

API Execution Graph & Quick Overview

This guide explains the runtime execution graph of the Instat NG Backend API and gives newcomers simple, commented links and examples so they can understand how requests flow through the codebase.

Purpose: help a newcomer understand the main runtime path, where to look for behavior, and how to run and test key operations.

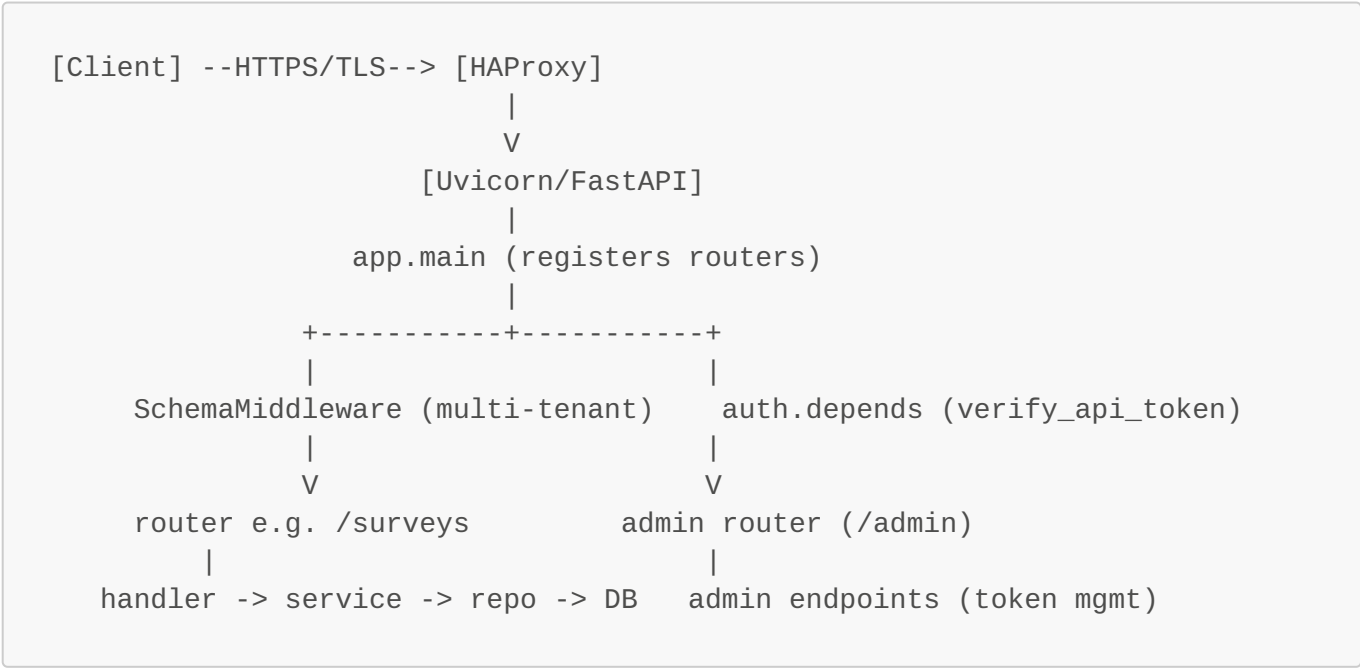
Table of Contents

- Execution graph (ASCII diagram)
- Request lifecycle (step-by-step) with file references
- Key components and where they live (files to open)
- Admin & RBAC notes
- Rate limiting & TLS (HAProxy)
- Quick run & test commands
- Simple examples (curl)
- Where to extend and common pitfalls

1) Execution Graph (simple ASCII)

Client -> HAProxy (TLS, rate-limit) -> API (uvicorn / FastAPI) -> `app.main` (app startup, middleware, routers)
-> Middleware: `SchemaMiddleware` (tenant/schema selection) -> Global dependency: `verify_api_token` (if enabled) -> Router (e.g. `/surveys`, `/domains`, `/admin`) -> Router handler -> Service layer -> Repository -> Database (Postgres)

ASCII diagram:



2) Request lifecycle — step-by-step (with file references)

1. Client issues request to API base URL (e.g. `https://api.example.org/api/surveys/`).
 - HAProxy handles TLS termination and rate limiting first (ACL + stick-table). If rate limit exceeded, HAProxy returns `429`.
 2. HAProxy forwards the request to the FastAPI service (uvicorn) where `app.main` is the entry point.
 - File: `app/main.py`
 - `app` registers middleware and routers here and applies a default dependency `verify_api_token` to data routers (unless `DISABLE_AUTH=true`).
 3. `SchemaMiddleware` runs for each request (multi-tenant / schema selection).
 - File: `app/middleware/schema_middleware.py`
 - Purpose: set DB schema or tenant context in the request lifecycle.
 4. Authentication step (if enabled): `verify_api_token` dependency inspects `Authorization: Bearer <token>`.
 - File: `app/core/auth.py`
 - Uses `APITokenManager` class methods to validate token, check expiration, and retrieve `user_id/role`.
 5. RBAC (role) checks when present: the `require_role([...])` dependency verifies user's role.
 - File: `app/core/auth.py` (function `require_role`)
 - Example: Admin routes use `require_role([Role.ADMIN])`.
 6. Router handler executes (e.g. `app/routers/survey.py`), parsing request body via Pydantic schemas and calling a service.
 - Files: `app/routers/*.py` (for each domain)
 - Example routers include: `app/routers/survey.py`, `app/routers/domain.py`, `app/routers/response.py`, `app/routers/admin.py`.
 7. Service layer performs business logic and validation.
 - Files: `app/core/services/implementations.py` (and `app/core/services/base.py`)
 - Services use repositories for DB access.
 8. Repository layer interacts with DB via SQLAlchemy.
 - Files: `app/core/repositories/implementations.py`
 - DB models are defined in `app/models.py`.
 9. Database: Postgres (configured by `DATABASE_URL`), DDL in `db/init/01_schema.sql`.
 - Connection/session helpers are in `app/database.py` and `app/deps.py`.
 10. Response is returned to the client; middleware or proxy may add headers (HSTS), logs, etc.
-

