# SCSAC : Board 시스템 설계서

## 1. 시스템 아키텍처

### 프로젝트 개요 및 전체 구성

SCSAC 내부 게시판 시스템은 SCSA 구성원 간의 지식 공유를 위한 **내부 커뮤니티 플랫폼으로**, 선후배 간 정보 단절을 해소하고 체계적인 지식 축적을 목적으로 개발되었습니다. 이 시스템은 **클라이언트-서버 아키텍처 기반의** 웹 애플리케이션으로, **React 기반 프론트엔드와 Spring Boot 기반 백엔드가** REST API를 통해 통신하는 구조입니다.

### 프런트엔드

* SPA(Single Page Application) 형태로 구성되어 빠른 페이지 전환과 사용자 경험 향상을 도모합니다.
* React를 기반으로 컴포넌트를 분리하여 재사용성과 유지보수를 용이하게 하고, TypeScript를 도입하여 정적 타입 검사를 통해 코드의 안정성과 가독성을 확보하였습니다.
* Redux Toolkit을 통해 로그인 정보, 게시판 목록, 사용자 상태 등 전역 상태를 효율적으로 관리합니다. 로그인, 사용자 프로필, 알림 수 등 다양한 전역 상태를 슬라이스 단위로 나눠 관리하며, 각 슬라이스는 기능별로 명확하게 역할을 분담합니다.
* React Router DOM으로 클라이언트 사이드 라우팅을 구성하여 페이지 새로고침 없이 경로별로 컴포넌트를 렌더링합니다. 예: /login, /category/:id, /article/:id
* Axios로 백엔드와 통신하며, JWT 토큰을 Authorization 헤더에 포함하여 인증합니다. 오류 응답 처리도 포함하여 사용자 경험을 개선합니다.
* 개발환경으로는 Vite를 사용하여 빠른 번들링과 HMR(Hot Module Replacement)을 지원합니다.

### 백엔드

* Spring Boot 프레임워크를 사용하여 RESTful API 서버를 구축하였으며, 도메인 기반의 Controller Service Repository 3계층 아키텍처로 구성되어 유지보수성을 높였습니다.
* Spring Security와 JWT(Json Web Token)를 기반으로 인증 및 권한 관리를 수행합니다. 로그인 시 서버는 사용자 정보를 포함한 JWT를 발급하고, 클라이언트는 이후 모든 요청에 해당 토큰을 Authorization: Bearer 형식으로 포함하여 전송합니다.
* JPA(Hibernate)를 이용해 객체와 관계형 테이블 간 매핑을 처리하며, DB 트랜잭션 관리를 Spring의 AOP 기반으로 처리합니다. Lazy Loading과 Fetch Join 전략을 적절히 혼합하여 성능과 기능을 균형 있게 구현했습니다.
* 서버발송이벤트(Server-Sent Events, SSE)를 도입하여 알림 기능을 실시간으로 제공합니다. 알림은 댓글 작성 등 이벤트 발생 시 서버에서 SSE로 연결된 클라이언트에 전달됩니다.
* OpenAI API 연동을 통해 사용자가 질문을 하면 AI 챗봇이 응답하도록 구현하여 지식형 플랫폼으로서의 부가 가치를 제공합니다.

### 디렉토리 구조

├─ scsac\_fe/scsac-front/ # 프런트엔드 React 프로젝트

│ ├─ src/ # React 소스 (컴포넌트, 페이지, 라우터, Redux store 등)

│ ├─ public/ # 정적 파일 및 HTML 템플릿

│ └─ package.json # 프런트엔드 의존성 및 스크립트 설정

│

├─ scsac\_back/scsac-project/ # 백엔드 Spring Boot 프로젝트

│ ├─ src/main/java/com/scsac/... # 메인 소스

│ ├─ src/main/resources/ # 설정 파일 및 리소스

│ ├─ pom.xml # Maven 프로젝트 설정 (의존성, 빌드 정보)

│ └─ ... # 기타 백엔드 구성 파일

│

├─ DB/ # 데이터베이스 관련 (ERD 다이어그램, 초기 SQL 등)

│

└─ README.md # 프로젝트 소개 및 기술 스택, 기능 설명 (한글)

### 시스템 흐름 요약

1. 사용자가 웹 브라우저에서 프론트엔드에 접근합니다.
2. React 앱이 로드되며, 사용자가 로그인하거나 특정 게시판을 탐색합니다.
3. 사용자 요청은 Axios를 통해 백엔드의 REST API로 전달되며, JWT 토큰을 Authorization 헤더에 포함하여 인증합니다.

