

Architecture

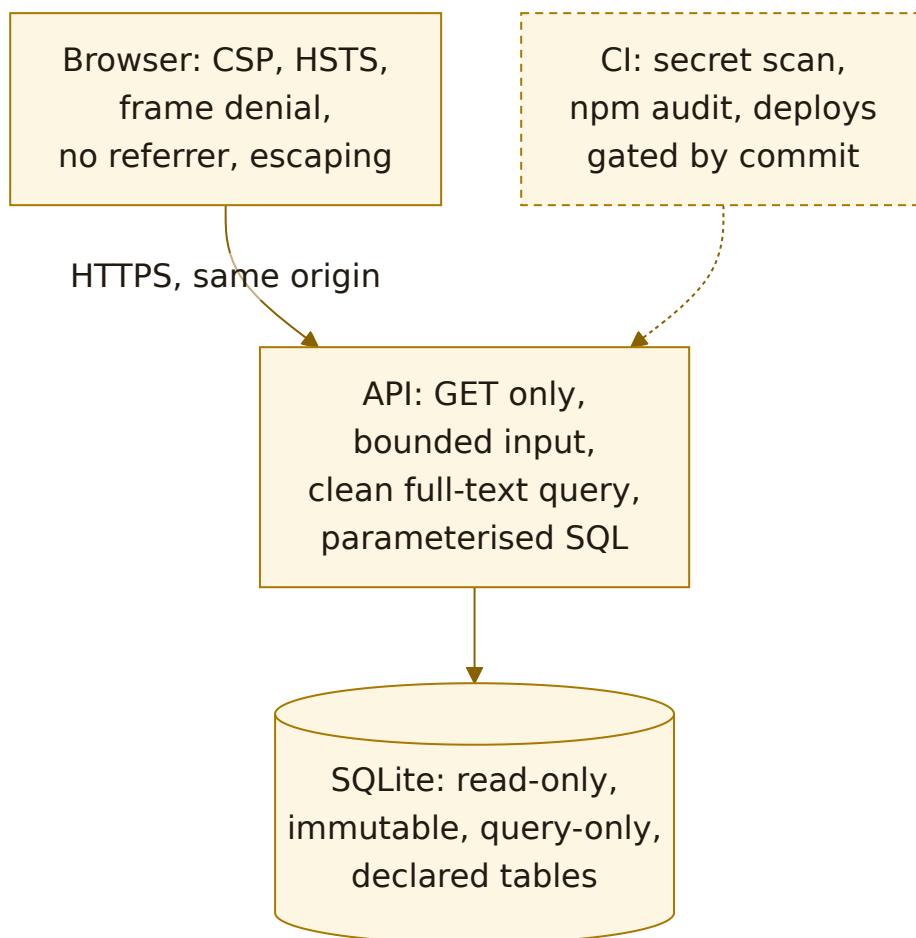
Security and privacy

🕒 5 min read · Updated 11 October 2026 · Verified against [05086f5](#) on 2026-09-27

The defences at each layer — browser headers, a read-only API with bounded input, an immutable database, gated CI — what is and is not collected about readers, and the known gaps.

<https://docs.gurbanisoul.com/architecture/security-and-privacy/>

The system has an unusually small attack surface, by design. It has no accounts, no logins, no user data, no request bodies and no writes: the API answers GET requests from a database it opens read-only. What it must protect is **the integrity of the text** and **the privacy of the people reading it**. This page lists the defences at each layer, what is collected, and the gaps that are known and open. To report a vulnerability, follow [SECURITY.md](#)¹ (email, not a public issue).



Defences at each layer between a reader and the database

The API

The same handler serves every host (`serve.py`, and each Vercel function subclasses it).

Defence	How	Where
Read-only by construction	only <code>do_GET</code> and <code>do_HEAD</code> exist, so there are no request bodies	<code>serve.py</code>
A database that cannot be written	<code>mode=ro&immutable=1</code> and <code>PRAGMA query_only=ON</code> ; the image makes the file <code>0444</code>	<code>sggs/core.py</code> , <code>webapp/Dockerfile</code>
Least privilege per context	each context declares its <code>TABLES</code> ; a test runs every route under an SQLite authorizer that refuses anything else, and a function will not load if its slice lacks a declared table	<code>webapp/tests/test_declared_tables.py</code> , <code>tools/build_api_functions.py</code>
Bounded input	every integer through <code>_int()</code> (at most 12 characters, clamped to a range); search <code>q</code> at most 300 characters, <code>limit</code> at most 200; a verify claim at most 600 characters; id lists at most 300	<code>sggs/core.py</code> , <code>sggs/search.py</code> , <code>sggs/verification.py</code>
No injection	every value bound with <code>?</code> ; the only built SQL strings are constant columns and <code>?</code> lists; full-text input loses <code>"</code> and <code>*</code> and each token is quoted; FTS columns come from an allowlist	<code>sggs/search.py</code> , <code>invariants</code>
No path traversal	static paths are resolved with <code>realpath</code> and must stay under the static root; %-encoded input is never decoded	<code>serve.py_resolve_static</code>
No leaks on error	a bad parameter is a 400 with a message the code wrote; anything unexpected is a 500 with the fixed body <code>{"error": "internal server error"}</code> and the traceback only in the log	<code>serve.py</code>
Slow clients	15-second socket timeout; at most 48 worker threads and a backlog of 128	<code>serve.py</code>
Response headers	<code>X-Content-Type-Options: nosniff</code> , <code>X-Frame-Options: DENY</code> , <code>Referrer-Policy: no-referrer</code> , <code>HSTS</code> , <code>X-Request-Id</code> , <code>X-Service</code> on every response	<code>serve.py_sec_headers</code>

