김아현

연락처 : 010-9790-9022

Email : gracias9022@gmail.com

Github : https://github.com/naro-Kim

Velog : https://velog.io/@naro-kim

미디어 도메인을 이해하는 Web Platform Frontend Engineer

미디어 재생·업로드·결제·인증처럼 실패 비용이 큰 사용자 여정을 안정적인 Web architecture로 전환해왔습니다. 대용량 미디어 스트리밍 최적화 및 공통 인프라(디자인 시스템, 패키지) 구축에 특화되어 있습니다.

- HLS 미디어 최적화 : 2시간 이상 대용량 비디오의 멀티트랙 재생 및 싱크 오차(Audio/Video Sync) 누적 문제를 디버깅하여 미디어 재생 신뢰성 확보
- 성능 최적화 및 아키텍처 안정화 : Next.js App Router 점진적 마이그레이션을 주도하여 **번들 사이즈 64% 감소 및 사용자 체류 시간 20% 향상 기여**
- 생산성 고도화 : 사내 디자인 시스템 및 DesignOps 구축으로 전사적인 생산성 향상에 기여. 디자인-개발 **협업 리소스 53.6%에서 6.5%로 대폭 절감**

프로젝트 경험

Gaudio Studio Pro - AI 영상 컨텐츠 현지화 플랫폼

2026.01 - 2026.06

담당 역할 : Frontend Engineer, 아키텍처 설계 및 개발 리드

- 플랫폼 내 HLS 재생 방식을 기획하고 Multitrack Player Architecture 및 플랫폼 내 애플리케이션 공통 package 설계, 구현
- Playback/Buffer/Region lifecycle과 동시 track 제어, runtime track 추가, 사용자 comment 기능 구현
- RBAC PermissionGuard와 action provider 기반 페이지 권한 별 panel composition 제어 구축
- Audio/Video Sync 문제를 진단하고 Media workflow 개발자 및 Media QA와 원인 분석, 수정 및 재검증, 플랫폼 운영 안정성 기여

기술 스택 : TypeScript, Next.js, HLS.js, wavesurfer.js, Zustand, Playwright

배경

Gaudio Studio Pro는 콘텐츠 편집자와 검토자가 기존 영상의 입 모양과 더빙을 현지화하고, dialogue/music/effect track을 개별 또는 통합 검수하는 B2B 플랫폼이었습니다. 2시간이 넘는 영상과 여러 HLS track, Region 기반 comment를 하나의 Player에서 다루기 때문에, 각 미디어의 독립적인 lifecycle을 유지하면서도 **하나의 timeline으로 동작하는 재생 안정성과 Audio/Video Sync 정확도를 보장**하는 것이 핵심 과제였습니다.

