남다른 성장곡선의 개발자, 조성배 입니다

FRONTEND DEVELOPER

010-6437-9796

tmsprqo@gmail.com

https://github.com/ChoBae

https://chobae.vercel.app/

INTRODUCE

"순수 함수"의 원칙을 따라, 저는 **'배움'이라는 인자에 대해 일관되게 '지식 공유'라는 결과를 내놓는 개발자로 성장**하고자 합니다. **카이스트에서의 5개월 간의 집중적인 프로젝트 경험**을 통해, 다양한 팀원들과 아이디어를 교류하며 **개발의 다면성을 탐구하고, 솔루션에 대한 절대적인 '정답'이 존재하지 않음을 이해**하게 되었습니다. 이러한 경험은 저에게 지식 공유의 가치와 중요성을 깊이 인식시켰습니다.

현재, CSR의 한계를 느껴 **Nextjs를 기반으로한 사이드 프로젝트를 진행**하고 있고, 지식 공유를 실천하고자 **UI 컴포넌트 라이브러리도 배포**하였습니다.

SKILLS

Programming Language

- TypeScript(JS)

Frontend

- React, Next.js
- Recoil, React Query

Styling

- Tailwind, Styled-Component

Testing

- Jest

EDUCATION

전자공학과 배재대학교
2014 ~ 2020

- **정보처리기사 취득**

EXPERIENCE

SW 정글 KAIST

2022.09 ~ 2023.02

- 카이스트에서 진행한 **CS 위주의 부트캠프**
- **5개월간 매주 100시간 이상 개발** 몰입 경험
- 매주 20문제 이상의 알고리즘 문제를 풀이하며 알고리즘 이해
- RB트리, Malloc Lab, 웹 프록시 서버 등 구체적인 컴퓨팅 동작을 C언어로 구현
- 카이스트의 OS 프로젝트인 'PintOS'의 스레드, 프로세스, 가상메모리, 파일시스템 등 OS 핵심 구성 요소를 직접 구현
- 한달간의 팀 프로젝트를 통해 기획부터 배포까지 모든 과정을 함께 경험하며, 협업 능력과 실전 경험을 동시에 향상

참여 프로젝트 (최근 순으로 정렬)

DIA 2023.11 ~ 진행 중🔥 (바로가기)
#NEXTJS #TAILWINDCSS #JEST
TTS와 음성인식을 활용해 개발자들의 실전 같은 면접을 돕는 서비스

SummaryWagon 2023.05.01 ~ 06.15 (바로가기)
#NEXTJS #REACT QUERY #JEST
블로그나 테크 아티클 같은 웹사이트의 내용을 세줄 요약 해주는 서비스

Highlighters 2022.12.23 ~ 2023.01.27 (바로가기)
#REACT #RECOIL #REACT QUERY #TAILWIND CSS
크롬 익스텐션을 활용한 중요한 텍스트, 이미지, 유튜브 동영상을 강조하고 공유하며 협업하는 아카이빙 플랫폼

DIA

2023.11.22 ~ 진행중🔥

#NEXTJS #TAILWINDCSS #JEST

참여 인원 : 3명 (FRONTEND, BACKEND, UI/UX 디자이너)
담당 역할 : FRONTEND
깃허브 : 바로가기
사이트 : 바로가기

음성인식과 TTS를 활용하여 보다 리얼한 면접 환경을 경험할 수 있습니다💁

"TTS와 음성인식을 활용해 개발자들의 실전 같은 면접을 돕는 서비스 "

프로젝트 요약

- 실제 면접에서 자주 등장하는 질문을 제공하며, 각 질문에 대한 스크립트 작성을 지원합니다.
- TTS(Text To Speech) 기능을 활용하여 사용자에게 입체감 있는 모의 면접 경험을 제공합니다.
- 모의 면접 이후 STT(Speech To Text)를 통해 사용자가 작성한 스크립트와 비교하여 향상된 학습 기회를 제공합니다.

프로젝트 기여

- **웹 UI 구현:** 사용자 친화적이고 직관적인 인터페이스를 개발하여 면접 연습을 원활하게 할 수 있도록 디자인하고 구현하였습니다.
- **모의 면접 시 STT 구현:** 음성 입력 기능을 통해 사용자가 모의 면접에서 대답한 내용을 텍스트로 변환하여 분석할 수 있게 구현하였습니다.
- **TDD (Test-Driven Development) 활용:** 개발 과정에서 테스트 주도 개발을 적극적으로 활용하여 코드의 신뢰성과 유지보수성을 높였습니다.
- **CSS 최적화:**
  - **UI 컴포넌트 최적화:** 테일윈드의 기존 단점인 코드 가독성 문제를 해결하기 위해, 재사용성을 고려하여 UI 컴포넌트를 구성했습니다.
  - **커스텀 최적화:** Tailwind-merge를 활용하여 CSS 클래스 병합을 최적화함으로써, 코드의 가독성과 유지보수성을 향상시켰습니다.

