Overview

This document provides a comprehensive, step-by-step implementation plan for implementing a **simplified** Product Management System in Willow CMS starting from version tag v1.4.0. The system features a single products table with essential fields, unified tagging across articles and products, and optional article associations for detailed product information.

Prerequisites

- Docker environment running (`./setup_dev_env.sh`)
- Queue worker running (`cake_queue_worker`)
- All development aliases installed (`./setup_dev_aliases.sh`)
- Willow CMS v1.4.0 baseline

Phase 1: Foundation & Database Schema (Week 1-2)

Day 1: Simplified Database Schema Creation

1.1 Create Simplified Products Migration

```
# Create the simplified products migration
docker compose exec willowcms bin/cake bake migration
CreateSimplifiedProducts
```

Migration Content:

`config/Migrations/YYYYMMDD_HHMMSS_CreateSimplifiedProducts.php`

```
<?php
declare(strict_types=1);

use Migrations\AbstractMigration;

class CreateSimplifiedProducts extends AbstractMigration
{
    public function change(): void
    {
        // Create simplified products table
        $table = $this->table('products', [
            'id' => false,
            'primary_key' => ['id'],
        ]);

        $table->addColumn('id', 'uuid', [
            'default' => null,
            'null' => false,
        ])
        ->addColumn('user_id', 'uuid', [
            'default' => null,
            'null' => false,
        ])
        ->addColumn('article_id', 'uuid', [
            'default' => null,
            'null' => true,
            'comment' => 'Optional reference to detailed article'
        ])
        ->addColumn('title', 'string', [
            'default' => null,
            'limit' => 255,
            'null' => false,
        ])
        ->addColumn('slug', 'string', [
            'default' => null,
            'limit' => 191,
            'null' => false,
        ])
        ->addColumn('description', 'text', [
            'default' => null,
            'null' => true,
            'comment' => 'Brief product description'
        ])
```

```
            ->addColumn('manufacturer', 'string', [
                'default' => null,
                'limit' => 255,
                'null' => true,
            ])
            ->addColumn('model_number', 'string', [
                'default' => null,
                'limit' => 255,
                'null' => true,
            ])
            ->addColumn('price', 'decimal', [
                'default' => null,
                'null' => true,
                'precision' => 10,
                'scale' => 2,
            ])
            ->addColumn('currency', 'char', [
                'default' => 'USD',
                'limit' => 3,
                'null' => true,
            ])
            ->addColumn('image', 'string', [
                'default' => null,
                'limit' => 255,
                'null' => true,
                'comment' => 'Primary product image'
            ])
            ->addColumn('alt_text', 'string', [
                'default' => null,
                'limit' => 255,
                'null' => true,
            ])
            ->addColumn('is_published', 'boolean', [
                'default' => false,
                'null' => false,
            ])
            ->addColumn('featured', 'boolean', [
                'default' => false,
                'null' => false,
            ])
            ->addColumn('verification_status', 'string', [
                'default' => 'pending',
                'limit' => 20,
                'null' => false,
            ])
            ->addColumn('reliability_score', 'decimal', [
                'default' => '0.00',
                'null' => true,
                'precision' => 3,
                'scale' => 2,
            ])
            ->addColumn('view_count', 'integer', [
                'default' => 0,
                'null' => false,
            ])
            ->addColumn('created', 'datetime', [
                'default' => null,
                'null' => false,
            ])
            ->addColumn('modified', 'datetime', [
                'default' => null,
                'null' => false,
            ])
            ->addIndex(['slug'], ['unique' => true, 'name' =>
'idx_products_slug'])
            ->addIndex(['user_id'], ['name' => 'idx_products_user'])
            ->addIndex(['article_id'], ['name' => 'idx_products_article'])
            ->addIndex(['is_published'], ['name' => 'idx_products_published'])
            ->addIndex(['featured'], ['name' => 'idx_products_featured'])
            ->addIndex(['verification_status'], ['name' =>
'idx_products_verification'])
            ->addIndex(['manufacturer'], ['name' =>
'idx_products_manufacturer'])
            ->addIndex(['reliability_score'], ['name' =>
'idx_products_reliability'])
            ->addIndex(['created'], ['name' => 'idx_products_created'])
            ->create();

            // Create products_tags junction table for unified tagging
            $tagsTable = $this->table('products_tags', [
                'id' => false,
                'primary_key' => ['product_id', 'tag_id'],
            ]);

            $tagsTable->addColumn('product_id', 'uuid', [
                'default' => null,
                'null' => false,
            ])
            ->addColumn('tag_id', 'uuid', [
                'default' => null,
                'null' => false,
            ])
            ->addIndex(['product_id'], ['name' => 'idx_products_tags_product'])
            ->addIndex(['tag_id'], ['name' => 'idx_products_tags_tag'])
            ->create();
        }
    }
```

```
# Run the migration after editing
docker compose exec willowcms bin/cake migrations migrate
```

1.2 Generate Core Model Structure

```
# Generate the simplified model structure using AdminTheme
docker compose exec willowcms bin/cake bake model Products --theme
AdminTheme

# This creates:
```

```
# - src/Model/Table/ProductsTable.php
# - src/Model/Entity/Product.php
```

1.3 Generate Model Tests

```
# Generate unit tests for the models
docker compose exec willowcms bin/cake bake test table Products
docker compose exec willowcms bin/cake bake test entity Product

# This creates:
# - tests/TestCase/Model/Table/ProductsTableTest.php
# - tests/TestCase/Model/Entity/ProductTest.php
```

1.4 Generate Fixtures for Testing

```
# Generate test fixtures
docker compose exec willowcms bin/cake bake fixture Products

# This creates:
# - tests/Fixture/ProductsFixture.php
```

