오준석 Oh Junseok

Full-Stack Engineer · AI / LLM Application

2001.09.17 010-7593-4447
ojspp000@naver.com 가톨릭대학교 컴퓨터정보공학과 졸업
서울특별시 강서구 화곡동 1111, 1719호
portfolio-nextjs-puce-pi.vercel.app
github.com/ojspp41

ABOUT ME 문제를 직접 정의하고 끝까지 푸는 풀스택 개발자

기획자·디자이너 없는 환경에서 사용자 문제를 직접 정의하고 끝까지 푸는 풀스택 개발자입니다. 한솔그룹 13개 계열사·1만 명 사내 AI(LLM) 서비스 'AI Atlas'에서 프론트를 오너로 구현하고, 미리보기 엔드포인트·다운로드/미리보기 통로 분리와 미터링 조회 API·사용 한도 제어 API까지 서버 계약을 직접 설계·구현했습니다. 그 프론트가 붙는 Go 기반 중앙 API Gateway(gRPC·Kafka·Redis·MinIO)는 설계 의도를 이해하고 연동했습니다. 화면부터 서버 계약까지 직접 설계하며, 모든 결정을 극단 폭주 시 리렌더 99.6%↓·도커 이미지 50%↓·회복률 60→97.5%로 증명합니다.

1만 명
한솔그룹 사내 LLM 서비스

리렌더 99.6%↓
응답 폭주 시, Profiler 실측

장관상
과학기술정보통신부 · Githru

2,000명
상용 서비스 (매출 800만원)

TECH STACK

AI / LLM	LLM Streaming · Generative UI Pipeline · RAG Integration · Markdown Parser · MCP
Frontend	React 19 · Next.js 15 · TypeScript (strict) · Tailwind v4
State / RT	Zustand · Recoil · TanStack Query · WebSocket / STOMP · EventBus
Infra / Test	Docker Multi-stage · Kubernetes · CI/CD · Jest (TDD) · Playwright
Backend	Go 1.24 · MySQL · MongoDB · Redis · Gin · gRPC · Protobuf · Kafka · MinIO · K8s

AI 활용 경험

- 생성형 UI 처리 흐름 (Pipeline) 설계 — AI 응답을 안전한 UI로 변환
- 검색 증강 (RAG) · Chunking · 임베딩 과정 시각화
- RBAC 권한 경계 설계 — can() 추상화로 미확정 역할 스펙과 FE 격리 (BE permission.go 설계 이해·연동)
- 스트리밍 마크다운 파서 직접 설계 (Progressive Markdown Parser)
- VOC 에이전트 직접 개발 — 생성형 UI 폼으로 Jira 티켓 자동 생성 (가이드 문서 기반)
- MES 데이터 분석 에이전트 — 자연어를 SQL로 변환된 차트 시각화
- playwright-mcp + Confluence MCP 결합 가이드북 자동화 — Claude Code로 화면 영역 캡처 및 문서 자동 생성
- 사용량·비용 미터링 대시보드 — 화면 요구를 백엔드 집계로 정의 (Kafka·SCD-2 연동, FE 훅·ExcelJS 직접 구현)
- 파일 인앱 미리보기 풀스택 — 미리보기 엔드포인트·통로 분리 직접 구현 + 악성코드 차단 3겹(서버 1 + 화면 2)

강점 & 협업 역량

리더십
팀장 역임 3회 (COMAtching · Favus · 사내 CoP)

오너십
1만 명 서비스 런칭 리더

임팩트 정량화
리렌더 99.6%↓ · 도커 이미지 50%↓ · 실패율 90%↓

지속적 기여
오픈소스 컨트리뷰션 + vs code extension 개발

경력

한솔 PNS IT

AI 개발팀 · On-Premise

2025.10 — 재직 중
Full-Stack (단독)

풀스택 개발 · AI Atlas 사내 LLM 서비스 (13개 계열사 · 1만 명) 프론트 오너 구현 + Go Gateway 일부 개발(백오피스, 미터링)

> 5년차 시니어 사수가 이탈한 상황에서 입사 2개월 만에 기획·개발·테스트·운영 전 과정을 주도해 런칭했습니다. 기획자·디자이너가 없는 환경에서 AI 도메인(파싱·청킹·임베딩) 프로세스를 시각화하고 Smart Tooltip 모듈을 npm에 배포했습니다. 프론트를 오너로 직접 구현하면서, 그 프론트가 붙는 Go 기반 중앙 API Gateway(gRPC·Kafka·Redis·MinIO)의 설계 의도까지 이해하고 개발하였습니다.

01

사용량·비용 미터링 대시보드 (백오피스)

AI 호출 이벤트를 백오피스 대시보드로 — 화면이 요구하는 데이터 모양을 먼저 정의하고 백엔드 집계를 그 모양으로 역설계했습니다.

FE 대시보드·조회 훅·엑셀 내보내기, BE 미터링 조회 API·사용 한도 제어 직접 구현 / BE Go 파이프라인(Kafka 인입·일배치 사전집계·SCD-2 정산)은 설계 의도 이해·연동.

조회 지연 상수화 · 복합키 upsert 멱등 재실행 · SCD-2로 정산 시점 재현

02

RBAC 권한 경계 — FE/BE 경계 분리

무엇에 권한을 묻는지는 FE가 정하고, 어떻게 강제하는지는 BE가 맡습니다. 리소스 10 × 액션 12(부분집합) · 역할 6, shares.shared_with[]가 SSOT.

can() 추상화로 미확정 역할 스펙과 FE를 격리 + 3-상태 렌더링으로 플리커 제거 (FE 직접 구현) / BE permission.go(멀티 소스 SSOT·cascade)는 설계 이해·연동, ownership-first 소비가 FE·BE 접점.

