Contents

LFIT Admin API 아키텍처 문서

1. 프로젝트 구조

```
src/
  application/        # 애플리케이션 서비스 계층
     ad.service.ts
     mission.service.ts
     jwt.service.ts
     auth.service.ts
  domain/             # 도메인 계층
     auth.ts
     errors.ts
  infrastructure/     # 인프라스트럭처 계층
     ad.repository.ts
     mission.repository.ts
     dashboard.repository.ts
  interfaces/         # 인터페이스 계층
     ad.controller.ts
     mission.controller.ts
     ad.routes.ts
     mission.routes.ts
     reward.claims.routes.ts
```

```
middleware/         # 미들웨어
    auth.ts
types/              # 타입 정의
    ad.ts
    mission.ts
    user.ts
app.ts              # 애플리케이션 진입점
```

2. 아키텍처 다이어그램

2.1 계층 구조

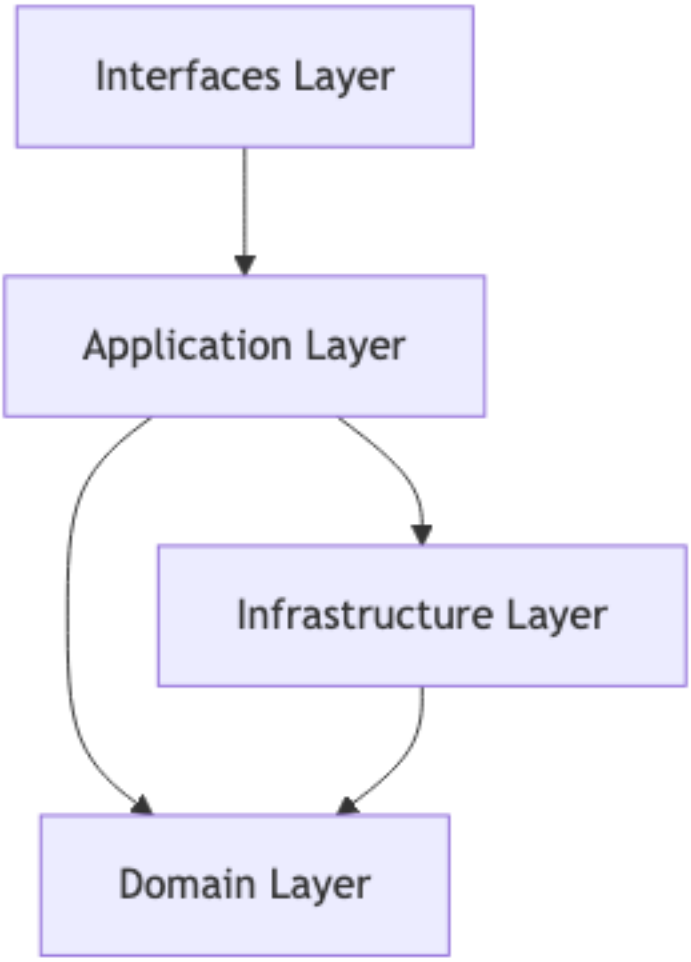

Figure 1: Architecture Diagram

2.2 요청 흐름도

3. 계층별 상세 설명

3.1 인터페이스 계층 (interfaces/)