Day 2: Enhanced Model Relationships & Methods

2.1 Configure Products Model Associations

Update `src/Model/Table/ProductsTable.php`:

```php
<?php
declare(strict_types=1);

namespace App\Model\Table;

use Cake\ORM\Table;
use Cake\Validation\Validator;

class ProductsTable extends Table
{
    public function initialize(array $config): void
    {
        parent::initialize($config);
        $this->setTable('products');
        $this->setDisplayField('title');
        $this->setPrimaryKey('id');

        $this->addBehavior('Timestamp');
        $this->addBehavior('Sluggable', [
            'slug' => 'slug',
            'displayField' => 'title'
        ]);

        // Core relationships
        $this->belongsTo('Users', [
            'foreignKey' => 'user_id',
            'joinType' => 'INNER',
        ]);

        $this->belongsTo('Articles', [
            'foreignKey' => 'article_id',
            'joinType' => 'LEFT',
            'propertyName' => 'article'
        ]);

        // Many-to-many with Tags (unified tagging system)
        $this->belongsToMany('Tags', [
            'foreignKey' => 'product_id',
            'targetForeignKey' => 'tag_id',
            'joinTable' => 'products_tags',
        ]);
    }

    public function validationDefault(Validator $validator): Validator
    {
        $validator
            ->uuid('id')
            ->allowEmptyString('id', null, 'create');

        $validator
            ->scalar('title')
            ->maxLength('title', 255)
            ->requirePresence('title', 'create')
            ->notEmptyString('title');

        $validator
            ->scalar('slug')
            ->maxLength('slug', 191)
            ->requirePresence('slug', 'create')
            ->notEmptyString('slug')
            ->add('slug', 'unique', ['rule' => 'validateUnique', 'provider'
=> 'table']);

        $validator
            ->scalar('description')
            ->allowEmptyString('description');

        $validator
            ->scalar('manufacturer')
            ->maxLength('manufacturer', 255)
            ->allowEmptyString('manufacturer');

        $validator
            ->scalar('model_number')
            ->maxLength('model_number', 255)
            ->allowEmptyString('model_number');

        $validator
            ->decimal('price')
            ->allowEmptyString('price');

        $validator
            ->boolean('is_published')
            ->notEmptyString('is_published');

        $validator
            ->boolean('featured')
            ->notEmptyString('featured');

        return $validator;
    }

    /**
     * Get published products with optional filtering
     */
    public function getPublishedProducts(array $options = []):
\Cake\ORM\Query
    {
        $query = $this->find()
            ->where(['Products.is_published' => true])
            ->contain(['Users', 'Tags', 'Articles'])
            ->order(['Products.created' => 'DESC']);

        // Apply filters
        if (!empty($options['tag'])) {
            $query->matching('Tags', function ($q) use ($options) {
                return $q->where(['Tags.slug' => $options['tag']]);
            });
        }

        if (!empty($options['manufacturer'])) {
            $query->where(['Products.manufacturer LIKE' => '%' .
$options['manufacturer'] . '%']);
        }

        if (!empty($options['featured'])) {
            $query->where(['Products.featured' => true]);
        }

        return $query;
    }

    /**
     * Get products by verification status
     */
    public function getProductsByStatus(string $status): \Cake\ORM\Query
    {
        return $this->find()
            ->where(['verification_status' => $status])
            ->contain(['Users', 'Tags'])
            ->order(['created' => 'ASC']);
    }

    /**
     * Search products across title, description, manufacturer
     */
    public function searchProducts(string $term): \Cake\ORM\Query
    {
        return $this->find()
            ->where([
                'OR' => [
                    'Products.title LIKE' => "%{$term}%",
                    'Products.description LIKE' => "%{$term}%",
                    'Products.manufacturer LIKE' => "%{$term}%",
                    'Products.model_number LIKE' => "%{$term}%"
                ]
            ])
            ->contain(['Tags', 'Users'])
            ->where(['Products.is_published' => true]);
    }

    /**
     * Get products with same tags (for related products)
     */
    public function getRelatedProducts(string $productId, int $limit = 5):
array
    {
        $product = $this->get($productId, ['contain' => ['Tags']]);

        if (empty($product->tags)) {
            return [];
        }

        $tagIds = array_map(fn($tag) => $tag->id, $product->tags);

        return $this->find()
            ->matching('Tags', function ($q) use ($tagIds) {
                return $q->where(['Tags.id IN' => $tagIds]);
            })
            ->where([
                'Products.id !=' => $productId,
                'Products.is_published' => true
            ])
            ->limit($limit)
            ->toArray();
    }

    /**
     * Increment view count
     */
    public function incrementViewCount(string $productId): bool
    {
        return $this->updateAll(
            ['view_count = view_count + 1'],
            ['id' => $productId]
        );
    }
}
```

2.2 Update Articles Table for Product Association

Update `src/Model/Table/ArticlesTable.php` to add the reverse relationship:

```
// Add to initialize() method in ArticlesTable
$this->hasMany('Products', [
    'foreignKey' => 'article_id',
    'dependent' => false, // Don't delete products when article is deleted
]);
```

2.3 Update Tags Table for Products Association

Update `src/Model/Table/TagsTable.php`:

```
// Add to initialize() method in TagsTable
$this->belongsToMany('Products', [
    'foreignKey' => 'tag_id',
    'targetForeignKey' => 'product_id',
    'joinTable' => 'products_tags',
]);
```

Day 3: Slug System Enhancement

3.1 Update Slug Behavior for Products

Create `src/Model/Behavior/SlugBehavior.php` enhancement or update existing:

```
// In the existing SlugBehavior, ensure Products model is supported
// Update the behavior to handle the new 'Product' model type

// Add to the model types array:
protected $supportedModels = [
    'Article',
    'Tag',
    'User',
    'Page',
    'Product' // New addition
];
```

3.2 Update Slugs Table if Needed

If the slugs table needs enhancement, create a migration:

```
docker compose exec willowcms bin/cake bake migration
EnhanceSlugsForProducts
```

Migration Content: `config/Migrations/YYYYMMDD_HHMMSS_EnhanceSlugsForProducts.php`

```
<?php
declare(strict_types=1);

use Migrations\AbstractMigration;

class EnhanceSlugsForProducts extends AbstractMigration
{
    public function change(): void
    {
        // Add any necessary indexes for better performance with Products
        $table = $this->table('slugs');

        // Ensure we have proper composite index for model + foreign_key
lookups
        if (!$table->hasIndex(['model', 'foreign_key'])) {
            $table->addIndex(['model', 'foreign_key'], [
                'name' => 'idx_slugs_model_foreign'
            ]);
        }

        $table->update();
    }
}
```

