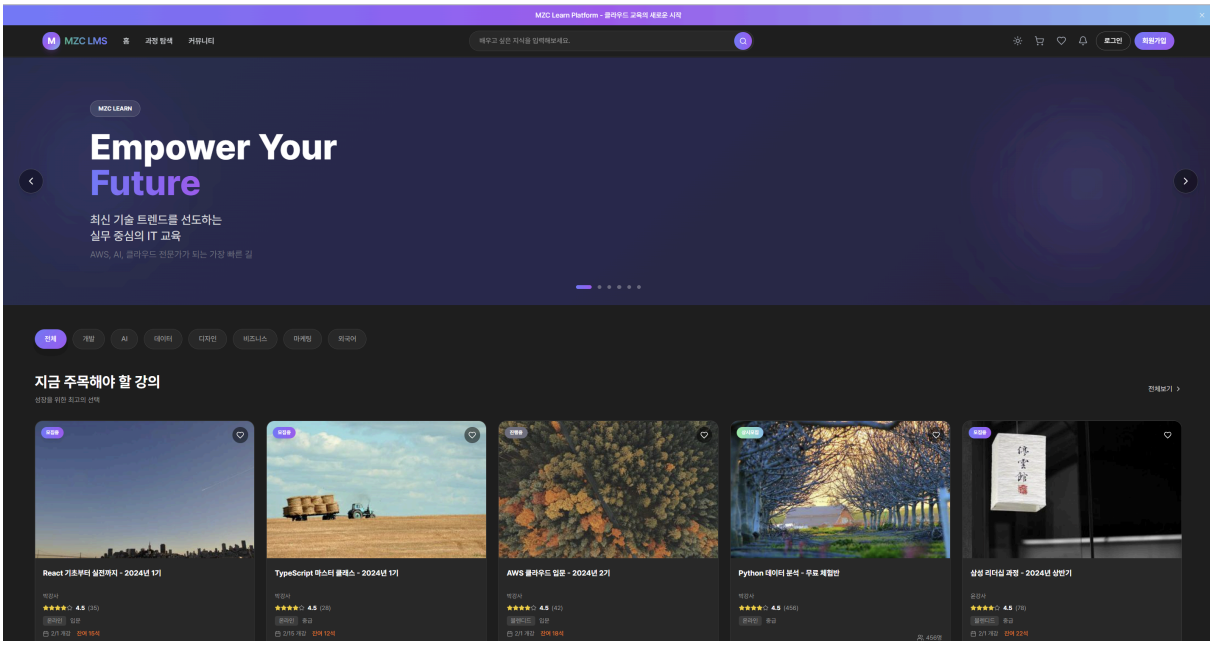
MZC-LMS 플랫폼 포트폴리오

프로젝트 요약

> 기업 전용 온라인 교육 플랫폼을 빠르게 구축해주는 **멀티테넌트 SaaS 기반 학습 관리 플랫폼**



프로젝트 소개

연계/소속	메가존클라우드(주) Ability Training Unit
개발 기간	2025.11.03 ~ 2026.01.23
플랫폼	멀티테넌트 SaaS 기반 LMS 플랫폼
개발 인원	과천3 부산4 (디자이너 각1명) 분산 팀
역할	팀장
개발 방법론	Agile/Scrum (12주 12스프린트, 1주 단위 스프린트)
관련 링크	하단의 관련 링크로 이동합니다.

