KAI Platform Documentation

Generated on: 2025-05-07 12:50:57

# Table of Contents

(Table of contents will be generated when opened in Word)

# 3D Reconstruction Pipeline

Source: readme/3d-reconstruction-pipeline.md

---

## 3D Reconstruction Pipeline

This document outlines the 3D reconstruction pipeline implementation for the CrewAI 3D Visual Builder.

### Components Overview

#### 1. Room Layout Extraction

* HorizonNet + CubeMap
* Uses HorizonNet for accurate room boundary detection
* CubeMap generation for complete room visualization
* Handles complex room geometries and layouts
* Dependencies: horizon-net>=1.0.0

horizon-net>=1.0.0

#### 2. Depth Estimation

* MiDaS Integration
* High-quality monocular depth estimation
* Post-processing pipeline for NeRF compatibility
* Confidence map generation
* Dependencies: midas-py>=1.0.0

midas-py>=1.0.0

#### 3. Room Segmentation

* Segment Anything Model (SAM)
* Precise room element segmentation
* Wall, floor, ceiling detection
* Object boundary identification
* Dependencies: segment-anything>=1.0

segment-anything>=1.0

#### 4. Object Detection

* YOLO v8
* Real-time object detection and classification
* Furniture and fixture identification
* High-confidence scoring system
* Dependencies: ultralytics>=8.0.0

ultralytics>=8.0.0

#### 5. NeRF-based Scene Reconstruction

* NerfStudio/Instant-NGP
* Parallel training implementation
* Multi-view synthesis
* High-quality scene reconstruction
* Dependencies:  
  nerfstudio>=0.3.0  
  instant-ngp>=1.0.0
* nerfstudio>=0.3.0
* instant-ngp>=1.0.0

nerfstudio>=0.3.0

instant-ngp>=1.0.0

#### 6. Gaussian Splatting as an Alternative

* Gaussian Splatting Implementation
* 10-20x faster rendering speeds compared to traditional NeRF
* Comparable or better visual quality with improved detail retention
* More efficient training (hours instead of days)
* Better handling of complex geometries and transparent/reflective surfaces

Technical Implementation:  
 - Based on 3D Gaussian Splatting framework and NVIDIA's Splatfacto  
 - Custom Python service (gaussian\_splatting\_service.py) handles:  
 - 3D point cloud to Gaussian primitives conversion  
 - Optimization of 3D Gaussians (position, scale, rotation, opacity)  
 - Progressive coarsening for LOD management  
 - Export to mesh and point-cloud formats

gaussian\_splatting\_service.py

Integration Points:  
 - TypeScript bridge (gaussian-splatting-bridge.ts) connects frontend to Python backend  
 - Enhanced ThreeJS viewer with dedicated GaussianSplattingLoader  
 - Support for real-time Gaussian rendering with WebGL  
 - Progressive loading and streaming for large scenes

gaussian-splatting-bridge.ts

Compatibility Considerations:  
 - Hardware requirements:   
 - GPU with 8GB+ VRAM for training  
 - Standard WebGL-capable GPU for rendering  
 - Browser compatibility:  
 - Full support in Chrome/Edge/Firefox with WebGL 2.0  
 - Limited support in Safari (iOS performance limitations)  
 - Memory usage:  
 - Can require 1.5-2x more memory than mesh-based formats for complex scenes  
 - Progressive streaming helps mitigate memory issues on mobile devices

Potential Integration Issues:  
 - Non-trivial conversion from Gaussian representation to traditional meshes  
 - May require custom shader implementation for optimal rendering  
 - Cannot use standard PBR material system directly on Gaussian points  
 - Limited multi-user editing capabilities for Gaussian-based scenes

Advantages over NeRF:  
 - Real-time rendering without separate mesh extraction step  
 - Better preservation of fine details and transparency  
 - More efficient training pipeline (3-5x faster)  
 - Direct export to optimized point cloud formats  
 - Better interaction with scene lighting and global illumination

#### 6. 3D Model Processing

* BlenderProc
* Automated texturing pipeline
* UV mapping optimization
* Material property extraction
* Dependencies: blenderproc>=2.6.0

blenderproc>=2.6.0

#### 7. Edge Refinement

* Marching Cubes (Open3D)
* Mesh optimization
* Edge detection and refinement
* Surface smoothing
* Dependencies: open3d>=0.17.0

open3d>=0.17.0

### Setup and Installation

1. Install Python dependencies:

cd packages/ml  
pip install -r requirements.txt

cd packages/ml  
pip install -r requirements.txt

1. Install Node.js dependencies:

npm install

npm install

1. Configure environment variables:

cp .env.example .env  
# Edit .env with your settings

cp .env.example .env  
# Edit .env with your settings

### Pipeline Workflow

1. Input Processing
2. Image validation
3. Format conversion
4. Resolution optimization
5. Layout Analysis  
   mermaid  
    graph TD  
    A[Input Image] --> B[HorizonNet]  
    B --> C[Layout Extraction]  
    C --> D[CubeMap Generation]  
    D --> E[Room Structure]
6. Depth and Segmentation  
   mermaid  
    graph TD  
    A[Processed Image] --> B[MiDaS]  
    A --> C[SAM]  
    B --> D[Depth Map]  
    C --> E[Room Segments]  
    D --> F[NeRF Input]  
    E --> F
7. Object Recognition  
   mermaid  
    graph TD  
    A[Scene] --> B[YOLO v8]  
    B --> C[Object Detection]  
    C --> D[Classification]  
    D --> E[Spatial Mapping]
8. 3D Reconstruction  
   mermaid  
    graph TD  
    A[Processed Data] --> B[NeRF Training]  
    B --> C[Scene Reconstruction]  
    C --> D[BlenderProc]  
    D --> E[Final Model]

Resolution optimization

Layout Analysis  
mermaid  
 graph TD  
 A[Input Image] --> B[HorizonNet]  
 B --> C[Layout Extraction]  
 C --> D[CubeMap Generation]  
 D --> E[Room Structure]

mermaid  
 graph TD  
 A[Input Image] --> B[HorizonNet]  
 B --> C[Layout Extraction]  
 C --> D[CubeMap Generation]  
 D --> E[Room Structure]

Depth and Segmentation  
mermaid  
 graph TD  
 A[Processed Image] --> B[MiDaS]  
 A --> C[SAM]  
 B --> D[Depth Map]  
 C --> E[Room Segments]  
 D --> F[NeRF Input]  
 E --> F

mermaid  
 graph TD  
 A[Processed Image] --> B[MiDaS]  
 A --> C[SAM]  
 B --> D[Depth Map]  
 C --> E[Room Segments]  
 D --> F[NeRF Input]  
 E --> F

Object Recognition  
mermaid  
 graph TD  
 A[Scene] --> B[YOLO v8]  
 B --> C[Object Detection]  
 C --> D[Classification]  
 D --> E[Spatial Mapping]

mermaid  
 graph TD  
 A[Scene] --> B[YOLO v8]  
 B --> C[Object Detection]  
 C --> D[Classification]  
 D --> E[Spatial Mapping]

3D Reconstruction  
mermaid  
 graph TD  
 A[Processed Data] --> B[NeRF Training]  
 B --> C[Scene Reconstruction]  
 C --> D[BlenderProc]  
 D --> E[Final Model]

mermaid  
 graph TD  
 A[Processed Data] --> B[NeRF Training]  
 B --> C[Scene Reconstruction]  
 C --> D[BlenderProc]  
 D --> E[Final Model]

### Implementation Details

#### TypeScript Bridge

The ReconstructionBridge class (packages/ml/src/reconstruction-bridge.ts) handles communication between the frontend and Python pipeline:

ReconstructionBridge

packages/ml/src/reconstruction-bridge.ts

interface PipelineConfig {  
 useParallel?: boolean;  
 gpuAcceleration?: boolean;  
 optimizationLevel?: 'fast' | 'balanced' | 'quality';  
 exportFormat?: 'glb' | 'obj' | 'fbx';  
}

interface PipelineConfig {  
 useParallel?: boolean;  
 gpuAcceleration?: boolean;  
 optimizationLevel?: 'fast' | 'balanced' | 'quality';  
 exportFormat?: 'glb' | 'obj' | 'fbx';  
}

#### Python Pipeline

The main reconstruction pipeline (packages/ml/python/room\_reconstruction\_pipeline.py) orchestrates all components:

packages/ml/python/room\_reconstruction\_pipeline.py

1. Layout Extraction
2. Room boundary detection
3. Structural element identification
4. CubeMap generation
5. Depth Processing
6. MiDaS inference
7. Depth map refinement
8. Confidence estimation
9. Segmentation
10. SAM model initialization
11. Room element segmentation
12. Boundary refinement
13. Object Detection
14. YOLO v8 inference
15. Object classification
16. Spatial relationship mapping
17. NeRF Processing
18. Parallel training setup
19. View synthesis
20. Quality optimization
21. Model Processing
22. Mesh extraction
23. UV mapping
24. Texture application
25. Edge Refinement
26. Marching Cubes implementation
27. Edge detection
28. Surface optimization

CubeMap generation

Depth Processing

Confidence estimation

Segmentation

Boundary refinement

Object Detection

Spatial relationship mapping

NeRF Processing

Quality optimization

Model Processing

Texture application

Edge Refinement

### Performance Considerations

* GPU acceleration for NeRF training
* Parallel processing for multiple views
* Memory optimization for large scenes
* Caching for intermediate results

### Error Handling

* Input validation
* Component failure recovery
* Resource cleanup
* Error reporting

### Future Improvements

1. Enhanced Parallelization
2. Multi-GPU support
3. Distributed training
4. Quality Improvements
5. Higher resolution support
6. Better texture mapping
7. Advanced material recognition
8. Pipeline Optimization
9. Faster processing
10. Reduced memory usage
11. Improved caching

Distributed training

Quality Improvements

Advanced material recognition

Pipeline Optimization

### References

* HorizonNet Paper
* MiDaS Documentation
* SAM Paper
* YOLO v8 Documentation
* NeRF Documentation
* BlenderProc Guide
* Open3D Documentation

# 3D Visualization

Source: readme/3d-visualization.md

---

## 3D Visualization System

This document outlines the 3D visualization system's architecture, components, and integration with crewAI.

### Overview

The 3D visualization system provides comprehensive capabilities for:  
- 3D reconstruction from images using NeRF-based models  
- Text-to-3D generation using multiple models  
- Scene understanding and material recognition  
- Integration with existing knowledge base  
- Gaussian Splatting support for enhanced realism  
- WebGPU and WebXR optimizations for improved performance

### Core Components

#### 1. Visualization Layer

##### ThreeJsViewer Component

The core visualization component built with Three.js that provides:  
- Real-time 3D rendering with WebGL  
- WebXR support for AR/VR experiences  
- BVH-optimized ray tracing  
- Efficient scene management

##### EnhancedThreeJsViewer Component

An advanced viewer extension that provides:  
- WebGPU rendering support for modern hardware with performance monitoring  
- Gaussian Splatting for photorealistic point cloud rendering with custom shaders  
- Adaptive Level of Detail (LOD) optimization with distance-based adjustment  
- Hierarchical occlusion culling for performance optimization  
- Improved BVH integration with three-mesh-bvh and spatial partitioning  
- Progressive texture loading for faster initial rendering  
- Deferred rendering pipeline for complex lighting scenarios  
- Dynamic memory management for large scene optimization  
- Texture compression with automatic format selection  
- Instance batching for similar objects  
- Support for multiple model formats (GLTF, GLB, FBX, OBJ, PLY, Gaussian Splats)

// Example usage of EnhancedThreeJsViewer  
<EnhancedThreeJsViewer  
 modelUrl="path/to/model.splat"  
 modelType="gaussian"  
 initialPosition={{ x: 0, y: 0, z: 5 }}  
 enableVR={true}  
 enableAR={true}  
 enableBVH={true}  
 enableLOD={true}  
 enableOcclusionCulling={true}  
 preferWebGPU={true}  
 onSceneReady={(scene) => {  
 // Scene is ready for interaction  
 }}  
/>

// Example usage of EnhancedThreeJsViewer  
<EnhancedThreeJsViewer  
 modelUrl="path/to/model.splat"  
 modelType="gaussian"  
 initialPosition={{ x: 0, y: 0, z: 5 }}  
 enableVR={true}  
 enableAR={true}  
 enableBVH={true}  
 enableLOD={true}  
 enableOcclusionCulling={true}  
 preferWebGPU={true}  
 onSceneReady={(scene) => {  
 // Scene is ready for interaction  
 }}  
/>

// Enhanced configuration options  
interface EnhancedViewerOptions {  
 // Rendering options  
 renderMode: 'webgl' | 'webgl2' | 'webgpu';  
 renderPipeline: 'forward' | 'deferred';  
  
 // Performance options  
 enableInstancing: boolean;  
 enableCompression: boolean;  
  
 // Feature options  
 enableShadows: boolean;  
 shadowType: 'basic' | 'pcss' | 'raytraced';  
  
 // Optimization options  
 cullingStrategy: 'frustum' | 'occlusion' | 'hierarchical';  
 lodStrategy: 'distance' | 'performance' | 'quality';  
  
 // Progressive loading  
 progressiveLoadingEnabled: boolean;  
 initialLoadQuality: 'low' | 'medium' | 'high';  
  
 // Gaussian splat options  
 splatQuality: 'low' | 'medium' | 'high';  
 adaptiveSplatRendering: boolean;  
 maxSplatCount: number;  
}

// Enhanced configuration options  
interface EnhancedViewerOptions {  
 // Rendering options  
 renderMode: 'webgl' | 'webgl2' | 'webgpu';  
 renderPipeline: 'forward' | 'deferred';  
  
 // Performance options  
 enableInstancing: boolean;  
 enableCompression: boolean;  
  
 // Feature options  
 enableShadows: boolean;  
 shadowType: 'basic' | 'pcss' | 'raytraced';  
  
 // Optimization options  
 cullingStrategy: 'frustum' | 'occlusion' | 'hierarchical';  
 lodStrategy: 'distance' | 'performance' | 'quality';  
  
 // Progressive loading  
 progressiveLoadingEnabled: boolean;  
 initialLoadQuality: 'low' | 'medium' | 'high';  
  
 // Gaussian splat options  
 splatQuality: 'low' | 'medium' | 'high';  
 adaptiveSplatRendering: boolean;  
 maxSplatCount: number;  
}

// Example usage of ThreeJsViewer  
<ThreeJsViewer  
 modelUrl="path/to/model.glb"  
 modelType="3d"  
 enableVR={true}  
 enableAR={true}  
 enableBVH={true}  
 onSceneReady={(scene) => {  
 // Scene is ready for interaction  
 }}  
/>

// Example usage of ThreeJsViewer  
<ThreeJsViewer  
 modelUrl="path/to/model.glb"  
 modelType="3d"  
 enableVR={true}  
 enableAR={true}  
 enableBVH={true}  
 onSceneReady={(scene) => {  
 // Scene is ready for interaction  
 }}  
/>

##### SceneController Component

Manages scene modifications and real-time updates:  
- Batch processing for performance  
- Real-time preview system  
- Export capabilities for multiple formats  
- Object selection and manipulation

// Example usage of SceneController  
<SceneController  
 scene={scene}  
 enableRealTimePreview={true}  
 previewInterval={100}  
>  
 {/\* Child components receive scene control props \*/}  
</SceneController>

// Example usage of SceneController  
<SceneController  
 scene={scene}  
 enableRealTimePreview={true}  
 previewInterval={100}  
>  
 {/\* Child components receive scene control props \*/}  
</SceneController>

##### Export Capabilities

Support for multiple 3D formats:  
- GLB/GLTF with metadata preservation  
- FBX export  
- OBJ export  
- Configurable texture and quality settings

##### BVH Optimization

Automatic Bounding Volume Hierarchy for improved performance:  
- Faster ray tracing and intersection tests  
- Optimized scene traversal  
- Automatic updates on geometry changes  
- Enhanced with three-mesh-bvh library integration  
- Optimized ray casting for interactive applications

##### Level of Detail (LOD) System

Dynamic mesh simplification based on camera distance:  
- Automatic creation of multiple detail levels  
- Progressive rendering for complex scenes  
- Exponential distance-based detail reduction  
- Optimized for mobile and low-power devices

##### Occlusion Culling

Advanced rendering optimization techniques:  
- Multi-level hierarchical occlusion culling  
- Hardware-accelerated occlusion queries (WebGPU)  
- Only renders objects within the view frustum  
- Skips rendering for occluded objects  
- Software-based occlusion prediction  
- Temporal coherence optimization to reduce occlusion testing  
- Significant performance boost for complex scenes (up to 70% fewer draw calls)  
- Adaptive culling based on object size, distance, and scene complexity  
- Dynamic occlusion thresholds based on device performance  
- Pre-computed visibility sets for static scenes

// Occlusion culling configuration  
const occlusionSystem = new HierarchicalOcclusionCulling({  
 // Use hardware queries when available  
 useHardwareQueries: renderer.capabilities.hasFeature('occlusion-query'),  
  
 // How many frames to skip between full occlusion tests  
 temporalCoherenceFrames: 5,  
  
 // Minimum object size to consider for culling (prevents culling small objects)  
 minimumObjectSize: 0.5,  
  
 // Pre-compute visibility for static objects  
 precomputeStaticVisibility: true,  
  
 // Debug visualization  
 debugVisualization: false  
});  
  
// Register with the renderer  
renderer.setOcclusionCulling(occlusionSystem);

// Occlusion culling configuration  
const occlusionSystem = new HierarchicalOcclusionCulling({  
 // Use hardware queries when available  
 useHardwareQueries: renderer.capabilities.hasFeature('occlusion-query'),  
  
 // How many frames to skip between full occlusion tests  
 temporalCoherenceFrames: 5,  
  
 // Minimum object size to consider for culling (prevents culling small objects)  
 minimumObjectSize: 0.5,  
  
 // Pre-compute visibility for static objects  
 precomputeStaticVisibility: true,  
  
 // Debug visualization  
 debugVisualization: false  
});  
  
// Register with the renderer  
renderer.setOcclusionCulling(occlusionSystem);

##### WebXR Integration

Built-in support for immersive experiences:  
- VR mode with full scene navigation  
- AR mode for real-world integration  
- Device capability detection  
- Optimized rendering for XR  
- Automatic VR/AR button injection  
- Performance optimizations for mobile XR

##### Gaussian Splatting Support

Integration with state-of-the-art point cloud rendering:  
- Photorealistic rendering of captured environments with advanced point cloud representation  
- Progressive loading of splat data with dynamic level of detail  
- Integration with Python Gaussian Splatting service for processing and conversion  
- Custom shader implementation with adaptive point sizing and alpha blending  
- Real-time environment lighting integration for realistic appearance  
- Support for large-scale scenes with millions of points  
- Adaptive performance optimization based on device capabilities  
- Custom rendering pipeline with optimized draw calls

// The GaussianSplattingShader provides custom rendering for splats  
const splattingMaterial = new THREE.ShaderMaterial({  
 vertexShader: GaussianSplattingShader.vertexShader,  
 fragmentShader: GaussianSplattingShader.fragmentShader,  
 uniforms: {  
 pointSize: { value: 2.0 },  
 alphaTest: { value: 0.5 },  
 splatTexture: { value: null },  
 adaptiveScaling: { value: true },  
 maxDistance: { value: 100.0 }  
 },  
 transparent: true,  
 depthTest: true,  
 blending: THREE.NormalBlending  
});  
  
// The GaussianSplattingLoader handles splat file formats  
const loader = new GaussianSplattingLoader();  
const model = await loader.loadAsync("path/to/model.splat");  
scene.add(model);

// The GaussianSplattingShader provides custom rendering for splats  
const splattingMaterial = new THREE.ShaderMaterial({  
 vertexShader: GaussianSplattingShader.vertexShader,  
 fragmentShader: GaussianSplattingShader.fragmentShader,  
 uniforms: {  
 pointSize: { value: 2.0 },  
 alphaTest: { value: 0.5 },  
 splatTexture: { value: null },  
 adaptiveScaling: { value: true },  
 maxDistance: { value: 100.0 }  
 },  
 transparent: true,  
 depthTest: true,  
 blending: THREE.NormalBlending  
});  
  
// The GaussianSplattingLoader handles splat file formats  
const loader = new GaussianSplattingLoader();  
const model = await loader.loadAsync("path/to/model.splat");  
scene.add(model);

##### WebGPU Integration

Next-generation graphics API support:  
- Automatic capability detection with feature-level testing  
- Seamless fallback to WebGL when WebGPU is unavailable  
- Performance optimization with up to 50% better frame rates on compatible hardware  
- Advanced rendering features including compute shaders for complex calculations  
- Hardware-accelerated ray tracing on supported devices  
- Pipeline state caching for efficient render state management  
- Bindless textures for improved material rendering performance  
- Prepared for future rendering pipeline upgrades with extensible architecture

// WebGPU initialization with fallback  
const renderer = await initRenderer({  
 preferWebGPU: true,  
 fallbackToWebGL: true,  
 powerPreference: 'high-performance',  
 antialias: true,  
 enableRayTracing: hasRayTracingSupport()  
});  
  
// Feature detection example  
if (renderer.capabilities.hasFeature('compute-shaders')) {  
 // Enable advanced compute features  
 scene.enableParticleSimulation();  
 scene.enableFluidDynamics();  
}

// WebGPU initialization with fallback  
const renderer = await initRenderer({  
 preferWebGPU: true,  
 fallbackToWebGL: true,  
 powerPreference: 'high-performance',  
 antialias: true,  
 enableRayTracing: hasRayTracingSupport()  
});  
  
// Feature detection example  
if (renderer.capabilities.hasFeature('compute-shaders')) {  
 // Enable advanced compute features  
 scene.enableParticleSimulation();  
 scene.enableFluidDynamics();  
}

#### 2. Image Processing Pipeline

* Room Layout Extraction
* HorizonNet for initial layout analysis
* CubeMap for room mapping
* Scene cleanup with BlenderProc
* Integration with Gaussian Splatting for photorealistic reconstruction
* Scene Understanding
* YOLO v8 for object detection
* MiDaS for depth estimation
* SAM for scene segmentation

Integration with Gaussian Splatting for photorealistic reconstruction

Scene Understanding

#### 3. Text Processing Pipeline

* Base Structure Generation
* Shap-E for generating base house structure
* GET3D for detailed scene generation
* Hunyuan3D-2 for alternative generation
* Support for direct GLB/GLTF output formats

#### 4. Material Integration

* Leverages existing knowledge base
* Vector similarity search
* Material suggestions based on context
* Integration with FurnitureMaterialEditor component
* Real-time material previews using PBR workflows

### Model Integration

#### NeRF-based Models

* NerfStudio Integration
* Scene reconstruction from multiple views
* Lighting estimation
* Material property extraction
* Conversion pipeline to Gaussian Splatting format
* Instant-NGP
* Fast reconstruction capabilities
* Real-time preview generation
* Optimization for performance
* Direct export to Three.js compatible formats

Conversion pipeline to Gaussian Splatting format

Instant-NGP

#### Text-to-3D Models

* Shap-E
* Base structure generation
* Coarse layout definition
* Initial scene composition
* GET3D
* Detailed object generation
* Furniture placement
* Scene refinement
* Hunyuan3D-2
* Alternative generation approach
* Style-based modifications
* Scene variations

Initial scene composition

GET3D

Scene refinement

Hunyuan3D-2

#### Scene Understanding Models

* YOLO v8
* Object detection and classification
* Spatial relationship analysis
* Scene composition understanding
* MiDaS
* Depth estimation from single images
* Spatial understanding
* Scene structure analysis
* SAM (Segment Anything Model)
* Object and wall segmentation
* Material boundary detection
* Scene component isolation

Scene composition understanding

MiDaS

Scene structure analysis

SAM (Segment Anything Model)

### CrewAI Integration

#### 3D Designer Agent

The system includes a specialized 3D Designer agent that:  
- Processes both images and text descriptions  
- Coordinates multiple model pipelines  
- Integrates with material knowledge base  
- Provides natural language interaction

// Example agent configuration  
const config: ThreeDDesignerConfig = {  
 knowledgeBaseUrl: process.env.KNOWLEDGE\_BASE\_URL,  
 modelEndpoints: {  
 nerfStudio: process.env.NERF\_STUDIO\_ENDPOINT,  
 instantNgp: process.env.INSTANT\_NGP\_ENDPOINT,  
 shapE: process.env.SHAPE\_E\_ENDPOINT,  
 get3d: process.env.GET3D\_ENDPOINT,  
 hunyuan3d: process.env.HUNYUAN3D\_ENDPOINT,  
 blenderProc: process.env.BLENDER\_PROC\_ENDPOINT  
 }  
};

// Example agent configuration  
const config: ThreeDDesignerConfig = {  
 knowledgeBaseUrl: process.env.KNOWLEDGE\_BASE\_URL,  
 modelEndpoints: {  
 nerfStudio: process.env.NERF\_STUDIO\_ENDPOINT,  
 instantNgp: process.env.INSTANT\_NGP\_ENDPOINT,  
 shapE: process.env.SHAPE\_E\_ENDPOINT,  
 get3d: process.env.GET3D\_ENDPOINT,  
 hunyuan3d: process.env.HUNYUAN3D\_ENDPOINT,  
 blenderProc: process.env.BLENDER\_PROC\_ENDPOINT  
 }  
};

#### LLM Integration

* Uses ChatOpenAI for natural language processing
* Handles multimodal inputs (text + images)
* Provides detailed explanations and suggestions

### Usage Examples

#### Image-based Reconstruction

// Process an image for 3D reconstruction  
const result = await threeDService.processImageInput(image, {  
 detectObjects: true,  
 estimateDepth: true,  
 segmentScene: true  
});

// Process an image for 3D reconstruction  
const result = await threeDService.processImageInput(image, {  
 detectObjects: true,  
 estimateDepth: true,  
 segmentScene: true  
});

#### Text-based Generation

// Generate a 3D scene from text description  
const scene = await threeDService.processTextInput(description, {  
 style: "modern",  
 constraints: {  
 roomSize: "large",  
 lighting: "natural"  
 }  
});

// Generate a 3D scene from text description  
const scene = await threeDService.processTextInput(description, {  
 style: "modern",  
 constraints: {  
 roomSize: "large",  
 lighting: "natural"  
 }  
});

#### Scene Refinement

// Refine generated scene based on feedback  
const refined = await threeDService.refineResult(scene, feedback, {  
 focusAreas: ["lighting", "materials"],  
 preserveStructure: true  
});

// Refine generated scene based on feedback  
const refined = await threeDService.refineResult(scene, feedback, {  
 focusAreas: ["lighting", "materials"],  
 preserveStructure: true  
});

### Dependencies

#### Required Packages

* @langchain/openai for LLM integration
* Three.js for 3D visualization
* TensorFlow.js for client-side inference

#### Model Dependencies

* NeRF-based models (NerfStudio, Instant-NGP)
* Text-to-3D models (Shap-E, GET3D, Hunyuan3D-2)
* Scene understanding models (YOLO v8, MiDaS, SAM)

### Setup Instructions

1. Install required packages:

npm install @langchain/openai three @tensorflow/tfjs

npm install @langchain/openai three @tensorflow/tfjs

1. Configure environment variables:

OPENAI\_API\_KEY=your\_key\_here  
KNOWLEDGE\_BASE\_URL=your\_kb\_url  
NERF\_STUDIO\_ENDPOINT=your\_endpoint  
# ... additional endpoints

OPENAI\_API\_KEY=your\_key\_here  
KNOWLEDGE\_BASE\_URL=your\_kb\_url  
NERF\_STUDIO\_ENDPOINT=your\_endpoint  
# ... additional endpoints

1. Initialize the service:

const threeDService = new ThreeDService(config);

const threeDService = new ThreeDService(config);

### Best Practices

#### Image Input

* Provide clear, well-lit images
* Include multiple angles when possible
* Ensure good contrast and minimal noise

#### Text Descriptions

* Be specific about spatial relationships
* Include material preferences
* Specify style and constraints clearly

#### Scene Refinement

* Provide focused feedback
* Specify areas for improvement
* Include reference images when possible

### Error Handling

The system includes comprehensive error handling:  
- Input validation  
- Model availability checks  
- Processing pipeline monitoring  
- Graceful fallbacks

### Performance Considerations

* Model selection based on requirements
* Caching for frequent operations
* Progressive loading for large scenes
* Optimization options for different devices

### Future Improvements

Planned enhancements include:  
- Additional model integrations  
- Real-time collaboration features  
- Enhanced material suggestions  
- Improved performance optimization

# Ai System

Source: readme/AI-SYSTEM.md

---

## Kai Material Recognition AI System

This document provides comprehensive documentation on the AI models and machine learning capabilities integrated into the Kai Material Recognition system.

### Overview

The Kai system combines multiple AI technologies to provide powerful material recognition, classification, and information extraction capabilities:

1. OCR System - Extracts text from images and documents
2. Computer Vision Models - Recognizes and classifies materials based on visual features
3. Vector Embedding System - Creates searchable vector representations for similarity matching
4. Metadata Extraction System - Extracts structured information using pattern recognition and hints

### AI Models and Components

#### OCR System

The OCR (Optical Character Recognition) system is built around Tesseract OCR with custom enhancements:

##### Components:

* Tesseract OCR Engine - Core text recognition capabilities
* Region-Based OCR - Specialized extraction from specific image regions
* Enhanced Preprocessing Pipeline - Custom image preprocessing to improve OCR accuracy
* Error Recovery System - Fallback strategies for handling OCR failures

##### Key Features:

* Multiple OCR engines with automatic selection based on content type
* Specialized preprocessing techniques for catalog materials
* Region detection to focus OCR on relevant areas
* Confidence scoring for extracted text

##### Integration Points:

* ocrService.ts - Core OCR functionality
* regionBasedOCR.ts - Targeted extraction from specific regions
* enhancedPreprocessing.ts - Image quality improvements for OCR

ocrService.ts

regionBasedOCR.ts

enhancedPreprocessing.ts

#### Computer Vision Models

The system uses a hybrid approach for material recognition, combining traditional feature-based methods with deep learning:

##### Models:

1. Feature-based Recognition:
2. Algorithm: Scale-Invariant Feature Transform (SIFT)
3. Implementation: OpenCV through Python bindings
4. Strengths: Works well with limited training data, effective for material textures
5. Location: material\_recognizer.py (feature-based mode)
6. Neural Network Models:
7. TensorFlow Model: MobileNetV2 (optimized for mobile/edge devices)
8. PyTorch Model: ResNet18 (residual network with 18 layers)
9. Strengths: Better at understanding higher-level visual concepts
10. Location: material\_recognizer.py (ml-based mode)
11. Hybrid Approach:
12. Combines confidence scores from both feature-based and neural network approaches
13. Adaptive weighting based on detection confidence
14. Location: material\_recognizer.py (hybrid mode)

Location: material\_recognizer.py (feature-based mode)

material\_recognizer.py

Neural Network Models:

Location: material\_recognizer.py (ml-based mode)

material\_recognizer.py

Hybrid Approach:

material\_recognizer.py

##### Key Features:

* Multiple recognition strategies (feature-based, ML-based, hybrid)
* Confidence thresholds for reliable recognition
* Customizable fusion algorithms for combining results
* Fall-back mechanisms when one approach fails

#### Vector Embedding System

The vector embedding system transforms materials into mathematical vectors for similarity search:

##### Models:

* Feature-based Embedding: Extracts and compresses SIFT features
* Neural Network Embedding: Uses the same neural networks as recognition but removes classification layers
* Hybrid Embedding: Combines both approaches for more robust representations

##### Implementation:

* Located in embedding\_generator.py
* Generates fixed-size embeddings (default: 128 dimensions)
* Supports different frameworks (TensorFlow, PyTorch)
* Includes dimensionality reduction techniques

embedding\_generator.py

##### Search Capabilities:

* Cosine similarity calculation
* Fast nearest-neighbor search
* Material filtering by type and attributes
* Result ranking by similarity score

### AI Enhancement through Administrative Functionality

Our recently implemented administrative functionality enhances these AI capabilities through better metadata management:

#### 1. Category Management System

The Category system provides context for AI models:

* Implementation: category.model.ts
* Key Features:
* Hierarchical category structure
* Parent/child relationships between categories
* Category-specific extraction rules

category.model.ts

AI Enhancement: By organizing materials into categories, the system can apply specialized extraction rules, improving recognition accuracy by narrowing the domain.

#### 2. Metadata Field System

The MetadataField system defines structured fields with AI extraction guidance:

* Implementation: metadataField.model.ts
* Field Types:
* Text/TextArea - For descriptions, names
* Number - For dimensions, weights
* Dropdown - For standardized attributes (R9, R10, R11 ratings)
* AI Guidance:
* Each field includes a "hint" property that guides AI extraction
* Hints explain context, patterns, or locations to find information

metadataField.model.ts

AI Enhancement: Metadata fields provide structured guidance to the OCR and ML systems, significantly improving extraction accuracy.

#### 3. Metadata Extraction with Hints

The core extraction functionality now uses hints to guide the process:

* Implementation: Enhanced extractMaterialInfoFromTexts and extractFieldValueUsingHint functions in material.model.ts
* Extraction Strategies:
* Pattern-based extraction using regular expressions derived from hints
* Context-based extraction (finding text near specific keywords)
* Type-specific extraction strategies for numbers, dropdowns, etc.
* Confidence Tracking:
* Each extracted field includes a confidence score
* Administrators can use these scores to improve hint quality

extractMaterialInfoFromTexts

extractFieldValueUsingHint

material.model.ts

### AI Processing Pipeline

The complete AI pipeline for processing materials works as follows:

1. Document Processing:
2. PDF processor extracts images from catalogs
3. Images are processed for quality improvement
4. OCR Processing:
5. Tesseract OCR extracts text from images
6. Region-based OCR targets specific areas for detailed extraction
7. Text is processed and normalized
8. Material Recognition:
9. Images go through feature extraction (SIFT)
10. Neural networks classify the material
11. Results are combined in hybrid mode for better accuracy
12. Metadata Extraction:
13. Extracted text is processed using metadata field hints
14. Type-specific extraction strategies are applied
15. Confidence scores are calculated for each extracted field
16. Vector Generation:
17. Material images are converted to vector embeddings
18. These vectors enable similarity search
19. Multiple embedding strategies ensure robust matching
20. Storage and Indexing:
21. Materials with metadata and vectors are stored in the database
22. Vector indexes enable fast similarity search
23. Categories and metadata provide structured filtering

Images are processed for quality improvement

OCR Processing:

Text is processed and normalized

Material Recognition:

Results are combined in hybrid mode for better accuracy

Metadata Extraction:

Confidence scores are calculated for each extracted field

Vector Generation:

Multiple embedding strategies ensure robust matching

Storage and Indexing:

### System Requirements and Dependencies

The AI system relies on the following components:

* Python 3.8+ - For ML components
* Node.js 16+ - For server integration
* TensorFlow 2.12+ / PyTorch 2.0+ - ML frameworks
* OpenCV 4.5+ - Computer vision functionality
* Tesseract OCR - Text extraction
* NumPy, SciPy, Scikit-learn - Scientific computing

### Extending the AI System

The system is designed for extensibility:

1. Adding New Models:
2. Place new TensorFlow models in models/material\_classifier\_tf/
3. Place new PyTorch models in models/material\_classifier\_torch.pt
4. Update the model loader in material\_recognizer.py
5. Improving Extraction:
6. Add new metadata fields with hints through the admin interface
7. Refine existing hints based on confidence scores
8. Extend the pattern library in extractFieldValueUsingHint
9. Training New Models:
10. Use the training scripts in ml/scripts/
11. Organize training data by material category
12. Configure training parameters through the API

models/material\_classifier\_tf/

models/material\_classifier\_torch.pt

Update the model loader in material\_recognizer.py

material\_recognizer.py

Improving Extraction:

Extend the pattern library in extractFieldValueUsingHint

extractFieldValueUsingHint

Training New Models:

ml/scripts/

By leveraging the administrative functionality for metadata management, the system becomes more accurate over time without requiring new AI models. Instead, administrators provide better guidance to the existing models through carefully defined metadata fields and extraction hints.

# Readme

Source: readme/README.md

---

## Kai Documentation

Welcome to the Kai documentation. This index provides links to all documentation files organized by category.

### Project Overview

* Main Project Overview - Complete introduction to the Kai system
* Folder Structure - Organization and component interactions
* Analytics System - Analytics architecture and predictive capabilities

### ML Components

#### Machine Learning Core

* ML Package - Overview of ML functionality for material recognition
* ML Python Modules - Python implementation of ML components
* OCR Enhancements - Text extraction improvements
* Training API Improvements - Enhanced training capabilities

#### Model Context Protocol

* MCP Client - TypeScript client for the MCP server
* MCP Server - Centralized model management service
* MCP Server Guide - Detailed implementation guide

#### Material Recognition

* Material Recognition - ML-powered material identification
* PDF Processing Pipeline - Processing PDFs for material data
* PDF Processing - Catalog extraction capabilities
* OCR - Optical Character Recognition system

### Data Management

* Knowledge Base - Material storage and retrieval system
* Datasets and Models - Integration of premade datasets with AI models

#### Metadata

* Material Metadata Fields - Specifications for material metadata
* Material Metadata Extraction - Extraction system for metadata

### Infrastructure & Development

#### Infrastructure

* Queue System - Message broker and async processing
* Web Crawling - Web content extraction for materials
* Testing Approach - Testing strategy and best practices
* Kubernetes Architecture - Kubernetes implementation details
* HPA Configuration Guide - Horizontal Pod Autoscaling configuration
* Advanced Scaling Features - Predictive scaling and cross-service dependencies
* Monitoring System - System health monitoring and operational visibility

#### Deployment

* Deployment & Development - Environment setup guide
* Deployment Guide - Production deployment instructions
* Digital Ocean Kubernetes Setup - DO-specific deployment
* Vercel Deployment Guide - Vercel-specific deployment

#### Database & Storage

* Database & Vector DB - Database architecture
* Supabase - Supabase integration overview
* Supabase Setup Guide - Detailed Supabase setup

### User Interfaces

* Admin Panel - Administration interface
* HeroUI Integration - UI component system implementation
* MoodBoard Feature - Material collection and organization feature

### API & Integration

* API Reference - Comprehensive API endpoints
* Implementation Plans - Roadmap for future features

### Quality & Metrics

* Quality of Data and Metrics - Data quality standards

### Additional Documentation

Additional documentation can be found in the following locations:

* Package-specific documentation (moved to this central location)
* Code comments for implementation details
* API documentation generated from code

### Documentation Tools

* Documentation Consolidation - Automated system for generating consolidated DOCX documentation

# Adaptive Hybrid Embeddings

Source: readme/adaptive-hybrid-embeddings.md

---

## Adaptive Hybrid Embedding System

### Overview

The Adaptive Hybrid Embedding System provides a sophisticated approach to generating high-quality embeddings for material recognition by dynamically selecting and switching between embedding methods based on real-time quality assessment. The system continuously evaluates embedding quality and adapts its approach to optimize results without requiring human intervention.

This system represents a significant advancement in material recognition technology, as it overcomes the limitations of using a single embedding approach by intelligently combining multiple techniques based on their strengths.

### Key Features

* Dynamic Method Selection: Automatically switches between feature-based, ML-based, and hybrid embedding approaches based on quality metrics
* Quality-Based Adaptation: Continuously evaluates embedding quality using multiple metrics and adapts in real-time
* Material-Specific Optimization: Learns optimal embedding methods for different material types
* Performance Tracking: Maintains historical performance data to inform future decisions
* Robust Fallbacks: Gracefully handles failures with a cascade of fallback methods
* Self-Improving: Continuously refines its decision-making through performance history

#### Embedding Approaches in Detail

1. Feature-based approach
2. Uses enhanced SIFT/SURF algorithms to extract distinctive visual features
3. Focuses on texture, pattern, and color characteristics
4. Works well even with partial images or varying lighting conditions
5. Particularly strong for materials with distinctive surface patterns
6. ML-based approach
7. Uses deep neural networks (TensorFlow/PyTorch) to generate high-dimensional feature vectors
8. Trained specifically for material recognition tasks
9. Excels at category classification and semantic understanding
10. Provides excellent general material identification capabilities
11. Hybrid approach
12. Combines feature-based and ML-based methods with a sophisticated confidence fusion system
13. Uses weighted scoring to merge results based on confidence levels
14. Dynamically adjusts the influence of each method based on historical performance
15. Superior performance for specialized material types and challenging recognition scenarios

Particularly strong for materials with distinctive surface patterns

ML-based approach

Provides excellent general material identification capabilities

Hybrid approach

### Architecture

The system consists of three primary components that work together to create an intelligent embedding selection system:

1. Embedding Generators: Multiple embedding generation approaches (feature-based, ML-based, hybrid)
2. Quality Evaluation System: Real-time assessment of embedding quality with multiple metrics
3. Adaptive Controller: Decision-making logic for method selection and switching

┌─────────────────────────┐ ┌─────────────────────────┐  
│ │ │ │  
│ Embedding Generators │ │ Quality Evaluator │  
│ - Feature-based │ │ - Vector Coherence │  
│ - ML-based (TF/PyTorch)│ │ - Discrimination Power │  
│ - Hybrid │ │ - Anomaly Detection │  
│ │ │ - Clustering Alignment │  
└───────────┬─────────────┘ └─────────────┬───────────┘  
 │ │  
 ▼ ▼  
┌─────────────────────────────────────────────────────────┐  
│ │  
│ Adaptive Controller │  
│ - Method Selection Logic │  
│ - Quality Thresholds │  
│ - Performance Tracking │  
│ - Material-Specific Adaptation │  
│ │  
└─────────────────────────────────────────────────────────┘

┌─────────────────────────┐ ┌─────────────────────────┐  
│ │ │ │  
│ Embedding Generators │ │ Quality Evaluator │  
│ - Feature-based │ │ - Vector Coherence │  
│ - ML-based (TF/PyTorch)│ │ - Discrimination Power │  
│ - Hybrid │ │ - Anomaly Detection │  
│ │ │ - Clustering Alignment │  
└───────────┬─────────────┘ └─────────────┬───────────┘  
 │ │  
 ▼ ▼  
┌─────────────────────────────────────────────────────────┐  
│ │  
│ Adaptive Controller │  
│ - Method Selection Logic │  
│ - Quality Thresholds │  
│ - Performance Tracking │  
│ - Material-Specific Adaptation │  
│ │  
└─────────────────────────────────────────────────────────┘

### Quality Evaluation Metrics

The system employs multiple quality metrics to evaluate embeddings:

#### Vector Coherence

Measures the internal quality of the embedding vector by analyzing its statistical properties. Higher coherence indicates a more structured, information-rich embedding.

#### Discrimination Power

Evaluates how well the embedding can distinguish between different materials. A good embedding will place similar materials close together and different materials far apart in the vector space.

#### Anomaly Detection

Identifies embeddings that deviate from expected patterns, detecting potential issues like near-zero vectors, uniform distributions, or extreme values.

#### Clustering Alignment

Assesses how well the embedding aligns with expected clustering behavior for known material categories. Good embeddings will cluster well with other embeddings from the same material category.

### Adaptation Mechanism

The adaptation mechanism operates through the following process:

1. Initial Method Selection:
2. For new materials, starts with the default method (usually hybrid)
3. For previously encountered materials, uses the historically best-performing method
4. Quality Assessment:
5. Generates an embedding using the selected method
6. Evaluates the embedding quality using multiple metrics
7. Computes an overall quality score
8. Adaptation Decision:
9. If quality exceeds the threshold, uses the current embedding
10. If quality falls below the threshold, tries an alternative method
11. Method Switching:
12. Generates a new embedding with the recommended alternative method
13. Evaluates the quality of the new embedding
14. Compares quality scores and selects the better result
15. Performance Tracking:
16. Records method performance for the material
17. Updates historical statistics
18. Refines future method selection based on accumulated knowledge

For previously encountered materials, uses the historically best-performing method

Quality Assessment:

Computes an overall quality score

Adaptation Decision:

If quality falls below the threshold, tries an alternative method

Method Switching:

Compares quality scores and selects the better result

Performance Tracking:

### Performance Optimization

The system continuously optimizes its performance through several mechanisms:

#### Material-Specific Learning

* Maintains a mapping of material IDs to their optimal embedding methods
* Tracks quality scores for each material-method combination
* Adapts method selection based on historical performance

#### Statistical Tracking

* Records quality metrics and processing times
* Computes exponential moving averages to favor recent performance
* Maintains category-specific statistics for tailored decisions

#### Automatic Fallbacks

* Gracefully handles errors in any embedding method
* Provides cascade fallbacks to ensure successful embedding generation
* Tracks fallback events to improve future decisions

### Implementation Components

The implementation consists of two main Python modules:

#### 1. Embedding Quality Evaluator (embedding\_quality\_evaluator.py)

embedding\_quality\_evaluator.py

This module handles quality assessment of embeddings and provides recommendations for method selection:

* EmbeddingQualityMetrics class: Implements various quality evaluation metrics
* EmbeddingQualityEvaluator class: Evaluates embedding quality and recommends methods

EmbeddingQualityMetrics

EmbeddingQualityEvaluator

#### 2. Adaptive Hybrid Embeddings (adaptive\_hybrid\_embeddings.py)

adaptive\_hybrid\_embeddings.py

This module implements the adaptive embedding generation system:

* AdaptiveEmbeddingGenerator class: Core class that handles adaptive method selection
* generate\_adaptive\_embedding function: Primary entry point for generating embeddings

AdaptiveEmbeddingGenerator

generate\_adaptive\_embedding

### Usage

#### Basic Usage

from adaptive\_hybrid\_embeddings import generate\_adaptive\_embedding  
  
# Generate an embedding with adaptive method selection  
result = generate\_adaptive\_embedding(  
 image\_path="path/to/image.jpg",  
 material\_id="example\_material\_123"  
)  
  
# Access the embedding vector  
embedding\_vector = result["vector"]  
  
# Check which method was ultimately used  
final\_method = result["method"]  
  
# Examine quality scores  
quality\_scores = result["quality\_scores"]

from adaptive\_hybrid\_embeddings import generate\_adaptive\_embedding  
  
# Generate an embedding with adaptive method selection  
result = generate\_adaptive\_embedding(  
 image\_path="path/to/image.jpg",  
 material\_id="example\_material\_123"  
)  
  
# Access the embedding vector  
embedding\_vector = result["vector"]  
  
# Check which method was ultimately used  
final\_method = result["method"]  
  
# Examine quality scores  
quality\_scores = result["quality\_scores"]

#### Advanced Configuration

result = generate\_adaptive\_embedding(  
 image\_path="path/to/image.jpg",  
 material\_id="example\_material\_123",  
 method="feature-based", # Initial method suggestion  
 reference\_path="path/to/refs", # Reference embeddings for quality evaluation  
 cache\_dir="path/to/cache", # Cache for performance tracking  
 model\_path="path/to/model", # Custom model for ML-based methods  
 output\_dimensions=256, # Embedding dimensionality  
 quality\_threshold=0.7, # Threshold for method switching  
 adaptive=True # Enable/disable adaptation  
)

result = generate\_adaptive\_embedding(  
 image\_path="path/to/image.jpg",  
 material\_id="example\_material\_123",  
 method="feature-based", # Initial method suggestion  
 reference\_path="path/to/refs", # Reference embeddings for quality evaluation  
 cache\_dir="path/to/cache", # Cache for performance tracking  
 model\_path="path/to/model", # Custom model for ML-based methods  
 output\_dimensions=256, # Embedding dimensionality  
 quality\_threshold=0.7, # Threshold for method switching  
 adaptive=True # Enable/disable adaptation  
)

#### Command Line Interface

The module can also be used from the command line:

python adaptive\_hybrid\_embeddings.py path/to/image.jpg \  
 --material-id example\_material\_123 \  
 --method hybrid \  
 --reference-path path/to/refs \  
 --cache-dir path/to/cache \  
 --quality-threshold 0.65

python adaptive\_hybrid\_embeddings.py path/to/image.jpg \  
 --material-id example\_material\_123 \  
 --method hybrid \  
 --reference-path path/to/refs \  
 --cache-dir path/to/cache \  
 --quality-threshold 0.65

### Reference Data

The system can optionally use reference embeddings to improve quality evaluation and method selection:

* Per-Category References: Collections of known-good embeddings for each material category
* Distribution Statistics: Statistical properties of embeddings for anomaly detection
* Material Categorization: Mapping of material IDs to their categories

### Performance Tracking and Analysis

The system maintains detailed performance statistics:

* Method Usage: Tracks how often each method is used
* Quality Scores: Records average quality for each method
* Processing Times: Monitors computational efficiency
* Method Switches: Counts how often methods are switched
* Material Performance: Maintains material-specific statistics

These statistics can be analyzed to gain insights into system performance and further optimize the embedding generation process.

### Integration with Material Recognizer

The adaptive embedding system is fully integrated with the MaterialRecognizer class, enabling quality-based method switching during material recognition:

from material\_recognizer import MaterialRecognizer  
  
# Initialize material recognizer with adaptive embedding enabled  
recognizer = MaterialRecognizer(  
 method="hybrid",  
 adaptive=True, # Enable adaptive embedding selection  
 quality\_threshold=0.7, # Set quality threshold for method switching  
 use\_gpu=True  
)  
  
# Recognize material with adaptive method selection  
result = recognizer.recognize(  
 image\_path="path/to/image.jpg",   
 material\_id="example\_material\_123" # Optional but enables material-specific optimization  
)  
  
# Access the recognition results  
material\_type = result["material\_type"]  
confidence = result["confidence"]  
embedding = result["embedding"]  
  
# Examine which embedding method was used  
used\_method = result["embedding\_method"]  
quality\_score = result["quality\_score"]  
method\_switches = result["method\_switches"]

from material\_recognizer import MaterialRecognizer  
  
# Initialize material recognizer with adaptive embedding enabled  
recognizer = MaterialRecognizer(  
 method="hybrid",  
 adaptive=True, # Enable adaptive embedding selection  
 quality\_threshold=0.7, # Set quality threshold for method switching  
 use\_gpu=True  
)  
  
# Recognize material with adaptive method selection  
result = recognizer.recognize(  
 image\_path="path/to/image.jpg",   
 material\_id="example\_material\_123" # Optional but enables material-specific optimization  
)  
  
# Access the recognition results  
material\_type = result["material\_type"]  
confidence = result["confidence"]  
embedding = result["embedding"]  
  
# Examine which embedding method was used  
used\_method = result["embedding\_method"]  
quality\_score = result["quality\_score"]  
method\_switches = result["method\_switches"]

#### Material Recognizer Command Line Interface

The MaterialRecognizer CLI now supports adaptive embedding features:

python material\_recognizer.py path/to/image.jpg \  
 --method hybrid \  
 --adaptive \  
 --quality-threshold 0.7 \  
 --material-id example\_material\_123

python material\_recognizer.py path/to/image.jpg \  
 --method hybrid \  
 --adaptive \  
 --quality-threshold 0.7 \  
 --material-id example\_material\_123

#### Web API Integration

When using the material recognition through the web API, the adaptive embedding features can be enabled via query parameters:

POST /api/recognition  
{  
 "image": "base64\_encoded\_image",  
 "adaptive": true,  
 "quality\_threshold": 0.7,  
 "material\_id": "example\_material\_123"  
}

POST /api/recognition  
{  
 "image": "base64\_encoded\_image",  
 "adaptive": true,  
 "quality\_threshold": 0.7,  
 "material\_id": "example\_material\_123"  
}

### Integration with Vector Search

The adaptive embedding system seamlessly integrates with the existing vector search implementation:

from adaptive\_hybrid\_embeddings import generate\_adaptive\_embedding  
from vector\_search import VectorSearchIndex  
  
# Load search index  
index = VectorSearchIndex("path/to/index")  
  
# Generate adaptive embedding  
result = generate\_adaptive\_embedding("path/to/query\_image.jpg")  
query\_embedding = np.array(result["vector"])  
  
# Search for similar materials  
material\_ids, similarities = index.search(query\_embedding, k=5)

from adaptive\_hybrid\_embeddings import generate\_adaptive\_embedding  
from vector\_search import VectorSearchIndex  
  
# Load search index  
index = VectorSearchIndex("path/to/index")  
  
# Generate adaptive embedding  
result = generate\_adaptive\_embedding("path/to/query\_image.jpg")  
query\_embedding = np.array(result["vector"])  
  
# Search for similar materials  
material\_ids, similarities = index.search(query\_embedding, k=5)

### Server Integration

#### Configuration Options

The adaptive embedding system can be configured at the server level through environment variables or configuration files:

{  
 "ml": {  
 "embedding": {  
 "adaptive": true,  
 "defaultMethod": "hybrid",  
 "qualityThreshold": 0.7,  
 "cacheEnabled": true,  
 "cachePath": "./cache/embedding\_performance",  
 "fallbackOrder": ["hybrid", "ml-based", "feature-based"]  
 }  
 }  
}

{  
 "ml": {  
 "embedding": {  
 "adaptive": true,  
 "defaultMethod": "hybrid",  
 "qualityThreshold": 0.7,  
 "cacheEnabled": true,  
 "cachePath": "./cache/embedding\_performance",  
 "fallbackOrder": ["hybrid", "ml-based", "feature-based"]  
 }  
 }  
}

#### Performance Monitoring

The system exposes metrics that can be monitored in real-time:

* Method usage distribution: Percentage of requests using each method
* Quality score averages: Average quality scores per method and material type
* Adaptation events: Frequency of method switching events
* Processing times: Average, min, max processing times per method

These metrics can be visualized in dashboards to track system performance and optimization opportunities.

### Customization

#### Adding New Embedding Methods

The system is extensible and can incorporate new embedding methods:

1. Create a new embedding generator class
2. Add it to the \_initialize\_embedding\_generators method in AdaptiveEmbeddingGenerator
3. Update the available\_methods list in the adaptation logic

\_initialize\_embedding\_generators

AdaptiveEmbeddingGenerator

available\_methods

#### Customizing Quality Metrics

You can customize or add new quality metrics:

1. Add new metric methods to the EmbeddingQualityMetrics class
2. Update the evaluate\_quality method in EmbeddingQualityEvaluator to include new metrics
3. Adjust the weighting in the overall quality score calculation

EmbeddingQualityMetrics

evaluate\_quality

EmbeddingQualityEvaluator

### Best Practices

1. Reference Data: Provide representative reference embeddings for optimal quality evaluation
2. Cache Directory: Enable caching to leverage historical performance data
3. Material IDs: Use consistent material IDs to benefit from material-specific optimization
4. Quality Threshold: Adjust the quality threshold based on your application's requirements
5. Periodic Analysis: Review performance statistics to identify patterns and optimization opportunities

### Technical Specifications

* Embedding Dimensions: Configurable, default is 256
* Quality Threshold: Configurable, default is 0.65
* Supported Methods: feature-based, ml-based (TensorFlow or PyTorch), hybrid
* Caching: Optional file-based caching for performance history
* Threading: Thread-safe implementation for concurrent usage
* Fallbacks: Automatic fallbacks to ensure robustness

### Conclusion

The Adaptive Hybrid Embedding System provides a sophisticated approach to embedding generation that continuously improves over time. By dynamically selecting the optimal method for each material and learning from performance history, the system can generate high-quality embeddings without human intervention, resulting in improved material recognition accuracy.

# Admin Panel

Source: readme/admin-panel.md

---

## Admin Panel

The Admin Panel is a comprehensive management interface for the Kai platform, providing administrators with tools for system configuration, content management, user control, and performance monitoring. This document details the panel's features, architecture, and implementation.

### Features

#### Dashboard and Analytics

The admin panel provides comprehensive system visibility:

1. System Dashboard
2. Real-time system health indicators
3. Resource utilization metrics
4. Active job monitoring
5. Error rate tracking
6. Key performance indicators
7. Usage Analytics
8. User activity metrics
9. Feature utilization statistics
10. Response time tracking
11. Search pattern analysis
12. Content engagement metrics
13. Performance Monitoring
14. Processing throughput metrics
15. Queue status and performance
16. Database query performance
17. API response times
18. Resource consumption patterns

Key performance indicators

Usage Analytics

Content engagement metrics

Performance Monitoring

#### User and Access Management

The admin panel includes comprehensive user management:

1. User Administration
2. User account creation and management
3. Role assignment (admin, manager, user, guest)
4. Permission configuration
5. Activity logging and auditing
6. Bulk user operations
7. Role-Based Access Control
8. Predefined role templates
9. Custom permission sets
10. Feature-level access control
11. Content-level permissions
12. API access management
13. Authentication Management
14. Authentication method configuration
15. Session policy settings
16. Password policy enforcement
17. Multi-factor authentication options
18. Access token management

Bulk user operations

Role-Based Access Control

API access management

Authentication Management

#### Dataset Management

The admin panel provides comprehensive dataset control:

1. Dataset Operations
2. Dataset upload and import (ZIP, CSV)
3. Dataset visualization and exploration
4. Version control and history
5. Quality assessment and metrics
6. Export and distribution capabilities
7. Data Preparation Tools
8. Dataset cleaning and validation
9. Class balancing and organization
10. Data augmentation workflows
11. Synthetic data generation
12. Incremental learning dataset preparation
13. Quality Management
14. Automated quality assessment
15. Issue detection and resolution
16. Class distribution visualization
17. Image quality metrics
18. Dataset comparison tools

Export and distribution capabilities

Data Preparation Tools

Incremental learning dataset preparation

Quality Management

#### Queue Management

The admin panel includes unified queue management:

1. Multi-Queue Interface
2. PDF processing queue management
3. Web crawler queue management
4. Job status monitoring
5. Priority configuration
6. Resource allocation
7. Job Controls
8. Job creation and scheduling
9. Status monitoring and tracking
10. Cancellation and retry operations
11. Result inspection and validation
12. Error handling and resolution
13. Performance Optimization
14. Queue performance analytics
15. Throughput monitoring
16. Bottleneck identification
17. Resource utilization tracking
18. Scheduling optimization tools

Resource allocation

Job Controls

Error handling and resolution

Performance Optimization

#### Knowledge Base Management

The admin panel provides comprehensive knowledge base tools:

1. Content Management
2. Material entry creation and editing
3. Collection and category organization
4. Relationship management
5. Version control and history
6. Bulk operations and imports
7. Taxonomy Management
8. Category hierarchy management
9. Metadata field configuration
10. Attribute management
11. Classification scheme maintenance
12. Controlled vocabulary management
13. Quality Control
14. Content validation tools
15. Consistency checking
16. Duplicate detection
17. Relationship verification
18. Missing data identification

Bulk operations and imports

Taxonomy Management

Controlled vocabulary management

Quality Control

#### Model and Training Management

The admin panel includes ML model management:

1. Model Operations
2. Model training initialization
3. Training progress monitoring
4. Model evaluation and validation
5. Model deployment and activation
6. Version management and rollback
7. Feature Engineering
8. Feature descriptor generation
9. Vector index management
10. Embedding visualization
11. Feature importance analysis
12. Feature selection tools
13. Training Configuration
14. Hyperparameter configuration
15. Dataset selection and preparation
16. Validation strategy setup
17. Performance metric selection
18. Resource allocation management

Version management and rollback

Feature Engineering

Feature selection tools

Training Configuration

#### System Configuration

The admin panel provides system-wide configuration:

1. General Settings
2. System-wide parameters
3. Default values configuration
4. Notification settings
5. Integration management
6. Feature toggles and flags
7. Integration Management
8. External API credential management
9. Service provider configuration
10. Connection testing and validation
11. Usage quota monitoring
12. Authentication management
13. Backup and Maintenance
14. Database backup creation
15. Restore operations
16. System maintenance scheduling
17. Log management and rotation
18. Storage optimization tools

Feature toggles and flags

Integration Management

Authentication management

Backup and Maintenance

### Technical Implementation

#### Admin Panel Architecture

The admin panel is built with Next.js for a performant React-based UI:

// pages/index.tsx - Main entry point with redirection to dashboard  
import { useEffect } from 'react';  
import { useRouter } from 'next/router';  
  
/\*\*  
 \* Admin index page - redirects to dashboard  
 \*/  
export default function AdminIndex() {  
 const router = useRouter();  
  
 useEffect(() => {  
 router.replace('/dashboard');  
 }, [router]);  
  
 return (  
 <div className="flex h-screen items-center justify-center">  
 <div className="text-center">  
 <h1 className="text-2xl font-semibold text-gray-800">Redirecting to dashboard...</h1>  
 <div className="mt-4">  
 <p>Loading the admin dashboard...</p>  
 </div>  
 </div>  
 </div>  
 );  
}

// pages/index.tsx - Main entry point with redirection to dashboard  
import { useEffect } from 'react';  
import { useRouter } from 'next/router';  
  
/\*\*  
 \* Admin index page - redirects to dashboard  
 \*/  
export default function AdminIndex() {  
 const router = useRouter();  
  
 useEffect(() => {  
 router.replace('/dashboard');  
 }, [router]);  
  
 return (  
 <div className="flex h-screen items-center justify-center">  
 <div className="text-center">  
 <h1 className="text-2xl font-semibold text-gray-800">Redirecting to dashboard...</h1>  
 <div className="mt-4">  
 <p>Loading the admin dashboard...</p>  
 </div>  
 </div>  
 </div>  
 );  
}

#### Layout and Navigation

The admin panel uses a consistent layout with sidebar navigation:

// components/Layout.tsx  
import { ReactNode } from 'react';  
import Header from './Header';  
import Sidebar from './Sidebar';  
  
/\*\*  
 \* Layout component for the admin panel  
 \*/  
export default function Layout({ children }: { children: ReactNode }) {  
 return (  
 <div className="min-h-screen bg-gray-100">  
 <div className="flex">  
 <Sidebar />  
 <div className="flex-1">  
 <Header />  
 <main className="p-6">  
 {children}  
 </main>  
 </div>  
 </div>  
 </div>  
 );  
}  
  
// components/Sidebar.tsx  
import Link from 'next/link';  
import { useRouter } from 'next/router';  
import {   
 HomeIcon, UsersIcon, DatabaseIcon,   
 CogIcon, ChartBarIcon, CollectionIcon,  
 CloudUploadIcon, QueueListIcon  
} from '@heroicons/react/24/outline';  
  
/\*\*  
 \* Sidebar component for admin navigation  
 \*/  
export default function Sidebar() {  
 const router = useRouter();  
 const navItems = [  
 { name: 'Dashboard', href: '/dashboard', icon: HomeIcon },  
 { name: 'User Management', href: '/users', icon: UsersIcon },  
 { name: 'Datasets', href: '/datasets', icon: DatabaseIcon },  
 { name: 'Knowledge Base', href: '/knowledge-base', icon: CollectionIcon },  
 { name: 'Queue Management', href: '/queue', icon: QueueListIcon },  
 { name: 'Reports', href: '/reports', icon: ChartBarIcon },  
 { name: 'Settings', href: '/settings', icon: CogIcon },  
 ];  
  
 return (  
 <div className="w-64 bg-white h-screen shadow-sm overflow-y-auto">  
 <div className="flex items-center justify-between h-16 px-4 border-b border-gray-200">  
 <Link href="/dashboard">  
 <div className="flex items-center cursor-pointer">  
 <span className="text-xl font-semibold text-gray-800">Kai Admin</span>  
 </div>  
 </Link>  
 </div>  
 <nav className="mt-4">  
 <ul>  
 {navItems.map((item) => {  
 const isActive = router.pathname === item.href || router.pathname.startsWith(`${item.href}/`);  
 return (  
 <li key={item.name} className="px-2 py-1">  
 <Link href={item.href}>  
 <div className={`flex items-center px-4 py-2 rounded-md ${isActive ? 'bg-blue-50 text-blue-700' : 'text-gray-700 hover:bg-gray-50'}`}>  
 <item.icon className="w-5 h-5 mr-3" />  
 <span>{item.name}</span>  
 </div>  
 </Link>  
 </li>  
 );  
 })}  
 </ul>  
 </nav>  
 </div>  
 );  
}

// components/Layout.tsx  
import { ReactNode } from 'react';  
import Header from './Header';  
import Sidebar from './Sidebar';  
  
/\*\*  
 \* Layout component for the admin panel  
 \*/  
export default function Layout({ children }: { children: ReactNode }) {  
 return (  
 <div className="min-h-screen bg-gray-100">  
 <div className="flex">  
 <Sidebar />  
 <div className="flex-1">  
 <Header />  
 <main className="p-6">  
 {children}  
 </main>  
 </div>  
 </div>  
 </div>  
 );  
}  
  
// components/Sidebar.tsx  
import Link from 'next/link';  
import { useRouter } from 'next/router';  
import {   
 HomeIcon, UsersIcon, DatabaseIcon,   
 CogIcon, ChartBarIcon, CollectionIcon,  
 CloudUploadIcon, QueueListIcon  
} from '@heroicons/react/24/outline';  
  
/\*\*  
 \* Sidebar component for admin navigation  
 \*/  
export default function Sidebar() {  
 const router = useRouter();  
 const navItems = [  
 { name: 'Dashboard', href: '/dashboard', icon: HomeIcon },  
 { name: 'User Management', href: '/users', icon: UsersIcon },  
 { name: 'Datasets', href: '/datasets', icon: DatabaseIcon },  
 { name: 'Knowledge Base', href: '/knowledge-base', icon: CollectionIcon },  
 { name: 'Queue Management', href: '/queue', icon: QueueListIcon },  
 { name: 'Reports', href: '/reports', icon: ChartBarIcon },  
 { name: 'Settings', href: '/settings', icon: CogIcon },  
 ];  
  
 return (  
 <div className="w-64 bg-white h-screen shadow-sm overflow-y-auto">  
 <div className="flex items-center justify-between h-16 px-4 border-b border-gray-200">  
 <Link href="/dashboard">  
 <div className="flex items-center cursor-pointer">  
 <span className="text-xl font-semibold text-gray-800">Kai Admin</span>  
 </div>  
 </Link>  
 </div>  
 <nav className="mt-4">  
 <ul>  
 {navItems.map((item) => {  
 const isActive = router.pathname === item.href || router.pathname.startsWith(`${item.href}/`);  
 return (  
 <li key={item.name} className="px-2 py-1">  
 <Link href={item.href}>  
 <div className={`flex items-center px-4 py-2 rounded-md ${isActive ? 'bg-blue-50 text-blue-700' : 'text-gray-700 hover:bg-gray-50'}`}>  
 <item.icon className="w-5 h-5 mr-3" />  
 <span>{item.name}</span>  
 </div>  
 </Link>  
 </li>  
 );  
 })}  
 </ul>  
 </nav>  
 </div>  
 );  
}

#### Role-Based Access Control

The admin panel implements robust role-based access control:

// middleware/auth.middleware.ts  
import { Request, Response, NextFunction } from 'express';  
import jwt from 'jsonwebtoken';  
import { ApiError } from '../utils/apiError';  
  
/\*\*  
 \* Middleware to authenticate users via JWT  
 \*/  
export const authMiddleware = (req: Request, res: Response, next: NextFunction) => {  
 try {  
 const token = req.headers.authorization?.split(' ')[1];  
  
 if (!token) {  
 throw new ApiError(401, 'Authentication required');  
 }  
  
 const decoded = jwt.verify(token, process.env.JWT\_SECRET as string);  
 req.user = decoded;  
  
 next();  
 } catch (error) {  
 next(new ApiError(401, 'Invalid or expired token'));  
 }  
};  
  
