

Architecture

Request lifecycle

🕒 2 min read · Updated 23 October 2026 · Verified against [a661527](#) on 2026-09-26

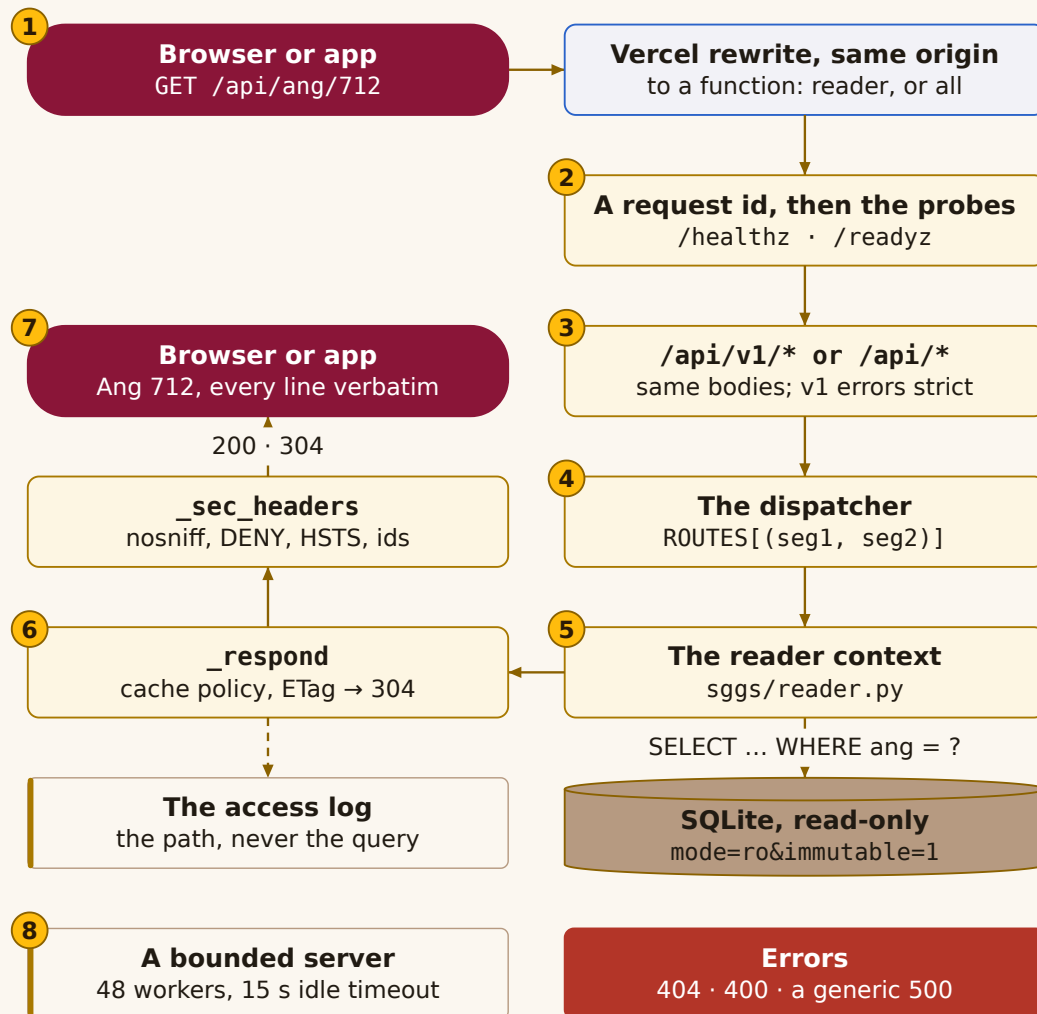
One request, GET /api/ang/712, followed from the browser through the host rewrite, the request handler, the dispatcher and the bounded context to SQLite and back, in the order the code does it.

<https://docs.gurbanisoul.com/architecture/request-lifecycle/>

Step through what `webapp/serve.py` does with one request. Everything in the poster is read from the code named in its footer; the wiki gate keeps the footer honest.

08 · Request lifecycle

What happens to GET /api/ang/712 between the browser and the database — in the order the code does it



■ a person □ external system □ component in this project ■ data at rest
 □ explanation ■ gate: fails the build or the deploy
 — flow: data or control moves this way
 --- reference: pins, reads or derives from

Source of truth: webapp/serve.py · webapp/sggs/core.py · gateway/routes.json · tools/gen_gateway.py

v1.3.10 · verified 2026-09-27 · d85c2ad

Sri Guru Granth Sahib Ji — Knowledge Base · engineering wiki

Poster 08 — the request lifecycle, from the browser to SQLite and back

1. **Same-origin call.** The browser (or a wiki widget) calls /api/ang/712 on the site's own host. There is no CORS and no API base URL in the frontend: a Vercel rewrite forwards the request — on staging to the context's own function (here reader); in production today (1.3.10) to all, the whole API as one function, and from the release after 1.3.10 to the context's function as on staging. The rules are generated from the route table.

