

Architecture

Bounded contexts and the gateway

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

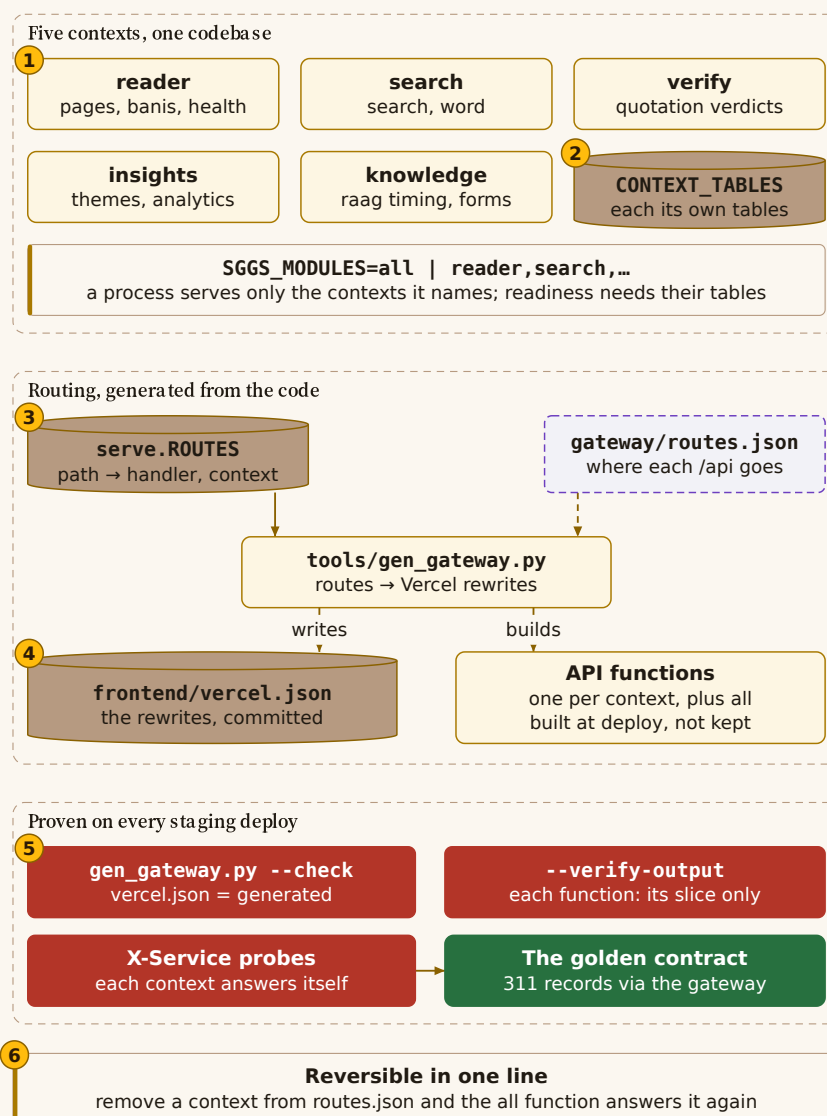
How one codebase runs as five contexts, how a process is sliced with `SGGS_MODULES`, and how the `/api` routing is generated from the route table and proven on every staging deploy.

<https://docs.gurbanisoul.com/architecture/bounded-contexts-and-gateway/>

The API is one stdlib module tree, `webapp/sggs/`, factored into five **bounded contexts**. Any of them can run alone as a service from the same image, and the routing that sends each `/api` prefix to the right place is generated from the code, never written by hand ([ADR-0010](#), [ADR-0011](#)).

09 • Bounded contexts and the gateway

One codebase, five contexts: how a function serves a slice of it, and how routing is generated so it cannot drift



□ component in this project ■ data at rest □ explanation or setting
 📌 pin: a reviewed lock file ■ gate: fails the build or the deploy
 ■ proven on every staging deploy — flow: data or control moves this way
 --- reference: pins, reads or derives from

Source of truth: webapp/serve.py · gateway/routes.json ·
 tools/gen_gateway.py · tools/build_api_functions.py
 v1.3.10 · verified 2026-09-27 · d2f2e41
 Sri Guru Granth Sahib Ji — Knowledge Base · engineering wiki

Poster 09 — five bounded contexts, module slicing, and the gateway generated from the code

- 1. Five contexts, 29 routes.** Every route in `serve.ROUTES` is tagged with the bounded context that owns it: reader (meta, health, Angs, compositions, random, lines, banis), search (search, word), verify, insights (the theme network, analytics, related, line concepts, neighbours) and knowledge (raag timing and forms). No two contexts share a path prefix, so any route can be sent to exactly one function.

