LastMile

Developer Manual

초정밀 보행 최단 경로 최적화 플랫폼

Team 모세의 기적

곽민석 | PM

남진우 | Tech Lead

김영서 | Data Engineer

이준혁 | Research Lead

App Version: **V2.3.1** | Doc Version: **V3.0**

개발 기간: **2025.03.25 ~ 2025.04.03**

목차 (Table of Contents)

서론

이 문서의 목적

본 문서는 LastMile 서비스(V2.3.1)의 개발자 매뉴얼입니다. 신규 참여 개발자, 유지보수 담당자, 기술 심사자가 코드베이스를 빠르게 파악하고 독립적으로 작업할 수 있도록 작성되었습니다. 단순한 API 나열이 아니라, 설계 결정의 이유와 도메인 간 연결 구조까지 담는 것을 목표로 합니다.

대상 독자

- 신규 개발자 — 온보딩 시 코드 구조와 도메인 역할을 파악하기 위한 첫 번째 참고 문서
- 유지보수 담당자 — 버그 수정·기능 추가 시 영향 범위를 파악하기 위한 기술 레퍼런스
- 기술 심사자 — 아키텍처·보안·의존성 현황을 한 문서에서 확인하기 위한 종합 리포트

문서 범위

본 문서는 다음 내용을 포함합니다:

- 기능 명세 — Users·Navigation·Spatial·ETL 도메인별 비즈니스 로직 및 핵심 알고리즘
- API 명세 — 전 엔드포인트 요청/응답 스펙 및 cURL 예시
- DB 스키마 — PostgreSQL + PostGIS 전체 테이블 구조 및 관계
- SBOM 및 취약점 분석 — syft v1.42.3 으로 의존성 목록 추출, grype v0.110.0 으로 CVE 스캔 (2026-04-03 기준, 총 244 개 패키지, CRITICAL·HIGH 0 건)
- CI/CD 및 빌드 — Docker Compose 기반 빌드·실행 방법 및 데이터 시딩 절차
- 도메인 독립성 분석 — 각 도메인의 결합도 및 독립 업데이트 가능 여부
- 모니터링 현황 — 현재 구성된 관측 수단 및 향후 권장 도입 방안

※ 본 문서는 프론트엔드 UI/UX 상세 설명, 인프라 운영(서버 설정·SSL 등), 비즈니스 기획 내용은 다루지 않습니다.

버전 이력

문서 버전	앱 버전	주요 변경 사항
V1.0	V2.3.0	초안 작성 — 기능 명세·API·DB 스키마·SBOM 포함

V2.0	V2.3.0	표지 팀원 정보·포지션 추가, 개발 기간 명시, SBOM 취약점 분석 결과 반영
V3.0	V2.3.1	서론 추가, 모니터링 현황 섹션(§12) 신설, 버전 정보 갱신

1. 프로젝트 개요

LastMile은 서울 도심의 보행 이동성을 위한 경로 최적화 플랫폼입니다. 단순 최단 거리를 넘어 계단·경사·조명·CCTV 등의 안전 요소를 반영한 3가지 경로 프로파일(최단/균형/안전)을 동시에 제공합니다. 표준 보행자·휠체어 사용자·유모차 사용자를 모두 지원하며, 주민 제보로 발견된 비공식 지름길(쪽문·출입구)도 관리자 승인 후 실시간 라우팅 그래프에 반영됩니다.

1.1 기술 스택

레이어	기술	버전
Backend	Go + gin-gonic/gin	Go 1.26.1 / Gin v1.12.0
DB Driver	jmoiron/sqlx + lib/pq	v1.4.0 / v1.12.0
Database	PostgreSQL 16 + PostGIS 3.4	Docker: postgis/postgis:16-3.4
Auth	lestrrat-go/jwx/v2 + golang.org/x/crypto	v2.1.6 / v0.49.0
API Docs	swaggo/gin-swagger	v1.6.1
Frontend	React 19 + TypeScript + Vite	React v19 / TS ~5.9.3 / Vite v8
UI	Radix UI + shadcn/ui + Tailwind CSS	Tailwind v4.2.2
Map	Naver Maps SDK (주) + Leaflet (관리자)	Leaflet v1.9.4
HTTP Client	Axios	v1.13.6
External API	Naver Local Search API + OpenStreetMap	REST / 공개 데이터

1.2 디렉토리 구조

```
lastmile-Lastmile_V-2.3.0/
├── docker-compose.yml          # DB / backend / frontend / seed 오케스트레이션
├── last_mile_backend/          # Go 백엔드 (메인 애플리케이션)
│   ├── main.go                 # 진입점: DI 연결 + Gin 서버 시작
│   ├── go.mod / go.sum         # Go 모듈 정의
│   ├── Dockerfile              # 멀티스테이지 백엔드 이미지
│   ├── Dockerfile.seed         # 원샷 시드 이미지
│   ├── .env.example            # 환경 변수 템플릿
│   ├── core/                   # JWT 매니저 + Gin 미들웨어
│   ├── users/                  # 인증·프로필·선호도 도메인
```

```
│   ├── navigation/              # Dijkstra 라우팅 엔진 도메인
│   ├── spatial/                 # 그래프 데이터·지름길·Naver 검색 도메인
│   ├── etl/                     # 데이터 수집·GPX 분석·피드백 도메인
│   ├── db/                      # 스키마 SQL (DB 초기화 시 자동 적용)
│   ├── docs/                    # Swagger 자동 생성 문서
│   ├── cmd/                     # CLI 도구 (seed, enrich_edges, import_slope_json)
│   └── scripts/                 # 데이터 준비 쉘/파이썬 스크립트 + SQL 패치
└── last_mile_frontend/          # React/TypeScript SPA
    ├── src/
    │   ├── main.tsx             # React 진입점
    │   ├── App.tsx              # 루트: AuthContext + React Router
    │   ├── api/client.ts        # Axios 인스턴스 + JWT 인터셉터
    │   ├── pages/               # 페이지 컴포넌트
    │   └── components/          # 공유 UI 컴포넌트
    ├── index.html               # Naver Maps SDK 스크립트 삽입
    └── Dockerfile               # 프론트엔드 컨테이너 이미지
```

1.3 진입점 파일

서비스	파일
Go 백엔드	last_mile_backend/main.go
React 프론트엔드	last_mile_frontend/src/main.tsx
DB 스키마 초기화	last_mile_backend/db/schema.sql
OSM 그래프 시드	last_mile_backend/cmd/seed/main.go
엣지 안전 데이터 보강	last_mile_backend/cmd/enrich_edges/main.go
경사 데이터 임포트	last_mile_backend/cmd/import_slope_json/main.go

2. 아키텍처

2.1 전체 시스템 구조

백엔드는 도메인별 엄격한 계층 아키텍처를 따릅니다:

```
HTTP Request
  ↓
Gin Router (main.go)
  ↓
Handler (users/handler.go 등)      ← HTTP: 바인딩, 검증, 에러 → 상태코드 변환
  ↓
Service (users/service.go 등)      ← 비즈니스 로직: 규칙, 오케스트레이션
  ↓
Repository (users/repository.go)   ← 데이터 접근: sqlx + lib/pq 파라미터화 SQL
  ↓
PostgreSQL + PostGIS
```

Navigation 도메인의 특이사항: NavigationService는 SpatialRepository 인터페이스를 통해 라우팅 그래프를 로드한 후 순수 인메모리 Engine 레이어(navigation/engine.go)로 전달합니다. 즉 Navigation 엔진은 DB 의존성이 없습니다.

