CIE v2.3.2

DEVELOPER STEP-BY-STEP BUILD GUIDE

Plain English Walkthrough + LLM/Cursor Prompt Guidance + Validation Stages

Build it right. Validate with AI. Ship with confidence.

February 2026 | CONFIDENTIAL

Companion to: CIE v2.3.2 Complete Developer Handoff Package (12 files)

0. How to Use This Guide

This document is your roadmap. It tells you WHAT to build, in WHAT order, and HOW to use AI tools (Cursor, Claude, GPT) to speed up development and catch mistakes before they hit production.

0.1 The Three Box Types

Throughout this guide you will see three colour-coded boxes:

CURSOR PROMPT: Blue boxes = Cursor/LLM prompts

```
Copy-paste these into Cursor, Claude, or GPT.
They give the AI the right context to generate code for you.
Always feed the relevant spec doc as context alongside the prompt.
```

LLM VALIDATION: Green boxes = LLM validation prompts

```
After you build a feature, paste your code + these prompts into an LLM.
The LLM reviews your code against the spec and flags gaps.
Do this BEFORE merging any PR. It is your AI code reviewer.
```

Red boxes = Non-negotiable rules
These are hard constraints that cannot be changed.
If your code violates these, it fails review. Full stop.

0.2 Reference Documents

Every step references specific files from the handoff package. Always feed these to the LLM when prompting:

File	Use It For	Feed to LLM?
CIE_v2.3_Enforcement_Edition.docx	Business rules, gates, tier logic	Yes - as system context
CIE_v2.3.1_Enforcement_Dev_Spec.docx	API endpoints, vector logic, code samples	Yes - primary dev reference
CIE_v231_Developer_Build_Pack.docx	DB schema, RBAC, sequences, OpenAPI	Yes - for each relevant step
CIE_v232_Hardening_Addendum.docx	Edge cases, fail-soft, degradation	Yes - during hardening phase

CIE_Cable_Canonical_Schema_v2_3_2.xlsx	Field definitions, data types, enums	Export to CSV first
cie_v231_openapi.yaml	API contract generation	Yes - paste directly
golden_test_data.json	Test fixtures, seed data	Yes - for test generation
CIE_Doc5_UIUX_Specification.docx	Screen layouts, component spec	Yes - for frontend steps
CIE_v232_UI_Mockup.jsx	Interactive prototype reference	Yes - visual reference

PHASE 1: Foundation (Steps 1-3)

You are building the skeleton. Database, enums, and the validation API. Nothing else matters until this works.

Step 1: Create the Database

What you are doing: Creating all the tables that store SKU data, tiers, clusters, intents, audit results, and user permissions. The database is the single source of truth.

Plain English: You need about 15 tables. The most important ones are: skus (stores every product), clusters (groups of related products), sku_intents (which intents are assigned to each SKU), audit_results (AI citation scores), and content_briefs (auto-generated refresh tasks). Every table uses UUID primary keys. Every row has created_at and updated_at timestamps.

CURSOR PROMPT: Generate Database Migration

```
You are building a PHP/Python hybrid product content management system
called CIE (Catalog Intelligence Engine).

I will paste the database schema from our spec. Please generate:
1. A complete MySQL/PostgreSQL migration file
2. All tables, foreign keys, indexes, and enum types
3. Seed data for the 9-intent taxonomy (these are LOCKED, never user-editable)
4. Seed data for the 4 tier types: hero, support, harvest, kill

Key rules:
- UUIDs for all primary keys
- created_at/updated_at on every table
- The 9 intents are: problem_solving, comparison, compatibility,
  specification, installation, troubleshooting, inspiration,
  regulatory, replacement
- Tier enum: hero, support, harvest, kill
- Include the tier_intent_rules table that maps which intents
  are allowed per tier

[PASTE: Section 1 from CIE_v231_Developer_Build_Pack.docx]
```

LLM VALIDATION: After building the DB, validate it

```
Here is my database migration file. Please review it against the
CIE v2.3.2 spec and check:

1. Are all 15+ tables from the spec present?
2. Do the tier_intent_rules correctly restrict intents per tier?
   - Hero: all 9 intents, max 3 secondary
   - Support: all 9 intents, max 2 secondary
   - Harvest: only problem_solving, compatibility, specification; max 1 secondary
   - Kill: no intents allowed
3. Is the audit_results table structured for weekly scoring (0-3)?
4. Are there proper indexes on sku_id, cluster_id, and tier?
5. Is there a content_briefs table for auto-generated refresh briefs?
6. Is there an audit_log table for tracking all changes?

Flag anything missing or wrong. Be specific about what to add.

[PASTE: your migration file]
[PASTE: Section 1 from Developer_Build_Pack.docx for reference]
```

DATABASE NON-NEGOTIABLES
The 9-intent taxonomy is SEEDED, not user-created. It must be locked.
Tier assignments come from Finance/ERP sync, not manual entry.
Every audit_result row must reference a specific question_id + engine + week.
The clusters table must store the intent vector (embedding) as a JSON/BLOB field.

