Introduction

안녕하세요!

꾸준히 고민하며 성장하는
신입 백엔드 개발자 **노승준**입니다.

Email : roh100438@gmail.com

Phone : 010 - 5582 - 8626

GitHub : https://github.com/ZeroZoa

About Me

Award[Project : Crewer]

캡스톤디자인 경진대회 | 최우수상
2025.03.01 - 2025.10.31 | 한경국립대학교

Education

한경국립대학교 소프트웨어&서비스컴퓨팅학과
2020.02.28 - 2026.08.20

Certification

정보처리기사 | 2025.09.12 | 한국산업인력공단

Skills

Backend

Java · Spring · JPA
PostgreSQL

- REST API 설계 및 구현
- 인증 · 보안
- JPA / 쿼리 최적화
- 파일 처리 / 외부API

Frontend

Flutter (Main)
React (Sub)

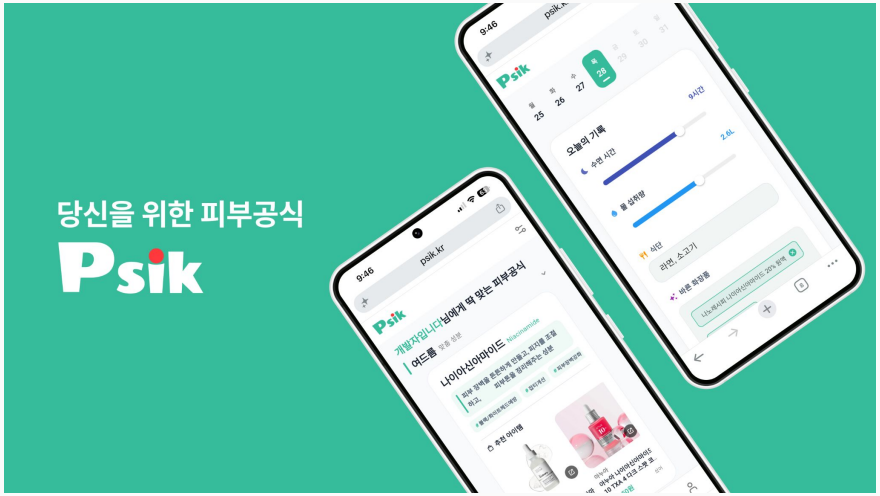
- Provider 상태관리
- go router 라우팅 관리
- Flutter Web CanvasKit 구조 이해 및 로딩 속도 최적화

Tools & DevOps

Docker · Git · GCP

- Docker를 통한 Spring Boot 컨테이너화 및 배포
- 브랜치 전략 (Develop/Main) 수립 및 운영
- CI/CD 파이프라인 구축 (GitHub Actions)
- GCP 기반 클라우드 환경 활용

Project 1



당신을 위한 피부공식 | Psik

psik.kr

Psik의 To

[As-Is] 기존 서비스의 한계

과도한 마케팅 : 흡수율이 낮거나 임상적 근거가 부족한 성분의 과도한 마케팅

획일화된 제품 추천 : 사용자의 피부 상태를 고려하지 않은 성분 추천

01. 검증된 성분 사전 구축

의학 논문 및 객관적 데이터를 통해 실제 효능이 입증된 화장품 성분만을 선별했습니다.

02. AI 피부 진단

AI 기술로 피부 상태를 분석하고 객관적으로 점수화된 지표를 제공합니다.

03. 데일리 루틴 최적화

일기를 통해 기록을 트래킹하여 진정한 맞춤형 스킨케어 솔루션을 완성합니다.

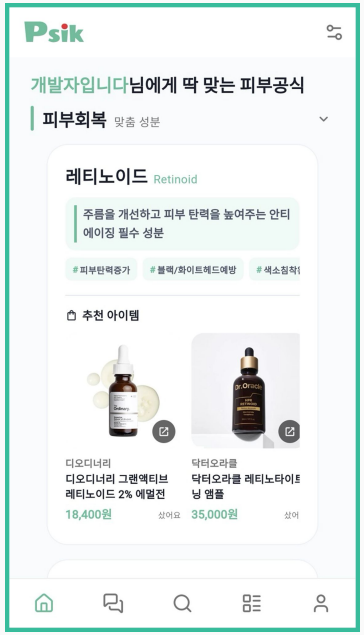
한줄소개 : 당신의 피부 고민을 해결하기 위한 피부 성분 추천 및 AI 피부 분석 서비스

