CIE v2.3.2 - Implementation Guide (Part 2)

Continued from Part 1...

---

7. Intent Assignment Logic

Intent Assignment Service

IntentAssignmentService.php

```
php
```

```php
<?php

namespace App\Services;

use App\Models\Sku;
use App\Models\Cluster;
use App\Models\SkuIntent;
use Illuminate\Support\Collection;

class IntentAssignmentService
{
    /**
     * Assign clusters to SKU and auto-derive intents
     */
    public function assignClusters(Sku $sku, array $clusterIds): array
    {
        // Validate cluster IDs
        $clusters = Cluster::whereIn('id', $clusterIds)->get();

        if ($clusters->count() !== count($clusterIds)) {
            throw new \InvalidArgumentException('One or more invalid cluster IDs');
        }

        // Clear existing assignments
        $sku->skuIntents()->delete();

        $assignedIntents = [];
        $isPrimary = true;

        foreach ($clusters as $cluster) {
            // Create intent assignment
            $skuIntent = SkuIntent::create([
                'sku_id' => $sku->id,
                'intent_id' => $cluster->primary_intent_id,
                'cluster_id' => $cluster->id,
                'is_primary' => $isPrimary
            ]);

            $assignedIntents[] = [
                'intent' => $cluster->primaryIntent->name,
                'intent_display' => $cluster->primaryIntent->display_name,
                'cluster' => $cluster->name,
                'cluster_id' => $cluster->id,
                'is_primary' => $isPrimary
            ];

            $isPrimary = false; // Only first cluster is primary
        }

        // Update SKU primary cluster
        $sku->update([
            'primary_cluster_id' => $clusters->first()->id
        ]);

        // Log the assignment
        logger()->info('Intents assigned to SKU', [
            'sku_id' => $sku->id,
            'clusters' => $clusterIds,
            'intents' => array_column($assignedIntents, 'intent')
        ]);

        return $assignedIntents;
    }

    /**
     * Validate SKU has at least one intent
     */
    public function validateIntentAssignment(Sku $sku): bool
    {
        return $sku->skuIntents()->count() > 0;
    }

    /**
     * Get all intents for SKU (auto-derived from clusters)
     */
    public function getSkuIntents(Sku $sku): Collection
    {
        return $sku->skuIntents()
            ->with(['intent', 'cluster'])
            ->get()
            ->map(fn($si) => [
                'intent_name' => $si->intent->name,
                'intent_display' => $si->intent->display_name,
                'cluster_name' => $si->cluster->name,
                'is_primary' => $si->is_primary,
                'assigned_at' => $si->assigned_at
            ]);
    }
}
```

Cluster Controller (SEO Governor Only)

ClusterController.php

```
php
```

```php
<?php

namespace App\Controllers;

use App\Models\Cluster;
use App\Enums\RoleType;
use Illuminate\Http\Request;

class ClusterController extends Controller
{
    public function __construct()
    {
        // Only SEO_GOVERNOR and ADMIN can modify clusters
        $this->middleware('rbac:' . RoleType::SEO_GOVERNOR->value . ',' . RoleType::ADMIN->value)
            ->except(['index', 'show']);
    }

    public function store(Request $request)
    {
        $validated = $request->validate([
            'name' => 'required|string|max:255',
            'intent_statement' => 'required|string|min:100',
            'primary_intent_id' => 'required|uuid|exists:intents,id'
        ]);

        $cluster = Cluster::create([
            ...$validated,
            'created_by' => auth()->id(),
            'approval_status' => 'PENDING' // Requires approval workflow
        ]);

        // Generate centroid vector from intent statement
        $this->generateCentroidVector($cluster);

        return response()->json([
            'message' => 'Cluster created (pending approval)',
            'cluster' => $cluster
        ], 201);
    }

    public function update(Request $request, string $id)
    {
        $cluster = Cluster::findOrFail($id);
        if ($cluster->is_locked) {
            return response()->json([
                'error' => 'Cluster is locked',
                'message' => 'This cluster is locked and cannot be modified. Contact admin to unlock.'
            ], 403);
        }

        $validated = $request->validate([
            'name' => 'sometimes|string|max:255',
            'intent_statement' => 'sometimes|string|min:100',
        ]);

        $cluster->update($validated);

        // If intent statement changed, regenerate vector
        if (isset($validated['intent_statement'])) {
            $this->generateCentroidVector($cluster);
        }

        return response()->json([
            'message' => 'Cluster updated',
            'cluster' => $cluster
        ]);
    }

    private function generateCentroidVector(Cluster $cluster): void
    {
        // Call Python vector service to embed intent statement
        $client = new \GuzzleHttp\Client();
        $response = $client->post(env('PYTHON_VECTOR_SERVICE') . '/embed', [
            'json' => ['text' => $cluster->intent_statement]
        ]);

        $data = json_decode($response->getBody()->getContents(), true);
        $vector = $data['vector'];

        // Store in database as JSON
        $cluster->update([
            'centroid_vector' => $vector,
            'last_vector_update' => now()
        ]);

        // Also cache in Redis (calls your Python cluster_cache.py)
        $client->post(env('PYTHON_VECTOR_SERVICE') . '/cache-vector', [
            'json' => [
                'cluster_id' => $cluster->id,
                'vector' => $vector
            ]
        ]);

        logger()->info('Cluster vector generated', [
            'cluster_id' => $cluster->id,
            'dimensions' => count($vector)
        ]);
    }
}
```

