# Teoram — AI-Powered Technology CMS (Master Guide)

> **Goal**: An automated, trustworthy, and observable pipeline that discovers trending *technology* topics, clusters them, collects multi-source evidence, drafts high-quality articles, routes them through human-in-the-loop review, publishes, and continuously learns from user sentiment.

---

## 0) Repo Map — files you'll touch

**API (Node / Fastify / Prisma)** - `apps/api/src/index.ts` — server boot; load dotenv, register routes, start keep-alive & scheduler. - `apps/api/src/lib/trends/googleTrends.ts` — fetch Google Trends (tech). - `apps/api/src/lib/trends/youtubeTrends.ts` — fetch YouTube trending (category 28 Tech). - `apps/api/src/lib/gemini.ts` — embeddings + LLM grouping, drafting; deterministic configs & guardrails. - `apps/api/src/lib/qdrant.ts` — vector store client (upsert/search in `topics` collection). - `apps/api/src/lib/scrape.ts` — HTML/article scrapers + YouTube transcript fetcher (with polite crawling). - `apps/api/src/lib/pipeline.ts` — orchestration helpers; status machine; retries; metrics. - `apps/api/src/jobs/discovery.cron.ts` — cron job to run discovery hourly. - `apps/api/src/routes/topics.ts` — topic CRUD, discovery endpoint (idempotent). - `apps/api/src/routes/agents.ts` — long-running jobs: collect → draft → review → publish. - `apps/api/src/lib/keepAlive.ts` — DB ping every 4 min to prevent Neon cold-start.

**CMS (Next.js)** - `apps/cms/src/app/dashboard/page.tsx` — loads topics for the Discovery board. - `apps/cms/src/app/dashboard/DashboardClient.tsx` — "Run Discovery", status bars, actions. - `apps/cms/src/app/topics/[id]/page.tsx` — topic detail, title selection, step tracker. - `apps/cms/src/components/StatusPill.tsx` — uniform status chip. - `apps/cms/src/components/ProgressStepper.tsx` — shows pipeline steps with ticks/crosses. - `apps/cms/src/lib/api.ts` — axios/fetch client with POST defaults & error toasts. - `apps/cms/src/styles/globals.css` — already aligned with pills, buttons, cards.

**DB / Vector** - `packages/db/prisma/schema.prisma` — models & enum (✅ `Topic.slug @unique`). - Qdrant collection: `topics` (dimension = embedding size), payload: `{ id, title }`.

**Schedulers / Queues (optional but recommended)** - `apps/api/src/queue/index.ts` — BullMQ (Redis) or lightweight in-memory queue for local dev.

---

## 1) Environment & Keys

- `GEMINI_API_KEY` — AI Studio key (LLM + embeddings).
- `YOUTUBE_API_KEY` — Cloud Console key w/ YouTube Data API v3 enabled.
- `QDRANT_URL`, `QDRANT_API_KEY` — Qdrant Cloud.

- `POSTGRES_URL` — used by Prisma in `schema.prisma`.
- `JWT_SECRET`, `REVALIDATE_TOKEN` — API & ISR revalidation.
- **Load** `.env` at API bootstrap via `import "dotenv/config"`.

---

## 2) Status Machine (canonical states)

We track per **Topic**:

1. **NEW** — discovered & grouped, awaiting curation.
2. **APPROVED** — curator selected source set & confirmed.
3. **PROCESSING** — background job started.
4. **COLLECTED** — sources scraped, transcripts pulled, citations extracted.
5. **DRAFTED** — LLM produced draft (article + FAQ + metadata).
6. **ASSIGNED** — category/subcategory & content type decided.
7. **READY** — ready to publish.
8. **PUBLISHED** — live article pushed.

   Transitions are linear with error branches. Each sub-step logs a **tick** or **cross** with a message.

