강경규

프론트엔드

주소: 경기도 수원시 행궁로26번길 25-2

연락처: 010-8019-7134

이메일: qwzx16@naver.com

블로그: https://velog.io/@qwzx16/posts

깃허브: https://github.com/ganggyunggyu

프론트 엔드와 백엔드 로직을 함께 고민하기 위해 노력하고 있습니다.

TavaScript, React, Redux, React-Query, TailwindCSS 활용에 익숙합니다.

약 3년의 사회 생활과 6개월간 온라인 부트캠프 경험을 토대로 의사소통에 능숙합니다.

듣는 사람이 명확하고 이해하기 쉽게 설명을 하기 위해 다양한 스터디 활동을 하는 등의 노력을 기울이고 있습니다.

경력 사항

컬쳐커넥션 : 2024.06 ~ 2024.12 (6개월)

디자이너, 기획자 분들과 원활한 소통을 하며 프론트엔드 개발 업무를 진행하였습니다.

- 광주 비엔날레 미술관의 온라인 도슨트 서비스
  Vue.js를 이용한 프론트엔드 Service Deck 부분을 개발하였습니다.
  https://playar.syrup.co.kr/culture/index.html#/ready
- 파주 임진각의 다양한 명소에서 즐길수 있는 게임 및 온라인 도슨트 서비스
  Vue.js를 이용한 프론트엔드 Service Deck 부분을 개발하였습니다.
  https://playar.syrup.co.kr/culture/index.html#/paju
- 파주 임진각의 다양한 명소에서 즐길수 있는 게임 및 온라인 도슨트 서비스
  Vue.js를 이용한 프론트엔드 Service Deck 부분을 개발하였습니다.
  https://playar.syrup.co.kr/culture/index.html#/pin

