Software Design Specification (SDS)

Version: 1.0

Date: 2025-10-18

Authors: LogMate Team (강찬욱, 양승원, 정주연)

1. Introduction (소개)

1.1 Purpose (문서 목적)

본 문서는 LogMate 시스템의 소프트웨어 설계를 기술한다. SRS에서 정의된 요구사항을 충족하기 위해 선택한 소프트웨어 구조, 컴포넌트, 인터페이스, 데이터 설계, 처리 규칙을 상세히 설명하며, 구현·테스트·운영 단계에서 공통 기준으로 사용된다.

1.2 Scope (범위)

- 주요 서브시스템: Agent / Streaming / API / Frontend / AI Server
- 공용 인프라: Kafka, OpenSearch, MySQL

1.3 Definitions, Acronyms, Abbreviations (용어)

Acronym / Term (약어 / 용어)	Definition (정의)
Subsystem (서브시스템)	전체 시스템 내에서 독립적인 책임과 기능을 갖는 구성 단위. LogMate는 여러 서브시스템(Agent, Streaming Server 등)으로 구성된다.
Component (컴포넌트)	하나의 서브시스템 내에서 특정 기능을 수행하는 내부 구성 요소. 예: Kafka Producer, File Tailer.
Module (모듈)	특정 기능을 구현한 논리적 코드 단위. 컴포넌트와 유사하며 재사용 가능하다.

Interface (인터페이스)	서브시스템 또는 컴포넌트 간 데이터 교환 지점. REST API, WebSocket 등이 대표적이다.
Pipeline (파이프라인)	데이터를 단계적으로 처리하는 구조. LogMate에서는 로그 수집, 분석, 저장을 포함한다.
Deployment Environment (배포 환경)	소프트웨어가 실제로 실행되는 시스템 환경. LogMate는 Docker 기반 컨테이너 환경에 배포된다.
Client (클라이언트)	서버에 요청을 보내는 주체. 사용자 브라우저(Frontend) 또는 Agent가 해당된다.
Server (서버)	클라이언트의 요청을 받아 처리하는 시스템. API, Streaming, AI Server가 이에 해당한다.
Endpoint (엔드포인트)	외부 시스템이 접근 가능한 API의 URI 또는 소켓 주소.
Event (이벤트)	시스템 내 상태 변화 또는 트리거 발생. 예: 로그 수집, 설정 변경, 알림 발생.
Log (로그)	시스템 또는 애플리케이션의 동작 이력을 기록한 데이터. LogMate는 로그를 수집, 분석, 시각화한다.
Timezone (시간대)	시간 기준의 지역 차이. 시스템은 UTC 기준을 사용하고, 사용자는 userTimezone을 설정할 수 있다.
Agent	로그를 수집하고 HTTP를 통해 Streaming Server로 전송하는 클라이언트 프로그램. 사용자 서버에 배포된다.
Streaming Server	로그 파이프라인을 처리하는 핵심 서버. Kafka,

	OpenSearch, AI Server와 연동한다.
API Server	사용자 인증, 팀 관리, 대시보드 설정 등을 처리하는 RESTful API 제공 백엔드 서버.
Frontend	React 기반 사용자 인터페이스. 실시간 로그 시각화, 설정 관리 등을 지원한다.
AI Server	수집된 로그 데이터를 AI 모델로 분석하여 이상 탐지 결과를 반환하는 서버.
Kafka	로그 데이터를 비동기적으로 전달하는 메시지 브로커. log-stream, dlq 토픽을 사용한다.
OpenSearch	로그 데이터 인덱싱, 검색, 집계 등을 위한 분산 검색 엔진.
MySQL	사용자, 팀, 설정 등 메타데이터 저장용 관계형 데이터베이스.
Docker	서버 구성요소를 컨테이너 단위로 배포하기 위한 가상화 도구.
EC2	AWS의 가상 서버 인스턴스. LogMate 구성요소가 실행된다.
Netlify	정적 웹 프론트엔드를 호스팅하는 플랫폼. CI/CD로 자동 배포된다.
GitHub Actions	자동화된 빌드, 테스트, 배포를 수행하는 CI/CD 도구.
Discord Webhook	알림 메시지를 Discord로 전송하기 위한 외부 API 인터페이스.
JWT (JSON Web Token)	인증 정보를 담아 API 호출 시 사용하는 토큰 기반 인증 방식.

REST API	HTTP를 통해 자원을 처리하는 대표적인 웹 API 스타일.
WebSocket	실시간 로그 스트리밍을 위한 양방향 통신 프로토콜.
HTTPS / WSS	보안이 적용된 HTTP / WebSocket 프로토콜.
DLQ (Dead Letter Queue)	Kafka 처리 실패 로그를 저장하는 보조 큐.
eTag	설정 변경 여부를 판단하기 위한 버전 식별자.
AI Score	AI Server가 분석한 결과로, 로그 이벤트의 이상 여부를 수치화한 값.
ParsedLogData	표준화된 로그 데이터 구조. 로그 종류에 따라 서브 클래스(SpringBoot, Tomcat 등)를 가짐.
LogEnvelope	log 데이터와 메타정보(agentId 등)를 포함하는 전송용 포맷.
userTimezone	사용자별 로컬 시간대 설정. UI 표시 시 기준으로 사용됨.
UTC	협정 세계시. 시스템 내 모든 시간 처리 기준이 된다.
CI/CD	지속적 통합(Continuous Integration), 지속적 배포(Continuous Deployment)를 의미하는 자동화 개발 파이프라인.

1.4 References (참고 문헌)

- IEEE Std 1016-1998: Recommended Practice for Software Design Descriptions

- IEEE Std 830-1998: Recommended Practice for Software Requirements Specifications
- LogMate SRS (2025-10-18. version 2.0)

1.5 Overview (개요)

본 SDS 는 IEEE 1016 표준에 따라 작성되었으며, 설계 엔티티 속성 10개와 설계 뷰 4개를 충실히 반영한다.

주요 내용은 다음과 같다.

- **2장**: Design Considerations
- **3장**: Architectural Design 미완
- **4장**: Data Design
- **5장**: Interface Design
- **6장**: IEEE 1016 Design Views
- **7장**: Detailed Design

2. Design Considerations (설계 고려사항)

2.1 Assumptions & Dependencies

2.1.1 Assumptions

2.1.1.1 실행/배포 환경

LogMate 시스템은 각 구성요소가 완전히 분리된 실행 환경에서 독립적으로 운영되는 분산형 배포 환경을 가지고 있다.

Agent, Frontend를 제외한 모든 주요 서브시스템은 컨테이너 단위로 배포 및 관리되며, 로그 수집 Agent 는 별도의 클라이언트 실행 파일 형태로 사용자 환경에 배포된다.

시스템은 다음과 같은 실행/배포 환경을 가정한다.

- **컨테이너 기반 아키텍처**
    - 서버 구성요소(API, Streaming, AI) 는 Docker 를 통해 각각 독립된 컨테이너로 실행된다.
    - 각 컨테이너는 고유한 네트워크와 리소스 제한을 가지며, 다른 서비스와 격리된 환경에서 동작한다.
    - 필요한 환경 변수의 경우 배포 시 외부에서 주입된다.

- **Agent**
    - 로그 수집 기능을 수행하는 Agent는 실행 파일 형태로 빌드되어 사용자 서버 또는 로컬 환경에 배포된다.
    - Agent 는 로컬 로그 파일에 대한 읽기 권한이 확보된 환경에서 동작함을 전제로 한다.
- **API Server**
    - 데이터베이스(MySQL) 와 연결된다.
    - 단일 컨테이너 내에서 독립적으로 실행된다.
    - 필요한 환경 변수는 배포 시 외부에서 주입된다.
- **Streaming Server**
    - Kafka, OpenSearch, AI Server 등 외부 컴포넌트와 연동된다.
    - 단일 컨테이너 내에서 동작하며, 외부 시스템과의 통신은 네트워크 인터페이스를 통해 수행된다.
    - 필요한 환경 변수는 배포 시 외부에서 주입된다.
- **AI Server**
    - AI 기반 로그 분석 및 이상 탐지 기능을 담당한다.
    - 모델 학습 및 추론 기능은 Python 기반으로 구현되며, FastAPI 프레임워크를 사용한다.
    - 실시간 로그 데이터를 수신하고, 분석 결과를 Streaming Server로 전달한다.
- **Frontend**
    - 정적 자산(HTML, JS, CSS)은 Netlify를 통해 배포된다.
    - Netlify는 CI/CD 파이프라인과 연동되어 GitHub 리포지토리의 변경사항을 자동으로 반영한다.
    - 런타임 환경 변수는 빌드 시점에 주입된다.

2.1.1.2 네트워크/보안

LogMate 시스템은 모듈 간 완전한 네트워크 분리와 안전한 통신 경로 확보를 전제로 설계되었다. 각 서버는 개별 컨테이너 내부에서 고유한 네트워크 네임스페이스를 사용하며, 외부 클라이언트(Agent, Frontend)는 인증된 요청을 통해서만 내부 네트워크에 접근할 수 있다.

- **네트워크 구조**
    - 서버 구성요소(API, Streaming, AI)는 각각 별도의 EC2 인스턴스에서 Docker 컨테이너 단위로 실행되며, 서버 간 통신은 퍼블릭 또는 VPC 네트워크를 통해 이루어진다. Compose 기반 단일 네트워크는 사용되지 않으며, 각 서버는 독립된 호스트 환경에서 운영된다. 외부로 노출되는 포트는 최소화한다.
- **데이터 전송 및 암호화**

- 외부 통신(Agent↔API/Streaming, Frontend↔API/Streaming, Streaming↔AI) 은 TLS(HTTPS/WSS)로 암호화된다. Kafka 및 OpenSearch, MySQL 에 대한 접근은 보안 그룹 정책에 따라 접근이 제한되고 SSL을 통해 보호된다.

- **인증 및 접근 제어**
  - Agent는 API Server에서 발급받은 Agent ID와 클라이언트 ID, Password를 사용하여 인증을 수행하며, 이를 통해 JWT 기반 토큰을 획득한다. 이 토큰을 이용해 Streaming Server로 로그를 안전하게 전송할 수 있다. 사용자 인증은 로그인 절차를 통해 발급받은 JWT 토큰으로 수행된다.
- **보안 정책 및 관리**
  - 민감 정보(API 키, DB 비밀번호)는 GitHub Actions Secrets 또는 환경 변수를 통해 안전하게 주입된다. CORS 정책은 API Server, Streaming Server 에서 제어되며, 허용된 도메인 외 요청은 거부된다.

2.1.1.3 사용자/운영 형태

LogMate의 사용자는 일반 사용자와 운영자로 구분되며, 각자의 역할과 권한에 따라 시스템을 이용한다.

- 일반 사용자
  - 일반 사용자는 Frontend를 통해 실시간 로그를 조회하고, 대시보드를 구성한다. JWT 기반 로그인 후 사용자 권한에 따라 접근 범위가 제한된다.
- 운영자
  - 운영자는 Kafka, OpenSearch, API, Streaming 서버를 포함한 전체 인프라를 관리하며, 장애 발생 시 로그 분석 및 재배포를 수행한다.

2.1.2 Dependencies

2.1.2.1 내부 서비스/컴포넌트 의존성

LogMate는 분산 아키텍처를 기반으로 하며, 각 서버 간의 역할과 책임이 명확히 분리되어 있다. 모든 내부 구성요소는 REST API, WebSocket, 또는 Kafka를 통해 연동된다.

- **Streaming Server → AI Server**
  - 로그 분석 요청을 비동기적으로 전달하며, AI Server로부터 이상 탐지 결과를 수신한다.
- **Streaming Server → OpenSearch**
  - 파싱 및 필터링이 완료된 로그를 OpenSearch로 전송하여 저장 및 인덱싱을 수행하며 로그 검색을 수행한다.

- **Streaming Server ↔ Kafka**
    - Kafka는 로그 파이프라인의 핵심 버퍼 역할을 하며, 로그를 Streaming Server Producer 로부터 제공받고, Streaming Server Consumer 에 전달한다. 장애 시 Kafka가 로그를 보존하여 재처리를 가능하게 한다.
- **Frontend → API/Streaming Server**
    - Frontend는 JWT 인증 기반으로 API Server에 REST API 요청을 보낸다.
    - Streaming Server의 WebSocket을 통해 실시간 로그를 구독하며 로그 검색 API 요청을 보낸다.
- **Agent → API Server**
    - Agent는 API Server에 Agent ID, ID, Password 를 통해 인증하고, 주기적으로 설정 정보를 Pulling 한다.
- **Agent → Streaming Server**
    - Agent는 Streaming Server로 로그를 전송한다.

2.1.2.2 외부 인프라

LogMate는 AWS 인프라를 기반으로 배포되며, 다음 외부 구성요소에 의존한다.

- **AWS EC2**
    - 서버 구성요소(API, Streaming, AI)는 각기 독립된 EC2 인스턴스에서 실행된다. 서버 간 통신은 VPC 내부 또는 보안 그룹 정책에 따라 허용된 포트에서만 허용된다.
- **AWS RDS (MySQL)**
    - 사용자, 팀, 대시보드, Agent 설정 데이터 등 주요 비즈니스 데이터를 저장한다. 자동 백업과 스냅샷 복구 기능을 이용하여 데이터 손실을 방지한다.
- **Netlify**
    - Frontend는 Netlify를 통해 정적 웹 애플리케이션 형태로 배포된다.
    - GitHub 리포지토리와 연결된 CI/CD 파이프라인을 통해 코드 변경 시 자동으로 빌드 및 배포된다.
- **Kafka**
    - 로그 스트림 버퍼 및 메시지 큐 역할을 수행한다.
- **OpenSearch**
    - 로그 인덱싱 및 검색을 담당한다.
- **Webhook API**

    - 사용자가 Slack, Discord 등 외부 협업 툴의 Webhook URL을 등록하면, 이벤트 발생 시 해당 외부 시스템으로 POST 요청을 전송한다.
    - Webhook 모듈은 내부 알림 로직과 외부 플랫폼 간의 통신을 담당하는 외부 연동 인터페이스 계층(External Integration Layer) 으로 분류된다
- **GitHub Actions**
  CI/CD 파이프라인을 통해 각 서버의 빌드, 테스트, 배포를 자동화한다. 민감 정보는 GitHub Actions Secrets 으로 관리된다.

2.1.2.3 프로토콜

LogMate의 내부 통신 및 데이터 전송은 보안성과 확장성을 고려하여 다음 프로토콜 기반으로 수행된다.

구분	프로토콜	용도	특징
Agent ↔ API	HTTPS	인증 및 설정 Pull	AgentId, ID, Password 기반 인증, eTag 캐싱 적용
Agent ↔ Streaming	HTTPS	로그 전송	TLS 암호화, 실시간 스트림 처리
Streaming ↔ AI	HTTPS	로그 분석 요청	JSON 기반 요청/응답, 비동기 처리
Streaming ↔ OpenSearch	HTTP	로그 인덱싱	Bulk API 이용, 지연 허용 구조
Streaming ↔ Kafka	TCP (Kafka Binary Protocol)	로그 스트림 버퍼링	고가용성 메시징, 장애 시 재처리 가능

Frontend ↔ API	HTTPS	사용자 요청, 대시보드 관리	JWT 인증, REST 기반
Frontend ↔ Streaming	WSS	실시간 로그 구독	비동기 양방향 통신
Frontend ↔ Streaming	HTTPS	로그 검색	JWT 인증, REST 기반

2.4 Architectural Rationale

2.4.1 전체 아키텍처 선정 근거

LogMate 는 분산형 아키텍처를 기반으로 설계되었다. 이는 로그 수집, 처리, 분석, 시각화 기능을 개별 서비스로 분리하여 유지보수성과 확장성을 확보하기 위함이며 각 서버의 특성에 맞는 프레임워크와 기술스택을 적용하기 위함이다. 이러한 구조를 선택한 주요 이유는 다음과 같다.

2.4.2 기술 스택 선정 근거

- **Spring Boot (API / Streaming)**
    - REST API와 Reactive Stream(WebFlux) 구현에 적합하며, 대규모 비동기 로그 스트림을 효율적으로 처리할 수 있다. 또한, 강력한 DI 및 Validation 기능을 통해 안정적인 서버 운영이 가능하다.
- **FastAPI (AI Server)**
    - Python 생태계를 활용하여 AI 모델을 신속하게 서빙할 수 있다. 경량화된 ASGI 서버 구조로 고속 응답을 지원하며, 모델 변경이나 추가가 용이하다.
- **Isolation Forest**
    - 비지도 학습 기반 이상 탐지 알고리즘으로, 로그 데이터 내 정상/비정상 패턴을 빠르게 탐지할 수 있고 대규모 로그 데이터에서도 트리 기반 구조를 활용해 계산 효율이 높으며, 노이즈나 비정상 샘플이 포함된 환경에서도 강건한 성능을 보인다.
- **Java (Agent)**
    - Agent는 Java로 개발되었으며, JVM 기반으로 어떤 운영체제에서도 동일하게 실행

가능하다. 이는 Windows, macOS, Linux 등 다양한 환경에서의 로그 수집 호환성을 보장한다. 또한, Java는 비교적 러닝 커브가 낮고 풍부한 표준 라이브러리를 제공하여 파일 입출력, 스레드 관리, 네트워크 통신 등 시스템 수준 기능을 쉽게 구현할 수 있다.

- **Kafka (Streaming Backbone)**
    - 대용량 로그 데이터의 버퍼링과 비동기 전송을 위한 핵심 인프라로 채택되었다. 장애 발생 시에도 로그 보존이 가능하며, Consumer Group 구조를 통해 병렬 처리가 가능하다.
- **OpenSearch (로그 저장 및 분석)**
    - 로그 검색과 대시보드 시각화를 위해 OpenSearch를 선택하였다. 전통적인 RDBMS에 비해 대용량 비정형 데이터 처리에 특화되어 있으며, 인덱싱 및 전문 검색(Full-Text Search)에 최적화되어 있다. 또한 Elasticsearch 기반의 오픈소스 솔루션으로 수평 확장성과 빠른 검색 응답 속도를 제공한다..
- **React (Frontend)**
    - SPA(Single Page Application) 구조를 통해 사용자 경험을 개선하고, 실시간 로그 시각화에 필요한 WebSocket 구독을 지원한다.

2.4.3 보안 구성 선정 근거

- **인증 구조 선택**
    - LogMate는 단일 서비스 내에서 관리 가능한 JWT 기반 인증 구조를 채택하였다. 사용자는 로그인 후 JWT 토큰을 발급받아 API 및 Streaming 서버와의 통신에 사용하며, Agent는 API 서버에서 발급받은 Agent ID와 ID, Password를 통해 인증 후 JWT를 획득한다. 이러한 구조는 외부 인증 서버 의존도를 제거하고, 단일 인증 흐름 내에서 서비스 간 신뢰를 확보하기 위함이다.
- **통신 보안**
    - 모든 외부 통신(Agent–API/Streaming, Frontend–API/Streaming)은 TLS(HTTPS/WSS) 암호화를 적용하여 데이터 전송 중 도청 및 변조를 방지한다. 내부 통신(API–DB, Streaming–AI 등)은 AWS 내부 네트워크를 통해 수행되어 외부 노출을 차단한다.
- **비밀 관리 정책**
    - 민감한 자격 증명(API 키, DB 비밀번호 등)은 GitHub Actions Secrets 또는 런타임 환경 변수로 주입된다. 이는 별도의 시크릿 관리 서버 없이도 배포 자동화 파이프라인 내에서 안전한 보안값 전달을 보장한다.
- **네트워크 보호 구조**
    - 각 서버는 AWS EC2 보안 그룹을 통해 격리되어 있으며, 허용된 포트(예: 443, 8080,

9200)만 개방된다. 불필요한 인바운드 트래픽은 차단되며, SSH 접근은 특정 관리자 IP로 제한된다.