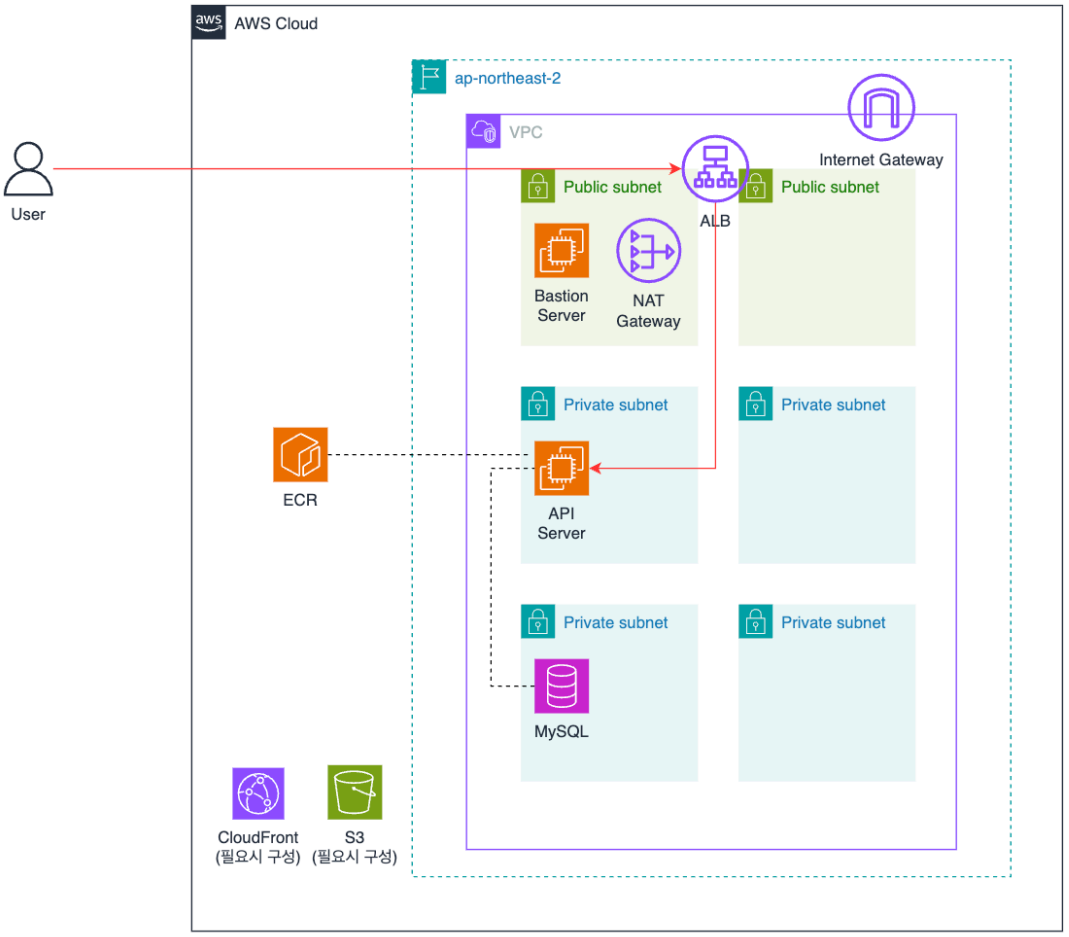
핵심 성과

- 7명 분산팀(과천-부산) 팀장으로 12주 12스프린트 전체 일정 완주
- 멀티테넌트 SaaS 아키텍처 설계 — 3단계 데이터 격리, 6계층 RBAC 권한 체계
- 맥도날드 시스템 도입으로 LLM 표준 체계 도입 성공. 코드 리뷰 컨벤션 위반 71% 감소 (5-7건 → 1-2건)
- **메가존클라우드 사내 교육 플랫폼 도입 검토 대상 선정**, 인수인계 완료

기술 스택

영역	스택
Frontend	React 19, TypeScript 5.6, TailwindCSS 3.4
Backend	Java 21, Spring Boot 3.4.12, Gradle 8.5, Spring Data JPA, MySQL 8.0
DevOps / Infra	AWS (EC2, RDS, S3, CloudFront), GitHub Actions CI/CD, Docker, SonarQube
AI-Assisted Dev	Claude Opus 4.5

시스템 아키텍처



아키텍처 인터랙티브 뷰: LMS Architecture

프로젝트 배경

기업들이 자사 교육 플랫폼을 구축할 때 선택지는 크게 세 가지였습니다.

1.단발성 자체 구축(SI)은 기술 부채가 누적되고 지속적 업데이트가 어렵고,

2.설치형 LMS는 서버·보안 인프라를 직접 관리해야 하며,

3.Enterprise SaaS는 자사 고유 브랜딩 적용이 불가능합니다.

MZC-LMS는 "합리적 비용으로 빠르게 자사 브랜드 교육 플랫폼을 구축하고 싶다"는 니즈에 대응해, 기업별 독립 서비스(브랜드·도메인·데이터 완전 분리)를 SaaS로 제공하는 플랫폼입니다.

담당 기능

모듈 개발

- **System Admin 모듈**: 도메인 기반 전체 테넌트(고객사) 생성 및 관리
- **Tenant Admin 모듈**: 테넌트별 설정, 브랜딩(로고/컬러), 사용자 관리
- **User Master 모듈**: JWT 기반 인증/인가, 6계층 RBAC 권한 체계

아키텍처 및 설계

- 멀티테넌시 구조 설계 (도메인 기반 테넌트 분리, JWT 클레임에 TID 포함)
- 전체 모듈 구조 설계 및 문서화
- 6계층 RBAC 권한 체계 설계 및 구현

개발 프로세스 구축

- Backend/Frontend 코딩 컨벤션 문서 제작
- Git 브랜치 전략 도입 (main/develop/feature, 브랜치 보호 규칙)
- 맥도날드 시스템 도입 (LLM 활용 개발 프로세스 표준화)
- CI/CD 파이프라인 구축 (GitHub Actions + SonarQube 연동)
- AWS 인프라 배포

디자이너 협업

- Figma 기반 디자인 시스템 연동 및 컴포넌트 명세 협의
- 데일리 스크럼, 주간 디자인 리뷰 미팅 운영
- UX 요구사항과 기술적 제약 사이 합의점 도출

AI 기반 개발 프로세스 설계

Context Engineering 관점의 LLM 활용 체계 설계 – McDonald System 상세

> "AI를 단순히 쓰는 것이 아니라, 팀의 표준에 맞게 AI를 활용하는 체계를 만든 것"

도입 배경

프로젝트에서 7명의 개발자가 LLM(Claude)을 활용해 개발을 진행했지만 각자 사용하는 방식이 달랐습니다.

그 결과 다음과 같은 문제가 발생했습니다.

- 코드 스타일 불일치
- 환각(Hallucination)으로 인한 잘못된 코드 생성
- 프로젝트 컨벤션 위반
- 코드 리뷰에서 스타일 수정에 많은 시간 소모

문제의 원인은 **프롬프트 작성 능력의 차이가 아니라 LLM이 이해하는 프로젝트 맥락(Context)의 부재**였습니다.

그래서 프롬프트를 개선하는 방식 대신 **프로젝트 맥락을 구조화하여 LLM에 전달하는 Context Engineering 기반 접근**으로 전환했습니다.

McDonald System 개념

맥도날드 시스템이란 **누가 일을 하더라도 동일한 결과가 나오도록 작업 과정을 표준화하는 방식**을 의미합니다.