Day 4: Unified Search Service

4.1 Create Unified Search Service

Create `src/Service/Search/UnifiedSearchService.php`:

```
<?php
declare(strict_types=1);

namespace App\Service\Search;

use Cake\ORM\TableRegistry;
use Cake\Utility\Text;

class UnifiedSearchService
{
    private $articlesTable;
    private $productsTable;
    private $tagsTable;

    public function __construct()
    {
        $this->articlesTable = TableRegistry::getTableLocator()-
>get('Articles');
        $this->productsTable = TableRegistry::getTableLocator()-
>get('Products');
        $this->tagsTable = TableRegistry::getTableLocator()->get('Tags');
    }

    /**
     * Perform unified search across articles, products, and tags
     */
    public function search(string $term, array $options = []): array
    {
        $results = [
            'products' => [],
            'articles' => [],
            'tags' => [],
            'total' => 0
        ];

        $limit = $options['limit'] ?? 10;
        $includeProducts = $options['include_products'] ?? true;
        $includeArticles = $options['include_articles'] ?? true;
        $includeTags = $options['include_tags'] ?? true;

        // Search products
        if ($includeProducts) {
            $products = $this->productsTable->searchProducts($term)
                ->limit($limit)
                ->toArray();

            $results['products'] = array_map(function($product) {
                return [
                    'id' => $product->id,
                    'title' => $product->title,
                    'type' => 'product',
                    'url' => '/products/' . $product->slug,
                    'description' => Text::truncate($product->description ??
'', 150),
                    'image' => $product->image,
                    'manufacturer' => $product->manufacturer,
                    'price' => $product->price,
                    'currency' => $product->currency,
                    'tags' => array_map(fn($tag) => $tag->title, $product-
>tags ?? [])
                ];
            }, $products);
        }

        // Search articles
        if ($includeArticles) {
            $articles = $this->articlesTable->find()
                ->where([
                    'OR' => [
                        'Articles.title LIKE' => "%{$term}%",
                        'Articles.body LIKE' => "%{$term}%",
                        'Articles.lede LIKE' => "%{$term}%"
                    ],
                    'Articles.is_published' => true
                ])
                ->contain(['Tags', 'Users'])
                ->limit($limit)
                ->toArray();

            $results['articles'] = array_map(function($article) {
                return [
                    'id' => $article->id,
                    'title' => $article->title,
                    'type' => 'article',
                    'url' => '/articles/' . $article->slug,
                    'description' => Text::truncate($article->lede ?? '',
150),
                    'author' => $article->user->username ?? '',
                    'created' => $article->created,
                    'tags' => array_map(fn($tag) => $tag->title, $article-
>tags ?? [])
                ];
            }, $articles);
        }

        // Search tags (and include content with those tags)
        if ($includeTags) {
            $tags = $this->tagsTable->find()
                ->where(['Tags.title LIKE' => "%{$term}%"])
                ->contain(['Articles', 'Products'])
                ->limit($limit)
                ->toArray();

            $results['tags'] = array_map(function($tag) {
                return [
                    'id' => $tag->id,
                    'title' => $tag->title,
                    'type' => 'tag',
                    'url' => '/search?tag=' . $tag->slug,
                    'description' => "Tag with " . count($tag->articles ??
[]) . " articles and " . count($tag->products ?? []) . " products",
                    'article_count' => count($tag->articles ?? []),
                    'product_count' => count($tag->products ?? [])
                ];
            }, $tags);
        }

        $results['total'] = count($results['products']) +
count($results['articles']) + count($results['tags']);

        // Sort all results by relevance (simple scoring)
        $allResults = array_merge($results['products'],
$results['articles'], $results['tags']);
        usort($allResults, function($a, $b) use ($term) {
            $scoreA = $this->calculateRelevanceScore($a, $term);
            $scoreB = $this->calculateRelevanceScore($b, $term);
            return $scoreB <=> $scoreA;
        });

        $results['mixed'] = array_slice($allResults, 0, $limit);

        return $results;
    }

    /**
     * Search by tag across all content types
     */
    public function searchByTag(string $tagSlug): array
    {
        $tag = $this->tagsTable->find()
            ->where(['slug' => $tagSlug])
            ->contain([
                'Articles' => function($q) {
                    return $q->where(['is_published' => true])
                        ->contain(['Users'])
                        ->order(['created' => 'DESC']);
                },
                'Products' => function($q) {
                    return $q->where(['is_published' => true])
                        ->contain(['Users'])
                        ->order(['created' => 'DESC']);
                }
            ])
            ->first();

        if (!$tag) {
            return ['tag' => null, 'articles' => [], 'products' => []];
        }

        return [
            'tag' => [
                'id' => $tag->id,
                'title' => $tag->title,
                'slug' => $tag->slug
            ],
            'articles' => $tag->articles ?? [],
            'products' => $tag->products ?? []
        ];
    }
    /**
     * Get search suggestions for autocomplete
     */
    public function getSuggestions(string $term, int $limit = 5): array
    {
        $suggestions = [];

        // Product suggestions
        $products = $this->productsTable->find()
            ->select(['title', 'slug', 'manufacturer'])
            ->where([
                'title LIKE' => "%{$term}%",
                'is_published' => true
            ])
            ->limit($limit)
            ->toArray();

        foreach ($products as $product) {
            $suggestions[] = [
                'text' => $product->title,
                'type' => 'product',
                'url' => '/products/' . $product->slug,
                'subtitle' => $product->manufacturer
            ];
        }

        // Article suggestions
        $articles = $this->articlesTable->find()
            ->select(['title', 'slug'])
            ->where([
                'title LIKE' => "%{$term}%",
                'is_published' => true
            ])
            ->limit($limit)
            ->toArray();

        foreach ($articles as $article) {
            $suggestions[] = [
                'text' => $article->title,
                'type' => 'article',
                'url' => '/articles/' . $article->slug,
                'subtitle' => 'Article'
            ];
        }

        // Tag suggestions
        $tags = $this->tagsTable->find()
            ->select(['title', 'slug'])
            ->where(['title LIKE' => "%{$term}%"])
            ->limit($limit)
            ->toArray();

        foreach ($tags as $tag) {
            $suggestions[] = [
                'text' => $tag->title,
                'type' => 'tag',
                'url' => '/search?tag=' . $tag->slug,
                'subtitle' => 'Tag'
            ];
        }

        return array_slice($suggestions, 0, $limit * 3);
    }

    /**
     * Calculate relevance score for search results
     */
    private function calculateRelevanceScore(array $item, string $term): int
    {
        $score = 0;
        $termLower = strtolower($term);
        $titleLower = strtolower($item['title']);

        // Exact title match gets highest score
        if ($titleLower === $termLower) {
            $score += 100;
        }
        // Title starts with term
        elseif (strpos($titleLower, $termLower) === 0) {
            $score += 80;
        }
        // Title contains term
        elseif (strpos($titleLower, $termLower) !== false) {
            $score += 60;
        }

        // Description contains term
        if (isset($item['description']) &&
strpos(strtolower($item['description']), $termLower) !== false) {
            $score += 20;
        }

        // Boost for certain content types
        if ($item['type'] === 'product') {
            $score += 10; // Slight boost for products
        }

        return $score;
    }
}
```

Day 5: Settings Integration

5.1 Create Product Settings Migration

```
# Create settings migration for product configuration
docker compose exec willowcms bin/cake bake migration
InsertSimplifiedProductSettings
```

```
# Run the migration after editing
docker compose exec willowcms bin/cake migrations migrate
```

