

API

Versioning and caching

🕒 3 min read · Updated 27 September 2026 · Verified against [c761e51](#) on 2026-09-27

The two API surfaces and their errors, request ids, which responses the browser and the CDN may cache and how ETags make revalidation cheap, the security headers, and X-Service.

<https://docs.gurbanisoul.com/api/versioning-and-caching/>

Two surfaces, one body

Every route is served twice. `/api/*` is the **legacy** surface whose responses are pinned byte for byte by the golden contract and the iOS app. `/api/v1/*` is the **strict** surface: the same handlers and the same bodies, but an unknown endpoint is 404 (not 400) and every error is a structured envelope:

```
{ "error": { "code": "not_found", "message": "no such endpoint: /api/v1/nope",  
  "request_id": "..." } }
```

code is `not_found` (404), `invalid_request` (a bad parameter, 400) or `internal` (500); every case is on the **error reference**. New clients should use `/api/v1`.

What happens to `/api`. Nothing is scheduled. The legacy surface is what the golden contract pins and what existing clients and the website call, so it keeps answering byte for byte. It has no retirement date; if one is ever set, it will be announced in the **API changelog** well ahead, and `/api/v1` is already the surface to build on.

webapp/serve.py · _handle_v1 · lines 363–388

```

def _handle_v1(self, u):
    """/api/v1/* – the same routes and bodies as /api/*, with strict semantics: an
    unknown
    endpoint is 404 (not 400), and every error is {"error": {"code", "message",
    "request_id"}}.
    The legacy /api/* responses stay byte-identical (the golden contract pins them)."""
    def err(status, code, message):
        body = {'error': {'code': code, 'message': message, 'request_id': self._rid}}
        return self._respond(status, json.dumps(body).encode(), 'application/json')
    legacy = '/api/' + u.path[len('/api/v1/'):]
    try:
        body = json.dumps(api(legacy, parse_qs(u.query)), ensure_ascii=False).encode()
        seg = legacy.split('/')[2] if legacy.count('/') >= 2 else ''
        cache = _CACHE_IMMUTABLE if seg in _CACHEABLE else 'no-store'
        return self._respond(200, body, 'application/json; charset=utf-8', cache)
    except ApiError as e:
        return err(e.status, 'not_found' if e.status == 404 else 'error', e.message)
    except (ValueError, IndexError, OverflowError) as e:
        if str(e) in ('unknown endpoint', 'missing endpoint'):
            return err(404, 'not_found', f'no such endpoint: {u.path}')
        return err(400, 'invalid_request', str(e))
    except Exception:
        import traceback
        traceback.print_exc(file=sys.stderr)
        try:
            if hasattr(_local, 'con'): _local.con.close(); del _local.con
        except Exception: pass
        return err(500, 'internal', 'internal server error')

```

Read from the source at build time — [view these lines on GitHub at c3d9360](#) .

Source: [webapp/serve.py · H. _handle_v1](#)², read at build time.

Request ids

Every response carries X-Request-Id, and the same id is written to the access log (with the method, path, status and duration — **never the query string**, so what people search for is not recorded anywhere). An incoming X-Request-Id, or Vercel's x-vercel-id, is reused when it is a plain token of 8–128 safe characters; anything else is replaced and never logged as sent. Quote the id when reporting a problem and one request can be followed from the edge to the service.

What may be cached

Only responses that depend on nothing but the immutable database are cacheable — the tuple `_CACHEABLE` in `serve.py`: `ang`, `shabad`, `lines`, `bani`, `banis`, `word`, `analytics`, `themes`, `timing`, `forms`, `neighbors`, `related`, `line_concepts`. They are sent with

```

Cache-Control: public, max-age=300, s-maxage=3600
ETag: "<sha256 of the body, 32 hex>"

```

so a browser keeps them for five minutes, the CDN in front of the API for an hour, and a repeat request with `If-None-Match` is answered `304 Not Modified` without a body. Everything else — `meta`, `health`, `random`, `search`, `verify` — is `Cache-Control: no-store`: a search is answered afresh and a health check is never stale. The rule is the same on both surfaces.

webapp/serve.py · `_CACHEABLE` · lines 184–185

```
_CACHEABLE = ('ang', 'shabad', 'lines', 'bani', 'banis', 'word', 'analytics', 'themes',
'timing',
'forms', 'neighbors', 'related', 'line_concepts')
```

Read from the source at build time — [view these lines on GitHub at c3d9360](#).

Source: [webapp/serve.py](#)² — the cacheable set, read at build time.

A dataset change is a new deploy (the pin is baked into the image), and the deploy workflows purge the CDN after each promotion (`vercel cache purge`, in `deploy-production` and `deploy-staging`), so no cached body outlives the database it came from.

Headers on every response

Header	Value	Why
X-Content-Type-Options	<code>nosniff</code>	the body is what the type says
X-Frame-Options	<code>DENY</code>	the API is never framed
Referrer-Policy	<code>no-referrer</code>	a request never carries where it came from
Strict-Transport-Security	<code>max-age=31536000</code>	honoured on the hosted API
X-Request-Id	the request id	one id from the edge to the log
X-Service	all, or the contexts this service runs	which service answered; the gateway's routing is verified by it

Limits a client will meet

- `limit ≤ 200` and `offset ≤ 1,000,000` on search; ids lists read at most 300 ids; a query is at most 300 characters; every integer is clamped rather than rejected, except where a value is required.
- The server is bounded: a fixed worker pool, a 15-second socket timeout, idle clients dropped.
- A composition id that is a permanent gap (see [the comp_id rule](#)) is 404 on both surfaces.

Links

1. <https://github.com/Algohythmios-AI/siggs-platform/blob/c3d936031f0c283283c5d1102705affe1f8cf404/webapp/serve.py#L363-L388>

2. <https://github.com/Algorythmos-AI/siggs-platform/blob/integration/webapp/serve.py>

3. <https://github.com/Algorythmos-AI/siggs-platform/blob/c3d936031f0c283283c5d1102705affe1f8cf404/webapp/serve.py#L184-L185>

Sri Guru Granth Sahib Ji — KnowledgeBase wiki. Generated from commit c3d9360. Updated 27 September 2026. Verified against c761e51 on 2026-09-27. This copy may be out of date; the page is at <https://docs.gurbanisoul.com/api/versioning-and-caching/>