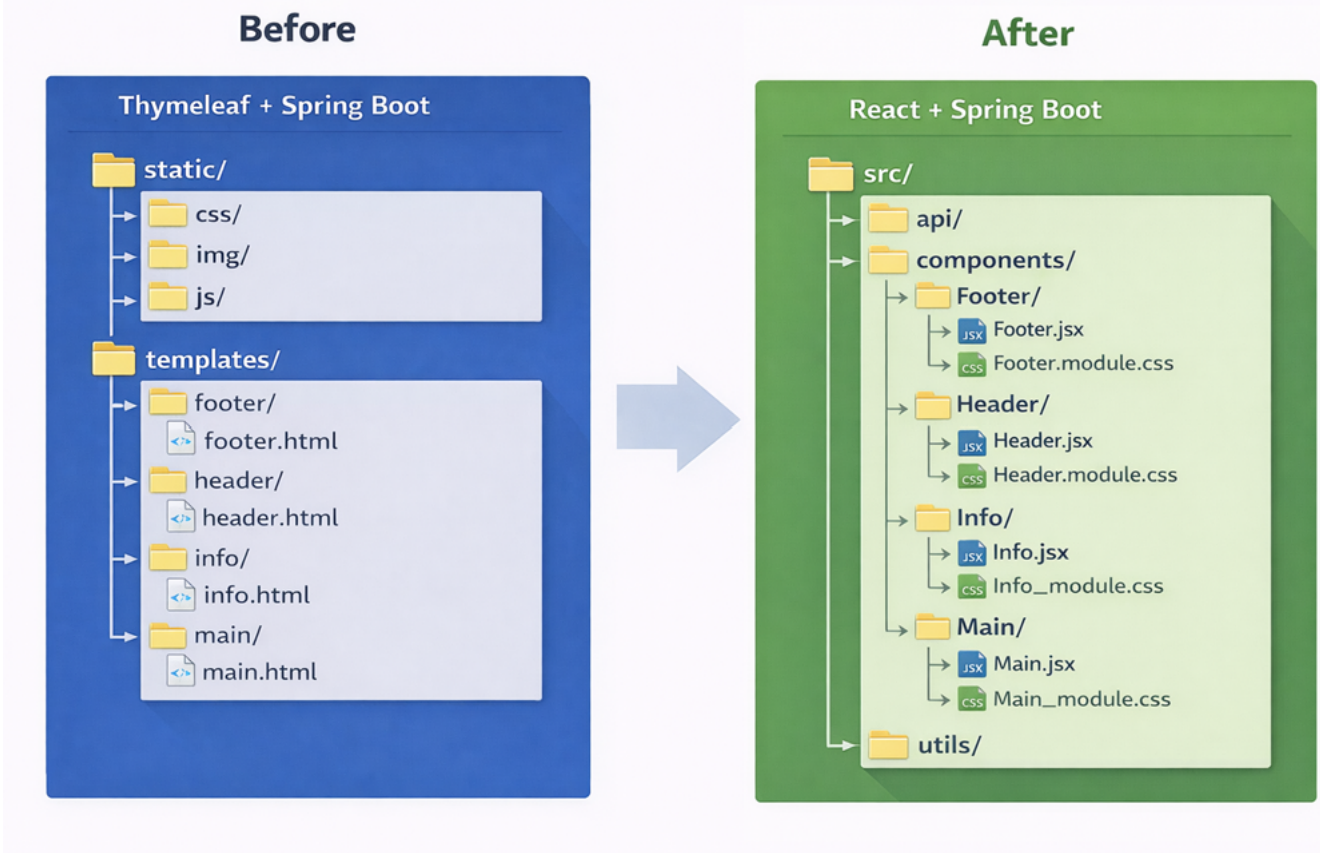
프로젝트 수행이력

프로젝트 명: CALTIZM SPA 프로젝트

git address : https://github.com/tkdals1144/React-migration-project



수행기간	2025.11.25 ~ 2025.12.24

개발 PLATFORM	프로젝트 개요
▶ **Server :** Tomcat v10.1.33 Server	▶ **기존 CALTIZM 프로젝트 아키텍처 리팩터링 (SSR -> CSR)** 기존 Thymeleaf + Spring Boot 기반의 SSR 구조로 구현된 해외 직구 쇼핑몰 CALTZIM 을 React + Spring Boot 기반의 CSR(SPA) 구조로 전환한 프로젝트입니다. 데이터 흐름 개선, 유지보수성 향상, React 실전 감각 향상, 그리고 무뎌진 Spring Boot 코딩 감각 개선을 목표로 진행한 개인 프로젝트입니다.
▶ **DB :** mySQL	
▶ **Client :** HTML5, CSS, React Js	
▶ **H/W :** Windows 11	

