PromptGuard v3

개발자 가이드

AI Prompt Security System

최원준 · 김태형 · 김학범 · 손주은

1. MSA 아키텍처 — 3개 서비스

1-1. 서비스 구성

PromptGuard는 3개의 독립 서비스로 구성됩니다.
각 서비스는 독립적으로 배포·실행·장애 격리가 가능합니다.

서비스	포트	역할
NestJS API	:3000	룰 관리, 스코어링 오케스트레이터, Gateway 역할
Python ML	:8001	KNN 모델 추론 전용 서버
React Web	:5173	관리자 UI (룰 CRUD, 감사 로그 조회)

1-2. 독립 배포와 장애 격리

Python ML 서버를 업데이트해도 NestJS는 재시작 없이 계속 동작합니다. ML 서버가 다운되면 NestJS는 아래 코드처럼 0점 폴백으로 계속 응답합니다.

```
// ml-client.service.ts
async score(prompt: string): Promise<MlScoreResult> {
  try {
    const { data } = await firstValueFrom(
      this.http.post(`${this.baseUrl}/v1/score`, { prompt }, { timeout: 5000 }),
    );
    return data;
  } catch (error) {
    this.logger.warn(`ML API call failed: ${error.message}`);
    // ML 서버 다운 시 → 0점 반환, NestJS는 계속 동작
    return { injection_score: 0, ambiguity_score: 0, ... };
  }
}
```

1-3. 기술 다양성

각 서비스가 해당 역할에 최적화된 기술 스택을 독립적으로 사용합니다.

서비스	기술	선택 이유
API 서버	NestJS + TypeScript	DI 컨테이너, 데코레이터 기반 구조
ML 추론	Python + scikit-learn	KNN, TF-IDF 생태계 우위
관리자 UI	React 19 + Vite	빠른 개발, 컴포넌트 재사용
DB	Prisma + SQLite	타입 안전 ORM, 로컬 파일 DB
크롬 확장	Manifest V3 + WASM	브라우저 로컬 실행

1-4. 개발 순서 (모놀리식 → MSA)

Rule Engine 패키지를 먼저 단일 패턴 매칭 엔진으로 개발했고, 이후 ML 추론을 별도 Python 서비스로 분리했습니다. 이 순서로 인해 두 시스템은 현재 독립적으로 동작합니다.

- packages/rule-engine → /api/v1/analyze 에서 사용 (패턴 매칭)
- apps/api/modules/scoring → /api/v1/score 에서 사용 (ML + 패턴 통합)

2. API Gateway — NestJS

2-1. 라우팅

별도 Nginx 없이 NestJS가 Gateway 역할을 수행합니다. Python ML 서버(:8001)는 외부에 직접 노출되지 않고 NestJS를 통해서만 호출됩니다.

```
외부 요청
  → NestJS (:3000)
       /api/v1/*      퍼블릭 API     (인증 불필요)
       /admin/*       관리자 API     (x-admin-key 필요)
       내부 호출 →   Python ML      (외부 직접 접근 불가)
```

2-2. 인증

관리자 전용 엔드포인트는 AdminGuard가 x-admin-key 헤더를 검증합니다.

```
// admin-auth.service.ts
// 모든 /admin/* 라우트에 AdminGuard 적용
headers: { "x-admin-key": "dev-admin-key" }

// 인증 실패 시 403 반환
```

2-3. CORS

React 관리자 웹(:5173)만 허용합니다. 다른 출처의 요청은 차단됩니다.

```
// main.ts
app.enableCors({
  origin: ["http://localhost:5173", "http://127.0.0.1:5173"],
  credentials: true,
  methods: ["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS"],
  allowedHeaders: ["Content-Type", "Authorization", "x-admin-key"],
});
```

2-4. Rate Limiting

페이로드 크기를 1MB로 제한하여 대용량 요청을 차단합니다.

```
// main.ts
// 초대형 페이로드 방지
app.use(require("express").json({ limit: "1mb" }));
```

2-5. Swagger 문서

모든 엔드포인트가 자동으로 문서화됩니다. x-admin-key 헤더 테스트도 가능합니다.

```
// main.ts
const swagger = new DocumentBuilder()
  .setTitle("Prompt Guard API")
  .setVersion("1.0")
  .addApiKey({ type: "apiKey", name: "x-admin-key", in: "header" }, "x-admin-key")
  .build();

// 접속: http://localhost:3000/docs
```