4. Spring Boot 백엔드는 Controller → Service → Repository 계층을 거쳐 요청을 처리하며, MySQL과 JPA를 통해 데이터 처리 후 응답을 생성합니다.

5. 프론트엔드는 JSON 응답을 받아 UI에 반영하며, 상태 변화는 Redux를 통해 전역 관리됩니다.

6. 댓글 작성, 게시글 작성 등 특정 이벤트 발생 시, 서버는 SSE를 통해 실시간 알림을 사용자에게 푸시합니다.

7. 사용자가 AI 챗봇에 질문을 전송하면 서버는 OpenAI API에 요청하고, 응답 결과를 반환하여 화면에 출력합니다.

## 2. 데이터베이스 설계

### 핵심 테이블 및 엔티티 관계

* User (users): 사용자 계정 정보 및 권한을 관리. Article, Comment, Alert와 연관.
* Category (categories): 게시판 정보. Article과 연관.
* Article (articles): 게시글 데이터. User, Category, Comment, Alert와 연관.
* Comment (comments): 댓글 데이터. Article, User, Alert와 연관.
* Alert (alerts): 알림 정보. User, Article, Comment와 연관.

### 설계 특징

* 외래키 ON DELETE 제약
* 알림의 읽음 여부를 위한 boolean 필드 포함
* created\_at, updated\_at에 인덱스 부여
* 권한은 User.authority와 Category.access\_level로 관리

## 3. REST API 명세

### 인증 및 사용자 관리 API

* POST /auth/login: JWT 발급
* POST /user: 회원가입
* GET /user/{id}: 사용자 정보 조회
* PUT /user/{id}: 사용자 정보 수정
* DELETE /user/{id}: 사용자 삭제
* GET /user: 전체 사용자 목록 조회 (관리자 전용)

### 게시판(Category) API

* GET /category: 전체 게시판 목록
* GET /category/{id}?sort=&page=: 특정 게시판의 글 목록
* POST /category: 게시판 생성
* PUT /category/{id}: 게시판 수정
* DELETE /category/{id}: 게시판 삭제

### 게시글(Article) API

* GET /article/{id}: 게시글 상세 보기
* POST /article: 게시글 작성
* PUT /article/{id}: 게시글 수정
* DELETE /article/{id}: 게시글 삭제

### 댓글(Comment) API

* GET /comment?articleId=ID: 댓글 목록 조회
* POST /comment: 댓글 작성
* PUT /comment/{id}: 댓글 수정
* DELETE /comment/{id}: 댓글 삭제

### 알림(Alert) API

* GET /alert?id=사용자ID: 알림 목록 조회
* DELETE /alert/{id}: 알림 삭제 또는 읽음 처리

### SSE(Server-Sent Event) API

* GET /sse?token=JWT: 실시간 알림 수신

### AI 챗봇 API

* POST /ai: 질문 요청 및 AI 응답 반환

## 4. 프론트엔드 기술 스택 분석

### React

* 컴포넌트 기반 UI
* Hook 기반 상태 관리 및 효과 처리

### TypeScript

* interface 기반 명시적 타입 정의
* 컴파일 타임 타입 체크

### Redux Toolkit

* 전역 상태 저장소 구성
* createSlice 및 configureStore

### React Router DOM

* SPA 구조의 라우팅 관리
* URL 기반 페이지 컴포넌트 렌더링

### Axios

* HTTP 통신 및 에러 핸들링
* JWT 토큰 포함 요청 구성