2. **Each context declares its tables.** Each module lists the tables it reads (CONTEXT_TABLES). SGGG_MODULES chooses the contexts a process serves — all by default, or a comma list to run one context from the same code; an unknown name is an error, a route outside the enabled set is a 404, and readiness refuses a database missing any declared table. The same lists cut each function's database slice.
3. **Routing is generated, never hand-written.** gateway/routes.json says, per environment, where /api goes: on staging to Vercel functions, one per context plus all; in production (1.3.10 and 1.4.0) the entry is empty, so all answers every path, and the five contexts go live together in a later release, after the app is on the App Store. tools/gen_gateway.py combines it with the route table to produce the rewrites — every prefix of a routed context and its /api/v1 twin, then a catch-all to all.
4. **What ships.** The committed frontend/vercel.json carries only the rewrites: staging's, host-conditioned, first; then production's, host-less, one per routed prefix; then the catch-all to all. At deploy time tools/build_api_functions.py generates the functions — each fixing its SGGG_MODULES, its database slice and the commit — and adds their configuration; nothing generated is committed.
5. **Proven on every deploy.** CI fails if vercel.json is not what the code generates, if a function bundles anything but its own code and slice, if any context answers with the wrong X-Service (or an unknown prefix does not fall through to all), or if the golden contract — the 311 records the seven HTTP suites replay — does not pass through the gateway on the staging site.
6. **And reversible in one line.** Removing a context from gateway/routes.json sends its routes back to the all function on the next deploy. Production answers /api with all today and moves all five contexts in one release, each gated on its own — the functions proven at the commit, the golden contract, the performance baseline (p95 within +10 % per context) and the data canary; setting api_platform back to render returns it to the Render single API, which keeps deploying as the rollback.

The contexts and their routes

Context	Routes (/api/...)	Module
reader	meta,ang,shabad,health,random,lines,banis,bani	sggs/reader.py
search	search,word	sggs/search.py
verify	verify	sggs/verification.py
insights	themes/network,analytics/*,related,line_concepts,neighbors	sggs/insights.py
knowledge	timing/clock, timing/raag, timing/divergence, forms	sggs/knowledge.py

No two contexts share a path prefix, so any route can be sent to exactly one service. Each module declares the tables it reads (TABLES); CONTEXT_TABLES in serve.py collects them and is the source for the per-service database slices (make slices).

Running one context

```
SGGS_MODULES=search SGGG_DB=build/slices/search.sqlite python3 webapp/serve.py
```

SGGS_MODULES is all by default (the monolith) or a comma list; an unknown name is an error, a route outside the enabled set is a 404, and startup refuses a database missing any declared table (/readyz reports ready, contexts,missing_tables,version,commit).

Routing, generated

`gateway/routes.json` says per environment where `/api` goes. Staging and — on integration — production both list all five contexts, so each has its own Vercel function (`api/svc/<context>`) and `all` catches the rest. Production *today* (v1.3.10, released before the split) still answers every path with `all`; the five contexts go live there together in a later release, after the app is on the App Store (owner decision 2026-09-26, services-production step 3). Setting `api_platform: render` proxies `/api` back to the Render single API — the rollback. `tools/gen_gateway.py` combines it with `serve.ROUTES` and writes the rewrites in `frontend/vercel.json`: every prefix of a routed context and its `/api/v1` twin to its function, everything else to `all` — staging's rules first, host-conditioned, then production's, host-less. At deploy time `tools/build_api_functions.py` generates the functions themselves; nothing generated is committed.

What CI proves

- `gen_gateway.py --check`: the committed rewrites equal what the code generates.
- `build_api_functions.py --verify-output`: each function bundles only its entry, the API code and its own database slice.
- X-Service probes: each routed context answers its own routes; `/api/nope` falls through to `all`.
- The whole golden contract (311 records: the seven suites `tools/contract_http.py` replays) replays through the gateway on the staging site.

Reversal is one line: remove a context from `gateway/routes.json` and `all` answers it again on the next deploy. Production changes only when the move is released through the normal pipeline, and that release owes the performance gate — `tools/perf_baseline.py --compare docs/perf/baseline-2026-09.json` from the same vantage, p95 within +10 % per context — besides the data canary ([runbook](#)).