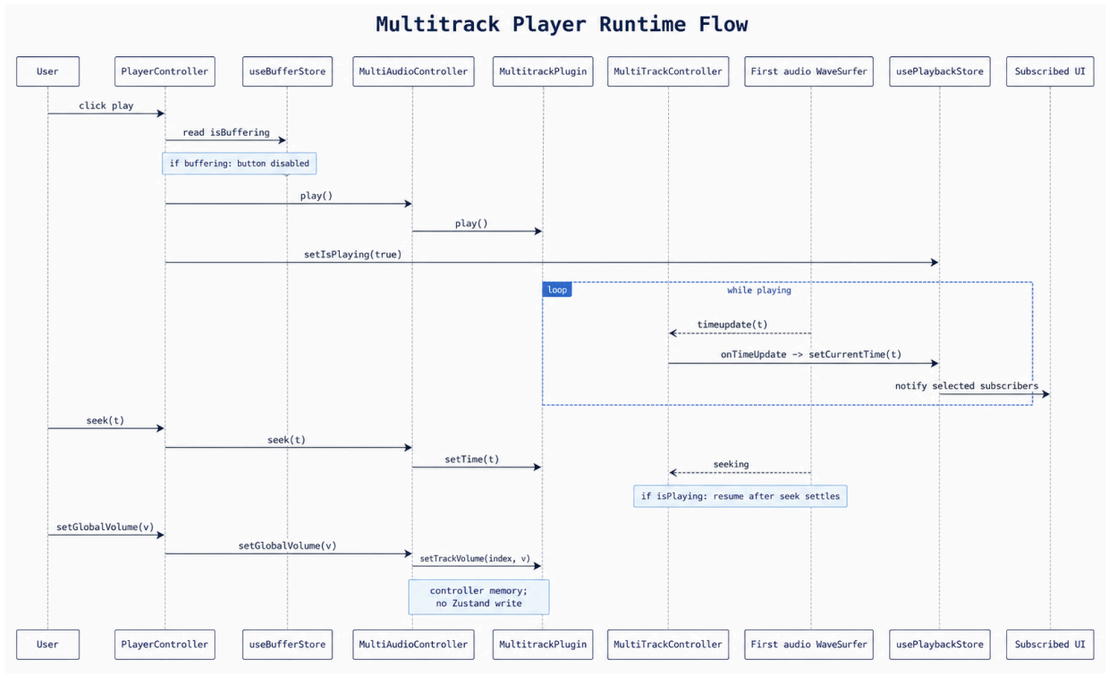
문제 인식

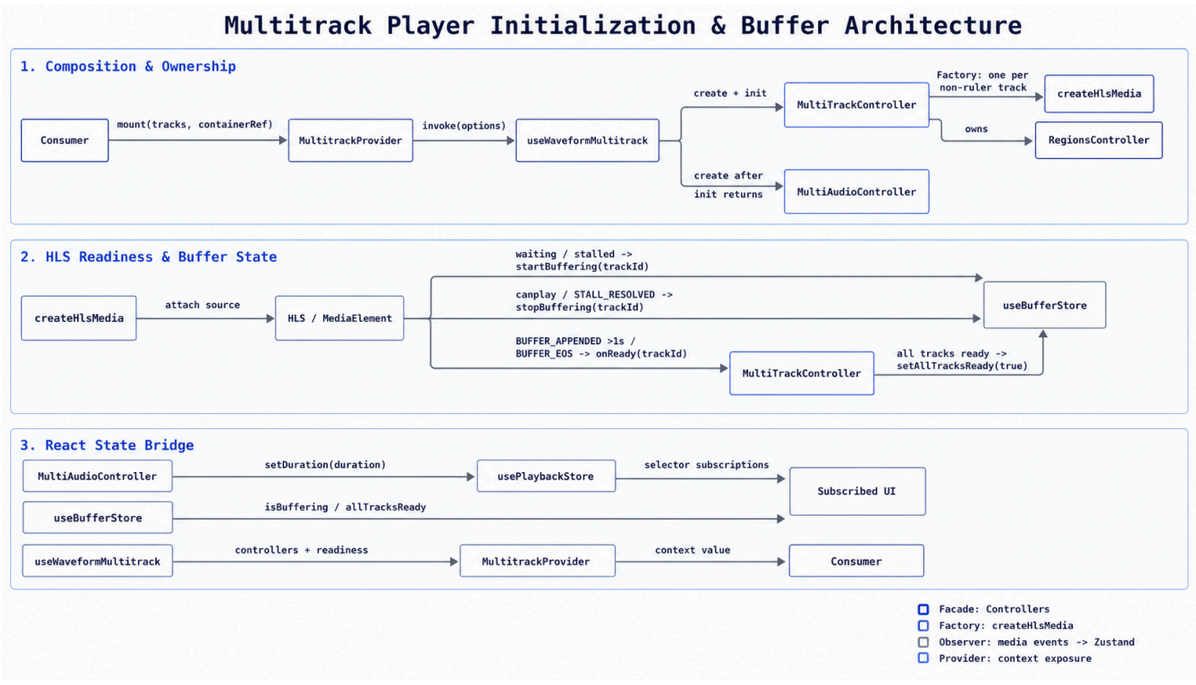
테스트 미디어가 준비되기 전, 예상 HLS format을 기준으로 Player를 선행 설계하고, 미디어 샘플이 준비된 후 구현한 Core Package에 실제 사용 시나리오를 검증하며 다음과 같은 문제들을 확인했습니다.

