 통해개선

TMDB API 활용 페이지 구성
필요한 영화 데이터들을 수집

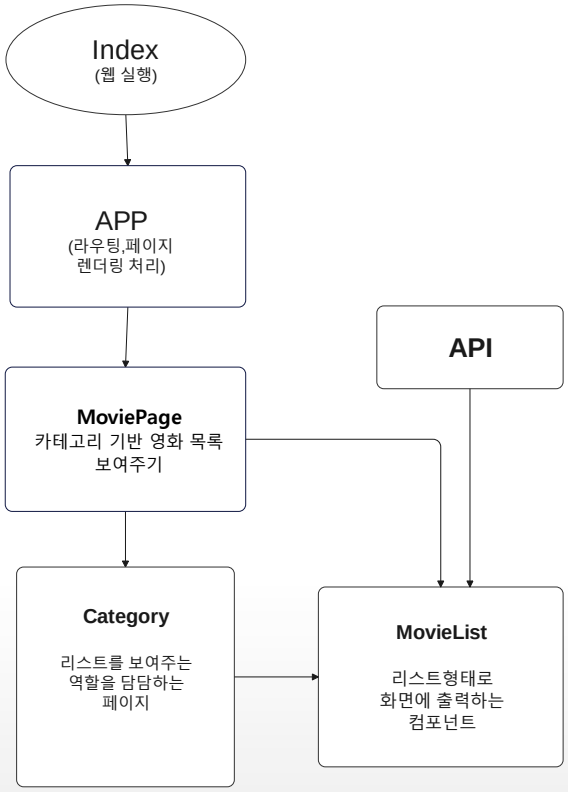
핵심 기능 설명 및 구현



1.플로우차트

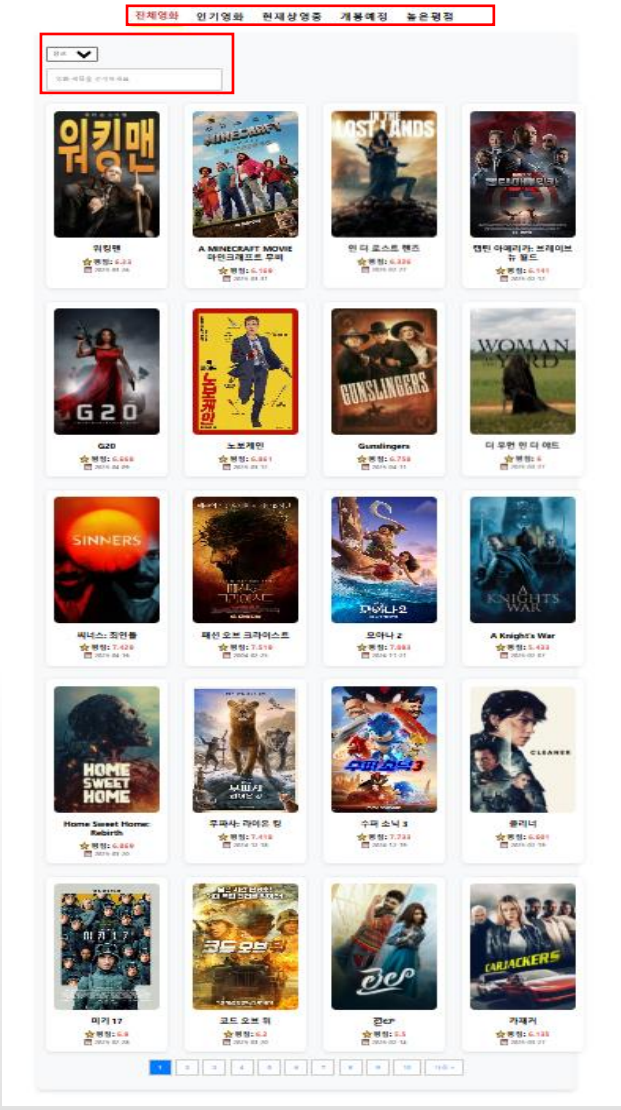

Index
(웹 실행)
APP
(라우팅,페이지
렌더링 처리)
MoviePage
카테고리 기반 영화 목록
보여주기
Category
리스트를 보여주는
역할을 담담하는
페이지
MovieList
리스트형태로
화면에 출력하는
컴포넌트
MovieItem
화면에
보여주는
UI컴포넌트
MovieDetailPage
영화에 대한
상세정보제공
페이지
ReviewPage
전체 리뷰페이지 구성
ReviewForm
리뷰 입력 폼
컴포넌트
ReviewList
리뷰 리스트
컴포넌트
ReviewItem
각각의
리뷰카드(수정/삭제
포함)