맥도날드 매장에서는 직원의 숙련도와 관계없이 동일한 햄버거가 만들어지도록 다음과 같은 요소가 표준화되어 있습니다.

- 조리 과정
- 재료 구성
- 작업 순서
- 품질 기준

즉, **개인의 능력에 의존하지 않고 시스템으로 결과 품질을 유지하는 구조**입니다.

이 개념을 LLM 개발 환경에 적용했습니다.

왜 Prompt Engineering이 아니라 Context Engineering인가

프로젝트 초기에 LLM을 활용할 때는 일반적인 **Prompt Engineering 방식**을 사용했습니다.

즉, 기능을 구현할 때마다 상세한 프롬프트를 작성하여 원하는 코드가 생성되도록 하는 방식이었습니다.

방법	문제점
Spring Boot 기반 API 생성 요청	개발자마다 프롬프트 작성 방식이 달라 결과물 품질이 달라짐
JPA Repository 패턴 적용 요청	프로젝트 컨벤션을 매번 프롬프트에 반복 작성해야 함
ControllerAdvice 기반 예외 처리 요구	프롬프트가 길어질수록 유지보수가 어려움
DTO/Entity 분리 규칙 명시	팀 규모가 커질수록 일관성을 유지하기 어려움

결국 문제의 본질은 **프롬프트 작성 능력이 아니라 LLM이 이해하는 프로젝트 맥락이 부족한 것**이었습니다.

그래서 접근 방식을 **Prompt Engineering → Context Engineering**으로 전환했습니다.

핵심 차이

구분	Prompt Engineering	Context Engineering
접근 방식	매번 좋은 프롬프트 작성	LLM이 이해할 맥락을 구조화
문제 해결 방식	질문을 더 잘 작성	배경 정보를 먼저 제공
결과물 품질	프롬프트 작성자 역량 의존	팀 전체에서 동일한 품질 유지
유지보수	프롬프트 반복 작성 필요	컨텍스트 문서 최신화

구성 요소

1. 컨텍스트 문서 (Context Document)

LLM에게 프로젝트 맥락을 전달하기 위한 문서입니다.

포함 내용:

- 프로젝트 아키텍처 개요 (멀티테넌트 구조, 모듈 관계도)

- 기술 스택 및 버전 정보
- 디렉토리 구조 및 레이어별 책임 (Controller → Service → Repository)
- 네이밍 규칙, 에러 처리 패턴, API 응답 포맷 표준

이 문서를 통해 **LLM이 프로젝트 맥락을 이해한 상태에서 코드를 생성하도록 설계했습니다.**

2. 코딩 컨벤션 문서

영역	주요 규칙
Backend	패키지 구조, DTO/Entity 분리 원칙, Repository 작성 규칙, 예외 처리 패턴, 응답 래퍼 클래스
Frontend	컴포넌트 구조, 상태 관리 패턴, API 호출 규칙, TailwindCSS 스타일링 규칙
공통	커밋 메시지 포맷 (Conventional Commits), PR 템플릿, 브랜치 네이밍

3. 상황별 프롬프트 템플릿

상황	용도	예시
신규 API 개발	요구사항 → 컨벤션에 맞는 Controller/Service/Repository 코드 생성	"회원 목록 조회 API 만들어줘" → 팀 표준에 맞는 3계층 코드
버그 수정	에러 로그 + 관련 코드 → 원인 분석 및 수정안	스택 트레이스 붙여넣기 → 원인과 컨벤션 준수 수정안
코드 리뷰	작성 코드 → 컨벤션 위반 사항 검토	PR 전 셀프 리뷰 용도
테스트 코드	대상 메서드 → 테스트 케이스 자동 생성	Service 메서드 → JUnit 테스트

적용 프로세스

1. [생산성] 맥도날드 시스템: LLM 컨텍스트 엔지니어링 7단계

1단계	**Role (역할 정의)**	**숙련된 Spring Boot 아키텍트 페르소나 부여.**
2단계	**Context (맥락 주입)**	**프로젝트 구조, 레이어별 책임(Controller-Service-Repository) 정보 전달.**
3단계	**Constraint (제약 설정)**	**'Setter 금지', 'JPA Row-level 필터링 필수' 등 컨벤션 강제.**
4단계	**Input (데이터 제공)**	**구현 기능 요구사항 및 DB 스키마/API 명세 전달.**

5단계	**Format (형식 지정)**	**팀 표준 패키지 구조에 맞춘 코드 출력 요구.**
6단계	**Few-Shot (예시 활용)**	**기존에 작성된 모범 코드 샘플을 제공하여 스타일 학습.**
7단계	**Refinement (자가 검토)**	**보안 취약점 및 로직 결함에 대한 LLM의 2차 검토 요청.**

➔ **성과:** LLM 환각 현상 및 컨벤션 위반 **70% 감소**, 코드 리뷰 효율성 극대화.

2. [LLM 통제 전략]MoSCoW 기반 프로젝트 우선순위 관리 (Strategic Planning)

배경 : 컨텍스트가 누적되거나, 복잡한 작업을 요청하는 경우 **구현 계획 없이 바로 코드 생성**을 진행하거나 핵심 기능이 아닌 기능을 우선 구현하는 경우가 발생

해결 : LLM에게 작업을 요청할 때 단순히 기능 구현을 요청하지 않고 MoSCoW 우선순위 기준을 **컨텍스트 문서에 포함**하여 LLM이 항상 참고하도록 했습니다.