2. **Identity and probes first.** `H._handle` stamps the clock and a request id — an incoming `X-Request-Id` or `x-vercel-id` is reused only if it is a plain token, otherwise a `uuid4` — then answers `/healthz` (liveness) and `/readyz` (every table each enabled context declares is present) before anything else.
3. **Two API surfaces, one body.** `/api/v1/*` has strict semantics: an unknown endpoint is 404 and every error is `{code, message, request_id}`. `/api/*` is the legacy surface whose responses are byte-pinned by the golden contract. Anything else is a static file, served from a realpath jail.
4. **The dispatcher.** `api()` splits the path, ensures FTS5 is detected, and looks up (first segment, second segment) in `ROUTES` — 29 routes, each tagged with its bounded context. In split mode a route whose context is not enabled in this process is a 404: a service never answers for another.
5. **The context reads SQLite.** The handler in `sggs/reader.py` runs parameterised SQL (never string-built; FTS columns allowlisted; every integer through `_int()`) on a connection opened read-only and immutable, one per handler thread. The app can never write to the scripture.
6. **Respond: caching and headers.** Responses that depend only on the immutable database (`ang`, `shabad`, `lines`, `analytics`, `timing` ...) are cacheable for the browser and the CDN, with a `sha256` ETag that turns a repeat into a 304. Every response carries `nosniff`, `X-Frame-Options: DENY`, `no-referrer`, `HSTS`, the request id and `X-Service` (which service answered).
7. **Log the path, never the query.** One JSON line per request goes to `stderr`: `method`, `path`, `status`, `milliseconds` and the request id. The query string is never written, so what people search for is not recorded anywhere. The verbatim lines of `Ang 712` reach the browser.
8. **Errors and limits.** An `ApiError` keeps its status (a missing composition is 404); a bad parameter is 400 with a short message; anything unexpected is a generic 500 — the detail goes to the server log, never over the wire. The server caps handler threads and times out idle sockets so a slow client cannot hold a slot.

The order, in prose

1. **Same-origin.** The browser calls `/api/ang/712` on the site's own host. A Vercel rewrite, generated from the route table (**bounded contexts**), forwards it to the API. No CORS, no API base URL in the frontend.
2. **Request id, then probes.** `H._handle` records the start time and a request id — an incoming `X-Request-Id` or `x-vercel-id` is reused only when it is a plain token — and answers `/healthz` (liveness) and `/readyz` (every declared table present) before anything else.
3. **Two API surfaces.** `/api/v1/*` is strict: unknown endpoint → 404; every error is `{error: {code, message, request_id}}`. `/api/*` is the legacy surface whose bodies the golden contract pins byte for byte. Anything else is a static file from a realpath jail.
4. **Dispatch.** `api()` looks up (first segment, second segment) in `ROUTES` — 29 routes, each tagged with its bounded context. In split mode a route outside `ENABLED` is a 404.
5. **The context reads SQLite.** The handler runs parameterised SQL on a connection opened `mode=ro&immutable=1` with `query_only`; the app can never write to the scripture.
6. **Respond.** Cacheable segments get `public`, `max-age=300`, `s-maxage=3600` and a `sha256` ETag (a repeat becomes a 304); everything else is `no-store`. Every response carries `nosniff`, `X-Frame-Options: DENY`, `Referrer-Policy: no-referrer`, `HSTS`, `X-Request-Id` and `X-Service`.

7. **Log the path only.** One JSON line per request with method, path, status, milliseconds and the id. The query string is never written ([ADR-0009](#)).
8. **Errors and limits.** `ApiError` keeps its status; a bad parameter is a 400 with a short message; anything unexpected is a generic 500 with the detail on the server log only. Handler threads are capped (`SGGS_MAX_WORKERS`) and idle sockets time out after 15 s.

Try it

With `serve.py` running locally:

```
curl -si 'http://localhost:7777/api/ang/712' | sed -n '1,12p'
```

Look for `X-Request-Id`, `X-Service: all`, `Cache-Control: public, max-age=300, s-maxage=3600` and the `ETag`. Send the `ETag` back in `If-None-Match` and the answer is a 304.

Related: [Engineering invariants](#) · the API reference on this site (generated from `contract/openapi.json`)