기술 스택

Lang: TypeScript

FE: React, Vue, Nextjs, Redux, ReactQuery

BE: Nextjs, Expressjs, MongoDB

AWS: EC2, LoadBalancer, Route53, S3, CloudFront

프로젝트

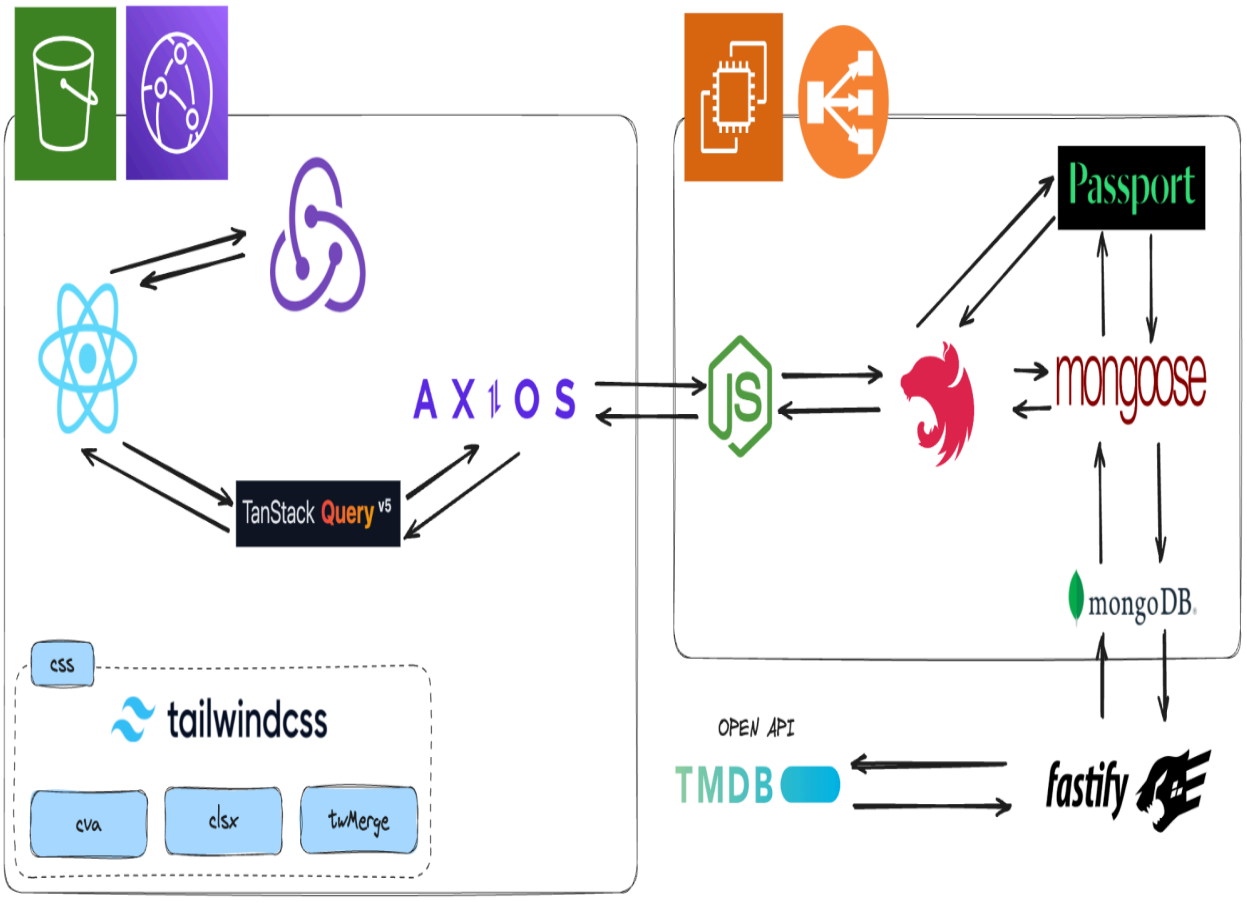
룸크리틱

- TMDB API를 활용해 영화 정보를 제공하고, 사용자 리뷰 기능을 구현한 영화 정보 서비스
- React와 Tailwind CSS로 UI 구현, Nest.js와 MongoDB를 활용한 인증/인가와 데이터 관리 로직 개발
- 서비스 링크: https://room-critic.online
- API 명세서: https://api.room-critic.online/api
- Github 링크

  https://github.com/ganggyunggyu/ROOMCRITIC_FE_V2

  https://github.com/ganggyunggyu/ROOMCRITIC_BE_V2

테크 & 인프라 아키텍처



프로젝트에서 배운 점

1.session 기반의 인증/인가 구현

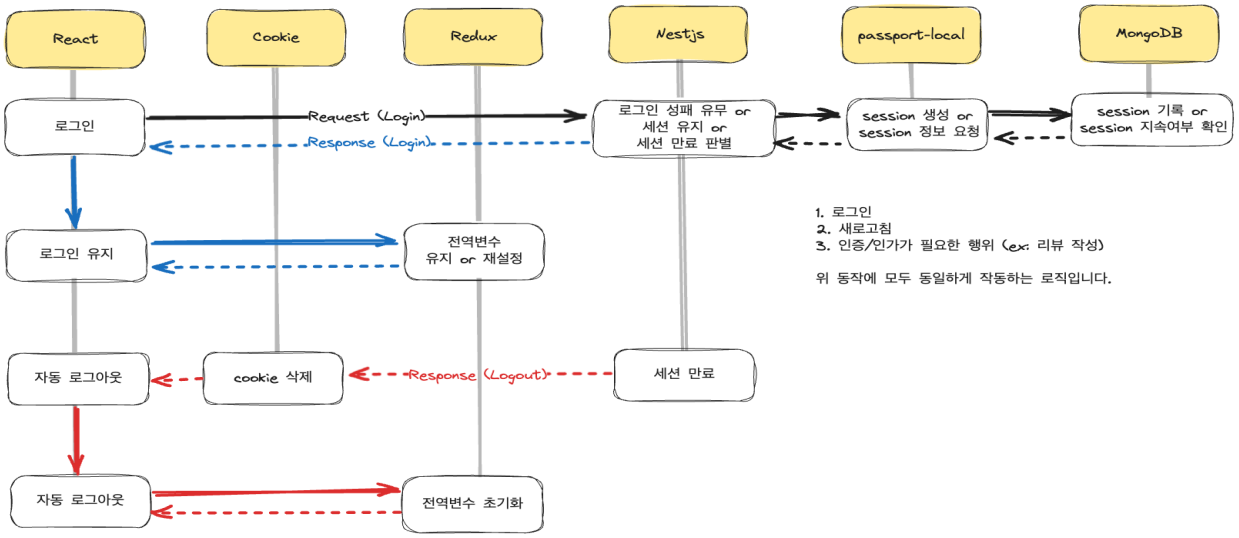
기존에는 팀프로젝트를 해왔기 때문에 프론트엔드 개발을 맡아서 했으나 제 개인 프로젝트는 서버를 직접 구현해서 인증/인가를 구현해야 했습니다.

