김대건 Frontend Developer

Email toothless042@gmail.com
Github https://github.com/toothlessdev
Blog https://www.toothlessdev.com

INTRODUCE.

사용자 경험과 구현 과정의 비효율을 구조의 문제로 바라보고 해결하는 프론트엔드 개발자입니다.

하드코딩 좌표 대신 삼각함수 기반 파라미터형 레이아웃으로 변경 대응과 QA 비용을 줄였고,

관리자 대량 업로드에서는 이미지 압축과 청크 분할 처리, 실패 항목 재시도 구조를 설계해
운영 안정성을 높였습니다.

앞으로도 반복되는 문제를 구조적으로 해석하고 더 나은 흐름으로 바꾸는 개발자가 되고자 합니다.

SKILLS.

Language	JavaScript, TypeScript
Core & Styling	React, Next.js, SCSS, Emotion, TailwindCSS
State Management	Redux Toolkit, Zustand, React Query

CORE VALUES.

1. **안정적인 구조와 명확한 흐름을 지향합니다**
   변하는 기술 위에서, 변하지 않는 구조와 흐름을 설계하는 것을 중요하게 생각합니다.
   보이지 않는 영역까지 고려한 구조적 설계를 지향합니다.

2. **기본기를 중요하게 생각합니다**
   문제 해결 과정에서 구현 능력보다 CS 기반의 이해가 더 중요함을 체감했습니다.
   기능이 아닌 원인을 추상화하고, 구조적으로 해석하려고 합니다.

3. **지식은 공유될 때 더 큰 가치를 만든다고 믿습니다**
   동아리 활동을 진행하며 학습과 토론을 통해 이해를 확장해왔으며,
   개인의 성장을 넘어 팀의 역량 향상에 기여하고자 합니다.

CAREERS.

Nuvilab Frontend Engineer Intern 2026.02 ~ 03

관리자 대용량 이미지 배치 업로드 최적화 및 UX 레벨 원자성 보장 (p.4~) [블로그]

- 이미지 압축 및 청크 분할 병렬 처리로 단일 요청 페이로드를 90% 절감
- 부분 성공 허용 구조를 도입해, 실패 항목만 잔존시키는 UX 복구 흐름 구현

삼각함수 기반 별자리 레이아웃 생성 알고리즘 설계 및 구현 (p.6~) [블로그]

- 하드코딩 좌표 대신 파라미터 기반 레이아웃 생성 구조로 전환해 확장성과 유지보수성 개선
- 디자인 QA 단계의 반복적인 수치 조정을 해결하기 위해 파라미터 튜닝 DevTool 제작

XState 기반 교육게임 상태 머신 아키텍처 설계 및 구현 (p.8~)

- 오디오, 인터랙션, 애니메이션 전이를 선언적 상태 차트로 모델링
- Top-level 이벤트 기반 상태 점핑 DevTool 을 구현해 테스트 및 QA 효율 개선

PROJECT.

#에디터 개발 # 인터랙션 고도화

이력서 피드백 플랫폼 KareerFit 2025.09 ~ 2025.11 [Github]

인원 – 프론트엔드 3명 (기여도 40%), 백엔드 4명
기술스택 – Typescript, React(v18), TailwindCSS, AWS S3, CloudFront

드래그 기반 선택 인터랙션 성능 개선 (p.10)

- 마우스 이동마다 setState 호출로 발생하던 고빈도 리렌더링 및 이벤트 드롭 문제 해결
- 좌표를 useRef 로 관리 및 명령형 DOM 업데이트로 인터랙션 중 리렌더링 최소화

EventBus 기반 인터랙션 아키텍쳐 설계 (p.11) [블로그]

- 상태·로직이 단일 컴포넌트에 집중되던 구조를 EventBus로 분리해 유지보수성과 가독성 개선
- EventPlugin 레이어를 도입해 원시 이벤트를 고수준 인터랙션 이벤트로 추상화
- 향후 Zoom·Pan 등 기능 확장을 고려한 구조로 설계

#최대 MAU 8.3천 #800석 완판 #UX개선

모의수능 온라인 신청 플랫폼 MOSU 2025.03 ~ 2025.09 [Github]

인원 – 프론트엔드 2명 (기여도 60%), 백엔드 4명, 디자이너 1명
기술스택 – TypeScript, Next.js, SCSS, TailwindCSS, GSAP, Tanstack Query, Vitest, Sentry, Google Analytics

모의수능 신청 플로우 구현 및 데이터 기반 UX 개선 [블로그]

- 메인 페이지 할인 혜택 배너 제안 및 도입을 통해 신청페이지 진입률 12% 개선
- 모바일 환경에서 다수 집계되던 유효성 오류를 자동스크롤 및 포커스 이동으로 UX 개선

외부 PG사 SDK 연동을 포함한 결제 플로우 설계 및 테스트 자동화

- 각 결제 단계를 핸들러 단위로 분리해 결제 수단 확장 및 테스트가 용이한 구조로 설계
- PG사 SDK 호출 제약 환경에서도 검증이 가능하도록 Mock 기반 단위/통합 시나리오 테스트
- 결제 플로우 테스트 커버리지 89%를 확보하고 프로덕션 환경에서의 결제 문제를 사전에 방지

공지 및 FAQ 페이지 렌더링 전략 설계 [블로그]

- 관리자 페이지에서 콘텐츠 수정시 API Route 를 통해 Revalidate 트리거
- 콘텐츠 수정 빈도에 맞춘 Next.js 주문형 ISR 렌더링 전략 적용

#디자이너와의 협업 #성능 최적화

2024 경북대학교 디자인학과 졸업전시 웹사이트 2024.08 ~ 2025.02 [Site] [Github]

인원 – 프론트엔드 2명 (기여도 70%), 디자이너 4명
기술스택 – Typescript, React(v18), EmotionCSS, S3, CloudFront, Lambda, Google Analytics

페이지 전환 애니메이션 구현 및 성능 최적화 (p.12~)

- 가로로 이어진 아코디언 형태의 페이지 전환 UI를 구현하고 전환 과정의 성능 병목 개선
- 동적 언마운트 구조를 도입해 불필요한 DOM을 제거하고 메모리 사용량을 약 56% 절감
- Stacking Context 분리와 GPU 가속 전환을 적용해 렌더링 부하 및 프레임 드랍 최소화

대용량 이미지 갤러리 성능 최적화 (p.14) [블로그]

- 다수의 고해상도 이미지가 동시에 로드되며 발생했던 초기 렌더링 지연 문제 개선
- 사전로드 및 지연로드 전략을 활용해 FCP 32% 개선, CLS 0 달성

콘텐츠 관리 시스템 구현 (p.14)

- 졸업생들이 전시 콘텐츠를 직접 수정 및 관리할 수 있는 환경 구축
- 업로드 단계에서 Lambda 기반 이미지 자동 변환 파이프라인 구축

ACTIVITY.

경북대학교 멋쟁이사자처럼 11기 멤버 & 12기 대표 2023. 02. ~ 2024.12.

- 프론트엔드 기초 교육 세션 기획 및 진행 (Context 와 Compound Component 패턴)
- 2024 경북대학교 IT 연합동아리 GLOW 해커톤 주최 및 운영 (6개 동아리, 132명 참가)
- 경북대학교 멋쟁이사자처럼 13기 해커톤 프론트엔드 멘토

경북대학교 Google Developer on Campus KNU 운영진 2024. 06. ~ now

- React Fiber 와 Reconciliation 세션 발표
- JavaScript 실행컨텍스트 뜯어보기 세션 발표 [발표자료]
- NextJS 렌더링패턴과 Core Web Vitals 세션 발표 [발표자료]
- 전역상태 잘 설계하기 (Nested 한 전역상태에서 탈출하기) 세션 발표 [발표자료]

Storybook 오픈소스 컨트리뷰션 [PR #29480] 2024. 10.

- CLI 실행 시 가이드 부족으로 인한 불편함을 직관적인 커맨드 옵션을 추가하여 사용성 개선
- Monorepo 환경에서 CLI 패키지 구조와 테스트, Lint, CI 기반 기여 흐름 경험

AWARD.