- **인증 체계 구조**
    - LogMate는 3단계 인증 흐름을 기반으로 보안을 설계하였다. Frontend에서 사용자 회원가입을 통해 ID, Password를 만들 수 있으며, Agent를 등록하면 Agent ID를 발급한다. Agent는 실행 시 이를 이용해 API 서버로부터 JWT를 발급받고, 이 토큰을 이용하여 Streaming 서버로 로그를 전송한다. Streaming 서버는 검증된 JWT를 통해 인증된 사용자(Frontend 로그인 사용자)에게만 로그를 스트리밍으로 전달한다. 이를 통해 각 계층 간 명시적 신뢰 관계를 형성하고, 외부 공격 표면을 최소화하였다.

2.4.4 배포 및 운영 설계 근거

- **Docker 기반 배포**
    - 서버 구성요소(API, Streaming, AI) 는 Docker 이미지를 통해 빌드 및 배포되며, 환경 간 일관성을 확보한다. 각 서비스는 독립 컨테이너로 관리되어 장애 복구 및 롤백이 용이하다.
- **GitHub Actions CI/CD**
    - 빌드·테스트·배포 전 과정을 자동화하기 위해 GitHub Actions를 채택하였다. 보안 토큰 및 환경 변수는 GitHub Secrets를 통해 안전하게 관리된다.
- **AWS EC2 중심 네트워크 및 보안 설계**
    - LogMate의 서버 구성요소 (API, Streaming, AI)는 개별 AWS EC2 인스턴스에 배포된다. 각 인스턴스는 AWS 보안 그룹(Security Group)을 통해 포트 접근이 제한되며, 불필요한 인바운드 트래픽은 차단된다. 데이터베이스(RDS), Kafka, OpenSearch는 AWS 내부 통신을 통해 연결되어 외부 네트워크로 노출되지 않는다. 이러한 설계는 인스턴스 간 트래픽 제어를 단순화하고, AWS 인프라 수준의 네트워크 보안과 접근 통제 이점을 활용하기 위함이다.

3. Architectural Design (아키텍처 설계)

3.1 System Overview

3.1.1 System Context

LogMate 는 경량 로그 모니터링 플랫폼으로, 다양한 애플리케이션의 로그를 수집 -> 저장 -> 분석/시각화 하는 과정을 지원한다.

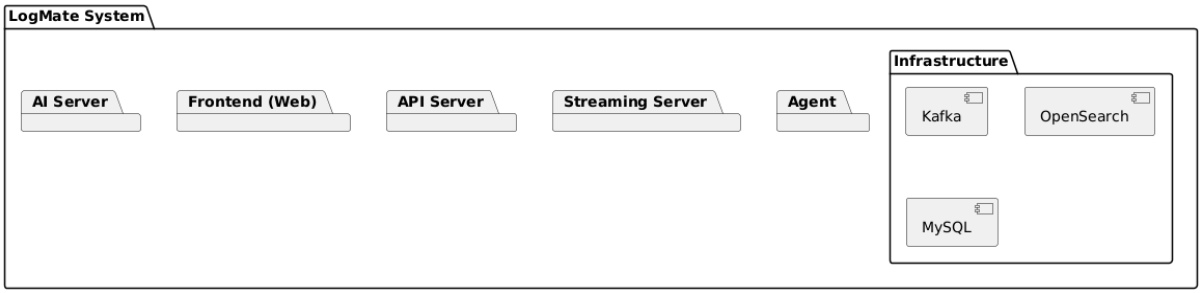
3.1.2 High-Level Architecture

시스템은 크게 다섯 개의 서브시스템과 공용 인프라로 구성된다.

1. **Agent**
    - 애플리케이션/시스템 로그 파일을 실시간으로 수집하고, 표준 포맷으로 변환 후 전송한다.
    - 필터링, 멀티라인 처리, 실패 시 로컬 버퍼링 기능 등을 제공한다.
2. **Streaming Server**
    - Kafka 메시지를 소비하여 로그를 처리하고, 필요 시 AI Server 에 전달한다.
    - 결과 로그를 OpenSearch 에 저장하고, 동시에 WebSocket 을 통해 Frontend 에 브로드캐스트한다.
    - OpenSearch 에 저장된 로그를 검색하는 기능을 제공한다.
3. **API Server**
    - 사용자 인증/인가, 팀대시보드 관리, 파싱/필터 규칙 관리 등의 REST API 를 제공한다.
    - MySQL 을 저장소로 이용한다.
4. **Frontend**
    - React 기반 SPA로 WebSocket 을 통해 스트리밍 로그를 실시간 표시한다.
    - 대시보드/검색 UI, 룰 관리 화면 등 사용자 UI 를 제공한다.
5. **AI Server**
    - Isolation Forest 등 모델을 통해 로그 이벤트의 이상 여부를 점수화 한다.
    - Streaming Server 로부터 요청을 받아 결과를 반환한다.
6. **공용 인프라**
    - **Kafka:** 로그 메시지 큐잉 및 비동기 이벤트 처리
    - **OpenSearch:** 로그 색인/검색/시각화
    - **MySQL:** 사용자, 팀, 대시보드, Agent 설정 등 저장

3.2 Subsystem Decomposition (분해 뷰)

3.2.1 Top-level Decomposition



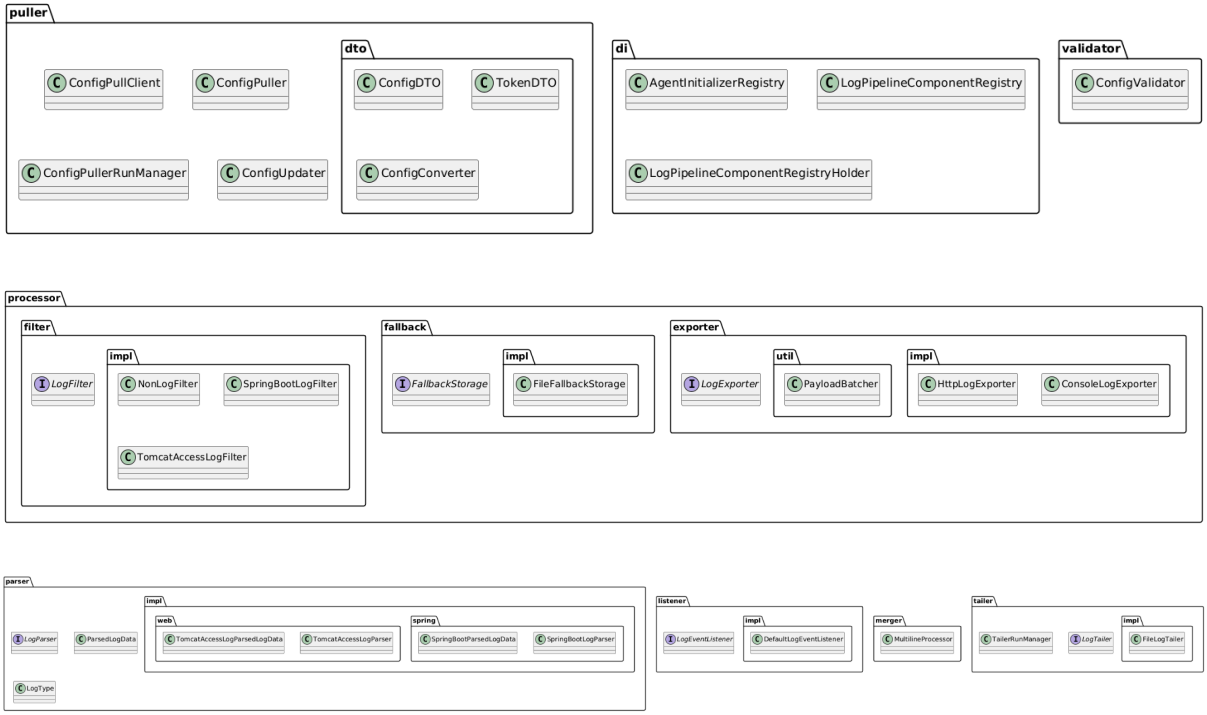
Identification	Type	Purpose	Function
Agent	Service	로컬 로그 수집·표준화·전송	- 로그 파일 Tailing - 파싱/필터링 - 멀티라인 처리 - 실패 시 로컬 버퍼링 - HTTPS/Kafka 전송
Streaming Server	Reactive Service	실시간 로그 처리 및 스트리밍	- Kafka 메시지 소비 - ProcessorChain 실행 - AI Server 호출 - OpenSearch 저장, 조회 - WebSocket 브로드캐스트
API Server	REST Service	사용자 인증/설정 관리/로그 검색·알림	- JWT 인증/인가 - 사용자, 팀, 폴더, 대시보드 관리 - Webhook 알림 - agent server 대시보드 설정 pulling 지원

Frontend (Web)	SPA (Web App)	사용자 UI 및 실시간 로그 시각화	- WebSocket 구독 - 실시간 로그 뷰 렌더링 - 대시보드/검색 UI - 규칙 관리 UI
AI Server	Service	로그 이상 탐지 및 점수화	- Feature 추출 - ML 모델 기반 Inference - Score 반환
Infrastructure	Infra	공용 데이터 처리 및 저장소	- Kafka: 로그 이벤트 메시징 - OpenSearch: 로그 색인/검색 - MySQL: 메타데이터 저장

3.2.2 Component-level Decomposition

3.2.2.1 Agent

3.2.2.1.1 Component Diagram



3.2.2.1.2 Component Description

Identification	Type	Purpose	Function	Subordinates
AgentArguments	Class	실행 인자 DTO	프로그램 시작 시 전달된 args를 보관	—
ArgsExtractor	Class	실행 인자 파서	CLI 인자를 파싱해 AgentArguments로 변환	—
AgentAuthenticator	Class	Agent 인증 수행	Agent ID/Token을 기반으로 인증 서버 검증	AuthClient, AuthToken
AuthClient	Class	인증	REST 호출을	—

		서버 클라이언트	통해 인증 요청 수행	
AuthenticationRequestDTO	Class	인증 요청 DTO	인증 API 호출 시 사용되는 요청 데이터	—
AuthToken	Class	인증 토큰	Agent 인증 후 발급되는 토큰 객체	—
ConfigLoader	Interface	설정 로딩 추상화	외부/로컬 설정 파일 로딩 규약 정의	YamlConfigLoader
YamlConfigLoader	Class	YAML 설정 로더	YAML 기반 설정 파일 로드	—
ConfigInitializer	Class	설정 초기화	로딩된 설정의 유효성 검사 및 초기화	—
AgentInitializer	Class	Agent 초기화	실행 시 args, config, 인증 초기화 orchestrator	—
ExporterConfig	Class	Exporter 설정	로그 Exporter 동작 설정	—
FallbackStorageConfig	Class	Fallback 저장소 설정	로컬 실패 로그 저장소 동작 정의	—
FilterConfig	Class	필터 설정	로그 필터링 동작 정의	—

LogPipelineConfig	Class	파이프라인 설정	파이프라인 실행 순서 정의	—
MultilineConfig	Class	멀티라인 설정	스택트레이스 등 멀티라인 병합 정책 정의	—
ParserConfig	Class	파서 설정	로그 파서 동작 정의	—
AgentConfig	Class	Agent 전역 설정	Agent 전체 실행 환경 정의	—
PullerConfig	Class	Puller 설정	Config Puller의 주기, eTag 등 동작 정의	—
TailerConfig	Class	Tailer 설정	파일 tailing 동작 정의	—
AgentConfigHolder	Class	Agent 설정 홀더	로딩된 AgentConfig를 전역 보관	—
LogPipelineConfigHolder	Class	파이프라인 설정 홀더	LogPipelineConfig를 전역 보관	—
PullerConfigHolder	Class	Puller 설정 홀더	PullerConfig를 전역 보관	—
ConfigDTO	Class	Config DTO	Config 서버와 주고받는 데이터 객체	—

TokenDTO	Class	Token DTO	인증 토큰 교환용 데이터 객체	—
ConfigConverter	Class	Config 변환기	수신된 설정 JSON→객체 변환	—
ConfigPullClient	Class	Config 서버 클라이언트	설정 서버와 통신	—
ConfigPuller	Class	Config Pull 실행	주기적으로 Config 서버에서 설정을 pull	ConfigPullerRunManager, ConfigUpdater
ConfigPullerRunManager	Class	Config Puller 실행 관리	Puller 스레드 실행 제어	—
ConfigUpdater	Class	설정 업데이트기	수신된 설정을 무중단 반영	—
ConfigValidator	Class	설정 유효성 검사기	로딩된 설정의 유효성 검증	—
AgentInitializerRegistry	Class	초기화 레지스트리	초기화 관련 컴포넌트 관리	—
LogPipelineComponentRegistry	Class	파이프라인 컴포넌트 레지스	LogPipeline 등록 및 관리	LogPipelineComponentRegistryHolder

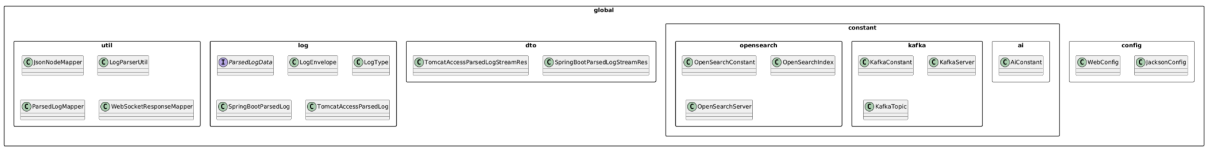
		트리		
LogExporter	Interface	로그 Export 추상화	로그 Exporter 규약 정의	ConsoleLogExporter, HttpLogExporter
ConsoleLogExporter	Class	Console Exporter	로그를 콘솔에 출력	—
HttpLogExporter	Class	HTTP Exporter	로그를 HTTP API 서버에 전송	—
PayloadBatcher	Class	배치 유틸	로그 전송 시 배치 묶음 관리	—
FallbackStorage	Interface	Fallback 저장소 추상화	실패 로그 저장 규약 정의	FileFallbackStorage
FileFallbackStorage	Class	Fallback 저장소 구현	로컬 파일로 로그 저장	—
LogFilter	Interface	로그 필터링 추상화	로그 수락/거부 규약 정의	NonLogFilter, SpringBootLogFilter, TomcatAccessLogFilter
NonLogFilter	Class	Non-Log 필터	로그 형식이 아닌 데이터 제거	—
SpringBootLogFilter	Class	SpringBoot 로그 필터	Spring Boot 로그만 필터링	—
TomcatAccessLogFilter	Class	Tomcat Access 로그 필터	Tomcat Access 로그만 필터링	—

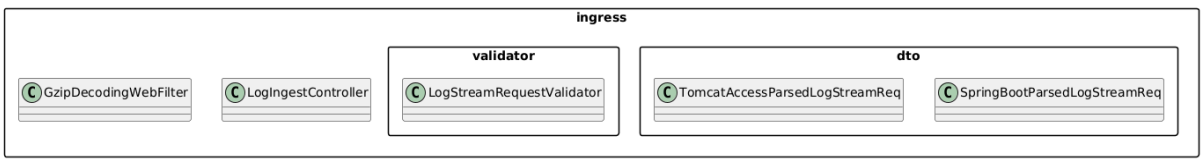
LogEventListener	Interface	로그 이벤트 리스너	로그 이벤트 구독 규약 정의	DefaultLogEventListener
DefaultLogEventListener	Class	기본 로그 이벤트 리스너	기본 이벤트 구독 및 처리	—
MultilineProcessor	Class	멀티라인 병합기	멀티라인 로그를 단일 이벤트로 병합	—
LogParser	Interface	로그 파서 추상화	로그 문자열→객체 변환 규약 정의	SpringBootLogParser, TomcatAccessLogParser
SpringBootLogParser	Class	SpringBoot 로그 파서	Spring Boot 로그를 구조화 객체로 변환	SpringBootParsedLogData
SpringBootParsedLogData	Class	SpringBoot 로그 데이터	파싱된 Spring Boot 로그 결과	—
TomcatAccessLogParser	Class	Tomcat 로그 파서	Tomcat Access 로그를 구조화 객체로 변환	TomcatAccessLogParsedLogData
TomcatAccessLogParsedLogData	Class	Tomcat 로그 데이터	파싱된 Tomcat Access 로그 결과	—
ParsedLogData	Class	공용 로그 데이터	공통 파싱 데이터 모델	—

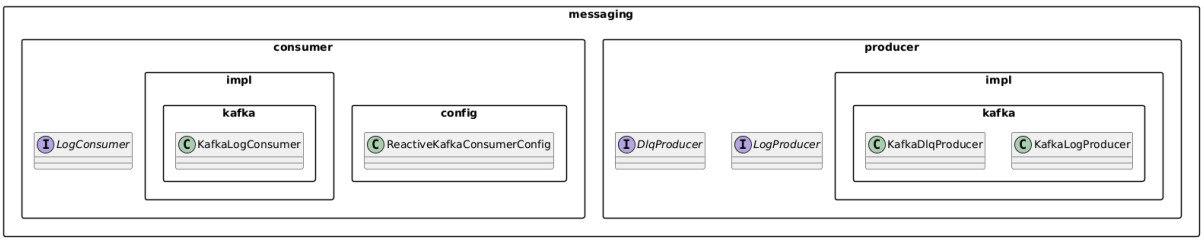
LogType	Enum	로그 유형	로그 종류(SpringBoot/Tomcat) 정의	—
LogTailer	Interface	로그 Tailer 추상화	파일 tailing 규약 정의	FileLogTailer
FileLogTailer	Class	파일 Tailer	파일을 실시간 tailing	—
TailerRunManager	Class	Tailer 실행 관리	Tailer 스레드 실행 제어	—
Main	Class	Agent 실행 진입점	Agent 어플리케이션 실행	—

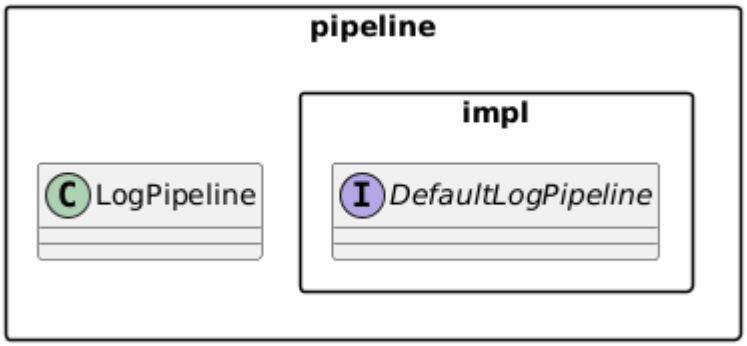
3.2.2.2 Streaming Server

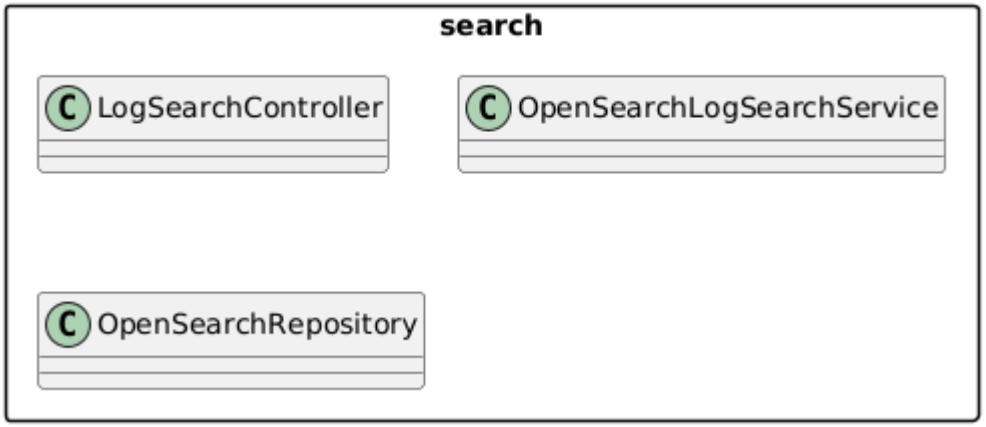
3.2.2.2.1 Component Diagram











3.2.2.2.2 Component Description

Identification	Type	Purpose	Function	Subordinates
LogConsumer	Interface	로그 소비를 위한 표준화 인터페이스	Kafka Consumer의 공통 기능 정의	KafkaLogConsumer
KafkaLogConsumer	Class	Kafka 메시지 소비 구현체	Kafka 토픽 구독 및 로그 이벤트 처리	—