GitHub URL : https://github.com/ZeroZoa/Project_Psik_Backend

기획 및 개발 : 2026.02.01 - 2026.06.01

팀원 : 노승준

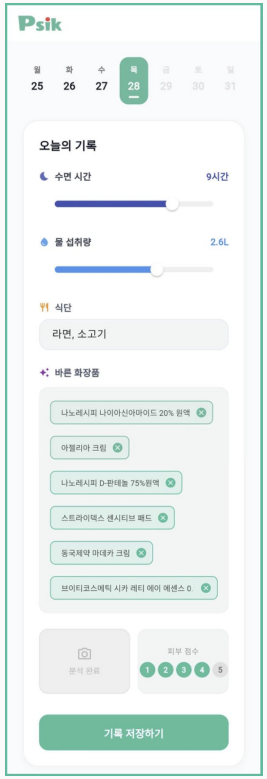
프로젝트에 대하여

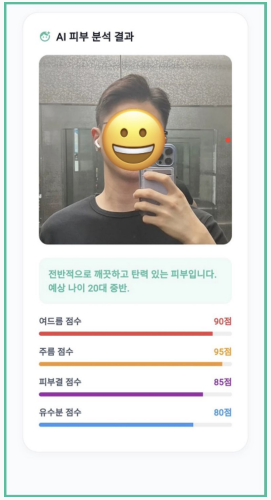


홈화면 [성분 추천]

마이페이지 & 피부 일기 요약

피부 커뮤니티

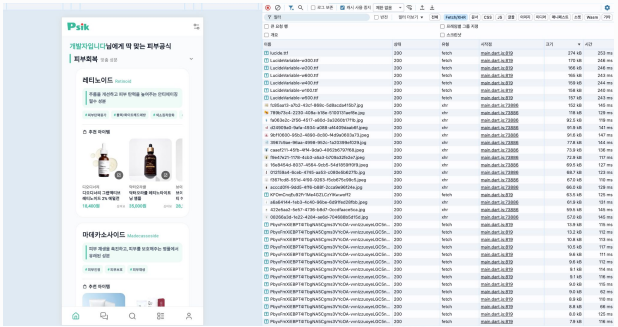




피부 일기 & AI 피부 분석

프로젝트 문제해결 및 배운 점1

문제인식



초기 home_screen.dart에서 Ingredient List를 불러올때 조회 성능에서 문제를 느꼈습니다.

문제해결 전 Psik의 홈화면 네트워크 페이로드 확인 결과입니다.
[동일한 구간의 일부분만 측정, 약 6초 중반대으로 측정되었습니다.]

원인1 및 해결

HTTP 레벨 N + 1문제

1.초기 home_screen은 IngredientController.getIngredients()가 page<IngredientResponse> 즉 요약본을 반환하기 때문에 초기 home_screen에서는 상세한 성분 정보를 조회하기 위해 성분을 순회하며 개별 호출을 반복함

-부가적으로 BatchSize를 추가해 성분 조회를 위한 컬렉션 쿼리 N회를 1회(IN절 배치)로 줄여 해결할 수 있었습니다.

원인2 및 해결

home_screen의 너무 큰 이미지의 네트워크 페이로드

1. 백엔드 프론트엔드에서 이중 압축 문제

-클라이언트측의 프론트엔드단에서 이뤄지는 imageQuality: 80압축을 제거

2. 이미지를 사용자가 입력한 원본 포맷으로 저장

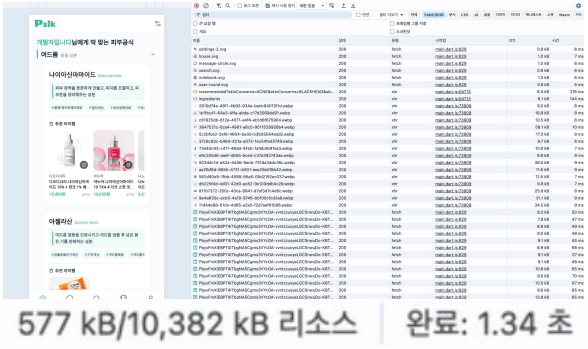
-Scrimage를 통해 문제를 해결했습니다. WebP를 자체 모듈로 지원하며, 불변 객체기반 API로 원본 데이터 오염 없이 안전하게 WebP로 이미지를 저장