1. **장시간 media loading :** 2시간 이상의 media를 전체 file 단위로 처리하면 초기 재생 대기와 browser memory 사용량이 증가할 위험 존재
2. **Multitrack 상태 제어 복잡성** : Track마다 독립된 MediaElement와 buffer lifecycle을 가지면서도 동시 playback 제어와 개별 트랙의 volume 제어를 하나의 Player 객체에서 제어해야 하는 복잡성
3. **Player UI 결합도 문제** : Customer/Agent 등 사용자 권한에 따른 panel UI 조합, Region comment의 CRUD 기능이 playback state와 직접 결합
4. **Audio/Video 트랙의 동기화 오차 발생**: 시간이 긴 샘플 미디어를 끝까지 재생할수록 video와 audio의 duration 및 실제 종료 시점 차이가 점진적으로 증가

해결 과정

처음에는 HLS playback에 널리 사용되는 Shaka Player의 사용도 검토했지만, waveform과 Multitrack의 상태 동기화 및 동시 제어를 위해 별도의 orchestration layer가 필요함을 발견했습니다. 따라서 장시간 미디어 loading에는 HLS 사용을 유지하되, 서비스의 핵심 요구와 가까운 wavesurfer-multitrack 기술을 채택하고 내부 플레이어 구현을 Controller로 감싸 공통 패키지로 확장했습니다.





1. HLS 기반 Multitrack playback 구조 설계

- 장시간 media 전체 download 대신 HLS.js 기반 segment loading·buffer 관리 적용
- Track별 `BUFFER_APPENDED`·`BUFFER_EOS` event의 중앙 store 수집 및 `allTracksReady` 상태와 Player control gate 연결
- 첫 번째 audio track을 master로 지정하고 `MultiTrackController`,`MultiAudioController`에서 동시 playback 조율
- 기본 4개 HLS track과 runtime track 추가 지원 및 테스트 환경에서 최대 10개 track의 사용 시나리오 검증

2. Controller-Store-Hook-UI 기반 공통 Player package 구축

- Facade 기반 Controller 객체로 multitrack 제어를 캡슐화하고 Factory 기반 HLS MediaElement 생성 패턴 설계
- 어플리케이션 및 권한 별 track, comment panel 등 UI composition을 manifest로 제어
- Region comment의 CRUD와 playback의 lifecycle 분리 및 200개 이상의 Region 생성, 편집 시나리오 검증

3. Aaudio/Video 동기화 오차 원인 분석 및 source mediaConvert가 포함된 workflow 수정 기여

- 발생한 동기화 오차 원인으로 예상된 Player rendering 가설 검증을 위해 타임 스탬프와 buffer lifecycle, HLS fragment event logging
- Fragment `startPTS` 비교로 오차 원인을 Player state에서 media resource의 encoding configuration에 있음을 진단하여 수정 요청

4. AI Agent Orchestration 기반 Engineering workflow 구축

- Orchestrator 요구사항 분해, Generator A/B의 독립 구현, Critic과 Validator의 설계, 성능, E2E 검증 rule 구성
- Generator별 독립적인 worktree 할당 및 채택으로 Critic을 통과한 branch만 PR로 전달하는 리뷰 시스템 구축
- 애플리케이션 공용 Playwright storageState에 auth context를 보존하여 permission, order, player flow 검증 환경 구성
- critic에서 검증 실패 시 최대 3회의 retry feedback loop와 deadlock report를 적용하여 무한 반복 방지 조건 명시

성과

- Video와 Audio를 통합한 HLS Multitrack Player package 단독 설계, 구현 및 플랫폼 내 2개 어플리케이션에 해당 패키지 적용
- 기본 4개 track 옵션, 실시간 track 추가, 버전 기능 지원 및 최대 10개 HLS track과 200개 이상 comment 기능을 claude 기반 E2E 테스트로 검증
- Audio/Video 트랙의 동기화 오차 누적 원인을 resource encoding으로 규명하고 MediaConvert 수정 후 이후 4 경로 검증으로 해결 확인
- AI 에이전트를 활용한 코드 검증 및 자동 리뷰 파이프라인(Orchestrator-Critic)을 설계하여 팀 내 개발 생산성 및 코드 품질 상향 평준화 기여

