프로젝트 개발 보고서

LinkFolio

목차

1. 프로젝트 개요

1.1. 프로젝트 소개

『Linkfolio』는 개발자・디자이너 등 전문 직군 구직자를 위한 **개인 브랜딩 및 채용 통합 관리 플랫폼**이다.

현대 채용 시장에서 구직자들은 GitHub, LinkedIn, 기술 블로그 등 다양한 채널에 자신의 이력을 분산하여 관리해야 하는 번거로움이 있다. Linkfolio는 이러한 파편화된 정보를 하나의 세련된 웹 포트폴리오 페이지로 통합(Aggregation)하여, 구직자가 자신의 전문성을 효과적으로 전달할 수 있도록 돕는다.

단순한 링크 모음 서비스를 넘어, 시각적으로 구조화된 포트폴리오 빌더와 기업과의 실시간 커뮤니케이션 기능을 제공함으로써, 구직자에게는 퍼스널 브랜딩의 기회를, 기업에는 검증된 인재를 발굴할 수 있는 양방향 채용 생태계를 제공하는 것을 목표로 한다.

1.2. 추친 배경 및 필요성

1.2.1. 시장 환경 분석

국내 채용 시장은 지속적인 성장세를 보이며, 주요 플랫폼들은 이미 수백만 단위의 MAU를 확보하고 있다. 이는 Linkfolio가 진입할 시장의 규모가 매우 크고, 잠재 고객층이 충분함을 의미한다.

사이트명	전체 이용자 수	MAU	출처
잡코리아	개인 약 1,700만 명, 기업 약 360만	약 761만 명	유저수/MAU
잡플래닛	개인 약 574만 명	약 152만 명	유저수/MAU
사람인	약 1700만 명, 기업 약 120만	약 922만 명	유저수/MAU
크몽	약 400만 명	약 250만명	유저수/MAU
링커리어	비공개	약 200명	-/MAU
원티브	글로벌 약 351만 명, 기업 약 3만	약 61만명	유저수/MAU

[표 1] 채용 사이트 시장 규모

특히 크몽과 같은 프리랜서 시장의 성장세는 '개인 브랜딩의 중요성 증가'를 명확히 보여주며, 구직자가 단순 이력서를 넘어 자신의 브랜드를 효과적으로 표현할 필요성이 커지고 있다.

1.2.2. 문제 인식 (Pain Points)

거대한 시장 규모에도 불구하고, 기존 채용 생태계는 다음과 같은 구조적 한계를 가지고 있다.

- **개인 브랜딩 정보의 파편화 (Fragmentation)**
  구직자의 역량을 증명할 코드(GitHub), 글(블로그), 경력(LinkedIn) 등이 여러 플랫폼에 흩어져 있어, 이를 종합적으로 보여주기 위해 이력서에 수많은 링크를 나열해야 하는 비효율이 발생한다 또한, 포트폴리오가 갱신 될 경우 여러 사이트에서 갱신이 요구되어 갱신이 누락 될 가능성이 높다.
- **기존 Link-in-Bio 서비스의 한계**
  인스타그램 프로필용 링크 서비스 등은 단순한 URL 나열에 그쳐, 프로젝트의 상세 내용이나 기술 스택(Skill Set) 같은 전문적인 정보를 시각적으로 표현하기 어렵다.
- **채용 시장의 단방향성 (Unidirectional)**
  대부분의 채용 플랫폼은 구직자가 공고를 찾아 지원하는 수동적 구조다. 이로 인해 역량 있는 인재가 먼저 자신을 알리거나, 기업이 비공식적으로 접촉(Coffee Chat 등)하기 어려운 문제가 있다.

1.3. 서비스 목표 및 해결 방안

Linkfolio는 위에서 제기된 문제점을 해결하기 위해 다음과 같은 핵심 가치를 제공한다.

- **All-in-One 통합 브랜딩**
  흩어진 링크를 단순 통합하는 것을 넘어, 프로젝트 결과물과 기술 스택을 시각적으로 돋보이게 하는 전문 포트폴리오 템플릿을 제공한다.
- **양방향(Bidirectional) 소통 채널 구축**

기업 채용 담당자가 구직자의 포트폴리오를 보고 직접 제안을 보내거나, 구직자 간에 피드백을 주고받을 수 있는 실시간 커뮤니케이션 기능을 탑재한다.

- **사용자 경험(UX) 중심의 포트폴리오**
  별도의 코딩 없이도 누구나 디자이너가 만든 것 같은 고품질의 웹 페이지를 가질 수 있도록 커스터마이징 기능을 강화한다.

1.4. 경쟁사 분석 및 차별화 전략

1.4.1. 경쟁 서비스 분석

기존 시장의 주요 플레이어들은 각자의 영역에서 강점을 가지고 있으나, '통합 포트폴리오 관리'와 '네트워킹'을 동시에 만족시키는 서비스는 부재하다.

서비스	장점	단점
JOBKOREA(잡코리아)	국내 대표 채용 플랫폼, 방대한 채용 정보 제공	포트폴리오 중심의 브랜딩 기능 부족
wanted(원티드)	AI 추천 기반 매칭, 커리어 성장 콘텐츠 제공	링크 통합 및 개인 브랜딩 기능 미흡
inflearn(인프런)	학습 중심의 역량 강화 플랫폼	채용 및 포트폴리오 관리 기능 부재
잡플래닛	면접, 회사 내 분위기 등의 내부 데이터를 제공	채용 공고보단 회사 정보 파악에 강점
사람인	기업 입장에서 채용 공고를 한 번에 여러 플랫폼 게시, 채용 데이터 혼선 방지, 뛰어난 자동 완성 기능	많은 양의 데이터가 있어 질이 낮은 포트폴리오 다수 존재
크몽	쉽게 게시글을 작성, 작업 가능	전문 채용이 아닌 외주 위주로 전문성이 타 플랫폼에 비해 약함

[표 2] 주요 경쟁 서비스 비교 분석

1.4.2. 차별화 전략

Linkfolio는 경쟁사들이 제공하지 못하는 '소통'과 '커스터마이징'에 집중하여 틈새시장을 공략한다.

- **1:1 실시간 다이렉트 메시지 (DM):**
  이메일이나 서류 전형을 거치지 않고도 플랫폼 내에서 기업과 구직자가 즉시 대화할 수 있어 채용 리드 타임을 획기적으로 단축한다.
- **커뮤니티형 피드백 시스템**
  구직자끼리 서로의 포트폴리오에 댓글과 피드백을 남길 수 있는 기능을 제공하여, 단순한 문서 저장소가 아닌 '함께 성장하는 커리어 커뮤니티'를 지향한다.
- **고도화된 커스터마이징**
  정해진 양식에 텍스트만 채워 넣는 기존 이력서와 달리, 사용자가 원하는 대로 섹션을 배치하고 스타일을 적용할 수 있는 유연한 빌더 기능을 제공한다.

1.5. 비즈니스 모델(BM)

1.5.1. 벤치마킹 모델 분석

안정적인 수익 창출을 위해 기존 성공 사례들의 BM을 분석하였으며, 시장에서 검증된 '광고'와 '구독' 모델을 벤치마킹하였다.

사이트명	BM	기능
잡코리아	기간제 구독제, AD, 서비스 이용료(기업)	광고 배너, 이력서 강조(구직), 공고 상단 노출 및 갱신(구인)
잡플래닛	기간제 구독제(무료/스탠다드/프리미엄), 서비스 이용료(기업)	리뷰 열람, 회사 정보 열람, 프리미엄 데이터 열람(구직), 채용 성공 시, 수수료 제공(구인)
크리에이터 링크	기간제 구독제(무료/베이직/비즈니스/쇼핑)	사이트 제작 기능 변경(저장공간, 다국어 번역, 결제 기능 등)
크몽	사이트 화폐, 수수료	서비스 이용료, 결제 수수료 및 결제망 이용료, 부가세
링커리어	광고, 서비스 이용료(기업)	광고 배너, 패키지
원티드	서비스 이용료(기업)	취업 성공 수수료

[표 3] 레퍼런스 서비스 BM 분석

1.5.2. Linkfolio 수익화 전략

Linkfolio는 초기 사용자 확보를 위해 진입 장벽이 낮은 Freemium(부분 유료화) 전략을 채택하고, 트래픽이

확보된 지면을 활용하여 광고 수익을 창출하는 하이브리드 모델을 구축하였다.

- **타겟 광고 (Advertisement)**

포트폴리오 열람 페이지 및 메인 대시보드 등 트래픽이 집중되는 구간에 배너 광고를 유치한다.
타 사이트 대비 낮은 단가로 서비스 초반 CTR의 위험 부담을 줄이고, CPC를 낮출 수 있도록 한다.

광고	노출 위치	개수	규격	단가
메인 광고	PC/Mobile 메인 상단	5개	1130px * 76px	주 40만 원

[표 4] 광고

- **프리미엄 구독제 (Free & Pro)**

구직 경쟁력을 높여주는 상위 노출 및 편의 기능을 유료화하여 충성 고객을 확보한다.

구분	Free-Tier	Pro-Tier
PR 우선 노출	미제공	공고 상단 고정 + 배너 노출
PR 검색 노출	미제공	검색 상단 고정
PR 카드 양식	기본 양식	PR 카드 커버 제공
이력서 자동 업데이트	미제공	자동 갱신(7일 주기)
강좌 연계	미제공	지원
유지 기간	제한 없음(기본 이용)	28일
가격	무료	월 4,900원

[표 5] 프리미엄 구독제

2. 요구사항 정의

2.1. 기능적 요구사항 (Functional Requirements)

- **인증/인가 (Auth Service)**

ID	요구사항명	상세 내용	중요도
REQ-AUTH-01	자체 회원가입	이메일 인증(Redis 기반 코드 검증)을 거쳐 계정을 생성한다. 가입 시 Transactional Outbox 패턴 과 SAGA를 통해 유저 프로필 생성을 보장한다.	상
REQ-AUTH-02	소셜 로그인	Google, Naver, Kakao OAuth2를 통한 로그인 및 회원가입을 지원한다.	상
REQ-AUTH-03	로그인/로그아웃	JWT(Access/Refresh Token) 기반의 로그인을 처리하며, 로그아웃 시 Refresh Token을 만료시킨다.	상
REQ-AUTH-04	토큰 재발급	Refresh Token Rotation(RTR) 방식을 적용하여 Access Token 만료 시 보안성을 유지하며 토큰을 재발급한다.	상
REQ-AUTH-05	아이디/비밀번호 찾기	이메일 인증을 통해 가입된 아이디를 찾거나 비밀번호를 재설정할 수 있어야 한다.	중

- **사용자 관리 (User Service)**

ID	요구사항명	상세 내용	중요도
REQ-USER-01	내 프로필 조회	로그인한 사용자의 상세 정보(이름, 이메일, 생년월일 등)를 조회한다.	상
REQ-USER-02	내 프로필 수정	사용자의 이름, 생년월일, 성별 등 개인정보를 수정한다. 수정 사항은 타 서비스(Auth, Portfolio, Chat)로 전파되어야 한다.	상

- **포트폴리오 (Portfolio Service)**

ID	요구사항명	상세 내용	중요도
REQ-PORT-01	포트폴리오 관리	사용자는 자신의 포트폴리오(한줄 소개, 기술 스택, 본문 등)를 생성 및 수정할 수 있다. 사용자 프로필 정보는 비정규화하여 자동 동기화된다.	상
REQ-PORT-02	포트폴리오 목록 조회	메인 페이지에서 무한 스크롤(Slice) 방식으로 목록을 조회한다. 직군별 필터링 및 최신순 정렬과 Hacker News 알고리즘 기반의 인기순 정렬 을 지원한다.	상
REQ-PORT-03	포트폴리오 상세 조회	포트폴리오의 상세 내용을 조회한다. Split Caching 전략 을 통해 정적 데이터(본문)와 동적 데이터(조회수, 좋아요)를 분리하여 처리한다.	상
REQ-PORT-04	관심(좋아요) 기능	타인의 포트폴리오에 관심을 표시하거나 취소할 수 있다. 좋아요 수는 Redis에서 실시간 집계 후 DB에 배치 반영된다.	중
REQ-PORT-05	내 관심 목록 조회	사용자가 관심을 표시한 포트폴리오 목록을 별도로 조회할 수 있다.	중