예를 들어 기능 개발을 요청할 때 단순히 구현을 요청하는 대신 다음과 같은 방식으로 작업을 진행했습니다.

```
회원가입 / 로그인 / 로그아웃 기능을 구현하려고 합니다.
MoSCoW 컨텍스트와 claude.md 문서를 참고해
먼저 기능 구현 계획을 세워주세요.

[MoSCoW]
• Must Have (치명적 필수): 미구현 시 프로젝트 실패로 간주되는 핵심 기능.
◦ 3단계 데이터 격리 구조(Multi-tenancy), 6계층 RBAC 권한 체계.

• Should Have (중요하지만 대체 가능): 중요도는 높으나 수동 프로세스로 일시적 대체가 가능한 기능.
◦ 모듈 간 단방향 통신 구조 최적화, 학습 진도율 체크 로직.

• Could Have (구현 시 가치 증대): 자원 여유 시 반영하며 사용자 경험(UX)을 향상시키는 기능.
◦ 테넌트별 UI 커스터마이징, 실시간 학습 알림 서비스.

• Won't Have (이번 범위 제외): 핵심 가치 집중을 위해 명확히 배제한 고비용 기능.
◦ 외부 결제 모듈 연동 고도화, AI 기반 자동 채점 시스템.
```

그 결과 LLM이

1. 기능 구현 계획 생성
2. Must → Should 순서로 작업 정리
3. 계획에 따라 코드 생성
4. 빠른 MVP 선정 → 핵심 기능부터 구현

과 같은 **계획 기반 개발 흐름을 따르는 결과**를 얻을 수 있었습니다.

LLM 활용 체계 도입 성과

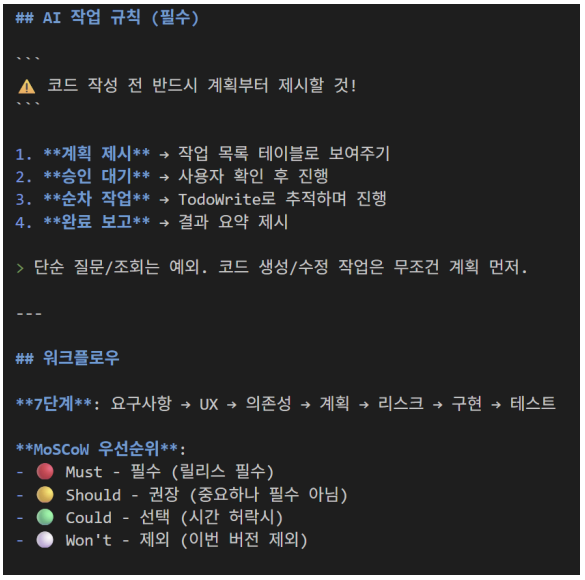
지표	Before	After
코드 리뷰 시 컨벤션 위반 지적	리뷰당 평균 5-7건	평균 1-2건
신규 팀원 온보딩	1주일 이상 적응 기간	컨텍스트 문서 숙지 후 즉시 투입
코드 스타일 일관성	작성자별 상이	전체 통일

```
# mzc-lp - AI 작업 가이드

> **핵심 원칙**: 이 문서만으로 대부분의 작업 시작 가능. 부족하면 부록 참조.
> **필수 작업 규칙**: 모든 작업은 **반드시 계획 먼저 제시** → 승인 → 순차 진행 → 완료 보고

---

## 프로젝트 개요

| 항목 | 내용 |
|------|------|
| **Backend** | Spring Boot 3.4.12, Java 21 |
| **Frontend** | React 19.x, TypeScript 5.x, Vite 7.x, TailwindCSS |
| **Database** | MySQL 8.0 |
| **Infra** | AWS (ECS, RDS, S3, CloudFront), Docker |
| **인증** | JWT (Access + Refresh Token) |

### 저장소 구조

> 폴리레포 (3개 저장소) → [POLY-REPO.md](./POLY-REPO.md)

---

## 핵심 규칙 (Must-Know)

### Backend
```
✅ Entity: Setter 금지 → 비즈니스 메서드 사용
✅ Service: @Transactional(readOnly=true) 클래스 레벨
✅ Controller: try-catch 금지 → GlobalExceptionHandler
✅ DTO: Java Record + from() 정적 팩토리
✅ Enum: @Enumerated(EnumType.STRING)
```

### Frontend
```
✅ any 타입 금지 → 명시적 타입 정의
✅ 서버 상태: React Query (useState는 UI 상태만)
✅ API: Axios Instance + handleApiError
✅ 컴포넌트: Props Destructuring + Early Return
```
```

```
## 컨벤션 로딩 (작업별 선택)

| 작업 | 필수 컨벤션 |
|------|------------|
| 프로젝트 구조 | `01-PROJECT-STRUCTURE` |
| Git | `02-GIT-CONVENTIONS` |
| Controller | `03-CONTROLLER-CONVENTIONS` |
| Service | `04-SERVICE-CONVENTIONS` |
| Repository | `05-REPOSITORY-CONVENTIONS` |
| Entity | `06-ENTITY-CONVENTIONS` |
| DTO | `07-DTO-CONVENTIONS` |
| Exception | `08-EXCEPTION-CONVENTIONS` |
| React Core | `10-REACT-TYPESCRIPT-CORE` |
| React 구조 | `11-REACT-PROJECT-STRUCTURE` |
| Component | `12-REACT-COMPONENT-CONVENTIONS` |
| State 관리 | `13-REACT-STATE-MANAGEMENT` |
| API Service | `14-REACT-API-INTEGRATION` |
| Backend Test | `15-BACKEND-TEST-CONVENTIONS` |
| Frontend Test | `16-FRONTEND-TEST-CONVENTIONS` |
| Design/UI | `design/00-DESIGN-CONVENTIONS` |
| Docker | `17-DOCKER-CONVENTIONS` |
| Database | `18-DATABASE-CONVENTIONS` |
| AWS 배포 | `19-AWS-CONVENTIONS` |
| 보안 | `20-SECURITY-CONVENTIONS` |
| 성능 | `21-PERFORMANCE-CONVENTIONS` |
| 외부 API | `22-EXTERNAL-API-CONVENTIONS` |
| 멀티테넌시 | `23-MULTI-TENANCY` |
| Ignore 설정 | `24-IGNORE-CONVENTIONS` |
| 다국어(i18n) | `25-I18N-CONVENTIONS` |
| 코드 품질 분석 | `26-CODE-QUALITY-ANALYSIS` |
| AI 시대 개발자 역할 | `27-DEVELOPER-ROLE-AI-ERA` |

> 컨벤션 위치: `docs/conventions/`
```

```
## 작업 순서

**Backend CRUD**: Entity → Repository → DTO → Exception → Service → Controller → Test

**Frontend 페이지**: Types → API Service → React Query Hook → Component → Test

---

## 프로젝트 구조