/\*\*  
 \* Middleware to authorize users based on roles  
 \*/  
export const authorizeRoles = (roles: string[]) => {  
 return (req: Request, res: Response, next: NextFunction) => {  
 if (!req.user) {  
 return next(new ApiError(401, 'Authentication required'));  
 }  
  
 if (!roles.includes(req.user.role)) {  
 return next(new ApiError(403, 'Unauthorized access'));  
 }  
  
 next();  
 };  
};  
  
// Usage in routes  
router.use('/admin', authMiddleware, authorizeRoles(['admin']));

// middleware/auth.middleware.ts  
import { Request, Response, NextFunction } from 'express';  
import jwt from 'jsonwebtoken';  
import { ApiError } from '../utils/apiError';  
  
/\*\*  
 \* Middleware to authenticate users via JWT  
 \*/  
export const authMiddleware = (req: Request, res: Response, next: NextFunction) => {  
 try {  
 const token = req.headers.authorization?.split(' ')[1];  
  
 if (!token) {  
 throw new ApiError(401, 'Authentication required');  
 }  
  
 const decoded = jwt.verify(token, process.env.JWT\_SECRET as string);  
 req.user = decoded;  
  
 next();  
 } catch (error) {  
 next(new ApiError(401, 'Invalid or expired token'));  
 }  
};  
  
/\*\*  
 \* Middleware to authorize users based on roles  
 \*/  
export const authorizeRoles = (roles: string[]) => {  
 return (req: Request, res: Response, next: NextFunction) => {  
 if (!req.user) {  
 return next(new ApiError(401, 'Authentication required'));  
 }  
  
 if (!roles.includes(req.user.role)) {  
 return next(new ApiError(403, 'Unauthorized access'));  
 }  
  
 next();  
 };  
};  
  
// Usage in routes  
router.use('/admin', authMiddleware, authorizeRoles(['admin']));

#### API Routes

The admin panel integrates with dedicated API routes:

// routes/admin.routes.ts  
import { Router } from 'express';  
import { authMiddleware, authorizeRoles } from '../middleware/auth.middleware';  
import { asyncHandler } from '../utils/asyncHandler';  
import { Request, Response } from 'express';  
  
// Import the admin routes  
import modelRoutes from './admin/model.routes';  
import queueRoutes from './admin/queue.routes';  
import categoryRoutes from './admin/category.routes';  
import metadataFieldRoutes from './admin/metadataField.routes';  
import knowledgeBaseRoutes from './admin/knowledgeBase.routes';  
import datasetRoutes from './admin/dataset.routes';  
  
const router = Router();  
  
// All routes in this file require admin authentication  
router.use(authMiddleware, authorizeRoles(['admin']));  
  
/\*\*  
 \* @route GET /api/admin/dashboard  
 \* @desc Get admin dashboard statistics  
 \* @access Private (Admin)  
 \*/  
router.get('/dashboard', asyncHandler(async (req: Request, res: Response) => {  
 const stats = await getDashboardStats();  
 res.json(stats);  
}));  
  
// Mount specialized admin routes  
router.use('/models', modelRoutes);  
router.use('/queue', queueRoutes);  
router.use('/categories', categoryRoutes);  
router.use('/metadata-fields', metadataFieldRoutes);  
router.use('/knowledge-base', knowledgeBaseRoutes);  
router.use('/datasets', datasetRoutes);  
  
export default router;

// routes/admin.routes.ts  
import { Router } from 'express';  
import { authMiddleware, authorizeRoles } from '../middleware/auth.middleware';  
import { asyncHandler } from '../utils/asyncHandler';  
import { Request, Response } from 'express';  
  
// Import the admin routes  
import modelRoutes from './admin/model.routes';  
import queueRoutes from './admin/queue.routes';  
import categoryRoutes from './admin/category.routes';  
import metadataFieldRoutes from './admin/metadataField.routes';  
import knowledgeBaseRoutes from './admin/knowledgeBase.routes';  
import datasetRoutes from './admin/dataset.routes';  
  
const router = Router();  
  
// All routes in this file require admin authentication  
router.use(authMiddleware, authorizeRoles(['admin']));  
  
/\*\*  
 \* @route GET /api/admin/dashboard  
 \* @desc Get admin dashboard statistics  
 \* @access Private (Admin)  
 \*/  
router.get('/dashboard', asyncHandler(async (req: Request, res: Response) => {  
 const stats = await getDashboardStats();  
 res.json(stats);  
}));  
  
// Mount specialized admin routes  
router.use('/models', modelRoutes);  
router.use('/queue', queueRoutes);  
router.use('/categories', categoryRoutes);  
router.use('/metadata-fields', metadataFieldRoutes);  
router.use('/knowledge-base', knowledgeBaseRoutes);  
router.use('/datasets', datasetRoutes);  
  
export default router;

#### Dataset Management Implementation

The admin panel includes comprehensive dataset management:

// routes/admin/dataset.routes.ts  
import { Router } from 'express';  
import { authMiddleware, authorizeRoles } from '../../middleware/auth.middleware';  
import { asyncHandler } from '../../utils/asyncHandler';  
import { Request, Response } from 'express';  
import multer from 'multer';  
import supabaseDatasetService from '../../services/supabase/supabase-dataset-service';  
import { zipExtractorService } from '../../services/datasets/zip-extractor.service';  
import { csvParserService } from '../../services/datasets/csv-parser.service';  
import { datasetManagementService } from '../../services/datasets/dataset-management.service';  
import { ApiError } from '../../utils/apiError';  
import { logger } from '../../utils/logger';  
  
const router = Router();  
const upload = multer({ dest: 'uploads/' });  
  
// All routes protected by admin authentication  
router.use(authMiddleware, authorizeRoles(['admin']));  
  
/\*\*  
 \* @route GET /api/admin/datasets  
 \* @desc Get all datasets with pagination  
 \*/  
router.get('/', asyncHandler(async (req: Request, res: Response) => {  
 const page = parseInt(req.query.page as string) || 1;  
 const limit = parseInt(req.query.limit as string) || 10;  
 const query = req.query.q;  
  
 // Search datasets  
 const result = await supabaseDatasetService.searchDatasets({  
 query: query as string,  
 page,  
 limit  
 });  
  
 return res.json(result);  
}));  
  
/\*\*  
 \* @route POST /api/admin/datasets/upload/zip  
 \* @desc Upload and process a ZIP dataset  
 \*/  
router.post('/upload/zip', upload.single('file'), asyncHandler(async (req: Request, res: Response) => {  
 if (!req.file) {  
 throw new ApiError(400, 'No file uploaded');  
 }  
  
 const { name, description } = req.body;  
  
 if (!name) {  
 throw new ApiError(400, 'Dataset name is required');  
 }  
  
 // Process the uploaded ZIP file  
 const result = await zipExtractorService.extractDataset(  
 req.file.path,  
 name,  
 description  
 );  
  
 return res.json(result);  
}));  
  
// Additional routes for dataset management...  
  
export default router;

// routes/admin/dataset.routes.ts  
import { Router } from 'express';  
import { authMiddleware, authorizeRoles } from '../../middleware/auth.middleware';  
import { asyncHandler } from '../../utils/asyncHandler';  
import { Request, Response } from 'express';  
import multer from 'multer';  
import supabaseDatasetService from '../../services/supabase/supabase-dataset-service';  
import { zipExtractorService } from '../../services/datasets/zip-extractor.service';  
import { csvParserService } from '../../services/datasets/csv-parser.service';  
import { datasetManagementService } from '../../services/datasets/dataset-management.service';  
import { ApiError } from '../../utils/apiError';  
import { logger } from '../../utils/logger';  
  
const router = Router();  
const upload = multer({ dest: 'uploads/' });  
  
// All routes protected by admin authentication  
router.use(authMiddleware, authorizeRoles(['admin']));  
  
/\*\*  
 \* @route GET /api/admin/datasets  
 \* @desc Get all datasets with pagination  
 \*/  
router.get('/', asyncHandler(async (req: Request, res: Response) => {  
 const page = parseInt(req.query.page as string) || 1;  
 const limit = parseInt(req.query.limit as string) || 10;  
 const query = req.query.q;  
  
 // Search datasets  
 const result = await supabaseDatasetService.searchDatasets({  
 query: query as string,  
 page,  
 limit  
 });  
  
 return res.json(result);  
}));  
  
/\*\*  
 \* @route POST /api/admin/datasets/upload/zip  
 \* @desc Upload and process a ZIP dataset  
 \*/  
router.post('/upload/zip', upload.single('file'), asyncHandler(async (req: Request, res: Response) => {  
 if (!req.file) {  
 throw new ApiError(400, 'No file uploaded');  
 }  
  
 const { name, description } = req.body;  
  
 if (!name) {  
 throw new ApiError(400, 'Dataset name is required');  
 }  
  
 // Process the uploaded ZIP file  
 const result = await zipExtractorService.extractDataset(  
 req.file.path,  
 name,  
 description  
 );  
  
 return res.json(result);  
}));  
  
// Additional routes for dataset management...  
  
export default router;

#### Queue Management Implementation

The admin panel provides unified queue management:

// routes/admin/queue.routes.ts  
import { Router } from 'express';  
import { authMiddleware, authorizeRoles } from '../../middleware/auth.middleware';  
import { asyncHandler } from '../../utils/asyncHandler';  
import { Request, Response } from 'express';  
import { ApiError } from '../../utils/apiError';  
import {   
 getJobs,   
 getJobById,   
 retryJob,   
 cancelJob,   
 clearQueue,   
 getQueueStats,  
 getSourceFilters  
} from '../../controllers/queue.controller';  
  
const router = Router();  
  
// All routes in this file require admin authentication  
router.use(authMiddleware, authorizeRoles(['admin']));  
  
/\*\*  
 \* @route GET /api/admin/queue  
 \* @desc Get all jobs from both queue systems with filtering  
 \*/  
router.get('/', asyncHandler(async (req: Request, res: Response) => {  
 const queueSystem = req.query.system as 'pdf' | 'crawler' | 'all' | undefined;  
 const status = req.query.status as string | undefined;  
 const page = parseInt(req.query.page as string) || 1;  
 const limit = parseInt(req.query.limit as string) || 10;  
  
 const jobs = await getJobs({  
 queueSystem,  
 status,  
 page,  
 limit  
 });  
  
 res.json(jobs);  
}));  
  
/\*\*  
 \* @route GET /api/admin/queue/stats  
 \* @desc Get statistics for both queue systems  
 \*/  
router.get('/stats', asyncHandler(async (req: Request, res: Response) => {  
 const stats = await getQueueStats();  
 res.json(stats);  
}));  
  
// Additional routes for queue management...  
  
export default router;

// routes/admin/queue.routes.ts  
import { Router } from 'express';  
import { authMiddleware, authorizeRoles } from '../../middleware/auth.middleware';  
import { asyncHandler } from '../../utils/asyncHandler';  
import { Request, Response } from 'express';  
import { ApiError } from '../../utils/apiError';  
import {   
 getJobs,   
 getJobById,   
 retryJob,   
 cancelJob,   
 clearQueue,   
 getQueueStats,  
 getSourceFilters  
} from '../../controllers/queue.controller';  
  
const router = Router();  
  
// All routes in this file require admin authentication  
router.use(authMiddleware, authorizeRoles(['admin']));  
  
/\*\*  
 \* @route GET /api/admin/queue  
 \* @desc Get all jobs from both queue systems with filtering  
 \*/  
router.get('/', asyncHandler(async (req: Request, res: Response) => {  
 const queueSystem = req.query.system as 'pdf' | 'crawler' | 'all' | undefined;  
 const status = req.query.status as string | undefined;  
 const page = parseInt(req.query.page as string) || 1;  
 const limit = parseInt(req.query.limit as string) || 10;  
  
 const jobs = await getJobs({  
 queueSystem,  
 status,  
 page,  
 limit  
 });  
  
 res.json(jobs);  
}));  
  
/\*\*  
 \* @route GET /api/admin/queue/stats  
 \* @desc Get statistics for both queue systems  
 \*/  
router.get('/stats', asyncHandler(async (req: Request, res: Response) => {  
 const stats = await getQueueStats();  
 res.json(stats);  
}));  
  
// Additional routes for queue management...  
  
export default router;

### Integration with Other Systems

#### PDF Processing Integration

The admin panel integrates with the PDF processing system:

1. Queue Management
2. PDF job creation and submission
3. Status monitoring and tracking
4. Result inspection and validation
5. Error handling and resolution
6. Performance analytics
7. Configuration Management
8. OCR settings configuration
9. Processing parameters adjustment
10. Custom extraction rule creation
11. Template management
12. Output format configuration
13. Result Management
14. Extracted content review
15. Manual correction interface
16. Quality assessment tools
17. Approval workflows
18. Batch processing controls

Performance analytics

Configuration Management

Output format configuration

Result Management

#### Web Crawler Integration

The admin panel integrates with the web crawler system:

1. Crawler Management
2. Crawler configuration creation
3. Job scheduling and execution
4. Status monitoring and control
5. Result inspection and validation
6. Performance tuning
7. Credential Management
8. API key storage and encryption
9. Provider configuration
10. Connection testing
11. Usage monitoring
12. Security management
13. Data Integration
14. Crawler-to-training pipeline configuration
15. Data transformation settings
16. Quality threshold configuration
17. Auto-training settings
18. Result analysis tools

Performance tuning

Credential Management

Security management

Data Integration

#### Vector Database Integration

The admin panel integrates with the vector database:

1. Index Management
2. Vector index creation and configuration
3. Embedding dimension configuration
4. Similarity metric selection
5. Index performance monitoring
6. Reindexing and optimization tools
7. Search Configuration
8. Search parameter adjustment
9. Weight configuration for hybrid search
10. Threshold setting for matches
11. Result count configuration
12. Performance monitoring
13. Data Visualization
14. Embedding space visualization
15. Cluster analysis tools
16. Similarity mapping
17. Outlier detection
18. Dimensionality reduction visualization

Reindexing and optimization tools

Search Configuration

Performance monitoring

Data Visualization

#### ML Pipeline Integration

The admin panel integrates with the ML pipeline:

1. Training Management
2. Training job creation and configuration
3. Progress monitoring with visualizations
4. Performance metric tracking
5. Early stopping controls
6. Resource allocation management
7. Model Management
8. Model comparison and selection
9. A/B testing configuration
10. Deployment and activation
11. Version control and rollback
12. Performance monitoring
13. Dataset Pipeline
14. Dataset-to-training workflow management
15. Data preprocessing configuration
16. Validation strategy selection
17. Augmentation parameter tuning
18. Cross-validation configuration

Resource allocation management

Model Management

Performance monitoring

Dataset Pipeline

### Usage Examples

#### Dashboard Navigation

import { useEffect, useState } from 'react';  
import Layout from '../components/Layout';  
import StatCard from '../components/StatCard';  
import QueueStatusChart from '../components/QueueStatusChart';  
import RecentActivityList from '../components/RecentActivityList';  
import SystemHealthIndicator from '../components/SystemHealthIndicator';  
  
export default function DashboardPage() {  
 const [stats, setStats] = useState({  
 users: 0,  
 materials: 0,  
 datasets: 0,  
 activeJobs: 0,  
 systemHealth: 'normal'  
 });  
  
 useEffect(() => {  
 // Fetch dashboard statistics  
 async function fetchStats() {  
 try {  
 const response = await fetch('/api/admin/dashboard');  
 const data = await response.json();  
 setStats(data);  
 } catch (error) {  
 console.error('Failed to fetch dashboard stats:', error);  
 }  
 }  
  
 fetchStats();  
  
 // Set up polling for real-time updates  
 const interval = setInterval(fetchStats, 30000); // Update every 30 seconds  
  
 return () => clearInterval(interval);  
 }, []);  
  
 return (  
 <Layout>  
 <div className="mb-6">  
 <h1 className="text-2xl font-semibold text-gray-800">Dashboard</h1>  
 <p className="text-gray-600">Welcome to the Kai Material Recognition System admin panel.</p>  
 </div>  
  
 <SystemHealthIndicator status={stats.systemHealth} />  
  
 <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 mb-8">  
 <StatCard title="Users" value={stats.users} icon="users" />  
 <StatCard title="Materials" value={stats.materials} icon="database" />  
 <StatCard title="Datasets" value={stats.datasets} icon="folder" />  
 <StatCard title="Active Jobs" value={stats.activeJobs} icon="cog" />  
 </div>  
  
 <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 mb-8">  
 <QueueStatusChart />  
 <RecentActivityList />  
 </div>  
 </Layout>  
 );  
}

import { useEffect, useState } from 'react';  
import Layout from '../components/Layout';  
import StatCard from '../components/StatCard';  
import QueueStatusChart from '../components/QueueStatusChart';  
import RecentActivityList from '../components/RecentActivityList';  
import SystemHealthIndicator from '../components/SystemHealthIndicator';  
  
export default function DashboardPage() {  
 const [stats, setStats] = useState({  
 users: 0,  
 materials: 0,  
 datasets: 0,  
 activeJobs: 0,  
 systemHealth: 'normal'  
 });  
  
 useEffect(() => {  
 // Fetch dashboard statistics  
 async function fetchStats() {  
 try {  
 const response = await fetch('/api/admin/dashboard');  
 const data = await response.json();  
 setStats(data);  
 } catch (error) {  
 console.error('Failed to fetch dashboard stats:', error);  
 }  
 }  
  
 fetchStats();  
  
 // Set up polling for real-time updates  
 const interval = setInterval(fetchStats, 30000); // Update every 30 seconds  
  
 return () => clearInterval(interval);  
 }, []);  
  
 return (  
 <Layout>  
 <div className="mb-6">  
 <h1 className="text-2xl font-semibold text-gray-800">Dashboard</h1>  
 <p className="text-gray-600">Welcome to the Kai Material Recognition System admin panel.</p>  
 </div>  
  
 <SystemHealthIndicator status={stats.systemHealth} />  
  
 <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 mb-8">  
 <StatCard title="Users" value={stats.users} icon="users" />  
 <StatCard title="Materials" value={stats.materials} icon="database" />  
 <StatCard title="Datasets" value={stats.datasets} icon="folder" />  
 <StatCard title="Active Jobs" value={stats.activeJobs} icon="cog" />  
 </div>  
  
 <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 mb-8">  
 <QueueStatusChart />  
 <RecentActivityList />  
 </div>  
 </Layout>  
 );  
}

#### Dataset Upload Management

import { useState } from 'react';  
import { useRouter } from 'next/router';  
import Layout from '../../components/Layout';  
import { Button, TextField, FileUpload, ProgressIndicator, Alert } from '../../components/ui';  
  
export default function DatasetUploadPage() {  
 const router = useRouter();  
 const [file, setFile] = useState<File | null>(null);  
 const [name, setName] = useState('');  
 const [description, setDescription] = useState('');  
 const [loading, setLoading] = useState(false);  
 const [progress, setProgress] = useState(0);  
 const [error, setError] = useState<string | null>(null);  
  
 const handleFileChange = (selectedFile: File | null) => {  
 setFile(selectedFile);  
 // Auto-generate name from filename if not already set  
 if (!name && selectedFile) {  
 setName(selectedFile.name.split('.')[0]);  
 }  
 };  
  
 const handleSubmit = async (e: React.FormEvent) => {  
 e.preventDefault();  
  
 if (!file) {  
 setError('Please select a file to upload');  
 return;  
 }  
  
 if (!name) {  
 setError('Dataset name is required');  
 return;  
 }  
  
 setLoading(true);  
 setError(null);  
  
 // Create form data  
 const formData = new FormData();  
 formData.append('file', file);  
 formData.append('name', name);  
 formData.append('description', description);  
  
 try {  
 // Determine endpoint based on file type  
 const endpoint = file.name.endsWith('.zip')   
 ? '/api/admin/datasets/upload/zip'  
 : '/api/admin/datasets/upload/csv';  
  
 // Upload the dataset  
 const response = await fetch(endpoint, {  
 method: 'POST',  
 body: formData,  
 headers: {  
 'Authorization': `Bearer ${localStorage.getItem('token')}`  
 }  
 });  
  
 if (!response.ok) {  
 const errorData = await response.json();  
 throw new Error(errorData.message || 'Failed to upload dataset');  
 }  
  
 const result = await response.json();  
  
 // Redirect to the dataset details page  
 router.push(`/datasets/${result.dataset.id}`);  
 } catch (error) {  
 setError(error instanceof Error ? error.message : 'An unknown error occurred');  
 setLoading(false);  
 }  
 };  
  
 const handleDownloadTemplate = () => {  
 window.open('/api/admin/datasets/templates/csv', '\_blank');  
 };  
  
 return (  
 <Layout>  
 <div className="mb-6">  
 <h1 className="text-2xl font-semibold text-gray-800">Upload Dataset</h1>  
 <p className="text-gray-600">Upload a new dataset for training and recognition.</p>  
 </div>  
  
 {error && (  
 <Alert type="error" title="Upload Error" message={error} onClose={() => setError(null)} />  
 )}  
  
 <div className="bg-white shadow-sm rounded-lg p-6">  
 <form onSubmit={handleSubmit}>  
 <div className="mb-6">  
 <TextField  
 label="Dataset Name"  
 value={name}  
 onChange={setName}  
 required  
 placeholder="Enter a descriptive name for the dataset"  
 />  
 </div>  
  
 <div className="mb-6">  
 <TextField  
 label="Description"  
 value={description}  
 onChange={setDescription}  
 multiline  
 rows={3}  
 placeholder="Optional description of the dataset"  
 />  
 </div>  
  
 <div className="mb-6">  
 <FileUpload  
 label="Upload Dataset File"  
 accept=".zip,.csv"  
 onChange={handleFileChange}  
 helpText="Accepted formats: ZIP (for image datasets) or CSV (for reference data)"  
 />  
 </div>  
  
 <div className="flex items-center justify-between">  
 <Button   
 type="button"   
 variant="secondary"   
 onClick={handleDownloadTemplate}  
 >  
 Download CSV Template  
 </Button>  
  
 <Button   
 type="submit"   
 variant="primary"   
 disabled={loading || !file || !name}  
 >  
 {loading ? 'Uploading...' : 'Upload Dataset'}  
 </Button>  
 </div>  
  
 {loading && (  
 <div className="mt-4">  
 <ProgressIndicator progress={progress} />  
 <p className="text-sm text-gray-600 mt-2">  
 Uploading and processing dataset. This may take several minutes for large files.  
 </p>  
 </div>  
 )}  
 </form>  
 </div>  
 </Layout>  
 );  
}

import { useState } from 'react';  
import { useRouter } from 'next/router';  
import Layout from '../../components/Layout';  
import { Button, TextField, FileUpload, ProgressIndicator, Alert } from '../../components/ui';  
  
export default function DatasetUploadPage() {  
 const router = useRouter();  
 const [file, setFile] = useState<File | null>(null);  
 const [name, setName] = useState('');  
 const [description, setDescription] = useState('');  
 const [loading, setLoading] = useState(false);  
 const [progress, setProgress] = useState(0);  
 const [error, setError] = useState<string | null>(null);  
  
 const handleFileChange = (selectedFile: File | null) => {  
 setFile(selectedFile);  
 // Auto-generate name from filename if not already set  
 if (!name && selectedFile) {  
 setName(selectedFile.name.split('.')[0]);  
 }  
 };  
  
 const handleSubmit = async (e: React.FormEvent) => {  
 e.preventDefault();  
  
 if (!file) {  
 setError('Please select a file to upload');  
 return;  
 }  
  
 if (!name) {  
 setError('Dataset name is required');  
 return;  
 }  
  
 setLoading(true);  
 setError(null);  
  
 // Create form data  
 const formData = new FormData();  
 formData.append('file', file);  
 formData.append('name', name);  
 formData.append('description', description);  
  
 try {  
 // Determine endpoint based on file type  
 const endpoint = file.name.endsWith('.zip')   
 ? '/api/admin/datasets/upload/zip'  
 : '/api/admin/datasets/upload/csv';  
  
 // Upload the dataset  
 const response = await fetch(endpoint, {  
 method: 'POST',  
 body: formData,  
 headers: {  
 'Authorization': `Bearer ${localStorage.getItem('token')}`  
 }  
 });  
  
 if (!response.ok) {  
 const errorData = await response.json();  
 throw new Error(errorData.message || 'Failed to upload dataset');  
 }  
  
 const result = await response.json();  
  
 // Redirect to the dataset details page  
 router.push(`/datasets/${result.dataset.id}`);  
 } catch (error) {  
 setError(error instanceof Error ? error.message : 'An unknown error occurred');  
 setLoading(false);  
 }  
 };  
  
 const handleDownloadTemplate = () => {  
 window.open('/api/admin/datasets/templates/csv', '\_blank');  
 };  
  
 return (  
 <Layout>  
 <div className="mb-6">  
 <h1 className="text-2xl font-semibold text-gray-800">Upload Dataset</h1>  
 <p className="text-gray-600">Upload a new dataset for training and recognition.</p>  
 </div>  
  
 {error && (  
 <Alert type="error" title="Upload Error" message={error} onClose={() => setError(null)} />  
 )}  
  
 <div className="bg-white shadow-sm rounded-lg p-6">  
 <form onSubmit={handleSubmit}>  
 <div className="mb-6">  
 <TextField  
 label="Dataset Name"  
 value={name}  
 onChange={setName}  
 required  
 placeholder="Enter a descriptive name for the dataset"  
 />  
 </div>  
  
 <div className="mb-6">  
 <TextField  
 label="Description"  
 value={description}  
 onChange={setDescription}  
 multiline  
 rows={3}  
 placeholder="Optional description of the dataset"  
 />  
 </div>  
  
 <div className="mb-6">  
 <FileUpload  
 label="Upload Dataset File"  
 accept=".zip,.csv"  
 onChange={handleFileChange}  
 helpText="Accepted formats: ZIP (for image datasets) or CSV (for reference data)"  
 />  
 </div>  
  
 <div className="flex items-center justify-between">  
 <Button   
 type="button"   
 variant="secondary"   
 onClick={handleDownloadTemplate}  
 >  
 Download CSV Template  
 </Button>  
  
 <Button   
 type="submit"   
 variant="primary"   
 disabled={loading || !file || !name}  
 >  
 {loading ? 'Uploading...' : 'Upload Dataset'}  
 </Button>  
 </div>  
  
 {loading && (  
 <div className="mt-4">  
 <ProgressIndicator progress={progress} />  
 <p className="text-sm text-gray-600 mt-2">  
 Uploading and processing dataset. This may take several minutes for large files.  
 </p>  
 </div>  
 )}  
 </form>  
 </div>  
 </Layout>  
 );  
}

#### Queue Management Interface

import { useEffect, useState } from 'react';  
import Layout from '../../components/Layout';  
import {   
 Table,   
 Button,   
 Select,   
 Pagination,   
 Badge,   
 Dialog,  
 Alert  
} from '../../components/ui';  
import {  
 getQueueJobs,  
 getQueueStats,  
 retryQueueJob,  
 cancelQueueJob,  
 clearQueue  
} from '../../services/queue.service';  
import JobDetailsDialog from '../../components/admin/JobDetailsDialog';  
  
export default function QueueDashboardPage() {  
 const [jobs, setJobs] = useState([]);  
 const [stats, setStats] = useState({  
 pdf: { total: 0, byStatus: {} },  
 crawler: { total: 0, byStatus: {}, byProvider: {} }  
 });  
  
 const [filter, setFilter] = useState({  
 queueSystem: 'all',  
 status: '',  
 page: 1,  
 limit: 10  
 });  
  
 const [detailsDialog, setDetailsDialog] = useState({  
 open: false,  
 jobId: '',  
 system: 'pdf'  
 });  
  
 const [clearQueueDialog, setClearQueueDialog] = useState({  
 open: false,  
 system: 'pdf'  
 });  
  
 const [action, setAction] = useState(null);  
 const [actionSuccess, setActionSuccess] = useState(null);  
 const [error, setError] = useState(null);  
  
 // Load jobs and stats  
 useEffect(() => {  
 fetchJobs();  
 fetchStats();  
 }, [filter]);  
  
 const fetchJobs = async () => {  
 try {  
 const data = await getQueueJobs({  
 system: filter.queueSystem,  
 status: filter.status,  
 page: filter.page,  
 limit: filter.limit  
 });  
  
 setJobs(data.jobs);  
 } catch (err) {  
 setError('Failed to fetch jobs: ' + err.message);  
 }  
 };  
  
 const fetchStats = async () => {  
 try {  
 const statsData = await getQueueStats();  
 setStats(statsData);  
 } catch (err) {  
 setError('Failed to fetch queue statistics: ' + err.message);  
 }  
 };  
  
 // Handle job actions  
 const handleJobAction = async (type, jobId, system) => {  
 setAction({ type, jobId, system });  
 setActionSuccess(null);  
 setError(null);  
  
 try {  
 if (type === 'retry') {  
 const result = await retryQueueJob(jobId, system);  
 setActionSuccess(`Job ${jobId} queued for retry`);  
 } else if (type === 'cancel') {  
 const result = await cancelQueueJob(jobId, system);  
 setActionSuccess(`Job ${jobId} cancelled successfully`);  
 }  
  
 // Refresh data  
 fetchJobs();  
 fetchStats();  
 } catch (err) {  
 setError(`Failed to ${type} job: ${err.message}`);  
 } finally {  
 setAction(null);  
 }  
 };  
  
 // Handle clear queue  
 const handleClearQueue = async (system) => {  
 try {  
 const result = await clearQueue(system);  
 setActionSuccess(`Cleared ${result.count} jobs from ${system} queue`);  
 setClearQueueDialog({ open: false, system: 'pdf' });  
  
 // Refresh data  
 fetchJobs();  
 fetchStats();  
 } catch (err) {  
 setError(`Failed to clear queue: ${err.message}`);  
 }  
 };  
  
 return (  
 <Layout>  
 <div className="mb-6 flex justify-between items-center">  
 <div>  
 <h1 className="text-2xl font-semibold text-gray-800">Queue Management</h1>  
 <p className="text-gray-600">Manage PDF processing and crawler job queues</p>  
 </div>  
  
 <div className="flex space-x-4">  
 <Button   
 variant="secondary"  
 onClick={() => handleOpenClearQueueDialog('pdf')}  
 >  
 Clear PDF Queue  
 </Button>  
  
 <Button   
 variant="secondary"  
 onClick={() => handleOpenClearQueueDialog('crawler')}  
 >  
 Clear Crawler Queue  
 </Button>  
 </div>  
 </div>  
  
 {/\* Status cards \*/}  
 <div className="grid grid-cols-1 md:grid-cols-2 gap-6 mb-8">  
 <div className="bg-white shadow-sm rounded-lg p-6">  
 <h2 className="text-lg font-semibold mb-4">PDF Processing Queue</h2>  
 <div className="flex justify-between items-center">  
 <div>  
 <div className="text-3xl font-bold">  
 {stats.pdf.total}  
 </div>  
 <div className="text-sm text-gray-500">Total Jobs</div>  
 </div>  
 <div>  
 {Object.entries(stats.pdf.byStatus).map(([status, count]) => (  
 <Badge   
 key={status}  
 color={getStatusColor(status)}  
 className="mr-2 mb-2"  
 >  
 {status}: {count}  
 </Badge>  
 ))}  
 </div>  
 </div>  
 </div>  
  
 <div className="bg-white shadow-sm rounded-lg p-6">  
 <h2 className="text-lg font-semibold mb-4">Crawler Queue</h2>  
 <div className="flex justify-between items-center">  
 <div>  
 <div className="text-3xl font-bold">  
 {stats.crawler.total}  
 </div>  
 <div className="text-sm text-gray-500">Total Jobs</div>  
 </div>  
 <div>  
 {Object.entries(stats.crawler.byStatus).map(([status, count]) => (  
 <Badge   
 key={status}  
 color={getStatusColor(status)}  
 className="mr-2 mb-2"  
 >  
 {status}: {count}  
 </Badge>  
 ))}  
 </div>  
 </div>  
 </div>  
 </div>  
  
 {/\* Filter controls \*/}  
 <div className="bg-white shadow-sm rounded-lg p-6 mb-6">  
 <div className="flex flex-wrap gap-4">  
 <div className="w-full md:w-auto">  
 <Select  
 label="Queue System"  
 value={filter.queueSystem}  
 onChange={(value) => setFilter({...filter, queueSystem: value, page: 1})}  
 options={[  
 { value: 'all', label: 'All Systems' },  
 { value: 'pdf', label: 'PDF Processing' },  
 { value: 'crawler', label: 'Web Crawler' }  
 ]}  
 />  
 </div>  
  
 <div className="w-full md:w-auto">  
 <Select  
 label="Status"  
 value={filter.status}  
 onChange={(value) => setFilter({...filter, status: value, page: 1})}  
 options={[  
 { value: '', label: 'All Statuses' },  
 { value: 'pending', label: 'Pending' },  
 { value: 'processing', label: 'Processing' },  
 { value: 'completed', label: 'Completed' },  
 { value: 'failed', label: 'Failed' },  
 { value: 'canceled', label: 'Canceled' }  
 ]}  
 />  
 </div>  
 </div>  
 </div>  
  
 {/\* Jobs table \*/}  
 <div className="bg-white shadow-sm rounded-lg overflow-hidden">  
 <Table  
 columns={[  
 { key: 'id', header: 'Job ID' },  
 { key: 'queueSystem', header: 'System' },  
 { key: 'status', header: 'Status' },  
 { key: 'source', header: 'Source' },  
 { key: 'createdAt', header: 'Created' },  
 { key: 'actions', header: 'Actions' }  
 ]}  
 data={jobs.map(job => ({  
 ...job,  
 status: (  
 <Badge color={getStatusColor(job.status)}>  
 {job.status}  
 </Badge>  
 ),  
 createdAt: new Date(job.createdAt).toLocaleString(),  
 actions: (  
 <div className="flex space-x-2">  
 <Button  
 size="sm"  
 variant="text"  
 onClick={() => handleOpenDetails(job.id, job.queueSystem)}  
 >  
 Details  
 </Button>  
  
 {job.status === 'failed' && (  
 <Button  
 size="sm"  
 variant="secondary"  
 onClick={() => handleJobAction('retry', job.id, job.queueSystem)}  
 disabled={action && action.jobId === job.id}  
 >  
 Retry  
 </Button>  
 )}  
  
 {['pending', 'processing'].includes(job.status) && (  
 <Button  
 size="sm"  
 variant="danger"  
 onClick={() => handleJobAction('cancel', job.id, job.queueSystem)}  
 disabled={action && action.jobId === job.id}  
 >  
 Cancel  
 </Button>  
 )}  
 </div>  
 )  
 }))}  
 emptyState={{  
 message: 'No jobs found matching the current filters',  
 action: {  
 label: 'Clear Filters',  
 onClick: () => setFilter({  
 queueSystem: 'all',  
 status: '',  
 page: 1,  
 limit: 10  
 })  
 }  
 }}  
 />  
  
 <div className="p-4 border-t">  
 <Pagination  
 currentPage={filter.page}  
 pageSize={filter.limit}  
 totalItems={100} // This would come from the API  
 onPageChange={(page) => setFilter({...filter, page})}  
 />  
 </div>  
 </div>  
  
 {/\* Job details dialog \*/}  
 <JobDetailsDialog  
 open={detailsDialog.open}  
 jobId={detailsDialog.jobId}  
 system={detailsDialog.system}  
 onClose={() => setDetailsDialog({...detailsDialog, open: false})}  
 onActionComplete={() => {  
 fetchJobs();  
 fetchStats();  
 }}  
 />  
  
 {/\* Clear queue confirmation dialog \*/}  
 <Dialog  
 open={clearQueueDialog.open}  
 title={`Clear ${clearQueueDialog.system === 'pdf' ? 'PDF Processing' : 'Web Crawler'} Queue`}  
 content={`Are you sure you want to clear all jobs from the ${clearQueueDialog.system} queue? This action cannot be undone.`}  
 actions={[  
 {  
 label: 'Cancel',  
 variant: 'text',  
 onClick: () => setClearQueueDialog({...clearQueueDialog, open: false})  
 },  
 {  
 label: 'Clear Queue',  
 variant: 'danger',  
 onClick: () => handleClearQueue(clearQueueDialog.system)  
 }  
 ]}  
 onClose={() => setClearQueueDialog({...clearQueueDialog, open: false})}  
 />  
  
 {/\* Action messages \*/}  
 {actionSuccess && (  
 <Alert  
 type="success"  
 title="Success"  
 message={actionSuccess}  
 onClose={() => setActionSuccess(null)}  
 className="fixed bottom-4 right-4 z-50"  
 />  
 )}  
  
 {error && (  
 <Alert  
 type="error"  
 title="Error"  
 message={error}  
 onClose={() => setError(null)}  
 className="fixed bottom-4 right-4 z-50"  
 />  
 )}  
 </Layout>  
 );  
}  
  
// Helper function to map status to color  
const getStatusColor = (status) => {  
 switch (status) {  
 case 'completed': return 'green';  
 case 'processing': return 'blue';  
 case 'pending': return 'yellow';  
 case 'failed': return 'red';  
 case 'canceled': return 'gray';  
 default: return 'gray';  
 }  
};

import { useEffect, useState } from 'react';  
import Layout from '../../components/Layout';  
import {   
 Table,   
 Button,   
 Select,   
 Pagination,   
 Badge,   
 Dialog,  
 Alert  
} from '../../components/ui';  
import {  
 getQueueJobs,  
 getQueueStats,  
 retryQueueJob,  
 cancelQueueJob,  
 clearQueue  
} from '../../services/queue.service';  
import JobDetailsDialog from '../../components/admin/JobDetailsDialog';  
  
export default function QueueDashboardPage() {  
 const [jobs, setJobs] = useState([]);  
 const [stats, setStats] = useState({  
 pdf: { total: 0, byStatus: {} },  
 crawler: { total: 0, byStatus: {}, byProvider: {} }  
 });  
  
 const [filter, setFilter] = useState({  
 queueSystem: 'all',  
 status: '',  
 page: 1,  
 limit: 10  
 });  
  
 const [detailsDialog, setDetailsDialog] = useState({  
 open: false,  
 jobId: '',  
 system: 'pdf'  
 });  
  
 const [clearQueueDialog, setClearQueueDialog] = useState({  
 open: false,  
 system: 'pdf'  
 });  
  
 const [action, setAction] = useState(null);  
 const [actionSuccess, setActionSuccess] = useState(null);  
 const [error, setError] = useState(null);  
  
 // Load jobs and stats  
 useEffect(() => {  
 fetchJobs();  
 fetchStats();  
 }, [filter]);  
  
 const fetchJobs = async () => {  
 try {  
 const data = await getQueueJobs({  
 system: filter.queueSystem,  
 status: filter.status,  
 page: filter.page,  
 limit: filter.limit  
 });  
  
 setJobs(data.jobs);  
 } catch (err) {  
 setError('Failed to fetch jobs: ' + err.message);  
 }  
 };  
  
 const fetchStats = async () => {  
 try {  
 const statsData = await getQueueStats();  
 setStats(statsData);  
 } catch (err) {  
 setError('Failed to fetch queue statistics: ' + err.message);  
 }  
 };  
  
 // Handle job actions  
 const handleJobAction = async (type, jobId, system) => {  
 setAction({ type, jobId, system });  
 setActionSuccess(null);  
 setError(null);  
  
 try {  
 if (type === 'retry') {  
 const result = await retryQueueJob(jobId, system);  
 setActionSuccess(`Job ${jobId} queued for retry`);  
 } else if (type === 'cancel') {  
 const result = await cancelQueueJob(jobId, system);  
 setActionSuccess(`Job ${jobId} cancelled successfully`);  
 }  
  
 // Refresh data  
 fetchJobs();  
 fetchStats();  
 } catch (err) {  
 setError(`Failed to ${type} job: ${err.message}`);  
 } finally {  
 setAction(null);  
 }  
 };  
  
 // Handle clear queue  
 const handleClearQueue = async (system) => {  
 try {  
 const result = await clearQueue(system);  
 setActionSuccess(`Cleared ${result.count} jobs from ${system} queue`);  
 setClearQueueDialog({ open: false, system: 'pdf' });  
  
 // Refresh data  
 fetchJobs();  
 fetchStats();  
 } catch (err) {  
 setError(`Failed to clear queue: ${err.message}`);  
 }  
 };  
  
 return (  
 <Layout>  
 <div className="mb-6 flex justify-between items-center">  
 <div>  
 <h1 className="text-2xl font-semibold text-gray-800">Queue Management</h1>  
 <p className="text-gray-600">Manage PDF processing and crawler job queues</p>  
 </div>  
  
 <div className="flex space-x-4">  
 <Button   
 variant="secondary"  
 onClick={() => handleOpenClearQueueDialog('pdf')}  
 >  
 Clear PDF Queue  
 </Button>  
  
 <Button   
 variant="secondary"  
 onClick={() => handleOpenClearQueueDialog('crawler')}  
 >  
 Clear Crawler Queue  
 </Button>  
 </div>  
 </div>  
  
 {/\* Status cards \*/}  
 <div className="grid grid-cols-1 md:grid-cols-2 gap-6 mb-8">  
 <div className="bg-white shadow-sm rounded-lg p-6">  
 <h2 className="text-lg font-semibold mb-4">PDF Processing Queue</h2>  
 <div className="flex justify-between items-center">  
 <div>  
 <div className="text-3xl font-bold">  
 {stats.pdf.total}  
 </div>  
 <div className="text-sm text-gray-500">Total Jobs</div>  
 </div>  
 <div>  
 {Object.entries(stats.pdf.byStatus).map(([status, count]) => (  
 <Badge   
 key={status}  
 color={getStatusColor(status)}  
 className="mr-2 mb-2"  
 >  
 {status}: {count}  
 </Badge>  
 ))}  
 </div>  
 </div>  
 </div>  
  
 <div className="bg-white shadow-sm rounded-lg p-6">  
 <h2 className="text-lg font-semibold mb-4">Crawler Queue</h2>  
 <div className="flex justify-between items-center">  
 <div>  
 <div className="text-3xl font-bold">  
 {stats.crawler.total}  
 </div>  
 <div className="text-sm text-gray-500">Total Jobs</div>  
 </div>  
 <div>  
 {Object.entries(stats.crawler.byStatus).map(([status, count]) => (  
 <Badge   
 key={status}  
 color={getStatusColor(status)}  
 className="mr-2 mb-2"  
 >  
 {status}: {count}  
 </Badge>  
 ))}  
 </div>  
 </div>  
 </div>  
 </div>  
  
 {/\* Filter controls \*/}  
 <div className="bg-white shadow-sm rounded-lg p-6 mb-6">  
 <div className="flex flex-wrap gap-4">  
 <div className="w-full md:w-auto">  
 <Select  
 label="Queue System"  
 value={filter.queueSystem}  
 onChange={(value) => setFilter({...filter, queueSystem: value, page: 1})}  
 options={[  
 { value: 'all', label: 'All Systems' },  
 { value: 'pdf', label: 'PDF Processing' },  
 { value: 'crawler', label: 'Web Crawler' }  
 ]}  
 />  
 </div>  
  
 <div className="w-full md:w-auto">  
 <Select  
 label="Status"  
 value={filter.status}  
 onChange={(value) => setFilter({...filter, status: value, page: 1})}  
 options={[  
 { value: '', label: 'All Statuses' },  
 { value: 'pending', label: 'Pending' },  
 { value: 'processing', label: 'Processing' },  
 { value: 'completed', label: 'Completed' },  
 { value: 'failed', label: 'Failed' },  
 { value: 'canceled', label: 'Canceled' }  
 ]}  
 />  
 </div>  
 </div>  
 </div>  
  
 {/\* Jobs table \*/}  
 <div className="bg-white shadow-sm rounded-lg overflow-hidden">  
 <Table  
 columns={[  
 { key: 'id', header: 'Job ID' },  
 { key: 'queueSystem', header: 'System' },  
 { key: 'status', header: 'Status' },  
 { key: 'source', header: 'Source' },  
 { key: 'createdAt', header: 'Created' },  
 { key: 'actions', header: 'Actions' }  
 ]}  
 data={jobs.map(job => ({  
 ...job,  
 status: (  
 <Badge color={getStatusColor(job.status)}>  
 {job.status}  
 </Badge>  
 ),  
 createdAt: new Date(job.createdAt).toLocaleString(),  
 actions: (  
 <div className="flex space-x-2">  
 <Button  
 size="sm"  
 variant="text"  
 onClick={() => handleOpenDetails(job.id, job.queueSystem)}  
 >  
 Details  
 </Button>  
  
 {job.status === 'failed' && (  
 <Button  
 size="sm"  
 variant="secondary"  
 onClick={() => handleJobAction('retry', job.id, job.queueSystem)}  
 disabled={action && action.jobId === job.id}  
 >  
 Retry  
 </Button>  
 )}  
  
 {['pending', 'processing'].includes(job.status) && (  
 <Button  
 size="sm"  
 variant="danger"  
 onClick={() => handleJobAction('cancel', job.id, job.queueSystem)}  
 disabled={action && action.jobId === job.id}  
 >  
 Cancel  
 </Button>  
 )}  
 </div>  
 )  
 }))}  
 emptyState={{  
 message: 'No jobs found matching the current filters',  
 action: {  
 label: 'Clear Filters',  
 onClick: () => setFilter({  
 queueSystem: 'all',  
 status: '',  
 page: 1,  
 limit: 10  
 })  
 }  
 }}  
 />  
  
 <div className="p-4 border-t">  
 <Pagination  
 currentPage={filter.page}  
 pageSize={filter.limit}  
 totalItems={100} // This would come from the API  
 onPageChange={(page) => setFilter({...filter, page})}  
 />  
 </div>  
 </div>  
  
 {/\* Job details dialog \*/}  
 <JobDetailsDialog  
 open={detailsDialog.open}  
 jobId={detailsDialog.jobId}  
 system={detailsDialog.system}  
 onClose={() => setDetailsDialog({...detailsDialog, open: false})}  
 onActionComplete={() => {  
 fetchJobs();  
 fetchStats();  
 }}  
 />  
  
 {/\* Clear queue confirmation dialog \*/}  
 <Dialog  
 open={clearQueueDialog.open}  
 title={`Clear ${clearQueueDialog.system === 'pdf' ? 'PDF Processing' : 'Web Crawler'} Queue`}  
 content={`Are you sure you want to clear all jobs from the ${clearQueueDialog.system} queue? This action cannot be undone.`}  
 actions={[  
 {  
 label: 'Cancel',  
 variant: 'text',  
 onClick: () => setClearQueueDialog({...clearQueueDialog, open: false})  
 },  
 {  
 label: 'Clear Queue',  
 variant: 'danger',  
 onClick: () => handleClearQueue(clearQueueDialog.system)  
 }  
 ]}  
 onClose={() => setClearQueueDialog({...clearQueueDialog, open: false})}  
 />  
  
 {/\* Action messages \*/}  
 {actionSuccess && (  
 <Alert  
 type="success"  
 title="Success"  
 message={actionSuccess}  
 onClose={() => setActionSuccess(null)}  
 className="fixed bottom-4 right-4 z-50"  
 />  
 )}  
  
 {error && (  
 <Alert  
 type="error"  
 title="Error"  
 message={error}  
 onClose={() => setError(null)}  
 className="fixed bottom-4 right-4 z-50"  
 />  
 )}  
 </Layout>  
 );  
}  
  
// Helper function to map status to color  
const getStatusColor = (status) => {  
 switch (status) {  
 case 'completed': return 'green';  
 case 'processing': return 'blue';  
 case 'pending': return 'yellow';  
 case 'failed': return 'red';  
 case 'canceled': return 'gray';  
 default: return 'gray';  
 }  
};

### Configuration

#### Deployment

The admin panel can be deployed in various configurations:

1. Development Environment
2. Local development server with hot reloading
3. Environment variables for development settings
4. Mock data for rapid UI development
5. Debugging tools enabled
6. Performance profiling available
7. Production Deployment
8. Optimized production build with code splitting
9. Server-side rendering for improved performance
10. Content delivery network integration
11. Secure cookie settings
12. Error monitoring and reporting
13. Customization Options
14. Branding and theme customization
15. Feature toggle configuration
16. Role permission mapping
17. Notification settings
18. Integration points with external systems

Performance profiling available

Production Deployment

Error monitoring and reporting

Customization Options

#### Access Control Configuration

The admin panel supports customizable access control:

// Settings for role-based access control  
const rolePermissions = {  
 admin: {  
 // Full access to all features  
 dashboard: ['view', 'manage'],  
 users: ['view', 'create', 'edit', 'delete'],  
 datasets: ['view', 'create', 'edit', 'delete', 'upload', 'train'],  
 knowledgeBase: ['view', 'create', 'edit', 'delete', 'import', 'export'],  
 queue: ['view', 'manage', 'cancel', 'retry', 'clear'],  
 settings: ['view', 'edit']  
 },  
 manager: {  
 // Limited administrative access  
 dashboard: ['view'],  
 users: ['view'],  
 datasets: ['view', 'create', 'edit', 'upload', 'train'],  
 knowledgeBase: ['view', 'create', 'edit', 'import'],  
 queue: ['view', 'manage', 'cancel', 'retry'],  
 settings: ['view']  
 },  
 user: {  
 // Basic access for regular users  
 dashboard: ['view'],  
 datasets: ['view'],  
 knowledgeBase: ['view'],  
 queue: ['view']  
 }  
};  
  
// Configuration for authentication methods  
const authConfig = {  
 methods: ['jwt', 'oauth'],  
 sessionTimeout: 3600, // 1 hour  
 refreshTokens: true,  
 passwordPolicy: {  
 minLength: 10,  
 requireSpecialChars: true,  
 requireNumbers: true,  
 requireUppercase: true,  
 expiration: 90 // days  
 },  
 mfa: {  
 enabled: true,  
 methods: ['app', 'email']  
 }  
};

// Settings for role-based access control  
const rolePermissions = {  
 admin: {  
 // Full access to all features  
 dashboard: ['view', 'manage'],  
 users: ['view', 'create', 'edit', 'delete'],  
 datasets: ['view', 'create', 'edit', 'delete', 'upload', 'train'],  
 knowledgeBase: ['view', 'create', 'edit', 'delete', 'import', 'export'],  
 queue: ['view', 'manage', 'cancel', 'retry', 'clear'],  
 settings: ['view', 'edit']  
 },  
 manager: {  
 // Limited administrative access  
 dashboard: ['view'],  
 users: ['view'],  
 datasets: ['view', 'create', 'edit', 'upload', 'train'],  
 knowledgeBase: ['view', 'create', 'edit', 'import'],  
 queue: ['view', 'manage', 'cancel', 'retry'],  
 settings: ['view']  
 },  
 user: {  
 // Basic access for regular users  
 dashboard: ['view'],  
 datasets: ['view'],  
 knowledgeBase: ['view'],  
 queue: ['view']  
 }  
};  
  
// Configuration for authentication methods  
const authConfig = {  
 methods: ['jwt', 'oauth'],  
 sessionTimeout: 3600, // 1 hour  
 refreshTokens: true,  
 passwordPolicy: {  
 minLength: 10,  
 requireSpecialChars: true,  
 requireNumbers: true,  
 requireUppercase: true,  
 expiration: 90 // days  
 },  
 mfa: {  
 enabled: true,  
 methods: ['app', 'email']  
 }  
};

#### Notification Configuration

The admin panel includes configurable notifications:

// Notification settings for admin users  
const notificationConfig = {  
 channels: {  
 email: {  
 enabled: true,  
 throttle: 3600 // seconds between similar notifications  
 },  
 inApp: {  
 enabled: true,  
 maxUnread: 100  
 },  
 slack: {  
 enabled: false,  
 webhookUrl: ''  
 }  
 },  
 events: {  
 userActivity: {  
 loginAttempts: {  
 failed: {  
 threshold: 5,  
 channels: ['email', 'inApp']  
 }  
 }  
 },  
 system: {  
 queueErrors: {  
 threshold: 10,  
 channels: ['email', 'inApp']  
 },  
 storageWarning: {  
 threshold: 0.9, // 90% usage  
 channels: ['email', 'inApp']  
 }  
 },  
 training: {  
 modelCompleted: {  
 channels: ['inApp']  
 },  
 modelError: {  
 channels: ['email', 'inApp']  
 }  
 }  
 },  
 recipients: {  
 admins: ['admin@example.com'],  
 technicalTeam: ['tech@example.com']  
 }  
};

// Notification settings for admin users  
const notificationConfig = {  
 channels: {  
 email: {  
 enabled: true,  
 throttle: 3600 // seconds between similar notifications  
 },  
 inApp: {  
 enabled: true,  
 maxUnread: 100  
 },  
 slack: {  
 enabled: false,  
 webhookUrl: ''  
 }  
 },  
 events: {  
 userActivity: {  
 loginAttempts: {  
 failed: {  
 threshold: 5,  
 channels: ['email', 'inApp']  
 }  
 }  
 },  
 system: {  
 queueErrors: {  
 threshold: 10,  
 channels: ['email', 'inApp']  
 },  
 storageWarning: {  
 threshold: 0.9, // 90% usage  
 channels: ['email', 'inApp']  
 }  
 },  
 training: {  
 modelCompleted: {  
 channels: ['inApp']  
 },  
 modelError: {  
 channels: ['email', 'inApp']  
 }  
 }  
 },  
 recipients: {  
 admins: ['admin@example.com'],  
 technicalTeam: ['tech@example.com']  
 }  
};

#### Performance Considerations

1. UI Optimization
2. Code splitting for faster initial loading
3. Virtualized lists for large datasets
4. Lazy loading for secondary content
5. Debounced search inputs
6. Optimized rendering for large tables
7. API Efficiency
8. Pagination for large result sets
9. Filtered queries to minimize data transfer
10. Caching for frequently accessed data
11. ETags for conditional requests
12. Compression for response payload
13. Authentication Performance
14. Token-based authentication for stateless scaling
15. Refresh token rotation for long sessions
16. Permission caching for access checks
17. Rate limiting for security
18. Efficient role hierarchy traversal

Optimized rendering for large tables

API Efficiency

Compression for response payload

Authentication Performance

# Advanced Alerting Condition Types

Source: readme/advanced-alerting-condition-types.md

---

## Advanced Alerting Condition Types

This document describes the advanced condition types implemented in the alerting service. These condition types provide more sophisticated alerting capabilities, enabling proactive monitoring and issue detection.

### Overview

The alerting service now supports several advanced condition types:

1. Trend Condition: Detects trends in metrics over time
2. Anomaly Condition: Detects anomalies in metrics using statistical methods
3. Composite Condition: Combines multiple conditions with logical operators
4. Dynamic Threshold Condition: Uses dynamic thresholds based on historical data

Additionally, the service now supports advanced aggregation functions:

1. Median: Calculates the median value
2. Percentiles (P90, P95, P99): Calculates percentile values
3. Standard Deviation: Calculates the standard deviation
4. Variance: Calculates the variance

### Advanced Condition Types

#### Trend Condition

The trend condition detects trends in metrics over time. It uses linear regression to calculate the slope of the trend line and compares it to a threshold.

##### Properties

* type: AlertRuleConditionType.TREND
* metric: The metric to monitor (required)
* timeWindow: The time window in seconds (optional)
* properties: Additional properties (required)
* trendDirection: The direction of the trend (required)  
  TrendDirection.INCREASING: Increasing trend  
  TrendDirection.DECREASING: Decreasing trend  
  TrendDirection.STABLE: Stable trend
* trendThreshold: The threshold for the trend slope (optional, default: 0)

AlertRuleConditionType.TREND

* TrendDirection.INCREASING: Increasing trend
* TrendDirection.DECREASING: Decreasing trend
* TrendDirection.STABLE: Stable trend

TrendDirection.INCREASING

TrendDirection.DECREASING

TrendDirection.STABLE

##### Example

const trendCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.TREND,  
 metric: 'response\_time',  
 timeWindow: 3600, // 1 hour  
 properties: {  
 trendDirection: TrendDirection.INCREASING,  
 trendThreshold: 0.1  
 }  
};

const trendCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.TREND,  
 metric: 'response\_time',  
 timeWindow: 3600, // 1 hour  
 properties: {  
 trendDirection: TrendDirection.INCREASING,  
 trendThreshold: 0.1  
 }  
};

##### Implementation

The trend condition is implemented as follows:

1. Filter events within the time window if specified
2. Extract metric values from events (from properties or measurements)
3. Calculate the linear regression slope
4. Compare the slope to the trend threshold based on the trend direction

#### Anomaly Condition

The anomaly condition detects anomalies in metrics using statistical methods. It calculates the z-score of recent values compared to historical values and triggers if the z-score exceeds a threshold.

##### Properties

* type: AlertRuleConditionType.ANOMALY
* metric: The metric to monitor (required)
* properties: Additional properties (optional)
* sensitivity: The sensitivity of the anomaly detection (optional, default: 0.5)
* trainingWindow: The training window in seconds (optional, default: 24 hours)

AlertRuleConditionType.ANOMALY

##### Example

const anomalyCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.ANOMALY,  
 metric: 'response\_time',  
 properties: {  
 sensitivity: 0.7,  
 trainingWindow: 86400 // 24 hours  
 }  
};

const anomalyCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.ANOMALY,  
 metric: 'response\_time',  
 properties: {  
 sensitivity: 0.7,  
 trainingWindow: 86400 // 24 hours  
 }  
};

##### Implementation

The anomaly condition is implemented as follows:

1. Filter events within the training window
2. Split events into training and test sets
3. Calculate the mean and standard deviation of the training values
4. Calculate the z-scores of the test values
5. Check if any z-score exceeds the threshold (based on sensitivity)

#### Composite Condition

The composite condition combines multiple conditions with logical operators. It allows for complex conditions that can't be expressed using a single condition.

##### Properties

* type: AlertRuleConditionType.COMPOSITE
* logicalOperator: The logical operator to use (required)
* LogicalOperator.AND: All conditions must be met
* LogicalOperator.OR: At least one condition must be met
* LogicalOperator.NOT: The condition must not be met
* conditions: The child conditions to combine (required)

AlertRuleConditionType.COMPOSITE

LogicalOperator.AND

LogicalOperator.OR

LogicalOperator.NOT

##### Example

const compositeCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.COMPOSITE,  
 logicalOperator: LogicalOperator.AND,  
 conditions: [  
 {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt'  
 },  
 {  
 type: AlertRuleConditionType.FREQUENCY,  
 timeWindow: 300, // 5 minutes  
 minCount: 5  
 }  
 ]  
};

const compositeCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.COMPOSITE,  
 logicalOperator: LogicalOperator.AND,  
 conditions: [  
 {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt'  
 },  
 {  
 type: AlertRuleConditionType.FREQUENCY,  
 timeWindow: 300, // 5 minutes  
 minCount: 5  
 }  
 ]  
};

##### Implementation

The composite condition is implemented as follows:

1. Evaluate each child condition
2. Combine the results based on the logical operator

#### Dynamic Threshold Condition

The dynamic threshold condition uses dynamic thresholds based on historical data. It calculates the mean and standard deviation of historical values and triggers if the current value exceeds the dynamic threshold.

##### Properties

* type: AlertRuleConditionType.DYNAMIC\_THRESHOLD
* metric: The metric to monitor (required)
* operator: The comparison operator (required)
* properties: Additional properties (optional)
* baselinePeriod: The baseline period in seconds (optional, default: 24 hours)
* deviationFactor: The deviation factor (optional, default: 2)
* aggregation: The aggregation function for current values (optional, default: 'avg')

AlertRuleConditionType.DYNAMIC\_THRESHOLD

##### Example

const dynamicThresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.DYNAMIC\_THRESHOLD,  
 metric: 'response\_time',  
 operator: 'gt',  
 properties: {  
 baselinePeriod: 86400, // 24 hours  
 deviationFactor: 3,  
 aggregation: AggregationFunction.AVG  
 }  
};

const dynamicThresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.DYNAMIC\_THRESHOLD,  
 metric: 'response\_time',  
 operator: 'gt',  
 properties: {  
 baselinePeriod: 86400, // 24 hours  
 deviationFactor: 3,  
 aggregation: AggregationFunction.AVG  
 }  
};

##### Implementation

The dynamic threshold condition is implemented as follows:

1. Filter events within the baseline period
2. Calculate the mean and standard deviation of the baseline values
3. Calculate the dynamic threshold (mean ± deviationFactor \* stdDev)
4. Calculate the aggregate value for current values
5. Compare the current value to the dynamic threshold

### Advanced Aggregation Functions

The alerting service now supports several advanced aggregation functions:

#### Median

Calculates the median value of a set of metrics.

const thresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt',  
 properties: {  
 aggregation: AggregationFunction.MEDIAN  
 }  
};

const thresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt',  
 properties: {  
 aggregation: AggregationFunction.MEDIAN  
 }  
};

#### Percentiles (P90, P95, P99)

Calculates the 90th, 95th, or 99th percentile of a set of metrics.

const thresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt',  
 properties: {  
 aggregation: AggregationFunction.P95  
 }  
};

const thresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt',  
 properties: {  
 aggregation: AggregationFunction.P95  
 }  
};

#### Standard Deviation

Calculates the standard deviation of a set of metrics.

const thresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 100,  
 operator: 'gt',  
 properties: {  
 aggregation: AggregationFunction.STDDEV  
 }  
};

const thresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 100,  
 operator: 'gt',  
 properties: {  
 aggregation: AggregationFunction.STDDEV  
 }  
};

#### Variance

Calculates the variance of a set of metrics.

const thresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 10000,  
 operator: 'gt',  
 properties: {  
 aggregation: AggregationFunction.VARIANCE  
 }  
};

const thresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 10000,  
 operator: 'gt',  
 properties: {  
 aggregation: AggregationFunction.VARIANCE  
 }  
};

### Multiple Metrics Support

The alerting service now supports conditions that involve multiple metrics. This allows for more complex conditions that compare multiple metrics.

const compositeCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.COMPOSITE,  
 logicalOperator: LogicalOperator.AND,  
 conditions: [  
 {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt'  
 },  
 {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'error\_rate',  
 threshold: 0.05,  
 operator: 'gt'  
 }  
 ]  
};

const compositeCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.COMPOSITE,  
 logicalOperator: LogicalOperator.AND,  
 conditions: [  
 {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt'  
 },  
 {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'error\_rate',  
 threshold: 0.05,  
 operator: 'gt'  
 }  
 ]  
};

### Benefits

The implementation of advanced condition types provides several benefits:

1. Sophisticated Alerting: Support for advanced condition types allows for more sophisticated alerting
2. Trend Detection: Trend conditions allow for detecting trends in metrics over time
3. Anomaly Detection: Anomaly conditions allow for detecting anomalies in metrics
4. Complex Conditions: Composite conditions allow for complex conditions that combine multiple conditions
5. Dynamic Thresholds: Dynamic threshold conditions allow for thresholds that adapt to historical data
6. Advanced Aggregation: Support for advanced aggregation functions allows for more precise alerting
7. Multiple Metrics: Support for multiple metrics allows for conditions that compare multiple metrics

### Next Steps

The following steps are recommended to further improve the alerting service:

1. Add More Condition Types: Add support for more condition types (seasonality, correlation, etc.)
2. Improve Anomaly Detection: Improve the anomaly detection algorithm with more sophisticated methods
3. Add Support for Machine Learning: Add support for machine learning models for anomaly detection
4. Add Support for Time Series Forecasting: Add support for time series forecasting for predictive alerting
5. Add Support for Alert Correlation: Add support for correlating alerts to reduce noise
6. Add Support for Alert Suppression: Add support for suppressing alerts based on maintenance windows or other criteria
7. Add Support for Alert Escalation: Add support for escalating alerts based on severity and time

# Advanced Property Validation

Source: readme/advanced-property-validation.md

---

## Advanced Property Validation

This document describes the Advanced Property Validation feature, which provides sophisticated validation rules for material properties to ensure data consistency and accuracy.

### Overview

Advanced Property Validation enables the definition and enforcement of complex validation rules for material properties. It helps maintain data quality by validating property values against defined constraints, ensuring that all property data meets the required standards.

Key capabilities include:

1. Multiple Validation Types: Support for various validation types including range, pattern, enumeration, dependency, custom, and composite validations
2. Conditional Validation: Ability to define validation rules that depend on the values of other properties
3. Severity Levels: Different severity levels for validation results (error, warning, info)
4. Composite Rules: Ability to combine multiple validation rules using logical operators (AND, OR, NOT)
5. Real-time Validation: Integration with property forms for real-time validation feedback

### Architecture

The Advanced Property Validation feature consists of the following components:

#### Database Schema

* validation\_rules: Stores validation rule definitions, including type-specific configuration
* validation\_rule\_dependencies: Stores relationships between composite validation rules and their component rules
* validation\_results: Stores validation results for analytics purposes

#### Types

The following validation rule types are supported:

1. Range Validation: Validates numeric values against minimum and maximum constraints
2. Parameters: min, max, step, unit
3. Pattern Validation: Validates string values against regular expression patterns
4. Parameters: pattern, flags
5. Enumeration Validation: Validates values against a list of allowed values
6. Parameters: allowedValues
7. Dependency Validation: Validates values based on the values of other properties
8. Parameters: condition (property, operator, value), requiredValue/requiredPattern/requiredRange
9. Custom Validation: Applies custom validation functions
10. Parameters: functionName, parameters
11. Composite Validation: Combines multiple validation rules with logical operators
12. Parameters: operator (AND, OR, NOT), rules

Parameters: min, max, step, unit

Pattern Validation: Validates string values against regular expression patterns

Parameters: pattern, flags

Enumeration Validation: Validates values against a list of allowed values

Parameters: allowedValues

Dependency Validation: Validates values based on the values of other properties

Parameters: condition (property, operator, value), requiredValue/requiredPattern/requiredRange

Custom Validation: Applies custom validation functions

Parameters: functionName, parameters

Composite Validation: Combines multiple validation rules with logical operators

#### API Endpoints

The following API endpoints are available for managing validation rules:

##### Validation Rules

* GET /api/validation/rules: Get validation rules with filtering options
* GET /api/validation/rules/:id: Get a validation rule by ID
* POST /api/validation/rules: Create a new validation rule
* PUT /api/validation/rules/:id: Update a validation rule
* DELETE /api/validation/rules/:id: Delete a validation rule

GET /api/validation/rules

GET /api/validation/rules/:id

POST /api/validation/rules

PUT /api/validation/rules/:id

DELETE /api/validation/rules/:id

##### Validation

* POST /api/validation/validate: Validate a property value
* POST /api/validation/validate-batch: Validate multiple properties in batch

POST /api/validation/validate

POST /api/validation/validate-batch

#### Client Components

The following client components are available for working with validation rules:

* ValidationRuleManager: Admin component for managing validation rules
* ValidationRuleForm: Form for creating and editing validation rules
* ValidationTester: Component for testing validation rules with sample values
* ValidationDisplay: Component for displaying validation results

### Usage

#### Creating a Validation Rule

Validation rules can be created through the admin interface or by using the API.

