**🎯 차별화된 기술적 구현 - 해설 강화판 (발표/심사위원용)**

*※ 이 문서는 기능 개념 → 문제 맥락 → 동작 흐름 → 데모/검증 방법 → 보안/성능 주의점 → 확장 아이디어 순으로 설명합니다. 실제 구현된 코드 기준으로 작성했습니다.*

————————————————————————————————————————

# **🔧 백엔드: 엔터프라이즈급 아키텍처 설계**

1. 1) JWT + DB 이중 검증 기반 권한 시스템

**개념 요약**

* 클라이언트가 보낸 JWT의 유효성(서명/만료)을 검사하고, 동시에 DB에서 사용자의 현재 역할/권한을 조회하여 일치하지 않으면 요청을 거부합니다.

**왜 필요한가**

* JWT는 발급 시점의 정보를 담습니다. 이후 사용자의 권한이 변경되었을 때(강등/정지), DB 진실원장과 불일치하면 보안 리스크가 발생합니다.

**동작 흐름**

* ① 요청 → Authorization: Bearer <accessToken>
* ② JwtAuthenticationFilter: 서명/만료 검증, 클레임 파싱(sub/roles)
* ③ DB 조회: User/Role 테이블에서 현재 역할 로드
* ④ 비교/검증: 토큰의 role과 DB의 role이 불일치하면 401/403 반환
* ⑤ SecurityContext에 인증 주입 → 컨트롤러 접근 허용

**데모/검증 방법(30초)**

* 사전 준비: 같은 계정에서 DB role을 바꿔 둡니다(예: STUDENT → SUSPENDED).
* 1) 토큰 발급 후 정상 접근 확인 → 2) DB에서 role 변경 → 3) 동일 토큰으로 접근 시 403 거부 로그 확인

**보안/주의점**

* 토큰 저장소: 로컬스토리지 사용 시 XSS에 주의. HttpOnly 쿠키 + CSRF 전략 고려 가능
* 토큰 회수(로그아웃/강제 만료): 서버 측 denylist/토큰 버전 관리로 즉시 차단

**관련 파일**

* security/service/CustomUserDetailsService.java
* security/jwt/JwtAuthenticationFilter.java

1. 2) N+1 쿼리 문제 해결 + 집계 최적화 🚀

**개념 요약**

* N+1 문제: 부모 N개를 조회한 후, 각 부모마다 자식 데이터를 별도 쿼리로 N번 더 불러오는 비효율 패턴

**왜 필요한가**

* 대시보드처럼 다수 프로젝트/과제를 한 화면에서 볼 때, 쿼리 수가 선형적으로 증가하여 지연이 크게 발생합니다.

**해결 전략**

* ① 배치 조회(findByProject\_IdIn): 한 번의 IN 쿼리로 모든 자식(과제) 로드
* ② Stream 집계(groupingBy, mapToDouble().average()): 메모리 효율적으로 지표 계산
* ③ 필요 시 Fetch Join/@EntityGraph로 연관 로딩 최적화(상황별 선택)

**전/후 비교 코드 스냅샷**

// Before: N+1  
for (Project p : myProjects) {  
 assignmentRepository.findByProject\_Id(p.getId()); // N번 쿼리  
}

// After: Batch + Grouping  
List<Assignment> all = assignmentRepository.findByProject\_IdIn(projectIds);  
Map<Long, List<Assignment>> byProject = all.stream()  
 .collect(Collectors.groupingBy(a -> a.getProject().getId()));

**데모/검증 방법(30초)**

* A/B 실행 로그: 최적화 전/후의 쿼리 수와 응답 시간(예: 312→18쿼리, 1.2s→260ms)을 콘솔/프로파일러 캡처로 제시
* 데이터 결과 동일성 확인: 동일 입력에서 동일 결과가 나오는지 단위테스트 스냅샷 비교

**주의점/확장**

* IN 리스트가 매우 클 때는 페이징/샤딩 고려
* 집계 컬럼만 필요하면 Projection/Native 쿼리로 전송량 축소

**관련 파일**

* service/DashboardService.java
* repository/AssignmentRepository.java

1. 3) Presigned Token 기반 파일 업로드 보안 📁

**개념 요약**

* 서버가 발급한 1회용·유효시간 제한 토큰으로 특정 경로에만 업로드를 허용합니다. 저장은 임시 파일로 받은 뒤 원자적 이동으로 무결성을 보장합니다.

**위협 모델**

* 재사용 공격(Replay), 만료 우회, 경로 탈출(../), 부분 파일로 인한 데이터 손상

**동작 흐름**

* ① 클라이언트: 업로드 사전요청 → ② 서버: 업로드 토큰 발급(jti, exp, path)
* ③ 업로드 호출 시 토큰 소비(remove) + 만료검사(exp)
* ④ 경로 정규화(resolve/normalize)로 루트 밖 접근 차단
* ⑤ temp 파일에 저장 후 ATOMIC\_MOVE로 교체(원자적 쓰기)

**데모/검증 방법(30초)**

* 만료된 토큰/중복 사용 토큰으로 업로드 → 401/403 및 로그 확인
* '../../secret.txt' 같은 키로 시도 → 예외 발생 확인

**주의점/확장**

* 현재 구현: TTL/1회용/경로 보안/원자적 쓰기/사용자 검증
* 추가 권장: 파일 사이즈/Content-Type 제한, 서버 서명·검증, 바이러스 스캔

**관련 파일**

* service/LocalFileStorageService.java, controller/LocalUploadController.java

————————————————————————————————————————

# **⚛️ 프론트엔드: 고급 React 패턴과 상태 관리**

1. 1) useSyncExternalStore 기반 커스텀 상태 관리 🎯

**개념 요약**

* React 18의 구독/스냅샷 모델을 이용해 전역 상태를 최소 보일러플레이트로 관리합니다. Redux 없이도 예측 가능하고 타입 안전한 상태 흐름을 제공합니다.