Step 2: Build the Pre-Publish Validation API

What you are doing: Building the most important endpoint in the entire system. Every time someone clicks Save or Publish on a SKU, this API checks 8 gates. If any gate fails, the save is blocked with a 400 error. There is NO override button.

Plain English: Think of it like airport security. Every SKU must pass through 8 checkpoints (G1 to G7 plus G6.1) before it can be published. If any check fails, the SKU gets rejected with a clear error message telling the user exactly what to fix. The PHP CMS calls this Python API on every save.

CURSOR PROMPT: Build the Validation Endpoint

```
Build a Python FastAPI endpoint: POST /api/v1/sku/validate

It receives a JSON payload with these fields:
  sku_id, cluster_id, tier, primary_intent, secondary_intents[],
  title, description, answer_block, best_for[], not_for[],
  expert_authority, action (save/publish)

It must check these 8 gates IN ORDER:
  G1: cluster_id exists in master cluster list
  G2: primary_intent is one of exactly 9 valid enums
  G3: 1-3 secondary_intents, all different from primary, all valid enums
  G4: answer_block is 250-300 chars AND contains primary intent keyword
  G5: at least 2 best_for entries AND at least 1 not_for entry
  G6: tier is one of: hero, support, harvest, kill
  G6.1: intents match tier restrictions (harvest=spec+1, kill=none)
  G7: expert_authority is non-empty for hero/support tiers

CRITICAL RULES:
- Return 200 with status:pass if ALL gates pass
- Return 400 with status:fail listing ALL failures (not just the first)
- Each failure includes: error_code, detail, user_message
- Harvest tier: G4, G5, G7 are SUSPENDED (skip them)
- Kill tier: ALL content gates blocked, only G1 and G6 checked
- Must respond within 500ms

[PASTE: Section 7.2 and 7.3 from CIE_v2.3.1_Enforcement_Dev_Spec.docx]
[PASTE: the OpenAPI yaml file for endpoint schemas]
```

LLM VALIDATION: Validate your gate logic

```
Here is my validation endpoint code. Test it against these scenarios:

SCENARIO 1 (should PASS): Hero SKU with all fields valid
SCENARIO 2 (should FAIL G4): Answer block = 312 chars
SCENARIO 3 (should FAIL G6.1): Harvest SKU trying troubleshooting intent
SCENARIO 4 (should FAIL G3): Secondary intent same as primary
SCENARIO 5 (should PASS): Harvest SKU with only spec + compatibility
   (G4, G5, G7 suspended)
```

```
SCENARIO 6 (should FAIL): Kill SKU with ANY content fields filled
SCENARIO 7 (should FAIL G7): Support SKU with empty expert_authority

For each scenario, tell me if my code would produce the correct result.
If not, show me exactly what line needs to change.

[PASTE: your validation code]
[PASTE: golden_test_data.json for test fixtures]
```

Step 3: Implement the Intent Taxonomy + Tier Lock (G6.1)

What you are doing: Loading the 9-intent taxonomy as a locked enum list, and building the tier-lock logic that physically removes intent fields from the CMS form based on the SKU's tier.

Plain English: When a content editor opens a Hero SKU, they see all 9 intent fields. When they open a Harvest SKU, they only see Specification plus one other. When they open a Kill SKU, all fields are gone and a red banner says 'This SKU is flagged for delisting. No edits permitted.' The fields don't just get greyed out, they get removed from the page entirely.