- HTTP 요청/응답 처리
- 입력 유효성 검증

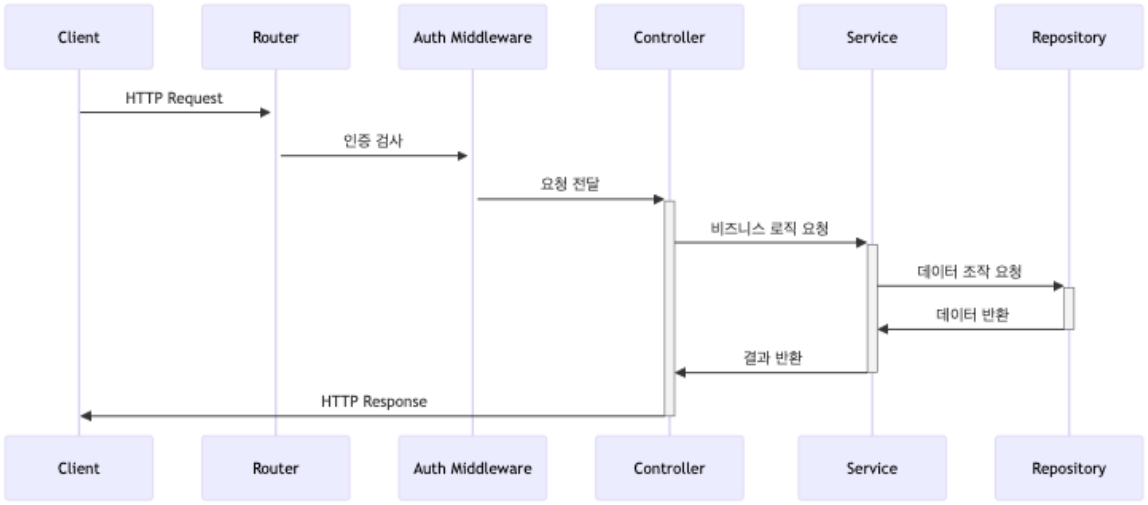

Figure 2: Flow Diagram

- 라우팅 설정
- 컨트롤러를 통한 요청 처리

컨트롤러 구조

```
export class AdController {
    constructor(private readonly adService: AdService) {}

    // CRUD 작업 메서드
    async getAll(): Promise<void>
    async getById(id: string): Promise<void>
    async create(dto: CreateDto): Promise<void>
    async update(id: string, dto: UpdateDto): Promise<void>
}
```

3.2 애플리케이션 계층 (application/)

- 비즈니스 로직 구현
- 트랜잭션 관리
- 도메인 객체 조작
- 서비스 간 조율

서비스 구조

```
export class AdService {
    constructor(private readonly repository: AdRepository) {}

    // 비즈니스 로직 메서드
    async getAllAds(): Promise<Ad[]>
    async createAd(dto: CreateAdDto): Promise<Ad>
    async updateAd(id: string, dto: UpdateAdDto): Promise<Ad>
}
```

3.3 도메인 계층 (domain/)

- 핵심 비즈니스 규칙
- 도메인 모델 정의
- 도메인 이벤트

도메인 모델 예시

```
export class Ad {
    private readonly id: string;
    private title: string;
    private reward: number;
    private isActive: boolean;

    constructor(props: AdProps) {
        this.validateProps(props);
        // ... 초기화 로직
    }

    // 도메인 규칙 및 비즈니스 로직
    activate(): void
    deactivate(): void
    updateReward(amount: number): void
}
```

3.4 인프라스트럭처 계층 (infrastructure/)

- 데이터베이스 연동
- 외부 서비스 통합
- 저장소 구현

리포지토리 구조

```
export class AdRepository {
    constructor(private readonly db: Database) {}

    // CRUD 작업
    async findAll(): Promise<Ad[]>
    async findById(id: string): Promise<Ad>
    async create(ad: Ad): Promise<Ad>
    async update(id: string, ad: Ad): Promise<Ad>
}
```

4. 데이터베이스 스키마

4.1 광고 보상 관련 테이블

4.2 미션 보상 관련 테이블

5. API 상세 명세

5.1 광고 보상 API

엔드포인트	메서드	설명	권한
/api/ads	GET	모든 광고 보상 조회	Admin
/api/ads/:id	GET	특정 광고 보상 조회	Admin

엔드포인트	메서드	설명	권한
/api/ads	POST	새로운 광고 보상 생성	Admin
/api/ads/:id	PUT	광고 보상 수정	Admin

요청/응답 예시:

```
// POST /api/ads
Request:
{
    "title": " 동영상 시청",
    "reward": 100,
    "daily_limit": 5
}

Response:
{
    "success": true,
    "data": {
        "id": "uuid",
        "title": " 동영상 시청",
        "reward": 100,
        "daily_limit": 5,
        "is_active": true,
        "created_at": "2024-03-20T00:00:00Z"
    }
}
```

6. 보안 아키텍처

6.1 인증 흐름도

6.2 권한 체계

- USER: 일반 사용자 권한
- ADMIN: 관리자 권한
- SUPER_ADMIN: 최고 관리자 권한

7. 개선 필요 사항

7.1 도메인 계층 강화

- 도메인 이벤트 추가 필요
- 도메인 객체의 불변성 보장 강화
- Value Object 패턴 적용 검토

7.2 보안 강화

- Rate Limiting 구현
- API 키 관리 체계 구축
- CORS 설정 상세화
- 입력값 검증 강화

7.3 에러 처리

```
// 글로벌 에러 핸들러 예시
app.use((err: Error, req: Request, res: Response, next: NextFunction) => {
```