- **커뮤니티 (Community Service)**

ID	요구사항명	상세 내용	중요도
REQ-COMM-01	게시글 관리	QnA, 정보공유, 팀원모집 등 카테고리별 게시글을 작성, 수정, 삭제할 수 있다.	상
REQ-COMM-02	계층형 댓글	게시글에 댓글 및 대댓글(답글)을 작성할 수 있으며, 계층형 구조로 조회된다.	상
REQ-COMM-03	QnA 특화 기능	질문 해결 여부(isSolved)를 표시할 수 있으며, 작성자는 원하는 답변(댓글)을 채택할 수 있다.	중
REQ-COMM-04	팀원 모집 및 지원	모집 상태(OPEN /CLOSED)를 관리하며, 지원자는 '지원하기' 버튼을 통해 작성자에게 즉시 1:1 지원 메시지를 전송할 수 있다.	상
REQ-COMM-05	활동 내역 조회	사용자가 작성한 게시글 목록과 북마크한 게시글 목록을 조회할 수 있다.	중

- **채팅 (Chat Service)**

ID	요구사항명	상세 내용	중요도
REQ-CHAT-01	실시간 1:1 채팅	WebSocket(STOMP)과 MongoDB를 이용하여 대용량 실시간 메시지를 전송 및 저장한다.	상
REQ-CHAT-02	채팅방 목록 조회	참여 중인 채팅방 목록을 조회한다. 상대방의 프로필 정보는 로컬 캐싱된 데이터(chat_user_profile)를 사용하여 N+1 문제 없이 조회한다.	상
REQ-CHAT-03	메시지 상태 관리	상대방의 메시지 읽음 여부(Read Count)를 실시간으로 동기화하며, 상대방이 '입력 중(Typing)'인 상태를 표시한다.	중
REQ-CHAT-04	안 읽은 메시지 배지	사용자가 확인하지 않은 총 메시지 개수를 Redis에서 고속으로 조회하여 네비게이션 바 등에 표시한다.	하

- **고객 지원 (Support Service)**

ID	요구사항명	상세 내용	중요도
REQ-SUPP-01	공지사항 조회	전체 공지사항을 조회하며, 중요 공지(isImportant)는 상단에 고정한다. 캐싱을 통해 조회 부하를 최소화한다.	하
REQ-SUPP-02	FAQ 조회	카테고리별(계정, 포트폴리오 등) 자주 묻는 질문을 조회한다.	하
REQ-SUPP-03	관리자 기능	ADMIN 권한을 가진 사용자는 공지사항 및 FAQ를 등록, 수정, 삭제할 수 있다.	중

2.2. 비기능적 요구사항 (Non-Functional Requirements)

- **보안 (Security)**

  ❐ **Gateway Header Spoofing 방지**

  API Gateway는 외부 요청의 인증 헤더를 검증한 후, 기존 헤더를 제거하고 신뢰할 수 있는 내부 헤더(X-User-Id 등)로 재작성(Re-write)하여 내부 서비스로 전달한다.

  ❐ **WebSocket 보안**

  WebSocket Handshake 단계에서 Gateway가 주입한 헤더를 인터셉트하여 세션 속성에 저장함으로써, 연결 수립 이후에도 STOMP 프로토콜 내에서 보안 컨텍스트(Principal)를 유지한다.

- **성능 (Performance)**

  ❐ **Split Caching Strategy**

  포트폴리오 상세 조회 시, 변경이 적은 본문은 Look-aside (Cache-Aside) 방식으로, 변경이 잦은 통계(조회수/좋아요)는 Write-Back 방식으로 분리하여 캐싱한다.

  ❐ **Batch Processing**

  조회수, 좋아요 수와 같은 빈번한 업데이트는 Redis HyperLogLog 및 Hash를 통해 인메모리에서 처리하고, 스케줄러를 통해 일정 주기로 DB에 일괄 반영하여 Row Lock을 최소화한다.

  ❐ **Global Caching**

  공지사항, FAQ 등 읽기 비중이 높은 데이터는 직렬화 이슈를 해결한 Wrapper Class(CustomPageResponse)를 사용하여 적극적으로 캐싱한다.

- **데이터 일관성 (Consistency)**

  ❐ **Eventual Consistency**

  서비스 간 데이터 동기화(예: 회원가입 시 프로필 생성, 프로필 변경 시 포트폴리오 작성자 정보 갱신)는 Kafka와 CDC(Debezium)를 활용한 결과적 일관성 모델을 따른다.

  ❐ **Transactional Outbox Pattern**

  분산 트랜잭션 시 DB 저장과 이벤트 발행의 원자성을 보장하기 위해 Outbox 테이블을 사용하여 'Dual Write' 문제를 방지한다.

- **확장성 (Scalability)**

  ❐ Stateless Architecture

  모든 마이크로서비스는 세션 상태를 저장하지 않으며(Stateless), Redis를 통한 중앙화된 세션/토큰 관리로 수평 확장을 지원한다.

  ❐ Polyglot Persistence

  데이터의 특성에 맞게 MySQL(관계형 데이터), MongoDB(비정형 대용량 채팅 로그), Redis(캐시 및 고속 카운터)를 적재적소에 활용한다.

  ❐ Hybrid Infra

  Kubernetes 클러스터(App)와 외부 VM(Kafka/DB)을 분리한 하이브리드 구조로 리소스 간섭을 최소화한다.

2.3. 기능 명세서

2.3.1. 기능 요약표

ID	페이지	기능명	목적
P-AUTH-001-01	회원가입	이메일 인증 코드 발송	가입 이메일 검증을 위한 코드 발송 (Redis TTL 관리)
P-AUTH-001-02	회원가입	이메일 인증 코드 검증	발송된 코드 일치 여부 확인
P-AUTH-001-03	회원가입	ID 중복 검사	아이디 사용 가능 여부 확인
P-AUTH-001-04	회원가입	비밀번호 일치 검사	비밀번호 오입력 방지
P-AUTH-001-05	회원가입	자체 회원가입 요청	최종 가입 정보 제출 및 SAGA 트랜잭션 시작
P-AUTH-002-01	로그인	자체 로그인	아이디/비밀번호 인증 및 JWT 토큰 발급
P-AUTH-002-02	로그인	소셜 로그인 (OAuth2)	외부 계정(Google, Naver, Kakao) 연동 및 자동 가입
P-COMM-001-01	공통	토큰 재발급	만료된 Access Token 갱신 및 RTR(Rotation) 적용
P-COMM-001-02	공통	로그아웃	서버 Redis 토큰 삭제 및 클라이언트 쿠키 만료
P-FIND-001-01	아이디찾기	아이디 찾기	실명/이메일 기반 아이디 조회 및 발송
P-FIND-002-01	비번찾기	비밀번호 재설정 (코드 발송)	본인 확인을 위한 인증 코드 발송
P-FIND-002-02	비번찾기	비밀번호 재설정 (코드 검증)	인증 코드 확인 및 권한 획득
P-FIND-002-03	비번찾기	비밀번호 재설정 (변경)	새로운 비밀번호로 변경
P-USER-001-01	마이페이지	내 정보 조회	로그인한 사용자 프로필(캐시 적용) 조회
P-USER-001-02	마이페이지	내 정보 수정	이름, 생년월일, 성별 수정 및 CDC 전파
P-USER-001-03	마이페이지	비밀번호 변경	로그인 상태에서 비밀번호 변경
P-PORT-001-01	메인	포트폴리오 목록 조회	필터/정렬(최신순/인기순) 조건에 따른 목록 조회
P-PORT-002-01	에디터	내 포트폴리오 조회	수정용 데이터(기존 내용) 조회
P-PORT-002-02	에디터	내 포트폴리오 저장	포트폴리오 내용 저장 및 발행(공개)
P-PORT-003-01	상세	포트폴리오 상세 조회	개별 포트폴리오 열람 (Split Caching 적용)
P-PORT-003-02	상세	관심(좋아요) 추가	포트폴리오 스크랩 및 카운트 증가 (Redis Write-Back)
P-PORT-003-03	상세	관심(좋아요) 취소	스크랩 취소 및 카운트 감소
P-USER-002-01	관심목록	내 관심 목록 조회	내가 좋아요한 포트폴리오 모아보기
P-CMTY-001-01	커뮤니티	게시글 작성	QnA, 정보공유, 팀원모집 게시글 작성
P-CMTY-001-02	커뮤니티	게시글 목록 조회	카테고리별 목록 및 검색 (인덱스 최적화)
P-CMTY-001-03	커뮤니티	게시글 상세 조회	본문 및 계층형 댓글 조회
P-CMTY-002-01	상세	댓글/대댓글 작성	게시글에 대한 의견 작성
P-CMTY-002-02	상세	QnA 답변 채택	질문자가 원하는 답변 채택 및 해결 처리
P-CMTY-002-03	상세	팀원 지원하기	모집글 작성자에게 1:1 채팅으로 지원 메시지 전송
P-USER-002-02	활동내역	내 활동 조회	내가 쓴 글 및 북마크한 글 모아보기
P-CHAT-001-01	채팅목록	채팅방 목록 조회	참여 중인 대화방 및 안 읽은 메시지 수 조회
P-CHAT-001-02	채팅목록	전체 안 읽은 수 조회	메인 뱃지용 총 Unread Count 조회
P-CHAT-002-01	채팅방	채팅방 입장/생성	1:1 대화방 개설 또는 입장
P-CHAT-002-02	채팅방	메시지 내역 조회	이전 대화 내용 페이징 로딩 (MongoDB)
P-CHAT-002-03	채팅방	실시간 메시지 전송	텍스트 메시지 발송 및 Pub/Sub 전파
P-CHAT-002-04	채팅방	메시지 읽음 처리	상대방 메시지 확인 및 Read Count 갱신
P-CHAT-002-05	채팅방	입력 중 상태 전송	Typing Indicator 실시간 표시
P-SUPP-001-01	공지사항	공지사항 목록 조회	전체 공지사항 확인 (Global Caching)
P-SUPP-001-02	공지사항	공지사항 상세 조회	공지 내용 열람
P-SUPP-002-01	FAQ	FAQ 목록 조회	자주 묻는 질문 확인 (카테고리별)
P-ADM-001-01	관리자	공지사항 등록/수정/삭제	공지사항 컨텐츠 관리 및 캐시 초기화
P-ADM-002-01	관리자	FAQ 등록/수정/삭제	FAQ 컨텐츠 관리 및 캐시 초기화
P-SYS-001-01	시스템	사용자 프로필 동기화	Kafka CDC를 통한 서비스 간 데이터 일관성 유지
P-SYS-001-02	시스템	SAGA 보상 트랜잭션	분산 트랜잭션 실패 시 롤백 처리

2.3.2. 기능 상세

[1] 인증/인가 (Auth)

[P-AUTH-001-01] 이메일 인증 코드 발송

❒ 목적: 무분별한 가입 방지 및 본인 확인

❒ 업무 규칙:

- 이미 가입된 이메일(로컬/소셜)은 발송이 제한된다.
- 인증 코드는 6자리 난수이며, Redis에 3분간 저장된다 (TTL).

❒ 예외 처리:EMAIL_DUPLICATION (중복 이메일), ALREADY_SOCIAL (소셜 가입됨)

❒ 기타: EmailVerificationService 사용

❒ 시나리오:

1. 사용자는 이메일을 입력하고 '인증' 버튼을 클릭한다.
2. 시스템은 이메일 중복 여부를 확인한다.
3. 중복이 아니면 6자리 난수 코드를 생성한다.
4. 생성된 코드를 Redis에 저장하고 이메일로 발송한다.

[P-AUTH-001-02] 이메일 인증 코드 검증

❒ 목적: 발송된 코드 유효성 확인

❒ 업무 규칙: Redis에 저장된 코드와 일치해야 하며, 유효시간(3분) 이내여야 한다. 검증 성공 시 '인증 완료' 상태(VE:{email})를 Redis에 임시 저장한다.

❒ 예외 처리:INVALID_CODE (코드 불일치), EXPIRED_CODE (시간 만료)

❒ 시나리오:

1. 사용자는 수신한 코드를 입력하고 '확인' 버튼을 클릭한다.
2. 시스템은 Redis에 저장된 코드와 비교한다.
3. 일치하면 '인증 완료' 상태를 Redis에 저장한다.

