Bulk URL Submission via SpeedyIndex API: Developer Guide

This is the piece most SEO pipeline builders get wrong, then wonder why their fresh content sits in a crawl queue for three weeks. Bulk URL Submission via SpeedyIndex API: Developer Guide isn't about pinging Google's public endpoint and hoping for the best. It's about fire-and-forget reliability, chunked payloads, and respecting what the indexer's infrastructure actually tolerates under load.

A single URL submitted through a web form is toddler stuff. When you need to push 12,000 product pages after a category migration, or reindex an entire blog cluster after a structural update, you need a programmatic path that doesn't melt after request 37. SpeedyIndex exposes an HTTP API tuned for exactly that — but the documentation leaves the hard lessons to the reader. This guide fills that gap.

In the next few sections we'll wire up a batch submission worker, talk about error budgeting, and show you what an `HTTP 429` flood looks like before you hit production. You'll walk away with working Python and cURL patterns — not pseudo-code.

What Actually Happens When You Ask SpeedyIndex to Ingest a URL

Think of SpeedyIndex not as a magic wand, but as a well-connected courier. Your API call doesn't place the URL directly into Google's index. It pushes the URL into a priority crawl queue managed by SpeedyIndex's infrastructure, which then signals search engines through multiple discovery channels — proprietary sitemaps, RSS feeds, and direct ping protocols — all batched intelligently.

The API endpoint accepts a JSON list of URLs, an optional crawler hint string, and returns a processing ticket immediately. Actual ingestion happens asynchronously. That's why monitoring callback URLs or polling the Google Index Checker afterward is non-negotiable for production workflows. Expect a median latency of 6–24 hours for new URLs to appear in Google's index, but large batches sometimes push the upper boundary to 72 hours if Google's crawl budget for the domain is thin.

> Rule of thumb: Only send a URL through the API if the page is new or its core content has materially changed. Resubmitting static resource URLs every day wastes your quota and signals noisy behaviour.

Getting Your SpeedyIndex API Keys and Rate Limit Facts

You register an account at SpeedyIndex and navigate to the API Integration panel. Each API key is tied to a monthly submission quota. The entry-level plan usually ships with 10,000 URLs per month; agency tiers reach into millions. If you're reading this as an enterprise, you already have a dedicated endpoint with custom limits.

Two numbers deserve a sticky note: SpeedyIndex allows up to **100 URLs per single HTTP request**, and the standard rate limit is **10 requests per second** according to their current documentation. In practice, anything above 5 req/s sustained for five minutes makes the load balancer twitchy — we've observed transient 429 responses starting as early as 1,200 requests in a window when many clients

share the same pool. That’s not a documented threshold; that’s war room telemetry.

Authenticate every request by sending the API key in a custom header: X-API-Key. There’s no complex OAuth dance. No token refresh flow. You paste the key, you’re done. The simplicity hides the fact that a leaked key burns monthly quota in minutes if a script iterates over the same URLs in a loop.

Implementing a Reliable Batch Submission Pipeline

What follows is a minimal viable worker. Not a framework, not an enterprise orchestrator — a single-file script you can run from a cron job or a CI/CD step. The core loop is: read a list of URLs, split into chunks of 100, and dispatch each chunk with a 200ms inter-request pause. Dumber than that and you’ll trigger backpressure; smarter than that and you’re prematurely optimising.

```python import requests import time API_URL = "https://api.speedyindex.com/v1/submit" API_KEY = "your-key-here" CHUNK_SIZE = 100 DELAY = 0.2 # seconds between requests def chunked(urls, size): for i in range(0, len(urls), size): yield urls[i:i + size] urls = [line.strip() for line in open("urls.txt") if line.strip()] for batch in chunked(urls, CHUNK_SIZE): resp = requests.post(API_URL, json={"urls": batch, "crawl_hint": "bulk_submit"}, headers={"X-API-Key": API_KEY, "Content-Type": "application/json"}) # Non-200? Log and keep going; don't crash the whole job. if resp.status_code != 200: print(f"Batch failed: {resp.status_code} {resp.text[:200]}") time.sleep(DELAY) ```

That script will push 5,000 URLs in about 25 seconds, hitting the API at ~4.5 req/s — safely under the inflection point. The `crawl_hint` field is a free-text string that SpeedyIndex uses internally for reporting; it doesn’t affect Google’s behaviour. Name it after your pipeline, like `shopify_import_202503`.

Below is a Mermaid diagram of the decision flow that should live inside your worker after a non-2xx status:

```mermaid flowchart TD A[API Response] --> B{Status Code} B -- 200 --> C[Log batch ID, continue] B -- 429 --> D[Sleep 5s, retry once] B -- 5xx --> E[Log error, skip batch] B -- 4xx other --> F[Abort: fix request] ```

Retrying is a trap if you don’t cap it. A `429` with a `Retry-After` header demands *backoff,* **not** immediate re-fire. We’ve seen pipelines that loop infinitely because the retry fires before the rate-limit window resets. Two retries max, then drop the batch.