// Example: Creating a range validation rule  
const rangeRule = {  
 name: 'Thickness Range',  
 description: 'Validates that thickness is within the acceptable range',  
 type: 'range',  
 propertyName: 'thickness',  
 materialType: 'tile',  
 severity: 'error',  
 message: 'Thickness must be between 5mm and 20mm',  
 min: 5,  
 max: 20,  
 unit: 'mm'  
};  
  
const response = await fetch('/api/validation/rules', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(rangeRule)  
});  
  
const data = await response.json();  
console.log('Created rule:', data.rule);

// Example: Creating a range validation rule  
const rangeRule = {  
 name: 'Thickness Range',  
 description: 'Validates that thickness is within the acceptable range',  
 type: 'range',  
 propertyName: 'thickness',  
 materialType: 'tile',  
 severity: 'error',  
 message: 'Thickness must be between 5mm and 20mm',  
 min: 5,  
 max: 20,  
 unit: 'mm'  
};  
  
const response = await fetch('/api/validation/rules', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(rangeRule)  
});  
  
const data = await response.json();  
console.log('Created rule:', data.rule);

#### Validating a Property Value

Property values can be validated using the validation API.

// Example: Validating a property value  
const validationRequest = {  
 propertyName: 'thickness',  
 value: 25,  
 materialType: 'tile'  
};  
  
const response = await fetch('/api/validation/validate', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(validationRequest)  
});  
  
const data = await response.json();  
console.log('Validation passed:', data.isValid);  
console.log('Validation results:', data.results);

// Example: Validating a property value  
const validationRequest = {  
 propertyName: 'thickness',  
 value: 25,  
 materialType: 'tile'  
};  
  
const response = await fetch('/api/validation/validate', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(validationRequest)  
});  
  
const data = await response.json();  
console.log('Validation passed:', data.isValid);  
console.log('Validation results:', data.results);

#### Batch Validation

Multiple properties can be validated in a single request.

// Example: Batch validation  
const batchValidationRequest = {  
 materialType: 'tile',  
 properties: {  
 thickness: 15,  
 width: 300,  
 finish: 'matte'  
 }  
};  
  
const response = await fetch('/api/validation/validate-batch', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(batchValidationRequest)  
});  
  
const data = await response.json();  
console.log('All validations passed:', data.isValid);  
console.log('Validation results:', data.results);

// Example: Batch validation  
const batchValidationRequest = {  
 materialType: 'tile',  
 properties: {  
 thickness: 15,  
 width: 300,  
 finish: 'matte'  
 }  
};  
  
const response = await fetch('/api/validation/validate-batch', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(batchValidationRequest)  
});  
  
const data = await response.json();  
console.log('All validations passed:', data.isValid);  
console.log('Validation results:', data.results);

### Integration with Material Forms

The validation system can be integrated with material property forms to provide real-time validation feedback.

import { useState, useEffect } from 'react';  
import { TextField, FormHelperText } from '@mui/material';  
  
// Example: Property field with validation  
const PropertyField = ({ propertyName, value, onChange, materialType }) => {  
 const [error, setError] = useState(null);  
  
 // Validate on value change  
 useEffect(() => {  
 const validateProperty = async () => {  
 try {  
 const response = await fetch('/api/validation/validate', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 propertyName,  
 value,  
 materialType  
 })  
 });  
  
 const data = await response.json();  
  
 if (!data.isValid) {  
 // Find the first error message  
 const errorResult = data.results.find(result => !result.isValid);  
 setError(errorResult?.message || 'Invalid value');  
 } else {  
 setError(null);  
 }  
 } catch (err) {  
 console.error('Validation error:', err);  
 }  
 };  
  
 validateProperty();  
 }, [propertyName, value, materialType]);  
  
 return (  
 <div>  
 <TextField  
 label={propertyName}  
 value={value}  
 onChange={(e) => onChange(e.target.value)}  
 error={!!error}  
 />  
 {error && <FormHelperText error>{error}</FormHelperText>}  
 </div>  
 );  
};

import { useState, useEffect } from 'react';  
import { TextField, FormHelperText } from '@mui/material';  
  
// Example: Property field with validation  
const PropertyField = ({ propertyName, value, onChange, materialType }) => {  
 const [error, setError] = useState(null);  
  
 // Validate on value change  
 useEffect(() => {  
 const validateProperty = async () => {  
 try {  
 const response = await fetch('/api/validation/validate', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 propertyName,  
 value,  
 materialType  
 })  
 });  
  
 const data = await response.json();  
  
 if (!data.isValid) {  
 // Find the first error message  
 const errorResult = data.results.find(result => !result.isValid);  
 setError(errorResult?.message || 'Invalid value');  
 } else {  
 setError(null);  
 }  
 } catch (err) {  
 console.error('Validation error:', err);  
 }  
 };  
  
 validateProperty();  
 }, [propertyName, value, materialType]);  
  
 return (  
 <div>  
 <TextField  
 label={propertyName}  
 value={value}  
 onChange={(e) => onChange(e.target.value)}  
 error={!!error}  
 />  
 {error && <FormHelperText error>{error}</FormHelperText>}  
 </div>  
 );  
};

### Validation Rule Types

#### Range Validation

Range validation rules validate numeric values against minimum and maximum constraints.

// Example: Range validation rule  
const rangeRule = {  
 name: 'Temperature Range',  
 type: 'range',  
 propertyName: 'temperature',  
 materialType: 'ceramic',  
 severity: 'error',  
 message: 'Temperature must be between 0°C and 100°C',  
 min: 0,  
 max: 100,  
 step: 0.1,  
 unit: '°C'  
};

// Example: Range validation rule  
const rangeRule = {  
 name: 'Temperature Range',  
 type: 'range',  
 propertyName: 'temperature',  
 materialType: 'ceramic',  
 severity: 'error',  
 message: 'Temperature must be between 0°C and 100°C',  
 min: 0,  
 max: 100,  
 step: 0.1,  
 unit: '°C'  
};

#### Pattern Validation

Pattern validation rules validate string values against regular expression patterns.

// Example: Pattern validation rule  
const patternRule = {  
 name: 'Color Code Format',  
 type: 'pattern',  
 propertyName: 'colorCode',  
 materialType: 'paint',  
 severity: 'error',  
 message: 'Color code must be a valid hex color (e.g., #FF0000)',  
 pattern: '^#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})$'  
};

// Example: Pattern validation rule  
const patternRule = {  
 name: 'Color Code Format',  
 type: 'pattern',  
 propertyName: 'colorCode',  
 materialType: 'paint',  
 severity: 'error',  
 message: 'Color code must be a valid hex color (e.g., #FF0000)',  
 pattern: '^#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})$'  
};

#### Enumeration Validation

Enumeration validation rules validate values against a list of allowed values.

// Example: Enumeration validation rule  
const enumRule = {  
 name: 'Valid Finishes',  
 type: 'enum',  
 propertyName: 'finish',  
 materialType: 'tile',  
 severity: 'error',  
 message: 'Finish must be one of: matte, glossy, satin, textured',  
 allowedValues: ['matte', 'glossy', 'satin', 'textured']  
};

// Example: Enumeration validation rule  
const enumRule = {  
 name: 'Valid Finishes',  
 type: 'enum',  
 propertyName: 'finish',  
 materialType: 'tile',  
 severity: 'error',  
 message: 'Finish must be one of: matte, glossy, satin, textured',  
 allowedValues: ['matte', 'glossy', 'satin', 'textured']  
};

#### Dependency Validation

Dependency validation rules validate values based on the values of other properties.

// Example: Dependency validation rule  
const dependencyRule = {  
 name: 'Slip Resistance for Outdoor Tiles',  
 type: 'dependency',  
 propertyName: 'slipResistance',  
 materialType: 'tile',  
 severity: 'error',  
 message: 'Outdoor tiles must have a slip resistance rating of R10 or higher',  
 condition: {  
 propertyName: 'usage',  
 operator: 'equals',  
 value: 'outdoor'  
 },  
 requiredPattern: '^R(1[0-3]|9)$'  
};

// Example: Dependency validation rule  
const dependencyRule = {  
 name: 'Slip Resistance for Outdoor Tiles',  
 type: 'dependency',  
 propertyName: 'slipResistance',  
 materialType: 'tile',  
 severity: 'error',  
 message: 'Outdoor tiles must have a slip resistance rating of R10 or higher',  
 condition: {  
 propertyName: 'usage',  
 operator: 'equals',  
 value: 'outdoor'  
 },  
 requiredPattern: '^R(1[0-3]|9)$'  
};

#### Custom Validation

Custom validation rules apply custom validation functions.

// Example: Custom validation rule  
const customRule = {  
 name: 'Valid Email',  
 type: 'custom',  
 propertyName: 'contactEmail',  
 materialType: 'supplier',  
 severity: 'error',  
 message: 'Must be a valid email address',  
 functionName: 'isEmail'  
};

// Example: Custom validation rule  
const customRule = {  
 name: 'Valid Email',  
 type: 'custom',  
 propertyName: 'contactEmail',  
 materialType: 'supplier',  
 severity: 'error',  
 message: 'Must be a valid email address',  
 functionName: 'isEmail'  
};

#### Composite Validation

Composite validation rules combine multiple validation rules with logical operators.

// Example: Composite validation rule  
const compositeRule = {  
 name: 'Outdoor Tile Requirements',  
 type: 'composite',  
 propertyName: 'isValid',  
 materialType: 'tile',  
 severity: 'error',  
 message: 'Tile does not meet outdoor requirements',  
 operator: 'and',  
 rules: [  
 'slip-resistance-rule-id',  
 'frost-resistance-rule-id',  
 'water-absorption-rule-id'  
 ]  
};

// Example: Composite validation rule  
const compositeRule = {  
 name: 'Outdoor Tile Requirements',  
 type: 'composite',  
 propertyName: 'isValid',  
 materialType: 'tile',  
 severity: 'error',  
 message: 'Tile does not meet outdoor requirements',  
 operator: 'and',  
 rules: [  
 'slip-resistance-rule-id',  
 'frost-resistance-rule-id',  
 'water-absorption-rule-id'  
 ]  
};

### Best Practices

#### Creating Validation Rules

When creating validation rules, follow these best practices:

1. Clear Names and Messages: Use clear, descriptive names and error messages
2. Appropriate Severity: Use the appropriate severity level for each rule
3. Reusable Rules: Create reusable rules that can be applied to multiple material types
4. Logical Grouping: Group related rules using composite validation
5. Consistent Validation: Ensure consistent validation across similar properties

#### Validation Rule Management

To maintain an organized validation rule system:

1. Regular Review: Regularly review and update validation rules
2. Documentation: Document the purpose and behavior of each rule
3. Testing: Test validation rules with various inputs to ensure they work as expected
4. Versioning: Consider versioning validation rules for backward compatibility

### Benefits

Advanced Property Validation provides several benefits:

1. Data Quality: Ensures that property data meets defined standards
2. User Guidance: Guides users to enter correct property values
3. Consistency: Maintains consistency across similar materials
4. Error Prevention: Prevents invalid data from entering the system
5. Business Rules: Enforces business rules and domain-specific constraints

### Future Enhancements

Potential future enhancements to the Advanced Property Validation feature:

1. Rule Templates: Create reusable validation rule templates
2. Rule Import/Export: Support for importing and exporting validation rules
3. Validation Pipelines: Define validation pipelines for complex validation scenarios
4. Machine Learning Integration: Use machine learning to suggest validation rules based on existing data
5. Validation Analytics: Provide analytics on validation rule effectiveness and common validation errors

### Conclusion

Advanced Property Validation provides a comprehensive system for ensuring data quality and consistency in material properties. By defining and enforcing validation rules, it helps maintain high-quality data that can be used reliably throughout the system.

# Advanced Scaling Features

Source: readme/advanced-scaling-features.md

---

## Advanced Scaling Features

This document describes the advanced scaling features implemented in the KAI platform, including predictive scaling, cross-service scaling dependencies, and enhanced HPA event logging.

### Table of Contents

1. Overview
2. Predictive Scaling
3. Cross-Service Scaling Dependencies
4. Enhanced HPA Event Logging
5. Monitoring and Visualization
6. API Reference

### Overview

The KAI platform implements several advanced scaling features to optimize resource utilization and ensure system stability:

* Predictive Scaling: Analyzes historical metrics to predict future load and proactively adjust HPA settings
* Cross-Service Scaling Dependencies: Ensures that when one service scales, dependent services are also scaled appropriately
* Enhanced HPA Event Logging: Provides detailed information about scaling decisions and their triggers

These features are implemented in the Coordinator service and can be enabled or disabled through environment variables.

### Predictive Scaling

Predictive scaling analyzes historical metrics to predict future load and proactively adjust HPA settings for services with predictable load patterns.

#### How It Works

1. The system collects and analyzes historical metrics to identify patterns in service load
2. Based on these patterns, it predicts future load for specific time windows
3. The system proactively adjusts HPA settings to ensure that services have the right number of replicas before load increases

#### Configuration

Predictive scaling is controlled by the following environment variables:

* ENABLE\_PREDICTIVE\_SCALING: Set to true to enable predictive scaling
* REDIS\_URL: Redis connection URL for storing predictions and patterns

ENABLE\_PREDICTIVE\_SCALING

true

REDIS\_URL

#### Service Load Patterns

Service load patterns define when a service is expected to experience increased load:

{  
 "service": "coordinator",  
 "patternType": "daily",  
 "timeWindows": [  
 {  
 "dayOfWeek": 1,  
 "hourOfDay": 9,  
 "expectedLoad": 0.8  
 },  
 {  
 "dayOfWeek": 5,  
 "hourOfDay": 16,  
 "expectedLoad": 0.9  
 }  
 ]  
}

{  
 "service": "coordinator",  
 "patternType": "daily",  
 "timeWindows": [  
 {  
 "dayOfWeek": 1,  
 "hourOfDay": 9,  
 "expectedLoad": 0.8  
 },  
 {  
 "dayOfWeek": 5,  
 "hourOfDay": 16,  
 "expectedLoad": 0.9  
 }  
 ]  
}

#### API Endpoints

The following API endpoints are available for managing predictive scaling:

* GET /api/predictive-scaling/patterns: Get all service load patterns
* GET /api/predictive-scaling/patterns/:service: Get service load pattern for a specific service
* POST /api/predictive-scaling/patterns/:service: Create or update service load pattern
* DELETE /api/predictive-scaling/patterns/:service: Delete service load pattern
* GET /api/predictive-scaling/predictions: Get recent predictions

GET /api/predictive-scaling/patterns

GET /api/predictive-scaling/patterns/:service

POST /api/predictive-scaling/patterns/:service

DELETE /api/predictive-scaling/patterns/:service

GET /api/predictive-scaling/predictions

### Cross-Service Scaling Dependencies

Cross-service scaling dependencies ensure that when one service scales, dependent services are also scaled appropriately to maintain system balance.

#### How It Works

1. The system monitors the replica count of source services
2. When a source service scales, the system automatically adjusts the replica count of dependent services based on the defined dependency type

#### Dependency Types

* Proportional: Scale the target service proportionally to the source service (e.g., 2:1 ratio)
* Fixed: Set a fixed number of replicas for the target service when the source service scales
* Minimum: Ensure that the target service has at least a minimum number of replicas

#### Configuration

Cross-service scaling dependencies are controlled by the following environment variables:

* ENABLE\_SCALING\_DEPENDENCIES: Set to true to enable cross-service scaling dependencies
* REDIS\_URL: Redis connection URL for storing dependencies

ENABLE\_SCALING\_DEPENDENCIES

true

REDIS\_URL

#### API Endpoints

The following API endpoints are available for managing scaling dependencies:

* GET /api/scaling-dependencies: Get all scaling dependencies
* GET /api/scaling-dependencies/:sourceService/:targetService: Get a specific scaling dependency
* POST /api/scaling-dependencies/:sourceService/:targetService: Create or update a scaling dependency
* DELETE /api/scaling-dependencies/:sourceService/:targetService: Delete a scaling dependency
* POST /api/scaling-dependencies/:sourceService/:targetService/enable: Enable a scaling dependency
* POST /api/scaling-dependencies/:sourceService/:targetService/disable: Disable a scaling dependency

GET /api/scaling-dependencies

GET /api/scaling-dependencies/:sourceService/:targetService

POST /api/scaling-dependencies/:sourceService/:targetService

DELETE /api/scaling-dependencies/:sourceService/:targetService

POST /api/scaling-dependencies/:sourceService/:targetService/enable

POST /api/scaling-dependencies/:sourceService/:targetService/disable

### Enhanced HPA Event Logging

Enhanced HPA event logging provides detailed information about scaling decisions and their triggers, helping to understand and optimize scaling behavior.

#### How It Works

1. The system monitors HPA objects in the Kubernetes cluster
2. When a scaling event occurs, the system logs detailed information about the event, including the trigger metric and its value
3. The system also calculates scaling effectiveness metrics to help optimize scaling behavior

#### Configuration

Enhanced HPA event logging is controlled by the following environment variables:

* ENABLE\_HPA\_EVENT\_LOGGING: Set to true to enable enhanced HPA event logging
* REDIS\_URL: Redis connection URL for storing events

ENABLE\_HPA\_EVENT\_LOGGING

true

REDIS\_URL

#### Event Types

* scale-up: HPA decided to increase the number of replicas
* scale-down: HPA decided to decrease the number of replicas
* no-scale: HPA decided not to change the number of replicas
* limited-scale: HPA wanted to scale but was limited by constraints

#### API Endpoints

The following API endpoints are available for accessing HPA event logs:

* GET /api/hpa-events: Get recent HPA events
* GET /api/hpa-events/:service: Get recent HPA events for a specific service
* GET /api/hpa-events/:service/effectiveness: Get scaling effectiveness for a specific service

GET /api/hpa-events

GET /api/hpa-events/:service

GET /api/hpa-events/:service/effectiveness

### Monitoring and Visualization

The KAI platform includes comprehensive monitoring and visualization for advanced scaling features:

#### Grafana Dashboards

* HPA Metrics Dashboard: Shows current and desired replica counts, scaling events, and their triggers
* Coordinator Service Dashboard: Shows queue depths, workflow durations, and processing metrics
* Supabase Connection Pool Dashboard: Shows database connection pool metrics and performance

#### Admin Panel Integration

The admin panel includes a dedicated Grafana Dashboards page that embeds these dashboards, providing a unified interface for monitoring the system.

### API Reference

#### Predictive Scaling API

##### Get All Service Load Patterns

GET /api/predictive-scaling/patterns

GET /api/predictive-scaling/patterns

Response:

{  
 "patterns": [  
 {  
 "service": "coordinator",  
 "patternType": "daily",  
 "timeWindows": [  
 {  
 "dayOfWeek": 1,  
 "hourOfDay": 9,  
 "expectedLoad": 0.8  
 }  
 ],  
 "lastUpdated": 1623456789000  
 }  
 ]  
}

{  
 "patterns": [  
 {  
 "service": "coordinator",  
 "patternType": "daily",  
 "timeWindows": [  
 {  
 "dayOfWeek": 1,  
 "hourOfDay": 9,  
 "expectedLoad": 0.8  
 }  
 ],  
 "lastUpdated": 1623456789000  
 }  
 ]  
}

##### Create or Update Service Load Pattern

POST /api/predictive-scaling/patterns/:service

POST /api/predictive-scaling/patterns/:service

Request:

{  
 "patternType": "daily",  
 "timeWindows": [  
 {  
 "dayOfWeek": 1,  
 "hourOfDay": 9,  
 "expectedLoad": 0.8  
 }  
 ]  
}

{  
 "patternType": "daily",  
 "timeWindows": [  
 {  
 "dayOfWeek": 1,  
 "hourOfDay": 9,  
 "expectedLoad": 0.8  
 }  
 ]  
}

#### Scaling Dependencies API

##### Get All Scaling Dependencies

GET /api/scaling-dependencies

GET /api/scaling-dependencies

Response:

{  
 "dependencies": [  
 {  
 "sourceService": "coordinator",  
 "targetService": "mobile-optimization",  
 "dependencyType": "proportional",  
 "ratio": 0.5,  
 "enabled": true,  
 "lastUpdated": 1623456789000  
 }  
 ]  
}

{  
 "dependencies": [  
 {  
 "sourceService": "coordinator",  
 "targetService": "mobile-optimization",  
 "dependencyType": "proportional",  
 "ratio": 0.5,  
 "enabled": true,  
 "lastUpdated": 1623456789000  
 }  
 ]  
}

##### Create or Update Scaling Dependency

POST /api/scaling-dependencies/:sourceService/:targetService

POST /api/scaling-dependencies/:sourceService/:targetService

Request:

{  
 "dependencyType": "proportional",  
 "ratio": 0.5,  
 "enabled": true  
}

{  
 "dependencyType": "proportional",  
 "ratio": 0.5,  
 "enabled": true  
}

#### HPA Events API

##### Get Recent HPA Events

GET /api/hpa-events

GET /api/hpa-events

Response:

{  
 "events": [  
 {  
 "service": "coordinator",  
 "eventType": "scale-up",  
 "currentReplicas": 2,  
 "desiredReplicas": 4,  
 "actualReplicas": 4,  
 "triggerMetric": "resource:cpu",  
 "triggerValue": 85,  
 "triggerThreshold": 70,  
 "timestamp": 1623456789000  
 }  
 ]  
}

{  
 "events": [  
 {  
 "service": "coordinator",  
 "eventType": "scale-up",  
 "currentReplicas": 2,  
 "desiredReplicas": 4,  
 "actualReplicas": 4,  
 "triggerMetric": "resource:cpu",  
 "triggerValue": 85,  
 "triggerThreshold": 70,  
 "timestamp": 1623456789000  
 }  
 ]  
}

##### Get Scaling Effectiveness

GET /api/hpa-events/:service/effectiveness

GET /api/hpa-events/:service/effectiveness

Response:

{  
 "service": "coordinator",  
 "effectiveness": 0.95  
}

{  
 "service": "coordinator",  
 "effectiveness": 0.95  
}

# Agents Crewai

Source: readme/agents-crewai.md

---

## CrewAI Integration for KAI Platform

This document provides a comprehensive guide to the CrewAI integration in the KAI platform, covering architecture, installation, environment setup, implementation details, and usage examples.

Note: Installation instructions are available in the Deployment Guide.

### Table of Contents

1. Overview
2. Architecture
3. Core Components
4. Directory Structure
5. MCP Integration Structure
6. Agent Types
7. Frontend Agents
8. Backend Agents
9. Environment Setup
10. Environment Variables
11. API Keys and Service URLs
12. Authentication Configuration
13. Implementation Status
14. Setup and Usage
15. Prerequisites
16. Example Usage
17. MCP Integration for Agents
18. Benefits of MCP for Agents
19. MCP-Enabled Agent Factory
20. LLM Inference Adapter
21. Batch Processing and Request Optimization
22. Agent Tools
23. Material Search Tool
24. Image Analysis Tool
25. Vector Search Tool
26. MCP-Enabled Tools
27. Integration with Existing KAI Components
28. Frontend Integration
29. Backend Integration
30. MCP Integration
31. Implementation Roadmap
32. Next Development Steps
33. Known Issues
34. Verification and Testing
35. Troubleshooting
36. Technical Considerations
37. Related Documentation

### Overview

The CrewAI integration adds intelligent agent capabilities to the KAI platform, enabling:

1. User-facing intelligence: Agents that assist users during material recognition, provide detailed information about materials, and help organize projects
2. System-level intelligence: Agents that monitor the knowledge base, analyze system metrics, and optimize platform operations
3. MCP-powered inference: Optimized language model operations through the Model Context Protocol server architecture

### Architecture

#### Core Components

The integration is structured around several key components:

* Agent System: Centralized management of agent initialization, configuration, and lifecycle
* Agent Types: Specialized agents for different roles and capabilities
* Agent Tools: Functions that allow agents to interact with KAI systems
* Utilities: Common functionality for logging, error handling, and data processing
* MCP Integration: Adapters that enable agents to leverage the MCP server architecture
* Unified Type System: Consolidated TypeScript definitions for consistent agent development
* Enhanced Material Experts: Advanced material analysis capabilities with inheritance-based architecture

#### Directory Structure

packages/  
└── agents/  
 ├── package.json # Package configuration and dependencies  
 ├── tsconfig.json # TypeScript configuration  
 ├── setup.sh # Setup script for dependencies and missing files  
 ├── src/  
 │ ├── index.ts # Main exports  
 │ ├── core/ # Core system implementation  
 │ │ ├── agentSystem.ts # Agent initialization and management  
 │ │ └── types.ts # Type definitions and interfaces  
 │ ├── frontend/ # User-facing agents  
 │ │ ├── recognitionAssistant.ts # Image recognition enhancement  
 │ │ ├── materialExpert.ts # Material information provider  
 │ │ └── projectAssistant.ts # Project planning assistant  
 │ ├── backend/ # System-level agents  
 │ │ ├── knowledgeBaseAgent.ts # Knowledge base management  
 │ │ ├── analyticsAgent.ts # System metrics analysis  
 │ │ └── operationsAgent.ts # System operations monitoring  
 │ ├── tools/ # Agent interaction capabilities  
 │ │ ├── materialSearch.ts # Material database searches  
 │ │ ├── imageAnalysis.ts # Image property extraction  
 │ │ ├── vectorSearch.ts # Semantic similarity search  
 │ │ └── index.ts # Tool exports  
 │ └── utils/ # Common utilities  
 │ ├── logger.ts # Logging system  
 │ └── index.ts # Utility exports  
 └── logs/ # Agent operation logs

packages/  
└── agents/  
 ├── package.json # Package configuration and dependencies  
 ├── tsconfig.json # TypeScript configuration  
 ├── setup.sh # Setup script for dependencies and missing files  
 ├── src/  
 │ ├── index.ts # Main exports  
 │ ├── core/ # Core system implementation  
 │ │ ├── agentSystem.ts # Agent initialization and management  
 │ │ └── types.ts # Type definitions and interfaces  
 │ ├── frontend/ # User-facing agents  
 │ │ ├── recognitionAssistant.ts # Image recognition enhancement  
 │ │ ├── materialExpert.ts # Material information provider  
 │ │ └── projectAssistant.ts # Project planning assistant  
 │ ├── backend/ # System-level agents  
 │ │ ├── knowledgeBaseAgent.ts # Knowledge base management  
 │ │ ├── analyticsAgent.ts # System metrics analysis  
 │ │ └── operationsAgent.ts # System operations monitoring  
 │ ├── tools/ # Agent interaction capabilities  
 │ │ ├── materialSearch.ts # Material database searches  
 │ │ ├── imageAnalysis.ts # Image property extraction  
 │ │ ├── vectorSearch.ts # Semantic similarity search  
 │ │ └── index.ts # Tool exports  
 │ └── utils/ # Common utilities  
 │ ├── logger.ts # Logging system  
 │ └── index.ts # Utility exports  
 └── logs/ # Agent operation logs

#### MCP Integration Structure

packages/  
└── agents/  
 ├── src/  
 │ ├── core/  
 │ │ └── mcpAgentFactory.ts # MCP-enabled agent creation  
 │ ├── services/  
 │ │ └── adapters/  
 │ │ ├── llmInferenceMcpAdapter.ts # LLM operations adapter  
 │ │ ├── vectorSearchMcpAdapter.ts # Vector search adapter  
 │ │ ├── imageAnalysisMcpAdapter.ts # Image analysis adapter  
 │ │ └── ocrMcpAdapter.ts # OCR adapter  
 │ ├── utils/  
 │ │ ├── mcpIntegration.ts # MCP connection utilities  
 │ │ ├── mcpBatchProcessor.ts # Request batching system  
 │ │ └── llmInferenceHelper.ts # LLM inference utilities  
 └── docs/  
 └── mcp-integration.md # MCP integration documentation

packages/  
└── agents/  
 ├── src/  
 │ ├── core/  
 │ │ └── mcpAgentFactory.ts # MCP-enabled agent creation  
 │ ├── services/  
 │ │ └── adapters/  
 │ │ ├── llmInferenceMcpAdapter.ts # LLM operations adapter  
 │ │ ├── vectorSearchMcpAdapter.ts # Vector search adapter  
 │ │ ├── imageAnalysisMcpAdapter.ts # Image analysis adapter  
 │ │ └── ocrMcpAdapter.ts # OCR adapter  
 │ ├── utils/  
 │ │ ├── mcpIntegration.ts # MCP connection utilities  
 │ │ ├── mcpBatchProcessor.ts # Request batching system  
 │ │ └── llmInferenceHelper.ts # LLM inference utilities  
 └── docs/  
 └── mcp-integration.md # MCP integration documentation

### Agent Types

#### Frontend Agents

1. Recognition Assistant
2. Purpose: Enhance the material recognition workflow
3. Capabilities:  
   Pre-upload guidance for optimal image capture  
   Enhanced analysis of recognition results   
   Detailed explanations of material properties  
   Alternative matching suggestions
4. Integration Points:  
     
   Image upload component  
   Recognition results display  
   Material detail views
5. Material Expert
6. Purpose: Provide in-depth material knowledge
7. Capabilities:  
   Detailed material specifications  
   Comparative analysis with similar materials  
   Installation and maintenance recommendations  
   Technical compliance information
8. Integration Points:  
     
   Material detail pages  
   Comparison views  
   Search and filtering interfaces
9. Project Assistant
10. Purpose: Help users plan and organize material projects
11. Capabilities:  
    Material selection guidance  
    Quantity estimation and calculations  
    Compatibility checking between materials  
    Timeline and phasing suggestions
12. Integration Points:  
    Project planning interfaces  
    Material selection workflows  
    Cart and ordering systems

* Pre-upload guidance for optimal image capture
* Enhanced analysis of recognition results
* Detailed explanations of material properties
* Alternative matching suggestions

Integration Points:

* Image upload component
* Recognition results display
* Material detail views

Material Expert

* Detailed material specifications
* Comparative analysis with similar materials
* Installation and maintenance recommendations
* Technical compliance information

Integration Points:

* Material detail pages
* Comparison views
* Search and filtering interfaces

Project Assistant

* Material selection guidance
* Quantity estimation and calculations
* Compatibility checking between materials
* Timeline and phasing suggestions
* Project planning interfaces
* Material selection workflows
* Cart and ordering systems

#### Backend Agents

1. Knowledge Base Agent
2. Purpose: Maintain and enhance the knowledge base
3. Capabilities:  
   Data quality assessment  
   Relationship detection between materials  
   Metadata optimization  
   Search index enhancement
4. Integration Points:  
     
   Knowledge base management interfaces  
   Admin dashboards  
   Metadata management systems
5. Analytics Agent
6. Purpose: Analyze system data for insights
7. Capabilities:  
   Usage pattern detection  
   Anomaly identification  
   User behavior analysis  
   Trend forecasting
8. Integration Points:  
     
   Analytics dashboards  
   Reporting systems  
   Admin interfaces
9. Operations Agent
10. Purpose: Monitor and optimize system operations
11. Capabilities:  
    Proactive issue detection  
    Performance optimization recommendations  
    Resource allocation suggestions  
    Automated maintenance tasks
12. Integration Points:  
    System monitoring dashboards  
    Operations interfaces  
    Notification systems

* Data quality assessment
* Relationship detection between materials
* Metadata optimization
* Search index enhancement

Integration Points:

* Knowledge base management interfaces
* Admin dashboards
* Metadata management systems

Analytics Agent

* Usage pattern detection
* Anomaly identification
* User behavior analysis
* Trend forecasting

Integration Points:

* Analytics dashboards
* Reporting systems
* Admin interfaces

Operations Agent

* Proactive issue detection
* Performance optimization recommendations
* Resource allocation suggestions
* Automated maintenance tasks
* System monitoring dashboards
* Operations interfaces
* Notification systems

### Environment Setup

#### Environment Variables

All CrewAI integration environment variables should be added to the main application's .env file. Do not create a separate environment file for the agent system.

.env

Here's a complete list of the required and optional environment variables:

# === CrewAI Agent System ===  
  
# OpenAI API Configuration (required)  
OPENAI\_API\_KEY=your\_openai\_api\_key\_here  
OPENAI\_DEFAULT\_MODEL=gpt-4  
OPENAI\_TEMPERATURE=0.7  
  
# Redis Configuration (for agent state persistence)  
REDIS\_URL=redis://localhost:6379  
REDIS\_PASSWORD=  
  
# Logging Configuration  
LOG\_LEVEL=info  
LOG\_FILE\_PATH=logs/agent.log  
  
# KAI Service URLs  
KAI\_API\_URL=http://localhost:3000/api  
KAI\_VECTOR\_DB\_URL=http://localhost:5000/api/vector  
KAI\_ML\_SERVICE\_URL=http://localhost:7000/api/ml  
  
# Authentication  
KAI\_API\_KEY=your\_kai\_api\_key\_here  
KAI\_AUTH\_TOKEN=your\_auth\_token\_here  
  
# Agent Behavior Settings  
AGENT\_VERBOSE\_MODE=false  
AGENT\_MEMORY\_ENABLED=true  
AGENT\_MAX\_ITERATIONS=10  
AGENT\_DEFAULT\_TIMEOUT=30000  
  
# MCP Configuration  
MCP\_SERVER\_URL=http://localhost:8000  
MCP\_AUTH\_TOKEN=your\_mcp\_auth\_token  
MCP\_ENABLED\_agentInference=true  
MCP\_BATCHING\_ENABLED\_agentInference=true

# === CrewAI Agent System ===  
  
# OpenAI API Configuration (required)  
OPENAI\_API\_KEY=your\_openai\_api\_key\_here  
OPENAI\_DEFAULT\_MODEL=gpt-4  
OPENAI\_TEMPERATURE=0.7  
  
# Redis Configuration (for agent state persistence)  
REDIS\_URL=redis://localhost:6379  
REDIS\_PASSWORD=  
  
# Logging Configuration  
LOG\_LEVEL=info  
LOG\_FILE\_PATH=logs/agent.log  
  
# KAI Service URLs  
KAI\_API\_URL=http://localhost:3000/api  
KAI\_VECTOR\_DB\_URL=http://localhost:5000/api/vector  
KAI\_ML\_SERVICE\_URL=http://localhost:7000/api/ml  
  
# Authentication  
KAI\_API\_KEY=your\_kai\_api\_key\_here  
KAI\_AUTH\_TOKEN=your\_auth\_token\_here  
  
# Agent Behavior Settings  
AGENT\_VERBOSE\_MODE=false  
AGENT\_MEMORY\_ENABLED=true  
AGENT\_MAX\_ITERATIONS=10  
AGENT\_DEFAULT\_TIMEOUT=30000  
  
# MCP Configuration  
MCP\_SERVER\_URL=http://localhost:8000  
MCP\_AUTH\_TOKEN=your\_mcp\_auth\_token  
MCP\_ENABLED\_agentInference=true  
MCP\_BATCHING\_ENABLED\_agentInference=true

#### API Keys and Service URLs

##### OpenAI API

The CrewAI integration requires an OpenAI API key for agent operations. To get an API key:

1. Create an account at OpenAI's platform
2. Navigate to API Keys section
3. Generate a new key and add it to your .env file as OPENAI\_API\_KEY

.env

OPENAI\_API\_KEY

##### KAI Service URLs

The integration connects to several KAI services. The default URLs are configured for local development, but you should adjust them based on your deployment environment:

# Local Development  
KAI\_API\_URL=http://localhost:3000/api  
KAI\_VECTOR\_DB\_URL=http://localhost:5000/api/vector  
KAI\_ML\_SERVICE\_URL=http://localhost:7000/api/ml  
  
# Staging Environment Example  
KAI\_API\_URL=https://staging-api.kai-platform.com/api  
KAI\_VECTOR\_DB\_URL=https://staging-vector.kai-platform.com/api/vector  
KAI\_ML\_SERVICE\_URL=https://staging-ml.kai-platform.com/api/ml  
  
# Production Environment Example  
KAI\_API\_URL=https://api.kai-platform.com/api  
KAI\_VECTOR\_DB\_URL=https://vector.kai-platform.com/api/vector  
KAI\_ML\_SERVICE\_URL=https://ml.kai-platform.com/api/ml

# Local Development  
KAI\_API\_URL=http://localhost:3000/api  
KAI\_VECTOR\_DB\_URL=http://localhost:5000/api/vector  
KAI\_ML\_SERVICE\_URL=http://localhost:7000/api/ml  
  
# Staging Environment Example  
KAI\_API\_URL=https://staging-api.kai-platform.com/api  
KAI\_VECTOR\_DB\_URL=https://staging-vector.kai-platform.com/api/vector  
KAI\_ML\_SERVICE\_URL=https://staging-ml.kai-platform.com/api/ml  
  
# Production Environment Example  
KAI\_API\_URL=https://api.kai-platform.com/api  
KAI\_VECTOR\_DB\_URL=https://vector.kai-platform.com/api/vector  
KAI\_ML\_SERVICE\_URL=https://ml.kai-platform.com/api/ml

#### Authentication Configuration

The integration uses the KAI authentication system for accessing various services. There are two ways to authenticate:

##### 1. API Key Authentication (Recommended for server environments)

Set the KAI\_API\_KEY environment variable to authenticate using an API key:

KAI\_API\_KEY

KAI\_API\_KEY=your\_kai\_api\_key\_here

KAI\_API\_KEY=your\_kai\_api\_key\_here

##### 2. Token Authentication (Used in browser environments)

The system will automatically use token-based authentication in browser environments. If you've implemented a custom authentication flow, you can manually set:

KAI\_AUTH\_TOKEN=your\_auth\_token\_here

KAI\_AUTH\_TOKEN=your\_auth\_token\_here

### Implementation Status

The integration of crewAI agents into the KAI platform includes:

1. Core Framework:  
     
   An agents package with core types, interfaces, and configuration (agentSystem.ts).  
   Agent system management using environment variables for configuration (API keys, service URLs).  
   Optional Redis integration for agent memory persistence.  
   Logging and error handling utilities.
2. MCP Integration (LLM):  
     
   Implemented via adapters (llmInferenceMcpAdapter.ts) and helpers (llmInferenceHelper.ts).  
   Supports chat, completion, and embedding operations.  
   Automatically routes requests to the MCP server when enabled (isMCPEnabledForComponent).  
   Supports batching (mcpBatchProcessor.ts) and streaming (via WebSockets).  
   Includes fallback to a local LLMService implementation if MCP is disabled or fails.  
   Connects to real KAI backend services (configured via environment variables).
3. Agent Types & Factory:  
     
   Defined frontend agents (RecognitionAssistant, MaterialExpert, ProjectAssistant) and backend agents (KnowledgeBaseAgent, AnalyticsAgent, OperationsAgent).  
   mcpAgentFactory.ts creates MCP-enabled frontend agents, leveraging the LLM adapter. Backend agent MCP usage needs further review.  
   createEnhancedMaterialExpert is used for MaterialExpert.  
   createImageCapableMaterialExpert adds image analysis via imageAnalysisMcpAdapter.
4. Frontend Components:  
     
   UI components for agent interaction exist (AgentChat, panels).  
   Further integration work (WebSockets, state persistence) may be needed.
5. Tools & Adapters:  
     
   Basic agent tools (materialSearch, imageAnalysis, vectorSearch) exist.  
   Adapters (llmInferenceMcpAdapter, imageAnalysisMcpAdapter, etc.) handle communication with MCP or local services.  
   Completeness and robustness of individual agent/tool implementations require ongoing review.

Core Framework:

* An agents package with core types, interfaces, and configuration (agentSystem.ts).
* Agent system management using environment variables for configuration (API keys, service URLs).
* Optional Redis integration for agent memory persistence.
* Logging and error handling utilities.

agents

agentSystem.ts

MCP Integration (LLM):

* Implemented via adapters (llmInferenceMcpAdapter.ts) and helpers (llmInferenceHelper.ts).
* Supports chat, completion, and embedding operations.
* Automatically routes requests to the MCP server when enabled (isMCPEnabledForComponent).
* Supports batching (mcpBatchProcessor.ts) and streaming (via WebSockets).
* Includes fallback to a local LLMService implementation if MCP is disabled or fails.
* Connects to real KAI backend services (configured via environment variables).

llmInferenceMcpAdapter.ts

llmInferenceHelper.ts

isMCPEnabledForComponent

mcpBatchProcessor.ts

LLMService

Agent Types & Factory:

* Defined frontend agents (RecognitionAssistant, MaterialExpert, ProjectAssistant) and backend agents (KnowledgeBaseAgent, AnalyticsAgent, OperationsAgent).
* mcpAgentFactory.ts creates MCP-enabled frontend agents, leveraging the LLM adapter. Backend agent MCP usage needs further review.
* createEnhancedMaterialExpert is used for MaterialExpert.
* createImageCapableMaterialExpert adds image analysis via imageAnalysisMcpAdapter.

RecognitionAssistant

MaterialExpert

ProjectAssistant

KnowledgeBaseAgent

AnalyticsAgent

OperationsAgent

mcpAgentFactory.ts

createEnhancedMaterialExpert

MaterialExpert

createImageCapableMaterialExpert

imageAnalysisMcpAdapter

Frontend Components:

* UI components for agent interaction exist (AgentChat, panels).
* Further integration work (WebSockets, state persistence) may be needed.

Tools & Adapters:

* Basic agent tools (materialSearch, imageAnalysis, vectorSearch) exist.
* Adapters (llmInferenceMcpAdapter, imageAnalysisMcpAdapter, etc.) handle communication with MCP or local services.
* Completeness and robustness of individual agent/tool implementations require ongoing review.

materialSearch

imageAnalysis

vectorSearch

llmInferenceMcpAdapter

imageAnalysisMcpAdapter

### Setup and Usage

#### Prerequisites

* Node.js 16+
* Yarn package manager
* OpenAI API key (or other supported LLM provider)
* MCP server (optional, for optimized performance)

#### Example Usage

##### Initializing the Agent System with Service Connections

import { initializeAgentSystem, connectToServices } from '@kai/agents';  
import { env } from '@kai/agents/utils/environment'; // Assuming env is exported  
  
// Initialize core system (uses env vars for keys, redis, etc.)  
await initializeAgentSystem();   
  
// Configure connections to KAI services (uses env vars by default)  
await connectToServices();   
  
// --- OR Initialize with explicit config ---  
/\*  
await initializeAgentSystem({  
 apiKey: process.env.OPENAI\_API\_KEY,  
 redis: env.redis.url ? { host: new URL(env.redis.url).hostname, port: parseInt(new URL(env.redis.url).port || '6379'), password: env.redis.password } : undefined,  
 logLevel: 'debug'  
});  
  
await connectToServices({  
 apiUrl: env.services.kaiApiUrl,  
 vectorDbUrl: env.services.vectorDbUrl,  
 mlServiceUrl: env.services.mlServiceUrl,  
 apiKey: env.services.apiKey,  
 enableMockFallback: env.services.enableMockFallback  
});  
\*/

import { initializeAgentSystem, connectToServices } from '@kai/agents';  
import { env } from '@kai/agents/utils/environment'; // Assuming env is exported  
  
// Initialize core system (uses env vars for keys, redis, etc.)  
await initializeAgentSystem();   
  
// Configure connections to KAI services (uses env vars by default)  
await connectToServices();   
  
// --- OR Initialize with explicit config ---  
/\*  
await initializeAgentSystem({  
 apiKey: process.env.OPENAI\_API\_KEY,  
 redis: env.redis.url ? { host: new URL(env.redis.url).hostname, port: parseInt(new URL(env.redis.url).port || '6379'), password: env.redis.password } : undefined,  
 logLevel: 'debug'  
});  
  
await connectToServices({  
 apiUrl: env.services.kaiApiUrl,  
 vectorDbUrl: env.services.vectorDbUrl,  
 mlServiceUrl: env.services.mlServiceUrl,  
 apiKey: env.services.apiKey,  
 enableMockFallback: env.services.enableMockFallback  
});  
\*/

##### Creating a Recognition Assistant

import { createAgent, AgentType } from '@kai/agents';  
  
const agent = await createAgent({  
 id: 'recognition-assistant-1',  
 type: AgentType.RECOGNITION\_ASSISTANT,  
 name: 'Recognition Helper',  
 description: 'Assists users with material recognition',  
});  
  
// Process an uploaded image  
const insights = await agent.instance.processImage(imageUrl, {  
 originalFileName: file.name,  
 fileSize: file.size,  
 uploadedBy: user.id,  
});

import { createAgent, AgentType } from '@kai/agents';  
  
const agent = await createAgent({  
 id: 'recognition-assistant-1',  
 type: AgentType.RECOGNITION\_ASSISTANT,  
 name: 'Recognition Helper',  
 description: 'Assists users with material recognition',  
});  
  
// Process an uploaded image  
const insights = await agent.instance.processImage(imageUrl, {  
 originalFileName: file.name,  
 fileSize: file.size,  
 uploadedBy: user.id,  
});

##### Creating an MCP-enabled agent with real service connections

import { createMCPEnabledAgent, AgentType } from '@kai/agents/core/mcpAgentFactory';  
import { AgentConfig } from '@kai/agents'; // Assuming AgentConfig is exported  
  
// Configuration for the agent  
const agentConfig: AgentConfig = {  
 id: 'mcp-material-expert-1',  
 type: AgentType.MATERIAL\_EXPERT,  
 name: 'MCP Material Expert',  
 description: 'Provides detailed material information using MCP',  
 // Agent-specific tools can be added here if needed  
 // tools: [customTool]   
};  
  
// Model settings, potentially overriding defaults  
const modelSettings = {  
 provider: 'openai',  
 name: 'gpt-4-turbo', // Use desired model  
 temperature: 0.5,  
 enableBatching: true, // Enable batching via MCP if desired  
 // maxTokens: 2048   
};  
  
// Create the agent  
const mcpAgent = await createMCPEnabledAgent(agentConfig, modelSettings);  
  
// Use the agent (it will use MCP for LLM ops if enabled)  
const materialDetails = await mcpAgent.processUserInput('Tell me about Carrara marble.');  
console.log(materialDetails);  
  
// Example using image analysis capability (if created via createImageCapableMaterialExpert)  
// const imageAnalysis = await mcpAgent.analyzeImage(imageBase64String);  
// console.log(imageAnalysis);

import { createMCPEnabledAgent, AgentType } from '@kai/agents/core/mcpAgentFactory';  
import { AgentConfig } from '@kai/agents'; // Assuming AgentConfig is exported  
  
// Configuration for the agent  
const agentConfig: AgentConfig = {  
 id: 'mcp-material-expert-1',  
 type: AgentType.MATERIAL\_EXPERT,  
 name: 'MCP Material Expert',  
 description: 'Provides detailed material information using MCP',  
 // Agent-specific tools can be added here if needed  
 // tools: [customTool]   
};  
  
// Model settings, potentially overriding defaults  
const modelSettings = {  
 provider: 'openai',  
 name: 'gpt-4-turbo', // Use desired model  
 temperature: 0.5,  
 enableBatching: true, // Enable batching via MCP if desired  
 // maxTokens: 2048   
};  
  
// Create the agent  
const mcpAgent = await createMCPEnabledAgent(agentConfig, modelSettings);  
  
// Use the agent (it will use MCP for LLM ops if enabled)  
const materialDetails = await mcpAgent.processUserInput('Tell me about Carrara marble.');  
console.log(materialDetails);  
  
// Example using image analysis capability (if created via createImageCapableMaterialExpert)  
// const imageAnalysis = await mcpAgent.analyzeImage(imageBase64String);  
// console.log(imageAnalysis);

### MCP Integration for Agents

The agent system integrates with the Model Context Protocol (MCP) server architecture to optimize language model operations and resource utilization.

#### Benefits of MCP for Agents

1. Performance Optimization
2. Reduced latency by eliminating model loading overhead
3. Improved throughput through token batching
4. More efficient GPU/TPU utilization for inference
5. Resource Efficiency
6. Multiple agents share the same model instances
7. Lower memory footprint
8. More efficient scaling of agent capabilities
9. Enhanced Capabilities
10. Streaming responses for real-time agent interactions
11. Centralized model version management
12. Seamless model upgrades without restarts

More efficient GPU/TPU utilization for inference

Resource Efficiency

More efficient scaling of agent capabilities

Enhanced Capabilities

#### MCP-Enabled Agent Factory

The mcpAgentFactory.ts provides enhanced agent creation with MCP integration that connects directly to your API and MCP server:

mcpAgentFactory.ts

import { createMCPEnabledAgent } from '@kai/agents/core/mcpAgentFactory';  
  
// Create an MCP-enabled agent that uses real services  
const agent = await createMCPEnabledAgent({  
 id: 'material-expert-1',  
 name: 'Material Expert',  
 description: 'Provides detailed information about materials',  
 modelSettings: {  
 provider: 'openai',  
 model: 'gpt-4-turbo',  
 temperature: 0.7  
 },  
 tools: [materialSearchTool, vectorSearchTool]  
});  
  
// Agent uses MCP server for LLM operations when available  
// Falls back to local implementation if MCP server is unavailable  
const response = await agent.execute(userQuery);

import { createMCPEnabledAgent } from '@kai/agents/core/mcpAgentFactory';  
  
// Create an MCP-enabled agent that uses real services  
const agent = await createMCPEnabledAgent({  
 id: 'material-expert-1',  
 name: 'Material Expert',  
 description: 'Provides detailed information about materials',  
 modelSettings: {  
 provider: 'openai',  
 model: 'gpt-4-turbo',  
 temperature: 0.7  
 },  
 tools: [materialSearchTool, vectorSearchTool]  
});  
  
// Agent uses MCP server for LLM operations when available  
// Falls back to local implementation if MCP server is unavailable  
const response = await agent.execute(userQuery);

For image analysis capabilities:

import { createImageCapableMaterialExpert } from '@kai/agents/core/mcpAgentFactory';  
  
// Create a material expert with image analysis capabilities via MCP  
const imageCapableAgent = await createImageCapableMaterialExpert({  
 id: 'image-material-expert-1',  
 name: 'Image Material Expert',  
 description: 'Analyzes material images and provides detailed information',  
 modelSettings: {  
 provider: 'openai',  
 model: 'gpt-4-vision',  
 temperature: 0.5  
 }  
});  
  
// Agent can process images and provide detailed analysis  
const imageAnalysis = await imageCapableAgent.analyzeImage(imageUrl);

import { createImageCapableMaterialExpert } from '@kai/agents/core/mcpAgentFactory';  
  
// Create a material expert with image analysis capabilities via MCP  
const imageCapableAgent = await createImageCapableMaterialExpert({  
 id: 'image-material-expert-1',  
 name: 'Image Material Expert',  
 description: 'Analyzes material images and provides detailed information',  
 modelSettings: {  
 provider: 'openai',  
 model: 'gpt-4-vision',  
 temperature: 0.5  
 }  
});  
  
// Agent can process images and provide detailed analysis  
const imageAnalysis = await imageCapableAgent.analyzeImage(imageUrl);

#### LLM Inference Adapter

The llmInferenceMcpAdapter.ts handles all language model operations through MCP with direct connections to your API and MCP server:

llmInferenceMcpAdapter.ts

* Chat completions - For conversational agent interactions with direct API connections
* Text completions - For structured text generation through your ML API endpoints
* Embeddings - For semantic representation of text using your vector models
* Streaming responses - For real-time interactions using WebSocket connections to your API

Each of these operations:  
1. Checks if MCP is enabled for the component  
2. If enabled, routes requests through the MCP server  
3. If disabled or if MCP is unavailable, falls back to direct API calls  
4. Provides comprehensive error handling and logging

#### Batch Processing and Request Optimization

The MCP integration optimizes performance by batching similar operations to reduce API call overhead:

// These operations will be automatically batched if they occur within  
// the configured time window (default: 50ms)  
const [resultA, resultB, resultC] = await Promise.all([  
 agent.generateEmbedding(textA),  
 agent.generateEmbedding(textB),  
 agent.generateEmbedding(textC)  
]);

// These operations will be automatically batched if they occur within  
// the configured time window (default: 50ms)  
const [resultA, resultB, resultC] = await Promise.all([  
 agent.generateEmbedding(textA),  
 agent.generateEmbedding(textB),  
 agent.generateEmbedding(textC)  
]);

The batch processor:  
1. Collects similar requests within a configurable time window  
2. Combines them into a single MCP server call  
3. Routes the combined request to the appropriate service  
4. Distributes the results back to the original callers  
5. Provides detailed performance metrics for monitoring and optimization

### Agent Tools

#### Material Search Tool

Enables agents to search the KAI material database using text queries, filtering, and metadata.

const results = await agent.invoke('search\_materials', {  
 query: 'white marble',  
 filters: {  
 material\_type: 'tile',  
 color: 'white',  
 finish: 'polished'  
 },  
 limit: 10  
});

const results = await agent.invoke('search\_materials', {  
 query: 'white marble',  
 filters: {  
 material\_type: 'tile',  
 color: 'white',  
 finish: 'polished'  
 },  
 limit: 10  
});

#### Image Analysis Tool

Allows agents to analyze images to extract properties, characteristics, and assess image quality.

const analysis = await agent.invoke('analyze\_image', {  
 imageUrl: 'https://example.com/material.jpg',  
 mode: 'full',  
 detail\_level: 'detailed'  
});

const analysis = await agent.invoke('analyze\_image', {  
 imageUrl: 'https://example.com/material.jpg',  
 mode: 'full',  
 detail\_level: 'detailed'  
});

#### Vector Search Tool

Enables semantic similarity searches using vector embeddings rather than keyword matching.

const similarMaterials = await agent.invoke('vector\_search', {  
 mode: 'text',  
 query: 'luxury italian marble with gold veining',  
 limit: 5,  
 threshold: 0.75  
});

const similarMaterials = await agent.invoke('vector\_search', {  
 mode: 'text',  
 query: 'luxury italian marble with gold veining',  
 limit: 5,  
 threshold: 0.75  
});

#### MCP-Enabled Tools with API Integration

These tools leverage the MCP architecture and connect directly to your API services:

// Vector search using MCP  
const similarMaterials = await agent.invoke('vector\_search', {  
 mode: 'text',  
 query: 'luxury italian marble with gold veining',  
 useMCP: true, // Explicitly use MCP for this operation  
 limit: 5,  
 threshold: 0.75  
});  
  
// Image analysis using MCP with direct API connections  
const analysis = await agent.invoke('analyze\_image', {  
 imageUrl: 'https://example.com/material.jpg',  
 useMCP: true, // Explicitly use MCP for this operation  
 mode: 'full',  
 detail\_level: 'detailed',  
 extractColors: true, // Use actual API parameters  
 extractPatterns: true // that match your ML service  
});

// Vector search using MCP  
const similarMaterials = await agent.invoke('vector\_search', {  
 mode: 'text',  
 query: 'luxury italian marble with gold veining',  
 useMCP: true, // Explicitly use MCP for this operation  
 limit: 5,  
 threshold: 0.75  
});  
  
// Image analysis using MCP with direct API connections  
const analysis = await agent.invoke('analyze\_image', {  
 imageUrl: 'https://example.com/material.jpg',  
 useMCP: true, // Explicitly use MCP for this operation  
 mode: 'full',  
 detail\_level: 'detailed',  
 extractColors: true, // Use actual API parameters  
 extractPatterns: true // that match your ML service  
});

The tools use the following connections:  
- Vector Search: Connects to your Supabase vector database service  
- Image Analysis: Integrates with your ML image processing service  
- LLM Inference: Uses your API endpoints for LLM operations  
- OCR Processing: Connects to your document processing service

### Integration with Existing KAI Components

#### Frontend Integration

The frontend integration connects agents with the user interface:

1. Material Recognition Flow
2. Enhanced file upload with agent-assisted guidance
3. Intelligent analysis of recognition results
4. Interactive Q&A about recognized materials
5. Material Browsing
6. Agent-assisted search and filtering
7. Comparative analysis of similar materials
8. Personalized recommendations based on history and preferences
9. Project Planning
10. Material selection assistance
11. Quantity and cost estimation
12. Compatibility checking and suggestions

Interactive Q&A about recognized materials

Material Browsing

Personalized recommendations based on history and preferences

Project Planning

#### Backend Integration

The backend integration enables system-level intelligence:

1. Knowledge Base Management
2. Automated quality assurance and improvement
3. Intelligent indexing and relationship mapping
4. Anomaly detection and correction
5. System Monitoring
6. Performance analysis and optimization
7. Usage pattern detection
8. Proactive issue identification
9. Admin Interface
10. Natural language querying for complex operations
11. Insight generation and reporting
12. Automated task handling and delegation

Anomaly detection and correction

System Monitoring

Proactive issue identification

Admin Interface

#### MCP Integration

The MCP integration enhances both frontend and backend components:

1. Frontend Optimization
2. Faster response times for agent interactions
3. Streaming responses for a more interactive experience
4. Lower resource usage during agent operations
5. Backend Efficiency
6. Centralized management of ML models
7. Shared model instances across multiple requests
8. Better scalability for high-volume agent operations
9. Cross-Component Communication
10. MCP server as a bridge between packages
11. Standardized protocol for model operations
12. Consistent versioning across the system

Lower resource usage during agent operations

Backend Efficiency

Better scalability for high-volume agent operations

Cross-Component Communication

### Implementation Roadmap

#### Phase 1: Foundation (Completed)

* Basic package structure and configuration
* Core agent system architecture
* Agent type definitions and interfaces
* Initial tool implementations
* Base agent implementations (RecognitionAssistant, KnowledgeBaseAgent)

#### Phase 2: Frontend Integration (Months 1-2)

* Frontend interface components for agent interaction
* Integration with existing recognition workflow
* User feedback collection and adaptation
* Comprehensive testing and refinement

#### Phase 3: Backend Integration (Months 3-4)

* Admin interface enhancements for agent interaction
* System monitoring and analytics dashboards
* Integration with existing knowledge base operations
* Automated workflows and processes

#### Phase 4: Advanced Capabilities (Months 5-6)

* Multi-agent collaboration for complex tasks
* Adaptive learning based on user feedback
* Performance optimization and scaling
* Additional specialized agents and tools

#### Phase 5: MCP Migration (Months 7-8) - Completed

* MCP integration for LLM operations ✓
* Performance measurement and optimization ✓
* Batch processing implementation ✓
* Advanced security and monitoring ✓
* Real service connections established ✓
* Mock implementations removed ✓

### Next Development Steps

While the core MCP integration for LLM is in place, further work includes:

1. Complete Agent/Tool Logic: Flesh out the specific implementations within agent classes and tools, replacing any remaining placeholders with robust logic and error handling.
2. Streaming Fallback: Implement streaming support in the local LLMService fallback path or clearly document the limitation.
3. Backend Agent MCP Usage: Review and potentially implement MCP integration for backend agents if required.
4. Enhance Frontend Integration: Improve real-time communication (WebSockets), state management, and UI feedback for agent interactions.
5. Testing: Develop comprehensive unit, integration, and potentially end-to-end tests for agents, tools, and MCP interactions.
6. Configuration & Deployment: Finalize environment variable documentation, create deployment configurations, implement monitoring, and plan for scaling.

LLMService

### Known Issues

1. Local Streaming Fallback: The local LLMService used when MCP is unavailable does not currently support streaming responses.
2. Placeholder Implementations: Some specific agent behaviors or tool functionalities might still be placeholders requiring full implementation.
3. Error Handling Granularity: Error handling in some tools or specific agent logic might need refinement beyond basic try/catch blocks.

LLMService

### Verification and Testing

After setting up the environment variables, you can verify your configuration using the provided verification script:

# Navigate to the agents package  
cd packages/agents  
  
# Run the verification script  
yarn verify  
# or  
npm run verify

# Navigate to the agents package  
cd packages/agents  
  
# Run the verification script  
yarn verify  
# or  
npm run verify

This script will:  
- Validate all required environment variables  
- Check connections to OpenAI API and KAI services  
- Report any issues or missing configurations

You can also test the integration with actual services using:

# Run integration tests  
yarn test:integration  
# or  
npm run test:integration

# Run integration tests  
yarn test:integration  
# or  
npm run test:integration

### Troubleshooting

#### Common Issues

##### OpenAI API Authentication Failures

If you see errors like "Authentication failed with OpenAI":

1. Verify your OPENAI\_API\_KEY is correctly set
2. Ensure your OpenAI account has billing information if required
3. Check that you're using a supported model name in OPENAI\_DEFAULT\_MODEL

OPENAI\_API\_KEY

OPENAI\_DEFAULT\_MODEL

##### Service Connection Issues

If you encounter errors connecting to KAI services:

1. Verify all service URLs are correctly set
2. Ensure the services are running (for local development)
3. Check network connectivity and firewall settings
4. Verify your authentication credentials are valid

##### Token Refresh Failures

If authentication tokens aren't refreshing properly:

1. Check that your authentication configuration is correct
2. Ensure the auth service is available
3. Verify user permissions for the required operations

#### Fallback Mechanisms

The integration includes fallback mechanisms that use mock implementations when services are unavailable. This is useful during development or when certain services aren't yet deployed.

To control fallback behavior, you can use:

# Enable mock fallbacks (default: true in development, false in production)  
ENABLE\_MOCK\_FALLBACK=true

# Enable mock fallbacks (default: true in development, false in production)  
ENABLE\_MOCK\_FALLBACK=true

### Technical Considerations

#### Performance and Scaling

* Agent operations can be resource-intensive, especially for complex reasoning tasks
* Consider implementing:
* Caching for common agent responses
* Rate limiting for API-dependent operations
* Asynchronous processing for non-interactive tasks
* Horizontal scaling for high-volume deployments
* MCP integration for optimized model operations

#### Security

* Agents operate with least privilege principle
* All agent operations are logged for audit purposes
* User data is handled according to existing platform policies
* Input validation is implemented for all agent inputs
* Output filtering ensures appropriate agent responses
* MCP authentication is handled securely with automatic token rotation

#### Error Handling

* Agents implement graceful degradation on API failures
* Fallback mechanisms ensure continuity of service
* Comprehensive logging aids in debugging and issue resolution
* Monitoring systems alert administrators to repeated errors
* MCP health checks prevent requests to unavailable services

### Related Documentation

* Material Recognition
* Knowledge Base
* PDF Processing
* Queue System
* MCP Server
* MCP Integration Documentation
* CrewAI Documentation
* OpenAI API Reference
* Redis Documentation

# Agents Unified Services

Source: readme/agents-unified-services.md

---

## Agents Package Unified Services

This document describes the implementation of the unified services architecture in the agents package. The unified services architecture completely removes backward compatibility layers and uses the unified services directly.

### Implementation Overview

The agents package has been updated to use the unified services from the shared package directly. The following changes were made:

1. Removed duplicate files:
2. Removed authService.ts - Replaced with unified auth service
3. Removed baseService.ts - Replaced with unified API client
4. Removed logger.ts - Replaced with unified logger
5. Removed environment.ts - Replaced with unified config
6. Created a unified services export:
7. Created services/index.ts to export all unified services from the shared package
8. Updated components to use unified services directly:
9. Updated agentSystem.ts to use the unified auth service, logger, and config
10. Created activityLogger.ts to provide specialized logging for agent activities

authService.ts

baseService.ts

logger.ts

Removed environment.ts - Replaced with unified config

environment.ts

Created a unified services export:

Created services/index.ts to export all unified services from the shared package

services/index.ts

Updated components to use unified services directly:

agentSystem.ts

activityLogger.ts

### Unified Services Export

The services/index.ts file exports all the unified services from the shared package for use throughout the agents package. It also provides a function to initialize all services.

services/index.ts

/\*\*  
 \* Unified Services Export  
 \*   
 \* This file exports all the unified services from the shared package  
 \* for use throughout the agents package.  
 \*/  
  
// Export auth service  
export {   
 auth,   
 initializeAuth,   
 User,   
 LoginCredentials,   
 RegisterCredentials,   
 AuthResult   
} from '@kai/shared';  
  
// Export API client  
export {   
 apiClient,   
 createApiClient,   
 ApiClientConfig,   
 ApiError,  
 BaseService,  
 ServiceConfig  
} from '@kai/shared';  
  
// Export MCP client  
export {   
 mcpClient,   
 createMCPClient   
} from '@kai/shared';  
  
// Export Supabase client  
export {   
 supabase   
} from '@kai/shared';  
  
// Export storage service  
export {   
 storage,   
 initializeStorage,   
 StorageProvider   
} from '@kai/shared';  
  
// Export logger  
export {   
 createLogger,   
 LogLevel   
} from '@kai/shared';  
  
// Export config  
export {   
 config   
} from '@kai/shared';  
  
/\*\*  
 \* Initialize all services  
 \*   
 \* This function initializes all the unified services.  
 \* It should be called early in the application lifecycle.  
 \*/  
export function initializeServices(): void {  
 // Initialize auth service  
 initializeAuth();  
  
 // Initialize storage service  
 initializeStorage();  
  
 // Log that services have been initialized  
 const logger = createLogger('Services');  
 logger.info('Unified services initialized');  
}

/\*\*  
 \* Unified Services Export  
 \*   
 \* This file exports all the unified services from the shared package  
 \* for use throughout the agents package.  
 \*/  
  
// Export auth service  
export {   
 auth,   
 initializeAuth,   
 User,   
 LoginCredentials,   
 RegisterCredentials,   
 AuthResult   
} from '@kai/shared';  
  
// Export API client  
export {   
 apiClient,   
 createApiClient,   
 ApiClientConfig,   
 ApiError,  
 BaseService,  
 ServiceConfig  
} from '@kai/shared';  
  
// Export MCP client  
export {   
 mcpClient,   
 createMCPClient   
} from '@kai/shared';  
  
// Export Supabase client  
export {   
 supabase   
} from '@kai/shared';  
  
// Export storage service  
export {   
 storage,   
 initializeStorage,   
 StorageProvider   
} from '@kai/shared';  
  
// Export logger  
export {   
 createLogger,   
 LogLevel   
} from '@kai/shared';  
  
// Export config  
export {   
 config   
} from '@kai/shared';  
  
/\*\*  
 \* Initialize all services  
 \*   
 \* This function initializes all the unified services.  
 \* It should be called early in the application lifecycle.  
 \*/  
export function initializeServices(): void {  
 // Initialize auth service  
 initializeAuth();  
  
 // Initialize storage service  
 initializeStorage();  
  
 // Log that services have been initialized  
 const logger = createLogger('Services');  
 logger.info('Unified services initialized');  
}

### Agent System

The agentSystem.ts file has been updated to use the unified services directly. It now imports the auth service, logger, and config from the shared package and uses them for all operations.

agentSystem.ts

Key changes:  
- Imports the unified auth service, logger, and config from the shared package  
- Uses the unified auth service for authentication  
- Uses the unified config for configuration  
- Uses the unified logger for logging

### Activity Logger

The activityLogger.ts file provides specialized logging for agent activities. It uses the unified logger from the shared package and adds agent-specific context to log entries.

activityLogger.ts

/\*\*  
 \* Agent Activity Logger  
 \*   
 \* Provides specialized logging for agent activities, including task execution,  
 \* agent creation, and other agent-related events.  
 \*/  
  
import { createLogger } from '../services';  
  
// Create a specialized logger for agent activities  
const activityLogger = createLogger('AgentActivity');  
  
/\*\*  
 \* Log agent activity  
 \*   
 \* @param agentId - ID of the agent  
 \* @param activity - Activity details  
 \*/  
export function logAgentActivity(  
 agentId: string,  
 activity: {  
 action: 'agent\_creation' | 'agent\_deletion' | 'task\_execution' | 'task\_completion' | 'error';  
 status: 'start' | 'success' | 'error' | 'warning';  
 details?: Record<string, any>;  
 error?: Error;  
 }  
): void {  
 const { action, status, details, error } = activity;  
  
 // Create a structured log entry  
 const logEntry = {  
 agentId,  
 action,  
 status,  
 timestamp: new Date().toISOString(),  
 ...details  
 };  
  
 // Log at the appropriate level based on status  
 switch (status) {  
 case 'start':  
 activityLogger.info(`Agent ${agentId} ${action} started`, logEntry);  
 break;  
 case 'success':  
 activityLogger.info(`Agent ${agentId} ${action} succeeded`, logEntry);  
 break;  
 case 'warning':  
 activityLogger.warn(`Agent ${agentId} ${action} warning`, logEntry);  
 break;  
 case 'error':  
 activityLogger.error(`Agent ${agentId} ${action} failed`, error, logEntry);  
 break;  
 }  
}

/\*\*  
 \* Agent Activity Logger  
 \*   
 \* Provides specialized logging for agent activities, including task execution,  
 \* agent creation, and other agent-related events.  
 \*/  
  
import { createLogger } from '../services';  
  
// Create a specialized logger for agent activities  
const activityLogger = createLogger('AgentActivity');  
  
/\*\*  
 \* Log agent activity  
 \*   
 \* @param agentId - ID of the agent  
 \* @param activity - Activity details  
 \*/  
export function logAgentActivity(  
 agentId: string,  
 activity: {  
 action: 'agent\_creation' | 'agent\_deletion' | 'task\_execution' | 'task\_completion' | 'error';  
 status: 'start' | 'success' | 'error' | 'warning';  
 details?: Record<string, any>;  
 error?: Error;  
 }  
): void {  
 const { action, status, details, error } = activity;  
  
 // Create a structured log entry  
 const logEntry = {  
 agentId,  
 action,  
 status,  
 timestamp: new Date().toISOString(),  
 ...details  
 };  
  
 // Log at the appropriate level based on status  
 switch (status) {  
 case 'start':  
 activityLogger.info(`Agent ${agentId} ${action} started`, logEntry);  
 break;  
 case 'success':  
 activityLogger.info(`Agent ${agentId} ${action} succeeded`, logEntry);  
 break;  
 case 'warning':  
 activityLogger.warn(`Agent ${agentId} ${action} warning`, logEntry);  
 break;  
 case 'error':  
 activityLogger.error(`Agent ${agentId} ${action} failed`, error, logEntry);  
 break;  
 }  
}

### Benefits

The direct unified services architecture provides several benefits:

1. Simplified codebase: No more compatibility layers or adapter files
2. Reduced code duplication: Common functionality is implemented once in the shared package
3. Improved maintainability: Changes to common functionality only need to be made in one place
4. Consistent behavior: All parts of the application use the same implementation of common functionality
5. Type safety: The unified services provide type-safe interfaces for common operations
6. Extensibility: The provider pattern allows adding new implementations without changing client code

### Next Steps

The following steps are recommended to further improve the unified services architecture:

1. Update the ml package to use the unified services directly
2. Add more storage providers (Google Cloud Storage, Azure Blob Storage, etc.)
3. Add more authentication providers (SAML, etc.)
4. Implement caching mechanisms for improved performance
5. Add more comprehensive monitoring and telemetry

# Alerting Condition Types

Source: readme/alerting-condition-types.md

---

## Alerting Service Condition Types

This document describes the implementation of all condition types in the alerting service. The alerting service provides a unified interface for generating alerts based on telemetry data, enabling proactive monitoring and issue detection.