[P-AUTH-001-03] ID 중복 검사

❒ 목적: 아이디 유일성 보장

❒ 업무 규칙: AuthUserRepository를 조회하여 중복된 username이 있는지 확인한다.

❒ 예외 처리: USERNAME_DUPLICATION

❒ 시나리오:

1. 사용자가 ID를 입력하면 실시간 혹은 버튼 클릭으로 중복 여부를 응답받는다.

[P-AUTH-001-04] 비밀번호 일치 검사

❒ 목적: 오입력 방지

❒ 업무 규칙: 비밀번호와 비밀번호 확인 필드의 값이 정확히 일치해야 한다.

❒ 예외 처리: PASSWORD_MISMATCH

[P-AUTH-001-05] 자체 회원가입 요청

❒ 목적: 계정 생성 및 프로필 생성 트리거

❒ 업무 규칙:

- 이메일 인증 완료 상태(VE:{email})가 Redis에 존재해야 한다.
- 계정은 PENDING 상태로 생성되며, Outbox 테이블에 이벤트를 발행하여 SAGA 트랜잭션을 시작한다.

❒ 시나리오:

1. 모든 정보 입력 후 가입 요청.
2. Auth DB 저장 및 Kafka 이벤트 발행(via Debezium).
3. User Service가 이벤트를 받아 프로필을 생성한다.

[P-AUTH-002-01] 자체 로그인

❒ 목적: 인증 및 JWT 발급
❒ 업무 규칙: username으로 조회한 계정의 상태가 COMPLETED여야 한다 (SAGA 완료 계정). / 소셜 가입 계정은 ID/PW 로그인이 불가능하다.
❒ 예외 처리: USER_NOT_FOUND (계정 없음/비밀번호 불일치/미완료 계정)
❒ 결과: Access Token(Body), Refresh Token(HttpOnly Cookie) 반환.
❒ 시나리오:
1. 사용자는 아이디와 비밀번호를 입력하고 로그인을 시도한다.
2. 시스템은 계정 존재 여부와 비밀번호 일치를 검증한다.
3. 계정 상태가 'COMPLETED'인지 확인한다.
4. 성공 시 Access Token과 Refresh Token을 발급한다.

[P-AUTH-002-02] 소셜 로그인 (OAuth2)
❒ 목적: 외부 계정 연동 및 간편 가입
❒ 업무 규칙:
- Google, Naver, Kakao 지원
- RedisBasedAuthorizationRequestRepository를 사용하여 state를 관리(Stateless)한다
- 신규 가입 시 자동으로 SAGA 트랜잭션을 시작한다.
❒ 예외 처리: EMAIL_ALREADY_REGISTERED_AS_SOCIAL (타 소셜로 이미 가입됨)

[P-COMM-001-01] 토큰 재발급
❒ 업무 규칙: Refresh Token 탈취 감지 시 모든 토큰을 무효화하며, 재발급 시 Refresh Token도 교체된다 (RTR).
❒ 예외 처리: 401 Unauthorized (토큰 없음/만료/불일치)
❒ 시나리오:
1. 클라이언트는 401 만료 응답을 감지한다.
2. 저장된 Refresh Token(Cookie)으로 재발급 API를 호출한다.
3. 시스템은 토큰을 검증하고 새로운 Access/Refresh Token을 발급한다.

[P-COMM-001-02] 로그아웃
❒ 목적: 세션 종료
❒ 업무 규칙: Redis에서 해당 유저의 Refresh Token을 삭제하고, 클라이언트 쿠키의 유효기간을 0으로 설정한다.
❒ 시나리오: 로그아웃 요청 시 서버와 클라이언트 양쪽의 인증 수단을 무효화한다.

[P-FIND-001-01] 아이디 찾기
❒ 목적: 분실 아이디 조회
❒ 업무 규칙: 실명(name)과 이메일(email)이 일치하는 LOCAL 계정을 조회한다.
❒ 예외 처리: USER_NOT_FOUND_BY_NAME_AND_EMAIL

[P-FIND-002-01] ~ [03] 비밀번호 재설정
❒ 목적: 비밀번호 분실 시 재설정
❒ 프로세스:
1. 코드 발송: ID(이메일) 기준으로 코드를 발송한다 (PW_RESET:{email} 저장).
2. 코드 검증: 코드 일치 시 PW_VERIFIED:{email} 상태를 Redis에 저장한다 (6분 유효).
3. 변경: PW_VERIFIED 상태가 확인되면 비밀번호를 변경하고 상태를 삭제한다.

[2] 사용자 관리 (User)

[P-USER-001-01] 내 정보 조회

❒ 목적: 프로필 확인
❒ 업무 규칙: Redis 캐시(user:profile:{id})를 우선 조회하고, 없으면 DB 조회 후 캐싱한다.
❒ 시나리오: 마이페이지 접속 시 기본 정보(이름, 이메일 등)를 로드한다.

[P-USER-001-02] 내 정보 수정
❒ 업무 규칙: 이메일, 아이디는 수정 불가하며, 수정된 정보는 타 서비스로 자동 동기화된다.
❒ 예외 처리: INTERNAL_SERVER_ERROR (DB 처리 오류)
❒ 기타: CDC(Debezium) 기반 데이터 Fan-out

[P-USER-001-03] 비밀번호 변경
❒ 목적: 로그인 상태에서의 비밀번호 변경
❒ 업무 규칙:현재 비밀번호가 일치해야 변경 가능하다.

[3] 포트폴리오 (Portfolio)

[P-PORT-001-01] 포트폴리오 목록 조회
❒ 업무 규칙: '발행(isPublished=true)'된 포트폴리오만 노출되며, 페이징(Slice) 방식을 사용한다.
❒ 예외 처리: 데이터 없음 (빈 목록 반환)
❒ 기타: QueryDSL 동적 쿼리 사용
❒ 시나리오:
1. 사용자는 직군 필터 또는 정렬(최신순/인기순)을 선택한다.
2. 시스템은 조건에 맞는 포트폴리오 카드를 조회하여 반환한다.

[P-PORT-002-01] 내 포트폴리오 조회 (수정용)
❒ 목적: 에디터 진입 시 데이터 로드
❒ 업무 규칙: userId로 조회하며, 데이터가 없어도 기본 객체를 반환하거나 예외 처리한다.

[P-PORT-002-02] 내 포트폴리오 저장
❒ 업무 규칙: 1인당 1개의 포트폴리오만 생성 가능하며, 태그는 최대 4개까지 저장된다.
❒ 예외 처리: PORTFOLIO_NOT_FOUND (사용자 정보 캐시 누락)
❒ 기타:
❒ 시나리오:
1. 사용자는 내용(사진, 소개, 마크다운 등)을 입력하고 저장한다.
2. 시스템은 정보를 DB에 저장한다.
3. 최초 저장 시 상태를 '발행(Published)'으로 변경한다.

[P-PORT-003-01] 포트폴리오 상세 조회
❒ 목적: 본문 열람
❒ 업무 규칙: Split Caching 적용
- 본문: Redis 캐시(portfolio:details:{id}) 조회.
- 통계(조회수): Redis Hash(portfolio:stats:{id}) INCR 연산 및 배치용 키에 누적.
- 좋아요 여부: 로그인 시 DB 조회하여 병합.

[P-PORT-003-02/03] 관심(좋아요) 추가/취소
❒ 업무 규칙: 중복 좋아요는 불가능하며(DB Unique), 본인 포트폴리오는 좋아요 할 수 없다.
❒ 시나리오:
1. 사용자는 포트폴리오 상세 화면에서 좋아요 버튼을 클릭한다.
2. 시스템은 관계 테이블에 데이터를 저장한다.
3. 해당 포트폴리오의 좋아요 카운트를 증가시킨다.

[P-USER-002-01] 내 관심 목록 조회

❒ 목적: 북마크 모아보기

❒ 업무 규칙: 내가 좋아요를 누른 포트폴리오를 최신순으로 조회한다.

[4] 커뮤니티 (Community Service)

[P-CMTY-001-01] 게시글 작성

❒ 목적: 콘텐츠 생성

❒ 업무 규칙: QNA, INFO, RECRUIT 카테고리 중 하나를 선택하여 저장한다.

[P-CMTY-001-02] 게시글 목록 조회

❒ 목적: 정보 탐색

❒ 업무 규칙:카테고리, 해결 여부(isSolved) 등으로 필터링하며, 작성자 정보를 포함(Join)하여 조회한다. Redis 캐싱 적용 시 CustomPageResponse 래퍼를 사용한다.

[P-CMTY-001-03] 게시글 상세 조회

❒ 목적: 내용 및 댓글 열람

❒ 업무 규칙: 본문은 캐싱, 조회수 등은 Redis 실시간 카운팅. 댓글은 계층형 구조(Parent-Child)로 변환하여 반환한다.

[P-CMTY-002-01] 댓글/대댓글 작성

❒ 목적: 소통

❒ 업무 규칙: parentId 유무에 따라 댓글/대댓글로 구분 저장. 댓글 수 카운트를 Redis에서 증가시킨다

[P-CMTY-002-02] QnA 답변 채택

❒ 목적: 질문 해결

❒ 업무 규칙: QNA 카테고리 작성자만 가능. 채택 시 댓글의 isAccepted와 게시글의 isSolved가 true로 변경된다.

[P-CMTY-002-03] 팀원 지원하기

❒ 목적: 모집글 지원

❒ 업무 규칙: RECRUIT 카테고리 &OPEN 상태일 때만 가능. Feign Client를 통해 Chat Service로 시스템 메시지를 발송한다.

[P-USER-002-02] 활동 내역 조회

❒ 목적: 마이페이지 활동 관리

❒ 업무 규칙: 내가 쓴 글(findMyPosts) 및 북마크한 글(findMyBookmarkedPosts)을 페이징 조회한다.

[5] 채팅 (Chat)

[P-CHAT-001-01] 채팅방 목록 조회

❒ 업무 규칙: 최근 메시지가 온 순서대로 정렬되며, 상대방 정보는 로컬 캐시(ChatUserProfile)를 사용한다.

❒ 기타: MongoDB Aggregation

❒ 시나리오:

1. 사용자는 채팅 메뉴에 진입한다.
2. 시스템은 내가 속한 방 목록, 상대방 정보, 안 읽은 메시지 수를 조회한다.

[P-CHAT-001-02] 전체 안 읽은 수 조회

❒ 목적: 알림 배지

❒ 업무 규칙: Redis의 USER_TOTAL_UNREAD:{userId} 키 값을 조회하여 반환한다.

[P-CHAT-002-01] 채팅방 입장/생성
❒ 목적: 대화 시작
❒ 업무 규칙: 두 사용자 ID(minId, maxId)를 기준으로 유니크한 방을 조회하거나 생성한다.

[P-CHAT-002-02] 메시지 내역 조회
❒ 목적: 이전 대화 확인
❒ 업무 규칙: MongoDB chat_message 컬렉션에서 roomId 기준 최신순으로 페이징 조회한다.

[P-CHAT-002-03] 실시간 메시지 전송
❒ 업무 규칙: 전송 시 상대방의 '안 읽은 메시지 수'를 1 증가시키고, 나의 카운트는 0으로 초기화한다.
❒ 예외 처리: 웹소켓 연결 끊김 (재연결 시도)
❒ 기타: WebSocket STOMP 프로토콜
❒ 시나리오:
1. 사용자는 메시지를 입력하고 전송한다.
2. 시스템은 메시지를 DB에 저장한다.
3. Redis Pub/Sub을 통해 상대방이 접속한 서버로 메시지를 전달한다.
4. 상대방에게 실시간으로 메시지가 표시된다.

[P-CHAT-002-04] 메시지 읽음 처리
❒ 목적: 수신 확인
❒ 업무 규칙: 채팅방 입장 시 호출. unreadCount를 0으로 초기화하고, 상대방에게 READ 타입 메시지를 보내 숫자 1을 지우게 한다.

[P-CHAT-002-05] 입력 중 상태 전송
❒ 목적: UX 향상
❒ 업무 규칙: DB 저장 없이 Redis Pub/Sub을 통해 TYPING 타입 메시지만 실시간 전파한다.