Migration Content: `config/Migrations/YYYYMMDD_HHMMSS_InsertSimplifiedProductSettings.php`

```
<?php
declare(strict_types=1);

use Cake\Utility\Text;
use Migrations\AbstractMigration;

class InsertSimplifiedProductSettings extends AbstractMigration
{
    public function change(): void
    {
        $this->table('settings')
            ->insert([
                'id' => Text::uuid(),
                'ordering' => 50,
                'category' => 'Products',
                'key_name' => 'enabled',
                'value' => '1',
                'value_type' => 'bool',
                'value_obscure' => false,
                'description' => 'Enable the products system. When disabled,
products will not be accessible on the frontend.',
                'data' => null,
                'column_width' => 2,
            ])
            ->insert([
                'id' => Text::uuid(),
                'ordering' => 51,
                'category' => 'Products',
                'key_name' => 'userSubmissions',
                'value' => '1',
                'value_type' => 'bool',
                'value_obscure' => false,
                'description' => 'Allow users to submit products for review.
When enabled, registered users can add products that require approval.',
                'data' => null,
                'column_width' => 2,
            ])
            ->insert([
                'id' => Text::uuid(),
                'ordering' => 52,
                'category' => 'Products',
                'key_name' => 'aiVerificationEnabled',
                'value' => '1',
                'value_type' => 'bool',
                'value_obscure' => false,
                'description' => 'Enable AI-powered verification of product
submissions. Uses AI to validate product information and suggest
improvements.',
                'data' => null,
                'column_width' => 2,
            ])
            ->insert([
                'id' => Text::uuid(),
                'ordering' => 53,
                'category' => 'Products',
                'key_name' => 'peerVerificationEnabled',
                'value' => '1',
                'value_type' => 'bool',
                'value_obscure' => false,
                'description' => 'Enable peer verification where users can
verify and rate product accuracy.',
                'data' => null,
                'column_width' => 2,
            ])
            ->insert([
                'id' => Text::uuid(),
                'ordering' => 54,
                'category' => 'Products',
                'key_name' => 'minVerificationScore',
                'value' => '3.0',
                'value_type' => 'numeric',
                'value_obscure' => false,
                'description' => 'Minimum verification score (0-5) required
for automatic approval. Products below this score require manual review.',
                'data' => null,
                'column_width' => 2,
            ])
            ->insert([
                'id' => Text::uuid(),
                'ordering' => 55,
                'category' => 'Products',
                'key_name' => 'autoPublishThreshold',
                'value' => '4.0',
                'value_type' => 'numeric',
                'value_obscure' => false,
                'description' => 'Reliability score threshold for automatic
publishing. Products scoring above this will be automatically published.',
                'data' => null,
                'column_width' => 2,
            ])
            ->insert([
                'id' => Text::uuid(),
                'ordering' => 56,
                'category' => 'Products',
                'key_name' => 'maxUserSubmissionsPerDay',
                'value' => '5',
                'value_type' => 'numeric',
                'value_obscure' => false,
                'description' => 'Maximum number of products a user can
submit per day. Set to 0 for unlimited submissions.',
                'data' => null,
                'column_width' => 2,
            ])
            ->insert([
                'id' => Text::uuid(),
                'ordering' => 57,
                'category' => 'Products',
                'key_name' => 'duplicateDetectionEnabled',
                'value' => '1',
                'value_type' => 'bool',
                'value_obscure' => false,
                'description' => 'Enable duplicate detection to prevent
submission of identical products based on title and manufacturer.',
                'data' => null,
                'column_width' => 2,
            ])
            ->insert([
                'id' => Text::uuid(),
                'ordering' => 58,
                'category' => 'Products',
                'key_name' => 'productImageRequired',
                'value' => '1',
                'value_type' => 'bool',
                'value_obscure' => false,
                'description' => 'Require at least one product image for
publication. Helps maintain visual consistency.',
                'data' => null,
                'column_width' => 2,
            ])
            ->insert([
                'id' => Text::uuid(),
                'ordering' => 59,
                'category' => 'Products',
                'key_name' => 'technicalSpecsRequired',
                'value' => '1',
                'value_type' => 'bool',
                'value_obscure' => false,
                'description' => 'Require basic technical specifications
(description, manufacturer, model) for product approval.',
                'data' => null,
                'column_width' => 2,
            ])
            ->save();
    }
}
```

Day 6: Basic Job System Integration

6.1 Create Product Verification Job

Create `src/Job/ProductVerificationJob.php`:

```
<?php
declare(strict_types=1);
```

```
namespace App\Job;

use App\Job\AbstractJob;
use App\Service\Api\Anthropic\ProductAnalyzer;
use App\Utility\SettingsManager;

class ProductVerificationJob extends AbstractJob
{
    public function execute(array $data): void
    {
        $this->validateArguments($data, ['product_id']);

        $productId = $data['product_id'];
        $this->log("Starting verification for product {$productId}",
'info');

        try {
            // Get product data
            $productsTable = $this->getTableInstance('Products');
            $product = $productsTable->get($productId, [
                'contain' => ['Tags', 'Article']
            ]);

            $verificationScore = $this-
>calculateVerificationScore($product);

            // Use AI verification if enabled
            if (SettingsManager::read('Products.aiVerificationEnabled',
true)) {
                $aiScore = $this->runAIVerification($product);
                $verificationScore = ($verificationScore + $aiScore) / 2;
            }

            // Update product with verification score
            $product->reliability_score = $verificationScore;

            // Auto-publish if score is high enough
            $autoPublishThreshold =
(float)SettingsManager::read('Products.autoPublishThreshold', 4.0);
            if ($verificationScore >= $autoPublishThreshold) {
                $product->is_published = true;
                $product->verification_status = 'approved';
                $this->log("Product {$productId} auto-published with score
{$verificationScore}", 'info');
            } else {
                $product->verification_status = 'pending';
                $this->log("Product {$productId} pending review with score
{$verificationScore}", 'info');
            }

            $productsTable->save($product);

        } catch (\Exception $e) {
            $this->log("Product verification job failed: " . $e-
>getMessage(), 'error');
            throw $e;
```

```php
        }
    }

    /**
     * Calculate basic verification score based on completeness
     */
    private function calculateVerificationScore($product): float
    {
        $score = 0;
        $maxScore = 0;

        // Title is required (weight: 1.0)
        $maxScore += 1.0;
        if (!empty($product->title)) {
            $score += 1.0;
        }

        // Description (weight: 1.5)
        $maxScore += 1.5;
        if (!empty($product->description) && strlen($product->description)
>= 50) {
            $score += 1.5;
        } elseif (!empty($product->description)) {
            $score += 0.75;
        }

        // Manufacturer (weight: 1.0)
        $maxScore += 1.0;
        if (!empty($product->manufacturer)) {
            $score += 1.0;
        }

        // Model number (weight: 0.5)
        $maxScore += 0.5;
        if (!empty($product->model_number)) {
            $score += 0.5;
        }

        // Image (weight: 1.0)
        $maxScore += 1.0;
        if (!empty($product->image)) {
            $score += 1.0;
        }

        // Tags (weight: 0.5)
        $maxScore += 0.5;
        if (!empty($product->tags) && count($product->tags) > 0) {
            $score += 0.5;
        }

        // Convert to 5-point scale
        return ($score / $maxScore) * 5.0;
    }

    /**
     * Run AI verification if service is available
     */
    private function runAIVerification($product): float
    {
        try {
            // This would integrate with the AI service
            // For now, return a basic score
            return 4.0;
        } catch (\Exception $e) {
            $this->log("AI verification failed: " . $e->getMessage(),
'warning');
            return 3.0; // Fallback score
        }
    }
}
```

Day 7: Phase 1 Testing & Validation

7.1 Run Initial Test Suite

```
# Run all new model tests
docker compose exec willowcms php vendor/bin/phpunit
tests/TestCase/Model/Table/ProductsTableTest.php
docker compose exec willowcms php vendor/bin/phpunit
tests/TestCase/Model/Entity/ProductTest.php

# Test the search service
docker compose exec willowcms php vendor/bin/phpunit
tests/TestCase/Service/Search/

# Generate coverage report
docker compose exec willowcms php vendor/bin/phpunit --coverage-html
webroot/coverage tests/TestCase/Model/
```

7.2 Validate Database Schema

```
# Verify all migrations ran successfully
docker compose exec willowcms bin/cake migrations status

# Test database structure
docker compose exec mysql mysql -u cms_user -ppassword cms -e "DESCRIBE
products;"
docker compose exec mysql mysql -u cms_user -ppassword cms -e "DESCRIBE
products_tags;"
```

Phase 2: Admin Interface & CRUD Operations (Week 3-4)

Day 8: Admin Controllers Creation

8.1 Generate Admin Controllers

```
# Create admin controllers using AdminTheme
docker compose exec willowcms bin/cake bake controller Admin/Products --
theme AdminTheme

# This creates:
# - src/Controller/Admin/ProductsController.php
```

8.2 Generate Admin Templates

```
# Generate all admin templates using AdminTheme
docker compose exec willowcms bin/cake bake template Admin/Products --theme
AdminTheme

# This creates:
# - plugins/AdminTheme/templates/Admin/Products/index.php
# - plugins/AdminTheme/templates/Admin/Products/view.php
# - plugins/AdminTheme/templates/Admin/Products/add.php
# - plugins/AdminTheme/templates/Admin/Products/edit.php
```