정보 누출 차단 · 플리커 제거 · 역할 늘어도 FE 무변경

03

파일 인앱 미리보기 — 풀스택 대표작

원본은 그대로 두고 DRM 처리 사본만 보여줍니다. FE 렌더 3층(판단→가공→표현) 직접 구현 + BE 미리보기 엔드포인트·통로 분리 직접 구현.

다운로드는 원본 / 미리보기는 DRM 처리 사본만 서빙 — 다운로드·미리보기 통로를 서버에서 분리 (Breaking Change 0). 악성코드 차단 3겹(서버 1겹 + 화면 2겹)으로 방어. 6종 형식 단일 엔드포인트.

Breaking Change 0 · 3겹 방어 · object URL 누수 0 · 6종 형식

경력 WORK EXPERIENCE — CONTINUED

한솔 PNS IT · AI 개발팀 — Full-Stack · 2025.10 ~ 재직 중

04

Generative UI 파이프라인 — 깨진 JSON 복구

모델 출력을 믿지 않는다는 전제로, 깨진 JSON을 받아도 위젯이 뜨도록 6단계 정리 + 2겹 파싱으로 복구했습니다. 회복률을 40개 fixture 벤치로 측정해 규칙을 바꿔도 회귀를 즉시 잡습니다. (FE 직접 구현)

회복률 60% → 97.5% · 위젯 장애 95%↓ · 위젯 하나 깨져도 채팅 유지

05

실시간 통신 6단계 방어 (WebSocket)

AI 응답이 이어질 때 생기는 응답 섞임·백그라운드 탭 끊김·화면 버벅임을 서버 변경 없이 클라이언트에서 막았습니다. 실시간·AI 도메인 폭을 넓혔습니다.

끊김 감지(Heartbeat) → 지수 백오프(Backoff+Jitter) → 화면에 보일 때만 연결 유지 + 응답이 몰릴 때 화면 갱신을 묶어서 처리. (FE 직접 구현 — 클라이언트 방어)

극단 폭주 시 리렌더 99.6%↓ · 실사용 패턴 50%↓ · 정상 응답 추가 비용 0%

06

Dockerfile 고도화 — 인프라 / 운영 보강

운영팀이 두 번 반려한 이미지를 멀티스테이지 빌드 + Next.js standalone + Non-root로 슬림화했습니다. 이후 dev 의존성이 부활해 1.56GB가 되살아난 것을 레이어별 측정으로 찾아내 제거했습니다.

Non-root × APM(WhaTap) 로그 권한 충돌(EACCES)은 이미지 안에서 해소하고, 부팅 회귀는 스모크 테스트로 머지 전에 막았습니다. (Infra 직접 구현)

이미지 3.63GB → 1.82GB 50%↓ · cold pull 24s → 13s

수상 내역

1. 2025 오픈소스 컨트리뷰션 아카데미 우수상 — 과학기술정보통신부 장관상 · Githru
2. 2025 오픈소스 개발자 대회 우수작 — 직장인 부문 · Favus (Go CLI · WebSocket)
3. GGUM 교내 해커톤 우수상 — 도서관 좌석 / 빈 강의실 시각화 (React, TS)
4. 교내 ICPC 우수상 — 알고리즘 대회

자격 · 어학 · 활동

- 어학: 오픽 IH (2025.08)
- 자격증: 정보처리기사 (2025.09) · 컴퓨터활용능력 2급 (2023.12) · 운전면허 1종 보통
- 기타: 국제근로팀 근로 (2023.03~2025.06) — 외국인 재학생 기숙사 영어 응대 / Excel 업무

학력

가톨릭대학교
컴퓨터 정보공학과 졸업
2020.03 — 2026.02 · 학점 4.05 / 4.5

광명북고등학교 (광명)
2017.03 — 2020.02

동아리 활동

COMA (임원)
교내 IT 개발 중앙동아리 · 2023.03 ~ 2026.03

COMATCHING (팀장)
IT 창업 동아리 · 2023.09 ~ 2025.08

GDG CUK (임원진)
구글 개발자 그룹 · 2024.08 ~ 2025.03

UMC
교외 Node.js 파트 동아리 · 2023.08 ~ 2023.12

인턴 경력

PTKOREA (삼성 제일기획 자회사, 외국계)

2025.06 — 2025.09

대기업 글로벌 사이트 QA 자동화 인턴 — 풀스택

Full-Stack · Internship

TypeScript | Django | React | Zustand | Python · AEM · Jira

- DNT(Do Not Translate) 검증 자동화 — 'SAMSUNG/AI' 등 브랜드명 번역 오류 자동 검출, 96개국 일괄 검증 시스템.
- 초기 로딩 65% 단축 (3.7s → 1.3s) — 데이터 요청 방식 개선(전체 → 페이지 단위).
- 월 10시간+ 수작업 절감 — DNT 검증 + Excel 보고서 자동화.
- 폐쇄망 환경 Electron 기반 .exe 사내 앱 개발 + 96개국 AEM 컴포넌트 유지보수.

오픈소스 & 사이드 프로젝트

Githru (VSCode Extension) — 컨트리뷰션

2025.06 ~

2025 오픈소스 컨트리뷰션 아카데미 우수상 · 과학기술정보통신부 장관상

TypeScript | Tailwind | React · Zustand · D3.js

- TemporalFilter 컴포넌트 렌더링 성능 최적화 PR (#812) — 라인 차트 데이터 처리 로직 최적화로 렌더링 성능 18.9% 개선, 변동성 93% 감소.
- D3.js 차트 컴포넌트의 불필요한 재계산 방지로 대용량 커밋 데이터 처리 성능 향상.

