

Architecture

Architecture Overview

🕒 2 min read · Updated 13 October 2026 · Verified against [a661527](#) on 2026-09-26

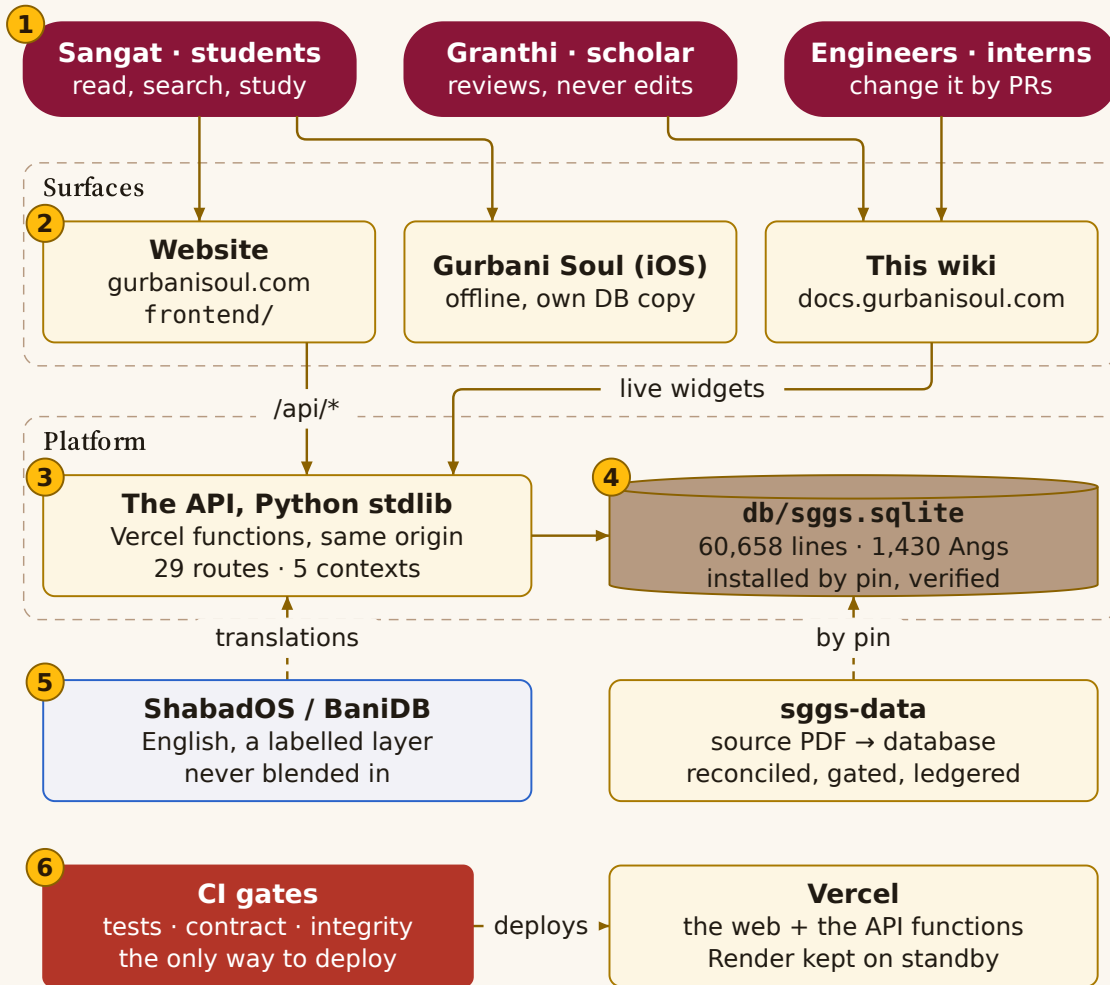
The system in one page: who uses it, the containers across three repositories, production deployment and the five data layers.

