

Architecture

Architecture

Updated 27 September 2026 · Verified against [05086f5](#) on 2026-09-27

How the SGGS Knowledge Base is built: three repositories, one read-only API in five bounded contexts, a pinned database, and the surfaces that show the verbatim text.

<https://docs.gurbanisoul.com/architecture/>

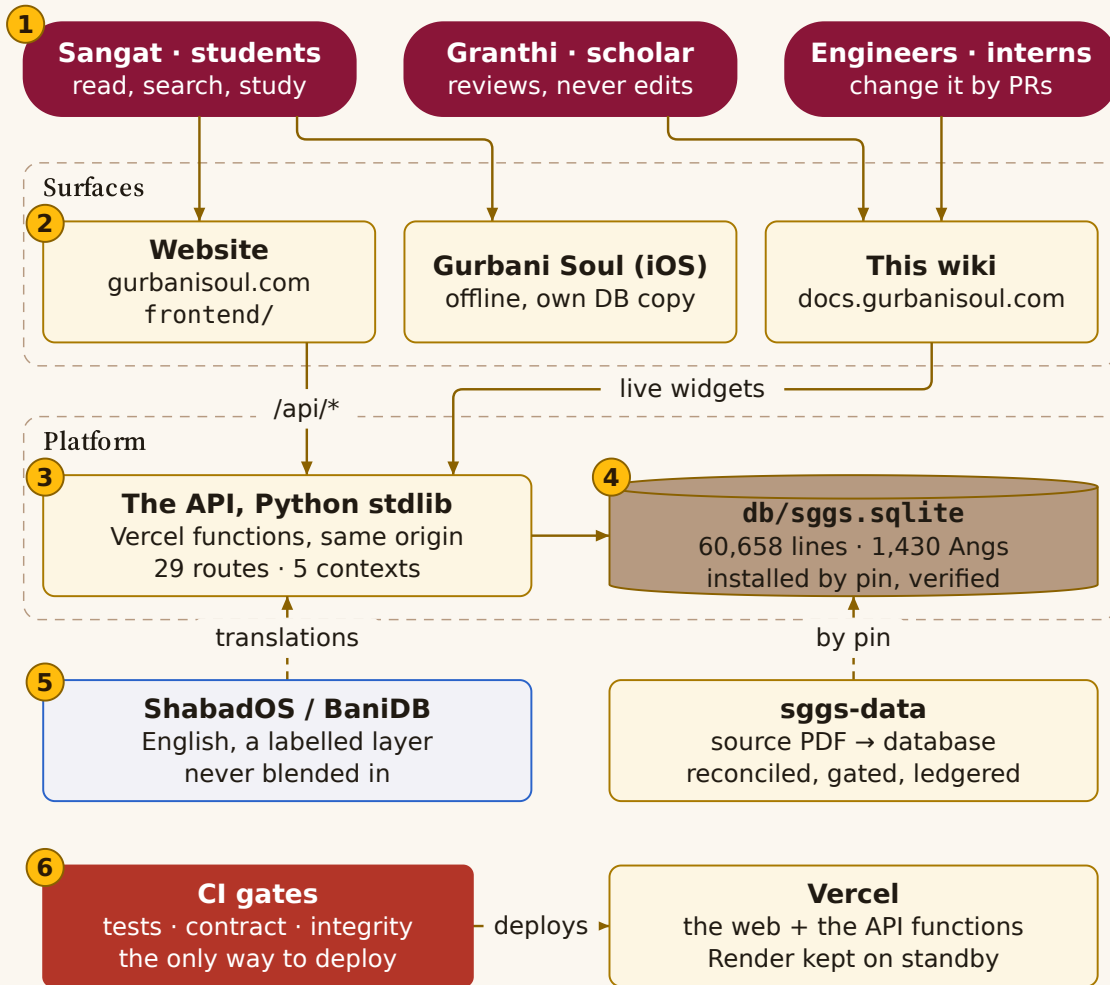
The system does one thing and is built around it: it shows **Sri Guru Granth Sahib Ji exactly as printed**, cited by Ang, and proves at every step that nothing changed. Everything else — search, verification, the insights, the app — is read-only around that text.