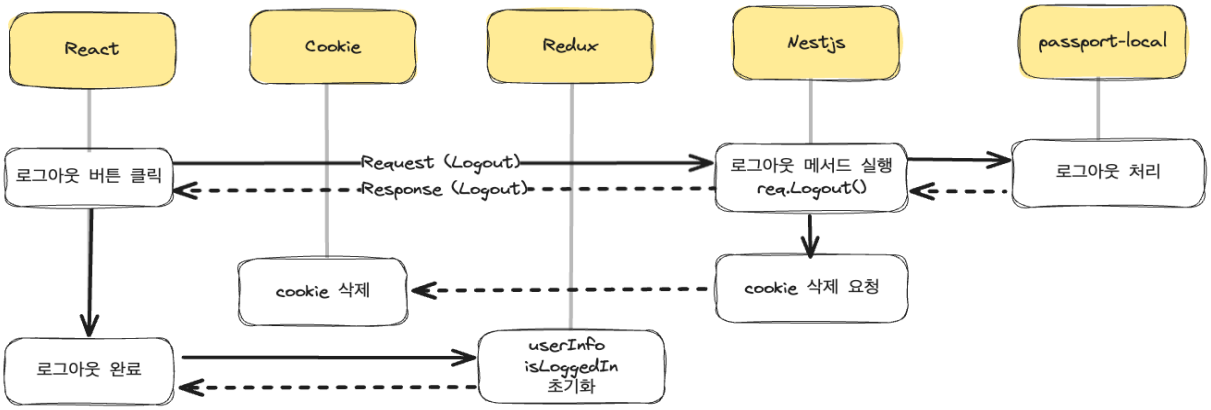
프로젝트에서는 클라이언트 웹서버가 하나이며, 이러한 환경에서는 JWT보다는 세션을 사용하는 것이 적절하다고 생각했습니다.

→ JWT는 주로 마이크로서비스 아키텍처에서 활용되며, 클라이언트 측에서 로그인 상태를 관리할 때 유용합니다.

→ 그러나 현재 프로젝트에서는 JWT를 구현하기에는 난이도가 높을 뿐만 아니라 세션을 사용함으로써 보안 레벨을 높일 수 있고, 개발 과정이 단순해지며, 개발 속도와 생산성을 향상시킬 수 있다고 판단했습니다.

→ 따라서 세션을 사용하는 것이 프로젝트에 적합하다고 판단했습니다.





결론

클라이언트에서 인증/인가 여부를 관리하는 것이 아닌 직접 서버를 구현하며 JWT와 Session의 차이를 알 수 있었고 직접 구현한 서버에서는 보안 걱정 없이 로그인 상태와 사용자 정보를 효과적으로 관리할 수 있게 됐습니다.

2.JWT 기반의 인증/인가 구현

프로젝트 초기에는 세션 기반 인증을 도입했으나, 이후 JWT를 통해 인증 방식을 개선해 보았습니다.