#프로젝트 트러블 슈팅

☒Tailwind 단점 개선

개발 과정에서 저에게 친숙한 Tailwind CSS를 활용하여 스타일링의 효율성과 속도를 대폭 향상시켰습니다. 그러나 프로젝트 규모의 증가와 함께 Tailwind의 일부 제한 사항이 두드러지기 시작했습니다. 가장 큰 문제는 CSS 클래스의 중복과 길어진 클래스 명명으로 인한 가독성 저하였습니다.

이 문제에 대응하기 위해 처음에는 인라인 CSS의 사용을 최소화하고 UI 컴포넌트를 분리하는 방식으로 가독성을 개선하려고 시도했습니다. 이 접근법은 인라인 CSS를 현저히 줄여 가독성을 개선하는 데는 효과적이었지만, 기존 Tailwind의 한계로 인해 UI 컴포넌트의 커스터마이징에는 제약이 있었습니다. 이에 따라, 저는 Tailwind Merge를 도입해 CSS 클래스 관리를 더욱 최적화하는 전략으로 전환했습니다.

Tailwind Merge의 도입은 클래스 중복을 효과적으로 감소시키고 코드베이스의 가독성을 크게 개선하는 데 중요한 역할을 했습니다. 다수의 CSS 클래스를 효율적으로 결합하고 중복을 제거함으로써 스타일시트의 크기를 최소화하고 렌더링 성능을 향상시켰습니다. 이러한 간결하고 체계적인 클래스 명명 규칙은 개발자들이 스타일 코드를 더 빠르고 쉽게 이해하고 수정할 수 있게 만들어, 프로젝트의 전반적인 유지보수성을 개선하는 중요한 요소가 되었습니다. 이 경험을 통해, 저는 제가 주로 사용하는 Tailwind CSS를 더욱 효과적으로 활용할 수 있게 되었으며, 프론트엔드 개발에서 스타일링을 효과적으로 관리하는 방법에 대해 깊이 이해하게 되었습니다.

SUMMARYWAGON

2023.05.01 ~ 06.15

#NEXTJS #REACT QUERY #JEST

참여 인원 : 3명 (FRONTEND 1명, BACKEND 2명)
담당 역할 : FRONTEND
깃허브 : [바로가기]

“블로그나 테크 아티클 같은 웹사이트의 내용을 세줄 요약 해주는 서비스”

#프로젝트 요약

- 테크 아티클이나 블로그의 내용을 링크를 통해 세줄 요약 할 수 있습니다.
- 요약한 아티클을 아카이브해서 언제든 찾아 볼 수 있습니다.
- 최근 가장 많이 요약된 아티클을 쉽게 볼 수 있습니다.
- 키워드 추출 알고리즘을 통해 키워드를 추출하였고, 키워드별로 아티클을 볼 수 있습니다.

#프로젝트 개요

평소 개발 공부를 할 때 블로그나 기술 아티클을 참고하는 경우가 많았습니다. 하지만 많은 글들을 읽고 이해하는 데에는 상당한 시간과 노력이 소요되곤 했습니다. 이러한 경험으로부터, 글 요약 기능이 있다면 학습 시간을 단축하고 핵심 내용을 빠르게 파악하는 데에 도움이 될 것이라고 생각하게 되었습니다. 또한, 요약된 내용을 먼저 읽고 글을 본문까지 읽으면 이해하는 데에 더욱 효율적일 것으로 기대하게 되었습니다.
그래서 'SummaryWagon'이라는 웹 애플리케이션을 개발하게 되었습니다. 저희 서비스는 OpenAI를 활용하여 사용자가 원하는 링크를 입력하면 해당 글의 세 줄 요약을 즉시 확인할 수 있도록 제공합니다. 이렇게 간결하게 요약된 내용은 개인 히스토리에 저장되어, 언제든지 다시 찾아볼 수 있습니다. 이러한 기능을 통해 사용자들에게 시간을 절약하면서도 중요한 내용을 빠르게 파악하는데에 큰 도움을 주고자 합니다.