```
backend/.../domain/
├── user/   # controller, service, repository, entity, dto, exception
├── course/
└── enrollment/
global/   # config, exception, common

frontend/src/
├── pages/   # 페이지 컴포넌트
├── components/   # 재사용 컴포넌트 (common, layout)
├── services/   # API 호출
├── hooks/   # Custom Hooks (React Query 래핑)
├── stores/   # 전역 상태 (필요시)
├── types/   # TypeScript 타입
└── utils/   # 유틸리티 함수
```
```



```
## AI 작업 규칙 (필수)

```
⚠️ 코드 작성 전 반드시 계획부터 제시할 것!
```

1. **계획 제시** → 작업 목록 테이블로 보여주기
2. **승인 대기** → 사용자 확인 후 진행
3. **순차 작업** → TodoWrite로 추적하며 진행
4. **완료 보고** → 결과 요약 제시

> 단순 질문/조회는 예외. 코드 생성/수정 작업은 무조건 계획 먼저.

워크플로우

7단계: 요구사항 → UX → 의존성 → 계획 → 리스크 → 구현 → 테스트

MoSCoW 우선순위:
- 🔴 Must - 필수 (릴리스 필수)
- 🟡 Should - 권장 (중요하나 필수 아님)
- 🟢 Could - 선택 (시간 허락시)
- 🔵 Won't - 제외 (이번 버전 제외)
```

CLAUDE.md

CONTEXT-ENGINEERING.md

CONTEXT-ENGINEERING-MCDONALD-SYSTEM.md

핵심 인사이트

- 도구 자체보다 **도구를 어떻게 쓰느냐**가 중요
- LLM은 단순한 코드 작성 도구가 아닌 **팀 컨벤션 전파 도구** 로 활용
- 새로운 기술을 개인의 생산성이 아닌 **팀 전체의 생산성 향상**으로 연결

기술적 의사결정

1. 멀티테넌시 데이터 격리 전략

문제

하나의 애플리케이션에서 여러 기업(테넌트)의 데이터를 서비스하므로, 한 테넌트의 사용자가 다른 테넌트의 데이터에 접근할 수 없도록 격리해야 합니다.

검토한 선택지

전략	격리 수준	장점	단점
DB per Tenant	완벽	물리적 분리로 보안 최상	테넌트 증가 시 인프라 비용 급증 RDS 인스턴스마다 커넥션 풀 별도 관리 필요
Schema per Tenant	높음	DB 하나에서 논리적 분리	런타임 동적 스키마 전환 구현이 복잡. 마이그레이션 시 전 스키마 일괄 적용 부담
Row-level 격리 (Shared DB/Schema)	보통	단일 DB·커넥션 풀로 리소스 효율적, JPA 레벨에서 자연스럽게 구현 가능	쿼리마다 tenant_id 조건 누락 시 데이터 유출 위험

선택: Row-level 격리 + 3단계 방어 체계

선택 이유

- 인턴 프로젝트 환경에서 **DB per Tenant 구조는 비용적으로 비현실적**
- Shared DB 구조가 **인프라 복잡도를 최소화**
- Row-level의 단점인 데이터 누락 위험을 **3단계 방어 구조로 보완**

Row-level 격리의 약점인 tenant_id 누락 시 데이터 유출을 방어하기 위해 3단계 방어 체계

(도메인 식별 → JWT 검증 → Hibernate Filter)로 구조적으로 방지했습니다.


클라이언트 요청 mzc.lms.com/api/courses
LAYER 1 · NETWORK
도메인 기반 테넌트 식별
TenantResolver Filter
서브도메인에서 tenant_id 추출 → ThreadLocal 에 저장
mzc.lms.com → mzc
samsung.lms.com → samsung
PASS
LAYER 2 · AUTH
JWT 클레임 검증
Security Filter
토큰 내 tenant_id 와 Layer 1에서 식별한 값 비교
✓ MATCH → 통과
✗ MISMATCH → 403
PASS
LAYER 3 · DATA
JPA Row-level 자동 필터링
Repository Layer
@Filter + @FilterDef 로 모든 쿼리에 tenant_id 자동 주입
SELECT * FROM courses WHERE ...
AND tenant_id = 'mzc' ← 자동
PASS
테넌트 격리 완료
해당 테넌트 데이터만 안전하게 응답

이 구조의 핵심은 **어느 한 레이어가 실패해도 다른 레이어가 방어**한다는 점입니다.