### Overview

The alerting service supports several types of conditions for triggering alerts:

1. Threshold Condition: Triggers when a metric exceeds a threshold
2. Frequency Condition: Triggers when events occur at a certain frequency
3. Absence Condition: Triggers when no events are received for a period of time
4. Change Condition: Triggers when a metric changes by a certain amount
5. Custom Condition: Triggers based on custom logic

### Condition Types

#### Threshold Condition

The threshold condition triggers an alert when a metric exceeds a threshold. It supports various comparison operators and aggregation functions.

##### Properties

* type: AlertRuleConditionType.THRESHOLD
* metric: The metric to monitor (required)
* threshold: The threshold value (required)
* operator: The comparison operator (required)
* gt: Greater than
* lt: Less than
* eq: Equal to
* ne: Not equal to
* ge: Greater than or equal to
* le: Less than or equal to
* timeWindow: The time window in seconds (optional)
* properties: Additional properties (optional)
* aggregation: The aggregation function (optional)  
  avg: Average (default)  
  max: Maximum  
  min: Minimum  
  sum: Sum  
  count: Count  
  last: Last value

AlertRuleConditionType.THRESHOLD

gt

lt

eq

ne

ge

le

* avg: Average (default)
* max: Maximum
* min: Minimum
* sum: Sum
* count: Count
* last: Last value

avg

max

min

sum

count

last

##### Example

const thresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt',  
 timeWindow: 300, // 5 minutes  
 properties: {  
 aggregation: 'avg'  
 }  
};

const thresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt',  
 timeWindow: 300, // 5 minutes  
 properties: {  
 aggregation: 'avg'  
 }  
};

##### Implementation

The threshold condition is implemented as follows:

1. Filter events within the time window if specified
2. Extract metric values from events (from properties or measurements)
3. Calculate the aggregate value based on the aggregation function
4. Compare the aggregate value to the threshold using the specified operator

#### Frequency Condition

The frequency condition triggers an alert when events occur at a certain frequency. It counts the number of events within a time window and triggers if the count exceeds a threshold.

##### Properties

* type: AlertRuleConditionType.FREQUENCY
* timeWindow: The time window in seconds (required)
* minCount: The minimum count of events (required)

AlertRuleConditionType.FREQUENCY

##### Example

const frequencyCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.FREQUENCY,  
 timeWindow: 300, // 5 minutes  
 minCount: 5  
};

const frequencyCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.FREQUENCY,  
 timeWindow: 300, // 5 minutes  
 minCount: 5  
};

##### Implementation

The frequency condition is implemented as follows:

1. Filter events within the time window
2. Count the number of events
3. Compare the count to the minimum count

#### Absence Condition

The absence condition triggers an alert when no events are received for a period of time. It's useful for detecting when a service or component is down.

##### Properties

* type: AlertRuleConditionType.ABSENCE
* timeWindow: The time window in seconds (required)

AlertRuleConditionType.ABSENCE

##### Example

const absenceCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.ABSENCE,  
 timeWindow: 300 // 5 minutes  
};

const absenceCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.ABSENCE,  
 timeWindow: 300 // 5 minutes  
};

##### Implementation

The absence condition is implemented as follows:

1. Filter events within the time window
2. Check if there are no events in the time window

#### Change Condition

The change condition triggers an alert when a metric changes by a certain amount. It supports both absolute and percentage changes.

##### Properties

* type: AlertRuleConditionType.CHANGE
* metric: The metric to monitor (required)
* threshold: The threshold value (required)
* operator: The comparison operator (required)
* timeWindow: The time window in seconds (optional)
* properties: Additional properties (optional)
* usePercentage: Whether to use percentage change (optional, default: false)

AlertRuleConditionType.CHANGE

##### Example

const changeCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.CHANGE,  
 metric: 'response\_time',  
 threshold: 50,  
 operator: 'gt',  
 timeWindow: 300, // 5 minutes  
 properties: {  
 usePercentage: true  
 }  
};

const changeCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.CHANGE,  
 metric: 'response\_time',  
 threshold: 50,  
 operator: 'gt',  
 timeWindow: 300, // 5 minutes  
 properties: {  
 usePercentage: true  
 }  
};

##### Implementation

The change condition is implemented as follows:

1. Filter events within the time window if specified
2. Extract metric values from events (from properties or measurements)
3. Calculate the change between the first and last values
4. Calculate the percentage change if usePercentage is true
5. Compare the change to the threshold using the specified operator

#### Custom Condition

The custom condition allows for custom logic to evaluate events. It's useful for complex conditions that can't be expressed using the other condition types.

##### Properties

* type: AlertRuleConditionType.CUSTOM
* evaluate: A function that evaluates events and returns a boolean (required)

AlertRuleConditionType.CUSTOM

##### Example

const customCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.CUSTOM,  
 evaluate: (events) => {  
 // Custom logic to evaluate events  
 return events.some(event =>   
 event.properties?.statusCode === 500 &&   
 event.properties?.endpoint === '/api/users'  
 );  
 }  
};

const customCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.CUSTOM,  
 evaluate: (events) => {  
 // Custom logic to evaluate events  
 return events.some(event =>   
 event.properties?.statusCode === 500 &&   
 event.properties?.endpoint === '/api/users'  
 );  
 }  
};

##### Implementation

The custom condition is implemented by calling the evaluate function with the events.

### Usage

Alert conditions are used in alert rules to define when an alert should be triggered. Multiple conditions can be combined in a single rule, and all conditions must be met for the alert to be triggered.

const rule: AlertRule = {  
 id: 'api-error-alert',  
 name: 'API Error Alert',  
 description: 'Alert on API error events',  
 severity: AlertSeverity.ERROR,  
 eventTypes: ['error'],  
 eventNames: ['api\_error'],  
 conditions: [  
 {  
 type: AlertRuleConditionType.FREQUENCY,  
 timeWindow: 300, // 5 minutes  
 minCount: 3  
 },  
 {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt',  
 timeWindow: 300, // 5 minutes  
 properties: {  
 aggregation: 'avg'  
 }  
 }  
 ],  
 enabled: true  
};  
  
alerting.addRule(rule);

const rule: AlertRule = {  
 id: 'api-error-alert',  
 name: 'API Error Alert',  
 description: 'Alert on API error events',  
 severity: AlertSeverity.ERROR,  
 eventTypes: ['error'],  
 eventNames: ['api\_error'],  
 conditions: [  
 {  
 type: AlertRuleConditionType.FREQUENCY,  
 timeWindow: 300, // 5 minutes  
 minCount: 3  
 },  
 {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt',  
 timeWindow: 300, // 5 minutes  
 properties: {  
 aggregation: 'avg'  
 }  
 }  
 ],  
 enabled: true  
};  
  
alerting.addRule(rule);

### Benefits

The implementation of all condition types provides several benefits:

1. Flexibility: Support for different types of conditions allows for flexible alert rules
2. Customization: Custom conditions allow for complex logic that can't be expressed using the other condition types
3. Aggregation: Support for different aggregation functions allows for more precise alerting
4. Time Windows: Support for time windows allows for alerting based on recent events
5. Comparison Operators: Support for different comparison operators allows for more precise alerting

### Next Steps

The following steps are recommended to further improve the alerting service:

1. Add More Condition Types: Add support for more condition types (trend, anomaly, etc.)
2. Add More Aggregation Functions: Add support for more aggregation functions (median, percentile, etc.)
3. Add Support for Multiple Metrics: Add support for conditions that involve multiple metrics
4. Add Support for Composite Conditions: Add support for conditions that combine multiple conditions with logical operators (AND, OR, NOT)
5. Add Support for Dynamic Thresholds: Add support for thresholds that are calculated dynamically based on historical data

# Alerting Service

Source: readme/alerting-service.md

---

## Alerting Service

This document describes the alerting service implementation in the KAI platform. The alerting service provides a unified interface for generating alerts based on telemetry data, enabling proactive monitoring and issue detection.

### Overview

The alerting service is designed to monitor telemetry events and trigger alerts when specific conditions are met. It provides a consistent API for defining alert rules, notification channels, and alert management. The service integrates with the telemetry service to collect events and the event bus to publish alert notifications.

### Architecture

The alerting service follows a rule-based architecture, where alert rules define conditions that trigger alerts. The service consists of the following components:

1. Alerting Service: The main service that provides a unified interface for alerting operations.
2. Alert Rules: Rules that define conditions for triggering alerts.
3. Notification Channels: Channels for sending alert notifications.
4. Alert Management: Functions for managing active alerts.

### Usage

#### Basic Usage

import { alerting, AlertSeverity, AlertRuleConditionType } from '@kai/shared';  
  
// Add an alert rule  
alerting.addRule({  
 id: 'error-alert',  
 name: 'Error Alert',  
 description: 'Alert on error events',  
 severity: AlertSeverity.ERROR,  
 eventTypes: ['error'],  
 conditions: [  
 {  
 type: AlertRuleConditionType.FREQUENCY,  
 timeWindow: 300, // 5 minutes  
 minCount: 5  
 }  
 ],  
 enabled: true  
});  
  
// Add a notification channel  
alerting.addChannel({  
 id: 'console',  
 name: 'Console',  
 type: AlertNotificationChannelType.CONSOLE,  
 config: {},  
 enabled: true  
});  
  
// Get active alerts  
const alerts = alerting.getAlerts();  
  
// Acknowledge an alert  
alerting.acknowledgeAlert('alert-id');  
  
// Resolve an alert  
alerting.resolveAlert('alert-id');

import { alerting, AlertSeverity, AlertRuleConditionType } from '@kai/shared';  
  
// Add an alert rule  
alerting.addRule({  
 id: 'error-alert',  
 name: 'Error Alert',  
 description: 'Alert on error events',  
 severity: AlertSeverity.ERROR,  
 eventTypes: ['error'],  
 conditions: [  
 {  
 type: AlertRuleConditionType.FREQUENCY,  
 timeWindow: 300, // 5 minutes  
 minCount: 5  
 }  
 ],  
 enabled: true  
});  
  
// Add a notification channel  
alerting.addChannel({  
 id: 'console',  
 name: 'Console',  
 type: AlertNotificationChannelType.CONSOLE,  
 config: {},  
 enabled: true  
});  
  
// Get active alerts  
const alerts = alerting.getAlerts();  
  
// Acknowledge an alert  
alerting.acknowledgeAlert('alert-id');  
  
// Resolve an alert  
alerting.resolveAlert('alert-id');

#### Alert Rules

Alert rules define conditions that trigger alerts. Each rule consists of:

* ID: Unique identifier for the rule
* Name: Human-readable name for the rule
* Description: Description of the rule
* Severity: Severity level of alerts triggered by the rule
* Event Types: Types of telemetry events to monitor
* Event Names: Optional names of telemetry events to monitor
* Conditions: Conditions that must be met to trigger an alert
* Tags: Optional tags for categorizing the rule
* Properties: Optional additional properties for the rule
* Enabled: Whether the rule is enabled

const rule: AlertRule = {  
 id: 'api-error-alert',  
 name: 'API Error Alert',  
 description: 'Alert on API error events',  
 severity: AlertSeverity.ERROR,  
 eventTypes: ['error'],  
 eventNames: ['api\_error'],  
 conditions: [  
 {  
 type: AlertRuleConditionType.FREQUENCY,  
 timeWindow: 300, // 5 minutes  
 minCount: 3  
 }  
 ],  
 tags: ['api', 'error'],  
 properties: {  
 team: 'backend'  
 },  
 enabled: true  
};  
  
alerting.addRule(rule);

const rule: AlertRule = {  
 id: 'api-error-alert',  
 name: 'API Error Alert',  
 description: 'Alert on API error events',  
 severity: AlertSeverity.ERROR,  
 eventTypes: ['error'],  
 eventNames: ['api\_error'],  
 conditions: [  
 {  
 type: AlertRuleConditionType.FREQUENCY,  
 timeWindow: 300, // 5 minutes  
 minCount: 3  
 }  
 ],  
 tags: ['api', 'error'],  
 properties: {  
 team: 'backend'  
 },  
 enabled: true  
};  
  
alerting.addRule(rule);

#### Alert Conditions

Alert conditions define the specific criteria that must be met to trigger an alert. The alerting service supports several types of conditions:

* Threshold: Triggers when a metric exceeds a threshold
* Change: Triggers when a metric changes by a certain amount
* Absence: Triggers when no events are received for a period of time
* Frequency: Triggers when events occur at a certain frequency
* Custom: Triggers based on custom logic

// Threshold condition  
const thresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt'  
};  
  
// Frequency condition  
const frequencyCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.FREQUENCY,  
 timeWindow: 300, // 5 minutes  
 minCount: 5  
};  
  
// Custom condition  
const customCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.CUSTOM,  
 evaluate: (events) => {  
 // Custom logic to evaluate events  
 return events.some(event =>   
 event.properties?.statusCode === 500 &&   
 event.properties?.endpoint === '/api/users'  
 );  
 }  
};

// Threshold condition  
const thresholdCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.THRESHOLD,  
 metric: 'response\_time',  
 threshold: 1000,  
 operator: 'gt'  
};  
  
// Frequency condition  
const frequencyCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.FREQUENCY,  
 timeWindow: 300, // 5 minutes  
 minCount: 5  
};  
  
// Custom condition  
const customCondition: AlertRuleCondition = {  
 type: AlertRuleConditionType.CUSTOM,  
 evaluate: (events) => {  
 // Custom logic to evaluate events  
 return events.some(event =>   
 event.properties?.statusCode === 500 &&   
 event.properties?.endpoint === '/api/users'  
 );  
 }  
};

#### Notification Channels

Notification channels define how alerts are delivered. The alerting service supports several types of channels:

* Console: Logs alerts to the console
* Email: Sends alerts via email
* Webhook: Sends alerts to a webhook
* Custom: Sends alerts using custom logic

// Console channel  
const consoleChannel: AlertNotificationChannel = {  
 id: 'console',  
 name: 'Console',  
 type: AlertNotificationChannelType.CONSOLE,  
 config: {},  
 enabled: true  
};  
  
// Email channel  
const emailChannel: AlertNotificationChannel = {  
 id: 'email',  
 name: 'Email',  
 type: AlertNotificationChannelType.EMAIL,  
 config: {  
 recipients: ['alerts@example.com'],  
 subject: '[KAI] Alert: {alert.name}'  
 },  
 enabled: true  
};  
  
// Webhook channel  
const webhookChannel: AlertNotificationChannel = {  
 id: 'webhook',  
 name: 'Webhook',  
 type: AlertNotificationChannelType.WEBHOOK,  
 config: {  
 url: 'https://hooks.slack.com/services/...',  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 }  
 },  
 enabled: true  
};

// Console channel  
const consoleChannel: AlertNotificationChannel = {  
 id: 'console',  
 name: 'Console',  
 type: AlertNotificationChannelType.CONSOLE,  
 config: {},  
 enabled: true  
};  
  
// Email channel  
const emailChannel: AlertNotificationChannel = {  
 id: 'email',  
 name: 'Email',  
 type: AlertNotificationChannelType.EMAIL,  
 config: {  
 recipients: ['alerts@example.com'],  
 subject: '[KAI] Alert: {alert.name}'  
 },  
 enabled: true  
};  
  
// Webhook channel  
const webhookChannel: AlertNotificationChannel = {  
 id: 'webhook',  
 name: 'Webhook',  
 type: AlertNotificationChannelType.WEBHOOK,  
 config: {  
 url: 'https://hooks.slack.com/services/...',  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 }  
 },  
 enabled: true  
};

### Configuration

The alerting service can be configured through environment variables or the unified configuration system. The following configuration options are available:

// In .env file  
ALERTING\_ENABLED=true  
ALERTING\_BUFFER\_SIZE=1000  
ALERTING\_EVALUATION\_INTERVAL\_MS=60000

// In .env file  
ALERTING\_ENABLED=true  
ALERTING\_BUFFER\_SIZE=1000  
ALERTING\_EVALUATION\_INTERVAL\_MS=60000

### Implementation Details

#### Alerting Service

The alerting service provides a unified interface for alerting operations. It manages alert rules, notification channels, and active alerts.

class AlertingService {  
 // Initialize the alerting service  
 initialize(): void;  
  
 // Enable alerting  
 enable(): void;  
  
 // Disable alerting  
 disable(): void;  
  
 // Add an alert rule  
 addRule(rule: AlertRule): void;  
  
 // Remove an alert rule  
 removeRule(ruleId: string): void;  
  
 // Get an alert rule  
 getRule(ruleId: string): AlertRule | undefined;  
  
 // Get all alert rules  
 getRules(): AlertRule[];  
  
 // Add a notification channel  
 addChannel(channel: AlertNotificationChannel): void;  
  
 // Remove a notification channel  
 removeChannel(channelId: string): void;  
  
 // Get a notification channel  
 getChannel(channelId: string): AlertNotificationChannel | undefined;  
  
 // Get all notification channels  
 getChannels(): AlertNotificationChannel[];  
  
 // Get an alert  
 getAlert(alertId: string): Alert | undefined;  
  
 // Get all alerts  
 getAlerts(): Alert[];  
  
 // Acknowledge an alert  
 acknowledgeAlert(alertId: string): void;  
  
 // Resolve an alert  
 resolveAlert(alertId: string): void;  
  
 // Process a telemetry event  
 processEvent(event: TelemetryEvent): void;  
  
 // Evaluate all alert rules  
 evaluateRules(): Promise<void>;  
}

class AlertingService {  
 // Initialize the alerting service  
 initialize(): void;  
  
 // Enable alerting  
 enable(): void;  
  
 // Disable alerting  
 disable(): void;  
  
 // Add an alert rule  
 addRule(rule: AlertRule): void;  
  
 // Remove an alert rule  
 removeRule(ruleId: string): void;  
  
 // Get an alert rule  
 getRule(ruleId: string): AlertRule | undefined;  
  
 // Get all alert rules  
 getRules(): AlertRule[];  
  
 // Add a notification channel  
 addChannel(channel: AlertNotificationChannel): void;  
  
 // Remove a notification channel  
 removeChannel(channelId: string): void;  
  
 // Get a notification channel  
 getChannel(channelId: string): AlertNotificationChannel | undefined;  
  
 // Get all notification channels  
 getChannels(): AlertNotificationChannel[];  
  
 // Get an alert  
 getAlert(alertId: string): Alert | undefined;  
  
 // Get all alerts  
 getAlerts(): Alert[];  
  
 // Acknowledge an alert  
 acknowledgeAlert(alertId: string): void;  
  
 // Resolve an alert  
 resolveAlert(alertId: string): void;  
  
 // Process a telemetry event  
 processEvent(event: TelemetryEvent): void;  
  
 // Evaluate all alert rules  
 evaluateRules(): Promise<void>;  
}

### Benefits

The alerting service provides several benefits:

1. Proactive Monitoring: Detect issues before they impact users.
2. Customizable Rules: Define alert rules based on specific criteria.
3. Multiple Notification Channels: Send alerts through different channels.
4. Alert Management: Acknowledge and resolve alerts.
5. Integration with Telemetry: Use telemetry data to trigger alerts.
6. Event-Based Architecture: Publish alert events for other services to consume.

### Next Steps

The following steps are recommended to further improve the alerting service:

1. Implement Condition Types: Complete the implementation of all condition types.
2. Add More Notification Channels: Add support for more notification channels (SMS, PagerDuty, etc.).
3. Add Alert Aggregation: Group related alerts to reduce noise.
4. Add Alert Escalation: Escalate alerts based on severity and time.
5. Add Alert History: Store alert history for analysis.
6. Add Alert Dashboards: Create dashboards for visualizing alerts.
7. Add Alert Templates: Create templates for alert notifications.

# Alternatives

Source: readme/alternatives.md

---

## Alternative Applications for the Recognition System

This document outlines how our material recognition system can be adapted to recognize other types of objects, such as shoes, clothing, furniture, or other product categories. The modular architecture of our system makes it highly adaptable to new domains with relatively modest changes.

### Current Architecture Overview

Our material recognition system employs a sophisticated modular design with several key components:

1. Core ML Engines
2. Feature-based recognition (OpenCV, SIFT features)
3. Neural network-based recognition (TensorFlow/PyTorch)
4. Hybrid recognition combining multiple approaches
5. Adaptive method selection framework
6. Embedding Generation System
7. Traditional feature embeddings
8. Neural network embeddings
9. Adaptive hybrid embeddings
10. Quality Assessment
11. Real-time quality evaluation
12. Method switching based on confidence
13. Material-specific optimization
14. Service Layer
15. Clean API interfaces
16. Preprocessing pipelines
17. Result formatting and standardization
18. Client Integration
19. Image upload endpoints
20. URL-based recognition
21. Batch processing
22. Vector search

Adaptive method selection framework

Embedding Generation System

Adaptive hybrid embeddings

Quality Assessment

Material-specific optimization

Service Layer

Result formatting and standardization

Client Integration

### Adapting to New Recognition Domains

#### 1. Architectural Advantages for Adaptation

The current system's architecture offers several advantages when adapting to new domains:

* Method Independence: The core recognition logic is implemented with clear separation between different methods (feature-based, ML-based, hybrid), allowing selective replacement or retraining
* Quality-Driven Selection: The adaptive framework can evaluate recognition quality and dynamically switch methods, which is valuable for any recognition domain
* Modular Embedding Generation: The embedding generation pipeline is modular and can be retrained for new domains while maintaining the same interfaces
* Consistent API Structure: The service layer provides well-defined interfaces that can remain largely unchanged regardless of the underlying recognition domain
* Robust Category Management Backend: While the current UI implementation uses hardcoded categories, the system's backend has a fully-developed Category model with hierarchical structure support, CRUD operations, and API endpoints that could be leveraged for dynamic category management

Method Independence: The core recognition logic is implemented with clear separation between different methods (feature-based, ML-based, hybrid), allowing selective replacement or retraining

Quality-Driven Selection: The adaptive framework can evaluate recognition quality and dynamically switch methods, which is valuable for any recognition domain

Modular Embedding Generation: The embedding generation pipeline is modular and can be retrained for new domains while maintaining the same interfaces

Consistent API Structure: The service layer provides well-defined interfaces that can remain largely unchanged regardless of the underlying recognition domain

Robust Category Management Backend: While the current UI implementation uses hardcoded categories, the system's backend has a fully-developed Category model with hierarchical structure support, CRUD operations, and API endpoints that could be leveraged for dynamic category management

#### 2. Required Modifications for New Domains

##### ML Component Changes

Table content:

Component | Current (Materials) | Required Changes for Other Domains

material\_recognizer.py | Core recognition logic for materials | Rename to domain\_recognizer.py, update class names, modify feature extraction for domain-specific attributes

embedding\_bridge.py | Unified interface for embedding generation | Minimal changes - update logging and variable names

adaptive\_hybrid\_embeddings.py | Dynamic method selection | Update quality assessment metrics to be relevant for the new domain

ML Models | Trained on material dataset | Retrain with domain-specific data

material\_recognizer.py

domain\_recognizer.py

embedding\_bridge.py

adaptive\_hybrid\_embeddings.py

##### Service Layer Changes

Table content:

Component | Current (Materials) | Required Changes for Other Domains

material-recognizer-service.ts | Server-side recognition service | Rename to domain-recognizer-service.ts, update terminology and domain-specific preprocessing

recognition.routes.ts | API endpoints for material recognition | Minimal path changes, update response fields to match new domain

material-recognizer-service.ts

domain-recognizer-service.ts

recognition.routes.ts

##### Data Model Changes

Table content:

Aspect | Current (Materials) | Required Changes for Other Domains

Recognition Result | MaterialRecognitionResult with material-specific properties | Create DomainRecognitionResult with domain-specific properties

Metadata Fields | Material-specific (patternFamily, dimensions) | Replace with domain-specific attributes (e.g., for shoes: style, size, brand)

Database Schema | Optimized for material properties | Update for domain-specific properties

MaterialRecognitionResult

DomainRecognitionResult

##### Category Management Enhancements

Table content:

Current Implementation | Potential Enhancement for New Domains

Hardcoded categories in UI components: ['Tile', 'Stone', 'Wood', 'Ceramic', 'Porcelain', 'Vinyl', 'Laminate', 'Other'] | Leverage existing Category model in backend to create a dynamic category management UI

No dedicated UI for category management | Build a new admin panel component for CRUD operations on categories using existing backend APIs

Static material type checks in code | Replace with dynamic category lookups from database

Material-specific field groups in metadata panel | Generate field groups dynamically based on category definition

['Tile', 'Stone', 'Wood', 'Ceramic', 'Porcelain', 'Vinyl', 'Laminate', 'Other']

Category

#### 3. Domain-Specific Considerations

##### For Shoe Recognition:

Domain-Specific Features:  
- Silhouette/shape detection  
- Logo/brand recognition  
- Style classification  
- Color pattern recognition  
- Material composition identification

Recognition Result Fields:

interface ShoeRecognitionResult {  
 shoeType: string; // e.g., "sneaker", "boot", "sandal"  
 confidence: number;  
 brand?: string;  
 style?: string;  
 colorway?: string;  
 materials?: string[];  
 similarModels?: {  
 modelId: string;  
 similarity: number;  
 imageUrl: string;  
 }[];  
}

interface ShoeRecognitionResult {  
 shoeType: string; // e.g., "sneaker", "boot", "sandal"  
 confidence: number;  
 brand?: string;  
 style?: string;  
 colorway?: string;  
 materials?: string[];  
 similarModels?: {  
 modelId: string;  
 similarity: number;  
 imageUrl: string;  
 }[];  
}

Category Management Implementation:  
Instead of hardcoding shoe categories like:

const shoeCategories = ['Sneaker', 'Boot', 'Sandal', 'Flat', 'Heel', 'Athletic'];

const shoeCategories = ['Sneaker', 'Boot', 'Sandal', 'Flat', 'Heel', 'Athletic'];

You would:  
1. Use the existing Category model to create and manage categories  
2. Develop a UI component to display and edit these categories  
3. Fetch categories dynamically from the database:

const getShoeCategories = async () => {  
 // Using existing category API endpoints  
 const response = await fetch('/api/admin/categories?type=shoe');  
 const data = await response.json();  
 return data.categories;  
};

const getShoeCategories = async () => {  
 // Using existing category API endpoints  
 const response = await fetch('/api/admin/categories?type=shoe');  
 const data = await response.json();  
 return data.categories;  
};

##### For Clothing Recognition:

Domain-Specific Features:  
- Garment type detection  
- Fabric pattern recognition  
- Style classification  
- Fashion trend analysis  
- Occasion categorization

Recognition Result Fields:

interface ClothingRecognitionResult {  
 garmentType: string; // e.g., "shirt", "dress", "pants"  
 confidence: number;  
 style?: string;  
 pattern?: string;  
 fabric?: string;  
 season?: string;  
 occasion?: string[];  
 similarItems?: {  
 itemId: string;  
 similarity: number;  
 imageUrl: string;  
 }[];  
}

interface ClothingRecognitionResult {  
 garmentType: string; // e.g., "shirt", "dress", "pants"  
 confidence: number;  
 style?: string;  
 pattern?: string;  
 fabric?: string;  
 season?: string;  
 occasion?: string[];  
 similarItems?: {  
 itemId: string;  
 similarity: number;  
 imageUrl: string;  
 }[];  
}

##### For Furniture Recognition:

Domain-Specific Features:  
- Furniture category detection  
- Style classification  
- Material composition identification  
- Structural elements recognition  
- Period/era identification

Recognition Result Fields:

interface FurnitureRecognitionResult {  
 furnitureType: string; // e.g., "chair", "table", "sofa"  
 confidence: number;  
 style?: string;  
 period?: string;  
 materials?: string[];  
 dimensions?: {  
 estimated: boolean;  
 width?: number;  
 height?: number;  
 depth?: number;  
 unit: string;  
 };  
 similarItems?: {  
 itemId: string;  
 similarity: number;  
 imageUrl: string;  
 }[];  
}

interface FurnitureRecognitionResult {  
 furnitureType: string; // e.g., "chair", "table", "sofa"  
 confidence: number;  
 style?: string;  
 period?: string;  
 materials?: string[];  
 dimensions?: {  
 estimated: boolean;  
 width?: number;  
 height?: number;  
 depth?: number;  
 unit: string;  
 };  
 similarItems?: {  
 itemId: string;  
 similarity: number;  
 imageUrl: string;  
 }[];  
}

#### 4. Implementation Roadmap

When adapting the system to a new domain, follow this implementation roadmap:

1. Data Collection & Preparation (4-6 weeks)
2. Collect domain-specific dataset (5,000+ labeled images)
3. Define classification taxonomy
4. Implement data augmentation specific to the domain
5. Annotate with domain-specific attributes
6. Model Adaptation & Training (3-4 weeks)
7. Modify feature extraction for domain-specific needs
8. Adapt neural network architecture if needed
9. Train domain-specific models
10. Validate with test set
11. Fine-tune hyperparameters
12. Code Refactoring (2-3 weeks)
13. Update class and variable names
14. Implement domain-specific preprocessing
15. Update result formatting for domain attributes
16. Add domain-specific quality metrics
17. API Integration (1-2 weeks)
18. Update API routes
19. Modify response schemas
20. Update documentation
21. Implement domain-specific endpoint features
22. Testing & Validation (2-3 weeks)
23. Benchmark against baseline
24. User acceptance testing
25. Performance optimization
26. Edge case handling
27. Deployment & Monitoring (1-2 weeks)
28. Containerize updated models
29. Set up monitoring for new domain
30. Implement continuous learning for domain

Annotate with domain-specific attributes

Model Adaptation & Training (3-4 weeks)

Fine-tune hyperparameters

Code Refactoring (2-3 weeks)

Add domain-specific quality metrics

API Integration (1-2 weeks)

Implement domain-specific endpoint features

Testing & Validation (2-3 weeks)

Edge case handling

Deployment & Monitoring (1-2 weeks)

#### 5. Resource Requirements for Adaptation

Table content:

Resource | Requirement | Notes

Dataset | 5,000+ labeled images | Minimum for initial training; more is better

Computing | GPU with 12GB+ VRAM | For training new neural network models

Storage | 50GB+ | For dataset, intermediate files, and models

Development | 3-4 engineers | ML engineer, backend dev, data scientist recommended

Timeline | 12-16 weeks | From data collection to production deployment

#### 6. Case Study: Adaptation to Shoe Recognition

To illustrate the adaptation process, here's how we would adapt the system specifically for shoe recognition:

1. Dataset Requirements
2. 10,000+ shoe images across different types
3. Labels for shoe type, brand, style, color
4. Multiple angles per shoe model
5. Varied backgrounds and lighting conditions
6. Feature Extraction Modifications
7. Enhance edge detection for shoe silhouettes
8. Add specific feature extractors for logo detection
9. Implement color pattern analysis for distinctive shoe features
10. Develop specialized preprocessing for handling reflective materials
11. Model Training Approach
12. Use transfer learning from existing material models
13. Fine-tune with shoe-specific dataset
14. Implement specialized models for brand logo detection
15. Train separate models for athletic vs. formal shoes
16. API Changes
17. Update endpoints from /api/recognition/\* to /api/shoe-recognition/\*
18. Modify response schema to include shoe-specific attributes
19. Add specialized endpoints for brand verification
20. Implementation Timeline
21. Data collection & preparation: 5 weeks
22. Model adaptation & training: 4 weeks
23. Code refactoring: 2 weeks
24. API integration: 1 week
25. Testing & validation: 3 weeks
26. Deployment & monitoring: 1 week
27. Total: ~16 weeks

Varied backgrounds and lighting conditions

Feature Extraction Modifications

Develop specialized preprocessing for handling reflective materials

Model Training Approach

Train separate models for athletic vs. formal shoes

API Changes

/api/recognition/\*

/api/shoe-recognition/\*

Add specialized endpoints for brand verification

Implementation Timeline

### Conclusion

The existing recognition system provides an excellent foundation for adaptation to new domains. With its modular design, adaptive capability, and clean interfaces, transitioning from material recognition to other product domains requires moderate effort focused primarily on data collection, model training, and terminology updates rather than structural changes.

Most importantly, the investment in the adaptive embedding system and quality-based method selection provides significant advantages for any new recognition domain, allowing the system to automatically optimize based on performance metrics specific to that domain.

When planning an adaptation, prioritize:

1. High-quality, diverse training data for the new domain
2. Domain-specific feature extraction enhancement
3. Careful validation and quality assessment
4. Consistent API design across domains

Following this guide, the team can efficiently adapt the material recognition system to new domains, leveraging the robust architecture while focusing resources on the domain-specific aspects that drive recognition quality and user experience.

# Analytics Agent

Source: readme/analytics-agent.md

---

## Analytics Agent

This document provides detailed information about the Analytics Agent, a specialized crewAI agent designed for data analytics, market research, and decision support within the KAI platform.

### Overview

The Analytics Agent uses the KAI platform's analytics data to provide intelligent insights, market research, trend analysis, and decision support. It connects to the analytics API, processes historical data, and delivers context-aware recommendations based on actual usage patterns.

### Key Capabilities

The Analytics Agent offers multiple specialized functions:

1. User Behavior Analysis
2. Understand how users interact with the platform
3. Identify usage patterns and engagement metrics
4. Segment users based on behavior and preferences
5. Recommend enhancements based on actual usage
6. Market Trend Analysis
7. Identify emerging trends in material interests
8. Detect shifts in search patterns over time
9. Analyze seasonal variations in user behavior
10. Provide competitive market insights
11. Decision Support
12. Offer data-driven recommendations for strategic decisions
13. Compare options using historical performance data
14. Identify potential risks and opportunities
15. Quantify expected outcomes of different choices
16. Product Performance Analysis
17. Track the performance of specific products or categories
18. Compare performance across different time periods
19. Identify factors affecting product popularity
20. Recommend improvements based on usage data
21. Natural Language Analytics Queries
22. Process questions about data in plain English
23. Generate appropriate analytics queries from natural language
24. Present findings in an accessible, conversational format
25. Explain technical metrics in business-relevant terms

Recommend enhancements based on actual usage

Market Trend Analysis

Provide competitive market insights

Decision Support

Quantify expected outcomes of different choices

Product Performance Analysis

Recommend improvements based on usage data

Natural Language Analytics Queries

### Architecture

The Analytics Agent integrates with the broader KAI platform through several key components:

#### Component Structure

packages/  
├── agents/  
│ ├── src/  
│ │ ├── backend/  
│ │ │ └── analyticsAgent.ts # Agent implementation  
│ │ ├── services/  
│ │ │ ├── analyticsService.ts # Analytics data access layer  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ ├── analytics.ts # Analytics-specific tools  
│ │ │ └── index.ts # Tool exports  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── client/  
 └── src/  
 └── components/  
 └── agents/  
 ├── AnalyticsPanel.tsx # Client-side interface  
 └── AgentDashboard.tsx # Agent integration in UI

packages/  
├── agents/  
│ ├── src/  
│ │ ├── backend/  
│ │ │ └── analyticsAgent.ts # Agent implementation  
│ │ ├── services/  
│ │ │ ├── analyticsService.ts # Analytics data access layer  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ ├── analytics.ts # Analytics-specific tools  
│ │ │ └── index.ts # Tool exports  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── client/  
 └── src/  
 └── components/  
 └── agents/  
 ├── AnalyticsPanel.tsx # Client-side interface  
 └── AgentDashboard.tsx # Agent integration in UI

#### Architectural Layers

1. Agent Layer (analyticsAgent.ts)
2. Implements the agent's reasoning capabilities
3. Defines specialized methods for analytics tasks
4. Handles event processing and response generation
5. Manages context and memory for ongoing analysis
6. Service Layer (analyticsService.ts)
7. Provides data access methods for analytics information
8. Handles API communication with error management
9. Formats requests and responses appropriately
10. Acts as a centralized client for the analytics API
11. Tool Layer (analytics.ts)
12. Implements specialized tools for the agent to use
13. Translates agent intents into service operations
14. Formats results for agent consumption
15. Handles errors and provides fallbacks
16. UI Layer (AnalyticsPanel.tsx)
17. Presents the agent's capabilities in the user interface
18. Manages agent session and communication
19. Displays analytics visualizations and insights
20. Provides interaction mechanisms for users

analyticsAgent.ts

Manages context and memory for ongoing analysis

Service Layer (analyticsService.ts)

analyticsService.ts

Acts as a centralized client for the analytics API

Tool Layer (analytics.ts)

analytics.ts

Handles errors and provides fallbacks

UI Layer (AnalyticsPanel.tsx)

AnalyticsPanel.tsx

### Implementation Details

#### Agent Implementation

The Analytics Agent is a SystemAgent type that implements specific methods for analytics tasks:

export class AnalyticsAgent implements SystemAgent {  
 // Standard SystemAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // Analytics-specific methods  
 public async getUserBehaviorInsights(startDate?: Date, endDate?: Date, segment?: string): Promise<string>;  
 public async getMarketTrendAnalysis(timeframe?: 'day' | 'week' | 'month', category?: string): Promise<string>;  
 public async getCompetitiveAnalysis(competitorData: Array<{name: string; metrics: Record<string, any>}>, focusAreas?: string[]): Promise<string>;  
 public async getDecisionSupport(decision: string, options: Array<{name: string; pros: string[]; cons: string[]}>, criteria?: Array<{name: string; weight: number}>): Promise<string>;  
 public async getProductPerformanceAnalysis(productId: string, startDate?: Date, endDate?: Date): Promise<string>;  
 public async processAnalyticsQuery(query: string): Promise<string>;  
}

export class AnalyticsAgent implements SystemAgent {  
 // Standard SystemAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // Analytics-specific methods  
 public async getUserBehaviorInsights(startDate?: Date, endDate?: Date, segment?: string): Promise<string>;  
 public async getMarketTrendAnalysis(timeframe?: 'day' | 'week' | 'month', category?: string): Promise<string>;  
 public async getCompetitiveAnalysis(competitorData: Array<{name: string; metrics: Record<string, any>}>, focusAreas?: string[]): Promise<string>;  
 public async getDecisionSupport(decision: string, options: Array<{name: string; pros: string[]; cons: string[]}>, criteria?: Array<{name: string; weight: number}>): Promise<string>;  
 public async getProductPerformanceAnalysis(productId: string, startDate?: Date, endDate?: Date): Promise<string>;  
 public async processAnalyticsQuery(query: string): Promise<string>;  
}

#### Analytics Tools

The Analytics Agent leverages specialized tools to interact with the analytics system:

// Tool for querying analytics events  
const queryTool = await createAnalyticsQueryTool();  
  
// Tool for analyzing trends over time  
const trendsTool = await createTrendAnalysisTool();  
  
// Tool for generating statistics about platform usage  
const statsTool = await createAnalyticsStatsTool();  
  
// Tool for getting top search queries  
const searchQueriesTool = await createTopSearchQueriesTool();  
  
// Tool for getting top agent prompts  
const agentPromptsTool = await createTopAgentPromptsTool();  
  
// Tool for getting top viewed materials  
const materialsTool = await createTopMaterialsTool();  
  
// Create a complete set of tools for the agent  
const tools = await createAnalyticsTools();

// Tool for querying analytics events  
const queryTool = await createAnalyticsQueryTool();  
  
// Tool for analyzing trends over time  
const trendsTool = await createTrendAnalysisTool();  
  
// Tool for generating statistics about platform usage  
const statsTool = await createAnalyticsStatsTool();  
  
// Tool for getting top search queries  
const searchQueriesTool = await createTopSearchQueriesTool();  
  
// Tool for getting top agent prompts  
const agentPromptsTool = await createTopAgentPromptsTool();  
  
// Tool for getting top viewed materials  
const materialsTool = await createTopMaterialsTool();  
  
// Create a complete set of tools for the agent  
const tools = await createAnalyticsTools();

#### Client-Side Integration

The Analytics Agent is integrated into the client interface through a specialized panel with multiple tabs:

1. Query Analytics - Natural language analytics queries
2. Usage Trends - Visualization of platform usage patterns
3. Market Research - Competitive analysis and market insights
4. Decision Support - Data-driven recommendations for decisions

Each tab provides relevant context and examples to help users interact effectively with the agent.

### Setup Instructions

#### Prerequisites

* Functioning KAI platform with analytics system enabled
* CrewAI integration set up according to CrewAI installation guide
* Analytics API available and accessible

#### Installation

The Analytics Agent is included in the standard crewAI integration package:

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

#### Configuration

Configure the agent in your application initialization:

import { initializeAgentSystem, AgentType } from '@kai/agents';  
  
// Initialize the agent system  
await initializeAgentSystem({  
 apiKey: process.env.OPENAI\_API\_KEY,  
 defaultModel: {  
 provider: 'openai',  
 name: 'gpt-4',  
 temperature: 0.7,  
 },  
 logLevel: 'info'  
});  
  
// Create an Analytics Agent instance  
const analyticsAgent = await createAgent({  
 id: 'analytics-expert-1',  
 type: AgentType.ANALYTICS,  
 name: 'Analytics Expert',  
 description: 'Provides data-driven insights and decision support',  
 // Optional additional tools  
 additionalTools: [customAnalyticsTool],  
 // Optional agent-specific settings  
 settings: {  
 dataRetentionPeriod: 90, // days  
 defaultTimeframe: 'month'  
 }  
});

import { initializeAgentSystem, AgentType } from '@kai/agents';  
  
// Initialize the agent system  
await initializeAgentSystem({  
 apiKey: process.env.OPENAI\_API\_KEY,  
 defaultModel: {  
 provider: 'openai',  
 name: 'gpt-4',  
 temperature: 0.7,  
 },  
 logLevel: 'info'  
});  
  
// Create an Analytics Agent instance  
const analyticsAgent = await createAgent({  
 id: 'analytics-expert-1',  
 type: AgentType.ANALYTICS,  
 name: 'Analytics Expert',  
 description: 'Provides data-driven insights and decision support',  
 // Optional additional tools  
 additionalTools: [customAnalyticsTool],  
 // Optional agent-specific settings  
 settings: {  
 dataRetentionPeriod: 90, // days  
 defaultTimeframe: 'month'  
 }  
});

### Usage Examples

#### Backend Usage

import { createAgent, AgentType } from '@kai/agents';  
  
// Create the Analytics Agent  
const analyticsAgent = await createAgent({  
 id: 'analytics-agent-1',  
 type: AgentType.ANALYTICS,  
 name: 'Analytics Expert'  
});  
  
// Get user behavior insights  
const userInsights = await analyticsAgent.instance.getUserBehaviorInsights(  
 new Date('2024-01-01'),  
 new Date('2024-03-31'),  
 'premium-subscribers'  
);  
  
// Analyze market trends  
const trendAnalysis = await analyticsAgent.instance.getMarketTrendAnalysis(  
 'month',  
 'floor-tiles'  
);  
  
// Get decision support for a strategic question  
const decisionSupport = await analyticsAgent.instance.getDecisionSupport(  
 'Should we expand our ceramic or natural stone inventory?',  
 [  
 {  
 name: 'Expand ceramic inventory',  
 pros: ['Lower cost', 'Higher margin', 'More popular in searches'],  
 cons: ['Market saturation', 'Price competition']  
 },  
 {  
 name: 'Expand natural stone inventory',  
 pros: ['Premium segment', 'Less competition', 'Higher ticket value'],  
 cons: ['Higher inventory cost', 'Slower turnover']  
 }  
 ],  
 [  
 { name: 'Profit potential', weight: 0.4 },  
 { name: 'Market demand', weight: 0.3 },  
 { name: 'Operational complexity', weight: 0.2 },  
 { name: 'Brand alignment', weight: 0.1 }  
 ]  
);  
  
// Process natural language analytics query  
const queryResult = await analyticsAgent.instance.processAnalyticsQuery(  
 'Which product categories showed the highest growth in the last quarter?'  
);

import { createAgent, AgentType } from '@kai/agents';  
  
// Create the Analytics Agent  
const analyticsAgent = await createAgent({  
 id: 'analytics-agent-1',  
 type: AgentType.ANALYTICS,  
 name: 'Analytics Expert'  
});  
  
// Get user behavior insights  
const userInsights = await analyticsAgent.instance.getUserBehaviorInsights(  
 new Date('2024-01-01'),  
 new Date('2024-03-31'),  
 'premium-subscribers'  
);  
  
// Analyze market trends  
const trendAnalysis = await analyticsAgent.instance.getMarketTrendAnalysis(  
 'month',  
 'floor-tiles'  
);  
  
// Get decision support for a strategic question  
const decisionSupport = await analyticsAgent.instance.getDecisionSupport(  
 'Should we expand our ceramic or natural stone inventory?',  
 [  
 {  
 name: 'Expand ceramic inventory',  
 pros: ['Lower cost', 'Higher margin', 'More popular in searches'],  
 cons: ['Market saturation', 'Price competition']  
 },  
 {  
 name: 'Expand natural stone inventory',  
 pros: ['Premium segment', 'Less competition', 'Higher ticket value'],  
 cons: ['Higher inventory cost', 'Slower turnover']  
 }  
 ],  
 [  
 { name: 'Profit potential', weight: 0.4 },  
 { name: 'Market demand', weight: 0.3 },  
 { name: 'Operational complexity', weight: 0.2 },  
 { name: 'Brand alignment', weight: 0.1 }  
 ]  
);  
  
// Process natural language analytics query  
const queryResult = await analyticsAgent.instance.processAnalyticsQuery(  
 'Which product categories showed the highest growth in the last quarter?'  
);

#### Frontend Integration

The Analytics Agent is available through the AgentDashboard component:

import React from 'react';  
import { AgentDashboard } from '../components/agents/AgentDashboard';  
  
const AgentsPage: React.FC = () => {  
 return (  
 <div>  
 <h1>KAI Intelligent Assistants</h1>  
 <AgentDashboard />  
 </div>  
 );  
};  
  
export default AgentsPage;

import React from 'react';  
import { AgentDashboard } from '../components/agents/AgentDashboard';  
  
const AgentsPage: React.FC = () => {  
 return (  
 <div>  
 <h1>KAI Intelligent Assistants</h1>  
 <AgentDashboard />  
 </div>  
 );  
};  
  
export default AgentsPage;

Users can access the Analytics Agent through the "Analytics Expert" tab in the dashboard, which provides:

* Natural language query interface
* Interactive visualizations
* Example questions and prompts
* Tabs for different analytics functions

### Event Processing

The Analytics Agent can process various system events:

// Process a usage spike event  
await analyticsAgent.instance.processEvent('usage\_spike', {  
 timestamp: new Date(),  
 metric: 'search\_volume',  
 baseline: 150,  
 actual: 450,  
 duration: '2h'  
});  
  
// Process a search pattern change event  
await analyticsAgent.instance.processEvent('search\_pattern\_change', {  
 oldTopQueries: ['white marble', 'ceramic tile', 'porcelain'],  
 newTopQueries: ['sustainable materials', 'recycled tile', 'eco-friendly'],  
 changeVelocity: 'rapid',  
 detectedAt: new Date()  
});  
  
// Process a new material trend event  
await analyticsAgent.instance.processEvent('new\_material\_trend', {  
 trendingMaterial: 'recycled glass mosaic',  
 growthRate: 215, // percentage  
 searchVolume: 850,  
 relatedQueries: ['eco-friendly bathroom', 'sustainable kitchen']  
});

// Process a usage spike event  
await analyticsAgent.instance.processEvent('usage\_spike', {  
 timestamp: new Date(),  
 metric: 'search\_volume',  
 baseline: 150,  
 actual: 450,  
 duration: '2h'  
});  
  
// Process a search pattern change event  
await analyticsAgent.instance.processEvent('search\_pattern\_change', {  
 oldTopQueries: ['white marble', 'ceramic tile', 'porcelain'],  
 newTopQueries: ['sustainable materials', 'recycled tile', 'eco-friendly'],  
 changeVelocity: 'rapid',  
 detectedAt: new Date()  
});  
  
// Process a new material trend event  
await analyticsAgent.instance.processEvent('new\_material\_trend', {  
 trendingMaterial: 'recycled glass mosaic',  
 growthRate: 215, // percentage  
 searchVolume: 850,  
 relatedQueries: ['eco-friendly bathroom', 'sustainable kitchen']  
});

### Advanced Configuration

#### Custom Analytics Tools

Create custom analytics tools for specialized functionality:

import { Tool } from 'crewai';  
  
// Create a custom predictive analytics tool  
const createPredictiveAnalyticsTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'predictive\_analytics',  
 description: 'Predict future trends based on historical data',  
 func: async (args) => {  
 const { metric, timeframe, parameters } = JSON.parse(args);  
  
 // Use the predictive analytics service  
 let prediction;  
  
 if (metric === 'time\_series') {  
 // Generate time-series forecast  
 prediction = await predictiveAnalyticsService.generateTimeSeriesForecast({  
 eventType: parameters.eventType,  
 resourceType: parameters.resourceType,  
 startDate: new Date(parameters.startDate),  
 endDate: new Date(parameters.endDate),  
 forecastPeriods: parameters.forecastPeriods || 7,  
 interval: parameters.interval || 'day'  
 });  
 } else if (metric === 'anomalies') {  
 // Detect anomalies  
 prediction = await predictiveAnalyticsService.detectAnomalies({  
 eventType: parameters.eventType,  
 resourceType: parameters.resourceType,  
 startDate: new Date(parameters.startDate),  
 endDate: new Date(parameters.endDate),  
 interval: parameters.interval || 'day',  
 threshold: parameters.threshold || 2.0  
 });  
 } else if (metric === 'user\_behavior') {  
 // Predict user behavior  
 prediction = await predictiveAnalyticsService.predictUserBehavior({  
 userId: parameters.userId,  
 predictionType: parameters.predictionType || 'next\_action',  
 lookbackDays: parameters.lookbackDays || 30,  
 includeUserProfile: parameters.includeUserProfile !== false  
 });  
 }  
  
 return JSON.stringify(prediction);  
 }  
 });  
};  
  
// Add it to the agent  
const analyticsAgent = await createAgent({  
 id: 'advanced-analytics-1',  
 type: AgentType.ANALYTICS,  
 additionalTools: [await createPredictiveAnalyticsTool()]  
});

import { Tool } from 'crewai';  
  
// Create a custom predictive analytics tool  
const createPredictiveAnalyticsTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'predictive\_analytics',  
 description: 'Predict future trends based on historical data',  
 func: async (args) => {  
 const { metric, timeframe, parameters } = JSON.parse(args);  
  
 // Use the predictive analytics service  
 let prediction;  
  
 if (metric === 'time\_series') {  
 // Generate time-series forecast  
 prediction = await predictiveAnalyticsService.generateTimeSeriesForecast({  
 eventType: parameters.eventType,  
 resourceType: parameters.resourceType,  
 startDate: new Date(parameters.startDate),  
 endDate: new Date(parameters.endDate),  
 forecastPeriods: parameters.forecastPeriods || 7,  
 interval: parameters.interval || 'day'  
 });  
 } else if (metric === 'anomalies') {  
 // Detect anomalies  
 prediction = await predictiveAnalyticsService.detectAnomalies({  
 eventType: parameters.eventType,  
 resourceType: parameters.resourceType,  
 startDate: new Date(parameters.startDate),  
 endDate: new Date(parameters.endDate),  
 interval: parameters.interval || 'day',  
 threshold: parameters.threshold || 2.0  
 });  
 } else if (metric === 'user\_behavior') {  
 // Predict user behavior  
 prediction = await predictiveAnalyticsService.predictUserBehavior({  
 userId: parameters.userId,  
 predictionType: parameters.predictionType || 'next\_action',  
 lookbackDays: parameters.lookbackDays || 30,  
 includeUserProfile: parameters.includeUserProfile !== false  
 });  
 }  
  
 return JSON.stringify(prediction);  
 }  
 });  
};  
  
// Add it to the agent  
const analyticsAgent = await createAgent({  
 id: 'advanced-analytics-1',  
 type: AgentType.ANALYTICS,  
 additionalTools: [await createPredictiveAnalyticsTool()]  
});

#### Integration with External Analytics Systems

Connect the Analytics Agent to external analytics platforms:

import { createAnalyticsService } from '../services/analyticsService';  
  
// Create a custom analytics service with external system connection  
const externalAnalyticsService = createAnalyticsService({  
 baseURL: 'https://external-analytics-system.example.com/api',  
 apiKey: process.env.EXTERNAL\_ANALYTICS\_API\_KEY,  
 timeout: 30000  
});  
  
// Create tools using the external service  
const externalQueryTool = await createAnalyticsQueryTool(externalAnalyticsService);  
  
// Add it to the agent  
const analyticsAgent = await createAgent({  
 id: 'multi-source-analytics-1',  
 type: AgentType.ANALYTICS,  
 additionalTools: [externalQueryTool]  
});

import { createAnalyticsService } from '../services/analyticsService';  
  
// Create a custom analytics service with external system connection  
const externalAnalyticsService = createAnalyticsService({  
 baseURL: 'https://external-analytics-system.example.com/api',  
 apiKey: process.env.EXTERNAL\_ANALYTICS\_API\_KEY,  
 timeout: 30000  
});  
  
// Create tools using the external service  
const externalQueryTool = await createAnalyticsQueryTool(externalAnalyticsService);  
  
// Add it to the agent  
const analyticsAgent = await createAgent({  
 id: 'multi-source-analytics-1',  
 type: AgentType.ANALYTICS,  
 additionalTools: [externalQueryTool]  
});

### Performance Considerations

#### Optimizing Response Times

1. Caching Strategy
2. Cache frequently requested analytics data
3. Implement TTL-based cache invalidation
4. Use materialized views for common queries
5. Query Optimization
6. Limit date ranges for historical queries
7. Use pagination for large result sets
8. Apply appropriate filters before processing
9. Parallel Processing
10. Execute independent analytics queries in parallel
11. Batch related requests when possible
12. Use Promise.all for concurrent operations

Use materialized views for common queries

Query Optimization

Apply appropriate filters before processing

Parallel Processing

### Security Considerations

1. Data Access Control
2. Enforce appropriate permissions for analytics data
3. Filter sensitive information from responses
4. Audit analytics queries for security compliance
5. PII Management
6. Anonymize personally identifiable information
7. Apply data minimization principles
8. Implement proper data retention policies
9. Agent Boundaries
10. Restrict the agent to analytics operations only
11. Validate inputs to prevent injection attacks
12. Sanitize outputs to prevent sensitive data leakage

Audit analytics queries for security compliance

PII Management

Implement proper data retention policies

Agent Boundaries

### Related Documentation

* Analytics System - Core analytics system architecture
* CrewAI Integration - Overall agent system architecture
* CrewAI Implementation - Implementation details
* Agent Installation - Setup instructions
* Client Integration - Frontend framework details

# Analytics System

Source: readme/analytics-system.md

---

## Analytics System

This document outlines the analytics system architecture, implementation details, and data storage considerations for tracking searches, Agent AI prompts, and API requests on our platform.

### Overview

The analytics system captures and stores user interaction data to provide insights into:

* Search patterns and most common search queries
* Agent AI usage and popular prompts
* API request volumes, patterns, and performance metrics
* Material view trends and popular materials

This data enables trend analysis, performance monitoring, feature optimization, and predictive analytics based on actual usage patterns.

### Data Captured

The analytics system collects the following data points:

* Event Type: The type of interaction (search, agent\_prompt, api\_request, material\_view)
* Timestamp: When the interaction occurred
* User ID: Which user performed the action (if authenticated)
* Resource Type: What resource was accessed (materials, collections, etc.)
* Query: The actual search query or agent prompt text
* Response Time: How long the operation took to complete (ms)
* Response Status: HTTP status code for API requests
* Additional Context: JSON field for variable additional data

### Implementation Architecture

The analytics system consists of these key components:

1. Analytics Middleware: An Express middleware that automatically captures all API requests
2. Analytics Service: Service methods to track specific events (searches, agent prompts, etc.)
3. Database Schema: Table structure and indexes for storing analytics data
4. Database Functions: PostgreSQL functions that process raw data into insights
5. Predictive Analytics: Advanced capabilities for forecasting, anomaly detection, and user behavior prediction
6. Real-Time Analytics: WebSocket-based system for processing and delivering analytics events in real-time
7. Admin Dashboard: UI for visualizing analytics data, trends, and predictions

### Data Storage Options

#### Option 1: Supabase PostgreSQL (Current Implementation)

Our current implementation uses Supabase's PostgreSQL database:

##### Advantages

* Seamless Integration: Works within our existing Supabase infrastructure
* Simplicity: Single database system for application and analytics data
* Real-time Access: Low latency for dashboard visualizations
* SQL Power: Full PostgreSQL capabilities for complex queries
* Data Integrity: ACID compliance ensures reliable analytics
* Implementation Speed: Faster to implement with existing Supabase knowledge

##### Limitations

* Scaling Challenges: May struggle with very high volumes (billions of records)
* Cost Structure: Storage costs increase linearly with data volume
* Mixed Workload: Analytics queries compete with operational queries

##### Implementation Details

The Supabase implementation uses:

* A dedicated analytics\_events table with appropriate indexes
* PostgreSQL functions for aggregation and trend analysis
* Row-level security policies to protect sensitive data

analytics\_events

Database schema details are defined in the migration file:  
packages/server/src/services/supabase/migrations/005\_analytics\_system.sql

packages/server/src/services/supabase/migrations/005\_analytics\_system.sql

#### Option 2: BigQuery Alternative

For larger-scale deployments, Google BigQuery provides an alternative:

##### Advantages

* Massive Scale: Handles petabytes of data with ease
* Query Performance: Superior performance for complex analytical queries
* Separation of Concerns: Analytics workload isolated from operational database
* Cost Efficiency: Pay-per-query pricing can be cost-effective for intermittent analysis
* Advanced Features: Machine learning integrations and advanced analytics
* No Storage Management: Serverless architecture requires no capacity planning

##### Limitations

* Implementation Complexity: Requires ETL pipelines to load data from application
* Higher Latency: Not ideal for real-time dashboard updates
* Learning Curve: Team needs to learn BigQuery-specific SQL dialect
* Additional Service: Adds another external dependency

##### Implementation Approach

To implement BigQuery storage:

1. Create a BigQuery dataset and table schema matching our analytics structure
2. Implement a data export service that regularly:
3. Queries recent analytics events from Supabase
4. Transforms data if needed
5. Loads data into BigQuery tables
6. Update admin dashboard to query BigQuery for historical analytics
7. Optionally implement a hybrid approach:
8. Recent data (30-90 days) stays in Supabase for real-time access
9. Historical data moves to BigQuery for long-term storage and analysis

### Hybrid Approach Considerations

For systems with high analytics volume, consider a hybrid approach:

1. Real-time Tier: Keep recent data (last 30-90 days) in Supabase
2. Powers real-time dashboards with low latency
3. Handles rapid writes efficiently
4. Enables immediate analysis of current activity
5. Historical Tier: Archive older data to BigQuery
6. Stores historical data cost-effectively
7. Enables complex analytical queries over large datasets
8. Removes pressure from operational database

Enables immediate analysis of current activity

Historical Tier: Archive older data to BigQuery

This approach provides the benefits of both systems while minimizing their limitations.

### Setup Instructions

#### Supabase Setup

1. Run the migration script:  
    bash  
    cd packages/server  
    npm run migration:run
2. The migration creates:
3. The analytics\_events table
4. Required indexes
5. PostgreSQL functions for aggregation
6. Row-level security policies
7. The analytics service will use the new table automatically

Run the migration script:  
 bash  
 cd packages/server  
 npm run migration:run

bash  
 cd packages/server  
 npm run migration:run

The migration creates:

Row-level security policies

The analytics service will use the new table automatically

#### BigQuery Setup (Optional)

1. Create a BigQuery project and dataset
2. Create tables matching our analytics schema
3. Set up the data export service to run on schedule
4. Update dashboard services to query BigQuery for historical data

### Using the Analytics System

The system automatically tracks all API requests through the middleware. For additional manual tracking:

// Track a search operation  
await analyticsService.trackSearch(  
 query, // The search query  
 resourceType, // Type of resource being searched  
 userId, // Optional user ID  
 parameters, // Optional additional parameters  
 responseTimeMs, // Optional response time  
 responseStatus // Optional HTTP status code  
);  
  
// Track an agent prompt  
await analyticsService.trackAgentPrompt(  
 prompt, // The agent prompt  
 agentType, // Type of agent  
 userId, // Optional user ID  
 sessionId, // Optional session ID  
 parameters // Optional additional parameters  
);  
  
// Track a material view  
await analyticsService.trackMaterialView(  
 materialId, // ID of the material being viewed  
 userId, // Optional user ID  
 parameters // Optional additional parameters  
);

// Track a search operation  
await analyticsService.trackSearch(  
 query, // The search query  
 resourceType, // Type of resource being searched  
 userId, // Optional user ID  
 parameters, // Optional additional parameters  
 responseTimeMs, // Optional response time  
 responseStatus // Optional HTTP status code  
);  
  
// Track an agent prompt  
await analyticsService.trackAgentPrompt(  
 prompt, // The agent prompt  
 agentType, // Type of agent  
 userId, // Optional user ID  
 sessionId, // Optional session ID  
 parameters // Optional additional parameters  
);  
  
// Track a material view  
await analyticsService.trackMaterialView(  
 materialId, // ID of the material being viewed  
 userId, // Optional user ID  
 parameters // Optional additional parameters  
);

### Accessing Analytics Data

Analytics data can be accessed through:

1. Admin Dashboard: Navigate to /admin/analytics for visualizations
2. Analytics API: Endpoints for programmatic access:
3. GET /api/admin/analytics/events - List raw events
4. GET /api/admin/analytics/stats - Get summary statistics
5. GET /api/admin/analytics/trends - Get time-based trends
6. GET /api/admin/analytics/top-searches - Get most common searches
7. GET /api/admin/analytics/top-prompts - Get most common agent prompts
8. GET /api/admin/analytics/top-materials - Get most viewed materials
9. POST /api/analytics/predictive/forecast - Generate time-series forecasts
10. POST /api/analytics/predictive/anomalies - Detect anomalies in analytics data
11. POST /api/analytics/predictive/user-behavior - Predict user behavior patterns
12. POST /api/analytics/real-time/event - Track real-time analytics events
13. GET /api/analytics/real-time/events - Get recent real-time events

/admin/analytics

GET /api/admin/analytics/events

GET /api/admin/analytics/stats

GET /api/admin/analytics/trends

GET /api/admin/analytics/top-searches

GET /api/admin/analytics/top-prompts

GET /api/admin/analytics/top-materials

POST /api/analytics/predictive/forecast

POST /api/analytics/predictive/anomalies

POST /api/analytics/predictive/user-behavior

POST /api/analytics/real-time/event

GET /api/analytics/real-time/events

### Best Practices

1. Data Retention: Define a clear data retention policy:
2. How long to keep detailed events
3. When to aggregate and discard raw data
4. Legal compliance considerations (GDPR, etc.)
5. Privacy Considerations:
6. Store only necessary data
7. Anonymize sensitive information
8. Ensure proper authorization for analytics access
9. Performance Optimization:
10. Use time-based partitioning for large datasets
11. Create materialized views for common queries
12. Consider a read replica for analytics queries
13. Monitoring:
14. Set up alerts for unusual patterns
15. Monitor storage growth and query performance
16. Periodically audit access to analytics data

Legal compliance considerations (GDPR, etc.)