AI 기술은 빠르게 발전하고 있으며, 이제는 우리 생활과 업무에 깊이 녹아들어 많은 부분을 효율적으로 지원해줍니다. 그러나 기술적인 내용에 대한 이해가 부족한 사용자들에게는 AI 기술을 활용하는 것이 쉬운 일이 아닐 수 있습니다. 따라서 저희 서비스는 이러한 사용자들에게 AI 기술을 친숙하고 접근하기 쉬운 형태로 제공하여, 정보의 격차를 해소하고 더욱 포괄적인 지식 공유를 실현하고자 합니다.

프로젝트 설명

테스트 코드를 도입하여 개발 효율 증대

기존 프로젝트를 완성한 이후, 개발 과정에서 코드 작성과 디버깅에 많은 시간 낭비가 발생하는 문제를 인식하게 되었습니다. 이러한 문제는 프로젝트의 규모가 커지고 코드베이스가 복잡해짐에 따라 점점 더 심화되었습니다. 이로 인해 기능 개발을 위한 시간보다 디버깅에 많은 시간을 사용하게 되었고, 이러한 악순환을 극복하고자, 완성된 프로젝트에 Jest를 도입하여 개발 효율을 증대시키는 시도를 하게 되었습니다.

기존 프로젝트에 Jest를 도입하면서 전통적인 TDD(Test-Driven Development) 방법론을 사용하기 어렵다는 점을 인식하였습니다. 프로젝트가 이미 완성된 상태이기 때문에, 기존 코드를 테스트 가능하도록 Jest를 활용하여 테스트 코드를 작성하는 방식을 선택하였습니다. 이에 따라 기존 코드를 더 깊이 이해하고 수정할 필요가 있었습니다.

처음에는 기존 코드에 대한 테스트 케이스를 작성하는 것이 어려움을 겪었습니다. 완성된 코드들을 테스트하기 위해서는 코드를 수정하고 테스트 가능한 형태로 리팩토링해야 했습니다. 그러나 이러한 노력을 통해 기존 코드에 대한 이해도가 크게 향상되었습니다. 더불어 테스트 코드를 작성하면서 기존 코드의 동작 방식과 흐름을 더욱 명확하게 파악할 수 있었습니다. 기존 코드를 테스트 가능한 상태로 만들면서 코드 품질과 유지보수성을 향상시키는데 큰 도움이 되었습니다.

Jest를 도입한 결과, 테스트 코드 작성에 소요되는 시간이 있었지만, 이를 통해 코드를 개선하고 검증하는 과정에서 많은 이점을 얻을 수 있었습니다. 기존 코드를 수정하면서 안정성이 향상되었고, 수정한 코드의 예상대로 동작함을 테스트를 통해 확실히 확인할 수 있었습니다. 이로 인해 기존 코드의 버그를 사전에 발견하고 수정할 수 있어서 개발 과정에서 발생하는 잠재적인 문제를 예방할 수 있었습니다.

또한, 테스트 코드를 추가함으로써 새로운 기능을 추가하거나 기존 기능을 수정할 때 더욱 안심하고 코드를 변경할 수 있었습니다. 테스트 케이스를 작성함으로써 기존 코드의 예상 동작을 명확하게 파악하고, 변경 사항이 다른 부분에 영향을 미치는지 사전에 파악할 수 있었습니다. 이는 기능 추가 및 변경 작업에 대한 리스크를 최소화하고 안정적인 업데이트를 할 수 있도록 도왔습니다.

Jest를 도입한 이후로 개발 효율이 크게 향상되었습니다. 코드 작성과 디버깅에 소요되는 시간이 현저히 줄어들었으며, 코드를 안정적으로 유지하고 개선하는 데에 도움이 되었습니다. 또한, 테스트를 통해 기존 코드의 버그를 발견하고 미리 수정함으로써 안정성을 확보할 수 있었습니다.

프론트엔드 개발 효율 향상을 위한 Mocking 활용

비효율적인 백엔드 API 개발로 인한 프론트엔드 업무의 지연되면서 이러한 과정이 비효율적이라는 것을 느꼈고,개선하고자 여러 가지 방법을 시도했습니다. 처음에는 Mocking JSON 데이터를 생성하여 프론트엔드 개발을 진행하는 방법을 선택했습니다. 이는 빠르게 활용 가능하다는 장점이 있었지만, API 응답 상태에 따른 대기, 로딩, 에러 등의 구현에 어려움이 있었습니다. 실제 API와 유사한 동작을 보장하지 못했기 때문에 테스트에서 예상치 못한 문제가 발생하기도 했습니다.