ReactiveKafkaConsumerConfig	Class	Kafka Consumer 환경 설정	Reactive Kafka Consumer Bean 및 설정 관리	—
LogIngestController	Class	외부 로그 인입 관리	HTTP 엔드포인트를 통해 로그를 수신	—
LogPipeline	Interface	로그 파이프라인 처리 정의	로그 표준화 및 파이프라인 실행 순서 정의	DefaultLogPipeline
DefaultLogPipeline	Class	로그 파이프라인 기본 구현	LogProcessor 체인을 실행	—
LogProcessor	Interface	로그 처리 단계 정의	파싱, 필터링, AI 분석, 저장 등 처리 로직 정의	LogProcessorRegistry, AiAnalyzeTomcatAccessLogProcessor, LogStorageProcessor, WebSocketProcessor
LogProcessorRegistry	Class	Processor 관리	등록된 Processor를 순서대로 실행	—
AiClient	Class	AI 서버 연동	로그를 AI 서버에 전송하고 응답 점수 수신	AiClientConfig, AiAnalyzeTomcatAccessLogRequest, AiResponse

AiClientConfig	Class	AI Client 설정	AI 서버 연동을 위한 WebClient Bean 생성	—
AiAnalyzeTomcatAccessLogRequest	Class	AI 분석 요청 DTO	Tomcat Access 로그를 분석 요청할 때 사용	—
AiResponse	Class	AI 분석 응답 DTO	AI 서버로부터 받은 분석 결과 캡슐화	—
AiAnalyzeTomcatAccessLogProcessor	Class	AI 로그 처리기	Tomcat Access 로그를 AI 서버에 전송하여 분석 점수를 부여	—
OpenSearchConfig	Class	OpenSearch 연결 설정	OpenSearch 클러스터 연결 환경 설정	—
LogStorageService	Interface	로그 저장 추상화	로그 저장 동작 정의	OpenSearchLogStorageService
OpenSearchLogStorageService	Class	OpenSearch 저장 구현체	로그를 OpenSearch 인덱스에 저장	—
LogStorageProcessor	Class	저장	로그를	—

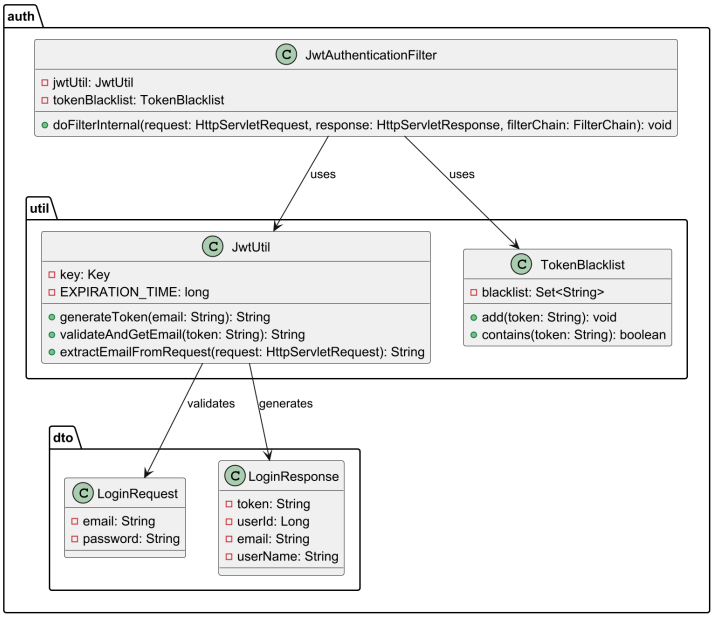
		Processor	LogStorageService를 통해 저장	
WebSocketRouteConfig	Class	WebSocket 라우팅 설정	WebSocket 엔드포인트 매핑	—
LogWebSocketHandler	Class	WebSocket 로그 핸들러	로그 스트리밍 세션 관리 및 메시지 송신	—
WebSocketProcessor	Class	WebSocket 처리 Processor	로그를 WebSocket Broadcaster로 전달	—
LogProducer	Interface	로그 메시지 프로듀서 정의	Kafka 전송 동작 정의	KafkaLogProducer
DlqProducer	Interface	DLQ 프로듀서 정의	실패 로그를 Dead Letter Queue로 전송	KafkaDlqProducer
KafkaLogProducer	Class	Kafka 로그 프로듀서	Kafka 토픽으로 로그 이벤트 전송	—
KafkaDlqProducer	Class	Kafka DLQ 프로듀서	Kafka DLQ 토픽으로 로그 전송	—
JacksonConfig	Class	Jackson	JSON	—

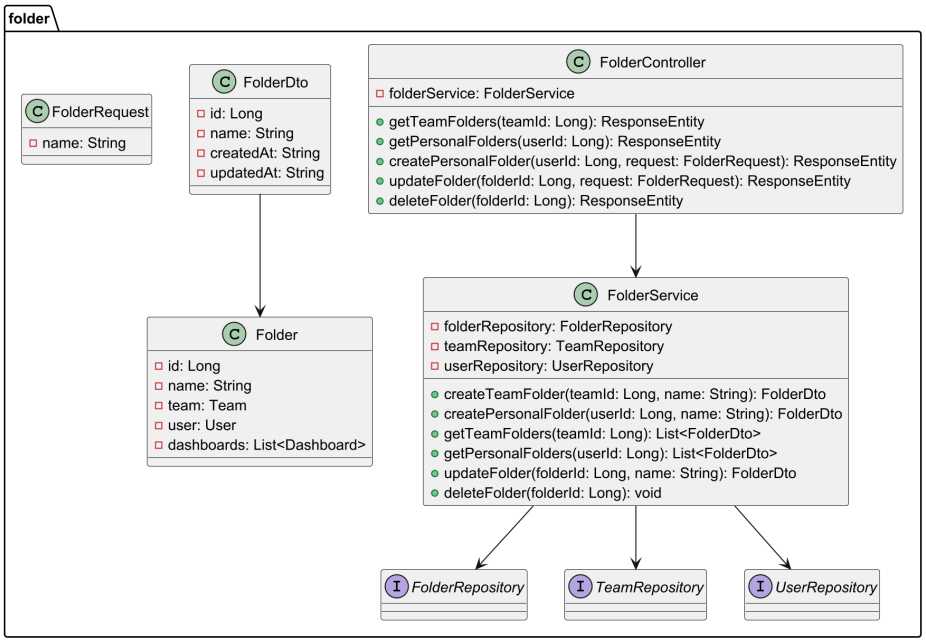
		직렬화 설정	직렬화/역직렬화 설정	
KafkaConstant / KafkaServer / KafkaTopic	Class	Kafka 상수 및 설정	Kafka 서버/토픽 관련 상수 정의	—
OpenSearchConstant / OpenSearchIndex / OpenSearchServer	Class	OpenSearch 상수 및 설정	OpenSearch 관련 상수 정의	—
AiConstant	Class	AI 관련 상수	AI 서버와 연동 시 필요한 상수 관리	—
LogEnvelope / LogType / SpringBootParsedLog / TomcatAccessParsedLog	Class	로그 모델	로그 데이터 타입 및 파싱 결과 정의	—
LogParserUtil	Class	로그 파서 유틸리티	로그 파싱 관련 공용 함수 제공	—
LogmateStreamingApplication	Class	Streaming Server 메인 엔트리	Spring Boot 애플리케이션 실행	—
LogSearchController	Class	로그 조회 API 컨트롤러	API 엔드포인트 제공	-
OpenSearchLogSearchService	Class	로그 조회 서비스	로그 조회 서비스 함수 제공	-

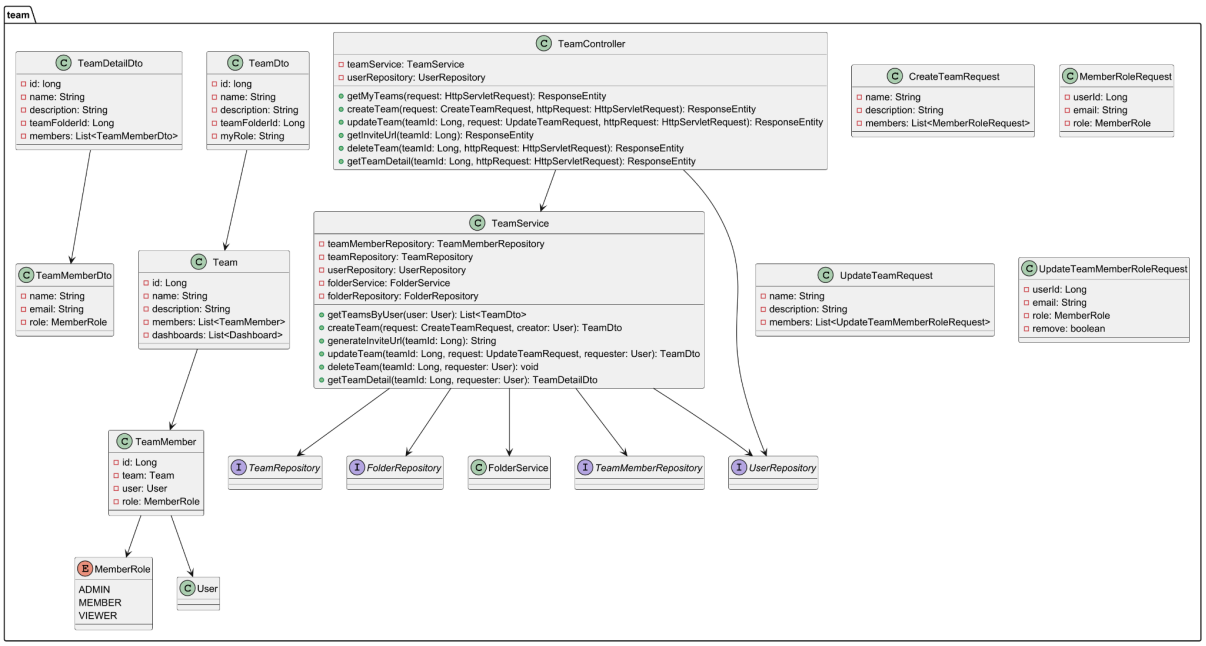
OpenSearchRepository	Class	OpenSearch 로그 조회 쿼리 작성 및 쿼리 요청	OpenSearch 쿼리 함수 제공	-

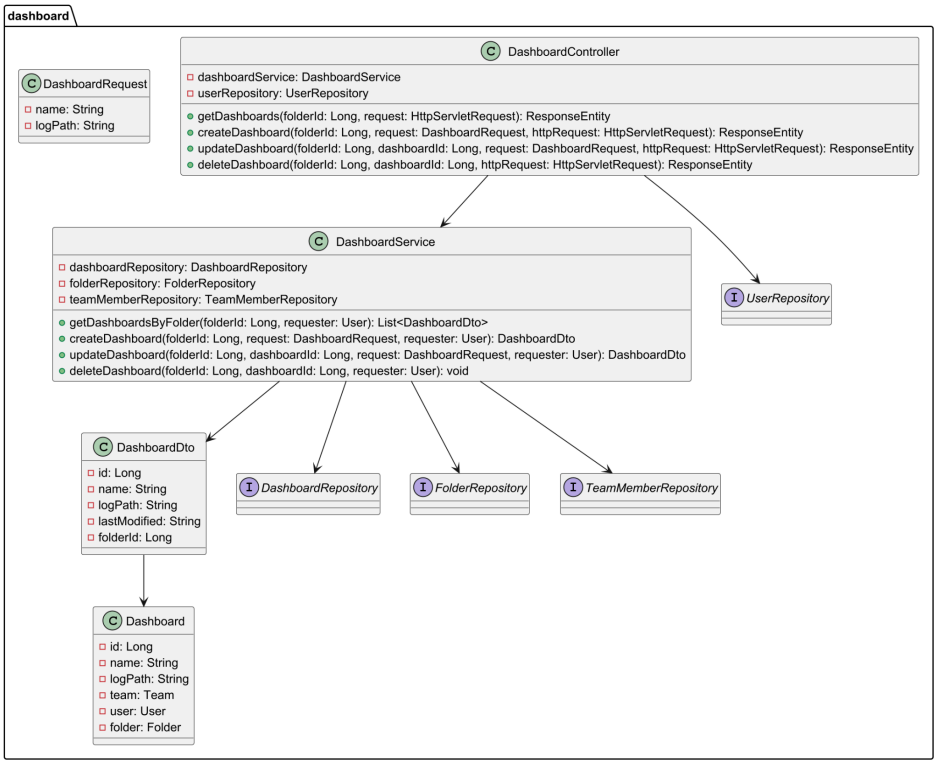
3.2.2.3 API Server

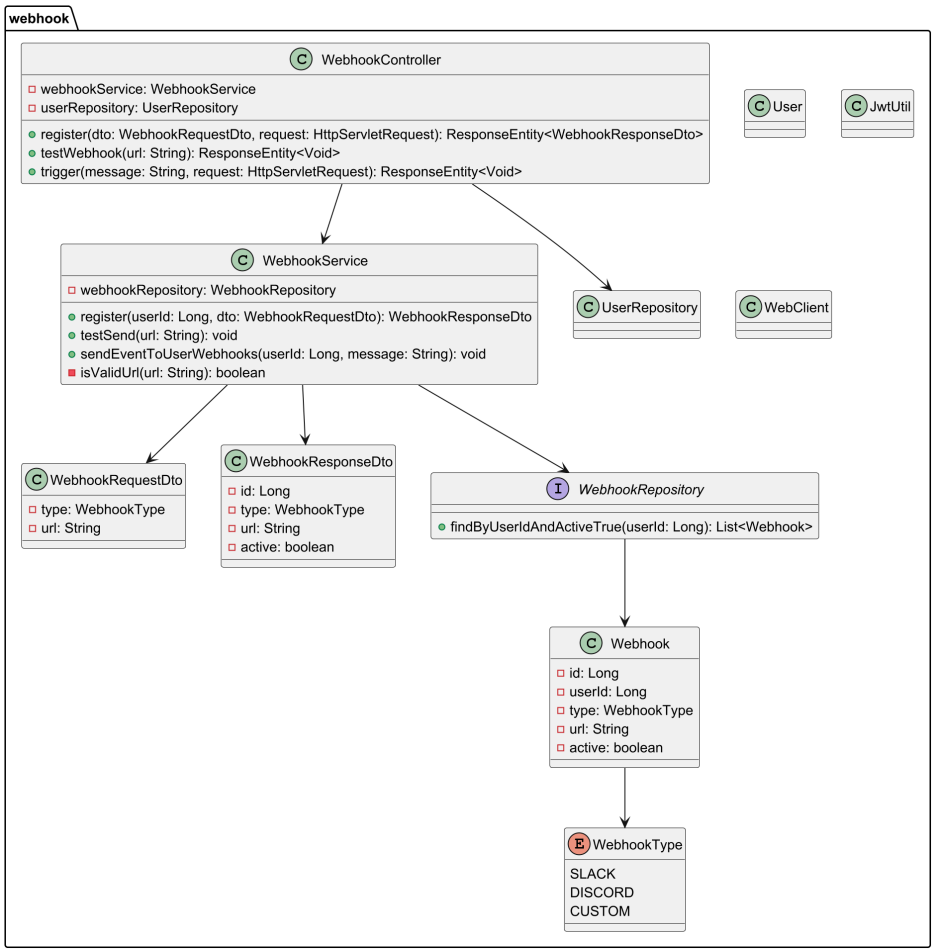
3.2.2.3.1 Component Diagram

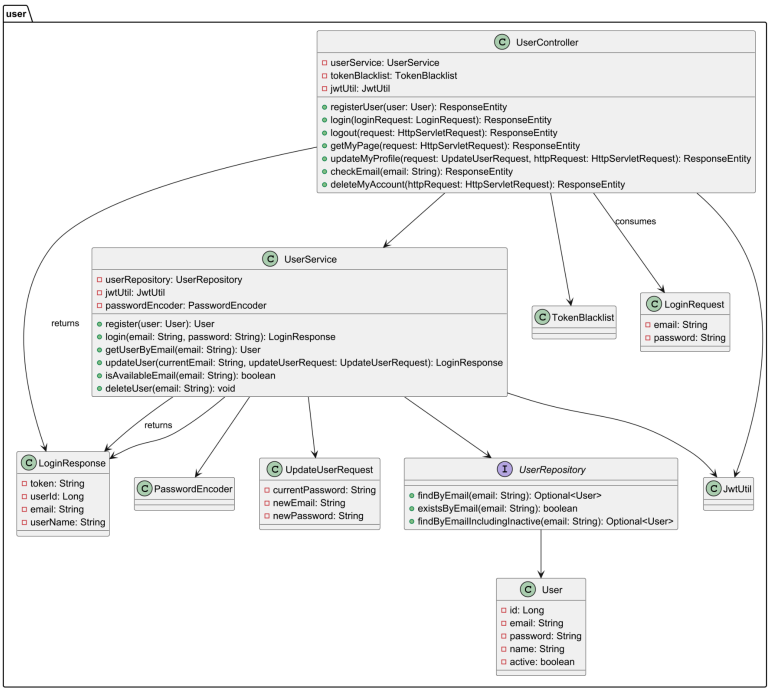


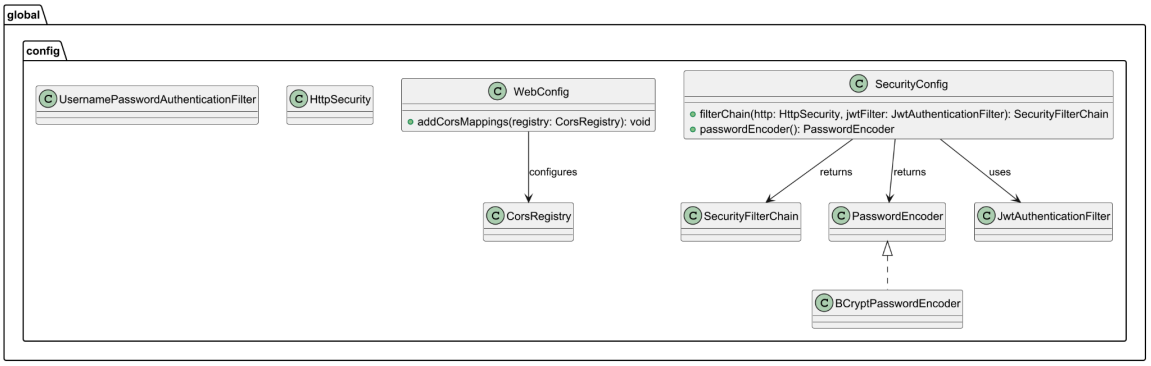

folder
FolderRequest
name: String
FolderDto
id: Long
name: String
createdAt: String
updatedAt: String
FolderController
folderService: FolderService
getTeamFolders(teamId: Long): ResponseEntity
getPersonalFolders(userId: Long): ResponseEntity
createPersonalFolder(userId: Long, request: FolderRequest): ResponseEntity
updateFolder(folderId: Long, request: FolderRequest): ResponseEntity
deleteFolder(folderId: Long): ResponseEntity
Folder
id: Long
name: String
team: Team
user: User
dashboards: List<Dashboard>
FolderService
folderRepository: FolderRepository
teamRepository: TeamRepository
userRepository: UserRepository
createTeamFolder(teamId: Long, name: String): FolderDto
createPersonalFolder(userId: Long, name: String): FolderDto
getTeamFolders(teamId: Long): List<FolderDto>
getPersonalFolders(userId: Long): List<FolderDto>
updateFolder(folderId: Long, name: String): FolderDto
deleteFolder(folderId: Long): void
FolderRepository
TeamRepository
UserRepository