예를 들어, 개발자가 커스텀 쿼리에서 tenant_id 조건을 빠뜨려도 ③의 Hibernate Filter가 자동으로 걸리고, 토큰 탈취 시에도 ②에서 도메인-토큰 불일치를 탐지합니다.

결과

- 테넌트 간 데이터 완벽 격리 달성
- 신규 테넌트 추가 시 코드 수정 없이 DB에 tenant 레코드만 추가하면 자동 적용
- Hibernate Filter 기반이므로 개발자 실수에 의한 데이터 유출 구조적 방지

---

2. 6계층 RBAC 권한 체계

문제

멀티테넌트 LMS에서는 "시스템 전체를 관리하는 역할"부터 "한 테넌트 내에서 수강만 하는 역할"까지 권한 범위가 극단적으로 다릅니다.
단순한 Role 기반 접근 제어로는 "이 Operator는 수강신청 관리만 가능하고, 저 Operator는 과정 개설까지 가능"과 같은 세분화된 제어가 어렵습니다.

검토한 선택지

방식	장점	단점
단순 Role 기반 (ROLE_ADMIN, ROLE_USER)	구현 간단, Spring Security 기본 지원	역할 내 세부 권한 차이 표현 불가. 역할이 늘어나면 Role 폭발
URL 패턴 기반 필터링	설정 파일로 관리 가능	동일 URL에서 역할별로 다른 동작(조회만 vs 수정 가능)을 구분하기 어려움
계층형 RBAC (Role → Authority → Privilege → Resource)	역할/권한/자원을 독립적으로 관리, 조합으로 세밀한 제어 가능	테이블 구조 복잡, 초기 설계 비용 높음

선택: 6계층 RBAC

B2C와 B2B를 동시에 지원하는 구조에서, 테넌트마다 활성화하는 역할과 권한 조합이 달라질 수 있습니다.

예를 들어 B2B 기업교육 테넌트에서는 Designer 역할이 필요 없을 수 있고, 특정 테넌트에서는 Instructor에게 수강생 관리 권한까지 부여하고 싶을 수 있습니다.

이런 유연성을 위해 Role과 권한을 분리하는 계층형 구조가 필요했습니다.


System Admin
테넌트 생성/삭제, 전체 시스템 설정
Tenant Admin
테넌트 내 사용자 관리, 브랜딩 설정
Course Operator
수강신청 관리, 과정 개설, 차수 승인, 강사 배정
Designer
강의 콘텐츠 설계 및 개설
Instructor
배정된 강의 운영, 수강생 평가
User
수강신청, 장바구니, 결제, 커뮤니티

런타임 권한 검증 흐름


API 요청 POST /api/mzc/courses
STEP 1 · FILTER
Spring Security FilterChain
• JWT에서 role, tenant_id 추출
• URL 패턴 + HTTP Method 기반 1차 접근 제어
/api/{tenantId}/admin/** → TENANT_ADMIN 이상만 허용
✗ URL 패턴 불일치 → 403
PASS
STEP 2 · METHOD
@PreAuthorize 메서드 레벨 검증
• 비즈니스 로직 단위 세밀한 권한 검증
• Role이 아닌 Authority 단위 검증 → 역할 변경 시 코드 수정 불필요
@PreAuthorize("hasAuthority('COURSE_CREATE')")
✗ Authority 미보유 → 403
PASS
STEP 3 · DATA
Service Layer 데이터 레벨 검증
• 본인 테넌트 데이터만 조작 가능한지 최종 확인
• tenant_id 일치 + 리소스 소유권 검증
courseService.update(id) → 내부에서 tenant 소유권 체크
✗ 소유권 불일치 → 403
PASS
요청 처리 완료

URL 패턴 + 메서드 레벨 검증

URL 패턴으로 1차 접근 제어를 수행하고, `@PreAuthorize`로 메서드 레벨 권한 검증을 적용해 **불필요한 요청 차단과 세밀한 권한 제어를 동시에 확보했습니다.**

Authority 기반 권한 검증

`hasRole()` 대신 `hasAuthority()` 기반 검증을 사용해 **Role 변경 시 코드 수정 없이 DB 매핑만으로 권한 정책을 확장할 수 있도록 설계했습니다.**

---

3. 모듈 간 양방향 → 단방향 통신 재설계

문제 상황 (구체적 시나리오)

초기 설계에서는 TS(타임스케줄) 모듈과 IIS(강사정보) 모듈이 서로를 호출하는 양방향 의존 관계였습니다.