**UI**: `ProgressStepper` renders steps with status, percentage, and per-step logs.

---

## 3) Discovery Pipeline (hourly)

### Step 1 — Fetch tech-only trends

- **Google Trends** (category Computers & Electronics, e.g. `30`) via `google-trends-api`:
- If API returns topics only → augment with **Google News**/Search (first few News results).
- Each item mapped to `{ title, url?, kind: "NEWS" }`.
- **YouTube Trending** (category 28 Science & Technology) via `youtube.videos.list(chart=mostPopular)`:
- Map to `{ title, url, kind: "YOUTUBE" }`.
- **Filters**:
- English (optional), remove non-tech using heuristic keyword allowlist/denylist.
- Deduplicate by URL and normalized title.
- **Scheduler**: run **hourly** via `node-cron` or BullMQ repeatable job.

### Step 2 — Grouping (Gemini “Topic Curator”)

- Prompt Gemini with the mixed list; ask for JSON:

```
{ "topics": [ { "master": "string", "children": [{ "title": "…", "url":
"…", "kind": "NEWS|YOUTUBE|BLOG|SPEC" }] } ] }
```

- Enforce low temperature (`0.2`), JSON-only. Fallback to safe local grouping if LLM fails.

- Produce clusters: `[{ master, children[] }]`.

### Step 3 — Qdrant canonicalization

- For each `master`:
- Embed title → cosine search in `topics` collection.
- If score >= **0.86** (tunable), **reuse existing master title** from Qdrant.
- Else **upsert** new point `{ id: uuid, vector, payload: { id, title } }`.
- Output canonical master (`master_canonical`).

### Step 4 — Persist to Postgres (idempotent)

- `Topic.slug` unique; `Source.url` **not unique** (by your decision).
- `upsert Topic by slug(master_canonical)`, then `createMany(skipDuplicates: true)` sources.
- Status stays **NEW**.

### Step 5 — UI

- Discovery board lists **master topics** with source bullets, **Approve** button, checkbox per source.
- "Approve" posts `{ selectedSourceUrls[], contentType? }` → sets status **APPROVED**.

---

## 4) Post-approval Pipeline

### 4.1 Assign Content Type

Heuristic + LLM classification ("news", "compare", "launch", "unboxing", "analysis", "how-to").
Persist on `Topic` or `Article` draft metadata.

### 4.2 Collect Content

- Crawl selected URLs politely (User-Agent, robots.txt respected, 1 req/sec per host, 3 retries with backoff).
- Extract HTML main content via readability + boilerplate removal.
- **YouTube**: pull transcripts (captions API or third-party library), fallback to ASR only if legal & permitted.
- Normalize to `blocks[]`, keep **source citations** (URL, author, timestamp, quoted span).

### 4.3 Draft Article (Gemini "Tech Writer")

- Inputs: master topic, selected titles, extracted blocks, transcripts.
- Output: `{ title, tl_dr, body_html, faq_html, outline_json, metaTitle, metaDescription }`.
- Constraints:
- **No hallucination**: every claim must be traceable to a source; include `[source n]` markers.
- **Neutral tone**, disclosure of uncertainty, avoid sensationalism.

- **Non-derivative**: synthesize across sources; quote sparingly.
- **SEO**: descriptive H2s, internal linking candidates (later step).

### 4.4 Review / QA (LLM “Copy Editor” + Human)

- Lint: spelling, grammar, readability, passive voice.
- **Fact-check**: verify each numeric/date/spec with sources; flag ungrounded sentences.
- Safety: remove PII, defamatory language, or policy violations.
- Output status: **DRAFTED** → **READY** or **NEEDS_CHANGES**.

### 4.5 Categorize & Publish

- Auto-assign category/subcategory by taxonomy (LLM + rules).
- Publish to site; revalidate ISR; move to **PUBLISHED**; notify.

---

## 5) Sentiment & Product Score (post-publish agent)

- Sources: Reddit, YouTube comments, X/Twitter, Hacker News, Quora (respect terms; use official APIs where required).
- Pipeline:
- Collect comments/posts for the product/entity.
- De-dupe, language filter, spam filter.
- LLM or small classifier for **sentiment (−1..+1)** and **aspect tags** (battery, camera, price, build).
- Aggregate to per-aspect scores + overall **Product Score**.
- Store on `Article` or `Product` entity; update widgets on the article.