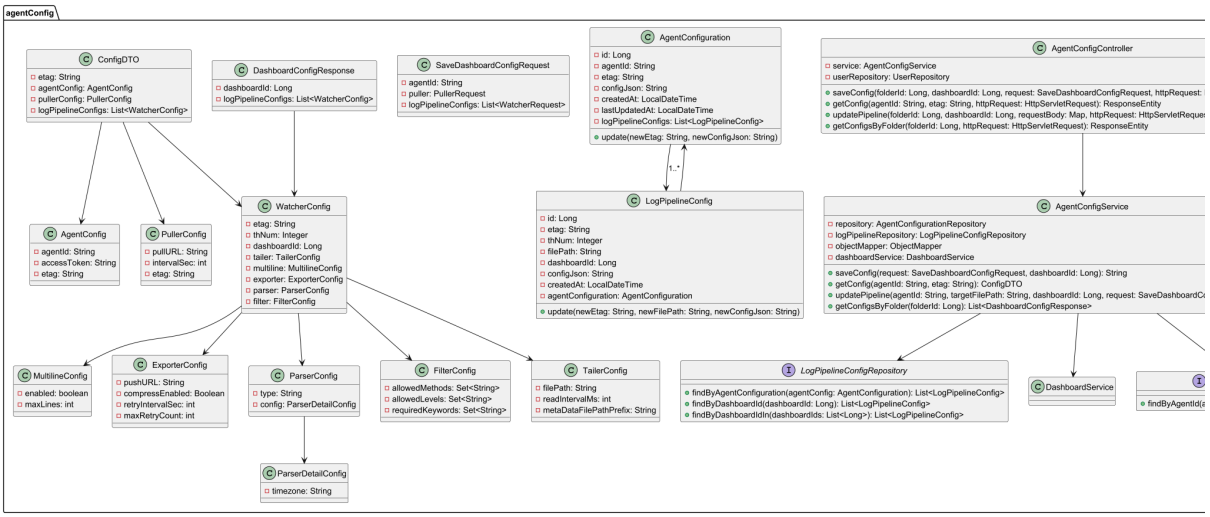

team
TeamDetailDto
id: long
name: String
description: String
teamFolderId: Long
members: List<TeamMemberDto>
TeamDto
id: long
name: String
description: String
teamFolderId: Long
myRole: String
TeamController
teamService: TeamService
userRepository: UserRepository
getMyTeams(request: HttpServletRequest): ResponseEntity
createTeam(request: CreateTeamRequest, httpRequest: HttpServletRequest): ResponseEntity
updateTeam(teamId: Long, request: UpdateTeamRequest, httpRequest: HttpServletRequest): ResponseEntity
getInviteUrl(teamId: Long): ResponseEntity
deleteTeam(teamId: Long, httpRequest: HttpServletRequest): ResponseEntity
getTeamDetail(teamId: Long, httpRequest: HttpServletRequest): ResponseEntity
CreateTeamRequest
name: String
description: String
members: List<MemberRoleRequest>
MemberRoleRequest
userId: Long
email: String
role: MemberRole
TeamMemberDto
name: String
email: String
role: MemberRole
Team
id: Long
name: String
description: String
members: List<TeamMember>
dashboards: List<Dashboard>
TeamService
teamMemberRepository: TeamMemberRepository
teamRepository: TeamRepository
userRepository: UserRepository
folderService: FolderService
folderRepository: FolderRepository
getTeamsByUser(user: User): List<TeamDto>
createTeam(request: CreateTeamRequest, creator: User): TeamDto
generateInviteUrl(teamId: Long): String
updateTeam(teamId: Long, request: UpdateTeamRequest, requester: User): TeamDto
deleteTeam(teamId: Long, requester: User): void
getTeamDetail(teamId: Long, requester: User): TeamDetailDto
UpdateTeamRequest
name: String
description: String
members: List<UpdateTeamMemberRoleRequest>
UpdateTeamMemberRoleRequest
userId: Long
email: String
role: MemberRole
remove: boolean
TeamMember
id: Long
team: Team
user: User
role: MemberRole
TeamRepository
FolderRepository
FolderService
TeamMemberRepository
UserRepository
MemberRole
ADMIN
MEMBER
VIEWER
User


dashboard
DashboardController
- dashboardService: DashboardService
- userRepository: UserRepository
+ getDashboards(folderId: Long, request: HttpServletRequest): ResponseEntity
+ createDashboard(folderId: Long, request: DashboardRequest, httpRequest: HttpServletRequest): ResponseEntity
+ updateDashboard(folderId: Long, dashboardId: Long, request: DashboardRequest, httpRequest: HttpServletRequest): ResponseEntity
+ deleteDashboard(folderId: Long, dashboardId: Long, httpRequest: HttpServletRequest): ResponseEntity
DashboardRequest
- name: String
- logPath: String
DashboardService
- dashboardRepository: DashboardRepository
- folderRepository: FolderRepository
- teamMemberRepository: TeamMemberRepository
+ getDashboardsByFolder(folderId: Long, requester: User): List<DashboardDto>
+ createDashboard(folderId: Long, request: DashboardRequest, requester: User): DashboardDto
+ updateDashboard(folderId: Long, dashboardId: Long, request: DashboardRequest, requester: User): DashboardDto
+ deleteDashboard(folderId: Long, dashboardId: Long, requester: User): void
UserRepository
DashboardDto
- id: Long
- name: String
- logPath: String
- lastModified: String
- folderId: Long
DashboardRepository
FolderRepository
TeamMemberRepository
Dashboard
- id: Long
- name: String
- logPath: String
- team: Team
- user: User
- folder: Folder


webhook
WebhookController
- webhookService: WebhookService
- userRepository: UserRepository
+ register(dto: WebhookRequestDto, request: HttpServletRequest): ResponseEntity<WebhookResponseDto>
+ testWebhook(url: String): ResponseEntity<Void>
+ trigger(message: String, request: HttpServletRequest): ResponseEntity<Void>
User
JwtUtil
WebhookService
- webhookRepository: WebhookRepository
+ register(userId: Long, dto: WebhookRequestDto): WebhookResponseDto
+ testSend(url: String): void
+ sendEventToUserWebhooks(userId: Long, message: String): void
- isValidUrl(url: String): boolean
UserRepository
WebClient
WebhookRequestDto
- type: WebhookType
- url: String
WebhookResponseDto
- id: Long
- type: WebhookType
- url: String
- active: boolean
WebhookRepository
+ findByUserIdAndActiveTrue(userId: Long): List<Webhook>
Webhook
- id: Long
- userId: Long
- type: WebhookType
- url: String
- active: boolean
WebhookType
SLACK
DISCORD
CUSTOM







3.2.2.3.2 Component Description

Identification	Type	Purpose	Function	Subordinates
JwtUtil	Class	JWT 토큰 발급 및 검증 유틸	generateToken(), validateAndGetEmail(), extractEmailFromRequest()	LoginResponse, LoginRequest
TokenBlacklist	Class	로그아웃된 토큰 관리	add(), contains()	
JwtAuthenticationFilter	Class(Filter)	JWT 인증 필터링	doFilterInternal() - 토큰 검증, SecurityContext 세팅	JwtUtil, TokenBlacklist
LoginRequest/ LoginResponse	DTO	로그인 요청, 응답 데이터	email, password/ Token, userID, email, userName	
UserController	Class	회원가입, 로그인, 로그아웃, 마이페이지 관리	registerUser(), login(), logout(), getMyPage(), updateMyProfile(), deleteMyAccount()	UserService, TokenBlacklist, JwtUtil

UserService	Class	사용자 등록/인증/수정/삭제 비즈니스 로직	register(), login(), updateUser(), deleteUser(), getUserByEmail()	UserRepository, JwtUtil, PasswordEncoder
UserRepository	Interface	사용자 데이터 접근	findByEmail(), existsByEmail(), findByEmailIncludingInactive()	User
User	Entity	사용자 정보 저장	id, email, password, name, active	BaseEntity
UpdateUserRequest	DTO	사용자 정보 수정 요청	currentPassword, newEmail, newPassword	
TeamController	Class	팀 생성/조회/수정/삭제 API 제공	getMyTeams(), createTeam(), updateTeam(), deleteTeam(), getTeamDetail()	TeamService, UserRepository
TeamService	Class	팀관리 및 멤버 관리 로직	getTeamsByUser(), createTeam(), updateTeam(), deleteTeam(), getTeamDetail()	TeamRepository, TeamMemberRepository, FolderService
Team	Entity	팀 정보 저장		BaseEntity

			id, name, description, members, dashboards	
TeamMember	Entity	팀-사용자 관계, 역할 저장	team, user, role	Team, User
TeamRepository	Interface	팀 데이터 접근	save(), findById()	Team
TeamMemberRepository	Interface	팀 멤버 데이터 접근	findByUser(), findByUserIdAndTeamId()	TeamMember
MemberRole	Enum	팀 내 사용자 권한 구분	ADMIN, MEMBER, VIEWER	
TeamDto/TeamDetailDto/ TeamMemberDto/ CreateTeamRequest/ UpdateTeamRequest	DTO	팀 관련 요청, 응답 객체들		
DashboardController	Class	대시보드 CRUD API	getDashboards(), createDashboard(), updateDashboard(), deleteDashboard()	DashboardService, UserRepository
DashboardService	Class	대시보드 생성/조회/수정/삭제	getDashboardsByFolder(), createDashboard(),	DashboardRepository, FolderRepository, TeamMemberRepository

			updateDashboard(), deleteDashboard()	
Dashboard	Entity	대시보드 데이터 저장	Id, name, logpath, folder, user, team	BaseEntity
DashboardRepository	Interface	대시보드 DB 접근	findByFolderId(), findIdsByFolderId()	Dashboard
DashboardDto/ DashboardRequest	DTO	대시보드 응답 데이터/ 대시보드 생성 및 수정 요청	id, name, logPath, lastModified, folderId/ name, logPath	
FolderController	Class	폴더 CRUD API	getTeamFolders(), getPersonalFolders(), createPersonalFolder(),	FolderService

			updateFolder(), deleteFolder()	
FolderService	Class	폴더 생성/조회/수정/삭제	createTeamFolder(), createPersonalFolder(), getTeamFolders(), getPersonalFolders(), updateFolder(), deleteFolder()	FolderRepository, TeamRepository, UserRepository
Folder	Entity	폴더 데이터 저장	id, name, team, user, dashboards	BaseEntity
FolderRepository	Interface	폴더 DB 접근	findByTeamIdAndStatus(), findByUserIdAndStatus()	Folder
FolderDto/ FolderRequest	DTO	폴더 요청, 응답 데이터	id,name, createdAt, updatedAt/ name	
WebhookController	Class	Webhook 등록/테스트/트리거 API	register(), testWebhook(), trigger()	WebhookService, UserRepository
WebhookService	Class	Webhook 관리 및 호출 로직	register(), testSend(), sendEventToUserWebhooks()	WebhookRepository

Webhook	Entity	Webhook 데이터 저장	id, userId, type, url, active	
WebhookType	Enum	Webhook 타입 정의	SLACK, DISCORD, CUSTOM	
WebhookRepository	Interface	Webhook DB 접근	findByUserIdAndActiveTrue()	Webhook
WebhookRequestDto/ WebhookResponseDto	DTO	Webhook 등록 요청/ 응답	type, url/ Id, type, url, active	

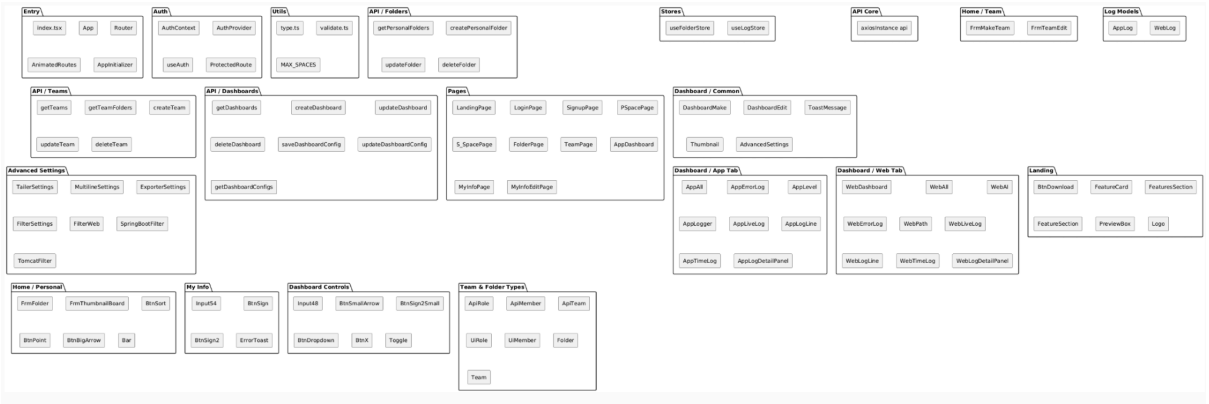
		데이터		
AgentConfigController	Class	대시보드 고급설정 API	saveConfig(), getConfig(), updatePipeline(), getConfigsByFolder()	AgentConfigService, UserRepository
AgentConfigService	Class	AgentConfig 관리 및 LogPipeline 관리	saveConfig(), getConfig(), updatePipeline(), getConfigsByFolder()	AgentConfigurationRepository, LogPipelineConfigRepository, DashboardService
AgentConfiguration	Entity	Agent 설정 저장	Id, agentId, etag, configJson, logPipelineConfigs	LogPipelineConfig

LogPipelineConfig	Entity	개별 파이프라인 설정 저장	Id, etag, thNum, filePath, configJson, dashboardId	AgentConfiguration
AgentConfigurationRepository	Interface	AgentConfiguration DB 접근	findByAgentId()	AgentConfiguration
LogPipelineConfigRepository	Interface	LogPipelineConfig DB 접근	findByDashboardIdIn(), findByAgentConfiguration()	LogPipelineConfig
ConfigDTO/ DashboardConfigResponse/ SaveDashboardConfigRequest	DTO	Agnet 설정 응답 DTO/ 대시보드별 파이프라인 응답 DTO/ 고급정보 설정 저장 요청 DTO	Etag, agentConfig, pullerConfig, logPipelineConfigs/ dashboardId, logPipelineConfigs/ agentId, puller, logPipelineConfigs	AgentConfig, PullerConfig, WatcherConfig/ WatcherConfig/ TailerRequest, MultilineRequest, ExporterRequest, FilterRequest, ParserConfigRequest

WatcherConfig	Config Object	로그 수집기 단위 설정	etag, thNum, tailer, multiline, exporter, parser, filter	TailerConfig, MultilineConfig, ExporterConfig, ParserConfig, FilterConfig
TailerConfig/ MultilineConfig/ ExporterConfig/ ParserConfig/ FilterConfig/ PullerConfig	Config Object	Agent 세부 수집 설정	filePath, readInterval, compress, parser type, filter rules 등	

3.2.2.4 Frontend Server

3.2.2.4.1 Component Diagram



3.2.2.4.2 Component Description

Identification	Type	Purpose	Function	Subordinates
axioslnstace	module	벡엔드 API 통신 유틸리티	Axios.create 로 인스턴스를 생성하여 기본 URL 설정, 요청마다 토큰 자동 첨부(로그인/회원가입 제외)	Request Interceptor
dashboard	Module	대시보드 API 유틸리티	대시보드 생성, 조회, 수정, 삭제 및 고급설정 관리	createDashboard, getDashboards, getDashboardConfigs, updateDashboard, deleteDashboard, saveDashboardConfig, updateDashboardConfig
folders	Modul	폴더 API 유틸리티	개인/팀 폴더 생성, 조회,	createPersonalFolder,

	e		수정, 삭제 기능 제공	getPersonalFolders, updateFolder, deleteFolder
team	Module	팀 API 유틸리티	팀 생성, 조회, 상세조회, 폴더조회, 수정, 삭제 기능 제공	createTeam, getTeams, getTeamDetail, getTeamFolders, updateTeam, deleteTeam
user	Module	사용자 인증 API 유틸리티	회원가입 및 로그인 요청 처리, 로그인 시 토큰 및 사용자 정보 로컬스토리지 저장	Signup, login
btn-add	Component	+ 버튼 컴포넌트	클리시 onclick 이벤트 실행 가능	—
btn-addmember	Component	팀원 추가 버튼 컴포넌트	클릭시 onclick 실행, 비활성화 가능	—
btn-big-arrow	Component	방향 화살표 버튼 컴포넌트	방향(up, down, left, right) 및 스타일(default, variant2) 지정 가능	—
btn-checkbox	Component	체크박스 버튼 컴포넌트	체크 상태(checked) 토글 on/off 이벤트	—
btn-delete	Comp	삭제	클릭시	—

	onent	버튼 컴포넌트	onclick 실행, nudgeY로 텍스트 위치 보정	
btn-dropdown	Component	드롭다운 버튼 컴포넌트	옵션 목록을 열고 닫을 수 있으며, 선택/해제 시 onSelect 이벤트 실행	—
btn-minus	Component	마이너스 버튼 컴포넌트	클릭시 onClick 실행가능	—
btn-more-text	Component	옵션 텍스트 선택 컴포넌트	옵션목록 표시, 선택 시 onSelect 실행 및 onClose 호출	—
btn-more	Component	더보기 버튼 컴포넌트	클릭시 onClick 실행	—
btn-plus	Component	플러스 버튼 컴포넌트	클릭시 onClick 실행 가능	—
btn-point	Component	강조 포인트 버튼 컴포넌트	Children 렌더링, 클릭 시 onCLick 실행	—
btn-role	Component	역할 선택 버튼 컴포넌트	현재 역할 표시, 드롭다운으로 역할 변경	—

btn-setting	Component	설정버튼 컴포넌트	마우스 hover 시 배경색 변경, 클릭시 onClick 실행	—
btn-sign-2-small	Component	작은 화원가입/로그인 버튼	isActive 여부에 따라 스타일 변경, 클릭시 onClick 실행	—
btn-sign-2	Component	회원가입/로그인 버튼(활성 제어)	isActive에 따라 활성/비활성, disabled 적용, onClick 실행	—
btn-sign	Component	회원가입/로그인 버튼(기본 활성)	isActive 여부에 따라 스타일 변경, 클릭 시 onClick 실행	—
btn-sort	Component	정렬 버튼	클릭 시 정렬 순서(newest와 oldest 토글)	—
btn-x	Component	닫기 버튼	클락시 onClick 실행	—
btn-small-arrow	Component	작은 화살표 버튼	Direction에 따라 회전, hover 시 색상 변경	—
frm-folder	Component	폴더/팀 보드 컴포넌트	폴더(또는 팀)의 이름 표시 및 수정,	—

