

Architecture

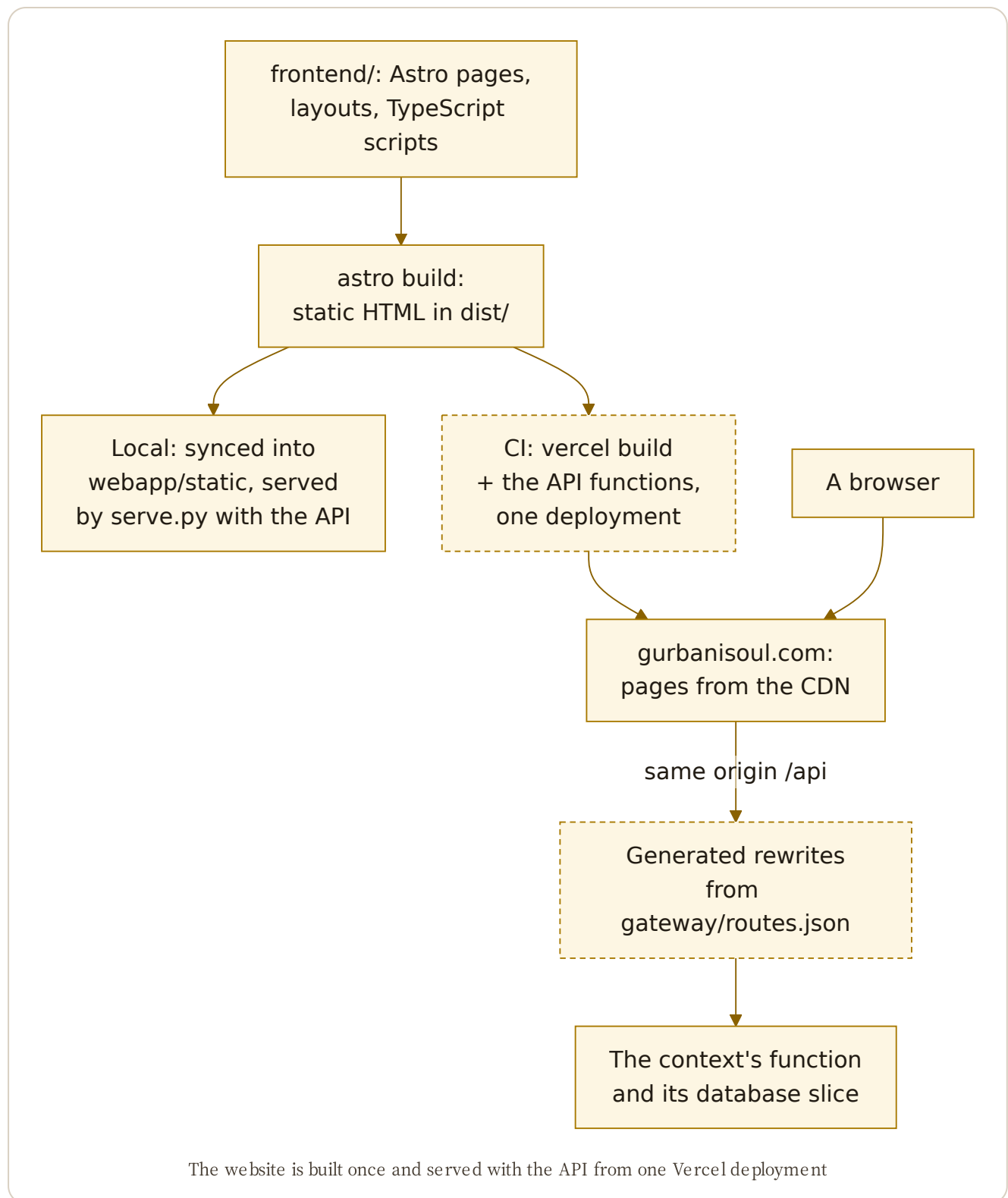
The website

🕒 5 min read · Updated 23 October 2026 · Verified against 05086f5 on 2026-09-27

How gurbanisoul.com is built: a static Astro site with no framework runtime, two layouts, plain TypeScript page scripts, and the API on the same origin through generated rewrites.

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

`gurbanisoul.com` is one static site, built with Astro from `frontend/`. Every page is plain HTML built ahead of time; there is no server rendering, no client framework and no service worker. Each page loads a few small TypeScript modules that call the read-only API on the same origin, `/api/...`, so the browser never talks to another host for scripture. This page explains how those pieces fit. The operational details (domains, SEO, images, the App Store switch) are in [the website runbook](#).



Two sites in one build

The build holds two properties that are deliberately kept apart (**brand**):

	Layout	Pages	Scripts it loads
Gurbani Soul (the app's home)	Marketing.astro	/, /features, /watch, /learn/*, /privacy, /support	landing.ts only (it never imports the Knowledge Base code), Vercel Web Analytics on gurbanisoul.com
Sri Guru Granth Sahib Ji Knowledge Base	Base.astro	/library (its home), /search, /reader, /nitnem, /browse, /themes, /lineage, /trail, /analytics, /constellation, /raag-clock, /divergence, /404	core, panel, studytrail, saroop on every page, plus the page's own controller; no analytics

frontend/src/routes.ts is the single list of public routes; the sitemap is built from it, and a repository gate fails if the sitemap and the built pages disagree.