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

The whole system on one poster — press *Next* on the site to build it up step by step:

01 · System landscape

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



■ a person ■ component in this project ■ data at rest ■ external system
 ■ gate: fails the build or the deploy — flow: data or control moves this way
 --- reference: pins, reads or derives 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: people, surfaces, API, pinned database, sources, CI

- 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/sggs/: 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/sggs.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 sggs-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.** sggs-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.

System context (C4 level 1)

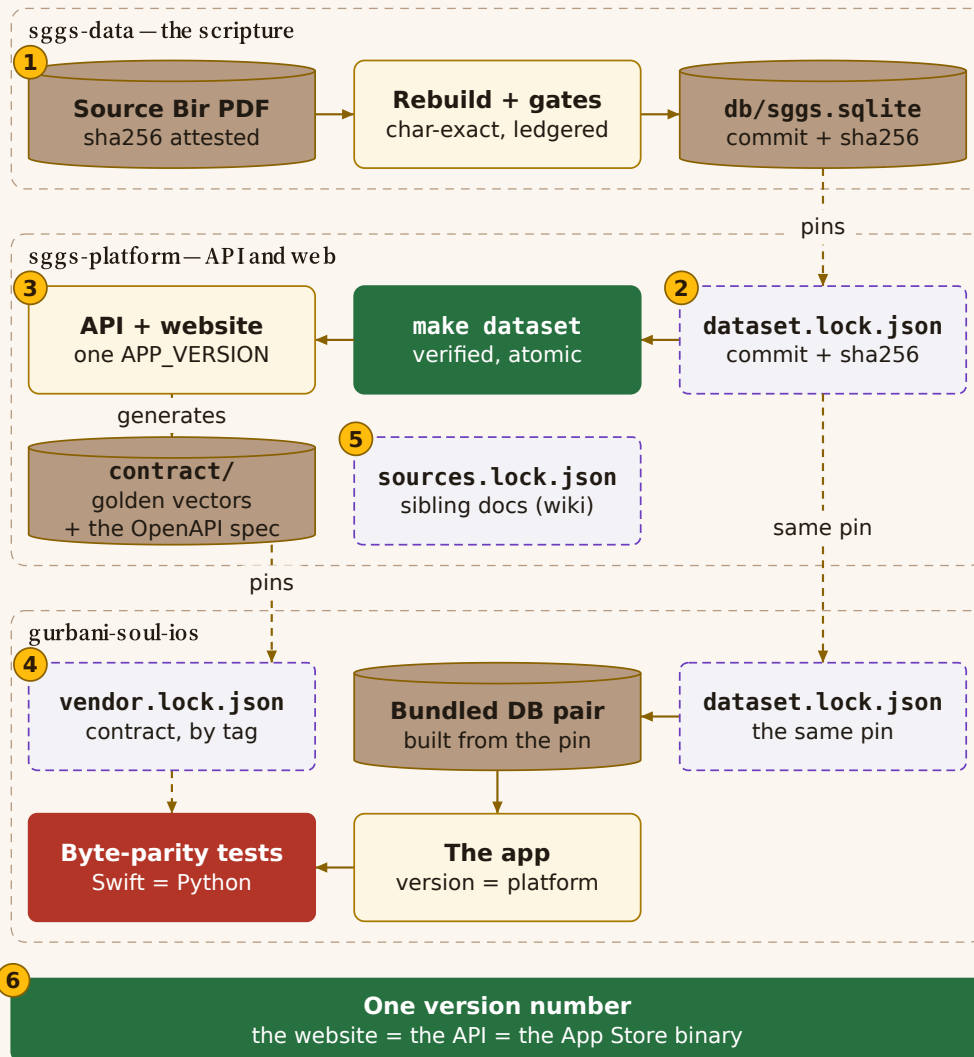
Poster 01 above is the system context: three kinds of people, three surfaces, one read-only API, the pinned database, the source Bir and the labelled English layer, and CI as the only way to production.