2.2 도메인 브레이크다운

도메인	폴더	역할	핵심 파일
Core	core/	JWT 토큰 생명주기 + Gin 인증·관리자 미들웨어	jwt.go, middleware.go
Users	users/	등록·로그인·토큰 로테이션·선호도 관리	handler.go, service.go, repository.go
Navigation	navigation/	그래프 로드·Dijkstra·3+1 경로 변형	handler.go, service.go, engine.go
Spatial	spatial/	그래프 영속성·지름길 크라우드소싱·Naver 검색	handler.go, service.go, repository.go
ETL	etl/	안전 데이터 수집·GPX 분석·피드백 가중치 조정	handler.go, service.go, collector.go

2.3 의존성 주입 (main.go)

모든 의존성은 IoC 컨테이너 없이 main.go에서 명시적으로 하향식으로 구성됩니다:

```
// 1. DB 연결
db, _ := sqlx.Connect("postgres", dbURL)

// 2. JWT 매니저 (개발 시 RSA 키 자동 생성 — 프로덕션에서는 영속 키 필수)
jwtManager, _ := core.NewJWTManagerWithGeneratedKey(jwtSecret)

// 3. Spatial (NavigationService와 SpatialService가 공유하는 Repo)
spatialRepo := spatial.NewPostgresRepository(db)
naverClient, _ := spatial.NewNaverSearchClient()
spatialSvc  := spatial.NewSpatialService(spatialRepo, naverClient)

// 4. Navigation (SpatialRepository 인터페이스로 spatialRepo 주입)
navSvc  := navigation.NewNavigationService(spatialRepo)

// 5. Users
usersRepo := users.NewPostgresUserRepository(db)
usersSvc  := users.NewUserService(usersRepo, jwtManager)

// 6. ETL (MockCollector → FileCollector로 교체 시 운영 데이터 활성화)
etlCollector := etl.NewMockCollector()
etlSvc       := etl.NewETLService(etlCollector, db)
```

2.4 미들웨어 스택

미들웨어	파일	동작
CORS	main.go (gin-contrib/cors)	localhost:5173 허용, GET/POST/PATCH/DELETE/OPTIONS, credentials 허용
Recovery	main.go (gin.Recovery())	패닉 캐치 → 500 반환
AuthMiddleware	core/middleware.go	Bearer 토큰 추출 → RS256 JWT 검증 → *JWTClaims를 Gin 컨텍스트 'jwt_claims'에 주입
AdminMiddleware	core/middleware.go	AuthMiddleware 이후 체이닝 필수; claims.Role == 'admin' 확인, 불일치 시 403

※ V2.3.1에는 레이트 리밋 미들웨어가 없습니다.

3. 기능 명세서

3.1 Users 도메인

사용자 전체 생명주기를 관리합니다: 역할 선택(초대 코드로 admin 등록) 계정 생성, 타이밍 어택 방어 로그인, JWT 토큰 발급·로테이션, 선호도(라우팅 가중치 슬라이더 0~3) 영속화, GDPR 준수 소프트 삭제(PII 익명화).

데이터 모델

구조체	주요 필드	설명
User	ID, Email, UserType, Role, IsActive, DeletedAt	UUID PK; UserType: standard|wheelchair|stroller; Role: user|admin; DeletedAt: 소프트 삭제 타임스탬프
UserPreferences	StairAvoidance, SlopeAvoidance, SafetyPriority, MainRoadPrefer	모두 0~3 범위 정수 (0=비활성화, 3=최대); users.id FK
RefreshToken	TokenHash, ExpiresAt, Revoked	SHA-256 해시 저장; Revoked 플래그로 재사용 감지

핵심 알고리즘

- 비밀번호 검증 (validatePassword): 최소 8 자, 대문자·소문자·숫자·특수문자 각 1 개 이상
- 타이밍 어택 방어 (Login): 사용자 조회 실패 시 더미 bcrypt 비교 실행 → 응답 시간 일정화, 이메일 열거 방지
- 토큰 로테이션 (RefreshTokens):
  - ① JWT 서명 검증 (HS256) → ② refresh_tokens 테이블에서 SHA-256 해시 조회 + 유효성 확인
  - ③ 사용된 토큰 즉시 폐기 → ④ 재사용 감지 시 해당 유저의 모든 토큰 일괄 폐기
  - ⑤ 새 Access + Refresh 토큰 쌍 발급 + 새 해시 저장
- 소프트 삭제 (SoftDeleteUser): 트랜잭션 내 이메일 익명화(deleted_<uuid>@removed.invalid) + 비밀번호 해시 삭제 + deleted_at 설정 + 모든 Refresh Token 폐기

3.2 Navigation 도메인

BBox 범위 도로 그래프를 Spatial 레포지터리에서 로드하고, 출발지·목적지 좌표를 가장 가까운 연결 노드에 스냅한 후, 4가지 Dijkstra 병렬 연산으로 최단·균형·안전·커스텀 경로를 반환합니다.

엣지 비용 공식

```
Total_Cost = Distance × (1 + ΣPenalties)

stair_penalty  = coeff(StairAvoidance) × 1.0          [계단 존재 시]
slope_penalty  = coeff(SlopeAvoidance) × SlopeGrade
light_penalty  = coeff(SafetyPriority) × (1 − LightDensity)
cctv_penalty   = coeff(SafetyPriority) × (1 − CCTVDensity)
safety_penalty = (light_penalty + cctv_penalty) / 2
road_penalty   = coeff(MainRoadPrefer) × 0.3           [메인도로 아닌 경우]
```

강도→계수 변환 (intensityToCoefficient)

레벨	계수	효과
3 (최대)	5.0	엣지 비용 최대 ~6배
2 (중)	2.0	엣지 비용 ~3배
1 (최소)	0.5	엣지 비용 ~1.5배
0 (비활성)	0.0	페널티 없음

경로 변형

변형	동작
shortest	모든 UserWeights = 0 강제 → 순수 거리
balanced	사용자의 base_weights 그대로 적용
safe	SafetyPriority=3, StairAvoidance=3, SlopeAvoidance=2 강제 (사용자 설정 무시)
custom	사용자의 custom_weights 그대로 적용

특수 로직

- BBox 재시도: 그래프 로드 실패 시 패딩 ×1 → ×2 → ×4 순서로 재시도 (기본 0.01° ≈ 1.1km)
- 노드 스냅 (mapToConnectedNodes): 양 끝점 각 8 개 후보 → 총 거리 기준 정렬 → BFS 연결성 검사 → 첫 번째 연결된 쌍 선택

- 병렬 처리 (computeAllEdgeCosts): 각 엣지 비용을 별도 고루틴에서 계산, sync.WaitGroup 동기화, cloneGraph()로 데이터 경쟁 방지
- Dijkstra: container/heap 기반 min-heap, dist(가중치 비용)과 realDist(실제 미터) 동시 추적, 목적지 팝 시 조기 종료
- 거리 계산: Haversine(도로 엣지) + Euclidean(공원 내 단거리)

3.3 Spatial 도메인

라우팅 그래프 영속성(PostGIS 쿼리), 시민 지름길 크라우드소싱(관리자 승인 워크플로우), Naver Local Search API 프록시, 공원 가시성 그래프, 관리자 안전 지도 데이터를 제공합니다.

핵심 알고리즘

- Gateway → Graph 주입 (ApplyGatewayGraph): 관리자 승인 시 트랜잭션 내 실행
  - path_coords ≥2 포인트: gw_<id>_a / gw_<id>_b 노드 생성 + 가장 가까운 도로 노드에 연결 + 지름길 엣지 추가
  - 단일 포인트: gw_<id>_p 노드 생성 + 가장 가까운 도로 노드에 연결
  - 거부/비활성화: gw_<id>% 접두사 노드·엣지 is_active=FALSE 설정
- PostGIS BBox 쿼리 (LoadGraphContext): ST_MakeEnvelope + ST_Within 으로 활성 노드·엣지 조회
- KNN 최근접 노드 (FindNearestNodeID): ST_DWithin + <-> 연산자 (GIST 인덱스 필수)
- Naver API (NaverSearchClient.Search): KATECH 좌표(×10^7) → WGS-84 변환, 30 개 조회 후 서버사이드 5 개로 재정렬

3.4 ETL 도메인

외부 안전 데이터를 수집·처리하고, 사용자 피드백으로 엣지 가중치를 조정합니다. X-Internal-Token 또는 관리자 JWT로 보호됩니다.

