

포트폴리오

박준선

Software Engineer

SURF 동아리 커뮤니티 서비스

카카오, Apple의 웹, 앱 로그인을 지원하고, Redis 기반 토큰 갱신의 동시성 문제를 개선한 동아리 커뮤니티 서비스

Backend

Java 17 · Spring Boot · Spring Security · JPA · MySQL · Redis · Lua · OAuth · OIDC · JWT

<https://portfolio-and-notes.pzs2001926.workers.dev/portfolio/#project-surf>

[GitHub](#) [소셜 로그인 확장 과정](#) [Refresh Token 동시성 개선](#)

프로젝트 소개

SURF는 TAVE 동아리 구성원의 소통과 활동을 지원하는 커뮤니티 서비스입니다. 전체 11명, 백엔드 4명으로 구성된 팀에서 인증과 인가, 금칙어 마스킹을 담당했습니다. 팀은 서비스를 Google Play와 App Store에 출시했습니다.

기존 카카오 웹 로그인을 카카오와 Apple의 웹 로그인과 앱 로그인으로 유연한 구조로 확장하고 토큰 갱신 과정에서 발생하는 경쟁 조건과 정상적인 중복 요청을 처리하는 데 집중했습니다.

카카오 웹 로그인에서 네 가지 인증 흐름으로 확장

기존에는 컨트롤러가 카카오 토큰 교환부터 회원 조회와 저장, JWT 발급, 쿠키 설정까지 처리했습니다. 회원도 카카오 전용 ID로 식별하고 있어 Apple 로그인과 앱 로그인을 추가하려면 인증 구조를 함께 정리해야 했습니다.

로그인 흐름을 조율하는 Usecase와 외부 API 호출, 인증 로그 처리의 책임을 분리했습니다. 회원은 provider + providerId 조합으로 식별하고 복합 Unique 제약을 적용했습니다. 공통 토큰 발급기인 LoginTokenIssuer에는 JWT 발급과 응답 조립을 모았습니다.

요청 형식이 다른 웹과 앱은 로그인 URL을 분리했습니다. Refresh Token은 웹에 HttpOnly 쿠키로, 앱에 JSON 응답으로 전달하도록 구성했습니다.

Apple 인증에서는 Identity Token의 서명과 클레임을 검증하고 웹의 Service ID와 앱의 Bundle ID를 구분했습니다. 웹 콜백 검증에 필요한 state와 nonce는 Redis에 저장하고 Lua로 조회와 삭제를 한 번에 처리해 재사용을 막았습니다.

이렇게 제공자별 인증과 공통 회원 처리, 클라이언트별 토큰 전달을 분리해 카카오와 Apple 각각에 웹과 앱 로그인을 제공하는 네 가지 흐름을 지원했습니다.

같은 Refresh Token이 두 번 소비되는 문제 개선

기존 토큰 회전 로직은 Redis의 조회, 값 비교, 삭제를 각각 실행했습니다. 두 요청이 삭제 전에 같은 값을 읽으면 동일한 토큰으로 모두 검증을 통과할 수 있었습니다.

저장값 비교와 삭제를 Lua 스크립트로 묶어 같은 토큰을 소비하는 요청이 하나만 성공하도록 변경했습니다. 처리 결과도 소비 완료, 불일치, 미존재로 구분하고 인프라 오류는 별도로 처리했습니다.

하지만 토큰을 한 번만 소비하도록 만드는 것만으로는 충분하지 않았습니다. 앱의 여러 API가 동시에 만료 응답을 받으면 정상 사용자도 같은 Refresh Token으로 갱신을 요청할 수 있었습니다. 뒤늦은 요청을 재사용 공격으로 판단하면 다른 기기 세션까지 폐기되는 문제가 남았습니다.

이를 보완하기 위해 **직전 토큰과 새 토큰을 30초 동안 보관하는 유예 키**를 추가했습니다. 유예 키가 저장된 뒤 들어온 중복 요청은 제출한 토큰이 직전 토큰과 일치할 때 같은 새 토큰을 받습니다. 로그아웃할 때는 현재 토큰과 유예 키를 함께 삭제했습니다.

전체 세션을 폐기하는 처리도 KEYS에서 SCAN 커서 순회와 100개 단위 삭제로 변경했습니다.

