Frontend
Next Generation
Project

Turborepo Platform Architecture

JUNG TAEINN
2025.07

INDEX

1. 차세대 프로젝트 소개

- 터보레포 플랫폼 프로젝트란
- 프로젝트 추진 배경
- 프로젝트 개요

2. 차세대 프로젝트 전환목적 및 기대효과

- 기존 프로젝트의 문제점 및 한계
- 전환을 통한 기대효과

3. 차세대 프로젝트 적용결과

- 서비스성능 개선
- 코드품질 개선 및 코드컨벤션 정립
- SEO/GA 개선
- 프로세스 개선
- 지식자산 산출물 자동화
- QA 품질 개선
- 프로젝트 비용 개선

4. NEXT STEP

- 프로젝트 TRADE-OFF
- 프로젝트 TO-DO
  - QA 테스트 엑셀 시나리오를 활용한 AI 테스트 자동화 설계
  - Figma MCP 활용을 통한 디자인-개발 연동 자동화 설계

5. APPENDIX

- 프로젝트 기술설명
  - 주요 기술스택
  - 아키텍처 구조
  - UI/UX 컴포넌트 구조
  - 코어 패키지 구조설계
  - Openapi 통한 API 자동화 설계
  - SEO/GA 등 외부 스크립트 관리
  - QA 단위/통합 테스트 설계
  - Docusaurus/Storybook 등 지식자산 산출물 자동화 설계
  - Docker 빌드/배포 설계
  - 기타 서비스 산출물

APPENDIX 섹션은 관련 담당자의 색인이 포함되어 있습니다

해당 Chapter 에서는 차세대 프로젝트의 전환 배경과 간략한 소개를 다룹니다.

프로젝트와 관련한 기획/디자인/개발/QA/보안/IT PMO 의 니즈를 다룹니다.

위 프로젝트에 참여한 분들과 역할, 기간에 대해 소개합니다.

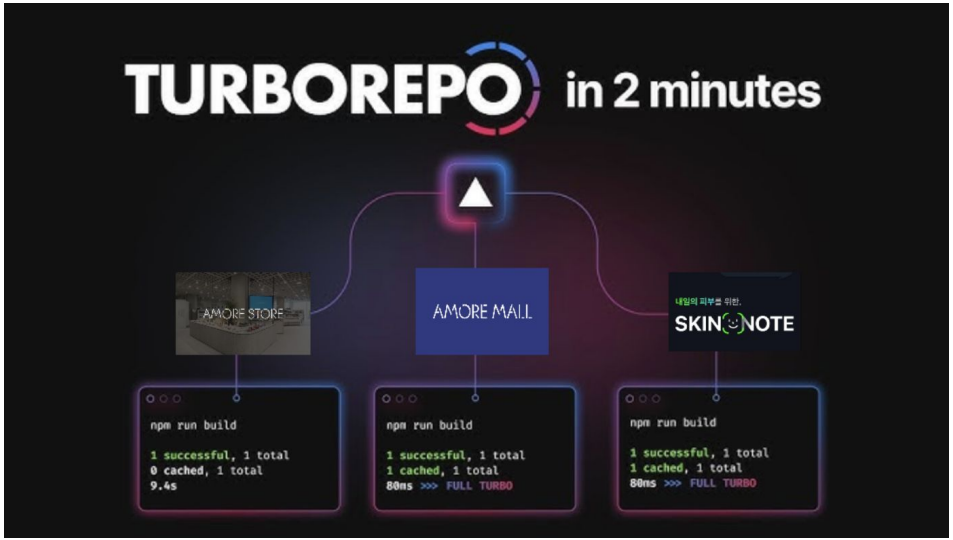
INTRO

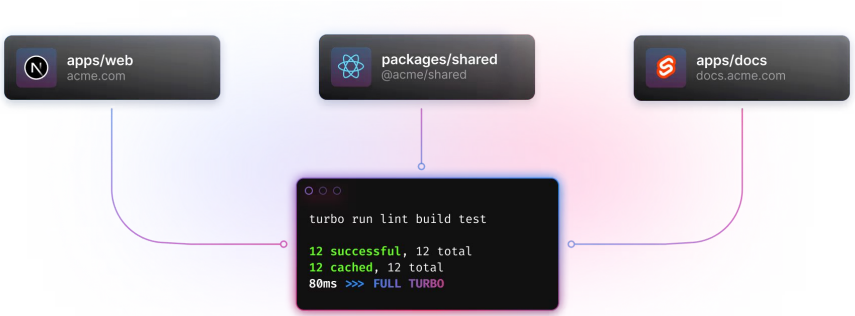
1. 차세대 프로젝트 소개

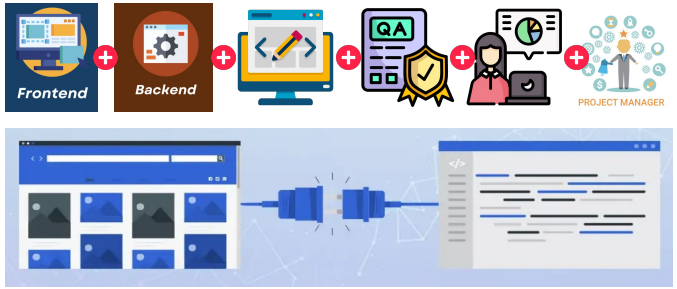
- 프로젝트 소개
- 프로젝트 배경
- 프로젝트 개요

INTRO

터보레포 플랫폼 프로젝트 소개







1. 여러 애플리케이션을 한 곳에서 관리

다양한 서비스가 확장됨에 따라 이를 효율적으로 관리하는 것이 매우 중요한데, 여러 애플리케이션을 한 곳에서 관리하면서도 빌드/배포를 최적화하고 서로의 서비스에는 영향을 미치지 않으면서 코드의 정형화를 이룰 수 있는 플랫폼 프로젝트

2. 전영역 자동화 연계

프론트엔드, 백엔드, 디자인, QA, PM 영역의 구간을 연결해주고, 해당 구간에서 발생하는 업무를 기술적으로 자동화하도록 설계되었습니다. 구간별 업무 자동화를 통해 불필요한 공수를 제거하고 코드 품질을 극대화하였습니다.

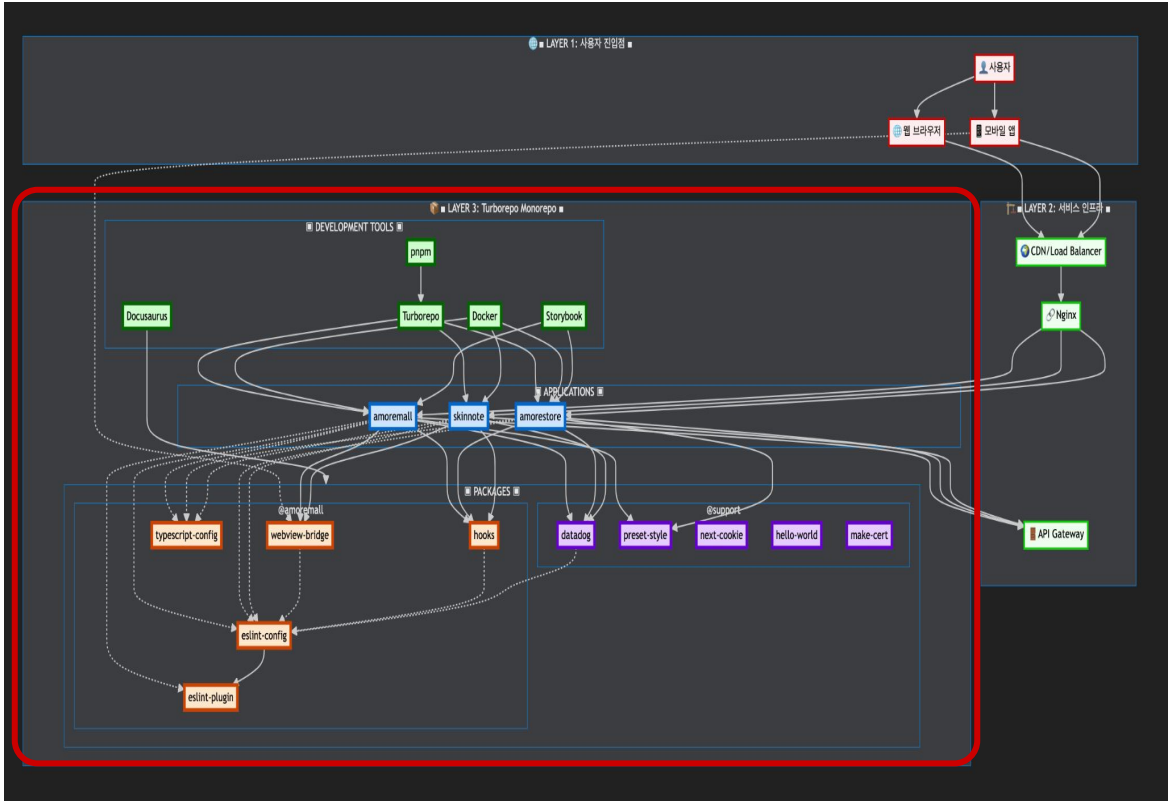
3. FE-AI 융합 연계 시스템 환경 구성

AI 와의 융합 연계 환경 기반을 생성하였습니다. 이후에 AI 패키지를 구성하여 프론트엔드와 타영역의 자연스러운 융합 환경을 이룰 수 있도록 하였습니다. A/B 테스트, 초개인화, QA 자동화 등등의 AI 라이브러리가 구성되어 프론트엔드 코드와 자연스레 융합되도록 구성하였습니다.

모노레포 구조 기반

OpenAI LINE 배 N Tripadvisor ebay

INTRO



핵심 아키텍처 다이어그램

1. Applications

아모레몰, 아모레스토어, 스킨노트 등 B2C 의 다양한 서비스가 독립적으로 구성되어 있으며,
B2B 같은 서비스를 추가해서 B2B, B2C의 플랫폼 프로젝트로 운영이 가능합니다.
독립적으로 동작하기 때문에 타 서비스의 빌드/배포 및 구동에 영향을 끼치지 않아
보다 안정적으로 동작합니다.
터보레포 캐싱을 활용하여 30개 이상의 애플리케이션을 구성하지 않는 한, 하나의 저장소
프로젝트에서 성능적으로 안정적이고 빠른 빌드/배포 지원을 유지합니다.

2. Packages

회사의 기술 코어 역할을 하는 패키지 영역입니다.
현재는 @amoremall, @support 로 구성되어 있으나, 애플리케이션의 확장 및 성격에 따라
추가로 관리하여 라이브러리를 효율적으로 관리할 수 있습니다.
패키지들은 애플리케이션들에서 범용적으로 사용하는 주요 기술들을 위주로 구성되어 있으며,
해당 룰은 내부 코어 개발자들만 관리하여 운영합니다.

3. Development Tools

Turborepo 를 통해 Monorepo 구조를 적절히 캐싱하여 여러 프로젝그를 효율적이고 빠르게
개발/운영 관리합니다.
Docker 기반의 CI/CD 로 구성되어 있으며, Turborepo 아키텍처 기반으로 설계가 되어
빌드/배포를 최적화시켰습니다.
Docusaurus, Storybook 등을 통해 개발자는 개발코드만 잘 작성하면, 개발코드 기반으로
문서를 생성할 수 있도록 자동화하였습니다.

INTRO

" 왜? 이 프로젝트를 진행하게 되었는가 "

회사 내 프론트엔드 개발 초기멤버로 5년동안 근무하며 React 내재화, 서비스 그랜드오픈, PC/모바일 통합, 신규 서비스 오픈, 성능 최적화 등 다양한 프로젝트를 진행하였습니다.
기획자, 디자이너, 개발자, QA, 보안 담당자, IT PMO 등 다양한 협업을 하며, 협업 과정에서 담당자들의 불편했던 프로세스 개선 및 회사의 기술 경쟁력을 확보하고자 프로젝트를 진행하였습니다.

기획자, 디자이너

- 운영 SR 진행 시, "기획 - 디자인 - 마크업 개발 - 프론트 개발" 프로세스에서 **중복 프로세스로 인한 불필요 공수를 줄여 빠른 서비스 결과물 도출**
- 기획서, 디자인 산출물, 개발 코드에서 사용하는 컴포넌트 네이밍 통일성 부재로 소통 미스 상황을 타개하고, **회사의 네이밍 자산을 강화**

IT PMO, 현업 담당자

- SI 프로젝트 시, B2C/B2B 서비스에 대해 초기 설계로 공수가 많이 발생하는데, 이를 절감 가능

외주 개발자에게 단순 생산성 위주의 개발을 관리하고, 내부 직원이 코어 개발을 하는 방식으로 구분하여 회사 기술력을 확보하고자 함

FE/BE 개발자

- 서비스를 운영하며 의존성이 서로 심하고 비즈니스가 비대해지는데, 관리 부재로 인한 빌드 성능이 저하되고 코드품질 관리가 어려웠음.

유연하게 확장성이 가능하고, 의존성을 최소화하여 관리가 편리하고 서비스 확장과 빌드 성능을 높이는데 포커스함

보안 담당자

- 프로젝트 신규 오픈 시, 보안성 검토는 필수인데

프로젝트 초기 환경을 템플릿화하여 중복 보안성 검토에 대한 보안/개발 담당자 부담이 줄어듬

QA 담당자

- 비즈니스 기반의 복합적 QA는 상대적으로 균일한 품질을 유지하기가 어려움

개발 레벨에서 단위/통합 QA를 지원하여 QA가 비즈니스 위주로 검증 강화 목적

INTRO

“ Turborepo 기반의 Tech 기술 및 프로세스 전반을 개선하고 플랫폼화 하는 Frontend Superset Platform Project “

담당역할

성명	프로젝트 역할
정태인	Project Lead, Project Architect, Frontend System Development
최성은	Project Tech Advisor, Frontend System Development
김윤아	Markup Architect, Markup System Development
안재민	DevOps Support

개발기간

1 **24.12 - 25.01**

프로젝트 아키텍처 설계 및
개발방향 전략 구상

2 **25.01 - 25.05**

프로젝트 시스템 개발
(Frontend, Markup, DevOps 등)

3 **25.05 - 25.07**

운영환경 배포를 위한 준비 B2C
서비스 페이지 개발 착수 및
문서화 정리

4 **25.07 -**

애플리케이션 개발 및 자동화 R&D 확대
B2C, B2B 템플릿 개발/운영 검증 및
AI 자동화 코어 라이브러리 개발 확대

해당 Chapter 에서는 차세대 프로젝트의 전환목적과 적용에 따른 기대효과를 보여줍니다.

기존 프로젝트의 문제점 및 한계를 간략히 짚어보고,

프로젝트 전환을 통한 얻는 이점에 대해 다루어 봅니다.

ISSUE

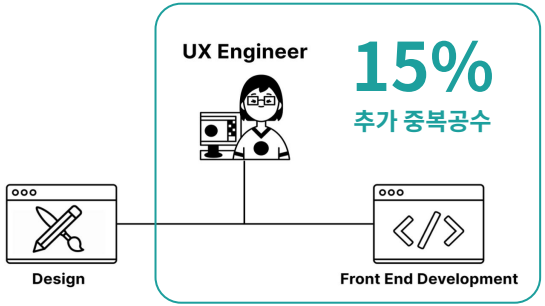
2. 차세대 프로젝트 전환목적 및 기대효과

- 기존프로젝트의 문제점 및 한계
- 전환을 통한 기대효과

ISSUE

1 마크업-프론트엔드 개발/운영 간 중복공수 요소발생

기존 시스템은 디자인 산출물을 토대로 마크업 개발을 진행하며, 마크업 결과물을 토대로 프론트엔드 개발을 진행합니다. 디자인 검수가 진행되며, 마크업-프론트 개발 간 동기화로 인한 중복공수가 발생하며 동기화 누락이 발생하여, 에러의 요소가 되기도 합니다.



2 서비스 확장을 고려치 못한 시스템 노후화로 빌드 성능저하

기존 시스템은 의존성에 의거하여 공통화가 컨셉이었습니다. 서비스의 확장으로 인해 다양한 패키지들이 추가되었고 관리의 부재로 인해 빌드시간은 10분이상 소요되며 3.2배 저하되었습니다.

6,428
전체 패키지 의존성

14.5%
전체 패키지 중복률

3.2x
빌드시간 저하

중복 의존성 및 빌드량 증가로 인해
평균 10min 소요

3 프로젝트 구축 시 기술자산 부재로 발생하는 다양한 문제

당사에서 진행하는 여러 프로젝트 구축시 초기 설계비용이 중복 발생합니다. 또한 상이한 설계구조로 인해 기술자산이 쌓이지 못하고 지식자산 산출물은 제대로 활용되지 못하고 있습니다.

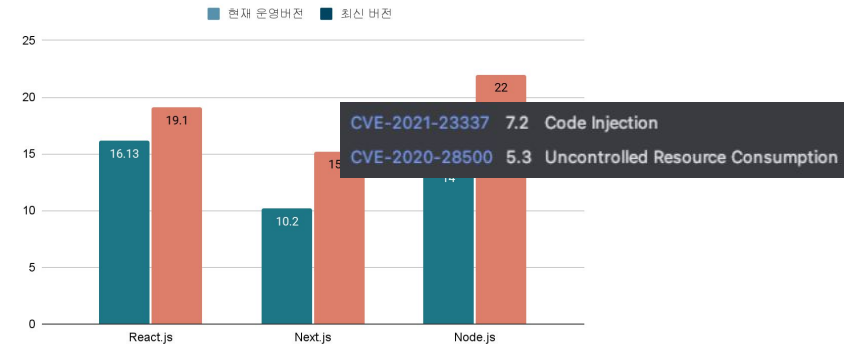
중복 설계비용
평균 3개월의
초기 설계구축

환경 이분화
프로젝트 이분화로 인한
다른 환경 운영부담 증가

지식자산 부재
산출물의 일관되지 못한 구조와
수동 업데이트로 인한 가치저하

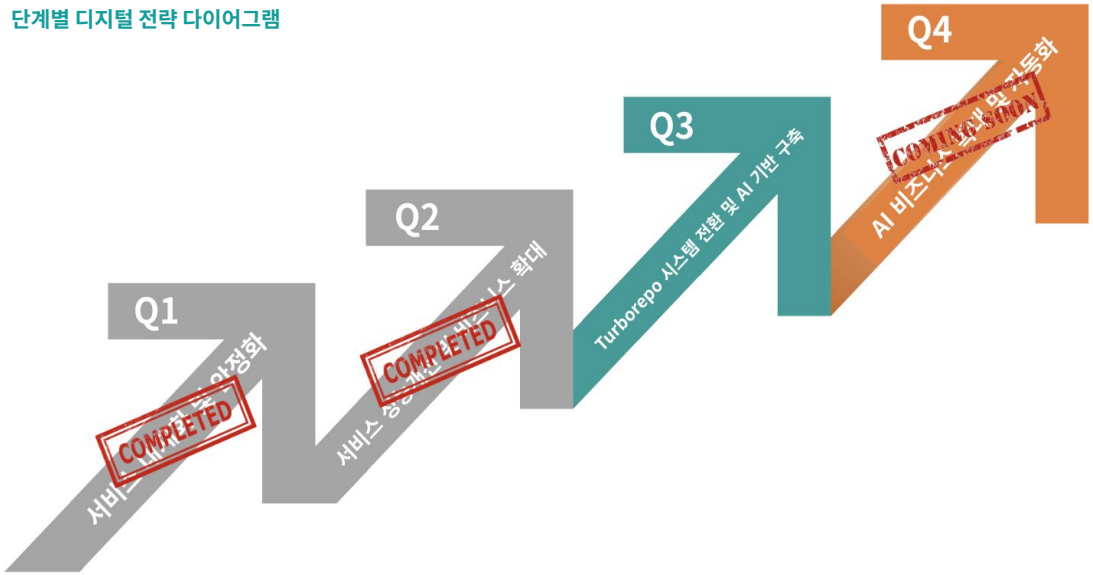
4 프로젝트 노후화로 인한 지원기능 저하 및 보안성 약화

기존 시스템은 의존성 있는 패키지 구조로 인해 서비스 확장에 따라 프로젝트가 자연스럽게 노후화되었습니다. 보안성이 약화되지만 의존성으로 버전을 올리지 못하는 역설이 발생합니다.