2.TMDB영화 API

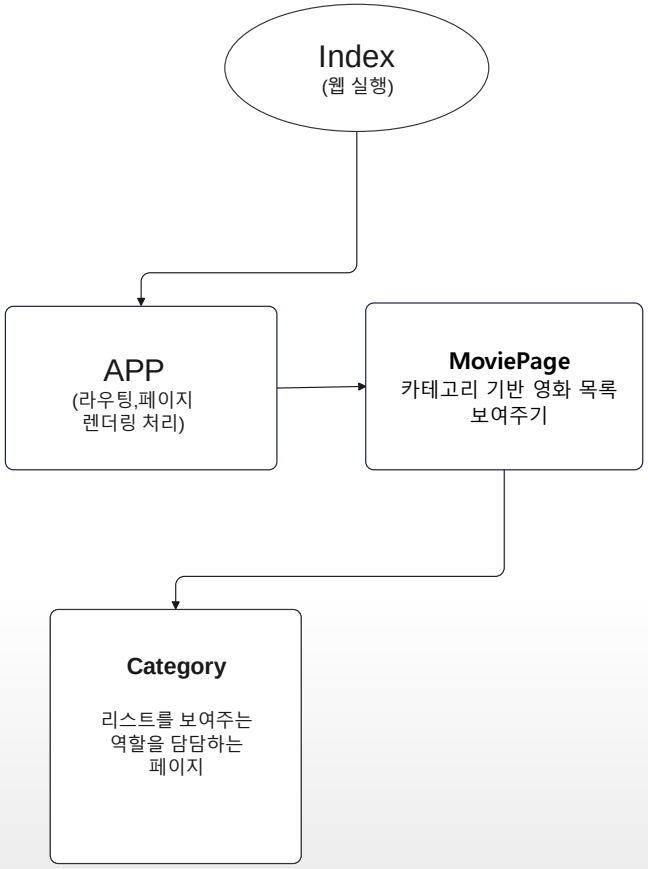



```
//카테고리 별 영화 API, 검색API,전체 영화API
useEffect(() => {
  const fetchData = async () => {
    setLoading(true);
    try {
      const scrollY = window.scrollY;
      let url = "";
      const params = {
        api_key: API_KEY,
        language: "ko-KR",
        page: currentPage,
      };

      if (searchTerm) {
        url = `https://api.themoviedb.org/3/search/movie`;
        params.query = searchTerm;
      } else if (category === "all") {
        url = `https://api.themoviedb.org/3/discover/movie`;
        params.sort_by = "popularity.desc";
        if (selectedGenre) {
          params.with_genres = selectedGenre;
        }
      } else {
        url = `https://api.themoviedb.org/3/movie/${category}`;
      }
```

```
useEffect(() => {
  const fetchGenres = async () => {
    try {
      const res = await axios.get(
        `https://api.themoviedb.org/3/genre/movie/list?api_key=${API_KEY}&language=ko-KR`
      );
      setGenres(res.data.genres);
```