			보드 리스트 표시, 메뉴를 통한 삭제/이름 변경/ 팀 설정 관리	
frm-maketeam	Component	팀 생성 폼 컴포넌트	팀 이름/설명/팀원 입력을 받아 새로운 팀을 생성, 유효성 검사 및 에러 메시지 출력	—
frm-memberline	Component	팀 멤버 라인 아이템 컴포넌트	팀원의 이름, 이메일 표시, 역할 변경 버튼과 삭제 버튼 제공, readOnly 모드 자원	—
frm-memberlist	Component	팀 멤버 리스트 컴포넌트	이메일 입력을 통한 팀원 추가, 초대링크 복사, 멤버 목록 펴시 및 역할 변경/삭제 기능 제공	—
frm-more-team	Component	팀 메뉴 드롭다운 컴포넌트	팀 관련 옵션(팀 설정 변경, 팀 나가기) 목록 표시	—

frm-more-user	Component	사용자 메뉴 드롭다운 컴포넌트	사용자 관련 옵션(폴더 이름 바꾸기, 폴더 삭제 등) 목록 표시	—
frm–teamedit	Component	팀 설정 수정 폼 컴포넌트	팀 상세 조회 후 이름, 설명, 멤버 편집 및 저장/삭제/나가기 기능 제공	Input48, FrmMemberList, BtnX, BtnSign2Small
frm-thumbnail-board	Component	대시보드 썸네일 카드 컴포넌트	보드 상태(collecting, unresponsive, before)관리, 썸네일/메타정보 표시, 보드 열기, 삭제, 설정기능 제공	BoardMenu, Thumbnail, AgentStatusUnresponsive, AgentStatusCollecting, AgentStatusBefore, BtnMore, BtnMoreText
BoardMenu	Component	보드 메뉴 드롭다운 컴포넌트	보드설정, 보드삭제 옵션 제공, 외부 클릭시 닫힘, 선택시 핸들러(onEditSettings/onDelete 실행)	BtnMore, BtnMoreText
icon-blind	Component	아이콘	눈 모양 아이콘	—
icon-sign	Component	아이콘	로그인 모양 아이콘	—

logo.tsx	Component	로고	로고	—
48	Component	입력 필드 UI 컴포넌트	값 입력, Enter 시 완료 상태(isDone) 반영, 좌/우/가운데 정렬 옵션 제공	—
54	Component	입력 필드 + 아이콘 토글 제공	입력 값 표시, 완료 상태(done)에 따라 텍스트/아이콘 색상 변경, 아이콘 클릭시 onDone 상태 제어	IconBlind, IconSign
memberEM	Component	멤버 이메일 입력 UI	이메일 입력 필드 제공, 값 유무에 따라 버튼 활성화 및 멤버 추가 버튼 제어	BtnAddMember
add-folding	Component	스페이스 라벨 + 추가/토글 버튼 UI	라벨 클릭 시 페이지 이동, + 버튼으로 폴더 추가, 화살표 버튼으로 리스트 접기/펼치기	BtnAdd, BtnBigArrow
bar	Component	사이드바 전체	로고, 유저 정보, 로그아웃, 내정보,	AddFolding, MyPage, SpaceNameG, SpaceNameS, Logo

			개인/팀 스페이스 접힘/펼침 및 네비게이션 관리	
my-page	Component	내 정보 사이드 메뉴 항목	클릭 시 /myinfo로 이동, 활성 여부에 따라 색상 변경	—
spacename-g	Component	개인 스페이스 항목(폴더 이름)	클릭시 이동, 입력 편집 모드 지원(이름 변경/취소), 활성 상태에 따라 스타일 변경	—
spacename-s	Component	팀 스페이스 항목(폴더 이름)	클릭 시 이동, 입력 편집 모드 지원(취소가능), 활성 상태에 따라 스타일 변경	—
error-toast	Component	에러 메시지 토스트 UI 표시	Message와 trigger 값이 변하면 애니메이션 재시작, autoHideMs 설정 시 자동 닫힘 지원, 에러 아이콘과 메시지를 함께 표시	—

AppAll	Component	앱 로그 요약 카드(대시보드 위젯)	useLogStore에서 applogs를 가져옴, 총 로그수와 오늘 발생한 로그 수를 계산 후 표시, 실시간 누적 로그 현황을 시각화	useLogStore
AppErrorlog	Component	앱의 에러 로그 카운트 표시	useLogStore에서 appLogs 불러오기, level이 ERROR 인 로그 개수 계산 후 UI에 표시	useLogStore
AppLevel	Component	로그레벨(INFO/WARN/ERROR) 비율 시각화	useLogStore로 앱 로그 불러오기, 로그 레벨별 개수 집계, Recharts를 활용한 반원형 파이 차트 렌더링, 섹터 클릭 시 선택 항목 정보 표시	useLogStore, PieChart, Pie, Cell, ResponsiveContainer
AppLiveLog	Component	앱 실시간 로그 테이블 UI 표시	useLogStore에서 appLogs 불러오기, 검색/필터 적용, timestamp 정렬 토글, 클릭시 onSelectLog 이벤트 호출	useLogStore, BtnDropdown, SearchRefresh

AppLogger	Component	Logger 별 로그 카운트를 시각화	useLogStore 에서 로그 수집, logger 별 카운트 집계 및 정렬, 상위 5개 범례 표시	PieChart, Sector, truncateLabel
AppLogLine	Component	시간대별 이상 로그(WARN/ERROR/FATAL) 변화	1h/6h/12h 단위별 구간 생성 후 카운트 집계	LineChart, XAxis, YAxis, CartesianGrid, Tooltip, CustomTick
AppTimeLog	Component	시간대별 앱 로그 발생량을 시각화	useLogStore에서 로그 불러오기, 1h, 6h, 12h 구간별 로그 발생량 집계, 라인차트로 시각화	LineChart, XAxis, YAxis, CartesianGrid, Tooltip, CustomTick
searchrefresh	Component	실시간 로그 검색 및 새로고침 UI 제공	키워드 입력시 onSearch 콜백 호출, 새로고침 버튼 클릭 시 onRefresh 콜백 호출	Input, button
WebAI	Component	웹 로그를 AI 점수 기반으로 정상/경고/위험/분류 및 시각화	useLogStore에서 webLogs 가져오기, AI 점수에 따라 분류	PieChart, Cell, ResponsiveContainer, useMemi, useState
WebAll	Comp	웹로그	useLogStore로	useMemo

	onent	총 개수 및 오늘 발생한 로그 수 시각화	webLogs불러오기, 총로그 수 계산, 오늘 날짜 기준 필터링 후 todayLogs 계산	
WebErrorlog	Component	HTTP 에러 상태 코드(400, 500) 개수 시각화	webLogs의 status 별 카운트 집계, 400,500 상태코드 로그 수 표시	useLogStore
WebLiveLog	Component	실시간 웹 로그 테이블 뷰어	검색 키워드 필터링, 드롭다운 필터링, timestamp 정렬, AI score 색상 표시	SearchRefresh, BtnDropdown, useMemo, useState
WebLogLine	Component	AI Score 기반 시간대별 변화 라인 차트	시간 구간(1h, 6h, 12h)에 따라 로그 분포 계산	LineChart, Line, CustomTick, useEffect
WebPath	Component	상위 10개 Path 집계 및 비율 원형 차트	url/path 기준 로그 수 집계, 상위 10개 path 정렬 후 색상 매핑,	PieChart, AnimatedSector, useMemo, useState
WebTimeLog	Component	시간대별 로그량 변화 라인	1h/6h/12h 단위별 로그 발생량 집계, 시간 라벨	LineChart, Line, CustomTick, useEffect

		차트	생성 후 구간별 로그 수 표시	
exporter	Component	Exporter 설정	체크박, 증감버튼 두개	Input48,BtnPlus, BtnMinus
filter	Component	로그레벨/키워드 필터(springboot)	토글버튼, 입력필드	Input48
filterweb	Component	로그레벨/키워드 필터(Tomcat)	토글버튼, 입력필드	Input48
multiline	Component	멀티라인 로그 병합 설정	토글버튼, 증감 버튼 및 입력	Input48, BtnPlus, BtnMinus
tailer	Component	로그 파일 읽기 간격 및 메타데이터 저장	증감버튼, 입력 필드	Input48, BtnPlus, BtnMinus
advancedsetting	Component	고급 설정 제공	로그 수집기/필터링 등 고급 설정 UI 제공	TailerSettings, MultilineSettings, ExporterSettings, FilterSettings(Spring Boot), FilterWeb(Tomcat)
appdashboard	Page	애플리케이션.웹 로그 대시보드 화면 관리	인증 가드, 폴더/보드 데이터 로딩, WebSocket 연결, 탭(App/Web) 전환, 상세	Bar, Toggle, WebDashboard, AppAll, AppErrorLog, AppLevel, AppLogger, AppLiveLog, AppLogLine, AppTimeLog, AppLogDetailPanel

			패널 관리	
applogdetail	Component	로그 상세 패널 UI	선택된 로그의 상세 정보를 오른쪽 패널로 표시, 외부 클릭/버튼으로 닫기 처리	BtnX, buildRaw
dashboardedit	Component	대시보드 설정 편집 모달	대시보드 이름, 로그 경로, 로그 유형, 타임존, Agent ID, 고급 설정을 수정 및 저장	Input48, BtnSmallArrow, BtnSign2Small, BtnDropdown, AdvancedSettings, updateDashboard, updateDashboardConfig, getDashboards, getDashboardConfigs
dashboardmake	Component	새로운 대시보드 생성 모달	대시보드 이름, 로그 경로, 로그 유형, 타임존, Agent ID, 고급 설정을 입력받아 생성	Input48, BtnSmallArrow, BtnSign2Small, BtnDropdown, AdvancedSettings, defaultConfigs, deepClone
Thumbnail	Component	대시보드 썸네일 표시	iframe을 이용해 대시보드를 축소/확대/오프셋 이동하여 썸네일로 보여줌	useMemo
toastmessage	Component	보드 등록 완료 토스트 메시지	Agent ID 복사/안내 및 설치 가이드 표시	Input48

toggle	Component	탭 전환 버튼	App ↔ Web 대시보드 전환	—
webdashboard	Component	웹 대시보드 전체 레이아웃	WebAll, WebErrorLog, WebAI, WebPath, WebLiveLog, WebLogLine, WebTimeLog, WebLogDetailPanel 조합	WebAll, WebErrorLog, WebAI, WebPath, WebLiveLog, WebLogLine, WebTimeLog, WebLogDetailPanel
weblogdetail	Component	웹 로그 상세 패널	클릭한 로그의 상세 정보, Raw log line, AI Score 표시	BtnX, motion, AnimatePresence
folderpage	Page	개인 폴더 상세 페이지	폴더 내 대시보드(보드) 조회, 생성, 수정, 삭제 및 썸네일 보드 관리	Bar, FrmThumbnailBoard, DashboardMake, DashboardEdit, ToastMessage, BtnBigArrow
Myinfo	Page	내 정보 관리 페이지	사용자 정보 확인 및 비밀번호 인증 → 정보 수정, Webhook 등록/테스트/트리거 기능 제공	Bar, BtnSign2, Input54, ErrorToast, Toast(Webhook 관리), api(axiosInstance)
MyinfoEdit	Page	내 정보 수정 페이지	이메일/비밀번호 변경, 중복확인, 서버에 PUT 요청하여 계정 정보 갱신 후	Bar, BtnSign2, Input54, ErrorToast, api(axiosInstance), useAuth, isValidEmail, isValidPassword

			상태 업데이트	
P-spacePage	Page	개인 스페이스 관리 페이지	개인 폴더 목록 표시, 추가/수정/삭제, 폴더별 대시보드 관리, DashboardMake 모달 호출	Bar, BtnSort, BtnPoint, FrmFolder, DashboardMake, api(folders/dashboard), useFolderStore, AnimatePresence
S-SpacePage	Page	팀 스페이스 관리	팀 목록 조회/정렬, 팀 추가/수정/삭제, 대시보드 생성	Bar, BtnSort, BtnPoint, FrmFolder, FrmMakeTeam, FrmTeamEdit, DashboardMake, useFolderStore, api/teams
teampage	Page	특정 팀 상세 관리	보드 조회/정렬, 보드 추가/수정/삭제, 팀 정보 표시, 팀 생성, Toast 알림	Bar, BtnBigArrow, FrmThumbnailBoard, DashboardMake, DashboardEdit, FrmMakeTeam, ToastMessage, useFolderStore, api
btn-download	Component	다운로드/액션 버튼	hover glow, primary/secondary 스타일, onClick 실행	—
feature-card	Component	기능 카드	이미지 + 타이틀 카드 표시	—
feature-section	Component	기능 섹션 래퍼	FeatureCard 여러 개 렌더링	FeatureCard

landingpage	Page	랜딩 페이지 전체	Hero/Preview/Feature/Footer 렌더링, 애니메이션 처리, 버튼 액션	BtnDownload, PreviewBox, FeatureSection, Logo, motion, useInView, useNavigate
preview-box	Component	서비스 미리보기 박스	고정된 프리뷰 이미지(Web2.png) 표시	—
LoginPage	Page	기존 계정 로그인 페이지	이메일/비밀번호 입력 후 로그인 요청, 에러 토스트 표시, 성공 시 PersonalSpace로 이동	Input54, BtnSign, ErrorToast, Logo, useAuth(login), useNavigate, motion
SignupPage	Page	신규 계정 회원가입 페이지	사용자명/이메일/비밀번호 입력 및 유효성 검사, 이메일 중복 확인 API 호출, 회원가입 처리	Input54, BtnSign, ErrorToast, Logo, useAuth(signup), api(check-email), motion
AuthContext	context	인증 상태 전역 관리	User/토큰 관리, login·logout·signup 제공	AuthProvider, useAuth, ProtectedRoute
AuthProvider	Provider	전역 사용자 상태 제공	user 상태/토큰/에러 관리, login, logout, signup 제공	AuthContext

ProtectedRoute	Component	보호라우트	로그인 여부 확인 후 children 노출 or /login 리다이렉트	—
folderStore	Store	개인/팀 폴더 상태 관리	개인/팀 폴더·정렬 순서 저장, setter·정렬함수 제공	folders, teamFolders
logstore	Store	실시간 로그 상태 관리	appLogs/webLogs 관리, WebSocket 연결·해제, 로그 추가	Connect, disconnect
type	Type	API & UI 타입 정의	User, Folder, Team, Member, Role 정의	—
validate	Utils	유효성 검사	이메일/비밀번호/유저이름/비밀번호 확인 검증	—
Appinitializer	Component	로그인 후 초기 데이터 로딩	로그인 사용자 기준 개인/팀 폴더 API 호출 후 store 초기화	useFolderStore, getTeams, getFolders
AnimatedRouted	Compinent	라우팅 관리	/login, /signup, /personal, /team, /myinfo 등	각 Page 컴포넌트 들

			라우트 분기	
App	Component	앱 루트	AuthProvider + Router + 초기화/라우팅	AppInitalizer, AnimatedRoutes

3.2.2.5 AI Server

3.2.2.5.1 Component Diagram

3.2.2.5.2 Component Description

Identification	Type	Purpose	Function	Subordinates
main	Entry point	서버 실행 진입점	FastAPI 실행 및 라우터 등록록	predict
predict	Module	로그 점수화 API 엔드포인트	/receive_logs HTTP POST 요청을 통해 로그 수신후 Isolation Forest 모델로 점수 계산 후 반환	Extract_features, compute_score_for_log
extract_features	Function	로그 특성 추출	URL, User-Agent, Referer 기반	IOC_PATTERNS

			피처 생성(길이, 깊이, 특수문자, IOC 키워드 포함)	
Compute_socre_for_log	Function	로그 스코어 산출	전처리된 로그를 One-hot 인코딩하고 Isolation Forest 추론을 통해 나온 수치를 점수 스케일링	Load_model, scale_score
model	module	모델 로딩 유틸	Joblib을 사용해 모델(isolation_forest.pkl)과 feature 정보 로드	—
scaler	Module	점수 스케일링 유틸	scaler_params.json 기반 최소/최대 값 적용하여 0~100 정규화	—
ioc	Module	IOC 패턴 로딩	Ioc_keywords.pkl 파일에서 URL/User-Agent IOC 패턴 로드	—
parser	Module	로그 파싱 유틸리티	Access 로그 문자열을 정규식으로 파싱하여 dict로 변환	parse_log_line

parse_log_line	Function	로그 문자열 파싱	Tomcat 형식 로그 한줄을 파싱하여 dict(method, url, status, size, referer, user_agent)생성	—
isolation_model	Model File	Isolation Forest 모델	로그 이상탐지 수행(decision_function 기반)	—
features	Data File	Feature 리스트	로그 입력 벡터 생성 시 피처 순서 제공	—
method_cols	Data File	HTTP 메서드 One-Hot 인코딩 컬럼 정보	메서드 기반 피처 매핑	—
scaler_params	Config File	점수 스케일링 파라미터	min/max 점수 저장	—
ioc_keywords	Data File	IOC 키워드 사전	URL/UA IOC 패턴 매칭용	—
IF_model	Jupyter Notebook	모델학습/실험 기록	Isolation Forest 모델 학습, 피처 선정, 스케일링 파라미터 산출	—

3.3 Dependency View (의존 뷰)

3.3.1 System-level Dependency View (전체 모듈 간 의존성)

LogMate 시스템은 크게 Agent, API Server, Streaming Server, AI Server, Frontend로 구성되며, 각 모듈은 다음과 같은 의존성을 가진다.

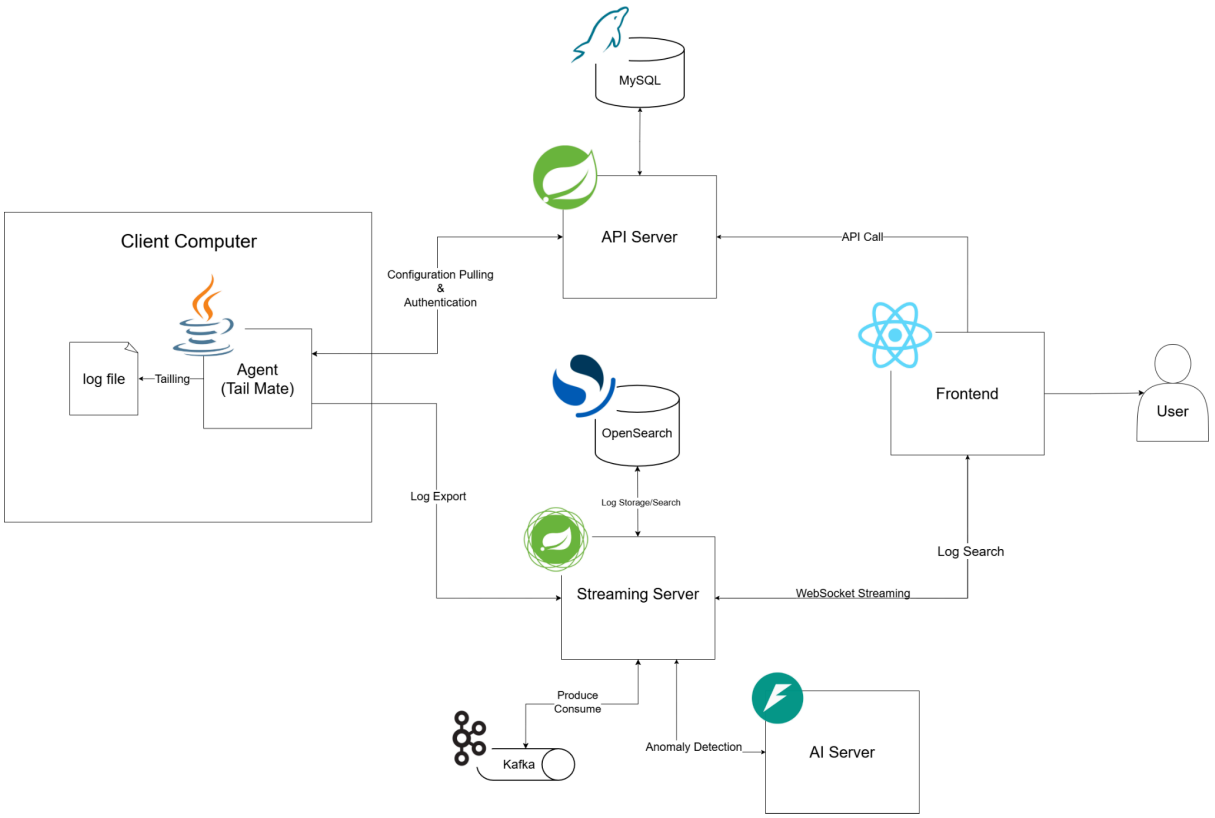
- **Agent**는 클라이언트 로그 파일을 수집하여 Streaming Server로 전송(Log Export)하며, 동시에 API Server와 통신하여 설정을 주기적으로 Pulling 하고 인증을 수행한다.

- **API Server**는 MySQL을 기반으로 사용자 및 설정 데이터를 관리하며, Frontend와 REST API를 통해 연결된다. JWT를 활용하여 인증 인가를 관리한다. 또한 Agent의 설정·인증을 담당한다.