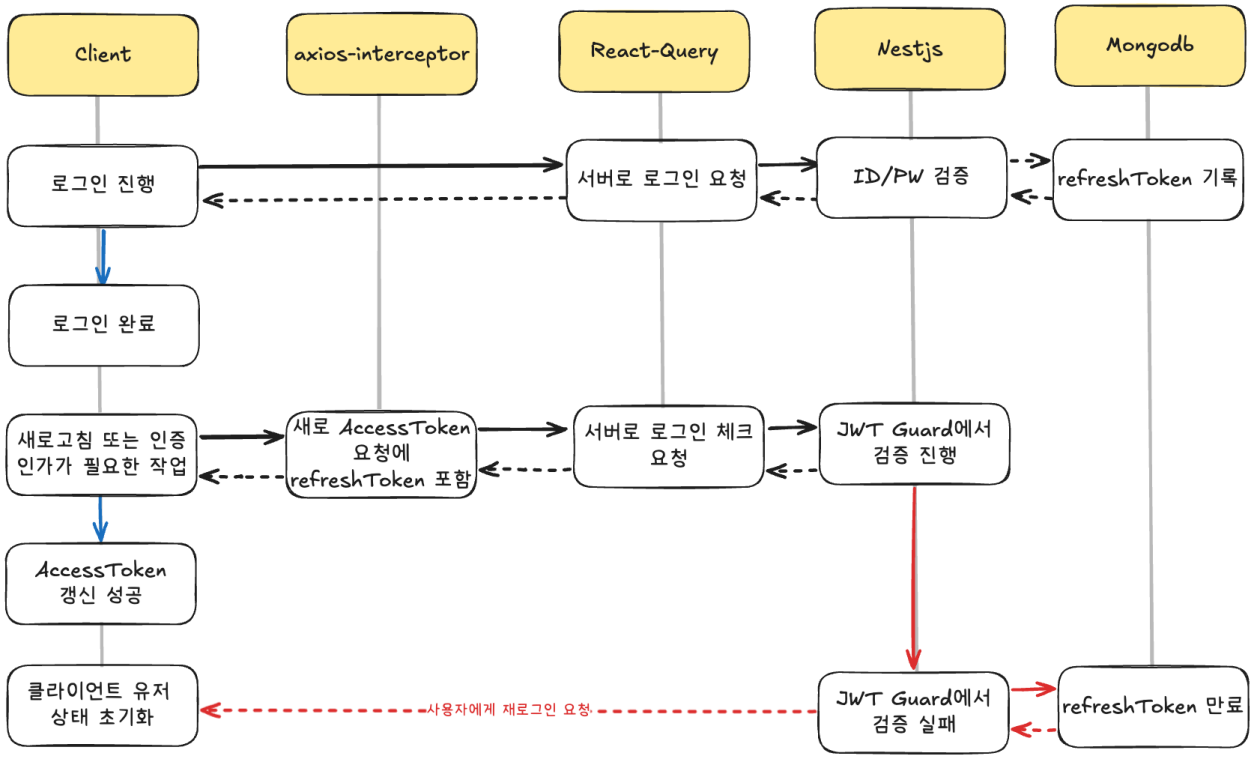
기존의 세션 인증/인가의 경우 서버에서 관리하기 때문에, 요즘처럼 마이크로 프론트엔드 아키텍처를 많이사용하고 서버도 하나만 두지 않는 인프라 환경에서는 확장성이 부족하다고 느꼈습니다.

→ 이러한 부분에서 서비스의 추후 확장을 고려하여 JWT가 더 적합하다고 판단하였습니다.

RESTful API 방식에서는 클라이언트가 요청마다 서버와 인증 데이터를 주고받아야 합니다.

→ 토큰 기반의 인증은 이를 간소화하는 데 큰 도움을 주었습니다.

→ 또한 JWT의 **Stateless** 특성은 API 설계를 단순하게 만드는 데도 도움이 되었습니다.



결론

JWT를 도입하면서 인증/인가의 **확장성**과 **편의성**을 강화할 수 있었습니다.

세션 기반 인증이 보안적 장점과 구현의 단순성을 제공했지만, 확장성과 API 인증 효율성 측면에서는 한계를 느꼈습니다.