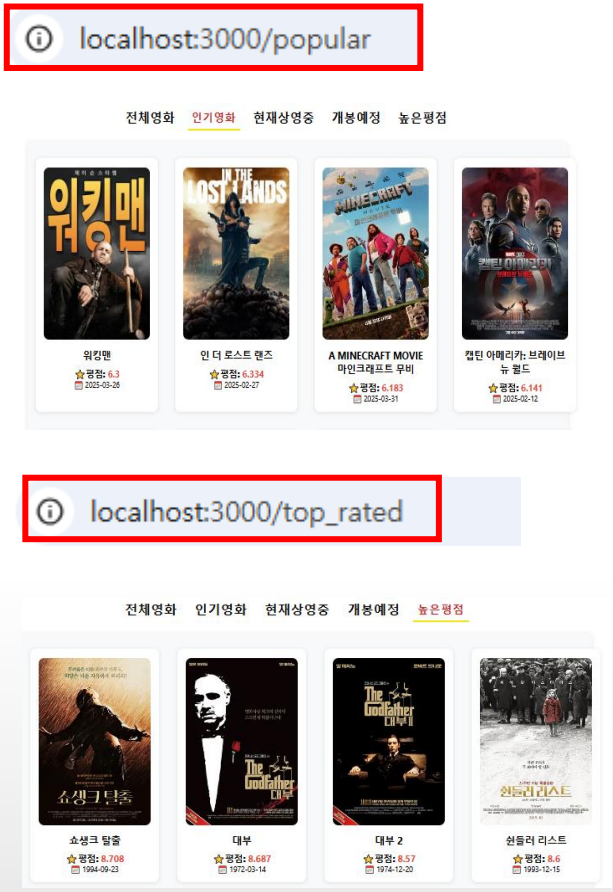


3.메인 카테고리 구현

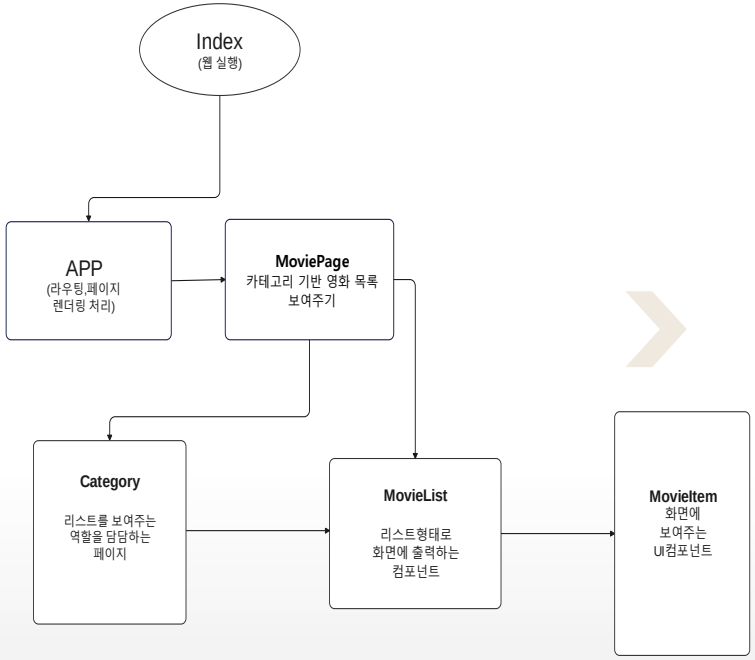


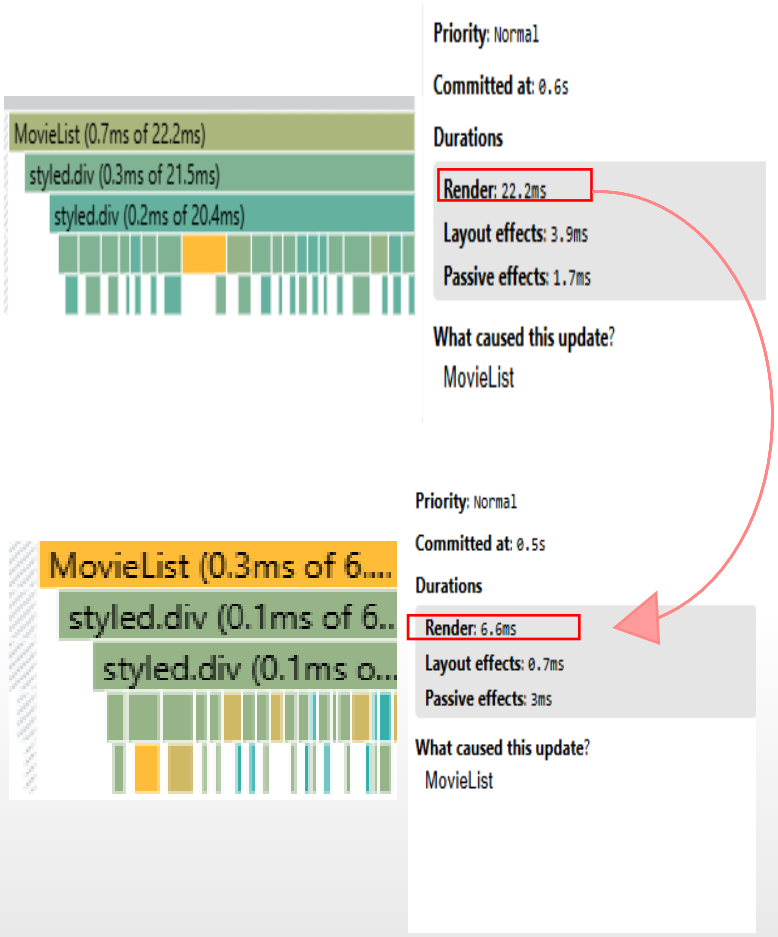

```jsx
import "./App.css";

import { Route, Routes } from "react-router-dom";
import MoviePage from "./pages/MoviePage";

import MovieDetailPage from "./pages/MovieDetailPage";

function App() {
  return (
    <Routes>
      <Route path="/" element={<MoviePage />}></Route>
      <Route path="/:category" element={<MoviePage />}></Route>
      <Route path="/movie/:id" element={<MovieDetailPage />} />
    </Routes>
  );
}

export default App;
```

```jsx
const Categories = () => {
  return (
    <CategoriesBlock>
      {categories.map((c) => (
        // c에담긴것:select됐을때 선택된애) name: 'popular',text: '인기영화',
        <Category
          key={c.name}
          className={({ isActive }) => (isActive ? "active" : undefined)}
          to={c.name === "all" ? "/" : `/${c.name}`}
        >
          {c.text}
        </Category>
      ))}
    </CategoriesBlock>
  );
};
```