CURSOR PROMPT: Build the Tier-Lock UI Logic

```
Build a JavaScript module for the PHP CMS that controls which form
fields are visible based on SKU tier.

Input: tier string (hero/support/harvest/kill)
Output: show/hide specific form sections

Rules:
- Hero: all 9 intent fields visible, max 3 secondary, all CIE fields shown
- Support: all 9 intent fields visible, max 2 secondary, hide wikidata_uri
- Harvest: only show specification + problem_solving + compatibility fields,
  max 1 secondary, HIDE answer_block/best_for/not_for/expert_authority
- Kill: remove ALL fields, show readonly banner, disable save button

Also build the PHP backend check: GET /api/v1/taxonomy/intents?tier=X
Returns only the allowed intent enums for that tier.

[PASTE: Section 11.2 from CIE_v2.3.1_Enforcement_Dev_Spec.docx]
[PASTE: intent taxonomy JSON from Section 8.3]
```

PHASE 2: Content Engine (Steps 4-7)

You are building the intelligence layer. Vector validation, title engine, and schema output.

Step 4: Build the Vector Validation Service

What you are doing: Building a Python service that checks whether a product description actually matches the cluster it is assigned to. It does this by converting both the description and the cluster intent into vectors (lists of numbers) and measuring how similar they are.

Plain English: Imagine every cluster has a 'meaning fingerprint'. Every description also has a fingerprint. This service compares the two. If they are similar enough (cosine similarity >= 0.72), the description passes. If not, it fails and the user gets a message saying 'Your description does not match this cluster's purpose. Rewrite it.' This prevents staff from copying random marketing text into the wrong product.

CURSOR PROMPT: Build Vector Validation

```
Build a Python service with two endpoints:

1. POST /api/v1/sku/embed
   - Takes: { text: string }
   - Returns: { vector: float[], model: string, dimensions: int }
   - Uses OpenAI text-embedding-3-small (1536 dimensions)

2. POST /api/v1/sku/similarity
   - Takes: { description: string, cluster_id: string }
   - Loads the cluster's stored centroid vector from DB/Redis
   - Embeds the description
   - Computes cosine similarity
   - Returns: { cosine_similarity: float, threshold: 0.72,
     status: pass/fail, message: string|null }

IMPORTANT: Cache cluster vectors in Redis. They only change when
the SEO Governor updates a cluster intent statement.

Fail-soft rule (from v2.3.2 Hardening): If the embedding API is
down, LOG the failure but allow the save with a warning flag.
Do NOT hard-block saves when the AI service is unavailable.

[PASTE: Section 8 from CIE_v2.3.1_Enforcement_Dev_Spec.docx]
[PASTE: Patch 1 from CIE_v232_Hardening_Addendum.docx]
```

LLM VALIDATION: Validate vector service

```
Review my vector validation code and check:
```

```
1. Does it correctly compute cosine similarity using numpy dot product?
2. Is the threshold hardcoded at 0.72 (not configurable by users)?
3. Does it handle embedding API timeouts with fail-soft (warn, not block)?
4. Does it cache cluster vectors (not re-embed them on every request)?
5. Does the error message tell the user WHAT to fix, not just 'failed'?
6. Does it log every similarity check for audit purposes?

[PASTE: your vector service code]
```

Step 5: Build the Title Construction Engine

What you are doing: Building a service that generates or validates product titles. Titles must follow the formula: Intent phrase FIRST, then attributes after a pipe separator. No attribute-stacking allowed.

Plain English: A good title reads like: 'Warm Glare-Free Lighting for Living Rooms | Fabric Drum Shade 35cm E27'. A bad title reads like: 'Blue Fabric Drum Lampshade 30cm E27 Pendant'. The first half must tell the customer what problem the product solves. The second half (after the |) is the technical details. If a title starts with attributes instead of intent, it fails G2.

CURSOR PROMPT: Build Title Validation

```
Build a Python function that validates product titles against CIE rules:

Rules:
- Title must contain a pipe separator '|'
- Text BEFORE the pipe must relate to user intent/problem (not attributes)
- Text AFTER the pipe contains product attributes (size, colour, fitting)
- The primary intent keyword (or synonym) must appear before the pipe
- Title must not start with brand name
- Max 120 characters total

Function signature:
  validate_title(title: str, primary_intent: str, cluster_id: str)
  -> { valid: bool, issues: string[], suggested_fix: string|null }

Also build a title SUGGESTION function that takes cluster + intent + SKU
attributes and generates a compliant title the user can accept or edit.

[PASTE: Title Construction Rule from Section 4.2 of Enforcement doc]
[PASTE: Section 6 Worked Examples for reference titles]
```

Step 6: Build JSON-LD Schema Renderer

What you are doing: Building a PHP function that automatically generates structured data (JSON-LD) for each product page. This tells Google, AI engines, and other crawlers exactly what the product is, using Schema.org vocabulary. Hero SKUs get full schema with Wikidata links. Kill SKUs get nothing.