Favus — S3 멀티파트 업로드 도구 (팀장)

2025.06 — 2025.09

2025 오픈소스 개발자 대회 우수작 · 직장인 부문

Go (CLI) | React · Next.js | Python · WebSocket

- Go 기반 병렬 청킹 + 상태 저장 메커니즘 — 네트워크 중단 시 자동 재개, 대용량 파일 전송 실패율 90% 감소.
- WebSocket 3계층 모니터링 (CLI ↔ Python Server → React UI) — 업로드 진행률·파트별 상태 실시간 시각화.

COMAtching v3 ~ v4 (팀장)

2023.06 — 2025.05

서비스 상용화 · 2년간 누적 사용자 2,000명 · 매출 800만원

React · Recoil | SockJS/STOMP | Node.js · MySQL

- 팀장 · 기획·개발·운영 전 과정 책임(프론트 주도) — MySQL 스키마·매칭 조회(user–match 조인 + status 인덱스) 설계, 백엔드 매칭 로직은 일부 구현, Docker/Jenkins CI/CD.
- Toss Payments SDK + Idempotency-Key 기반 중복 결제 방지 시스템 구축.

부천 FC | AI 응원 매칭 — 기업 협업 프로젝트

2024.09 — 2024.10

부천 FC 경기장 실서비스 배포 · 700명 참여

React | Recoil | jsQR | Chart.js

- 화면 기획 및 프론트엔드 단독 개발 — 입장 QR 인증 → 성향 분석 → 매칭까지 전체 사용자 플로우 설계.
- AI 기반 성향 분석 시스템 — 6가지 응원 유형을 Chart.js 레이더 차트로 시각화.

심층분석 01

AI 사용량·비용 미터링 & 과금 백오피스

백오피스 어드민

백오피스 어드민 · REST API · MongoDB 집계 · 사용 한도 제어(쿼터/레이트리밋) · Redis 분산락 · 가격 이력 버전 관리 · Kafka · Go · React Query · ExcelJS

FE 직접 구현

대시보드 · 목록 조회 공통 훅 · 엑셀 여러 시트 내보내기(ExcelJS) · 필터 6종을 주소(URL)에 저장해 새로고침·공유해도 상태 유지

BE 담당 범위

직접 구현 — 미터링 조회 API(필터 6종·페이지 나눔·엑셀 내보내기) · 사용 한도 제어 API(요금제별 한도, 실시간 카운터로 초과 호출 차단·경고).
설계 이해·연동 — Go 데이터 파이프라인(Kafka 수집·일배치 사전집계·멱등·가격/환율 이력 버전).

핵심 성과

빠른 조회

매 조회 즉석집계 → 일배치 사전집계로 설계 전환. 하루 한 번 만든 요약본만 읽게 해, 조회 1,970ms → 2.6ms(약 750배)·원본 3,000,000건 → 요약 1,440건으로 상수화.

재실행 안전

집계를 두 번 돌려도 숫자가 중복으로 더해지지 않게 멱등하게 구현. 2회 실행에도 요약 1,440건 · 비용 합계 완전 동일

비용 폭주 방지

사용량이 분당 100회 한도를 넘으면, 호출이 실제로 나가기 전 게이트웨이에서 Redis 카운터가 즉시 차단(429)하고 경고.

정산 시점 재현

가격·환율이 바뀌어도 예전 값을 지우지 않고 버전으로 쌓게 구현. 몇 달 전 청구도 그때 가격 그대로 다시 계산·검증 가능.

S 상황 (S)

AI 서비스가 커지면서 '누가 얼마 쓰고 얼마 청구되는지'를 관리자가 보고 통제할 백오피스가 필요했다. 이건 기능 세 개를 따로 붙이는 문제가 아니라, 하나의 파이프라인으로 이어져야 하는 문제였다.

호출 기록은 Kafka로 들어오는데, Kafka는 재전송 때문에 같은 기록이 두 번 이상 들어올 수 있다(최소 한 번 전달). 그래서 집계를 두 번 돌려도 숫자가 중복으로 더해지지 않게 멱등하게 만들었다. 또 매번 수백만 건 원본을 즉석 계산하면 느리고 비싸서, 하루 한 번 요약본을 미리 만들어 두고 화면은 그것만 읽게 했다. 마지막으로 가격·환율은 시간에 따라 바뀌는데, 예전 값을 지우면 과거 청구를 다시 만들 수 없어(정산·감사 불가) 값을 버전으로 쌓아 과거 청구를 그 시점 기준으로 재현했다. 한도 없이 호출되면 비용이 통제 불능이 되므로, 사용량이 요금제 한도를 넘으면 호출 길목에서 바로 막게 했다.

T 과제 (T)

1) 중복이 섞여 쏟아지는 호출 기록을, 두 번 처리해도 안전하게(멱등) 집계하기
2) 대시보드가 추가 가공 없이 바로 그릴 수 있는 요약 데이터로 사전집계
3) 가격·환율이 바뀌어도 과거 청구를 그 시점 기준으로 다시 계산·검증
4) 사용량이 요금제 한도를 넘으면 호출 길목에서 실시간으로 차단·경고

심층분석 01

AI 사용량·비용 미터링 & 과금 백오피스

백오피스 어드민

백오피스 어드민 · REST API · MongoDB 집계 · 사용 한도 제어(쿼터/레이트리밋) · Redis 분산락 · 가격 이력 버전 관리 · Kafka · Go · React Query · ExcelJS

> 사용자가 [내보내기]를 누르면 → 내가 만든 미터링 조회 API가 필터 6종·페이지 처리로 데이터를 모으고 → ExcelJS로 요약 시트 + 상세 시트를 만들어 → 파일로 내려준다. 이 내보내기 흐름(요청 → API → 가공 → 파일)은 요청부터 파일까지 직접 구현했다.