[6] 관리자/지원 (Support/Admin)
[P-SUPP-001-01] 공지사항 목록 조회
❒ 업무 규칙: 모든 사용자가 조회 가능하다.
❒ 시나리오:
1. 사용자는 공지사항 메뉴에 접속한다.
2. 시스템은 중요 공지를 상단에, 나머지를 최신순으로 정렬하여 보여준다.

[P-SUPP-001-02] 공지사항 상세 조회
❒ 목적: 공지 상세 확인
❒ 업무 규칙: noticeDetail:{id} 키로 캐싱된다.

[P-SUPP-002-01] FAQ 목록 조회
❒ 목적: 자주 묻는 질문
❒ 업무 규칙: 카테고리별(ACCOUNT, PAYMENT 등)로 캐싱하여 조회한다 (faqs:{category}).

[P-ADM-001-01 / 00201] 관리자 기능
❒ 목적: 콘텐츠 관리
❒ 업무 규칙: ADMIN 권한을 가진 사용자만 접근 가능하다.
❒ 예외 처리: 403 Forbidden (권한 없음), INVALID_INPUT_VALUE (입력값 누락)

❒ 시나리오:

1. 관리자는 관리자 페이지에 접속한다.
2. 제목, 내용, 중요 여부를 입력하고 등록/수정한다.
3. 필요 시 게시물을 삭제한다.

[6] 시스템 (System)

[P-SYS-001-01] 사용자 프로필 동기화

❒ 업무 규칙: 비동기 처리로 인해 일시적인 데이터 불일치(Eventual Consistency)가 발생할 수 있다.

❒ 기타: Kafka Consumer

❒ 시나리오:

1. User Service에서 사용자 정보가 변경된다.
2. CDC(Debezium)가 변경 사항을 감지하여 Kafka 이벤트를 발행한다.
3. 각 서비스는 이벤트를 수신하여 로컬 DB(캐시)를 갱신한다.

[P-SYS-001-02] SAGA 보상 트랜잭션

❒ 업무 규칙: 원자성이 보장되어야 한다.

❒ 기타: Kafka Consumer

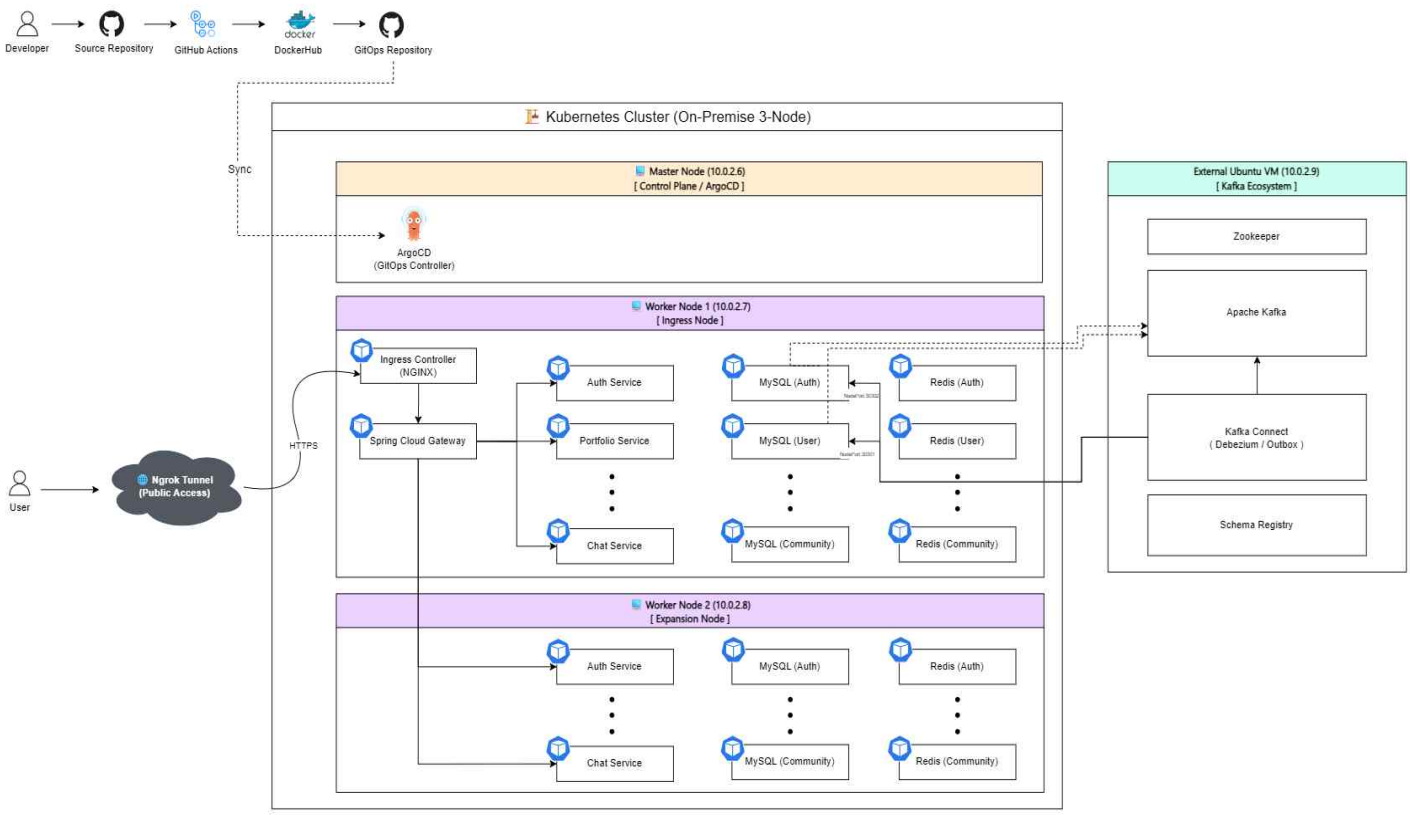
❒ 시나리오:

1. User Service에서 프로필 생성 중 오류가 발생한다.
2. 실패 이벤트를 Kafka로 발행한다.
3. Auth Service가 이를 수신하여 계정 상태를 'CANCELLED'로 변경한다.

3. 시스템 아키텍처

3.1. 서비스 간 구성도

LinkFolio는 On-Premise Kubernetes 환경과 외부 메시징 인프라가 결합된 하이브리드 아키텍처로 구성되어 있다.



- **인프라 토폴로지 (Infrastructure Topology)**
  - ❐ Hybrid Structure

    애플리케이션 실행 환경(Kubernetes Cluster)과 메시징 처리 환경(External VM)을 물리적으로 분리하여, 대용량 메시지 처리 시 발생할 수 있는 리소스 간섭을 최소화하고 안정성을 확보한다.
  - ❐ Kubernetes Cluster (On-Premise)

    1개의 Master Node와 2개의 Worker Node로 구성된 자체 클러스터를 구축하여 운영한다. 모든 마이크로서비스와 데이터베이스는 이 클러스터 내에서 Pod 형태로 관리된다.

- **서비스 계층 (Service Layer)**
  - ❐ Ingress &Gateway

    외부의 모든 요청은 NGINX Ingress Controller를 통해 클러스터로 진입하며, Spring Cloud Gateway가 이를 받아 인증/인가(JWT 검증) 및 라우팅을 수행한다.
  - ❐ Microservices

    업무 도메인별로 분리된 6개의 마이크로서비스(Auth, User, Portfolio, Community, Chat, Support)가 독립적으로 배포되어 운영된다. 각 서비스는 Stateless하게 설계되어 수평 확장이 가능하다.

- **데이터 계층 (Data Persistence Layer)**
  - ❐ Polyglot Persistence: 서비스의 특성에 따라 최적의 저장소를 선택하여 사용한다.
    - MySQL: 회원 정보, 게시글 등 정형 데이터의 무결성 보장 (Auth, User, Community 등).
    - MongoDB: 대용량 실시간 채팅 로그의 고속 쓰기 및 조회 (Chat Service).
    - Redis: 세션 관리(Refresh Token), 조회수 집계, Pub/Sub 메시징 브로커 역할 수행

- **이벤트 및 메시징 계층 (Event &Messaging Layer)**
  - ❐ Apache Kafka Ecosystem

    서비스 간 결합도를 낮추기 위해 Kafka를 도입했습니다. 외부의 독립된 Ubuntu VM에서 운영되며, Zookeeper, Kafka Connect, Schema Registry로 구성된 완전한 이벤트 스트리밍 파이프라인을 제공한다.
  - ❐ CDC (Change Data Capture)

    Debezium을 활용하여 DB의 변경 사항(Transaction Log)을 실시간으로 감지하고 Kafka로 전파하여, 서비스 간 데이터 일관성(Eventual Consistency)을 유지한다.

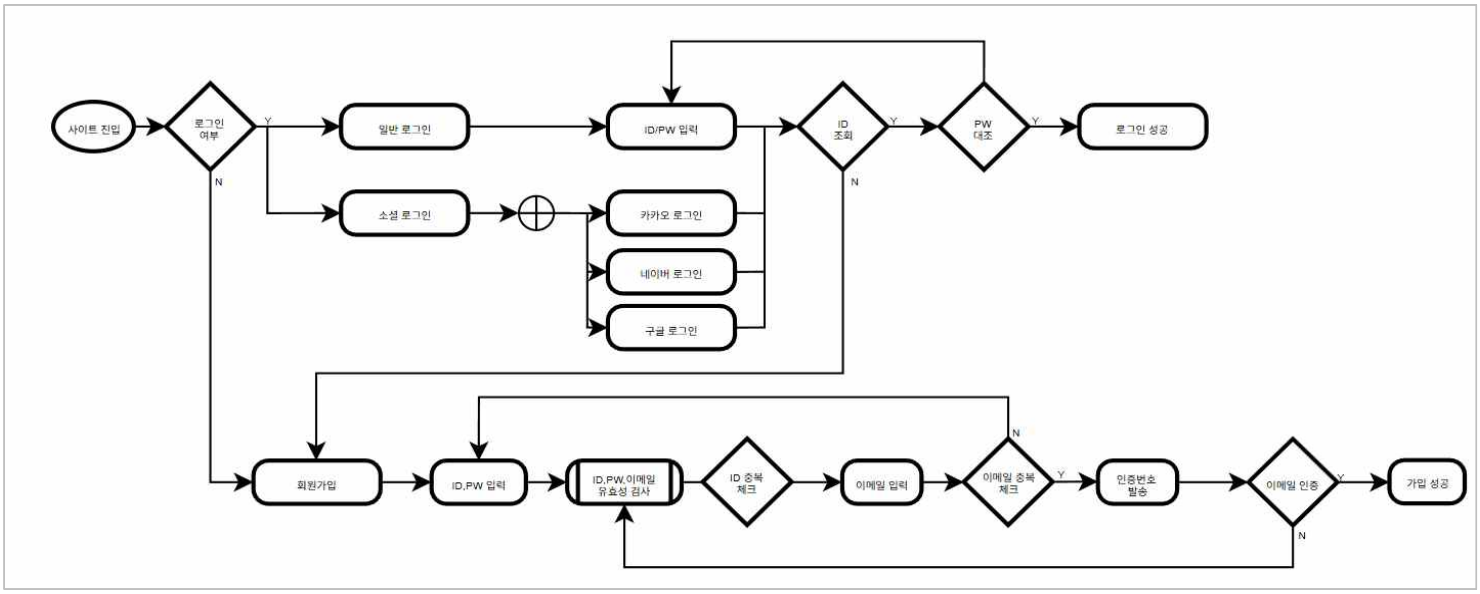
- **CI/CD 및 관제 (DevOps)**
  - ❐ GitOps Workflow

    GitHub Actions를 통해 CI(빌드/테스트)를 수행하고 Docker 이미지를 생성하며, ArgoCD가 Kubernetes Manifest 변경을 감지하여 클러스터에 자동으로 배포(Sync)하는 GitOps 체계를 구축한다.