CURSOR PROMPT: Build JSON-LD Renderer

```
Build a PHP function: render_cie_jsonld($sku)
It generates Schema.org Product JSON-LD based on the SKU's tier:

Hero: Full schema with @type Product, name, description (from answer_block),
  brand, offers, material with sameAs Wikidata URI, additionalProperty for
  Expert Authority, Best For, Not For
Support: Basic schema - Product, name, description, brand, offers
Harvest: Minimal - Product, name, offers only
Kill: Return empty string (no schema)

Output as <script type='application/ld+json'> block for page <head>

[PASTE: Section 11.1 from CIE_v2.3.1_Enforcement_Dev_Spec.docx]
```

PHASE 3: Commercial Layer (Steps 8-10)

You are connecting the engine to money. ERP data flows in, tiers get computed, and effort gets allocated based on ROI.

Step 7: Build ERP Sync Endpoint

What you are doing: Building the endpoint that receives financial data from the ERP system (margin, CPPC, velocity, return rate) and recalculates which tier each SKU belongs to.

Plain English: Once a month, Finance pushes a file of numbers for every SKU. Your system takes those numbers, runs them through a formula, and assigns each SKU a tier: Hero (top 20%), Support (middle 50%), Harvest (next 20%), Kill (bottom 10%). If a SKU's tier changes, the CMS automatically shows or hides fields the next time someone opens it.

CURSOR PROMPT: Build ERP Sync + Tier Computation

```
Build: POST /api/v1/erp/sync

Receives JSON array of SKUs with: sku_id, contribution_margin_pct,
cppc, velocity_90d, return_rate_pct

For each SKU, compute the Commercial Priority Score:
  score = (margin/100 * 0.40) + (1/max(cppc,0.01) * 0.25) +
          (velocity/max_velocity * 0.20) + ((1-return_rate/100) * 0.15)

Then assign tiers based on percentile bands:
  Top 20% (>= p80) = hero
  Middle 50% (>= p30) = support
  Next 20% (>= p10) = harvest
  Bottom 10% = kill

Also implement auto-promotion rule:
  If a harvest SKU's velocity increases >30% vs previous quarter,
  auto-promote to support

Store tier history (old_tier, new_tier, changed_at, reason)

[PASTE: Sections 3.2 and 9.2 from Dev Spec]
```

Step 8: Build Channel Readiness Scoring

What you are doing: Building a scoring system that calculates how 'ready' each SKU is for each sales channel (website, Amazon, eBay, Google Shopping, etc.). The score goes from 0 to 100. Managers use this to see gaps at a glance.

CURSOR PROMPT: Build Readiness Score Calculator

```
Build: GET /api/v1/sku/{id}/readiness

Returns a readiness score (0-100) per channel per SKU.

Score components (weighted by tier):
- Has valid cluster_id: 10 points
- Has primary intent: 10 points
- Has secondary intents: 10 points
- Answer block passes G4: 15 points
- Best-for/not-for populated: 10 points
- Expert authority present: 10 points
- JSON-LD renders valid: 10 points
- Images meet channel requirements: 10 points
- Pricing present: 10 points
- Category-specific fields complete: 5 points

For Harvest tier: only score the gates that apply (max = 45 not 100)
  then normalise to 0-100 scale

[PASTE: Section 11.3 from Dev Spec]
[PASTE: Channel mapping from Developer Build Pack]
```

PHASE 4: AI Audit Engine (Steps 11-14)

You are building the self-healing loop. The system asks AI engines about your products every week, scores the answers, and auto-generates fix tasks when scores drop.

Step 9: Build the AI Citation Audit Runner

What you are doing: Building a Python service that sends 20 pre-written questions per product category to ChatGPT, Gemini, Perplexity, and Google SGE, then scores whether your products get mentioned in the answers.

Plain English: Every Monday at 6am, the system asks 20 questions like 'What pendant cable do I need for a ceiling rose?' to four different AI engines. It then checks: did any of those engines mention our product? Score 0 = not mentioned, 1 = mentioned, 2 = summarised, 3 = recommended as best. These scores are stored and trended over time.