검증 범위와 남은 과제

다른 기기 세션 보존, 직전 토큰 재시도, 로그아웃 후 유예 키 삭제, 여러 세션 키 삭제를 다루는 Redis Testcontainers 테스트 5개와 인프라 예외 단위 테스트 2개를 추가했습니다.

현재 원자적으로 처리하는 범위는 기존 토큰의 검증과 삭제입니다. 토큰을 소비한 직후 유예 키가 저장되기 전에는 중복 요청이 실패할 수 있습니다. 유예 키 저장 후 현재 토큰 저장이 실패하는 경우도 보완해야 합니다.

토큰 소비와 저장을 하나의 연산으로 묶고 요청의 실행 순서를 제어해 두 요청의 결과를 함께 확인하는 테스트를 후속 과제로 정리했습니다. 30초 유예 시간도 실제 중복 요청 간격을 측정해 조정할 필요가 있습니다.

운영자가 관리할 수 있는 금치어 마스크

게시글, 댓글, 쪽지, 프로필에 입력되는 텍스트를 서버에서 검사하도록 구현했습니다. Aho-Corasick 기반으로 원문과 정규화된 텍스트를 검사하고 탐지한 위치를 원문에 대응시켜 마스크했습니다.

금치어와 허용 표현은 DB에서 관리하고 관리자 API로 수정할 수 있도록 했습니다. 변경된 사전은 스냅샷을 교체하는 방식으로 반영하고 필요할 때 기능을 비활성화할 수 있는 설정도 마련했습니다.

PASSTIVAL 대학 축제 서비스

축제 3일간 방문자 10,005명이 이용한 대학 축제 서비스. 기획과 백엔드 개발, 배포, 운영 및 후속 성능 개선을 담당했습니다.

PM, Backend(Lead)

Java 17 · Spring Boot · JPA · JDBC · MySQL · Apache POI · AWS EC2 · Docker · Nginx · GitHub Actions · AWS OIDC · SSM

<https://portfolio-and-notes.pzs2001926.workers.dev/portfolio/#project-passtival>

[GitHub](#)

프로젝트 소개

PASSTIVAL은 안양대학교 축제 정보, 상품 응모와 추천, 공지사항 및 분실물 제보 등을 제공한 서비스입니다. 전체 12명, 백엔드 3명으로 구성된 팀에서 기획과 백엔드 리드를 맡았습니다.

총학생회와 요구사항을 조율하고 응모, 추천, 관리자 기능, 서버 환경과 배포 흐름을 구현했습니다. 6주간 개발한 서비스를 2025년 9월 축제 기간에 3일간 운영했습니다.

운영 지표	결과
GA 기준 방문자	10,005명
페이지뷰	102,226회
최대 동시 접속자	106명

위 수치는 팀이 함께 운영한 서비스 전체의 지표입니다.

축제 현장에서 사용할 수 있는 응모와 추천 기능

축제 일정을 늦출 수 없었기 때문에, 남은 기간 안에 구현하고 검증할 수 있는 범위를 정해야 했습니다. 총학생회와 기능 우선순위, 응모 방식, 추천 절차를 조율하고 현장에서 직접 추천을 실행할 수 있는 관리자 기능을 개발했습니다.

중복 응모를 방지하고 활동 스탬프에 따른 등급을 적용했습니다. 추천에서는 등급에 비례해 당첨 확률이 달라지는 가중 추천을 구현했습니다. 관리자는 응모와 당첨 현황을 확인하며 무대 현장에서 서비스를 운영할 수 있었습니다.

팀 내에서는 운영 기간과 구현 비용, 검증에 필요한 시간을 기준으로 설계 의견을 조율했습니다. 합의한 구조를 함께 구현할 수 있도록 코드 컨벤션과 Docker 실행 및 배포 가이드도 정리했습니다.

운영 이후 개선: 인증키 10만 행 처리 시간을 85.7% 단축

축제 운영 이후에는 인증키 엑셀 업로드의 대량 저장 경로를 개선했습니다.

기존 코드는 JPA의 `saveAll()`을 사용했지만 IDENTITY 식별자 전략으로 인해 행마다 INSERT가 실행되고 있었습니다. 10만 행을 처리할 때 INSERT도 10만 번 실행됐고 기존 측정에서는 DB 저장이 전체 처리 시간의 약 84%를 차지했습니다.

같은 파일과 파서, 트랜잭션 조건에서 네 가지 저장 방식을 비교했습니다.