JWT를 통해 더 유연하고 확장 가능한 인증 방식을 구현할 수 있었으며, 프로젝트의 성장 가능성을 고려한 구조 설계의 중요성을 배울 수 있었습니다.

3.crontab을 사용하여 정기적으로 TMDB 데이터를 받아오는 스크립트 구현

Open API를 이용하다 보니 직접 구현할수 있는 부분도 Open API에 의존하게 되는 경우가 생겼습니다.

TMDB(영화 및 TV프로그램의 데이터를 제공하는 Open API)의 최신 콘텐츠를 일정한 간격으로 가져오는 서버를 구축하였습니다.

→ crontab을 이용해 일정한 시간 간격으로 TMDB로 요청을 보냅니다.

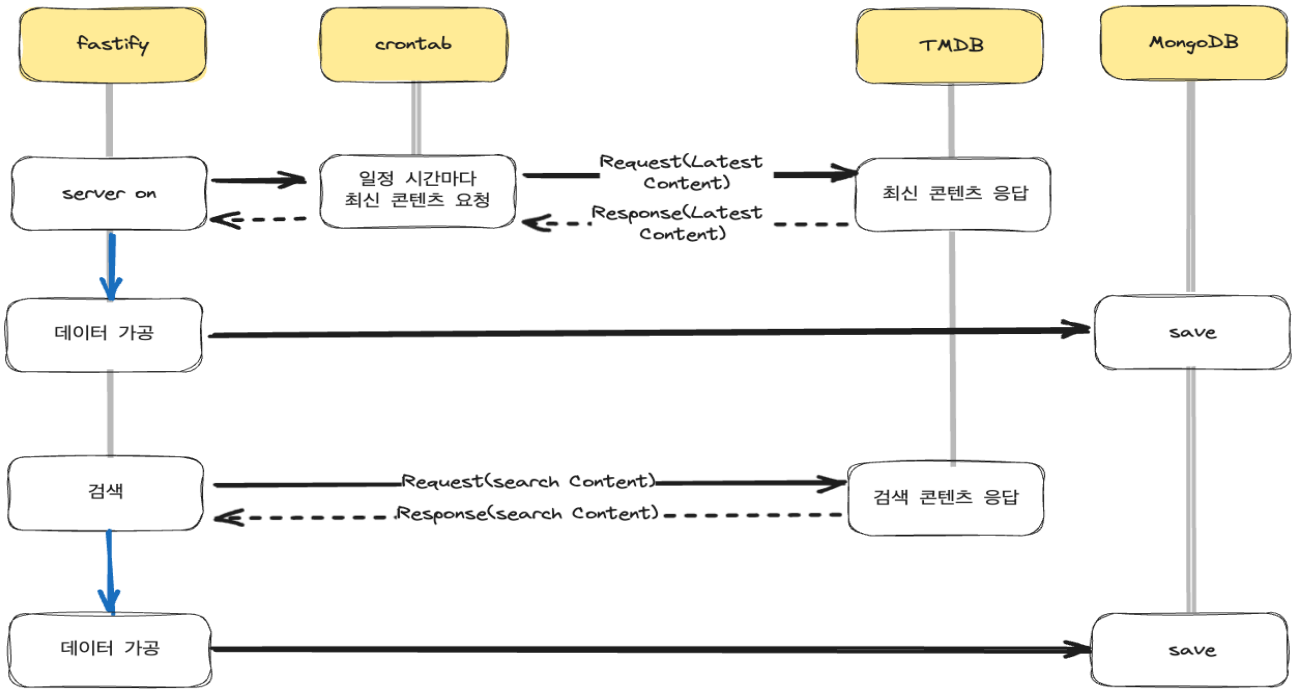
→ 미디어 데이터를 룸크리틱 데이터 요구 사항에 맞게 가공한 후에 데이터베이스에 저장합니다.

자동화를 시켜서 최대한 많은 미디어 데이터를 가져왔지만 마이너한 작품의 경우 없는 경우도 생겼습니다.