3. 표시크기 대비 과도한 저장 해상도

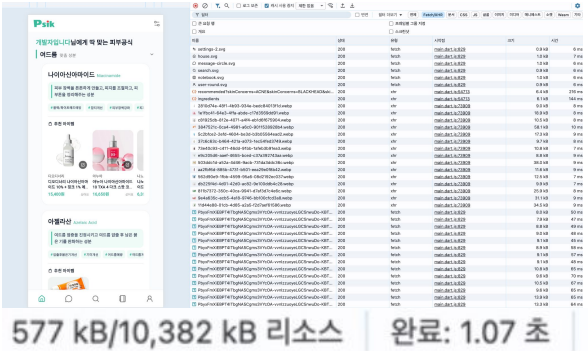
-이미지 저장 사이즈를 600*600으로 수정

프로젝트 문제해결 및 배운 점1

첫 방문 시 네트워크 페이로드[N + 1 해결, 이미지 압축]



재방문 시 네트워크 페이로드[N + 1 해결, 이미지 압축]



문제해결 전

약 6.44s

약 70% 속도 개선

초기 방문시

약 1.34s

재방문시

약 1.07s

최적화 결과 및 분석

- 첫 방문 (캐시 없음): 1.34초 / 재방문 (캐시 있음): 1.07초
- Google Core Web Vitals LCP 기준 **좋음 구간(2.5초 미만)** 달성
- 다만 Flutter Web은 CanvasKit 렌더러를 통해 UI를 그리는 구조로, 1.5MB 규모의 WASM 파일을 로드·파싱하는 비용이 필연적으로 발생합니다. 이는 React/Vue 같은 일반 웹앱과의 근본적인 차이로, 네트워크 최적화만으로는 해결할 수 없는 영역입니다. 서비스를 웹으로 지속한다면 **추후 HTML 렌더러 전환을 추가적으로 검토해볼 수 있습니다.**

프로젝트 문제해결 및 배운 점2

문제 인식 및 원인

[문제상황]

OAuth2 STATELESS 세션 문제로 로그인 과정에서 authorization_request_not_found 예외가 발생하였습니다.

[원인 분석]

JWT 기반 서버는 **SessionCreationPolicy.STATELESS** 설정으로 서버가 세션을 생성하지 않습니다.

따라서 클라이언트에서 로그인을 완료해도 생성한 **state**를 세션에 저장하지 못해 검증 오류가 발생하였습니다.

해결 방안

HttpCookieOAuth2AuthorizationRequestRepository 구현

- 기존의 HTTP 세션 저장 방식 대신, 세션을 대체할 수 있는 **별도의 Repository**를 구현하였습니다.

- 생성된 state 값을 **쿠키에 직렬화하여 저장**하는 방식을 채택하였습니다.

- 인증 검증이 필요할 때마다 쿠키에서 값을 꺼내어 역직렬화 후 검증함으로써 **STATELESS 환경**에서의 인증 문제를 해결하였습니다.

노력한 점

GitHub Actions CI/CD

- develop/main 브랜치 역할 분리
- CI: 자동 빌드 및 코드 검사
- CD: Merge 시 자동 빌드 및 배포

@Slf4j를 통한 로깅

- 배포를 위한 전 서비스 로깅 적용
- 환경 및 중요도별 로그 레벨 분리

트랜잭션 관리

- 클래스 레벨 readOnly=true 적용으로 읽기 성능 최적화
- 쓰기 메서드만 오버라이드하여 쓰기 잠금 최소화

예외처리

- ErrorCode enum으로 도메인별 에러 코드 체계화
- GlobalExceptionHandler를 이용한 통합 예외 처리
- 심각도 기반 로그 레벨 분리 (5xx: error / 4xx: warn)
- 401(미인증) vs 403(권한없음) 명확한 구분 처리

인증/보안

- JWT + Refresh Token Rotation (기존 토큰 자동 무효화)
- IP + UserAgent 기반 토큰 발급으로 기기 추적 보안 강화
- OAuth2 소셜 로그인 (카카오, 구글) 연동
- Secure 쿠키 플래그 환경별 분기 처리 (Local/Prod)
- GCP Secret Manager를 활용한 보안 정보 중앙 관리

시스템 아키텍처 다이어그램

Flutter Web | Spring Boot 3.x | GCP Cloud Run

사용자 (Browser)

Flutter Web (CanvasKit)