ETL 파이프라인 단계

1. 계단 GeoJSON 파싱 (ParseStairsGeoJSON): OSM 내보내기 → weight=0.2 (난간·조명 없으면 0.1)
2. 가로등 CSV 파싱 (ParseStreetlightsCSV): data.go.kr → weight=min(lux/1000, 1.0)

3. CCTV CSV 파싱 (ParseCCTVsCSV): 보안등 CSV 에서 'cctv'/'카메라' 필터 → weight=0.8(고정)
4. 경사 JSON 파싱 (ParseSlopeJSON): weight=max(1.0−slope_deg/30.0, 0.1) (0° → 1.0, ≥30° → 0.1)
5. 피드백 적용 (ApplyRouteFeedback): 평점 1→−0.15 / 2→−0.10 / 3→0.0 / 4→+0.05 / 5→+0.10 델타 맵
6. GPX 분석 (MapMatchGPX): trkpt 파싱 → haversine 거리 → 100m 초과 세그먼트를 신규 엣지 후보로 플래그

Collector 인터페이스

```
// etl/collector.go
type Collector interface {
    FetchSafetyData(regionCode string, ...) ([]RawData, error)
    MapMatchGPX(gpxBytes []byte)  ([]NewEdgeCandidate, float64, error)
}
```

FileCollector(운영) ↔ MockCollector(개발) 교체만으로 실제/모의 데이터 전환 가능합니다.

4. API 명세서 및 예시

Base URL: `http://localhost:8081/api/v1` | **Swagger UI:** `http://localhost:8081/swagger/index.html`

인증이 필요한 엔드포인트는 Authorization: Bearer <access_token> 헤더를 포함해야 합니다.

4.1 Users — /api/v1/users

POST /api/v1/users/signup

인증: 없음 (공개)

신규 계정 생성. admin_code가 ADMIN_SIGNUP_CODE 환경변수와 일치하면 admin 역할 부여. 이메일 중복 시 409.

요청 본문:

```
{ "email": "user@example.com", "password": "Str0ng!Pass",
  "user_type": "standard", "admin_code": "" }
```

응답 201:

```
{ "user": { "id": "a1b2c3d4-...", "email": "user@example.com",
            "user_type": "standard", "role": "user", "is_active": true },
  "tokens": { "access_token": "eyJ...", "refresh_token": "eyJ...",
              "expires_at": "2026-04-01T09:15:00Z" } }
```

cURL:

```
curl -X POST http://localhost:8081/api/v1/users/signup \
  -H "Content-Type: application/json" \
  -d '{"email":"user@example.com","password":"Str0ng!Pass","user_type":"standard"}'
```

POST /api/v1/users/login

인증: 없음 (공개)

타이밍 어택 방어 포함. 오류 시 의도적으로 401 Generic 메시지 반환.

요청 본문:

```
{ "email": "user@example.com", "password": "Str0ng!Pass" }
```

응답 200:

```
{ "access_token": "eyJ...", "refresh_token": "eyJ...", "expires_at": "..." }
```

cURL:

```
curl -X POST http://localhost:8081/api/v1/users/login \
  -H "Content-Type: application/json" -d
'{"email":"user@example.com","password":"Str0ng!Pass"}'
```

POST /api/v1/users/refresh

인증: 없음 (Refresh Token 본문)

토큰 로테이션: 사용된 Refresh Token은 즉시 폐기, 재사용 감지 시 전체 세션 폐기.

```
curl -X POST http://localhost:8081/api/v1/users/refresh \
```

```
  -H "Content-Type: application/json" -d '{"refresh_token":"eyJ..."}'
```

GET /api/v1/users/me

인증: Bearer (모든 사용자)

현재 인증 사용자 프로필 + 선호도 조회.

응답 200:

```
{ "user": { "id": "a1b2c3d4-...", "email": "user@example.com",
            "user_type": "wheelchair", "role": "user" },
  "preferences": { "stair_avoidance": 3, "slope_avoidance": 2,
                   "safety_priority": 1, "main_road_prefer": 0 } }
curl http://localhost:8081/api/v1/users/me -H "Authorization: Bearer eyJ..."
```

PATCH /api/v1/users/me/preferences

인증: Bearer (모든 사용자)

선호도 부분 업데이트. 전송하지 않은 필드는 기존 값 보존. 유효 범위: 0~3.

```
curl -X PATCH http://localhost:8081/api/v1/users/me/preferences \
  -H "Authorization: Bearer eyJ..." -H "Content-Type: application/json" \
  -d '{"stair_avoidance":3,"safety_priority":2}'
```

DELETE /api/v1/users/me

인증: Bearer (모든 사용자)

소프트 삭제: PII 익명화 + 모든 Refresh Token 폐기 + 계정 비활성화. 응답: 204 No Content.

```
curl -X DELETE http://localhost:8081/api/v1/users/me -H "Authorization: Bearer eyJ..."
```

4.2 Navigation — /api/v1/navigation

POST /api/v1/navigation/route

인증: 없음 (공개)

핵심 라우팅 엔드포인트. 최단·균형·안전·커스텀 4가지 경로 동시 반환. 좌표는 한국 BBox(lat 33~43, lng 124~132) 내부여야 합니다.

요청 본문:

```
{ "origin":      { "lat": 37.5563, "lng": 126.9236 },
  "destination": { "lat": 37.5602, "lng": 126.9280 },
  "base_weights":   { "stair_avoidance":3, "slope_avoidance":2, "safety_priority":1,
"main_road_prefer":0 },
  "custom_weights": { "stair_avoidance":0, "slope_avoidance":0, "safety_priority":3,
"main_road_prefer":1 } }
```

응답 200:

```
{ "shortest": { "path": [{"lat":37.5563,"lng":126.9236}, ...],
                "total_distance_meters": 460.0, "detour_index": 1.12,
                "summary": "총 거리: 460m | 우회율: 1.12x | 경유 지점: 1곳" },
  "balanced": { ... },
```

```
  "safe":     { ... },
  "custom":   { ... } }
```

cURL:

```
curl -X POST http://localhost:8081/api/v1/navigation/route \
  -H "Content-Type: application/json" \
  -d
'{"origin":{"lat":37.5563,"lng":126.9236},"destination":{"lat":37.5602,"lng":126.9280},

"base_weights":{"stair_avoidance":3,"slope_avoidance":2,"safety_priority":1,"main_road_pr
efer":0},

"custom_weights":{"stair_avoidance":0,"slope_avoidance":0,"safety_priority":3,"main_road_
prefer":1}}'
```

GET /api/v1/navigation/detour-index

인증: 없음 (공개)

실제 보행 최단 경로 거리 vs 직선 거리 비율. index > 1.0 = 우회 발생.

```
curl "http://localhost:8081/api/v1/navigation/detour-
index?origin_lat=37.5563&origin_lng=126.9236&dest_lat=37.5602&dest_lng=126.9280&type=safe
"
```

응답 예시:

```
{ "route_type": "safe", "detour_index": 1.35, "explanation": "직선 거리 대비 35% 우회" }
```

GET /api/v1/navigation/edges/:edge_id/safety-score

인증: 없음 (공개)

엣지 안전 점수 조회. 등급 기준: A≥0.8, B≥0.6, C≥0.4, D<0.4. V2.3.1에서는 페이크 그래프 폴백 사용.

```
curl http://localhost:8081/api/v1/navigation/edges/e1/safety-score
# 응답:
{ "edge_id":"e1","safety_score":0.85,"light_density":0.9,"cctv_density":0.8,"grade":"A" }
```

4.3 Spatial — /api/v1/spatial

GET /api/v1/spatial/graph-context

인증: 없음 (공개)

BBox 내 노드·엣지 그래프 컨텍스트 조회. Navigation 서비스가 내부적으로 호출합니다.

```
curl "http://localhost:8081/api/v1/spatial/graph-
context?min_lat=37.50&min_lng=126.90&max_lat=37.60&max_lng=127.00"
```

GET /api/v1/spatial/search

인증: 없음 (공개)

Naver Local Search API 프록시. 키워드로 장소 검색, WGS-84 좌표 변환 포함.

```
curl "http://localhost:8081/api/v1/spatial/search?query=홍대입구역"
# 응답: { "query":"홍대입구역", "count":5, "results":[{ "title":"홍대입구역 2호선",
#          "lat":37.5563, "lng":126.9236, "category":"교통 > 지하철역" }] }
```

POST /api/v1/spatial/gateways

인증: Bearer (모든 사용자)

지름길(쪽문·출입구) 제보. 초기 상태는 pending, 관리자 승인 후 라우팅 그래프에 반영.

```
curl -X POST http://localhost:8081/api/v1/spatial/gateways \
  -H "Authorization: Bearer eyJ..." -H "Content-Type: application/json" \
  -d '{"lat":37.5595,"lng":126.9268,"type":"shortcut",
       "properties":{"description":"아파트 쪽문
","path_coords":[{"lat":37.5594,"lng":126.9267},{"lat":37.5596,"lng":126.9269}]}}'
```