EFFECT

단계별 디지털 전략 다이어그램



20%

개선된 업무확장 및 안정성

설계비용 감소 및 안정성 확보가 가능하며 보다
빠른 성능과 체계화된 기술자산 확보

5x

빠른 빌드 속도와 서비스 안정성

기존 대비 5배 이상 빠른 빌드/배포 성능과
애플리케이션 최소 20개까지의 성능유지

30%

프로세스 자동화

디자인-개발/QA/지식자산 프로세스의
업무자동화 효율증대

Q3 단계의 Turborepo 아키텍처 도입에 따른
다음과 같은 효과를 기대합니다.

개발품질 관점

최신 기술 스택을 통해 개발품질을
다음과 같은 측면에서 향상시킵니다.

- 빠른 페이지 렌더링 성능
- 빠르고 안정적인 빌드/배포
- 보다 강력한 보안 취약성 보완
- 정형화된 구조를 통한 코드 품질 개선
- 단위/통합 테스트 및 UI/UX 문서화 자동화를 통한 개선

프로세스 개선 관점

백엔드/디자이너/기획자/보안 담당자/DevOps 엔지니어와의 협업 프로세스를
최소화하고, 이를 통해 프로세스 중복 리소스를 개선합니다.
프로세스 중복 리소스 개선을 통해, 보다 빠른 서비스 오픈을 도와
현업 담당자의 실험적인 비즈니스 환경을 더욱 빠르고 안정적이게 제공합니다.

개발조직 운영 관점

외주 개발인력은 애플리케이션의 단순 서비스 개발/운영으로 배치하고
내부 개발인력은 코어 개발과 R&D 중심의 개발에 집중하여,
AI 와 같은 차세대 기술에 대한 경쟁력을 확보하는 발판이 됩니다.
프로젝트는 AI 로의 교체가 아닌 AI 로의 융합이 될 수 있는 기반이 될 것입니다.

해당 Chapter 에서는 본격적으로 프로젝트의 적용결과를 다양한 시선으로 다루어 봅니다.

프로젝트 적용한 결과를 서비스성능/코드품질/프로세스/기술자산/비용

측면에서 보다 객관적인 지표로 분석합니다.

RESULT

3. 차세대 프로젝트 적용결과

- 프로젝트 적용결과 다각도 분석

RESULT

서비스 성능 결과 분석

42%

페이지 로드 시간 단축

코드 스플리팅과 레이지 로딩 전략 도입으로 초기 페이지 로드 시간이 평균 42% 감소했습니다. 이는 사용자 이탈률 감소와 직접적인 연관이 있습니다.

68%

Core Web Vitals 점수 향상

LCP, INP, CLS 지표 최적화를 통해 Core Web Vitals 통과율이 68% 증가했습니다. 이는 SEO 순위와 사용자 경험에 긍정적인 영향을 미칩니다.

3.2x

서버 응답 속도 개선

캐싱과 서버 사이드 렌더링 최적화를 통해 서버 응답 속도가 3.2배 향상되었습니다. 특히 모바일 환경에서의 성능 개선이 두드러집니다.

성능 최적화를 위해 SSR,CSR 컴포넌트를 사전 구성하고 스켈레톤 UI 적용

화면 렌더링(LCP)와 빠른 응답 전환(INP)를 위해 화면의 서버/클라이언트 렌더링을 이해하여 컴포넌트 구성.

스켈레톤 UI 적용을 API 조회 콘텐츠에 삽입하여, 고객에게 빠른 화면진입 효과 제공

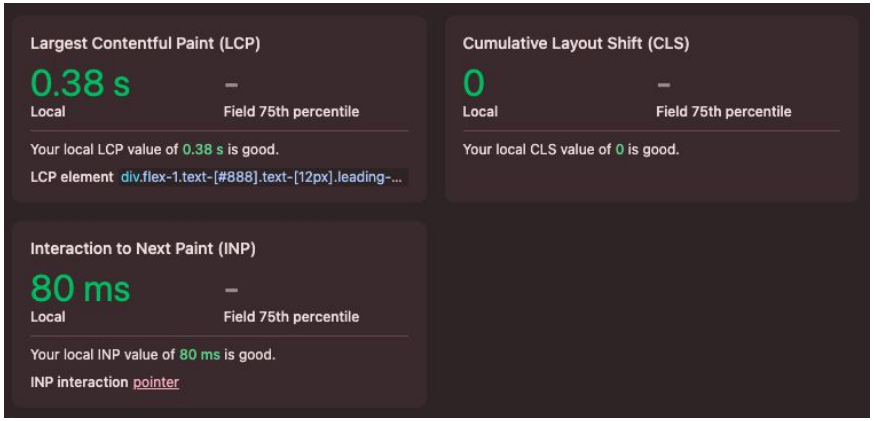
코드 스플리팅을 통한 최소한의 코드 구성을 통해 애플리케이션 경량화

빌드 단계에서 코드 스플리팅을 통해 최소한의 필요한 코드 구성을 통해 애플리케이션을 경량화하여 성능을 개선.

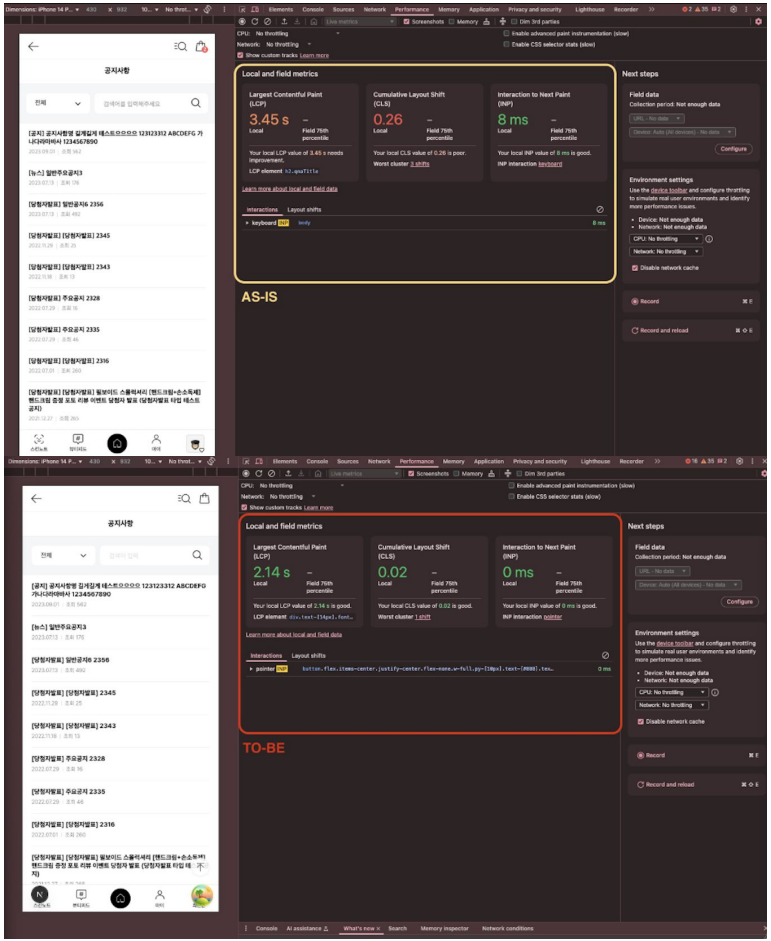
최신 프레임워크 공식 가이드라인 적용을 통해 캐싱과 INP 최적화

프레임워크에서 제공하는 가이드라인 적용을 통해 LCP, INP, CLS 최적화 개선

적절한 Memoization 을 통해 함수 캐싱과 React Query 등을 활용해 API 캐싱 최적화



RESULT



서비스 페이지 성능 결과 비교

해당 프로젝트를 진행하며 서비스 페이지의 성능 개선 확인을 위해
실제 운영되고 있는 페이지와 개선한 페이지의 성능을 비교해 보았습니다.
프론트엔드의 서비스 페이지 대표 지표에 해당하는 LCP, CLS, INP 지표로 확인하였습니다.

1. LCP 대상에 대한 서버사이드 우선 호출로 LCP 지표 개선

사용자에게 가장 빠르게 체감이 되는 LCP 지표를 페이지 개발 시에 확인하여, 이를 API 데이터 상에서 서버사이드 호출로 우선 호출되도록 처리하였습니다. 이를 통해 빠른 LCP 지표 확보와 함께 고객이 보다 빠르게 화면에 접근되는 인식을 느낄 수 있도록 개선하였습니다. **평균 3.4s -> 2.0s** 로 개선되었습니다.

2. 스켈레톤 UI 적용을 통한 빠른 페이지 랜딩 효과와 CLS 지표 개선

기존과 달리 서비스 페이지의 API 조회 특성에 따른 전체 영역 또는 일부 영역의 스켈레톤 UI 적용을 통해 빠르게 레이아웃 윤곽을 그려, 사용자에게 빠른 랜딩 효과를 제공하였습니다. 또한 스켈레톤 UI 를 통한 API 조회 영역에 대한 레이아웃 사전 확보로 CLS 성능이 **평균 0.4 -> 0.02** 로 개선되었습니다.

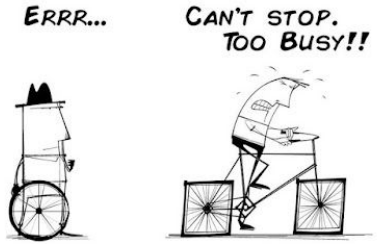
3. 화면 레이아웃에 대한 캐싱을 적절히 적용하여 INP 지표 개선

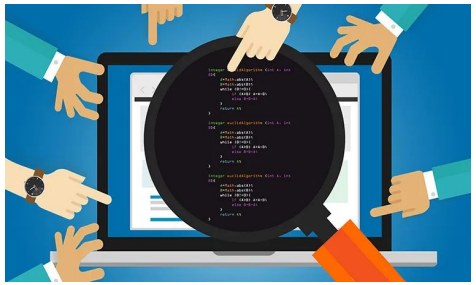
Interaction 에 의해 수치가 결정되는 INP 지표 개선을 위해, Memorize 등 레이아웃과 컴포넌트에 대한 캐싱을 적절히 활용하여 INP 수치를 개선하였습니다. 화면의 복잡도가 더욱 클수록 해당 설계없이 구성하면 INP 지표는 많이 저하되기 때문에, 이를 고려하여 컴포넌트 구성 및 데이터의 캐싱을 처리하였습니다.

(참고로, AS-IS 는 기존 레거시 Staging 화면이며, TO-BE 는 차세대 프로젝트에서 실행한 Local 화면입니다)

RESULT

코드품질 결과 분석





78%

코드 커버리지

단위 테스트 및 통합 테스트를 통한 코드 커버리지가 45%에서 78%로 증가했습니다. 핵심 비즈니스 로직은 90% 이상의 커버리지를 달성했습니다.

91점

정적 분석 점수

ESLint, TypeScript, SonarQube를 통한 정적 분석 점수가 62점에서 91점으로 향상되었습니다. 잠재적 버그와 보안 취약점이 대폭 감소했습니다.

35%

기술부채 감소

최신 아키텍처 구성과 라이브러리 도입으로 커머스 비즈니스에서 요구하는 복잡한 요구사항을 기술적으로 해결하였습니다. 기술 부채가 35% 감소했습니다. 이는 향후 기능 개발 속도와 안정성 향상에 기여합니다.

75%

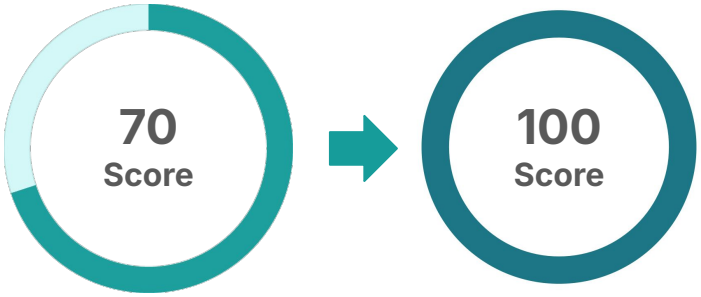
리뷰 시간 감소

표준화된 코딩 스타일과 네이밍 규칙을 도입하여 코드 일관성이 크게 향상되었습니다. 자동화된 포맷팅으로 스타일 관련 리뷰 시간이 75% 감소했습니다.

코드 품질 개선은 단순한 기술적 성취를 넘어 실질적인 비즈니스 가치를 창출합니다.

버그 발생률이 65% 감소하였고, 기능 개발 속도는 평균 40% 향상되었습니다.

RESULT



Lighthouse 측정을 통한 개발된 페이지 점수

1 GA4 최적화 구현 효과

@next/third-parties 라이브러리를 활용한 성능 최적화된 GA4 구현으로 Core Web Vitals에 미치는 영향을 최소화했습니다. 이벤트 추적 자동화를 통해 사용자 행동 데이터 수집의 일관성과 안정성이 향상되었습니다.
또한 스크립트 로드전략을 통해 화면 렌더링과 분리하여, 화면 성능을 최적화하고, GA 스크립트를 안정화하였습니다.

2 데이터 품질 모니터링 시스템

GA4 데이터 품질을 실시간으로 모니터링하고 이슈 발생 시 즉시 알림을 받을 수 있는 시스템을 구축했습니다.
이를 통해 신규 기능 출시 시 데이터 수집의 정확성을 즉시 검증할 수 있게 되었습니다.

인덱싱 개선을 통한 검색 최상단 올리기

Next.js의 서버 사이드 렌더링을 통해 완전히 렌더링된 HTML을 검색엔진에 제공하여 인덱싱 효율을 기존대비 20% 높였습니다.

기존 기술이슈로 지원 못하던 20% 페이지에 대해 인덱싱 효율을 통해 검색 최상단으로 올리는 효과가 있습니다.

구조화된 데이터 구현을 통한 스니펫 상승

schema-dts 라이브러리로 TypeScript 기반 JSON-LD를 구현하여 리치 스니펫 노출 기회를 증가시켰습니다.

기존 검색결과 1순위 웹사이트보다 클릭률이 30% 이상 높습니다. 또한 이로인해 고객의 체류 시간이 길고, 반환율이 높은 결과가 나왔습니다.

동적 메타데이터 최적화

Next.js의 동적 메타데이터 기능으로 페이지별 최적화된 태그를 생성하고 자동 업데이트합니다.

검색결과에서 페이지의 노출을 늘리는 효과가 있습니다.

RESULT

AS-IS

1 기존 프로세스

디자인 확인 → **마크업 개발(HTML) → 프론트엔드 개발(React 변환)** → **수동** 테스트 → 문서 **수동** 업데이트

중복 작업과 반복적인 수정으로 인해 단일 기능 개발에 **평균 5-7일** 소요

TO-BE

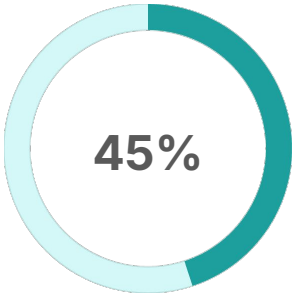
2 **개선된 프로세스**

디자인 확인 → **컴포넌트 기반 직접** 개발 → **자동화** 테스트 →문서 **자동** 업데이트

중간 단계 제거와 자동화를 통해 동일 기능 개발에 기존 대비 **평균 2-3일 단축**

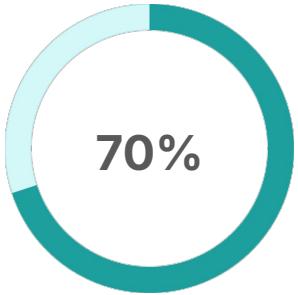
개발 시간 단축

중복 프로세스 제거 및 문서 자동 업데이트를 통한 개발 능률 효율화



피드백 반영 속도 향상

컴포넌트 단위 개발로 수정 사항 빠른 반영



개발자 만족도 증가

효율적인 프로세스로 개발 경험 개선

RESULT

개발-QA 개선 결과 분석

QA 진행단계

1 NEW **Jest 기반 단위테스트 자동화 단계(개발자)**

Jest 기술을 활용하여 유틸리티 함수 등을 테스트코드 작성

테스트코드 평균 개발 10분

단위 테스트 커버리지 100%

2 NEW **Cypress 기반 통합테스트 자동화 단계(개발자)**

QA 테스트 엑셀 시나리오 기반 E2E 테스트 코드 작성

테스트코드 평균 개발 30분

비즈니스 테스트 커버리지 60%

3 **QA 검증 단계(QA)**

1-2단계 단순 테스트 패스 이후 복합 비즈니스 중심의 테스트 진행

단순 기능 및 동작 테스트가 제외되므로

테스트 공수 대폭 축소

QA 인력은 복합 비즈니스 중심의 테스트 진행을 통해 **QA 품질 증가**

4 **배포 및 운영**

3단계 테스트 진행 후 발생하는 케이스 또한 AI 학습을 통해 커버리지 확대

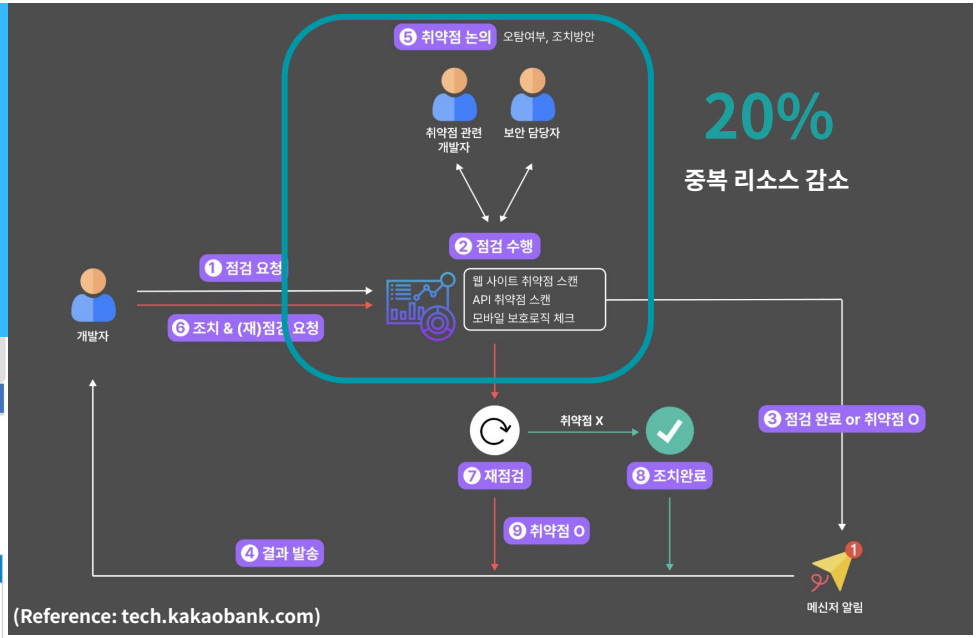
자동화된 테스트 파이프라인 구축으로 **QA 공수가 60% 감소하면서도 테스트 깊이는 오히려 45% 확대**되었습니다.

개발자가 코드 작성 시점에 품질을 검증할 수 있게 되어 결함 발견 시점이 크게 앞당겨졌습니다.

QA 는 복합 비즈니스 위주의 테스트 진행을 통해, QA 품질이 개선됩니다.

RESULT

보안-QA 개선 결과 분석



프로젝트 구축 시, 보안성점검은 필수입니다. 다만 기존 프로젝트들은 애플리케이션과 인프라시스템 레벨에 대해 보안성검토가 중복 발생합니다.

Turborepo 는 같은 인프라 레이어 내에 여러 애플리케이션이 구축되므로, **공통의 인프라 환경을 제공하며 애플리케이션 레벨에서는 인증, 암호화 등 유틸리티를 공유**합니다.

이는 검토 받은 보안성점검을 기반으로 신규 추가되는 애플리케이션 중복체크리스트 부담을 줄여 줍니다.