Day 9: Enhanced Admin Controllers

9.1 Enhance Products Admin Controller

Update `src/Controller/Admin/ProductsController.php`:

```
<?php
declare(strict_types=1);

namespace App\Controller\Admin;

use App\Controller\Admin\AppController;
use App\Service\Search\UnifiedSearchService;
```

```php
class ProductsController extends AppController
{
    protected UnifiedSearchService $searchService;

    public function initialize(): void
    {
        parent::initialize();
        $this->searchService = new UnifiedSearchService();
    }

    /**
     * Index method - Enhanced with filtering and search
     */
    public function index(): void
    {
        $query = $this->Products->find()
            ->contain(['Users', 'Tags', 'Articles'])
            ->order(['Products.created' => 'DESC']);

        // Apply filters
        if ($this->request->getQuery('status')) {
            $query->where(['verification_status' => $this->request-
>getQuery('status')]);
        }

        if ($this->request->getQuery('published')) {
            $published = $this->request->getQuery('published') === '1';
            $query->where(['is_published' => $published]);
        }

        if ($this->request->getQuery('featured')) {
            $query->where(['featured' => true]);
        }

        if ($this->request->getQuery('search')) {
            $search = $this->request->getQuery('search');
            $query->where([
                'OR' => [
                    'Products.title LIKE' => "%{$search}%",
                    'Products.description LIKE' => "%{$search}%",
                    'Products.manufacturer LIKE' => "%{$search}%",
                    'Products.model_number LIKE' => "%{$search}%"
                ]
            ]);
        }

        $this->set('products', $this->paginate($query));

        // Get filter options
        $tags = $this->Products->Tags
            ->find('list', ['keyField' => 'id', 'valueField' => 'title'])
            ->order(['title' => 'ASC']);

        $this->set(compact('tags'));
    }

    /**
```

```
     * Dashboard method - Product overview
     */
    public function dashboard(): void
    {
        // Basic statistics
        $totalProducts = $this->Products->find()->count();
        $publishedProducts = $this->Products->find()->where(['is_published'
=> true])->count();
        $pendingProducts = $this->Products->find()-
>where(['verification_status' => 'pending'])->count();
        $featuredProducts = $this->Products->find()->where(['featured' =>
true])->count();

        // Recent products
        $recentProducts = $this->Products->find()
            ->contain(['Users', 'Tags'])
            ->order(['created' => 'DESC'])
            ->limit(10)
            ->toArray();

        // Top manufacturers
        $topManufacturers = $this->Products->find()
            ->select([
                'manufacturer',
                'count' => $this->Products->find()->func()->count('*')
            ])
            ->where(['manufacturer IS NOT' => null])
            ->group('manufacturer')
            ->order(['count' => 'DESC'])
            ->limit(10)
            ->toArray();

        // Popular tags
        $popularTags = $this->Products->Tags->find()
            ->select([
                'Tags.title',
                'count' => $this->Products->Tags->find()->func()-
>count('ProductsTags.product_id')
            ])
            ->leftJoinWith('Products')
            ->group('Tags.id')
            ->order(['count' => 'DESC'])
            ->limit(10)
            ->toArray();

        $this->set(compact(
            'totalProducts',
            'publishedProducts',
            'pendingProducts',
            'featuredProducts',
            'recentProducts',
            'topManufacturers',
            'popularTags'
        ));
    }

    /**
```

```
     * View method - Enhanced with related products
     */
    public function view($id = null): void
    {
        $product = $this->Products->get($id, [
            'contain' => ['Users', 'Tags', 'Articles'],
        ]);

        // Get related products
        $relatedProducts = $this->Products->getRelatedProducts($id, 5);

        $this->set(compact('product', 'relatedProducts'));
    }

    /**
     * Add method - Enhanced with unified tagging
     */
    public function add(): void
    {
        $product = $this->Products->newEmptyEntity();

        if ($this->request->is('post')) {
            $data = $this->request->getData();
            $data['user_id'] = $this->getRequest()-
>getAttribute('identity')->id;

            $product = $this->Products->patchEntity($product, $data, [
                'associated' => ['Tags']
            ]);

            if ($this->Products->save($product)) {
                $this->Flash->success(__('The product has been saved.'));

                // Queue verification job
                $this->queueJob('ProductVerificationJob', [
                    'product_id' => $product->id
                ]);

                return $this->redirect(['action' => 'index']);
            }
            $this->Flash->error(__('The product could not be saved. Please,
try again.'));
        }

        // Get form options
        $users = $this->Products->Users->find('list', ['limit' => 200])-
>all();
        $articles = $this->Products->Articles
            ->find('list', ['keyField' => 'id', 'valueField' => 'title'])
            ->where(['is_published' => true])
            ->order(['title' => 'ASC']);
        $tags = $this->Products->Tags->find('list', ['limit' => 200])-
>all();

        $this->set(compact('product', 'users', 'articles', 'tags'));
    }
    /**
     * Edit method - Enhanced with change tracking
     */
    public function edit($id = null): void
    {
        $product = $this->Products->get($id, [
            'contain' => ['Tags'],
        ]);

        if ($this->request->is(['patch', 'post', 'put'])) {
            $originalScore = $product->reliability_score;

            $product = $this->Products->patchEntity($product, $this-
>request->getData(), [
                'associated' => ['Tags']
            ]);

            if ($this->Products->save($product)) {
                $this->Flash->success(__('The product has been saved.'));

                // Re-verify if significant changes were made
                if ($this->hasSignificantChanges($product)) {
                    $this->queueJob('ProductVerificationJob', [
                        'product_id' => $product->id
                    ]);
                }

                return $this->redirect(['action' => 'index']);
            }
            $this->Flash->error(__('The product could not be saved. Please,
try again.'));
        }

        $users = $this->Products->Users->find('list', ['limit' => 200])-
>all();
        $articles = $this->Products->Articles
            ->find('list', ['keyField' => 'id', 'valueField' => 'title'])
            ->where(['is_published' => true])
            ->order(['title' => 'ASC']);
        $tags = $this->Products->Tags->find('list', ['limit' => 200])-
>all();

        $this->set(compact('product', 'users', 'articles', 'tags'));
    }

    /**
     * Delete method
     */
    public function delete($id = null): void
    {
        $this->request->allowMethod(['post', 'delete']);
        $product = $this->Products->get($id);

        if ($this->Products->delete($product)) {
            $this->Flash->success(__('The product has been deleted.'));
        } else {
            $this->Flash->error(__('The product could not be deleted.
Please, try again.'));
        }

        return $this->redirect(['action' => 'index']);
    }

    /**
     * Verify method - Manual verification trigger
     */
    public function verify($id = null): void
    {
        $this->request->allowMethod(['post']);

        $this->queueJob('ProductVerificationJob', [
            'product_id' => $id
        ]);

        $this->Flash->success(__('Product verification has been queued.'));

        return $this->redirect(['action' => 'view', $id]);
    }

    /**
     * Toggle featured status
     */
    public function toggleFeatured($id = null): void
    {
        $this->request->allowMethod(['post']);

        $product = $this->Products->get($id);
        $product->featured = !$product->featured;

        if ($this->Products->save($product)) {
            $status = $product->featured ? 'featured' : 'unfeatured';
            $this->Flash->success(__('Product has been {0}.', $status));
        } else {
            $this->Flash->error(__('Could not update product status.'));
        }

        return $this->redirect($this->referer(['action' => 'index']));
    }

    /**
     * Check if product has significant changes requiring re-verification
     */
    private function hasSignificantChanges($product): bool
    {
        return $product->isDirty(['title', 'description', 'manufacturer',
'model_number']);
    }

    /**
     * Queue a background job
     */
    private function queueJob(string $jobClass, array $data): void
    {
        $this->loadComponent('Queue.Queue');
```

```
            $this->Queue->createJob($jobClass, $data);
        }
    }
```