---

8. AI Audit Engine

Multi-Engine Orchestrator

backend/python/src/ai_audit/audit_engine.py

```
python
```

```
import asyncio
import logging
from typing import Dict, List, Optional
from datetime import datetime
from .engines import PerplexityEngine, OpenAIEngine, AnthropicEngine, GeminiEngine

logger = logging.getLogger(__name__)

class AuditEngine:
    """
    Orchestrates multi-engine AI citation audits.
    Queries 4 engines in parallel, handles timeouts, returns aggregated score.
    """

    def __init__(self):
        self.engines = [
            PerplexityEngine(),
            OpenAIEngine(),
            AnthropicEngine(),
            GeminiEngine()
        ]

    async def audit_sku(self, sku_title: str, description: str) -> Dict:
        """
        Run citation audit across all engines.
        Returns: {
            'scores': {engine: score},
            'avg_score': float,
            'engines_succeeded': int,
            'status': 'SUCCESS|PARTIAL|FAILED'
        }
        """
        prompt = f"""Does your training data contain information about: {sku_title}.

Description: {description}

Respond with a citation score 0-100 where:
- 0 = no citations found in training data
- 50 = some mentions but not extensively cited
- 100 = extensively cited across multiple sources

Respond with ONLY a number 0-100, no explanation."""
```

```python
        # Query all engines in parallel
        tasks = [self._query_engine(engine, prompt) for engine in self.engines]
        results = await asyncio.gather(*tasks, return_exceptions=True)

        scores = {}
        engines_succeeded = 0

        for engine, result in zip(self.engines, results):
            engine_name = engine.__class__.__name__.replace('Engine', '').upper()

            if isinstance(result, Exception):
                logger.warning(f"{engine_name} failed: {result}")
                scores[engine_name] = {'score': None, 'status': 'ERROR', 'error': str(result)}
            elif result['status'] == 'TIMEOUT':
                logger.warning(f"{engine_name} timed out")
                scores[engine_name] = {'score': None, 'status': 'TIMEOUT'}
            else:
                scores[engine_name] = result
                engines_succeeded += 1

        # Calculate average (only from successful engines)
        successful_scores = [s['score'] for s in scores.values() if s['score'] is not None]

        if len(successful_scores) >= 3:  # Require at least 3 engines
            avg_score = sum(successful_scores) / len(successful_scores)
            status = 'SUCCESS'
        elif len(successful_scores) > 0:
            avg_score = sum(successful_scores) / len(successful_scores)
            status = 'PARTIAL'
        else:
            avg_score = None
            status = 'FAILED'

        return {
            'scores': scores,
            'avg_score': avg_score,
            'engines_succeeded': engines_succeeded,
            'status': status,
            'timestamp': datetime.utcnow().isoformat()
        }

    async def _query_engine(self, engine, prompt: str) -> Dict:
        """Query single engine with 3s timeout"""
        try:
            result = await asyncio.wait_for(
                engine.query(prompt),
                timeout=3.0
            )
            return result
        except asyncio.TimeoutError:
            return {'score': None, 'status': 'TIMEOUT'}
        except Exception as e:
            logger.exception(f"Engine {engine.__class__.__name__} error: {e}")
            return {'score': None, 'status': 'ERROR', 'error': str(e)}
```

Individual Engine Implementations

backend/python/src/ai_audit/engines/openai_engine.py

```
python
```

```
import os
import re
from openai import AsyncOpenAI

class OpenAIEngine:
    def __init__(self):
        self.client = AsyncOpenAI(api_key=os.getenv('OPENAI_API_KEY'))

    async def query(self, prompt: str) -> dict:
        response = await self.client.chat.completions.create(
            model="gpt-4-turbo-preview",
            messages=[
                {"role": "system", "content": "You are a citation analyzer. Respond only with a number 0-100."},
                {"role": "user", "content": prompt}
            ],
            max_tokens=10,
            temperature=0
        )

        text = response.choices[0].message.content.strip()
        score = self._parse_score(text)

        return {
            'score': score,
            'status': 'SUCCESS',
            'raw_response': text
        }

    def _parse_score(self, text: str) -> int:
        """Extract number from response"""
        match = re.search(r'\d+', text)
        if match:
            score = int(match.group())
            return max(0, min(100, score))  # Clamp to 0-100
        return 0
```