보안관리자와 개발자 상호간의 보안성검토 중 단순리스트 검토에 대한 중복 리소스를 20% 줄여주고, 더욱 심화된 보안에 집중할 수 있도록 도와줍니다.

RESULT

지식자산 개선 결과 분석

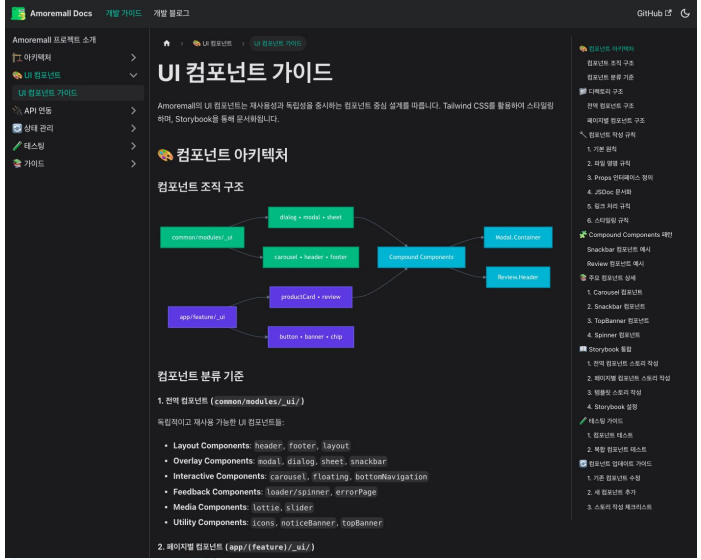
코드 작성 → 자동 동기화 → 웹페이지 형식 문서 생성

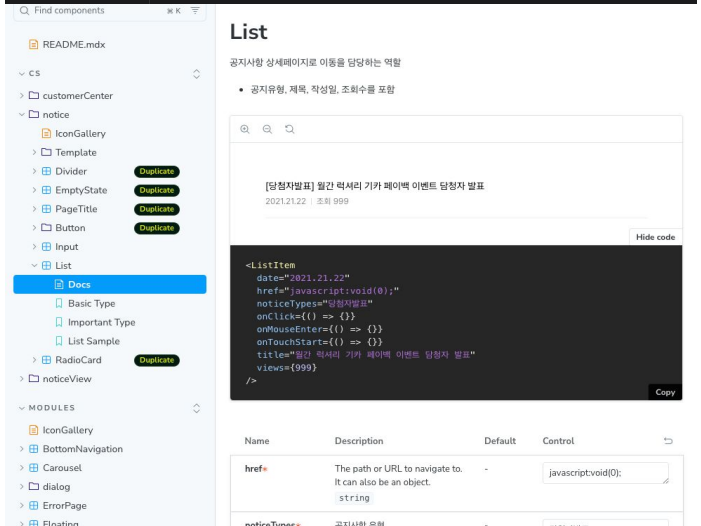
AS-IS

프로젝트 개발 이후, Confluence, Jira 등의 특정 페이지에
개발내용과 개발과정 등을 별도로 작성하고 관리하였습니다.
이는 운영을 진행하며 발생하는 개발자의 성향, 업무 강도, 배포 시급도 등의 다양한 상황에 영향을 받아 문서의 일관성을 잃고, 누락이 되기도 하는 등 문서의 품질을 저하시키는 요소로 작용하였습니다.

TO-BE

개발자는 개발을 진행하며 별도의 문서 작성없이,
개발 코드의 주석을 정형화된 형식(JSDoc)에 맞게 작성합니다.
해당 주석을 기반으로 자동 동기화되어, **개발자/디자이너/기획자/IT PMO 등 유관 담당자들이 쉽게 보고 이해하기 편한 문서로 자동 동기화**해줍니다.
이를 통해 특별한 문서작성 시간이 별도로 존재하지 않아, **문서 작성 공수 평균 2 md 감소시킵니다.**
또한 정형화된 양식을 제공하여, 업데이트 되므로 **최적화된 문서품질을 유지**합니다.





해당 Chapter 에서는 프로젝트의 한계점과 극복해야할 과제를 통해 다음 청사진을 그려봅니다.

프로젝트가 지니는 한계점과 극복해야할 과제를 통해

보다 객관적인 시선에서 앞으로의 발전방향을 모색합니다.

NEXT STEP

4. 차세대 프로젝트 TRADE-OFF 및 TO-DO

TRADE-OFF

Turborepo 아키텍처와 신규 개발 방식 도입에 따른 장단점을
다음과 같이 균형 있게 고찰하고 객관적으로 평가했습니다.

초기 투자 비용 증가

Trade-Off: 아키텍처 설계와 초기 개발에 상당한 투자가 필요했으며, 팀의 학습 곡선도 고려해야 했습니다.

대응 전략: 단계적 도입과 명확한 ROI 분석을 통해 투자 정당성을 확보했습니다.
비교적 영향도가 적은 서비스를 시작으로 전략적으로 적용하였으며,
팀 교육과 지식자산 시스템을 구축하여 학습 곡선을 완화했습니다.

기술 복잡성 증가

Trade-Off: Monorepo 구조와 새로운 도구 도입으로 인한 기술적 복잡성이 증가했습니다.

대응 전략: 철저한 문서화와 내부 교육을 통해 복잡성을 관리하고, 자동화 도구를 활용하여 개발자 경험을 개선했습니다. 복잡한 부분은 내부 개발자가 담당하고 외주 개발자에게는 단순화된 인터페이스를 제공했습니다.

유연성과 표준화 균형

Trade-Off: 표준화된 접근 방식은 일관성을 높이지만 유연성을 제한할 수 있습니다.

대응 전략: 코어 패키지는 표준화하되 애플리케이션 레벨에서는 의존성을 최소화하고 독립적인 서비스로 구성하여, 유연함과 확장성을 강조하였습니다. 표준에서 벗어나야 하는 예외 케이스를 위한 명확한 프로세스를 정의했습니다.

의존성 관리 복잡성

Trade-Off: 공유 패키지의 버전 관리 및 의존성 관계가 복잡해질 수 있습니다.

대응 전략: 패키지의 명확한 버전 관리 정책과 Changesets 도구를 도입하여 의존성 관리를 자동화하고 체계화하였습니다. 패키지 인터페이스의 안정성을 유지하고 하위 호환성을 보장하는 원칙을 수립했습니다.
일반적으론 의존성이 아닌 독립성을 강조하여 의존 관계를 최소화하였습니다.

TO-DO

AI 테스트 자동화 설계는 Q4 단계에 적용예정이며, BY 27-28(2026년말) 을 계획하고 있습니다.

AI 테스트 코드 생성 프로세스

테스트 요구사항 정의

개발자는 기획서의 스크립트 또는 QA가 테스트하는 엑셀 시나리오를 데이터로 변환

AI Prompt 생성

해당 데이터를 포함하여 테스트 요구사항을 AI가 이해할 수 있는 형식의 프롬프트로 변환

3 테스트 코드 자동 생성

AI가 단위 테스트 또는 통합 테스트 코드를 생성

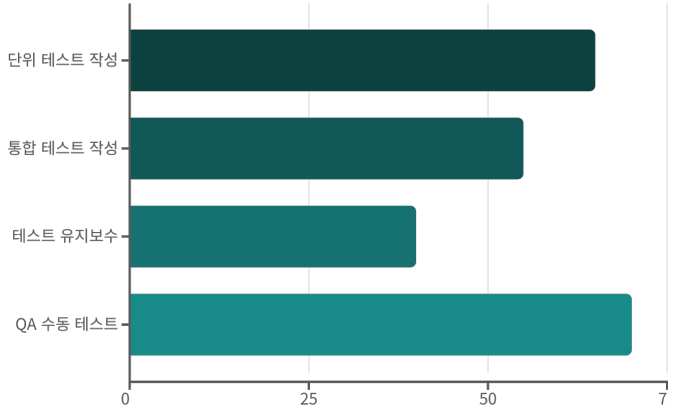
검토 및 보완

개발자가 생성된 코드를 검토하고 필요시 수정

테스트 실행 및 결과 분석

자동화된 테스트 실행 및 결과 보고

자동화 예상효과



AI 기반 테스트 자동화를 통해 테스트 코드 작성 시간이 **평균 60% 단축**되었으며,

테스트 커버리지는 35% 증가했습니다.

특히 반복적인 테스트 패턴에서 높은 효율성을 보였습니다.

개발자 생산성 향상

테스트 코드 작성에 소요되는 시간이 크게 줄어 개발자가 비즈니스 로직 구현에 더 집중할 수 있게 되었습니다. 이는 전체적인 개발 생산성 향상으로 이어졌습니다.

테스트 자동화율

AI 기반 테스트 코드 생성으로 인해 테스트 자동화율이 크게 증가했습니다. 이전에는 리소스 부족으로 자동화하지 못했던 영역까지 테스트 범위가 확대되었습니다.

QA 공수 감소

반복적인 테스트가 자동화되어 QA 팀이 탐색적 테스트와 사용자 경험 검증에 더 집중할 수 있게 되었습니다. 이는 전반적인 품질 향상으로 이어졌습니다.

TO-DO

Figma MCP 활용을 통한 디자인-개발 연동 자동화 설계

Figma MCP 기반의 자동화 설계는 Q4 단계에 적용예정이며, BY 28-29(2026년말) 을 계획하고 있습니다.

토큰 기반 디자인 시스템 구축

디자인 토큰(색상, 타이포그래피, 간격 등)을 중앙 저장소에서 관리하고 Figma 변수와 연동하여 일관된 시각 언어를 유지합니다. JSON 형식으로 추출하여 개발 환경에 자동 동기화합니다.

컴포넌트 변환 자동화 파이프라인

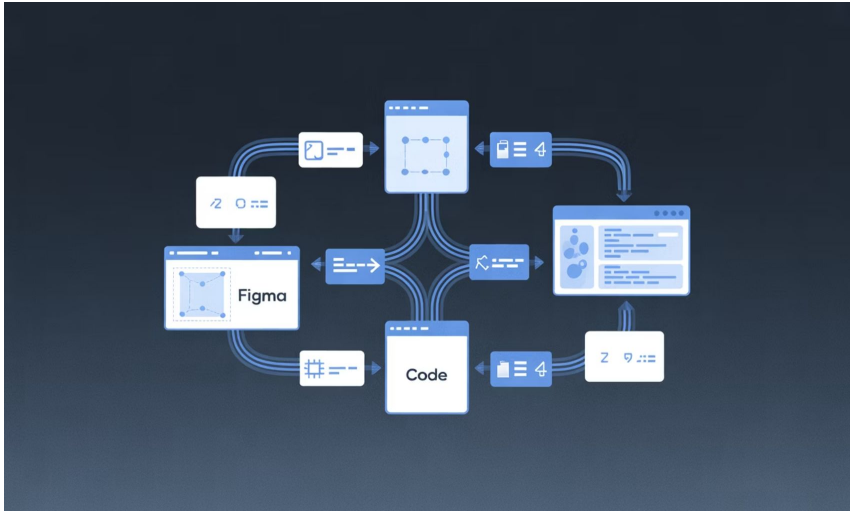
Figma API를 활용하여 디자인 컴포넌트의 구조 및 속성을 추출하고, 맞춤형 스크립트를 통해 React 컴포넌트로 변환합니다. 프레임워크별 최적화된 코드 생성이 가능합니다.
다만, 아직까진 그 정확도가 높지 않으므로, 정확성을 높이기 위한 기술 접근이 중요합니다.

CI/CD 통합 및 버전 관리

Git 워크플로우와 통합하여 디자인 변경사항을 자동으로 감지하고 개발 브랜치에 반영합니다. 변경 이력 추적 및 롤백 기능을 통해 안정적인 디자인 시스템 관리가 가능합니다.

디자인-개발 워크플로우 자동화의 기술적 접근

Figma MCP(Multi-player Collaboration Platform)는 디자인 시스템과 개발 프로세스를 유기적으로 연결하는 강력한 도구입니다.
디자인 토큰에서 코드 컴포넌트까지 자동화된 파이프라인을 구축함으로써 팀의 생산성을 획기적으로 향상시킬 수 있습니다.
이러한 자동화 파이프라인을 구축함으로써 디자인 변경사항이 개발 환경에 즉시 반영되어 개발자의 수동 코딩 시간을 **최대 78%** 절감할 수 있습니다.
또한 디자인-개발 간 불일치 문제를 해소하여 **품질 향상 및 출시 주기 단축에 기여**합니다.
효율적인 MCP 자동화 설계를 위해서는 팀 내 디자인 시스템 거버넌스 확립과 명확한 네이밍 컨벤션 수립이 선행되어야 합니다.



Figma MCP 기반 자동화 파이프라인 구조도

78%

개발 시간 절감

수동 코드 작성 대비

95%

디자인-코드 일치율

자동화 시스템 적용 후

3.5x

반복 작업 효율성

기존 워크플로우 대비

해당 Chapter 에서는 설계한 아키텍처 구조와 상세한 기술중심의 코드 설명을 다룹니다.

프로젝트의 전체 아키텍처 및 주요 영역별 폴더구조에 대해 왜 이렇게 구성하였는지를

설명하며 이를 통해 얻는 이점에 대해 기술위주로 설명합니다.

페이지마다 우측 상단에 기술관련 담당자 색인 마크를 포함하고 있습니다.

관련 담당자분들은 색인을 기준으로 집중해서 보시면 좋습니다.

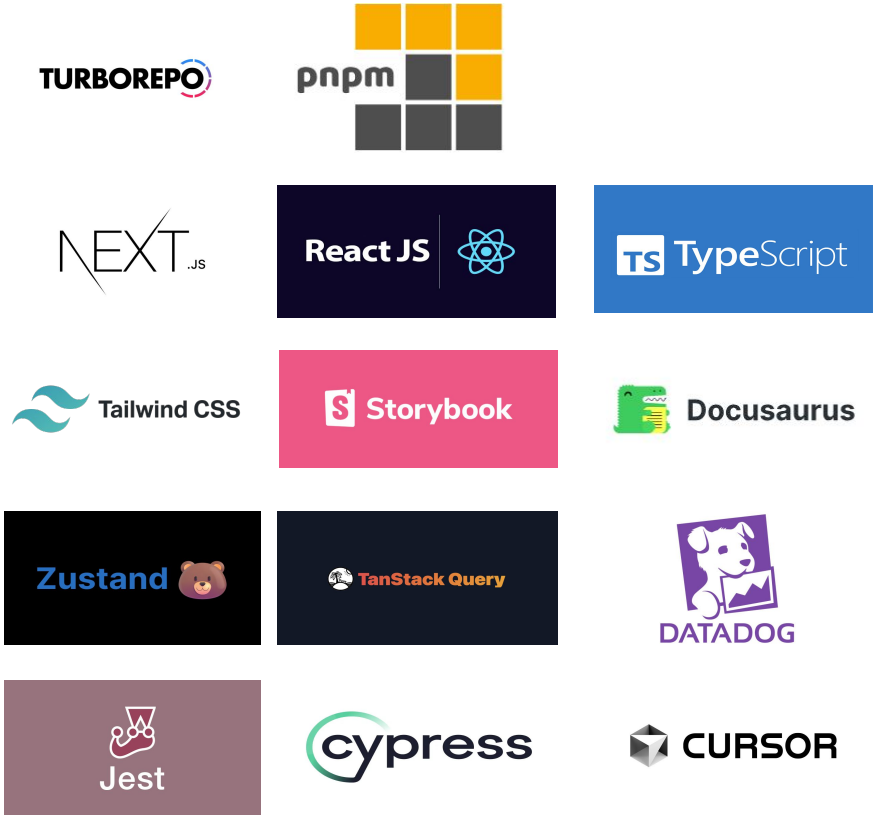
APPENDIX

5. 차세대 프로젝트 상세기술 부록

- 프로젝트 기술 상세설명

TECH 주요 기술스택

FE 개발자 | BE 개발자 | IT PMO
디자이너 | 기획자 | QA



무한한 서비스 확장 및 효율적인 관리

- Turborepo 아키텍처를 통해 여러 프로젝트를 단일저장소에 관리가 가능해짐.
- 이를 통해 프로젝트간 다양한 패키지 공유가 가능해짐.
- 코드컨벤션을 통일할 수 있어 코드품질/서비스품질 개선에 효과적.
- 패키지 공유를 통해 초기 구축 비용 단축이 가능해짐.
- PNPM 패키지매니저를 통해 빌드/배포 성능을 최적화함.
- 다양한 프로젝트가 단일저장소에 관리됨으로써 발생하는 부하를 최소화.

지식자산화 품질 개선과 자동화

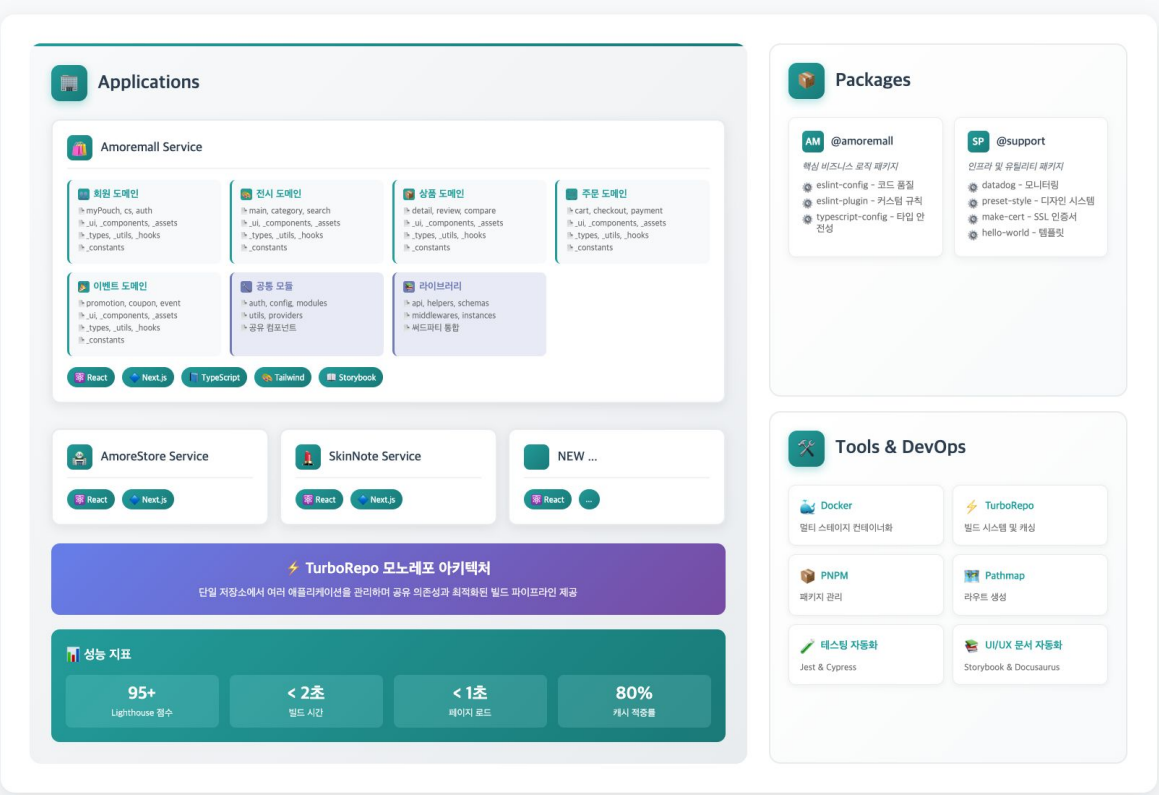
- Storybook, Docusaurus 통해 개발환경 지식자산화가 자동화됨.
- 개발한 내용을 토대로 자동 동기화함으로써, 중복되는 지식자산 공수 및 수동 업데이트로 인한 누락을 방지하며 품질 개선에 효율적임.
- Storybook 은 디자이너 협업 시, Figma 연계.
- Docusaurus 는 PMO 협업 시, 개발 산출물 공수 축소에 효과적임.
- Cursor AI 의 Prompt 기능을 결합하여, 지식자산을 정형화함.