---

## 6) Guardrails & Quality

- **Deterministic LLM settings**: `temperature: 0.2`, `top_p: 0.9`, `top_k: 40` (tune conservatively).
- **JSON-only outputs** with schema validation (Zod) and robust fallback.
- **Citations required** in drafts; show sources at end.
- **Rate limits**: YouTube & Google; exponential backoff; cache last hour’s results.
- **Observability**: structured logs, per-step timings, failure reasons; Grafana/ELK if needed.
- **Security**: sanitize HTML; no remote code; strict CORS; JWT on protected endpoints.
- **Ethics & Robots**: obey robots.txt; `User-Agent` string; time-boxed retries; no aggressive crawling.

---

## 7) API surfaces

### Discovery

- `POST /api/v1/topics/discover` → run Step 1–4 now; returns `[{ topic, sources[] }]` (status NEW).

- `GET /api/v1/topics/discovery` → list NEW topics with sources.
- `POST /api/v1/topics/:id/approve` `{ selectedUrls[], contentType? }` → status APPROVED.

### Processing

- `POST /api/v1/agents/topics/:id/collect` → scrape/transcripts → status COLLECTED.
- `POST /api/v1/agents/topics/:id/draft` → LLM draft → status DRAFTED.
- `POST /api/v1/agents/topics/:id/review` → QA pipeline → READY.
- `POST /api/v1/articles/:id/publish` → publish + revalidate → PUBLISHED.

### Search (Qdrant)

- `GET /api/v1/search?q=iphone` → semantic search on `topics` collection; return matches.

---

## 8) Cron & Queue

- Hourly discovery: `node-cron` in `discovery.cron.ts` (or BullMQ repeatable job).
- Long jobs (collect/draft/review) → queue with retries, dead-letter, and progress events to feed UI.
- Keep-alive ping to DB every 4 minutes to avoid Neon cold starts.

---

## 9) Example Prompts

### 9.1 “Topic Curator” (Grouping)

**System**: *You are a technology news/topic curator… Output strict JSON matching this schema…*
**User**: bullet list of `[{kind}] title | url`

**Constraints**: merge only highly similar; master = concise noun phrase; drop off-topic items.

### 9.2 “Tech Writer” (Drafting)

**System**: *You are a senior tech journalist… Create original, non-hallucinated synthesis. Every claim must map to a source; include citation markers [1], [2]…*
**User**: master topic + selected sources + extracted blocks/transcripts.

**Deliver**: `title`, `tl_dr`, `body_html` with H2/H3s, `faq_html`, `outline_json`, `metaTitle`, `metaDescription`.

### 9.3 “Copy Editor” (QA)

**System**: *You check facts, style, clarity, and safety. Flag speculative or unsupported claims. Return a JSON report + suggested edits.*

---

## 10) Data Shapes (DB)

**Topic**: `{ id, slug (unique), title, status, score, createdAt, updatedAt }`
**Source**: `{ id, url, kind, topicId, approved, createdAt }`
**Article**: `{ id, topicId, sourceId?, title, tl_dr, body_html, faq_html, outline_json, metaTitle, metaDescription, publishedAt }`
**Citation**: `{ id, articleId, sourceUrl, sourceType, title?, author?, timestamp?, quote? }`

---

## 11) Qdrant

- Collection: `topics`
- vectors: `float32[embedding_dim]` (Gemini `text-embedding-004` size)
- payload: `{ id, title }`
- Create collection once with cosine distance; upsert on discovery; search on approval/dedup.

---

## 12) Acceptance Checklist

- [ ] Hourly discovery adds only tech-relevant clusters.
- [ ] Idempotent: repeated runs don’t duplicate topics/sources.
- [ ] Grouping always returns valid JSON (or safe fallback).
- [ ] Qdrant canonicalization reuses near-duplicate topics.
- [ ] UI shows per-topic stepper with ticks/crosses and live progress.
- [ ] Drafts contain citations and pass QA checks.
- [ ] Publishing moves topic to Published list and revalidates pages.
- [ ] Logging & metrics expose timing and error causes.

---

## 13) Next Up (nice-to-have)

- Auto internal links by semantic similarity to existing articles.
- Image selection (YouTube thumbnail / OpenGraph) with license checks.
- A/B titles; CTR monitoring.
- Multi-lingual expansion with machine translation + native review.
- Active learning: feed fact-check failures back to prompts.

---

**End**