→ 이를 통해 운영자는 검색을 통해 원하는 콘텐츠를 쉽게 찾고, 필요한 경우 데이터베이스에 저장할 수 있습니다.

→ 서비스의 유연성을 높이고, 운영자가 콘텐츠를 관리하는 데 편의성을 제공합니다.

→ 사용자들에게는 다양한 콘텐츠를 제공할 수 있으며, 운영자는 새로운 컨텐츠를 쉽게 추가하여 서비스를 업데이트할 수 있습니다.



결론

서버는 주기적으로 최신 미디어 데이터를 데이터베이스에 저장하게 되었고, 사용자는 실시간으로 업데이트 된 미디어 컨텐츠를 볼 수 있습니다.

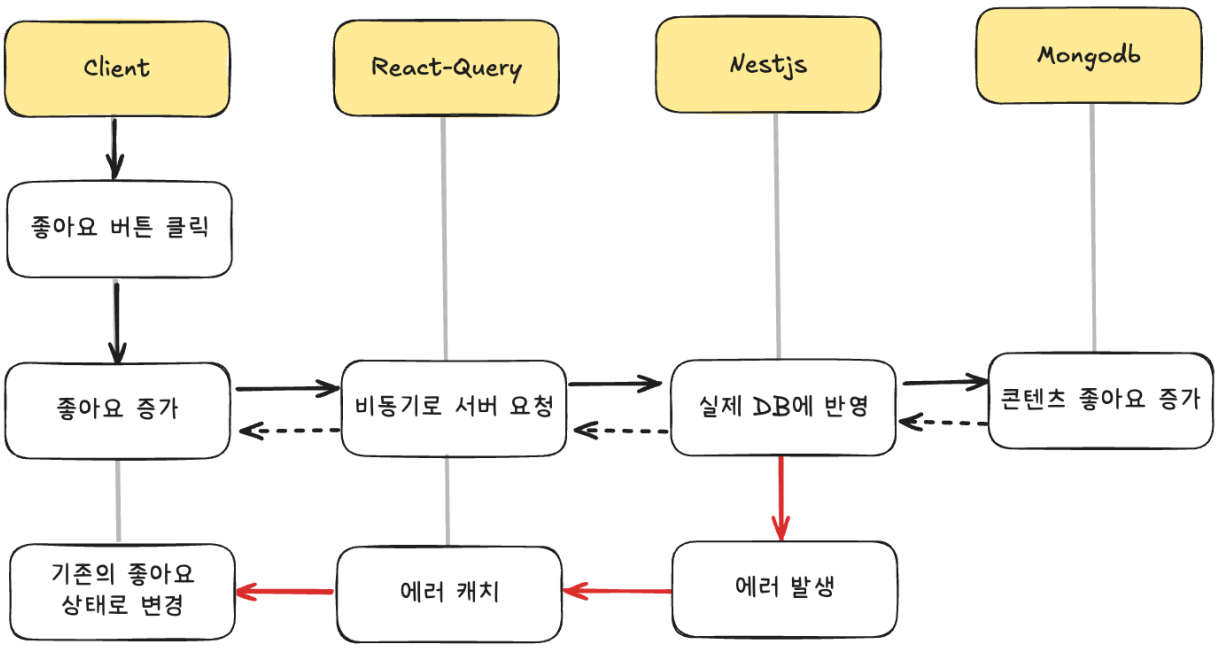
자동화 서버를 만들어 반복적인 작업을 줄이고 효율적으로 코어 개발에 집중할수 있게 됐습니다.

4.낙관적 업데이트를 이용한 좋아요 기능 구현

우리가 자주 볼수 있는 좋아요 기능은 누르는 즉시 유저의 눈에 반영이 되는 모습을 보여주어야 합니다.

그런데 서버에 보내는 요청은 클라이언트 측에서 즉시 반응할 것이라는 확신이 불가능 합니다.