Coretone - 사내 디자인 시스템

2025.03 - 2025.05

담당 역할 : Frontend Engineer, 1인 단독 개발

- 제품별 UI 일관성 확보와 협업 비용 절감을 위해 디자인 시스템 도입 제안 및 아키텍처 설계, 구체적인 기대 효과 제시로 유관 부서(디자이너, 개발 조직) 합의 도출
- Semantic token, component composition 등 디자인 시스템 foundation 구축, 모노레포 구조의 package architecture 설계
- Figma Variables export plugin과 React, Config, Lint rule, Agent Rule, CSS package, Storybook review 환경 구현
- Changeset, AWS CodeArtifact 기반 Release 구조 도입 및 이후 platform 운영, 유지보수

기술 스택 : React, Tailwind CSS, ShadcnUI, Storybook, Figma Plugin, AWS CodeArtifact

배경

Gaudio Lab은 사내 제품을 다다른 repository에서 개발하고 있었습니다. 때문에 같은 UI를 제품별로 복사하거나 개발자마다 다르게 구현하면서 매 프로젝트 마다 컴포넌트 구축에 시간이 불필요하게 투입되었고, 신규 디자인 적용 주에는 designer QA meeting에 주 2회 이상 총합 약 10시간이 소요되기도 했습니다. 이런 문제를 해결하기 위해 **커뮤니케이션 시간을 단축하면서 디자인 언어와 구현 품질을 일관된 version으로 관리할 공통 시스템**이 필요했습니다.