이 구조에서 다음 문제가 발생했습니다.

```
TS → IIS 호출
IIS → TS 재호출
→ 동일 트랜잭션 내 순환 호출
```

실제 동시 요청 테스트 에서

- 응답 지연
- 트랜잭션 타임아웃

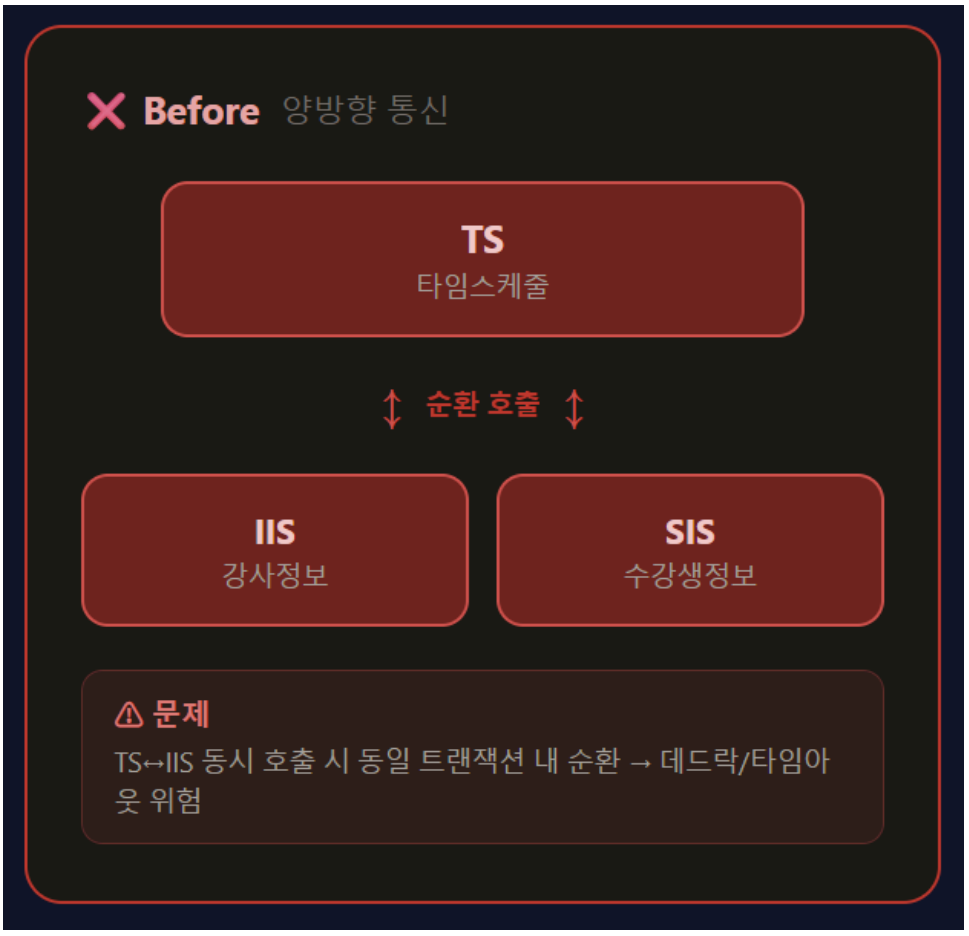
문제가 발생했습니다.

검토한 선택지

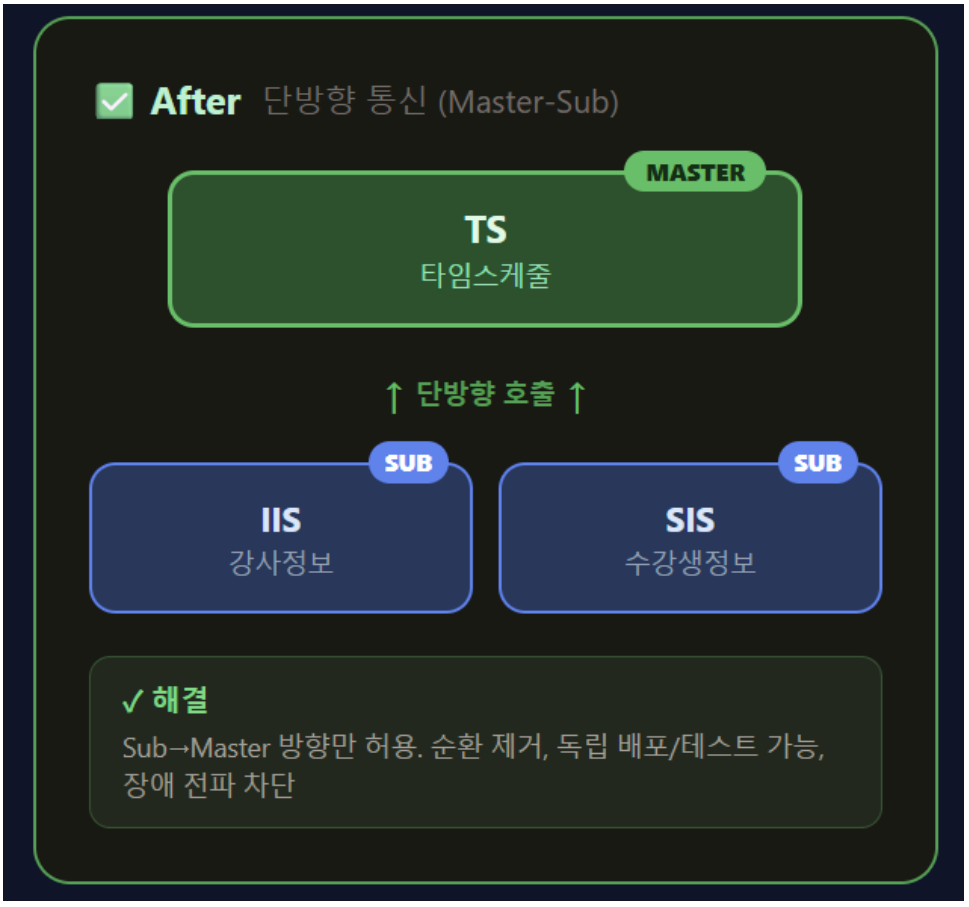
방식	장점	단점
이벤트 기반 비동기 통신 (Spring Events / Message Queue)	모듈 간 완전 디커플링, 확장성 우수	이벤트 순서 보장 어려움, 디버깅 복잡, 러닝커브 (팀원 대부분 첫 경험)
API Gateway 중재	순환 호출 방지 가능	프로젝트 규모 대비 오버엔지니어링, 게이트웨이 단일 장애점
Master-Sub 단방향 호출 규칙	의존 방향 명확, 디버깅 용이, 즉시 적용 가능	설계 제약으로 일부 기능 구현 시 우회 필요

선택: Master-Sub 단방향 호출

변경 전



변경 후



이벤트 기반을 선택하지 않은 이유:

의존 방향이 명확하고 즉시 적용 가능한 Master-Sub 규칙을 선택했습니다.