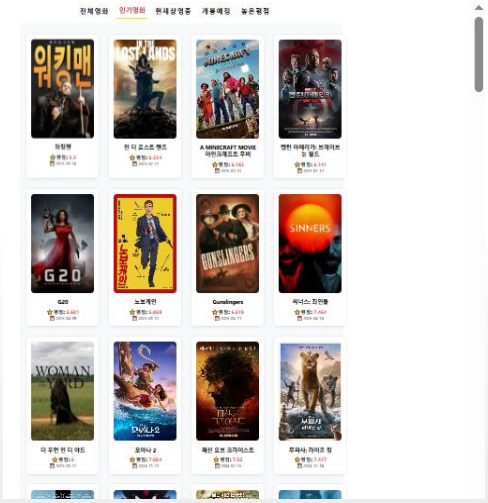
4.무한 스크롤





useCallback 함수 사용
이벤트핸들러(handleScroll, shouldFetchMore)가
메모이제이션되어 렌더링 최적화

상태 조건 분기 (loading,currentPage)
Fetch과다호출을 방지하여 데이터 병합 조건
관리 등으로 렌더링을 최소화



5.CRUD

ReivewPage
전체 리뷰 페이지 구성

ReivewForm
리뷰 입력 폼 컴포넌트

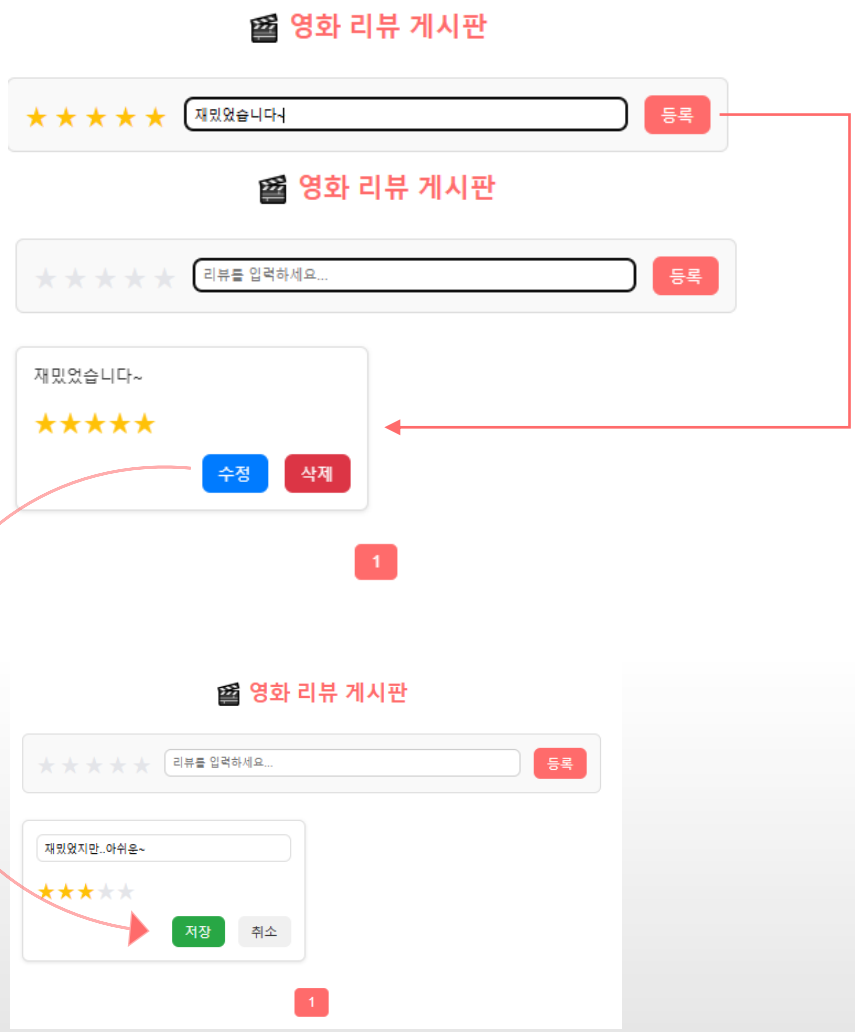
ReviewList
리뷰 데이터 저장
컴포넌트

ReviewItem
각각의 리뷰카드
(수정/삭제 포함)

```jsx
// 폼 컴포넌트
const ReviewForm = ({ onAdd }) => {
  const [text, setText] = useState("");
  const [rating, setRating] = useState(0);

  const handleSubmit = (e) => {
    e.preventDefault();
    if (text.trim() === "" || rating === 0) {
      alert("리뷰 내용과 별점을 모두 입력해주세요!");
      return;
    }

    onAdd({ text, rating });
    setText("");
    setRating(0);
  };
```

```jsx
const ReviewItem = ({ review, onUpdate, onDelete }) => {
  const [isEdit, setIsEdit] = useState(false);
  const [editText, setEditText] = useState(review.text);
```

```jsx
<Button className="edit" onClick={() => setIsEdit(true)}>
  수정
</Button>
<Button className="delete" onClick={() => onDelete(review.id)}>
  삭제
</Button>
```



구현 결과

SPA	React 기반의 SPA 구조로 개발하여 페이지 전환시 화면 전체가 아닌 필요한 데이터만 로딩되도록 하여 사용자가 직관적으로 영화 정보를 볼수 있도록 구현
성능 최적화	스크롤 이벤트처리에 대한 최적화를 통해 데이터 요청 시점을 적절히 조절하여 불필요한 리렌더링이나 과도한 API요청을 방지
CRUD	로컬 기반의 리뷰 게시판을 만들어 사용자와 원활한 의사소통이 가능하고 별점 기능을 추가하여 직관적인 평가 가능
영화정보API -TMDB	TMDB영화 목록 데이터를 가져와 검색 기능 및 장르선택 , 카테고리 목록에 맞게 구현