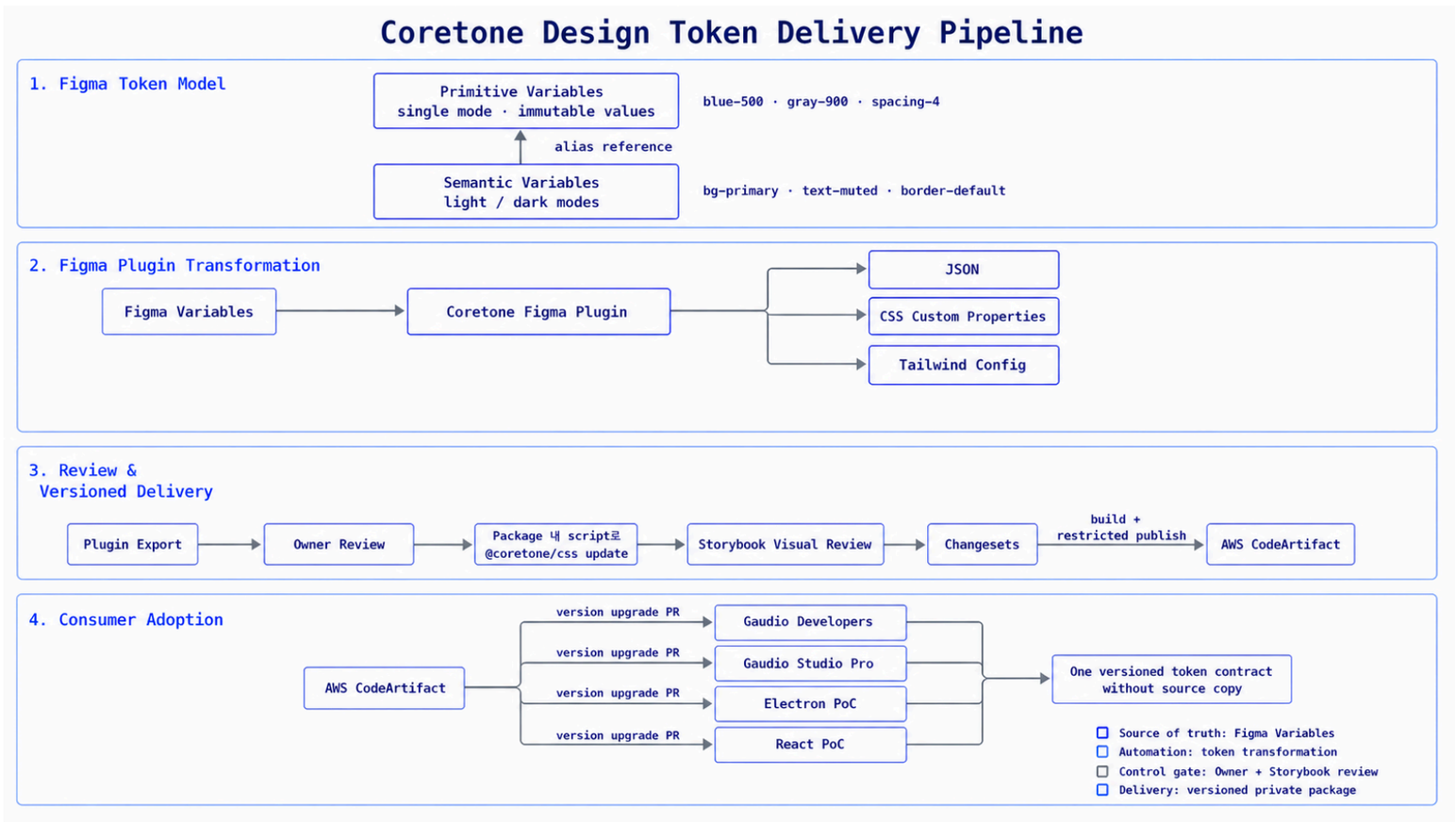
문제 인식

기존 Homepage 운영 이슈와 Studio의 신규 디자인 적용 과정을 검토하며 다음 문제를 확인했습니다.

- **UI component 중복과 불일치** : 동일 component의 소스코드를 제품 별 복사, 변형으로 반복적인 구현 수정과 designer QA 발생
- **Design-code token drift :** Figma의 color, typography, spacing, theme 값을 수동으로 맞추며 application별 token이 달라질 위험 증가
- **Private package 배포 경험 부재 :** 사내 resource를 보호하면서 여러 application에 공통으로 활용 가능한 패키지를 배포할 경험 부재

해결 과정

기존 개발 프로세스에서는 source code들을 신규 프로젝트에 복사하는 방식에 의해 변경 이력과 디자인 기준이 분산되는 한계가 있었습니다. 디자인 시스템 패키지는 초기 구축 비용이 들더라도 token과 component contract를 중앙에서 관리할 수 있어, 사내의 기존 기술 스택과 adoption cost를 기준으로 디자인 시스템 도입을 결정했습니다.



1. 사내 기술 스택에 맞춘 Component Architecture 설계

- Type Safety의 Vanilla Extract 도입을 고려하였으나 기존 product 호환성과 팀 학습 비용을 우선해 Tailwind CSS 선택
- HeadlessUI primitive로 컴포넌트의 action/visual rule 분리 및 `Slot`, `asChild` 기반 compound component 확장성 확보
- Code의 nested composition과 Figma의 Atomic hierarchy를 활용한 도구별 component 조합 최적화
- Figma의 semantic token contract를 공통 source of truth로 채택하여 디자인-UI개발 프로세스 간 token 일관성 통합

2. Figma Variables 기반 Token Pipeline 구축

- Semantic token → dark mode → responsive breakpoint → size variable 순서로 foundation 정의 및 Typography 기반 component 표준화 착수
- Figma Variables의 collection, mode, alias를 JSON와 CSS custom property 및 Tailwind config로 변환하는 Figma plugin 구현
- Immutable Primitive Token의 alias를 mode별로 참조하는 Semantic token alias 구조 설계
- Token Naming 표준화와 `px` → `rem` 변환 규칙, RGBA → HSLA 규칙, default/dark/custom theme selector 생성 자동화

3. 사내 Private Package Delivery 구축

- React component, CSS token, utility, TypeScript, ESLint, Tailwind config를 통합하여 Turborepo 기반 독립 package 구성
- Public npm의 source 노출이나 Verdaccio의 registry 운영 부담, GitHub Packages의 별도 권한 관리 등 사내 리스크 최소화 방안을 비교, 개발 비용 계산
- 기존 AWS 인프라와 권한 체계를 재사용하는 CodeArtifact 채택, 이후 Changeset versioning과 restricted publish 적용
- Pre-push 스크립트를 작성하여 package build와 publish, Storybook CI 배포 파이프라인 구축
- CodeArtifact 인증 만료 대응·신규 component publish 절차의 README 문서화 및 개발자·AI Agent 운영 경로 표준화

4. Storybook 기반 Component Review 및 사내 디자인-개발 통합 프로세스 확산

- Shadcn template 생성 → 디자인 구조 수정 → Storybook review → package publish → consumer update의 delivery flow 표준화
- 사내에서 디자인 시스템을 활용하여 리서치 부서, 프로덕션 부서 비개발자가 React, Electron demo 제작