CURSOR PROMPT: Build AI Audit Runner

```
Build a Python service that runs weekly AI citation audits.

Core flow:
1. Load golden_queries JSON for a category (20 questions each)
2. For each question, query 4 AI engines (ChatGPT API, Gemini API,
   Perplexity API, Google SGE scraper)
3. For each response, evaluate citation score (0-3):
   0 = not mentioned, 1 = cited, 2 = summarised, 3 = selected
4. Store results in audit_results table
5. Compare to previous 2 weeks for decay detection

Evaluation logic: Check if the AI response mentions:
  - Our brand name, OR
  - Our product name, OR
  - Text matching our answer_block (fuzzy match > 60%)

CRITICAL: Handle API rate limits and failures gracefully.
If an engine is down, mark that result as 'unavailable' not '0'.
(From v2.3.2 Hardening Addendum, Patch 2)

[PASTE: Sections 10.1 and 10.2 from Dev Spec]
[PASTE: golden_test_data.json]
[PASTE: Patch 2 from Hardening Addendum]
```

Step 10: Build the Citation Decay + Auto-Brief System

What you are doing: Building the self-healing loop. When a Hero SKU gets zero citations for 3 weeks in a row, the system automatically generates a content refresh brief and adds it to the team's work queue.

Plain English: Week 1 of zero = yellow flag in dashboard (no action needed). Week 2 = sends an alert to the content owner and SEO governor. Week 3 = the system writes a brief that says 'This SKU is failing on these questions. Here is the current answer block. Here is what competitors are saying. Fix this within 7 days.' Week 4 = escalates to the Commercial Director as a revenue risk.

CURSOR PROMPT: Build Citation Decay N8N Workflow

```
Build an N8N workflow (or Python cron equivalent) that runs after
each weekly audit and implements the citation decay escalation:

Logic:
1. Query audit_results for Hero SKUs
2. For each question, check consecutive_zero_weeks
3. Switch on count:
   - 1 week: set decay_status = 'yellow_flag'
   - 2 weeks: send Slack/email alert, set 'alert'
   - 3 weeks: call POST /api/v1/brief/generate to auto-create
     content refresh brief, set 'auto_brief'
   - 4+ weeks: escalate to Commercial Director, set 'escalated'

Auto-brief must include:
  SKU ID + name + tier + margin, failing questions with score=0,
  current answer_block, top 3 competitor answers (from AI responses),
  suggested revision direction, 7-day deadline

[PASTE: Section 5.3 and 9.3 from Dev Spec]
```

PHASE 5: Dashboard + CMS UI (Steps 15-18)

You are building what people see. The dashboard shows the numbers, the CMS enforces the rules, and the UI makes it impossible to do the wrong thing.

Step 11: Build the CIE Dashboard

What you are doing: Building the management dashboard with 6 key views: SKU detail cards, category heatmap, tier summary, decay monitor, effort allocation tracker, and bulk operations.

CURSOR PROMPT: Build CIE Dashboard

```
Build a React/PHP dashboard with these views:

1. SKU MATURITY GRID: Table showing all SKUs with columns for
   tier, readiness score per channel, decay status, last audit score
2. CATEGORY HEATMAP: Grid of category x channel showing avg readiness
   (colour coded: green >85, yellow 60-85, red <60)
3. TIER SUMMARY: Card per tier showing count, avg readiness, avg margin
4. DECAY MONITOR: List of Hero SKUs with consecutive zero scores,
   showing decay stage (yellow/alert/auto-brief/escalated)
5. EFFORT ALLOCATION: Pie chart showing % of content hours by tier
   with red alert if Hero < 60%
6. STAFF KPI CARDS: Per-user gate pass rate, avg time per SKU,
   rework count

Use the UI mockup JSX as visual reference for layout.

[PASTE: Section 11.3 from Dev Spec]
[PASTE: CIE_Doc5_UIUX_Specification.docx]
[PASTE: CIE_v232_UI_Mockup.jsx for component reference]
```

Step 12: Build the SKU Edit Form with Gate Enforcement

What you are doing: Building the CMS edit screen where staff fill in product data. The form dynamically changes based on tier (Step 3). When they click Save, it calls the validation API (Step 2). If validation fails, the form shows inline errors and blocks the save.

CURSOR PROMPT: Build SKU Edit Form

```
Build a PHP/JS CMS form for editing a single SKU with:

- Tier badge (readonly, colour coded) at top
- Cluster selector (dropdown from master list)
- Primary intent selector (dropdown, filtered by tier)
- Secondary intent multi-select (max based on tier)
- Title field with live validation indicator
- Description textarea with character count
- Answer Block textarea with LIVE char counter (must show 250-300 range)
- Best-For / Not-For repeater fields (add/remove)
- Expert Authority textarea
- Readiness score display per channel (readonly)

On Save click:
1. Call POST /api/v1/sku/validate
2. If pass: save to DB, show green toast
3. If fail: show red inline errors next to each failing field,
   DO NOT save, DO NOT close the form

Tier-lock: On page load, call applyTierRestrictions(tier) to
show/hide fields per Step 3 logic.

[PASTE: CIE_Doc5_UIUX_Specification.docx screen 2]
[PASTE: UI mockup JSX SKUEditScreen component]
```

PHASE 6: Hardening + Testing (Steps 19-21)

You are bulletproofing everything. Edge cases, fail-soft behaviour, RBAC, and end-to-end testing.

