CIE v2.3.2 - Complete Implementation Guide

From Zero to Production: A Developer's Technical Walkthrough

This document provides concrete code examples for implementing each component of the CIE v2.3.2 system.

Table of Contents

---

1. Database Setup

Complete Migration Files

001_create_users_table.sql

```sql
```

```
CREATE TABLE users (
    id CHAR(36) PRIMARY KEY DEFAULT (UUID()),
    email VARCHAR(255) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    INDEX idx_email (email),
    INDEX idx_active (is_active)
);
```

002_create_roles_table.sql

```
sql

CREATE TABLE roles (
    id CHAR(36) PRIMARY KEY DEFAULT (UUID()),
    name VARCHAR(50) NOT NULL UNIQUE,
    display_name VARCHAR(100),
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE user_roles (
    user_id CHAR(36) NOT NULL,
    role_id CHAR(36) NOT NULL,
    assigned_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    assigned_by CHAR(36),
    PRIMARY KEY (user_id, role_id),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (role_id) REFERENCES roles(id) ON DELETE CASCADE,
    FOREIGN KEY (assigned_by) REFERENCES users(id) ON DELETE SET NULL
);
```

003_create_skus_table.sql

```
sql
```

```
CREATE TABLE skus (
    id CHAR(36) PRIMARY KEY DEFAULT (UUID()),
    sku_code VARCHAR(100) NOT NULL UNIQUE,
    title VARCHAR(255) NOT NULL,
    short_description TEXT,
    long_description TEXT,
    meta_title VARCHAR(60),
    meta_description VARCHAR(160),

    -- Tier and clustering
    tier ENUM('HERO', 'SUPPORT', 'HARVEST', 'KILL') NOT NULL DEFAULT 'SUPPORT',
    tier_rationale TEXT,
    primary_cluster_id CHAR(36),

    -- Validation
    validation_status ENUM('DRAFT', 'PENDING', 'VALID', 'INVALID', 'DEGRADED') DEFAULT 'DRAFT',
    last_validated_at TIMESTAMP NULL,
    can_publish BOOLEAN DEFAULT false,

    -- ERP data
    current_price DECIMAL(10, 2),
    cost DECIMAL(10, 2),
    margin_percent DECIMAL(5, 2),
    annual_volume INT,
    last_sale_date DATE,
    strategic_hero BOOLEAN DEFAULT false,

    -- Readiness scores
    content_score INT DEFAULT 0,
    schema_score INT DEFAULT 0,
    commercial_score INT DEFAULT 0,
    readiness_score INT DEFAULT 0,

    -- Metadata
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    created_by CHAR(36),
    updated_by CHAR(36),

    INDEX idx_sku_code (sku_code),
    INDEX idx_tier (tier),
    INDEX idx_validation_status (validation_status),
    INDEX idx_cluster (primary_cluster_id),
```

```
    FOREIGN KEY (primary_cluster_id) REFERENCES clusters(id) ON DELETE SET NULL,
    FOREIGN KEY (created_by) REFERENCES users(id) ON DELETE SET NULL,
    FOREIGN KEY (updated_by) REFERENCES users(id) ON DELETE SET NULL
);
```

004_create_clusters_table.sql

```sql
CREATE TABLE clusters (
    id CHAR(36) PRIMARY KEY DEFAULT (UUID()),
    name VARCHAR(255) NOT NULL,
    intent_statement TEXT NOT NULL,
    primary_intent_id CHAR(36) NOT NULL,

    -- Vector storage (JSON array of 1536 floats)
    centroid_vector JSON,
    last_vector_update TIMESTAMP NULL,

    -- Governance
    is_locked BOOLEAN DEFAULT false,
    requires_approval BOOLEAN DEFAULT true,
    approval_status ENUM('DRAFT', 'PENDING', 'APPROVED', 'REJECTED') DEFAULT 'APPROVED',

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    created_by CHAR(36),

    INDEX idx_primary_intent (primary_intent_id),
    INDEX idx_approval_status (approval_status),
    FOREIGN KEY (primary_intent_id) REFERENCES intents(id),
    FOREIGN KEY (created_by) REFERENCES users(id) ON DELETE SET NULL
);
```

005_create_intents_table.sql

```sql
```

```sql
CREATE TABLE intents (
    id CHAR(36) PRIMARY KEY DEFAULT (UUID()),
    name VARCHAR(100) NOT NULL UNIQUE,
    display_name VARCHAR(150),
    description TEXT,
    is_locked BOOLEAN DEFAULT true,  -- The 9 canonical intents are LOCKED
    sort_order INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_name (name)
);
```

006_create_sku_intents_table.sql

```sql
CREATE TABLE sku_intents (
    id CHAR(36) PRIMARY KEY DEFAULT (UUID()),
    sku_id CHAR(36) NOT NULL,
    intent_id CHAR(36) NOT NULL,
    cluster_id CHAR(36) NOT NULL,
    is_primary BOOLEAN DEFAULT false,
    assigned_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE KEY unique_sku_intent (sku_id, intent_id),
    FOREIGN KEY (sku_id) REFERENCES skus(id) ON DELETE CASCADE,
    FOREIGN KEY (intent_id) REFERENCES intents(id) ON DELETE CASCADE,
    FOREIGN KEY (cluster_id) REFERENCES clusters(id) ON DELETE CASCADE,
    INDEX idx_sku (sku_id),
    INDEX idx_intent (intent_id)
);
```