01	02	03	04
FE	BE	BE	FE
내보내기 요청	**미터링 조회 API**	**ExcelJS 생성**	**파일 다운로드**
사용자 클릭	필터 6종·페이지 처리	요약 + 상세 시트	완성 리포트 제공

A 행동 (A) · 구현

BE

- ✓ 대용량 엑셀 내보내기를 브라우저에서 통째로 만드는 방식 → 서버 스트리밍으로 전환. ExcelJS 스트리밍 라이터로 행을 받는 즉시 흘려보내고, 데이터 소스(게이트웨이)도 페이지 단위로 조회해 메모리를 행 수와 무관하게 평탄하게 유지.
- ✓ 필터 6종을 하나의 조회 쿼리로 조합하고, 결과를 요약 시트 + 상세 시트 여러 장으로 구성해 파일 생성.
- ✓ 생성한 파일을 응답 스트림으로 바로 내려 브라우저가 즉시 다운로드.

FE

- ✓ 필터 6종을 주소(URL)에 저장해, 새로고침·링크 공유해도 같은 화면이 그대로 뜸.
- ✓ 목록 조회를 공통 훅으로 묶어 여러 화면이 로딩·에러·페이지 처리를 재사용.
- ✓ [내보내기] 요청부터 파일 다운로드까지의 상태(진행/완료/실패)를 화면에서 처리.

R 결과 (R)

성능

측정 결과 · 재현 데이터 300만 건 · 실제 MongoDB · 실제 집계 파이프라인 기준

조회 속도 p95	매 조회 즉석집계 → 일배치 사전집계로 전환 · 1,970ms→2.6ms (약 750배) — 원본 즉석 계산 없이 요약본만 읽음
조회 대상	원본 3,000,000건 → 요약 1,440건 (약 2,000분의 1)
멱등	집계 2회 실행 후 요약 1,440건·비용 합계 불변 — 집계키 유니크 인덱스 위 upsert
한도 차단	한도 초과 호출은 게이트웨이 Redis 카운터(분당 100회)가 즉시 차단(429)
엑셀 메모리	20만 행 1,871MB(브라우저 통째 생성) → 8.2MB(서버 스트리밍, 약 230배↓) · 50만 행도 22.4MB로 평탄

업무 개선

· 정산 리포트를 엑셀에 손으로 채우던 작업을 자동 생성으로 대체.

· 관리자가 사용량·비용을 직접 조회·내보내기 할 수 있게 되어, 반복되던 데이터 추출 요청(VOC) 해소.

권한 접근 제어

> *이 제품은 — AI 에이전트로 대화를 만들고 그 대화를 남에게 공유할 수 있다. 공유받은 사람은 대화는 보지만 에이전트(AI 설정)의 주인은 아니다.*

문제

서버 · 실시간 연결
대화 응답을 실시간으로 멈추거나 다른 세션으로 바꾸는 통로가, 로그인 여부만 확인하고 그 세션이 그 사람 것인지는 안 봤다. → 권한 없는 사람이 남의 대화 응답을 중간에 멈추거나 가로챌 수 있었다.

화면 · 사이드바
에이전트 정보 사이드바를 열지 말지를 '대화를 수정할 수 있나'로 판단했다. 보여주는 건 에이전트인데 묻는 건 대화라 기준이 어긋나, 권한 없는 정보가 새거나 봐도 되는데 가려졌다.

진단

두 곳 다 "어느 것에 권한을 걸지"가 실제 보여주는 것과 어긋나 있었다. 그리고 고치는 순서 규칙이 두 곳에서 같아야 했다 — 주인이면 즉시 통과, 아니면 권한 확인.

행동 — 설계의 무게중심

① 실시간 통로에 '권한 확인'을 넣었다 (핵심·서버)
로그인만 보던 두 진입점에 '이 세션의 주인인가 → 아니면 다룰 권한이 있나'를 붙였다. 로그인 확인과 권한 확인을 분리해 우회 경로를 닫았다.

② 화면의 기준을 실제 보여주는 것과 맞췄다
'대화 수정' → '에이전트 읽기'. 대화는 가졌어도 에이전트 읽기 권한이 없으면 사이드바를 안 띄운다.

③ 권한부터 확인하고, 통과했을 때만 데이터를 부른다
권한 없는 데이터는 요청 자체를 안 한다.

④ '확인 중'을 따로 뒀다
상태를 셋으로: 아직 확인 안 함 / 없음 / 있음. '있음'이 확실할 때만 화면을 연다. 확인하는 찰나엔 닫혀 있어 깜빡임이 원천부터 없다.

⑤ 빠르게 옮길 때를 대비했다
에이전트를 연달아 바꾸면 먼저 누른 것의 뒤늦은 응답이 최신 화면을 덮지 않게 이전 응답을 버린다.