Day 10: Admin Templates Enhancement

10.1 Create Enhanced Products Dashboard Template

Create `plugins/AdminTheme/templates/Admin/Products/dashboard.php`:

```
<?php
$this->assign('title', __('Products Dashboard'));
$this->Html->css('willow-admin', ['block' => true]);
?>

<div class="row">
    <div class="col-md-12">
        <div class="actions-card">
            <h3><?= __('Products Dashboard') ?></h3>
            <p class="text-muted"><?= __('Simplified product management
overview') ?></p>
        </div>
    </div>
</div>

<!-- Summary Statistics -->
<div class="row">
    <div class="col-md-3">
        <div class="card">
            <div class="card-body text-center">
                <h5 class="card-title"><?= __('Total Products') ?></h5>
                <h2 class="text-primary"><?= number_format($totalProducts) ?
></h2>
                <small class="text-muted"><?= __('All products') ?></small>
            </div>
        </div>
    </div>

    <div class="col-md-3">
        <div class="card">
            <div class="card-body text-center">
                <h5 class="card-title"><?= __('Published') ?></h5>
                <h2 class="text-success"><?=
number_format($publishedProducts) ?></h2>
                <small class="text-muted"><?= __('Live products') ?></small>
            </div>
        </div>
    </div>

    <div class="col-md-3">
        <div class="card">
```

```
                <div class="card-body text-center">
                    <h5 class="card-title"><?= __('Pending Review') ?></h5>
                    <h2 class="text-warning"><?= number_format($pendingProducts)
?></h2>
                    <small class="text-muted"><?= __('Awaiting verification') ?>
</small>
                </div>
            </div>
        </div>

        <div class="col-md-3">
            <div class="card">
                <div class="card-body text-center">
                    <h5 class="card-title"><?= __('Featured') ?></h5>
                    <h2 class="text-info"><?= number_format($featuredProducts) ?
></h2>
                    <small class="text-muted"><?= __('Featured products') ?>
</small>
                </div>
            </div>
        </div>
    </div>

    <!-- Recent Products -->
    <div class="row mt-4">
        <div class="col-md-6">
            <div class="card">
                <div class="card-header">
                    <h5><?= __('Recent Products') ?></h5>
                </div>
                <div class="card-body">
                    <?php if (!empty($recentProducts)): ?>
                        <div class="list-group list-group-flush">
                            <?php foreach ($recentProducts as $product): ?>
                            <div class="list-group-item d-flex justify-content-
between align-items-center">
                                <div>
                                    <h6 class="mb-1"><?= h($product->title) ?>
</h6>
                                    <small class="text-muted">
                                        <?= $product->created->format('M j, Y')
?>
                                        by <?= h($product->user->username) ?>
                                        <?php if ($product->manufacturer): ?>
                                            • <?= h($product->manufacturer) ?>
                                        <?php endif; ?>
                                    </small>
                                    <?php if (!empty($product->tags)): ?>
                                        <div class="mt-1">
                                            <?php foreach ($product->tags as
$tag): ?>
                                                <span class="badge badge-
secondary badge-sm"><?= h($tag->title) ?></span>
                                            <?php endforeach; ?>
                                        </div>
                                    <?php endif; ?>
                                </div>
                                <div>
                                    <span class="badge badge-<?= $product-
>is_published ? 'success' : 'warning' ?>">
                                        <?= $product->is_published ?
__('Published') : __(ucfirst($product->verification_status)) ?>
                                    </span>
                                </div>
                            </div>
                            <?php endforeach; ?>
                        </div>
                    <?php else: ?>
                        <p class="text-muted"><?= __('No recent products') ?>
</p>
                    <?php endif; ?>
                </div>
            </div>
        </div>

        <div class="col-md-6">
            <div class="card">
                <div class="card-header">
                    <h5><?= __('Top Manufacturers') ?></h5>
                </div>
                <div class="card-body">
                    <?php if (!empty($topManufacturers)): ?>
                        <div class="list-group list-group-flush">
                            <?php foreach ($topManufacturers as $manufacturer):
?>
                            <div class="list-group-item d-flex justify-content-
between align-items-center">
                                <span><?= h($manufacturer->manufacturer) ?>
</span>
                                <span class="badge badge-primary badge-pill"><?=
$manufacturer->count ?></span>
                            </div>
                            <?php endforeach; ?>
                        </div>
                    <?php else: ?>
                        <p class="text-muted"><?= __('No manufacturer data
available') ?></p>
                    <?php endif; ?>
                </div>
            </div>
        </div>
    </div>

    <!-- Popular Tags -->
    <?php if (!empty($popularTags)): ?>
    <div class="row mt-4">
        <div class="col-md-12">
            <div class="card">
                <div class="card-header">
                    <h5><?= __('Popular Tags') ?></h5>
                </div>
                <div class="card-body">
                    <div class="row">
                        <?php foreach ($popularTags as $tag): ?>
                        <div class="col-md-3 mb-2">
                            <span class="badge badge-info p-2">
                                <?= h($tag->title) ?>
                                <span class="badge badge-light ml-1"><?= $tag-
    >count ?></span>
                            </span>
                        </div>
                        <?php endforeach; ?>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <?php endif; ?>

    <!-- Quick Actions -->
    <div class="row mt-4">
        <div class="col-md-12">
            <div class="card">
                <div class="card-header">
                    <h5><?= __('Quick Actions') ?></h5>
                </div>
                <div class="card-body">
                    <div class="btn-group" role="group">
                        <?= $this->Html->link(
                            '<i class="fas fa-plus"></i> ' . __('Add Product'),
                            ['action' => 'add'],
                            ['class' => 'btn btn-success', 'escape' => false]
                        ) ?>
                        <?= $this->Html->link(
                            '<i class="fas fa-list"></i> ' . __('All Products'),
                            ['action' => 'index'],
                            ['class' => 'btn btn-primary', 'escape' => false]
                        ) ?>
                        <?= $this->Html->link(
                            '<i class="fas fa-clock"></i> ' . __('Pending
    Review'),
                            ['action' => 'index', '?' => ['status' =>
    'pending']],
                            ['class' => 'btn btn-warning', 'escape' => false]
                        ) ?>
                        <?= $this->Html->link(
                            '<i class="fas fa-star"></i> ' . __('Featured
    Products'),
                            ['action' => 'index', '?' => ['featured' => '1']],
                            ['class' => 'btn btn-info', 'escape' => false]
                        ) ?>
                    </div>
                </div>
            </div>
        </div>
    </div>
```

Phase 3: Integration Testing & Navigation (Week 5+)