PATCH /api/v1/spatial/gateways/:gateway_id/status

인증: Bearer + Admin 역할

지름길 승인/거부. approved 시 ApplyGatewayGraph(active=true)로 라우팅 그래프 주입. rejected 시 비활성화.

```
curl -X PATCH http://localhost:8081/api/v1/spatial/gateways/<id>/status \
  -H "Authorization: Bearer <admin-token>" -H "Content-Type: application/json" \
  -d '{"status":"approved","reason":"현장 확인 완료"}'
```

GET /api/v1/spatial/gateways/pending

인증: Bearer + Admin 역할

승인 대기 중인 지름길 목록 조회.

```
curl http://localhost:8081/api/v1/spatial/gateways/pending -H "Authorization: Bearer
<admin-token>"
```

GET /api/v1/spatial/areas/:area_id/visibility-graph

인증: 없음 (공개)

공원 등 구역 내 진입점→출입점 경로. 장애물 없으면 직선, 있으면 회피 경유점 반환.

```
curl "http://localhost:8081/api/v1/spatial/areas/park_001/visibility-
graph?entry_lat=37.5580&entry_lng=126.9240&exit_lat=37.5588&exit_lng=126.9255"
```

GET /api/v1/spatial/safety-map

인증: Bearer + Admin 역할

관리자용 안전 지도 데이터 (CCTV·보안등·계단 엣지·경사 엣지·경사 샘플) BBox 조회.

```
curl "http://localhost:8081/api/v1/spatial/safety-
map?min_lat=37.42&min_lng=126.76&max_lat=37.70&max_lng=127.22" \
  -H "Authorization: Bearer <admin-token>"
```

그래프 수동 관리 (관리자)

메서드	경로	설명
POST	/api/v1/spatial/graph/nodes	노드 수동 추가 ({lat, lng, type})
DELETE	/api/v1/spatial/graph/nodes/:node_id	노드 삭제 (연결된 모든 엣지도 비활성화)
POST	/api/v1/spatial/graph/edges	엣지 수동 추가 ({from_id, to_id})
DELETE	/api/v1/spatial/graph/edges/:edge_id	엣지 삭제

4.4 ETL — /api/v1/internal/etl

PUT /api/v1/internal/etl/safety-data/:region_code

인증: Bearer + Admin 역할

지역 안전 데이터 수집 트리거. 계단·가로등·CCTV·경사 파일 경로를 쿼리 파라미터로 전달.

```
curl -X PUT "http://localhost:8081/api/v1/internal/etl/safety-
data/seoul?stairs_geojson=/data/stairs.geojson&streetlight_csv=/data/lights.csv" \
  -H "Authorization: Bearer <admin-token>"
# 응답:
{ "status":"success","region_code":"seoul","records_updated":1523,"records_failed":2 }
```

GET /api/v1/internal/etl/logs

인증: Bearer + Admin 역할

최근 ETL 로그 조회.

```
curl http://localhost:8081/api/v1/internal/etl/logs -H "Authorization: Bearer <admin-
token>"
```

POST /api/v1/internal/etl/run

인증: Bearer + Admin 역할

ETL 프로세스 전체 실행.

DELETE /api/v1/internal/etl/safety-data/outdated

인증: Bearer + Admin 역할

지정 일수(days, 기본 180일) 이상 된 안전 데이터 파기. V2.3.1에서는 로그만 기록, 실제 SQL 삭제는 TODO.

```
curl -X DELETE "http://localhost:8081/api/v1/internal/etl/safety-data/outdated?days=90" \
  -H "Authorization: Bearer <admin-token>"
```

POST /api/v1/spatial/gpx/upload

인증: Bearer (모든 사용자)

GPX 트랙 파일 업로드 (최대 10MB). haversine 거리 계산 및 신규 엣지 후보 탐지.

```
curl -X POST http://localhost:8081/api/v1/spatial/gpx/upload \
  -H "Authorization: Bearer eyJ..." -F "file=@track.gpx"
# 응답: { "status":"analyzed","new_edges_discovered":3,"matched_distance_m":1240.50 }
```

POST /api/v1/navigation/routes/:route_id/feedback

인증: Bearer (모든 사용자)

경로 평가 제출. issue_tags: unmapped_stairs | dark_road | construction | dangerous.

```
curl -X POST http://localhost:8081/api/v1/navigation/routes/route_123/feedback \
  -H "Authorization: Bearer eyJ..." -H "Content-Type: application/json" \
  -d '{"rating":4,"issue_tags":["dark_road"],"comment":"가로등이 부족합니다"}'
```

5. 주요 코드 포인트

신규 개발자가 반드시 이해해야 할 핵심 코드 위치와 그 이유입니다.

파일 + 함수/구조체	중요한 이유
`main.go` (전체)	모든 도메인 DI 연결이 여기서 명시적으로 이루어짐. 의존성 그래프의 단일 진실 소스.
`core/jwt.go : JWTManager`	RS256(액세스) + HS256(리프레시) 토큰 생명주기 관리. 개발 시 RSA 키 자동 생성 — 운영에서는 반드시 영속 키로 교체.
`core/middleware.go : AuthMiddleware()`	모든 인증 엔드포인트가 이 함수를 통과함. Bearer 토큰 추출 → RS256 검증 → JWTClaims Gin 컨텍스트 주입.
`core/middleware.go : AdminMiddleware()`	AuthMiddleware 이후 체이닝 필수. claims.Role == 'admin' 확인, 불일치 시 403.
`navigation/engine.go : applyWeightsToEdge()`	핵심 비용 공식. 라우팅 동작(페널티 가중치, 새 피처) 변경은 여기서 시작.
`navigation/engine.go : dijkstra()`	container/heap 기반 min-heap. 가중치 비용과 실제 거리 동시 추적. 목적지 팝 시 조기 종료.
`navigation/engine.go : computeAllEdgeCosts()`	고루틴 팬아웃으로 엣지 비용 병렬 계산. cloneGraph()로 경로 변형 간 데이터 경쟁 방지.
`navigation/service.go : GetRoutes()`	메인 라우팅 진입점. BBox 재시도(×1/×2/×4 패딩), 4개 병렬 경로 고루틴, 개발 모드 페이크 그래프 폴백.
`navigation/service.go : mapToConnectedNodes()`	좌표→노드 스냅. 양 끝점 각 8개 후보, BFS 연결성 검사로 첫 번째 연결 쌍 선택. 라우팅 성공률의 핵심.
`spatial/repository.go : LoadGraphContext()`	PostGIS BBox 쿼리로 DB 행을 인메모리 navigation.Graph로 변환. 영속 저장소 ↔ 라우팅 엔진의 가교.
`spatial/repository.go : FindNearestNodeID()`	PostGIS <-> 연산자 KNN 쿼리. idx_nodes_geom GIST 인덱스 필수. 없으면 전체 스캔으로 성능 급락.
`spatial/repository.go : ApplyGatewayGraph()`	관리자 승인 시 지름길 → 그래프 주입을 트랜잭션으로 원자적 실행.
`spatial/service.go : NaverSearchClient.Search()`	Naver Local Search 외부 API 호출. KATECH 좌표 변환 처리. NAVER_CLIENT_ID/SECRET 필수.

`users/service.go : Login()`	사용자 미발견 시 더미 bcrypt 비교 실행 → 타이밍 어택 방어.
`users/service.go : RefreshTokens()`	토큰 로테이션 핵심. 재사용 감지 시 해당 유저 전체 세션 일괄 폐기.
`users/repository.go : SoftDeleteUser()`	GDPR 준수 PII 삭제. 트랜잭션 내 이메일 익명화 + 해시 삭제 + 토큰 폐기.
`etl/collector.go : FileCollector.FetchSafetyData()`	운영 ETL 파이프라인 진입점. OSM 계단 GeoJSON, 가로등/CCTV CSV, 경사 JSON 순차 처리.
`cmd/seed/main.go`	2패스 OSM 시더: 1패스 노드 로드 → 2패스 엣지 로드 (FK 위반 방지). OSM_FILE 환경변수 사용.
`db/schema.sql`	PostGIS 확장, 모든 테이블, 인덱스, updated_at 트리거 포함 완전한 DB 스키마. 컨테이너 시작 시 자동 적용.

