Fingoo API Architecture Overview (Backend)

1) 한 줄 요약

Fingoo API는 **NestJS** 기반의 모듈형 백엔드이며, 도메인별로 기능을 분리하고 **CQRS(Command/Query) + Ports/Adapters(**클린/헥사고날 스타일**)** 로 요청을 처리한다.

---

2) 런타임 구성 **(System / Container View)**

API Server (NestJS)

- Entry: `src/main.ts`
- Swagger: `GET /api`
- Listen: `8000`
- Global:
    - ValidationPipe(whitelist/transform/forbidNonWhitelisted)
    - CORS origin allowlist
    - Global HttpExceptionFilter

Infra Dependencies

- **PostgreSQL (TypeORM)**
    - 설정: `src/config/typeorm.config.ts`
- **Redis (ioredis)**
    - 설정: `src/config/redis.config.ts`

- **Supabase**
    - 공통 모듈: `src/commons/supabase.module.ts`
    - 연결/스토리지: `SupabaseConnection`, `SupabaseStorage`
- **External APIs**
    - 금융/지표 데이터: FRED, TwelveData, (기타 Public API)
    - 운영/분석: AWS Cost Explorer 등

---

3) 요청 처리 흐름 (Architecture Pattern)

표준 **Request Pipeline**

대부분의 요청은 아래 흐름을 따른다.

Client → Controller(api) → CQRS(CommandBus/QueryBus) → Handler(application) → Port(interface/token) → Adapter(infrastructure) → DB/Redis/Supabase/External API

- **Controller**: HTTP 입구(Validation/DTO), 로직 최소화
- **Handler**: 유즈케이스(비즈니스 규칙, 흐름 제어)
- **Port**: 외부 의존성 “계약” (Repository, Cache, External Client 등)
- **Adapter**: Port의 실제 구현 (TypeORM/Redis/Supabase/HTTP Client)

---

4) 폴더 구조 (Component View)

`src/` 아래가 도메인 경계이며, 도메인 내부는 거의 동일한 형태를 유지한다.

Root

- `src/main.ts` : 서버 부트스트랩, Swagger, Global Pipe/Filter, CORS
- `src/app.module.ts` : 전체 모듈 조립, TypeORM/Redis/Scheduler/Supabase 로딩

- `src/config/` : typeorm/redis 등 인프라 설정
- `src/commons/` : 공통 모듈/유틸 (Supabase 모듈 포함)
- `src/filter/` : 예외 필터
- `src/migrations/` : 마이그레이션(존재 시)

Domain Modules

- `src/numerical-guidance/`
- `src/user/`
- `src/education/`
- `src/admin/`
- `src/system-maintenance/`

도메인 내부 기본 규칙(공통)

- `api/` : Controller + DTO(요청/응답)
- `application/` : CQRS Commands/Queries + Handlers (유즈케이스)
- `domain/` : 도메인 모델/규칙(필요 시)
- `infrastructure/` : Adapter(Repository/Redis/External API/Supabase 등)
- `scheduler/` : 배치/크론 (있는 도메인만)

---

5) 모듈별 책임 (Domain Map)

5.1 Numerical Guidance (`src/numerical-guidance/`)

역할: 지표/차트데이터/예측/트렌드라인 등 “수치 가이던스” 도메인

- API Controllers 위치: `src/numerical-guidance/api/**`
    - `chart-data`, `indicator`, `trend-line` 등 + (base `/api/numerical-guidance` 아래 기능 컨트롤러)
- Application: CQRS 핸들러 중심(`application/command`, `application/query`)
- Infrastructure:
    - 외부 데이터 연동(FRED/Twelve/Public API 등)
    - (필요 시) Redis 캐시/집계
    - (필요 시) Supabase Storage 업로드

5.2 User (`src/user/`)

역할: 사용자/인증/멤버십(구독) 관련

- API Controllers:
    - `src/user/api/user/user.controller.ts` → `/api/user`
    - `src/user/api/membership/membership.controller.ts` → `/api/membership`
- 특징:
    - 인증/토큰/멤버십 흐름이 application에 모이고
    - 외부 인증/스토리지 연동은 infrastructure adapter로 분리됨(Supabase 등)
    - 스케줄러가 존재할 수 있음(멤버십 상태 갱신 등)

5.3 Education (`src/education/`)

역할: 교육/튜토리얼/게임 코인/이해도 저장 등 교육 도메인

- API Controllers:
    - `src/education/api/education/education.controller.ts` → `/api/education`
    - `src/education/api/tutorial/education-tutorial.controller.ts` → `/api/education/tutorial`
    - `src/education/api/game-coin/game-coin.controller.ts` → `/api/education/game-coin`
    - `src/education/api/understanding/education-understanding.controller.ts` → `/api/education/understanding`
- CQRS로 저장/조회 흐름 분리가 명확한 편(온보딩에 가장 좋은 예시 도메인)

5.4 Admin (`src/admin/`)

역할: 운영/통계/집계/분석(어드민 전용)

- API Controllers (`src/admin/api/**`)
    - `/api/admin/aws-cost`
    - `/api/admin/chartdata-avg`
    - `/api/admin/predict-chartdata-avg`
    - `/api/admin/telemetry`
    - `/api/admin/sojourn`
    - `/api/admin/user-stats`
    - `/api/admin/analytics`
- Scheduler (`src/admin/scheduler/**`)
    - 특정 기간 단위 집계, 예측값 생성 등 “백그라운드 작업” 포함

- Infrastructure:
    - Redis 기반 집계/캐시
    - AWS Cost Explorer 연동 등 외부 운영 데이터 수집

5.5 System Maintenance (`src/system-maintenance/`)

역할**:** 시스템 점검 상태 제공(프로덕션 프론트가 참조)

- Controller: `src/system-maintenance/api/system-maintenance.controller.ts`
    - Public: `GET /api/system/maintenance/status`
    - Admin:
        - `POST /api/admin/system/maintenance/reservation`
        - `DELETE /api/admin/system/maintenance/reservation`
        - `POST /api/admin/system/maintenance/inspection`
- CQRS 기반으로 “예약 저장/삭제/강제종료/상태조회”를 분리
- Scheduler로 오래된 예약/데이터 cleanup 등 운영성 작업 수행 가능

---

6) Cross-cutting Concerns (공통 관심사)

- **Validation**: DTO 기반 + Global ValidationPipe로 입력 안전성 확보
- **Error Handling**: Global `HttpExceptionFilter`로 예외 응답 표준화
- **Docs**: Swagger `/api`
- **Config**: `ConfigModule` 전역 사용 (환경변수 기반)

- **Transactions**: `typeorm-transactional` 사용(필요 시 유즈케이스 단위 트랜잭션)

---

7) “기능 추가” 시 개발자가 따라야 할 규칙 **(Onboarding 핵심)**

새 **API**를 추가할 때

1. `src/<domain>/api/`에 **Controller + DTO** 추가
2. 비즈니스 로직은 `src/<domain>/application/`에 **Command/Query + Handler**로 작성
3. DB/Redis/외부 API가 필요하면 `src/<domain>/infrastructure/`에 **Adapter** 구현
4. Handler는 Adapter를 직접 의존하지 않고, **Port(**계약**)** 을 통해 호출
5. `src/<domain>/<domain>.module.ts`에서 Controller/Handler/Adapter를 등록

“어디에 코드를 두면 되는지” 빠른 판단 기준

- HTTP 입구/응답 형태: `api/`
- 정책/흐름/유즈케이스: `application/`
- 영속성/외부연동/캐시: `infrastructure/`
- 규칙/모델(필요할 때): `domain/`