007_create_audit_results_table.sql

```sql
```

```sql
CREATE TABLE audit_results (
    id CHAR(36) PRIMARY KEY DEFAULT (UUID()),
    sku_id CHAR(36) NOT NULL,
    engine_type ENUM('PERPLEXITY', 'OPENAI', 'ANTHROPIC', 'GEMINI') NOT NULL,
    score INT,  -- 0-100, NULL if engine unavailable
    status ENUM('SUCCESS', 'TIMEOUT', 'ERROR', 'UNAVAILABLE') DEFAULT 'SUCCESS',
    response_text TEXT,
    error_message TEXT,
    queried_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (sku_id) REFERENCES skus(id) ON DELETE CASCADE,
    INDEX idx_sku_date (sku_id, queried_at),
    INDEX idx_engine (engine_type)
);
```

008_create_content_briefs_table.sql

```sql
```

```sql
CREATE TABLE content_briefs (
    id CHAR(36) PRIMARY KEY DEFAULT (UUID()),
    sku_id CHAR(36) NOT NULL,
    brief_type ENUM('DECAY_REFRESH', 'NEW_PRODUCT', 'MANUAL', 'SEASONAL') DEFAULT 'MANUAL',
    priority ENUM('LOW', 'MEDIUM', 'HIGH', 'URGENT') DEFAULT 'MEDIUM',

    title VARCHAR(255) NOT NULL,
    description TEXT,
    current_content TEXT,
    suggested_actions JSON,  -- Array of action items

    status ENUM('OPEN', 'IN_PROGRESS', 'COMPLETED', 'CANCELLED') DEFAULT 'OPEN',
    assigned_to CHAR(36),
    deadline DATE,
    effort_estimate_hours DECIMAL(5, 2),
    actual_hours DECIMAL(5, 2),

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    completed_at TIMESTAMP NULL,

    FOREIGN KEY (sku_id) REFERENCES skus(id) ON DELETE CASCADE,
    FOREIGN KEY (assigned_to) REFERENCES users(id) ON DELETE SET NULL,
    INDEX idx_sku (sku_id),
    INDEX idx_status (status),
    INDEX idx_assigned (assigned_to),
    INDEX idx_deadline (deadline)
);
```

009_create_validation_logs_table.sql

```sql
```

```sql
CREATE TABLE validation_logs (
    id CHAR(36) PRIMARY KEY DEFAULT (UUID()),
    sku_id CHAR(36) NOT NULL,
    gate_type ENUM('G1_BASIC_INFO', 'G2_IMAGES', 'G3_SEO', 'G4_VECTOR',
                   'G5_TECHNICAL', 'G6_COMMERCIAL', 'G7_EXPERT') NOT NULL,
    passed BOOLEAN NOT NULL,
    reason TEXT,
    is_blocking BOOLEAN DEFAULT true,
    similarity_score DECIMAL(5, 4),  -- For G4 gate
    validated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    validated_by CHAR(36),

    FOREIGN KEY (sku_id) REFERENCES skus(id) ON DELETE CASCADE,
    FOREIGN KEY (validated_by) REFERENCES users(id) ON DELETE SET NULL,
    INDEX idx_sku_gate (sku_id, gate_type),
    INDEX idx_validated_at (validated_at)
);
```

010_create_tier_history_table.sql

sql

```sql
CREATE TABLE tier_history (
    id CHAR(36) PRIMARY KEY DEFAULT (UUID()),
    sku_id CHAR(36) NOT NULL,
    old_tier ENUM('HERO', 'SUPPORT', 'HARVEST', 'KILL'),
    new_tier ENUM('HERO', 'SUPPORT', 'HARVEST', 'KILL') NOT NULL,
    reason TEXT,
    margin_percent DECIMAL(5, 2),
    annual_volume INT,
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    changed_by CHAR(36),

    FOREIGN KEY (sku_id) REFERENCES skus(id) ON DELETE CASCADE,
    FOREIGN KEY (changed_by) REFERENCES users(id) ON DELETE SET NULL,
    INDEX idx_sku (sku_id),
    INDEX idx_changed_at (changed_at)
);
```

011_create_audit_log_table.sql

sql

```
CREATE TABLE audit_log (
    id CHAR(36) PRIMARY KEY DEFAULT (UUID()),
    user_id CHAR(36),
    entity_type VARCHAR(50) NOT NULL,  -- 'sku', 'cluster', 'user', etc.
    entity_id CHAR(36) NOT NULL,
    action VARCHAR(50) NOT NULL,  -- 'create', 'update', 'delete'
    field_name VARCHAR(100),
    old_value TEXT,
    new_value TEXT,
    ip_address VARCHAR(45),
    user_agent TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL,
    INDEX idx_entity (entity_type, entity_id),
    INDEX idx_user (user_id),
    INDEX idx_created_at (created_at)
);
```