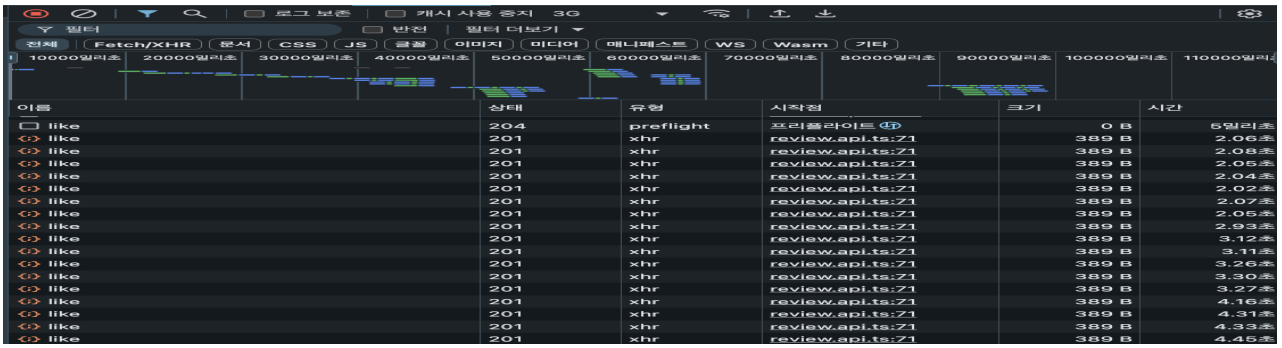
그래서 낙관적 업데이트를 하여 클라이언트 측에서 미리 좋아요와 같은 행위가 반영되었다는 것을 유저에게 보여줄수 있도록 구현하였습니다.



결론

에러 처리를 간결하고 명확하게 할 수 있게 되었습니다.

느린 3g 환경에서 요청에 대한 처리가 3초로 늘어났음에도 여러 요청에 대한 응답이 제대로 반영되는 모습을 확인했습니다.



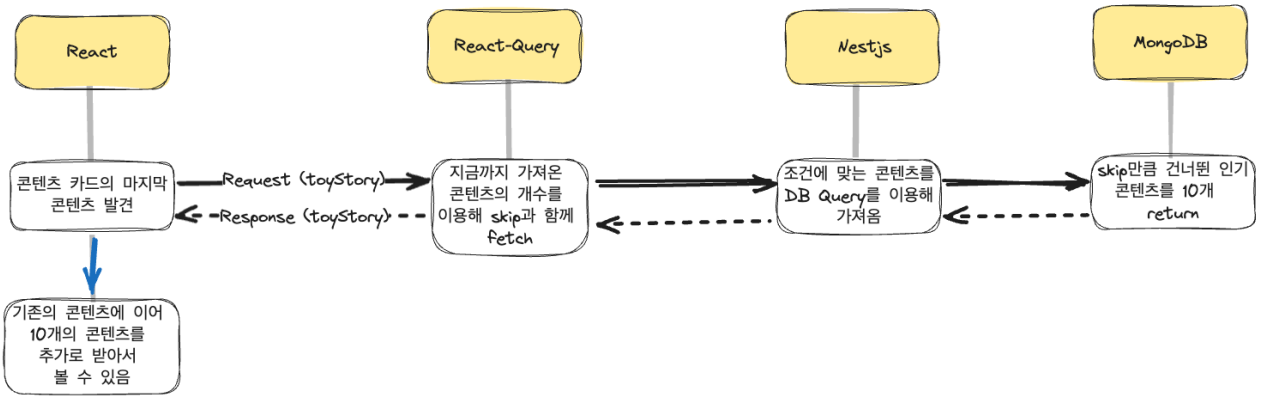
5. Cursor 기반 페이지네이션을 이용한 무한 스크롤 구현

콘텐츠에 대한 정보를 볼때 고정적으로 10개의 데이터만 보여주는 것이 문제라고 생각했습니다.

→ 무한 스크롤을 이용하여 콘텐츠를 10개씩 불러오면 사용성도 높아지고, 콘텐츠에 대한 데이터를 10개씩만 불러오기 때문에 서버에 부하도 작을 것이라고 생각하고 구현을 시작했습니다.