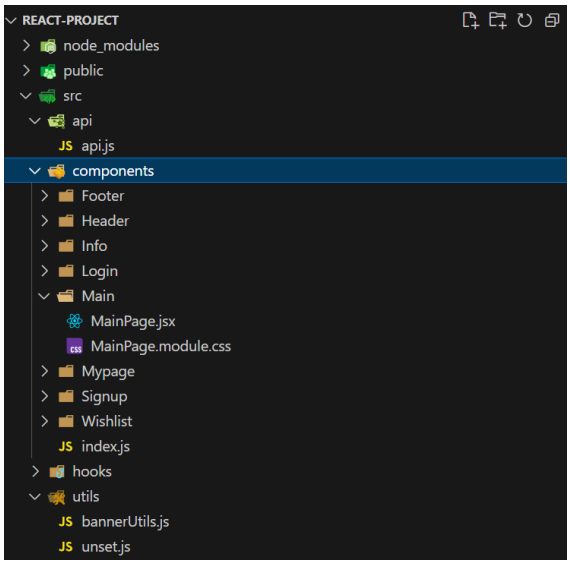
프 로 젝 트 특 징

렌더링 구조 전환

▶ **Before** : 서버에서 HTML 을 생성하여 반환 (Thymeleaf SSR)
▶ **After** : 화면 전환을 클라이언트 라우팅으로 처리 (React CSR)

기능 단위 컴포넌트 설계

▶ 역할별로 프론트 컴포넌트를 분리
▶ 컴포넌트 단위로 모듈화



(디렉토리 구조)

프록시 설정

▶ CORS 에러 방지를 위한 프록시 설정
▶ 프론트엔드와 백엔드 간의 통신을 설정

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

// https://vite.dev/config/
export default defineConfig({
    plugins: [react()],
    server: {
        proxy: {
            "/api": {
                target: "http://localhost:8080",
                changeOrigin: false,
            },
        },
    },
});
```

(프록시 설정 화면)

백엔드 역할 재정립

▶ Controller 책임 변경 (HTML 반환 -> JSON 데이터 반환)
▶ DTO 중심 데이터 전달 구조 확립

```java
@Controller  no usages  tkdals1144
@ResponseBody
public class signupController {

    @Autowired  4 usages
    SignupService service;
    @Autowired  1 usage
    PasswordEncoder passwordEncoder;

    public record AuthResponse(boolean success, String message) { }  4 usages  tkdals1144

    @PostMapping("/signup")  no usages  tkdals1144
    public ResponseEntity<AuthResponse> register(@RequestBody SignupRequestDTO user) {

        // 이메일 중복 검사
        if(service.userCheck(user.getEmail())) {
            return ResponseEntity.badRequest().body(new AuthResponse( success: false,  message: "중복된 이메일이 존재합니다."));
        }
        // 전화번호 중복 검사
        if(!service.telDupCheck(user.getPhone_number())) {
            return ResponseEntity.badRequest().body(new AuthResponse( success: false,  message: "중복된 전화번호가 존재합니다."));
        }
        String encodePassword = passwordEncoder.encode(user.getPassword());
        user.setPassword(encodePassword);

        service.registUser(user);
        service.registUserAddr(user);

        return ResponseEntity.ok().body(new AuthResponse( success: true,  message: "회원가입이 완료되었습니다."));
    }
}
```

(변경된 회원가입 Controller)

보안 및 인증 구조 강화

▶ Spring Security 적용
▶ SecurityConfig 를 생성하여 들어오는 요청에 대한 권한 확인

```java
@Configuration  no usages  tkdals1144
@EnableWebSecurity
public class SecurityConfig {

    @Bean  no usages  tkdals1144
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .csrf( CsrfConfigurer<HttpSecurity> csrf -> csrf.disable()) // REST API라면 비활성화
                .authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
                        .requestMatchers( ...patterns: "/error").permitAll()
                        .requestMatchers( ...patterns: "/api/**").permitAll()
                        .anyRequest().authenticated()
                );
        return http.build();
    }
```

```java
@Bean  no usages  tkdals1144
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new CorsConfiguration();
    configuration.addAllowedOrigin("http://localhost:5173"); // 프론트엔드 주소
    configuration.addAllowedMethod("*");
    configuration.addAllowedHeader("*");
    configuration.setAllowCredentials(true); // 세션/쿠키 허용

    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration( pattern: "/**", configuration);

    return source;
}
```

(SecurityConfig 로직)

UI 밀접 로직 프론트엔드로 이동

▶ UI 와 밀접한 단순 계산, 가공, 분기 로직을 프론트로 이동

▶ 이동한 로직을 utils / hooks 로 분리하여 책임 명확화