선택

Master → Sub 단방향 호출 구조

```
TS (Master)
→ IIS (Sub)
→ SIS (Sub)
```

선택 이유

- 데이터 흐름 기준이 **스케줄** 이었고
- 대부분의 유스케이스의 흐름이

```
스케줄 → 강사 / 수강생 정보 조회
```

구조였기 때문입니다.

결과

- 순환 호출로 인한 Deadlock 제거
- 모듈별 독립적 테스트 가능 — IIS 단독으로 단위 테스트 시 TS 모킹 불필요
- 장애 전파 범위 축소 — Sub 모듈 장애가 Master에 영향 미치지 않음

---

4. 개발-디자인 관점 충돌 해결

문제 상황

디자이너가 관리자 기능을 하나의 화면에서 통합 관리하는 UI를 제안했지만, 해당 설계는 RBAC 권한 구조를 고려하지 않은 접근 방식이었습니다.

멀티테넌트 환경에서는 역할별 접근 권한이 다르기 때문에 단일 관리자 화면에서 모든 기능을 제공할 경우
권한 검증과 데이터 접근 제어가 깨질 가능성이 있었습니다.

해결

Daily Scrum에 디자이너를 정식 참여시켜 매일 10분간 진행 상황을 공유했습니다. 기술적 제약사항을 비개발자도 이해할 수 있게 시각화("이 기능은 3일, 대안은 1일")하여 설명하고, 반대로 UX 관점의 근거(사용자 시나리오, 경쟁사 사례)를 공유받는 양방향 소통 구조를 정착시켰습니다.

결과

"기술적으로 가능하면서 UX도 만족하는" 대안을 함께 도출하는 프로세스가 자리잡았습니다. 사용자 중심 UX를 유지하면서 개발 일정 내 구현 가능한 설계를 완성했습니다.

업무 성과 요약

리더십

- 7명 분산팀(과천-부산) 개발 팀장으로 12주 12스프린트 전체 일정 완주
- Daily Scrum 기반 협업 체계 정착, 개발자-디자이너 협업 문화 경험

기술/아키텍처

- 멀티테넌트 SaaS 아키텍처 설계 — B2C/B2B 단일 플랫폼, 3단계 데이터 격리

- 6계층 RBAC 권한 체계 — Authority 단위 검증으로 역할 변경 시 코드 수정 최소화
- 모듈 간 Master-Sub 단방향 통신 재설계로 순환 호출/데드락 위험 제거

AI 기반 개발 프로세스 설계

- McDonald System 도입, 적용

비즈니스 임팩트

- **메가존클라우드 사내 교육 플랫폼 도입 검토 대상 선정**
- 메가존 Definition 팀에서 실제 사용 검토 중, 인수인계 완료

Communication Skill

- **Slack** 메신저 기반 소통 + GitHub 연동 실시간 PR 확인 + SonarQube 코드 리뷰
- **Notion**으로 트러블 슈팅, 학습 내용 기록 및 공유
- 1주 단위 스프린트, 일일/주간 회의를 통한 진행사항 공유
- **"아니 근데"** 금지 등 **그라운드 룰** 적용으로 협업 시 불편한 상황 방지

관련 링크

아키텍처 인터랙티브: LMS Architecture

Figma 디자인: MZC LMS Figma 1 | Figma 2

프론트엔드 데모: MZCLMS

팀 노션: MZ 인턴 팀 노션

시연 가이드: 시연 가이드

프로젝트 발표 자료: 발표 슬라이드

▼ **프론트엔드 데모 더미 계정 정보** (비밀번호 공통: `1q2w3e4r!`)

시스템 관리자 (테넌트 없음)

이메일	역할	리다이렉트
`sa@demo.com`	SYSTEM_ADMIN	`/sa/dashboard`

MZC 아카데미 (subdomain: mzc)

이메일	역할	리다이렉트
`ta-mzc@demo.com`	TENANT_ADMIN	`/mzc/ta/dashboard`

이메일	역할	리다이렉트
`co-mzc@demo.com`	OPERATOR	`/mzc/co/dashboard`
`instructor-mzc@demo.com`	INSTRUCTOR	`/mzc/tu/teaching`
`designer-mzc@demo.com`	DESIGNER	`/mzc/tu/teaching`
`user-mzc@demo.com`	USER	`/mzc/tu/b2c`

삼성 러닝센터 (subdomain: samsung)

이메일	역할	리다이렉트
`ta-samsung@demo.com`	TENANT_ADMIN	`/samsung/ta/dashboard`
`co-samsung@demo.com`	OPERATOR	`/samsung/co/dashboard`
`instructor-samsung@demo.com`	INSTRUCTOR	`/samsung/tu/teaching`
`designer-samsung@demo.com`	DESIGNER	`/samsung/tu/teaching`
`user-samsung@demo.com`	USER	`/samsung/tu/b2c`