테스트코드를 통한 코드/서비스 품질 개선

- Jest, Cypress 통해 기능 구현에 대한 단위 품질테스트, 서비스 품질테스트가 개발단계에서 진행되어 코드/서비스 품질 개선됨.
- QA 조직의 시나리오 스크립트를 Cursor AI 에 학습하여, E2E 테스트를 자동화할 수 있음. 테스트 품질 개선에 효과적.

TECH 아키텍처 구조

FE 개발자



Turborepo 프로젝트 아키텍처 구조도 상세

1 Applications

Turborepo를 활용해 여러 Application을 하나의 저장소에서 관리합니다. 각각의 Application은 독립적이며, 서로 의존적이지 않습니다. 이를 통해 신규 서비스를 빠르게 기획/오픈하고 안전하게 실험할 수 있습니다.

Application 내부는 도메인들끼리도 독립적이며, 서로 의존하여 import하지 않고 중복을 허용함으로써 서비스의 확장성을 토대로 영향도를 최소화하고 risk를 감소시킵니다.

2 Packages

Applications에서 사용하는 코어 라이브러리/패키지입니다. 해당 패키지는 코어의 역할을 하므로, 외주 개발자가 아닌 내부 개발직원이 버전별로 관리합니다.

@application, @support에 추가하여 공통 코어 관리를 합니다. 이는 패키지 배포를 통해 회사의 핵심기술 코어 자산으로 활용합니다.

3 Tools & DevOps

Docker를 통해 배포/관리합니다. Docker의 구성은 터보레포 환경 레이어 / 어플리케이션 레이어 / 패키지 레이어 등 3중 레이어로 빌드/배포합니다.

패키지매니저를 PNPM으로 활용하여, YARN의 Ghost Dependency 이슈를 방지하고 최적화합니다. Jest와 Cypress를 통해 테스트를, Storybook과 Docusaurus를 통해 문서 자동화를 관리합니다.

FE 개발자

TECH 아키텍처 구조

Turborepo 아키텍처

1. TurboRepo 기반 모노레포 설계

단일 저장소에서 여러 서비스와 패키지를 효율적으로 관리하는 현대적 모노레포 아키텍처를 구축했습니다. 증분 빌드와 지능형 캐싱을 통해 변경된 부분만 빌드하여 개발 생산성을 극대화하며, 의존성 그래프 기반 병렬 빌드로 전체 빌드 시간을 대폭 단축시켰습니다.

2. Apps 독립 서비스 컨테이너

아모레몰, 아모레스토어 등 각 이커머스 서비스를 완전히 독립된 애플리케이션으로 구성했습니다. 각 앱은 자체 빌드설정, 환경변수, 배포 파이프라인을 가지며 서로 간섭없이 독립적으로 개발하고 배포할 수 있습니다. 공통 패키지만 선택적으로 공유하여 서비스간 결합도를 최소화했습니다.

3. Packages 계층형 공통 라이브러리

@amoremall과 @support 등 네임스페이스로 분리된 이중 패키지 시스템을 구축했습니다. 아모레몰 생태계 전용 도구와 범용 지원도구를 명확히 구분하여 관리하며, 각 패키지는 Vite 기반 ESM 빌드로 최적화된 배포 환경을 제공합니다. 버전 관리와 의존성 추적이 자동화되어 안정적인 패키지로 생태계를 운영합니다.

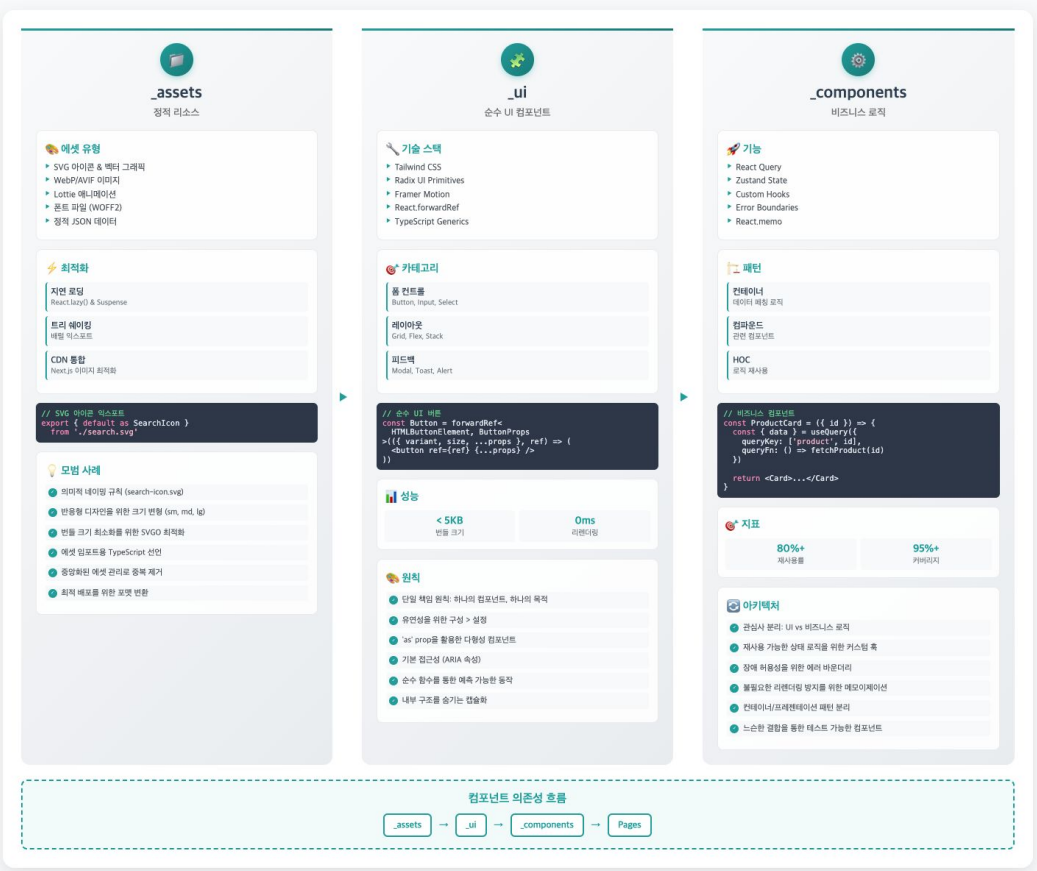
4. Docs 서비스별 독립 문서화

각 서비스마다 독립된 Docusaurus 기반 문서화 시스템을 구성하여 서비스별 맞춤형 개발 가이드를 제공합니다. JSDoc 자동 추출과 Mermaid 다이어그램을 통한 시각적 아키텍처 문서화가 가능하며, 코드 변경 시 문서가 자동으로 업데이트되어 개발 문서와 실제 코드 간의 일관성을 보장합니다.

5. DevOps 자동화 인프라

Docker 멀티스테이지 빌드와 TurboRepo 캐싱을 결합한 최적화된 배포 시스템을 구축했습니다. Prune 단계에서 불필요한 코드를 제거하고, BuildKit 캐시 마운트로 빌드 속도를 향상시키며 앱 타입 자동 감지를 통해 Next.js와 React 앱을 구분하여 최적의 실행 환경을 자동 구성합니다.

TECH UI/UX 컴포넌트 구조



Application 내부의 UI/UX 컴포넌트 구성 상세

1 _assets

application 에서 사용하는 이미지, 파비콘 등의 자산을 등록하여 사용하되 경량화를 통해 최적화합니다.
해당 서비스에서는 Next.js 에서 제공하는 next/image 를 활용하여 이미지 성능 최적화를 진행하였습니다.

2 _ui

Zeplin, Figma 디자인을 토대로 구현한 순수 UI 컴포넌트이며, [UiComponent].tsx, [UiComponent].stories.tsx 세트구성됩니다.

[UiComponent].tsx 는 _components 에 비즈니스 로직 개발에 활용되며, [UiComponent].stories.tsx 는 Storybook 에 적용되어 디자이너와의 커뮤니케이션에 활용합니다.

네이밍과 컴포넌트 구성의 기준은 Material Design, shadcn/ui, chakra-ui 등 12 개의 공식 Document 를 참고하였습니다.

기획/디자인/개발 등의 네이밍 룰 통일을 통해 개발/운영 시 커뮤니케이션 miss 를 최소화하며, 회사의 공식 네이밍 자산으로 기반을 쌓았습니다.

_ui 컴포넌트에서는 기능을 규정하지 않기에, props 로 해당 컴포넌트의 기능을 제한하기 보단 확장성을 우선하여 다형성 컴포넌트 및 rest 형태의 자유로운 props 로 구성하였습니다.

3 _components

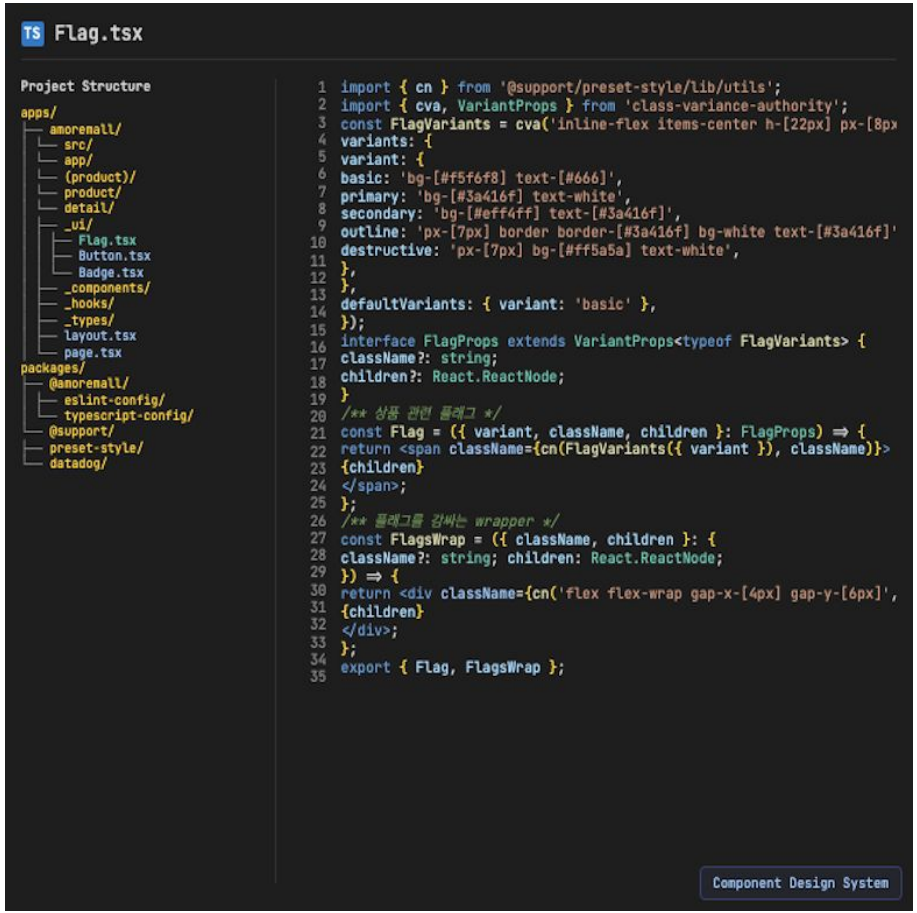
비즈니스 로직을 녹인 실제 페이지에 활용하는 컴포넌트입니다.

해당 프로젝트는 의존성이 큰 디자인 시스템 컨셉이 아닌, 의존성이 적으나 확장성이 용이한 컨셉으로 프로젝트를 구성하였습니다.

'_ui 컴포넌트들 + 비즈니스 로직'으로 구성됩니다.

TECH UI/UX 컴포넌트

FE 개발자



{page}/_ui/{component}

1. 페이지별 UI 격리 설계

각 페이지마다 독립적인 UI 폴더를 구성하여 서비스 간 의존성을 완전히 차단합니다. 커머스 5대 서비스 (고객/전시/이벤트/주문/상품)별로 UI 컴포넌트를 격리하여 관리하며, 동일한 기능이라도 서비스별로 독립 구현하여 확장성과 유지보수성을 확보했습니다.

2. 3계층 컴포넌트 아키텍처

UI 관리를 3개 계층으로 분리하여 관심사를 명확히 구분했습니다. 순수 UI 컴포넌트는 스타일링만 담당하고, 비즈니스 로직 컴포넌트는 UI를 조합하여 실제 기능을 구현하며, 리소스 파일은 별도 관리하여 역할과 책임을 분명히 했습니다.

3. 컴포넌트 컴포지션 패턴

대형 컴포넌트를 작은 단위로 분해하여 조합 가능한 구조로 설계했습니다. 각 컴포넌트는 단일 책임을 가지며, 필요에 따라 자유롭게 조합하여 다양한 UI 패턴을 구현할 수 있습니다. 이를 통해 재사용성과 유연성을 동시에 확보하였습니다.

4. 타입 안전 다형성 컴포넌트

하나의 컴포넌트가 다양한 HTML 요소나 React 컴포넌트를 렌더링될 수 있도록 설계했습니다. 버튼이 링크로도 작동하고 일반 버튼으로도 작동하는 등 유연한 사용이 가능하면서도 Typescript 를 통해 타입 안전성을 보장합니다.

5. 업계 표준 네이밍 컨벤션

Material UI, Chakra UI 등 주요 UI 라이브러리 네이밍 패턴을 준수하여 개발자 친화적인 인터페이스를 제공합니다. 일관된 네이밍으로 개발 생산성을 향상시켰습니다.

6. 서비스 독립성 보장

대규모 서비스에서 발생할 수 있는 의존성 문제를 원천 차단하기 위해 서비스 간 컴포넌트 공유를 금지했습니다. 각 서비스는 독립적으로 발전할 수 있으며, 한 서비스의 변경이 다른 서비스에 영향을 주지 않아 안정적인 운영이 가능합니다.

7. 현대적 개발 스택 적용

TailWind CSS로 일관된 디자인 시스템을 구축하고, TypeScript 로 타입 안전성을 확보했습니다. px 단위 사용으로 디자인 가이드와 정확히 일치하는 구현을 보장하며, JSDoc 을 통한 자동문서화로 개발자 경험을 최적화했습니다.

TECH UI/UX 컴포넌트

FE 개발자

TopBanner.tsx

```
Project Structure
apps/
  ├── amoremall/
  │   ├── src/
  │   ├── app/
  │   ├── (product)/
  │   ├── product/
  │   ├── detail/
  │   ├── common/
  │   ├── modules/
  │   │   ├── ui/
  │   │   └── topBanner/
  │   │       ├── TopBanner.tsx
  │   │       └── TopBanner.stories.tsx
  │   ├── header/
  │   ├── footer/
  │   ├── icons/
  │   └── image/
packages/
  ├── @amoremall/
  │   ├── eslint-config/
  │   └── typescript-config/
  ├── @support/
  ├── preset-style/
  └── datadog/
```

```tsx
import { Link } from '@/common/modules/_ui/link';
import { cn } from '@support/preset-style/lib/utils';
import { SvgIconCloseS24 } from '../icons/svgIcon/index';
import { Image } from '@/common/modules/_ui/image';
import React from 'react';
/**
 * 배너 닫기 버튼 컴포넌트
 * - 상단 배너를 닫을 수 있는 버튼을 제공합니다.
 * - 접근성을 위해 스크린 리더용 텍스트를 포함합니다.
 * @example
 *   tsx
 *   <TopBannerClose onClick={() => console.log('배너 닫기')} />
 *
 */
const TopBannerClose = ({ ...rest }: React.ButtonHTMLAttributes<HTMLButtonElement>) => {
return (
<button
type="button"
{...rest}
className="flex-none w-[44px] h-full hover:bg-black/10 transition-colors duration-200"
aria-label="배너 닫기"
>
<SvgIconCloseS24 width={24} height={24} aria-hidden="true" className="mx-auto" />
<span className="sr-only">닫기</span>
</button>
);
};
/** 배너 이미지 링크 컴포넌트 Props */
interface TopBannerImageLinkProps extends React.ComponentProps<typeof Link> {
/** 링크 URL */
href: string;
/** 이미지 속성 */
imageProps: React.ComponentProps<typeof Image>;
}
/**
 * 배너 이미지 링크 컴포넌트
 * - 배너 이미지와 링크 기능을 결합한 컴포넌트입니다.
 * - Next.js Image 컴포넌트를 사용하여 최적화된 이미지를 제공합니다.
 * @example
 *   tsx
 *   <TopBannerImageLink
 *   href="/promotion"
 *   imageProps={{
 *   src: "/banner.jpg",
 *   alt: "프로모션 배너"
 *   }}
 *   />
 *
 */
const TopBannerImageLink = ({ href, imageProps, ...rest }: TopBannerImageLinkProps) => {
<Link
href={href}
{...rest}
className="flex flex-1 items-center justify-start relative hover:opacity-90 transition-opacity duration-200"
>
<Image {...imageProps} width={331} height={44} className="w-full h-full object-contain object-left" priority />
</Link>
);
/** 상단 배너 컨테이너 Props */
interface TopBannerContainerProps {
/** 배경 컬러 (예: #ff0000) */
isFullscreen?: boolean;
/** 닫기 버튼 표시 여부 */
bgColor?: string;
/** 배너 이미지, 닫기버튼 등을 포함 */
children: React.ReactNode;
}
/**
 * 상단 배너 컨테이너 컴포넌트
 * - 화면 최상단에 위치한 프로모션, 앱 다운로드 유도 배너의 컨테이너입니다.
 * - CSS 변수를 사용하여 높이를 관리하고, 반응형 디자인을 지원합니다.
 * @example
 *   tsx
 *   <TopBannerContainer bgColor="#ff0000">
 *   <TopBannerImageLink href="/promo" imageProps={{...}} />
 *   <TopBannerClose onClick={handleClose} />
 *   </TopBannerContainer>
 *
 */
const TopBannerContainer = ({ children, bgColor, isFullscreen = false }: TopBannerContainerProps) => {
<aside
className={cn(
'sticky top-0 z-1020 w-full h-[var(--top-banner-height)] mx-auto bg-[#171b39] top-banner peer',
isFullscreen ? '' : 'max-w-screen-lg',
)}
style={{ backgroundColor: bgColor }}
>
<div className="flex w-full h-full max-w-screen-lg mx-auto">{children}</div>
</aside>
);
export { TopBannerClose, TopBannerImageLink, TopBannerContainer };
```

Banner Component System

common/modules/_ui/{component}

1. 비즈니스 무관 공통 UI 집중화

서비스별 독립성을 유지하면서도 전체 애플리케이션에서 공통으로 사용되는 순수 UI 컴포넌트만을 별도 관리합니다.

2. 전역 사용 컴포넌트 아키텍처

헤더, 푸터, 네비게이션, 모달, 캐러셀 등 애플리케이션 전반에서 사용되는 핵심 UI 컴포넌트들을 체계적으로 관리합니다. 각 컴포넌트는 독립적인 폴더 구조를 가지며, 재사용성과 확장성을 고려한 설계로 다양한 상황에서 유연하게 활용할 수 있습니다.

3. 고도화된 인터렉션 구현

Motion 라이브러리를 활용한 부드러운 애니메이션과 사용자 상호작용을 구현했습니다. 공지사항의 자동 롤링, 마우스 호버 시 일시정지, 캐러셀의 드래그 제스처 등 사용자 경험을 향상시키는 세밀한 인터렉션 디테일을 적용했습니다.

4. 모듈형 컴포지션 설계

각 컴포넌트를 작은 단위로 분해하여 조합 가능한 구조로 설계했습니다. 공지사항 배너의 경우 아이템, 롤링 영역, 더보기 버튼, 컨테이너가 독립적으로 구성되어 필요에 따라 자유롭게 조합하여 다양한 UI 패턴을 구현할 수 있습니다.

5. 상태 관리 통합 Client 컴포넌트

사용자 상호작용이 필요한 복잡한 UI 로직을 효율적으로 관리하기 위해 Client 컴포넌트로 구현했습니다. 자동 재생 제어, 드래그 상태 추적, 애니메이션 동기화 등의 상태를 안정적으로 관리하여 일관된 사용자 경험을 제공합니다.