```jsx
function useLogin(refreshAuth) {
    const [error, setError] = useState(null);
    const [isLoading, setIsLoading] = useState(false);
    const navigate = useNavigate();

    const login = async (email, password) => {
        setIsLoading(true);
        setError(null);

        try {
            //1. 통신 로직과 await 비동기 처리
            const res = await axios.post("/api/login", { email, password }, { withCredentials: true });

            // 2. 비즈니스 로직 및 상태 처리
            if (res.data.success) {
                await refreshAuth();
                navigate("/main", { replace: true });
            } else {
                setError(res.data.message || "로그인 실패");
                console.log(error);
            }
        } catch (err) {
            console.error(err);
            setError("로그인 서버 오류 발생");
        } finally {
            setIsLoading(false);
        }
    };

    return { login, error, isLoading, setError };
}

export { useLogin };
```

(로그인 Custom Hook)

프로젝트 후기	본 프로젝트는 React 에 대한 실전 감각을 확보하고, 다소 무뎌진 Spring Boot 백엔드 감각을 되살리기 위해 약 한 달간 진행한 개인 프로젝트이다. 기존 Spring Boot + Thymeleaf 기반의 서버 사이드 렌더링(SSR) 구조를 React 기반의 클라이언트 사이드 렌더링(CSR) 구조로 전환하는 것을 주요 목표로 삼았다. 학습이 목적이었기 때문에, 가능한 한 AI 의 도움을 최소화하고 직접 설계 · 구현 · 디버깅 하는 방식으로 프로젝트를 진행하였다. 프로젝트 초기에는 "Thymeleaf 템플릿을 JSX 로 옮기고, HTML 대신 JSON 을 주고받도록 Controller 만 수정하면 비교적 빠르게 마무리할 수 있겠다" 라는 판단을 했다. 그러나 실제로는 렌더링 방식이 바뀌는 것 자체가 단순한 View 변경이 아니라 시스템 전반의 책임 구조를 바꾸는 작업임을 프로젝트를 진행하며 깨닫게 되었다. CSR 구조로 전환하면서 기존에는 서버에서 암묵적으로 처리되던 화면 분기, 데이터 가공, 상태 판단 로직을 프론트엔드로 이전하게 되었고, 해당 과정에서 기존에는 필요 없던 데이터 요구사항들이 새롭게 발생했다. 그 결과 새로운 DTO 를 추가와 기존 Repository, Service, mapper.xml 로직의 확장이 이루어졌다. 이 과정에서 기존 프로젝트의 DTO, Controller, Service 간 책임이 어긋난 부분들을 발견하였고 이로 인해 예상보다 많은 디버깅 시간을 소모하였다. 이를 통하여 **초기 설계의 중요성**을 매우 강하게 체감하게 되었다. React 전환 이후, 프론트엔드 로직을 utils 와 hooks 로 분리하는 과정에서 props drilling 이 깊어지며 한글 IME 입력 조합이 깨지는 버그를 경험하였다. 이 문제를 통하여 zustand 나 redux 와 같은 전역 상태 관리 도구의 필요성을 느끼게 되었다. 프로젝트 후반부에는 보안 강화를 위해 Spring Security 를 도입하였다. 그러나 초기 설정 단계에서 permitAll 처리를 충분히 고려하지 않아 다른 에러 경로에서 403 에러가 발생하는 문제를 겪었다. 이를 해결하며, 보안 역시 기능 구현 이후 덧붙이는 요소가 아니라 **초기 설계 단계부터 함께 고려**되어야 하는 영역임을 다시 한 번 인식하게 되었다. 이 개인 프로젝트를 진행하며 가장 크게 체감한 것은 초기 설계가 프로젝트의 유지보수에 지대한 영향을 끼친다는 점이다. 설계 자체가 미흡하면 작은 변경을 하려고 해도 그것이 연쇄적인 수정으로 이어지고, 디버깅 비용이 기하급수적으로 증가한다. 그렇기에 다음 프로젝트에서는 구현 이전에 설계가 보다 많은 시간을 투자할 계획이다. 구체적으로 팀원들과의 코드 컨벤션을 통하여 네이밍과 디렉터리 규칙을 사전에 정의할 생각이다. 그리고 컴포넌트, 유스케이스 등 다양한 다이어그램을 그려 전체 흐름을 확실히 잡은 상태에서 개발을 진행할 생각이다. 이를 통해 단순히 개발 속도보다는 전체적인 구조의 안정성과 확장성을 우선하는 방향으로 목표로 나아갈 생각이다.

끝까지 읽어 주셔서 감사합니다.