Seed Data

001_seed_intents.sql

```
sql

-- THE 9 LOCKED INTENTS - These are NEVER editable by users
INSERT INTO intents (id, name, display_name, description, is_locked, sort_order) VALUES
(UUID(), 'problem_solving', 'Problem Solving', 'Troubleshooting, fixes, and solutions', true, 1),
(UUID(), 'comparison', 'Comparison', 'Product A vs Product B comparisons', true, 2),
(UUID(), 'compatibility', 'Compatibility', 'What works with what', true, 3),
(UUID(), 'product_specs', 'Product Specifications', 'Technical specifications and features', true, 4),
(UUID(), 'installation', 'Installation', 'Setup and installation guides', true, 5),
(UUID(), 'troubleshooting', 'Troubleshooting', 'Diagnostic and repair information', true, 6),
(UUID(), 'buyer_guide', 'Buyer Guide', 'Purchasing decision support', true, 7),
(UUID(), 'use_case', 'Use Case', 'Real-world application scenarios', true, 8),
(UUID(), 'product_overview', 'Product Overview', 'General product information', true, 9);
```

002_seed_roles.sql

```
sql
```

```
INSERT INTO roles (id, name, display_name, description) VALUES
(UUID(), 'ADMIN', 'Administrator', 'Full system access'),
(UUID(), 'SEO_GOVERNOR', 'SEO Governor', 'Manage clusters and taxonomy'),
(UUID(), 'CONTENT_EDITOR', 'Content Editor', 'Edit SKU content'),
(UUID(), 'CONTENT_LEAD', 'Content Lead', 'Manage content team and briefs'),
(UUID(), 'PRODUCT_SPECIALIST', 'Product Specialist', 'Edit technical and expert fields'),
(UUID(), 'CHANNEL_MANAGER', 'Channel Manager', 'View readiness and manage channels'),
(UUID(), 'FINANCE', 'Finance', 'ERP sync and tier management'),
(UUID(), 'AI_OPS', 'AI Operations', 'Run audits and manage AI systems'),
(UUID(), 'VIEWER', 'Viewer', 'Read-only access');
```

---

2. Core Enums & Types

PHP Enums

TierType.php

```php
```

```php
<?php

namespace App\Enums;

enum TierType: string
{
    case HERO = 'HERO';
    case SUPPORT = 'SUPPORT';
    case HARVEST = 'HARVEST';
    case KILL = 'KILL';

    public function displayName(): string
    {
        return match($this) {
            self::HERO => 'Hero',
            self::SUPPORT => 'Support',
            self::HARVEST => 'Harvest',
            self::KILL => 'Kill',
        };
    }

    public function description(): string
    {
        return match($this) {
            self::HERO => 'Top 20% margin+volume products',
            self::SUPPORT => 'Profitable products',
            self::HARVEST => 'Low margin but still selling',
            self::KILL => 'Negative margin or no sales',
        };
    }

    public function color(): string
    {
        return match($this) {
            self::HERO => '#10B981',    // Green
            self::SUPPORT => '#3B82F6',  // Blue
            self::HARVEST => '#F59E0B',  // Yellow
            self::KILL => '#EF4444',    // Red
        };
    }

    public function fieldsAreLocked(): bool
    {
```

```
        return in_array($this, [self::HARVEST, self::KILL]);
    }
}
```

GateType.php

```php
```

```php
<?php

namespace App\Enums;

enum GateType: string
{
    case G1_BASIC_INFO = 'G1_BASIC_INFO';
    case G2_IMAGES = 'G2_IMAGES';
    case G3_SEO = 'G3_SEO';
    case G4_VECTOR = 'G4_VECTOR';
    case G5_TECHNICAL = 'G5_TECHNICAL';
    case G6_COMMERCIAL = 'G6_COMMERCIAL';
    case G7_EXPERT = 'G7_EXPERT';

    public function displayName(): string
    {
        return match($this) {
            self::G1_BASIC_INFO => 'G1 - Basic Information',
            self::G2_IMAGES => 'G2 - Images',
            self::G3_SEO => 'G3 - SEO Metadata',
            self::G4_VECTOR => 'G4 - Semantic Validation',
            self::G5_TECHNICAL => 'G5 - Technical Specifications',
            self::G6_COMMERCIAL => 'G6 - Commercial Data',
            self::G7_EXPERT => 'G7 - Expert Authority',
        };
    }

    public function isBlockingForTier(TierType $tier): bool
    {
        // G1-G6 always blocking
        if ($this !== self::G7_EXPERT) {
            return true;
        }

        // G7 only blocking for Hero and Support
        return in_array($tier, [TierType::HERO, TierType::SUPPORT]);
    }
}
```

ValidationStatus.php

```php
```

```php
<?php

namespace App\Enums;

enum ValidationStatus: string
{
    case DRAFT = 'DRAFT';
    case PENDING = 'PENDING';
    case VALID = 'VALID';
    case INVALID = 'INVALID';
    case DEGRADED = 'DEGRADED';  // Fail-soft state

    public function canPublish(): bool
    {
        return $this === self::VALID;
    }

    public function color(): string
    {
        return match($this) {
            self::DRAFT => '#9CA3AF',
            self::PENDING => '#F59E0B',
            self::VALID => '#10B981',
            self::INVALID => '#EF4444',
            self::DEGRADED => '#F97316',
        };
    }
}
```