The browser

Both sites send their headers from `vercel.json`:

- **Content Security Policy.** `default-src 'self'`, `frame-ancestors 'none'`, `object-src 'none'`, `base-uri 'self'`. The website allows `'unsafe-inline'` for scripts and styles (inline theme and handler code) and one outside origin, the newsletter provider, for `connect-src` and `form-action`. The wiki allows inline scripts only by **SHA-256 hash**, checked in CI by `npm run check:csp`.

- **Transport and isolation.** HSTS for two years with `includeSubDomains`; `preload`, `Cross-Origin-Opener-Policy: same-origin`, and a `Permissions-Policy` that turns off camera, microphone and payment (the website keeps geolocation for its own origin, for the raag clock's solar mode).
- **Escaping.** All scripture and API text is escaped by `esc()` before it reaches the page (the website).
- **Tests.** `frontend/e2e/csp.spec.ts` loads every route under the enforced policy and expects zero violations; the wiki's e2e server sends its production headers to every test.

CI and the supply chain

- **Deploys only through CI.** Production and staging deploy only from their workflows, after the required checks pass on the exact commit, and are verified by the commit the running API reports (ADR-0005). The platforms' own Git deployments are off.
- **Secrets** live in the production and staging GitHub Environments (and production-approval for an optional reviewer). Values are never typed into chat, logs or files; one that is pasted anywhere is treated as leaked and rotated (delivery).
- **Least-privilege workflows.** Every workflow declares `permissions::`; most are `contents: read`, with `issues: write` only where a workflow files an issue, and `contents: write` only for tagging a release.
- **Scans** (`security.yml`): gitleaks on every change (the required secrets check); `npm audit --audit-level=high` for the website and the wiki; bandit, semgrep and actionlint. Dependabot proposes npm, Actions and Docker updates weekly.
- **The dataset is fetched only by pin.** `fetch_dataset.py` downloads the exact object named in `dataset.lock.json` and checks its SHA-256 and size; `build_api_functions.py --verify-output` refuses a build that would publish the database or the API source as static files.

Privacy

What is and is not collected, as the public privacy policy² states it (every statement there is checked against the code):

- **The app** collects nothing: no accounts, no network connections, no analytics or third-party SDKs. Its App Store label is “Data Not Collected”. Location for the raag clock is rounded to about 1 km and stays on the device.
- **The Knowledge Base pages** load no analytics and set no cookies. Preferences, reading positions, pins and a rounded location (about 1 km) stay in the browser's storage.
- **The Gurbani Soul pages** on `gurbanisoul.com` use Vercel Web Analytics: cookieless totals of page, referrer, country, browser, OS and device type, with a visitor hash discarded daily.

- **The API log** is one JSON line per request with the method, the path, the status, the duration and the request id. It never records the query string, the IP address or the user agent; a test fails if a query string appears (ADR-0009). Hosting providers keep their own standard logs.
- **The newsletter**, when enabled, sends only an email address to Buttondown, with double opt-in (newsletter runbook).

Known gaps

These are open and tracked here so nobody mistakes them for guarantees:

- **Static analysis does not block.** bandit runs with `|| true` and semgrep with `continue-on-error`; only the secret scan is a required check. `npm audit` fails its job but is not in the branch rulesets.
- **Not every Action is pinned to a SHA.** The deploy and docs workflows are; `web-ci`, `security`, `e2e` and several others use version tags. `gitleaks` is downloaded without a checksum check.
- **The website's CSP allows inline script** (`'unsafe-inline'`), unlike the wiki's hash-based policy.
- **No application rate limiting.** The API relies on its bounded input and the platform's own protection.
- **The API container runs as the image's default user** (no `USER` line in the `Dockerfile`). Production runs as Vercel functions; the container is the Render standby.
- **frontend/e2e/headers.spec.ts never runs in CI.** It needs `SMOKE_BASE_URL`, which no workflow sets, so nothing checks the website's deployed headers.
- **No written threat model** for the web and API.

New risks go into the risk register.

Links

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