Day 11: Integration Testing Setup

11.1 Create Comprehensive Controller Tests

Create `tests/TestCase/Controller/Admin/ProductsControllerTest.php`:

```php
<?php
declare(strict_types=1);

namespace App\Test\TestCase\Controller\Admin;

use Cake\TestSuite\IntegrationTestTrait;
use Cake\TestSuite\TestCase;
use Cake\ORM\TableRegistry;

class ProductsControllerTest extends TestCase
{
    use IntegrationTestTrait;

    protected $fixtures = [
        'app.Products',
        'app.ProductsTags',
        'app.Articles',
        'app.Tags',
        'app.Users'
    ];

    public function setUp(): void
    {
        parent::setUp();
        $this->configRequest([
            'environment' => [
                'PHP_AUTH_USER' => 'admin@test.com',
                'PHP_AUTH_PW' => 'password'
            ]
        ]);
    }

    public function testIndex(): void
    {
        $this->get('/admin/products');

        $this->assertResponseOk();
        $this->assertResponseContains('Products');

        // Test that products are displayed
        $viewVars = $this->viewVariable('products');
        $this->assertNotEmpty($viewVars);
    }

    public function testIndexWithFilters(): void
    {
        // Test status filter
        $this->get('/admin/products?status=pending');
        $this->assertResponseOk();

        // Test published filter
        $this->get('/admin/products?published=1');
        $this->assertResponseOk();

        // Test search filter
        $this->get('/admin/products?search=test');
        $this->assertResponseOk();
    }

    public function testDashboard(): void
    {
        $this->get('/admin/products/dashboard');

        $this->assertResponseOk();
        $this->assertResponseContains('Products Dashboard');

        // Check that required view variables are set
        $this->assertNotNull($this->viewVariable('totalProducts'));
        $this->assertNotNull($this->viewVariable('publishedProducts'));
        $this->assertNotNull($this->viewVariable('pendingProducts'));
        $this->assertNotNull($this->viewVariable('featuredProducts'));
    }

    public function testView(): void
    {
        $products = TableRegistry::getTableLocator()->get('Products');
        $product = $products->find()->first();

        $this->get('/admin/products/view/' . $product->id);

        $this->assertResponseOk();
        $this->assertResponseContains($product->title);
    }

    public function testAdd(): void
    {
        $this->get('/admin/products/add');

        $this->assertResponseOk();
        $this->assertResponseContains('Add Product');
    }

    public function testAddPost(): void
    {
        $data = [
            'title' => 'Test Product',
            'slug' => 'test-product',
            'description' => 'Test product description',
            'manufacturer' => 'Test Manufacturer',
            'model_number' => 'TM-001',
            'is_published' => false,
            'verification_status' => 'pending'
        ];

        $this->post('/admin/products/add', $data);

        $this->assertResponseSuccess();
        $this->assertFlashMessage('The product has been saved.');

        // Verify product was created
        $products = TableRegistry::getTableLocator()->get('Products');
        $product = $products->find()->where(['title' => 'Test Product'])-
>first();
        $this->assertNotNull($product);
    }

    public function testEdit(): void
    {
        $products = TableRegistry::getTableLocator()->get('Products');
        $product = $products->find()->first();

        $this->get('/admin/products/edit/' . $product->id);

        $this->assertResponseOk();
        $this->assertResponseContains('Edit Product');
        $this->assertResponseContains($product->title);
    }

    public function testToggleFeatured(): void
    {
        $products = TableRegistry::getTableLocator()->get('Products');
        $product = $products->find()->first();
        $originalStatus = $product->featured;

        $this->post('/admin/products/toggle-featured/' . $product->id);

        $this->assertResponseSuccess();

        // Verify featured status was toggled
        $updatedProduct = $products->get($product->id);
        $this->assertEquals(!$originalStatus, $updatedProduct->featured);
    }

    public function testDelete(): void
    {
        $products = TableRegistry::getTableLocator()->get('Products');
        $product = $products->find()->first();

        $this->post('/admin/products/delete/' . $product->id);

        $this->assertResponseSuccess();
        $this->assertFlashMessage('The product has been deleted.');

        // Verify product was deleted
        $this->assertFalse($products->exists(['id' => $product->id]));
    }

    public function testUnifiedTagging(): void
    {
        // Test that products can be associated with same tags as articles
        $data = [
            'title' => 'Tagged Product',
            'slug' => 'tagged-product',
            'description' => 'Product with tags',
            'tags' => [
                ['_ids' => [1, 2]] // Assuming tags with IDs 1 and 2 exist
            ]
        ];

        $this->post('/admin/products/add', $data);
        $this->assertResponseSuccess();

        // Verify tags were associated
        $products = TableRegistry::getTableLocator()->get('Products');
        $product = $products->find()
            ->contain(['Tags'])
            ->where(['title' => 'Tagged Product'])
            ->first();

        $this->assertNotEmpty($product->tags);
    }
}
```

11.2 Create Search Service Tests

Create `tests/TestCase/Service/Search/UnifiedSearchServiceTest.php`:

```
<?php
declare(strict_types=1);

namespace App\Test\TestCase\Service\Search;

use App\Service\Search\UnifiedSearchService;
use Cake\TestSuite\TestCase;
use Cake\ORM\TableRegistry;

class UnifiedSearchServiceTest extends TestCase
{
    protected $fixtures = [
        'app.Products',
        'app.Articles',
        'app.Tags',
        'app.ProductsTags',
        'app.ArticlesTags',
        'app.Users'
    ];
    protected $service;

    public function setUp(): void
    {
        parent::setUp();
        $this->service = new UnifiedSearchService();
    }

    public function testSearch(): void
    {
        $results = $this->service->search('test');

        $this->assertIsArray($results);
        $this->assertArrayHasKey('products', $results);
        $this->assertArrayHasKey('articles', $results);
        $this->assertArrayHasKey('tags', $results);
        $this->assertArrayHasKey('total', $results);
        $this->assertArrayHasKey('mixed', $results);
    }

    public function testSearchByTag(): void
    {
        $results = $this->service->searchByTag('technology');

        $this->assertIsArray($results);
        $this->assertArrayHasKey('tag', $results);
        $this->assertArrayHasKey('articles', $results);
        $this->assertArrayHasKey('products', $results);
    }

    public function testGetSuggestions(): void
    {
        $suggestions = $this->service->getSuggestions('test', 5);

        $this->assertIsArray($suggestions);
        $this->assertLessThanOrEqual(15, count($suggestions)); // 5 * 3
types max

        if (!empty($suggestions)) {
            $this->assertArrayHasKey('text', $suggestions[^0]);
            $this->assertArrayHasKey('type', $suggestions[^0]);
            $this->assertArrayHasKey('url', $suggestions[^0]);
        }
    }

    public function testSearchRelevanceScoring(): void
    {
        // This tests the internal scoring mechanism
        $results = $this->service->search('exact product title');

        if (!empty($results['mixed'])) {
            // Results should be ordered by relevance
            $scores = [];
            foreach ($results['mixed'] as $result) {
                // Check that each result has required fields
                $this->assertArrayHasKey('title', $result);
                $this->assertArrayHasKey('type', $result);
                $this->assertArrayHasKey('url', $result);
            }
        }
    }

    public function tearDown(): void
    {
        unset($this->service);
        parent::tearDown();
    }
}
```