---

3. Vector Validation Integration

Python Vector Module (Your Earlier Code)

backend/python/src/vector/cluster_cache.py

```python
# YOUR EARLIER CODE - Already implemented
# See the code you provided at the beginning
```

backend/python/src/vector/embedding.py

```python
# YOUR EARLIER CODE - Already implemented
# See the code you provided at the beginning
```

backend/python/src/vector/validation.py

```python
```

```
from typing import Optional
import logging
from . import cluster_cache
from . import embedding

logger = logging.getLogger(__name__)

SIMILARITY_THRESHOLD = 0.72

def validate_cluster_match(request_vector: list[float], cluster_id: str) -> dict:
    """
    Validate that request vector matches cluster centroid within threshold.
    Returns {'valid': bool, 'similarity': float, 'reason': str}
    Logs every check for audit purposes.
    """
    cluster_vec = cluster_cache.get_cluster_vector(cluster_id)
    if not cluster_vec:
        logger.warning("Validation blocked: cluster %s has no vector", cluster_id)
        return {
            'valid': False,
            'similarity': 0.0,
            'reason': f'Cluster {cluster_id} vectors not initialized. Wait for SEO Governor sync or contact support.'
        }

    try:
        similarity = embedding.cosine_similarity(request_vector, cluster_vec)
    except ValueError as e:
        logger.error("Vector validation error for cluster %s: %s", cluster_id, e)
        return {
            'valid': False,
            'similarity': 0.0,
            'reason': f'Vector dimension mismatch. Expected {embedding.DIMENSIONS} dimensions.'
        }

    # AUDIT LOG - every check gets logged
    logger.info(
        "AUDIT: cluster_validation cluster_id=%s similarity=%.4f threshold=%.2f valid=%s",
        cluster_id, similarity, SIMILARITY_THRESHOLD, similarity >= SIMILARITY_THRESHOLD
    )

    if similarity < SIMILARITY_THRESHOLD:
        return {
            'valid': False,
```

```
            'similarity': similarity,
            'reason': (
                f'Content semantic mismatch (similarity {similarity:.2f} < threshold {SIMILARITY_THRESHOLD}). '
                f'ACTION: Revise content to match cluster intent, or request cluster reassignment.'
            )
        }

    return {'valid': True, 'similarity': similarity, 'reason': 'Passed validation'}
```

PHP G4 Gate (Calls Python)

backend/php/src/Validators/Gates/G4_VectorGate.php

```php
```

```php
<?php

namespace App\Validators\Gates;

use App\Models\Sku;
use App\Enums\GateType;
use App\Validators\GateResult;
use App\Validators\GateInterface;

class G4_VectorGate implements GateInterface
{
    private const PYTHON_ENDPOINT = 'http://python-worker:5000/validate-vector';

    public function validate(Sku $sku): GateResult
    {
        // Check if SKU has a cluster assigned
        if (!$sku->primary_cluster_id) {
            return new GateResult(
                gate: GateType::G4_VECTOR,
                passed: false,
                reason: 'No cluster assigned. SKU must belong to at least one cluster.',
                blocking: true
            );
        }

        // Check if long_description exists
        if (!$sku->long_description || strlen(trim($sku->long_description)) < 100) {
            return new GateResult(
                gate: GateType::G4_VECTOR,
                passed: false,
                reason: 'Long description missing or too short (minimum 100 characters required for vector validation).',
                blocking: true
            );
        }

        try {
            // Call Python microservice for vector validation
            $response = $this->callPythonValidator($sku->long_description, $sku->primary_cluster_id);

            if ($response['valid']) {
                return new GateResult(
                    gate: GateType::G4_VECTOR,
                    passed: true,
```

```
                reason: sprintf('Semantic match confirmed (similarity: %.2f)', $response['similarity']),
                blocking: false,
                metadata: ['similarity' => $response['similarity']]
            );
        } else {
            return new GateResult(
                gate: GateType::G4_VECTOR,
                passed: false,
                reason: $response['reason'],
                blocking: true,
                metadata: ['similarity' => $response['similarity']]
            );
        }

    } catch (\Exception $e) {
        // Fail-soft: If embedding API times out or errors, allow DEGRADED save
        logger()->warning('G4 Vector validation failed (fail-soft)', [
            'sku_id' => $sku->id,
            'error' => $e->getMessage()
        ]);

        return new GateResult(
            gate: GateType::G4_VECTOR,
            passed: false,
            reason: 'Vector validation temporarily unavailable. Save allowed with DEGRADED status. Validation will retry aut
            blocking: false,  // Not blocking - fail-soft
            metadata: ['degraded' => true, 'error' => $e->getMessage()]
        );
    }
}

private function callPythonValidator(string $description, string $clusterId): array
{
    $client = new \GuzzleHttp\Client(['timeout' => 3.0]);  // 3 second timeout

    $response = $client->post(self::PYTHON_ENDPOINT, [
        'json' => [
            'description' => $description,
            'cluster_id' => $clusterId
        ]
    ]);

    return json_decode($response->getBody()->getContents(), true);
```

```
    }
}
```