Features
- home / auth
- community
- diary / mypage

State & Routing
- Provider (상태 관리)
- GoRouter (ShellRoute)

Network
- Dio (HTTP + JWT Bearer)

정적 파일 서빙 ↔

HTTPS API 호출 →

소셜 로그인 ↔

Firebase Hosting

psik.kr (Frontend URL)

- ✓ Google CDN
- ✓ 정적파일 엣지캐싱

Cloudflare

api.psik.kr (Backend URL)

- ✓ DNS / HTTPS / Proxy

OAuth2 Provider

Kakao | Google

API 트래픽 프록시 →

인가 코드 교환 및 프로필 조회 ↔

GCP Cloud Run

Spring Boot 3.x / Java 21

Controller → Service → Repository → Domain

Spring Security

JWT (Access/Refresh + Rotation)

OAuth2 + HttpCookie STATELESS

min 1 / max 3 인스턴스 | Mem 1Gi / CPU 1 | 동시요청 80

데이터/API 연동 →

Infra & External APIs

Cloud SQL
- PostgreSQL / Flyway

GCS (이미지 저장)
- Flutter 브라우저 직접 조회
- Cloud Run 업로드

Secret Manager
- DB/JWT 등 16개 시크릿 관리

Gemini API
- 피부 분석 / 점수 / 얼굴 감지

CI/CD Pipeline

develop push → CI 빌드 검증 → PR / main merge → CD 자동 배포

Backend: Docker → Artifact Registry → Cloud Run

Frontend: flutter build web → Firebase Hosting

사용 기술 및 라이브러리

백엔드 · Backend

언어 & 프레임워크	Java 21, Spring Boot 3.5	
인증 & 보안	Spring Security, JWT	Access Token(헤더) + Refresh Token(쿠키)
소셜 로그인	Spring OAuth2 Client	Google, Kakao / STATELESS 세션 대응 쿠키 기반 state 저장
AI 피부 분석 & 상담	Gemini API [gemini-2.5-flash]	Spring WebFlux 비동기 호출 및 RAG 구축
ID전략	UUID v7 [uuid-creator]	내부 Long ID + 외부 노출용 UUID 이중 구조
ORM	Spring Data JPA, Hibernate	
Build	Gradle	
Container	Docker	

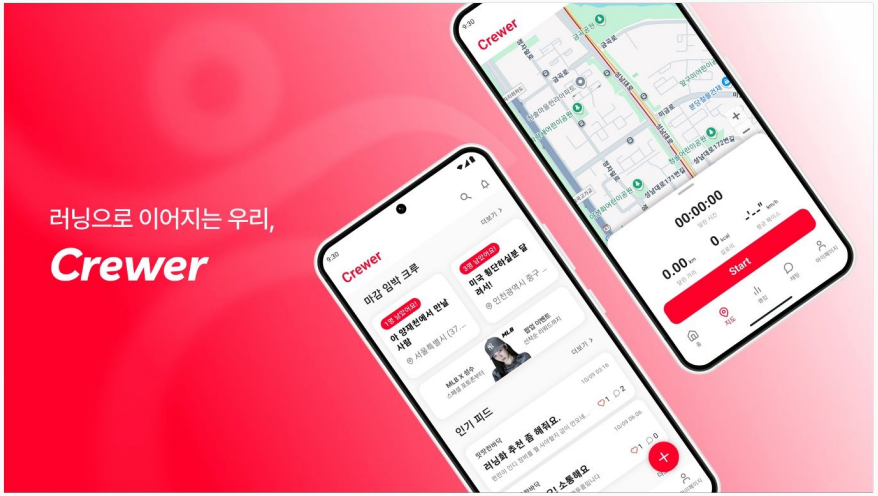
프론트엔드 · Frontend

언어 & 프레임워크	Dart, Flutter Web	
상태관리	Provider	
라우팅	go_router	
HTTP	Dio + AuthInterceptor	토큰 만료 시 reissue를 통해 자동 갱신
토큰 저장	flutter_secure_storage	
차트	fl_chart	스킨 다이어리 시각화
이미지	image_picker, flutter_svg	
Logging	logger	

인프라 · Infra

백엔드 배포	Google Cloud Run	컨테이너 기반, min 1 / max 3 인스턴스 최소수 유지
프론트엔드 배포	Firebase Hosting	CDN, SPA 라우팅
DB	Cloud SQL [PostgreSQL]	Unix 소켓 연결
이미지 저장	Google Cloud Storage	
시크릿 관리	Google Secret Manager	
CI/CD	GitHub Actions	빌드 및 배포 자동화
DNS/프록시	Cloudflare Worker	api.psik.kr → Cloud Run 프록시
컨테이너 저장소	Google Artifact Registry	

Project 2



러닝으로 이어지는 우리 | Crewer

Crewer의 To Be

[As-Is] 기존 서비스의 한계

파편화된 사용자 경험: 러닝 기록 측정과 크루 소통이 분리되어 있어, 사용자가 커뮤니티 활동을 위해 여러 앱을 오가야 하는 불편함 존재.