성과

- 디자인 시스템 표준화 및 토큰 파이프라인 구축을 통해 불필요한 UI 수정 소통 리소스 절감. 디자인 리뷰 미팅 시간 **6.6→2.8시간(-57.6% 단축)**, 사내 타 부서 비개발자의 프로토타입 제작 공수 개선
- 재사용 가능한 컴포넌트 및 디자인 토큰 표준화를 통해 UI 관련 작업 효율성 향상. **신규 프로젝트 리드타임 28.4% 단축**, 기간 대비 디자인 관련 완료 티켓 수 49% 증가
- 디자인 시스템과 DesignOps를 구축하여 디자인-개발 협업 프로세스를 표준화

Gaudio Studio | AI 음원 · 영상 분리 SaaS

2024.04 - 2025.05

담당 역할 : Frontend Engineer, 2인 팀 개발

- App Router 전환을 제안하여 route, SSR, i18n, Suspense 등 Frontend Core Architecture 설계
- 업로드 Funnel 재설계, 결제 기능 구현으로 사용자 경험 개선
- 미디어 업로드의 duration, channel, format 검증 구현
- 이외 반응형 디자인 구현, 웹 성능 최적화, 서비스 운영, 유지 보수

기술 스택 : TypeScript, Next.js App Router, Zustand, TanStack Query, next-intl, Paddle, Sentry

배경

Gaudio Studio는 사용자가 업로드한 음원 및 영상에서 음성과 악기를 분리하고 결과를 재생, 다운로드하는 B2C SaaS였습니다. 따라서 업로드 및 결제 단계에서 의도치않은 페이지 전환이 발생하거나 파일 및 결제 상태가 유실되면 사용자 요청 실패와 매출 손실로 이어질 수 있어, **URL과 UI 상태의 정합성** 및 업로드와 결제 **데이터 일관성**을 보장하는 것이 핵심 과제였습니다.

문제 인식

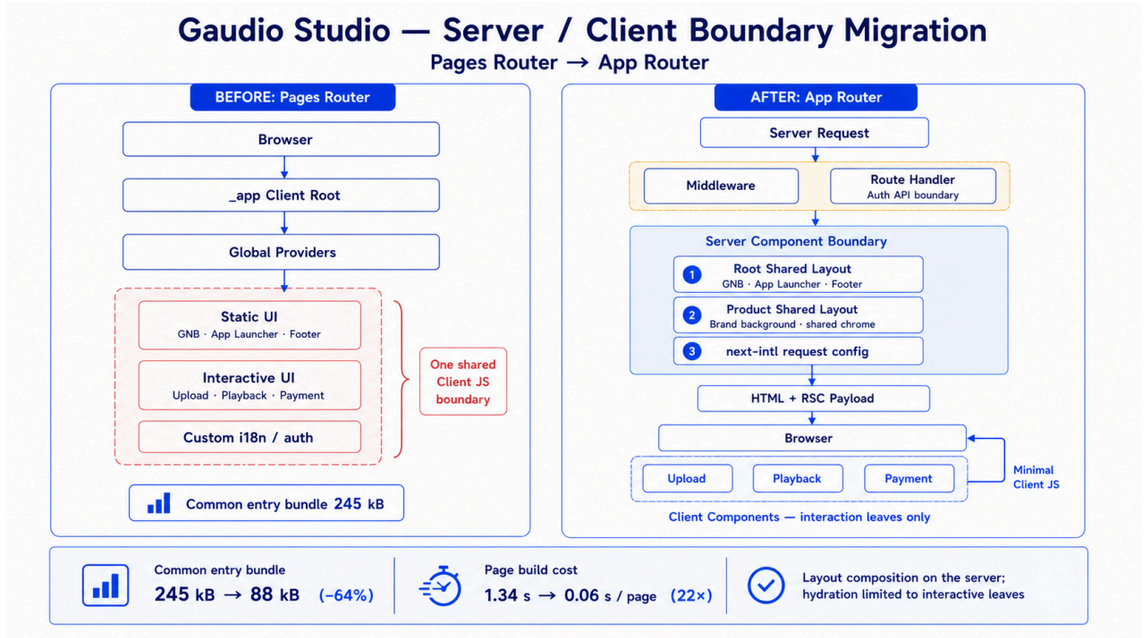
입사 후 온보딩 과정에서 기존 Pages Router application을 분석하며 다음 문제를 확인했습니다.