저장 방식	전체 처리 시간	DB INSERT 횟수
JPA <code>saveAll()</code>	4,054ms	100,000회

저장 방식	전체 처리 시간	DB INSERT 횟수
JDBC 단건 저장	3,235ms	100,000회
JDBC Batch 1,000행	2,629ms	100,000회
JDBC Batch 1,000행 + SQL 재작성	578ms	100회

생성된 ID를 반환할 필요가 없는 업로드 경로를 JDBC 저장소로 분리하고 JdbcTemplate.batchUpdate()로 1,000행씩 처리했습니다. 여기에 rewriteBatchedStatements=true를 적용해 다중 행 INSERT로 실행되도록 했습니다.

그 결과 로컬 환경의 10만 행 처리 시간이 **4.05초에서 0.58초로 약 85.7% 감소**했습니다. JDBC Batch 호출 여부뿐 아니라 드라이버의 SQL 재작성과 실제 DB 실행 횟수를 함께 확인했습니다.

Apple M3 16GiB, Java 17, MySQL 8.0.46 환경에서 방식별 5회 측정한 중앙값입니다. 서비스 호출부터 파싱과 DB 커밋 후 반환까지 측정했으며 HTTP 업로드 시간은 제외했습니다. 운영 트래픽에 대한 부하 테스트 수치는 아닙니다.

중간 실패 시 파일 전체가 롤백되는지 검증

처리 속도와 함께 일부 데이터만 저장되는 문제도 확인했습니다.

JDBC가 기존 JPA 트랜잭션에 참여하도록 같은 DataSource와 JpaTransactionManager를 사용했습니다. 배치마다 커밋하지 않고 파일 전체가 성공했을 때 한 번만 커밋하도록 구성했습니다.

후속 검증에서 로컬 API 테스트 13개와 MySQL 통합 테스트 4개를 통과했습니다. 2,501행을 INSERT 3회로 빠짐없이 저장했고 두 번째 배치에서 오류를 발생시키면 첫 번째 배치까지 롤백되는 것을 확인했습니다. 업로드 전에 존재하던 데이터와 level 값도 유지됐습니다.

JDBC 적용 범위는 인증키 업로드로 한정했습니다. 다른 JPA 저장소와 기존 트랜잭션을 유지하는 대신, 해당 경로의 SQL과 컬럼 매핑을 직접 관리하는 비용을 받아들였습니다.

정상 응답을 확인한 뒤 다음 서버로 배포

Nginx로 두 대의 Spring 서버에 요청을 분산했습니다. 배포 파이프라인은 SSH 키와 SCP를 사용하던 방식에서 GitHub Actions의 AWS OIDC 인증과 SSM 원격 실행을 사용하는 방식으로 개선했습니다.

커밋 SHA로 지정한 Docker 이미지를 서버별로 순차 배포하고 첫 번째 서버의 헬스체크에서 HTTP 200 응답을 확인한 뒤 두 번째 서버로 진행하도록 구성했습니다.

제한 시간 안에 정상 응답을 받지 못하면 컨테이너 상태와 최근 로그를 출력하고 후속 배포를 중단하도록 했습니다. 배포 스크립트의 실행 완료와 애플리케이션이 요청을 받을 준비가 된 상태를 구분한 것입니다.

이전 버전으로 자동 복구하는 기능과 배포 중 사용자 요청의 연속성 검증은 후속 과제로 남겼습니다.

운영 상황을 공유하는 오류 알림

Spring Cloud OpenFeign과 Discord Webhook으로 서버 예외를 팀에 전달했습니다. 실행 환경, 발생 시간, 요청 URL, 오류 메시지와 주요 스택 트레이스를 함께 보내고 알림 전송 실패가 서비스 처리에 영향을 주지 않도록 예외를 내부에서 처리했습니다.

반복된 Broken pipe 알림은 로그와 요청 경로를 대조해 정상 이용 흐름과 무관한 탐색성 요청 패턴으로 범위를 좁혔습니다. 해당 패턴을 별도 엔드포인트로 리다이렉트해 불필요한 예외 알림과 로그 노이즈를 줄였습니다.

ONECO 부모, 자녀 경제 교육 서비스

자녀가 매일 학습해야 할 경제 콘텐츠 학습 기능 개발과 자녀와 부모 연결 기능을 개발했습니다. CloudFront 도입으로 캐시 적중 시 이미지 응답 시간을 약 65% 줄였습니다.