권한 접근 제어

결과

- **우회 차단**
  권한 없는 사람이 남의 대화 스트림을 제어하던 경로 2개 제거 (로그인 확인 ≠ 권한 확인, 분리)
- **정보 누출 차단**
  읽기 권한 있는 사람에게만 사이드바 노출, 권한 없거나 없는 페이지는 팝업 없이 그냥 미표시
- **깜빡임 제거**
  확인 전 닫힘 보장으로 초기 깜빡임 소멸
- **역할 확장에 안 흔들림**
  역할이 6개에서 더 늘어도 화면은 그대로 (서버가 준 '허용된 일 목록'만 소비)

현재의 한계

- 권한이 바뀌면 새로고침해야 반영됨 — 실시간 아님
- '왜 안 보이지?' 디버깅이 어려움 — 판단 과정이 코드 안에 묻혀 있음
- 같은 권한을 매번 새로 물어봄 — 아직 호출이 적어 문제는 아님
- 팀 단위 등 나머지 역할 권한은 정의 전

다시 한다면

- 이미 깔려 있는 실시간 통신으로 권한 변경 즉시 반영
- '주인인지 / 허용된 일 / 최종 판정'을 보여주는 개발용 확인 패널
- 호출이 늘면 자원별로 한 번 물어본 권한을 재사용
- 권한 단위로 묻는 구조라 새 역할이 생겨도 그대로 흡수 (응답 형태만 통합 검증)

백엔드

화면이 잘 가려도 요청을 직접 쏘면 그만이라, 최종 차단은 서버가 한다. 서버도 화면과 같은 순서로 판단한다 — 주인이면 바로 통과, 아니면 '누구에게 공유했는지' 목록을 확인. 주인일 땐 공유 목록을 아예 안 들여다본다.

기준 통일
권한은 '공유 목록' 하나로만 판단

조직 격리
다른 회사 간 공유는 저장되기 전에 차단

정합성
권한 변경은 전부 반영되거나 전부 취소 (중간 상태 없음)

연쇄 취소
위에서 권한을 거두면 재공유 사슬을 따라 아래까지 함께 취소

세 곳(실시간 연결·화면·데이터)은 같은 권한을 보지 않는다. 화면은 '에이전트를 읽을 수 있나', 실시간 연결은 '이 세션을 다룰 수 있나' — 각자 다른 걸 본다. 같은 건 대상이 아니라 '주인이면 먼저 통과, 아니면 확인'이라는 순서다. 이 순서가 세 곳에 똑같이 흘러 가장 흔한 '내 것에 접근'은 어디서든 추가 확인 없이 끝난다. 권한을 강제하는 건 서버지만, 어느 것에 기준을 걸지는 화면을 만드는 프론트의 판단이었다.

실제 구현 근거 — 실시간 진입점 2곳에 소유자-우선 권한 확인 추가 · 화면 게이트를 '에이전트 읽기'로 단순화(공유/수정 게이트 제거) · 3-상태로 깜빡임 차단 · 조직 간 공유는 저장 전 차단

심층분석 03

파일 인앱 미리보기 풀스택 · 보안

Go(Gin) · MinIO · DRM · Next.js 15 · React 19 · TypeScript · DOMPurify

사내 보안 파일을 원본 노출·악성코드 실행 없이 인앱으로 미리보는 파이프라인. 서버가 DRM 처리 사본만 격리 정책과 함께 내려준다.

BE 통로 분리 & 서버 강제 격리

서버가 격리 정책을 강제

- 다운로드/미리보기 엔드포인트를 대칭 분리 — 다운로드는 원본을 첨부(attachment)로, 미리보기는 DRM 사본만 인라인(inline)으로 서빙
- 미리보기 경로는 원본 저장소를 참조조차 하지 않도록 설계 → "원본은 미리보기로 새지 않는다"를 테스트로 고정
- 파일을 통째로 메모리에 올리지 않고 스트리밍 전달, 접근 시 다운로드 권한 재검증 + 경로 우회(path-traversal) 차단
- 능동형 콘텐츠(HTML·SVG·XML)는 서버가 격리 정책(CSP sandbox + MIME 스니핑 차단)을 강제 → 실질 신뢰 경계

FE 단일 파이프라인 & 자원 정리

6종 · 하나의 파이프라인

- 형식 판별(확장자→MIME 폴백) 후 6종을 하나의 렌더링 파이프라인으로 처리, 미지원 형식은 자동 다운로드 안내
- 미리보기용 임시 데이터·진행 중 요청을 모든 종료 경로에서 정리 (누수 0)
- 능동형 콘텐츠는 살균(DOMPurify) + 분리 프레임(iframe sandbox) 2차 방어, PDF는 툴바·다운로드 숨김
- 미리보기 모듈은 실제로 열 때만 로드(코드 분할)

심층분석 03

파일 인앱 미리보기

성과 & 예상 질문

검증으로 고정한 성과 · 면접에서 나올 질문에 대한 방어 논리

핵심 성과

신규 형식 추가 비용 = 매퍼 한 줄

설계 임팩트

6종을 하나의 렌더링 파이프라인으로 처리하는 단일 설계라, 새 형식은 매핑 한 줄만 더하면 붙는다. 형식마다 따로 만들 필요가 없다.

원본 노출

0

BE 테스트로 고정

임시 데이터 누수

0

FE 테스트로 고정

악성 코드 차단

3겹

서버 1겹 + 화면 2겹 방어

예상 질문

Q 실질 신뢰 경계는?

↳ 서버가 강제하는 CSP sandbox + DRM 사본 서빙이 경계. 클라이언트 살균은 심층 방어일 뿐 경계가 아니다.

Q 형식 판단이 확장자면 우회 가능?

↳ 형식 판단은 렌더러 선택용일 뿐 보안 결정이 아니다. 콘텐츠는 어차피 격리 + 살균을 거친다.