저는 MSW(Mock Service Worker)를 도입한 후 문제를 해결할 수 있었습니다. MSW는 실제 네트워크 요청을 가로채서 Mocking Response를 보내주는 기능을 제공해줍니다. 이를 통해 API 개발이 완료되기를 기다릴 필요 없이 즉시 활용할 수 있었습니다. 더불어, MSW는 여러 응답 상태에 따른 구현을 간단하게 진행할 수 있게 해주어, 로딩 상태, 에러 처리 등을 쉽게 테스트하고 확인할 수 있었습니다.

MSW를 도입한 결과, 백엔드 API 개발 지연으로 인한 업무 중단을 최소화할 수 있었고, 저는 Mocking 된 데이터를 활용하여 독립적으로 개발을 진행할 수 있으며, 테스트 용이성이 증가했습니다. 더불어, 요구사항 변경에 더욱 유연하게 대응할 수 있어서 즉각적으로 API 명세를 변경하며 구현을 진행할 수 있었습니다.

또한, MSW를 활용한 모의 데이터를 통해 더욱 효과적인 테스트를 수행할 수 있었습니다. 테스트 시에 실제 API 응답이 아닌 Mocking된 데이터를 사용하므로, 테스트 환경을 더욱 안정화시키고 문제를 발견하는 속도가 향상되었습니다. 이는 개발 초기부터 예상치 못한 오류를 발견하여 수정하는 데에 큰 도움이 되었습니다.

MSW 도입으로 인해 팀은 프로젝트를 훨씬 더 효율적으로 진행할 수 있게 되었습니다. 실제로 프론트엔드와 백엔드 팀 간의 의존성을 최소화되어 동시에 개발을 진행할 수 있었고, 주간 미팅의 횟수를 줄여 개발에 몰두 할 수 있었습니다. 이를 통해 프로젝트를 보다 빠르게 완료할 수 있었습니다.

HIGHLIGHTERS

2022.12.23 ~ 2023.01.27

#REACT #RECOIL #REACT QUERY #TAILWIND CSS

참여 인원 : 5명 (FRONTEND 1명, 익스텐션 1명, BACKEND 3명)
담당 역할 : FRONTEND
깃허브 : 바로가기

“익스텐션을 활용한 중요한 텍스트, 이미지, 유튜브 동영상을 강조하고 공유하며 협업하는 아카이빙 플랫폼”

#프로젝트 요약

- 크롬 익스텐션을 통해 텍스트, 이미지, 영상에 하이라이팅하고, 기록을 저장할 수 있습니다.
- 동일한 웹사이트에 접속했을 때, 그룹원이 하이라이팅한 내용이 내 화면에도 실시간으로 반영됩니다.
- 공유한 자료를 아카이빙하고, 북마크 및 태그 기능을 통해 쉽게 찾아볼 수 있습니다.
- 그룹원에게 메시지 및 링크 알림을 보낼 수 있으며, 클릭시 해당 링크로 이동할 수 있습니다.
- 검색 기능을 통해 공유된 피드를 쉽게 찾을 수 있습니다. (과금 문제로 현재는 일반 검색 기능만 작동합니다.)

#프로젝트 개요

저희는 중요한 내용을 빠르고 간편하게 공유하고자 하는 3가지 고민에서 'Highlighters'라는 협업 툴을 개발하게 되었습니다.

이 협업 툴을 사용하면 웹페이지에서 한 눈에 중요한 내용을 파악할 수 있으며, 바로바로 그룹원들과 링크를 공유할 수 있습니다. 또한, 공유한 링크를 찾아보기 쉽게 모아둘 수 있어 편리합니다.

'Highlighters'는 크롬 익스텐션을 활용하여 텍스트, 이미지, 유튜브 동영상 등 다양한 요소를 하이라이팅할 수 있습니다. 그리고 이를 그룹 간에 공유하여 정보 공유를 간편하게 할 수 있습니다.

이러한 기능들을 통해 'Highlighters'는 메신저를 통한 정보 공유가 시간 낭비가 될 수 있는 경우, 중요한 부분을 빠르게 파악하고 공유하는 데 유용한 협업 툴이 되고자 합니다.

프로젝트 설명

세차례 걸친 메인 페이지 렌더링 성능 개선

서비스 배포 이전에 저희 팀은 베타버전 출시 후 사용자 30명의 피드백을 분석해보았고,
이 과정에서 메인 페이지의 로딩이 느리다는 피드백을 받았습니다.
이에 대응하기 위해 프로파일링을 실시하였고, **LCP를 2.5초로 개선할 필요성을 인지**하였습니다.