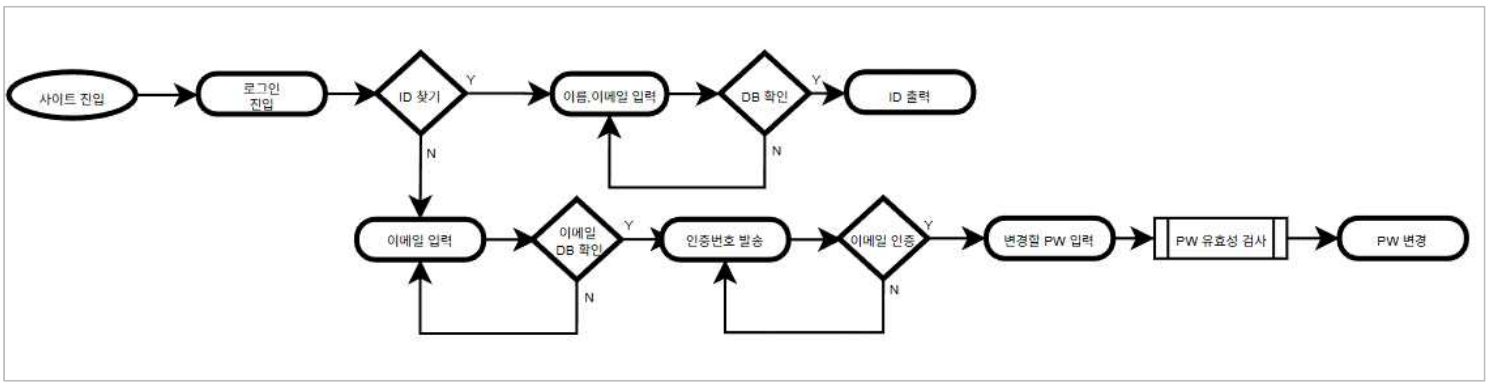
3.2. 유스 케이스 (Use Case)

3.3. 유저 플로우 (User Flow)