Q DRM 사본은 언제 만들어지나?

↳ 미리보기는 즉시 떠야 해 업로드 시 사전 생성이 유리하다. 저장 비용이 커지면 요청 시 변환으로 바꿀 수 있는 구조로 뒀다.

BE가 원본/사본 통로를 가르고 서버가 격리를 강제, FE는 6종을 단일 파이프라인으로 안전 사본만 렌더하고 자원까지 정리 — "원본은 지키고, 안전 사본만, 실수 없이, 가볍게"를 흐름 전체에 일관 적용. 기존 다운로드는 건드리지 않고 미리보기만 새로 얹었다.

심층분석 05 실시간 안정성

실시간 통신 6단계 방어 (WebSocket)

AI 응답이 끊기지 않는 실시간 채팅 만들기

프로젝트 배경

사내 AI 챗봇(LLM)을 여러 조직·여러 세션 환경으로 넓히면서 수십 초씩 이어지는 응답에서 세 가지 문제가 반복됐다 — 응답 섞임, 백그라운드 탭 끊김, 응답 몰릴 때 화면 버벅임. 서버는 그대로 두고 클라이언트(브라우저) 쪽에 6단계 방어 구조를 설계·구현했다.

핵심 성과

① 극단 폭주 · 1ms 간격

리렌더링 횟수 감소

99.6%

쉴 새 없이 응답이 쏟아지는 최악 조건

② 실사용 패턴

화면 갱신 횟수 감소

50.0%

정상 → 버스트 → 정상, 실제 사용 흐름

③ 정상 응답

정상 응답 속도 손해

0%

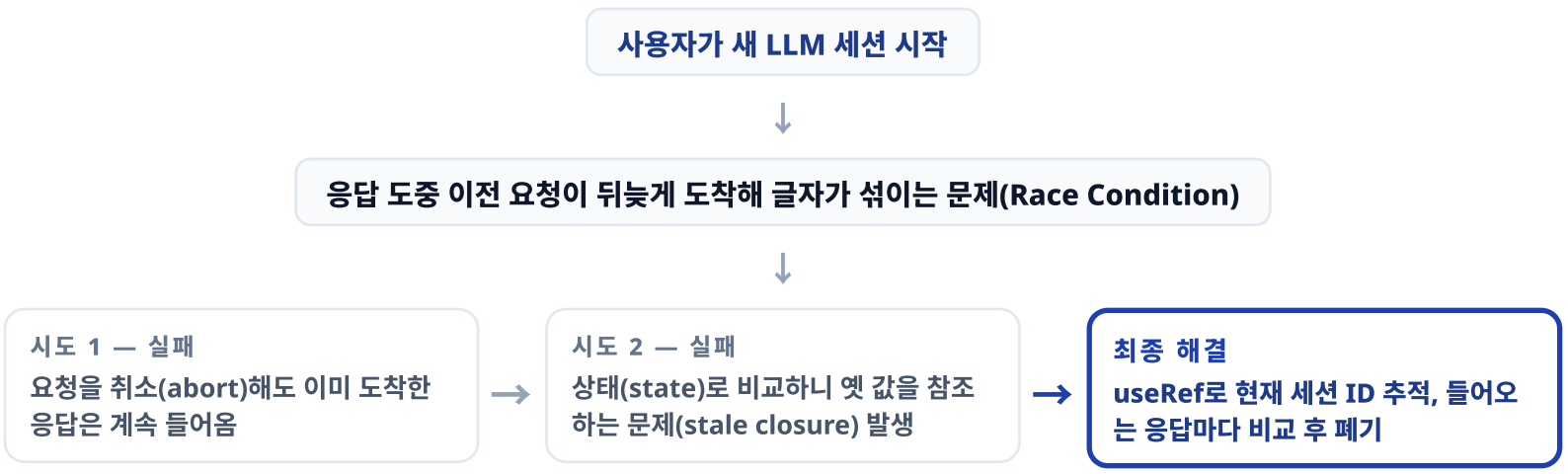
최적화로 인한 오버헤드 없음

6단계 방어 구조 위에서 아래로

1. **연결 상태 신호 (Heartbeat)**
   30초마다 신호를 주고받으며 연결이 살아있는지 확인한다. 응답이 없으면 끊긴 것으로 보고 즉시 다시 연결한다.
2. **지수 백오프 + 지터 (Backoff + Jitter)**
   재연결 간격을 1초 → 2초 → 4초 → 8초 → 16초로 늘리고, 무작위 지터(jitter)를 더해 간격을 흩뜨린다. 끊긴 사용자들이 같은 시점에 한꺼번에 몰려와 서버가 다시 멈추는 사태를 막는다.
3. **중복 연결 차단**
   같은 사용자에게 연결이 두 번 열리지 않도록 3단계로 막는다 — 중복 감지 → 소유권 잠금 → 인증키 검증.
4. **탭 상태 인식 (Page Visibility)**
   숨겨진 탭에서는 응답 수신을 잠시 멈추고, 다시 화면으로 돌아오면 즉시 재연결한다.
5. **인증키 자동 갱신 (Token Refresh)**
   인증키 만료를 감지하면 자동으로 새 키를 받아 끊김 없이 다시 연결한다.
6. **종료 코드별 처리**
   종료 코드에 따라 다르게 대응한다 — 1000 정상 종료, 1006 네트워크 단절, 1011 서버 오류, 4001 정책 위반(재로그인).

