

API

Contract and OpenAPI

🕒 3 min read · Updated 11 October 2026 · Verified against [d2f2e41](#) on 2026-09-27

contract/openapi.json is generated from real responses and checked on every pull request; the golden vectors beside it pin what every route returns. How both are made, enforced and changed.

<https://docs.gurbanisoul.com/api/contract-and-openapi/>

The API's shape is not written by hand. `tools/gen_openapi.py` declares each route once (tag, a one-sentence description, parameters, limits, sample requests, plus a short verb-first summary in SUMMARIES — “Search the Granth” — that the reference's sidebar and headings show) and then **infers the response schemas from real responses** of the in-process API on the pinned database, so the spec cannot describe a field the server does not return. The result, `contract/openapi.json`, drives the [API reference](#) on this site and the [routes page](#).

Two tables that must agree

The dispatcher (`webapp/serve.py:ROUTES`) says which handler and which bounded context serve a route; the generator (`tools/gen_openapi.py:ROUTES`) says what a route takes and means. A unit test fails if a route is on one side and not the other, and `tools/gen_route_table.py --check` fails the docs build if the [routes page](#) was not regenerated after either changed.

tools/gen_openapi.py · lines 40–60

```
# The description is the operation's full sentence; its short sidebar/heading label is
in SUMMARIES.
LENS = [
    q("author", STR, "Lens: only verses by this author."),
    q("raag", STR, "Lens: only verses in this raag."),
    q("raags", STR, "Lens: only verses in these raags (comma-separated; the \"This hour\"
lens)."),
    q("rahao", {"type": "string", "enum": ["1", "true"]}, "Lens: refrains (Rahao lines)
only."),
    q("min_themes", dict(INT, minimum=0, maximum=8, default=0), "Lens: verses carrying at
least this many themes."),
    q("ang_min", dict(INT, minimum=1, maximum=1430, default=1), "Lens: first Ang."),
    q("ang_max", dict(INT, minimum=1, maximum=1430, default=1430), "Lens: last Ang."),
]

ROUTES = {
    "/api/meta": ("Meta", "Build identity and corpus statistics (version, commit, counts).",
    [],
    [("/api/meta", {})], [ ]),
    "/api/health": ("Meta", "Integrity self-test: line/Ang counts, FTS, verbatim Mool
Mantar, ੴ count, live verify, bani registry.", [ ],
    [("/api/health", {})], [ ]),
    "/api/search": ("Search", "Search the Granth through the ranked waterfall (exact FTS,
seeker lexicon, variants, English, passage, fold).", [
    q("q", {"type": "string", "maxLength": 300}, "Query in Gurmukhi, romanised Gurmukhi
or English.", True),
    q("mode", {"type": "string", "enum": ["auto", "gurmukhi", "roman", "english",
"first", "theme"], "default": "auto"}, "Search mode."),
    q("limit", dict(INT, minimum=0, maximum=200, default=50), "Page size (clamped)."),
```

Read from the source at build time — [view these lines on GitHub at c3d9360](#) .

Source: [tools/gen_openapi.py](#)² — the head of the declarations table, read at build time.

How the spec is checked

```
python3 tools/gen_openapi.py          # regenerate contract/openapi.json (needs
db/sogs.sqlite)
python3 tools/gen_openapi.py --check  # what CI runs: regenerate in memory, exit 1 on any
difference
```

web-ci · python runs the check on every pull request, and asserts the spec covers every dispatcher route. Response schemas are inferred with types merged across the sample requests, so a field that is sometimes null is typed as nullable rather than guessed.

The golden vectors beside it

contract/ also holds the golden vectors — recorded outputs of the real functions — and _meta.json with their counts and the db_sha256 they were built against. They pin *values*, where the OpenAPI spec pins *shape*:

File	What it fixes
<code>openapi.json</code>	every route's parameters and response schema
<code>golden_search.ndjson</code> , <code>golden_verify.ndjson</code>	what search and verification return for recorded inputs
<code>golden_reader.ndjson</code> , <code>golden_timing.ndjson</code> , <code>golden_banis.ndjson</code> , <code>golden_analytics.ndjson</code>	the reader, knowledge, Nitnem and insight projections
<code>golden_roman_norm.ndjson</code> , <code>golden_diffilib.ndjson</code>	the two pure functions the iOS port must reproduce byte for byte

`make contract` regenerates the vectors and fails on drift; `tools/contract_http.py` replays them over HTTP against a running API, which every deploy does before promoting. Details on [Harnesses and golden vectors](#).

The API contract version

`API_CONTRACT_VERSION` in `tools/gen_openapi.py` is the spec's own version, bumped only for an API change; a breaking change gets a new major and a new path prefix. The **product** version (`APP_VERSION`, shown by `/api/meta` and `/api/health`) is a separate number shared by web, API and app — see [One version number](#).

Changing the API honestly

1. Add or change the handler in the bounded context module and its entry in `serve.py:ROUTES`.
2. Declare it in `tools/gen_openapi.py:ROUTES` with parameters, limits and at least one sample request, and give it a short label in `SUMMARIES` (at most 32 characters; a test holds both tables to the same routes); regenerate the spec.
3. Regenerate the golden vectors (`make contract`) and review the diff line by line.
4. Regenerate the [routes page](#) (`python3 tools/gen_route_table.py`) and, if the route belongs to a context that runs as its own function on staging, the gateway (`tools/gen_gateway.py`).
5. Ship through the normal pipeline; the deploy replays the contract through the gateway before promoting. The app re-vendors the contract at the next release tag.

Links

1. https://github.com/Algorythmos-AI/siggs-platform/blob/c3d936031f0c283283c5d1102705affe1f8cf404/tools/gen_openapi.py#L40-L60
2. https://github.com/Algorythmos-AI/siggs-platform/blob/integration/tools/gen_openapi.py