6. 범용 설정 옵션 시스템

다양한 사용 사례에 대응하기 위해 풍부한 설정 옵션을 제공합니다. 캐러셀의 경우 슬라이드 폭, 정렬 방식, 무한 루프, 자동 재생 주기, 드래그 민감도 등을 세밀하게 조정할 수 있어 각 페이지의 요구사항에 맞게 커스터마이징이 가능합니다.

7. 일관된 문서화 표준

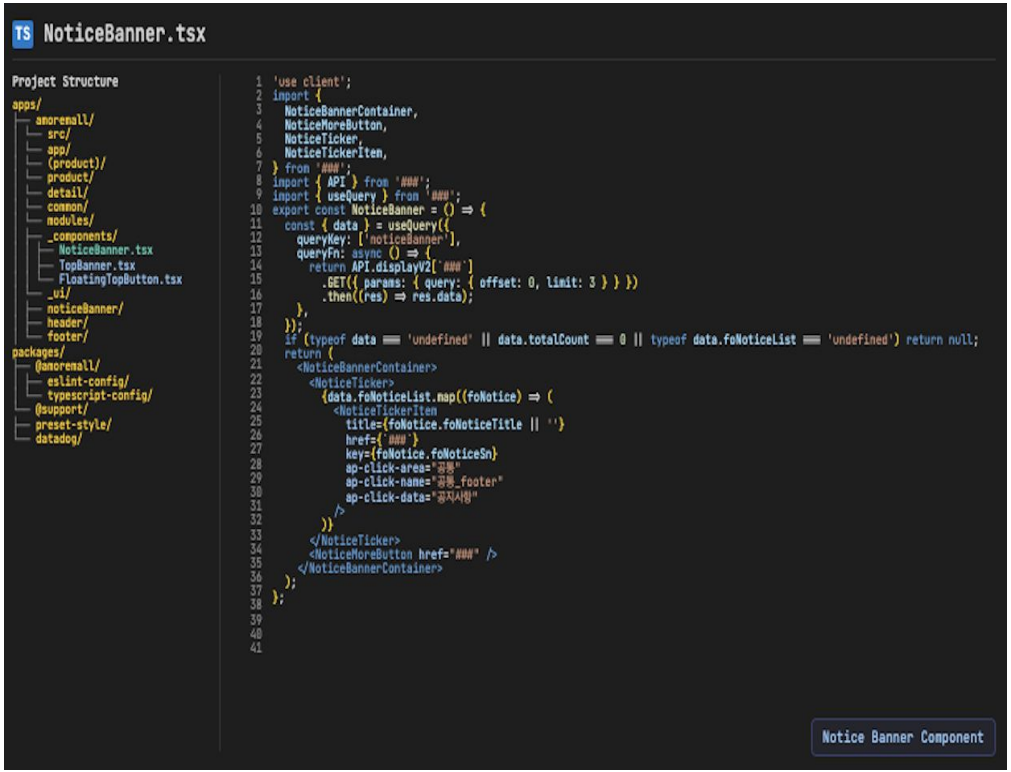
모든 공통 컴포넌트에 Storybook 기반 문서화와 JSDoc 주석을 적용하여 사용법과 옵션을 명확히 제시했습니다. 개발자가 컴포넌트의 기능과 사용방법을 쉽게 파악할 수 있도록 하여 개발 생산성과 코드 품질을 동시에 향상시켰습니다.

8. 확장 가능한 컨트롤 시스템

캐러셀과 같은 복합한 컴포넌트의 경우 템플릿 기반의 확장 가능한 컨트롤 시스템을 구축했습니다. 기본 제공 컨트롤 외에도 커스텀 컨트롤을 쉽게 추가할 수 있는 구조로 설계하여 다양한 디자인 요구사항에 유연하게 대응할 수 있습니다.

TECH UI/UX 컴포넌트

FE 개발자



_components/{component}

1. UI와 비즈니스 로직 통합 계층

순수 UI 컴포넌트와 실제 비즈니스 요구사항을 연결하는 중간계층입니다. API 호출, 상태관리, 조건부 렌더링 등의 비즈니스 로직을 포함하면서 UI 컴포넌트들을 조합하여 실제 사용가능한 완성된 컴포넌트를 제공합니다.

2. 디바이스 적응형 로직 구현

사용자 에이전트 정보를 기반으로 디바이스별 최적화된 경험을 제공합니다. 데스크톱과 모바일, iOS와 Android 환경에서 각각 다른 동작을 수행하며, 앱 설치 유도나 딥링크 처리 등 플랫폼별 특화 기능을 구현했습니다.

3. 사용자 인증 상태 연동

로그인 상태에 따라 다른 콘텐츠와 동작을 제공하는 개인화 시스템을 구축했습니다. 인증된 사용자와 비인증 사용자에게 서로 다른 배너를 표시하고, 사용자 행동 분석을 위한 클릭 데이터도 인증 상태에 따라 구분하여 수집합니다.

4. Client Component 기반 상호작용

사용자와의 실시간 상호작용이 필요한 기능들을 Client Component로 구현했습니다. 브라우저 API 접근, 이벤트 처리, 동적 상태 변경 등이 필요한 경우에만 클라이언트 사이드 렌더링을 활용하여 성능과 사용자 경험의 균형을 맞췄습니다.

5. 환경별 설정 및 구성 관리

개발, 스테이징, 프로덕션 환경에 다른 설정 값들을 체계적으로 관리합니다. API 엔드포인트, 앱 스킴, 원링크 설정 등 환경별로 달라져야 하는 값들을 중앙화된 설정을 통해 관리하여 배포 안정성을 화보했습니다.

6. 에러 처리 및 폴백 시스템

API 호출 실패나 데이터 부재 상황에 대한 견고한 에러 처리 시스템을 구축했습니다. 데이터 로드 실패 시, 컴포넌트 자체를 숨기거나 기본값을 표시하는 등 사용자에게 부정적인 경험을 주지 않는 우아한 실패 처리를 구현했습니다.

7. 추적 및 분석 시스템 통합

사용자 행동 분석을 위한 클릭 추적 시스템을 통합했습니다. 각 상호작용 요소에 추적 속성을 추가하여 사용자의 행동 패턴을 분석할 수 있도록 하고, 이를 통해 UI/UX 개선을 위한 데이터 기반 의사결정을 지원합니다.

TECH API 구조

FE 개발자 BE 개발자

lib/api

백엔드 Swagger 문서를 기반으로 자동으로 API를 정형화합니다.
instances/schemas/helpers/middlewares 등으로 구성되어 있으며,
대표적으로 instances 에는 schemas 를 기반으로 한 API 인스턴스가 생성됩니다.
API 의 버전관리가 가능하며, DTO 에 대한 스키마를 통해 애플리케이션 비즈니스 로직 영역인 _components 에서 사용할 때, 타입스크립트를 지원하여 타입을 통해 관리합니다.

구성한 API 구조는 다음과 같습니다.

instances

shell 스크립트와 openapi 를 통해 Swagger 의 json 데이터를 토대로 typescript 코드로 변환토록 하였습니다. 좌측 코드처럼 실제 API 인스턴스를 나타냅니다.

schemas

모든 API 에 대한 DTO 및 관련 코드들이 무거워지는 걸 방지하고자, schemas 를 통해 최소한의 이미지를 구현하였습니다.

middlewares

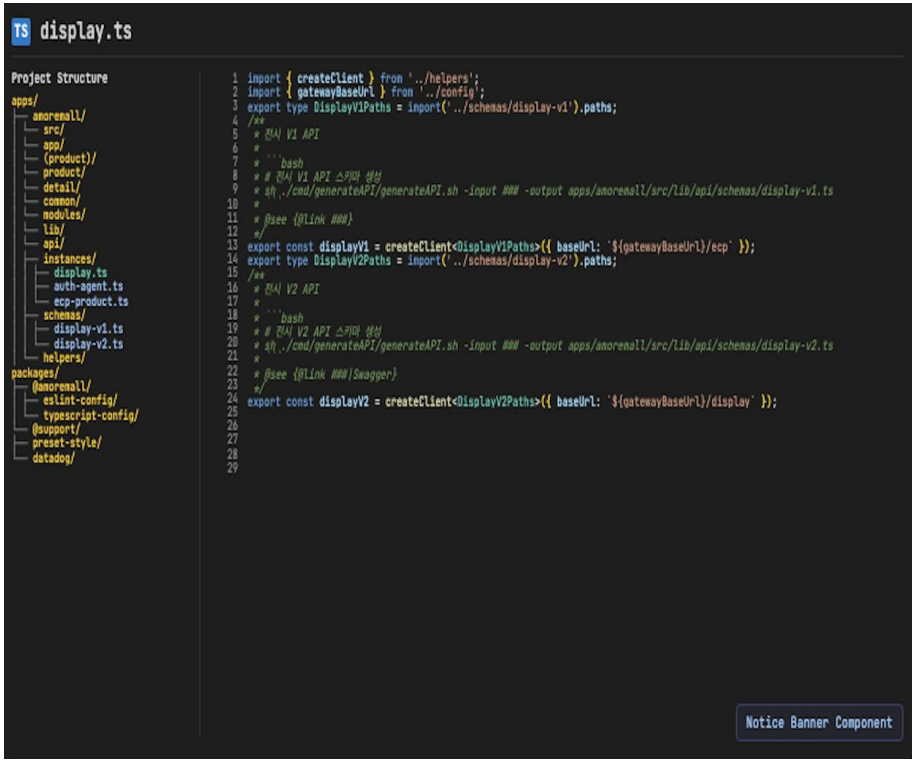
각 API 구성의 미들웨어 설정을 구현하여 wrapping 하였습니다.

helpers

middlewares 에서 공통으로 구성한 공통 헤더/인증 미들웨어/장바구니 미들웨어/타임아웃/로거 등에 대한 공통설정을 구현하여 wrapping 하였습니다.

TECH openapi 활용한 API 생성 자동화

FE 개발자 BE 개발자



lib/api

1. OpenAPI 기반 타입 자동 생성 시스템

Swagger/OpenAPI 명세서를 TypeScript 타입 정의로 자동 변환하는 완전 자동화된 파이프라인을 구축했습니다. 각 마이크로서비스별 API 스키마를 개별 TypeScript 파일로 생성하여 API 통신의 타입 안전성을 보장하고, 백엔드 API 변경사항을 프론트엔드에 즉시 반영할 수 있습니다.

2. 미들웨어 기반 횡단 관심사 처리

공통 헤더 추가, 인증 토큰 관리, 장바구니 정보 포함, 타임아웃 제어, 요청/응답 로깅 등 모든 HTTP 요청에 공통으로 적용되어야 하는 기능들을 미들웨어 패턴으로 구현했습니다. 각 미들웨어는 독립적으로 동작하며 필요에 따라 조합하여 사용할 수 있습니다.

3. 도메인별 API 인스턴스 격리

전시, 상품, 리뷰, 인증 등 각 비즈니스 도메인 별로 독립된 API 인스턴스를 생성하여 관리합니다. 각 인스턴스는 해당 도메인에 특화된 설정과 베이스 URL 을 가지며, 도메인 간 의존성을 최소화하고 서비스별 독립적인 확장이 가능합니다.

4. 환경별 설정 분리 및 게이트웨이 통합

서버 환경에서는 내부 통신용 게이트웨이를 사용하고, 클라이언트 환경에서는 외부 통신용 게이트웨이를 사용하는 이중화된 설정 구조입니다. 이를 통해 보안성을 높이고 네트워크 효율성을 최적화하며, 환경별로 다른 엔드포인트를 자동으로 선택합니다.

5. 지능형 타임아웃 및 리소스 관리

서버 환경과 클라이언트 환경에 따라 다른 타임아웃 전략을 적용합니다. AbortController 를 활용한 요청 취소 시스템과 메모리 누수 방지를 위한 자동 정리 메커니즘을 구현하여 안정적인 네트워크 통신을 보장합니다.

6. Path-Based 클라이언트 인터페이스

OpenAPI 경로를 그대로 활용하는 직관적인 API 호출 인터페이스를 제공합니다. API 경로와 메서드를 TypeScript 타입으로 정확히 매핑하여 자동완성과 타입 검사를 통한 개발 생산성 향상과 런타임 오류 방지를 동시에 달성했습니다.

7. 자동화된 스키마 관리 워크플로우

각 API 인스턴스마다 스키마 생성 명령어와 Swagger 문서 링크를 포함한 완전한 문서화 시스템을 구축했습니다. 개발자는 단일 스크립트 명령으로 최신 API 스키마를 업데이트할 수 있으며 모든 변경사항은 자동으로 TypeScript 타입에 반영됩니다.

TECH Packages 구조

packages

회사의 기술자산이 되는 코어 라이브러리입니다.

내부 개발직원만 관리하며, 외부 개발자는 이를 활용하여 애플리케이션 개발에 활용합니다.

@amoremall

amoremall 애플리케이션에서 사용하는 라이브러리를 구성합니다.

amoremall 기준의 lint, typescript, webview, hooks 등이 추가되어 있습니다.

@support

특정 애플리케이션과 무관하게 어느 애플리케이션에서도 사용할 수 있는 라이브러리를 구성합니다. 대표적인 예로, datadog 라이브러리를 회사에 맞게 wrapping하여 라이브러리화 합니다.

@...

특정 애플리케이션이 추가되면 이에 맞는 라이브러리를 구성합니다.

1단계: 페이지별 기능 선언
application 페이지별 기능을 먼저 선언하고 각 페이지마다 중복을 허용합니다.

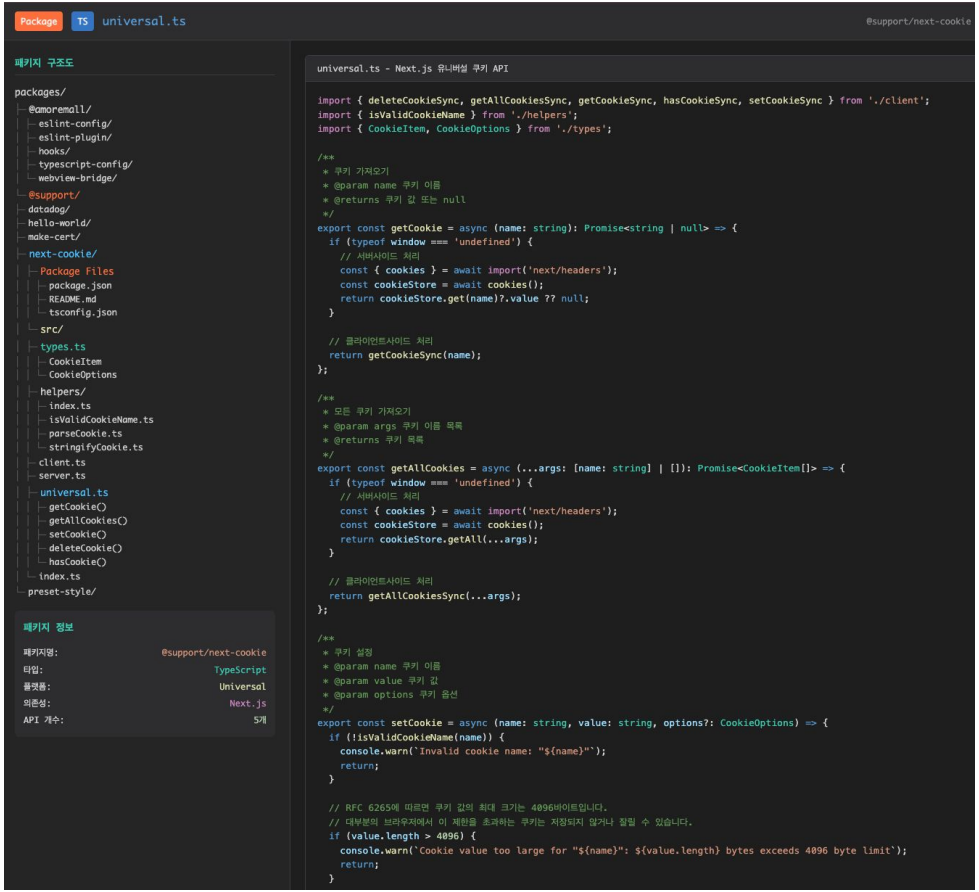
2단계: 애플리케이션 공통화
페이지 레벨에서 중복될 시, application의 common 레벨에 기능 배치합니다.

3단계: 애플리케이션 패키지화
common 레벨의 내용이 라이브러리로 관리가 필요할 시, "@application" 내 라이브러리로 구현합니다.

4단계: 범용 패키지화
"@application"에 국한되지 않는 성격인 경우, "@support"에 라이브러리로 구현합니다.

TECH Packages 구조

FE 개발자



packages

1. 도메인별 패키지 격리 설계

@amoremall 과 @support 등 네임스페이스로 패키지를 명확히 분리하여 역할과 책임을 구분합니다. 비즈니스에 특화된 패키지와 범용적으로 사용할 수 있는 지원 패키지로 분류하여 의존성 관리와 재사용성을 동시에 확보했습니다.

2. 개발 표준화 도구 패키지

ESLint 설정과 커스텀 플러그인, TypeScript 설정을 패키지화하여 프로젝트 전반의 코딩 표준을 일관성 있게 관리합니다. 특히 Next.js Image 컴포넌트 사용 금지 등 프로젝트별 규칙을 커스텀 ESLint 플러그인으로 구현하여 개발자 실수를 사전에 방지합니다.

3. Vite 기반 번들링 최적화

모든 패키지는 Vite를 활용한 ESM 모듈 빌드 시스템을 구축했습니다. TypeScript 선언 파일 자동 생성과 최적화된 번들링을 통해 타입 안전성과 성능을 동시에 확보하며, 모던 자바스크립트 생태계에 적합한 패키지 배포 환경을 구성했습니다.

4. 플랫폼 특화 유틸리티 구현

Next.js 서버/클라이언트 환경 모두에서 작동하는 쿠키 관리, SSL 인증서 자동 생성, Datadog 모니터링 설정 등 개발 생산성을 높이는 실용적 도구들을 패키지화했습니다. 각 도구는 독립적으로 사용 가능하면서도 전체 프로젝트 워크플로우와 자연스럽게 통합됩니다.

5. 프라이빗 레지스트리 연동 CI/CD

모든 패키지는 사내 Nexus 저장소와 연동된 자동 빌드 및 배포 파이프라인을 구축했습니다. 워크스페이스 프로토콜을 활용한 모노레포 내부 패키지 참조와 외부 배포를 동시에 지원하여 개발 환경과 운영 환경 간의 일관성을 보장합니다.

TECH 외부 스크립트 관리

FE 개발자

NethruInitializer.tsx

Project Structure

```
apps/
  amoremall/
    src/
      app/
        (product)/
        product/
        detail/
      common/
      modules/
    common/
    nethru/
      NethruInitializer.tsx
      index.tsx
      nethru.d.ts
      googleAnalytics/
      datadog/
      provider/
    app/
    lib/
packages/
  @amoremall/
    eslint-config/
    typescript-config/
  @support/
  preset-style/
  datadog/
```

```tsx
'use client';
import React, { useCallback, useEffect } from 'react';
import Script from 'next/script';
import { useUserAgent } from '@/common/utils/userAgent';
import { getPublicConfig } from '@/common/config';
const publicConfig = getPublicConfig();
/**
* Nethru 추천 관련 스크립트를 초기화하고, Nethru 추천 스크립트를 삽입합니다.
* - amoremall WEB 대장가이드: ###
* - amoremall APP 대장가이드: ###
*/
export function NethruInitializer() {
  const userAgent = useUserAgent();
  const checkWebview = useCallback(
    () => userAgent.deviceType === 'mobileApp' && userAgent.osType !== 'other',
    [userAgent.deviceType, userAgent.osType],
  );
  useEffect(() => {
    if (!window.AP) window.AP = {};
    window.AP = {
      ...window.AP,
      webview: {
        is: checkWebview,
      },
    };
    window.nethruLoginId = undefined;
    window.i_sDeviceNum = undefined;
  }, [checkWebview]);
  return (
    <>
      <Script
        id="amoremall-nethru-install"
        strategy="afterInteractive"
        src={`${publicConfig.basePath}/static/script/install.min.js`}
        data-s3-url={publicConfig.basePath}
      />
      <Script
        id="amoremall-nethru-solive"
        strategy="afterInteractive"
        src={`${publicConfig.basePath}/static/script/solive.min.js`}
      />
      <Script
        id="amoremall-nethru-seetoc"
        strategy="afterInteractive"
        src={`${publicConfig.basePath}/static/script/seetoc.min.js`}
      />
      <Script
        id="amoremall-nethru-event"
        strategy="afterInteractive"
        src={`${publicConfig.basePath}/static/script/AutoEvent.min.js`}
      />
    </>
  );
}
```

Nethru Recommendation Initializer

스크립트 전략

기존에는 외부 스크립트를 설치 시, 화면 렌더링에 영향을 받아왔습니다.
다음과 같은 스크립트 전략을 통해 화면 렌더링과 스크립트를 구성합니다.