backend/python/src/ai_audit/engines/anthropic_engine.py

```
python
```

```
import os
import re
from anthropic import AsyncAnthropic

class AnthropicEngine:
    def __init__(self):
        self.client = AsyncAnthropic(api_key=os.getenv('ANTHROPIC_API_KEY'))

    async def query(self, prompt: str) -> dict:
        message = await self.client.messages.create(
            model="claude-3-5-sonnet-20241022",
            max_tokens=10,
            messages=[{"role": "user", "content": prompt}]
        )

        text = message.content[0].text.strip()
        score = self._parse_score(text)

        return {
            'score': score,
            'status': 'SUCCESS',
            'raw_response': text
        }

    def _parse_score(self, text: str) -> int:
        match = re.search(r'\d+', text)
        if match:
            score = int(match.group())
            return max(0, min(100, score))
        return 0
```

Decay Detector

backend/python/src/ai_audit/decay_detector.py

```
python
```

```python
from typing import List, Dict
from datetime import datetime, timedelta
import logging

logger = logging.getLogger(__name__)

class DecayDetector:
    """
    Detects Hero SKUs with 3+ consecutive weeks of avg_score < 50
    """

    DECAY_THRESHOLD = 50
    CONSECUTIVE_WEEKS = 3

    def __init__(self, db_connection):
        self.db = db_connection

    def find_decaying_skus(self) -> List[Dict]:
        """
        Returns list of Hero SKUs that need content refresh briefs
        """
        three_weeks_ago = datetime.now() - timedelta(weeks=3)

        # Query for Hero SKUs with recent audit results
        query = """
        SELECT
            s.id,
            s.sku_code,
            s.title,
            s.tier,
            AVG(ar.score) as avg_score,
            DATE(ar.queried_at) as audit_date
        FROM skus s
        JOIN audit_results ar ON s.id = ar.sku_id
        WHERE s.tier = 'HERO'
          AND ar.queried_at >= %s
          AND ar.status = 'SUCCESS'
        GROUP BY s.id, DATE(ar.queried_at)
        HAVING avg_score < %s
        ORDER BY s.id, audit_date
        """

        results = self.db.execute(query, (three_weeks_ago, self.DECAY_THRESHOLD))
        # Group by SKU and check for consecutive weeks
        sku_audits = {}
        for row in results:
            sku_id = row['id']
            if sku_id not in sku_audits:
                sku_audits[sku_id] = {
                    'sku_code': row['sku_code'],
                    'title': row['title'],
                    'tier': row['tier'],
                    'audit_dates': []
                }
            sku_audits[sku_id]['audit_dates'].append(row['audit_date'])

        # Check for 3 consecutive weeks
        decaying_skus = []
        for sku_id, data in sku_audits.items():
            if self._has_consecutive_weeks(data['audit_dates'], self.CONSECUTIVE_WEEKS):
                logger.info(f"Decay detected: SKU {data['sku_code']} has {self.CONSECUTIVE_WEEKS}+ weeks below thresho
                decaying_skus.append({
                    'sku_id': sku_id,
                    'sku_code': data['sku_code'],
                    'title': data['title'],
                    'consecutive_low_weeks': len(data['audit_dates'])
                })

        return decaying_skus

    def _has_consecutive_weeks(self, dates: List, required_weeks: int) -> bool:
        """Check if dates represent consecutive weeks"""
        if len(dates) < required_weeks:
            return False

        sorted_dates = sorted(dates)
        consecutive = 1

        for i in range(1, len(sorted_dates)):
            days_diff = (sorted_dates[i] - sorted_dates[i-1]).days
            if 5 <= days_diff <= 9:  # Allow 5-9 days between (weekly audits)
                consecutive += 1
                if consecutive >= required_weeks:
                    return True
            else:
                consecutive = 1
        return False
```