2023 82스타트업 X LIKELION USA 아이디어톤 Security 부문 **대상** (Luddit Captcha)

2023 멋쟁이사자처럼 11기 대경권 연합해커톤 **대상** (우동사리)

2024 2학기 경북대학교 산학협력 프로젝트 경진대회 (Google) **우수상** (Look4Me)

2025 2학기 경북대학교 산학협력 프로젝트 경진대회 (위치스) **우수상** (StoryboardAI)

2025 한국정보기술학회 추계종합학술대회 논문 투고 및 수상 (위치스) **대상**

EDUCATION.

경북대학교 컴퓨터학부 졸업 (2026.02)

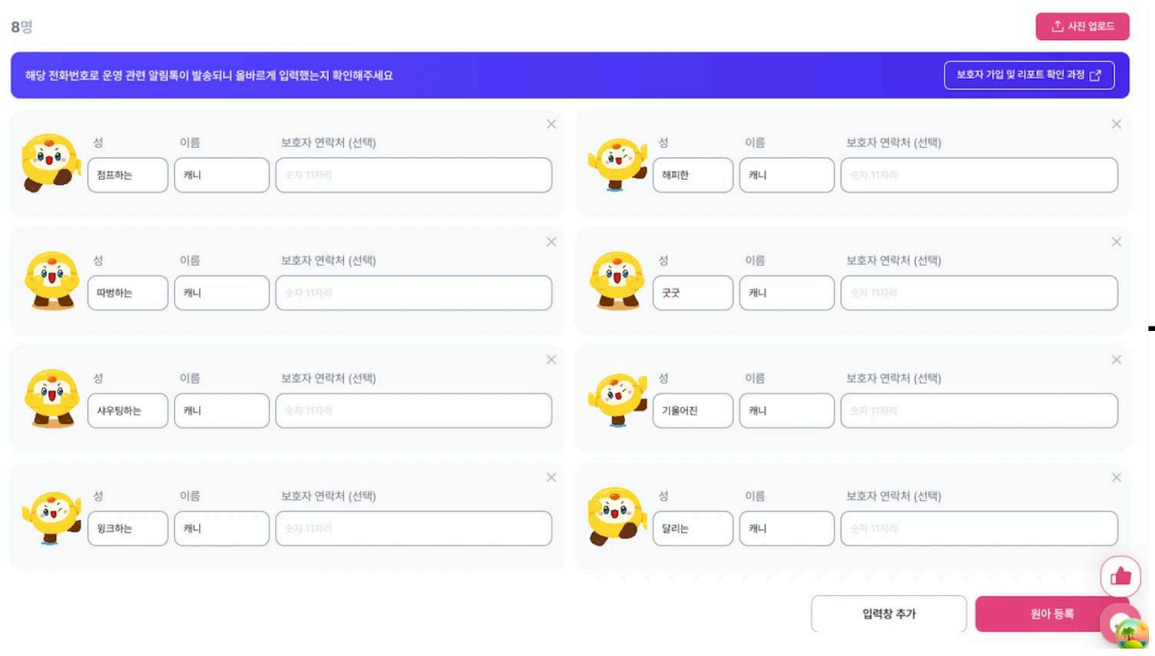
카카오테크캠퍼스 3기 프론트엔드 수료 (303시간)

관리자 대용량 이미지 배치 업로드 최적화 및 UX 레벨 원자성 보장

청크 분할 병렬 처리로 페이로드 90% 절감, 실패 항목만 재시도 가능하도록 구성

BACKGROUND

관리자 원아 일괄 등록 기능은 여러 원아 데이터를 단일 요청으로 전송하는 구조로 구현되어 있었습니다
각 원아는 프로필 이미지(평균 3~7MB)와 이름, 전화번호 등의 메타데이터가 하나의 단위로 묶여 함께 전송되었습니다

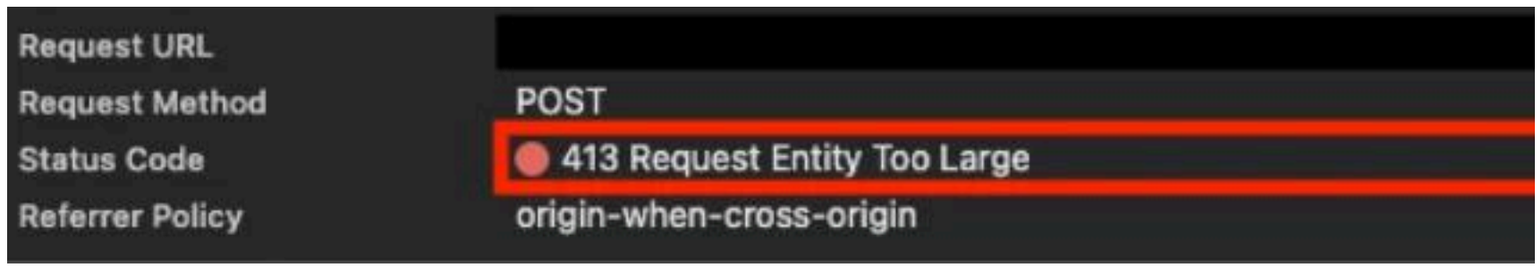



```
POST /batch-upload
Content-Type: multipart/form-data

files[]
metadata[]
```

PROBLEM - 01

대용량 데이터를 **단일 multipart 요청으로 처리하는 구조로 인해 요청 크기 제한이 초과**하는 문제가 발생했습니다
이로 인해 요청이 **서버에 도달하기도 전에 413 에러로 차단**되고 있었습니다

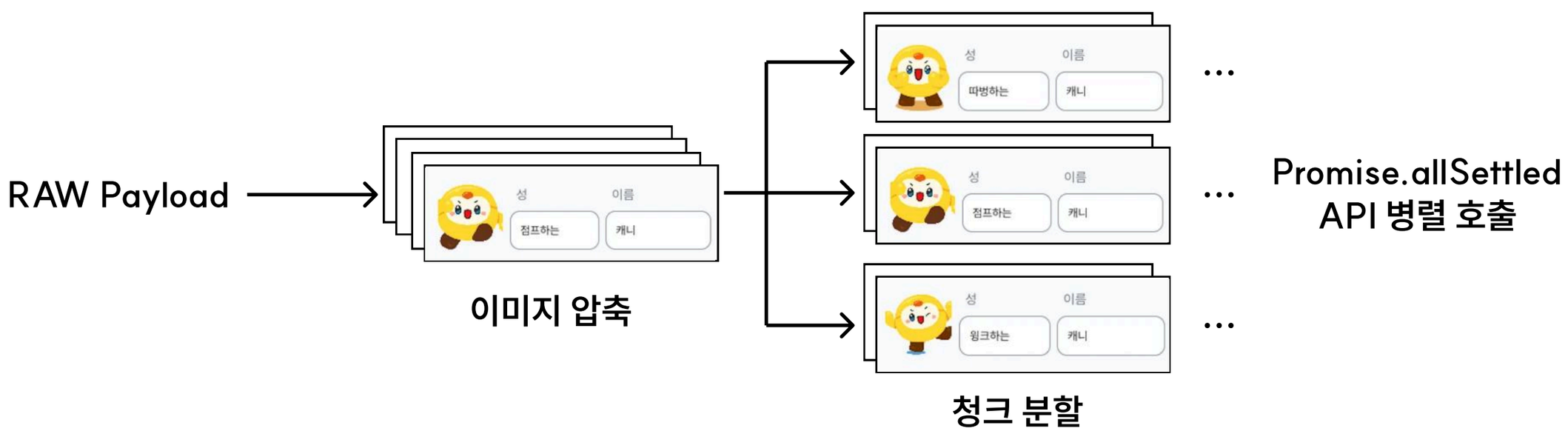


SOLUTION - 01

이미지 압축과 요청 청크 분할을 결합한 배치 업로드 구조를 도입하여
단일 요청을 여러 요청으로 분산함으로써 Payload 제한 문제를 해결했습니다.

대안으로 S3 Presigned URL 방식도 고려했으나,
백엔드/인프라의 변경이 필요하며, 이미지 업로드와 원아 등록을 하나의 흐름으로 보장하기 어려웠습니다.
또한 부분 실패 시 S3 고아 객체가 발생할 수 있는 문제가 존재했습니다.

이에 따라 기존 API를 유지하면서 프론트엔드에서 제어 가능한 구조를 선택했습니다.