응답이 섞이는 문제 (Race Condition)

문제 → 시도 → 해결 흐름



화면 갱신 묶음 처리 (Batching) 3ms debounce

응답 조각이 몰려서 도착할 때 매번 화면을 다시 그리면 갱신이 쌓인다. 3ms 동안 입력을 모았다가 한 번에 갱신(Debounce)으로 압축한다.

도착 패턴	조각 수	개선 전	개선 후	감소율
버스트 (약 1ms 간격) 네트워크 정체 해제 직후	500	500	2	99.6%
혼합 정상 → 버스트 → 정상	200	200	100	50.0%
정상 30조각/초 의도된 무변화	60	60	60	0%

측정 방식: React <Profiler> onRender 콜백으로 화면 갱신(커밋) 횟수를 집계. 도착 타임스탬프를 고정한 동일 입력을 debounce 적용 전·후로 각각 replay해 비교.

핵심 인사이트

정상 응답에서는 조각 간격이 3ms보다 커서 묶음 처리가 작동하지 않는다. 몰릴 때만 묶고, 정상일 때는 그대로 통과시키도록 의도한 설계다.

결과 및 학습

- 응답이 섞이는 현상 해소, 백그라운드 탭 복귀 시 끊김 제거, 응답 몰림 시 화면 버벅임 제거.
- React가 오래된 값을 기억하는 한계(stale closure)를 이해하고, 상태를 React 바깥(useRef)으로 분리하는 법을 배웠다.

심층분석 04

Generative UI 파이프라인

모델 출력을 믿을 수 없어, 깨진 JSON을 받아도 위젯이 뜨도록 클라이언트에서 복구하는 파이프라인이다. 회복률을 코드로 측정해 회귀를 자동으로 잡는다.

한 줄 요약

6단계 정제(sanitize) + 2단계 파싱으로 깨진 JSON을 복구한다. 회복률은 40개 테스트 세트로 자동 측정하고, 규칙을 바꿔도 회귀가 즉시 잡힌다.

JSON 회복률

60% → 97.5%

벤치마크 테스트 실행 후 실측값 측정

복구 계층

2겹 파싱

1차 JSON.parse 실패 시 sanitize 후 2차 시도

회귀 방어

벤치마크 테스트 자동화

규칙을 바꾸면 깨짐 유형별 샘플로 자동 재검사

다단계 복구 흐름 (Pipeline)

정상이면 통과, 깨지면 ①~⑥ 복구

① fence › ② 참거짓 › ③ 꼬리쉼표 › ④ 쉼표보강 › ⑤ 특수문자 › ⑥ 균형검사

#	라벨	깨진 것 → 고친 것
①	**fence**	```json{...}``` › **{...}**
②	**참거짓**	true" › **true**
③	**꼬리쉼표**	["a",] › **["a"]**
④	**쉼표보강**	}{ › **},{**
⑤	**특수문자**	"줄 ⏎ 바꿈" › **"줄 \n 바꿈"**
⑥	**균형검사**	{"a":[1,2} › **✗ 거부**

왜 이 순서인가 — 정제 먼저, 검사 나중

문제 ⑥(괄호 균형검사)을 정제 앞에 두니, 따옴표가 깨진 JSON(true")을 복구도 못 해보고 버렸다.

원인 깨진 따옴표가 "문자열 안/밖" 판단을 어긋나게 해 괄호 개수가 틀어진다.

해결 ②~⑤로 문자를 먼저 정상화한 뒤 ⑥에서 검사하도록 순서를 바꿨다. 회귀 방지용 fixture 테스트로 고정했다.

심층분석 04

위젯 격리 — 깨져도 채팅은 살리기

사고 과정 — 어떻게 알아냈나

문제정의 잘못된 데이터로 차트 위젯이 뜨자, 채팅 화면 전체가 언마운트되고 누적 대화까지 사라졌다.

가설 Error Boundary가 없으면 React가 트리 전체를 언마운트한다 → 위젯 예외가 채팅까지 끌고 내려간다고 봤다.

검증 일부러 깨뜨린 fixture(괄호 불균형 JSON)를 주입해 재현했다. Boundary 없는 버전은 세션 전체가 소실됐고, 위젯 단위 Boundary를 넣자 위젯만 fallback으로 처리됐다.

해결 1 위젯마다 안전망 두르기

위젯마다 안전망(GenUIErrorBoundary)을 둘렀다. 깨져도 그 자리에만 "위젯을 표시할 수 없습니다" 대체 메시지가 뜨고, 채팅은 그대로 산다.

```
<GenUIErrorBoundary>
  <CardSelectionWidget ... />
</GenUIErrorBoundary>
```

해결 2 이벤트 버스로 분리 — Context 대신 pub/sub

React Context로 전달하면 value가 바뀔 때마다 채팅 메시지 N개가 전부 리렌더돼 비용이 컸다. 그래서 전역 이벤트 버스(pub/sub)로 바꿔 상호작용이 React 트리를 거치지 않게 했다(무관한 메시지 리렌더 0). 약점인 이벤트명 오타·구독 해제 누락은 EventMap 타입과 useBusEvent(cleanup) 훅으로 막았다.

```
eventBus.emit('genui:interact', payload)
eventBus.on('genui:interact', handler)
```

회고 한계 발견과 재발 방지

검증하다 Error Boundary가 비동기·핸들러 예외는 못 잡는다는 한계를 발견해 try/catch 수렴 로직을 추가했다. 이후 같은 깨짐 패턴은 회귀 벤치가 자동으로 잡는다.