Python Flask API Endpoint

backend/python/src/api/vector_api.py

```python
```

```python
from flask import Flask, request, jsonify
from src.vector import embedding, validation
import logging

app = Flask(__name__)
logger = logging.getLogger(__name__)

@app.route('/validate-vector', methods=['POST'])
def validate_vector():
    """
    Endpoint called by PHP G4 gate.
    Embeds description and validates against cluster centroid.
    """
    data = request.get_json()
    description = data.get('description')
    cluster_id = data.get('cluster_id')

    if not description or not cluster_id:
        return jsonify({'error': 'Missing description or cluster_id'}), 400

    try:
        # Get embedding for description
        desc_vector = embedding.get_embedding(description)

        # Validate against cluster centroid
        result = validation.validate_cluster_match(desc_vector, cluster_id)

        return jsonify(result), 200

    except embedding.EmbeddingUnavailableError as e:
        logger.warning(f"Embedding API unavailable (fail-soft): {e}")
        return jsonify({
            'valid': False,
            'similarity': 0.0,
            'reason': 'Embedding service temporarily unavailable. Retry queued.',
            'degraded': True
        }), 503  # Service Unavailable
    except Exception as e:
        logger.exception(f"Vector validation error: {e}")
        return jsonify({'error': str(e)}), 500
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

---

4. Validation Gates (G1-G7)

Gate Interface

GateInterface.php

```
php
<?php

namespace App\Validators;

use App\Models\Sku;

interface GateInterface
{
    public function validate(Sku $sku): GateResult;
}
```

GateResult.php

```
php
```

```php
<?php

namespace App\Validators;

use App\Enums\GateType;

class GateResult
{
    public function __construct(
        public GateType $gate,
        public bool $passed,
        public string $reason,
        public bool $blocking = true,
        public array $metadata = []
    ) {}

    public function toArray(): array
    {
        return [
            'gate' => $this->gate->value,
            'gate_name' => $this->gate->displayName(),
            'passed' => $this->passed,
            'reason' => $this->reason,
            'blocking' => $this->blocking,
            'metadata' => $this->metadata
        ];
    }
}
```

Individual Gates

G1_BasicInfoGate.php

```php
```

```php
<?php

namespace App\Validators\Gates;

use App\Models\Sku;
use App\Enums\GateType;
use App\Validators\GateResult;
use App\Validators\GateInterface;

class G1_BasicInfoGate implements GateInterface
{
    public function validate(Sku $sku): GateResult
    {
        $missing = [];

        if (!$sku->sku_code || strlen(trim($sku->sku_code)) === 0) {
            $missing[] = 'SKU code';
        }

        if (!$sku->title || strlen(trim($sku->title)) === 0) {
            $missing[] = 'Title';
        }

        if (!$sku->short_description || strlen(trim($sku->short_description)) < 50) {
            $missing[] = 'Short description (min 50 characters)';
        }

        if (count($missing) > 0) {
            return new GateResult(
                gate: GateType::G1_BASIC_INFO,
                passed: false,
                reason: 'Missing required fields: ' . implode(', ', $missing),
                blocking: true
            );
        }

        return new GateResult(
            gate: GateType::G1_BASIC_INFO,
            passed: true,
            reason: 'All required basic information fields are present',
            blocking: false
        );
```

```
    }
}
```

G2_ImagesGate.php

```php
```

```php
<?php

namespace App\Validators\Gates;

use App\Models\Sku;
use App\Enums\GateType;
use App\Validators\GateResult;
use App\Validators\GateInterface;

class G2_ImagesGate implements GateInterface
{
    public function validate(Sku $sku): GateResult
    {
        // Check for at least one hero image
        if (!$sku->primary_image || !file_exists(storage_path('uploads/images/' . $sku->primary_image))) {
            return new GateResult(
                gate: GateType::G2_IMAGES,
                passed: false,
                reason: 'At least one hero image is required. Upload a primary product image.',
                blocking: true
            );
        }

        // Check image file size (< 5MB)
        $filePath = storage_path('uploads/images/' . $sku->primary_image);
        $fileSize = filesize($filePath);
        if ($fileSize > 5 * 1024 * 1024) {
            return new GateResult(
                gate: GateType::G2_IMAGES,
                passed: false,
                reason: 'Primary image exceeds 5MB limit. Compress or resize the image.',
                blocking: true
            );
        }

        // Check image dimensions (min 800x800)
        $imageInfo = getimagesize($filePath);
        if ($imageInfo[0] < 800 || $imageInfo[1] < 800) {
            return new GateResult(
                gate: GateType::G2_IMAGES,
                passed: false,
                reason: 'Primary image must be at least 800x800 pixels for quality standards.',
                blocking: true
```

```
            );
        }

        return new GateResult(
            gate: GateType::G2_IMAGES,
            passed: true,
            reason: sprintf('Primary image uploaded (%dx%d, %.1f MB)',
                $imageInfo[0], $imageInfo[1], $fileSize / 1024 / 1024),
            blocking: false
        );
    }
}
```

G3_SEOGate.php