Privacy Considerations:

Ensure proper authorization for analytics access

Performance Optimization:

Consider a read replica for analytics queries

Monitoring:

### Making the Decision: Supabase vs BigQuery

The choice between Supabase PostgreSQL and BigQuery depends on several factors:

#### Choose Supabase When:

* Expected analytics volume is under 100GB or ~50 million events
* Real-time analytics are critical
* Simplicity and fast implementation are priorities
* Team is already familiar with PostgreSQL
* Cost predictability is important

#### Choose BigQuery When:

* Expected analytics volume exceeds 100GB or ~50 million events
* Complex analytical queries are frequent
* Real-time access is less important than historical analysis
* Separate analytical workloads from operational database
* Team is comfortable with data pipeline management

#### Choose Hybrid Approach When:

* Both real-time and historical analytics are important
* Data volume is growing rapidly
* Different teams have different analysis needs
* Cost optimization for large datasets is a priority

### Predictive Analytics Features

The analytics system includes advanced predictive capabilities that leverage historical data to provide forward-looking insights:

#### 1. Time-Series Forecasting

The time-series forecasting feature predicts future trends based on historical analytics data:

* Forecasting Models: Uses statistical models to predict future values
* Configurable Parameters: Adjustable forecast periods and intervals
* Event Type Filtering: Generate forecasts for specific event types
* Resource Type Filtering: Focus on particular resource categories
* MCP Integration: Leverages the Model Context Protocol for advanced forecasting
* Visualization: Interactive charts in the admin dashboard

#### 2. Anomaly Detection

The anomaly detection system identifies unusual patterns in analytics data:

* Statistical Analysis: Uses standard deviation-based detection
* Severity Classification: Categorizes anomalies as low, medium, or high
* Threshold Configuration: Adjustable sensitivity settings
* Visual Indicators: Highlights anomalies in time-series charts
* Alert Potential: Foundation for anomaly-based alerting
* MCP Integration: Advanced detection through the Model Context Protocol

#### 3. User Behavior Prediction

The user behavior prediction system anticipates user actions and preferences:

* Next Action Prediction: Forecasts likely user actions
* Churn Risk Assessment: Identifies users at risk of disengagement
* Engagement Scoring: Predicts user engagement levels
* Content Preferences: Recommends content based on predicted interests
* User Insights: Provides activity level, interests, and usage patterns
* MCP Integration: Sophisticated prediction through the Model Context Protocol

#### 4. Real-Time Analytics Processing

The real-time analytics system processes events as they happen:

* WebSocket Delivery: Instant updates via WebSocket connections
* Event Buffering: Efficient processing of event batches
* Subscription Model: Clients can subscribe to specific event types
* Filtering Capabilities: Target specific event types or resources
* MCP Integration: Enhanced processing through the Model Context Protocol
* Low Latency: Minimal delay between event occurrence and processing

### Conclusion

The analytics system provides valuable insights into user behavior while maintaining system performance. The initial Supabase PostgreSQL implementation offers a good balance of simplicity and functionality, with options to migrate to BigQuery or implement a hybrid approach as data volume grows. The addition of predictive analytics and real-time processing capabilities enhances the system's value by providing forward-looking insights and immediate feedback on user activity.

# Api Endpoints Reference

Source: readme/api-endpoints-reference.md

---

## API Endpoints Reference

### Overview

This document provides a comprehensive reference of all API endpoints in the Kai platform, including their access control settings and rate limit configurations. This serves as both implementation documentation and a reference for administrators.

Important Note: All endpoint access settings are fully configurable through the Network Access Control panel in the admin dashboard. No endpoints have hardcoded access restrictions.

### Access Control Configuration

Each API endpoint can be configured with the following access settings:

Table content:

Setting | Description

Internal Access | When enabled, the endpoint can be accessed from defined internal networks

External Access | When enabled, the endpoint can be accessed from external networks

Rate Limit | Per-endpoint rate limits based on source network

### Rate Limit Configuration

Rate limits can be configured for different networks:

* Default Rate Limit: Applied to all requests from undefined networks (default: 30 req/min)
* Custom Rate Limits: Specific limits for particular IP addresses or CIDR ranges
* Endpoint Category Multipliers: Different endpoint categories have different rate limit multipliers

### Authentication APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/auth/login | POST | User login | Internal & External | 20 req/min

/api/auth/register | POST | User registration | Internal & External | 10 req/min

/api/auth/refresh-token | POST | Refresh authentication token | Internal & External | 60 req/min

/api/auth/forgot-password | POST | Request password reset | Internal & External | 5 req/min

/api/auth/reset-password | PUT | Reset password with token | Internal & External | 5 req/min

/api/auth/verify-email | POST | Verify user email address | Internal & External | 5 req/min

/api/auth/change-password | PUT | Change user password | Internal & External | 5 req/min

/api/auth/login

/api/auth/register

/api/auth/refresh-token

/api/auth/forgot-password

/api/auth/reset-password

/api/auth/verify-email

/api/auth/change-password

### User Management APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/users/profile | GET | Get user profile | Internal & External | 30 req/min

/api/users/profile | PUT | Update user profile | Internal & External | 10 req/min

/api/users/preferences | GET | Get user preferences | Internal & External | 30 req/min

/api/users/preferences | PUT | Update user preferences | Internal & External | 10 req/min

/api/users/:userId | GET | Get user by ID | Internal Only | 30 req/min

/api/users | GET | List all users | Internal Only | 20 req/min

/api/users/history | GET | Get user activity history | Internal & External | 20 req/min

/api/users/profile

/api/users/profile

/api/users/preferences

/api/users/preferences

/api/users/:userId

/api/users

/api/users/history

### Material APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/materials | GET | List materials | Internal & External | 60 req/min

/api/materials/:id | GET | Get material by ID | Internal & External | 60 req/min

/api/materials | POST | Create new material | Internal & External | 30 req/min

/api/materials/:id | PUT | Update material | Internal & External | 20 req/min

/api/materials/:id | DELETE | Delete material | Internal Only | 10 req/min

/api/materials/search | POST | Search materials | Internal & External | 60 req/min

/api/materials/batch-import | POST | Import multiple materials | Internal Only | 5 req/min

/api/materials/metadata | GET | Get material metadata fields | Internal & External | 30 req/min

/api/materials

/api/materials/:id

/api/materials

/api/materials/:id

/api/materials/:id

/api/materials/search

/api/materials/batch-import

/api/materials/metadata

### Recognition APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/recognition | POST | Recognize material | Internal & External | 20 req/min

/api/recognition/batch | POST | Batch recognition | Internal & External | 5 req/min

/api/recognition/history | GET | Get recognition history | Internal & External | 30 req/min

/api/recognition/feedback | POST | Submit recognition feedback | Internal & External | 20 req/min

/api/recognition/confidence | GET | Get recognition confidence scores | Internal Only | 30 req/min

/api/recognition/retrain | POST | Start recognition model retraining | Internal Only | 1 req/10min

/api/recognition

/api/recognition/batch

/api/recognition/history

/api/recognition/feedback

/api/recognition/confidence

/api/recognition/retrain

### Search APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/search | GET | Unified search | Internal & External | 60 req/min

/api/search/vector | POST | Vector similarity search | Internal & External | 30 req/min

/api/search/hybrid | POST | Hybrid (text + vector) search | Internal & External | 30 req/min

/api/search/autocomplete | GET | Search autocomplete suggestions | Internal & External | 100 req/min

/api/search/config | GET | Get search configuration | Internal & External | 10 req/min

/api/search/config | PUT | Update search configuration | Internal Only | 5 req/min

/api/search/reindex | POST | Rebuild search index | Internal Only | 1 req/hour

/api/search

/api/search/vector

/api/search/hybrid

/api/search/autocomplete

/api/search/config

/api/search/config

/api/search/reindex

### Analytics APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/analytics/events | POST | Track analytics event | Internal & External | 100 req/min

/api/analytics/events | GET | Get analytics events | Internal Only | 30 req/min

/api/analytics/trends | GET | Get analytics trends | Internal Only | 20 req/min

/api/analytics/stats | GET | Get analytics statistics | Internal Only | 20 req/min

/api/analytics/data | DELETE | Clear analytics data | Internal Only | 1 req/day

/api/analytics/dashboard | GET | Get analytics dashboard data | Internal Only | 20 req/min

/api/analytics/export | POST | Export analytics data | Internal Only | 3 req/hour

/api/analytics/events

/api/analytics/events

/api/analytics/trends

/api/analytics/stats

/api/analytics/data

/api/analytics/dashboard

/api/analytics/export

### ML Service APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/ml/models | GET | List ML models | Internal Only | 10 req/min

/api/ml/models/:id | GET | Get ML model details | Internal Only | 10 req/min

/api/ml/training/start | POST | Start model training | Internal Only | 3 req/hour

/api/ml/training/:jobId/status | GET | Get training job status | Internal Only | 30 req/min

/api/ml/training/:jobId/stop | POST | Stop training job | Internal Only | 5 req/min

/api/ml/inference | POST | Run model inference | Internal & External | 30 req/min

/api/ml/embeddings | POST | Generate embeddings | Internal & External | 30 req/min

/api/ml/models

/api/ml/models/:id

/api/ml/training/start

/api/ml/training/:jobId/status

/api/ml/training/:jobId/stop

/api/ml/inference

/api/ml/embeddings

### Dataset APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/admin/datasets | GET | List all datasets | Internal Only | 30 req/min

/api/admin/datasets/:id | GET | Get dataset details | Internal Only | 30 req/min

/api/admin/datasets/upload/zip | POST | Upload ZIP dataset | Internal Only | 5 req/min

/api/admin/datasets/upload/csv | POST | Upload CSV dataset | Internal Only | 5 req/min

/api/admin/datasets/import/premade | POST | Import premade dataset | Internal Only | 3 req/hour

/api/admin/datasets/:id/split | POST | Split dataset into train/validation/test sets | Internal Only | 10 req/min

/api/admin/datasets/:id/train | POST | Start training job for dataset | Internal Only | 3 req/hour

/api/admin/datasets/:id/quality | GET | Get dataset quality metrics | Internal Only | 20 req/min

/api/admin/datasets/:id | DELETE | Delete dataset | Internal Only | 5 req/min

/api/admin/datasets/:id/classes | GET | List dataset classes | Internal Only | 30 req/min

/api/admin/datasets/:id/images | GET | List dataset images | Internal Only | 30 req/min

/api/admin/training/:jobId/status | GET | Get training job status | Internal Only | 30 req/min

/api/admin/training/:jobId/stop | POST | Stop training job | Internal Only | 5 req/min

/api/admin/training/:jobId/metrics | GET | Get training job metrics | Internal Only | 30 req/min

/api/admin/datasets

/api/admin/datasets/:id

/api/admin/datasets/upload/zip

/api/admin/datasets/upload/csv

/api/admin/datasets/import/premade

/api/admin/datasets/:id/split

/api/admin/datasets/:id/train

/api/admin/datasets/:id/quality

/api/admin/datasets/:id

/api/admin/datasets/:id/classes

/api/admin/datasets/:id/images

/api/admin/training/:jobId/status

/api/admin/training/:jobId/stop

/api/admin/training/:jobId/metrics

### PDF Processing APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/pdf/extract | POST | Extract text from PDF | Internal & External | 10 req/min

/api/pdf/process | POST | Process PDF document | Internal & External | 5 req/min

/api/pdf/analyze | POST | Analyze PDF structure | Internal & External | 5 req/min

/api/pdf/ocr | POST | Run OCR on PDF | Internal & External | 3 req/min

/api/pdf/queue | GET | Get PDF processing queue status | Internal Only | 30 req/min

/api/pdf/jobs/:jobId | GET | Get PDF job status | Internal & External | 30 req/min

/api/pdf/extract

/api/pdf/process

/api/pdf/analyze

/api/pdf/ocr

/api/pdf/queue

/api/pdf/jobs/:jobId

### 3D Designer APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/3d-designer/scene | POST | Create new 3D scene | Internal & External | 10 req/min

/api/3d-designer/scene/:id | GET | Get 3D scene | Internal & External | 30 req/min

/api/3d-designer/scene/:id | PUT | Update 3D scene | Internal & External | 10 req/min

/api/3d-designer/scene/:id | DELETE | Delete 3D scene | Internal & External | 5 req/min

/api/3d-designer/render | POST | Render 3D scene | Internal & External | 5 req/min

/api/3d-designer/export | POST | Export 3D scene | Internal & External | 5 req/min

/api/3d-designer/scene

/api/3d-designer/scene/:id

/api/3d-designer/scene/:id

/api/3d-designer/scene/:id

/api/3d-designer/render

/api/3d-designer/export

### Agent APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/agents/chat | POST | Send message to agent | Internal & External | 30 req/min

/api/agents/sessions | GET | List agent sessions | Internal & External | 30 req/min

/api/agents/sessions/:id | GET | Get agent session | Internal & External | 30 req/min

/api/agents/sessions/:id | DELETE | Delete agent session | Internal & External | 10 req/min

/api/agents/feedback | POST | Submit agent feedback | Internal & External | 10 req/min

/api/agents/config | GET | Get agent configuration | Internal Only | 10 req/min

/api/agents/config | PUT | Update agent configuration | Internal Only | 5 req/min

/api/agents/chat

/api/agents/sessions

/api/agents/sessions/:id

/api/agents/sessions/:id

/api/agents/feedback

/api/agents/config

/api/agents/config

### Admin APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/admin/users | GET | List all users (admin) | Internal Only | 20 req/min

/api/admin/users/:id | PUT | Update user (admin) | Internal Only | 10 req/min

/api/admin/users/:id | DELETE | Delete user (admin) | Internal Only | 5 req/min

/api/admin/settings | GET | Get system settings | Internal Only | 10 req/min

/api/admin/settings | PUT | Update system settings | Internal Only | 5 req/min

/api/admin/metadata-fields | GET | Get metadata fields | Internal Only | 10 req/min

/api/admin/metadata-fields | POST | Create metadata field | Internal Only | 5 req/min

/api/admin/metadata-fields/:id | PUT | Update metadata field | Internal Only | 5 req/min

/api/admin/metadata-fields/:id | DELETE | Delete metadata field | Internal Only | 5 req/min

/api/admin/jobs | GET | List background jobs | Internal Only | 20 req/min

/api/admin/jobs/:id | GET | Get job details | Internal Only | 30 req/min

/api/admin/jobs/:id | DELETE | Cancel job | Internal Only | 10 req/min

/api/admin/users

/api/admin/users/:id

/api/admin/users/:id

/api/admin/settings

/api/admin/settings

/api/admin/metadata-fields

/api/admin/metadata-fields

/api/admin/metadata-fields/:id

/api/admin/metadata-fields/:id

/api/admin/jobs

/api/admin/jobs/:id

/api/admin/jobs/:id

### Network Access Control APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/admin/network/internal-networks | GET | List internal networks | Internal Only | 10 req/min

/api/admin/network/internal-networks | POST | Add internal network | Internal Only | 5 req/min

/api/admin/network/internal-networks/:id | DELETE | Remove internal network | Internal Only | 5 req/min

/api/admin/network/endpoints | GET | List endpoint permissions | Internal Only | 10 req/min

/api/admin/network/endpoints/:id | PUT | Update endpoint permissions | Internal Only | 5 req/min

/api/admin/network/rate-limits | GET | Get rate limit settings | Internal Only | 10 req/min

/api/admin/network/rate-limits | PUT | Update default rate limit | Internal Only | 5 req/min

/api/admin/network/rate-limits/custom | GET | List custom rate limits | Internal Only | 10 req/min

/api/admin/network/rate-limits/custom | POST | Add custom rate limit | Internal Only | 5 req/min

/api/admin/network/rate-limits/custom/:id | DELETE | Remove custom rate limit | Internal Only | 5 req/min

/api/admin/network/internal-networks

/api/admin/network/internal-networks

/api/admin/network/internal-networks/:id

/api/admin/network/endpoints

/api/admin/network/endpoints/:id

/api/admin/network/rate-limits

/api/admin/network/rate-limits

/api/admin/network/rate-limits/custom

/api/admin/network/rate-limits/custom

/api/admin/network/rate-limits/custom/:id

### Subscription APIs

Table content:

Endpoint | Method | Description | Default Access | Default Rate Limit

/api/subscription/plans | GET | List subscription plans | Internal & External | 30 req/min

/api/subscription/subscribe | POST | Subscribe to plan | Internal & External | 10 req/min

/api/subscription/status | GET | Get subscription status | Internal & External | 30 req/min

/api/subscription/cancel | POST | Cancel subscription | Internal & External | 10 req/min

/api/subscription/upgrade | POST | Upgrade subscription | Internal & External | 10 req/min

/api/subscription/invoices | GET | List subscription invoices | Internal & External | 20 req/min

/api/admin/subscription/plans | GET | List all plans (admin) | Internal Only | 10 req/min

/api/admin/subscription/plans | POST | Create plan (admin) | Internal Only | 5 req/min

/api/admin/subscription/plans/:id | PUT | Update plan (admin) | Internal Only | 5 req/min

/api/admin/subscription/plans/:id | DELETE | Delete plan (admin) | Internal Only | 5 req/min

/api/subscription/plans

/api/subscription/subscribe

/api/subscription/status

/api/subscription/cancel

/api/subscription/upgrade

/api/subscription/invoices

/api/admin/subscription/plans

/api/admin/subscription/plans

/api/admin/subscription/plans/:id

/api/admin/subscription/plans/:id

### How to Configure Access Settings

Access settings for each endpoint can be configured through the admin panel under Settings → Network Access:

1. Navigate to the API Endpoint Access Control section
2. Use the search box to find specific endpoints
3. Enable or disable internal/external access using the checkboxes
4. Save your changes

### How to Configure Rate Limits

Rate limits can be configured:

1. Default Rate Limit: Set the default requests per minute allowed from undefined networks
2. Custom Rate Limits: Define specific rate limits for particular IP addresses or CIDR ranges
3. Per-endpoint limits: Contact development team if specific endpoints need custom rate limits

### Implementation Notes

All endpoints in this document have the default recommended access settings that can be modified through the admin interface. The system provides a full audit trail of access changes for security monitoring purposes.

# Api Reference

Source: readme/api-reference.md

---

## Kai API Reference

### Overview

This document provides comprehensive documentation for the Kai API, including all available endpoints, authentication methods, access control mechanisms, rate limiting, and usage examples.

The Kai API follows RESTful principles and uses JSON for request and response bodies. All endpoints are prefixed with /api.

/api

### Table of Contents

* Authentication
* Authorization
* Role-Based Access Control
* Network-Based Access Control
* Error Handling
* Rate Limiting
* API Endpoints
* Authentication
* Users
* Materials
* Catalogs
* Recognition
* Search
* MoodBoard
* Analytics
* Admin
* AI
* Agents
* 3D Designer
* Subscription
* Network Access Control Implementation
* Scripts and Utilities
* Best Practices
* SDK Examples

### Authentication

The Kai API uses JWT (JSON Web Tokens) for authentication. To access protected endpoints, you must include an authorization token in the request headers.

#### Getting a Token

POST /api/auth/login  
Content-Type: application/json  
  
{  
 "email": "user@example.com",  
 "password": "yourpassword"  
}

POST /api/auth/login  
Content-Type: application/json  
  
{  
 "email": "user@example.com",  
 "password": "yourpassword"  
}

#### Using a Token

Include the token in the Authorization header for subsequent requests:

GET /api/users/profile  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

GET /api/users/profile  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

### Authorization

The Kai API implements multiple layers of authorization to secure endpoints.

#### Role-Based Access Control

Users have roles (e.g., admin, user) that determine their access level. Some endpoints are restricted to specific roles.

admin

user

Example:  
- Admin routes (/api/admin/\*) require the admin role  
- User management requires appropriate permissions

/api/admin/\*

admin

#### Network-Based Access Control

Some sensitive operations are restricted to internal network access only. This ensures that critical endpoints cannot be accessed from external networks, adding an additional security layer.

IMPORTANT: No API endpoints have hardcoded access restrictions. All endpoint access controls are configured through the admin panel and stored in the database.

##### Network Access Types

API endpoints can have one of the following access types:

1. ANY: Accessible from both internal and external networks (default for most endpoints)
2. INTERNAL\_ONLY: Only accessible from internal networks
3. EXTERNAL\_ALLOWED: Explicitly allows external access (same as ANY)

These access types are defined in the NetworkAccessType enum in packages/server/src/utils/network.ts.

NetworkAccessType

packages/server/src/utils/network.ts

##### Internal Networks

Internal networks are defined by CIDR ranges (e.g., 10.0.0.0/8, 192.168.0.0/16). Requests originating from these networks can access endpoints marked as "internal-only".

10.0.0.0/8

192.168.0.0/16

##### Access Verification

The system verifies network access as follows:

1. Determines the client's IP address (considering proxy headers if configured)
2. Checks if the IP falls within any defined internal network ranges
3. Verifies if the requested endpoint is allowed for the client's network location

Important Note: The "Default Access" column in endpoint tables shows the recommended access type for each endpoint, but these are not hardcoded restrictions. All access settings are fully configurable through the admin panel.

##### Admin Panel

The admin panel provides a user-friendly interface for managing API access control:

1. Internal Networks: Define CIDR ranges that should be considered internal networks
2. API Endpoint Access Control: Configure which endpoints can be accessed from internal or external networks
3. Rate Limits: Set default and custom rate limits for different networks

To access the admin panel, navigate to /admin/network-access in the application.

/admin/network-access

##### Testing Network Access

For development and testing purposes, you can:

1. Use the loopback address (127.0.0.1) which is always considered internal
2. Configure custom CIDR ranges for your development environment
3. Use the /api/admin/network-test endpoint (in development mode) to verify network detection

/api/admin/network-test

GET /api/admin/network-test  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

GET /api/admin/network-test  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Response:

{  
 "success": true,  
 "data": {  
 "ip": "192.168.1.100",  
 "isInternal": true,  
 "matchedNetwork": "192.168.0.0/16"  
 }  
}

{  
 "success": true,  
 "data": {  
 "ip": "192.168.1.100",  
 "isInternal": true,  
 "matchedNetwork": "192.168.0.0/16"  
 }  
}

### Error Handling

All API endpoints return consistent error responses with the following format:

{  
 "success": false,  
 "error": "Error message",  
 "statusCode": 400,  
 "details": {  
 // Additional error details if available  
 }  
}

{  
 "success": false,  
 "error": "Error message",  
 "statusCode": 400,  
 "details": {  
 // Additional error details if available  
 }  
}

Common HTTP status codes:

Table content:

Status Code | Description

400 | Bad Request - Invalid input data

401 | Unauthorized - Missing or invalid authentication

403 | Forbidden - Insufficient permissions or network restriction

404 | Not Found - Resource doesn't exist

429 | Too Many Requests - Rate limit exceeded

500 | Internal Server Error - Server-side issue

### Rate Limiting

The Kai platform implements a tiered rate limiting system to prevent abuse while allowing different levels of access for different clients.

#### Default Rate Limit

By default, most API endpoints are limited to 30 requests per minute. This is configurable through the admin panel.

#### Custom Rate Limits

Administrators can configure custom rate limits for specific IP addresses or CIDR ranges. For example:

* Internal networks (10.0.0.0/8): 300 requests per minute
* Office network (203.0.113.0/24): 100 requests per minute
* Specific partners (8.8.8.8): 50 requests per minute

#### Category-Specific Rate Limits

Different endpoint categories have different default rate limits:

Table content:

Category | Default Limit | Internal Multiplier | Notes

Standard API | 100 req/min | Configurable | General API endpoints

Authentication | 20 req/min | 2x | More strict to prevent brute force

ML Processing | 10 req/min | Configurable | Resource-intensive operations

Agent API | 30 req/min | Configurable | AI agent interactions

PDF Processing | 5 req/10 min | Configurable | Very resource-intensive

When a rate limit is exceeded, the API returns a 429 Too Many Requests response with a Retry-After header:

{  
 "error": "Too many requests",  
 "message": "Rate limit exceeded, please try again later",  
 "retryAfter": 60  
}

{  
 "error": "Too many requests",  
 "message": "Rate limit exceeded, please try again later",  
 "retryAfter": 60  
}

### API Endpoints

This section lists all available API endpoints, grouped by category. For each endpoint, we provide method, path, description, and default access recommendation.

#### Authentication Endpoints

Table content:

Method | Endpoint | Description | Default Access

POST | /api/auth/register | Register a new user | EXTERNAL\_ALLOWED

POST | /api/auth/login | Log in and get a token | EXTERNAL\_ALLOWED

POST | /api/auth/refresh | Refresh an auth token | EXTERNAL\_ALLOWED

POST | /api/auth/logout | Log out (invalidate token) | EXTERNAL\_ALLOWED

GET | /api/auth/me | Get current user info | EXTERNAL\_ALLOWED

##### Login Example

POST /api/auth/login  
Content-Type: application/json  
  
{  
 "email": "user@example.com",  
 "password": "yourpassword"  
}

POST /api/auth/login  
Content-Type: application/json  
  
{  
 "email": "user@example.com",  
 "password": "yourpassword"  
}

Response:

{  
 "success": true,  
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",  
 "user": {  
 "id": "123",  
 "email": "user@example.com",  
 "role": "user"  
 }  
}

{  
 "success": true,  
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",  
 "user": {  
 "id": "123",  
 "email": "user@example.com",  
 "role": "user"  
 }  
}

##### Register Example

POST /api/auth/register  
Content-Type: application/json  
  
{  
 "email": "newuser@example.com",  
 "password": "securepassword",  
 "name": "New User"  
}

POST /api/auth/register  
Content-Type: application/json  
  
{  
 "email": "newuser@example.com",  
 "password": "securepassword",  
 "name": "New User"  
}

Response:

{  
 "success": true,  
 "message": "User registered successfully",  
 "user": {  
 "id": "124",  
 "email": "newuser@example.com",  
 "role": "user"  
 }  
}

{  
 "success": true,  
 "message": "User registered successfully",  
 "user": {  
 "id": "124",  
 "email": "newuser@example.com",  
 "role": "user"  
 }  
}

#### User Endpoints

Table content:

Method | Endpoint | Description | Default Access

GET | /api/users/profile | Get current user profile | EXTERNAL\_ALLOWED

PUT | /api/users/profile | Update user profile | EXTERNAL\_ALLOWED

##### Get Current User Profile Example

GET /api/users/profile  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

GET /api/users/profile  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Response:

{  
 "success": true,  
 "data": {  
 "id": "123",  
 "email": "user@example.com",  
 "name": "User Name",  
 "role": "user",  
 "createdAt": "2023-01-01T00:00:00.000Z",  
 "subscription": {  
 "tier": "basic",  
 "expiresAt": "2024-01-01T00:00:00.000Z"  
 }  
 }  
}

{  
 "success": true,  
 "data": {  
 "id": "123",  
 "email": "user@example.com",  
 "name": "User Name",  
 "role": "user",  
 "createdAt": "2023-01-01T00:00:00.000Z",  
 "subscription": {  
 "tier": "basic",  
 "expiresAt": "2024-01-01T00:00:00.000Z"  
 }  
 }  
}

#### Material Endpoints

Table content:

Method | Endpoint | Description | Default Access

GET | /api/materials | Get all materials | EXTERNAL\_ALLOWED

GET | /api/materials/:id | Get material by ID | EXTERNAL\_ALLOWED

POST | /api/materials | Create a new material | EXTERNAL\_ALLOWED

PUT | /api/materials/:id | Update a material | EXTERNAL\_ALLOWED

DELETE | /api/materials/:id | Delete a material | EXTERNAL\_ALLOWED

GET | /api/materials/favorites | Get user's favorite materials | EXTERNAL\_ALLOWED

POST | /api/materials/:id/favorite | Add material to favorites | EXTERNAL\_ALLOWED

DELETE | /api/materials/:id/favorite | Remove material from favorites | EXTERNAL\_ALLOWED

##### Get All Materials Example

GET /api/materials?page=1&limit=10&category=tile  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

GET /api/materials?page=1&limit=10&category=tile  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Response:

{  
 "success": true,  
 "count": 120,  
 "page": 1,  
 "limit": 10,  
 "data": [  
 {  
 "id": "mat-123",  
 "name": "Marble Tile",  
 "category": "tile",  
 "imageUrl": "https://example.com/images/marble-tile.jpg",  
 "metadata": {  
 "color": "white",  
 "finish": "polished",  
 "size": "12x12"  
 }  
 },  
 // More materials...  
 ]  
}

{  
 "success": true,  
 "count": 120,  
 "page": 1,  
 "limit": 10,  
 "data": [  
 {  
 "id": "mat-123",  
 "name": "Marble Tile",  
 "category": "tile",  
 "imageUrl": "https://example.com/images/marble-tile.jpg",  
 "metadata": {  
 "color": "white",  
 "finish": "polished",  
 "size": "12x12"  
 }  
 },  
 // More materials...  
 ]  
}

#### Catalog Endpoints

These endpoints manage material catalogs, which are collections of related materials.

#### Recognition Endpoints

Table content:

Method | Endpoint | Description | Default Access

POST | /api/recognition | Recognize material from image | EXTERNAL\_ALLOWED

GET | /api/recognition/history | Get recognition history | EXTERNAL\_ALLOWED

POST | /api/recognition/feedback | Submit recognition feedback | EXTERNAL\_ALLOWED

##### Recognize Material Example

POST /api/recognition  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  
Content-Type: multipart/form-data

POST /api/recognition  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  
Content-Type: multipart/form-data

Form data:  
- image: Image file  
- modelType (optional): "standard", "enhanced", or "hybrid"  
- maxResults (optional): Number of results to return (default: 5)

image

modelType

maxResults

Response:

{  
 "success": true,  
 "data": {  
 "results": [  
 {  
 "materialId": "mat-123",  
 "name": "Marble Tile",  
 "confidence": 0.94,  
 "imageUrl": "https://example.com/images/marble-tile.jpg",  
 "metadata": {  
 "color": "white",  
 "finish": "polished"  
 }  
 },  
 // More results...  
 ],  
 "processingTime": 1.2  
 }  
}

{  
 "success": true,  
 "data": {  
 "results": [  
 {  
 "materialId": "mat-123",  
 "name": "Marble Tile",  
 "confidence": 0.94,  
 "imageUrl": "https://example.com/images/marble-tile.jpg",  
 "metadata": {  
 "color": "white",  
 "finish": "polished"  
 }  
 },  
 // More results...  
 ],  
 "processingTime": 1.2  
 }  
}

#### Search Endpoints

Table content:

Method | Endpoint | Description | Default Access

GET | /api/search | Search for materials | EXTERNAL\_ALLOWED

GET | /api/search/autocomplete | Get search suggestions | EXTERNAL\_ALLOWED

POST | /api/search/vector | Vector similarity search | EXTERNAL\_ALLOWED

GET | /api/search/recent | Get recent searches | EXTERNAL\_ALLOWED

#### MoodBoard Endpoints

Table content:

Method | Endpoint | Description | Default Access

GET | /api/boards | Get all boards for current user | EXTERNAL\_ALLOWED

GET | /api/boards/:boardId | Get board by ID | EXTERNAL\_ALLOWED

POST | /api/boards | Create a new board | EXTERNAL\_ALLOWED

PUT | /api/boards/:boardId | Update board details | EXTERNAL\_ALLOWED

DELETE | /api/boards/:boardId | Delete a board | EXTERNAL\_ALLOWED

GET | /api/boards/:boardId/items | Get all items in a board | EXTERNAL\_ALLOWED

POST | /api/boards/:boardId/items | Add an item to a board | EXTERNAL\_ALLOWED

PUT | /api/boards/:boardId/items/:itemId | Update item details | EXTERNAL\_ALLOWED

DELETE | /api/boards/:boardId/items/:itemId | Remove an item from a board | EXTERNAL\_ALLOWED

GET | /api/users/:userId/boards | Get public boards for a user | EXTERNAL\_ALLOWED

#### Analytics Endpoints

Table content:

Method | Endpoint | Description | Default Access

POST | /api/analytics/event | Track an analytics event | EXTERNAL\_ALLOWED

POST | /api/analytics/pageview | Track a page view | EXTERNAL\_ALLOWED

POST | /api/analytics/user-activity | Track user activity | EXTERNAL\_ALLOWED

GET | /api/admin/analytics/events | Get analytics events | INTERNAL\_ONLY

GET | /api/admin/analytics/trends | Get analytics trends | INTERNAL\_ONLY

DELETE | /api/admin/analytics/data | Clear analytics data | INTERNAL\_ONLY

POST | /api/admin/analytics/rebuild-index | Rebuild search index | INTERNAL\_ONLY

##### Track User Activity Example

POST /api/analytics/user-activity  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  
Content-Type: application/json  
  
{  
 "eventType": "view\_material",  
 "materialId": "mat-123",  
 "data": {  
 "timeSpent": 45,  
 "viewedImages": 3  
 }  
}

POST /api/analytics/user-activity  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  
Content-Type: application/json  
  
{  
 "eventType": "view\_material",  
 "materialId": "mat-123",  
 "data": {  
 "timeSpent": 45,  
 "viewedImages": 3  
 }  
}

Response:

{  
 "success": true,  
 "message": "Activity logged successfully"  
}

{  
 "success": true,  
 "message": "Activity logged successfully"  
}

#### Admin Endpoints

These endpoints require admin role and some are restricted to internal networks only.

Table content:

Method | Endpoint | Description | Default Access

GET | /api/admin/users | Get all users | INTERNAL\_ONLY

GET | /api/admin/users/:id | Get user by ID | INTERNAL\_ONLY

PUT | /api/admin/users/:id | Update a user | INTERNAL\_ONLY

DELETE | /api/admin/users/:id | Delete a user | INTERNAL\_ONLY

GET | /api/admin/settings | Get system settings | INTERNAL\_ONLY

PUT | /api/admin/settings | Update system settings | INTERNAL\_ONLY

GET | /api/admin/jobs | Get processing jobs | INTERNAL\_ONLY

POST | /api/admin/jobs/:id/cancel | Cancel a job | INTERNAL\_ONLY

GET | /api/admin/dashboard | Get admin dashboard stats | INTERNAL\_ONLY

#### Visual Reference Library API

Table content:

Method | Endpoint | Description | Default Access

GET | /api/property-references | Get property reference images | ANY

POST | /api/property-references | Create a property reference image | ANY

PUT | /api/property-references/:id | Update a property reference image | ANY

DELETE | /api/property-references/:id | Delete a property reference image | ANY

#### Visual Reference AI Training API

Table content:

Method | Endpoint | Description | Default Access

POST | /api/ai/visual-reference/datasets | Create a training dataset from visual references | INTERNAL\_ONLY

POST | /api/ai/visual-reference/models | Train a model using a visual reference dataset | INTERNAL\_ONLY

POST | /api/ai/visual-reference/train | Create a dataset and train a model in one step | INTERNAL\_ONLY

#### Visual Reference OCR API

Table content:

Method | Endpoint | Description | Default Access

POST | /api/ocr/visual-reference/enhance | Enhance OCR extraction with visual reference verification | ANY

POST | /api/ocr/visual-reference/enhance-multiple | Enhance multiple OCR extractions with visual reference verification | ANY

GET | /api/ocr/visual-reference/patterns/:propertyName/:materialType | Get extraction patterns for a property based on visual references | ANY

##### Get Admin Dashboard Stats Example

GET /api/admin/dashboard  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

GET /api/admin/dashboard  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Response:

{  
 "success": true,  
 "data": {  
 "users": {  
 "total": 1250,  
 "active": 850,  
 "new": 120  
 },  
 "materials": {  
 "total": 15000,  
 "views": 25600,  
 "searches": 8500  
 },  
 "recognition": {  
 "requests": 3500,  
 "successRate": 0.92  
 }  
 }  
}

{  
 "success": true,  
 "data": {  
 "users": {  
 "total": 1250,  
 "active": 850,  
 "new": 120  
 },  
 "materials": {  
 "total": 15000,  
 "views": 25600,  
 "searches": 8500  
 },  
 "recognition": {  
 "requests": 3500,  
 "successRate": 0.92  
 }  
 }  
}

##### Network Access Management

Table content:

Method | Endpoint | Description | Default Access

GET | /api/admin/network/internal | Get internal networks | INTERNAL\_ONLY

POST | /api/admin/network/internal | Add internal network | INTERNAL\_ONLY

DELETE | /api/admin/network/internal/:id | Remove internal network | INTERNAL\_ONLY

GET | /api/admin/network/endpoints | Get endpoint access rules | INTERNAL\_ONLY

POST | /api/admin/network/endpoints | Add endpoint access rule | INTERNAL\_ONLY

PUT | /api/admin/network/endpoints/:id | Update endpoint access | INTERNAL\_ONLY

##### Rate Limit Management

Table content:

Method | Endpoint | Description | Default Access

GET | /api/admin/rate-limits/settings | Get rate limit settings | INTERNAL\_ONLY

PUT | /api/admin/rate-limits/settings | Update rate limit settings | INTERNAL\_ONLY

GET | /api/admin/rate-limits/custom | Get custom rate limits | INTERNAL\_ONLY

POST | /api/admin/rate-limits/custom | Add custom rate limit | INTERNAL\_ONLY

PUT | /api/admin/rate-limits/custom/:id | Update custom rate limit | INTERNAL\_ONLY

DELETE | /api/admin/rate-limits/custom/:id | Remove custom rate limit | INTERNAL\_ONLY

#### AI Endpoints

Table content:

Method | Endpoint | Description | Default Access

POST | /api/ai/generate-description | Generate material description | EXTERNAL\_ALLOWED

##### Generate Material Description Example

POST /api/ai/generate-description  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  
Content-Type: application/json  
  
{  
 "materialId": "mat-123",  
 "tone": "professional",  
 "length": "medium"  
}

POST /api/ai/generate-description  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  
Content-Type: application/json  
  
{  
 "materialId": "mat-123",  
 "tone": "professional",  
 "length": "medium"  
}

Response:

{  
 "success": true,  
 "data": {  
 "description": "This premium quality marble tile features a stunning white polished finish that adds elegance to any space. Perfect for luxury residential or commercial flooring applications, the 12\"x12\" dimensions provide versatility for various design patterns."  
 }  
}

{  
 "success": true,  
 "data": {  
 "description": "This premium quality marble tile features a stunning white polished finish that adds elegance to any space. Perfect for luxury residential or commercial flooring applications, the 12\"x12\" dimensions provide versatility for various design patterns."  
 }  
}

#### Agent Endpoints

Table content:

Method | Endpoint | Description | Default Access

POST | /api/agents/material-expert/chat | Chat with material expert | EXTERNAL\_ALLOWED

##### Chat with Material Expert Example

POST /api/agents/material-expert/chat  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  
Content-Type: application/json  
  
{  
 "message": "What's the best material for a kitchen floor?",  
 "sessionId": "session-456"  
}

POST /api/agents/material-expert/chat  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  
Content-Type: application/json  
  
{  
 "message": "What's the best material for a kitchen floor?",  
 "sessionId": "session-456"  
}

Response:

{  
 "success": true,  
 "data": {  
 "reply": "For kitchen floors, I recommend porcelain or ceramic tiles due to their durability and water resistance. Porcelain is particularly good because it's less porous than ceramic, making it more stain-resistant and easier to clean. Natural stone like granite is also excellent but requires more maintenance. Would you like me to suggest some specific options?",  
 "suggestedMaterials": [  
 {  
 "id": "mat-456",  
 "name": "Premium Porcelain Tile",  
 "imageUrl": "https://example.com/images/porcelain-tile.jpg"  
 }  
 ]  
 }  
}

{  
 "success": true,  
 "data": {  
 "reply": "For kitchen floors, I recommend porcelain or ceramic tiles due to their durability and water resistance. Porcelain is particularly good because it's less porous than ceramic, making it more stain-resistant and easier to clean. Natural stone like granite is also excellent but requires more maintenance. Would you like me to suggest some specific options?",  
 "suggestedMaterials": [  
 {  
 "id": "mat-456",  
 "name": "Premium Porcelain Tile",  
 "imageUrl": "https://example.com/images/porcelain-tile.jpg"  
 }  
 ]  
 }  
}

#### 3D Designer Endpoints

Table content:

Method | Endpoint | Description | Default Access

POST | /api/3d-designer/generate-layout | Generate room layout | EXTERNAL\_ALLOWED

POST | /api/3d-designer/scene | Create a new 3D scene | EXTERNAL\_ALLOWED

GET | /api/3d-designer/scene/:id | Get scene by ID | EXTERNAL\_ALLOWED

PUT | /api/3d-designer/scene/:id | Update a scene | EXTERNAL\_ALLOWED

DELETE | /api/3d-designer/scene/:id | Delete a scene | EXTERNAL\_ALLOWED

POST | /api/3d-designer/render | Render a scene | EXTERNAL\_ALLOWED

POST | /api/3d-designer/export | Export a scene | EXTERNAL\_ALLOWED

##### Generate Room Layout Example

POST /api/3d-designer/generate-layout  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  
Content-Type: application/json  
  
{  
 "roomType": "kitchen",  
 "dimensions": {  
 "width": 4.5,  
 "length": 6.2,  
 "height": 2.8  
 },  
 "style": "modern"  
}

POST /api/3d-designer/generate-layout  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...  
Content-Type: application/json  
  
{  
 "roomType": "kitchen",  
 "dimensions": {  
 "width": 4.5,  
 "length": 6.2,  
 "height": 2.8  
 },  
 "style": "modern"  
}

Response:

{  
 "success": true,  
 "data": {  
 "layoutId": "layout-789",  
 "previewUrl": "https://example.com/layouts/preview-789.jpg",  
 "modelUrl": "https://example.com/layouts/model-789.gltf"  
 }  
}

{  
 "success": true,  
 "data": {  
 "layoutId": "layout-789",  
 "previewUrl": "https://example.com/layouts/preview-789.jpg",  
 "modelUrl": "https://example.com/layouts/model-789.gltf"  
 }  
}

#### Subscription Endpoints

Table content:

Method | Endpoint | Description | Default Access

GET | /api/subscriptions/tiers | Get subscription tiers | EXTERNAL\_ALLOWED

POST | /api/subscriptions/update | Update user subscription | EXTERNAL\_ALLOWED

##### Get Subscription Tiers Example

GET /api/subscriptions/tiers

GET /api/subscriptions/tiers

Response:

{  
 "success": true,  
 "data": [  
 {  
 "id": "tier-basic",  
 "name": "Basic",  
 "price": 9.99,  
 "features": [  
 "Material recognition (50/month)",  
 "Basic search",  
 "Standard API access"  
 ]  
 },  
 {  
 "id": "tier-pro",  
 "name": "Professional",  
 "price": 29.99,  
 "features": [  
 "Material recognition (500/month)",  
 "Advanced search with filters",  
 "Enhanced API access",  
 "Material expert agent"  
 ]  
 },  
 {  
 "id": "tier-enterprise",  
 "name": "Enterprise",  
 "price": 99.99,  
 "features": [  
 "Unlimited material recognition",  
 "Advanced search with custom filters",  
 "Full API access",  
 "All AI agents",  
 "Custom branding"  
 ]  
 }  
 ]  
}

{  
 "success": true,  
 "data": [  
 {  
 "id": "tier-basic",  
 "name": "Basic",  
 "price": 9.99,  
 "features": [  
 "Material recognition (50/month)",  
 "Basic search",  
 "Standard API access"  
 ]  
 },  
 {  
 "id": "tier-pro",  
 "name": "Professional",  
 "price": 29.99,  
 "features": [  
 "Material recognition (500/month)",  
 "Advanced search with filters",  
 "Enhanced API access",  
 "Material expert agent"  
 ]  
 },  
 {  
 "id": "tier-enterprise",  
 "name": "Enterprise",  
 "price": 99.99,  
 "features": [  
 "Unlimited material recognition",  
 "Advanced search with custom filters",  
 "Full API access",  
 "All AI agents",  
 "Custom branding"  
 ]  
 }  
 ]  
}

### Network Access Control Implementation

Network access controls are implemented using IP address detection to differentiate between internal and external requests. The system uses CIDR notation to define internal networks.

#### Configuration

Internal networks are defined using the INTERNAL\_NETWORKS environment variable as a comma-separated list of CIDR ranges:

INTERNAL\_NETWORKS

INTERNAL\_NETWORKS=127.0.0.1/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16

INTERNAL\_NETWORKS=127.0.0.1/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16

#### Implementation Details

Network-based access control is implemented alongside other authorization mechanisms:

1. Authentication verifies user identity
2. Role-based access control verifies user permissions
3. Network-based access control verifies request source
4. Subscription-based access control verifies feature access

This layered approach ensures that sensitive operations are protected by multiple security mechanisms.

#### Protected Endpoints

The following endpoints are protected by network-based access control:

Table content:

Endpoint | Method | Description

/api/admin/analytics/data | DELETE | Clear analytics data

/api/admin/backup | POST | Create database backup

/api/admin/restore | POST | Restore from backup

/api/admin/settings | PUT | Update system settings

/api/admin/training/:jobId/stop | POST | Stop ML training job

/api/admin/analytics/data

/api/admin/backup

/api/admin/restore

/api/admin/settings

/api/admin/training/:jobId/stop

### Scripts and Utilities

The following scripts are available to help manage API access control:

1. setup-api-access-control.ts: Sets up the API access control system by checking for unregistered endpoints and registering them
2. check-unregistered-endpoints.ts: Scans the codebase for API endpoints and checks if they are registered
3. register-api-endpoints.ts: Registers API endpoints with the network access control system

To run these scripts:

# From the project root  
cd packages/server  
yarn run-script setup-api-access-control  
yarn run-script check-unregistered-endpoints  
yarn run-script register-api-endpoints

# From the project root  
cd packages/server  
yarn run-script setup-api-access-control  
yarn run-script check-unregistered-endpoints  
yarn run-script register-api-endpoints

### Adding New API Endpoints

When adding new API endpoints, follow these steps to ensure they are properly registered with the access control system:

1. Create your route file and define your endpoints as usual
2. Use the authorize middleware with the appropriate NetworkAccessType:

authorize

NetworkAccessType

router.get(  
 '/your-endpoint',  
 authorize({ roles: ['admin'], accessType: NetworkAccessType.ANY }),  
 asyncHandler(async (req, res) => {  
 // Your handler code  
 })  
);

router.get(  
 '/your-endpoint',  
 authorize({ roles: ['admin'], accessType: NetworkAccessType.ANY }),  
 asyncHandler(async (req, res) => {  
 // Your handler code  
 })  
);

1. Run the setup script to register the endpoint:

# From the project root  
cd packages/server  
yarn run-script setup-api-access-control

# From the project root  
cd packages/server  
yarn run-script setup-api-access-control

1. Verify that your endpoint appears in the admin panel

Important: Do not block endpoints by default. Let administrators control access through the admin panel.

### Best Practices

1. Don't Block by Default: Let administrators control access through the admin panel
2. Register All Endpoints: Make sure all endpoints are registered with the access control system
3. Use Appropriate Access Types: Use INTERNAL\_ONLY for sensitive operations, ANY for public endpoints
4. Document Access Requirements: Document the access requirements for each endpoint
5. Run the Setup Script: Run the setup script after adding new endpoints
6. Check the Admin Panel: Verify that your endpoints appear in the admin panel

INTERNAL\_ONLY

ANY

By following these best practices, you can ensure that your API endpoints are properly secured while allowing appropriate access.

### SDK Examples

#### JavaScript/TypeScript

import KaiClient from '@kai/client';  
  
// Initialize client  
const client = new KaiClient({  
 apiKey: 'your-api-key',  
 baseUrl: 'https://api.example.com'  
});  
  
// Authentication  
async function login() {  
 try {  
 const { token, user } = await client.auth.login({  
 email: 'user@example.com',  
 password: 'password'  
 });  
  
 // Store token for subsequent requests  
 client.setToken(token);  
  
 return user;  
 } catch (error) {  
 console.error('Login failed:', error.message);  
 }  
}  
  
// Material recognition  
async function recognizeMaterial(imageFile) {  
 try {  
 const result = await client.recognition.recognize({  
 image: imageFile,  
 modelType: 'enhanced',  
 maxResults: 5  
 });  
  
 return result.results;  
 } catch (error) {  
 console.error('Recognition failed:', error.message);  
 }  
}  
  
// Admin operations (internal network only)  
async function clearAnalyticsData() {  
 try {  
 await client.admin.analytics.clearData();  
 console.log('Analytics data cleared successfully');  
 } catch (error) {  
 if (error.statusCode === 403) {  
 console.error('Network access denied: This operation requires internal network access');  
 } else {  
 console.error('Operation failed:', error.message);  
 }  
 }  
}

import KaiClient from '@kai/client';  
  
// Initialize client  
const client = new KaiClient({  
 apiKey: 'your-api-key',  
 baseUrl: 'https://api.example.com'  
});  
  
// Authentication  
async function login() {  
 try {  
 const { token, user } = await client.auth.login({  
 email: 'user@example.com',  
 password: 'password'  
 });  
  
 // Store token for subsequent requests  
 client.setToken(token);  
  
 return user;  
 } catch (error) {  
 console.error('Login failed:', error.message);  
 }  
}  
  
// Material recognition  
async function recognizeMaterial(imageFile) {  
 try {  
 const result = await client.recognition.recognize({  
 image: imageFile,  
 modelType: 'enhanced',  
 maxResults: 5  
 });  
  
 return result.results;  
 } catch (error) {  
 console.error('Recognition failed:', error.message);  
 }  
}  
  
// Admin operations (internal network only)  
async function clearAnalyticsData() {  
 try {  
 await client.admin.analytics.clearData();  
 console.log('Analytics data cleared successfully');  
 } catch (error) {  
 if (error.statusCode === 403) {  
 console.error('Network access denied: This operation requires internal network access');  
 } else {  
 console.error('Operation failed:', error.message);  
 }  
 }  
}

#### Python

from kai\_client import KaiClient  
  
# Initialize client  
client = KaiClient(  
 api\_key='your-api-key',  
 base\_url='https://api.example.com'  
)  
  
# Authentication  
def login():  
 try:  
 response = client.auth.login(  
 email='user@example.com',  
 password='password'  
 )  
  
 # Store token for subsequent requests  
 client.set\_token(response['token'])  
  
 return response['user']  
 except Exception as e:  
 print(f"Login failed: {str(e)}")  
  
# Material recognition  
def recognize\_material(image\_path):  
 try:  
 with open(image\_path, 'rb') as image\_file:  
 result = client.recognition.recognize(  
 image=image\_file,  
 model\_type='enhanced',  
 max\_results=5  
 )  
  
 return result['results']  
 except Exception as e:  
 print(f"Recognition failed: {str(e)}")  
  
# Admin operations (internal network only)  
def clear\_analytics\_data():  
 try:  
 client.admin.analytics.clear\_data()  
 print("Analytics data cleared successfully")  
 except Exception as e:  
 if getattr(e, 'status\_code', None) == 403:  
 print("Network access denied: This operation requires internal network access")  
 else:  
 print(f"Operation failed: {str(e)}")

from kai\_client import KaiClient  
  
# Initialize client  
client = KaiClient(  
 api\_key='your-api-key',  
 base\_url='https://api.example.com'  
)  
  
# Authentication  
def login():  
 try:  
 response = client.auth.login(  
 email='user@example.com',  
 password='password'  
 )  
  
 # Store token for subsequent requests  
 client.set\_token(response['token'])  
  
 return response['user']  
 except Exception as e:  
 print(f"Login failed: {str(e)}")  
  
# Material recognition  
def recognize\_material(image\_path):  
 try:  
 with open(image\_path, 'rb') as image\_file:  
 result = client.recognition.recognize(  
 image=image\_file,  
 model\_type='enhanced',  
 max\_results=5  
 )  
  
 return result['results']  
 except Exception as e:  
 print(f"Recognition failed: {str(e)}")  
  
# Admin operations (internal network only)  
def clear\_analytics\_data():  
 try:  
 client.admin.analytics.clear\_data()  
 print("Analytics data cleared successfully")  
 except Exception as e:  
 if getattr(e, 'status\_code', None) == 403:  
 print("Network access denied: This operation requires internal network access")  
 else:  
 print(f"Operation failed: {str(e)}")

For more examples, including implementation in other languages, rate limit handling, and error management, please refer to our Developer Guide.

# Cache Service

Source: readme/cache-service.md

---

## Cache Service

This document describes the cache service implementation in the KAI platform. The cache service provides a unified interface for caching operations across the application, with support for different cache providers.

### Overview

The cache service is designed to improve performance by caching frequently accessed data. It provides a consistent API for caching operations, regardless of the underlying cache provider. The service supports both in-memory caching for development and Redis caching for production.

### Architecture

The cache service follows a provider pattern, allowing different cache implementations to be used interchangeably. The service consists of the following components:

1. Cache Service: The main service that provides a unified interface for caching operations.
2. Cache Provider Interface: An interface that defines the contract for cache providers.
3. Memory Cache Provider: A provider that implements in-memory caching.
4. Redis Cache Provider: A provider that implements Redis-based caching.
5. Cache Initializer: A utility for initializing the cache service with different providers.

### Usage

#### Basic Usage

import { cache } from '@kai/shared';  
  
// Set a value in the cache  
await cache.set('user:123', { name: 'John', email: 'john@example.com' });  
  
// Get a value from the cache  
const user = await cache.get<User>('user:123');  
  
// Delete a value from the cache  
await cache.delete('user:123');  
  
// Check if a key exists in the cache  
const exists = await cache.has('user:123');  
  
// Clear all values from the cache  
await cache.clear();

import { cache } from '@kai/shared';  
  
// Set a value in the cache  
await cache.set('user:123', { name: 'John', email: 'john@example.com' });  
  
// Get a value from the cache  
const user = await cache.get<User>('user:123');  
  
// Delete a value from the cache  
await cache.delete('user:123');  
  
// Check if a key exists in the cache  
const exists = await cache.has('user:123');  
  
// Clear all values from the cache  
await cache.clear();

#### Caching Function Results

import { cache } from '@kai/shared';  
  
// Define a function to cache  
async function getUserById(id: string): Promise<User> {  
 // Expensive operation to get user  
 return db.users.findOne({ id });  
}  
  
// Create a cached version of the function  
const getCachedUserById = cache.cached(  
 getUserById,  
 (id) => `user:${id}`,  
 { ttl: 3600 } // Cache for 1 hour  
);  
  
// Use the cached function  
const user = await getCachedUserById('123');

import { cache } from '@kai/shared';  
  
// Define a function to cache  
async function getUserById(id: string): Promise<User> {  
 // Expensive operation to get user  
 return db.users.findOne({ id });  
}  
  
// Create a cached version of the function  
const getCachedUserById = cache.cached(  
 getUserById,  
 (id) => `user:${id}`,  
 { ttl: 3600 } // Cache for 1 hour  
);  
  
// Use the cached function  
const user = await getCachedUserById('123');

#### Namespaces

import { cache } from '@kai/shared';  
  
// Set a value in a specific namespace  
await cache.set('123', userData, { namespace: 'users' });  
  
// Get a value from a specific namespace  
const user = await cache.get<User>('123', { namespace: 'users' });  
  
// Clear all values in a specific namespace  
await cache.clear({ namespace: 'users' });

import { cache } from '@kai/shared';  
  
// Set a value in a specific namespace  
await cache.set('123', userData, { namespace: 'users' });  
  
// Get a value from a specific namespace  
const user = await cache.get<User>('123', { namespace: 'users' });  
  
// Clear all values in a specific namespace  
await cache.clear({ namespace: 'users' });

### Configuration

The cache service can be configured through environment variables or the unified configuration system. The following configuration options are available:

// In .env file  
CACHE\_TYPE=redis  
CACHE\_NAMESPACE=default  
CACHE\_TTL=3600  
CACHE\_REDIS\_URL=redis://localhost:6379  
CACHE\_REDIS\_HOST=localhost  
CACHE\_REDIS\_PORT=6379  
CACHE\_REDIS\_PASSWORD=password  
CACHE\_REDIS\_DB=0  
CACHE\_REDIS\_TLS=false  
CACHE\_REDIS\_CONNECT\_TIMEOUT=10000  
CACHE\_MEMORY\_CLEANUP\_INTERVAL=60000

// In .env file  
CACHE\_TYPE=redis  
CACHE\_NAMESPACE=default  
CACHE\_TTL=3600  
CACHE\_REDIS\_URL=redis://localhost:6379  
CACHE\_REDIS\_HOST=localhost  
CACHE\_REDIS\_PORT=6379  
CACHE\_REDIS\_PASSWORD=password  
CACHE\_REDIS\_DB=0  
CACHE\_REDIS\_TLS=false  
CACHE\_REDIS\_CONNECT\_TIMEOUT=10000  
CACHE\_MEMORY\_CLEANUP\_INTERVAL=60000

### Cache Providers

#### Memory Cache Provider

The memory cache provider implements in-memory caching using a Map. It's suitable for development and small-scale applications. Features include:

* In-memory storage using a Map
* TTL support with automatic expiration
* Periodic cleanup of expired entries
* Namespace support
* Memory usage statistics

#### Redis Cache Provider

The Redis cache provider implements Redis-based caching. It's suitable for production and distributed applications. Features include:

* Redis-based storage
* TTL support with automatic expiration
* Namespace support
* Connection management with automatic reconnection
* Error handling and logging
* Redis statistics

### Implementation Details

#### Cache Service

The cache service provides a unified interface for caching operations. It delegates all operations to the configured provider and adds additional functionality like namespaced keys and function result caching.

class CacheService {  
 private provider: CacheProvider | null = null;  
 private defaultNamespace: string = 'default';  
 private defaultTtl: number = 3600; // 1 hour  
  
 // Set the cache provider  
 setProvider(provider: CacheProvider): void;  
  
 // Set the default namespace  
 setDefaultNamespace(namespace: string): void;  
  
 // Set the default TTL  
 setDefaultTtl(ttl: number): void;  
  
 // Get a value from the cache  
 async get<T>(key: string, options?: CacheOptions): Promise<T | null>;  
  
 // Set a value in the cache  
 async set<T>(key: string, value: T, options?: CacheOptions): Promise<void>;  
  
 // Delete a value from the cache  
 async delete(key: string, options?: CacheOptions): Promise<void>;  
  
 // Clear all values from the cache  
 async clear(options?: CacheOptions): Promise<void>;  
  
 // Check if a key exists in the cache  
 async has(key: string, options?: CacheOptions): Promise<boolean>;  
  
 // Cache a function result  
 cached<T, Args extends any[]>(  
 fn: (...args: Args) => Promise<T>,  
 keyFn: (...args: Args) => string,  
 options?: CacheOptions  
 ): (...args: Args) => Promise<T>;  
}

class CacheService {  
 private provider: CacheProvider | null = null;  
 private defaultNamespace: string = 'default';  
 private defaultTtl: number = 3600; // 1 hour  
  
 // Set the cache provider  
 setProvider(provider: CacheProvider): void;  
  
 // Set the default namespace  
 setDefaultNamespace(namespace: string): void;  
  
 // Set the default TTL  
 setDefaultTtl(ttl: number): void;  
  
 // Get a value from the cache  
 async get<T>(key: string, options?: CacheOptions): Promise<T | null>;  
  
 // Set a value in the cache  
 async set<T>(key: string, value: T, options?: CacheOptions): Promise<void>;  
  
 // Delete a value from the cache  
 async delete(key: string, options?: CacheOptions): Promise<void>;  
  
 // Clear all values from the cache  
 async clear(options?: CacheOptions): Promise<void>;  
  
 // Check if a key exists in the cache  
 async has(key: string, options?: CacheOptions): Promise<boolean>;  
  
 // Cache a function result  
 cached<T, Args extends any[]>(  
 fn: (...args: Args) => Promise<T>,  
 keyFn: (...args: Args) => string,  
 options?: CacheOptions  
 ): (...args: Args) => Promise<T>;  
}

#### Cache Provider Interface

The cache provider interface defines the contract for cache providers. All providers must implement this interface.

interface CacheProvider {  
 // Get a value from the cache  
 get<T>(key: string, options?: CacheOptions): Promise<T | null>;  
  
 // Set a value in the cache  
 set<T>(key: string, value: T, options?: CacheOptions): Promise<void>;  
  
 // Delete a value from the cache  
 delete(key: string, options?: CacheOptions): Promise<void>;  
  
 // Clear all values from the cache  
 clear(options?: CacheOptions): Promise<void>;  
  
 // Check if a key exists in the cache  
 has(key: string, options?: CacheOptions): Promise<boolean>;  
}

interface CacheProvider {  
 // Get a value from the cache  
 get<T>(key: string, options?: CacheOptions): Promise<T | null>;  
  
 // Set a value in the cache  
 set<T>(key: string, value: T, options?: CacheOptions): Promise<void>;  
  
 // Delete a value from the cache  
 delete(key: string, options?: CacheOptions): Promise<void>;  
  
 // Clear all values from the cache  
 clear(options?: CacheOptions): Promise<void>;  
  
 // Check if a key exists in the cache  
 has(key: string, options?: CacheOptions): Promise<boolean>;  
}

### Benefits

The cache service provides several benefits:

1. Improved Performance: Caching frequently accessed data reduces database load and improves response times.
2. Consistent API: The unified interface provides a consistent API for caching operations, regardless of the underlying provider.
3. Provider Flexibility: The provider pattern allows different cache implementations to be used interchangeably.
4. Namespace Support: Namespaces allow for logical separation of cached data.
5. TTL Support: Time-to-live support ensures that cached data doesn't become stale.
6. Function Result Caching: The cached method makes it easy to cache function results.

cached

### Next Steps

The following steps are recommended to further improve the cache service:

1. Add More Providers: Add support for more cache providers (Memcached, DynamoDB, etc.).
2. Add Cache Invalidation: Add support for cache invalidation based on events or patterns.
3. Add Cache Warming: Add support for cache warming to pre-populate the cache.
4. Add Cache Metrics: Add support for cache metrics to monitor cache performance.
5. Add Cache Compression: Add support for cache compression to reduce memory usage.
6. Add Cache Sharding: Add support for cache sharding to distribute cache load.

# Cache Warming Cron Scheduling

Source: readme/cache-warming-cron-scheduling.md

---

## Cache Warming Cron Scheduling

This document describes the implementation of proper cron scheduling for the cache warming service. The cache warming service provides a unified interface for warming and prefetching cache entries, improving performance by proactively populating the cache with frequently accessed data.

### Overview

The cache warming service now supports proper cron scheduling for warming sources. This allows for more flexible and precise scheduling of cache warming operations, enabling administrators to define when cache warming should occur based on application usage patterns.

### Cron Parser

A custom cron parser has been implemented to support cron expressions for scheduling cache warming operations. The parser supports a subset of the cron syntax and provides functions for parsing cron expressions, calculating the next execution time, and validating cron expressions.

#### Cron Expression Format

The cron expression format is as follows:

\* \* \* \* \*  
| | | | |  
| | | | +----- Day of week (0 - 6) (Sunday = 0)  
| | | +-------- Month (1 - 12)  
| | +----------- Day of month (1 - 31)  
| +-------------- Hour (0 - 23)  
+----------------- Minute (0 - 59)

\* \* \* \* \*  
| | | | |  
| | | | +----- Day of week (0 - 6) (Sunday = 0)  
| | | +-------- Month (1 - 12)  
| | +----------- Day of month (1 - 31)  
| +-------------- Hour (0 - 23)  
+----------------- Minute (0 - 59)

#### Special Expressions

The parser also supports special expressions for common scheduling patterns:

* @yearly, @annually: Run once a year at midnight on January 1st (0 0 1 1 \*)
* @monthly: Run once a month at midnight on the first day (0 0 1 \* \*)
* @weekly: Run once a week at midnight on Sunday (0 0 \* \* 0)
* @daily, @midnight: Run once a day at midnight (0 0 \* \* \*)
* @hourly: Run once an hour at the beginning of the hour (0 \* \* \* \*)
* @every\_minute: Run once a minute ( \* \* \* )
* @every\_5\_minutes: Run every 5 minutes (/5 \* \* \* )
* @every\_10\_minutes: Run every 10 minutes (/10 \* \* \* )
* @every\_15\_minutes: Run every 15 minutes (/15 \* \* \* )
* @every\_30\_minutes: Run every 30 minutes (/30 \* \* \* )

#### Cron Parser Functions

The cron parser provides the following functions:

##### parseCronToMs

Parses a cron expression into a millisecond interval.

function parseCronToMs(expression: string): number;

function parseCronToMs(expression: string): number;

##### getNextExecutionTime

Gets the next execution time for a cron expression.

function getNextExecutionTime(expression: string, baseTime?: Date): Date;

function getNextExecutionTime(expression: string, baseTime?: Date): Date;

##### isValidCronExpression

Checks if a cron expression is valid.

function isValidCronExpression(expression: string): boolean;

function isValidCronExpression(expression: string): boolean;

### Cache Warming Service

The cache warming service has been updated to use the cron parser for scheduling cache warming operations. The following changes have been made:

#### CacheWarmingSource Interface

The CacheWarmingSource interface has been updated to include a schedule property for defining when cache warming should occur.