단조로운 기록 방식: 단순히 거리, 시간, 페이스만을 측정하는 기능에 국한되어 있어, 러닝에 지속적인 흥미를 유발할 색다른 요소가 부족함.

동기부여 및 소속감 저하: 혼자 뛰는 느낌을 주기 쉬우며, 크루원들 간의 즉각적인 교류나 건전한 경쟁을 유도할 시스템이 미흡함.

01. 러닝과 커뮤니티의 조화로운 결합

시간 러닝 측정 기능과 크루 네트워킹을 하나의 앱으로 통합하여 사용자를 유지하고, 이탈을 막기위한 기획설계를 하였습니다.

02. 스케치런을 통한 러닝의 엔터테인먼트화

지도 위에 자신만의 동선을 그리는 스케치런 기능을 도입하여, 러닝을 단순한 운동이 아닌 창의적이고 즐거운 액티비티로 확장하였습니다.

03. 실시간 랭킹 시스템 도입

크루원 간의 데이터를 기반으로 한 랭킹 시스템을 통해, 사용자가 지속적으로 달리게 만드는 동기부여 환경 구축하였습니다.

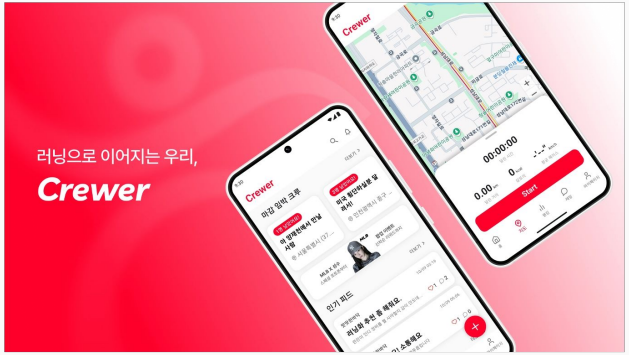
한줄소개 : 러닝으로 이어지는 달리기 커뮤니티 Crewer, 모여서! 경쟁하고! 기록하고!

기획 및 개발 : 2025.03.01 - 2026.10.31

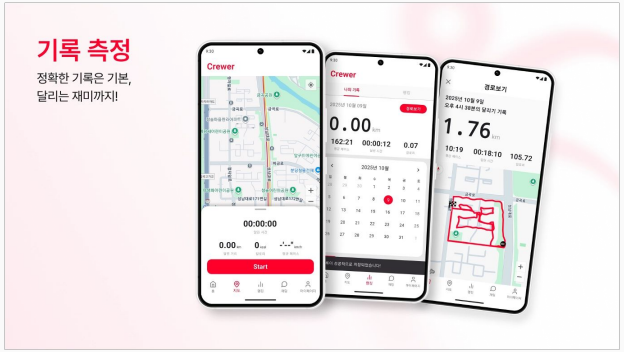
개발자 : 노승준[PM, Full Stack], 박근하[Full Stack], 조근희[Full Stack]