```php
```

```php
<?php

namespace App\Validators\Gates;

use App\Models\Sku;
use App\Enums\GateType;
use App\Validators\GateResult;
use App\Validators\GateInterface;

class G3_SEOGate implements GateInterface
{
    private const MAX_META_TITLE = 60;
    private const MAX_META_DESCRIPTION = 160;

    public function validate(Sku $sku): GateResult
    {
        $issues = [];

        // Meta title check
        if (!$sku->meta_title) {
            $issues[] = 'Meta title is missing';
        } elseif (strlen($sku->meta_title) > self::MAX_META_TITLE) {
            $issues[] = sprintf('Meta title too long (%d chars, max %d)',
                strlen($sku->meta_title), self::MAX_META_TITLE);
        }

        // Meta description check
        if (!$sku->meta_description) {
            $issues[] = 'Meta description is missing';
        } elseif (strlen($sku->meta_description) > self::MAX_META_DESCRIPTION) {
            $issues[] = sprintf('Meta description too long (%d chars, max %d)',
                strlen($sku->meta_description), self::MAX_META_DESCRIPTION);
        } elseif (strlen($sku->meta_description) < 50) {
            $issues[] = 'Meta description too short (min 50 characters for effective SEO)';
        }

        if (count($issues) > 0) {
            return new GateResult(
                gate: GateType::G3_SEO,
                passed: false,
                reason: implode('. ', $issues),
                blocking: true
            );
```

```
        }

        return new GateResult(
            gate: GateType::G3_SEO,
            passed: true,
            reason: sprintf('SEO metadata valid (title: %d chars, description: %d chars)',
                strlen($sku->meta_title), strlen($sku->meta_description)),
            blocking: false
        );
    }
}
```

G5_TechnicalGate.php

```php
```

```php
<?php

namespace App\Validators\Gates;

use App\Models\Sku;
use App\Enums\GateType;
use App\Validators\GateResult;
use App\Validators\GateInterface;

class G5_TechnicalGate implements GateInterface
{
    public function validate(Sku $sku): GateResult
    {
        // Get required specs from cluster schema
        $cluster = $sku->primaryCluster;
        if (!$cluster) {
            return new GateResult(
                gate: GateType::G5_TECHNICAL,
                passed: false,
                reason: 'No cluster assigned. Cannot validate technical specs.',
                blocking: true
            );
        }

        $requiredSpecs = $cluster->required_specifications ?? [];
        $skuSpecs = $sku->specifications ?? [];

        $missing = [];
        foreach ($requiredSpecs as $specName) {
            if (!isset($skuSpecs[$specName]) || empty($skuSpecs[$specName])) {
                $missing[] = $specName;
            }
        }

        if (count($missing) > 0) {
            return new GateResult(
                gate: GateType::G5_TECHNICAL,
                passed: false,
                reason: 'Missing required specifications: ' . implode(', ', $missing),
                blocking: true
            );
        }
```

```
        // Validate units (e.g., "10 lbs" not "10 pounds")
        $unitIssues = $this->validateUnits($skuSpecs);
        if (count($unitIssues) > 0) {
            return new GateResult(
                gate: GateType::G5_TECHNICAL,
                passed: false,
                reason: 'Unit format issues: ' . implode(', ', $unitIssues),
                blocking: true
            );
        }

        return new GateResult(
            gate: GateType::G5_TECHNICAL,
            passed: true,
            reason: sprintf('All %d required specifications completed with valid units',
                count($requiredSpecs)),
            blocking: false
        );
    }

    private function validateUnits(array $specs): array
    {
        $issues = [];
        $standardUnits = ['lbs', 'kg', 'oz', 'g', 'in', 'cm', 'ft', 'm', 'mm'];

        foreach ($specs as $name => $value) {
            // Check if value contains measurement
            if (preg_match('/\d+\s*([a-zA-Z]+)/', $value, $matches)) {
                $unit = strtolower($matches[1]);
                if (!in_array($unit, $standardUnits)) {
                    $issues[] = sprintf('%s: use standard units (found "%s")', $name, $unit);
                }
            }
        }

        return $issues;
    }
}
```

G6_CommercialGate.php

```
php
```

```php
<?php

namespace App\Validators\Gates;

use App\Models\Sku;
use App\Enums\GateType;
use App\Validators\GateResult;
use App\Validators\GateInterface;

class G6_CommercialGate implements GateInterface
{
    public function validate(Sku $sku): GateResult
    {
        $missing = [];

        if (!$sku->current_price || $sku->current_price <= 0) {
            $missing[] = 'Valid price';
        }

        if (!isset($sku->margin_percent)) {
            $missing[] = 'Margin data';
        }

        if (!$sku->last_sale_date) {
            $missing[] = 'Last sale date';
        }

        if (count($missing) > 0) {
            return new GateResult(
                gate: GateType::G6_COMMERCIAL,
                passed: false,
                reason: 'Missing ERP data: ' . implode(', ', $missing) . '. Ensure nightly ERP sync has completed.',
                blocking: true
            );
        }

        return new GateResult(
            gate: GateType::G6_COMMERCIAL,
            passed: true,
            reason: sprintf('Commercial data synced (price: $%.2f, margin: %.1f%%)',
                $sku->current_price, $sku->margin_percent),
            blocking: false
        );
```