PROBLEM - 02

요청을 여러 개로 분할하는 구조로 전환하면서, **일부만 성공하는 부분 성공 상태가 발생**할 수 있다고 판단했습니다.
이러한 구조에서는 다음과 같은 문제가 발생할 수 있었습니다.

1. **데이터 정합성 문제**
   일부 데이터만 등록된 상태에서 사용자가 흐름을 이탈할 경우, 불완전한 데이터가 시스템에 남아있을 수 있음
2. **UX 관점 일관성/신뢰성 붕괴**
   일부만 등록되고 일부는 실패하면서 사용자가 어떤 데이터가 정상 등록되었는지 파악하기 어려움

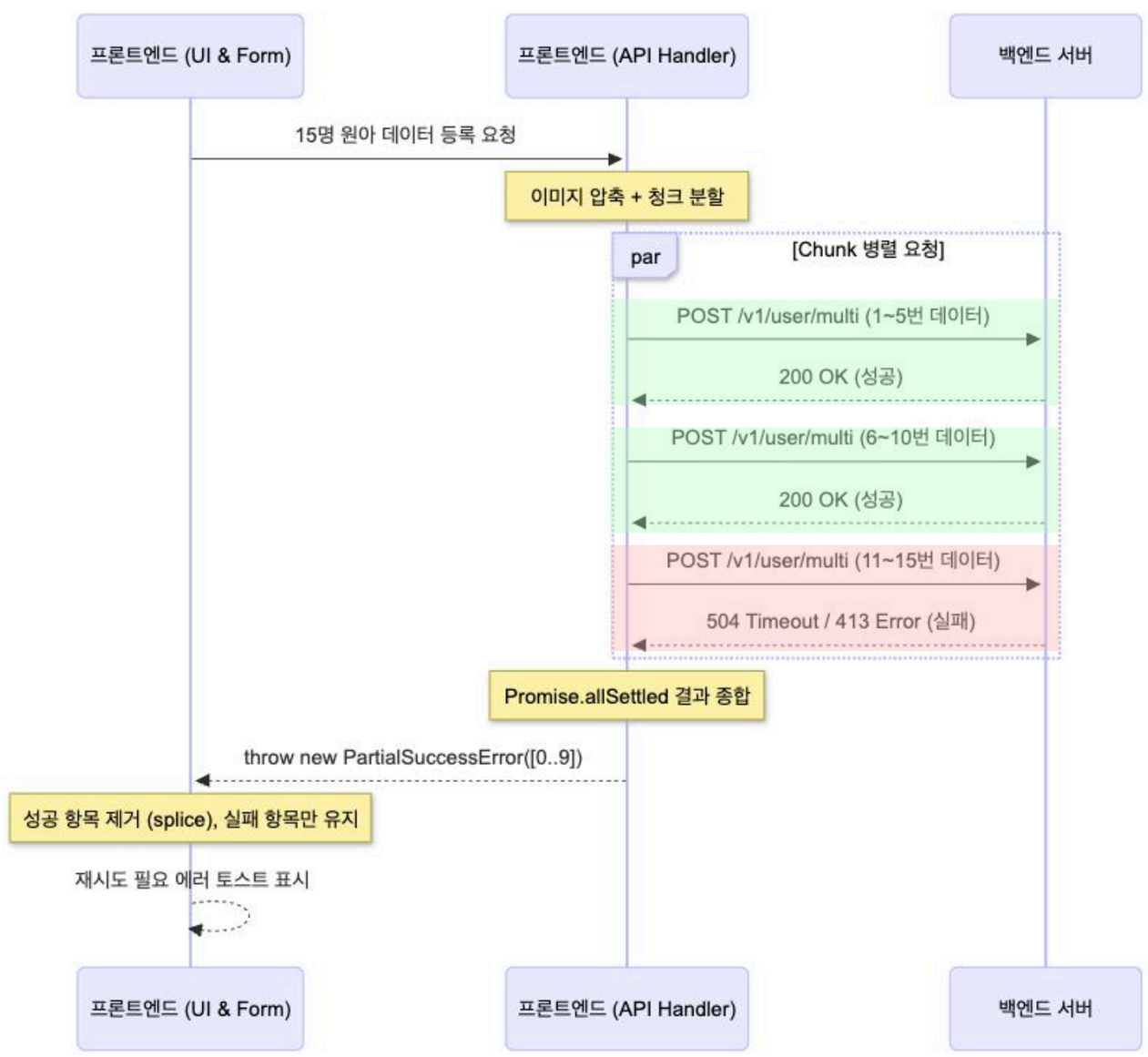
SOLUTION - 02

부분 성공 상태를 제어하기 위해 두 가지 방안을 고려했습니다.

1. **서버 데이터 롤백 (Delete API 기반 원자성 보장)**
   일부 요청만 성공한 경우, 이미 등록된 데이터를 삭제 요청으로 되돌리는 방식
2. **UX 레벨 원자성 보장**
   요청 결과 중 실패한 항목만 재시도 가능하도록 폼 상태를 재구성하는 방식

두 방식 중 서버 롤백 방식은 Delete API 호출 도중 네트워크 오류 등으로 롤백이 실패할 경우,
오히려 데이터 불일치가 심화될 수 있다는 리스크가 있었습니다

따라서 데이터 정합성을 서버에 위임하기보다,
프론트엔드에서 **요청 결과를 기준으로 상태를 재구성하는 UX 레벨 원자성 보장 방식**을 채택했습니다.



RESULT

- 부분 성공으로 인한 데이터 혼란을 제거하고,
- 사용자가 실패한 항목을 바로 확인하고 수정할 수 있도록 했으며,
- 실패 항목만 남겨 전체 필드 재입력을 하지 않아도 되도록 구현해 사용자 경험을 개선했습니다.

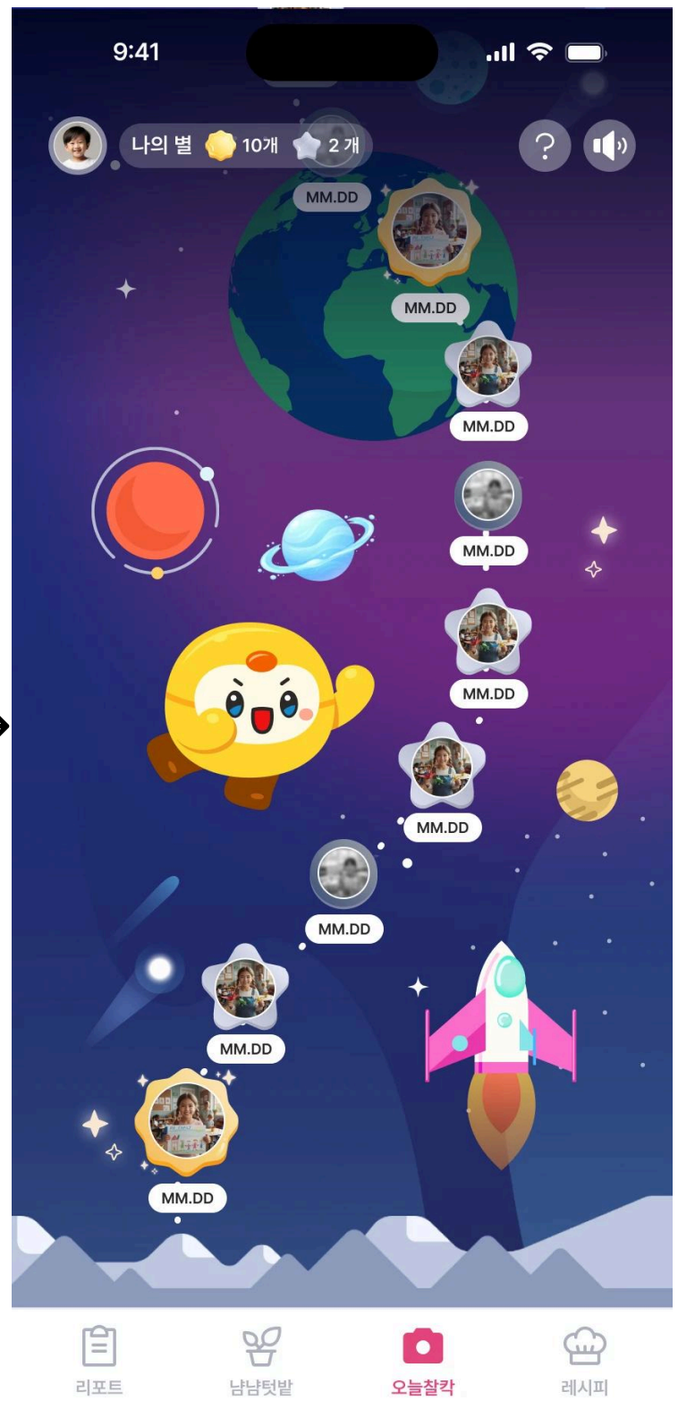
삼각함수 기반 별자리 레이아웃 생성 알고리즘 설계 및 구현

하드코딩 좌표 대신 파라미터 기반 레이아웃 생성 구조로 전환

BACKGROUND

기존 오늘찰칵 서비스는 단순 사진 나열 중심의 갤러리 UI로, 사용자 몰입 경험부족으로 인해 리텐션이 떨어졌습니다.
이를 개선하기 위해, 노드를 따라 이동하며 경험하는 별자리 기반 인터랙션 UI를 도입하게 되었습니다.