CacheWarmingSource

schedule

export interface CacheWarmingSource<T = any> {  
 /\*\* Source ID \*/  
 id: string;  
 /\*\* Source name \*/  
 name: string;  
 /\*\* Cache namespace \*/  
 namespace: string;  
 /\*\* Cache TTL in seconds \*/  
 ttl?: number;  
 /\*\* Warming strategy \*/  
 strategy: CacheWarmingStrategy;  
 /\*\* Schedule (cron expression) for scheduled warming \*/  
 schedule?: string;  
 /\*\* Function to fetch data \*/  
 fetch: () => Promise<Record<string, T>>;  
 /\*\* Description of the source \*/  
 description?: string;  
}

export interface CacheWarmingSource<T = any> {  
 /\*\* Source ID \*/  
 id: string;  
 /\*\* Source name \*/  
 name: string;  
 /\*\* Cache namespace \*/  
 namespace: string;  
 /\*\* Cache TTL in seconds \*/  
 ttl?: number;  
 /\*\* Warming strategy \*/  
 strategy: CacheWarmingStrategy;  
 /\*\* Schedule (cron expression) for scheduled warming \*/  
 schedule?: string;  
 /\*\* Function to fetch data \*/  
 fetch: () => Promise<Record<string, T>>;  
 /\*\* Description of the source \*/  
 description?: string;  
}

#### scheduleWarming Method

The scheduleWarming method has been updated to use the cron parser for scheduling cache warming operations. It now supports cron expressions for defining when cache warming should occur.

scheduleWarming

private scheduleWarming(source: CacheWarmingSource): void {  
 // Cancel existing schedule if any  
 if (this.schedules.has(source.id)) {  
 clearTimeout(this.schedules.get(source.id)!);  
 }  
  
 // Validate schedule  
 if (!source.schedule) {  
 logger.warn(`No schedule provided for source: ${source.id}, using default hourly schedule`);  
 source.schedule = '@hourly';  
 }  
  
 if (!isValidCronExpression(source.schedule)) {  
 logger.warn(`Invalid cron expression: ${source.schedule} for source: ${source.id}, using default hourly schedule`);  
 source.schedule = '@hourly';  
 }  
  
 // Get the interval from the cron expression  
 const interval = parseCronToMs(source.schedule);  
  
 // Schedule the first execution  
 const scheduleNextExecution = () => {  
 const now = new Date();  
 const nextExecutionTime = getNextExecutionTime(source.schedule!, now);  
 const delay = nextExecutionTime.getTime() - now.getTime();  
  
 logger.debug(`Scheduling next execution for source: ${source.id}`, {  
 currentTime: now.toISOString(),  
 nextExecutionTime: nextExecutionTime.toISOString(),  
 delayMs: delay  
 });  
  
 // Schedule the next execution  
 const timeout = setTimeout(() => {  
 // Execute the warming  
 this.warmSource(source.id)  
 .catch((error) => {  
 logger.error(`Error in scheduled warming for source: ${source.id}`, error);  
 })  
 .finally(() => {  
 // Schedule the next execution  
 scheduleNextExecution();  
 });  
 }, delay);  
  
 // Store the timeout  
 this.schedules.set(source.id, timeout);  
 };  
  
 // Start the scheduling  
 scheduleNextExecution();  
  
 // Log the scheduling  
 logger.info(`Scheduled warming for source: ${source.id}`, {  
 schedule: source.schedule,  
 intervalMs: interval  
 });  
}

private scheduleWarming(source: CacheWarmingSource): void {  
 // Cancel existing schedule if any  
 if (this.schedules.has(source.id)) {  
 clearTimeout(this.schedules.get(source.id)!);  
 }  
  
 // Validate schedule  
 if (!source.schedule) {  
 logger.warn(`No schedule provided for source: ${source.id}, using default hourly schedule`);  
 source.schedule = '@hourly';  
 }  
  
 if (!isValidCronExpression(source.schedule)) {  
 logger.warn(`Invalid cron expression: ${source.schedule} for source: ${source.id}, using default hourly schedule`);  
 source.schedule = '@hourly';  
 }  
  
 // Get the interval from the cron expression  
 const interval = parseCronToMs(source.schedule);  
  
 // Schedule the first execution  
 const scheduleNextExecution = () => {  
 const now = new Date();  
 const nextExecutionTime = getNextExecutionTime(source.schedule!, now);  
 const delay = nextExecutionTime.getTime() - now.getTime();  
  
 logger.debug(`Scheduling next execution for source: ${source.id}`, {  
 currentTime: now.toISOString(),  
 nextExecutionTime: nextExecutionTime.toISOString(),  
 delayMs: delay  
 });  
  
 // Schedule the next execution  
 const timeout = setTimeout(() => {  
 // Execute the warming  
 this.warmSource(source.id)  
 .catch((error) => {  
 logger.error(`Error in scheduled warming for source: ${source.id}`, error);  
 })  
 .finally(() => {  
 // Schedule the next execution  
 scheduleNextExecution();  
 });  
 }, delay);  
  
 // Store the timeout  
 this.schedules.set(source.id, timeout);  
 };  
  
 // Start the scheduling  
 scheduleNextExecution();  
  
 // Log the scheduling  
 logger.info(`Scheduled warming for source: ${source.id}`, {  
 schedule: source.schedule,  
 intervalMs: interval  
 });  
}

#### initialize Method

The initialize method has been updated to handle existing schedules. It now cancels any existing schedules and re-schedules any existing sources.

initialize

initialize(): void {  
 if (this.initialized) {  
 return;  
 }  
  
 this.initialized = true;  
 this.enabled = config.get('cache.warming.enabled', true);  
  
 // Cancel any existing schedules  
 for (const timeout of this.schedules.values()) {  
 clearTimeout(timeout);  
 }  
 this.schedules.clear();  
  
 // Re-schedule any existing sources  
 if (this.enabled) {  
 for (const source of this.sources.values()) {  
 if (source.strategy === CacheWarmingStrategy.SCHEDULED && source.schedule) {  
 this.scheduleWarming(source);  
 }  
 }  
 }  
  
 // Track with telemetry  
 telemetry.trackEvent({  
 type: 'cache',  
 name: 'cache\_warming\_initialized',  
 timestamp: Date.now(),  
 status: 'success',  
 properties: {  
 enabled: this.enabled,  
 sourceCount: this.sources.size,  
 scheduledCount: Array.from(this.sources.values()).filter(  
 source => source.strategy === CacheWarmingStrategy.SCHEDULED && source.schedule  
 ).length  
 }  
 });  
  
 logger.info('Cache warming service initialized', {  
 enabled: this.enabled,  
 sourceCount: this.sources.size,  
 scheduledCount: this.schedules.size  
 });  
}

initialize(): void {  
 if (this.initialized) {  
 return;  
 }  
  
 this.initialized = true;  
 this.enabled = config.get('cache.warming.enabled', true);  
  
 // Cancel any existing schedules  
 for (const timeout of this.schedules.values()) {  
 clearTimeout(timeout);  
 }  
 this.schedules.clear();  
  
 // Re-schedule any existing sources  
 if (this.enabled) {  
 for (const source of this.sources.values()) {  
 if (source.strategy === CacheWarmingStrategy.SCHEDULED && source.schedule) {  
 this.scheduleWarming(source);  
 }  
 }  
 }  
  
 // Track with telemetry  
 telemetry.trackEvent({  
 type: 'cache',  
 name: 'cache\_warming\_initialized',  
 timestamp: Date.now(),  
 status: 'success',  
 properties: {  
 enabled: this.enabled,  
 sourceCount: this.sources.size,  
 scheduledCount: Array.from(this.sources.values()).filter(  
 source => source.strategy === CacheWarmingStrategy.SCHEDULED && source.schedule  
 ).length  
 }  
 });  
  
 logger.info('Cache warming service initialized', {  
 enabled: this.enabled,  
 sourceCount: this.sources.size,  
 scheduledCount: this.schedules.size  
 });  
}

#### enable and disable Methods

The enable and disable methods have been updated to handle schedules. The enable method now re-schedules any existing sources, and the disable method now cancels all schedules.

enable

disable

enable

disable

enable(): void {  
 if (this.enabled) {  
 return;  
 }  
  
 this.enabled = true;  
  
 // Re-schedule any existing sources  
 for (const source of this.sources.values()) {  
 if (source.strategy === CacheWarmingStrategy.SCHEDULED && source.schedule) {  
 this.scheduleWarming(source);  
 }  
 }  
  
 // Track with telemetry  
 telemetry.trackEvent({  
 type: 'cache',  
 name: 'cache\_warming\_enabled',  
 timestamp: Date.now(),  
 status: 'success',  
 properties: {  
 sourceCount: this.sources.size,  
 scheduledCount: this.schedules.size  
 }  
 });  
  
 logger.info('Cache warming enabled', {  
 sourceCount: this.sources.size,  
 scheduledCount: this.schedules.size  
 });  
}  
  
disable(): void {  
 if (!this.enabled) {  
 return;  
 }  
  
 this.enabled = false;  
  
 // Cancel all schedules  
 for (const timeout of this.schedules.values()) {  
 clearTimeout(timeout);  
 }  
 this.schedules.clear();  
  
 // Track with telemetry  
 telemetry.trackEvent({  
 type: 'cache',  
 name: 'cache\_warming\_disabled',  
 timestamp: Date.now(),  
 status: 'success',  
 properties: {  
 sourceCount: this.sources.size  
 }  
 });  
  
 logger.info('Cache warming disabled', {  
 sourceCount: this.sources.size  
 });  
}

enable(): void {  
 if (this.enabled) {  
 return;  
 }  
  
 this.enabled = true;  
  
 // Re-schedule any existing sources  
 for (const source of this.sources.values()) {  
 if (source.strategy === CacheWarmingStrategy.SCHEDULED && source.schedule) {  
 this.scheduleWarming(source);  
 }  
 }  
  
 // Track with telemetry  
 telemetry.trackEvent({  
 type: 'cache',  
 name: 'cache\_warming\_enabled',  
 timestamp: Date.now(),  
 status: 'success',  
 properties: {  
 sourceCount: this.sources.size,  
 scheduledCount: this.schedules.size  
 }  
 });  
  
 logger.info('Cache warming enabled', {  
 sourceCount: this.sources.size,  
 scheduledCount: this.schedules.size  
 });  
}  
  
disable(): void {  
 if (!this.enabled) {  
 return;  
 }  
  
 this.enabled = false;  
  
 // Cancel all schedules  
 for (const timeout of this.schedules.values()) {  
 clearTimeout(timeout);  
 }  
 this.schedules.clear();  
  
 // Track with telemetry  
 telemetry.trackEvent({  
 type: 'cache',  
 name: 'cache\_warming\_disabled',  
 timestamp: Date.now(),  
 status: 'success',  
 properties: {  
 sourceCount: this.sources.size  
 }  
 });  
  
 logger.info('Cache warming disabled', {  
 sourceCount: this.sources.size  
 });  
}

### Usage

To use the cron scheduling feature, define a cache warming source with a cron expression for the schedule:

import { cacheWarming, CacheWarmingStrategy } from '@kai/shared';  
  
// Add a cache warming source with a cron schedule  
cacheWarming.addSource({  
 id: 'popular-products',  
 name: 'Popular Products',  
 namespace: 'products',  
 strategy: CacheWarmingStrategy.SCHEDULED,  
 schedule: '0 \*/2 \* \* \*', // Every 2 hours  
 fetch: async () => {  
 // Fetch popular products  
 const products = await fetchPopularProducts();  
  
 // Return products as a record of key-value pairs  
 return products.reduce((acc, product) => {  
 acc[product.id] = product;  
 return acc;  
 }, {});  
 },  
 description: 'Warm cache for popular products'  
});

import { cacheWarming, CacheWarmingStrategy } from '@kai/shared';  
  
// Add a cache warming source with a cron schedule  
cacheWarming.addSource({  
 id: 'popular-products',  
 name: 'Popular Products',  
 namespace: 'products',  
 strategy: CacheWarmingStrategy.SCHEDULED,  
 schedule: '0 \*/2 \* \* \*', // Every 2 hours  
 fetch: async () => {  
 // Fetch popular products  
 const products = await fetchPopularProducts();  
  
 // Return products as a record of key-value pairs  
 return products.reduce((acc, product) => {  
 acc[product.id] = product;  
 return acc;  
 }, {});  
 },  
 description: 'Warm cache for popular products'  
});

### Benefits

The implementation of proper cron scheduling provides several benefits:

1. Flexibility: Support for cron expressions allows for more flexible scheduling
2. Precision: Cron expressions allow for more precise scheduling based on application usage patterns
3. Efficiency: Scheduling cache warming operations during off-peak hours reduces the impact on application performance
4. Consistency: Using a standard cron syntax makes it easier for administrators to define schedules
5. Reliability: The scheduling mechanism is more reliable and handles edge cases better

### Next Steps

The following steps are recommended to further improve the cache warming service:

1. Add Support for More Cron Features: Add support for more cron features (e.g., step values, ranges, lists)
2. Add Support for Timezone: Add support for specifying the timezone for cron expressions
3. Add Support for Jitter: Add support for adding jitter to scheduled executions to prevent thundering herd problems
4. Add Support for Backoff: Add support for exponential backoff for failed executions
5. Add Support for Dependencies: Add support for defining dependencies between warming sources

# Cicd Pipeline

Source: readme/cicd-pipeline.md

---

## CI/CD Pipeline for Kai Platform

This document outlines the CI/CD (Continuous Integration/Continuous Deployment) pipeline configuration for the Kai platform, including branch structure, protection rules, and automated deployment processes.

### Branch Structure

The Kai repository uses a three-branch structure:

1. main: Production branch that triggers deployments
2. staging: Pre-production testing branch
3. development: Active development branch

The workflow follows this pattern:  
- Developers work in feature branches branched from development  
- Features are merged into development via pull requests  
- When ready for testing, development is merged to staging  
- After validation in staging, staging is merged to main to trigger production deployment

development

development

development

staging

staging

main

### Branch Protection Rules

Branch protection rules are set up to ensure code quality and control access:

#### Main Branch Protection

* Only user "Basilakis" can push directly to main
* Require pull request reviews before merging
* Require status checks to pass before merging
* Require linear history (no merge commits)
* Do not allow bypassing the above settings

main

#### Staging Branch Protection

* Require pull request reviews before merging
* Require status checks to pass before merging
* Allow administrators to bypass

#### Development Branch Protection

* Require status checks to pass before merging
* No restrictions on who can push

### CI/CD Pipeline with Reusable Workflows

The CI/CD pipeline is implemented using GitHub Actions with a modular, reusable workflow approach. This new implementation separates the pipeline into individual workflow files that can be called from the main workflow:

#### Main Workflow File Structure

The primary workflow is defined in .github/workflows/deploy.yml and orchestrates the entire CI/CD process by calling reusable workflows:

.github/workflows/deploy.yml

name: Kai Platform CI/CD Pipeline  
  
jobs:  
 # Build and test job using reusable workflow  
 build-and-test:  
 name: Build and Test  
 uses: ./.github/workflows/build-test.yml  
 with:  
 node-version: '16'  
 python-version: '3.9'  
  
 # Build Docker images using reusable workflow  
 build-docker-images:  
 name: Build Docker Images  
 needs: build-and-test  
 uses: ./.github/workflows/docker-build.yml  
 with:  
 environment: ${{ github.ref == 'refs/heads/main' && 'production' || 'staging' }}  
 tag-suffix: ${{ github.ref == 'refs/heads/main' && 'latest' || 'staging' }}  
 secrets:  
 github\_token: ${{ secrets.GITHUB\_TOKEN }}  
  
 # Deploy to staging or production using environment-specific workflows  
 deploy-staging:  
 name: Deploy to Staging  
 needs: build-docker-images  
 if: github.ref == 'refs/heads/staging'  
 uses: ./.github/workflows/deploy-staging.yml  
 with:  
 sha: ${{ github.sha }}  
 secrets: # Secrets passed to the workflow  
 # Various secrets needed for deployment  
  
 deploy-production:  
 name: Deploy to Production  
 needs: build-docker-images  
 if: github.ref == 'refs/heads/main'  
 uses: ./.github/workflows/deploy-production.yml  
 with:  
 sha: ${{ github.sha }}  
 secrets: # Secrets passed to the workflow  
 # Various secrets needed for deployment

name: Kai Platform CI/CD Pipeline  
  
jobs:  
 # Build and test job using reusable workflow  
 build-and-test:  
 name: Build and Test  
 uses: ./.github/workflows/build-test.yml  
 with:  
 node-version: '16'  
 python-version: '3.9'  
  
 # Build Docker images using reusable workflow  
 build-docker-images:  
 name: Build Docker Images  
 needs: build-and-test  
 uses: ./.github/workflows/docker-build.yml  
 with:  
 environment: ${{ github.ref == 'refs/heads/main' && 'production' || 'staging' }}  
 tag-suffix: ${{ github.ref == 'refs/heads/main' && 'latest' || 'staging' }}  
 secrets:  
 github\_token: ${{ secrets.GITHUB\_TOKEN }}  
  
 # Deploy to staging or production using environment-specific workflows  
 deploy-staging:  
 name: Deploy to Staging  
 needs: build-docker-images  
 if: github.ref == 'refs/heads/staging'  
 uses: ./.github/workflows/deploy-staging.yml  
 with:  
 sha: ${{ github.sha }}  
 secrets: # Secrets passed to the workflow  
 # Various secrets needed for deployment  
  
 deploy-production:  
 name: Deploy to Production  
 needs: build-docker-images  
 if: github.ref == 'refs/heads/main'  
 uses: ./.github/workflows/deploy-production.yml  
 with:  
 sha: ${{ github.sha }}  
 secrets: # Secrets passed to the workflow  
 # Various secrets needed for deployment

#### Reusable Workflow Components

The CI/CD pipeline is now broken down into these reusable workflow files:

1. Build and Test Workflow (.github/workflows/build-test.yml)
2. Accepts parameters for Node.js and Python versions
3. Handles dependency installation, linting, testing, and building
4. Uploads build artifacts for downstream jobs
5. Can be called independently for PR validation

.github/workflows/build-test.yml

yaml  
 on:  
 workflow\_call:  
 inputs:  
 node-version:  
 type: string  
 default: '20'  
 python-version:  
 type: string  
 default: '3.9'

yaml  
 on:  
 workflow\_call:  
 inputs:  
 node-version:  
 type: string  
 default: '20'  
 python-version:  
 type: string  
 default: '3.9'

1. Docker Build Workflow (.github/workflows/docker-build.yml)
2. Builds all required Docker images in two phases
3. First builds the centralized ML base image
4. Then builds all service images in parallel using the matrix strategy
5. Accepts parameters for environment-specific configuration
6. Passes the ML base image reference to service builds

.github/workflows/docker-build.yml

yaml  
 on:  
 workflow\_call:  
 inputs:  
 environment:  
 type: string  
 required: true  
 tag-suffix:  
 type: string  
 default: ''  
 secrets:  
 docker\_username:  
 required: true  
 # Other required secrets

yaml  
 on:  
 workflow\_call:  
 inputs:  
 environment:  
 type: string  
 required: true  
 tag-suffix:  
 type: string  
 default: ''  
 secrets:  
 docker\_username:  
 required: true  
 # Other required secrets

1. Deploy Staging Workflow (.github/workflows/deploy-staging.yml)
2. Handles all staging-specific deployment steps
3. Updates GitOps repository with new image tags
4. Deploys frontend applications to Vercel
5. Runs database migrations for staging environment
6. Deploy Production Workflow (.github/workflows/deploy-production.yml)
7. Similar to staging but with production-specific parameters
8. Applies stricter deployment controls
9. Uses production-specific secrets and configurations

.github/workflows/deploy-staging.yml

Runs database migrations for staging environment

Deploy Production Workflow (.github/workflows/deploy-production.yml)

.github/workflows/deploy-production.yml

#### Benefits of Reusable Workflows

This modular approach provides several key advantages:

1. DRY (Don't Repeat Yourself) Principle
2. Eliminates duplicated code between staging and production workflows
3. Common logic is defined once and reused across workflows
4. Simplified Maintenance
5. Easier to update individual components without affecting others
6. Clear separation of concerns between build, test, and deployment steps
7. Improved Readability
8. Each workflow file focuses on a specific responsibility
9. Main workflow file serves as a clean, high-level orchestrator
10. Easier Troubleshooting
11. Issues can be isolated to specific workflow components
12. Individual workflows can be tested independently
13. Consistent Environment Handling
14. Environment-specific logic is encapsulated in dedicated workflow files
15. Reduces risk of environment configuration inconsistencies
16. Scalable Architecture
17. New environments can be added with minimal changes
18. Additional workflow components can be easily integrated

Common logic is defined once and reused across workflows

Simplified Maintenance

Clear separation of concerns between build, test, and deployment steps

Improved Readability

Main workflow file serves as a clean, high-level orchestrator

Easier Troubleshooting

Individual workflows can be tested independently

Consistent Environment Handling

Reduces risk of environment configuration inconsistencies

Scalable Architecture

#### Enhanced Docker Build Process

The Docker build workflow has been optimized to build the centralized ML base image first, followed by all service images:

jobs:  
 # First build the ML base image that other images depend on  
 build-ml-base-image:  
 name: Build ML Base Image  
 runs-on: ubuntu-latest  
 steps:  
 # Steps to build and push the ML base image  
  
 # Then build all service images in parallel  
 build-service-images:  
 name: Build Service Images  
 needs: build-ml-base-image  
 runs-on: ubuntu-latest  
 strategy:  
 matrix:  
 include:  
 - name: api-server  
 dockerfile: ./Dockerfile.api  
 # Other service images in the matrix  
 steps:  
 # Steps to build and push service images  
 # Includes reference to the ML base image

jobs:  
 # First build the ML base image that other images depend on  
 build-ml-base-image:  
 name: Build ML Base Image  
 runs-on: ubuntu-latest  
 steps:  
 # Steps to build and push the ML base image  
  
 # Then build all service images in parallel  
 build-service-images:  
 name: Build Service Images  
 needs: build-ml-base-image  
 runs-on: ubuntu-latest  
 strategy:  
 matrix:  
 include:  
 - name: api-server  
 dockerfile: ./Dockerfile.api  
 # Other service images in the matrix  
 steps:  
 # Steps to build and push service images  
 # Includes reference to the ML base image

This approach ensures that:  
1. The ML base image is available for all service builds  
2. Common layers are properly cached and reused  
3. Service Dockerfiles are kept simple and focused

#### GitOps Integration

Both staging and production deployment workflows include steps to update the GitOps repository:

# From deploy-staging.yml  
jobs:  
 update-gitops:  
 name: Update GitOps Repository  
 steps:  
 - name: Checkout GitOps repository  
 uses: actions/checkout@v3  
 with:  
 repository: kai-platform/kai-gitops  
 path: gitops  
 token: ${{ secrets.gitops\_pat }}  
 ref: staging  
  
 - name: Update image tags in HelmReleases  
 # Steps to update and commit changes

# From deploy-staging.yml  
jobs:  
 update-gitops:  
 name: Update GitOps Repository  
 steps:  
 - name: Checkout GitOps repository  
 uses: actions/checkout@v3  
 with:  
 repository: kai-platform/kai-gitops  
 path: gitops  
 token: ${{ secrets.gitops\_pat }}  
 ref: staging  
  
 - name: Update image tags in HelmReleases  
 # Steps to update and commit changes

The production workflow uses a similar process but targets the main branch of the GitOps repository.

main

#### Docker Image Build Strategy

The pipeline builds multiple Docker images for different components of the system:

#### API Server Image

* Built from Dockerfile.api in the repository root
* Contains the main API server, authentication, and business logic

Dockerfile.api

#### Coordinator Service Image

* Built from packages/coordinator/Dockerfile.coordinator
* Provides orchestration for ML workflows via Argo Workflows
* Handles resource allocation, quality assessment, and caching

packages/coordinator/Dockerfile.coordinator

#### ML Base Image

* New centralized base image for all ML services
* Built from Dockerfile.ml-base in the repository root
* Provides consistent environment for ML workers
* Contains common dependencies and infrastructure

Dockerfile.ml-base

#### Worker Images

Each worker image is specialized for a specific task in the ML pipeline, now built using one of two centralized base images:

##### GPU-Based ML Services

These services inherit from the kai-ml-base image (built from Dockerfile.ml-base):  
- kai-quality-assessment: Assesses image quality and determines processing level  
- kai-image-preprocessing: Performs initial image preparation  
- kai-colmap-sfm: Runs Structure from Motion using COLMAP  
- kai-point-cloud: Generates point clouds from camera poses  
- kai-model-generator: Creates 3D models from point clouds or camera poses  
- kai-diffusion-nerf: Implements NeRF-based reconstruction  
- kai-nerf-mesh-extractor: Extracts mesh data from NeRF models  
- kai-format-converter: Converts models to different formats

kai-ml-base

Dockerfile.ml-base

kai-quality-assessment

kai-image-preprocessing

kai-colmap-sfm

kai-point-cloud

kai-model-generator

kai-diffusion-nerf

kai-nerf-mesh-extractor

kai-format-converter

##### Non-GPU Python Services

These services inherit from the kai-python-base image (built from Dockerfile.python-base):  
- kai-workflow-finalizer: Handles notifications and final cleanup

kai-python-base

Dockerfile.python-base

kai-workflow-finalizer

This dual base image strategy ensures optimal resource utilization, with GPU-dependent services using the TensorFlow GPU base and lightweight services using a simpler Python base image.

Each worker image is built from its respective Dockerfile in packages/ml/python/ with significantly reduced size and complexity due to the use of centralized base images.

packages/ml/python/

### Automated Canary Deployments

The CI/CD pipeline now includes support for automated canary deployments with health monitoring and automated rollback capabilities:

#### Canary Deployment Implementation

The production deployment workflow (deploy-production.yml) has been enhanced with canary deployment support:

deploy-production.yml

on:  
 workflow\_call:  
 inputs:  
 sha:  
 required: true  
 type: string  
 canary:  
 description: 'Whether to deploy as a canary release'  
 required: false  
 type: boolean  
 default: false  
 canary\_weight:  
 description: 'Percentage of traffic to route to canary'  
 required: false  
 type: number  
 default: 20

on:  
 workflow\_call:  
 inputs:  
 sha:  
 required: true  
 type: string  
 canary:  
 description: 'Whether to deploy as a canary release'  
 required: false  
 type: boolean  
 default: false  
 canary\_weight:  
 description: 'Percentage of traffic to route to canary'  
 required: false  
 type: number  
 default: 20

This allows the workflow to be triggered with canary deployment parameters:

deploy-production:  
 uses: ./.github/workflows/deploy-production.yml  
 with:  
 sha: ${{ github.sha }}  
 canary: true  
 canary\_weight: 10

deploy-production:  
 uses: ./.github/workflows/deploy-production.yml  
 with:  
 sha: ${{ github.sha }}  
 canary: true  
 canary\_weight: 10

#### Helm Chart Integration

The Helm deployment script (helm-charts/helm-deploy.sh) has been updated to support canary deployments with the following parameters:

helm-charts/helm-deploy.sh

* --canary: Enables canary deployment mode
* --canary-weight: Percentage of traffic to route to the canary (default: 20%)
* --health-threshold: Number of failures before marking deployment as degraded
* --critical-services: Comma-separated list of services to monitor for health

--canary

--canary-weight

--health-threshold

--critical-services

The Kai Helm chart's values.yaml has been updated with canary configuration:

canary:  
 enabled: false  
 weight: 20  
 maxTimeMinutes: 30  
 healthThreshold: 5  
 criticalServices:  
 - api-server  
 - coordinator-service

canary:  
 enabled: false  
 weight: 20  
 maxTimeMinutes: 30  
 healthThreshold: 5  
 criticalServices:  
 - api-server  
 - coordinator-service

#### Health Monitoring and Automated Promotion/Rollback

The canary deployment process includes automated health monitoring:

1. Initial Deployment: The canary version is deployed alongside the stable version
2. Traffic Splitting: Traffic is split between stable and canary according to the weight
3. Health Monitoring: The process continuously monitors the health of critical services
4. Automated Decision:
5. If health checks succeed throughout the monitoring period, the canary is automatically promoted to stable
6. If too many health checks fail, the canary is automatically rolled back

The workflow implementation includes:

- name: Monitor Deployment Health  
 id: health\_check  
 run: |  
 echo "Monitoring production deployment health for 5 minutes..."  
 FAILURES=0  
  
 for i in {1..30}; do  
 # Health check logic for critical services  
 # ...  
  
 if [ $FAILURES -ge 5 ]; then  
 echo "health\_status=degraded" >> $GITHUB\_OUTPUT  
 break  
 fi  
  
 sleep 10  
 done  
  
- name: Rollback if Needed  
 if: steps.health\_check.outputs.health\_status == 'degraded' && inputs.canary == 'true'  
 run: |  
 echo "::warning::Production health checks failed, rolling back canary deployment"  
 ./helm-charts/helm-deploy.sh \  
 --context=${{ env.KUBE\_CONTEXT }} \  
 --env=production \  
 --release=kai-production \  
 --rollback

- name: Monitor Deployment Health  
 id: health\_check  
 run: |  
 echo "Monitoring production deployment health for 5 minutes..."  
 FAILURES=0  
  
 for i in {1..30}; do  
 # Health check logic for critical services  
 # ...  
  
 if [ $FAILURES -ge 5 ]; then  
 echo "health\_status=degraded" >> $GITHUB\_OUTPUT  
 break  
 fi  
  
 sleep 10  
 done  
  
- name: Rollback if Needed  
 if: steps.health\_check.outputs.health\_status == 'degraded' && inputs.canary == 'true'  
 run: |  
 echo "::warning::Production health checks failed, rolling back canary deployment"  
 ./helm-charts/helm-deploy.sh \  
 --context=${{ env.KUBE\_CONTEXT }} \  
 --env=production \  
 --release=kai-production \  
 --rollback

#### Benefits of Automated Canary Deployments

This implementation provides several key advantages:

1. Reduced Deployment Risk: Only a small percentage of traffic is initially exposed to new versions
2. Early Problem Detection: Issues are identified with minimal user impact
3. Automated Verification: Health monitoring runs automatically without human intervention
4. Safety Net: Automatic rollback prevents prolonged service degradation
5. Configurable Parameters: Deployment teams can adjust canary settings based on risk tolerance

#### Usage in Workflow Dispatch

The canary deployment can be triggered manually via workflow dispatch:

workflow\_dispatch:  
 inputs:  
 environment:  
 description: 'Environment to deploy to'  
 required: true  
 default: 'staging'  
 type: choice  
 options:  
 - staging  
 - production  
 canary:  
 description: 'Use canary deployment (production only)'  
 required: false  
 default: false  
 type: boolean  
 canary\_weight:  
 description: 'Percentage of traffic to canary (1-50)'  
 required: false  
 default: '20'  
 type: string

workflow\_dispatch:  
 inputs:  
 environment:  
 description: 'Environment to deploy to'  
 required: true  
 default: 'staging'  
 type: choice  
 options:  
 - staging  
 - production  
 canary:  
 description: 'Use canary deployment (production only)'  
 required: false  
 default: false  
 type: boolean  
 canary\_weight:  
 description: 'Percentage of traffic to canary (1-50)'  
 required: false  
 default: '20'  
 type: string

This allows operations teams to make informed decisions about canary deployments on a case-by-case basis.

### Kubernetes Deployment with Helm Charts

The Kubernetes deployment now uses Helm charts for improved maintainability and consistency:

1. Helm Chart Structure: Organized as a parent chart with subcharts for components:  
    helm-charts/  
    ├── kai/ # Main parent chart  
    │ ├── Chart.yaml # Chart metadata with dependencies  
    │ ├── values.yaml # Default values  
    │ ├── values-staging.yaml # Staging environment values  
    │ └── values-production.yaml # Production environment values  
    └── coordinator/ # Example subchart  
    ├── Chart.yaml  
    ├── values.yaml  
    └── templates/ # Resource templates  
    ├── \_helpers.tpl # Reusable template snippets  
    ├── deployment.yaml # Deployment template  
    ├── service.yaml # Service template  
    ├── hpa.yaml # Autoscaling template  
    ├── pdb.yaml # Pod Disruption Budget template  
    ├── rbac.yaml # RBAC resources template  
    └── configmap.yaml # ConfigMap template
2. Environment-Specific Configuration: All environment differences are now managed in dedicated values files rather than script variables:
3. values.yaml contains default configurations
4. values-staging.yaml overrides for staging environment
5. values-production.yaml overrides for production environment
6. Deployment Process: The deployment uses the helm-charts/helm-deploy.sh script which:
7. Supports the same parameters as the previous script (--registry, --tag, --context)
8. Adds Helm-specific parameters (--release for release naming)
9. Provides enhanced rollback capability with version history
10. Enables more fine-grained control over deployment updates
11. Includes built-in deployment verification
12. CI/CD Integration: The GitHub Actions workflows use the Helm deployment script for both environments:  
     ```yaml
13. name: Deploy to Kubernetes with Helm  
     run: |  
     ./helm-charts/helm-deploy.sh \  
     --context=${{ inputs.kube-context }} \  
     --registry=${{ secrets.docker\_registry }}/${{ secrets.docker\_username }} \  
     --tag=${{ inputs.sha }} \  
     --env=${{ env.DEPLOY\_ENV }} \  
     --release=kai-${{ env.DEPLOY\_ENV }}  
     ```
14. Improved Rollback: The Helm-based rollback mechanism provides:
15. Versioned releases for deterministic rollbacks
16. Ability to roll back to any previous version, not just the last one
17. Comprehensive rollback including all related resources
18. Detailed history for auditing deployment changes

Helm Chart Structure: Organized as a parent chart with subcharts for components:  
 helm-charts/  
 ├── kai/ # Main parent chart  
 │ ├── Chart.yaml # Chart metadata with dependencies  
 │ ├── values.yaml # Default values  
 │ ├── values-staging.yaml # Staging environment values  
 │ └── values-production.yaml # Production environment values  
 └── coordinator/ # Example subchart  
 ├── Chart.yaml  
 ├── values.yaml  
 └── templates/ # Resource templates  
 ├── \_helpers.tpl # Reusable template snippets  
 ├── deployment.yaml # Deployment template  
 ├── service.yaml # Service template  
 ├── hpa.yaml # Autoscaling template  
 ├── pdb.yaml # Pod Disruption Budget template  
 ├── rbac.yaml # RBAC resources template  
 └── configmap.yaml # ConfigMap template

helm-charts/  
 ├── kai/ # Main parent chart  
 │ ├── Chart.yaml # Chart metadata with dependencies  
 │ ├── values.yaml # Default values  
 │ ├── values-staging.yaml # Staging environment values  
 │ └── values-production.yaml # Production environment values  
 └── coordinator/ # Example subchart  
 ├── Chart.yaml  
 ├── values.yaml  
 └── templates/ # Resource templates  
 ├── \_helpers.tpl # Reusable template snippets  
 ├── deployment.yaml # Deployment template  
 ├── service.yaml # Service template  
 ├── hpa.yaml # Autoscaling template  
 ├── pdb.yaml # Pod Disruption Budget template  
 ├── rbac.yaml # RBAC resources template  
 └── configmap.yaml # ConfigMap template

Environment-Specific Configuration: All environment differences are now managed in dedicated values files rather than script variables:

values.yaml

values-staging.yaml

values-production.yaml overrides for production environment

values-production.yaml

Deployment Process: The deployment uses the helm-charts/helm-deploy.sh script which:

helm-charts/helm-deploy.sh

--registry

--tag

--context

--release

Includes built-in deployment verification

CI/CD Integration: The GitHub Actions workflows use the Helm deployment script for both environments:  
 ```yaml

name: Deploy to Kubernetes with Helm  
 run: |  
 ./helm-charts/helm-deploy.sh \  
 --context=${{ inputs.kube-context }} \  
 --registry=${{ secrets.docker\_registry }}/${{ secrets.docker\_username }} \  
 --tag=${{ inputs.sha }} \  
 --env=${{ env.DEPLOY\_ENV }} \  
 --release=kai-${{ env.DEPLOY\_ENV }}  
 ```

Improved Rollback: The Helm-based rollback mechanism provides:

### Environment Variables and Secrets

The CI/CD pipeline uses a variety of environment variables and secrets to configure the deployment process. These are defined in the GitHub repository secrets and used across the workflows.

#### Environment Variables

These environment variables are commonly used across workflows:

Table content:

Variable | Description | Example Value

DEPLOY\_ENV | Target deployment environment | staging or production

TAG\_SUFFIX | Suffix for Docker image tags | staging or latest

VERCEL\_ARGS | Arguments for Vercel deployments | --prod

API\_URL | URL for the API service | https://api.kai.example.com

TEST\_SCRIPT | Script to run for tests | test:e2e

KUBE\_CONTEXT | Kubernetes context to use | kubernetes-cluster1

DEPLOY\_ENV

staging

production

TAG\_SUFFIX

staging

latest

VERCEL\_ARGS

--prod

API\_URL

https://api.kai.example.com

TEST\_SCRIPT

test:e2e

KUBE\_CONTEXT

kubernetes-cluster1

#### Required Secrets

The following secrets should be configured in your GitHub repository:

##### Docker Registry

* DOCKER\_USERNAME: Username for Docker registry
* DOCKER\_PASSWORD: Password for Docker registry
* DOCKER\_REGISTRY: Docker registry URL

DOCKER\_USERNAME

DOCKER\_PASSWORD

DOCKER\_REGISTRY

##### Kubernetes

* DIGITALOCEAN\_ACCESS\_TOKEN: Digital Ocean API token
* CLUSTER\_NAME: Base name for Kubernetes cluster (e.g., "kai")
* DO\_REGION: Digital Ocean region (e.g., "ams3")
* KUBE\_CONFIG\_DATA: Base64-encoded Kubernetes config (if not using DO provisioning)

DIGITALOCEAN\_ACCESS\_TOKEN

CLUSTER\_NAME

DO\_REGION

KUBE\_CONFIG\_DATA

##### Application

* DOMAIN\_NAME: Domain name for the application (e.g., "kai-platform.com")
* ADMIN\_EMAIL: Email for SSL certificate notifications
* MONGODB\_URI: MongoDB connection string
* JWT\_SECRET: Secret for JWT tokens
* OPENAI\_API\_KEY: OpenAI API key
* REDIS\_PASSWORD: Password for Redis

DOMAIN\_NAME

ADMIN\_EMAIL

MONGODB\_URI

JWT\_SECRET

OPENAI\_API\_KEY

REDIS\_PASSWORD

##### Supabase

* SUPABASE\_URL\_PRODUCTION, SUPABASE\_KEY\_PRODUCTION, SUPABASE\_ANON\_KEY\_PRODUCTION: Supabase production credentials
* SUPABASE\_URL\_STAGING, SUPABASE\_KEY\_STAGING, SUPABASE\_ANON\_KEY\_STAGING: Supabase staging credentials

SUPABASE\_URL\_PRODUCTION

SUPABASE\_KEY\_PRODUCTION

SUPABASE\_ANON\_KEY\_PRODUCTION

SUPABASE\_URL\_STAGING

SUPABASE\_KEY\_STAGING

SUPABASE\_ANON\_KEY\_STAGING

##### Payment Processing

* STRIPE\_SECRET\_KEY, STRIPE\_PUBLISHABLE\_KEY: Stripe payment credentials

STRIPE\_SECRET\_KEY

STRIPE\_PUBLISHABLE\_KEY

##### Frontend Deployment

* VERCEL\_TOKEN, VERCEL\_ORG\_ID: Vercel credentials
* VERCEL\_PROJECT\_ID\_CLIENT, VERCEL\_PROJECT\_ID\_ADMIN: Vercel project IDs

VERCEL\_TOKEN

VERCEL\_ORG\_ID

VERCEL\_PROJECT\_ID\_CLIENT

VERCEL\_PROJECT\_ID\_ADMIN

##### Notifications

* SLACK\_WEBHOOK: Slack notification webhook URL

SLACK\_WEBHOOK

### Removed Legacy Workflows and Scripts

As the CI/CD system has evolved, several workflows and scripts have been removed in favor of more modern approaches:

#### Removed Workflows

* workflow-env.yml: This was a reference file that documented environment variables but never actually ran. This information has been moved to this documentation file.
* dependency-scanner.yml: This workflow has been replaced by the more advanced dependency-update-testing.yml which includes additional features like creating PRs for safe updates and running targeted tests.

workflow-env.yml: This was a reference file that documented environment variables but never actually ran. This information has been moved to this documentation file.

dependency-scanner.yml: This workflow has been replaced by the more advanced dependency-update-testing.yml which includes additional features like creating PRs for safe updates and running targeted tests.

dependency-update-testing.yml

#### Removed Scripts

The following scripts have been removed as they are no longer needed with the current GitOps approach:

##### General Deployment Scripts

* helm-charts/helm-deploy.sh: Removed in favor of Flux GitOps

##### RAG System Deployment Scripts

The following scripts for the RAG system deployment have been removed as they are now superseded by the GitHub Actions workflow in .github/workflows/enhanced-rag.yml:

.github/workflows/enhanced-rag.yml

* deploy-rag.sh: Manual deployment script for the RAG system
* rag-deployment-pipeline.sh: Orchestrates the RAG deployment process
* build-push-rag.sh: Builds and pushes RAG Docker images
* verify-rag-deployment.sh: Verifies the RAG deployment
* monitor-rag-performance.sh: Monitors RAG system performance
* monitor-rag-api.sh: Monitors RAG API performance

#### Current Deployment Approach

All deployments are now handled through the GitOps approach with Flux. In case of emergency where the GitOps approach is not working, the deployment can be performed manually using kubectl and Helm commands directly, following the patterns established in the CI/CD workflows.

### Flux GitOps Integration

The CI/CD pipeline now integrates with Flux CD, providing a GitOps approach to Kubernetes deployments:

#### GitOps Repository Structure

flux/  
├── clusters/  
│ ├── staging/  
│ │ ├── flux-system/  
│ │ │ ├── gotk-sync.yaml # Flux synchronization configuration  
│ │ │ └── kustomization.yaml # Flux system components  
│ │ ├── sources/  
│ │ │ ├── helm-repository.yaml # Helm chart repository definition  
│ │ │ └── kustomization.yaml # Sources kustomization  
│ │ ├── releases/  
│ │ │ ├── coordinator.yaml # HelmRelease for coordinator service  
│ │ │ └── kustomization.yaml # Releases kustomization  
│ │ └── kustomization.yaml # Main kustomization including all components  
│ └── production/  
│ └── (Similar structure as staging)

flux/  
├── clusters/  
│ ├── staging/  
│ │ ├── flux-system/  
│ │ │ ├── gotk-sync.yaml # Flux synchronization configuration  
│ │ │ └── kustomization.yaml # Flux system components  
│ │ ├── sources/  
│ │ │ ├── helm-repository.yaml # Helm chart repository definition  
│ │ │ └── kustomization.yaml # Sources kustomization  
│ │ ├── releases/  
│ │ │ ├── coordinator.yaml # HelmRelease for coordinator service  
│ │ │ └── kustomization.yaml # Releases kustomization  
│ │ └── kustomization.yaml # Main kustomization including all components  
│ └── production/  
│ └── (Similar structure as staging)

#### CI/CD Workflow Integration

The deployment workflows include jobs that update the GitOps repository with new image versions:

update-gitops:  
 name: Update GitOps Repository  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout GitOps repository  
 # Checkout the GitOps repository...  
  
 - name: Update image tags in HelmReleases  
 # Update image tags and commit changes...

update-gitops:  
 name: Update GitOps Repository  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout GitOps repository  
 # Checkout the GitOps repository...  
  
 - name: Update image tags in HelmReleases  
 # Update image tags and commit changes...

This job:  
1. Determines the target environment (staging or production)  
2. Checks out the GitOps repository at the appropriate branch  
3. Updates image tags in the HelmRelease resources  
4. Commits and pushes changes to the GitOps repository

#### Flux Automation

Flux operates by continuously monitoring the GitOps repository:

1. Source Controller: Watches the Git repository for changes
2. Kustomize Controller: Applies Kubernetes resources defined via kustomize
3. Helm Controller: Manages Helm releases based on HelmRelease resources
4. Notification Controller: Provides alerts and notifications for events

When the CI/CD pipeline updates image tags in the GitOps repository, Flux automatically:  
1. Detects the changes via the Source Controller  
2. Processes the updated HelmRelease resources  
3. Deploys the new image versions to the cluster  
4. Reports status via the Notification Controller

#### Benefits of Flux GitOps

The Flux GitOps approach provides several advantages:

1. Declarative Configuration: All Kubernetes resources are defined as code in the GitOps repository
2. Automated Reconciliation: Flux continuously ensures the cluster state matches the desired state in Git
3. Audit Trail: All changes are tracked in Git with commit history and authorship
4. Self-Healing: Flux automatically recovers from drift by reapplying the desired state
5. Enhanced Security: Reduced need for direct cluster access; changes go through Git
6. Progressive Delivery: Support for canary deployments and A/B testing
7. Multi-Cluster Management: Simplified management of resources across multiple clusters
8. Simplified Rollbacks: Reverting to a previous state is as simple as reverting a Git commit

### Required GitHub Secrets

The workflow requires the following secrets to be set in your GitHub repository:

#### Vercel Deployment Secrets

* VERCEL\_TOKEN: API token from Vercel
* VERCEL\_ORG\_ID: Your Vercel organization ID
* VERCEL\_PROJECT\_ID\_CLIENT: Project ID for the client application
* VERCEL\_PROJECT\_ID\_ADMIN: Project ID for the admin application

VERCEL\_TOKEN

VERCEL\_ORG\_ID

VERCEL\_PROJECT\_ID\_CLIENT

VERCEL\_PROJECT\_ID\_ADMIN

#### GitOps Repository Access

* GITOPS\_PAT: Personal access token for the GitOps repository

GITOPS\_PAT

#### Digital Ocean Kubernetes Secrets

* KUBE\_CONFIG\_DATA: Base64-encoded kubeconfig file for your Kubernetes cluster

KUBE\_CONFIG\_DATA

#### GitHub Container Registry

* GITHUB\_TOKEN: GitHub token with write:packages permission (automatically provided by GitHub Actions)

GITHUB\_TOKEN

write:packages

#### Database and API Secrets

* SUPABASE\_URL\_STAGING: Supabase project URL for staging environment migrations
* SUPABASE\_KEY\_STAGING: Supabase service role key for staging environment migrations
* SUPABASE\_URL\_PRODUCTION: Supabase project URL for production environment migrations
* SUPABASE\_KEY\_PRODUCTION: Supabase service role key for production environment migrations

SUPABASE\_URL\_STAGING

SUPABASE\_KEY\_STAGING

SUPABASE\_URL\_PRODUCTION

SUPABASE\_KEY\_PRODUCTION

### Adding GitHub Secrets

To add these secrets to your repository:

1. Go to your GitHub repository → Settings → Secrets and variables → Actions
2. Click "New repository secret"
3. Enter the secret name (e.g., VERCEL\_TOKEN) and value
4. Click "Add secret"
5. Repeat for all required secrets

VERCEL\_TOKEN

### Customizing the Workflow

To customize the workflow for your specific needs:

1. Modify the core reusable workflow files for global changes:
2. .github/workflows/build-test.yml - Build and test process
3. .github/workflows/docker-build.yml - Docker image building
4. .github/workflows/deploy-staging.yml - Staging deployment
5. .github/workflows/deploy-production.yml - Production deployment
6. For environment-specific changes:
7. Update only the relevant environment workflow
8. Keep shared logic in the common workflow files
9. To add a new environment:
10. Create a new environment-specific deployment workflow file
11. Update the main workflow to call this new workflow with appropriate conditions

.github/workflows/build-test.yml

.github/workflows/docker-build.yml

.github/workflows/deploy-staging.yml

.github/workflows/deploy-production.yml - Production deployment

.github/workflows/deploy-production.yml

For environment-specific changes:

Keep shared logic in the common workflow files

To add a new environment:

### Troubleshooting

If you encounter issues with the pipeline:

1. Check the Actions tab in your GitHub repository to see detailed logs
2. Look at the specific workflow run that failed to identify the problem
3. For workflow-specific issues, check the individual reusable workflow files
4. Verify that all required secrets are properly configured
5. Ensure branch protection rules are set correctly

#### Troubleshooting Kubernetes Deployment

For Kubernetes-specific issues:

1. Check pod status and logs:  
    bash  
    kubectl get pods -n kai-ml  
    kubectl logs <pod-name> -n kai-ml
2. Check Argo Workflows:  
    bash  
    kubectl get workflows -n kai-ml  
    kubectl get workflowtemplates -n kai-ml
3. View the Coordinator service logs:  
    bash  
    kubectl logs -l app=coordinator-service -n kai-ml
4. Check for configuration issues:  
    bash  
    kubectl get configmaps -n kai-ml  
    kubectl get secrets -n kai-ml

Check pod status and logs:  
 bash  
 kubectl get pods -n kai-ml  
 kubectl logs <pod-name> -n kai-ml

bash  
 kubectl get pods -n kai-ml  
 kubectl logs <pod-name> -n kai-ml

Check Argo Workflows:  
 bash  
 kubectl get workflows -n kai-ml  
 kubectl get workflowtemplates -n kai-ml

bash  
 kubectl get workflows -n kai-ml  
 kubectl get workflowtemplates -n kai-ml

View the Coordinator service logs:  
 bash  
 kubectl logs -l app=coordinator-service -n kai-ml

bash  
 kubectl logs -l app=coordinator-service -n kai-ml

Check for configuration issues:  
 bash  
 kubectl get configmaps -n kai-ml  
 kubectl get secrets -n kai-ml

bash  
 kubectl get configmaps -n kai-ml  
 kubectl get secrets -n kai-ml

The Coordinator service provides detailed logs about workflow submissions and their status, which is the first place to look when troubleshooting ML pipeline issues.

# Client Heroui Integration

Source: readme/client-heroui-integration.md

---

## HeroUI Integration Guide

This document outlines the integration of HeroUI into the Kai application. HeroUI is being used for both frontend and backend UI components to provide a consistent design system across the application.

### Overview

HeroUI has been integrated into the project to standardize the UI components and provide a consistent look and feel. The integration is set up to:

1. Allow gradual adoption (existing components will be updated later)
2. Ensure new components use HeroUI from the start
3. Maintain a consistent theme across the application

### Files and Structure

The HeroUI integration consists of the following files:

* src/theme/heroui-theme.ts - Contains the HeroUI theme configuration
* src/providers/HeroUIProvider.tsx - Provider component that wraps the application with HeroUI theme
* gatsby-browser.js and gatsby-ssr.js - Wrap the application with the HeroUIProvider
* src/components/examples/HeroUIExample.tsx - Example component showcasing HeroUI usage

src/theme/heroui-theme.ts

src/providers/HeroUIProvider.tsx

gatsby-browser.js

gatsby-ssr.js

src/components/examples/HeroUIExample.tsx

### Usage Guidelines

#### For New Components

All new components should use HeroUI components and styling. Here's a basic example:

import React from 'react';  
import { Button, Card, TextField } from '@heroui/react';  
  
const MyNewComponent: React.FC = () => {  
 return (  
 <Card>  
 <h2>My New Component</h2>  
 <TextField label="Example Input" />  
 <Button variant="contained">Submit</Button>  
 </Card>  
 );  
};  
  
export default MyNewComponent;

import React from 'react';  
import { Button, Card, TextField } from '@heroui/react';  
  
const MyNewComponent: React.FC = () => {  
 return (  
 <Card>  
 <h2>My New Component</h2>  
 <TextField label="Example Input" />  
 <Button variant="contained">Submit</Button>  
 </Card>  
 );  
};  
  
export default MyNewComponent;

#### Theme Access

You can access the HeroUI theme in your components:

import React from 'react';  
import { useTheme } from '@heroui/react';  
  
const MyComponent: React.FC = () => {  
 const theme = useTheme();  
  
 return (  
 <div style={{ color: theme.colors.primary.main }}>  
 Themed content  
 </div>  
 );  
};

import React from 'react';  
import { useTheme } from '@heroui/react';  
  
const MyComponent: React.FC = () => {  
 const theme = useTheme();  
  
 return (  
 <div style={{ color: theme.colors.primary.main }}>  
 Themed content  
 </div>  
 );  
};

#### Common Components

Here are some commonly used HeroUI components:

* Button - For actions and triggers
* Card - For containing related content
* TextField - For text input
* Select - For dropdown selection
* Alert - For notifications and messages
* Modal - For dialogs and popups
* Table - For tabular data

Button

Card

TextField

Select

Alert

Modal

Table

#### Existing Components

Existing components will continue to use their current styling (Material-UI or styled-components) until they are updated to use HeroUI. Do not modify existing component styling unless specifically tasked with updating them to use HeroUI.

### Backend Integration

For backend admin interfaces, HeroUI is also available via the @heroui/node package. Backend templates should follow the same design system for consistency.

@heroui/node

### Best Practices

1. Consistency: Follow the HeroUI component patterns for consistent user experience
2. Theme Variables: Use theme variables instead of hardcoded values for colors, spacing, etc.
3. Responsive Design: Ensure components work across different screen sizes
4. Accessibility: Maintain accessibility standards when using HeroUI components

### Reference Example

See src/components/examples/HeroUIExample.tsx for a reference implementation showing how HeroUI components and styling should be used.

src/components/examples/HeroUIExample.tsx

### Resources

* HeroUI Documentation - Official documentation for HeroUI components and API
* Theme Customization - How to customize the HeroUI theme

# Client Unified Services Direct

Source: readme/client-unified-services-direct.md

---

## Client Package Direct Unified Services

This document describes the implementation of the direct unified services architecture in the client package. The direct unified services architecture completely removes backward compatibility layers and uses the unified services directly.

### Implementation Overview

The client package has been updated to use the unified services from the shared package directly. The following changes were made:

1. Removed compatibility layers:
2. Removed unifiedAuthAdapter.ts
3. Removed unifiedApiAdapter.ts
4. Removed unifiedLoggerAdapter.ts
5. Removed apiClient.ts
6. Removed logger.ts
7. Removed auth.service.ts
8. Removed supabaseAuth.service.ts
9. Removed supabaseClient.ts
10. Created a unified services export:
11. Created services/index.ts to export all unified services from the shared package
12. Updated components to use unified services directly:
13. Updated UserProvider.tsx to use the unified auth service directly
14. Updated gatsby-browser.js to initialize the unified services

unifiedAuthAdapter.ts

unifiedApiAdapter.ts

unifiedLoggerAdapter.ts

apiClient.ts

logger.ts

auth.service.ts

supabaseAuth.service.ts

Removed supabaseClient.ts

supabaseClient.ts

Created a unified services export:

Created services/index.ts to export all unified services from the shared package

services/index.ts

Updated components to use unified services directly:

UserProvider.tsx

gatsby-browser.js

### Unified Services Export

The services/index.ts file exports all the unified services from the shared package for use throughout the client package. It also provides a function to initialize all services.

services/index.ts

/\*\*  
 \* Unified Services Export  
 \*   
 \* This file exports all the unified services from the shared package  
 \* for use throughout the client package.  
 \*/  
  
// Export auth service  
export {   
 auth,   
 initializeAuth,   
 User,   
 LoginCredentials,   
 RegisterCredentials,   
 AuthResult   
} from '@kai/shared';  
  
// Export API client  
export {   
 apiClient,   
 createApiClient,   
 ApiClientConfig,   
 ApiError   
} from '@kai/shared';  
  
// Export MCP client  
export {   
 mcpClient,   
 createMCPClient   
} from '@kai/shared';  
  
// Export Supabase client  
export {   
 supabase   
} from '@kai/shared';  
  
// Export storage service  
export {   
 storage,   
 initializeStorage,   
 StorageProvider   
} from '@kai/shared';  
  
// Export logger  
export {   
 createLogger,   
 LogLevel   
} from '@kai/shared';  
  
// Export config  
export {   
 config   
} from '@kai/shared';  
  
/\*\*  
 \* Initialize all services  
 \*   
 \* This function initializes all the unified services.  
 \* It should be called early in the application lifecycle.  
 \*/  
export function initializeServices(): void {  
 // Initialize auth service  
 initializeAuth();  
  
 // Initialize storage service  
 initializeStorage();  
  
 // Log that services have been initialized  
 const logger = createLogger('Services');  
 logger.info('Unified services initialized');  
}

/\*\*  
 \* Unified Services Export  
 \*   
 \* This file exports all the unified services from the shared package  
 \* for use throughout the client package.  
 \*/  
  
// Export auth service  
export {   
 auth,   
 initializeAuth,   
 User,   
 LoginCredentials,   
 RegisterCredentials,   
 AuthResult   
} from '@kai/shared';  
  
// Export API client  
export {   
 apiClient,   
 createApiClient,   
 ApiClientConfig,   
 ApiError   
} from '@kai/shared';  
  
// Export MCP client  
export {   
 mcpClient,   
 createMCPClient   
} from '@kai/shared';  
  
// Export Supabase client  
export {   
 supabase   
} from '@kai/shared';  
  
// Export storage service  
export {   
 storage,   
 initializeStorage,   
 StorageProvider   
} from '@kai/shared';  
  
// Export logger  
export {   
 createLogger,   
 LogLevel   
} from '@kai/shared';  
  
// Export config  
export {   
 config   
} from '@kai/shared';  
  
/\*\*  
 \* Initialize all services  
 \*   
 \* This function initializes all the unified services.  
 \* It should be called early in the application lifecycle.  
 \*/  
export function initializeServices(): void {  
 // Initialize auth service  
 initializeAuth();  
  
 // Initialize storage service  
 initializeStorage();  
  
 // Log that services have been initialized  
 const logger = createLogger('Services');  
 logger.info('Unified services initialized');  
}

### UserProvider Component

The UserProvider.tsx component has been updated to use the unified auth service directly. It now imports the auth service from the shared package and uses it for all authentication operations.

UserProvider.tsx

Key changes:  
- Imports the unified auth service directly from the shared package  
- Uses the unified auth service for login, register, logout, and profile update operations  
- Converts the unified auth user to the client's user profile format

### Gatsby Browser

The gatsby-browser.js file has been updated to initialize the unified services before the application renders. It calls the initializeServices function from the services/index.ts file.

gatsby-browser.js

initializeServices

services/index.ts

/\*\*  
 \* Initialize services before the browser renders  
 \*/  
export const onClientEntry = () => {  
 // Initialize all unified services  
 initializeServices();  
};

/\*\*  
 \* Initialize services before the browser renders  
 \*/  
export const onClientEntry = () => {  
 // Initialize all unified services  
 initializeServices();  
};

### Benefits

The direct unified services architecture provides several benefits:

1. Simplified codebase: No more compatibility layers or adapter files
2. Reduced code duplication: Common functionality is implemented once in the shared package
3. Improved maintainability: Changes to common functionality only need to be made in one place
4. Consistent behavior: All parts of the application use the same implementation of common functionality
5. Type safety: The unified services provide type-safe interfaces for common operations
6. Extensibility: The provider pattern allows adding new implementations without changing client code

### Next Steps

The following steps are recommended to further improve the unified services architecture:

1. Update the agents package to use the unified services directly
2. Update the ml package to use the unified services directly
3. Add more storage providers (Google Cloud Storage, Azure Blob Storage, etc.)
4. Add more authentication providers (SAML, etc.)
5. Implement caching mechanisms for improved performance
6. Add more comprehensive monitoring and telemetry

# Client Unified Services

Source: readme/client-unified-services.md

---

## Client Package Unified Services

This document describes the implementation of the unified services architecture in the client package. The unified services architecture consolidates duplicate implementations of common functionality into a single, shared implementation to improve maintainability and ensure consistent behavior.

### Implementation Overview

The client package has been updated to use the unified services from the shared package. The following changes were made:

1. Created adapter files to provide a compatibility layer between the client package's existing code and the unified services:
2. unifiedAuthAdapter.ts - Adapts the unified auth service to the client's needs
3. unifiedApiAdapter.ts - Adapts the unified API client to the client's needs
4. unifiedLoggerAdapter.ts - Adapts the unified logger to the client's needs
5. Updated existing files to use the unified services through the adapters:
6. auth.service.ts - Updated to use the unified auth service
7. supabaseAuth.service.ts - Updated to use the unified auth service
8. apiClient.ts - Updated to use the unified API client
9. logger.ts - Updated to use the unified logger

unifiedAuthAdapter.ts

unifiedApiAdapter.ts

unifiedLoggerAdapter.ts - Adapts the unified logger to the client's needs

unifiedLoggerAdapter.ts

Updated existing files to use the unified services through the adapters:

auth.service.ts

supabaseAuth.service.ts

apiClient.ts

logger.ts

### Adapter Files

#### Unified Auth Adapter

The unifiedAuthAdapter.ts file provides a compatibility layer for the unified authentication service. It adapts the unified auth service to match the interface expected by the client package.

unifiedAuthAdapter.ts

Key features:  
- Exposes the same interface as the old Supabase auth service  
- Uses the unified auth service under the hood  
- Handles token management and authentication state  
- Provides backward compatibility for existing code

#### Unified API Client Adapter

The unifiedApiAdapter.ts file provides a compatibility layer for the unified API client. It adapts the unified API client to match the interface expected by the client package.

unifiedApiAdapter.ts

Key features:  
- Exposes an Axios-compatible interface for backward compatibility  
- Uses the unified API client under the hood  
- Handles error conversion between the unified API client and Axios  
- Provides backward compatibility for existing code

#### Unified Logger Adapter

The unifiedLoggerAdapter.ts file provides a compatibility layer for the unified logger. It adapts the unified logger to match the interface expected by the client package.

unifiedLoggerAdapter.ts

Key features:  
- Exposes the same interface as the old client logger  
- Uses the unified logger under the hood  
- Adds client-specific features like performance timing  
- Provides backward compatibility for existing code

### Updated Files

#### auth.service.ts

The auth.service.ts file has been updated to use the unified auth service through the adapter. It maintains the same interface for backward compatibility.

auth.service.ts

Key changes:  
- Imports the unified auth service from the adapter  
- Uses the unified auth service for authentication operations  
- Maintains the same interface for backward compatibility

#### supabaseAuth.service.ts

The supabaseAuth.service.ts file has been updated to use the unified auth service through the adapter. It maintains the same interface for backward compatibility.

supabaseAuth.service.ts

Key changes:  
- Imports the unified auth service from the adapter  
- Uses the unified auth service for Supabase-specific authentication operations  
- Maintains the same interface for backward compatibility

#### apiClient.ts

The apiClient.ts file has been updated to use the unified API client through the adapter. It maintains the same interface for backward compatibility.

apiClient.ts

Key changes:  
- Imports the unified API client from the adapter  
- Exports an Axios-compatible interface for backward compatibility  
- Maintains the same interface for backward compatibility

#### logger.ts

The logger.ts file has been updated to use the unified logger through the adapter. It maintains the same interface for backward compatibility.

logger.ts

Key changes:  
- Imports the unified logger from the adapter  
- Re-exports everything from the adapter  
- Maintains the same interface for backward compatibility

### Benefits

The unified services architecture provides several benefits for the client package:

1. Reduced code duplication: Common functionality is implemented once in the shared package.
2. Improved maintainability: Changes to common functionality only need to be made in one place.
3. Consistent behavior: All parts of the application use the same implementation of common functionality.
4. Type safety: The unified services provide type-safe interfaces for common operations.
5. Extensibility: The provider pattern allows adding new implementations without changing client code.

### Next Steps

The following steps are recommended to further improve the unified services architecture in the client package:

1. Update components to use the unified services directly where appropriate
2. Add more tests to ensure the unified services work correctly
3. Remove the adapter files once all code has been updated to use the unified services directly
4. Add more documentation to help developers understand how to use the unified services

# Credit System

Source: readme/credit-system.md

---

## Credit System for API Usage

The credit system allows users to consume third-party API services through a unified credit-based payment model. Credits are allocated through subscriptions and can be used for various services like OpenAI API calls, 3D model generation, and agent usage.