6. 소프트웨어 자재 명세서 (SBOM)

6.1 Go 직접 의존성 (go.mod)

모듈	버전	용도
github.com/gin-gonic/gin	v1.12.0	HTTP 프레임워크
github.com/gin-contrib/cors	v1.7.7	CORS 미들웨어
github.com/jmoiron/sqlx	v1.4.0	SQL 확장 (named query, struct scan)
github.com/lib/pq	v1.12.0	PostgreSQL 드라이버
github.com/lestrrat-go/jwx/v2	v2.1.6	JWT 인코딩/디코딩 (RS256 + HS256)
golang.org/x/crypto	v0.49.0	bcrypt 비밀번호 해싱
github.com/swaggo/gin-swagger	v1.6.1	Swagger UI 통합
github.com/swaggo/swag	v1.16.6	Swagger 코드 생성
github.com/swaggo/files	v1.0.1	Swagger 정적 파일

주요 간접 의존성

모듈	버전	비고
github.com/bytedance/sonic	v1.15.0	고속 JSON (gin 의존성)
github.com/go-playground/validator/v10	v10.30.1	요청 바인딩 유효성 검사
github.com/joho/godotenv	v1.5.1	.env 파일 로더

6.2 프론트엔드 의존성 (package.json)

런타임 의존성

패키지	버전	용도
react / react-dom	^19.2.4	UI 프레임워크

react-router-dom	^7.13.2	클라이언트 라우팅
axios	^1.13.6	HTTP 클라이언트
tailwindcss	^4.2.2	유틸리티 CSS
leaflet	^1.9.4	관리자 페이지 지도
@radix-ui/* (dialog, dropdown, select, slider 등)	다수	접근성 기반 UI 컴포넌트 (shadcn/ui)
lucide-react	^1.7.0	아이콘
class-variance-authority / clsx / tailwind-merge	최신	CSS 클래스 유틸리티

개발 의존성

패키지	버전	용도
vite / @vitejs/plugin-react	^8.0.1	빌드 도구 및 개발 서버
typescript	~5.9.3	TypeScript 컴파일러
eslint / eslint-plugin-react-hooks	^9.39.4	코드 품질 검사
@types/react / @types/leaflet	최신	TypeScript 타입 정의

6.3 외부 서비스 및 API

서비스	용도	인증 방식
PostgreSQL 16 + PostGIS 3.4	주 데이터 저장소; GIST 공간 인덱스	DATABASE_URL 연결 문자열
Naver Local Search API	장소/POI 검색 (/spatial/search)	X-Naver-Client-Id + X-Naver-Client-Secret
Naver Maps Web SDK	클라이언트 지도 렌더링 (index.html 삽입)	VITE_NAVER_MAP_CLIENT_ID 스크립트 파라미터
OpenStreetMap	보행자 도로 그래프 원본 데이터 (cmd/seed 처리)	공개 데이터 (인증 불필요)

6.4 SBOM 취약점 분석 결과

syft v1.42.3 + grype v0.110.0 사용, 2026-04-03 기준 분석. SPDX 2.3 JSON 포맷.

패키지 인벤토리

구성요소	탐지 패키지 수	취약점 수
Backend (last_mile_backend)	80개	1건 (LOW)
Frontend (last_mile_frontend)	164개	0건 — 클린
합계	244개	1건

취약점 상세 (전체 1 건)

패키지	설치 버전	수정 버전	취약점 ID	심각도 / 비고
`filippo.io/edwards25519`	v1.1.0	**v1.1.1**	GHSA-fw7p-63qq-7hpr	LOW / 간접 의존성

해당 패키지는 go.mod 직접 의존성이 아닌 lestrrat-go/jwx/v2(JWT 라이브러리)를 통한 간접 의존성입니다. EPSS 0.1% 미만으로 실제 악용 가능성이 매우 낮습니다.

조치 사항

- go get filippo.io/edwards25519@v1.1.1 후 go mod tidy 실행으로 해결 가능
- 또는 lestrrat-go/jwx/v2 최신 버전 업데이트 시 자동 해결
- CRITICAL·HIGH 취약점 0 건 — 직접 의존성 전체 안전
- 정기 재스캔 권장: syft + grype 를 CI/CD 파이프라인에 통합하여 주간 자동 실행

7. 환경 변수 및 설정

7.1 백엔드 (.env)

변수명	필수	예시 값	설명
DATABASE_URL	✓	postgres://user:pass@localhost:5432/lastmile?sslmode=disable	PostgreSQL DSN
JWT_REFRESH_SECRET	✓	my-super-secret-key-at-least-32-chars	Refresh Token HMAC 서명 비밀 (최소 32바이트)
NAVER_CLIENT_ID	선택	abc123xyz	Naver Local Search API 클라이언트 ID (미설정 시 검색 비활성화)
NAVER_CLIENT_SECRET	선택	secret456	Naver Local Search API 클라이언트 시크릿
ADMIN_SIGNUP_CODE	선택	LAST_MILE_ADMIN_2026	Admin 계정 생성 초대 코드 (미설정 시 Admin 가입 불가)
OSM_FILE	시드 시	/app/scripts/seoul_all.json	cmd/seed 전용 OSM JSON 파일 경로

7.2 프론트엔드 (.env)

변수명	필수	설명 및 예시

VITE_API_URL	✓	백엔드 Base URL (후행 슬래시 없음). 예: http://localhost:8081
VITE_NAVER_MAP_CLIENT_ID	✓	Naver Maps Web JS SDK 클라이언트 ID (ncpKeyId). 미설정 시 지도 미표시.

7.3 Docker Compose (.env)

변수명	기본값	설명
POSTGRES_PASSWORD	postgres	DB 슈퍼유저 비밀번호
DB_PORT	5433	호스트 측 DB 포트 매핑
BACKEND_PORT	8081	호스트 측 백엔드 포트 매핑
FRONTEND_PORT	5173	호스트 측 프론트엔드 포트 매핑
JWT_REFRESH_SECRET	(반드시 설정)	백엔드 컨테이너로 전달

⚠ .env 파일에서 값에 $ 문자가 포함된 경우 $$로 이스케이프해야 합니다.

8. 데이터베이스 스키마

모든 테이블은 last_mile_backend/db/schema.sql에 정의되어 있습니다. geometry 컬럼은 lat/lng에서 자동 생성(STORED)됩니다.

8.1 nodes

컬럼	타입	제약조건 / 설명
id	TEXT	PRIMARY KEY
lat / lng	DOUBLE PRECISION	NOT NULL; WGS-84 좌표
type	TEXT	NOT NULL; street | gateway | park_entry | park_exit
geom	GEOMETRY(Point,4326)	GENERATED STORED from ST_SetSRID(ST_MakePoint(lng,lat),4326)
is_active	BOOLEAN	DEFAULT TRUE; 인덱스: idx_nodes_geom (GIST on geom)

8.2 edges

컬럼	타입	제약조건 / 설명
id	TEXT	PRIMARY KEY
from_node_id / to_node_id	TEXT	FK → nodes(id); idx_edges_from_node, idx_edges_to_node (partial, is_active=TRUE)
distance_meters	DOUBLE PRECISION	NOT NULL; Haversine 계산값
has_stair	BOOLEAN	DEFAULT FALSE
slope_grade	DOUBLE PRECISION	DEFAULT 0.0; ETL에서 slope_samples 기반 업데이트
light_density	DOUBLE PRECISION	DEFAULT 0.5; ETL에서 security_lights 기반 업데이트
cctv_density	DOUBLE PRECISION	DEFAULT 0.5; ETL에서 cctv_points 기반 업데이트
is_main_road / is_in_park / is_active	BOOLEAN	DEFAULT FALSE/FALSE/TRUE

8.3 users

컬럼	타입	제약조건 / 설명
id	UUID	PRIMARY KEY DEFAULT gen_random_uuid()
email	TEXT	UNIQUE NOT NULL; idx_users_email (partial, deleted_at IS NULL)
password_hash	TEXT	NOT NULL; bcrypt cost 12
user_type	TEXT	NOT NULL DEFAULT 'standard'
role	TEXT	NOT NULL DEFAULT 'user'
is_active / deleted_at	BOOL/TIMESTAMPTZ	소프트 삭제; trg_users_updated_at 트리거로 updated_at 자동 갱신