기존 UI



개선된 UI

PROBLEM

초기 접근은 노드 좌표를 직접 하드코딩하는 방식이었으나 다음과 같은 문제가 존재했습니다

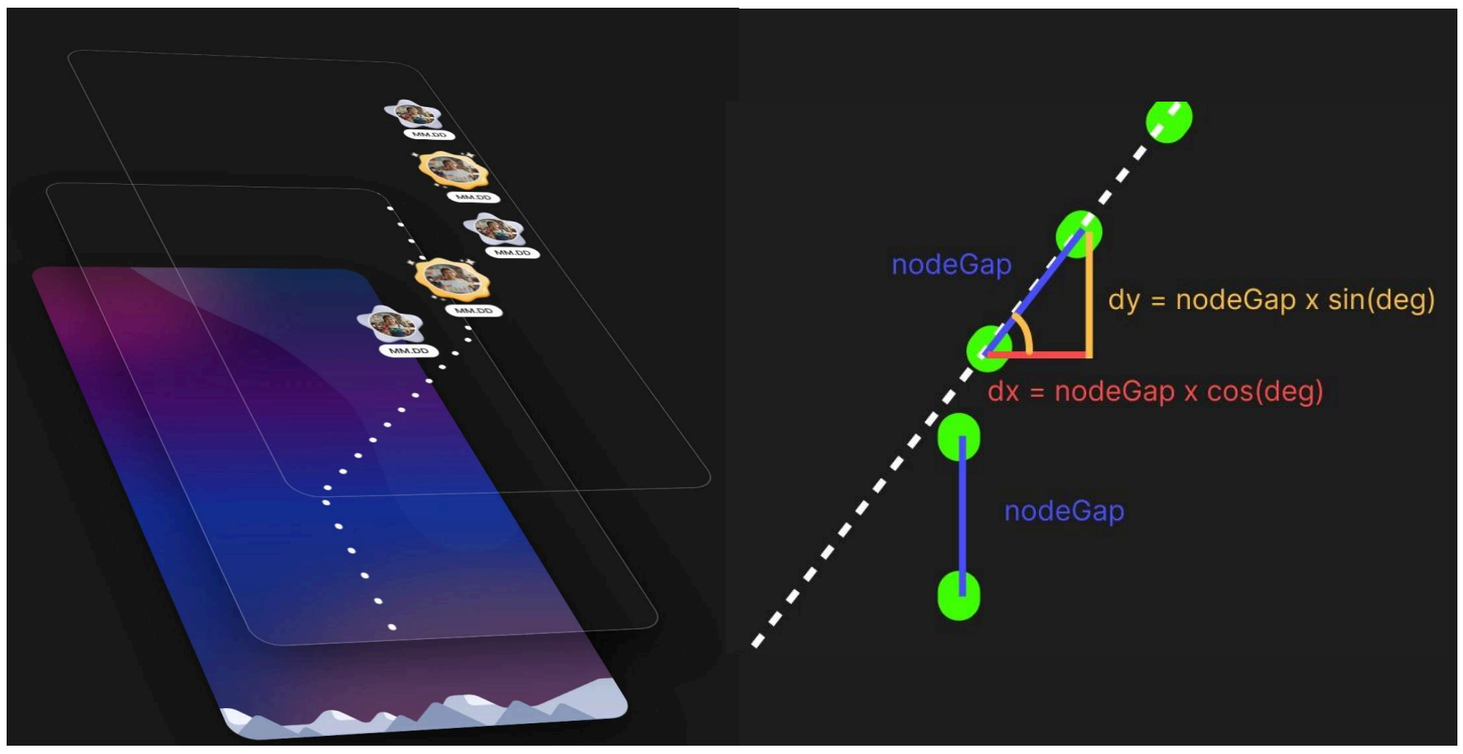
1. 노드 개수 또는 경로 변경 시 전체 좌표를 다시 계산해야 하는 구조
2. 디자인 QA 요청 대응 시 좌표 수정 비용 과다
3. SVG Path 기반 접근은 WebView 환경에서 성능 부담 및 개별 노드 제어 한계

결과적으로 **하드코딩된 좌표 기반 구현은 확장성과 운영 대응 측면에서 부적합**한 구조였습니다

SOLUTION

좌표를 직접 지정하는 방식 대신, **좌표를 규칙 기반으로 생성**하는 구조로 전환했습니다.

노드의 위치를 시작점, 간격(nodeGap), 각도(offsetAngle), 방향 패턴(offsetArray)으로 정의하고,
삼각함수를 활용해 dx, dy를 계산하여 좌표를 동적으로 생성하도록 설계했습니다.

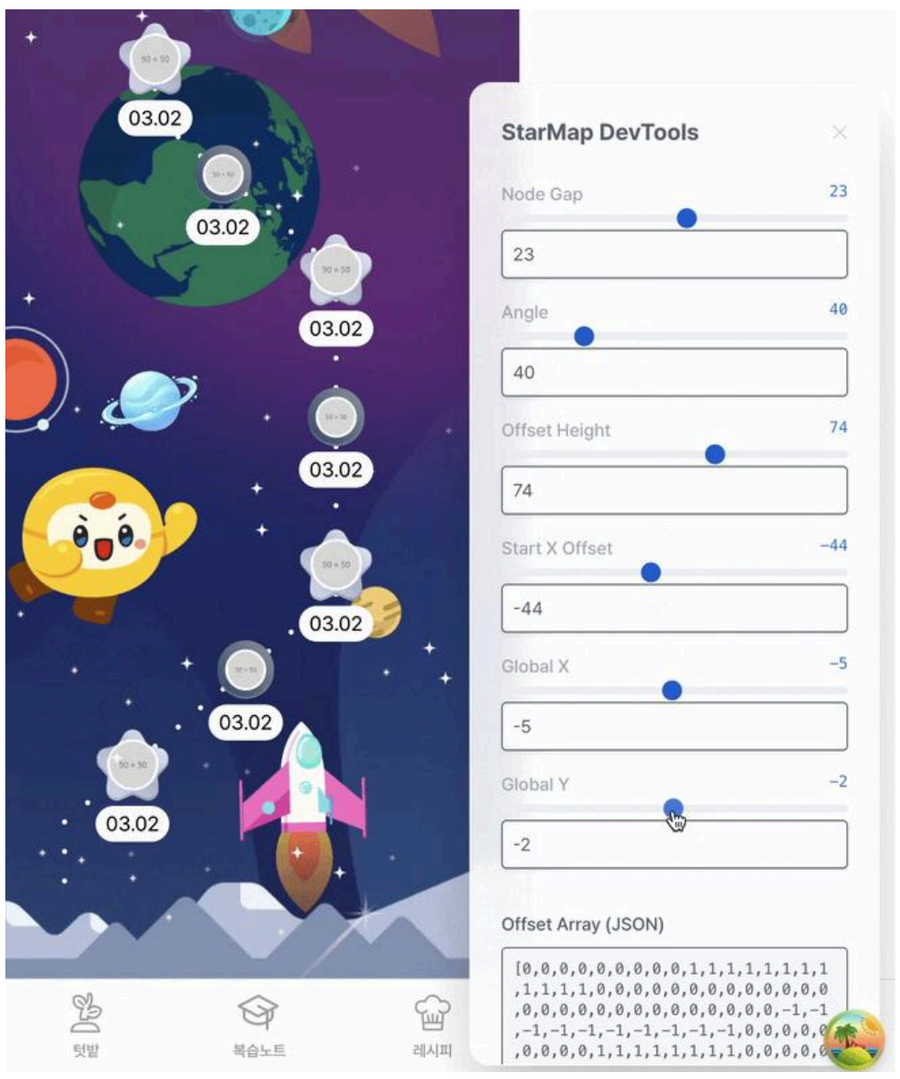


또한 **좌표 생성 로직과 UI를 분리하기 위해 Render Props 패턴을 적용**하여,
노드 표현 방식에 대한 확장성과 유연성을 확보했습니다.

```
<StarMapLayer
    totalNodes={offsetArray.length}
    nodeGap={20}
    angle={52}
    offsetArray={offsetArray}
    offsetStartX={-100}
    render={(node, index) => {
        // 5번째 노드만 별자리 마커로 표시
        if (index % 5 === 0) return <StarMapMarker node={node} index={index} />;
        return <DotMarker node={node} />;
    }}
/>;
```