2025.11 — 2026.01 Backend
Java 17 · Spring Boot · JPA · MySQL · Redis · Spring Scheduler · AWS EC2 · Docker · S3 · CloudFront

<https://portfolio-and-notes.pzs2001926.workers.dev/portfolio/#project-oneco>

[GitHub](#) [이미지 응답 개선 과정](#)

프로젝트 소개

ONECO는 부모와 자녀가 경제 및 금융 에 관한 키워드를 학습하고 퀴즈를 풀고, 관련 기사를 읽어 미션을 수행하는 게이미피케이션 서비스입니다. 자녀는 원하는 커리큘럼을 선택하여 부모에게 제안하고, 성실히 학습하면 그에 맞는 보상을 제안할 수 있습니다. 부모는 자녀의 학습 진행 상황을 검토하고, 코멘트를 남겨줄 수 있습니다.

총 6명으로 구성된 팀으로, 백엔드 로직을 개발했습니다.

결과적으로 TAVE 16기 연합 프로젝트에서 우수상 2등과 인기상을 수상했습니다.

미션의 상태와 전이 규칙을 도메인에 모으기

미션에는 수락과 거절, 진행, 성공과 실패에 따른 상태와 전이 규칙이 필요했습니다. 기능이 늘어나면서 정책이 여러 계층에 흩어지지 않도록 상태와 전이 규칙을 도메인 모델에 모았습니다.

미션 생성 시 가족 관계와 기간을 검증하고 저장은 MissionPersistencePort를 통해 수행하도록 구성했습니다. 실제 DB 접근은 어댑터와 JPA가 담당하도록 분리해 미션 정책과 저장 방식의 책임을 나눴습니다.

시간이 지나면 상태를 바꿔야 하는 미션은 Spring Scheduler로 처리했습니다. 기간이 지난 미션을 조회하고 상태 변경 로직을 호출해 실패 처리하도록 구현했습니다.

초대 코드와 역할을 검증하는 가족 연결

가족 연결에서는 Redis에 저장한 초대 코드를 사용했습니다. 초대를 수락할 때 코드와 동일한 여부, 부모와 자녀의 역할을 검증한 뒤 가족 관계를 연결했습니다.

가족 관계와 미션 데이터는 MySQL에 저장하고 초대 코드 관리는 Redis 어댑터가 담당하도록 구성했습니다. 미션을 생성할 때도 가족 관계를 확인해 연결된 부모와 자녀 사이에서 기능을 사용할 수 있도록 했습니다.

CloudFront 도입으로 이미지 응답 시간 약 65% 단축

기존에는 S3 객체 URL을 DB에 저장하고 클라이언트에 전달했습니다. 같은 이미지가 반복 조회되지만 CDN 캐시를 활용하지 않았고 이미지 주소도 버킷과 리전 구조에 묶여 있었습니다.

반복 조회되는 정적 이미지에 캐시를 적용하기 위해 CloudFront 도입을 제안하고 직접 적용했습니다. 이미지 제공 경로를 CloudFront로 변경하고 DB에서 관리하는 URL도 고정 CDN 도메인을 기준으로 정리했습니다.

Chrome DevTools에서 브라우저 캐시를 비활성화하고 개별 이미지 요청을 비교했습니다.

이미지 제공 방식	응답 시간
S3 직접 제공	843ms

이미지 제공 방식	응답 시간
CloudFront 캐시 적중	298ms

측정한 이미지의 응답 시간이 **843ms에서 298ms로 약 65% 감소**했습니다. CloudFront에 이미지가 캐시된 경우에는 S3에서 다시 가져오지 않고 캐시에서 응답하도록 제공 경로를 바꾼 결과입니다.

브라우저 캐시를 비활성화한 개별 이미지 요청의 비교 결과입니다. CloudFront 캐시 적중 조건의 수치이며 전체 페이지 로딩 시간이나 서비스 전체 평균을 의미하지 않습니다.

실행 환경과 개발 기준 공유

AWS EC2와 Docker를 사용해 서버 실행 환경을 구성하고 개발 규칙과 실행 방법을 문서로 정리해 팀에 공유했습니다.

미션 구현에서는 상태와 정책이 어디에 있어야 하는지, 이미지 개선에서는 어떤 조건에서 측정한 결과인지를 명확히 하는 경험을 쌓았습니다. 기능 구현과 함께 변경 이유와 확인 방법을 팀이 이해할 수 있도록 남기는 데 집중했습니다.