8.4 user_preferences

컬럼	타입	제약조건 / 설명
user_id	UUID	PRIMARY KEY FK → users(id) ON DELETE CASCADE
stair_avoidance / slope_avoidance / safety_priority / main_road_prefer	SMALLINT	DEFAULT 0; CHECK 0~3

8.5 refresh_tokens

컬럼	타입	제약조건 / 설명
id	UUID	PRIMARY KEY DEFAULT gen_random_uuid()
user_id	UUID	NOT NULL FK → users(id) ON DELETE CASCADE; idx_refresh_user_id
token_hash	TEXT	NOT NULL; SHA-256 hex; UNIQUE 인덱스: idx_refresh_token_hash
expires_at / revoked	TIMESTAMPTZ/BOOL	NOT NULL / DEFAULT FALSE

8.6 gateways / gateway_events

컬럼 (gateways)	타입	제약조건 / 설명
id	UUID	PK; idx_gateways_geom (GIST)
lat / lng / type	DOUBLE/TEXT	status: pending|approved|rejected

/ status		
properties	JSONB	path_coords 배열 포함 가능
reported_by / reviewed_by	UUID	FK → users(id)

8.7 안전 데이터 테이블

테이블	주요 컬럼 / 특이사항
cctv_points	BIGSERIAL PK, purpose, address, lat/lng, geom GENERATED STORED, source, data_date; idx_cctv_points_geom (GIST)
security_lights	BIGSERIAL PK, address, install_type, mgmt_org, lat/lng, geom GENERATED STORED; idx_security_lights_geom (GIST)
slope_samples	lat, lng, slope_deg, geom GENERATED STORED; idx_slope_samples_geom (GIST)
etl_logs	UUID PK, job_name, status (success|failed|in_progress), message, created_at
areas	TEXT PK, name, boundary GEOMETRY(Polygon,4326), obstacles GEOMETRY(MultiPolygon,4326)

8.8 테이블 관계도

```
users (1) ──────────────── (N) refresh_tokens
users (1) ──────────────── (1) user_preferences
users (1) ──────────────── (N) gateways [reported_by]
users (1) ──────────────── (N) gateways [reviewed_by]
gateways (1) ──────────────── (N) gateway_events
nodes (1) ──────────────── (N) edges [from_node_id]
nodes (1) ──────────────── (N) edges [to_node_id]
cctv_points → (SQL script) → edges.cctv_density
security_lights → (SQL script) → edges.light_density
slope_samples → (SQL script) → edges.slope_grade
```

9. CI/CD 및 빌드

9.1 Docker Compose (권장 방법)

docker-compose.yml은 4개 서비스를 정의합니다:

서비스	이미지/Dockerfile	역할
db	postgis/postgis:16-3.4	PostgreSQL+PostGIS; schema.sql 자동 적용 (첫 시작 시)
backend	last_mile_backend/Dockerfile	Go 멀티스테이지 빌드; 내부 8080 → 호스트 BACKEND_PORT(8081)
frontend	last_mile_frontend/Dockerfile	Vite 빌드 서버; FRONTEND_PORT(5173)
seed	last_mile_backend/Dockerfile.seed	원샷 OSM 시더; profile: manual (명시적 실행 필요)

Docker 실행 순서

```
# 1. 환경 변수 설정
cp last_mile_backend/.env.example last_mile_backend/.env
# .env에 JWT_REFRESH_SECRET, NAVER_CLIENT_ID 등 입력

# 2. DB + 백엔드 + 프론트엔드 시작
docker-compose up -d db backend frontend

# 3. 시드 실행 (최초 1회 또는 DB 초기화 후)
docker-compose --profile manual up seed

# 4. Swagger UI 확인
# http://localhost:8081/swagger/index.html
```

⚠ V2.3.1에는 .github/workflows/, Jenkins 파일, Makefile이 없습니다. 공식 CI/CD 파이프라인은 미구성 상태입니다.

9.2 로컬 수동 빌드 (개발 환경)

백엔드

필수 조건: Go ≥1.21, PostgreSQL 16 + PostGIS 3.4 로컬 실행

```
cd last_mile_backend
```

```
cp .env.example .env    # 실제 값으로 편집
go mod download
go run ./main.go        # 서버 :8080에서 시작
```

프론트엔드

필수 조건: Node.js ≥18

```
cd last_mile_frontend
cp .env.example .env    # VITE_API_URL, VITE_NAVER_MAP_CLIENT_ID 설정
npm install
npm run dev             # 개발 서버 http://localhost:5173
```

9.3 데이터베이스 초기화 및 시딩

스키마 적용

```
psql postgres://user:pass@localhost:5432/lastmile -f last_mile_backend/db/schema.sql
```

시드 파이프라인 (전체)

```
cd last_mile_backend
# OSM 그래프 시드
OSM_FILE=scripts/seoul_all.json go run ./cmd/seed/

# 경사 데이터 임포트
SLOPE_JSON=scripts/slope_latlng_samples.json go run ./cmd/import_slope_json/

# 엣지 안전 데이터 보강 (CCTV·가로등 밀도 업데이트)
go run ./cmd/enrich_edges/
```

9.4 스크립트 디렉토리

스크립트	역할
scripts/run_seed_and_enrich.sh	전체 시드 파이프라인: OSM → 경사 → 엣지 보강
scripts/docker_seed_pipeline.sh	Docker 기반 시드 파이프라인
scripts/import_seoul_safety_data.sh	서울시 전체 CCTV/가로등 데이터 임포트
scripts/import_songpa_safety.sh	송파구 안전 데이터 임포트

`scripts/convert_slope_geojson_to_samples.py`	경사 GeoJSON → 플랫 샘플 JSON 변환
`scripts/sql/10_update_edge_safety_density.sql`	cctv_points/security_lights → edges.light_density/cctv_density 업데이트
`scripts/sql/20_update_edges_slope_from_samples.sql`	slope_samples → edges.slope_grade 업데이트

10. 도메인 간 연동

10.1 호출 관계

호출자	피호출자	메커니즘
NavigationService	spatial.PostgresRepository	Go 인터페이스 — navigation.SpatialRepository는 navigation/service.go에 정의, spatial.PostgresRepository가 구현. navigation 패키지는 spatial 패키지를 직접 임포트하지 않음.
SpatialService	navigation.SpatialRepository + GatewayRepository + AreaRepository	복합 인터페이스 SpatialServiceRepository (spatial/service.go 정의)가 모든 레포지터리 기능을 집계.
SpatialService	navigation.Graph, Node, Edge, EdgeFeatures	직접 구조체 임포트 — Spatial이 Navigation 타입을 레포지터리·서비스 레이어에서 사용.
ETLService	*sqlx.DB (직접)	ETL은 레포지터리 추상화 없이 sqlx를 직접 사용해 etl_logs에 기록.
모든 핸들러	core.JWTManager / AuthMiddleware / AdminMiddleware	직접 임포트 — 모든 도메인이 core/ 패키지에 의존.

10.2 Navigation 이 ETL 데이터를 사용하는 방법

Navigation은 ETL 코드를 직접 호출하지 않습니다. 파이프라인은 다음과 같이 완전히 분리되어 있습니다:

7. ETL 스크립트가 cctv_points, security_lights, slope_samples 테이블을 채웁니다.
8. SQL 보강 스크립트(scripts/sql/10_..., 20_...)가 POI 테이블 값을 edges.light_density, cctv_density, slope_grade 로 전송합니다.
9. spatial.PostgresRepository.LoadGraphContext()가 보강된 edges 테이블을 읽어 인메모리 Graph 의 EdgeFeatures 를 채웁니다.
10. Navigation 엔진이 Graph 를 통해 EdgeFeatures 를 읽고 경로 비용을 계산합니다.