```
    }
}
```

G7_ExpertGate.php

```php
```

```php
<?php

namespace App\Validators\Gates;

use App\Models\Sku;
use App\Enums\GateType;
use App\Enums\TierType;
use App\Validators\GateResult;
use App\Validators\GateInterface;

class G7_ExpertGate implements GateInterface
{
    public function validate(Sku $sku): GateResult
    {
        // G7 is only blocking for Hero and Support tiers
        $isBlocking = in_array($sku->tier, [TierType::HERO, TierType::SUPPORT]);

        $missing = [];

        if (!$sku->expert_author) {
            $missing[] = 'Expert author';
        }

        if (!$sku->expert_credentials) {
            $missing[] = 'Expert credentials';
        }

        if (!$sku->review_date || strtotime($sku->review_date) < strtotime('-1 year')) {
            $missing[] = 'Recent review (within 1 year)';
        }

        if (count($missing) > 0) {
            return new GateResult(
                gate: GateType::G7_EXPERT,
                passed: false,
                reason: 'Missing expert authority fields: ' . implode(', ', $missing) .
                    ($isBlocking ? ' (REQUIRED for ' . $sku->tier->value . ' tier)' : ' (warning only)'),
                blocking: $isBlocking
            );
        }

        return new GateResult(
            gate: GateType::G7_EXPERT,
```

```
            passed: true,
            reason: sprintf('Expert authority confirmed (author: %s, reviewed: %s)',
                $sku->expert_author, $sku->review_date),
            blocking: false
        );
    }
}
```

Gate Orchestrator

GateValidator.php

php

```php
<?php

namespace App\Validators;

use App\Models\Sku;
use App\Enums\ValidationStatus;
use App\Validators\Gates\*;

class GateValidator
{
    private array $gates = [
        G1_BasicInfoGate::class,
        G2_ImagesGate::class,
        G3_SEOGate::class,
        G4_VectorGate::class,
        G5_TechnicalGate::class,
        G6_CommercialGate::class,
        G7_ExpertGate::class,
    ];

    public function validateAll(Sku $sku): ValidationResponse
    {
        $results = [];
        $overallPassed = true;
        $isDegraded = false;
        $blockingFailure = null;

        foreach ($this->gates as $gateClass) {
            $gate = app($gateClass);
            $result = $gate->validate($sku);

            $results[] = $result;

            // Log the gate check
            \App\Models\ValidationLog::create([
                'sku_id' => $sku->id,
                'gate_type' => $result->gate,
                'passed' => $result->passed,
                'reason' => $result->reason,
                'is_blocking' => $result->blocking,
                'similarity_score' => $result->metadata['similarity'] ?? null,
                'validated_by' => auth()->id()
            ]);
```

```
        if (!$result->passed) {
            $overallPassed = false;

            // Check if this is a degraded state (fail-soft)
            if ($result->metadata['degraded'] ?? false) {
                $isDegraded = true;
            }

            // Track first blocking failure
            if ($result->blocking && !$blockingFailure) {
                $blockingFailure = $result;
            }
        }
    }

    // Determine overall status
    if ($overallPassed) {
        $status = ValidationStatus::VALID;
        $canPublish = true;
        $nextAction = 'SKU is ready for publication';
    } elseif ($isDegraded) {
        $status = ValidationStatus::DEGRADED;
        $canPublish = false;
        $nextAction = 'Save allowed but publication blocked. Validation will retry automatically.';
    } else {
        $status = ValidationStatus::INVALID;
        $canPublish = false;
        $nextAction = $blockingFailure ? $blockingFailure->reason : 'Fix validation errors before publication';
    }

    // Update SKU status
    $sku->update([
        'validation_status' => $status,
        'can_publish' => $canPublish,
        'last_validated_at' => now()
    ]);

    return new ValidationResponse(
        skuId: $sku->id,
        overallStatus: $status,
        canPublish: $canPublish,
        gates: array_map(fn($r) => $r->toArray(), $results),
        nextAction: $nextAction
```

```
        );
    }
}
```

---

5. Tier Calculation Engine

TierCalculationService.php

```php
```

```php
<?php

namespace App\Services;

use App\Models\Sku;
use App\Enums\TierType;
use Illuminate\Support\Collection;

class TierCalculationService
{
    private const PROFITABILITY_THRESHOLD = 5.0; // 5% margin
    private const PERCENTILE_TOP = 20; // Top 20%

    public function recalculateAllTiers(): array
    {
        $allSkus = Sku::where('tier', '!=', TierType::KILL)->get();

        // Calculate percentiles
        $marginPercentile = $this->calculatePercentile($allSkus, 'margin_percent', self::PERCENTILE_TOP);
        $volumePercentile = $this->calculatePercentile($allSkus, 'annual_volume', self::PERCENTILE_TOP);

        $changes = [];

        foreach ($allSkus as $sku) {
            $oldTier = $sku->tier;
            $newTier = $this->calculateTierForSku($sku, $marginPercentile, $volumePercentile);

            if ($oldTier !== $newTier) {
                $this->updateSkuTier($sku, $oldTier, $newTier);
                $changes[] = [
                    'sku_id' => $sku->id,
                    'sku_code' => $sku->sku_code,
                    'old_tier' => $oldTier->value,
                    'new_tier' => $newTier->value,
                    'margin' => $sku->margin_percent,
                    'volume' => $sku->annual_volume
                ];
            }
        }

        return $changes;
    }
```

```php
private function calculateTierForSku(Sku $sku, float $marginPercentile, int $volumePercentile): TierType
{
    // Rule 1: Strategic Hero flag overrides everything
    if ($sku->strategic_hero) {
        return TierType::HERO;
    }