1. Next.js Third-parties 패키지 기반 최적화

공식 Next.js third-parties 패키지를 활용하여 Google Analytics 와 Google Tag Manager를 최적화된 방식으로 로드합니다. 자동 스크립트 최적화, 성능 향상, XSS 보안 기능이 내장되어 있어 안전하고 효율적인 분석 도구 통합을 구현했습니다.

2. 3 계층 전역변수 아키텍처

페이지 정보, 사용자 정보, 공통 객체로 분리된 전역변수 체계를 구축했습니다. 각 계층은 독립적으로 관리되며 TypeScript를 통해 완전한 타입 안전성을 보장합니다. 이를 통해 복잡한 이커머스 추적 요구사항을 체계적으로 관리합니다.

3. 동적 사용자 컨텍스트 추적

로그인 상태, 회원등급, 성별, 임직원 여부 등 다양한 사용자 속성을 실시간으로 추적하고 분석 도구에 전달합니다.
개인정보는 SHA512 해싱을 통해 암호화하여 개인정보보호 규정을 준수하면서도 정확한 사용자 행동 분석이 가능합니다.

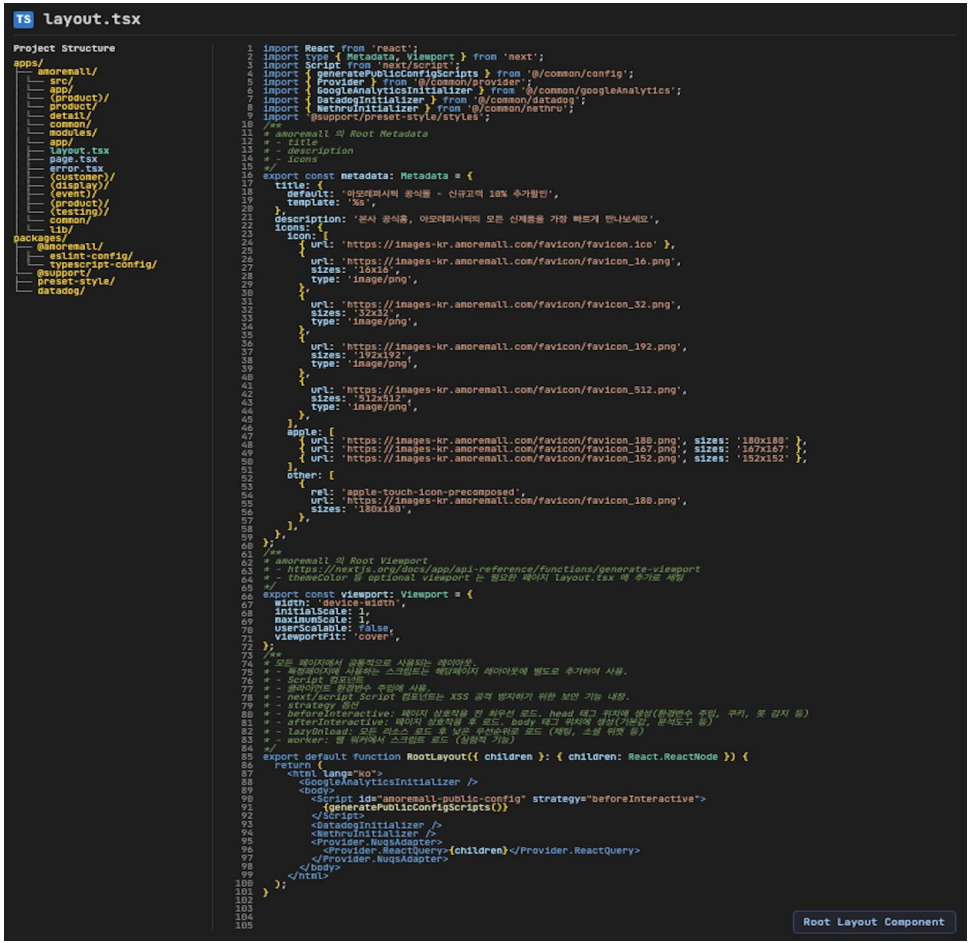
4. 페이지별 맞춤형 추적 설정

메인, 상품상세, 검색결과, 장바구니 등 각 페이지 타입별로 특화된 추적 변수를 자동설정합니다. 디바이스 타입 자동 감지와 함께 PC, 모바일, 웹 환경을 구분하여 채널별 사용자 행동 패턴을 정확하게 분석할 수 있습니다.

5. Script 로딩 전략 최적화

beforeInteractive, afterInteractive, lazyOnload 전략을 활용하여 분석 스크립트의 로딩 우선순위를 최적화했습니다. 필수 추적 변수는 페이지 상호작용 전에 로드하고 분석도구는 상호작용 후 로드하여 사용자 경험에 미치는 영향을 최소화합니다.

TECH SEO 스크립트 관리



Metadata 관리

RootLayout 에서 공통이 되는 Metadata 를 구성합니다.
Next.js 에서 제공하는 Metadata, viewport 타입을 통해 타입 구성을 합니다.

1. Next.js App Router 기반 계층형 메타데이터

페이지별 독립적인 메타데이터 관리 시스템을 구축했습니다. 루트 레이아웃에서 기본 메타데이터를 정의하고, 각 페이지 레이아웃에서 페이지별 특화 메타데이터를 설정합니다. 상품상세 페이지에서는 API 데이터를 활용한 동적 메타데이터 생성을 통해 실시간으로 최적화된 SEO 정보를 제공합니다.

2. 자동화된 검색엔진 최적화 파일 생성

sitemap.xml, robots.txt, manifest.json 파일을 Next.js 함수를 통해 자동 생성하는 시스템을 구현했습니다. MetadataRoute 타입을 활용하여 타입 안전성을 보장하면서도 검색엔진이 요구하는 표준형식을 준수합니다. 이를 통해 수동관리 없이도 검색엔진 크롤링과 색인 최적화를 달성합니다.

3. 구조화된 데이터 및 Open Graph 통합

상품 API에서 제공하는 SEO 정보와 메타 정보를 활용하여 Rich Snippet과 소셜 미디어 최적화를 동시에 구현했습니다. 상품명, 설명, 이미지, 키워드 등의 구조화된 데이터를 검색엔진이 이해하기 쉬운 형태로 제공하여 검색 결과에서의 노출도와 클릭률을 향상시킵니다.

4. 다중 해상도 파비콘 및 PWA 지원

16px부터 512px까지 다양한 해상도의 파비콘을 제공하고, Apple Touch Icon과 Maskable Icon 을 포함한 PWA 매니페스트를 구성했습니다. 이를 통해 데스크톱, 모바일, 앱 환경에서 일관된 브랜드 경험을 제공하며, 웹액으로서의 설치 가능성을 확보합니다.

5. 성능 중심 스크립트 로딩 및 보안 강화

Script 컴포넌트의 로딩 전략을 활용하여 SEO에 필수적인 스크립트는 beforeInteractive로, 분석 도구는 afterInteractive로 최적화했습니다. Content Security Policy 헤더를 통해 보안을 강화하면서도 검색엔진 크롤러의 접근은 허용하여 SEO 성능과 보안성을 동시에 확보했습니다.

FE 개발자 데이터 분석

TECH SEO 스크립트 관리

page.tsx

Project Structure

```
apps/
  amoremall/
    src/
      app/
        (product)/
          product/
            detail/
              common/
              modules/
                app/
                  (product)/
                    product/
                      detail/
                        page.tsx
                        layout.tsx
                        _ui/
                          (customer)/
                          (display)/
                          (event)/
                          common/
                          lib/
packages/
  @amoremall/
    eslint-config/
    typescript-config/
  @support/
  preset-style/
  datadog/
```

```tsx
import type { Metadata, ResolvingMetadata } from 'next';
import { notFound } from 'next/navigation';
import { createLoader, parseAsInteger, parseAsString } from 'nuqs/server';
import { API } from '@/lib/api';
const loadSearchParams = createLoader({
  /** 온라인상품번호 */
  ###: parseAsInteger,
  /** 온라인상품코드 */
  onlineProdCode: parseAsString,
});
/**
* 상품상세 동적 메타데이터
* - 상품상세 API 정보를 조회하여, 메타데이터로 사용한다.
* - 상품상세 API 정보가 없으면, RootLayout 메타데이터를 기본값으로 사용한다.
* - generateMetadata
* - 예약 함수명은 Next.js 예약어
* - 동적 메타데이터는 page 에서 정의해서 사용해야 한다.(layout 에서는 정의불가)
* - 우선순위: Page > PageLayout > RootLayout
*/
export async function generateMetadata(
  { searchParams }: AppPageProps,
  parent: ResolvingMetadata,
): Promise<Metadata | null> {
  const { ### } = await loadSearchParams(searchParams);
  if (!###) return null;
  const { data } = await API.productV1['###'].GET({ params: { path: { ### } } });
  if (!data?.metaInfo) return null;
  const previousImages = (await parent).openGraph?.images || [];
  return {
    title: data.metaInfo.title,
    description: data.metaInfo.desc,
    keywords: data.metaInfo.keyword,
    openGraph: {
      images: data.metaInfo.image ? [data.metaInfo.image, ...previousImages] : previousImages,
    },
  };
}
/** 상품상세 페이지 */
export default async function ProductDetailPage({ searchParams }: AppPageProps) {
  const { ### } = await loadSearchParams(searchParams);
  // 온라인상품번호 URL 파라미터 진입 오류 validation 처리
  if (!###) {
    console.error('상품상세 URL 파라미터 진입 오류', { ### });
    return notFound();
  }
  const { data, error } = await API.productV1['###'].GET({ params: { path: { ### } } });
  // 상품상세 API 조회 오류 validation 처리
  if (!data && error) {
    console.error('상품상세 API 조회 오류', { ###, error });
    return notFound();
  }
  return (
    <div>
```

Product Detail Page

동적 Metadata 관리

Metadata 중에 API 등을 통해 동적으로 생성되는 동적 Metadata 를
다음과 같이 구현하고, 관리합니다.

1. generateMetadata 함수 기반 동적 생성

Next.js App Router의 예약 함수를 활용하여 페이지별로 API 데이터를 기반으로 한 동적 메타데이터를 생성합니다.
상품상세 페이지에서는 온라인상품번호를 파라미터로 받아 실시간으로 상품 정보를 조회하고, 해당 데이터를
메타데이터로 변환하여 검색엔진 최적화를 수행합니다.

2. 계층형 메타데이터 상속 구조

ResolvingMetadata 매개변수를 통해 부모 컴포넌트의 메타데이터를 상속받아 확장하는 방식으로 구현했습니다. 페이지
레벨에서 정의된 동적 메타데이터가 페이지 레이아웃과 루트 레이아웃의 메타데이터보다 우선순위를 가지며, 기존 정보를
유지하면서 새로운 정보를 추가합니다.

3. 백엔드 API 연동 메타데이터 자동화

상품 API의 metaInfo 객체에서 제공하는 title, description, image, keyword 정보를 추출하여 메타데이터로
변환합니다. 이를 통해 수동으로 메타데이터를 관리할 필요 없이 백엔드에서 관리하는 상품 정보가 자동으로 SEO 최적화에
반영됩니다.

4. Open Graph 이미지 병합 처리

기존 상위 메타데이터의 Open Graph 이미지 목록을 유지하면서 새로운 상품 이미지를 앞쪽에 추가하는 방식으로
구현했습니다. 이를 통해 소셜 미디어 공유 시 상품 이미지가 우선적으로 표시되면서도 사이트 기본 이미지 정보를
백업으로 유지할 수 있습니다.

5. 안전한 Fallback 매커니즘

API 호출 실패나 데이터 부재 시 null 을 반환하여 상위 레벨의 메타데이터가 적용되도록 설계했습니다. 이를 통해 동적
메타데이터 생성 과정에서 오류가 발생하더라도 사용자에게는 항상 적절한 메타데이터가 제공되어 검색엔진 크롤링에
문제가 발생하지 않습니다.

TECH GA 스크립트 관리

FE 개발자 데이터 분석

GoogleAnalyticsInitializer.tsx

Project Structure

```
apps/
  amoremall/
    src/
      app/
        (product)/
          product/
            detail/
        common/
        modules/
          common/
            googleAnalytics/
              GoogleAnalyticsInitiali
              index.tsx
              googleAnalytics.d.ts
            datadog/
            modules/
            provider/
          app/
          lib/
packages/
  @amoremall/
    eslint-config/
    typescript-config/
  @support/
  preset-style/
  datadog/
```

```tsx
'use client';
import React, { useEffect } from 'react';
import { GoogleAnalytics, GoogleTagManager } from '@next/third-parties/google';
/**
* GoogleTagManager 관련 변수를 정의하고, 관련 스크립트를 삽입합니다.
* - 헤딩 태그는 body 태그 바깥에 위치해야 합니다.
* - nextjs 개발가이드: https://nextjs.org/docs/app/guides/third-party-libraries
* - amoremall 개발가이드: ###
*/
export function GoogleAnalyticsInitializer() {
  useEffect(() => {
    if (!window.AP) window.AP = {};
    window.AP = {
      ...window.AP,
    };
  }, []);
  return (
    <>
      <GoogleTagManager gtmId="###" />
      <GoogleAnalytics gaId="###" />
    </>
  );
}
```

Google Analytics Initializer

GA 스크립트

common 하위에 GoogleAnalytics 로 모듈화를 하였습니다.

1. 컴포넌트 기반 분리 초기화 설계

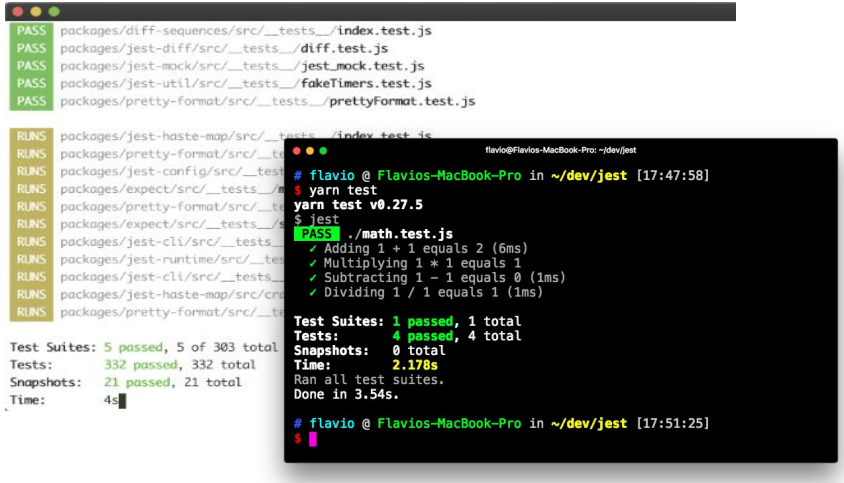
페이지 정보와 사용자 정보를 각각 독립된 초기화 컴포넌트로 분리하여 관리합니다. PageGlobalVariableInitializer와 UserGlobalVariableInitializer가 각각의 책임을 가지며, 필요에 따라 선택적으로 사용할 수 있어 코드 재사용성과 유지보수성을 크게 향상시킵니다.

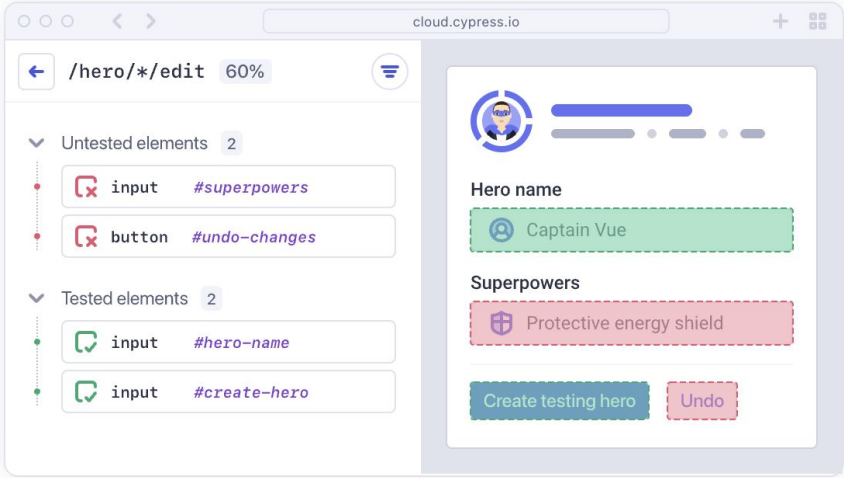
2. 비동기 초기화 및 타입 안전성

useEffect 훅을 통한 비동기 초기화로 서버사이드 렌더링과 클라이언트 하이드레이션 간의 충돌을 방지합니다. TypeScript 전역 타입 선언을 통해 Window 객체 확장 시에도 완전한 타입 안전성을 보장하여 런타임 오류를 사전에 방지합니다.

FE 개발자 QA

TECH QA 단위/통합테스트





유틸리티 단위 테스트

utils, hooks 등의 주요한 기능들을 단위 테스트하여, 방어로직 및 validation 체크를 합니다.

React Testing Library 통합

React 컴포넌트 렌더링 테스트를 위한 React Testing Library 통합으로 UI 검증 가능

사용자 통합 테스트

페이지별 UI/UX 종합 테스트를 합니다. 기획서와 테스트 시나리오를 기반으로 구성하며, 개발자 레벨에서 QA를 보완합니다.