- **Streaming Server**는 Agent가 전송한 로그를 수신하고, 이를 Kafka와 연동하여 처리하며, OpenSearch에 저장한다. 동시에 AI Server로 로그를 전달해 이상 탐지를 수행하고, 결과를 반영한다. 또한 Frontend로 WebSocket을 통해 실시간 로그를 스트리밍하며, 로그 조회 요청에 응답한다.

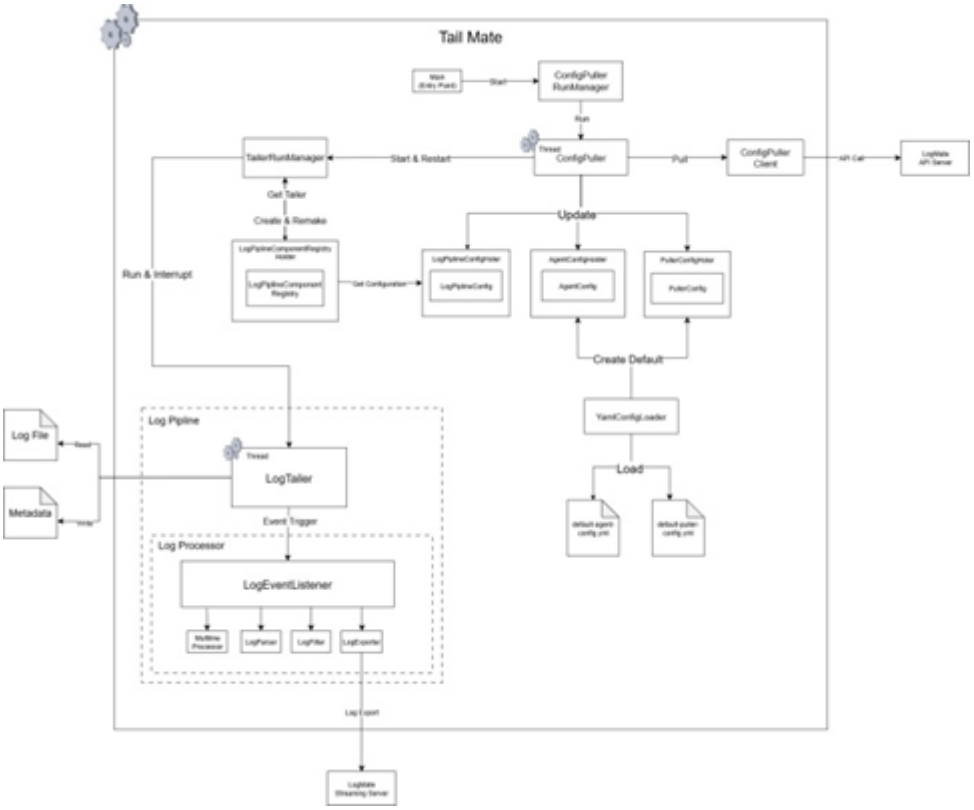
- **AI Server**는 Streaming Server로부터 로그를 전달받아 분석하고, 이상 탐지 결과를 다시 Streaming Server에 반환한다.

- **Frontend**는 사용자에게 API Server에서 받은 관리 데이터와 Streaming Server에서 전달받은 실시간 로그, 범위 조회 로그를 통합하여 시각화하고 제공한다.



3.3.2 Subsystem-level Dependency View (서버 내부 클래스 의존성)

3.3.2.1 Agent Dependency View



3.3.2.2 Streaming Server Dependency View

1) LogIngestController

- Agent(TailMate)로부터 HTTP Push 방식으로 로그를 수신.
- 수신된 로그를 LogProducer 인터페이스로 위임.

2) LogProducer → Kafka → LogConsumer

- GenericKafkaLogProducer가 로그를 Kafka에 저장.
- UnifiedLogConsumer가 Kafka에서 로그를 다시 구독(consume).

3) LogPipeline

- 소비된 로그를 LogPipeline으로 전달해 단계별 처리.
- DefaultLogPipeline이 체이닝을 관리하고, LogProcessor들을 순차 실행.

4) LogProcessor 체인

- (order 1) AILogProcessor: 로그를 AI Server에 전달해 이상 탐지를 요청.
- (order 2) LogStorageProcessor: 로그를 OpenSearch에 저장.
- (order 2) WebSocketProcessor: 로그를 WebSocket을 통해 브라우저(Frontend)로 스트리밍.

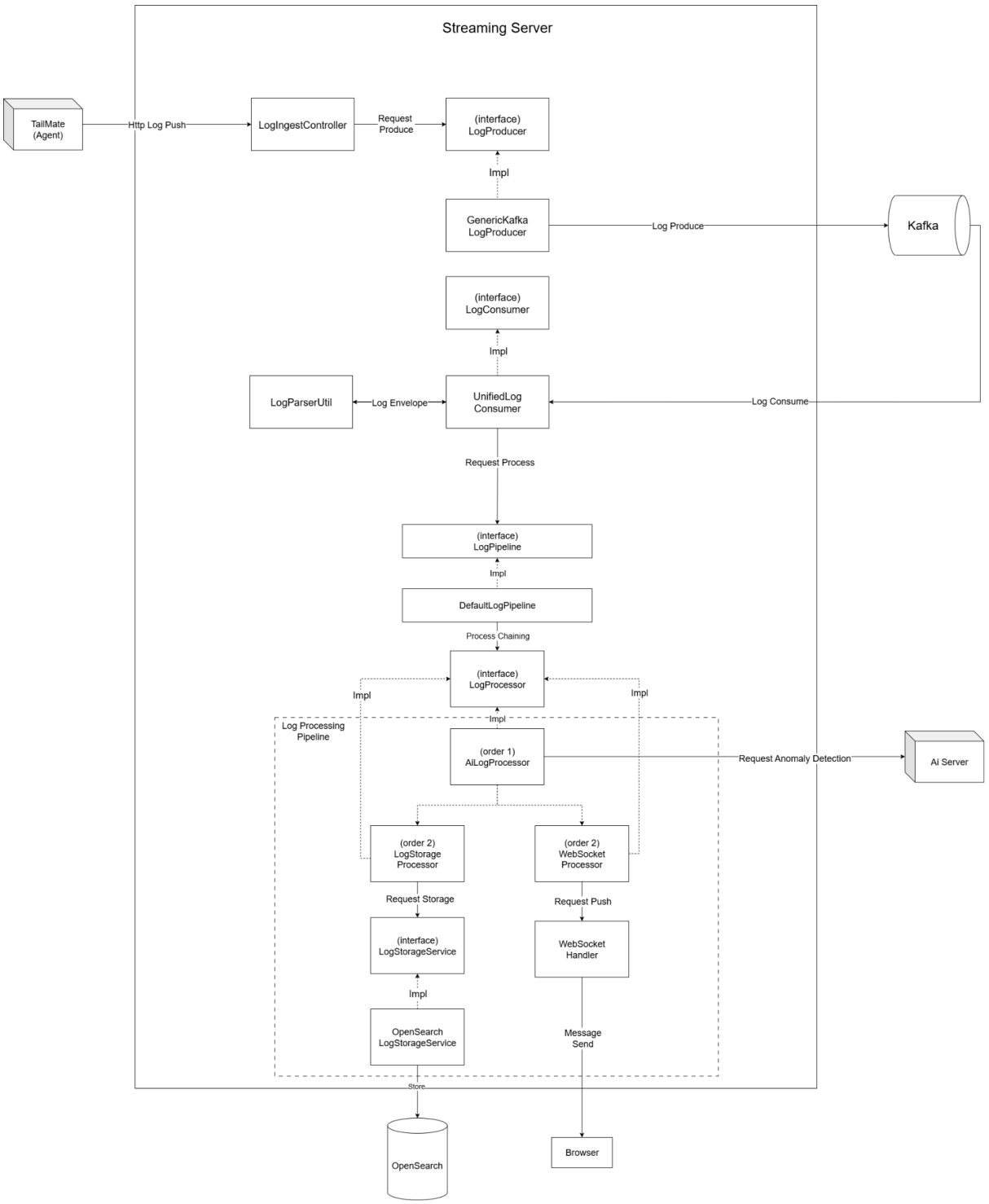
- Order 가 같은 값일 경우 병렬처리 진행.

5) OpenSearch

- OpenSearchLogStorageService를 통해 영구 저장 및 검색 가능.

6) WebSocket Handler

- 처리된 로그를 실시간으로 브라우저 클라이언트에 Push.



3.3.2.3 API Server Dependency View

1) UserController / AuthController

사용자 인증 및 계정 관련 HTTP 요청을 수신

수신된 요청은 UserService, AuthService로 위임되어 로그인, 회원가입, 로그아웃, 사용자 정보

수정 등을 처리

2) TeamController → TeamService → TeamRepository / TeamMemberRepository

로그인된 사용자의 팀 정보 및 멤버 관계를 관리

팀 생성, 수정, 삭제, 멤버 초대, 권한 변경 요청을 처리하며 권한 변경, 수정, 삭제의 경우 사용자의 권한에 따라 제한적임

TeamMember 엔티티를 통해 User와 Team을 연결하는 다대다 구조를 관리

3) DashboardController → DashboardService → DashboardRepository

대시보드 생성, 조회, 수정, 삭제 기능을 담당하며, 각 대시보드는 Team 또는 개인 폴더에 속하며, 대시보드 고급 설정 저장 시 AgentConfiguration과 연결되어 있음

설정 변경 시 AgentConfigService로 설정 전송을 요청

-> 대시보드 기반 로그 파이프라인 설정 관리의 중심 역할을 함

4) AgentConfigController→ AgentConfigService→ AgentConfigurationRepository / LogPipelineConfigRepository

에이전트(TailMate)와 직접 연결되는 도메인

Agent의 설정 저장, etag 버전 관리, Pull 요청 응답 처리 담당

DashboardService로부터 수신된 구성 정보를 기반으로 AgentConfiguration
및 LogPipelineConfig 엔티티를 관리

-> 설정은 JSON 형태로 직렬화되어 Agent에게 반환됨

5) WebhookController → WebhookService → WebhookRepository

외부 알림 연동 기능 담당

사용자가 등록한 Slack, Discord 등의 Webhook URL을 저장하고, 이벤트 발생 시 해당 URL들로 POST 요청 전송

전송 실패 시 재시도 로직 또는 실패 로그 기록 수행

→ User도메인과 연계되어 알림 전송

6) FolderController → FolderService → FolderRepository

대시보드를 폴더 단위로 그룹화 및 관리

팀 단위 폴더 구조를 구성하고, 각 폴더 내 대시보드를 트리 형태로 유지

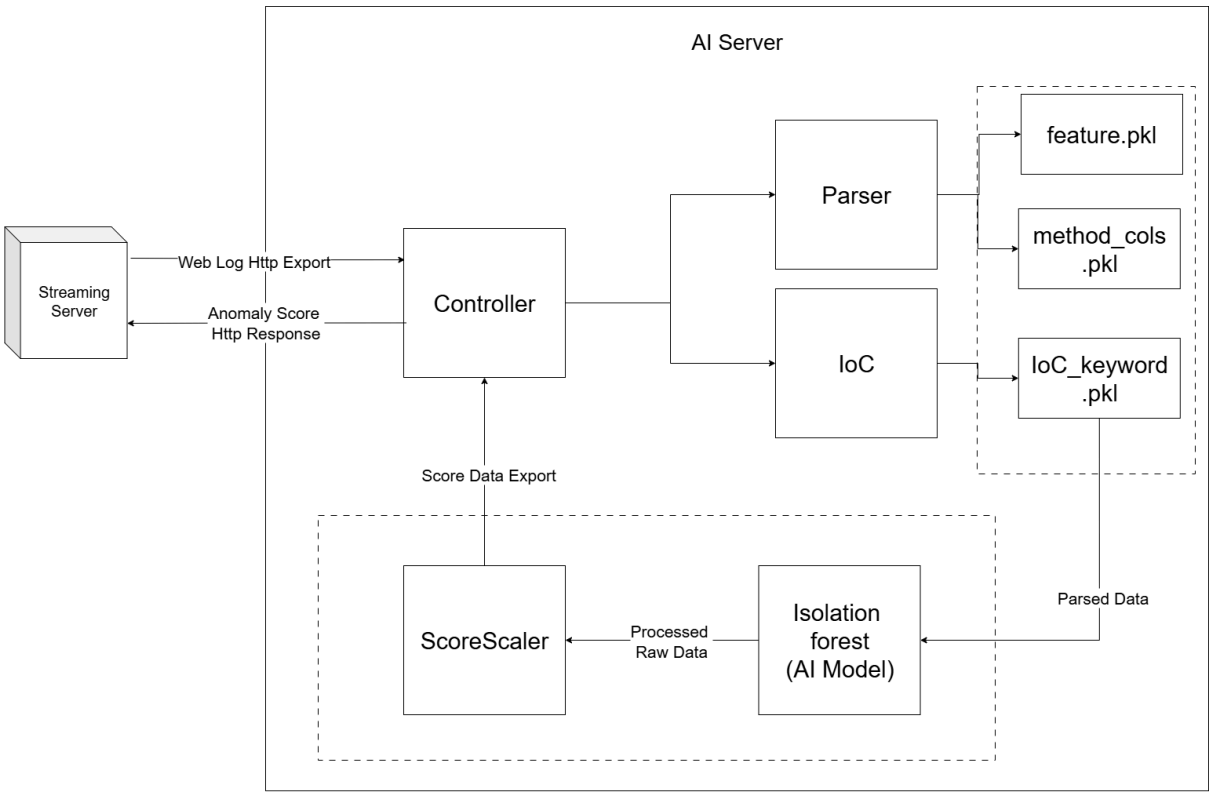
→ Team, Dashboard 도메인과 연계되어 계층적 구조 제공

7) Global Layer (공통 인프라 계층)

모든 도메인에서 공통 사용되는 기반 컴포넌트.

- BaseEntity: status, createdAt, updatedAt 공통 필드
- BaseResponse, BaseErrorResponse: 일관된 응답 구조 제공
- CustomException: 예외 처리 공통 규격
- SecurityConfig: JWT 인증/인가 및 필터 체인 설정
- WebConfig: CORS, 인터셉터, ArgumentResolver 등 글로벌 설정
  → 전 도메인의 안정성 및 일관성을 보장하는 기반 계층 역할

3.3.2.5 AI Server Dependency View



4. Data Design (데이터 설계)

4.1 Log Data Model and Structure

4.1.1 Overview

LogMate의 로그 데이터 모델은 두 가지 주요 유형으로 구성된다: Spring Boot Application Log와 Tomcat Access Log. 모든 로그 데이터 클래스는 ParsedLogData 인터페이스를 구현하며, 공통적으로 timestamp, isFormatCorrect, message, userTimezone 필드를 포함한다. 이 구조는 로그 소스별 형식 차이를 통합하여, Streaming 서버 및 OpenSearch 인덱싱 단계에서 일관된 파이프라인 처리를 가능하게 한다.

4.1.2 Time Role

LogMate 시스템은 시간 정보(Time Data)를 모든 구성요소 간에 일관되게 처리하기 위해 UTC(협정 세계시) 기준으로 변환 및 저장한다.

이는 로그 수집 시점의 지역 환경이나 서버 타임존에 상관없이 동일한 기준 시간 축을 유지하기 위함이다.

1. **UTC 변환 원칙**
    - 모든 로그의 timestamp 필드는 시스템 내부에서 UTC 기준으로 변환된 Instant 형태로 저장된다.
    - Agent가 수집한 원본 로그의 시간이 타임존 정보를 포함하고 있을 경우(2025-10-16T14:00:00+09:00 등), 해당 오프셋 정보를 기준으로 UTC 시간으로 변환한다.
    - Streaming Server 및 OpenSearch 인덱싱 단계에서도 동일하게 UTC 시각 기준으로 처리되어, 시계열 분석 및 기간 검색 시 지역 간 시간 차이에 의한 불일치를 방지한다.

2. **User Timezone 기반 유추 처리**
    - 일부 로그 소스(Tomcat Access Log 등)는 타임존 정보가 명시되지 않은 형태로 기록된다.
    - 이 경우, Agent 설정에 포함된 userTimezone 값을 참조하여 해당 로그의 지역 시각을 추정하고, 이를 UTC로 변환한 후 시스템에 저장한다.
    - 예를 들어, 로그가 2025-10-16 09:00:00으로 기록되어 있고 userTimezone이 Asia/Seoul인 경우, 내부적으로 2025-10-16T00:00:00Z로 변환되어 저장된다.

3. **표준 시간 저장 정책**
    - 내부 데이터 저장 및 인덱싱(OpenSearch, S3, RDS 등)에서는 모두 UTC 기반 시간값을 사용한다.
    - 외부 시스템(API 응답, 대시보드 UI 등)에 출력 시에는 사용자의 로컬 타임존 설정(userTimezone)에 맞추어 변환된 형태로 표시된다.

4. **UI 에서의 시간 변환**
    - 사용자 UI(Frontend)에서 표시되는 시간은, 사용자가 설정한 userTimezone에 맞춰 변환된 형태로 표시된다.
    - 즉, 백엔드에서 전달된 UTC 시각은 클라이언트 측에서 userTimezone 기준으로 변환되어 렌더링된다.
    - 이를 통해 서로 다른 지역의 사용자도 자신의 로컬 시간대에 맞춰 로그 발생 시점을 직관적으로 확인할 수 있다.

4.1.3 Common Interface: ParsedLogData

ParsedLogData는 LogMate의 모든 로그 데이터 클래스 (SpringBoot, Tomcat, 또는 향후 추가될 Nginx, System 등)가 반드시 구현해야 하는 공통 인터페이스이다.

이 인터페이스는 로그의 핵심 속성을 통일된 형태로 노출하여, 파이프라인의 모든 단계가 로그 소스에 관계없이 동일한 데이터 접근 방식을 사용할 수 있도록 보장한다.

메서드	반환 타입	설명	설계 의도
getMessage()	String	로그의 본문 메시지 또는 요청 URL	모든 로그 분석·검색의 기본 단위로 활용됨
getUserTimezone()	String	로그 발생 환경의 타임존	다중 지역 로그 분석 시 시간대 변환을 위해 필요
getTimestamp()	Instant (UTC Time)	로그 발생 시각	정렬, 시계열 분석, AI 이상 탐지의 핵심 기준
isFormatCorrect()	Boolean	포맷 유효성 여부	파싱 실패 라인을 식별·제외하여 데이터 품질 보장

4.1.4 SpringBootParsedLogData

필드명	타입	설명
isFormatCorrect	Boolean	로그 포맷이 파싱 규칙에 부합하는지 여부
timestamp	Instant	로그 발생 시각 (UTC Time, 초 단위 정밀도)
level	String	로그 레벨 (INFO, WARN, ERROR 등)
thread	String	로그 발생 스레드명

logger	String	로그를 남긴 클래스 또는 패키지명
message	String	로그 메시지 본문
userTimezone	String	로그 발생 환경의 타임존

4.1.5 TomcatAccessLogParsedLogData

필드명	타입	설명
isFormatCorrect	boolean	로그 포맷 파싱 성공 여부
ip	String	요청을 발생시킨 클라이언트 IP
timestamp	Instant	로그 발생 시각 (UTC Time, 초 단위 정밀도)
method	String	HTTP 메서드 (GET, POST 등)
url	String	요청 URL
protocol	String	사용된 프로토콜 (예: HTTP/1.1)
statusCode	int	HTTP 응답 코드
responseSize	int	응답 바이트 크기
referer	String	요청의 참조 URL
userAgent	String	요청을 발생시킨 클라이언트의 User-Agent

extra	String	부가 정보
userTimezone	String	로그 발생 환경의 타임존

4.2 Agent Config Schema (YAML)

4.2.1 Overview

LogMate Agent는 실행 시 API 서버로부터 Json 형식의 설정을 로드하여 동작한다. 이 설정은 세 개의 주요 블록(agentConfig, pullerConfig, logPipelineConfigs)으로 구성되며, 각각 Agent 자체 식별 정보, 설정 갱신 메커니즘, 로그 파이프라인 정의를 담당한다. 모든 구성은 eTag 기반 버전 관리와 무중단 재적용(hot reload)을 지원한다.