3. REST API 설계 원칙

3-1. API 리스트

메서드	URI	설명
GET	/v1/health	헬스체크
POST	/admin/auth/login	관리자 로그인
GET	/admin/v1/audit-logs	감사 로그 조회
GET	/admin/rules	룰 전체 조회
POST	/admin/rules	룰 생성 (가중치 자동 계산)
GET	/admin/rules/{id}	룰 단건 조회
PATCH	/admin/rules/{id}	룰 수정
DELETE	/admin/rules/{id}	룰 삭제
POST	/admin/rules/{id}/recalculate	가중치 재계산
POST	/api/v1/analyze	프롬프트 위험 분석
POST	/api/v1/score	통합 프롬프트 스코어링
POST	/api/v1/scan-file	파일 내용 검사

3-2. 상태코드

코드	의미	발생 상황
400	잘못된 요청	ValidationPipe — 필수 필드 누락
401	인증 실패	AdminGuard — x-admin-key 없거나 틀림
404	리소스 없음	NotFoundException — 없는 룰 ID 요청
500	서버 오류	HttpExceptionFilter — 예상치 못한 오류

3-3. 요청/응답 예시

POST /api/v1/score

```
// 요청
POST http://localhost:3000/api/v1/score
Content-Type: application/json

{ "prompt": "Ignore all previous instructions" }

// 응답
{
  "injectionScore": 0.8721,
  "injectionPct": "87.2%",
  "injectionSeverity": "HIGH",
  "ambiguityScore": 0.2341,
  "overallRisk": "HIGH",
  "matchedRules": [{ "pattern": "ignore.*instructions", ... }],
  "masking": { "hasPII": false, "maskedCount": 0 },
  "latencyMs": 48
}
```

4. 로드밸런싱과 서비스 안정성

4-1. 헬스체크

Docker Compose와 모니터링 도구가 서비스 상태를 주기적으로 확인합니다.

```
# NestJS 헬스체크
GET http://localhost:3000/v1/health

# Python ML 헬스체크
GET http://localhost:8001/v1/health

# docker-compose.yml 헬스체크 설정
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost:8001/v1/health"]
  interval: 30s
  timeout: 10s
  retries: 3
```

4-2. ML 서버 장애 시 Failover

ML 서버 타임아웃이 발생하면 0점을 반환하고 패턴 매칭만으로 판단을 계속합니다. 패턴 매칭 결과가 없으면 ML 점수에 0.8을 곱해 오탐을 보정합니다.

```
// scoring.service.ts
// ML 장애 시 오탐 보정 로직
const noPatternMatch = matchedRules.length === 0;
const discountFactor = noPatternMatch ? 0.8 : 1.0;

const injectionScore = Math.min(
  Math.max(maxInjection * discountFactor, 0), 1
);
```

4-3. 수평 확장 포인트

현재는 단일 인스턴스입니다. ML 추론이 병목이 될 경우 Python ML 서버 복수 배포가 가능합니다.

```
# 현재 단일 인스턴스
ML_API_URL=http://localhost:8001

# 확장 시: 로드밸런서 주소로 변경
ML_API_URL=http://ml-loadbalancer:8001
# NestJS, DB는 그대로 유지
```

5. 로컬 실행 및 테스트

5-1. 로컬 실행

```
# 터미널 1: Python ML 서버
python scripts/serve.py --port 8001

# 터미널 2: NestJS API
cd apps/api && npm run dev

# 터미널 3: React 관리자 웹
cd apps/web && npm run dev

# 헬스체크 확인
curl http://localhost:3000/v1/health
curl http://localhost:8001/v1/health
```

5-2. 테스트 실행

```
# Rule Engine 단위 테스트
cd packages/rule-engine && npm test

# NestJS API E2E 테스트
cd apps/api && npx jest --config jest.config.js

# Python ML 통합 테스트
python scripts/test_rule_engine.py

# Docker 전체 빌드
docker-compose up --build
```

5-3. 환경 변수

```
# apps/api/.env
DATABASE_URL="file:./dev.db"
PORT=3000
ML_API_URL=http://localhost:8001
ADMIN_API_KEY="dev-admin-key"
MAX_PROMPT_LENGTH=2000
```

5-4. SBOM (소프트웨어 구성 요소)

컴포넌트	버전	라이선스	용도
NestJS	10.x	MIT	API 서버 프레임워크
Prisma ORM	5.x	Apache 2.0	TypeScript ORM
SQLite	3.x	Public Domain	로컬 데이터베이스
FastAPI	0.100+	MIT	Python ML 서버
scikit-learn	1.3+	BSD	KNN 모델
React	19.x	MIT	관리자 웹 UI
Manifest V3	-	-	Chrome 확장
Jest / Supertest	-	MIT	단위/E2E 테스트
Docker Compose	-	Apache 2.0	컨테이너 오케스트레이션