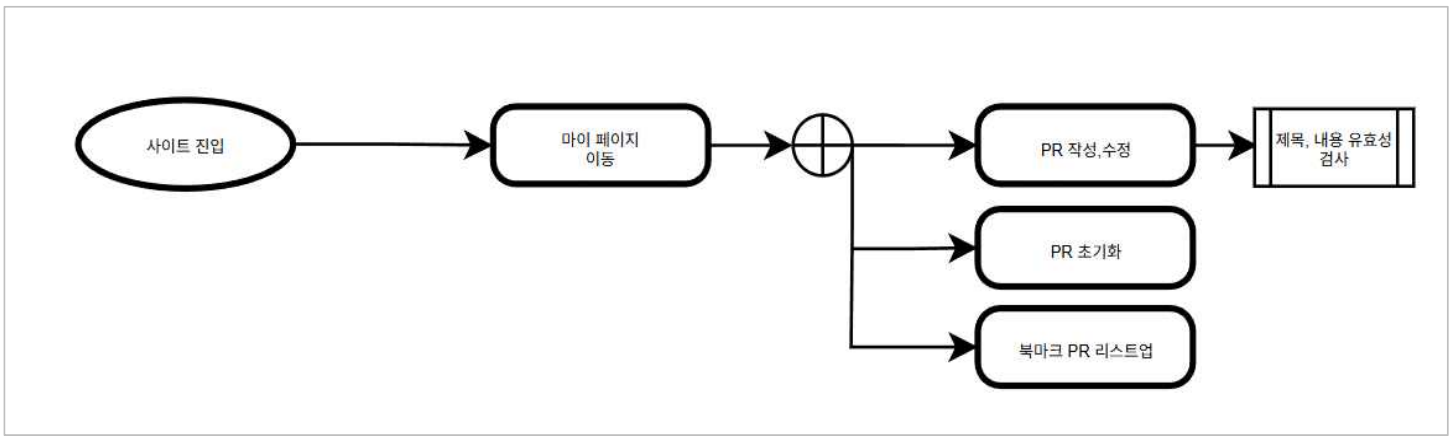
- 로그인/회원가입



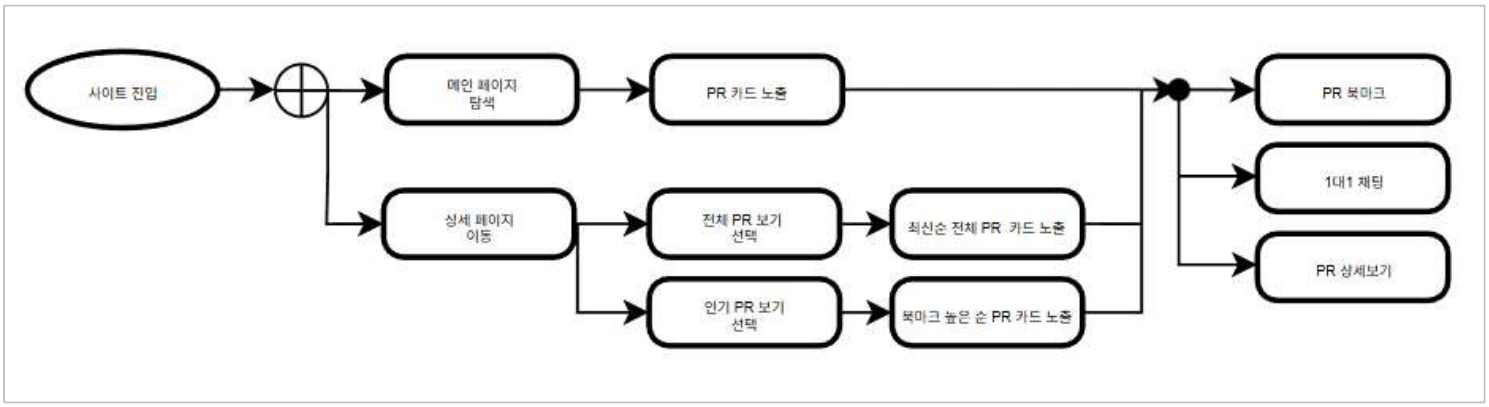
- ID/PW 찾기



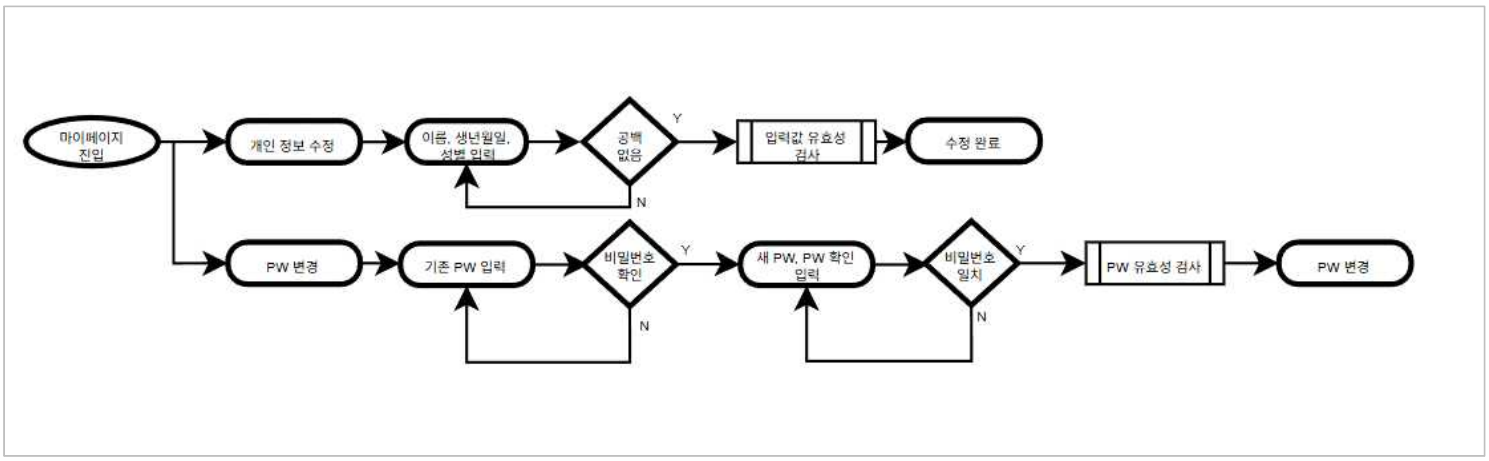
- PR 관리



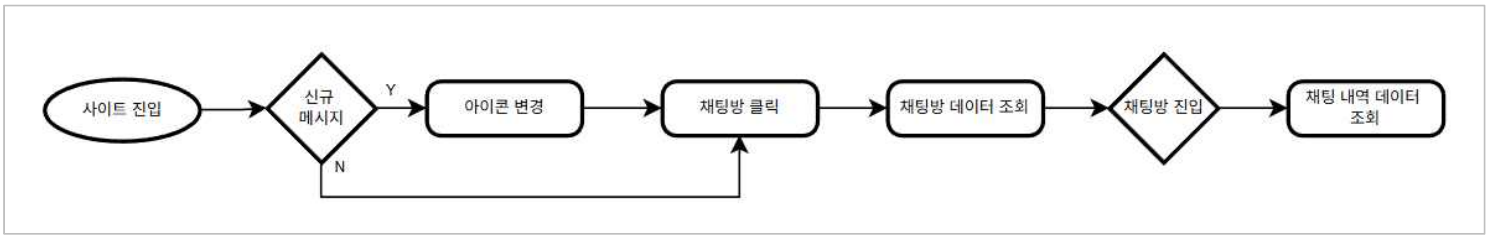
- PR 열람



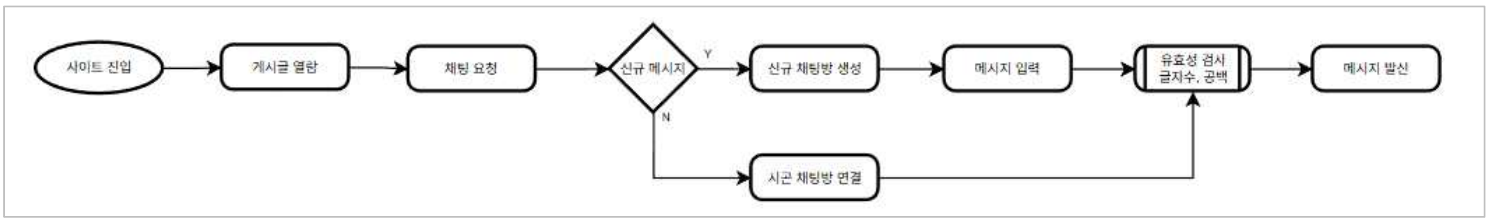
- 개인 정보 수정



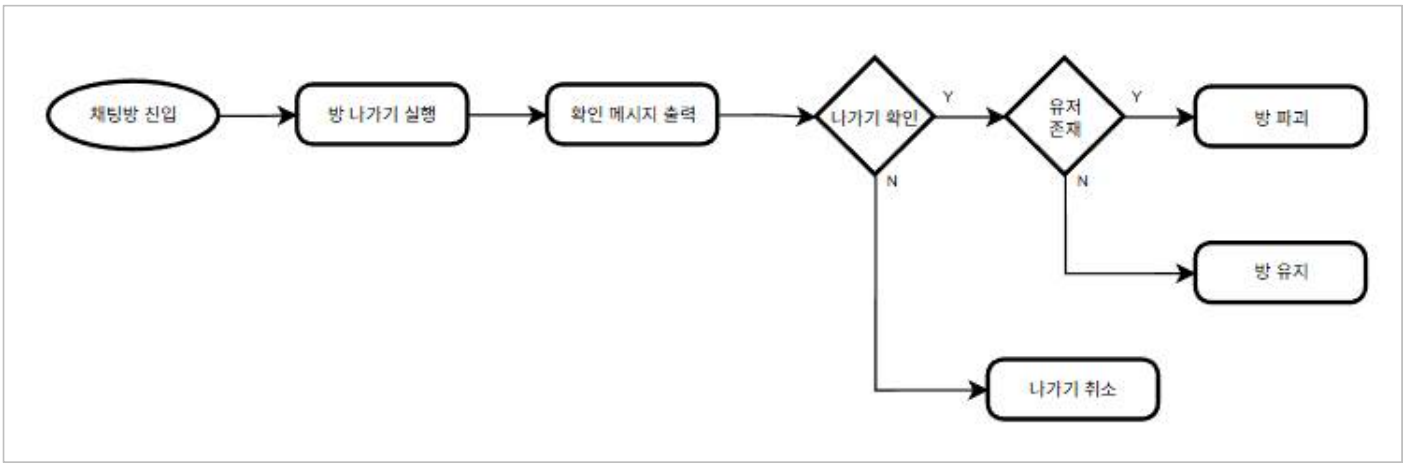
- 채팅방 입장



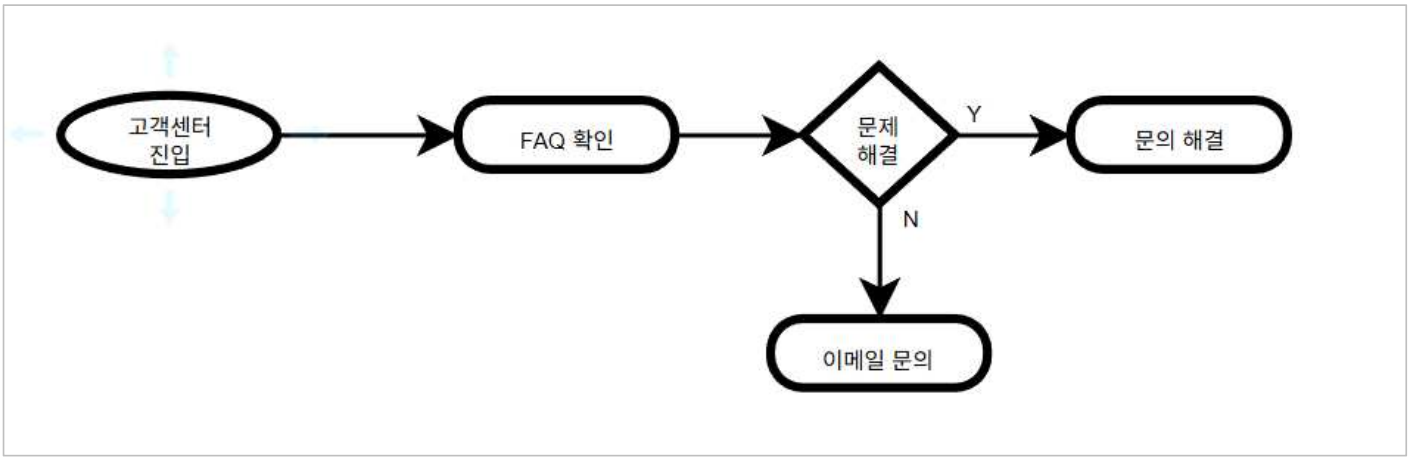
- 채팅방 메시지



- 채팅방 나가기


채팅방 진입
방 나가기 실행
확인 메시지 출력
나가기 확인
Y
N
유저 존재
Y
N
방 파괴
방 유지
나가기 취소

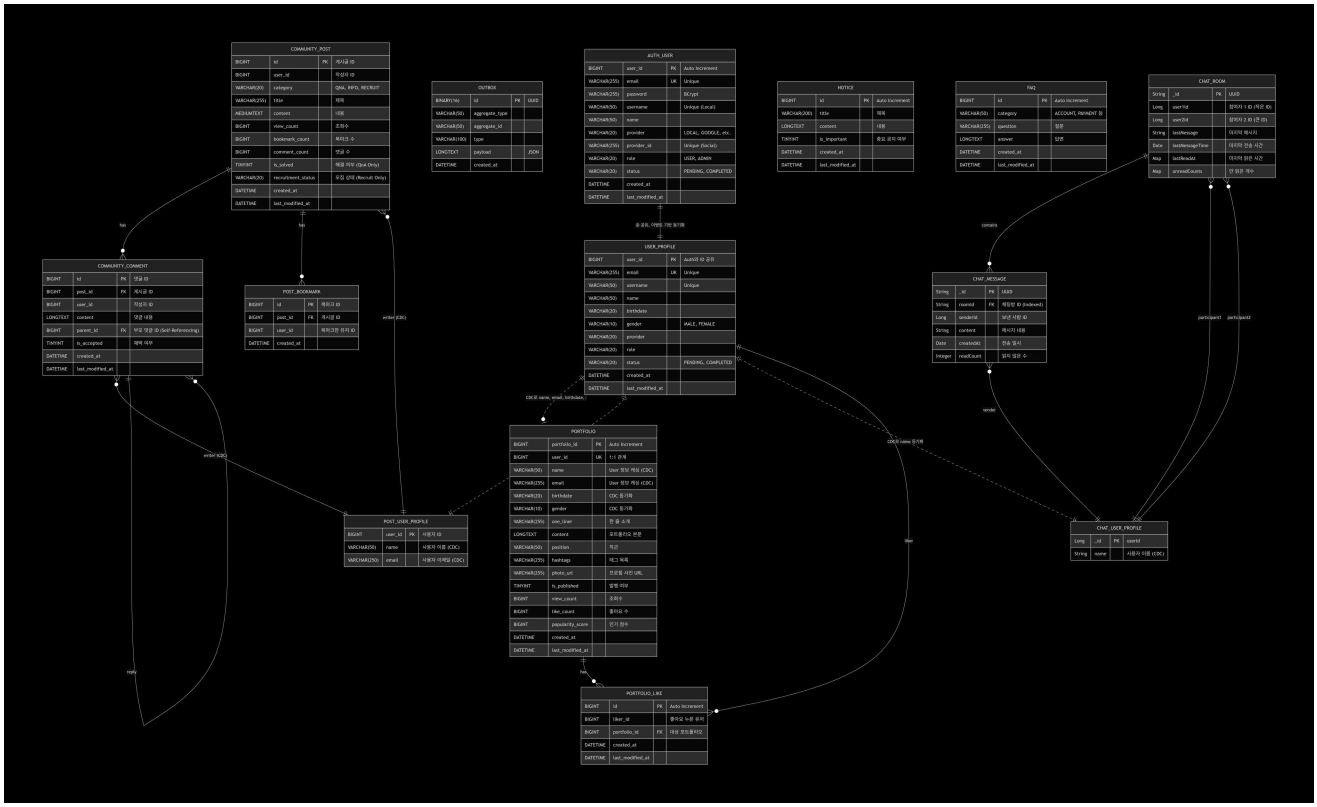
- 고객센터


고객센터 진입
FAQ 확인
문제 해결
Y
N
문의 해결
이메일 문의

3.4. Sequence Diagram

3.5. 데이터 모델

본 프로젝트는 데이터의 특성에 따라 Polyglot Persistence 전략을 적용하였다. 서비스 간 데이터 참조는 물리적 FK 대신 논리적 ID 매핑과 Kafka CDC를 통해 관리된다. (각 테이블의 상세 컬럼 정의, 데이터 타입, 제약 조건 등은 [부록 A. 테이블 정의서]를 참조)



4. 개발 및 배포

4.1 개발 환경

프로젝트의 개발 효율성과 운영 안정성을 확보하기 위해 다음과 같은 표준화된 개발 환경을 구축하였다.

구분	항목	상세 내용 및 버전
OS	Server	Ubuntu 22.04 LTS (On-Premise VM)
Language	Java	JDK 17 (LTS)
	JavaScript	Node.js 18.x (LTS)
Framework	Backend	Spring Boot 3.5.6, Spring Cloud 2025.0.0
	Frontend	Next.js 13+
IDE	Backend	IntelliJ IDEA Ultimate
	Frontend	Visual Studio Code
SCM	Version Control	Git, GitHub (Monorepo Strategy)
Build Tool	Backend	Maven 3.9.x (Wrapper 사용)

- **OS 및 가상화**

  로컬 개발 및 서버 환경 OS로 Ubuntu 22.04 LTS를 사용하며, VirtualBox를 통해 가상 머신을 관리한다.
  Kubernetes Cluster (총 3대): 1 Master Node, 2 Worker Node로 3-Tier 클러스터를 구축하였다.
  Message Broker (총 1대): SAGA 패턴 구현을 위해 별도 VM에 Zookeeper와 Kafka Broker를 구축하였다.

- **IDE**

  백엔드(Spring Cloud) 개발은 IntelliJ IDEA Ultimate을, 프론트엔드 및 Kubernetes 매니페스트(*.yaml) 관리는 VSCode를 사용한다.

- **형상관리**

  GitHub를 사용하며, 두 개의 분리된 레포지토리(Repository)를 운영한다.

- **빌드도구**

  Maven을 사용하여 Java 프로젝트 의존성을 관리하고 빌드한다.

- **배포**

  Kubernetes(K8s)를 최종 배포 환경으로 사용한다. linkfolio-manifest 레포지토리에 정의된 명세를 기반으로 ArgoCD가 GitOps 워크플로우를 통해 클러스터에 애플리케이션을 배포 및 동기화한다.

4.2. 기술 스택 및 선정 이유

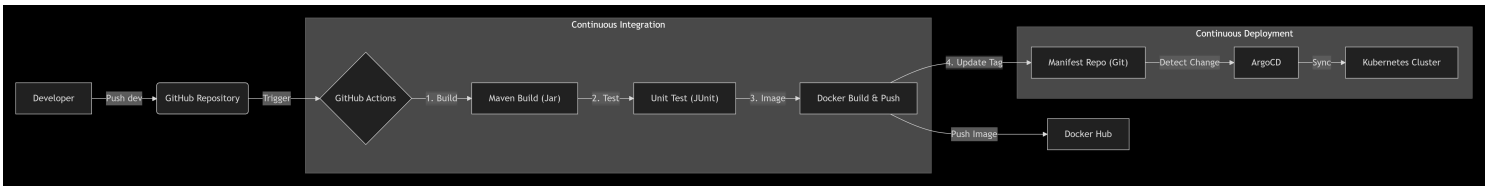
본 프로젝트는 MSA(Microservices Architecture)의 복잡성을 효율적으로 관리하고, 대용량 트래픽과 데이터 정합성을 보장하기 위해 다음과 같은 기술 스택을 선정하였다.

분류	기술명	버전	용도 및 선정 비고
Frontend	Next.js	13^	SSR 및 SEO 최적화를 통해 초기 로딩 속도를 개선하고 검색 엔진 노출을 극대화하기 위해 사용함
	HTML5	-	웹 표준 마크업
	React	18^	컴포넌트 기반 개발로 재사용성을 높이고 가상 DOM을 통한 효율적인 렌더링을 위해 사용함
	TypeScript	5^	정적 타입 검사를 통해 런타임 오류를 사전에 방지하고 개발 생산성을 향상시키기 위해 사용함
UI/Styling	Tailwind CSS	3	유틸리티 퍼스트 접근 방식으로 빠른 UI 개발과 일관된 디자인 시스템 구축을 위해 사용함
	MUI (Material-UI)	7.3+	엔터프라이즈급 컴포넌트 라이브러리로 복잡한 폼 및 관리자 페이지 구축에 활용함
	Emotion	11+	CSS-in-JS 방식으로 컴포넌트 레벨 스타일링과 동적 스타일링을 위해 사용함
	CVA(Class Variance Authority)	0.7+	컴포넌트의 variant 기반 스타일링을 타입 세이프하게 관리하여 조건부 클래스 적용을 체계화하기 위해 사용함
	clsx	2.1+	조건부 클래스명을 효율적으로 조합하여 동적 스타일링과 클래스 병합을 위해 사용함
Backend	Java	17	최신 LTS 버전의 안정성과 Record 등 모던 Java 문법을 활용하여 생산성을 높이기 위해 채택함
	Spring Boot	3.5.x	마이크로서비스 애플리케이션 구축을 위한 표준 프레임워크로, Native Image 등 클라우드 환경 최적화를 고려함
	Spring Cloud	2025.0.0	API Gateway, OpenFeign 등 MSA 구축에 필수적인 컴포넌트를 제공하여 서비스 디스커버리 및 라우팅을 효율화함
	Spring Security	-	JWT 기반의 Stateless 인증/인가 처리 및 OAuth2 소셜 로그인(Google, Naver, Kakao) 구현을 위해 사용함
	QueryDSL	5.1.0	복잡한 검색 조건(포트폴리오 필터링 등)을 Type-Safe하게 작성하여 런타임 오류를 방지하기 위해 사용함
	WebSocket / STOMP	-	실시간 1:1 채팅 기능을 구현하기 위해 표준 웹소켓 프로토콜 위에 메시징 규약을 얹어 사용함
Database	MySQL	8.0	회원(User), 인증(Auth), 포트폴리오(Portfolio) 등 정형 데이터의 ACID 트랜잭션을 보장하기 위해 메인 저장소로 사용함
	MongoDB	6.0+	채팅 서비스(chat-service)에서 발생하는 대량의 쓰기 트래픽을 감당하고 비정형 메시지 데이터를 유연하게 저장하기 위해 도입함
	Redis	-	인메모리 기반의 고속 I/O를 활용하여 Refresh Token 저장, 채팅 Pub/Sub 메시지 브로커, 이메일 인증 코드 캐싱에 사용함
Messaging	Apache Kafka	-	서비스 간 결합도를 낮추기 위한 비동기 통신과 회원가입 SAGA 패턴 및 데이터 동기화 처리를 안정적으로 수행하기 위해 사용함
	Avro / Schema Registry	-	JSON 대비 페이로드 크기를 줄여 네트워크 효율을 높이고, 스키마 변경(Evolution)을 엄격하게 관리하기 위해 채택함
	Debezium	-	DB 변경 사항(Transaction Log)을 실시간으로 감지하여 Kafka 이벤트를 자동으로 발행하는 CDC(Change Data Capture) 파이프라인 구축에 사용함
DevOps	Kubernetes (K8s)	-	Ubuntu VM 환경 위에 클러스터를 구축하여 다수의 컨테이너를 오케스트레이션하고 오토스케일링 및 자가 치유 환경을 구성함
	Docker	-	애플리케이션을 eclipse-temurin:17 기반의 경량 컨테이너로 패키징하여 배포 환경의 일관성을 유지함
	ArgoCD	-	Git 저장소의 형상 변경을 감지하여 K8s 클러스터에 자동으로 배포하는 GitOps 기반 CD 파이프라인을 구축함
	GitHub Actions	-	소스 코드 푸시 시 빌드, 테스트, Docker 이미지 생성을 자동화하는 CI 파이프라인 도구로 사용함
	Ubuntu	22.04 LTS	온프레미스(VM) 환경의 호스트 운영체제로 사용하여 인프라 제어권을 확보함
Tools	Swagger	2.5.0	REST API 명세를 자동으로 문서화하여 프론트엔드와의 협업 효율을 높이고 테스트를 자동화함
	MapStruct	1.5.5	Entity와 DTO 간의 변환 코드를 컴파일 시점에 자동으로 생성하여 반복 작업을 줄이고 성능을 최적화함
	Figma	-	UI·UX를 시각적으로 설계하고 화면 흐름을 명확히 정리해 프론트엔드·백엔드·디자인 간 협업을 효율화하는 데 사용됨