---

9. Content Brief Generator

backend/python/src/brief_generator/generator.py

```python
```

```python
from typing import Dict, List
from datetime import datetime, timedelta
import logging

logger = logging.getLogger(__name__)

class BriefGenerator:
    """
    Auto-generates content refresh briefs for decaying Hero SKUs
    """

    def __init__(self, db_connection):
        self.db = db_connection

    def generate_decay_brief(self, sku_id: str, sku_code: str, title: str) -> Dict:
        """
        Create content refresh brief for a decaying SKU
        """
        # Check if brief already exists
        existing = self.db.execute(
            "SELECT id FROM content_briefs WHERE sku_id = %s AND status IN ('OPEN', 'IN_PROGRESS')",
            (sku_id,)
        )

        if existing:
            logger.info(f"Brief already exists for SKU {sku_code}, skipping")
            return None

        # Get current content
        sku_data = self.db.execute(
            "SELECT long_description, specifications FROM skus WHERE id = %s",
            (sku_id,)
        ).fetchone()

        # Generate suggested actions
        suggested_actions = [
            "Add recent customer testimonials or reviews",
            "Update technical specifications if product has been revised",
            "Include new use case examples from recent support tickets",
            "Add FAQ section addressing common questions",
            "Update comparison data against competing products",
            "Include updated installation or setup guidance"
        ]
        # Calculate deadline (14 days from now)
        deadline = datetime.now() + timedelta(days=14)

        # Get Content Lead user ID
        content_lead = self.db.execute(
            """SELECT u.id FROM users u
               JOIN user_roles ur ON u.id = ur.user_id
               JOIN roles r ON ur.role_id = r.id
               WHERE r.name = 'CONTENT_LEAD' AND u.is_active = true
               LIMIT 1"""
        ).fetchone()

        brief_data = {
            'sku_id': sku_id,
            'brief_type': 'DECAY_REFRESH',
            'priority': 'HIGH',
            'title': f'Content Refresh: {title}',
            'description': f"""AI audit scores have been below 50% for 3 consecutive weeks for this Hero SKU.

Current description may be stale, generic, or missing recent updates. Rewrite with:
- Fresh examples and use cases
- Updated technical details
- Recent customer feedback integration
- Current market positioning""",
            'current_content': sku_data['long_description'],
            'suggested_actions': suggested_actions,
            'status': 'OPEN',
            'assigned_to': content_lead['id'] if content_lead else None,
            'deadline': deadline.strftime('%Y-%m-%d'),
            'effort_estimate_hours': 3.0
        }

        # Insert brief
        brief_id = self.db.execute(
            """INSERT INTO content_briefs
               (id, sku_id, brief_type, priority, title, description, current_content,
                suggested_actions, status, assigned_to, deadline, effort_estimate_hours)
               VALUES (UUID(), %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
               RETURNING id""",
            (
                brief_data['sku_id'],
                brief_data['brief_type'],
                brief_data['priority'],
            brief_data['title'],
            brief_data['description'],
            brief_data['current_content'],
            brief_data['suggested_actions'],
            brief_data['status'],
            brief_data['assigned_to'],
            brief_data['deadline'],
            brief_data['effort_estimate_hours']
        )
    ).fetchone()['id']

    logger.info(f"Brief {brief_id} created for SKU {sku_code}")

    # Send email notification
    self._send_notification(brief_data)

    return {
        'brief_id': brief_id,
        **brief_data
    }

def _send_notification(self, brief_data: Dict):
    """Send email to assigned Content Lead"""
    # Implementation depends on your email service
    logger.info(f"Sending brief notification for SKU {brief_data['sku_id']}")
    # TODO: Integrate with email service
```