Day 12: Navigation Integration

12.1 Add Products Navigation to Admin Menu

Update the AdminTheme navigation to include Products menu:

```
<!-- Add to admin navigation menu -->
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" id="productsDropdown"
role="button" data-toggle="dropdown">
        <i class="fas fa-box"></i> <?= __('Products') ?>
    </a>
    <div class="dropdown-menu">
        <?= $this->Html->link(
            '<i class="fas fa-chart-line"></i> ' . __('Dashboard'),
            ['controller' => 'Products', 'action' => 'dashboard'],
            ['class' => 'dropdown-item', 'escape' => false]
        ) ?>
        <div class="dropdown-divider"></div>
        <?= $this->Html->link(
            '<i class="fas fa-list"></i> ' . __('All Products'),
            ['controller' => 'Products', 'action' => 'index'],
            ['class' => 'dropdown-item', 'escape' => false]
        ) ?>
        <?= $this->Html->link(
            '<i class="fas fa-plus"></i> ' . __('Add Product'),
            ['controller' => 'Products', 'action' => 'add'],
            ['class' => 'dropdown-item', 'escape' => false]
        ) ?>
        <div class="dropdown-divider"></div>
        <?= $this->Html->link(
            '<i class="fas fa-clock"></i> ' . __('Pending Review'),
            ['controller' => 'Products', 'action' => 'index', '?' =>
['status' => 'pending']],
            ['class' => 'dropdown-item', 'escape' => false]
        ) ?>
        <?= $this->Html->link(
            '<i class="fas fa-star"></i> ' . __('Featured'),
            ['controller' => 'Products', 'action' => 'index', '?' =>
['featured' => '1']],
            ['class' => 'dropdown-item', 'escape' => false]
        ) ?>
    </div>
</li>
```

12.2 Update Routes Configuration

Add to `config/routes.php` in the admin prefix section:

```
// In the existing admin prefix block
$builder->prefix('Admin', function (RouteBuilder $routes): void {
    // ... existing routes ...

    // Products routes
    $routes->connect('/products/dashboard', [
        'controller' => 'Products',
        'action' => 'dashboard'
    ]);
    $routes->connect('/products/verify/*', [
        'controller' => 'Products',
        'action' => 'verify'
    ]);
    $routes->connect('/products/toggle-featured/*', [
        'controller' => 'Products',
        'action' => 'toggleFeatured'
    ]);

    $routes->fallbacks(DashedRoute::class);
});
```

Day 13-14: Final Integration Testing & Polish

13.1 Comprehensive Integration Tests

```
# Run complete test suite
docker compose exec willowcms php vendor/bin/phpunit --coverage-html
webroot/coverage

# Test specific components
docker compose exec willowcms php vendor/bin/phpunit
tests/TestCase/Controller/Admin/Products*
docker compose exec willowcms php vendor/bin/phpunit
tests/TestCase/Service/Search/
docker compose exec willowcms php vendor/bin/phpunit
tests/TestCase/Model/Table/Products*
```

```
# Test with queue worker
docker compose exec willowcms bin/cake queue worker &
docker compose exec willowcms php vendor/bin/phpunit
tests/TestCase/Job/Product*
```

13.2 Test Unified Search Functionality

```
# Test unified search across content types
docker compose exec willowcms bin/cake test_unified_search "technology"

# Test tag-based search
docker compose exec willowcms bin/cake test_tag_search "mobile"

# Test search suggestions
docker compose exec willowcms bin/cake test_search_suggestions "iphone"
```

13.3 Final Validation Checklist

- [ ] All migrations run successfully
- [ ] Products model has proper relationships with Articles and Tags
- [ ] Unified search works across products, articles, and tags
- [ ] Slug system supports products with proper redirects
- [ ] Admin CRUD interfaces work correctly
- [ ] Product verification system functions
- [ ] Settings are properly configured
- [ ] Navigation is integrated
- [ ] All tests pass
- [ ] Performance is acceptable

Testing Commands Summary

```
# Complete test suite
docker compose exec willowcms php vendor/bin/phpunit

# Products-specific tests only
docker compose exec willowcms php vendor/bin/phpunit
tests/TestCase/Model/Table/ProductsTableTest.php
docker compose exec willowcms php vendor/bin/phpunit
tests/TestCase/Controller/Admin/ProductsControllerTest.php
docker compose exec willowcms php vendor/bin/phpunit
tests/TestCase/Service/Search/
```

```
# Coverage report
docker compose exec willowcms php vendor/bin/phpunit --coverage-html
webroot/coverage

# Specific test methods
docker compose exec willowcms php vendor/bin/phpunit --filter
testUnifiedSearch
docker compose exec willowcms php vendor/bin/phpunit --filter
testProductTagging
```

File Structure Created

```
├── config/Migrations/
│   ├── YYYYMMDD_HHMMSS_CreateSimplifiedProducts.php
│   ├── YYYYMMDD_HHMMSS_EnhanceSlugsForProducts.php
│   └── YYYYMMDD_HHMMSS_InsertSimplifiedProductSettings.php
├── src/
│   ├── Controller/Admin/
│   │   └── ProductsController.php
│   ├── Model/
│   │   ├── Entity/
│   │   │   └── Product.php
│   │   └── Table/
│   │       └── ProductsTable.php (enhanced)
│   ├── Service/Search/
│   │   └── UnifiedSearchService.php
│   └── Job/
│       └── ProductVerificationJob.php
├── plugins/AdminTheme/templates/Admin/Products/
│   ├── index.php
│   ├── view.php
│   ├── add.php
│   ├── edit.php
│   └── dashboard.php
└── tests/
    ├── Fixture/
    │   └── ProductsFixture.php
    └── TestCase/
        ├── Controller/Admin/
        │   └── ProductsControllerTest.php
        ├── Model/Table/
        │   └── ProductsTableTest.php
        ├── Service/Search/
        │   └── UnifiedSearchServiceTest.php
        └── Job/
            └── ProductVerificationJobTest.php
```

Success Metrics

Phase 1 Completion:

- ✅ Simplified products table created with essential fields
- ✅ Unified tagging system operational across articles and products
- ✅ Product-Article association working (optional relationship)
- ✅ Slug system enhanced to support products
- ✅ Unified search service functional across all content types

Phase 2 Completion:

- ✅ Admin CRUD interfaces fully functional for products
- ✅ Product dashboard displays accurate metrics
- ✅ Settings integration works seamlessly
- ✅ All controller tests pass
- ✅ Enhanced templates provide rich functionality

Phase 3 Completion:

- ✅ Navigation properly integrated
- ✅ All integration tests pass
- ✅ Unified search working across products, articles, and tags
- ✅ Performance meets requirements
- ✅ Complete simplified product management system operational

Key Benefits of Simplified Approach

1. **Reduced Complexity**: Single products table instead of 12 separate tables
2. **Unified Tagging**: Same tag system for articles, products, and pages
3. **Flexible Content**: Optional article association for detailed product info
4. **Easy Migration**: Simple path from complex to simplified schema
5. **Better Performance**: Fewer JOINs and simpler queries
6. **Unified Search**: Single search interface across all content types
7. **Maintainable**: Less code to maintain and fewer edge cases

Rollback Procedures

If issues arise:

```
# Rollback migrations
docker compose exec willowcms bin/cake migrations rollback

# Disable new features via settings
# Set Products.enabled = 0 in admin settings

# Remove new routes if needed
# Comment out new routes in config/routes.php

# Run core test suite to ensure base functionality
docker compose exec willowcms php vendor/bin/phpunit
tests/TestCase/Model/Table/ArticlesTableTest.php
```