4.3 CI/CD

지속적인 통합(CI)과 배포(CD)를 위해 GitHub Actions와 ArgoCD를 결합한 GitOps 파이프라인을 구축하였다. 이를 통해 소스 코드 변경부터 운영 환경 배포까지의 과정을 자동화하여 배포 안정성을 확보하였다.

4.3.1.CI/CD 구성도



4.3.2. 파이프라인 상세 단계

pipeline-monorepo.yaml 워크플로우 정의에 따른 배포 과정은 다음과 같다.

1) Code Push & Trigger
   - 개발자가 dev 브랜치에 코드를 푸시하면 GitHub Actions가 트리거된다.

2) Build &Test (CI)
   - JDK 17 환경을 셋업하고, 의존성 순서에 따라 common-module을 포함한 전체 모듈을 Maven으로 빌드한다.
   - JUnit 5 기반의 단위 테스트를 수행하여 코드 무결성을 검증한다.

3) Docker Image Build
   - Matrix Strategy를 사용하여 6개 서비스(apigateway, auth, user, portfolio, chat, support)의 Docker 이미지를 병렬(Parallel)로 빌드한다.
   - 생성된 이미지는 Commit Short SHA 태그를 부여하여 Docker Hub에 푸시한다.

4) Manifest Update (CD Trigger)
   - kustomize를 사용하여 별도의 Manifest Repository(linkfolio-manifest)의 이미지 태그 정보를 최신 빌드 버전으로 업데이트하고 커밋한다.

5) Deployment (ArgoCD)
   - k8s 클러스터 내의 ArgoCD가 Manifest Repository의 변경 사항을 감지한다.
   - 기존 파드와 최신 설정 간의 차이(Diff)를 분석하고, Rolling Update 방식으로 새로운 버전의 컨테이너를 배포한다.

4.4 테스트 및 검증

- 단위 테스트 (JUnit5)
- 통합 테스트
- Postman 테스트 스크린샷

4.5. 트러블슈팅 및 핵심 문제 해결

4.6. 프로젝트 관리 및 개발 방법론

❐ 개발 방법론

본 프로젝트는 **애자일(Agile)** 방법론을 기반으로 한 **스크럼(Scrum)** 방식을 적용하였다. 1주 단위의 스프린트를 설정하여 주기적으로 개발 목표를 계획하고, 스프린트 종료 시점에 회고(Review &Retrospective)를 진행하였다. 이를 통해 개발 과정에서 발생하는 요구사항 변화를 유연하게 반영하고, 기능 단위로 빠르게 프로토타입을 검증할 수 있었다.

❐ 업무 관리 도구

프로젝트 일정 및 태스크 관리는 Jira를 중심으로 수행하였다. Kanban 보드를 활용하여 Todo · In Progress · Done 상태를 명확히 구분하였으며, 담당자와 마감일을 태스크 단위로 지정하여 책임 기반의 업무 분배를 실현하였다. 또한 전체적인 일정은 WBS로 관리하였으며 세부 주요 일정(스프린트 일정, 배포 계획)은 Google Calender와 연동하여 관리하였다.

❐ 커뮤니케이션 도구

팀 내 실시간 소통은 Gather Town을 주 채널로 사용하였고, 보조 커뮤니케이션 도구로는 Slack을 이용하였다.

5. 성능 및 보안

5.1 성능 개선

- DB 인덱스 적용
- 캐싱 전략 (Redis)
- API 응답 속도 개선 내역

5.2 보안 고려사항

- HTTPS, JWT, HttpOnly Cookie 적용
- CORS 정책 설정
- OAuth2 로그인 (Google, Kakao, Naver)

6. 결과물

6.1. 주요 화면 및 UI/UX

6.2. 시연 영상

- 링크

6.3. 테스트 결과 및 품질 지표

- 테스트 커버리지 (백엔드/프론트엔드)
- 성능 테스트 결과 (부하 테스트, 동시 사용자 처리 등)
- 코드 정적 분석 결과 (SonarQube 등)

7. 회고

7.1. 팀원 구성

이름	역할	담당 업무	세부 내용
허준형	Backend	시스템 설계	MSA 아키텍처 설계, DB 모델링(Polyglot Persistence), 인프라 구축 (On-Premise k8s)
		서비스 개발	Auth(인증), User, Portfolio, Chat(실시간), Support 등 전체 마이크로서비스 구현
	Architecture	데이터 파이프라인	Kafka &Debezium(CDC) 기반의 SAGA 패턴 및 데이터 동기화 처리
		DevOps	GitHub Actions &ArgoCD 기반의 CI/CD 파이프라인 구축
박성현	Frontend	화면 레이아웃	메인(포트폴리오 조회/필터링), 인증(로그인/회원가입/OAuth), 커뮤니티(QnA/Share/Team), 마이페이지(정보/포트폴리오), 관리자(FAQ/Notice)
		UI/UX 개발	반응형 웹 디자인(모바일/태블릿/노트북/데스크톱), Tailwind CSS, CVA 기반 디자인 시스템, 접근성 고려 시맨틱 마크업
		SEO & PWA	Next.js Metadata API, Open Graph, PWA 설정(오프라인 지원, 앱 설치)

7.2. 배운점 / 느낀점

7.3. 한계점 및 향후 개선 계획

7.4 차후 버전 로드맵

8. LINKS

8.1. GitHub Repository URL

8.2. 시연 영상 링크

8.3. 참고 자료

[부록 A] 테이블 정의서

1. Auth Service

사용 서비스	auth service	데이터베이스명	auth_db	테이블명	auth_user
테이블 설명	서비스 사용자의 로그인 계정 및 인증 상태를 관리하는 핵심 테이블				

No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고
1	user_id	BIGINT	N	Y	사용자 고유 ID	Auto Increment
2	email	VARCHAR(255)	N		이메일	Unique Key
3	password	VARCHAR(255)	Y		비밀번호	암호화 저장 (BCrypt)
4	username	VARCHAR(50)	Y		아이디	Unique Key (Local)
5	name	VARCHAR(50)	Y		사용자 실명	
6	provider	VARCHAR(20)	N		가입 경로 (Provider)	LOCAL, GOOGLE 등
7	provider_id	VARCHAR(255)	Y		소셜 식별자	Unique Key (Social)
8	role	VARCHAR(20)	N		사용자 권한	USER, ADMIN
9	status	VARCHAR(20)	N		계정 상태	PENDING, COMPLETED
10	created_at	DATETIME	N		생성 일시	BaseEntity
11	last_modified_at	DATETIME	N		수정 일시	BaseEntity

사용 서비스	auth service	데이터베이스명	auth_db	테이블명	outbox
테이블 설명	분산 트랜잭션(SAGA) 처리를 위한 이벤트 발행 기록 테이블 (CDC 연동)				

No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고
1	id	BINARY(16)	N	Y	이벤트 고유 ID	UUID
2	aggregate_type	VARCHAR(50)	N		도메인 유형	예: USER
3	aggregate_id	VARCHAR(50)	N		도메인 ID	예: userId
4	type	VARCHAR(100)	N		이벤트 타입	예: UserRegistration...
5	payload	LONGTEXT	N		이벤트 데이터	JSON 포맷
6	created_at	DATETIME	N		생성 일시	

2. User Service

사용 서비스	user service	데이터베이스명	user_db	테이블명	user_profile
테이블 설명	인증된 사용자의 상세 프로필 정보를 관리하는 테이블				

No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고
1	user_id	BIGINT	N	Y	사용자 고유 ID	Auth 서비스와 ID 공유
2	email	VARCHAR(255)	N		이메일	Unique Key
3	username	VARCHAR(50)	Y		아이디	Unique Key
4	name	VARCHAR(50)	Y		사용자 실명	
5	birthdate	VARCHAR(20)	Y		생년월일	
6	gender	VARCHAR(10)	Y		성별	MALE, FEMALE
7	provider	VARCHAR(20)	N		가입 경로	
8	role	VARCHAR(20)	N		사용자 권한	
9	status	VARCHAR(20)	N		프로필 상태	PENDING, COMPLETED
10	created_at	DATETIME	N		생성 일시	
11	last_modified_at	DATETIME	N		수정 일시	

3. Portfolio Service

사용 서비스	portfolio service	데이터베이스명	portfolio_db	테이블명	portfolio
테이블 설명	사용자가 작성한 포트폴리오 본문 및 메타데이터				

No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고
1	portfolio_id	BIGINT	N	Y	포트폴리오 ID	Auto Increment
2	user_id	BIGINT	N		소유자 ID	Unique Key (1:1)
3	name	VARCHAR(50)	N		작성자 이름	User 정보 캐싱
4	email	VARCHAR(255)	N		작성자 이메일	User 정보 캐싱
5	birthdate	VARCHAR(20)	Y		생년월일	
6	gender	VARCHAR(10)	Y		성별	HTML/Markdown
7	one_liner	VARCHAR(255)	Y		한 줄 소개	예: Backend
8	content	LONGTEXT	Y		포트폴리오 본문	쉼표(,) 구분
9	position	VARCHAR(50)	Y		직군	
10	hashtags	VARCHAR(255)	Y		태그 목록	Default: 0 (False)
11	photo_url	VARCHAR(255)	Y		프로필 사진 URL	Default: 0
12	is_published	TINYINT(1)	N		발행 여부	Default: 0
13	view_count	BIGINT	N		조회수	
14	like_count	BIGINT	N		좋아요 수	
15	popularity_score	BIGINT	Y		인기 점수	
16	created_at	DATETIME	N		생성 일시	
17	last_modified_at	DATETIME	N		수정 일시	

사용 서비스	portfolio service			데이터베이스명	portfolio_db	테이블명	portfolio_like
테이블 설명	사용자가 '북마크'를 누른 포트폴리오 관계 매핑 테이블						
No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고	
1	id	BIGINT	N	Y	좋아요 ID	Auto Increment	
2	liker_id	BIGINT	N		좋아요 누른 유저		
3	portfolio_id	BIGINT	N		대상 포트폴리오	FK	
4	created_at	DATETIME	N		생성 일시		
5	last_modified_at	DATETIME	N		수정 일시		