3) Key components & where to look (quick links)

- App entry point: `app/main.py` — router registration, global dependencies.
 - Middleware: `app/middleware/schema_middleware.py` — tenant selection.
 - Auth core: `app/core/auth.py` — `APITokenManager`, `verify_api_token`, `require_role`.
 - Routers: `app/routers/*.py` — per-resource endpoints (survey, domain, admin, question, response, etc.).
 - Admin: `app/routers/admin.py` — token generation, revoke, list; now protected by `require_role([Role.ADMIN])`.
 - Models: `app/models.py` — SQLAlchemy ORM models (User, Role, Permission, Survey, etc.).
 - Services: `app/core/services/implementations.py` — business logic layer.
 - Repositories: `app/core/repositories/implementations.py` — DB queries.
 - DB setup and session: `app/database.py`, `app/deps.py`.
 - Database schema (initialization): `db/init/01_schema.sql`.
 - Docker config: `Dockerfile`, `docker-compose.yml`.
 - Tests: `tests/` (E2E tests, run with `runTest.sh`).
-

4) Admin & RBAC notes

- Admin endpoints (create users, generate tokens, revoke tokens, list tokens) are in `app/routers/admin.py`.
 - Access control enforced with `require_role([Role.ADMIN])` and token validation `verify_api_token`.
 - Users, roles, permissions live in the DB (see `app/models.py`), so change roles through DB migrations or admin endpoints.
-

5) Rate limiting & TLS

- TLS termination and request rate limiting are expected to happen at HAProxy in front of the app.
- Example HAProxy ACL (already used in your infra):

```
acl api_path path_beg /api/  
acl high_rate sc0_http_req_rate(gt(20))  
http-request deny deny_status 429 if api_path high_rate
```

- If HAProxy returns `429`, requests are rejected before FastAPI processes them.
-

6) Quick run & test commands

Start locally with Docker Compose (recommended):

```
# from repo root  
docker-compose up --build
```

Start directly (dev) using uvicorn (inside virtualenv):

```
# activate venv
source .venv/bin/activate
# run uvicorn
uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload
```

Run tests (E2E):

```
# runTest.sh wraps pytest/httpx calls
./runTest.sh
# or basic pytest
pytest -q
```

Environment variables (examples):

```
export DATABASE_URL=postgresql+psycopg://user:pass@localhost:5432/dbname
export DISABLE_AUTH=false # set true for local debugging without token
validation
export TOKEN_EXPIRATION_DAYS=7
```

7) Example calls (curl)

1. Health:

```
curl http://localhost:8000/healthz
```

2. Create a user (ADMIN only):

```
curl -X POST http://localhost:8000/admin/users \
-H "Authorization: Bearer <ADMIN_TOKEN>" \
-H "Content-Type: application/json" \
-d '{"id":"alice","username":"Alice","role":"admin"}'
```

3. Generate token for user `alice` (ADMIN only):

```
curl -X POST http://localhost:8000/admin/tokens \
-H "Authorization: Bearer <ADMIN_TOKEN>" \
-H "Content-Type: application/json" \
-d '{"user_id":"alice","token_name":"cli"}'
```

4. Access protected data endpoint (example):

```
curl -H "Authorization: Bearer <USER_TOKEN>" http://localhost:8000/surveys/
```

If rate limit hit (HAProxy): response will be **429 Too Many Requests**.

8) Where to extend (high-level suggestions)

- Add audit logging in service layer to capture `user_id`, `action`, `resource_id`.
 - Put audit calls in `app/core/services/implementations.py` near DB commits.
 - Persist tokens in DB or Redis (instead of in-memory) for multi-instance deployments.
 - Modify `app/core/auth.py` to store tokens in a `api_tokens` table or Redis.
 - Add app-level rate limiting as a secondary defense (e.g. `slowapi`) even if HAProxy already limits.
 - Add more tests for RBAC paths (admin-only operations, insufficient role checks).
-

9) Troubleshooting quick tips

- 401 Unauthorized: token missing/invalid/expired; verify with `APITokenManager.list_tokens(user_id)` (admin) or re-generate token.
 - 403 Forbidden: user lacks required role; check DB `roles` assigned to user.
 - 429 Too Many Requests: rate limit hit at HAProxy — reduce request rate or adjust HAProxy ACL/stick-table.
 - DB connection errors: check `DATABASE_URL`, Postgres container/service availability.
-

10) Useful file list (for quick open)

- `app/main.py` (entry point)
 - `app/core/auth.py` (tokens, `verify_api_token`, `require_role`)
 - `app/routers/admin.py` (admin endpoints)
 - `app/routers/*.py` (domain APIs)
 - `app/models.py` (ORM models)
 - `app/database.py`, `app/deps.py` (DB session)
 - `app/middleware/schema_middleware.py` (multi-tenant logic)
 - `db/init/01_schema.sql` (DDL)
 - `docker-compose.yml`, `Dockerfile`
 - `runTest.sh`, `tests/` (testing)
-

Next steps I can do for you

- Add a simple diagram image (SVG) to the repo.
- Add example Postman/HTTP collection for quick testing.
- Add audit-logging scaffolding in service layer.

If you'd like any of those, tell me which one and I will implement it.