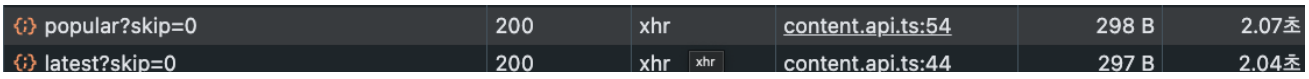
인기순으로 정렬해주는 쿼리를 작성하여 기존과 같은 개수인 10개의 콘텐츠를 가져옵니다.

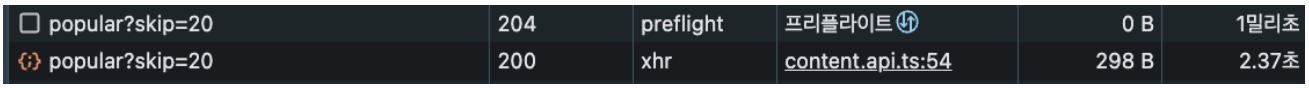
→ 그 후에 Observer가 콘텐츠 끝까지 스크롤이 됐다는 것이 감지되면 11번째 콘텐츠부터 20번째 콘텐츠를 가져오고 이것을 반복합니다.



결론

초기 로딩시간을 변함없이 유지 시키며 무한스크롤을 구현하였습니다.
3G 환경에서도 10개의 콘텐츠만 받아오면 되기 때문에 느린 네트워크에서도 안정적인 응답 속도를 보여줍니다.





위에 구현한 방법 외에도 MongoDB의 ObjectId를 이용하여 구현하는 방법도 시연해 보았기 때문에 더 다양한 카테고리의 콘텐츠들을 무한 스크롤로 구현할수 있게 됐습니다.

useObserver라는 훅을 구현하여 어떠한 요소의 값을 추출해 내는 기능을 범용성을 넓게 사용할수 있게 되었습니다. 이 훅의 기능을 숙지하면 무한 스크롤이 아닌 사용자의 동작에 따라 보여지는 어떤 요소에 대해 다양한 이벤트를 만들수 있게 되었습니다.

6.Feature-Slide-Design 프론트엔드 아키텍쳐 적용

기존의 코드 구성은 기능 추가 및 수정을 진행할 시 도메인별로 엮여있는 경우가 많아서 Side Effect가 많이 일어나 불편함을 많이 겪었습니다.

프로젝트의 확장성과 유지보수성을 높이기 위해 Feature-Slide-Design(FSD) 아키텍처를 도입했습니다.

→ 도메인별로 모듈화된 컴포넌트 설계를 하여 관심사를 분리하고 Side Effect를 효과적으로 줄일 수 있었습니다.

도메인별로 모듈화를 시키니 상태 관리에 어려움을 겪는 일도 생겼습니다.

→ Redux Toolkit을 이용해 중앙에서 도메인별로 상태를 관리하니 훨씬 직관적이고 단순하게 기능 구현을 할 수 있었습니다.

결론

프로젝트 구조가 단순 명료해져 개발 생산성뿐만 아니라 Side Effect가 확연히 줄어 안정성까지 챙길 수 있었습니다.

분리 된 도메인으로 인해 새로운 기능 추가 및 수정작업을 더욱 유연하게 할 수 있었고, 서버와 비슷한 형태의 폴더 구조가 만들어진 점도 개발 생산성 증가에 큰 도움이 되었습니다.

비록 개인 프로젝트에 적용한 아키텍처이지만, 팀으로 작업하면 더 큰 시너지가 날 것이라고 확신합니다. FSD 적용기를 담은 저의 포스팅입니다.

https://velog.io/@qwzx16/FSD-%ED%94%84%EB%A1%A0%ED%8A%B8%EC%97%94%EB%93%9C-%EC%95%84%ED%82%A4%ED%85%8D%EC%B3%90-%EC%A0%81%EC%9A%A9%EA%B8%B0

이때 당시 해당 아키텍처에 대한 이해가 좀 모자랐음을 느껴서 지금도 꾸준히 리펙토링을 진행하고 있습니다.