4. Community Service

사용 서비스	community service			데이터베이스명	community_db	테이블명	community_post
테이블 설명	커뮤니티 게시글 (QnA, 정보공유, 팀원모집) 정보						
No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고	
1	id	BIGINT	N	Y	게시글 ID		
2	user_id	BIGINT	N		작성자 ID		
3	category	VARCHAR(20)	N		카테고리		
4	title	VARCHAR(255)	N		제목		
5	content	MEDIUMTEXT	N		내용		
6	view_count	BIGINT	N		조회수		
7	bookmark_count	BIGINT	N		북마크 수		
8	comment_count	BIGINT	N		댓글 수		
9	is_solved	TINYINT(1)	N		해결 여부		
10	recruitment_status	VARCHAR(20)	Y		모집 상태		
11	created_at	DATETIME	N		생성 일시		
12	last_modified_at	DATETIME	N		수정 일시		

사용 서비스	community service			데이터베이스명	community_db	테이블명	community_comment
테이블 설명	게시글에 달린 댓글 정보 (계층형 구조 지원)						
No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고	
1	id	BIGINT	N	Y	댓글 ID		
2	post_id	BIGINT	N		게시글 ID		
3	user_id	BIGINT	N		작성자 ID		
4	content	LONGTEXT	N		댓글 내용		
5	parent_id	BIGINT	Y		부모 댓글 ID		
6	is_accepted	TINYINT(1)	N		채택 여부		
7	created_at	DATETIME	N		생성 일시		
8	last_modified_at	DATETIME	N		수정 일시		

사용 서비스	community service			데이터베이스명	community_db	테이블명	post-_bookmark
테이블 설명	게시글 북마크 매핑 테이블						
No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고	
1	id	BIGINT	N	Y	북마크 ID		
2	post_id	BIGINT	N		게시글 ID		
3	user_id	BIGINT	N		북마크한 유저 ID		
4	created_at	DATETIME	N		생성 일시		

사용 서비스	community service			데이터베이스명	community_db	테이블명	post_user_profile
테이블 설명	게시글 목록 조회를 위해 User 정보를 로컬에 복제(Sync)해두는 테이블						
No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고	
1	user_id	BIGINT	N	Y	사용자 ID		
2	name	VARCHAR(50)	N		사용자 이름		
3	email	VARCHAR(250)	N		사용자 이메일		

5. Support Service

사용 서비스	support service			데이터베이스명	support_db	테이블명	notice
테이블 설명	시스템 공지사항 게시글 정보						
No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고	
1	id	BIGINT	N	Y	공지 ID	Auto Increment	
2	title	VARCHAR(200)	N		제목		
3	content	LONGTEXT	N		내용		
4	is_important	TINYINT(1)	N		중요 공지 여부	상단 고정용	
5	created_at	DATETIME	N		생성 일시		
6	last_modified_at	DATETIME	N		수정 일시		

사용 서비스	support service			데이터베이스명	support_db	테이블명	faq
테이블 설명	카테고리별 FAQ 데이터						
No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고	
1	id	BIGINT	N	Y	FAQ ID	Auto Increment	
2	category	VARCHAR(50)	N		질문 카테고리	ACCOUNT, PAYMENT 등	
3	question	VARCHAR(255)	N		질문		
4	answer	LONGTEXT	N		답변		
5	created_at	DATETIME	N		생성 일시		
6	last_modified_at	DATETIME	N		수정 일시		

6. Chat Service (MongoDB)

사용 서비스	chat service			콜렉션 명	chat_db	테이블명	chat_room
테이블 설명	1:1 채팅방 메타데이터 및 읽음 상태 관리						
No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고	
1	_id	String	N	Y	채팅방 ID	UUID	
2	user1Id	Long	N		참여자 1 ID	(작은 ID) Indexed	
3	user2Id	Long	N		참여자 2 ID	(큰 ID) Indexed	
4	lastMessage	String	Y		마지막 메시지	목록 미리보기용	
5	lastMessageTime	Date	Y		마지막 전송 시간	정렬 기준	
6	lastReadAt	Map<String, DateTime>	Y		마지막 읽은 시간		
7	unreadCounts	Map	Y		안 읽은 개수	Key: userId	

사용 서비스	chat service			콜렉션 명	chat_db	테이블명	chat_message
테이블 설명	채팅방 내의 개별 메시지 이력						
No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고	
1	_id	String	N	Y	메시지 ID	UUID	
2	roomId	String	N		채팅방 ID	FK (Indexed)	
3	senderId	Long	N		보낸 사람 ID		
4	content	String	N		메시지 내용		
5	createdAt	Date	N		전송 일시		
6	readCount	Integer	N		읽지 않은 수	기본값: 1	

사용 서비스	chat service			콜렉션 명	chat_db	테이블명	chat_user_profile
테이블 설명	채팅방 내의 개별 메시지 이력						
No	컬럼명	데이터타입	NULL	PK	컬럼 정의	비고	
1	_id	Long	N	Y	사용자 ID		
2	name	String	N		사용자 이름		

[부록 B] API 명세서

1. Auth Service

Method	URI	기능	Request (Body / Param)	Response (200 OK)
POST	/auth/signup	회원가입	{ “email“: “user@test.com“, “password“: “...“, “username“: “user1“, “name“: “홍길동“, “birthdate“: “1990-01-01“, “gender“: “MALE“ }	(201 Created)
POST	/auth/login	로그인	{ “username“: “user1“, “password“: “...“ }	{ “accessToken“: “ey...“ } (Cookie: refresh_token)
POST	/auth/logout	로그아웃	(Header: X-User-Id)	(200 OK)
POST	/auth/refresh	토큰 재발급	(Cookie: refresh_token)	{ “accessToken“: “ey...“ } (Cookie: refresh_token 갱신)
POST	/auth/find-username	아이디 찾기	{ “name“: “홍길동“, “email“: “user@test.com“ }	{ “username“: “user1“ }
POST	/auth/check-username	ID 중복 검사	{ “username“: “user1“ }	(200 OK)
POST	/auth/check-password	비밀번호 확인	{ “password“: “...“, “passwordConfirm“: “...“ }	(200 OK)
PATCH	/auth/password	비밀번호 변경	{ “currentPassword“: “...“, “newPassword“: “...“, “newPasswordConfirm“: “...“ }	(200 OK)
POST	/auth/email-verification/send	인증 코드 발송	{ “email“: “user@test.com“ }	(200 OK)
POST	/auth/email-verification/check	인증 코드 검증	{ “email“: “user@test.com“, “code“: “123456“ }	(200 OK)
POST	/auth/password-reset/send-code	PW 재설정 코드 발송	{ “email“: “user@test.com“ }	(200 OK)
POST	/auth/password-reset/verify-code	PW 재설정 코드 검증	{ “email“: “user@test.com“, “code“: “123456“ }	(200 OK)
POST	/auth/password-reset/change	PW 재설정 (변경)	{ “email“: “...“, “newPassword“: “...“, “passwordConfirm“: “...“ }	(200 OK)

2. User Service

Method	URI	기능	Request (Body / Param)	Response (200 OK)
GET	/users/me	내 정보 조회	-	{ “email“: “...“, “username“: “...“, “name“: “...“, “birthdate“: “...“, “gender“: “...“ }
PUT	/users/me	내 정보 수정	{ “name“: “김철수“, “birthdate“: “1995-05-05“, “gender“: “MALE“ }	{ “id“: 1, “email“: “...“ }
GET	/users/{userId}	회원 정보 조회	(Path: userId)	{ “email“: “...“, “username“: “...“, “name“: “...“ ... }

3. Portfolio Service

Method	URI	기능	Request (Body / Param)	Response (200 OK)
GET	/portfolios	목록 조회	?page=0&size=10 &position=BACKEND &sort=popularityScore,desc	{ “content“: [{ “portfolioId“: 1, “name“: “...“, “position“: “...“, “photoUrl“: “...“, “viewCount“: 10, ... }], “last“: false }
GET	/portfolios/{id}	상세 조회	(Path: portfolioId)	{ “userId“: 1, “name“: “...“, “content“: “...“, “hashtags“: [“Java“], “viewCount“: 100, “likeCount“: 10, “liked“: false, ... }
GET	/me	내 이력서 조회	-	{ “userId“: 1, “name“: “...“, “content“: “...“, “published“: true, ... }
PUT	/me	내 이력서 저장	{ “photoUrl“: “...“, “oneLiner“: “...“, “content“: “...“, “position“: “...“, “hashtags“: [...] }	{ “userId“: 1, “name“: “...“, “published“: true, ... }
GET	/me/likes	관심 목록 조회	?page=0&size=10	{ “content“: [{ “portfolioId“: 2, “name“: “...“, ... }], “last“: false }
POST	/portfolios/{id}/like	북마크 추가	(Path: portfolioId)	(201 Created)
DELETE	/portfolios/{id}/like	북마크 취소	(Path: portfolioId)	(204 No Content)

4. Community Service

Method	URI	기능	Request (Body / Param)	Response (200 OK)
GET	/posts	게시글 목록	?category=QNA &isSolved=false &page=0&size=10	{ "content": [{ "id": 1, "title": "...", "writerName": "...", "viewCount": 0, "isSolved": false ... }], "totalPages": 5 }
POST	/posts	게시글 작성	{ "category": "QNA", "title": "질문있습니다", "content": "내용..." }	1 (Created Post ID)
GET	/posts/{id}	게시글 상세	(Path: postId)	{ "id": 1, "title": "...", "content": "...", "comments": [...], "bookmarked": false }
PUT	/posts/{id}	게시글 수정	{ "title": "수정 제목", "content": "수정 내용" }	(200 OK)
DELETE	/posts/{id}	게시글 삭제	(Path: postId)	(200 OK)
POST	/posts/{id}/comments	댓글 작성	{ "content": "댓글내용", "parentId": null }	(200 OK)
PATCH	/posts/.../comments/{id}	댓글 수정	{ "content": "수정댓글", "parentId": null }	(200 OK)
DELETE	/posts/.../comments/{id}	댓글 삭제	(Path: commentId)	(200 OK)
POST	/posts/.../adopt	답변 채택	(Path: commentId)	(200 OK)
POST	/posts/{id}/bookmark	북마크 토글	(Path: postId)	(200 OK)
POST	/posts/{id}/recruit/apply	팀원 지원	(Path: postId)	(200 OK)
PATCH	/posts/{id}/status	모집 상태 변경	{ "status": "CLOSED" }	(200 OK)
GET	/posts/me	내가 쓴 글	?category=QNA	{ "content": [{ "id": 1, "title": "..." }] }
GET	/posts/me/bookmarks	북마크한 글	?category=INFO	{ "content": [{ "id": 5, "title": "..." }] }

5. Support Service

Method	URI	기능	Request (Body / Param)	Response (200 OK)
GET	/notices	공지사항 목록	?page=0&size=10	{ "content": [{ "id": 1, "title": "...", "important": true }] }
GET	/notices/{id}	공지사항 상세	(Path: id)	{ "id": 1, "title": "...", "content": "...", "createdAt": "..." }
GET	/faqs	FAQ 목록	?category=PAYMENT	[{ "id": 1, "category": "PAYMENT", "question": "...", "answer": "..." }]
POST	/admin/notices	공지 등록	{ "title": "...", "content": "...", "important": true }	(201 Created)
PUT	/admin/notices/{id}	공지 수정	{ "title": "...", "content": "...", "important": false }	(200 OK)
DELETE	/admin/notices/{id}	공지 삭제	(Path: id)	(200 OK)
POST	/admin/faqs	FAQ 등록	{ "category": "ETC", "question": "...", "answer": "..." }	(201 Created)

6. Chat Service

Method	URI	기능	Request (Body / Param)	Response (200 OK)
GET	/chat/rooms	채팅방 목록	?page=0	{ "content": [{ "roomId": "uuid", "otherUserName": "...", "lastMessage": "...", "unreadCount": 2 }] }
GET	/chat/rooms/{id}/messages	메시지 내역	?page=0	{ "content": [{ "senderId": 1, "content": "안녕", "createdAt": "..." }] }
GET	/chat/unread-count	안 읽은 수	-	5 (Long)
WS	/ws-chat	소켓 연결	(Header: Authorization)	(101 Switching Protocols)