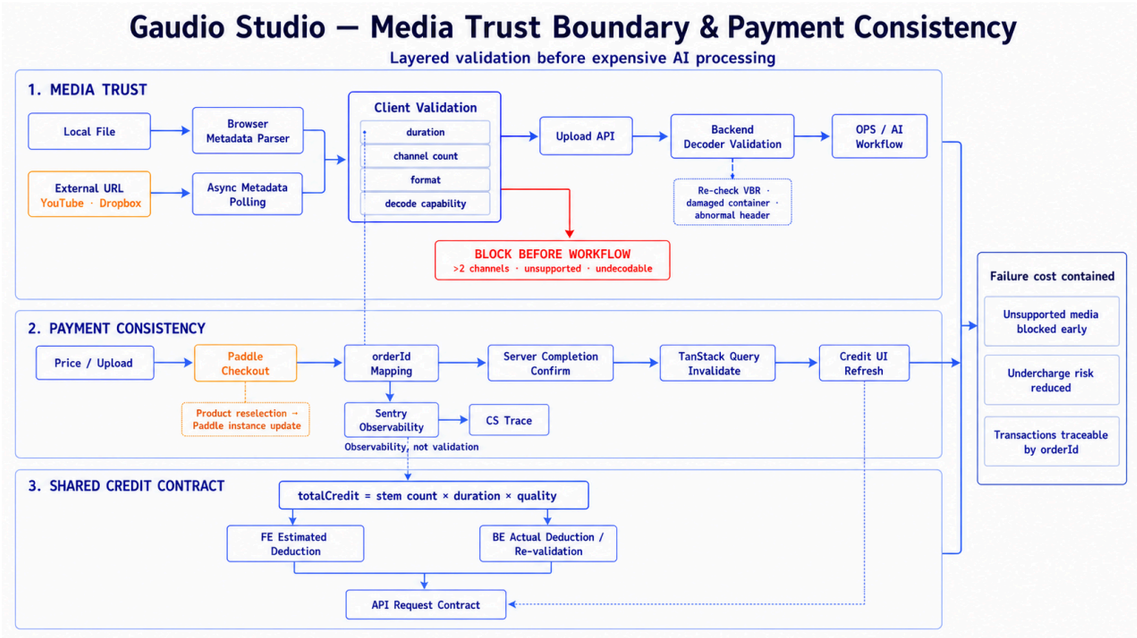
1. **페이지 번들링 문제 :** 기존 Pages Router의 _app 구조 하에서 정적/동적 컴포넌트가 강하게 결합되어 초기 번들(245kB)이 비대해지는 병목 진단
2. **업로드 상태 유실 가능성 :** 기존 업로드 Modal 클라이언트 컴포넌트에 쌓인 전역 상태 스토어에서 Suspense infinite loop가 발생하여 업로드 파일, 옵션, 크레딧 상태 유실과 사용자 이탈 유도
3. **크레딧 트랜잭션 정합성 문제 :** 네트워크 상황에 따라 외부 SDK와의 트랜잭션과 결제는 완료됐지만 credit이 갱신되지 않거나, 상품 재선택 후 이전 callback이 남는 사례 발생
4. **사용자 미디어 파일 신뢰성 문제 :** 조작된 header의 파일이나 비정상 metadata로 인해 실제 미디어 처리량보다 크레딧이 적게 차감되거나 지원하지 않는 다채널 audio가 AI workflow로 전달되는 사례 발생

이 문제들을 개별 UI 오류로만 수정하면 유사한 클라이언트 라이프사이클과 시스템 - 미디어 메타데이터의 신뢰 경계에서 같은 문제가 반복될 수 있다고 판단했습니다. 따라서 rendering architecture, Funnel lifecycle, payment transaction, media validation을 하나의 구조적인 업로드, 결제 사용자 여정으로 보고 **점진적으로 Pages Router를 App Router로 옮겨가며** 해결 방향을 검토했습니다.