문제 해결

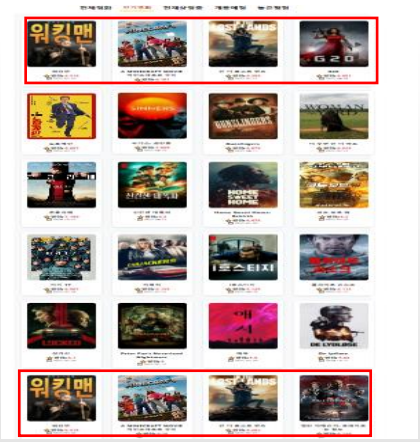
	문제점	해결방법
무한 스크롤 시 중복 영화 요청	스크롤 이벤트가 발생할 때 중복된 영화 API가 반복 호출되는 문제 발생	중복 호출을 막는 조건 처리와 shouldFetchMore() 함수를 사용하여 중복요청 방지
카테고리 변경	카테고리를 바꿀 때 이전 페이지나 검색어 상태가 남아있어 엉뚱한 결과 출력	useEffect 를 통해 category,searchTerm,selectedGenre가 바뀔때currentPage를 1로 초기화
검색	검색결과가 없을 때 다시'전체 목록'으로 돌아가는 기능 부재	handleGoBackToAllMovies 함수를 만들어 setCategory를 초기화 후 '전체영화'로 라우팅

문제 해결

무한 스크롤 시 중복 영화 요청

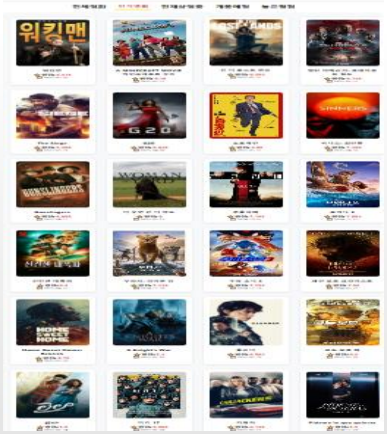
스크롤시 중복영화요청 방지

```
useEffect(() => {
  const handleScroll = () => {
    // 무한스크롤 조건 (이전 코드)
    if (
      window.innerHeight + window.scrollY >= document.body.offsetHeight - 500
    ) {
      setCurrentPage((prevPage) => prevPage + 1);
    }
  };

  window.addEventListener("scroll", handleScroll);
  return () => window.removeEventListener("scroll", handleScroll);
}, []);
```



```
const shouldFetchMore = useCallback(() => {
  return (
    window.innerHeight + window.scrollY >= document.body.offsetHeight - 500 &&
    !loading &&
    currentPage < totalPages &&
    category !== "all"
  );
}, [loading, currentPage, totalPages, category]);
```

(parameter) category: any



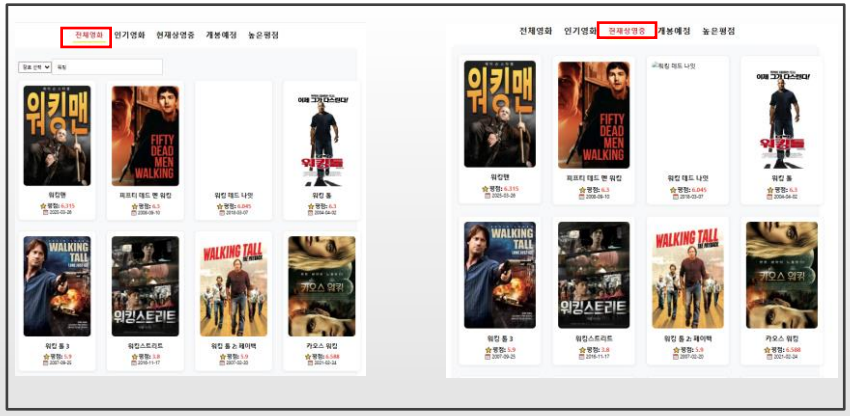
문제 해결

카테고리 변경

useEffect 를 통해 다른 카테고리에 중복요청이 되는 것을 방지

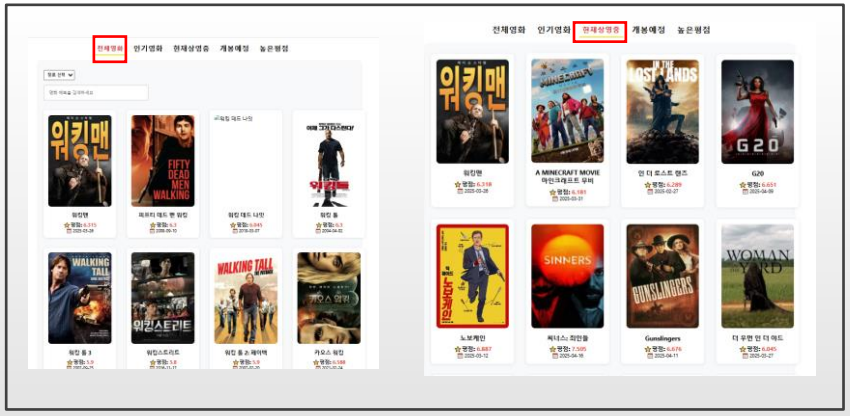
```
useEffect(() => {
  if (category === "all") {
    setSearchTerm("");
  }
  setSearchTerm("");
}, [category]);
```