RESULT

노드 개수나 경로가 변경되더라도 **전체 좌표를 다시 계산할 필요 없이 파라미터 조정만으로 대응**할 수 있게 되었습니다.
또한 DevTools를 통해 **QA 조정 시간을 수시간에서 수분 단위로 단축**했습니다.



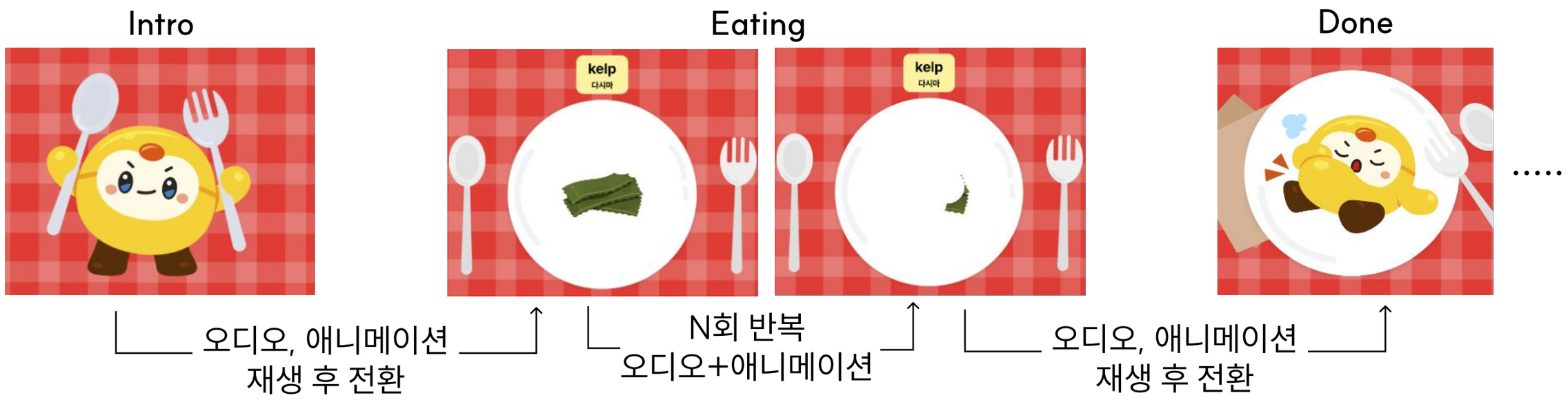
XState 기반 학습게임 상태 머신 아키텍쳐 설계 및 구현

분산된 상태 전이 로직을 단일 상태 차트 구조로 통합

BACKGROUND

키즈 플랫폼의 학습 게임(Fooding)은
오디오 재생 > 사용자 인터랙션 > 애니메이션 > 다음 단계 전이가 이어지는 멀티스텝 플로우였습니다.

기존에도 복잡한 조건 분기와 인터랙션 흐름을 다루는 로직은 존재했지만,
학습 게임은 **반복 루프, 조건 분기, 비동기 이벤트가 결합된 더 높은 수준의 상태 전이를 요구하는 구조**였습니다.



각 단계는 오디오/애니메이션 완료를 기준으로 다음 상태로 전이되며,
Eating 단계에서는 동일 상태 내 반복 루프가 발생하는 구조였습니다.

PROBLEM

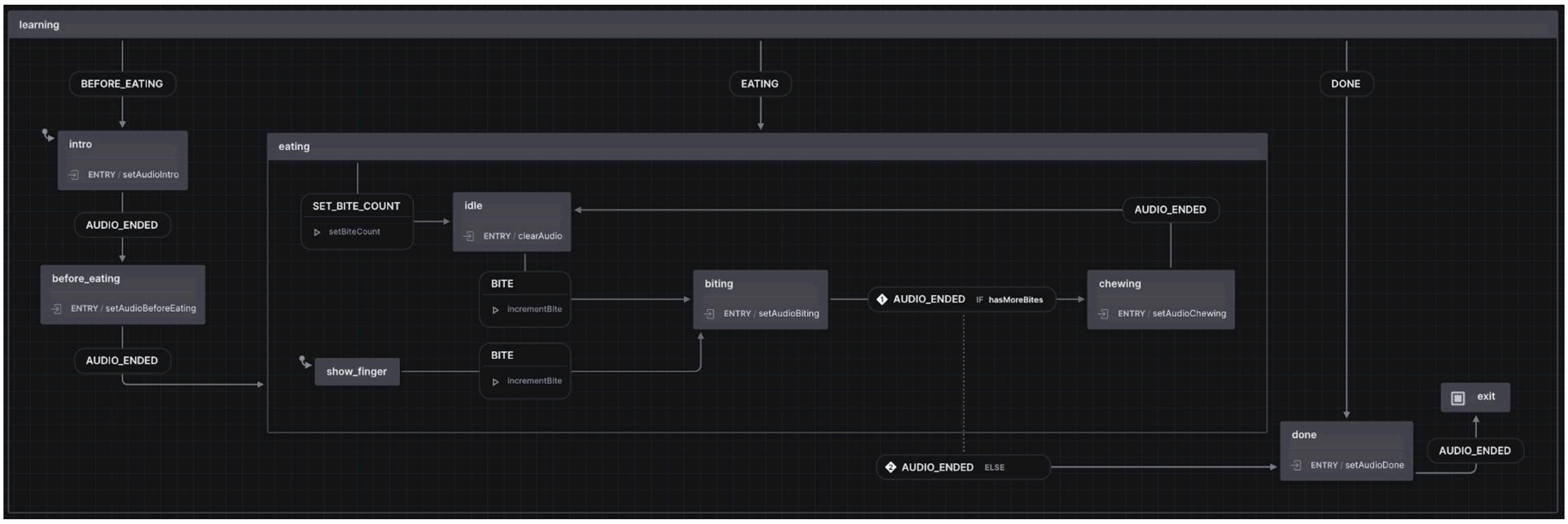
기존 방식 (useEffect, setTimeout, switch) 으로 구현할 경우 **상태 조합 증가와 전이 로직 분산**이 발생했습니다.
그 결과, 흐름을 안정적으로 관리하기 어려운 구조였습니다.

1. **상태 공간 추적 불가** : 독립적인 useState 조합 증가로 전체 상태 흐름 파악 어려움
2. **의존성 관리 복잡도 증가** : useEffect 의존성 수동 관리로 누락 및 race condition 발생
3. **전이 로직 분산** : setTimeout + switch 기반으로 로직이 흩어져 흐름 제어 어려움
4. **QA 비효율** : 특정 상태 재현을 위해 전 과정을 반복 수행해야 하는 구조

이러한 문제를 해결하기 위해, 상태 전이를 명시적으로 관리할 수 있는 구조가 필요했습니다.