---

10. ERP Sync Job

backend/python/src/erp_sync/sync_job.py

```python
```

```
import logging
from datetime import datetime
from typing import List, Dict
from .connectors import ODBCConnector, RESTConnector, CSVConnector

logger = logging.getLogger(__name__)

class ERPSyncJob:
    """
    Nightly job to sync pricing, margin, and volume data from ERP
    """

    def __init__(self, db_connection, connector_type='odbc'):
        self.db = db_connection
        self.connector = self._get_connector(connector_type)

    def _get_connector(self, connector_type: str):
        if connector_type == 'odbc':
            return ODBCConnector()
        elif connector_type == 'rest':
            return RESTConnector()
        elif connector_type == 'csv':
            return CSVConnector()
        else:
            raise ValueError(f"Unknown connector type: {connector_type}")

    def run(self) -> Dict:
        """
        Main sync process
        """
        logger.info("Starting ERP sync job")
        start_time = datetime.now()

        stats = {
            'total_records': 0,
            'updated': 0,
            'skipped': 0,
            'errors': 0,
            'tier_changes': []
        }

        try:
            # Extract data from ERP
            erp_data = self.connector.fetch_sku_data()
            stats['total_records'] = len(erp_data)

            logger.info(f"Fetched {len(erp_data)} records from ERP")

            # Process each SKU
            for record in erp_data:
                try:
                    self._process_sku_record(record, stats)
                except Exception as e:
                    logger.error(f"Error processing SKU {record.get('sku_code')}: {e}")
                    stats['errors'] += 1

            # Recalculate tiers for updated SKUs
            tier_changes = self._recalculate_tiers()
            stats['tier_changes'] = tier_changes

            # Log sync completion
            duration = (datetime.now() - start_time).total_seconds()
            self._log_sync_result(stats, duration)

            logger.info(f"ERP sync completed in {duration}s: {stats['updated']} updated, {stats['errors']} errors")

            return stats

        except Exception as e:
            logger.exception(f"ERP sync failed: {e}")
            raise

    def _process_sku_record(self, record: Dict, stats: Dict):
        """
        Update single SKU with ERP data
        """
        sku_code = record['sku_code']

        # Find SKU in CIE database
        sku = self.db.execute(
            "SELECT id, current_price, margin_percent, annual_volume FROM skus WHERE sku_code = %s",
            (sku_code,)
        ).fetchone()

        if not sku:
            logger.warning(f"SKU {sku_code} not found in CIE database, skipping")
            stats['skipped'] += 1
            return

        # Check if data changed
        if (sku['current_price'] == record['price'] and
            sku['margin_percent'] == record['margin'] and
            sku['annual_volume'] == record['volume']):
            stats['skipped'] += 1
            return

        # Update SKU
        self.db.execute(
            """UPDATE skus SET
               current_price = %s,
               cost = %s,
               margin_percent = %s,
               annual_volume = %s,
               last_sale_date = %s,
               updated_at = NOW()
               WHERE id = %s""",
            (
                record['price'],
                record['cost'],
                record['margin'],
                record['volume'],
                record['last_sale_date'],
                sku['id']
            )
        )

        stats['updated'] += 1
        logger.debug(f"Updated SKU {sku_code}: price=${record['price']}, margin={record['margin']}%")

    def _recalculate_tiers(self) -> List[Dict]:
        """
        Trigger tier recalculation via PHP service
        """
        import requests

        response = requests.post(
            'http://php-api:8080/api/v1/tiers/recalculate',
            headers={'Authorization': f"Bearer {self._get_service_token()}"}
        )

        if response.status_code == 200:
            return response.json()['changes']
        else:
            logger.error(f"Tier recalculation failed: {response.text}")
            return []

    def _get_service_token(self) -> str:
        """Get service account JWT token"""
        # Implementation depends on your auth system
        return "service_token_here"

    def _log_sync_result(self, stats: Dict, duration: float):
        """Log sync results to erp_sync_log table"""
        self.db.execute(
            """INSERT INTO erp_sync_log
               (id, total_records, updated, skipped, errors, tier_changes_count, duration_seconds, synced_at)
               VALUES (UUID(), %s, %s, %s, %s, %s, %s, NOW())""",
            (
                stats['total_records'],
                stats['updated'],
                stats['skipped'],
                stats['errors'],
                len(stats['tier_changes']),
                duration
            )
        )
```

ODBC Connector Example

backend/python/src/erp_sync/connectors/odbc_connector.py

```
python
```

```
import pyodbc
import os
from typing import List, Dict

class ODBCConnector:
    """
    Connect to ERP via ODBC (e.g., SQL Server, Oracle)
    """


    def __init__(self):
        self.connection_string = os.getenv('ERP_CONNECTION_STRING')

    def fetch_sku_data(self) -> List[Dict]:
        """
        Fetch SKU data from ERP database
        """
        conn = pyodbc.connect(self.connection_string)
        cursor = conn.cursor()

        # Query ERP tables
        # Adjust SQL to match your ERP schema
        query = """
        SELECT
            p.ProductCode as sku_code,
            p.CurrentPrice as price,
            p.Cost as cost,
            (p.CurrentPrice - p.Cost) / p.CurrentPrice * 100 as margin,
            COALESCE(s.AnnualVolume, 0) as volume,
            s.LastSaleDate as last_sale_date
        FROM Products p
        LEFT JOIN SalesData s ON p.ProductCode = s.ProductCode
        WHERE p.IsActive = 1
          AND p.LastModified >= DATEADD(day, -1, GETDATE())  -- Incremental sync
        """

        cursor.execute(query)

        results = []
        for row in cursor.fetchall():
            results.append({
                'sku_code': row.sku_code,
                'price': float(row.price),
                'cost': float(row.cost),
            'margin': float(row.margin),
            'volume': int(row.volume),
            'last_sale_date': row.last_sale_date.strftime('%Y-%m-%d') if row.last_sale_date else None
        })

    conn.close()
    return results
```

---

11. RBAC Implementation

RBACMiddleware.php

```
php
```

```php
<?php

namespace App\Middleware;

use Closure;
use Illuminate\Http\Request;
use App\Enums\RoleType;

class RBACMiddleware
{
    /**
     * Check if user has required role(s)
     * Usage: ->middleware('rbac:ADMIN,FINANCE')
     */
    public function handle(Request $request, Closure $next, ...$allowedRoles)
    {
        if (!auth()->check()) {
            return response()->json(['error' => 'Unauthenticated'], 401);
        }

        $user = auth()->user();
        $userRoles = $user->roles->pluck('name')->toArray();

        // Check if user has any of the allowed roles
        $hasPermission = !empty(array_intersect($userRoles, $allowedRoles));

        if (!$hasPermission) {
            return response()->json([
                'error' => 'Forbidden',
                'message' => sprintf(
                    'This action requires one of these roles: %s. Your roles: %s',
                    implode(', ', $allowedRoles),
                    implode(', ', $userRoles)
                ),
                'required_roles' => $allowedRoles,
                'your_roles' => $userRoles
            ], 403);
        }

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

Role-Based Route Protection

routes/api.php