Containers (C4 level 2)

The containers live in three repositories joined by reviewed lock files — the data repository builds and proves the database, this one pins and serves it and publishes the golden contract, the app vendors both at a release:

02 · Three repositories and their pins

Nothing is copied by hand between data, platform and app — every hand-off is a reviewed lock file with a commit and a checksum

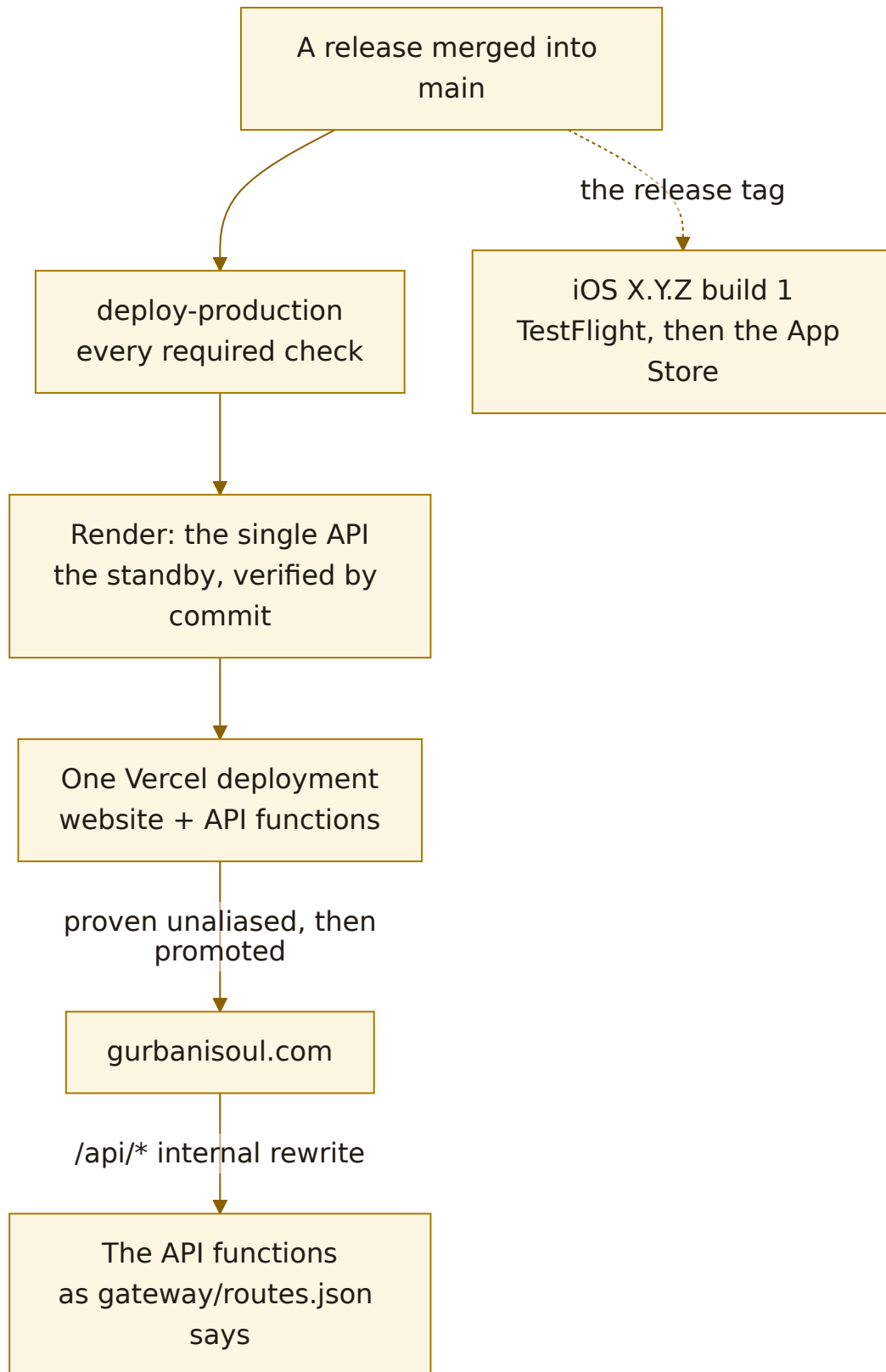


Poster 02 — three repositories and their pins: sggs-data builds and publishes the database, sggs-platform pins and serves it, gurbani-soul-ios pins the contract and the dataset, one version number

- The scripture is built and proven in sggs-data.** The source PDF's checksum is attested. The pipeline extracts the corpus, reconciles it character for character against the PDF, runs the golden checks, builds the SQLite database and passes the integrity gate. Any editorial repair is registered in the editorial ledger and approved in a scholar review. The database is published by commit, sha256 and size.