    // Rule 2: Check for KILL criteria first
    if ($this->shouldBeKilled($sku)) {
        return TierType::KILL;
    }

    // Rule 3: Top 20% margin AND volume = HERO
    if ($sku->margin_percent >= $marginPercentile &&
        $sku->annual_volume >= $volumePercentile) {
        return TierType::HERO;
    }

    // Rule 4: Profitable but not top 20% = SUPPORT
    if ($sku->margin_percent >= self::PROFITABILITY_THRESHOLD) {
        return TierType::SUPPORT;
    }

    // Rule 5: Positive margin but low = HARVEST
    if ($sku->margin_percent > 0) {
        return TierType::HARVEST;
    }

    // Default: KILL
    return TierType::KILL;
}

private function shouldBeKilled(Sku $sku): bool
{
    // Negative margin
    if ($sku->margin_percent <= 0) {
        return true;
    }

    // No sales in 90+ days
    if ($sku->last_sale_date &&
        strtotime($sku->last_sale_date) < strtotime('-90 days')) {
        return true;
    }
```

```
        // Zero volume
        if ($sku->annual_volume === 0) {
            return true;
        }

        return false;
    }

    private function calculatePercentile(Collection $skus, string $field, int $percentile): float|int
    {
        $values = $skus->pluck($field)->filter()->sort()->values();
        if ($values->isEmpty()) {
            return 0;
        }

        $index = (int) ceil($values->count() * ((100 - $percentile) / 100));
        return $values[$index] ?? $values->last();
    }

    private function updateSkuTier(Sku $sku, TierType $oldTier, TierType $newTier): void
    {
        $rationale = sprintf(
            'Margin: %.1f%% (top 20%% threshold: %.1f%%), Volume: %d units',
            $sku->margin_percent,
            $this->calculatePercentile(Sku::all(), 'margin_percent', self::PERCENTILE_TOP),
            $sku->annual_volume
        );

        $sku->update([
            'tier' => $newTier,
            'tier_rationale' => $rationale
        ]);

        // Log tier change
        \App\Models\TierHistory::create([
            'sku_id' => $sku->id,
            'old_tier' => $oldTier,
            'new_tier' => $newTier,
            'reason' => $rationale,
            'margin_percent' => $sku->margin_percent,
            'annual_volume' => $sku->annual_volume,
            'changed_by' => auth()->id()
        ]);
```

```
        logger()->info('Tier changed', [
            'sku_id' => $sku->id,
            'old' => $oldTier->value,
            'new' => $newTier->value
        ]);
    }
}
```

---

6. Tier Lock Middleware

TierLockMiddleware.php

```
php
```

```php
<?php

namespace App\Middleware;

use Closure;
use Illuminate\Http\Request;
use App\Models\Sku;
use App\Enums\TierType;

class TierLockMiddleware
{
    private const LOCKED_FIELDS_HARVEST = [
        'title',
        'short_description',
        'primary_image',
        'gallery_images'
    ];

    private const LOCKED_FIELDS_KILL = [
        'title',
        'short_description',
        'long_description',
        'meta_title',
        'meta_description',
        'primary_image',
        'gallery_images',
        'specifications',
        'expert_author',
        'expert_credentials'
    ];

    public function handle(Request $request, Closure $next)
    {
        // Only check on PUT/PATCH requests (updates)
        if (!in_array($request->method(), ['PUT', 'PATCH'])) {
            return $next($request);
        }

        $skuId = $request->route('id');
        $sku = Sku::findOrFail($skuId);

        $lockedFields = match($sku->tier) {
            TierType::HARVEST => self::LOCKED_FIELDS_HARVEST,
```

```
            TierType::KILL => self::LOCKED_FIELDS_KILL,
            default => []
        };

        if (empty($lockedFields)) {
            return $next($request);
        }

        // Check if any locked fields are being modified
        $attemptedChanges = array_intersect(
            array_keys($request->all()),
            $lockedFields
        );

        if (!empty($attemptedChanges)) {
            return response()->json([
                'error' => 'Field modification not allowed',
                'message' => sprintf(
                    'Fields [%s] are locked for %s tier SKUs. Request tier upgrade to edit these fields.',
                    implode(', ', $attemptedChanges),
                    $sku->tier->value
                ),
                'locked_fields' => $lockedFields,
                'tier' => $sku->tier->value,
                'next_action' => 'Contact Finance to upgrade tier or edit other fields'
            ], 403);
        }

        return $next($request);
    }
}
```

Due to length constraints, I'll continue this in the next file. Would you like me to create the rest of the implementation guide covering:

7. Intent Assignment Logic
8. AI Audit Engine
9. Content Brief Generator
10. ERP Sync Job
11. RBAC Implementation

12. Frontend Components

Let me create a second implementation file with the remaining sections.