반응형 디자인 검증

Cypress의 viewport 조정 기능을 통해 다양한 화면 크기에서의 UI 동작 확인 및 검증

API 인터셉트 & 모킹

Cypress의 네트워크 요청 인터셉트 기능으로 백엔드 의존성 없는 안정적인 E2E 테스트 구현

TECH QA 단위/통합테스트

HtmlParser.test.ts

```
Project Structure
apps/
  amoremall/
    src/
    common/
    utils/
    htmlParser/
    HtmlParser.test.ts
    HtmlParser.tsx
    index.ts
    googleAnalytics/
    datadog/
    provider/
    app/
    lib/
packages/
  @amoremall/
    eslint-config/
    typescript-config/
  @support/
  greset-style/
  datadog/
```

```ts
import { htmlParser } from './HtmlParser';
// eslint-disable-next-line @amoremall/no-xss-unsafety-libraries -- 테스트를 위한 허용
import parse from 'html-react-parser';
// Mock the dependencies
jest.mock('html-react-parser', () => ({
  __esModule: true,
  default: jest.fn((html) => html),
}));
describe('htmlParser', () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });
  it('빈 문자열이나 undefined 가 입력되면 null 을 반환해야 합니다', () => {
    expect(htmlParser('')).toBeNull();
    expect(htmlParser(undefined as unknown as string)).toBeNull();
  });
  it('HTML 문자열을 파싱하여 JSX 엘리먼트를 반환해야 합니다', () => {
    const testHtml = '<div>테스트 컨텐츠</div>';
    const mockParsedElement = { type: 'div', props: { children: '테스트 컨텐츠' } };
    (parse as jest.Mock).mockReturnValueOnce(mockParsedElement);
    const result = htmlParser(testHtml);
    expect(parse).toHaveBeenCalledWith(testHtml);
    expect(result).toBe(mockParsedElement);
    expect(result).toEqual(expect.any(Object));
  });
  it('복잡한 HTML 구조도 파싱할 수 있어야 합니다', () => {
    const complexHtml = `
      <div class="container">
        <h1>제목</h1>
        <p>문단 내용</p>
        <ul>
          <li>항목 1</li>
          <li>항목 2</li>
        </ul>
      </div>
    `;
    const mockParsedComplexElement = {
      type: 'div',
      props: {
        className: 'container',
        children: [
          { type: 'h1', props: { children: '제목' } },
          { type: 'p', props: { children: '문단 내용' } },
          {
            type: 'ul',
            props: {
              children: [
                { type: 'li', props: { children: '항목 1' } },
                { type: 'li', props: { children: '항목 2' } },
              ],
            },
          },
        ],
      },
    };
    (parse as jest.Mock).mockReturnValueOnce(mockParsedComplexElement);
    const result = htmlParser(complexHtml);
    expect(parse).toHaveBeenCalledWith(complexHtml);
    expect(result).toBe(mockParsedComplexElement);
    expect(result).toEqual(expect.any(Object));
  });
});
```

HTML Parser Test

```
PASS src/common/utils/htmlSanitizer/HtmlSanitizer.test.ts
  HtmlSanitizer
    sanitizeHtml
      ✓ 안전한 HTML 은 그대로 유지되어야 함 (16 ms)
      ✓ 허용되지 않은 태그는 제거되어야 함 (3 ms)
      ✓ 허용되지 않은 속성은 제거되어야 함 (5 ms)
      ✓ target="_blank"가 있는 링크에는 rel="noopener noreferrer"가 추가되어야 함 (3 ms)
      ✓ target="_blank"가 있는 링크에 이미 rel 값이 있다면 rel="noopener noreferrer"로 변경해야 함 (3 ms)
      ✓ XSS 공격 벡터가 제거되어야 함 (2 ms)
      ✓ 복잡한 HTML 구조에서도 안전한 콘텐츠만 유지되어야 함 (9 ms)

Test Suites: 1 passed, 1 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        1.284 s
Ran all test suites matching /src\/common\/utils\/htmlSanitizer\/HtmlSanitizer.test.ts/i.
```

단위/통합 테스트

1. Jest 기반 단위 테스트 환경구축

createJestConfig 를 활용하여 프레임워크와 완전 통합된 테스트 환경을 구성했습니다. V8 커버리지 제공자와 jsdom 테스트 환경을 통해 실제 브라우저 환경과 유사한 조건에서 컴포넌트와 유틸리티 함수를 테스트할 수 있으며, 빌드 시점의 설정과 일관성을 보장합니다.

2. 커스텀 HTML 비교 매처 시스템

toEqualHtml 커스텀 매처를 구현하여 HTML 문자열의 정규화 비교가 가능합니다. DOM 파싱, 속성 정렬, 공백 정규화를 통해 HTML 구조적 동일성을 검증하며, XSS 방지 및 HTML 살균 기능 테스트에서 핵심적인 역할을 수행합니다.

3. Given-When-Then 패턴 표준화

모든 테스트 케이스를 Given-When-Then 패턴으로 표준화하여 테스트 의도를 명확하게 표현합니다. 테스트 설명은 한국어로 작성하고, beforeEach 에서 Mock 초기화를 수행하여 테스트 간 격리를 보장하며, 에지 케이스와 보안 관련 테스트를 우선적으로 구현합니다.

4. Cypress E2E 테스트 인프라 준비

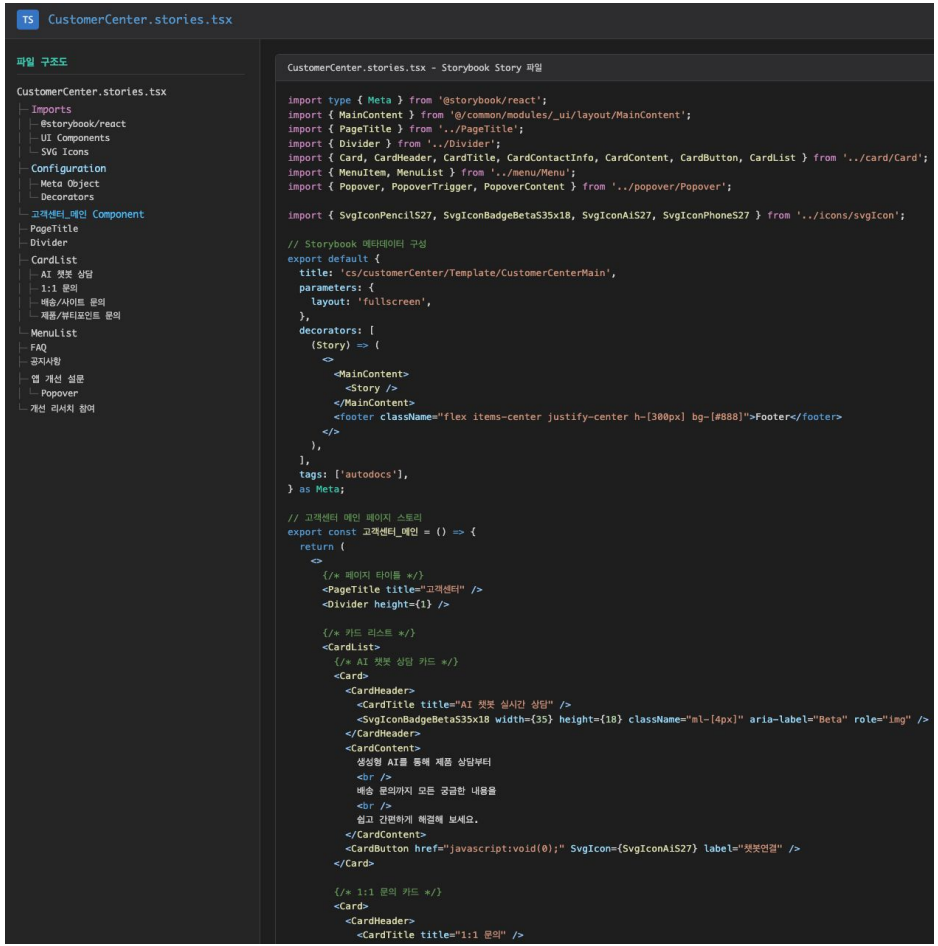
data-cy 속성을 활용한 요소 선별 시스템을 구축하여 안정적인 E2E 테스트 환경을 준비했습니다. 사용자 여정 기반 테스트 시나리오 작성이 가능하며, Page Object Model 패턴과 커스텀 명령어를 통해 유지보수성이 높은 통합 테스트 구조를 설계했습니다.

5. 보안 중심 테스트 전략

XSS 공격 벡터 차단, HTML 살균 처리, 안전하지 않은 라이브러리 사용 금지 등 보안에 특화된 테스트 케이스를 우선 구현했습니다. ESLint 플러그인과 연동된 테스트를 통해 개발단계에서부터 보안 취약점을 사전 차단하는 다층 방어 체계를 구축했습니다.

FE 개발자 디자이너

TECH Storybook 활용 지식자산화



Storybook 구성

Storybook은 UI 컴포넌트를 독립적으로 개발하고 시각적으로 테스트할 수 있는 도구로, 개발자와 디자이너 간의 협업을 크게 개선합니다.
각 컴포넌트를 격리된 환경에서 개발하고 다양한 상태와 상호작용을 테스트할 수 있습니다.

1. Next.js 통합 설정 아키텍처

Next.js 프레임워크와 완전 통합된 Storybook 환경을 구축하여 실제 운영환경과 동일한 조건에서 컴포넌트를 개발하고 테스트할 수 있습니다. 웹팩 설정 커스터마이징을 통해 SVG 아이콘 처리와 환경변수 주입을 자동화하여 개발 환경과 프로덕션 환경 간의 일관성을 보장합니다. .

2. 자동 문서화 및 타입 추론 시스템

TypeScript 와 JSDoc을 활용한 완전 자동화된 문서 생성 시스템을 구현했습니다. 컴포넌트의 Props, 이벤트 핸들러, 사용 예시가 코드에서 자동으로 추출되어 개발자 문서로 변환되며, 실시간으로 업데이트되어 항상 최신 상태의 API 문서를 제공합니다.

3. 멀티 디바이스 반응형 테스트 환경

기본 뷰포트뿐만 아니라 Galaxy Z Fold 와 같은 폴더블 디바이스까지 지원하는 포괄적인 디바이스 테스트 환경을 구축했습니다. 다양한 화면 크기와 해상도에서 컴포넌트의 반응형 동작을 실시간으로 검증할 수 있어 디바이스별 최적화 작업을 효율적으로 수행할 수 있습니다.

4. 고급 애드온 생태계 활용

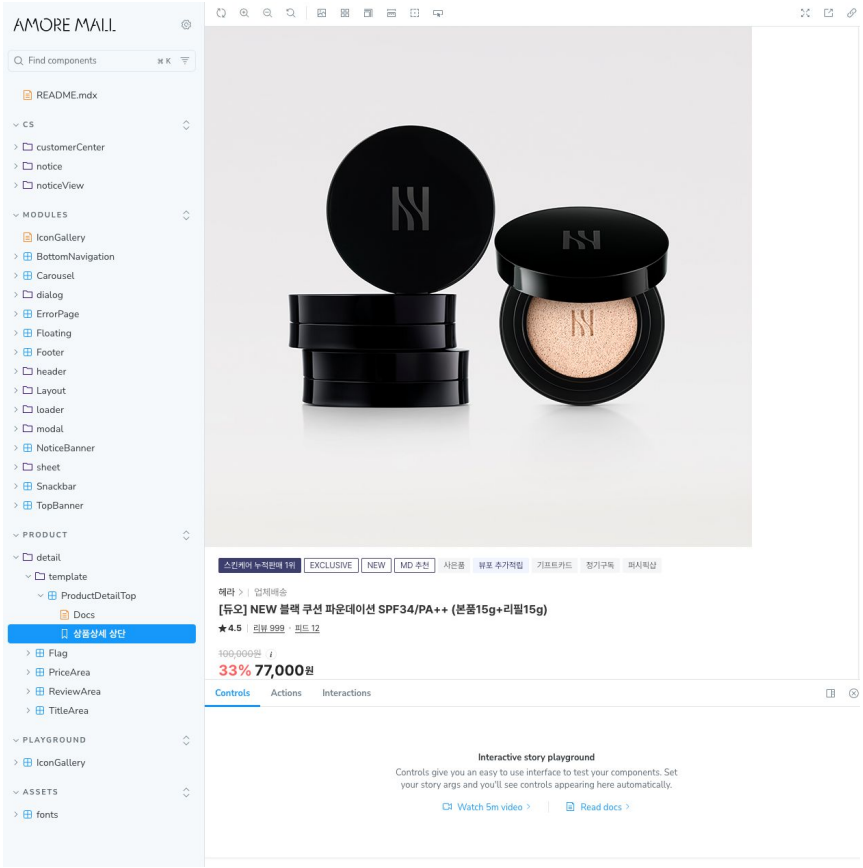
Pseudo-states, Tag-Badges, Interactions 등 전문적인 애드온을 통해 컴포넌트의 다양한 상태와 사용자 상호작용을 시뮬레이션합니다. 중복 컴포넌트 표시, 기능별 태그 분류, 자동 상호작용 테스트 등을 통해 개발 품질과 효율성을 동시에 향상시킵니다.

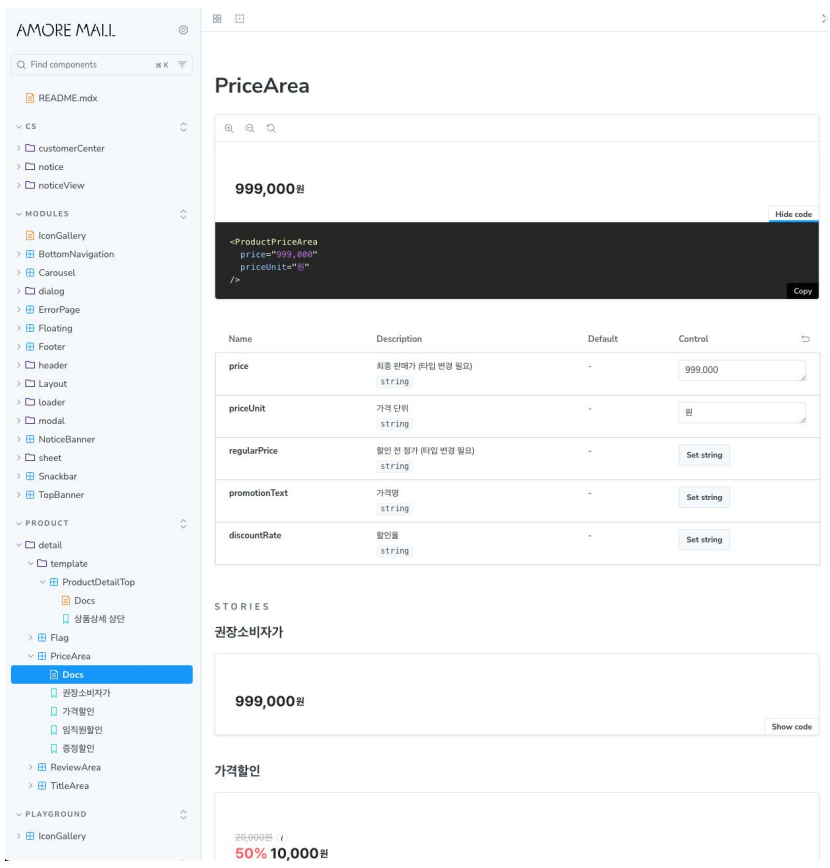
5. Docker 기반 독립 배포 시스템

Nginx 를 기반으로 한 정적 사이트 배포 시스템을 구축하여 Storybook 을 독립적인 문서 사이트로 배포합니다. 이를 통해 디자이너, 기획자, 개발자 간의 협업을 원활하게 하고 컴포넌트 가이드라인을 효과적으로 공유할 수 있는 환경을 제공합니다.

TECH Storybook 활용 산출물

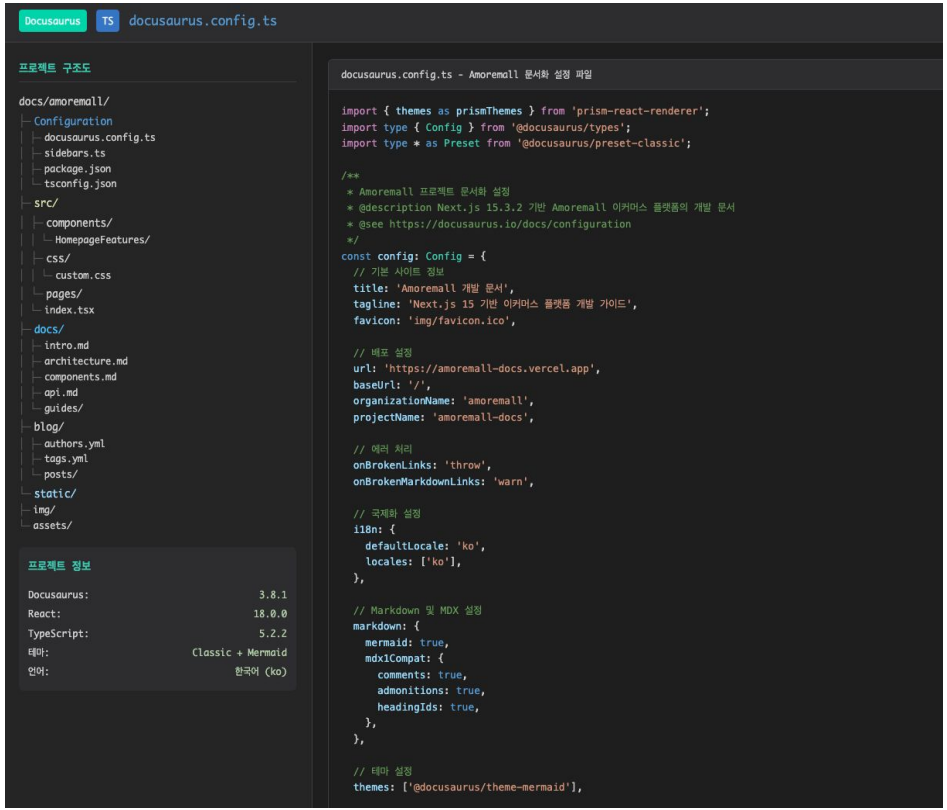
FE 개발자 | 디자이너





TECH Docusaurus 활용 지식자산화

FE 개발자 | BE 개발자 | IT PMO
디자이너 | 기획자 | QA



Docusaurus 구성

Docusaurus는 Facebook에서 개발한 정적 웹사이트 생성기로, 기술 문서화에 최적화되어 있습니다. 마크다운 기반의 문서 작성과 자동 배포를 통해 항상 최신 상태의 개발 문서를 유지할 수 있습니다.

1. JSDoc 기반 자동 문서 생성 파이프라인

TypeScript 코드에서 JSDoc 주석을 추출하여 Docusaurus 문서로 자동 변환하는 완전 자동화 시스템을 구축했습니다. 컴포넌트 Props, API 함수, 타입 정의 등이 코드 변경 시 실시간으로 문서에 반영되어 개발자 문서와 코드 간의 일관성을 보장하며, 수동 문서 관리 없이도 최신 API 레퍼런스를 제공합니다.

2. 다중 언어 기술 문서 통합 지원

TypeScript, JSX, MDX, JSON, Bash 등 다양한 프로그래밍 언어에 대한 문법 강조와 코드 블록 렌더링을 지원합니다. 실제 프로젝트에서 사용하는 모든 기술 스택의 코드 예제를 한 곳에서 통합 관리할 수 있으며, 개발자가 복사하여 바로 실행할 수 있는 검증된 코드 스니펫을 제공합니다.

3. Mermaid 다이어그램 통합 아키텍처 문서화

복잡한 시스템 아키텍처와 API 흐름을 Mermaid 다이어그램으로 시각화하여 개발자 이해도를 높입니다. 라이트/다크 테마 지원을 통해 다양한 환경에서 최적화된 다이어그램을 제공하며, 문서 내에서 인터렉티브한 아키텍처 설명이 가능합니다.

4. 카테고리별 구조화된 문서 네비게이션

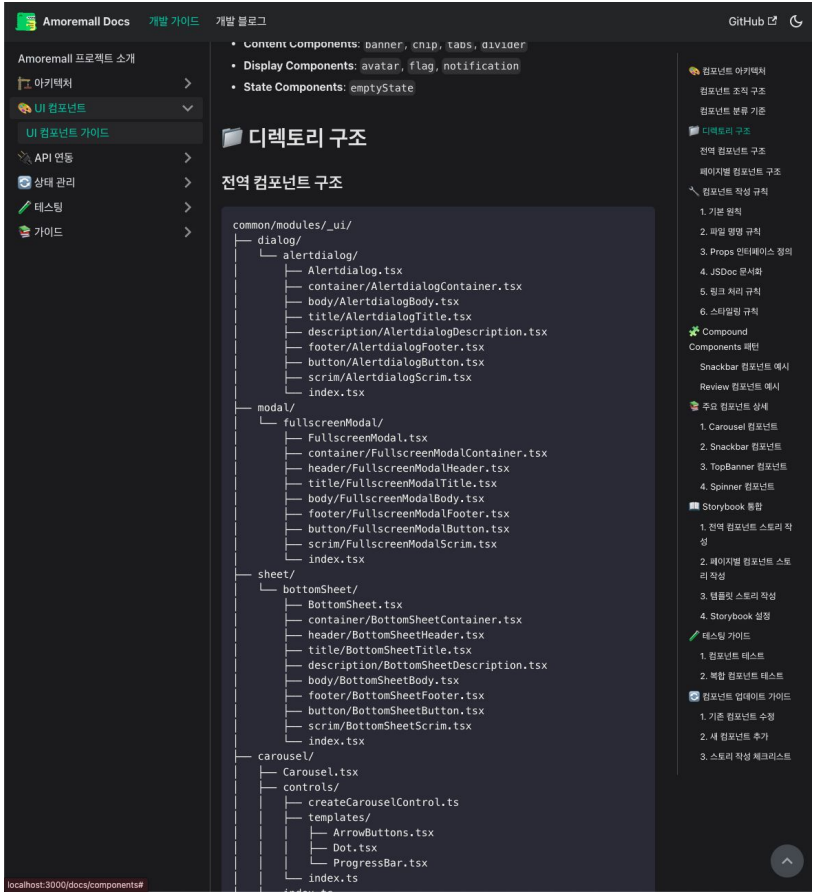
아키텍처, UI 컴포넌트, API 연동, 상태 관리, 테스팅 등 개발 영역별로 체계적으로 분류된 문서구조를 구축했습니다. 사이드바 자동 생성과 검색 기능을 통해 개발자가 필요한 정보를 빠르게 찾을 수 있으며, 문서 간 상호 참조와 링크 관리를 자동화했습니다.