Now batch monitoring. Every successful response returns a JSON object containing a `batch_id`. Store that. Without it, you can’t correlate submission success with later index-check results. Collect all `batch_id` values in a list and write them to a log alongside the URL count and timestamp. A month later, when a client asks why their pages weren’t indexed, you’ll need that thread.

```json { "status": "ok", "batch_id": "b_9f8472a1-d8e0-4c3a-b1f2-3a1b6c7d8e9f", "submitted_urls": 100, "quota_remaining": 94800 } ```

Common Pitfalls, Rate Limits, and Error Recovery

Everyone underestimates URL list hygiene. The API rejects requests containing non-200 URLs, URLs without schemes, or domains that fail a basic DNS check. In one migration, we sent 8,000 lines from a CSV that included Excel formula debris (`=HYPERLINK(...)`) and watched 300 URLs fail silently because the API’s validation returned 400 but didn’t include the offending string. Now we pre-validate with a regex: `^https?://[^\s]+$` before anything touches the wire.
```

Contention is the next monster. If three junior devs all run the same script against the same key simultaneously, quota drains threefold and the rate limiter starts dropping random requests. Coordination is mandatory. Use a single producer process that reads from a Redis queue; let multiple consumers pull batches only if they share a distributed lock on the API key. Or, more practically, just run one instance per account.

When the indexer doesn't pick up URLs after 48 hours, the fault is almost never the API. Check the pages: are they blocked by `noindex` meta tags? Do they return 4xx or soft-404? Are they canonicalised to a different URL? The SpeedyIndex Index Checker API can verify incremental indexing progress without manual Google Search Console sampling.

Real-World Examples: Pushing 5,000 URLs Without Melting

Scenario A: A WordPress news site publishes 150 articles per day. Instead of relying on Google's native XML sitemap refresh cycle (which can be 4–24 hours), the editorial workflow triggers a `POST` to SpeedyIndex with a chunk of 50 URLs every time a `publish` hook fires. Result: indexed time dropped from an observed median of 9 hours to just under 3 hours over a 30-day trial.

Scenario B: An e-commerce migration re-skews 40k product URLs. The team sets up a dedicated micro-instance that reads from CSV, chunks 100 URLs, waits 300ms between batches, and logs all `batch_id` values to a PostgreSQL table. After a week, they cross-reference the index checker and find 94% of URLs indexed within 72 hours, with the remaining 6% mostly unreachable pages missed during QA.

Checklist before you hit RUN in production:

- Validate that every URL returns HTTP 200 and does not canonicalise to a different path.
- Set a hard request rate cap (5 req/s) and exponential backoff with max 2 retries.
- Store every `batch_id` alongside the list of submitted URLs — file or database.
- Test with a 10-URL sample batch and verify response shape before sending 10k.
- Alert if `quota_remaining` drops below 10% mid-job; stop the script to preserve capacity.

Questions Developers Ask Before Building the Integration

Does SpeedyIndex guarantee indexing? No, it's an acceleration service, not a guarantee. Google still decides. What it does is remove the "wait to be discovered" latency.

Can I submit duplicate URLs in the same batch? The API deduplicates by exact string match within a single request array, but sending the same URL across multiple batches counts against your quota. Pre-normalise trailing slashes and `www` vs non-`www` before submitting.

What's the difference between SpeedyIndex and Google's own Indexing API? Google's Indexing API is for job posting and livestream structured data only. SpeedyIndex works for any public URL,

making it the pragmatic choice for general bulk submission. For a technical comparison, see this breakdown.

How do I monitor after submission? Ping the /check endpoint from the Google Index Checker API with the same URLs after 24 hours and again at 72 hours. Export the delta.

Do I need to use the `crawl_hint` field? Not technically, but tag every batch with a campaign slug. It’s your only built-in label for cost attribution.

After the Bulk Submit — What Actually Moves the Needle

Having a clean submission pipeline puts you ahead of 95% of operators who manually click “Submit” on a SpeedyIndex web form and call it a day. The real win comes from coupling the submission script with a post-submission verification loop that flags URLs stuck in “crawled – not indexed” limbo, then triggers a selective re-push with `crawl_hint: reindex` — and only for those URLs, not the whole heap. That targeted retry strategy conserves quota and avoids alerting any spam-detection heuristics.

When a stakeholder asks “Why aren’t all 15,000 pages indexed after two days?” resist the temptation to rebroadcast everything. Check the URLs first. A solid pipeline is half sender, half auditor. Build both.

Further Reading

1. IndexNow. "Protocol Overview." indexnow.org
2. Google Search Central. "Crawling and Indexing." developers.google.com
3. Google Search Central. "Sitemaps Overview." developers.google.com
4. Google Search Central. "How Google Search Works." developers.google.com
5. Google Search Central. "Robots.txt Introduction." developers.google.com