### Overview

The credit system consists of the following components:

1. Service Cost Management: Admin-configurable costs for third-party services
2. Credit Allocation: Credits included with subscription plans
3. Credit Usage Tracking: Tracking and deducting credits for API usage
4. Credit Purchasing: Allowing users to purchase additional credits

### Database Schema

#### Service Costs Table

Stores the costs of third-party API services:

CREATE TABLE service\_costs (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 service\_name TEXT NOT NULL,  
 service\_key TEXT NOT NULL UNIQUE,  
 cost\_per\_unit DECIMAL(10, 6) NOT NULL,  
 unit\_type TEXT NOT NULL,  
 multiplier DECIMAL(10, 2) NOT NULL DEFAULT 1.0,  
 description TEXT,  
 is\_active BOOLEAN NOT NULL DEFAULT true,  
 created\_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW()  
);

CREATE TABLE service\_costs (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 service\_name TEXT NOT NULL,  
 service\_key TEXT NOT NULL UNIQUE,  
 cost\_per\_unit DECIMAL(10, 6) NOT NULL,  
 unit\_type TEXT NOT NULL,  
 multiplier DECIMAL(10, 2) NOT NULL DEFAULT 1.0,  
 description TEXT,  
 is\_active BOOLEAN NOT NULL DEFAULT true,  
 created\_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW()  
);

#### User Credits Table

Stores user credit balances:

CREATE TABLE user\_credits (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 user\_id UUID NOT NULL REFERENCES auth.users(id) ON DELETE CASCADE,  
 balance INTEGER NOT NULL DEFAULT 0,  
 last\_updated\_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW(),  
 created\_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW(),  
 UNIQUE(user\_id)  
);

CREATE TABLE user\_credits (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 user\_id UUID NOT NULL REFERENCES auth.users(id) ON DELETE CASCADE,  
 balance INTEGER NOT NULL DEFAULT 0,  
 last\_updated\_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW(),  
 created\_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW(),  
 UNIQUE(user\_id)  
);

#### Credit Transactions Table

Tracks credit transactions:

CREATE TABLE credit\_transactions (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 user\_id UUID NOT NULL REFERENCES auth.users(id) ON DELETE CASCADE,  
 amount INTEGER NOT NULL, -- Positive for additions, negative for usage  
 balance INTEGER NOT NULL, -- Balance after transaction  
 description TEXT NOT NULL,  
 type TEXT NOT NULL CHECK (type IN ('purchase', 'usage', 'refund', 'expiration', 'adjustment', 'subscription')),  
 service\_key TEXT, -- Key of the service that used the credits  
 service\_usage JSONB, -- Details of service usage  
 metadata JSONB,  
 created\_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW(),  
 expires\_at TIMESTAMP WITH TIME ZONE  
);

CREATE TABLE credit\_transactions (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 user\_id UUID NOT NULL REFERENCES auth.users(id) ON DELETE CASCADE,  
 amount INTEGER NOT NULL, -- Positive for additions, negative for usage  
 balance INTEGER NOT NULL, -- Balance after transaction  
 description TEXT NOT NULL,  
 type TEXT NOT NULL CHECK (type IN ('purchase', 'usage', 'refund', 'expiration', 'adjustment', 'subscription')),  
 service\_key TEXT, -- Key of the service that used the credits  
 service\_usage JSONB, -- Details of service usage  
 metadata JSONB,  
 created\_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW(),  
 expires\_at TIMESTAMP WITH TIME ZONE  
);

### Core Components

#### Service Cost Model

The serviceCost.model.ts file provides functions for managing service costs:

serviceCost.model.ts

* getAllServiceCosts(): Get all service costs
* getServiceCostByKey(serviceKey): Get service cost by key
* createServiceCost(serviceCost): Create a new service cost
* updateServiceCost(id, updates): Update a service cost
* deleteServiceCost(id): Delete a service cost
* calculateCreditCost(serviceKey, units): Calculate credit cost for service usage

getAllServiceCosts()

getServiceCostByKey(serviceKey)

createServiceCost(serviceCost)

updateServiceCost(id, updates)

deleteServiceCost(id)

calculateCreditCost(serviceKey, units)

#### User Credit Model

The userCredit.model.ts file provides functions for managing user credits:

userCredit.model.ts

* getUserCredit(userId): Get user credit balance
* addCredits(userId, amount, description, type, metadata): Add credits to user
* useCredits(userId, amount, description, type, metadata): Use credits from user
* useServiceCredits(userId, serviceKey, units, description, metadata): Use credits for a specific service
* getCreditTransactions(userId, limit, offset): Get credit transactions for a user
* getCreditUsageByService(userId, limit, offset): Get credit usage by service
* hasEnoughCredits(userId, amount): Check if user has enough credits
* initializeUserCredit(userId, initialBalance): Initialize user credit if it doesn't exist

getUserCredit(userId)

addCredits(userId, amount, description, type, metadata)

useCredits(userId, amount, description, type, metadata)

useServiceCredits(userId, serviceKey, units, description, metadata)

getCreditTransactions(userId, limit, offset)

getCreditUsageByService(userId, limit, offset)

hasEnoughCredits(userId, amount)

initializeUserCredit(userId, initialBalance)

#### Credit Service

The creditService.ts file provides a unified interface for credit management:

creditService.ts

* hasEnoughCreditsForService(userId, serviceKey, units): Check if user has enough credits for a service
* useServiceCredits(userId, serviceKey, units, description, metadata): Use credits for a service
* getAllServiceCosts(): Get all service costs
* getServiceCostByKey(serviceKey): Get service cost by key
* createServiceCost(serviceCost): Create a new service cost
* updateServiceCost(id, updates): Update a service cost
* deleteServiceCost(id): Delete a service cost
* getUserCreditUsageByService(userId, limit, offset): Get user credit usage by service
* getUserCreditBalance(userId): Get user credit balance
* addCreditsToUser(userId, amount, description, type, metadata): Add credits to user
* initializeUserCredit(userId, initialBalance): Initialize user credit if it doesn't exist

hasEnoughCreditsForService(userId, serviceKey, units)

useServiceCredits(userId, serviceKey, units, description, metadata)

getAllServiceCosts()

getServiceCostByKey(serviceKey)

createServiceCost(serviceCost)

updateServiceCost(id, updates)

deleteServiceCost(id)

getUserCreditUsageByService(userId, limit, offset)

getUserCreditBalance(userId)

addCreditsToUser(userId, amount, description, type, metadata)

initializeUserCredit(userId, initialBalance)

#### API Credit Middleware

The apiCredit.middleware.ts file provides middleware for checking and deducting credits for API usage:

apiCredit.middleware.ts

* checkApiCredits(serviceKey, estimatedUnits): Check if user has enough credits for an API request
* trackApiUsage(req, actualUnits, additionalMetadata): Track actual API usage and deduct credits

checkApiCredits(serviceKey, estimatedUnits)

trackApiUsage(req, actualUnits, additionalMetadata)

### Integration with Subscription System

The credit system is integrated with the subscription system in the following ways:

1. Subscription Tiers: Each subscription tier includes a specified number of credits
2. Subscription Creation: Credits are allocated when a user subscribes to a plan
3. Subscription Renewal: Credits are replenished on subscription renewal
4. Credit Purchasing: Users can purchase additional credits beyond what's included in their subscription

### API Endpoints

#### User Endpoints

* GET /api/subscriptions/credits: Get user's credit balance
* GET /api/subscriptions/credits/transactions: Get user's credit transactions
* GET /api/subscriptions/credits/usage-by-service: Get credit usage by service
* POST /api/subscriptions/credits/purchase: Purchase credits
* POST /api/subscriptions/credits/use: Use credits
* POST /api/subscriptions/credits/use-service: Use credits for a specific service
* GET /api/subscriptions/service-costs: Get all service costs

GET /api/subscriptions/credits

GET /api/subscriptions/credits/transactions

GET /api/subscriptions/credits/usage-by-service

POST /api/subscriptions/credits/purchase

POST /api/subscriptions/credits/use

POST /api/subscriptions/credits/use-service

GET /api/subscriptions/service-costs

#### Admin Endpoints

* GET /api/admin/service-costs: Get all service costs
* GET /api/admin/service-costs/:id: Get service cost by ID
* POST /api/admin/service-costs: Create a new service cost
* PUT /api/admin/service-costs/:id: Update a service cost
* DELETE /api/admin/service-costs/:id: Delete a service cost

GET /api/admin/service-costs

GET /api/admin/service-costs/:id

POST /api/admin/service-costs

PUT /api/admin/service-costs/:id

DELETE /api/admin/service-costs/:id

### Usage Examples

#### Checking and Deducting Credits for API Usage

// In an API route handler  
import apiCreditMiddleware from '../middleware/apiCredit.middleware';  
  
// Check if user has enough credits for OpenAI GPT-4 usage  
router.post(  
 '/generate-text',  
 authMiddleware,  
 apiCreditMiddleware.checkApiCredits('openai.gpt-4', 10), // Estimated 10 units  
 async (req, res) => {  
 try {  
 // Process the request  
 const result = await openaiService.generateText(req.body.prompt);  
  
 // Track actual usage (e.g., based on tokens used)  
 await apiCreditMiddleware.trackApiUsage(  
 req,  
 result.usage.totalTokens / 1000, // Convert tokens to units  
 { model: 'gpt-4', promptTokens: result.usage.promptTokens, completionTokens: result.usage.completionTokens }  
 );  
  
 res.status(200).json({ success: true, data: result });  
 } catch (error) {  
 // Handle error  
 res.status(500).json({ success: false, error: error.message });  
 }  
 }  
);

// In an API route handler  
import apiCreditMiddleware from '../middleware/apiCredit.middleware';  
  
// Check if user has enough credits for OpenAI GPT-4 usage  
router.post(  
 '/generate-text',  
 authMiddleware,  
 apiCreditMiddleware.checkApiCredits('openai.gpt-4', 10), // Estimated 10 units  
 async (req, res) => {  
 try {  
 // Process the request  
 const result = await openaiService.generateText(req.body.prompt);  
  
 // Track actual usage (e.g., based on tokens used)  
 await apiCreditMiddleware.trackApiUsage(  
 req,  
 result.usage.totalTokens / 1000, // Convert tokens to units  
 { model: 'gpt-4', promptTokens: result.usage.promptTokens, completionTokens: result.usage.completionTokens }  
 );  
  
 res.status(200).json({ success: true, data: result });  
 } catch (error) {  
 // Handle error  
 res.status(500).json({ success: false, error: error.message });  
 }  
 }  
);

#### Adding Credits to a User

// Add 100 credits to a user  
const result = await creditService.addCreditsToUser(  
 userId,  
 100,  
 'Credit purchase',  
 'purchase',  
 { paymentId: 'pi\_123456' }  
);

// Add 100 credits to a user  
const result = await creditService.addCreditsToUser(  
 userId,  
 100,  
 'Credit purchase',  
 'purchase',  
 { paymentId: 'pi\_123456' }  
);

#### Using Credits for a Service

// Use credits for OpenAI GPT-4 usage  
const result = await creditService.useServiceCredits(  
 userId,  
 'openai.gpt-4',  
 5, // 5 units (e.g., 5000 tokens)  
 'Text generation with GPT-4',  
 { prompt: 'Summarize this article...', model: 'gpt-4' }  
);

// Use credits for OpenAI GPT-4 usage  
const result = await creditService.useServiceCredits(  
 userId,  
 'openai.gpt-4',  
 5, // 5 units (e.g., 5000 tokens)  
 'Text generation with GPT-4',  
 { prompt: 'Summarize this article...', model: 'gpt-4' }  
);

### Admin Panel

The admin panel provides the following functionality for managing the credit system:

1. Service Cost Management: Add, edit, and delete service costs
2. Credit Usage Analytics: View credit usage by service and user
3. User Credit Management: Add credits to users and view credit balances
4. Subscription Tier Configuration: Configure included credits for subscription tiers

### Future Enhancements

1. Credit Expiration: Implement credit expiration for time-limited credits
2. Credit Bundles: Create bundles of credits with discounted pricing
3. Service-Specific Credits: Implement credits that can only be used for specific services
4. Usage Forecasting: Provide usage forecasting based on historical data
5. Automated Cost Updates: Automatically update service costs based on third-party API pricing changes

# Crewai Implementation

Source: readme/crewai-implementation.md

---

## crewAI Integration Implementation

This document provides a comprehensive overview of the crewAI agent integration implementation for the KAI platform, including the current state, architecture, and next steps.

### Implementation Summary

The integration of crewAI agents into the KAI platform has been implemented with the following components:

#### Backend Components

1. Agent Controller (packages/server/src/controllers/agents.controller.ts)
2. Session management for agent conversations
3. Message processing and response generation
4. Image upload and analysis for recognition agents
5. Mock implementation of agent processing with hooks for crewAI integration
6. Agent Routes (packages/server/src/routes/agents.routes.ts)
7. RESTful API endpoints for agent interaction
8. Session creation, message sending, and retrieval
9. Image upload for recognition agents
10. Admin endpoints for system status monitoring
11. Server Integration (packages/server/src/server.ts)
12. Registration of agent routes at /api/agents
13. Integration with existing authentication and middleware

packages/server/src/controllers/agents.controller.ts

Mock implementation of agent processing with hooks for crewAI integration

Agent Routes (packages/server/src/routes/agents.routes.ts)

packages/server/src/routes/agents.routes.ts

Admin endpoints for system status monitoring

Server Integration (packages/server/src/server.ts)

packages/server/src/server.ts

/api/agents

#### Frontend Components

1. Agent Service (packages/client/src/services/agentService.ts)
2. Centralized service for agent communication
3. Session management and message handling
4. Image upload for recognition agents
5. Mock implementations with hooks for real API integration
6. Agent Chat Component (packages/client/src/components/agents/AgentChat.tsx)
7. Reusable chat interface for agent interactions
8. Real-time message display and input handling
9. Image upload capabilities for recognition agents
10. Typing indicators and message threading
11. Specialized Agent Panels
12. RecognitionPanel: Image upload and material recognition with agent assistance
13. MaterialExpertPanel: Detailed material information with expert guidance
14. ProjectAssistantPanel: Project planning and material organization with agent assistance
15. Agent Dashboard (packages/client/src/components/agents/AgentDashboard.tsx)
16. Unified interface for accessing different agent capabilities
17. Tab-based navigation between agent types
18. Consistent UI and experience across agent interactions

packages/client/src/services/agentService.ts

Mock implementations with hooks for real API integration

Agent Chat Component (packages/client/src/components/agents/AgentChat.tsx)

packages/client/src/components/agents/AgentChat.tsx

Typing indicators and message threading

Specialized Agent Panels

RecognitionPanel

MaterialExpertPanel

ProjectAssistantPanel: Project planning and material organization with agent assistance

ProjectAssistantPanel

Agent Dashboard (packages/client/src/components/agents/AgentDashboard.tsx)

packages/client/src/components/agents/AgentDashboard.tsx

#### Agent Package

1. Core Agent System (packages/agents/src/core/)
2. Agent type definitions and configuration interfaces
3. System initialization and management
4. Agent creation and registration
5. Frontend Agents (packages/agents/src/frontend/)
6. RecognitionAssistant: Helps identify materials from images
7. MaterialExpert: Provides detailed information about materials
8. ProjectAssistant: Helps organize materials into projects
9. Backend Agents (packages/agents/src/backend/)
10. KnowledgeBaseAgent: Maintains and improves the knowledge base
11. AnalyticsAgent: Processes usage patterns and system metrics
12. OperationsAgent: Monitors system health and performance
13. Agent Tools (packages/agents/src/tools/)
14. materialSearch: Search for materials in the database
15. imageAnalysis: Analyze images for recognition
16. vectorSearch: Perform semantic similarity searches

packages/agents/src/core/

Agent creation and registration

Frontend Agents (packages/agents/src/frontend/)

packages/agents/src/frontend/

RecognitionAssistant

MaterialExpert

ProjectAssistant: Helps organize materials into projects

ProjectAssistant

Backend Agents (packages/agents/src/backend/)

packages/agents/src/backend/

KnowledgeBaseAgent

AnalyticsAgent

OperationsAgent: Monitors system health and performance

OperationsAgent

Agent Tools (packages/agents/src/tools/)

packages/agents/src/tools/

materialSearch

imageAnalysis

vectorSearch

### Current State and Dependencies

The implementation currently includes all necessary files and structure for the crewAI integration, but has the following dependencies that need to be installed to complete the integration:

1. crewAI Package: Core dependency for agent functionality
2. Redis: For agent memory and session persistence
3. Winston: For structured logging
4. @types/node: For Node.js typings
5. @emotion/styled and @emotion/react: For frontend styling (client package)
6. react-dropzone: For image upload functionality (client package)

These dependencies are referenced in the setup script and package.json files but need to be installed. The current implementation includes mock functionality that allows the system to function without these dependencies during development.

### Integration Architecture

The crewAI integration follows a layered architecture:

1. User Interface Layer: React components for agent interaction
2. Service Layer: Client-side services for agent communication
3. API Layer: RESTful endpoints for agent operations
4. Agent System Layer: Core agent definitions and management
5. Tool Layer: Specialized capabilities for agents

Data flows through these layers as follows:

1. User interacts with a specialized agent panel
2. Panel sends requests through the agentService
3. Service communicates with the backend API
4. API routes the request to the appropriate controller method
5. Controller processes the request and invokes the agent system
6. Agent system delegates to the appropriate agent type
7. Agent utilizes tools to perform operations
8. Results flow back through the layers to the user

### Next Steps to Complete the Integration

To complete the crewAI integration, follow these steps:

#### 1. Install Dependencies

Run the setup script to install all required dependencies:

# Make the script executable  
chmod +x packages/agents/scripts/setup.sh  
  
# Run the setup script  
cd packages/agents/scripts  
./setup.sh  
  
# Install client dependencies  
cd ../../client  
npm install @emotion/styled @emotion/react react-dropzone  
  
# Install server dependencies  
cd ../server  
npm install uuid multer

# Make the script executable  
chmod +x packages/agents/scripts/setup.sh  
  
# Run the setup script  
cd packages/agents/scripts  
./setup.sh  
  
# Install client dependencies  
cd ../../client  
npm install @emotion/styled @emotion/react react-dropzone  
  
# Install server dependencies  
cd ../server  
npm install uuid multer

#### 2. Finalize Backend Integration

1. Update Type Definitions: Create a local type definition file for @kai/agents to resolve import errors:

// packages/server/src/types/kai-agents.d.ts  
declare module '@kai/agents' {  
 export enum AgentType {  
 RECOGNITION = 'recognition',  
 MATERIAL\_EXPERT = 'material',  
 PROJECT\_ASSISTANT = 'project',  
 KNOWLEDGE\_BASE = 'knowledge\_base',  
 ANALYTICS = 'analytics',  
 OPERATIONS = 'operations'  
 }  
  
 export interface AgentConfig {  
 // Add necessary properties  
 }  
  
 export function initializeAgentSystem(config: any): Promise<void>;  
}

// packages/server/src/types/kai-agents.d.ts  
declare module '@kai/agents' {  
 export enum AgentType {  
 RECOGNITION = 'recognition',  
 MATERIAL\_EXPERT = 'material',  
 PROJECT\_ASSISTANT = 'project',  
 KNOWLEDGE\_BASE = 'knowledge\_base',  
 ANALYTICS = 'analytics',  
 OPERATIONS = 'operations'  
 }  
  
 export interface AgentConfig {  
 // Add necessary properties  
 }  
  
 export function initializeAgentSystem(config: any): Promise<void>;  
}

1. Create uploads directory: Ensure the directory for image uploads exists:

mkdir -p uploads

mkdir -p uploads

1. Implement null checks: Update the controller to handle potentially undefined sessionId parameters:

// In packages/server/src/controllers/agents.controller.ts  
// Update the getMessages function:  
if (!sessionId) {  
 return res.status(400).json({ error: 'Session ID is required' });  
}  
const session = sessions.get(sessionId);

// In packages/server/src/controllers/agents.controller.ts  
// Update the getMessages function:  
if (!sessionId) {  
 return res.status(400).json({ error: 'Session ID is required' });  
}  
const session = sessions.get(sessionId);

#### 3. Complete Frontend Integration

1. Fix Type Issues: Update the client tsconfig.json to include proper TypeScript definitions:

{  
 "compilerOptions": {  
 "typeRoots": ["./node\_modules/@types", "./src/types"]  
 }  
}

{  
 "compilerOptions": {  
 "typeRoots": ["./node\_modules/@types", "./src/types"]  
 }  
}

1. Create Local Type Definitions: Add missing type definitions for React hooks:

// packages/client/src/types/react-hooks.d.ts  
import React from 'react';  
  
declare module 'react' {  
 export function useRef<T>(initialValue: T): React.RefObject<T>;  
 export function useRef<T>(initialValue: null): React.RefObject<T | null>;  
 export function useRef<T = undefined>(): React.RefObject<T | undefined>;  
  
 export interface KeyboardEvent<T = Element> extends React.SyntheticEvent<T> {  
 altKey: boolean;  
 charCode: number;  
 ctrlKey: boolean;  
 key: string;  
 keyCode: number;  
 locale: string;  
 metaKey: boolean;  
 repeat: boolean;  
 shiftKey: boolean;  
 which: number;  
 }  
  
 export function useCallback<T extends (...args: any[]) => any>(  
 callback: T,  
 deps: ReadonlyArray<any>  
 ): T;  
}

// packages/client/src/types/react-hooks.d.ts  
import React from 'react';  
  
declare module 'react' {  
 export function useRef<T>(initialValue: T): React.RefObject<T>;  
 export function useRef<T>(initialValue: null): React.RefObject<T | null>;  
 export function useRef<T = undefined>(): React.RefObject<T | undefined>;  
  
 export interface KeyboardEvent<T = Element> extends React.SyntheticEvent<T> {  
 altKey: boolean;  
 charCode: number;  
 ctrlKey: boolean;  
 key: string;  
 keyCode: number;  
 locale: string;  
 metaKey: boolean;  
 repeat: boolean;  
 shiftKey: boolean;  
 which: number;  
 }  
  
 export function useCallback<T extends (...args: any[]) => any>(  
 callback: T,  
 deps: ReadonlyArray<any>  
 ): T;  
}

1. Fix Null Reference Handling: Update the RecognitionPanel component to safely handle potential undefined values:

// In topResult usage, add null checks:  
content: `I've identified your material as ${topResult?.name || 'Unknown'} with ${topResult ? Math.round(topResult.confidence \* 100) : 0}% confidence...`

// In topResult usage, add null checks:  
content: `I've identified your material as ${topResult?.name || 'Unknown'} with ${topResult ? Math.round(topResult.confidence \* 100) : 0}% confidence...`

#### 4. Connect to Real API Endpoints

1. Update agentService: Modify the service to call the actual backend endpoints:

// In packages/client/src/services/agentService.ts  
// Replace mock implementations with actual API calls:  
async sendMessage(sessionId: string, message: string): Promise<void> {  
 const response = await fetch(`/api/agents/session/${sessionId}/message`, {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ message })  
 });  
  
 if (!response.ok) {  
 throw new Error('Failed to send message');  
 }  
}

// In packages/client/src/services/agentService.ts  
// Replace mock implementations with actual API calls:  
async sendMessage(sessionId: string, message: string): Promise<void> {  
 const response = await fetch(`/api/agents/session/${sessionId}/message`, {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ message })  
 });  
  
 if (!response.ok) {  
 throw new Error('Failed to send message');  
 }  
}

1. Update image upload: Connect the image upload functionality to the actual API endpoint:

// In upload functionality:  
const formData = new FormData();  
formData.append('image', file);  
  
const response = await fetch(`/api/agents/session/${sessionId}/image`, {  
 method: 'POST',  
 body: formData  
});

// In upload functionality:  
const formData = new FormData();  
formData.append('image', file);  
  
const response = await fetch(`/api/agents/session/${sessionId}/image`, {  
 method: 'POST',  
 body: formData  
});

#### 5. Testing and Validation

1. Start the development server:

cd packages/server  
npm run dev

cd packages/server  
npm run dev

1. Start the client application:

cd packages/client  
npm run dev

cd packages/client  
npm run dev

1. Navigate to the agents dashboard at /agents to test the integration.
2. Validate each agent type by interacting with them and ensuring proper responses.

Navigate to the agents dashboard at /agents to test the integration.

/agents

Validate each agent type by interacting with them and ensuring proper responses.

### Future Enhancements

Once the basic integration is complete, consider these enhancements:

1. Real-time Communication: Implement WebSocket communication for real-time agent responses
2. Context Awareness: Enhance agents with user history and preferences
3. Voice Interface: Add speech-to-text and text-to-speech capabilities
4. Integration with Knowledge Base: Connect agents directly to the KAI knowledge base
5. Performance Optimization: Implement caching and response optimization

### Conclusion

The crewAI integration has been successfully implemented with all necessary structure and components. By following the next steps outlined above, the integration can be completed and deployed, providing KAI users with intelligent assistance throughout their material discovery and project management workflows.

# Database Vector Db

Source: readme/database-vector-db.md

---

## Database Management and Vector DB

The Kai platform leverages advanced database management and vector database capabilities to provide powerful search, storage, and retrieval functionality. This document details the system's approach to database management, with a focus on vector-based similarity search and hybrid search implementations across multiple application domains.

### Features

#### Supabase Database Integration

The system uses Supabase as its primary database platform:

1. Unified Database Management
2. PostgreSQL-based database engine
3. Secure API access through Supabase SDK
4. Row-level security policies
5. Real-time subscription capabilities
6. Managed authentication and authorization
7. Relational Data Storage
8. Structured tables for materials, collections, categories
9. Normalized schema for efficient data storage
10. Foreign key relationships for data integrity
11. Indexing for performant queries
12. JSON/JSONB support for flexible metadata
13. Database Extensions
14. pgvector extension for vector similarity search
15. Full-text search capabilities
16. GIN and GIST indexing for complex data types
17. Trigger functions for automated actions
18. Custom PostgreSQL functions for specialized operations

Managed authentication and authorization

Relational Data Storage

JSON/JSONB support for flexible metadata

Database Extensions

#### Vector Similarity Search

The system implements advanced vector similarity search:

1. Vector Embedding Storage
2. Dedicated tables for vector embeddings
3. Support for high-dimensional vectors (384+ dimensions)
4. Efficient storage using PostgreSQL's vector type
5. Associated metadata storage for context
6. Linking to source entities (materials, images)
7. Similarity Search Algorithms
8. Cosine similarity for semantic matching
9. Euclidean distance for feature-based matching
10. Approximate nearest neighbor search using HNSW
11. IVF-Flat indexing for larger vector collections
12. Configurable similarity thresholds
13. Vector Index Optimization
14. HNSW indexing for near real-time queries
15. IVF-Flat for batch processing
16. Index parameterization based on data characteristics
17. Automated index rebuilding
18. Performance monitoring and optimization

Linking to source entities (materials, images)

Similarity Search Algorithms

Configurable similarity thresholds

Vector Index Optimization

#### Hybrid Search Capabilities

The system combines multiple search strategies:

1. Multi-Modal Search
2. Text-based search using full-text indexing
3. Vector similarity search for semantic understanding
4. Metadata filtering for structured attributes
5. Combined scoring with configurable weights
6. Unified ranking algorithm
7. Search Customization
8. Adjustable text/vector weight balancing
9. Domain-specific tokenization and preprocessing
10. Query expansion and enhancement
11. Fallback strategies when vectors are unavailable
12. Confidence scoring for result relevance
13. Search Optimization
14. Materialized search indexes
15. Request-time parameter tuning
16. Cached vector computations
17. Progressive loading of results
18. Performance analytics for search quality

Unified ranking algorithm

Search Customization

Confidence scoring for result relevance

Search Optimization

#### Dataset Management

The system provides comprehensive dataset management:

1. Dataset Organization
2. Hierarchical structure with datasets and classes
3. Image storage and classification
4. Metadata and annotation storage
5. Version control for datasets
6. Audit trail for data modifications
7. Data Operations
8. Bulk import and export
9. Dataset splitting and merging
10. Data augmentation and preprocessing
11. Quality assessment and cleanup
12. Incremental updates
13. Integration with ML Pipeline
14. Training/test/validation splitting
15. Feature extraction for ML models
16. Dataset statistics and analytics
17. Model-dataset relationship tracking
18. Performance metrics based on datasets

Audit trail for data modifications

Data Operations

Incremental updates

Integration with ML Pipeline

### Technical Implementation

#### Supabase Client Management

The system implements a Supabase client manager:

/\*\*  
 \* Supabase Client Manager  
 \* Manages connections to Supabase and provides access to the client instance.  
 \*/  
class SupabaseClientManager {  
 private client: SupabaseClient | null = null;  
 private config: {  
 url: string;  
 key: string;  
 options?: any;  
 } | null = null;  
  
 /\*\*  
 \* Initialize the Supabase client with configuration  
 \* @param config Supabase configuration  
 \*/  
 public init(config: { url: string; key: string; options?: any }): void {  
 this.config = config;  
 this.client = createClient(config.url, config.key, config.options);  
 logger.info('Supabase client initialized');  
 }  
  
 /\*\*  
 \* Get the Supabase client instance  
 \* @returns SupabaseClient instance  
 \*/  
 public getClient(): SupabaseClient {  
 if (!this.client) {  
 throw new Error('Supabase client not initialized');  
 }  
 return this.client;  
 }  
}  
  
// Export a singleton instance  
export const supabaseClient = new SupabaseClientManager();

/\*\*  
 \* Supabase Client Manager  
 \* Manages connections to Supabase and provides access to the client instance.  
 \*/  
class SupabaseClientManager {  
 private client: SupabaseClient | null = null;  
 private config: {  
 url: string;  
 key: string;  
 options?: any;  
 } | null = null;  
  
 /\*\*  
 \* Initialize the Supabase client with configuration  
 \* @param config Supabase configuration  
 \*/  
 public init(config: { url: string; key: string; options?: any }): void {  
 this.config = config;  
 this.client = createClient(config.url, config.key, config.options);  
 logger.info('Supabase client initialized');  
 }  
  
 /\*\*  
 \* Get the Supabase client instance  
 \* @returns SupabaseClient instance  
 \*/  
 public getClient(): SupabaseClient {  
 if (!this.client) {  
 throw new Error('Supabase client not initialized');  
 }  
 return this.client;  
 }  
}  
  
// Export a singleton instance  
export const supabaseClient = new SupabaseClientManager();

#### Vector Search Implementation

The system provides a dedicated vector search service:

/\*\*  
 \* Vector Search Service for Supabase  
 \* Provides methods for storing and querying vector embeddings  
 \*/  
class SupabaseVectorSearch {  
 /\*\*  
 \* Find similar vectors to the provided embedding  
 \*  
 \* @param embedding The query vector to find similar vectors for  
 \* @param tableName The table containing vector embeddings  
 \* @param vectorColumn The column name for the vector data (default: 'embedding')  
 \* @param config Search configuration options  
 \* @returns Array of matching records with similarity scores  
 \*/  
 async findSimilar(  
 embedding: number[],  
 tableName: string,  
 vectorColumn: string = 'embedding',  
 config: VectorSearchConfig = {}  
 ): Promise<Array<Record<string, any>>> {  
 try {  
 // Initialize query with proper client  
 const client = supabaseClient.getClient();  
  
 const limit = config.limit || 10;  
 const threshold = config.threshold || 0.75;  
  
 // Use vector operators in PostgreSQL query  
 const { data, error } = await client  
 .from(tableName)  
 .select(`  
 \*,  
 similarity:1 - (${vectorColumn} <=> ${'embedding'})  
 `)  
 .gte('similarity', threshold)  
 .order('similarity', { ascending: false })  
 .limit(limit);  
  
 if (error) throw error;  
  
 return data || [];  
  
 } catch (error) {  
 logger.error(`Vector search error: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Store a vector embedding in Supabase  
 \*  
 \* @param embedding The vector to store  
 \* @param metadata Additional metadata to store with the vector  
 \* @param tableName The table to store the vector in  
 \* @param vectorColumn The column name for the vector data (default: 'embedding')  
 \* @returns The created record ID  
 \*/  
 async storeVector(  
 embedding: number[],  
 metadata: Record<string, any>,  
 tableName: string,  
 vectorColumn: string = 'embedding'  
 ): Promise<string> {  
 try {  
 const record = {  
 ...metadata,  
 [vectorColumn]: embedding,  
 created\_at: new Date().toISOString()  
 };  
  
 // Insert the data using proper client  
 const client = supabaseClient.getClient();  
  
 const { data, error } = await client  
 .from(tableName)  
 .insert(record)  
 .select('id')  
 .single();  
  
 if (error) throw error;  
  
 return data.id;  
  
 } catch (error) {  
 logger.error(`Vector storage error: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Create a vector index for faster similarity searches  
 \*  
 \* @param tableName The table to create the index for  
 \* @param vectorColumn The column name for the vector data  
 \* @param indexMethod The index method to use ('ivfflat', 'hnsw')  
 \* @param dimensions Optional vector dimensions (default: 1536 for typical embeddings)  
 \* @returns True if the index was created successfully  
 \*/  
 async createIndex(  
 tableName: string,  
 vectorColumn: string = 'embedding',  
 indexMethod: 'ivfflat' | 'hnsw' = 'hnsw',  
 dimensions: number = 1536  
 ): Promise<boolean> {  
 try {  
 // Get the Supabase client  
 const client = supabaseClient.getClient();  
  
 try {  
 // For some operations, we need to use raw SQL to create indices  
 const indexName = `idx\_${tableName}\_${vectorColumn}\_${indexMethod}`;  
  
 // Note: The actual SQL execution will depend on whether Supabase allows raw SQL  
 // This might need to be done through database migrations or admin console  
 const sql = indexMethod === 'hnsw'  
 ? `CREATE INDEX IF NOT EXISTS ${indexName} ON ${tableName} USING hnsw (${vectorColumn} vector\_l2\_ops) WITH (dims=${dimensions});`  
 : `CREATE INDEX IF NOT EXISTS ${indexName} ON ${tableName} USING ivfflat (${vectorColumn} vector\_l2\_ops) WITH (lists=100);`;  
  
 // Execute SQL to create index  
 // Implementation details may vary based on Supabase access level  
  
 return true;  
  
 } catch (error) {  
 logger.error(`Error creating vector index: ${error}`);  
 throw error;  
 }  
  
 } catch (error) {  
 logger.error(`Index creation error: ${error}`);  
 throw error;  
 }  
 }  
}  
  
// Export singleton instance  
export const vectorSearch = new SupabaseVectorSearch();

/\*\*  
 \* Vector Search Service for Supabase  
 \* Provides methods for storing and querying vector embeddings  
 \*/  
class SupabaseVectorSearch {  
 /\*\*  
 \* Find similar vectors to the provided embedding  
 \*  
 \* @param embedding The query vector to find similar vectors for  
 \* @param tableName The table containing vector embeddings  
 \* @param vectorColumn The column name for the vector data (default: 'embedding')  
 \* @param config Search configuration options  
 \* @returns Array of matching records with similarity scores  
 \*/  
 async findSimilar(  
 embedding: number[],  
 tableName: string,  
 vectorColumn: string = 'embedding',  
 config: VectorSearchConfig = {}  
 ): Promise<Array<Record<string, any>>> {  
 try {  
 // Initialize query with proper client  
 const client = supabaseClient.getClient();  
  
 const limit = config.limit || 10;  
 const threshold = config.threshold || 0.75;  
  
 // Use vector operators in PostgreSQL query  
 const { data, error } = await client  
 .from(tableName)  
 .select(`  
 \*,  
 similarity:1 - (${vectorColumn} <=> ${'embedding'})  
 `)  
 .gte('similarity', threshold)  
 .order('similarity', { ascending: false })  
 .limit(limit);  
  
 if (error) throw error;  
  
 return data || [];  
  
 } catch (error) {  
 logger.error(`Vector search error: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Store a vector embedding in Supabase  
 \*  
 \* @param embedding The vector to store  
 \* @param metadata Additional metadata to store with the vector  
 \* @param tableName The table to store the vector in  
 \* @param vectorColumn The column name for the vector data (default: 'embedding')  
 \* @returns The created record ID  
 \*/  
 async storeVector(  
 embedding: number[],  
 metadata: Record<string, any>,  
 tableName: string,  
 vectorColumn: string = 'embedding'  
 ): Promise<string> {  
 try {  
 const record = {  
 ...metadata,  
 [vectorColumn]: embedding,  
 created\_at: new Date().toISOString()  
 };  
  
 // Insert the data using proper client  
 const client = supabaseClient.getClient();  
  
 const { data, error } = await client  
 .from(tableName)  
 .insert(record)  
 .select('id')  
 .single();  
  
 if (error) throw error;  
  
 return data.id;  
  
 } catch (error) {  
 logger.error(`Vector storage error: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Create a vector index for faster similarity searches  
 \*  
 \* @param tableName The table to create the index for  
 \* @param vectorColumn The column name for the vector data  
 \* @param indexMethod The index method to use ('ivfflat', 'hnsw')  
 \* @param dimensions Optional vector dimensions (default: 1536 for typical embeddings)  
 \* @returns True if the index was created successfully  
 \*/  
 async createIndex(  
 tableName: string,  
 vectorColumn: string = 'embedding',  
 indexMethod: 'ivfflat' | 'hnsw' = 'hnsw',  
 dimensions: number = 1536  
 ): Promise<boolean> {  
 try {  
 // Get the Supabase client  
 const client = supabaseClient.getClient();  
  
 try {  
 // For some operations, we need to use raw SQL to create indices  
 const indexName = `idx\_${tableName}\_${vectorColumn}\_${indexMethod}`;  
  
 // Note: The actual SQL execution will depend on whether Supabase allows raw SQL  
 // This might need to be done through database migrations or admin console  
 const sql = indexMethod === 'hnsw'  
 ? `CREATE INDEX IF NOT EXISTS ${indexName} ON ${tableName} USING hnsw (${vectorColumn} vector\_l2\_ops) WITH (dims=${dimensions});`  
 : `CREATE INDEX IF NOT EXISTS ${indexName} ON ${tableName} USING ivfflat (${vectorColumn} vector\_l2\_ops) WITH (lists=100);`;  
  
 // Execute SQL to create index  
 // Implementation details may vary based on Supabase access level  
  
 return true;  
  
 } catch (error) {  
 logger.error(`Error creating vector index: ${error}`);  
 throw error;  
 }  
  
 } catch (error) {  
 logger.error(`Index creation error: ${error}`);  
 throw error;  
 }  
 }  
}  
  
// Export singleton instance  
export const vectorSearch = new SupabaseVectorSearch();

#### Hybrid Search Integration

The system implements hybrid search combining text and vector similarity:

-- Function to perform hybrid search  
CREATE OR REPLACE FUNCTION hybrid\_search(  
 query\_text TEXT,  
 query\_embedding VECTOR,  
 table\_name TEXT,  
 text\_columns TEXT[] DEFAULT ARRAY['name', 'description'],  
 vector\_column TEXT DEFAULT 'embedding',  
 text\_weight FLOAT DEFAULT 0.5,  
 vector\_weight FLOAT DEFAULT 0.5,  
 match\_count INT DEFAULT 10,  
 score\_threshold FLOAT DEFAULT 0.5  
) RETURNS TABLE (  
 id UUID,  
 text\_score FLOAT,  
 vector\_score FLOAT,  
 combined\_score FLOAT  
) AS $$  
DECLARE  
 text\_query TEXT;  
 vector\_query TEXT;  
 final\_query TEXT;  
BEGIN  
 -- Construct the text search portion  
 text\_query := FORMAT('  
 WITH text\_search AS (  
 SELECT  
 id,  
 CASE WHEN %L = '''' THEN 0  
 ELSE ts\_rank(to\_tsvector(''english'', %s), websearch\_to\_tsquery(''english'', %L))  
 END AS text\_score  
 FROM %I  
 WHERE true  
 ' || CASE WHEN length(query\_text) > 0 THEN '  
 AND to\_tsvector(''english'', %s) @@ websearch\_to\_tsquery(''english'', %L)  
 ' ELSE '' END || '  
 ),  
 vector\_search AS (  
 SELECT  
 id,  
 CASE WHEN %I IS NULL THEN 0  
 ELSE 1 - (%I <=> %L::vector)  
 END AS vector\_score  
 FROM %I  
 ),  
 hybrid\_results AS (  
 SELECT  
 COALESCE(t.id, v.id) AS id,  
 COALESCE(t.text\_score, 0) AS text\_score,  
 COALESCE(v.vector\_score, 0) AS vector\_score,  
 (COALESCE(t.text\_score, 0) \* %L) + (COALESCE(v.vector\_score, 0) \* %L) AS combined\_score  
 FROM text\_search t  
 FULL OUTER JOIN vector\_search v ON t.id = v.id  
 WHERE (t.id IS NOT NULL OR v.id IS NOT NULL)  
 AND ((COALESCE(t.text\_score, 0) \* %L) + (COALESCE(v.vector\_score, 0) \* %L)) >= %L  
 ORDER BY combined\_score DESC  
 LIMIT %L  
 )  
 SELECT  
 id,  
 text\_score,  
 vector\_score,  
 combined\_score  
 FROM hybrid\_results;',  
 query\_text,  
 array\_to\_string(text\_columns, ' || '' '' || '),  
 query\_text,  
 table\_name,  
 array\_to\_string(text\_columns, ' || '' '' || '),  
 query\_text,  
 vector\_column,  
 vector\_column,  
 query\_embedding,  
 table\_name,  
 text\_weight,  
 vector\_weight,  
 text\_weight,  
 vector\_weight,  
 score\_threshold,  
 match\_count  
 );  
  
 RETURN QUERY EXECUTE text\_query;  
END;  
$$ LANGUAGE plpgsql;

-- Function to perform hybrid search  
CREATE OR REPLACE FUNCTION hybrid\_search(  
 query\_text TEXT,  
 query\_embedding VECTOR,  
 table\_name TEXT,  
 text\_columns TEXT[] DEFAULT ARRAY['name', 'description'],  
 vector\_column TEXT DEFAULT 'embedding',  
 text\_weight FLOAT DEFAULT 0.5,  
 vector\_weight FLOAT DEFAULT 0.5,  
 match\_count INT DEFAULT 10,  
 score\_threshold FLOAT DEFAULT 0.5  
) RETURNS TABLE (  
 id UUID,  
 text\_score FLOAT,  
 vector\_score FLOAT,  
 combined\_score FLOAT  
) AS $$  
DECLARE  
 text\_query TEXT;  
 vector\_query TEXT;  
 final\_query TEXT;  
BEGIN  
 -- Construct the text search portion  
 text\_query := FORMAT('  
 WITH text\_search AS (  
 SELECT  
 id,  
 CASE WHEN %L = '''' THEN 0  
 ELSE ts\_rank(to\_tsvector(''english'', %s), websearch\_to\_tsquery(''english'', %L))  
 END AS text\_score  
 FROM %I  
 WHERE true  
 ' || CASE WHEN length(query\_text) > 0 THEN '  
 AND to\_tsvector(''english'', %s) @@ websearch\_to\_tsquery(''english'', %L)  
 ' ELSE '' END || '  
 ),  
 vector\_search AS (  
 SELECT  
 id,  
 CASE WHEN %I IS NULL THEN 0  
 ELSE 1 - (%I <=> %L::vector)  
 END AS vector\_score  
 FROM %I  
 ),  
 hybrid\_results AS (  
 SELECT  
 COALESCE(t.id, v.id) AS id,  
 COALESCE(t.text\_score, 0) AS text\_score,  
 COALESCE(v.vector\_score, 0) AS vector\_score,  
 (COALESCE(t.text\_score, 0) \* %L) + (COALESCE(v.vector\_score, 0) \* %L) AS combined\_score  
 FROM text\_search t  
 FULL OUTER JOIN vector\_search v ON t.id = v.id  
 WHERE (t.id IS NOT NULL OR v.id IS NOT NULL)  
 AND ((COALESCE(t.text\_score, 0) \* %L) + (COALESCE(v.vector\_score, 0) \* %L)) >= %L  
 ORDER BY combined\_score DESC  
 LIMIT %L  
 )  
 SELECT  
 id,  
 text\_score,  
 vector\_score,  
 combined\_score  
 FROM hybrid\_results;',  
 query\_text,  
 array\_to\_string(text\_columns, ' || '' '' || '),  
 query\_text,  
 table\_name,  
 array\_to\_string(text\_columns, ' || '' '' || '),  
 query\_text,  
 vector\_column,  
 vector\_column,  
 query\_embedding,  
 table\_name,  
 text\_weight,  
 vector\_weight,  
 text\_weight,  
 vector\_weight,  
 score\_threshold,  
 match\_count  
 );  
  
 RETURN QUERY EXECUTE text\_query;  
END;  
$$ LANGUAGE plpgsql;

#### Database Schema Design

The system implements a comprehensive database schema:

-- Enable the pgvector extension for vector similarity search  
CREATE EXTENSION IF NOT EXISTS pgvector;  
  
-- Materials table with full-text search and vector support  
CREATE TABLE IF NOT EXISTS materials (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 name VARCHAR(255) NOT NULL,  
 description TEXT,  
 material\_type VARCHAR(100),  
 manufacturer VARCHAR(255),  
 product\_code VARCHAR(100),  
 dimensions JSONB,  
 color JSONB,  
 finish VARCHAR(100),  
 tags TEXT[],  
 metadata JSONB,  
 created\_by UUID REFERENCES users(id),  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
  
 -- Generated column for full-text search  
 search\_text TSVECTOR GENERATED ALWAYS AS (  
 to\_tsvector('english', coalesce(name, '') || ' ' || coalesce(description, '') || ' ' ||  
 coalesce(type, '') || ' ' || coalesce(array\_to\_string(tags, ' '), ''))  
 ) STORED  
);  
  
-- Vector embeddings table for similarity search  
CREATE TABLE IF NOT EXISTS vector\_embeddings (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 material\_id UUID REFERENCES materials(id) ON DELETE CASCADE,  
 embedding vector(384), -- Adjust dimension as needed  
 metadata JSONB,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Create an index for cosine distance similarity search  
CREATE INDEX IF NOT EXISTS vector\_embeddings\_embedding\_idx ON vector\_embeddings USING ivfflat (embedding vector\_cosine\_ops) WITH (lists = 100);  
  
-- Create an index for material ID lookups  
CREATE INDEX IF NOT EXISTS vector\_embeddings\_material\_id\_idx ON vector\_embeddings (material\_id);  
  
-- Function to find similar materials by vector embedding  
CREATE OR REPLACE FUNCTION find\_similar\_materials(  
 query\_embedding vector,  
 match\_threshold float,  
 match\_count int  
)  
RETURNS TABLE (  
 id UUID,  
 similarity float  
)  
LANGUAGE SQL  
AS $$  
SELECT  
 m.id,  
 1 - (ve.embedding <=> query\_embedding) AS similarity  
FROM  
 vector\_embeddings ve  
JOIN  
 materials m ON ve.material\_id = m.id  
WHERE  
 1 - (ve.embedding <=> query\_embedding) > match\_threshold  
ORDER BY  
 similarity DESC  
LIMIT  
 match\_count;  
$$;

-- Enable the pgvector extension for vector similarity search  
CREATE EXTENSION IF NOT EXISTS pgvector;  
  
-- Materials table with full-text search and vector support  
CREATE TABLE IF NOT EXISTS materials (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 name VARCHAR(255) NOT NULL,  
 description TEXT,  
 material\_type VARCHAR(100),  
 manufacturer VARCHAR(255),  
 product\_code VARCHAR(100),  
 dimensions JSONB,  
 color JSONB,  
 finish VARCHAR(100),  
 tags TEXT[],  
 metadata JSONB,  
 created\_by UUID REFERENCES users(id),  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
  
 -- Generated column for full-text search  
 search\_text TSVECTOR GENERATED ALWAYS AS (  
 to\_tsvector('english', coalesce(name, '') || ' ' || coalesce(description, '') || ' ' ||  
 coalesce(type, '') || ' ' || coalesce(array\_to\_string(tags, ' '), ''))  
 ) STORED  
);  
  
-- Vector embeddings table for similarity search  
CREATE TABLE IF NOT EXISTS vector\_embeddings (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 material\_id UUID REFERENCES materials(id) ON DELETE CASCADE,  
 embedding vector(384), -- Adjust dimension as needed  
 metadata JSONB,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Create an index for cosine distance similarity search  
CREATE INDEX IF NOT EXISTS vector\_embeddings\_embedding\_idx ON vector\_embeddings USING ivfflat (embedding vector\_cosine\_ops) WITH (lists = 100);  
  
-- Create an index for material ID lookups  
CREATE INDEX IF NOT EXISTS vector\_embeddings\_material\_id\_idx ON vector\_embeddings (material\_id);  
  
-- Function to find similar materials by vector embedding  
CREATE OR REPLACE FUNCTION find\_similar\_materials(  
 query\_embedding vector,  
 match\_threshold float,  
 match\_count int  
)  
RETURNS TABLE (  
 id UUID,  
 similarity float  
)  
LANGUAGE SQL  
AS $$  
SELECT  
 m.id,  
 1 - (ve.embedding <=> query\_embedding) AS similarity  
FROM  
 vector\_embeddings ve  
JOIN  
 materials m ON ve.material\_id = m.id  
WHERE  
 1 - (ve.embedding <=> query\_embedding) > match\_threshold  
ORDER BY  
 similarity DESC  
LIMIT  
 match\_count;  
$$;

#### Dataset Management Service

The system provides a comprehensive dataset management service:

/\*\*  
 \* Dataset Management Service  
 \* Provides methods for dataset creation, manipulation, and analysis  
 \*/  
export class DatasetManagementService {  
 /\*\*  
 \* Analyze a dataset to identify quality issues  
 \*   
 \* @param datasetId The ID of the dataset to analyze  
 \* @returns Dataset analysis results  
 \*/  
 public async analyzeDataset(datasetId: string): Promise<DatasetAnalysisResult> {  
 try {  
 const result: DatasetAnalysisResult = {  
 datasetId,  
 totalImages: 0,  
 issuesDetected: {  
 lowResolutionImages: { count: 0, details: [] },  
 poorQualityImages: { count: 0, details: [] },  
 duplicateImages: { count: 0, details: [] },  
 classImbalance: { totalClasses: 0, details: [] }  
 },  
 recommendations: []  
 };  
  
 // Get the dataset  
 const dataset = await supabaseDatasetService.getDatasetById(datasetId);  
 if (!dataset) {  
 throw new Error(`Dataset not found: ${datasetId}`);  
 }  
  
 // Get classes for the dataset  
 const classes = await supabaseDatasetService.getDatasetClasses(datasetId);  
 result.issuesDetected.classImbalance.totalClasses = classes.length;  
  
 // Analysis logic implementation...  
  
 return result;  
 } catch (error) {  
 logger.error(`Dataset analysis error: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Clean a dataset based on analysis results  
 \*   
 \* @param datasetId The ID of the dataset to clean  
 \* @param cleaningOptions Options for dataset cleaning  
 \* @returns Cleaning operation results  
 \*/  
 public async cleanDataset(  
 datasetId: string,  
 cleaningOptions: DatasetCleaningOptions  
 ): Promise<DatasetCleaningResult> {  
 try {  
 // Implementation details...  
  
 // Create target dataset if creating a new version  
 let targetDatasetId = datasetId;  
 let targetDataset = null;  
  
 if (createNewVersion) {  
 // Create a new dataset as a copy  
 targetDataset = await supabaseDatasetService.createDataset({  
 name: `${dataset.name} (Cleaned)`,  
 description: `Cleaned version of ${dataset.name}`,  
 sourceDatasetId: datasetId,  
 status: 'processing'  
 });  
  
 targetDatasetId = targetDataset.id;  
 }  
  
 // Copy and clean data  
 // Implementation details...  
  
 // Set the cleaned dataset status to ready  
 if (createNewVersion && targetDataset) {  
 await supabaseDatasetService.updateDataset(targetDatasetId, {  
 status: 'ready'  
 });  
 }  
  
 return result;  
 } catch (error) {  
 logger.error(`Dataset cleaning error: ${error}`);  
 throw error;  
 }  
 }  
  
 // Additional methods for dataset management...  
}

/\*\*  
 \* Dataset Management Service  
 \* Provides methods for dataset creation, manipulation, and analysis  
 \*/  
export class DatasetManagementService {  
 /\*\*  
 \* Analyze a dataset to identify quality issues  
 \*   
 \* @param datasetId The ID of the dataset to analyze  
 \* @returns Dataset analysis results  
 \*/  
 public async analyzeDataset(datasetId: string): Promise<DatasetAnalysisResult> {  
 try {  
 const result: DatasetAnalysisResult = {  
 datasetId,  
 totalImages: 0,  
 issuesDetected: {  
 lowResolutionImages: { count: 0, details: [] },  
 poorQualityImages: { count: 0, details: [] },  
 duplicateImages: { count: 0, details: [] },  
 classImbalance: { totalClasses: 0, details: [] }  
 },  
 recommendations: []  
 };  
  
 // Get the dataset  
 const dataset = await supabaseDatasetService.getDatasetById(datasetId);  
 if (!dataset) {  
 throw new Error(`Dataset not found: ${datasetId}`);  
 }  
  
 // Get classes for the dataset  
 const classes = await supabaseDatasetService.getDatasetClasses(datasetId);  
 result.issuesDetected.classImbalance.totalClasses = classes.length;  
  
 // Analysis logic implementation...  
  
 return result;  
 } catch (error) {  
 logger.error(`Dataset analysis error: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Clean a dataset based on analysis results  
 \*   
 \* @param datasetId The ID of the dataset to clean  
 \* @param cleaningOptions Options for dataset cleaning  
 \* @returns Cleaning operation results  
 \*/  
 public async cleanDataset(  
 datasetId: string,  
 cleaningOptions: DatasetCleaningOptions  
 ): Promise<DatasetCleaningResult> {  
 try {  
 // Implementation details...  
  
 // Create target dataset if creating a new version  
 let targetDatasetId = datasetId;  
 let targetDataset = null;  
  
 if (createNewVersion) {  
 // Create a new dataset as a copy  
 targetDataset = await supabaseDatasetService.createDataset({  
 name: `${dataset.name} (Cleaned)`,  
 description: `Cleaned version of ${dataset.name}`,  
 sourceDatasetId: datasetId,  
 status: 'processing'  
 });  
  
 targetDatasetId = targetDataset.id;  
 }  
  
 // Copy and clean data  
 // Implementation details...  
  
 // Set the cleaned dataset status to ready  
 if (createNewVersion && targetDataset) {  
 await supabaseDatasetService.updateDataset(targetDatasetId, {  
 status: 'ready'  
 });  
 }  
  
 return result;  
 } catch (error) {  
 logger.error(`Dataset cleaning error: ${error}`);  
 throw error;  
 }  
 }  
  
 // Additional methods for dataset management...  
}

### Extended Vector Applications

The Supabase Vector database is integrated across multiple domains within the system, providing powerful semantic capabilities beyond basic search.

#### Query Understanding

The system leverages Supabase Vector to enhance search with semantic understanding:

1. Query Embedding and Storage
2. Natural language queries converted to vector embeddings
3. Storage of historical queries with metadata
4. Semantic clustering of query embeddings
5. Contextual understanding of search intent
6. Association with user context and session data
7. Semantic Enhancement
8. Query expansion based on vector similarity
9. Identification of semantically similar terms
10. Domain-specific concept linking
11. Personalized query interpretation
12. Context-aware search term weighting
13. Search Personalization
14. User preference learning through query vectors
15. Category and domain-specific personalization
16. Continuous adaptation to user behavior
17. Vector-based user intent modeling
18. Hybrid personalization combining explicit and implicit signals

Association with user context and session data

Semantic Enhancement

Context-aware search term weighting

Search Personalization

#### Material Recognition System

The Database Management and Vector DB components provide advanced material recognition capabilities:

1. Feature Vector Management
2. Material visual features stored as vector embeddings
3. Multi-dimensional feature space for material properties
4. Classification confidence through vector similarity
5. Feature extraction and storage pipeline
6. Automatic vector indexing for performance
7. Similarity Matching
8. Recognition based on visual feature similarity
9. Confidence scoring through vector distance metrics
10. Similar material identification
11. "More like this" functionality
12. Handling of material variations and lighting conditions
13. Feedback Mechanisms
14. User feedback integration for recognition improvement
15. Vector space adaptation based on feedback
16. Accuracy tracking by material type
17. Continuous model refinement
18. Recognition confidence threshold optimization

Automatic vector indexing for performance

Similarity Matching

Handling of material variations and lighting conditions

Feedback Mechanisms

#### Recommendation Engine

The system implements vector-based recommendation capabilities:

1. User Preference Modeling
2. User preferences encoded as vector embeddings
3. Interaction-based preference vector updates
4. Category weighting through vector components
5. User profile evolution over time
6. Multi-dimensional interest representation
7. Recommendation Generation
8. Material-user similarity through vector operations
9. Diversity balancing in recommendations
10. Personalized relevance scoring
11. Category-aware recommendations
12. Cold-start handling for new users
13. Feedback Integration
14. Preference vector updates based on interactions
15. Positive and negative feedback weighting
16. Duration-based influence for engagement
17. Share-based significance boosting
18. Progressive adaptation to changing preferences

Multi-dimensional interest representation

Recommendation Generation

Cold-start handling for new users

Feedback Integration

#### Document Processing

The system uses vector embeddings for advanced document management:

1. Document Content Vectorization
2. Text chunks converted to semantic vectors
3. Document section embedding with metadata
4. Hierarchical content representation
5. Cross-document semantic linking
6. Vector-based document organization
7. Semantic Search Capabilities
8. Meaning-based document retrieval
9. Context-aware search within documents
10. Conceptual matching beyond keywords
11. Relevance ranking with confidence scores
12. Query-section matching at paragraph level
13. Entity Recognition and Linking
14. Named entity vectorization in documents
15. Entity relationship mapping through vectors
16. Cross-document entity tracking
17. Confidence-based entity identification
18. Semantic entity categorization

Vector-based document organization

Semantic Search Capabilities

Query-section matching at paragraph level

Entity Recognition and Linking

### Integration with Core Systems

#### Material Recognition Pipeline

The Database Management and Vector DB components integrate with the material recognition system:

1. Vector Generation
2. Material images converted to vector embeddings
3. Storage in vector database tables
4. Indexing for efficient similarity search
5. Association with material metadata
6. Vector quality monitoring
7. Search Integration
8. Recognition results matched against vector database
9. Multi-stage search process for candidate identification
10. Combined scoring for result ranking
11. Confidence thresholds for result filtering
12. Performance metrics for search quality
13. Result Processing
14. Vector search results enhanced with material details
15. Similarity scores included with search results
16. Fallback to text search when needed
17. Multi-modal result presentation
18. Continuous improvement through feedback loop

Vector quality monitoring

Search Integration

Performance metrics for search quality

Result Processing

#### Knowledge Base System

The Database Management and Vector DB components support the knowledge base:

1. Knowledge Representation
2. Structured storage of domain knowledge
3. Vector embeddings for semantic understanding
4. Entity relationships for knowledge graph
5. Attribute indexing for faceted search
6. History tracking for knowledge evolution
7. Query Processing
8. Natural language query embedding
9. Vector similarity for semantic matching
10. Hybrid search for comprehensive results
11. Contextual relevance ranking
12. Explainable search results
13. Knowledge Update
14. Automatic vector embedding updates
15. Cascading updates for related entities
16. Transaction safety for knowledge consistency
17. Versioning for knowledge snapshots
18. Audit trail for knowledge modifications

History tracking for knowledge evolution

Query Processing

Explainable search results

Knowledge Update

#### Admin Panel

The Database Management and Vector DB components are exposed through the admin panel:

1. Database Management
2. Dataset creation and management interface
3. Import/export tools for data migration
4. Schema visualization and management
5. Query builder for advanced operations
6. Performance monitoring dashboard
7. Vector Operations
8. Vector index management tools
9. Embedding visualization and analysis
10. Similarity search testing interface
11. Vector quality assessment
12. Performance optimization controls
13. System Monitoring
14. Database health and performance metrics
15. Query performance analytics
16. Storage utilization tracking
17. Index efficiency monitoring
18. Scheduled maintenance management

Performance monitoring dashboard

Vector Operations

Performance optimization controls

System Monitoring

### API Usage Examples

#### Basic Vector Search

import { vectorSearch } from '@kai/server/services/supabase/vector-search';  
  
async function findSimilarMaterials(embeddingVector: number[]) {  
 try {  
 // Configure search parameters  
 const searchConfig = {  
 limit: 10, // Number of results to return  
 threshold: 0.75, // Minimum similarity threshold (0-1)  
 includeMetadata: true // Include associated metadata  
 };  
  
 // Perform vector search  
 const results = await vectorSearch.findSimilar(  
 embeddingVector,  
 'vector\_embeddings',  
 'embedding',  
 searchConfig  
 );  
  
 console.log(`Found ${results.length} similar materials`);  
  
 // Process search results  
 results.forEach(result => {  
 console.log(`Material ID: ${result.material\_id}`);  
 console.log(`Similarity: ${result.similarity.toFixed(4)}`);  
 console.log(`Metadata: ${JSON.stringify(result.metadata)}`);  
 });  
  
 return results;  
 } catch (error) {  
 console.error('Vector search failed:', error);  
 throw error;  
 }  
}

import { vectorSearch } from '@kai/server/services/supabase/vector-search';  
  
async function findSimilarMaterials(embeddingVector: number[]) {  
 try {  
 // Configure search parameters  
 const searchConfig = {  
 limit: 10, // Number of results to return  
 threshold: 0.75, // Minimum similarity threshold (0-1)  
 includeMetadata: true // Include associated metadata  
 };  
  
 // Perform vector search  
 const results = await vectorSearch.findSimilar(  
 embeddingVector,  
 'vector\_embeddings',  
 'embedding',  
 searchConfig  
 );  
  
 console.log(`Found ${results.length} similar materials`);  
  
 // Process search results  
 results.forEach(result => {  
 console.log(`Material ID: ${result.material\_id}`);  
 console.log(`Similarity: ${result.similarity.toFixed(4)}`);  
 console.log(`Metadata: ${JSON.stringify(result.metadata)}`);  
 });  
  
 return results;  
 } catch (error) {  
 console.error('Vector search failed:', error);  
 throw error;  
 }  
}

#### Hybrid Search Implementation

import { supabaseClient } from '@kai/server/services/supabase/supabaseClient';  
  
interface HybridSearchOptions {  
 textWeight?: number;  
 vectorWeight?: number;  
 limit?: number;  
 threshold?: number;  
}  
  
async function performHybridSearch(  
 textQuery: string,  
 embeddingVector: number[],  
 options: HybridSearchOptions = {}  
) {  
 try {  
 // Configure search parameters  
 const searchParams = {  
 textWeight: options.textWeight ?? 0.5,  
 vectorWeight: options.vectorWeight ?? 0.5,  
 limit: options.limit ?? 10,  
 threshold: options.threshold ?? 0.5  
 };  
  
 // Get Supabase client  
 const client = supabaseClient.getClient();  
  
 // Call hybrid search function  
 const { data, error } = await client.rpc('hybrid\_search\_materials', {  
 query\_text: textQuery,  
 query\_embedding: embeddingVector,  
 text\_weight: searchParams.textWeight,  
 vector\_weight: searchParams.vectorWeight,  
 match\_count: searchParams.limit,  
 score\_threshold: searchParams.threshold  
 });  
  
 if (error) throw error;  
  
 console.log(`Hybrid search found ${data.length} results`);  
  
 // Process search results  
 const enhancedResults = data.map(result => ({  
 id: result.id,  
 name: result.name,  
 description: result.description,  
 scores: {  
 textScore: result.text\_score,  
 vectorScore: result.vector\_score,  
 combinedScore: result.combined\_score  
 }  
 }));  
  
 return enhancedResults;  
 } catch (error) {  
 console.error('Hybrid search failed:', error);  
 throw error;  
 }  
}

import { supabaseClient } from '@kai/server/services/supabase/supabaseClient';  
  
interface HybridSearchOptions {  
 textWeight?: number;  
 vectorWeight?: number;  
 limit?: number;  
 threshold?: number;  
}  
  
async function performHybridSearch(  
 textQuery: string,  
 embeddingVector: number[],  
 options: HybridSearchOptions = {}  
) {  
 try {  
 // Configure search parameters  
 const searchParams = {  
 textWeight: options.textWeight ?? 0.5,  
 vectorWeight: options.vectorWeight ?? 0.5,  
 limit: options.limit ?? 10,  
 threshold: options.threshold ?? 0.5  
 };  
  
 // Get Supabase client  
 const client = supabaseClient.getClient();  
  
 // Call hybrid search function  
 const { data, error } = await client.rpc('hybrid\_search\_materials', {  
 query\_text: textQuery,  
 query\_embedding: embeddingVector,  
 text\_weight: searchParams.textWeight,  
 vector\_weight: searchParams.vectorWeight,  
 match\_count: searchParams.limit,  
 score\_threshold: searchParams.threshold  
 });  
  
 if (error) throw error;  
  
 console.log(`Hybrid search found ${data.length} results`);  
  
 // Process search results  
 const enhancedResults = data.map(result => ({  
 id: result.id,  
 name: result.name,  
 description: result.description,  
 scores: {  
 textScore: result.text\_score,  
 vectorScore: result.vector\_score,  
 combinedScore: result.combined\_score  
 }  
 }));  
  
 return enhancedResults;  
 } catch (error) {  
 console.error('Hybrid search failed:', error);  
 throw error;  
 }  
}

#### Dataset Management

import { DatasetManagementService } from '@kai/server/services/datasets/dataset-management.service';  
import supabaseDatasetService from '@kai/server/services/supabase/supabase-dataset-service';  
  
async function createAndManageDataset() {  
 try {  
 // Create a new dataset  
 const dataset = await supabaseDatasetService.createDataset({  
 name: 'Ceramic Tiles Training Set',  
 description: 'Training dataset for ceramic tile recognition',  
 status: 'processing'  
 });  
  
 console.log(`Created dataset with ID: ${dataset.id}`);  
  
 // Create dataset classes  
 const classes = [  
 { name: 'Porcelain', description: 'Porcelain ceramic tiles' },  
 { name: 'Terracotta', description: 'Terracotta clay tiles' },  
 { name: 'Marble', description: 'Marble effect ceramic tiles' }  
 ];  
  
 for (const cls of classes) {  
 const datasetClass = await supabaseDatasetService.createDatasetClass({  
 datasetId: dataset.id,  
 name: cls.name,  
 description: cls.description  
 });  
  
 console.log(`Created class: ${datasetClass.name} (${datasetClass.id})`);  
 }  
  
 // Update dataset status  
 await supabaseDatasetService.updateDataset(dataset.id, {  
 status: 'ready'  
 });  
  
 // Initialize dataset management service  
 const datasetManager = new DatasetManagementService();  
  
 // Analyze dataset quality  
 const analysisResult = await datasetManager.analyzeDataset(dataset.id);  
  
 console.log('Dataset analysis results:');  
 console.log(`- Total images: ${analysisResult.totalImages}`);  
 console.log(`- Low resolution images: ${analysisResult.issuesDetected.lowResolutionImages.count}`);  
 console.log(`- Class imbalance issues: ${analysisResult.issuesDetected.classImbalance.details.length}`);  
  
 // Clean dataset based on analysis  
 if (analysisResult.recommendations.length > 0) {  
 const cleaningResult = await datasetManager.cleanDataset(dataset.id, {  
 createNewVersion: true,  
 removeDuplicates: true,  
 balanceClasses: true,  
 removeCorruptedImages: true  
 });  
  
 console.log(`Dataset cleaned: ${cleaningResult.cleanedDatasetId}`);  
 console.log(`- Removed images: ${cleaningResult.removedImages}`);  
 console.log(`- Balanced classes: ${cleaningResult.balancedClasses}`);  
 }  
  
 return dataset.id;  
 } catch (error) {  
 console.error('Dataset management failed:', error);  
 throw error;  
 }  
}

import { DatasetManagementService } from '@kai/server/services/datasets/dataset-management.service';  
import supabaseDatasetService from '@kai/server/services/supabase/supabase-dataset-service';  
  
async function createAndManageDataset() {  
 try {  
 // Create a new dataset  
 const dataset = await supabaseDatasetService.createDataset({  
 name: 'Ceramic Tiles Training Set',  
 description: 'Training dataset for ceramic tile recognition',  
 status: 'processing'  
 });  
  
 console.log(`Created dataset with ID: ${dataset.id}`);  
  
 // Create dataset classes  
 const classes = [  
 { name: 'Porcelain', description: 'Porcelain ceramic tiles' },  
 { name: 'Terracotta', description: 'Terracotta clay tiles' },  
 { name: 'Marble', description: 'Marble effect ceramic tiles' }  
 ];  
  
 for (const cls of classes) {  
 const datasetClass = await supabaseDatasetService.createDatasetClass({  
 datasetId: dataset.id,  
 name: cls.name,  
 description: cls.description  
 });  
  
 console.log(`Created class: ${datasetClass.name} (${datasetClass.id})`);  
 }  
  
 // Update dataset status  
 await supabaseDatasetService.updateDataset(dataset.id, {  
 status: 'ready'  
 });  
  
 // Initialize dataset management service  
 const datasetManager = new DatasetManagementService();  
  
 // Analyze dataset quality  
 const analysisResult = await datasetManager.analyzeDataset(dataset.id);  
  
 console.log('Dataset analysis results:');  
 console.log(`- Total images: ${analysisResult.totalImages}`);  
 console.log(`- Low resolution images: ${analysisResult.issuesDetected.lowResolutionImages.count}`);  
 console.log(`- Class imbalance issues: ${analysisResult.issuesDetected.classImbalance.details.length}`);  
  
 // Clean dataset based on analysis  
 if (analysisResult.recommendations.length > 0) {  
 const cleaningResult = await datasetManager.cleanDataset(dataset.id, {  
 createNewVersion: true,  
 removeDuplicates: true,  
 balanceClasses: true,  
 removeCorruptedImages: true  
 });  
  
 console.log(`Dataset cleaned: ${cleaningResult.cleanedDatasetId}`);  
 console.log(`- Removed images: ${cleaningResult.removedImages}`);  
 console.log(`- Balanced classes: ${cleaningResult.balancedClasses}`);  
 }  
  
 return dataset.id;  
 } catch (error) {  
 console.error('Dataset management failed:', error);  
 throw error;  
 }  
}

### Implementation Examples

#### Query Understanding Service

/\*\*  
 \* Query Understanding Service  
 \* Enhances search by understanding the semantic meaning of queries using vector embeddings  
 \*/  
export class QueryUnderstandingService {  
 private embeddingTableName = 'semantic\_concepts';  
 private vectorColumnName = 'embedding';  
 private queryHistoryTableName = 'query\_history';  
  
 /\*\*  
 \* Process and enhance a search query using semantic understanding  
 \*/  
 public async enhanceQuery(  
 query: string,  
 options: QueryUnderstandingOptions = {},  
 context?: QueryContext  
 ): Promise<ExpandedQuery> {  
 try {  
 // Generate embedding for the query  
 const queryEmbedding = await this.generateQueryEmbedding(query);  
  
 // Find semantically similar concepts  
 const similarConcepts = await this.findSimilarConcepts(  
 queryEmbedding,   
 options.domainContext || 'general',  
 options.minConfidence || 0.7  
 );  
  
 // Extract related terms from similar concepts  
 const relatedTerms = this.extractRelatedTerms(similarConcepts, options.maxRelatedTerms || 5);  
  
 // Create enhanced query with synonyms if requested  
 let enhancedQuery = query;  
 if (options.expandSynonyms && similarConcepts.length > 0) {  
 enhancedQuery = this.expandWithSynonyms(query, similarConcepts);  
 }  
  
 // Personalize based on user context if available  
 if (context?.userId) {  
 enhancedQuery = await this.personalizeQuery(enhancedQuery, context);  
 }  
  
 // Store query in history  
 await this.storeQueryHistory(query, queryEmbedding, context?.userId);  
  
 return {  
 originalQuery: query,  
 enhancedQuery,  
 relatedTerms,  
 queryEmbedding,  
 confidence: similarConcepts.length > 0 ? similarConcepts[0].similarity : 0.5  
 };  
 } catch (error) {  
 logger.error(`Failed to enhance query: ${error}`);  
 // Return basic result if enhancement fails  
 return {  
 originalQuery: query,  
 enhancedQuery: query,  
 relatedTerms: [],  
 queryEmbedding: await this.generateMockEmbedding(),  
 confidence: 0  
 };  
 }  
 }  
  
 private async findSimilarConcepts(embedding: number[], domainContext: string, minConfidence: number) {  
 // Use vector search to find similar concepts  
 return await vectorSearch.findSimilar(  
 embedding,  
 this.embeddingTableName,  
 this.vectorColumnName,  
 {  
 threshold: minConfidence,  
 limit: 10,  
 filters: {  
 domain\_context: domainContext === 'general'   
 ? { $in: [domainContext, 'general'] }  
 : domainContext  
 }  
 }  
 );  
 }  
  