5. 정적 사이트 배포 및 버전 관리

Vercel 을 통한 자동 배포 파이프라인과 Git 기반 버전 관리 시스템을 구축했습니다. 코드 변경 시 문서가 자동으로 업데이트되고 배포되며, 개발 브랜치별로 독립적인 문서 환경을 제공하여 개발 단계별 문서 관리가 가능합니다.

TECH Docusaurus 활용 산출물

FE 개발자 | BE 개발자 | IT PMO
디자이너 | 기획자 | QA

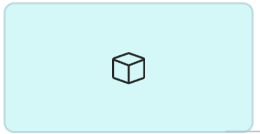


FE 개발자 DevOps

TECH Docker 빌드/배포

Package Image Layer

공통 패키지 라이브러리 이미지



App Image Layer

N개 애플리케이션 중 대상 애플리케이션 이미지

Base Image Layer

환경 구성을 위한 환경설정 이미지

멀티 스테이지 도커 빌드

각 빌드 단계를 분리하여 최종 이미지 크기를 최소화합니다.

Turborepo 캐싱

turbo.json 의 로컬 및 원격 캐시를 활용하여 빌드 시간을 단축합니다.

앱 타입 감지

Next.js 또는 React 를 shell 체크를 통해 자동으로 감지하고, 환경에 맞는 적절한 실행 환경을 구성합니다.

빌드 파일 경량화

Standalone 모드 등을 통해 빌드에 필요한 파일들만 추출하여 애플리케이션과 패키지가 확장되더라도, 경량화는 유지됩니다.

Storybook 별도 빌드

컴포넌트 문서화를 위한 별도의 Storybook 빌드 프로세스를 제공하여, 서비스 빌드의 성능에 영향을 미치지 않고 향상시킵니다.

Turborepo 기반 프로젝트의 효율적인 빌드 및 배포를 위한 Docker 기반 파이프라인을 구축했습니다.

1

다중 레이어 Docker 빌드

빌드 프로세스를 3개의 레이어로 분리하여 효율성을 극대화했습니다:

Turborepo 환경 레이어: 기본 Node.js 환경과 Turborepo 설정을 포함

애플리케이션 레이어: 특정 애플리케이션의 코드와 의존성을 포함

패키지 레이어: 공유 패키지 빌드 결과를 포함

2

캐시 최적화

Docker 빌드 캐시와 Turborepo 캐시를 결합하여 빌드 성능을 최적화했습니다. turbo.json 에 의해 레이어별 캐싱 전략으로 변경된 부분만 재빌드하여 배포 시간을 크게 단축했습니다.

3

환경별 배포 자동화

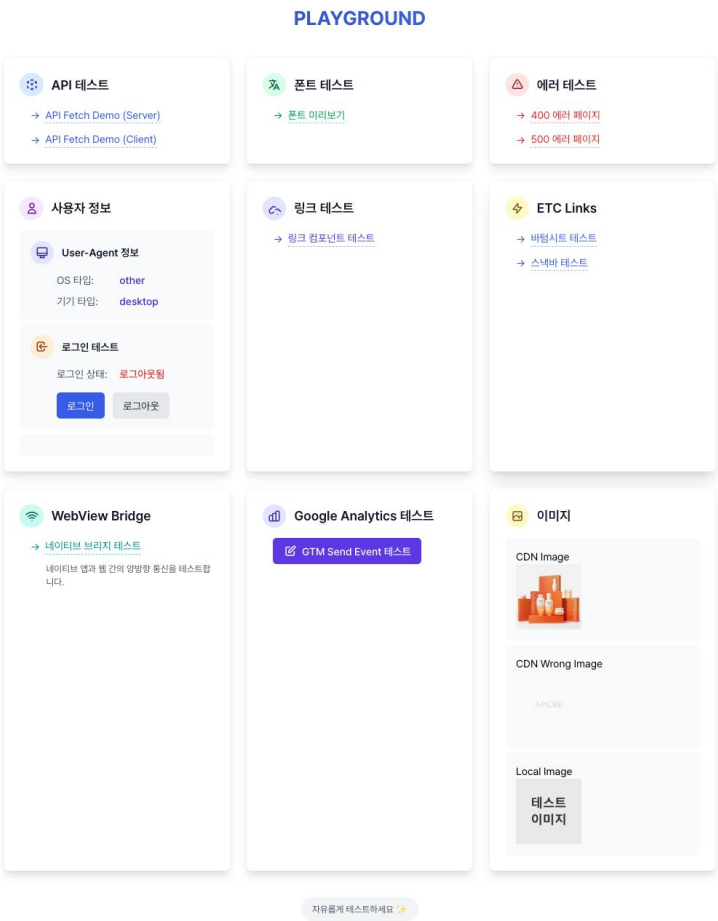
개발, 스테이징, 프로덕션 환경에 맞춘 빌드 설정을 자동화했습니다. 환경변수, 최적화 설정, 로깅 레벨 등이 환경에 따라 자동으로 조정됩니다.

4

확장 가능한 CI/CD 파이프라인

신규 애플리케이션 추가 시 최소한의 설정만으로 기존 파이프라인에 통합될 수 있도록 템플릿화된 CI/CD 구성을 제공합니다.

TECH 기타 서비스 산출물



개발자 Playground 서비스

해당 서비스는 개발자의 편의를 위해 구성한 서비스 페이지입니다. 개발자가 모듈별로 구성한 기능들을 테스트하고 이상 여부 및 정상 동작 여부를 확인할 수 있습니다. 모듈별로 구성이 되어 있다는 점에서 모듈의 기능 테스트와 핵심 기능을 예제처럼 활용이 가능합니다.

1. 개발자의 Playground

제목 그대로 개발자의 Playground 서비스 페이지입니다. 개발자가 유틸리티를 개발하고, 이를 예제로 구성하여 모듈별로 테스트하고 확인할 수 있는 페이지입니다. 링크간 이동, 폰트, API, 에러, 사용자 정보, 웹뷰, GTM 등 다양한 모듈로 구성이 되며, 모듈이 추가될 때마다 계속해서 구성하고 확장이 자유롭습니다.

2. 모듈별로 단위테스트 지원

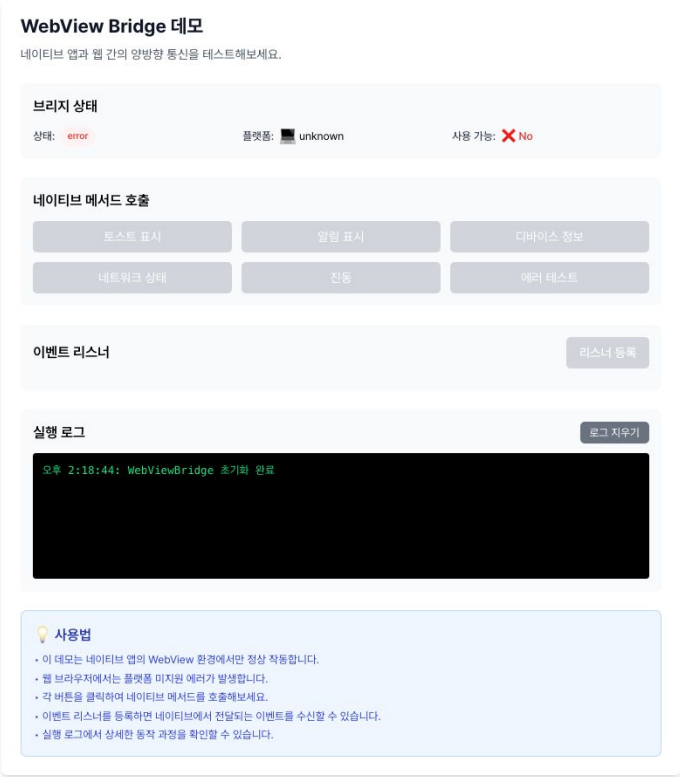
모듈에 대해 단위테스트를 지원하기 때문에 QA 또는 개발자 레벨에서 단위테스트로 로직 체크를 하거나, 기능의 정상작동 여부를 확인하는데 활용합니다.

3. 모듈별로 예제 기능으로 활용

개발자 간 모듈 개발 이후에 이를 사용할 때 예제 코드로 활용이 가능합니다. 모듈 단위의 순수 기능 로직이 들어가 있기에 이를 실 서비스에 활용하는 방식으로 정형화할 수 있습니다.

4. APP/GTM/API 통신 관련 디버깅 활용

해당 기능에는 APP 통신, GTM 통신 등 다양한 통신에 대한 확인이 가능합니다. 특정 APP의 통신이 불가능한 경우, Playground 에서 디버깅을 지원하고, 이를 빠르게 확인할 수 있도록 구성되어 있습니다.



Playground WebView Bridge

Playground 설명 중 4번에 해당하는 디버깅 활용에 가능한 서비스입니다.
가장 확인하기 어려운 디버깅 중 하나인 웹-앱 간 통신에 대해 디버깅을 중점으로 지원하는 서비스 페이지입니다. 실행 로그를 적용하여, 통신에 대한 유효성 여부를 확인할 수 있습니다.

1. 웹-앱 통신상태 디버깅 활용

브리지 상태를 통해 현재 웹-앱 통신상태를 확인할 수 있습니다. 연결 전/연결 중/연결 종료/에러 등의 상태가 존재하며, 연결된 플랫폼의 상태와 통신 가능여부를 확인할 수 있습니다. 이를 통해 본격적인 메서드 디버깅 전 웹-앱의 초기화 상태를 통해 빠른 디버깅이 가능합니다.

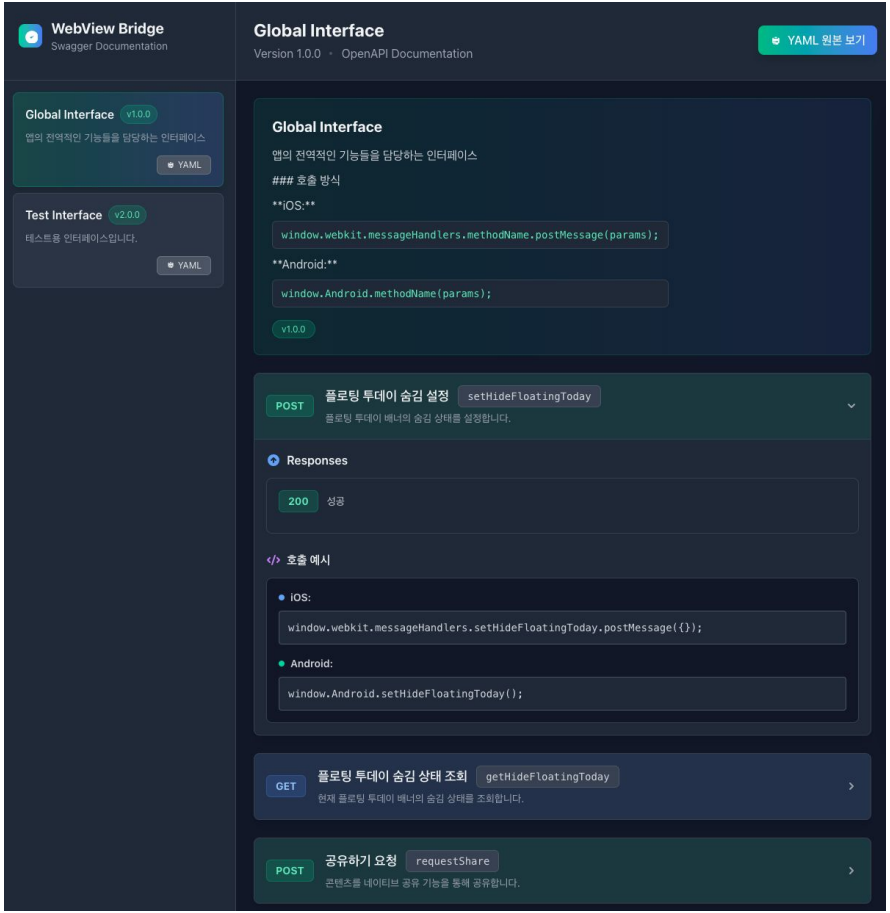
2. 네이티브 메서드 호출 디버깅 활용

본격적으로 네이티브와 정의한 메서드를 호출하여, 해당 메서드가 정상 호출되는지 확인이 가능합니다.
각각의 네이티브 테스트를 통해 앱의 세부적인 상태를 확인할 수 있으며, 이를 각각 신규로 추가하여 신규 메서드 확인을 손쉽게 테스트가 가능합니다. 이벤트 리스너도 등록하여, 특정 기능에 대한 동작호출이 가능한지도 확인 가능합니다.

3. 실행 로그를 통한 디버깅 모드 지원

실행 로그를 해당 화면에 표시하여, 웹의 콘솔모드가 아닌 웹-앱 통신의 로그만 집중적으로 확인할 수 있습니다.
로그 지우기를 통해 불필요한 로그 정리 또한 가능하며, 휘발성 로그이므로 서비스에 영향을 끼치지 않습니다.
이를 통해 브리지 상태와 네이티브 메서드 호출 시에 오류발생에 따른 근거자료로 활용하여 앱 개발자와의 협업에도 활용 가능합니다.

TECH 기타 서비스 산출물



WebView Bridge Swagger

Webview 통신을 위한 Webview 버전의 API Swagger 페이지를 생성하였습니다.
Interface 를 yaml 파일로 정형화하여 관리하고, 해당 yaml 들을 웹 UI 페이지로 구현할 수 있게 자동화하였습니다. 개발자는 yaml 파일만 생성하여, 관리하면 되며 이를 통해 웹-앱 개발자간 규약을 명세할 수 있습니다.

해당 yaml 파일을 open-api, MCP 등을 통해 스크립트를 실행하면 웹/앱의 인터페이스에 맞게 코드로 자동화해주도록 구현하였습니다.

1. 웹-앱 버전의 API Swagger 활용을 통한 자동화

웹-앱 버전의 API Swagger 를 만들었습니다. 정형화된 yaml 파일만 개발자는 관리하며, yaml 파일이 생성되거나 변경되면 빌드 단계에서 자동으로 해당 화면을 업데이트하여, 통신규약을 업데이트합니다. 이를 통해 개발자는 코드만을 현행화하여 관리하면, 문서 및 통신 규약에 대해 고려하지 않고 현행화가 가능합니다.

2. 웹-앱 yaml 파일을 통한 웹/앱 인터페이스 자동화 생성

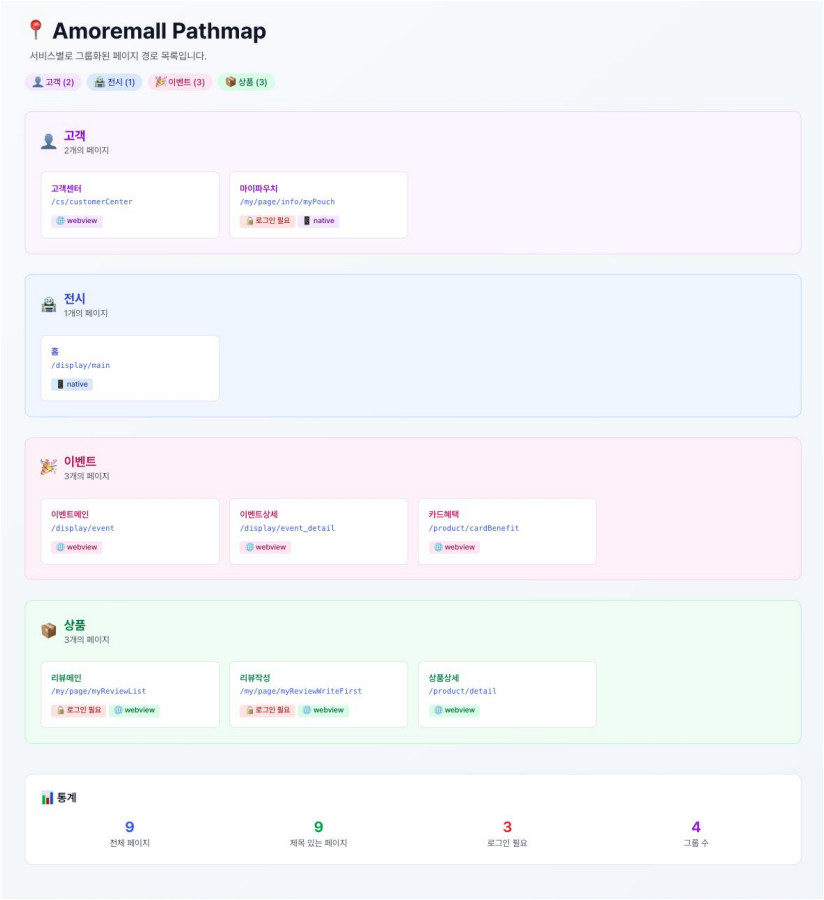
yaml 파일을 자동으로 웹/앱의 코드로 생성해주는 open-api, MCP 등을 통해, 스크립트 또는 프롬프트를 통해 yaml 에 명시된 인터페이스 명세를 구성할 수 있습니다. 웹의 경우 yaml 을 ts 파일로 생성하여 이를 활용해 웹-앱 간 양방향 통신을 원활히 합니다. 특히 자동화로 구현되기 때문에 개발자의 휴먼 에러를 방지하며, 품질 관리 및 개발 공수를 줄이는데 활용됩니다.

3. 문서 자동화를 통한 별도의 수기 문서 공수 축소

yaml 이 업데이트 되면 빌드 시에 문서가 자동 업데이트되어, 기존에 wiki 등의 수기 문서를 통해 관리하던 파편화 및 현행화의 한계를 해결하였습니다. yaml 을 웹/앱 관리자가 관리하여, 서로 통신에 대한 규약을 확인하고 사용방법에 대해 쉽게 확인이 가능하다는 장점이 있습니다.

TECH 기타 서비스 산출물

FE 개발자 | BE 개발자 | IT PMO
디자이너 | 기획자 | QA



개발자 Pathmap 서비스

해당 Pathmap 은 신규 개발자/QA/APP 담당자 분들이 애플리케이션의 페이지 종류와 상태를 한 곳에서 확인 가능한 서비스 페이지입니다. 좌측에 보이는 Amoremall Pathmap 을 활용해 해당 페이지로 이동이 가능하며, 페이지의 웹뷰/네이티브 여부, 로그인 필요 여부 등 정보 확인이 가능합니다.

1. 애플리케이션에서 제공하는 모든 서비스를 한 곳에서 관리

애플리케이션에서 제공하는 모든 서비스 페이지를 한 페이지에서 관리하여, 이를 통해 필요한 정보들을 관련자들이 확인하고, 이를 서비스 체크나 정리를 하는데 활용이 가능합니다. 특히 신규 개발자가 모든 서비스의 정보를 확인할 때 유용합니다.

2. 빌드 단계에서 Pathmap 구성 자동화 지원

개발 단계에서 각각의 page 레벨에 path.json 형태로 추가로 원하는 정보를 넣는 수동 방식을 제외하고, 보편적인 정보들은 빌드 단계에서 해당 애플리케이션의 트리를 찾아 경로를 탐색하고, 정형화된 공통 정보들과 각각의 path.json 의 데이터를 취합하여 좌측의 화면을 자동으로 구성해 줍니다.

3. 정적 사이트 배포 및 빌드 최적화

해당 서비스 페이지는 Staging 단계까지만 활용되는 개발자를 대상으로 하는 페이지이므로, 운영 단계에서는 빌드 제외됩니다. 또한 Pathmap 의 목적성에 맞게 최소한의 기능으로 목적에 집중하여 불필요한 리소스를 최소화하였습니다.

THANK YOU