해결 과정

위 문제들을 해결하기 위해 처음에는 App router로의 전체 전환을 고려했으나, 전면 전환은 실 사용 서비스를 동시에 바꾸는 위험이 컸습니다. 따라서 전체 어플리케이션 SSR 도입 및 인터랙션이 불필요한 레이아웃 영역을 Server Component로 격리하는 **Server-Client Boundary 설계**로 초기 렌더링 구조부터 개선하였고, 이후 **Parallel Route를 활용한 업로드 상태 관리와 제어** 전략을 도입하였습니다.





1. App Router 기반 Server, Client boundary 재설계

- Root shared layout에 GNB, App Launcher, Footer를 배치하여 정적 UI는 Server Component로 구성하고, upload, playback, payment 처럼 사용자 인터랙션과 browser API가 필요한 leaf만 Client Component로 분리
- route segment별 `layout`, `loading`, `not-found`와 provider 경계를 구성하고 auth API를 Route Handler로 이동
- 기존의 클라이언트 감지 기반의 i18n cookie 동기화와 request별 server instance 분기 로직을 next-intl의 root provider-typed navigation 구조로 전환

2. Parallel Route 기반 Upload, Payment Funnel UX 개선

- `[product]/upload` 아래에 `@upload`, `@preview`, `@options`, `@request` slot을 구성
- `upload` 디렉토리의 persistent Client layout 아래에 step, body, option Zustand store를 유지
- store 구독으로 현재 단계에 맞는 slot을 선택하고 validation 및 Funnel 전환, 초기화 시점을 제어
- route를 벗어나면 Funnel state를 초기화해 이전 작업 상태가 다음 upload 상태에 전이되지 않도록 방어
- credit이 부족하면 request 단계로 진행시키지 않고 Funnel 내부 pricing slot을 노출해, 충전 완료 후 작업 흐름으로 복귀하도록 연결

3. Payment transaction 관측 기능 추가 및 Paddle SDK lifecycle 보강

- Paddle transaction과 내부 주문을 연결하는 `orderId`를 Sentry에 기록해, 결제 완료 후 크레딧이 반영되지 않을 때 로그로 거래를 식별할 수 있도록 개선
- checkout 완료 후 크레딧 관련 TanStack Query cache를 invalidate해 크레딧이 노출되는 모든 UI의 최신 상태를 보장
- 사용자가 상품을 다시 선택할 때 이전 callback이 남는 문제를 Paddle instance update 방식으로 수정
- checkout이 준비되기 전에는 Loader를 노출하고, locale mapping과 fallback을 적용해 국가별 checkout UI의 안정성 개선

4. 업로드 Flow에서 크레딧과 파일의 다층 validation 구현

- Local file은 browser metadata와 `music-metadata-browser`를 사용해 duration·channel·decode 가능 여부를 검사
- 2채널을 초과하는 audio는 AI workflow에 진입하기 전에 `fileChannelError`로 차단
- YouTube·외부 URL은 비동기 metadata 결과를 polling한 뒤 성공 시점에 duration·extension·최대·최소 길이를 재검증
- 화면의 예상 차감량과 API request가 같은 `stem × duration × quality` 계산을 사용하도록 공통화
- VBR·손상 container·비정상 header처럼 parser별 duration이 달라질 수 있는 사례를 Backend와 공유하여 AI model workflow의 실제 처리 경계에서 검증

성과

- 기존 프로젝트의 빌드 병목 지점을 진단하고, Next.js App Router 마이그레이션을 주도하여 개발 빌드 시간을 페이지당 빌드 비용 **1.34s → 0.06s/page**로 22배 개선, app 공통 **Bundle Size 245kB → 88kB** (64% 감소), 사용자 평균 **체류 시간 5분 16초 → 6분 20초** (20% 증가)
- 유료화 v2의 프론트엔드 전체 아키텍처를 전담하고 파일 업로드 최적화, Paddle 결제 통합 연동, UX 개편을 구현하여, **누적 결제 2.5만 건 및 크레딧 170만 건** 이상의 대규모 실사용 트래픽 환경에서 결제 이탈 및 상태 유실 없는 안정적인 결제 라이프사이클 운영
- 이외에도 마케팅팀과의 협업으로 글로벌 다국어화(i18n) 플랫폼 및 글로벌 SEO 최적화 아키텍처를 구축하여, 전체 트래픽 중 **해외 유입 비중 47.5%에서 71.2%**로의 성장 및 글로벌 유저 인프라 안정화에 기여