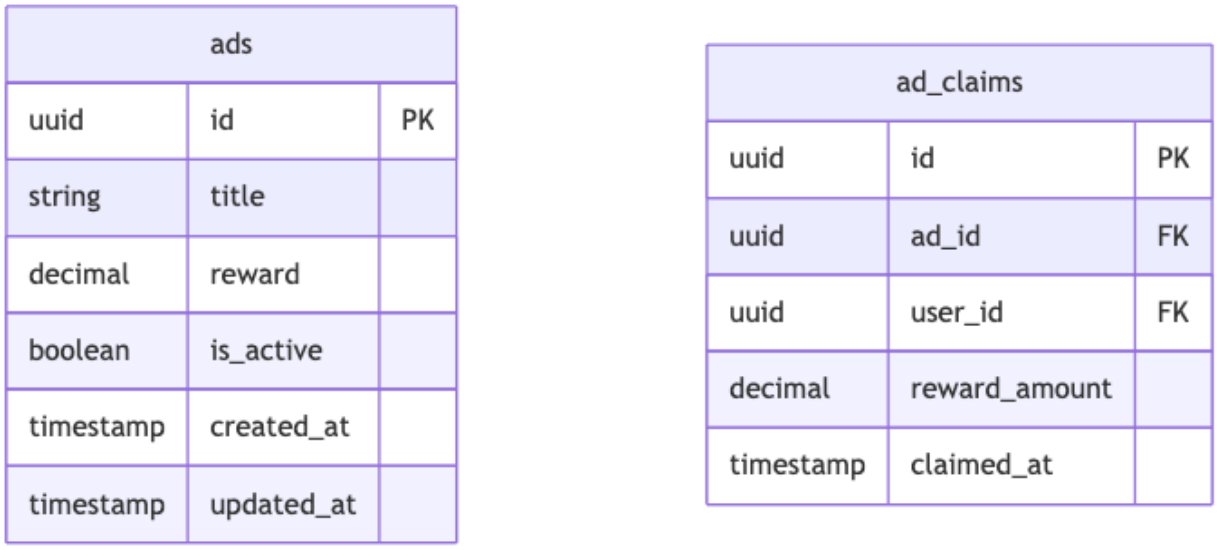

Figure 3: Ad Schema Diagram

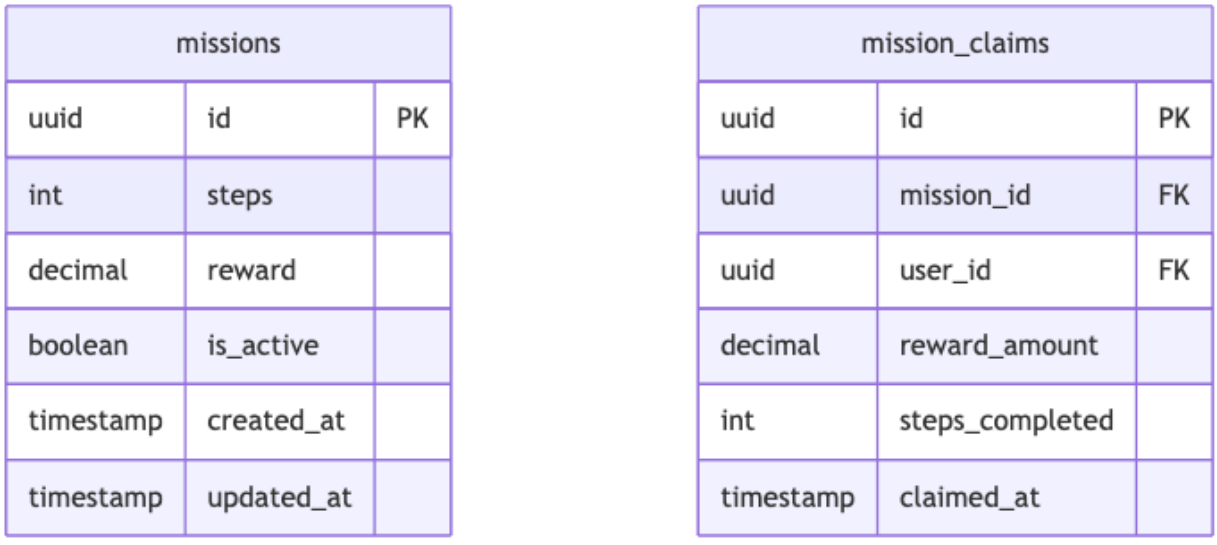

Figure 4: Mission Schema Diagram

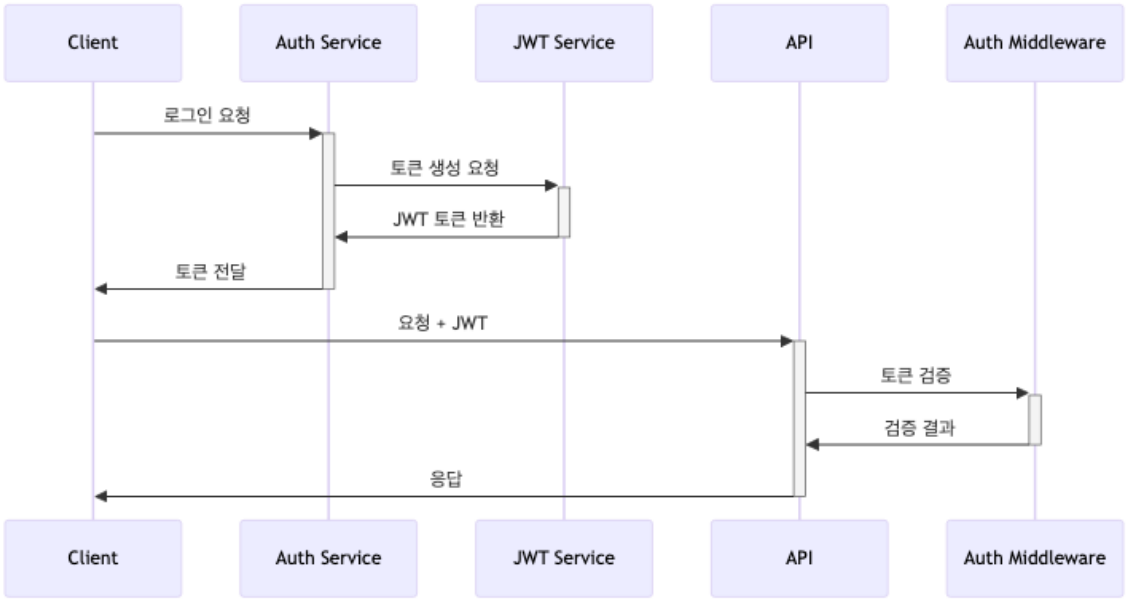

Figure 5: Auth Flow Diagram

```
    if (err instanceof ValidationError) {
        return res.status(400).json({
            success: false,
            error: {
                code: 'VALIDATION_ERROR',
                message: err.message,
                details: err.details
            }
        });
    }
    // ... 다른 에러 처리
});
```

7.4 성능 최적화

- 캐싱 전략 수립
  - Redis 를 활용한 캐시 레이어 구현
  - 캐시 무효화 전략 수립
- 데이터베이스 쿼리 최적화
  - 인덱스 최적화
  - 쿼리 실행 계획 분석
- N+1 문제 해결
  - DataLoader 패턴 적용
  - Join 쿼리 최적화