스트리밍 중 깜빡임 제거 — skeleton / dots UX 타이밍

위젯 JSON은 한 토큰씩 흘러들어오는 동안 계속 미완성 상태다. 도착하는 즉시 그리면 반쪽짜리 위젯이 마운트·언마운트를 반복해 화면이 깜빡인다. 그래서 스트리밍 구간에는 위젯을 그리지 않고 skeleton과 typing-dots로 자리만 잡았다가, 파이프라인이 완전한 JSON을 통과시킨 순간 한 번에 실제 위젯으로 바꿨다.

스트리밍 중	깨짐	완성
skeleton · dots	fallback	실제 위젯

스트리밍·깨짐·완성 세 상태를 분리해 사용자가 보는 첫 화면을 안정화했다.

위젯 카탈로그 종류

- 폼 위젯 · 필드 8종
- 카드 선택 · 단일/다중
- 차트 위젯 · 데이터 시각화

측정 결과 genui-parse.bench 실측 (sanitize 전후)

- 파싱 성공률 60% → 97.5%
- 렌더 실패 payload 47건 → 1건
- 위젯 장애 약 95% ↓

sanitize(1차 방어) → Error Boundary(2차 방어)의 2단 방어로 위젯 장애 자체를 약 95% 줄였다. fixture 5종(코드블록 감싸짐·Bool 따옴표·후행 쉼표·괄호 불균형·복합 깨짐)을 genui-parse.bench로 측정. 위젯 1개 장애의 영향 범위: 세션 전체(메시지 N개+입력 컨텍스트) → 위젯 1개.

심층분석 06

Dockerfile 고도화

한 줄 요약 운영팀이 두 번 반려한 이미지를 슬림화했는데, 기능이 붙는 동안 1.56GB가 슬그머니 되살아났다. 이를 레이어 측정으로 적발해 3.63GB → 1.82GB(-50%)로 되돌렸다. 숫자는 전부 직접 빌드해서 잴 값이다.

50% ↓
이미지(비압축) 3.63GB → 1.82GB (실측)

1.56GB
측정 없으면 안 보이던 숨은 회귀 적발

~10s ↓
cold pull ~24s → ~13s (초는 추정)

S 이 이미지는 받기 어렵습니다를 두 번 받았다

크기 초기 이미지가 비대해 K8s 기동이 느렸다

보안 non-root 정책과 APM(WhaTap)의 로그 write 권한이 충돌해 에이전트 부팅이 실패했다

안정성 필수 환경변수(NEXT_PUBLIC_*) 누락이 런타임에야 드러났다

1차로 슬림화해 운영에 투입했다. 문제는 그 다음이었다 — 인보이스 PDF 기능이 붙고 다시 재니 3.63GB였다.

T 동작한다가 아니라 왜 이 크기인가에 답한다

1. 레이어별 기여를 분리 측정해 정당한 비용과 결함을 가른다
2. 줄어든 게 왜 다시 늘었는지 한 줄로 설명할 수 있게 만든다
3. 장애 가능성은 가장 싼 자리(빌드 타임)에서 막는다

심층분석 06 · 계속

레이어를 까서 회귀를 잡았다

줄여놓은 게 되살아난 걸 측정으로 적발 · 부팅 회귀는 스모크 테스트가 포착

A Action — 측정으로 숨은 회귀를 잡았다

① 기반 · 1차 슬림화

멀티스테이지 분리(base · deps · builder · runner)로 빌드 전용 의존성을 최종 이미지에서 배제하고, Next standalone으로 런타임 실의존성만 남겼다. non-root × WhaTap 충돌은 로그 폴더 소유권을 에이전트 사용자에게 넘겨 해소. 필수 환경변수 누락 시 빌드 자체가 실패하도록 fail-fast.

② 적발 · 레이어 기여 분리 측정

1.56GB **모니터링 에이전트 런타임 설치** — 결함

npm install이 standalone으로 걷어낸 dev 의존성(typescript 등)을 통째로 부활

0.89GB **Chromium + 한글 폰트** — 정당

PDF 렌더에 필수

나머지 **앱 · 런타임 본체** — 정상

③ 수정 · 정리

에이전트를 격리 설치해 결과만 가져오게 바꿨더니 컨테이너가 부팅 실패. 커스텀 server.js가 Next 풀 패키지를 필요로 했는데 슬림 트리엔 없었다 — 스모크 테스트가 잡았다. 뜯어보니 그 서버는 기본 핸들러에 넘기는 게 전부라, Next 표준 서버로 바꾸고 에이전트는 실행 옵션으로 얹어 정리했다.

R Result — 측정 가능한 성과

지표	Before	After	Δ
이미지(비압축)	3.63GB	**1.82GB**	**−50%**
이미지(압축)	901MB	**540MB**	**−40%**
cold pull(풀 구간)	~24s	**~13s**	**~−10s**
숨은 회귀	1.56GB 잠복	**적발·제거**	—
non-root × APM	부팅 실패	**이미지서 해소**	—

인사이트

- **standalone·alpine은 누구나 쓴다. 차별은 줄여놓은 게 되살아난 걸 측정으로 잡아내는 데 있다.**
- '되긴 한다'는 끝이 아니다 — 운영팀이 받을 수 있어야 끝이다.
- 장애는 가장 싼 자리에서 막는다 — 런타임이 아니라 빌드 1초.

다음에 한다면

→ Chromium을 사이드카로 분리하고 PDF를 launch에서 connect로 전환하면 앱 이미지는 1GB 아래로. 단 앱 이미지가 줄뿐 Pod 전체 자원이 주는 건 아니라, 운영팀과 함께 정할 사안이라 백로그. (→ 서버사이드 PDF 인보이스)