구성은 세 부분으로 나뉜다:

1. **agentConfig**: 에이전트 자체 식별 정보 및 인증 토큰
2. **pullerConfig**: 설정 갱신(Hot Reload)을 위한 Puller 정보
3. **logPipelineConfigs**: 로그 수집·파싱·전송 파이프라인 정의

```
etag: config-dto-etag-12345

agentConfig: { ... }

pullerConfig: { ... }

logPipelineConfigs: [{ … }, { … } ... ]
```

4.2.2 agentConfig

필드	타입	설명

agentId	String	에이전트 고유 식별자. API 서버에서 발급됨
accessToken	String	Agent가 API 서버 인증에 사용할 토큰
etag	String	Agent 설정 버전 관리용 eTag

4.2.3 pullerConfig

필드	타입	설명
pullURL	String	최신 설정을 가져올 API 엔드포인트
intervalSec	int	설정 갱신 주기 (초 단위)
etag	String	Puller 설정 버전 관리용 eTag

4.2.4 logPipelineConfig

각 로그 파이프라인은 독립적인 스레드(thNum)에서 실행되며, Tailer–Merger-Parser–Filter–Exporter 모듈이 순차적으로 연결된다.

필드	타입	설명
thNum	int	파이프라인 고유 스레드 번호
tailer.filePath	String	읽을 로그 파일 경로
tailer.metaDataFilePathPrefix	String	마지막 읽은 위치를 저장할 메타데이터 파일 prefix

multiline.enabled	boolean	멀티라인 병합 활성화 여부
multiline.maxLines	int	병합 가능한 최대 라인 수
exporter.pushURL	String	로그 전송 대상 서버 URL
exporter.retryIntervalSec	int	전송 실패 시 재시도 간격 (초 단위)
exporter.maxRetryCount	int	최대 재시도 횟수
parser.type	String	로그 파서 유형 (springboot / tomcat 등)
parser.config.timezone	String	로그 해석 시 사용할 타임존
filter.allowedLevels	String[]	허용 로그 레벨
filter.allowedKeywords	String[]	메시지 키워드 필터
filter.allowedMethods	String[]	HTTP 메서드 필터

4.3 OpenSearch Data Management Strategy

4.3.1 Overview

OpenSearch는 LogMate의 주요 로그 저장소이자 검색·시각화 엔진으로 사용된다. 이 구성요소는 실시간 로그 수집, 인덱싱, 검색 및 통계 집계(Aggregation) 기능을 담당하며, 대시보드에서 표시되는 모든 로그 데이터와 통계 정보의 기반이 된다.

OpenSearch는 시간(event_time) 기반 인덱싱과 agentId 기반 라우팅(routing) 전략을 적용하여 성능과 확장성을 동시에 확보한다. 또한 ILM(Index Lifecycle Management) 정책을 통해 30일간의 로그를 검색 가능 상태로 유지된다.

추후 데이터는 S3 Snapshot Repository로 백업하는 기능을 구현할 예정이다.

4.3.2 Index Convention

규칙	설명	예시
접두어	식별 접두어	parsed
접미어	식별 접미어	logs
로그 타입	수집된 로그 종류 (springboot, tomcat, nginx 등)	springboot
날짜	인덱스 생성일 (event_time 기준)	2025.10.07
전체 이름	{prefix}-{logType}-{suffix}-{date}	parsed-springboot-logs-2025.10.07
라우팅 키	agentId	-
인덱스 생성 주기	1일 단위 롤오버	-

4.3.3 Retention & Backup Policy

구분	정책
보존 기간 (조회 가능)	30일

삭제 정책	ILM 기반 자동 삭제
백업 정책	삭제 전 S3 Snapshot Repository로 자동 백업
백업 구조 예시	s3://logmate-archive/logs/YYYY/MM/DD/
조회 범위 외 로그 처리	프론트엔드 UI에서 event_time 기준 필터링으로 제외 표시

4.4 Kafka Topics and Message Keys Strategy

4.4.1 Overview

Kafka는 LogMate의 실시간 로그 스트리밍 파이프라인의 중심(Message Backbone) 으로 사용된다. Agent에서 수집된 로그는 Streaming Server를 통해 Kafka로 전달되며, Kafka는 이를 실시간 버퍼링 및 비동기 처리 구조로 관리한다.

Kafka는 LogMate 내에서 다음과 같은 역할을 수행한다:

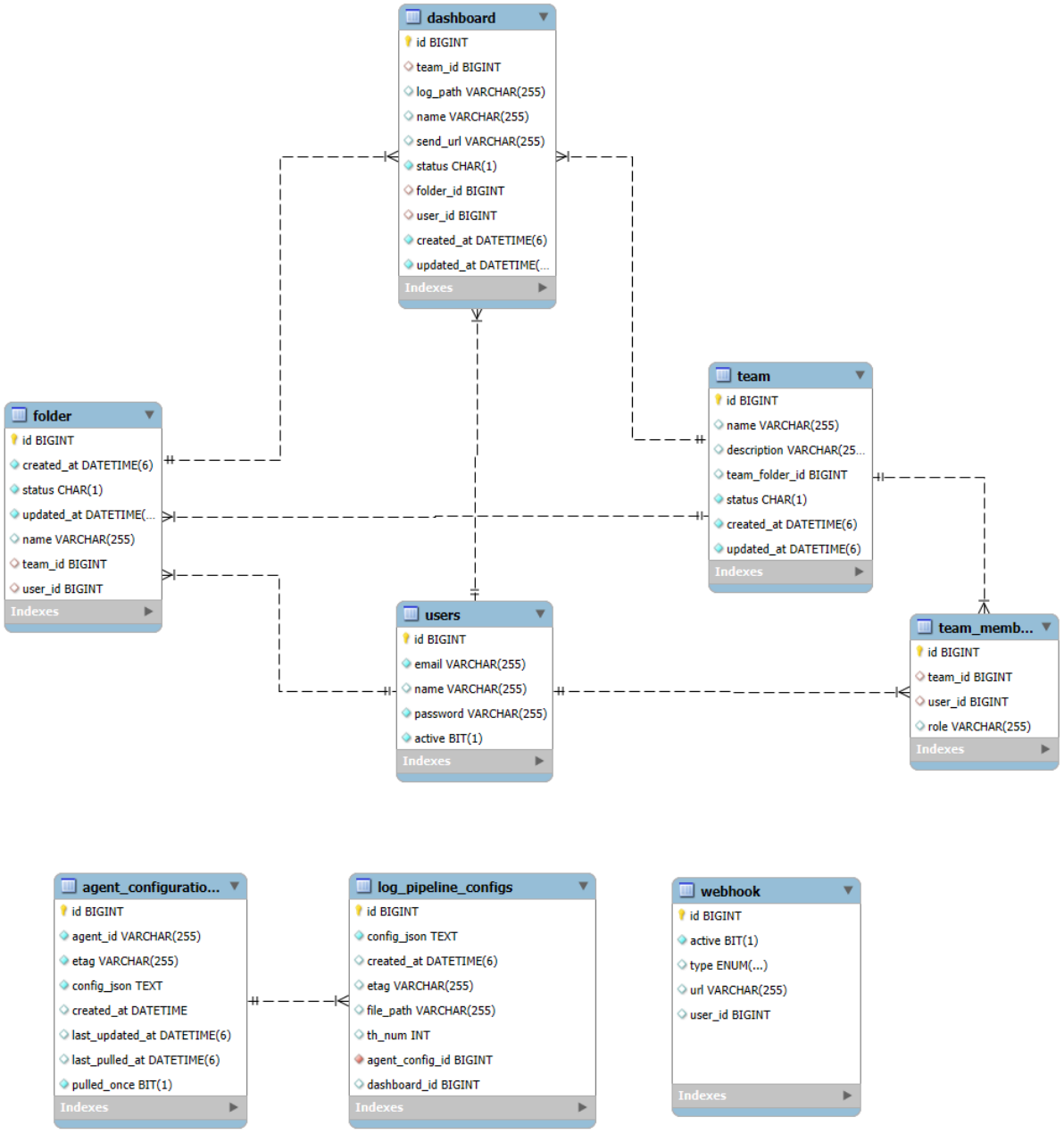
- 실시간 로그 수집과 병렬 분산 처리
- Consumer 장애나 네트워크 단절 시 재처리(Replay) 지원
- DLQ(Dead Letter Queue)를 통한 오류 로그 격리 및 복구

4.4.2 Topic Configuration

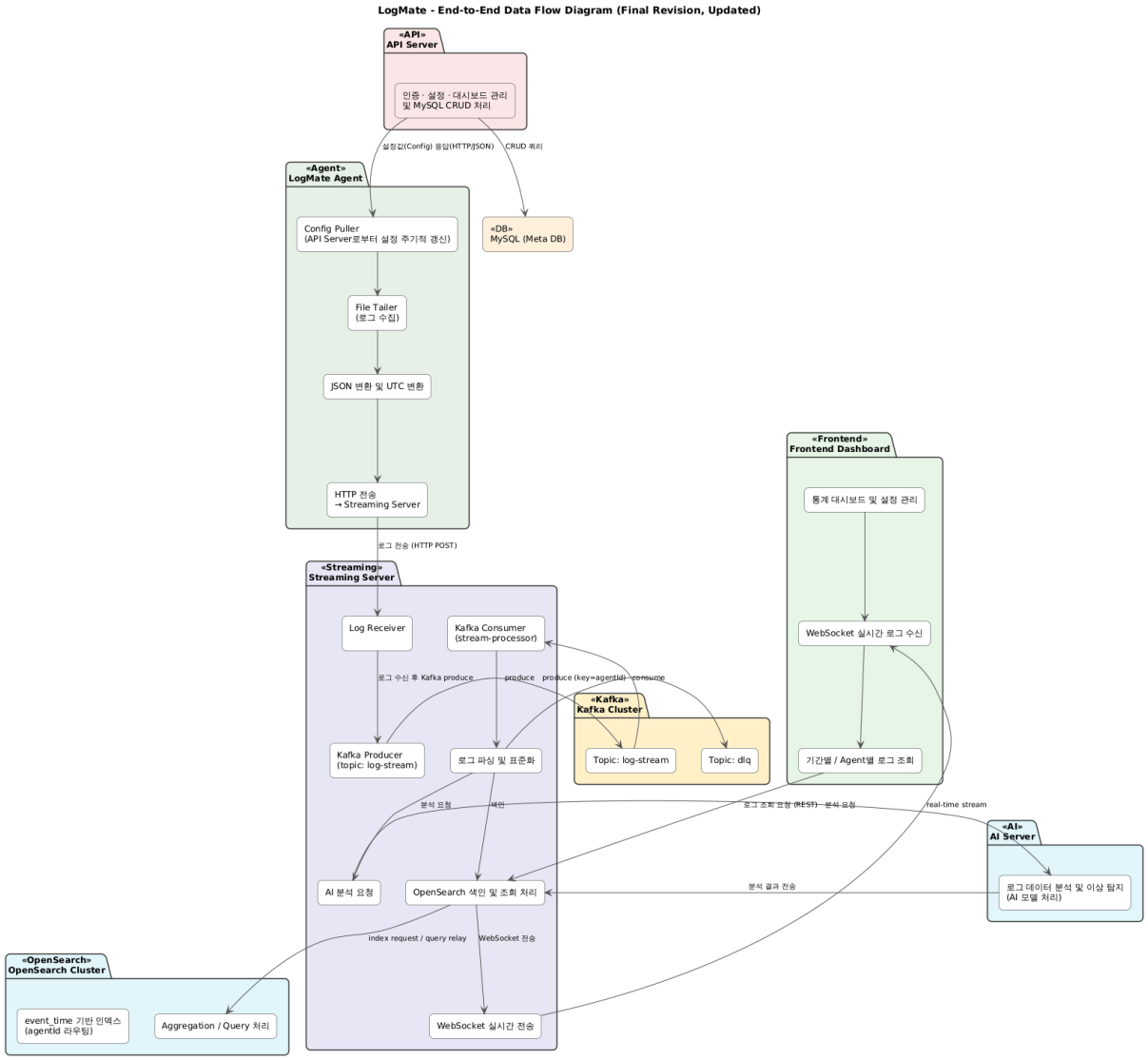
현재 LogMate의 Kafka는 단일 메인 토픽과 DLQ 토픽을 사용한다.

구분	토픽 이름	용도
Main Topic	log-stream	Agent → Streaming Server로부터 수집된 로그를 저장
DLQ Topic	dlq	처리 실패, 역직렬화 오류 등 비정상 로그를 보관

4.5 Relational Data Model (MySQL ERD)



4.6 End-to-End Data Flow Diagram



5. Interface Design (인터페이스 설계)

5.1 REST API

5.1.1 API Server

API Server는 사용자 인증, 팀 및 대시보드 관리, 파싱/필터링 규칙 설정, Webhook 관리 등

LogMate의 핵심 비즈니스 로직을 담당하며, Frontend 및 외부 서비스와의 통신을 위해 REST API를 제공한다.

세부 API 명세는 아래 Notion 문서에 별도로 관리된다.

https://www.notion.so/Logmate-API-255d02fced5a802582f3c7e557485ce2

5.1.2 Streaming Server

Streaming Server는 외부 Agent로부터 수집된 로그를 HTTP 기반으로 수신하고, Kafka 및 OpenSearch 등 내부 컴포넌트로 전달한다.

또한 OpenSearch 에 저장된 로그를 조회하는 API를 제공한다.

세부 API 정의는 아래 Notion 문서에 별도로 관리된다.

https://lime-shift-a9d.notion.site/API-26fd93ae9e4c8022a849eb3a8208a0d5?source=copy_link

https://lime-shift-a9d.notion.site/API-26fd93ae9e4c808ba055c40ddf524dc2?pvs=74

https://lime-shift-a9d.notion.site/28ad93ae9e4c80baba81ffbe778e5f10?source=copy_link

5.1.3 AI Server

AI Server는 Streaming Server로부터 전달받은 로그 데이터를 기반으로 이상 탐지(Anomaly Detection)를 수행한다.

세부 API 정의는 아래 Notion 문서에 별도로 관리된다.

https://lime-shift-a9d.notion.site/AI-254d93ae9e4c80c08ef4c0692e1ad35b?source=copy_link

5.2 WebSocket

Streaming Server는 Frontend 클라이언트와 WebSocket을 통해 양방향 실시간 통신을 수행하며, Kafka로부터 소비된 로그 데이터를 클라이언트 대시보드로 실시간 스트리밍한다.

세부 메시지 포맷 및 동작 방식은 아래 Notion 문서에 별도로 관리된다.

https://lime-shift-a9d.notion.site/26fd93ae9e4c802b866bce5ec38bf172?pvs=73

6. IEEE 1016 Design Views

6.1 Decomposition View

3장 3.3 참조

6.2 Dependency View

3장 3.4 참조

6.3 Interface View

5장 참조

6.4 Detail View

7장 참조

7. Detailed Design

7.1 Agent

7.1.1 Class: AgentInitializer

No	Attribute / Component	Type	Description	Notes
1	argsExtractor	ArgsExtractor	커맨드라인 인자 파서	Guice 주입
2	configInitializer	ConfigInitializer	Agent 설정 초기화기	Guice 주입
3	authenticator	AgentAuthenticator	API 서버 인증 수행기	Guice 주입
4	configPullerRunManager	ConfigPullerRunManager	설정 Puller 스레드	Guice

			실행 관리	주입
5	agentArguments	AgentArguments	파싱된 인자 DTO	지역 변수
6	authToken	AuthToken	인증용 토큰 객체	지역 변수
7	AgentConfigHolder	Singleton	전역 설정 객체 저장소	Static 싱글턴
8	log	Logger	초기화 로그 출력	Slf4j
9	YamlConfigLoader	Loader Impl	설정 파일 로드	내부 의존
10	init(args)	void	Agent 전체 초기화 프로시저	메인 엔트리

처리 로직

1. CLI 인자를 파싱하여 AgentArguments 객체를 생성한다.
2. ConfigInitializer를 통해 Agent 설정과 Puller 설정을 로드 및 검증한다.
3. AgentAuthenticator를 이용하여 API 서버 인증을 수행한다.
4. ConfigPuller 스레드를 실행하여 동적 설정 주입을 시작한다.
5. 초기화 완료 로그를 출력한다.

7.1.2 Class: ConfigPuller

No	Attribute	Type	Description	Notes

1	client	ConfigPullClient	설정 서버와 통신하는 HTTP 클라이언트	의존 객체
2	updater	ConfigUpdater	새 설정 적용기	의존 객체
3	etag	String	현재 설정 버전 식별자	초기 UUID
4	AgentConfigHolder	Holder	Agent 전역 설정 참조	Static
5	PullerConfigHolder	Holder	Puller 설정 참조	Static
6	run()	void	설정 주기적 Pull 루프	Runnable 구현
7	client.configPull()	Optional	HTTP 요청 수행 메서드	ConfigDTO 반환
8	updater.apply()	void	새 설정 반영	Hot reload 수행
9	Thread.sleep()	long	Pull 간격 제어	intervalSec × 1000
10	log	Logger	동작 상태 출력	Slf4j

처리로직

1. 무한 루프를 시작한다.
2. AgentConfig 및 PullerConfig를 로드한다.
3. 현재 eTag를 포함한 HTTPS 요청을 보낸다.
4. 응답이 존재하면 ConfigUpdater를 통해 새로운 설정을 반영하고 eTag를 갱신한다.
5. 변경이 없으면 그대로 다음 주기까지 대기한다.

6. PullerConfig의 intervalSec만큼 대기한 후 다시 반복한다.
7. InterruptedException 발생 시 스레드를 종료한다.

7.1.3 Class: FileLogTailer

No	Attribute	Type	Description	Notes
1	config	TailerConfig	로그 파일 경로, 읽기 주기 설정	불변 필드
2	listener	LogEventListener	로그 수신 이벤트 처리기	의존 객체
3	logFile	File	대상 로그 파일 객체	tail 대상
4	metaDataFile	File	마지막 읽은 위치 저장 파일	오프셋 관리
5	lastKnownPosition	long	최근 읽은 위치 (byte 단위)	변경 지속 저장
6	thNum	Integer	파이프라인 번호	MDC thread context
7	savePositionToFile()	void	포지션 정보 파일 저장	내부 메서드
8	loadPositionFromFile()	long	메타데이터에서 포지션	내부 메서드

			읽기	
9	run()	void	Tailing 루프 실행	LogTailer 구현
10	MDC	Context	Thread-local 로깅 Context 저장	SLF4J MDC

처리로직

1. 메타데이터 파일을 초기화한다 (없으면 생성, 있으면 기존 위치를 로드).
2. 로그 파일을 RandomAccessFile로 연다.
3. 루프를 반복하며 다음을 수행한다:
    - 파일 크기가 줄어들면 (롤링 발생) 포지션을 0으로 초기화한다.
    - 새로운 로그 데이터가 있으면 새 부분을 읽고 라인 단위로 분리한 후 listener.onLogReceive()로 전달한다.
    - 변경된 파일 위치를 메타데이터 파일에 저장한다.
    - 지정된 readIntervalMs 동안 대기한 후 다시 반복한다.