GitHub URL : https://github.com/ZeroZoa/Crewer

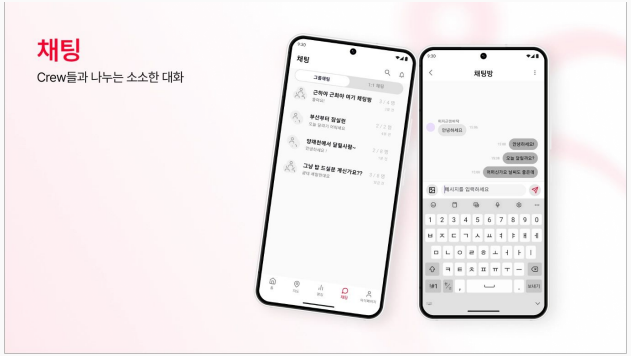
프로젝트에 대하여



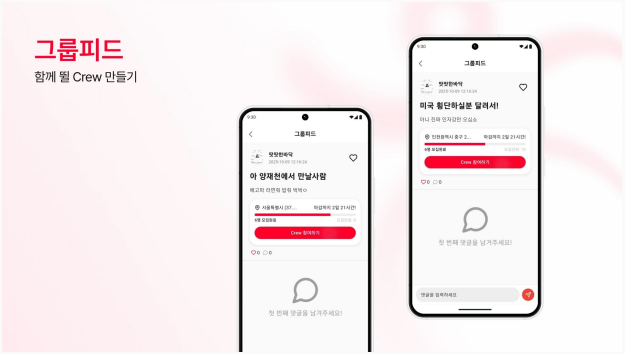
홈화면 및 기록측정



기록측정



채팅



그룹피드

담당 역할 : PM 및 풀스택 개발

1 프로젝트 리드 및 아키텍처 설계

프로젝트 매니징 및 최우수상 수상: PM으로서 전체 개발 일정을 조율하고 팀원들과 기획, 개발, 테스트, 배포 각각의 단계를 계획하고, 진행했습니다. 결과적으로 참가한 다른 팀 대비 좋은 빌드 퀄리티와 서버 안정성을 인정받아 캡스톤디자인 경진대회에서 최우수상을 수상했습니다.

협업 프로세스 구축: GitHub 브랜치 전략을 수립하여 병합 충돌을 최소화하고, Figma와 Notion을 이용해 To do List, 회의록, 트러블 슈팅 과정을 문서화하여 팀원 간 이슈를 공유하고 소통했습니다.

도메인 설계: 서비스 요구사항을 분석하고, 확장성과 유지보수성을 고려하여 전체 서비스의 도메인 모델(엔티티 및 연관관계)을 직접 설계했습니다.

2 인프라 구축 및 배포

AWS 기반 배포 및 형상 관리: AWS EC2 환경에 베이스 서버를 구축하고, Docker를 활용해 DB 및 브로커를 컨테이너화하여 안정적인 운영 및 형상 관리 환경을 구성했습니다.

3 백엔드 및 프론트엔드 통합 개발

JWT 기반 인증/인가 시스템 구현: 클라이언트(Flutter)부터 서버(Spring Security)까지 이어지는 JWT 기반의 로그인 체계를 풀스택으로 설계 및 구현했습니다.

핵심 비즈니스 서비스 구현: 사용자의 앱 내 활동 핵심인 Feed 및 GroupFeed 시스템, 실시간 Running 기록 연동, 사용자 Ranking 시스템의 백엔드 API 설계부터 프론트엔드 UI/UX 구현 및 연동까지 전체 생명주기를 전담하여 개발했습니다.

시스템 아키텍처 다이어그램

Flutter | Java 21 | Docker

Client Layer

Flutter App (iOS / Android)

Presentation

- 크로스 플랫폼 UI/UX 렌더링
- 사용자 인터랙션 처리

State & Routing

- 상태 관리 (State Management)
- 내부 라우팅 제어

Network

- REST API 통신 모듈
- JWT Bearer 인증 헤더 관리

HTTPS
REST API
(JSON)

→
←

Application Layer

Spring Boot 3.x / Java 21

Controller > Service > Repository > Domain

Security & Auth

- Spring Security (필터 체인)
- JWT (Access/Refresh Token)

Data Access Layer

- ✓ Spring Data JPA (Hibernate ORM)
- ✓ Lettuce (Redis Client 연동)

TCP/IP
Query &
Cache

→
←

Infrastructure

Docker Container Env

PostgreSQL

- 메인 RDBMS
- 관계형 비즈니스 데이터 저장
- 엔티티 영속성 관리

Redis

- In-Memory Cache 시스템
- Refresh Token 화이트/블랙리스트
- 빠른 응답이 필요한 데이터 캐싱

협업 파이프라인

Mac OS (Local Dev) → Git Push (Branch) → Pull Request → Code Review → Merge to Main