전체영화목록과 현재 상영중영화 정보가 동일



```
useEffect(() => {
  setCurrentPage(1);
}, [category, searchTerm, selectedGenre]);
```

변경 후 데이터에 맞게 받아오기

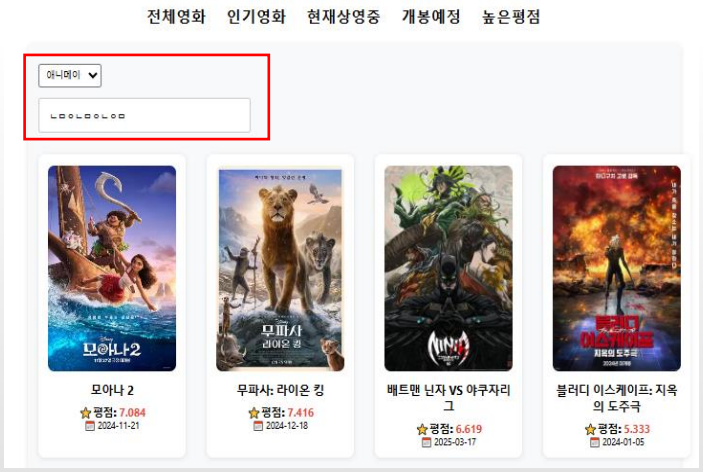


문제 해결

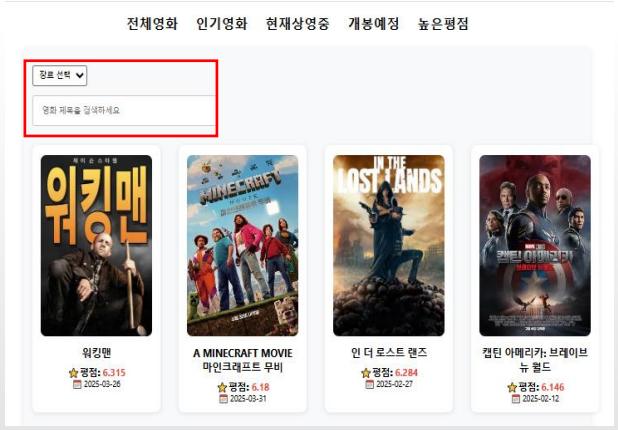
검색

검색결과 없을 시 확인버튼 클릭 후 이전 검색어,장르,페이지 값이 남지않도록handleGoBackToAllMovies 함수사용

```
const handleGoBackToAllMovies = useCallback(() => {
  setCategory("");
  setTimeout(() => {
    setCategory("all");
    navigate("/all", { replace: true });
  }, 0);
}, [setCategory, navigate]);
```




```
const handleGoBackToAllMovies = () => {
  setCategory("all");
  navigate("/all");
};
```



느낀 점

김찬영

프로젝트를 진행하면서 플로우를 신중하게 설계하는 것이 얼마나 중요한지
깨달았습니다. 초반에 구조를 잘 잡아두면 이후의 구현 흐름도 훨씬 유연해지고,
코드 작성이 수월해진다는 걸 경험했습니다.

또한, 불필요한 데이터가 중첩되어 생기는 문제는 성능 개선에 큰 방해가 된다는 점을
알게되어, 각각 상황의 맞는 렌더링 최적화 작업을 진행하여 문제를 개선해 나아갔습니다.

앞으로는 이러한 경험을 통해 코드를 작성하기 전 그림을 그리듯 구조를 구성하고
방향을 먼저 잡는 습관을 들이며, 이를 통해 구조 설계를 더욱 체계화하고 중요한 기능을
우선적으로 구현하는 연습으로 프로젝트의 완성도를 한층 더 높여갈 계획입니다.