결합은 완전히 분리됩니다: ETL → DB → Spatial 레포 → Navigation 엔진

10.3 Spatial 이 Navigation 에 제공하는 것

spatial.PostgresRepository는 navigation.SpatialRepository의 유일한 구현체로, NavigationService에 두 가지 메서드를 제공합니다:

- LoadGraphContext(): PostGIS BBox 쿼리로 DB 행을 인메모리 navigation.Graph 로 변환 (영속 저장소 ↔ 라우팅 엔진의 가교)
- FindNearestNodeID(): PostGIS KNN 쿼리로 좌표 → 노드 스냅 (idx_nodes_geom GIST 인덱스 필수)

두 메서드 모두 인터페이스로 접근하므로, 다른 그래프 백엔드로 교체 시 인터페이스만 구현하면 됩니다.

10.4 공유 패키지 및 유틸리티

패키지/리소스	공유하는 도메인
core/ (JWT + 미들웨어)	Users, Spatial, Navigation, ETL — 모든 핸들러
navigation 타입 (Node, Edge, Graph, EdgeFeatures)	Navigation (정의), Spatial (레포지터리·서비스에서 사용)
*sqlx.DB (PostgreSQL 연결)	users.repo, spatial.repo, etl.service — main.go에서 공유 인스턴스 주입

11. 독립적 업데이트 가능성 및 도메인 결합

11.1 도메인별 분석

Users 도메인

독립 업데이트: ✅ 가능

- users, user_preferences, refresh_tokens 테이블을 이 도메인이 단독으로 소유.
- 다른 도메인은 Users 코드를 직접 임포트하지 않고 core.JWTManager / AuthMiddleware 를 통해서만 간접 의존.
- 의존: core/ 패키지 (JWT 매니저), PostgreSQL (전용 테이블)
- 피의존: Navigation·Spatial·ETL — JWT 토큰을 통해서만 간접 의존
- 인터페이스 경계: Auth 미들웨어(core.AuthMiddleware)가 유일한 계약. Users 도메인은 다른 도메인에 Go 인터페이스를 노출하지 않음.

Navigation 도메인

독립 업데이트: ✅ 엔진 로직은 가능 / ⚠ SpatialRepository 인터페이스 변경 시 주의

- routing engine(engine.go)은 완전히 자립적 — DB 의존성 없음.
- 의존: navigation.SpatialRepository 인터페이스 (→ Spatial 구현), core/, navigation 패키지 자체 타입
- 피의존: Go 코드 레벨에서는 아무도 Navigation 을 호출하지 않음 (프론트엔드가 HTTP 로 직접 호출)
- 인터페이스 경계: navigation.SpatialRepository 가 공식 경계. 변경 시 navigation/service.go(정의)와 spatial/repository.go(구현) 양쪽 수정 필요.

Spatial 도메인

독립 업데이트: ⚠ 부분적 가능

- 지름길 CRUD, 검색, 가시성 그래프 로직은 자유롭게 변경 가능.
- LoadGraphContext(), FindNearestNodeID() SQL 변경은 Navigation 과의 인터페이스 계약에 영향.
- 의존: PostgreSQL (nodes/edges/gateways/areas/안전 POI 테이블), Naver Local Search API, navigation 패키지 타입

- 피의존: NavigationService (SpatialRepository 인터페이스로). main.go 가 spatialRepo 를 NavigationService 와 SpatialService 양쪽에 주입.
- 인터페이스 경계: navigation.SpatialRepository + spatial.SpatialServiceRepository (복합). navigation 구조체 타입 임포트는 인터페이스 이상의 결합.

ETL 도메인

독립 업데이트: ✅ 가능

- 다른 도메인과 Go 인터페이스를 공유하지 않음.
- 의존: core/ (auth 미들웨어), *sqlx.DB (직접), Collector 인터페이스, 파일 시스템
- 피의존: Navigation 이 SQL 스크립트로 보강된 edges 테이블을 간접 소비 (Go 호출 없음)
- 인터페이스 경계: Collector 인터페이스 (FileCollector ↔ MockCollector 교체). 나머지는 DB 직접 접근.

11.2 의존성 매트릭스

도메인	의존 대상	의존받는 대상	독립 업데이트 가능?
Users	core/ (JWT), PostgreSQL (전용 테이블)	Nav·Spatial·ETL (JWT 미들웨어 통해서만, 코드 직접 임포트 없음)	✅ **예 — 테이블 단독 소유**
Navigation	SpatialRepository 인터페이스 (→ Spatial 구현), core/	없음 (HTTP 호출만)	✅ **엔진 로직 / ⚠ 인터페이스 변경 시**
Spatial	PostgreSQL (다수 테이블), Naver API, navigation 타입	NavigationService (SpatialRepository 인터페이스로)	⚠ **부분적 — 그래프 쿼리 변경 시 Navigation 영향**
ETL	core/ (auth), *sqlx.DB (직접), Collector 인터페이스	Navigation (SQL 보강된 DB 간접 소비, Go 호출 없음)	✅ **예 — Go 레벨 완전 분리**

11.3 핵심 결합 지점 요약

새 개발자가 알아야 할 가장 중요한 결합 지점:

- main.go: 유일한 DI 연결 지점. spatialRepo 가 NavigationService 와 SpatialService 양쪽에 주입됨.
- navigation.SpatialRepository 인터페이스: Navigation ↔ Spatial 간의 공식 계약. 변경 시 양쪽 수정 필요.
- navigation 패키지 타입 (Node, Edge, Graph): Spatial 레포지터리가 직접 임포트. 타입 변경 시 Spatial 도 영향.
- SQL 보강 스크립트: ETL 출력(cctv_points, security_lights, slope_samples) → Navigation 입력(edges) 변환의 물리적 경계.

12. 모니터링 현황

V2.3.1 기준 전용 모니터링 인프라는 구성되어 있지 않습니다. 현재 관측 가능한 수단과 향후 도입 권장 방안을 기술합니다.

12.1 현재 구성된 관측 수단

Gin 기본 로그

gin.Default() 미들웨어가 모든 HTTP 요청을 표준 출력(stdout)에 기록합니다. Docker 환경에서는 docker logs lastmile-backend로 확인할 수 있습니다.

```
docker logs lastmile-backend --tail 100 -f
```

기록 항목: HTTP 메서드, 경로, 응답 코드, 응답 시간, 클라이언트 IP. 단, 구조화된 JSON 형태가 아니므로 대량 로그 분석에는 적합하지 않습니다.

ETL 로그 테이블

ETL 작업 결과는 etl_logs 테이블에 기록됩니다. GET /api/v1/internal/etl/logs 엔드포인트로 최근 작업 상태를 확인할 수 있습니다.

```
curl http://localhost:8081/api/v1/internal/etl/logs -H "Authorization: Bearer <admin-token>"
```

SBOM 취약점 스캔 (수동)

syft + grype를 사용한 의존성 취약점 분석은 현재 수동으로만 실행됩니다. 자동화된 주기적 스캔은 미구성 상태입니다.

12.2 현재 부재한 관측 수단

항목	현황
헬스체크 엔드포인트	GET /health 미구현 — 컨테이너 정상 여부를 외부에서 확인할 방법 없음
구조화 로그 (JSON)	미구성 — 현재 Gin 텍스트 로그만 출력, ELK·Loki 등과 연동 불가
메트릭 수집	미구성 — Prometheus 미들웨어 미적용, CPU·메모리·응답 지연 추적 불가

분산 트레이싱	미구성 — 도메인 간 요청 흐름 추적 수단 없음
알림·경보	미구성 — ETL 실패·DB 연결 오류 발생 시 자동 알림 없음
SBOM 자동 스캔	미구성 — CI/CD 파이프라인 자체가 없으므로 주기적 취약점 스캔 불가

12.3 향후 도입 권장 방안

현재 V2.3.1은 개발·데모 단계의 완성도를 갖추고 있으며, 운영 환경 투입을 위해서는 위의 관측 인프라 구축이 선행되어야 합니다.