Step 13: Implement Fail-Soft Rules (v2.3.2 Hardening)

What you are doing: Making sure the system does not break when external services go down. If the embedding API is offline, saves should still work (with a warning flag). If an AI audit engine is unreachable, scores should be marked 'unavailable' not 'zero'.

CURSOR PROMPT: Implement Fail-Soft Hardening

```
Review the 6 hardening patches from v2.3.2 and implement:

Patch 1 - VECTOR VALIDATION FAIL-SOFT:
  If embedding API times out after 3s, allow save with
  validation_status='degraded', log the failure, queue for retry

Patch 2 - AI AUDIT DEGRADATION:
  If an engine returns error, mark as 'unavailable' not score=0
  Decay trigger only fires when >=3 engines were successfully queried

Patch 3 - READINESS SCORE DECOMPOSITION:
  Break readiness into sub-scores: content_score, schema_score,
  commercial_score. Show breakdown not just total.

Patch 4 - FAQ TEMPLATES:
  Auto-generate FAQ blocks from best_for + not_for fields

Patch 5 - CLUSTER GOVERNANCE:
  Add approval workflow for cluster changes (propose > review > approve)

Patch 6 - TIER-MODE UX COPY:
  Different banner text per tier on the edit form

[PASTE: entire CIE_v232_Hardening_Addendum.docx]
```

Step 14: Implement RBAC (Role-Based Access Control)

What you are doing: Making sure different roles can only do what they are supposed to. Content editors can edit SKU content but cannot change tiers. Finance can push ERP data but cannot edit content. The SEO Governor can modify clusters but no one else can.

CURSOR PROMPT: Build RBAC Layer

```
Implement role-based access control with these roles:

ADMIN: Full access to everything
SEO_GOVERNOR: Manage clusters, lock/unlock taxonomy, approve cluster changes
CONTENT_EDITOR: Edit SKU content fields (subject to tier-lock)
CONTENT_LEAD: Edit + approve + assign briefs + view effort reports
PRODUCT_SPECIALIST: Edit expert authority + safety certs only
CHANNEL_MANAGER: View readiness, manage channel mappings
FINANCE: Push ERP sync, view tier assignments, no content edit
AI_OPS: Run audits, view decay monitor, manage golden queries
VIEWER: Read-only dashboard access

Key restrictions:
- Only ADMIN + FINANCE can trigger tier recalculation
- Only SEO_GOVERNOR can modify cluster intent statements
- Only ADMIN can modify the 9-intent taxonomy (should never happen)
- Content editors CANNOT override validation gate failures

[PASTE: RBAC matrix from Developer Build Pack Section 3]
```

Step 15: Run End-to-End Validation

What you are doing: Running the golden test data through the entire system to make sure everything works from ERP sync through to published output.

LLM VALIDATION: Final System Validation - paste EVERYTHING

```
I have built the complete CIE v2.3.2 system. Here is my codebase.
Please review it against the full specification and check:

1. GATE ENFORCEMENT: Can any SKU be published without passing G1-G7?
2. TIER LOCK: If I set a SKU to Kill, can I still edit content fields?
3. VECTOR CHECK: Does the similarity threshold block bad descriptions?
4. DECAY LOOP: If a Hero gets 3 weeks of zeros, does an auto-brief appear?
5. ERP SYNC: If margin data changes, do tiers recompute correctly?
6. READINESS: Does the score reflect actual field completion?
7. RBAC: Can a content editor change a SKU's tier? (should be NO)
8. FAIL-SOFT: If the embedding API goes down, do saves still work?
9. AUDIT LOG: Is every change tracked with who/when/what?
10. JSON-LD: Does Hero output include Wikidata sameAs?

Run each check against the golden_test_data.json fixtures.
Flag any failure with: file, line number, what is wrong, how to fix it.

[PASTE: your complete codebase or key files]
[PASTE: golden_test_data.json]
[PASTE: CIE_v232_Hardening_Addendum.docx for edge cases]
```