SOLUTION

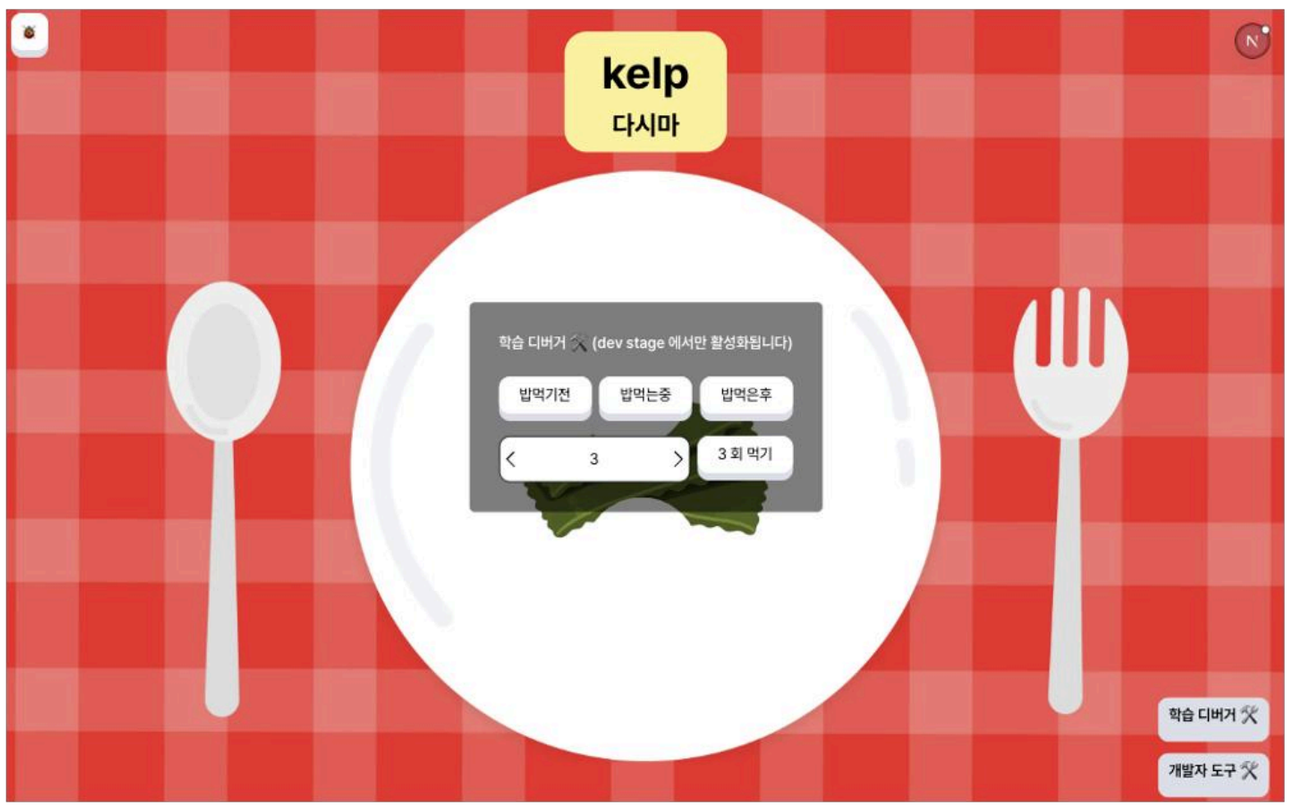
전체 학습 플로우를 상태 단위로 분리하고, 각 상태 간 전이를 이벤트 기반으로 명시적으로 정의했습니다.



1. 전체 학습 플로우를 **하나의 상태 머신으로 통합**하여 전이를 단일 구조로 관리
2. Eating 단계는 Nested State로 구성하여 반복 루프를 명시적으로 모델링
3. 상태와 UI를 1:1로 매핑하여 흐름과 화면을 일치시키는 구조로 설계

RESULT

상태 전이 로직이 **단일 상태머신 구조로 통합**되면서, 분산되어 있던 **흐름을 한 곳에서 관리**할 수 있게 되었습니다.
상태 머신 정의만으로 전체 플로우를 이해할 수 있어, **별도 문서 없이도 상태 흐름 공유가 가능**해졌습니다.




```
Open Visual Editor
return learningStateMachineSetup.createMachine({
    /** @xstate-layout N4IgpgJg5mDOIC5QBswEMBOA7AllqAxAEICi
    id: "learning",
    initial: "intro",
    context: {
        currentBiteIndex: 0,
        audioSrc: undefined,
    },
    on: {
        BEFORE_EATING: { target: ".intro" },
        EATING: { target: ".eating" },
        DONE: { target: ".done" },
    },
    states: {
        intro: {
            entry: "setAudioIntro",
            on: {
                AUDIO_ENDED: {
                    target: "before_eating",
```

Top-level Event를 통해 특정 상태로 즉시 전이할 수 있도록 구현하여,
QA 시 전 과정을 반복 수행하지 않고도 특정 상태를 즉시 재현할 수 있는 구조로 개선했습니다.

에디터 인터랙션 성능 최적화 및 구조개선

드래그 기반 선택 인터랙션 성능 개선 및 이벤트 아키텍처 설계

BACKGROUND

커리어핏은 사용자가 이력서/포트폴리오 위에서 드래그로 영역을 선택하고 댓글을 남기는 인터랙션이 핵심 기능입니다.
해당 기능은 마우스 드래그에 따라 실시간으로 UI가 반응해야하는 고빈도 인터랙션이 필요했습니다

PROBLEM

초기 구현에서는 드래그 좌표를 React 상태로 관리하며, onMouseMove 마다 setState 가 호출되는 구조였습니다.
이로인해 다음과 같은 문제가 발생할 수 있었습니다

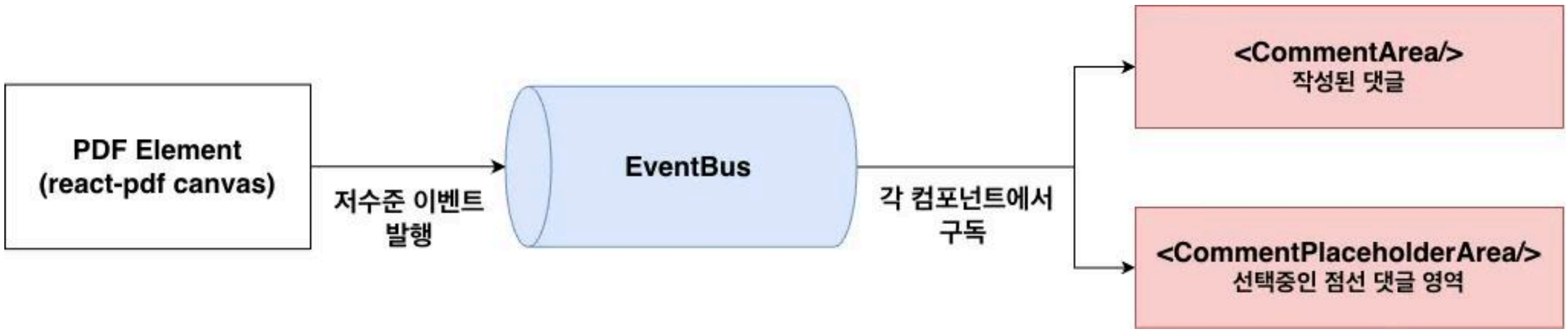
1. **고빈도 상태 업데이트로 인해 프레임 드롭 및 이벤트 드롭 발생**
   드래그 도중 영역을 표시하는 미리보기와 같은 임시 상태까지 React 렌더링 사이클에 포함
2. **상태, 이벤트 처리, 비즈니스 로직이 단일 컴포넌트에 결합되어 확장 및 유지보수 어려움**
   Zoom, Pan 등 추가 인터랙션 기능 도입시 복잡도 증가 예상

SOLUTION - 01

고빈도 인터랙션은 **React 상태 기반으로 처리할 경우 Reconciliation 비용이 병목**이 된다고 판단했습니다.
따라서 드래그 **좌표를 useRef 로 관리하고, 선택 박스를 명령형 DOM 업데이트로 제어**하여
고빈도 UI 갱신이 리액트의 렌더링을 유발하지 않도록 개선했습니다.

SOLUTION - 02

상태, 이벤트 처리, 비즈니스 로직이 단일 컴포넌트에 결합된 문제를 해소하기 위해
EventBus 를 사용해 이벤트 흐름을 분리했습니다.



PDF Element 는 이벤트를 발행하고, 각 컴포넌트는 이벤트를 구독하는 구조로 전환했습니다.