2. **The platform pins it.** dataset.lock.json names the sggs-data commit and the object's sha256 and size. make dataset downloads exactly that object, verifies it and installs it atomically; make dataset-check proves sggs-data publishes it at that commit. A new dataset arrives only as a reviewed bump of this file.
3. **The platform serves and publishes.** webapp/ and frontend/ serve the API and the website at one version. From the running code, tools/gen_golden_vectors.py records the golden contract — thousands of input and output vectors for the Roman fold, search, verification and the reader — and tools/gen_openapi.py the OpenAPI spec.
4. **The app vendors and re-proves.** gurbani-soul-ios pins the contract at a platform release tag (vendor.lock.json, a sha256 per file) and the same dataset pin. It builds its own bundled database pair from that pin and runs byte-parity tests so its Swift port equals the Python source of truth. Its MARKETING_VERSION must equal the platform version it vendored.
5. **This wiki pins the sibling docs.** docs-site/sources.lock.json records, per sibling repository, a commit and the sha256 of every published documentation file; the build installs them under /data/ and /ios/ with a banner naming the canonical source. A pin bump is a small reviewed pull request, opened by the weekly docs-pins job.
6. **One version number.** The web footer, /api/meta, /api/health, the iOS About screen and the App Store all report the same X.Y.Z, and every platform release is re-archived as iOS X.Y.Z (1) from the release tag. A release is complete only when web, API and the binary report the same number from the same commit.

Deployment (production)



Production deployment

The web frontend and API are **same-origin**: the browser calls `/api/*` and Vercel's internal rewrites send those paths to the API, which runs as Python functions in the same Vercel project ([ADR-0011](#)). There is no CORS and the frontend carries no API-base configuration. `gateway/routes.json` decides the routing per environment:

- **Staging** routes each of the five bounded contexts (reader, search, verify, insights, knowledge) to its own function, with `all` — the whole API — as the catch-all.
- **Production (v1.3.10, and 1.4.0)** answers every `/api` path with `all`.
- **From a later release, after the app is live on the App Store**, production routes the five contexts the same way as staging, all five in that one release ([services-production](#), step 3). It is one line in `gateway/routes.json`; 1.4.0 leaves it out.

Setting production's `api_platform` back to render proxies `/api` to the Render single API, which every release still deploys as the rollback target until it is retired. See [Environments](#) and [bounded contexts](#).

The five layers

1. **Scripture core** — lines (60,658 verbatim rows), raags, sections, authors, vaars.
2. **Search indexes** — `FTS5` (`fts`, `fts_en`, `fts_shabad`; `fts_tri` is built but no route reads it), `variants`, `word_freq`.
3. **Translations** — the Khalsa English layer, a separate labelled table (never blended into [Gurmukhi](#)).
4. **Analytics / Insight Engine** — theme network, stylometry, resonance, semantic neighbours.
5. **Knowledge layer** — attributed raag-timing claims (divergence preserved, never adjudicated).

Source of truth vs generated

- **In sggs-data**: the PDF-derived corpus and database, the timing seed, the concept definitions and the private translation inputs (fetched from `Algorythmos-AI/sggs-source`, checksum-verified).
- **Here, source of truth**: `webapp/serve.py` + `verify.py` + `romannorm.py` (the behaviour the Swift port mirrors), `frontend/src/`, and `dataset.lock.json` (which database is served).
- **Here, generated**: `contract/*.ndjson` + `contract/openapi.json`, `frontend/dist/`, `webapp/static/`; `db/sggs.sqlite` is installed from the pin, never committed.

See [microservices roadmap](#) for how the monolith is factored for a later service split.