7.5 모니터링

- 로깅 시스템 구축
  - ELK 스택 도입
  - 로그 레벨 체계화
- 성능 메트릭 수집

  - Prometheus + Grafana 구축
  - 주요 지표 대시보드 구성
- 알림 시스템 구축
  - 임계치 기반 알림
  - 에러 발생 시 알림

8. 향후 개발 계획

8.1 단기 목표 (1-3 개월)

- 테스트 커버리지 향상
  - 단위 테스트 80% 이상
  - 통합 테스트 시나리오 구축
- API 문서화 (Swagger/OpenAPI)
  - 전체 API 엔드포인트 문서화
  - 예제 코드 추가
- 환경 설정 관리 개선
  - 환경변수 검증 로직 추가
  - 설정 값 중앙화

8.2 중기 목표 (3-6 개월)

- 마이크로서비스 아키텍처 검토
  - 서비스 분리 계획 수립
  - 서비스 간 통신 구조 설계
- 이벤트 기반 아키텍처 도입
  - 이벤트 버스 구축
  - 비동기 처리 로직 구현
- 실시간 처리 기능 강화
  - WebSocket 기반 실시간 알림
  - 실시간 데이터 동기화

8.3 장기 목표 (6 개월 이상)

- 확장 가능한 인프라 구축
  - 컨테이너 오케스트레이션
  - 자동 스케일링
- 데이터 분석 파이프라인 구축
  - 데이터 웨어하우스 구축
  - BI 도구 연동
- AI/ML 기능 통합
  - 사용자 행동 분석
  - 보상 최적화 알고리즘