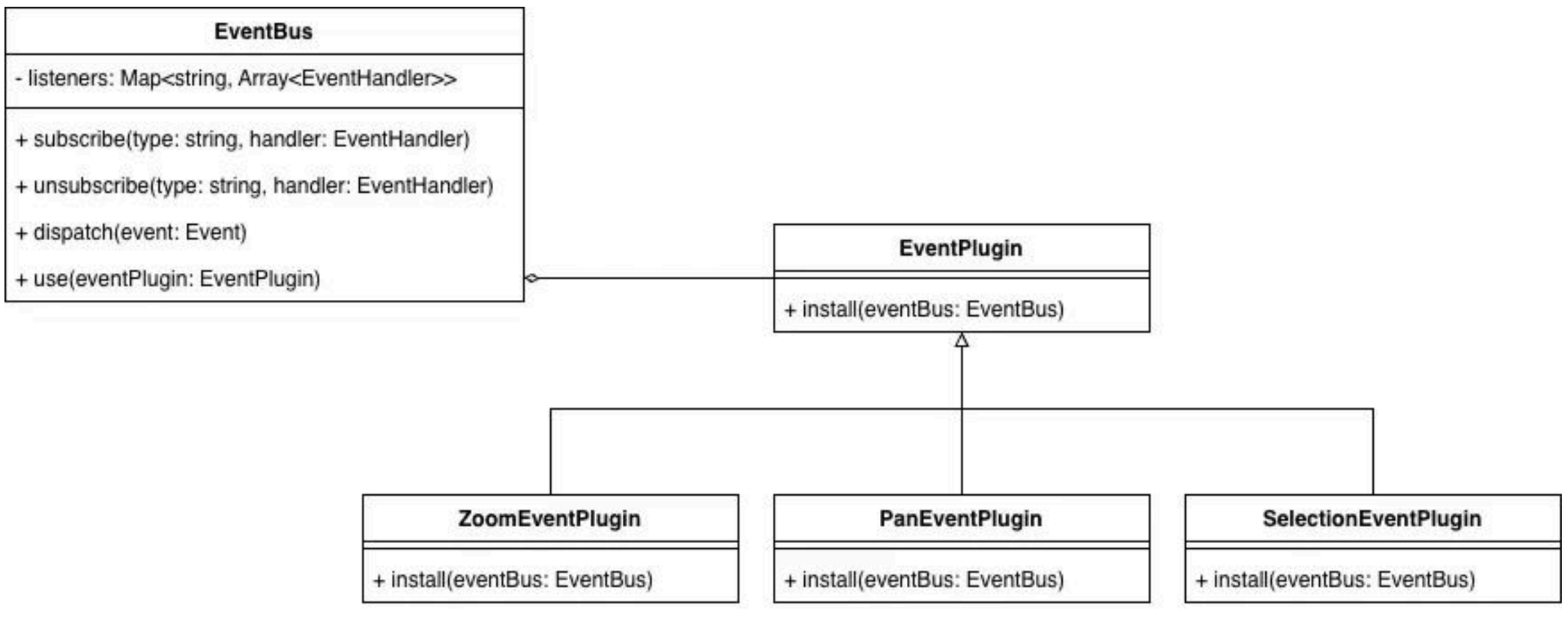
이를 통해 UI 표현과 인터랙션 처리 로직을 분리하고, 컴포넌트가 복잡한 이벤트 흐름을 직접 관리하지 않도록 개선했습니다.

다만, mousedown, mousemove, mouseup 과 같은 원시 이벤트를 그대로 전달할 경우,
EventBus 가 비즈니스 로직까지 떠맡게 되는 한계가 존재했습니다.

SOLUTION - 03

EventBus 의 한계를 극복하기 위해 EventPlugin 레이어를 도입했습니다.

EventPlugin 은 **저수준 이벤트를 selection:start, selection:move, selection:end 와 같은 도메인 이벤트로 변환**하여
이벤트의 의미 단위를 명확히 분리했습니다.



또한, 각 인터랙션을 **Plugin 단위로 확장 가능하도록 설계**하여
Zoom, Pan 등 **신규 인터랙션을 코드 수정 없이 독립적으로 추가할 수 있는 구조를 구축**했습니다.

RESULT

드래그 중 고빈도 상태 업데이트가 리렌더링을 유발하던 구조를 제거하여,
이벤트 드롭 없이 안정적인 선택 인터랙션을 제공하도록 개선했습니다.

또한 저수준 이벤트와 도메인 이벤트를 분리한 구조를 설계하여,
복잡한 인터랙션을 명시적으로 관리하고 Zoom, Pan 등 신규 기능을 독립적으로 확장할 수 있는 기반을 마련했습니다.

페이지간 아코디언 애니메이션 최적화

렌더링 구조 개선을 통한 성능 병목 해결

BACKGROUND

전시 웹사이트는 가로 방향으로 페이지가 확장, 축소되는 아코디언 형태의 전환구조로 설계되었습니다.
전체 사이트에 적용되는 애니메이션인 만큼, 부드러운 전환 경험을 제공하기 위해 애니메이션 품질이 핵심 요소였습니다.

PROBLEM

초기 구현에서는 모든 페이지를 동시에 마운트한 상태로 max-width 를 통해 전환되는 구조를 사용했습니다.
이로인해 다음과 같은 문제가 발생했습니다.

1. **보이지 않는 페이지의 DOM 이 메모리를 점유**
   페이지 수가 증가할수록 렌더링 비용이 지속적으로 증가하는 문제 발생
2. **max-width 기반 애니메이션으로 인한 프레임 드랍**
   반복적인 레이아웃 재계산으로 인해 Reflow 가 빈번하게 일어나 프레임 드랍과 끊김 발생

SOLUTION

렌더링 범위를 최소화하고, 레이아웃 기반 애니메이션을 컴포지터 기반으로 전환하는 방향으로 구조를 재설계했습니다

1. **ReactTransitionGroup 도입**
   보이지 않는 페이지는 애니메이션 종료 시점에 언마운트해, 불필요한 DOM 이 유지되지 않도록 개선했습니다

   단순히 display:none 을 활용해 DOM 을 숨기는 방식도 고려했으나
   DOM 은 여전히 메모리에 유지되며, 페이지 전환 흐름과 동기화된 애니메이션 처리가 어렵다는 한계가 있었습니다

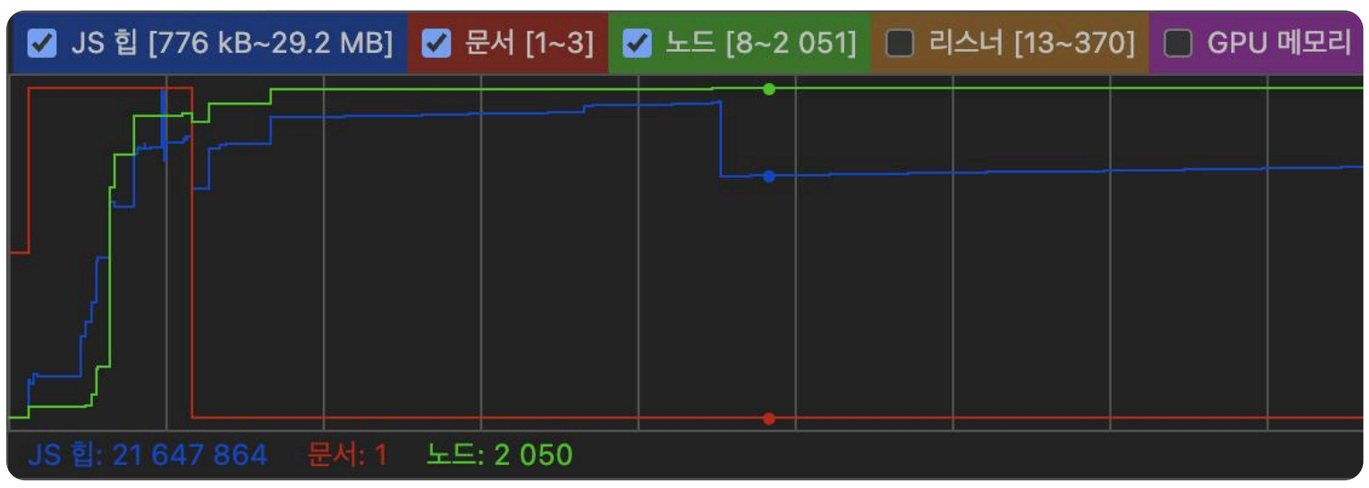
2. **페이지간 StackingContext 분리**
   각 페이지의 z-index 를 활용해 StackingContext 를 분리함으로써 애니메이션시 레이아웃 간섭이 발생하지 않도록 구현했습니다

3. **GPU 가속을 활용하는 transform 애니메이션으로 전환**
   기존 max-width 에서 transform 애니메이션으로 전환해 GPU Compositing 단계에서 처리하도록 개선했습니다.