7.1.4 Class: DefaultLogEventListener

No	Attribute	Type	Description	Notes
1	logParser	LogParser	문자열 로그 파싱	인터페이스
2	logFilter	LogFilter	필터 규칙 적용	인터페이스
3	logExporter	LogExporter	외부 서버 전송기	인터페이스
4	multilineProcessor	MultilineProcessor	멀티라인 병합기	인터페이스

5	fallbackStorage	FallbackStorage	실패 로그 저장소	인터페이스
6	onLogReceive()	void	수신 로그 처리 메서드	Listener 구현
7	exportLogs	List	Export 대상 버퍼	지역 변수
8	parse()	ParsedLogData	로그 파싱 수행	LogParser 사용
9	filter.accept()	boolean	필터링 통과 여부	LogFilter 사용
10	export()	List	로그 전송 및 실패 로그 반환	LogExporter 사용

처리로직

1. 멀티라인 병합기가 활성화된 경우, 입력된 로그 라인들을 병합한다.
2. 각 로그 라인에 대해 파서와 필터를 차례로 적용한다.
3. 필터를 통과한 로그만 export 버퍼에 추가한다.
4. 이전에 저장된 실패 로그(fallback)를 함께 로드한다.
5. 버퍼가 비어 있지 않으면:
    - Exporter를 통해 로그를 외부로 전송한다.
    - Fallback 저장소를 비우고, 실패한 로그를 다시 저장한다.

7.2 Streaming Server

7.2.1 Class: KafkaLogProducer

No	Attribute / Component	Type	Description	Notes

1	kafkaTemplate	KafkaTemplate<String, String>	Kafka 브로커와 통신하기 위한 템플릿 객체	Spring Bean 주입
2	kafkaConstant	KafkaConstant	Kafka 관련 설정 상수 (토픽명, DLQ 등)	상수 관리 객체
3	objectMapper	ObjectMapper	로그 데이터를 JSON으로 직렬화하는 Jackson 매퍼	의존성 주입
4	sendLog()	Mono	로그 데이터를 Kafka에 전송하는 메서드	LogProducer 인터페이스 구현
5	logData	Object	전송할 로그 객체	인자 값
6	logType	LogType	로그의 종류(SPRING_BOOT, TOMCAT 등)	enum 사용
7	agentId	String	로그를 보낸 에이전트 식별자	파티션 키로 사용
8	thNum	String	로그 파이프라인 스레드 번호	JSON 필드 포함
9	json	String	직렬화된 로그 데이터	내부 생성 값

10	log	Logger	Kafka 전송 로그 출력	Slf4j

처리 로직

1. 로그 데이터를 ObjectMapper를 사용하여 JSON 문자열로 직렬화한다.
2. 직렬화된 JSON에 logType, agentId, thNum 정보를 추가한다.
3. KafkaTemplate을 통해 지정된 로그 토픽으로 메시지를 전송한다.
4. CompletableFuture 콜백을 등록하여 전송 성공/실패 여부를 비동기적으로 처리한다.
5. 전송 성공 시 전송된 메시지의 크기, 토픽, 파티션, 오프셋 등을 로그로 남긴다.
6. 실패 시 예외를 로깅하고 sink.error()로 Mono 에러 신호를 전달한다.
7. 전체 과정은 별도의 쓰레드 풀(Schedulers.boundedElastic)에서 실행된다.

7.2.2 Class: KafkaLogConsumer

No	Attribute	Type	Description	Notes
1	pipeline	LogPipeline	로그 처리 파이프라인 (AI → WS → Storage)	의존 객체
2	dlqProducer	DlqProducer	DLQ(Dead Letter Queue) 전송기	의존 객체
3	consume()	void	Kafka로부터 로그 메시지를 수신	KafkaListener 어노테이션 적용
4	fallback()	void	DLQ 전송 처리 메서드	내부 호출
5	json	String	Kafka로부터 수신한 원본 로그 메시지	KafkaConsumer 인자

6	env	LogEnvelope	로그 데이터 래핑 객체	파싱 결과
7	LogParserUtil	Util Class	JSON 문자열을 LogEnvelope으로 변환	정적 유틸
8	onErrorResume()	Reactor Operator	파이프라인 오류 시 예외 처리	DLQ로 전송
9	log	Logger	로그 출력	Slf4j
10	groupId	String	Kafka consumer group ID	설정 기반

처리 로직

1. Kafka로부터 로그 메시지를 수신한다.
2. JSON 문자열을 LogParserUtil을 통해 LogEnvelope 객체로 파싱한다.
3. LogPipeline의 process() 메서드를 호출하여 로그를 처리한다.
4. 파이프라인 처리 중 오류가 발생하면 DLQ로 메시지를 전송한다.
5. 파싱 실패 시에도 동일하게 fallback()을 통해 DLQ로 메시지를 보낸다.
6. 모든 로그 처리 완료 후 Reactor Mono 구독을 통해 비동기 처리를 종료한다.

7.2.3 Class: DefaultLogPipeline

No	Attribute	Type	Description	Notes
1	processors	List	로그 처리 프로세서 목록 (AI, WebSocket, Storage 등)	의존 객체
2	process()	Mono	로그 처리 진입점	LogPipeline

				인터페이스 구현
3	env	LogEnvelope	로그 데이터 객체	입력 파라미터
4	groupBy(order)	Stream	order 기준으로 LogProcessor 그룹화	순서 보장
5	concatMap()	Reactor Operator	각 그룹의 처리를 순차 실행	비동기 흐름
6	Mono.when()	Reactor Operator	동일 order 그룹을 병렬 실행	병렬 처리
7	log	Logger	로그 처리 단계 출력	Slf4j
8	TreeMap	Map<Integer, List>	프로세서 정렬용 자료구조	order 기준 정렬
9	supports()	boolean	해당 로그 유형을 지원하는지 판단	Processor 내부 조건
10	then()	Mono	모든 처리 완료 후 반환	Reactor 체인 마감

처리 로직

1. 입력된 로그(LogEnvelope)를 수신하고 처리 시작 로그를 출력한다.
2. 등록된 LogProcessor 목록을 order 기준으로 TreeMap에 정렬한다.
3. 동일 order를 가진 프로세서들은 병렬로 실행하고, 다음 order는 순차적으로 진행한다.
4. 각 LogProcessor는 supports() 검사를 통해 대상 로그를 처리할지 결정한다.
5. process()를 호출하여 로그를 AI 분석, WebSocket 전송, Storage 저장 등으로 전달한다.
6. 모든 프로세서의 작업이 완료되면 Mono<Void>를 반환하여 파이프라인을 종료한다.

7.3 API Server

7.3.1 Class: DashboardService

No	Attribute	Type	Description	Notes
1	dashboardRepository	DashboardRepository	대시보드 엔티티에 대한 CRUD 수행 리포지토리	의존 객체
2	folderRepository	FolderRepository	폴더 정보 조회 및 접근 권한 확인 리포지토리	의존 객체
3	teamMemberRepository	TeamMemberRepository	팀 멤버 권한 검증을 위한 리포지토리	의존 객체
4	agentConfigurationRepository	AgentConfigurationRepository	대시보드와 연결된 Agent 설정 조회용 리포지토리	의존 객체
5	logPipelineConfigRepository	LogPipelineConfigRepository	로그 파이프라인 구성 정보 조회용 리포지토리	의존 객체
6	getDashboardsByFolder()	List<DashboardDto>	특정 폴더 내 대시보드 목록과 상태(Agent ID, thNum) 조회	AgentConfig 및 LogPipeline 기반으로 상태 파악

7	createDashboard()	DashboardDto	새로운 대시보드 생성	폴더 소유자 또는 팀 권한 확인 후 생성
8	updateDashboard()	DashboardDto	기존 대시보드 수정 (이름, 로그 경로 등)	폴더 ID 일치 여부 및 접근권한 검증 포함
9	deleteDashboard()	void	지정된 폴더 내 대시보드 삭제	트랜잭션 처리, 권한 확인 후 삭제
10	assertFolderReadable()	void	폴더 읽기 권한 검증	개인 폴더. 팀 폴더 구분
11	assertFolderWritable()	void	폴더 쓰기 권한 검증	Euquester Id 기반 권한 인
12	getDashboardIdsByFolder()	List<Long>	폴더 내 대시보드 Id 목록 조회	AgentConfigService 등 다른 모듈에서 사용

처리 로직

1. Folder ID를 기반으로 폴더의 존재 여부를 확인한다.
2. 요청자의 접근 권한을 검증한다.
3. 폴더에 속한 Dashboard 목록을 조회한다.
4. 각 Dashboard에 대해 AgentConfiguration에서 agentId를 조회한다.
5. LogPipelineConfig에서 thNum을 조회한다.
6. DashboardDto 형태로 매핑하여 반환한다.
7. 폴더 쓰기 권한이 있을 경우 새로운 Dashboard를 생성한다.
8. Folder 정보에 따라 개인 폴더 또는 팀 폴더로 구분하여 저장한다.
9. Dashboard 수정 요청 시 폴더 ID 일치 여부를 검증한다.

10. 권한 검증 후 이름과 로그 경로를 업데이트한다.
11. 대시보드 삭제 시 권한을 검증하고 트랜잭션으로 안전하게 삭제한다.
12. 폴더가 개인 소유인지 팀 소유인지에 따라 접근 권한을 구분한다.
13. TeamMemberRepository를 통해 팀 소속 여부를 확인한다.
14. 권한이 없을 경우 CustomException을 발생시킨다.
15. 폴더 ID를 기반으로 대시보드 ID 목록을 반환한다.

7.3.2 Class: FolderService

No	Attribute	Type	Description	Notes
1	folderRepository	FolderRepository	폴더 엔티티에 대한 CRUD 수행 리포지토리	의존 객체
2	teamRepository	FolderRepository	팀 폴더 생성을 위한 팀 정보 조회 리포지토리	의존 객체
3	userMemberRepository	UserRepository	개인 폴더 생성을 위한 사용자 정보 조회 리포지토리	의존 객체
4	createTeamFolder()	FolderDto	지정된 팀에 새 폴더를 생성	팀 ID를 기준으로 생성, 존재하지 않는 팀일 경우 예외 발생
5	createPersonalFolder()	FolderDto	사용자 개인 폴더를 생성	사용자 ID를 기준으로 폴더를 생성, 존재하지 않는 사용자일 경우 예외를 발생

6	getTeamFolders()	List<FolderDto>	특정 팀에 속한 모든 폴더 목록을 조회	BaseStatus.Y(활성) 상태의 폴더만 조회
7	getPersonalFolders()	List<FolderDto>	특정 사용자의 개인 폴더 목록을 조회	BaseStatus.Y(활성) 상태의 폴더만 조회
8	updateFolder()	FolderDto	폴더 이름을 수정	삭제된 폴더나 팀 폴더는 수정 안됨
9	deleteFolder()	void	폴더를 소프트 삭제	BaseStatus를 N으로 변경하여 상태만 비활성화

처리로직

1. 팀 폴더 생성 시 팀 ID로 팀 존재 여부를 확인한다.
2. 팀이 존재하지 않으면 예외를 발생시킨다.
3. 존재할 경우 폴더 엔티티를 생성하고 저장한다.
4. 개인 폴더 생성 시 사용자 ID로 사용자 존재 여부를 확인한다.
5. 사용자가 존재하지 않으면 예외를 발생시킨다.
6. 개인 폴더 엔티티를 생성하고 저장한다.
7. 팀 ID 또는 사용자 ID 기준으로 활성 상태(BaseStatus.Y)의 폴더 목록을 조회한다.
8. 폴더 수정 시 폴더 존재 여부를 확인한다.
9. 삭제된 폴더일 경우 수정할 수 없다는 예외를 발생시킨다.
10. 팀 폴더는 직접 수정이 불가능하며, 팀 수정 API를 통해 변경하도록 예외를 발생시킨다.
11. 개인 폴더의 이름을 수정하고 저장한다.
12. 폴더 삭제 시 존재 여부를 확인하고, 이미 삭제된 폴더는 예외를 발생시킨다.
13. 팀 폴더는 직접 삭제할 수 없으며, 팀 삭제 시 함께 처리되도록 한다.
14. 개인 폴더의 상태를 N으로 변경하여 소프트 삭제를 수행한다.

7.3.3 Class: AgentConfigService

No	Attribute	Type	Description	Notes
1	repository	AgentConfigurationRepository	Agent 설정 엔티티 관리 리포지토리	의존 객체
2	logPipelineRepository	LogPipelineConfigRepository	Log 파이프라인 설정 관리 리포지토리	의존 객체
3	objectMapper	ObjectMapper	객체 직렬화 및 JSON 파싱 유틸리티	의존 객체
4	dashboardService	DashboardService	대시보드 ID 조회용 서비스	의존 객체
5	saveConfig()	String	대시보드 설정을 저장 AgentConfig, PullerConfig, WatcherConfig를 조합하여 최종 ConfigDTO를 생성하고 저장	신규 Agent 생성 또는 기존 Agent 업데이트
6	getConfig()	ConfigDTO	Agent가 설정을 요청할 때, 최신 ConfigDTO를 반환	etag 비교로 변경 여부 판단
7	updatePipeline()	void	특정 로그 파일 경로(targetFilePath)에 해당하는 파이프라인 설정을	WatcherConfig, FilterConfig, PullerConfig 동적

			수정	갱신
8	getConfigsByFolder()	List<DashboardConfigResponse>	폴더 내 모든 대시보드의 설정 및 상태를 조회	에이전트의 응답 상태(수집 중/미응답) 포함

처리로직

1. saveConfig()는 요청받은 SaveDashboardConfigRequest를 기반으로 Agent ID를 확인한다.
2. Agent ID가 없으면 새 UUID로 생성하고 AgentConfiguration 엔티티를 저장한다.
3. PullerConfig, AgentConfig, WatcherConfig를 조립하여 ConfigDTO를 구성한다.
4. WatcherConfig는 로그 파일별 Tailer, Multiline, Exporter, Parser, Filter 정보를 포함한다.
5. 각 WatcherConfig는 직렬화되어 LogPipelineConfig 엔티티로 DB에 저장된다.
6. 최종 ConfigDTO를 JSON으로 변환하여 AgentConfiguration에 저장한다.
7. getConfig()는 Agent ID로 AgentConfiguration을 조회하고, etag가 동일하면 null을 반환한다.
8. etag가 다르면 ConfigDTO를 역직렬화하여 반환하며, 로그 파이프라인 구성을 다시 조립한다.
9. updatePipeline()은 대상 로그 파일 경로에 해당하는 파이프라인을 찾아 새로운 WatcherConfig로 갱신한다.
10. PullerConfig의 intervalSec이 요청에 포함된 경우 업데이트 후 전체 etag를 새로 생성한다.
11. getConfigsByFolder()는 폴더 내 모든 대시보드 ID를 조회하고, 각 대시보드의 파이프라인을 그룹화한다.
12. 에이전트의 마지막 pull 시간과 intervalSec을 기반으로 “수집 중” 또는 “에이전트 미응답” 상태를 계산한다.
13. 각 대시보드의 PullerConfig와 WatcherConfig 목록을 포함한 DashboardConfigResponse를 반환한다.

7.3.4 Class: UserService

No	Attribute	Type	Description	Notes

1	userRepository	UserRepository	사용자 엔티티 관리 리포지토리	의존 객체
2	jwtUtil	JwtUtil	JWT 토큰 발급 및 검증 유틸리티	의존 객체
3	passwordEncoder	PasswordEncoder	비밀번호 암호화 및 검증 유틸리티	의존 객체
4	validatePassword()	void	비밀번호의 조함, 길이, 이메일 중복 여부 검증	내부 유효성 검사 메서드
5	register()	User	신규 사용자를 등록 이메일 중복 검증, 비밀번호 유효성 검사 수행	비밀번호 암호화 후 저장
6	login()	LoginResponse	이메일과 비밀번호로 로그인한다. JWT 토큰을 발급	비밀번호 불일치 시 예외 발생
7	getUserByEmail()	User	이메일로 사용자 정보 조회	존재하지 않으면 예외 발생
8	updateUser()	LoginResponse	이메일 또는 비밀번호를 변경한다. 현재 비밀번호 일치 여부를 확인	변경 후 새 JWT 토큰 발급
9	isAvailableEmail()	boolean	이메일 중복 여부를 확인	회원가입 시 사용

10	deleteUser()	void	사용자 삭제 비활성 사용자 포함 검색 후 제거	Transactional 처리

처리로직

1. register()는 사용자 이메일 중복 여부를 검사한다.
2. 비밀번호를 validatePassword()로 검증하고, 조건(8자 이상, 영어+숫자+특수문자 조합, 이메일 불일치)을 만족해야 한다.
3. 비밀번호를 암호화한 뒤 User 엔티티를 저장한다.
4. login()은 이메일로 사용자를 조회하고, PasswordEncoder로 비밀번호 일치 여부를 확인한다.
5. 비밀번호가 일치하면 JwtUtil을 통해 JWT 토큰을 생성하고 LoginResponse를 반환한다.
6. getUserByEmail()은 이메일을 기준으로 사용자 정보를 반환한다.
7. updateUser()는 현재 비밀번호를 검증한 뒤, 새 이메일 또는 비밀번호를 갱신한다.
8. 새 비밀번호가 입력된 경우 validatePassword()를 다시 호출하여 유효성을 확인한다.
9. 사용자 정보 갱신 후 새 JWT 토큰을 발급하여 LoginResponse로 반환한다.
10. isAvailableEmail()은 입력된 이메일의 존재 여부를 반환한다.
11. deleteUser()는 비활성 사용자를 포함하여 해당 이메일의 사용자를 찾고, 삭제 처리한다.

7.3.5 Class: TeamService

No	Attribute	Type	Description	Notes
1	teamMemberRepository	TeamMemberRepository	팀 멤버 정보 관리 리포지토리	의존 객체
2	teamRepository	TeamRepository	팀 엔티티 관리 리포지토리	의존 객체

3	userRepository	UserRepository	사용자 정보 조회 리포지토리	의존 객체
4	folderService	FolderService	팀 생성 시 자동 폴더 생성을 위한 서비스	의존 객체
5	folderRepository	FolderRepository	팀 폴더 조회 및 상태 변경 리포지토리	의존 객체
6	getTeamsByUser()	List<TeamDTO>	사용자가 속한 모든 팀 목록 조회	BaseStatus.Y 상태의 팀만 반환
7	createTeam()	TeamDto	새로운 팀 생성 생성시 기본 폴더와 관리자, 멤버 등록	관리자 : 요청자
8	updateTeam()	TeamDto	팀 이름, 설명, 멤버 권한을 수정	관리자만 수정 가능
9	deleteTeam()	void	팀 삭제 팀 및 폴더를 soft delete 처리하며 멤버를 삭제	관리자 권한 필요
10	getTeamDetail()	TeamDetailDto	팀 상세 정보와 멤버 목록을 조회	요청자가 팀 멤버 이상의 권일 때만 접근 가능

처리로직

1. getTeamsByUser()는 사용자가 속한 모든 팀 멤버십을 조회한다.
2. BaseStatus.Y인 팀만 필터링하고, 각 팀의 폴더 ID를 함께 TeamDto로 반환한다.

3. createTeam()은 요청자의 정보를 기반으로 새로운 팀을 생성한다.
4. 팀 생성시 요청자를 ADMIN으로 등록하고, 나머지 멤버를 이메일 또는 ID로 조회하여 추가한다.
5. 생성된 팀을 저장한 후 FolderService를 통해 팀 전용 폴더를 자동 생성한다.
6. generateInviteUrl()은 UUID 기반으로 고유 초대 코드를 생성하고 초대 링크를 반환한다.
7. updateTeam()은 관리자 권한 검증 후 팀 이름, 설명, 멤버 구성을 갱신한다.
8. 팀 이름이 변경되면 폴더 이름도 동일하게 갱신한다.
9. 요청된 멤버 리스트를 기준으로 기존 멤버를 제거하거나 역할(Role)을 업데이트한다.
10. deleteTeam()은 관리자만 실행할 수 있으며, 팀과 관련된 폴더를 soft delete 처리한다.
11. 팀 멤버는 하드 삭제로 제거한다(추후 soft delete 전환 가능).
12. getTeamDetail()은 팀과 폴더 정보를 조회한 뒤, 팀 멤버 전체 목록을 TeamMemberDto로 변환하여 반환한다.
13. 요청자가 해당 팀의 멤버가 아닐 경우 접근이 거부된다.