The stack

- **Astro 7, static output** (output: 'static', no adapter). Pages are .astro files in frontend/src/pages/; Learn articles are MDX in src/content/learn/.
- **No islands.** There are no UI-framework components and no client: directives. Interactivity is plain TypeScript in frontend/src/scripts/, one controller per page. The charts use D3 and Chart.js, loaded only on the pages that draw them.
- **Tailwind CSS 4**, CSS-first (@import "tailwindcss" in global.css, which lists the ordered partials in styles/kb/; @theme is in kb/theme.css, no config file). The marketing pages take their colours from the brand tokens (a gate checks theme.ts against them). The Knowledge Base has its own theme set, generated from docs/brand/web-themes.json by scripts/brand/gen_web_themes.py, which proves every text pair before it writes anything. A theme, the appearance, the paper tone, the text size and the saroop switch are chosen in the display sheet (scripts/display.ts), on every Knowledge Base page.
- **One overlay stack.** The composition panel, Saved and the display sheet register with scripts/overlay.ts: Escape closes the top one, Tab stays inside it, focus returns to the opener, and a page's own keys (the Reader's arrow paging) ask overlayOpen() first.
- **The Reader's controls** (scripts/readerchrome.ts): the Go-to sheet, the page bar, reading options and the swipe. It only navigates; reader.ts draws.
- **Jump anywhere** (scripts/palette.ts, imported on first use by core.ts): plain matching on /api/meta and /api/banis, never a second copy of the search fold.
- **Page transitions without a router.** The Knowledge Base opts into cross-document view transitions in CSS (styles/kb/motion.css, inside prefers-reduced-motion: no-preference): the bars are held, the page cross-fades. The marketing pages keep Astro's <ClientRouter />. Navigation links carry data-astro-prefetch.
- **Fonts are self-hosted.** Sant Lipi (SIL OFL) renders Gurmukhi and is limited by unicode-range to the Gurmukhi block, so it never downloads for an English-only page; the marketing pages add Source Serif 4. Both are preloaded.
- **Node 22** (.nvmrc), used for the build only.

- **Open Graph cards** are drawn at build time with satori and resvg (`src/pages/og/`).

How a page talks to the API

Every call goes through `api(path)` in `core.ts`, which fetches `/api/ + path` on the same origin. Because the API is same-origin, there is no CORS and no API host in the page; the rewrites in `frontend/vercel.json`, generated from `gateway/routes.json`, send each prefix to its context's function ([bounded contexts and the gateway](#)).

Page	Calls
Every Knowledge Base page	<code>meta</code> (kept for the session), <code>random</code> , <code>shabad/{comp_id}</code> , <code>neighbors</code> , <code>lines</code> , <code>line_concepts</code>
<code>/search</code>	<code>search</code> , <code>verify</code>
<code>/reader</code>	<code>ang/{n}</code> (and prefetches the next and previous Ang), <code>neighbors</code> , <code>timing/clock</code>
<code>/nitnem</code>	<code>banis</code> , <code>bani/{key}</code>
<code>/lineage</code> , <code>/analytics</code>	<code>analytics/*</code> , <code>themes/network</code> , the build-time <code>contributors.json</code>
<code>/constellation</code>	<code>analytics/constellation</code> , <code>analytics/sky</code> , <code>analytics/pair</code> , <code>analytics/theme</code> , <code>timing/clock</code> (the This hour lens)
<code>/raag-clock</code> , <code>/divergence</code>	<code>timing/clock</code> , <code>timing/divergence</code>

The Gurbani Soul pages make no API calls.

Rules the scripts keep

- **Everything from the API is escaped.** Scripture and API text reach innerHTML only through `esc()` in `core.ts`. This is one of the [engineering invariants](#).
- **Saroop is display-only.** `saroop.ts` draws the traditional subjoined-ya form in the rendered glyphs only; the stored text, search, the API and Copy stay verbatim (a copy handler strips the display markup). It is on by default and remembers an explicit choice.
- **The pinned verse is read from the API model.** The Study Trail pins verbatim text from the API (`data-gm`), never from the rendered glyphs, which saroop may have changed.
- **State stays in the browser.** Preferences (theme, colour theme, paper tone, transliteration, text size, saroop, raag clock mode), reading positions and pinned verses (at most 500) live in `localStorage`; the meta and timing responses and the Semantic Trail path live in `sessionStorage`. There are no cookies and no accounts; the [privacy policy](#)¹ describes the same set.

Build, test and deploy

Step	Where	What
Build	<code>npm run build</code>	<code>astro build</code> into <code>frontend/dist/</code>
Local serving	<code>npm run sync</code>	copies <code>dist/</code> into <code>webapp/static/</code> for <code>serve.py</code> (a backup goes to <code>static.bak/</code>)
Checks	<code>web-ci</code> (frontend job)	build, <code>astro check</code> (advisory), the pahar vectors
Repository gates	<code>web-ci</code> (python job)	SEO, sitemap, RSS, the external-request allowlist, no secrets in the source, theme tokens, page-weight budgets for the Gurbani Soul pages
Browser tests	<code>e2e</code> (playwright job)	the Playwright specs in <code>frontend/e2e/</code> against <code>serve.py</code> : the CSP with zero violations on every route, axe on the main pages, the reader and search smoke
Staging	<code>deploy-staging.yml</code>	build the API functions, <code>vercel build</code> , deploy, alias <code>sggs-staging.vercel.app</code> , purge the CDN, then <code>@smoke</code> , the gateway probes and the golden contract
Production	<code>deploy-production.yml</code>	the same build deployed unaliased, verified, promoted to <code>gurbanisoul.com</code> , then <code>@smoke</code> on the public domain; a failure rolls back automatically (deploy runbook)

The Vercel project's own Git deployments are off; only these workflows deploy. The security headers the site sends are on [security and privacy](#).

Links

1. <https://github.com/Algorithmos-AI/sggs-platform/blob/integration/frontend/src/pages/privacy.astro>