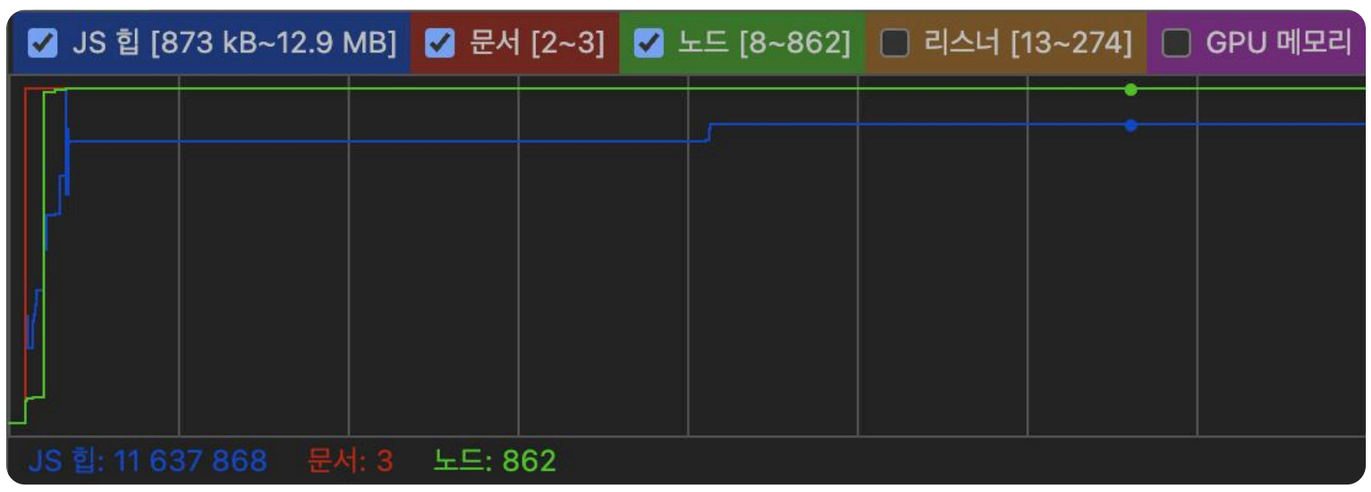
RESULT

보이지 않는 페이지를 언마운트하도록 구조를 개선함으로써

잔여 DOM 을 제거하고 메모리 사용량을 56% 절감했습니다 (29.2MB → 12.9MB)



개선 전



개선 후

또한 transform 기반 GPU compositing으로 전환함에 따라

페이지 전환 시 기존에 발생하던 **프레임 드랍과 끊김 현상이 해소**되었습니다.

그 결과, 페이지 수가 증가하는 대규모 DOM 구조에서도

성능 저하 없이 안정적인 전환 애니메이션을 유지할 수 있는 구조를 확보했습니다.

전시작품 상세 페이지로 전환 시, 이미지 로드로 인한 성능 병목 개선

이미지 전송 최적화를 통한 초기 로딩 병목 개선

BACKGROUND

졸업전시 웹사이트는 디자인 전공 학생들의 졸업작품을 감상하는 공간으로, 고화질 이미지로 구성된 페이지였습니다.
작품의 질감,색감을 정확히 전달해야 해 **이미지 품질 저하 없이 제공**하는 것이 도메인 특성상 필수였습니다.

특히 디자이너가 제작한 원본 이미지는 수천~만 픽셀 이상의 고해상도로 제공되는 경우가 많았고,
이미지를 그대로 사용하는 경우 성능 저하가 발생할 수 있는 상황이었습니다.

PROBLEM

초기에는 PNG 기반의 고용량 이미지를 그대로 사용하는 구조였기에 다음과 같은 문제가 발생했습니다.

1. **고해상도 이미지로 인한 초기 로딩 지연**
   갤러리 페이지 진입시 대용량 이미지 다운로드로 인해 초기 로딩 속도 저하
2. **메모리 사용량 증가 및 렌더링 부담**
   고용량 이미지가 브라우저에 로드되면서 OOM (Out Of Memory) 발생
3. **수동 이미지 최적화로 인한 운영 비효율**
   이미지 포맷 변환과 리사이징을 수동으로 처리하는 구조로 인한 운영 부담 증가

SOLUTION

이미지 품질을 유지하면서 전송 비용과 렌더링 부담을 줄이기 위해
포맷 최적화, 디바이스 대응 리사이징을 자동화 하는 구조로 변경했습니다.

1. **이미지 포맷 최적화 (PNG → WebP)**
   고효율 압축이 가능한 WebP 포맷으로 변환하여, 이미지 품질을 유지하면서 전송 크기를 절감
2. **디바이스 기반 리사이징 전략 도입**
   모바일/PC 해상도에 맞는 여러 크기의 이미지를 생성해, 불필요한 데이터 전송 감소
3. **이미지 처리 파이프라인 자동화 (S3 + Lambda)**
   이미지 업로드시 S3 이벤트를 트리거로 Lambda 를 실행하여 포맷변환가 리사이징이 자동으로 수행되도록 처리

이를 CMS 업로드 단계에 통합하여,
디자이너가 원본 이미지를 그대로 업로드하더라도 별도의 수작업 없이 일관된 최적화가 적용되도록 구조를 설계했습니다

RESULT

WebP 변환과 디바이스별 리사이징을 통해 이미지 전송 크기를 줄이고
상세 페이지 초기 로딩 속도를 개선했습니다.

또한 CMS 업로드 시 자동 변환 파이프라인을 구축해
운영자가 별도 최적화 없이도 일관된 품질과 성능을 유지할 수 있도록 개선했습니다.

이미지 로드 타이밍 제어를 통한 체감 성능 개선

BACKGROUND

상세 페이지는 전환 직후 사용자가 가장 먼저 접하는 상단 주요 이미지의 즉각적인 노출이 중요한 구조였습니다.

사용자는 페이지 진입과 동시에 콘텐츠가 준비되었다고 인지하기 때문에,
초기 화면에 해당하는 주요 이미지의 빠른 표시가 사용자 경험에 직접적인 영향을 미치는 상황이었습니다.

PROBLEM

초기 구현에서는 상단 주요 이미지와 하단 본문 이미지가 구분 없이 동시에 로드되는 구조를 사용하고 있었습니다.
이로인해 초기 렌더링 시점에 불필요한 리소스가 함께 로드되며 다음과 같은 문제가 발생했습니다.

1. **주요 이미지 노출 지연**
   우선순위가 낮은 하단 이미지까지 동시에 로드되며, 가장 먼저 보여야하는 상단 이미지 로드 지연
2. **초기 렌더링 부하 증가**
   하단의 불필요한 이미지 요청이 초기 시점에 집중되며 네트워크 및 디코딩 비용이 증가
3. **레이아웃 안정성 저하**
   이미지 로드 시점이 제어되지 않아 LayoutShift 발생 가능성 존재

SOLUTION

초기 렌더링에 필요한 리소스와 이후 로드해도 되는 리소스를 분리하고,
이미지 로드 타이밍을 사용자 인터랙션 흐름에 맞춰 제어하는 방향으로 구조를 개선했습니다.

1. **주요 이미지 사전로드**
   목록 페이지 호버 시점에 상세 페이지의 상단 주요 이미지를 preload 처리
2. **하단 이미지 지연 로드**
   Intersection Observer 를 활용해 하단 본문 이미지는 뷰포트에 진입하기 직전에만 로드되도록 구현

RESULT

상세 페이지 전환 직후 주요 이미지가 즉시 표시되도록 개선되며 FCP 를 32% 단축했습니다 (4.1s → 2.8s)

불필요한 초기 이미지 요청을 제거해 네트워크 및 렌더링 부하를 줄였고,
이미지 로드로 인한 LayoutShift 를 제거하여 CLS 를 0으로 개선했습니다 (0.214 → 0)