**왜 필요한가**

* Context 남용 시 리렌더 범위가 커지고 복잡도가 증가합니다. uSES는 구독 단위가 명확하고 성능 측정이 쉽습니다.

**동작 흐름**

* ① store 생성(createStore + subscribe/getSnapshot)
* ② useSyncExternalStore로 구독하여 상태 변경 시 필요한 컴포넌트만 리렌더
* ③ 이벤트 버스와 결합하여 전역으로 변경 사항을 브로드캐스트

**샘플 API**

const state = useSyncExternalStore(store.subscribe, store.getSnapshot);

**데모/검증 방법(30초)**

* 로그인/로그아웃 시 Header/메뉴/보호 라우트가 즉시 반응하는지 확인

**주의점/확장**

* 비동기 로딩 상태, 에러 상태를 일관된 형태로 유지
* SSR을 사용하지 않는 경우 getServerSnapshot 관련 문구 제거(본 문서 반영 완료)

**관련 파일**

* frontend/src/stores/auth.ts

1. 2) 멀티 레이어 이벤트 버스 아키텍처 🚌

**개념 요약**

* 범용 버스(app-bus)와 도메인 특화 버스(schedule-bus)로 이벤트를 분리해 관심사 분리와 타입 안전성을 확보합니다.

**왜 필요한가**

* 컴포넌트 간 Prop Drilling 없이 느슨한 결합으로 상태 변화를 전파하고, 구독 해제로 메모리 누수를 방지합니다.

**동작 흐름**

* ① 상태 변경(auth/notifications) → appBus.emit\* 호출
* ② 구독자(Header/Dropdown 등)가 on\* 핸들러로 즉시 갱신

**데모/검증 방법(30초)**

* 로그인 상태 변경 → Header/알림/프로젝트 스위처 동시 업데이트
* 알림 읽음 처리 → 배지 숫자와 목록 상태 즉시 반영

**주의점/확장**

* 외부 window 이벤트 연동은 현재 구현 범위 밖 → 문서에서 제거(과장 방지)

**관련 파일**

* frontend/src/lib/app-bus.ts, frontend/src/lib/schedule-bus.ts, stores/auth.ts, api/notifications.ts, components/Layout/Header.tsx

1. 3) HTTP 인터셉터 + 자동 토큰 갱신 🔄

**개념 요약**

* API 호출이 401이면 refresh 토큰으로 새 accessToken을 발급받고 원 요청을 자동 재시도하여 무중단 UX를 제공합니다.

**왜 필요한가**

* 세션 만료가 잦은 환경에서도 사용자가 중단을 체감하지 않도록 합니다.

**동작 흐름**

* ① axios 응답 인터셉터에서 401 감지 & \_retry 플래그 확인
* ② refresh 호출 성공 시 헤더 갱신 후 원 요청 재시도
* ③ 실패 시 로그아웃 및 사용자 안내

**주의점/확장**

* 동시 다발 401에 대한 중복 refresh 방지(큐잉/락) 고려 가능
* 토큰 저장 위치에 따른 보안 전략(XSS/CSRF) 점검

**관련 파일**

* frontend/src/api/http.ts

1. 4) 메모이제이션 최적화 패턴 ⚡

**개념 요약**

* 대용량 리스트에 다중 필터/정렬을 적용할 때 useMemo로 결과를 캐시해 불필요 연산과 리렌더를 줄입니다.

**왜 필요한가**

* 프로젝트/알림/사용자 목록처럼 조건이 자주 바뀌는 뷰에서 성능 체감이 큽니다.

**사용 패턴**

* 의존성 배열에 필터 조건을 명확히 기재
* 정렬/검색 함수는 useCallback으로 참조 안정화

**데모/검증 방법(30초)**

* 필터 조건 토글 시 FPS/렌더 횟수 비교(React Profiler 캡처)

**관련 파일**

* frontend/src/pages/Projects/ProjectManagement.tsx

————————————————————————————————————————

# **🔗 추가 연관 파일들**

* Header.tsx (useAuth/알림 구독), StudentDashboard.tsx (인증 상태 연동)
* EventEditor.tsx (schedule-bus), NotificationCenter.tsx (app-bus)
* FilesTab.tsx (토큰 기반 업로드 UI)

# **📎 Appendix (참고)**

* Java 17 Switch Expression: 가독성 향상(핵심 배점과 직접 연관 낮음)
* Optional chaining/Map 그룹핑: 기본 기법(보조 포인트로만 언급)