〈첫번째 개선〉

피드 컴포넌트 내부의 이미지 로딩 시간이 길다는 것을 파악하였습니다. 원본 이미지 대신 컴포넌트 크기에 맞게 리사이징한 이미지를 사용하면 성능 개선이 가능할 것으로 판단했습니다. sharp 라이브러리를 활용해 이미지를 리사이징하고, 리사이징된 이미지를 s3에 저장했습니다. 이를 통해 이미지 크기가 피드 5개 기준으로 1,008KB에서 472KB로 감소하여 LCP를 0.3초 가량 개선할 수 있었습니다.

〈두번째 개선〉

뷰포트 내에서 사용자가 볼 수 있는 피드는 3~4개 였기 때문에 모든 피드를 한꺼번에 로드하는 과정이 불필요하다고 판단하였습니다. 또한 불안정한 네트워크 환경까지 고려하여 한번에 로드하는 피드의 수를 12개로 셋팅하였고, 이 결과 LCP를 0.2초 가량 개선할 수 있었습니다.

〈세번째 개선〉

사용자에게 즉시 보이지 않는 피드를 렌더링하는 것은 불필요하다고 판단했습니다. 따라서 사용자가 피드를 로드할 때 모든 피드를 렌더링하는 대신, 스크롤을 내리며 보이는 피드만 렌더링하는 효율적인 방식을 선택하였습니다. 이를 구현하기 위해 Lazy Loading을 적용했고, 사용자가 스크롤을 내려 피드가 뷰포트에 들어올 때만 렌더링되는 방식으로 구현하였습니다.

〈결과〉

해당 개선사항을 통해 LCP를 1.6초로 크게 감소 시킬 수 있었습니다. 현재까지도 꾸준히 수정하고 있으며 최적화 훅이나 'virtual scroll'을 활용한 추가적인 개선도 고려하고 있습니다.

Suspense 기능을 활용하여 비동기처리 로직 개선

프로젝트가 커지면서 서버에서 데이터를 가져오는 비동기 로직이 많아지고, 이로 인해 코드의 가독성이 저하하는 문제가 발생했습니다.
특히, 로딩 중일 때 "isLoading" state를 사용하여 스켈레톤 UI를 렌더링하는 것이 코드 이해를 어렵게 만들었습니다. 이후 React v18에서 개선된 Susense 기능을 적용하여, 비동기 로직을 수행하는 컴포넌트에 Suspense 컴포넌트를 래핑한 것만으로 비동기 데이터 패칭이 가능했습니다.
이를 통해 비동기 로직을 간결하고 동기적으로 작성할 수 있었고, Error Boundary를 활용하여 유연한 에러 상황 처리까지 쉽게 처리해줄 수 있었습니다.

로딩 시 스켈레톤 UI 활용 FCP 개선

사용자 정보, 피드, 알림 등 많은 데이터를 서버에서 가져와 페이지에 렌더링 해야 했고, 데이터를 로딩 할 때는 사용자가 빈페이지를 보고 있어야 했습니다.
사용자가 빈페이지를 보고 있으면 이게 로딩 중인 건지 에러인 건지 파악할 수 없다고 판단했습니다.
이를 해결하기 위해 스켈레톤 UI를 적용하여 웹사이트나 애플리케이션에서 비동기 데이터를 로드하는 동안 사용자에게 로딩 중임을 시각적으로 알려주었습니다. 이를 통해 사용자는 비동기 작업이 완료될 때까지 빈 화면이나 로딩 스피너만 보는 것이 아니라 예상되는 콘텐츠가 어떻게 보일지 미리 확인할 수 있어 불안감을 줄이고 더 나은 사용자 경험을 제공할 수 있었고, 프로파일링을 통해 측정한 FCP 1.1초에서 0.7초로 개선할 수 있었습니다.

이미지 Lazy Loading 활용 LCP 개선

하이라이터즈의 '그리드 뷰' 기능을 사용하면 많은 피드들을 한번에 모아 볼 수 있습니다.
기존보다 많은 양의 피드가 뷰포트 내에 한번에 보여지는데 내부에 포함된 이미지가 많아서 로딩에 필요한 리소스가 많이 발생할 것 이라는 생각이 들어 프로파일링을 진행했고 LCP가 1.9초로 확인되었습니다.
이 후 뷰포트에 있는 이미지 즉, 당장 로드가 필요한 이미지만 로드하면 되겠다는 생각이 들었고,
이미지가 뷰포트에 들어왔을때만 로드하는 Lazy Loading을 적용하였고, 프로파일링 재측정 결과 LCP 1.9초에서 1.6초로 개선되었습니다.