단기 (즉시 적용 가능)

- 헬스체크 엔드포인트 추가: GET /health → DB ping 결과 반환 (Gin 라우터 1 줄 추가)
- 구조화 로그: zap 또는 zerolog 도입으로 JSON 로그 전환 — ELK·Grafana Loki 연동 준비
- SBOM 주기 스캔: GitHub Actions 워크플로 추가, 주 1 회 syft + grype 자동 실행

중기 (운영 단계 진입 시)

- Prometheus + Grafana: gin-prometheus 미들웨어로 HTTP 메트릭 수집, 대시보드 구성
- ETL 실패 알림: etl_logs.status == 'failed' 조건 감지 시 Slack·이메일 웹훅 발송
- DB 연결 모니터링: PostgreSQL exporter 로 커넥션 풀·쿼리 성능 추적

Last Mile — 서비스 개발 WBS | 2025.03.25 ~ 04.03 | 기획·설계·개발·마무리 전 단계

No	도메인	담당자	작업 항목	세부 내용	우선순위	상태	비고/출처	2026.3.25	2026.3.26	2026.3.27	2026.3.28	2026.3.29	2026.3.30	2026.3.31	2026.4.1	2026.4.2	2026.4.3
1	기획 · 요구사항	전원	문제 정의 & 서비스 방향 설정	기존 네이버/카카오맵 한계 분석	높음	FIN	latmile 제안서	•	•								
2				Last Mile 핵심 가치 정의	높음	FIN		•	•								
3			타겟 사용자 정의	주요 페르소나 설정 (야간 보행자, 계단 회피 사용자, 빠른 출근자, 도보 배달기사 등)	높음	FIN		•	•								
4			기능 요구사항 도출	핵심 기능 목록 작성: 경로 탐색 3종, 커스텀 경로, 제보 기능	높음	FIN			•								
5			비기능 요구사항 정의	성능(응답속도), 보안(JWT), 확장성(PostGIS) 요구사항 정의	중간	FIN			•								
6			데이터 요구사항 분석	OSM + 공공데이터 수집 범위 결정	높음	FIN			•	•	•						
7				edges 컬럼 설계 초안	높음	FIN			•	•	•						
8	설계 · 기능명세	전원	시스템 아키텍처 설계	Handler→Service→Repository 레이어 구조, 도메인 분리 확정	높음	FIN				•	•	•					
9			DB 스키마 설계	nodes/edges/users/gateways/areas 테이블 설계, PostGIS 타입 확정	높음	FIN					•	•					
10			API 명세 작성	전 도메인 엔드포인트 정의, 요청/응답 스펙 문서화	높음	FIN						•	•				
11			화면 정의 & 사용자 흐름	5개 페이지 와이어프레임, 인증/경로/제보 흐름 정의	높음	FIN						•	•				
12			기술 스택 선정	Go/Gin, PostgreSQL/PostGIS, React 19/Vite, Naver Maps 선정 및 근거 정리	중간	FIN					•	•					
13			R&R 확정 & 킥오프	4개 도메인 담당자 확정, 개발 일정 수립, 협업 규칙 설정	높음	FIN							•	•			
14	DB			Python → Go 포팅	높음	FIN	OpenStreetMap				•						
15			OSM ETL 파이프라인 구현	scripts/seed 구현	높음	FIN					•						
16				run_seed.sh (서울 4개 구역)	높음	FIN					•						
17	ETL · 기존 데이터		OSM 피처 수집	계단 유무 태그 수집	높음	ING	OSM tags					•			•		
18		김영서	추가 계단 위치 ETL	OSM 계단 태그 이외에도 다른 계단 데이터 반영	높음	TBC											
19			CCTV 데이터 ETL	서울 열린데이터광장 방범CCTV → nodes/edges 밀도 매핑	높음	FIN	data.seoul.go.kr		•		•						
20			보안등 데이터 ETL	보안등 위치 수집 → light_density 컬럼 반영	높음	FIN	data.go.kr		•		•						
21	ETL · 보행편의/접근성		경사도 DEM 데이터 ETL	오르막 회피 정밀도 향상	중간	TBC	국토지리정보원										
22	ETL · 시간대/동적		커스텀 경로용 DB 전체 반영	edges 테이블 업데이트	높음	ING	내부 PostGIS				•					•	•
23				seed 재실행 및 검증	높음	FIN								•			
24	구현		라우팅 엔진 설계	Dijkstra 가중치 구조 설계, 3가지 프리셋 정의	높음	FIN				•	•						
24			Dijkstra 엔진 구현	in-memory 그래프 로드, 병렬 goroutine + 그래프 클론 data race 방지	높음	FIN					•						
25	Backend (Navigation)	곽민석	공원 가시성 그래프	park polygon → 진입/출구 Euclidean 거리 처리	중간	ING								•	•	•	
26			프리셋 3종 하드코딩	최단·균형·안심 경로 가중치 고정값 설정 및 검증	높음	FIN					•	•	•	•			
27			커스텀 경로 API 분리	user_preferences 기반 동적 가중치 엔드포인트 신설 (/route/custom)	높음	ING							•	•	•	•	
28			통합 테스트 & QA	전 도메인 API 연동 테스트, 엣지 케이스 검증	높음	ING							•	•	•	•	
29			Naver Maps API 연동	Naver Cloud Maps api 발급	높음	FIN	네이버 클라우드 플랫폼			•							
30				장소 검색 핸들러 및 서비스 구현	높음	FIN	네이버 클라우드 플랫폼			•	•	•					
31	Backend (Spatial)	이준혁	PostGIS 공간 쿼리 최적화	반경 검색 가능화	중간	FIN					•	•					
32				ST_DWithin 인덱스 튜닝	중간	FIN	\repository.go				•	•					
33			검색 정확도 버그 수정	sort=comment→random 수정	중간	FIN	\service.go								•		
34				Sscanf 에러 처리 추가	중간	FIN	\service.go								•		
35			JWT 인증 인프라 구축	RS256 JWT Manager, 토큰 로테이션, Timing Attack 방어	높음	FIN									•	•	
36	Backend (Users)	남진우	회원가입·로그인 API	bcrypt 해시, refresh token 발급·검증	높음	FIN					•	•	•				
37			사용자 선호 설정 API	4개 파라미터(0-3) CRUD, user_preferences 테이블	높음	FIN					•	•	•				
38			제보 기능 설계	gateways 테이블 설계, pending→approved 워크플로	높음	FIN					•	•	•				
39			제보 등록 API	사용자 지름길·쪽문 위치 제보 엔드포인트 구현	높음	ING										•	
40			Admin 승인 API	제보 목록 조회, 승인/반려 처리 엔드포인트	높음	TBC											
41	Frontend / Users		프로젝트 초기 세팅	React 19 + Vite + TypeScript + Tailwind 환경 구성	높음	FIN									•		
42			인증 흐름 구현	LoginPage, AuthContext, JWT auto-refresh 401 인터셉터	높음	FIN										•	
43		남진우	Naver Maps 연동	SDK CDN 로드, naver-maps.d.ts 타입 정의	높음	FIN					•						
44			경로 탐색 UI	RoutePage, 출발·도착 검색창, 경로 결과 3종 탭 표시	높음	FIN					•	•					
45			커스텀 경로 UI 추가	슬라이더 4종(계단·오르막·안심·대로) + 커스텀 탭 신설	높음	ING										•	
46			제보 UI 구현	GatewayPage: 지도 핀 등록, 제보 폼 전송	중간	TBC											
47	테스트	전석남진우	도메인 간 API 연동 검증	프론트↔백엔드 전 엔드포인트 smoke test	높음	ING										•	
48	통합 / QA	전원	기능별 사용자 시나리오 검증	경로 탐색 3종, 커스텀 경로, 제보 흐름 end-to-end 테스트	높음	ING										•	
49	마무리		API 문서 최종화	전 엔드포인트 요청/응답 스펙 최종 정리	높음	ING										•	
50				README 업데이트	높음	ING										•	
51	문서화	이준혁		시스템 구조도 작성	높음	ING										•	
52			아키텍처 문서 작성	DB ERD 작성	중간	ING										•	
53				라우팅 알고리즘 흐름도 작성	중간	ING										•	
54		전원	발표 자료 (PPT) 작성	서비스 소개, 차별점, 기술 스택, 데모 시나리오, 향후 로드맵	높음	ING										•	
55	발표	이준혁	데모 시나리오 준비	발표용 시연 경로 선정, 커스텀 경로 비교 데모 스크립트 작성	높음	TBC											
56		전원	발표 리허설	데모 흐름 최종 확인, 질의응답 대비, 발표자(김영서) 시연 연습	높음	TBC											
57		전원	성과 정리 & 회고	1주 개발 성과 정리, 잘된 점/아쉬운 점, 향후 확장 방향 문서화	중간	TBC											

범례: FIN | ING | TBC | 확장 | • 작업일 | • 확장예정 | 주말