```
php
```

```php
<?php

use App\Controllers\*;
use App\Enums\RoleType;

// Public routes
Route::post('/auth/login', [AuthController::class, 'login']);

// Protected routes (all require auth)
Route::middleware('auth')->group(function () {

    // SKU routes (different permissions per action)
    Route::get('/skus', [SkuController::class, 'index']);  // All roles
    Route::get('/skus/{id}', [SkuController::class, 'show']);  // All roles

    Route::post('/skus', [SkuController::class, 'store'])
        ->middleware('rbac:' . RoleType::CONTENT_EDITOR->value . ',' . RoleType::CONTENT_LEAD->value . ',' . RoleTyp

    Route::put('/skus/{id}', [SkuController::class, 'update'])
        ->middleware(['rbac:' . RoleType::CONTENT_EDITOR->value . ',' . RoleType::CONTENT_LEAD->value . ',' . RoleTy

    // Validation (all authenticated users can validate)
    Route::post('/skus/{id}/validate', [ValidationController::class, 'validate']);

    // Tier management (FINANCE + ADMIN only)
    Route::post('/tiers/recalculate', [TierController::class, 'recalculate'])
        ->middleware('rbac:' . RoleType::FINANCE->value . ',' . RoleType::ADMIN->value);

    // Cluster management (SEO_GOVERNOR + ADMIN only)
    Route::post('/clusters', [ClusterController::class, 'store'])
        ->middleware('rbac:' . RoleType::SEO_GOVERNOR->value . ',' . RoleType::ADMIN->value);

    Route::put('/clusters/{id}', [ClusterController::class, 'update'])
        ->middleware('rbac:' . RoleType::SEO_GOVERNOR->value . ',' . RoleType::ADMIN->value);

    // AI Audit (AI_OPS + ADMIN)
    Route::post('/audit/{sku_id}', [AuditController::class, 'runAudit'])
        ->middleware('rbac:' . RoleType::AI_OPS->value . ',' . RoleType::ADMIN->value);

    // Content Briefs (CONTENT_LEAD + ADMIN)
    Route::get('/briefs', [BriefController::class, 'index']);  // All can view

    Route::post('/briefs', [BriefController::class, 'store'])
```

```
        ->middleware('rbac:' . RoleType::CONTENT_LEAD->value . ',' . RoleType::ADMIN->value);

    Route::put('/briefs/{id}/assign', [BriefController::class, 'assign'])
        ->middleware('rbac:' . RoleType::CONTENT_LEAD->value . ',' . RoleType::ADMIN->value);
});
```

---

12. Frontend Components

SKU Edit Form with Tier Badges

frontend/src/components/sku/SkuEditForm.jsx

```
jsx
```

```
import React, { useState, useEffect } from 'react';
import { useForm } from 'react-hook-form';
import { TierBadge } from './TierBadge';
import { TierLockBanner } from './TierLockBanner';
import { ValidationPanel } from './ValidationPanel';
import { ImageUploader } from './ImageUploader';
import { skuService, validationService } from '../../services';
import { useTierLock } from '../../hooks/useTierLock';

export function SkuEditForm({ skuId }) {
  const [sku, setSku] = useState(null);
  const [validationResults, setValidationResults] = useState(null);
  const [isSaving, setIsSaving] = useState(false);
  const [isValidating, setIsValidating] = useState(false);

  const { register, handleSubmit, formState: { errors }, watch } = useForm();
  const { isFieldLocked, getLockedFields } = useTierLock(sku?.tier);

  useEffect(() => {
    loadSku();
  }, [skuId]);

  const loadSku = async () => {
    const data = await skuService.get(skuId);
    setSku(data);
  };

  const onSaveDraft = async (data) => {
    setIsSaving(true);
    try {
      await skuService.update(skuId, data);
      alert('Draft saved successfully');
    } catch (error) {
      if (error.response?.status === 403) {
        alert(error.response.data.message);  // Tier lock error
      }
    } finally {
      setIsSaving(false);
    }
  };

  const onRunValidation = async () => {
    setIsValidating(true);
    try {
      const results = await validationService.validate(skuId);
      setValidationResults(results);
    } finally {
      setIsValidating(false);
    }
  };

  const onSubmitForPublication = async (data) => {
    // First validate
    await onRunValidation();

    // Check if can publish
    const results = await validationService.validate(skuId);
    if (!results.can_publish) {
      alert('Cannot publish: validation failed. See results below.');
      return;
    }

    // Publish
    await skuService.publish(skuId);
    alert('SKU published successfully!');
  };

  if (!sku) return <div>Loading...</div>;

  return (
    <div className="sku-edit-form">
      {/* Header with tier badge */}
      <div className="header">
        <h1>{sku.sku_code} - {sku.title}</h1>
        <TierBadge tier={sku.tier} />
      </div>