In three sentences:

- **Three repositories, joined by pins.** sggs-data builds the database from the source Bir and proves it character for character; this repository pins that exact database and serves it; the Gurbani Soul app pins the same database and this repository's golden contract.
- **One read-only API, five bounded contexts.** Reader, search, verify, insights and knowledge — each declares the tables it reads, runs as its own Vercel function (or all together), and opens SQLite read-only and immutable. The website and the wiki call it on the same origin.
- **Every deploy is gated and proven by commit.** Nothing reaches production except through CI, and production is re-checked against the pin every six hours.

01 · System landscape

Who uses the SGGs Knowledge Base, what they touch, and where the text comes from



Source of truth: docs/architecture/overview.md · webapp/serve.py · gateway/routes.json · dataset.lock.json · .github/workflows/deploy-production.yml

v1.3.10 · verified 2026-09-27 · d2f2e41

Sri Guru Granth Sahib Ji — Knowledge Base · engineering wiki

Poster 01 — the system landscape: the people, the three surfaces, the one read-only API, the pinned database, and where the verbatim text comes from

- 1. The people.** Three kinds of people meet the system: the sangat and students who read, search and study; Granthis and scholars who review anything flagged in the text but never edit it; and engineers and interns, who change code and documentation only through reviewed pull requests.

2. **Three surfaces.** The website (gurbanisoul.com, a static Astro site), the Gurbani Soul iOS app (fully offline, with its own copy of the database) and this wiki. All three show scripture verbatim, cited by Ang.
3. **One read-only API.** The API is Python standard-library code (webapp/serve.py and the five bounded contexts in webapp/siggs/: reader, search, verify, insights, knowledge) answering 29 routes. It runs as Python functions inside the same Vercel project as the website, so the browser calls it same-origin under /api/* with no CORS; gateway/routes.json decides which function answers which path. The wiki's live widgets call it the same way.
4. **The pinned database.** db/siggs.sqlite holds 60,658 verbatim lines over 1,430 Angs with FTS5 indexes, opened read-only. It is never committed here: dataset.lock.json names the siggs-data commit and the sha256 of the exact object, and make dataset installs it verified. The app builds its own copy from the same pin.
5. **Where the text comes from.** siggs-data turns the source Bir PDF (1,483 pages) into the corpus and the database through character-exact reconciliation, golden checks and a scholar-reviewed editorial ledger, then publishes the result by commit and checksum — the pin. The English layer from ShabadOS / BaniDB is a separate, labelled table the API reads beside the Gurmukhi, never blended into it.
6. **Only CI deploys.** A pull request runs the gates (tests, the golden contract, dataset integrity, security, the docs gates). Merging to integration deploys staging; a release merge to main runs the gated production deploy to Vercel — the website and the API functions together — verified by the running commit, and cuts the tag. The single Render API stays deployed as the rollback. Nobody deploys by hand.

The pages

The system on one page

who uses it, the containers across the three repositories, how production runs, the five data layers.

Request lifecycle

one request from the browser to SQLite and back, in the order the code does it (poster 08).

Bounded contexts and the gateway

the five contexts, how a function serves a slice of the database, and how routing is generated so it cannot drift (poster 09).

Three repositories and pins

every hand-off between data, platform and app as a reviewed lock file (poster 02).

Search waterfall

how a query becomes results, tier by tier (poster 05); part of Search & verification.

The insights context

theme networks, author and raag emphasis, Vaars and related verses: every number precomputed, descriptive only.

The knowledge context

The website

raag-timing claims credited to their sources, disagreement kept, and forms taken only from headings.

a static Astro site with no framework runtime, two layouts, and the API on the same origin.

Security and privacy

the defences at each layer, what is collected about readers, and the known gaps.

Quality and observability

each quality, how it is measured and gated, how production is watched, and what is not yet wired.

Microservices roadmap

how the monolith became modules, then functions, and what is left.

Read next

The decisions behind this shape are the ADRs — especially three repositories (ADR-0007), the dataset by pin (ADR-0008) and the API as functions (ADR-0011). How the text is kept whole end to end is the integrity chain.