Backend: Local Docker Compose 실행

Frontend: iOS Simulator / Android Emulator

핵심 기술 및 구현 내용

1 RESTful API 설계 및 JPA 도메인 모델링

-Member, GroupFeed, RunningRecord 등 핵심 도메인 엔티티를 JPA로 설계하고, 연관관계(1:1, 1:N, N:1)를 LAZY 로딩 기반으로 구성
-@NoArgsConstructor(access = PROTECTED)와 도메인 내부 생성자를 통해 무분별한 엔티티 생성을 방지하고 캡슐화 유지
-Spring Data JPA Auditing(@CreatedDate, @LastModifiedDate)으로 생성/수정 시각 자동 관리
-Feed, GroupFeed, Member 등 주요 도메인에 대한 CRUD REST API를 설계하고, @AuthenticationPrincipal로 인증된 사용자 정보를 컨트롤러에서 직접 주입받는 구조 적용

2 WebSocket / STOMP 기반 실시간 채팅 (RabbitMQ 확장 설계)

-Spring WebSocket + STOMP 프로토콜로 그룹 채팅방 / 1:1 채팅방을 분리 설계하여 실시간 채팅 구현
-StompHandler 인터셉터를 구현해 STOMP CONNECT 시점에 JWT를 검증하고 세션에 memberId를 저장함으로써, HTTP 필터 체인과 WebSocket 인증을 구조적으로 분리
-현재는 Spring 내장 SimpleBroker로 동작하며, 다중 서버(Scale-out) 환경 대응을 위해 RabbitMQ STOMP Relay(enableStompBrokerRelay) 전환을 고려하여 spring-boot-starter-amqp 의존성 및 RabbitMQ 컨테이너를 사전 구성해둔 상태

3 Google Maps API와 GPS 기반 러닝 경로 추적 및 랭킹 시스템

-API 키 노출 방지를 위해 Google Maps API Key를 서버에서 관리하고 인증된 클라이언트에만 내려주는 구조 적용
-Flutter 클라이언트에서 수집한 GPS 경로 좌표(latitude, longitude 배열)를 @Embeddable + @ElementCollection으로 DB에 순서 보장하여 저장
-PostgreSQL Native Query에서 CTE + ROW_NUMBER() OVER(PARTITION BY runner_id, distance_category) 윈도우 함수를 활용해 거리 카테고리별 사용자 최고 기록 기반 랭킹 산출

4 Spring Security 기반 JWT 인증 체계 구축

-Spring Security FilterChain을 Stateless로 구성하고, JwtAuthenticationFilter(HTTP)와
-StompHandler(WebSocket) 두 진입점에서 각각 JWT 인증을 처리하도록 인증 체계를 구조적으로 분리
-Redis를 활용한 이메일 인증 코드 발급 및 검증 흐름 구현 (인증 코드 5분 TTL, 인증 완료 토큰 별도 관리)
-BCryptPasswordEncoder를 통한 비밀번호 단방향 암호화 적용
-SecurityConfig에서 공개/인증 필요 엔드포인트를 명시적으로 분리하여 접근 제어 관리

5 DTO Projection을 통한 조회 성능 최적화

-페이징 + 컬렉션 JOIN 시 발생하는 카테시안 곱 문제를 해결하기 위해, 1단계에서 Page<Long>으로 ID만 페이징 조회 후 2단계에서 해당 ID 기반으로 데이터를 조회하는 2-Step 방식 적용
-JPQL Constructor Projection(new DTO(...))으로 엔티티 전체를 로딩하지 않고 필요한 필드만 조회하여 불필요한 데이터 로딩 제거
-상세 조회에서는 LEFT JOIN FETCH로 댓글을 즉시 로딩하고, 좋아요 컬렉션은 @BatchSize(size = 100)으로 IN 쿼리 배치 처리하여 N+1 문제 대응
-최신순 / 인기순(일주일 기준 좋아요) / 마감 임박순 등 다양한 정렬 기준에 각각 최적화된 쿼리 분리 설계

6 Docker 기반 컨테이너 환경 구축 및 AWS EC2 배포

-Multi-stage Dockerfile을 작성하여 빌드 이미지(gradle:8.5-jdk21)와 런타임 이미지 (eclipse-temurin:21-jre)를 분리, 최종 이미지 용량 최소화
-Docker Compose로 Spring Boot 앱 / PostgreSQL / Redis / RabbitMQ를 단일 내부 네트워크 (crewer-net bridge)로 묶고, DB 포트를 외부에 노출하지 않아 보안 구성
-.env 파일로 DB 계정, JWT Secret, API Key 등 민감 정보를 코드에서 분리하여 관리
-AWS EC2에 Docker 환경을 구성하고 컨테이너로 서비스 배포