      {/* Tier lock warning banner */}
      {(sku.tier === 'HARVEST' || sku.tier === 'KILL') && (
        <TierLockBanner tier={sku.tier} lockedFields={getLockedFields()} />
      )}

      <form onSubmit={handleSubmit(onSaveDraft)}>
        {/* Basic Info Section */}
        <section className="form-section">
          <h2>Basic Information</h2>
  <div className="form-field">
    <label>SKU Code *</label>
    <input type="text" value={sku.sku_code} disabled />
  </div>

  <div className="form-field">
    <label>Title * {isFieldLocked('title') && '🔒'}</label>
    <input
      type="text"
      {...register('title', { required: true, maxLength: 120 })}
      defaultValue={sku.title}
      disabled={isFieldLocked('title')}
      maxLength={120}
    />
    <CharCounter current={watch('title')?.length || 0} max={120} />
    {errors.title && <span className="error">Title is required (max 120 chars)</span>}
  </div>

  <div className="form-field">
    <label>Short Description * {isFieldLocked('short_description') && '🔒'}</label>
    <textarea
      {...register('short_description', { required: true, maxLength: 300 })}
      defaultValue={sku.short_description}
      disabled={isFieldLocked('short_description')}
      maxLength={300}
      rows={3}
    />
    <CharCounter current={watch('short_description')?.length || 0} max={300} />
  </div>

  <div className="form-field">
    <label>Long Description *</label>
    <textarea
      {...register('long_description', { required: true, minLength: 500 })}
      defaultValue={sku.long_description}
      rows={10}
    />
    <CharCounter current={watch('long_description')?.length || 0} min={500} />
  </div>
</section>

{/* SEO Section */}
<section className="form-section">
  <h2>SEO Metadata</h2>
```

```
      <div className="form-field">
        <label>Meta Title *</label>
        <input
          type="text"
          {...register('meta_title', { required: true, maxLength: 60 })}
          defaultValue={sku.meta_title}
          maxLength={60}
        />
        <CharCounter current={watch('meta_title')?.length || 0} max={60} />
      </div>

      <div className="form-field">
        <label>Meta Description *</label>
        <textarea
          {...register('meta_description', { required: true, maxLength: 160 })}
          defaultValue={sku.meta_description}
          maxLength={160}
          rows={3}
        />
        <CharCounter current={watch('meta_description')?.length || 0} max={160} />
      </div>
    </section>

    {/* Images Section */}
    <section className="form-section">
      <h2>Images {isFieldLocked('primary_image') && '🔒'}</h2>
      <ImageUploader
        skuId={skuId}
        disabled={isFieldLocked('primary_image')}
      />
    </section>

    {/* Action Buttons */}
    <div className="form-actions">
      <button
        type="submit"
        disabled={isSaving}
        className="btn-secondary"
      >
        {isSaving ? 'Saving...' : 'Save Draft'}
      </button>

      <button
```

```
            type="button"
            onClick={onRunValidation}
            disabled={isValidating}
            className="btn-secondary"
          >
            {isValidating ? 'Validating...' : 'Run Validation'}
          </button>

          <button
            type="button"
            onClick={handleSubmit(onSubmitForPublication)}
            disabled={!validationResults?.can_publish}
            className="btn-primary"
          >
            Submit for Publication
          </button>
        </div>
      </form>

      {/* Validation Results Panel */}
      {validationResults && (
        <ValidationPanel results={validationResults} />
      )}
    </div>
  );
}

// Character counter component
function CharCounter({ current, max, min }) {
  const percentage = (current / (max || min)) * 100;
  const color = max
    ? (percentage > 90 ? 'red' : percentage > 70 ? 'orange' : 'green')
    : (current >= min ? 'green' : 'red');

  return (
    <span className={`char-counter ${color}`}>
      {current || 0} / {max || `${min} (min)`}
    </span>
  );
}
```

Tier Badge Component

frontend/src/components/sku/TierBadge.jsx

```jsx
import React from 'react';

export function TierBadge({ tier }) {
  const tierConfig = {
    HERO: { label: 'Hero', color: '#10B981', icon: '⭐' },
    SUPPORT: { label: 'Support', color: '#3B82F6', icon: '🛠️' },
    HARVEST: { label: 'Harvest', color: '#F59E0B', icon: '🌾' },
    KILL: { label: 'Kill', color: '#EF4444', icon: '🚫' }
  };

  const config = tierConfig[tier] || tierConfig.SUPPORT;

  return (
    <span
      className="tier-badge"
      style={{
        backgroundColor: config.color,
        color: 'white',
        padding: '4px 12px',
        borderRadius: '4px',
        fontWeight: 'bold',
        fontSize: '14px'
      }}
    >
      {config.icon} {config.label}
    </span>
  );
}
```

Validation Panel

frontend/src/components/sku/ValidationPanel.jsx

```jsx
```

```
import React from 'react';

export function ValidationPanel({ results }) {
  return (
    <div className="validation-panel">
      <h3>Validation Results</h3>

      <div className={`overall-status ${results.overall_status.toLowerCase()}`}>
        <strong>Status:</strong> {results.overall_status}
        {results.can_publish ? '✅ Ready to publish' : '❌ Cannot publish'}
      </div>

      <div className="gates-list">
        {results.gates.map((gate, index) => (
          <div key={index} className={`gate ${gate.passed ? 'passed' : 'failed'}`}>
            <div className="gate-header">
              <span className="gate-icon">{gate.passed ? '✅' : '❌'}</span>
              <strong>{gate.gate_name}</strong>
              {gate.blocking && !gate.passed && <span className="blocking-badge">BLOCKING</span>}
            </div>
            <div className="gate-reason">{gate.reason}</div>
            {gate.metadata?.similarity && (
              <div className="gate-metadata">
                Similarity: {(gate.metadata.similarity * 100).toFixed(1)}%
              </div>
            )}
          </div>
        ))}
      </div>

      {results.next_action && (
        <div className="next-action">
          <strong>Next Action:</strong> {results.next_action}
        </div>
      )}
    </div>
  );
}
```

13. Cron Job Setup

jobs/nightly_erp_sync.sh

bash

```bash
#!/bin/bash
# Cron: 0 2 * * *  (Runs at 2 AM daily)

set -e

SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
LOG_FILE="/var/log/cie/erp_sync_$(date +%Y%m%d).log"

echo "[$(date)] Starting ERP sync job" >> $LOG_FILE

cd $SCRIPT_DIR/../backend/python

# Activate virtual env
source venv/bin/activate

# Run Python sync job
python -m src.erp_sync.sync_job >> $LOG_FILE 2>&1

if [ $? -eq 0 ]; then
    echo "[$(date)] ERP sync completed successfully" >> $LOG_FILE
else
    echo "[$(date)] ERP sync failed with exit code $?" >> $LOG_FILE
    # Send alert email
    echo "ERP sync failed. Check logs: $LOG_FILE" | mail -s "CIE: ERP Sync Failed" ops@company.com
fi
```

jobs/weekly_decay_check.sh

bash

```
#!/bin/bash
# Cron: 0 3 * * 1  (Runs at 3 AM every Monday)

set -e

SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
LOG_FILE="/var/log/cie/decay_check_$(date +%Y%m%d).log"

echo "[$(date)] Starting decay detection" >> $LOG_FILE

cd $SCRIPT_DIR/../backend/python
source venv/bin/activate

python -m src.ai_audit.decay_detector >> $LOG_FILE 2>&1

echo "[$(date)] Decay check completed" >> $LOG_FILE
```

---

14. Docker Setup

docker-compose.yml

```
yaml
```

```yaml
version: '3.8'

services:
  db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root_password
      MYSQL_DATABASE: cie_v232
      MYSQL_USER: cie_user
      MYSQL_PASSWORD: cie_password
    volumes:
      - db_data:/var/lib/mysql
      - ./database/migrations:/docker-entrypoint-initdb.d
    ports:
      - "3306:3306"

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data

  php-api:
    build:
      context: ./backend/php
      dockerfile: ../../infrastructure/docker/Dockerfile.php
    volumes:
      - ./backend/php:/app
      - ./storage:/app/storage
    environment:
      - DB_HOST=db
      - REDIS_URL=redis://redis:6379/0
    depends_on:
      - db
      - redis
    ports:
      - "9000:9000"

  python-worker:
    build:
      context: ./backend/python
      dockerfile: ../../infrastructure/docker/Dockerfile.python
    volumes:
      - ./backend/python:/app
    environment:
      - DB_HOST=db
      - REDIS_URL=redis://redis:6379/0
      - OPENAI_API_KEY=${OPENAI_API_KEY}
    depends_on:
      - db
      - redis
    ports:
      - "5000:5000"

  nginx:
    image: nginx:alpine
    volumes:
      - ./infrastructure/docker/nginx.conf:/etc/nginx/nginx.conf
      - ./frontend/dist:/usr/share/nginx/html
    ports:
      - "8080:80"
    depends_on:
      - php-api

volumes:
  db_data:
  redis_data:
```

Next Steps

1. **Initialize Database**: Run migrations and seed data
2. **Configure Environment**: Set all API keys in `.env`
3. **Start Services**: `docker-compose up -d`
4. **Run Tests**: `make test`
5. **Deploy**: Follow deployment guide in README.md

For questions, refer to:

- README.md (setup instructions)
- docs/api/openapi.yaml (API reference)
- docs/deployment/runbook.md (operations)

Happy Building! 🚀