 // Additional implementation details...  
}

/\*\*  
 \* Query Understanding Service  
 \* Enhances search by understanding the semantic meaning of queries using vector embeddings  
 \*/  
export class QueryUnderstandingService {  
 private embeddingTableName = 'semantic\_concepts';  
 private vectorColumnName = 'embedding';  
 private queryHistoryTableName = 'query\_history';  
  
 /\*\*  
 \* Process and enhance a search query using semantic understanding  
 \*/  
 public async enhanceQuery(  
 query: string,  
 options: QueryUnderstandingOptions = {},  
 context?: QueryContext  
 ): Promise<ExpandedQuery> {  
 try {  
 // Generate embedding for the query  
 const queryEmbedding = await this.generateQueryEmbedding(query);  
  
 // Find semantically similar concepts  
 const similarConcepts = await this.findSimilarConcepts(  
 queryEmbedding,   
 options.domainContext || 'general',  
 options.minConfidence || 0.7  
 );  
  
 // Extract related terms from similar concepts  
 const relatedTerms = this.extractRelatedTerms(similarConcepts, options.maxRelatedTerms || 5);  
  
 // Create enhanced query with synonyms if requested  
 let enhancedQuery = query;  
 if (options.expandSynonyms && similarConcepts.length > 0) {  
 enhancedQuery = this.expandWithSynonyms(query, similarConcepts);  
 }  
  
 // Personalize based on user context if available  
 if (context?.userId) {  
 enhancedQuery = await this.personalizeQuery(enhancedQuery, context);  
 }  
  
 // Store query in history  
 await this.storeQueryHistory(query, queryEmbedding, context?.userId);  
  
 return {  
 originalQuery: query,  
 enhancedQuery,  
 relatedTerms,  
 queryEmbedding,  
 confidence: similarConcepts.length > 0 ? similarConcepts[0].similarity : 0.5  
 };  
 } catch (error) {  
 logger.error(`Failed to enhance query: ${error}`);  
 // Return basic result if enhancement fails  
 return {  
 originalQuery: query,  
 enhancedQuery: query,  
 relatedTerms: [],  
 queryEmbedding: await this.generateMockEmbedding(),  
 confidence: 0  
 };  
 }  
 }  
  
 private async findSimilarConcepts(embedding: number[], domainContext: string, minConfidence: number) {  
 // Use vector search to find similar concepts  
 return await vectorSearch.findSimilar(  
 embedding,  
 this.embeddingTableName,  
 this.vectorColumnName,  
 {  
 threshold: minConfidence,  
 limit: 10,  
 filters: {  
 domain\_context: domainContext === 'general'   
 ? { $in: [domainContext, 'general'] }  
 : domainContext  
 }  
 }  
 );  
 }  
  
 // Additional implementation details...  
}

#### Material Recognition Service

/\*\*  
 \* Material Recognition Service  
 \* Identifies materials based on their visual feature vectors using Supabase Vector  
 \*/  
export class MaterialRecognitionService {  
 private embeddingTableName = 'material\_feature\_vectors';  
 private vectorColumnName = 'embedding';  
 private recognitionHistoryTableName = 'recognition\_history';  
  
 /\*\*  
 \* Recognize a material from its feature vector  
 \*/  
 public async recognizeMaterial(  
 featureVector: number[],  
 options: RecognitionOptions = {}  
 ): Promise<MaterialRecognitionResult> {  
 try {  
 // Prepare filters  
 const filters: Record<string, any> = {  
 feature\_type: options.featureType || 'global'  
 };  
  
 // Add material type filter if specified  
 if (options.materialType) {  
 if (Array.isArray(options.materialType)) {  
 filters.material\_type = { $in: options.materialType };  
 } else {  
 filters.material\_type = options.materialType;  
 }  
 }  
  
 // Find similar feature vectors  
 const similarVectors = await vectorSearch.findSimilar(  
 featureVector,  
 this.embeddingTableName,  
 this.vectorColumnName,  
 {  
 threshold: options.minConfidence || 0.7,  
 limit: options.maxResults || 5,  
 filters  
 }  
 );  
  
 // Process recognition results  
 if (!similarVectors || similarVectors.length === 0) {  
 return {  
 materialId: '',  
 materialName: 'Unknown',  
 materialType: 'Unknown',  
 similarity: 0,  
 confidence: 0,  
 alternatives: []  
 };  
 }  
  
 // Get best match and prepare result  
 const bestMatch = similarVectors[0];  
 const result = this.prepareRecognitionResult(  
 bestMatch,   
 similarVectors,   
 options.includeAlternatives || true,  
 options.includeAttributes || true  
 );  
  
 // Log recognition result  
 await this.logRecognitionResult(featureVector, result);  
  
 return result;  
 } catch (error) {  
 logger.error(`Recognition failed: ${error}`);  
 throw new Error(`Recognition failed: ${error instanceof Error ? error.message : String(error)}`);  
 }  
 }  
  
 // Additional implementation details...  
}

/\*\*  
 \* Material Recognition Service  
 \* Identifies materials based on their visual feature vectors using Supabase Vector  
 \*/  
export class MaterialRecognitionService {  
 private embeddingTableName = 'material\_feature\_vectors';  
 private vectorColumnName = 'embedding';  
 private recognitionHistoryTableName = 'recognition\_history';  
  
 /\*\*  
 \* Recognize a material from its feature vector  
 \*/  
 public async recognizeMaterial(  
 featureVector: number[],  
 options: RecognitionOptions = {}  
 ): Promise<MaterialRecognitionResult> {  
 try {  
 // Prepare filters  
 const filters: Record<string, any> = {  
 feature\_type: options.featureType || 'global'  
 };  
  
 // Add material type filter if specified  
 if (options.materialType) {  
 if (Array.isArray(options.materialType)) {  
 filters.material\_type = { $in: options.materialType };  
 } else {  
 filters.material\_type = options.materialType;  
 }  
 }  
  
 // Find similar feature vectors  
 const similarVectors = await vectorSearch.findSimilar(  
 featureVector,  
 this.embeddingTableName,  
 this.vectorColumnName,  
 {  
 threshold: options.minConfidence || 0.7,  
 limit: options.maxResults || 5,  
 filters  
 }  
 );  
  
 // Process recognition results  
 if (!similarVectors || similarVectors.length === 0) {  
 return {  
 materialId: '',  
 materialName: 'Unknown',  
 materialType: 'Unknown',  
 similarity: 0,  
 confidence: 0,  
 alternatives: []  
 };  
 }  
  
 // Get best match and prepare result  
 const bestMatch = similarVectors[0];  
 const result = this.prepareRecognitionResult(  
 bestMatch,   
 similarVectors,   
 options.includeAlternatives || true,  
 options.includeAttributes || true  
 );  
  
 // Log recognition result  
 await this.logRecognitionResult(featureVector, result);  
  
 return result;  
 } catch (error) {  
 logger.error(`Recognition failed: ${error}`);  
 throw new Error(`Recognition failed: ${error instanceof Error ? error.message : String(error)}`);  
 }  
 }  
  
 // Additional implementation details...  
}

#### Recommendation Engine

/\*\*  
 \* Recommendation Engine  
 \* Provides personalized material recommendations based on user preferences  
 \* and material feature vectors using Supabase Vector similarity search  
 \*/  
export class RecommendationEngine {  
 private userPreferenceTableName = 'user\_preference\_vectors';  
 private materialFeatureTableName = 'material\_feature\_vectors';  
 private interactionHistoryTableName = 'user\_material\_interactions';  
 private vectorColumnName = 'embedding';  
  
 /\*\*  
 \* Get personalized recommendations for a user  
 \*/  
 public async getRecommendations(  
 options: RecommendationOptions  
 ): Promise<RecommendationResult[]> {  
 try {  
 // Get user preference vector  
 const userPreference = await this.getUserPreference(options.userId);  
  
 if (!userPreference) {  
 // Fall back to general popularity-based recommendations for new users  
 return this.getPopularRecommendations(  
 options.count || 10,  
 options.materialTypes || [],  
 options.excludeMaterialIds || [],  
 options.categoryFilter  
 );  
 }  
  
 // Prepare filters for material search  
 const filters: Record<string, any> = {};  
  
 if (options.materialTypes?.length) {  
 filters.material\_type = { $in: options.materialTypes };  
 }  
  
 if (options.categoryFilter) {  
 filters.category = options.categoryFilter;  
 }  
  
 if (options.excludeMaterialIds?.length) {  
 filters.material\_id = { $nin: options.excludeMaterialIds };  
 }  
  
 // Find similar materials based on user preference vector  
 const similarMaterials = await vectorSearch.findSimilar(  
 userPreference.preferenceVector,  
 this.materialFeatureTableName,  
 this.vectorColumnName,  
 {  
 threshold: options.minRelevance || 0.6,  
 limit: (options.count || 10) \* 2, // Get more than needed for diversity filtering  
 filters  
 }  
 );  
  
 // Process and return recommendations  
 const recommendations = this.processRecommendations(  
 similarMaterials,  
 options.diversityFactor || 0.3,  
 options.count || 10,  
 options.includeExplanations || true,  
 userPreference  
 );  
  
 return recommendations;  
 } catch (error) {  
 logger.error(`Error getting recommendations: ${error}`);  
 throw new Error(`Error getting recommendations: ${error instanceof Error ? error.message : String(error)}`);  
 }  
 }  
  
 // Additional implementation details...  
}

/\*\*  
 \* Recommendation Engine  
 \* Provides personalized material recommendations based on user preferences  
 \* and material feature vectors using Supabase Vector similarity search  
 \*/  
export class RecommendationEngine {  
 private userPreferenceTableName = 'user\_preference\_vectors';  
 private materialFeatureTableName = 'material\_feature\_vectors';  
 private interactionHistoryTableName = 'user\_material\_interactions';  
 private vectorColumnName = 'embedding';  
  
 /\*\*  
 \* Get personalized recommendations for a user  
 \*/  
 public async getRecommendations(  
 options: RecommendationOptions  
 ): Promise<RecommendationResult[]> {  
 try {  
 // Get user preference vector  
 const userPreference = await this.getUserPreference(options.userId);  
  
 if (!userPreference) {  
 // Fall back to general popularity-based recommendations for new users  
 return this.getPopularRecommendations(  
 options.count || 10,  
 options.materialTypes || [],  
 options.excludeMaterialIds || [],  
 options.categoryFilter  
 );  
 }  
  
 // Prepare filters for material search  
 const filters: Record<string, any> = {};  
  
 if (options.materialTypes?.length) {  
 filters.material\_type = { $in: options.materialTypes };  
 }  
  
 if (options.categoryFilter) {  
 filters.category = options.categoryFilter;  
 }  
  
 if (options.excludeMaterialIds?.length) {  
 filters.material\_id = { $nin: options.excludeMaterialIds };  
 }  
  
 // Find similar materials based on user preference vector  
 const similarMaterials = await vectorSearch.findSimilar(  
 userPreference.preferenceVector,  
 this.materialFeatureTableName,  
 this.vectorColumnName,  
 {  
 threshold: options.minRelevance || 0.6,  
 limit: (options.count || 10) \* 2, // Get more than needed for diversity filtering  
 filters  
 }  
 );  
  
 // Process and return recommendations  
 const recommendations = this.processRecommendations(  
 similarMaterials,  
 options.diversityFactor || 0.3,  
 options.count || 10,  
 options.includeExplanations || true,  
 userPreference  
 );  
  
 return recommendations;  
 } catch (error) {  
 logger.error(`Error getting recommendations: ${error}`);  
 throw new Error(`Error getting recommendations: ${error instanceof Error ? error.message : String(error)}`);  
 }  
 }  
  
 // Additional implementation details...  
}

#### Document Processing Service

/\*\*  
 \* Document Processing Service  
 \* Leverages Supabase Vector to enable semantic search across document repositories  
 \*/  
export class DocumentProcessingService {  
 private documentsTableName = 'documents';  
 private documentChunksTableName = 'document\_chunks';  
 private entitiesTableName = 'document\_entities';  
 private vectorColumnName = 'embedding';  
  
 /\*\*  
 \* Process a document and store it with vector embeddings  
 \*/  
 public async processDocument(  
 documentContent: string,  
 metadata: Omit<DocumentMetadata, 'id'>  
 ): Promise<string> {  
 try {  
 // Generate document ID  
 const documentId = uuidv4();  
  
 // Store document metadata  
 await this.storeDocumentMetadata({  
 id: documentId,  
 ...metadata  
 });  
  
 // Split document into chunks  
 const chunks = this.splitDocumentIntoChunks(documentContent);  
  
 // Process chunks with embeddings  
 await this.processDocumentChunks(documentId, chunks);  
  
 // Extract and store entities  
 await this.extractAndStoreEntities(documentId, documentContent, metadata.title);  
  
 return documentId;  
 } catch (error) {  
 logger.error(`Document processing failed: ${error}`);  
 throw new Error(`Document processing failed: ${error instanceof Error ? error.message : String(error)}`);  
 }  
 }  
  
 /\*\*  
 \* Search documents using semantic search  
 \*/  
 public async searchDocuments(  
 options: DocumentSearchOptions  
 ): Promise<DocumentSearchResult[]> {  
 try {  
 // Generate embedding for the query  
 const queryEmbedding = await this.generateTextEmbedding(options.query);  
  
 // Get document IDs that match metadata filters  
 const documentIds = await this.getFilteredDocumentIds(  
 options.fileTypes,  
 options.dateRange,  
 options.categories,  
 options.tags,  
 options.author,  
 options.uploadedBy  
 );  
  
 // Prepare filters for vector search  
 const filters: Record<string, any> = {};  
  
 if (documentIds.length > 0) {  
 filters.document\_id = { $in: documentIds };  
 }  
  
 // Find similar chunks  
 const similarChunks = await vectorSearch.findSimilar(  
 queryEmbedding,  
 this.documentChunksTableName,  
 this.vectorColumnName,  
 {  
 threshold: options.minRelevance || 0.6,  
 limit: (options.limit || 10) \* 3,  
 filters  
 }  
 );  
  
 // Process and return search results  
 if (!similarChunks || similarChunks.length === 0) {  
 return this.fallbackKeywordSearch(  
 options.query,  
 options.limit || 10,  
 options.offset || 0,  
 documentIds,  
 options.includeMetadata || true,  
 options.highlightResults || true  
 );  
 }  
  
 return this.processSearchResults(  
 similarChunks,  
 options.includeMetadata || true,  
 options.highlightResults || true,  
 options.limit || 10,  
 options.offset || 0  
 );  
 } catch (error) {  
 logger.error(`Document search failed: ${error}`);  
 throw new Error(`Document search failed: ${error instanceof Error ? error.message : String(error)}`);  
 }  
 }  
  
 // Additional implementation details...  
}

/\*\*  
 \* Document Processing Service  
 \* Leverages Supabase Vector to enable semantic search across document repositories  
 \*/  
export class DocumentProcessingService {  
 private documentsTableName = 'documents';  
 private documentChunksTableName = 'document\_chunks';  
 private entitiesTableName = 'document\_entities';  
 private vectorColumnName = 'embedding';  
  
 /\*\*  
 \* Process a document and store it with vector embeddings  
 \*/  
 public async processDocument(  
 documentContent: string,  
 metadata: Omit<DocumentMetadata, 'id'>  
 ): Promise<string> {  
 try {  
 // Generate document ID  
 const documentId = uuidv4();  
  
 // Store document metadata  
 await this.storeDocumentMetadata({  
 id: documentId,  
 ...metadata  
 });  
  
 // Split document into chunks  
 const chunks = this.splitDocumentIntoChunks(documentContent);  
  
 // Process chunks with embeddings  
 await this.processDocumentChunks(documentId, chunks);  
  
 // Extract and store entities  
 await this.extractAndStoreEntities(documentId, documentContent, metadata.title);  
  
 return documentId;  
 } catch (error) {  
 logger.error(`Document processing failed: ${error}`);  
 throw new Error(`Document processing failed: ${error instanceof Error ? error.message : String(error)}`);  
 }  
 }  
  
 /\*\*  
 \* Search documents using semantic search  
 \*/  
 public async searchDocuments(  
 options: DocumentSearchOptions  
 ): Promise<DocumentSearchResult[]> {  
 try {  
 // Generate embedding for the query  
 const queryEmbedding = await this.generateTextEmbedding(options.query);  
  
 // Get document IDs that match metadata filters  
 const documentIds = await this.getFilteredDocumentIds(  
 options.fileTypes,  
 options.dateRange,  
 options.categories,  
 options.tags,  
 options.author,  
 options.uploadedBy  
 );  
  
 // Prepare filters for vector search  
 const filters: Record<string, any> = {};  
  
 if (documentIds.length > 0) {  
 filters.document\_id = { $in: documentIds };  
 }  
  
 // Find similar chunks  
 const similarChunks = await vectorSearch.findSimilar(  
 queryEmbedding,  
 this.documentChunksTableName,  
 this.vectorColumnName,  
 {  
 threshold: options.minRelevance || 0.6,  
 limit: (options.limit || 10) \* 3,  
 filters  
 }  
 );  
  
 // Process and return search results  
 if (!similarChunks || similarChunks.length === 0) {  
 return this.fallbackKeywordSearch(  
 options.query,  
 options.limit || 10,  
 options.offset || 0,  
 documentIds,  
 options.includeMetadata || true,  
 options.highlightResults || true  
 );  
 }  
  
 return this.processSearchResults(  
 similarChunks,  
 options.includeMetadata || true,  
 options.highlightResults || true,  
 options.limit || 10,  
 options.offset || 0  
 );  
 } catch (error) {  
 logger.error(`Document search failed: ${error}`);  
 throw new Error(`Document search failed: ${error instanceof Error ? error.message : String(error)}`);  
 }  
 }  
  
 // Additional implementation details...  
}

### Performance Considerations

1. Database Scaling
2. Vertical scaling for higher memory/CPU needs
3. Connection pooling for concurrent access
4. Read replicas for query distribution
5. Supabase tier selection based on data volume
6. Automated cleanup of temporary data
7. Vector Optimization
8. Dimensionality reduction for large vector spaces
9. Index parameter tuning for search performance
10. Batched vector operations for bulk processing
11. Vector storage compression
12. Cached vector computations for frequent queries
13. Multiple vector indices for different application domains
14. Selective pruning of outdated vectors
15. Parameterized indices based on domain requirements
16. Search Performance
17. Query optimization for complex searches
18. Result caching for common queries
19. Pagination for large result sets
20. Background indexing for vector updates
21. Asynchronous search for UI responsiveness
22. Resource Requirements
23. CPU: Multi-core recommended for vector operations
24. Memory: 8GB+ for larger vector indices
25. Storage: Scales with dataset size (approx. 5-10x raw data)
26. Network: Standard database bandwidth requirements
27. Backup: Regular point-in-time snapshots
28. Monitoring and Maintenance
29. Index rebuilding on significant data changes
30. Regular VACUUM operations for PostgreSQL
31. Performance tracking for query execution times
32. Storage utilization monitoring
33. Connection usage tracking

Automated cleanup of temporary data

Vector Optimization

Parameterized indices based on domain requirements

Search Performance

Asynchronous search for UI responsiveness

Resource Requirements

Backup: Regular point-in-time snapshots

Monitoring and Maintenance

# Datasets And Models

Source: readme/datasets-and-models.md

---

## Datasets and AI Model Integration

This document outlines the system's capabilities for working with datasets and AI models, including importing premade datasets and pre-trained models, and connecting them for training and inference.

### Overview

The Kai platform supports multiple ways to manage datasets and AI models:

1. Dataset Management
2. Upload custom datasets (ZIP archives with organized folders or CSV mapping files)
3. Generate vector embeddings from uploaded images for knowledge base and training
4. Import premade datasets from public repositories
5. Data quality assessment and enhancement
6. Dataset versioning and metadata management
7. Model Management
8. Upload custom trained models
9. Import pre-trained models from repositories
10. Model versioning and performance tracking
11. Transfer learning and fine-tuning capabilities
12. Training Pipeline
13. Connect datasets with models for training
14. Configure training parameters and techniques
15. Monitor progress with real-time metrics
16. Evaluate performance after completion

Dataset versioning and metadata management

Model Management

Transfer learning and fine-tuning capabilities

Training Pipeline

### Custom Dataset Upload

The system supports uploading custom datasets directly using ZIP files or CSV mapping files.

#### ZIP Archive Upload with Vector Embedding Generation

You can upload a ZIP archive containing organized image folders, with each folder representing a dataset class. The system will:

1. Extract images from the ZIP archive
2. Organize them into dataset classes
3. Generate vector embeddings for each image
4. Store both the images and their embeddings in the database
5. Make them available for both the knowledge base and model training

##### How to Upload a ZIP Dataset with Embeddings

1. Navigate to Admin Panel → Datasets → Upload Dataset
2. Select the ZIP Upload tab
3. Choose your ZIP file containing organized image folders
4. Enable the Generate Vector Embeddings option
5. Configure additional options if needed
6. Click Upload and Process

// Client-side code example for uploading a ZIP dataset with embeddings  
async function uploadZipDatasetWithEmbeddings(zipFile, options) {  
 try {  
 const formData = new FormData();  
 formData.append('file', zipFile);  
 formData.append('name', options.name);  
 formData.append('description', options.description);  
 formData.append('generateEmbeddings', 'true'); // Enable embedding generation  
  
 const response = await fetch('/api/admin/datasets/upload/zip', {  
 method: 'POST',  
 body: formData,  
 });  
  
 if (!response.ok) {  
 throw new Error('Failed to upload dataset');  
 }  
  
 const result = await response.json();  
 console.log(`Successfully processed ${result.imageCount} images across ${result.classCount} classes`);  
 console.log(`Generated ${result.embeddingsGenerated} vector embeddings`);  
  
 return result;  
 } catch (error) {  
 console.error('Error uploading ZIP dataset:', error);  
 throw error;  
 }  
}

// Client-side code example for uploading a ZIP dataset with embeddings  
async function uploadZipDatasetWithEmbeddings(zipFile, options) {  
 try {  
 const formData = new FormData();  
 formData.append('file', zipFile);  
 formData.append('name', options.name);  
 formData.append('description', options.description);  
 formData.append('generateEmbeddings', 'true'); // Enable embedding generation  
  
 const response = await fetch('/api/admin/datasets/upload/zip', {  
 method: 'POST',  
 body: formData,  
 });  
  
 if (!response.ok) {  
 throw new Error('Failed to upload dataset');  
 }  
  
 const result = await response.json();  
 console.log(`Successfully processed ${result.imageCount} images across ${result.classCount} classes`);  
 console.log(`Generated ${result.embeddingsGenerated} vector embeddings`);  
  
 return result;  
 } catch (error) {  
 console.error('Error uploading ZIP dataset:', error);  
 throw error;  
 }  
}

##### ZIP Dataset Structure

For best results, organize your ZIP file with the following structure:

dataset.zip  
├── class1/  
│ ├── image1.jpg  
│ ├── image2.png  
│ └── ...  
├── class2/  
│ ├── image1.jpg  
│ ├── image2.png  
│ └── ...  
└── ...

dataset.zip  
├── class1/  
│ ├── image1.jpg  
│ ├── image2.png  
│ └── ...  
├── class2/  
│ ├── image1.jpg  
│ ├── image2.png  
│ └── ...  
└── ...

Each top-level folder becomes a dataset class. Supported image formats include JPG, JPEG, PNG, WEBP, and GIF.

##### Generated Embeddings

The vector embeddings generated from your uploaded images are:

* Stored in the vector database for similarity search
* Available for both knowledge base integration and model training
* Accessible through the API for custom applications
* Used to improve material recognition accuracy

##### Vector Embedding Options

When uploading a ZIP file with embedding generation enabled, you can configure:

* Embedding Dimension: Control the vector size (default: 256)
* Embedding Method: Choose between hybrid, feature-based, or ML-based approaches
* Quality Threshold: Set minimum quality requirements for generated embeddings

### Premade Datasets Integration

The system provides a streamlined workflow for integrating public, premade datasets into your recognition pipeline.

#### Supported Dataset Repositories

The system integrates with popular dataset repositories:

Table content:

Repository | Description | Dataset Types

Kaggle Datasets | Public datasets with various licensing | General, materials, textures

TensorFlow Datasets | Ready-to-use datasets from TensorFlow | Classification, recognition

Hugging Face Datasets | Public ML datasets | Classification, multi-modal

ImageNet | Standard computer vision dataset | Object recognition

#### Importing a Premade Dataset

To import a premade dataset:

1. Navigate to Admin Panel → Datasets → Upload Dataset
2. Select the Premade Dataset tab
3. Browse available datasets or search by name/category
4. Select the dataset and configure import options
5. Click Import Dataset

// Client-side code example for importing a premade dataset  
async function importPremadeDataset(datasetId, options) {  
 try {  
 const response = await fetch('/api/admin/datasets/import/premade', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify({  
 sourceDatasetId: datasetId,  
 includeMetadata: options.includeMetadata,  
 selectedClasses: options.selectedClasses,  
 name: options.name,  
 description: options.description  
 }),  
 });  
  
 if (!response.ok) {  
 throw new Error('Failed to import dataset');  
 }  
  
 const result = await response.json();  
 return result;  
 } catch (error) {  
 console.error('Error importing premade dataset:', error);  
 throw error;  
 }  
}

// Client-side code example for importing a premade dataset  
async function importPremadeDataset(datasetId, options) {  
 try {  
 const response = await fetch('/api/admin/datasets/import/premade', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify({  
 sourceDatasetId: datasetId,  
 includeMetadata: options.includeMetadata,  
 selectedClasses: options.selectedClasses,  
 name: options.name,  
 description: options.description  
 }),  
 });  
  
 if (!response.ok) {  
 throw new Error('Failed to import dataset');  
 }  
  
 const result = await response.json();  
 return result;  
 } catch (error) {  
 console.error('Error importing premade dataset:', error);  
 throw error;  
 }  
}

#### Dataset Import Options

When importing a premade dataset, you can configure:

* Dataset Name: Custom name for the imported dataset (defaults to original name)
* Description: Optional description for reference
* Include Metadata: Import annotations, labels, and other metadata
* Class Selection: Choose specific classes to import (subset of the dataset)
* Sample Limitation: Limit the number of samples per class

#### Dataset Quality Verification

After importing a premade dataset, the system automatically analyzes its quality:

// Server-side code for dataset quality analysis  
import { datasetManagementService } from '@kai/server/services/datasets/dataset-management.service';  
  
async function verifyImportedDataset(datasetId: string) {  
 // Analyze dataset quality  
 const qualityMetrics = await datasetManagementService.analyzeDatasetQuality(datasetId);  
  
 // If quality issues are detected, provide recommendations  
 if (qualityMetrics.overallQualityScore < 70) {  
 return {  
 needsImprovement: true,  
 recommendations: qualityMetrics.recommendations,  
 metrics: qualityMetrics  
 };  
 }  
  
 return {  
 needsImprovement: false,  
 metrics: qualityMetrics  
 };  
}

// Server-side code for dataset quality analysis  
import { datasetManagementService } from '@kai/server/services/datasets/dataset-management.service';  
  
async function verifyImportedDataset(datasetId: string) {  
 // Analyze dataset quality  
 const qualityMetrics = await datasetManagementService.analyzeDatasetQuality(datasetId);  
  
 // If quality issues are detected, provide recommendations  
 if (qualityMetrics.overallQualityScore < 70) {  
 return {  
 needsImprovement: true,  
 recommendations: qualityMetrics.recommendations,  
 metrics: qualityMetrics  
 };  
 }  
  
 return {  
 needsImprovement: false,  
 metrics: qualityMetrics  
 };  
}

### AI Models Integration

The KAI platform integrates a comprehensive suite of AI models for various tasks, from material recognition to 3D visualization. This section details the models used and their specific purposes.

#### AI Model Registry

The platform includes a sophisticated model registry that manages model selection, performance tracking, and evaluation across different providers (OpenAI, Anthropic, HuggingFace, and local models). The system automatically selects the best model for each task based on performance metrics including accuracy, latency, and cost.

#### Text and Language Models

##### OpenAI Models

* GPT-4 (gpt-4)
* Purpose: Primary text generation model
* Used for: General text generation, complex reasoning, and high-quality responses
* Default for: Text generation tasks
* GPT-4 Turbo (gpt-4-turbo)
* Purpose: Faster version of GPT-4
* Used for: Agent-based interactions requiring quick responses
* Default for: Chat completions in agent system
* GPT-4 Vision (gpt-4-vision)
* Purpose: Multimodal model that can process both text and images
* Used for: Material recognition from images
* Default for: Material recognition tasks
* GPT-3.5 Turbo (gpt-3.5-turbo)
* Purpose: Faster, more cost-effective model for simpler tasks
* Used for: Classification, summarization, and translation tasks
* Default for: Classification, summarization, and translation

gpt-4

Default for: Text generation tasks

GPT-4 Turbo (gpt-4-turbo)

gpt-4-turbo

Default for: Chat completions in agent system

GPT-4 Vision (gpt-4-vision)

gpt-4-vision

Default for: Material recognition tasks

GPT-3.5 Turbo (gpt-3.5-turbo)

gpt-3.5-turbo

##### Anthropic Models

* Claude 2 (claude-2)
* Purpose: Alternative text generation model
* Used for: Text generation, classification, summarization, and translation
* Default for: Anthropic text tasks
* Claude 3 Opus (claude-3-opus-20240229)
* Purpose: High-quality text generation with multimodal capabilities
* Used for: 3D designer agent

claude-2

Default for: Anthropic text tasks

Claude 3 Opus (claude-3-opus-20240229)

claude-3-opus-20240229

##### Hugging Face Text Models

* Google/Flan-T5-XXL (google/flan-t5-xxl)
* Purpose: Text generation alternative to GPT models
* Used for: Text generation when OpenAI is unavailable
* Facebook/BART-Large-MNLI (facebook/bart-large-mnli)
* Purpose: Zero-shot text classification
* Used for: Classifying text without specific training
* Facebook/BART-Large-CNN (facebook/bart-large-cnn)
* Purpose: Text summarization
* Used for: Creating concise summaries of longer text
* Facebook/MBART-Large-50-Many-to-Many-MMT (facebook/mbart-large-50-many-to-many-mmt)
* Purpose: Multilingual translation
* Used for: Translating text between languages

google/flan-t5-xxl

Used for: Text generation when OpenAI is unavailable

Facebook/BART-Large-MNLI (facebook/bart-large-mnli)

facebook/bart-large-mnli

Used for: Classifying text without specific training

Facebook/BART-Large-CNN (facebook/bart-large-cnn)

facebook/bart-large-cnn

Used for: Creating concise summaries of longer text

Facebook/MBART-Large-50-Many-to-Many-MMT (facebook/mbart-large-50-many-to-many-mmt)

facebook/mbart-large-50-many-to-many-mmt

#### Embedding Models

* Text Embedding Ada 002 (text-embedding-ada-002)
* Purpose: Generate text embeddings for vector search
* Used for: Creating searchable vector representations of text
* Default for: Embedding generation in OpenAI
* Text Embedding 3 Small (text-embedding-3-small)
* Purpose: Newer, more efficient embedding model
* Used for: Default embedding model in agent system
* Sentence-Transformers/All-MiniLM-L6-v2 (sentence-transformers/all-MiniLM-L6-v2)
* Purpose: Generate text embeddings
* Used for: Creating vector representations for similarity search
* Default for: Hugging Face embedding tasks

text-embedding-ada-002

Default for: Embedding generation in OpenAI

Text Embedding 3 Small (text-embedding-3-small)

text-embedding-3-small

Used for: Default embedding model in agent system

Sentence-Transformers/All-MiniLM-L6-v2 (sentence-transformers/all-MiniLM-L6-v2)

sentence-transformers/all-MiniLM-L6-v2

#### Computer Vision Models

##### Material Recognition

* Google/ViT-Base-Patch16-224 (google/vit-base-patch16-224)
* Purpose: Vision Transformer for image analysis
* Used for: Material recognition from images
* EfficientNet (B0-B5 variants)
* Purpose: Image classification
* Used for: Material classification
* ResNet (18, 34, 50, 101, 152 variants)
* Purpose: Image classification with residual connections
* Used for: Material classification
* MobileNet (V2, V3Small, V3Large variants)
* Purpose: Lightweight image classification
* Used for: Mobile/edge device material classification

google/vit-base-patch16-224

Used for: Material recognition from images

EfficientNet (B0-B5 variants)

Used for: Material classification

ResNet (18, 34, 50, 101, 152 variants)

Used for: Material classification

MobileNet (V2, V3Small, V3Large variants)

##### Object Detection and Segmentation

* YOLO v8 (yolov8)
* Purpose: Object detection
* Used for: Detecting objects in images
* Default for: Local object detection
* Segment Anything Model (SAM)
* Purpose: Image segmentation
* Used for: Segmenting images into regions
* MiDaS
* Purpose: Depth estimation
* Used for: Estimating depth from single images
* CLIP
* Purpose: Connecting text and images
* Used for: Cross-modal understanding and validation

yolov8

Default for: Local object detection

Segment Anything Model (SAM)

Used for: Segmenting images into regions

MiDaS

Used for: Estimating depth from single images

CLIP

#### 3D Reconstruction and Visualization Models

##### NeRF-based Reconstruction

* NeRF Studio
* Format: Custom NeRF format
* Purpose: Neural Radiance Field implementation
* Used for: 3D reconstruction from images
* Integration: Via Python API
* Instant-NGP
* Format: .ngp
* Purpose: Fast Neural Graphics Primitives
* Used for: Accelerated 3D reconstruction
* Integration: CUDA-accelerated backend
* COLMAP
* Purpose: Structure-from-Motion
* Used for: Camera pose estimation and 3D reconstruction

Integration: Via Python API

Instant-NGP

Integration: CUDA-accelerated backend

COLMAP

##### Text-to-3D Generation

* Shap-E
* Format: PyTorch (.pth)
* Purpose: Text-to-3D base structure generation
* Used for: Creating basic 3D shapes from text descriptions
* Integration: REST API
* GET3D
* Format: PyTorch (.pth)
* Purpose: Text-to-3D detailed scene generation
* Used for: Creating detailed 3D scenes from text
* Integration: Python API
* Hunyuan3D
* Format: Custom (.h3d)
* Purpose: Alternative text-to-3D model
* Used for: Creating 3D models from text descriptions
* Integration: REST API

Integration: REST API

GET3D

Integration: Python API

Hunyuan3D

#### Material and Style Models

* ControlNet
* Purpose: Controlled image generation
* Used for: House outline generation
* Text2Material
* Purpose: Generate material textures from text
* Used for: Creating material textures based on descriptions
* HDRNet
* Purpose: Lighting estimation and HDR environment maps
* Used for: Creating realistic lighting for 3D scenes

Used for: House outline generation

Text2Material

Used for: Creating material textures based on descriptions

HDRNet

#### Interior Design Models

* SpaceFormer
* Purpose: Room layout and furniture placement
* Used for: Automated interior design
* Architectural Recognition
* Purpose: Process architectural drawings
* Used for: Understanding floor plans and architectural designs
* Room Layout Generator
* Purpose: Generate room layouts
* Used for: Creating new room designs

Used for: Automated interior design

Architectural Recognition

Used for: Understanding floor plans and architectural designs

Room Layout Generator

#### Local Models

* Llama-2-13B-Chat (llama-2-13b-chat)
* Purpose: Local text generation
* Used for: Text generation when cloud services are unavailable
* DistilBART-MNLI (distilbart-mnli)
* Purpose: Lightweight text classification
* Used for: Local text classification
* DistilBART-CNN (distilbart-cnn)
* Purpose: Lightweight text summarization
* Used for: Local text summarization

llama-2-13b-chat

Used for: Text generation when cloud services are unavailable

DistilBART-MNLI (distilbart-mnli)

distilbart-mnli

Used for: Local text classification

DistilBART-CNN (distilbart-cnn)

distilbart-cnn

#### Model Configuration

Example configuration for model endpoints:

interface ModelEndpoints {  
 // Image-based reconstruction  
 nerfStudio: string;  
 instantNgp: string;  
 blenderProc: string;  
  
 // Text-based generation  
 shapE: string;  
 get3d: string;  
 hunyuan3d: string;  
  
 // Scene understanding  
 yolo: string; // Object detection  
 sam: string; // Segmentation  
 midas: string; // Depth estimation  
  
 // Architectural  
 architecturalRecognition: string;  
 roomLayoutGenerator: string;  
  
 // Material and style  
 controlNet: string;  
 text2material: string;  
 clip: string;  
 hdrnet: string; // For lighting estimation and HDR environment maps  
  
 // Camera pose estimation and multi-view consistency  
 colmap: string; // For Structure-from-Motion processing  
  
 // Interior design automation  
 spaceFormer: string; // For room layout and furniture placement  
}  
  
const modelConfig: ModelEndpoints = {  
 nerfStudio: process.env.NERF\_STUDIO\_ENDPOINT,  
 instantNgp: process.env.INSTANT\_NGP\_ENDPOINT,  
 shapE: process.env.SHAPE\_E\_ENDPOINT,  
 get3d: process.env.GET3D\_ENDPOINT,  
 hunyuan3d: process.env.HUNYUAN3D\_ENDPOINT,  
 blenderProc: process.env.BLENDER\_PROC\_ENDPOINT,  
 // ... additional endpoints  
};

interface ModelEndpoints {  
 // Image-based reconstruction  
 nerfStudio: string;  
 instantNgp: string;  
 blenderProc: string;  
  
 // Text-based generation  
 shapE: string;  
 get3d: string;  
 hunyuan3d: string;  
  
 // Scene understanding  
 yolo: string; // Object detection  
 sam: string; // Segmentation  
 midas: string; // Depth estimation  
  
 // Architectural  
 architecturalRecognition: string;  
 roomLayoutGenerator: string;  
  
 // Material and style  
 controlNet: string;  
 text2material: string;  
 clip: string;  
 hdrnet: string; // For lighting estimation and HDR environment maps  
  
 // Camera pose estimation and multi-view consistency  
 colmap: string; // For Structure-from-Motion processing  
  
 // Interior design automation  
 spaceFormer: string; // For room layout and furniture placement  
}  
  
const modelConfig: ModelEndpoints = {  
 nerfStudio: process.env.NERF\_STUDIO\_ENDPOINT,  
 instantNgp: process.env.INSTANT\_NGP\_ENDPOINT,  
 shapE: process.env.SHAPE\_E\_ENDPOINT,  
 get3d: process.env.GET3D\_ENDPOINT,  
 hunyuan3d: process.env.HUNYUAN3D\_ENDPOINT,  
 blenderProc: process.env.BLENDER\_PROC\_ENDPOINT,  
 // ... additional endpoints  
};

#### Supported Model Repositories

Models can be imported from popular repositories:

Table content:

Repository | Description | Access Method

HuggingFace Hub | Collection of community models | Direct import via model ID

TensorFlow Hub | Reusable ML modules | URL-based import

PyTorch Hub | Pretrained PyTorch models | Model name reference

ONNX Model Zoo | Collection of ONNX models | GitHub reference

#### Importing a Pre-trained Model

To import a pre-trained model:

1. Navigate to Admin Panel → Models → Import Model
2. Choose the import method:
3. File Upload: Upload a model file from your computer
4. Repository: Import from a model repository using ID/path
5. URL: Provide a direct URL to the model file
6. Configure model details and import

// Client-side code for model import  
async function importModelFromRepository(repository, modelId, options) {  
 try {  
 const formData = new FormData();  
 formData.append('repository', repository);  
 formData.append('modelId', modelId);  
 formData.append('name', options.name);  
 formData.append('description', options.description);  
 formData.append('framework', options.framework);  
  
 const response = await fetch('/api/admin/models/import/repository', {  
 method: 'POST',  
 body: formData,  
 });  
  
 if (!response.ok) {  
 throw new Error('Failed to import model');  
 }  
  
 return await response.json();  
 } catch (error) {  
 console.error('Error importing model:', error);  
 throw error;  
 }  
}

// Client-side code for model import  
async function importModelFromRepository(repository, modelId, options) {  
 try {  
 const formData = new FormData();  
 formData.append('repository', repository);  
 formData.append('modelId', modelId);  
 formData.append('name', options.name);  
 formData.append('description', options.description);  
 formData.append('framework', options.framework);  
  
 const response = await fetch('/api/admin/models/import/repository', {  
 method: 'POST',  
 body: formData,  
 });  
  
 if (!response.ok) {  
 throw new Error('Failed to import model');  
 }  
  
 return await response.json();  
 } catch (error) {  
 console.error('Error importing model:', error);  
 throw error;  
 }  
}

#### Model Import Options

When importing a model, you can configure:

* Model Name: Custom name for reference
* Description: Optional description for documentation
* Framework: Specification of the model's framework if not auto-detected
* Advanced Options: Framework-specific configurations

### Connecting Datasets with Models

The system provides a training connector to link datasets with models for training or fine-tuning.

#### Dataset Splitting

Before training, you can split your dataset into training, validation, and test sets:

1. Split Ratio Configuration: Specify the percentage for each set
2. Stratified Splitting: Maintain class distribution across sets
3. API-driven Splitting: Use the dataset splitting API endpoint

// Client-side code for splitting a dataset  
async function splitDataset(datasetId, splitConfig) {  
 try {  
 const response = await fetch(`/api/admin/datasets/${datasetId}/split`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify({  
 trainRatio: splitConfig.trainRatio, // e.g., 70  
 validationRatio: splitConfig.validationRatio, // e.g., 20  
 testRatio: splitConfig.testRatio, // e.g., 10  
 stratified: splitConfig.stratified // Maintain class distribution if true  
 }),  
 });  
  
 if (!response.ok) {  
 throw new Error(`Failed to split dataset: ${await response.text()}`);  
 }  
  
 return await response.json();  
 } catch (error) {  
 console.error('Error splitting dataset:', error);  
 throw error;  
 }  
}

// Client-side code for splitting a dataset  
async function splitDataset(datasetId, splitConfig) {  
 try {  
 const response = await fetch(`/api/admin/datasets/${datasetId}/split`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify({  
 trainRatio: splitConfig.trainRatio, // e.g., 70  
 validationRatio: splitConfig.validationRatio, // e.g., 20  
 testRatio: splitConfig.testRatio, // e.g., 10  
 stratified: splitConfig.stratified // Maintain class distribution if true  
 }),  
 });  
  
 if (!response.ok) {  
 throw new Error(`Failed to split dataset: ${await response.text()}`);  
 }  
  
 return await response.json();  
 } catch (error) {  
 console.error('Error splitting dataset:', error);  
 throw error;  
 }  
}

#### Training Configuration

The training connector supports:

1. Dataset Selection: Choose from available datasets
2. Model Selection: Select a model for training/fine-tuning
3. Training Parameters:
4. Learning rate
5. Epochs
6. Batch size
7. Validation split
8. Advanced Options:
9. Transfer learning toggle
10. Freezing base model layers
11. Trainable layers count
12. Data augmentation
13. Early stopping
14. Hyperparameter tuning

#### Starting a Training Job

To train a model with a dataset:

1. Navigate to Admin Panel → Datasets → Dataset Details → Training Configuration
2. Configure model selection, split ratios, and data augmentation options
3. Click "Start Training" to begin the process
4. Monitor progress in real-time through the integrated visualization

The system now supports two methods for starting a training job:

##### Method 1: From Models Panel

// Client-side code for starting a training job from Models panel  
async function startModelTraining(config) {  
 try {  
 const response = await fetch('/api/admin/training/start', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify({  
 modelId: config.modelId,  
 datasetId: config.datasetId,  
 learningRate: config.learningRate,  
 epochs: config.epochs,  
 batchSize: config.batchSize,  
 useTranferLearning: config.useTranferLearning,  
 freezeBaseModel: config.freezeBaseModel,  
 trainableLayersCount: config.trainableLayersCount,  
 enableEarlyStopping: config.enableEarlyStopping,  
 enableDataAugmentation: config.enableDataAugmentation,  
 validationSplit: config.validationSplit,  
 enableHyperparameterTuning: config.enableHyperparameterTuning  
 }),  
 });  
  
 if (!response.ok) {  
 throw new Error('Failed to start training');  
 }  
  
 return await response.json();  
 } catch (error) {  
 console.error('Error starting training:', error);  
 throw error;  
 }  
}

// Client-side code for starting a training job from Models panel  
async function startModelTraining(config) {  
 try {  
 const response = await fetch('/api/admin/training/start', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify({  
 modelId: config.modelId,  
 datasetId: config.datasetId,  
 learningRate: config.learningRate,  
 epochs: config.epochs,  
 batchSize: config.batchSize,  
 useTranferLearning: config.useTranferLearning,  
 freezeBaseModel: config.freezeBaseModel,  
 trainableLayersCount: config.trainableLayersCount,  
 enableEarlyStopping: config.enableEarlyStopping,  
 enableDataAugmentation: config.enableDataAugmentation,  
 validationSplit: config.validationSplit,  
 enableHyperparameterTuning: config.enableHyperparameterTuning  
 }),  
 });  
  
 if (!response.ok) {  
 throw new Error('Failed to start training');  
 }  
  
 return await response.json();  
 } catch (error) {  
 console.error('Error starting training:', error);  
 throw error;  
 }  
}

##### Method 2: From Dataset Details Panel

// Client-side code for starting a training job directly from a dataset  
async function startDatasetTraining(datasetId, trainingConfig) {  
 try {  
 const response = await fetch(`/api/admin/datasets/${datasetId}/train`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify({  
 config: {  
 modelArchitecture: trainingConfig.modelArchitecture,  
 pretrainedWeights: trainingConfig.pretrainedWeights,  
 splitRatios: trainingConfig.splitRatios,  
 stratifiedSplit: trainingConfig.stratifiedSplit,  
 hyperparameters: trainingConfig.hyperparameters,  
 augmentation: trainingConfig.augmentation  
 }  
 }),  
 });  
  
 if (!response.ok) {  
 throw new Error(`Server responded with ${response.status}: ${await response.text()}`);  
 }  
  
 const data = await response.json();  
 return data;  
 } catch (error) {  
 console.error('Error starting dataset training:', error);  
 throw error;  
 }  
}

// Client-side code for starting a training job directly from a dataset  
async function startDatasetTraining(datasetId, trainingConfig) {  
 try {  
 const response = await fetch(`/api/admin/datasets/${datasetId}/train`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify({  
 config: {  
 modelArchitecture: trainingConfig.modelArchitecture,  
 pretrainedWeights: trainingConfig.pretrainedWeights,  
 splitRatios: trainingConfig.splitRatios,  
 stratifiedSplit: trainingConfig.stratifiedSplit,  
 hyperparameters: trainingConfig.hyperparameters,  
 augmentation: trainingConfig.augmentation  
 }  
 }),  
 });  
  
 if (!response.ok) {  
 throw new Error(`Server responded with ${response.status}: ${await response.text()}`);  
 }  
  
 const data = await response.json();  
 return data;  
 } catch (error) {  
 console.error('Error starting dataset training:', error);  
 throw error;  
 }  
}

#### Monitoring Training Progress

The system provides real-time training metrics visualization through WebSocket connections:

// Client-side code for monitoring training  
function subscribeToTrainingProgress(datasetId, callbacks) {  
 // Create WebSocket connection to the training progress server  
 const WS\_URL = process.env.REACT\_APP\_WS\_URL || window.location.origin.replace(/^http/, 'ws');  
 const socket = new WebSocket(`${WS\_URL}/ws/training-progress`);  
  
 socket.onopen = () => {  
 // Subscribe to updates for the specific dataset  
 socket.send(JSON.stringify({  
 type: 'subscribe',  
 datasetId: datasetId  
 }));  
  
 if (callbacks.onConnected) {  
 callbacks.onConnected();  
 }  
 };  
  
 socket.onmessage = (event) => {  
 const data = JSON.parse(event.data);  
  
 switch (data.type) {  
 case 'progress':  
 callbacks.onProgress(data.progress, data.currentEpoch, data.totalEpochs);  
 break;  
 case 'metrics':  
 callbacks.onMetrics(data.metrics);  
 break;  
 case 'completed':  
 callbacks.onCompleted(data.results);  
 break;  
 case 'error':  
 callbacks.onError(data.error);  
 break;  
 }  
 };  
  
 socket.onerror = (error) => {  
 console.error('WebSocket error:', error);  
 if (callbacks.onError) {  
 callbacks.onError('WebSocket connection error');  
 }  
 };  
  
 socket.onclose = () => {  
 if (callbacks.onDisconnected) {  
 callbacks.onDisconnected();  
 }  
 };  
  
 // Return a cleanup function  
 return () => {  
 if (socket.readyState === WebSocket.OPEN) {  
 socket.close();  
 }  
 };  
}

// Client-side code for monitoring training  
function subscribeToTrainingProgress(datasetId, callbacks) {  
 // Create WebSocket connection to the training progress server  
 const WS\_URL = process.env.REACT\_APP\_WS\_URL || window.location.origin.replace(/^http/, 'ws');  
 const socket = new WebSocket(`${WS\_URL}/ws/training-progress`);  
  
 socket.onopen = () => {  
 // Subscribe to updates for the specific dataset  
 socket.send(JSON.stringify({  
 type: 'subscribe',  
 datasetId: datasetId  
 }));  
  
 if (callbacks.onConnected) {  
 callbacks.onConnected();  
 }  
 };  
  
 socket.onmessage = (event) => {  
 const data = JSON.parse(event.data);  
  
 switch (data.type) {  
 case 'progress':  
 callbacks.onProgress(data.progress, data.currentEpoch, data.totalEpochs);  
 break;  
 case 'metrics':  
 callbacks.onMetrics(data.metrics);  
 break;  
 case 'completed':  
 callbacks.onCompleted(data.results);  
 break;  
 case 'error':  
 callbacks.onError(data.error);  
 break;  
 }  
 };  
  
 socket.onerror = (error) => {  
 console.error('WebSocket error:', error);  
 if (callbacks.onError) {  
 callbacks.onError('WebSocket connection error');  
 }  
 };  
  
 socket.onclose = () => {  
 if (callbacks.onDisconnected) {  
 callbacks.onDisconnected();  
 }  
 };  
  
 // Return a cleanup function  
 return () => {  
 if (socket.readyState === WebSocket.OPEN) {  
 socket.close();  
 }  
 };  
}

#### Stopping a Training Job

The system allows stopping a training job that is in progress:

// Client-side code for stopping a training job  
async function stopTrainingJob(jobId) {  
 try {  
 const API\_URL = process.env.REACT\_APP\_API\_URL || window.location.origin;  
  
 const response = await fetch(`${API\_URL}/api/admin/training/${jobId}/stop`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 // Add authorization headers if required  
 ...(localStorage.getItem('auth\_token')  
 ? { 'Authorization': `Bearer ${localStorage.getItem('auth\_token')}` }  
 : {})  
 }  
 });  
  
 if (!response.ok) {  
 throw new Error(`Server responded with ${response.status}: ${await response.text()}`);  
 }  
  
 return await response.json();  
 } catch (error) {  
 console.error('Failed to stop training job:', error);  
 throw error;  
 }  
}

// Client-side code for stopping a training job  
async function stopTrainingJob(jobId) {  
 try {  
 const API\_URL = process.env.REACT\_APP\_API\_URL || window.location.origin;  
  
 const response = await fetch(`${API\_URL}/api/admin/training/${jobId}/stop`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 // Add authorization headers if required  
 ...(localStorage.getItem('auth\_token')  
 ? { 'Authorization': `Bearer ${localStorage.getItem('auth\_token')}` }  
 : {})  
 }  
 });  
  
 if (!response.ok) {  
 throw new Error(`Server responded with ${response.status}: ${await response.text()}`);  
 }  
  
 return await response.json();  
 } catch (error) {  
 console.error('Failed to stop training job:', error);  
 throw error;  
 }  
}

### Transfer Learning

The system supports transfer learning to leverage pre-trained models for new tasks with limited data.

#### Transfer Learning Workflow

1. Base Model Selection: Choose a pre-trained model as the starting point
2. Layer Configuration: Determine which layers to freeze/unfreeze
3. Fine-tuning: Train the model on a new dataset (often smaller than original training data)
4. Evaluation: Assess performance on validation data

#### Transfer Learning Options

The system provides several configuration options:

* Layer Freezing: Control which layers remain fixed during training
* Learning Rate Scheduling: Adjust learning rates for different layers
* Progressive Unfreezing: Gradually unfreeze layers during training
* Feature Extraction: Use the model only for feature extraction without fine-tuning

# Python code example for transfer learning configuration  
def configure\_transfer\_learning(base\_model, num\_classes, freeze\_layers=True, trainable\_layers=3):  
 """Configure a model for transfer learning  
  
 Args:  
 base\_model: The pre-trained model to use as a base  
 num\_classes: Number of classes in the new dataset  
 freeze\_layers: Whether to freeze base model layers  
 trainable\_layers: Number of top layers to make trainable if freezing  
  
 Returns:  
 Configured model ready for training  
 """  
 # Create a model with the pre-trained base and new classification head  
 model = create\_transfer\_model(base\_model, num\_classes)  
  
 if freeze\_layers:  
 # Freeze all base model layers  
 for layer in base\_model.layers:  
 layer.trainable = False  
  
 # Make the last few layers trainable if specified  
 if trainable\_layers > 0:  
 for layer in base\_model.layers[-trainable\_layers:]:  
 layer.trainable = True  
  
 # Compile with appropriate parameters for fine-tuning  
 model.compile(  
 optimizer=tf.keras.optimizers.Adam(learning\_rate=0.0001),  
 loss='categorical\_crossentropy',  
 metrics=['accuracy']  
 )  
  
 return model

# Python code example for transfer learning configuration  
def configure\_transfer\_learning(base\_model, num\_classes, freeze\_layers=True, trainable\_layers=3):  
 """Configure a model for transfer learning  
  
 Args:  
 base\_model: The pre-trained model to use as a base  
 num\_classes: Number of classes in the new dataset  
 freeze\_layers: Whether to freeze base model layers  
 trainable\_layers: Number of top layers to make trainable if freezing  
  
 Returns:  
 Configured model ready for training  
 """  
 # Create a model with the pre-trained base and new classification head  
 model = create\_transfer\_model(base\_model, num\_classes)  
  
 if freeze\_layers:  
 # Freeze all base model layers  
 for layer in base\_model.layers:  
 layer.trainable = False  
  
 # Make the last few layers trainable if specified  
 if trainable\_layers > 0:  
 for layer in base\_model.layers[-trainable\_layers:]:  
 layer.trainable = True  
  
 # Compile with appropriate parameters for fine-tuning  
 model.compile(  
 optimizer=tf.keras.optimizers.Adam(learning\_rate=0.0001),  
 loss='categorical\_crossentropy',  
 metrics=['accuracy']  
 )  
  
 return model

### Examples and Use Cases

#### Example 1: Material Dataset with Vector Embeddings

// Upload a ZIP archive of material images and generate embeddings  
async function createMaterialDatasetWithEmbeddings() {  
 // Upload ZIP file with vector embedding generation  
 const zipFile = document.getElementById('dataset-file').files[0];  
  
 const uploadResult = await uploadZipDatasetWithEmbeddings(zipFile, {  
 name: 'Material Surfaces Dataset',  
 description: 'Dataset of various material surfaces with vector embeddings',  
 });  
  
 console.log(`Created dataset with ${uploadResult.imageCount} images`);  
 console.log(`Generated ${uploadResult.embeddingsGenerated} vector embeddings`);  
  
 // Now the dataset is ready for both knowledge base and training use  
  
 // Optional: Start training a model using this dataset  
 const trainingJob = await startModelTraining({  
 datasetId: uploadResult.dataset.id,  
 modelId: 'pretrained-material-classifier',  
 // Other training parameters...  
 });  
  
 return {  
 datasetId: uploadResult.dataset.id,  
 trainingJobId: trainingJob.id  
 };  
}

// Upload a ZIP archive of material images and generate embeddings  
async function createMaterialDatasetWithEmbeddings() {  
 // Upload ZIP file with vector embedding generation  
 const zipFile = document.getElementById('dataset-file').files[0];  
  
 const uploadResult = await uploadZipDatasetWithEmbeddings(zipFile, {  
 name: 'Material Surfaces Dataset',  
 description: 'Dataset of various material surfaces with vector embeddings',  
 });  
  
 console.log(`Created dataset with ${uploadResult.imageCount} images`);  
 console.log(`Generated ${uploadResult.embeddingsGenerated} vector embeddings`);  
  
 // Now the dataset is ready for both knowledge base and training use  
  
 // Optional: Start training a model using this dataset  
 const trainingJob = await startModelTraining({  
 datasetId: uploadResult.dataset.id,  
 modelId: 'pretrained-material-classifier',  
 // Other training parameters...  
 });  
  
 return {  
 datasetId: uploadResult.dataset.id,  
 trainingJobId: trainingJob.id  
 };  
}

#### Example 2: Ceramic Tile Classification

// Import ImageNet pretrained model and fine-tune on ceramic tiles dataset  
async function setupCeramicTileClassifier() {  
 // Import pre-trained MobileNetV2 from TensorFlow Hub  
 const model = await importModelFromRepository(  
 'tfhub',  
 'mobilenetv2\_1.00\_224',  
 {  
 name: 'Ceramic Tile Classifier Base',  
 description: 'Base model for ceramic tile classification',  
 framework: 'tensorflow'  
 }  
 );  
  
 // Import Ceramic Tile dataset from Kaggle  
 const dataset = await importPremadeDataset(  
 'materials-dataset',  
 {  
 name: 'Ceramic Tiles Dataset',  
 includeMetadata: true,  
 selectedClasses: ['ceramic-tile', 'porcelain-tile', 'natural-stone-tile']  
 }  
 );  
  
 // Start training with transfer learning  
 const trainingJob = await startModelTraining({  
 modelId: model.id,  
 datasetId: dataset.id,  
 learningRate: 0.0001,  
 epochs: 10,  
 batchSize: 32,  
 useTranferLearning: true,  
 freezeBaseModel: true,  
 trainableLayersCount: 5,  
 enableEarlyStopping: true,  
 enableDataAugmentation: true,  
 validationSplit: 0.2  
 });  
  
 return trainingJob;  
}

// Import ImageNet pretrained model and fine-tune on ceramic tiles dataset  
async function setupCeramicTileClassifier() {  
 // Import pre-trained MobileNetV2 from TensorFlow Hub  
 const model = await importModelFromRepository(  
 'tfhub',  
 'mobilenetv2\_1.00\_224',  
 {  
 name: 'Ceramic Tile Classifier Base',  
 description: 'Base model for ceramic tile classification',  
 framework: 'tensorflow'  
 }  
 );  
  
 // Import Ceramic Tile dataset from Kaggle  
 const dataset = await importPremadeDataset(  
 'materials-dataset',  
 {  
 name: 'Ceramic Tiles Dataset',  
 includeMetadata: true,  
 selectedClasses: ['ceramic-tile', 'porcelain-tile', 'natural-stone-tile']  
 }  
 );  
  
 // Start training with transfer learning  
 const trainingJob = await startModelTraining({  
 modelId: model.id,  
 datasetId: dataset.id,  
 learningRate: 0.0001,  
 epochs: 10,  
 batchSize: 32,  
 useTranferLearning: true,  
 freezeBaseModel: true,  
 trainableLayersCount: 5,  
 enableEarlyStopping: true,  
 enableDataAugmentation: true,  
 validationSplit: 0.2  
 });  
  
 return trainingJob;  
}

#### Example 3: Multi-Model Ensemble

// Create an ensemble model using multiple pretrained models and datasets  
async function createMaterialEnsemble() {  
 // Import models from different repositories  
 const textureModel = await importModelFromRepository('pytorch\_hub', 'resnet18');  
 const colorModel = await importModelFromRepository('huggingface', 'microsoft/resnet-50');  
 const patternModel = await importModelFromRepository('tfhub', 'efficientnet/b0');  
  
 // Import and prepare the combined dataset  
 const materialDataset = await importPremadeDataset('dtd', {  
 name: 'Material Textures Combined',  
 includeMetadata: true  
 });  
  
 // Train each model on the dataset with different focuses  
 const textureTraining = await startModelTraining({  
 modelId: textureModel.id,  
 datasetId: materialDataset.id,  
 useTranferLearning: true,  
 // Additional parameters for texture focus  
 });  
  
 const colorTraining = await startModelTraining({  
 modelId: colorModel.id,  
 datasetId: materialDataset.id,  
 useTranferLearning: true,  
 // Additional parameters for color focus  
 });  
  
 const patternTraining = await startModelTraining({  
 modelId: patternModel.id,  
 datasetId: materialDataset.id,  
 useTranferLearning: true,  
 // Additional parameters for pattern focus  
 });  
  
 // Create ensemble configuration after all training jobs complete  
 // ...  
}

// Create an ensemble model using multiple pretrained models and datasets  
async function createMaterialEnsemble() {  
 // Import models from different repositories  
 const textureModel = await importModelFromRepository('pytorch\_hub', 'resnet18');  
 const colorModel = await importModelFromRepository('huggingface', 'microsoft/resnet-50');  
 const patternModel = await importModelFromRepository('tfhub', 'efficientnet/b0');  
  
 // Import and prepare the combined dataset  
 const materialDataset = await importPremadeDataset('dtd', {  
 name: 'Material Textures Combined',  
 includeMetadata: true  
 });  
  
 // Train each model on the dataset with different focuses  
 const textureTraining = await startModelTraining({  
 modelId: textureModel.id,  
 datasetId: materialDataset.id,  
 useTranferLearning: true,  
 // Additional parameters for texture focus  
 });  
  
 const colorTraining = await startModelTraining({  
 modelId: colorModel.id,  
 datasetId: materialDataset.id,  
 useTranferLearning: true,  
 // Additional parameters for color focus  
 });  
  
 const patternTraining = await startModelTraining({  
 modelId: patternModel.id,  
 datasetId: materialDataset.id,  
 useTranferLearning: true,  
 // Additional parameters for pattern focus  
 });  
  
 // Create ensemble configuration after all training jobs complete  
 // ...  
}

### Best Practices

#### Dataset Selection

1. Quality over Quantity: Choose datasets with high-quality images rather than larger datasets with poor quality
2. Class Balance: Ensure classes are balanced or use techniques to address imbalance
3. Diversity: Select datasets with diverse examples covering the variance in your target domain
4. Metadata Richness: Prefer datasets with comprehensive metadata when available
5. Vector Embedding Generation: Enable vector embedding generation for ZIP uploads to leverage similarity search and improve recognition capabilities
6. Embedding Quality: For optimal embedding results, use clear, well-lit images with good contrast and minimal background noise

#### Model Selection

1. Task Alignment: Choose models pre-trained on tasks similar to your target application
2. Resource Consideration: Balance model complexity with available computation resources
3. Framework Compatibility: Select models from frameworks your team is familiar with
4. Inference Speed: Consider deployment requirements when selecting models

#### Training Configuration

1. Start Conservative: Begin with conservative learning rates and increase if needed
2. Validation Strategy: Use appropriate validation split or cross-validation
3. Augmentation Tuning: Adjust augmentation to match domain characteristics
4. Early Experimentation: Run short training cycles to validate approach before full training

### Troubleshooting

#### Common Dataset Issues

1. Import Failures:
2. Verify network connection to the repository
3. Check if the dataset ID is correct
4. Ensure sufficient storage space
5. Quality Problems:
6. Use the dataset quality analysis tools to identify issues
7. Apply preprocessing using the dataset management service

Ensure sufficient storage space

Quality Problems:

#### Common Model Issues

1. Import Failures:
2. Verify framework compatibility
3. Check model file integrity
4. Ensure correct model ID or URL
5. Training Issues:
6. Verify dataset format compatibility
7. Check for class imbalance issues
8. Monitor for overfitting or underfitting

Ensure correct model ID or URL

Training Issues:

### API Reference

For detailed API documentation, refer to the API Reference document.

### Related Documentation

* Quality of Data and Metrics
* Material Recognition
* Admin Panel
* Database and Vector DB

# Dependency Management System

Source: readme/dependency-management-system.md

---

## Dependency Management System

The Dependency Management System is a comprehensive solution for monitoring, analyzing, and updating dependencies across both Node.js and Python codebases. It integrates with Kubernetes infrastructure to provide efficient, on-demand scanning with intelligent compatibility analysis and automated updates.

### Architecture Overview

The system consists of the following main components:

1. Admin Panel Interface
2. Dependency Management Page
3. Deployment Dashboard Integration
4. Visual Package Analysis
5. Backend API Services
6. Kubernetes Integration
7. Scan Management
8. Package Analysis
9. Kubernetes Jobs
10. Resource-efficient scanning
11. Selective testing
12. Automated PR creation
13. AI Compatibility Analysis
14. Breaking change detection
15. Configuration impact assessment
16. Test selection

Visual Package Analysis

Backend API Services

Package Analysis

Kubernetes Jobs

Automated PR creation

AI Compatibility Analysis

### Features

#### Admin Panel Integration

* Dedicated Management Page
* Comprehensive view of all dependencies
* Filtering by package type, update type, and risk level
* One-click scan triggering and status monitoring
* Deployment Dashboard Panel
* Real-time dependency status
* Pending PRs and recent updates
* Quick scan triggering

One-click scan triggering and status monitoring

Deployment Dashboard Panel

#### Intelligent Analysis

* AI-Powered Compatibility Checking
* Risk categorization (safe, caution, major)
* Breaking change detection
* Configuration impact analysis
* Selective Testing
* Only tests affected components
* Reduces CI resource usage
* Improves update confidence

Configuration impact analysis

Selective Testing

#### Kubernetes Integration

* Resource-Efficient Scanning
* Dedicated pods only when needed
* Proper resource limits
* Automatic cleanup
* Schedule and On-Demand Options
* Weekly scheduled scans via CronJob
* On-demand scans from admin panel
* Command-line triggering option for automation

Automatic cleanup

Schedule and On-Demand Options

#### Cross-Technology Support

* Node.js and Python Support
* Consistent handling across technologies
* Uniform admin interface
* Technology-specific analysis

### Admin Panel Usage

#### Dependency Management Page

1. Accessing the Page
2. Navigate to the admin panel
3. Click "Dependency Management" in the sidebar
4. Triggering a Scan
5. Click the "Trigger Scan" button
6. Select scan options (all, Node.js, Python)
7. View real-time scan status
8. Viewing Results
9. Review outdated packages list
10. See compatibility analysis for each
11. Filter by various criteria
12. Managing Updates
13. Select packages to update
14. Review potential impact
15. Apply updates individually or in batches

Click "Dependency Management" in the sidebar

Triggering a Scan

View real-time scan status

Viewing Results

Filter by various criteria

Managing Updates

#### Deployment Dashboard Integration

The dependency status panel in the deployment dashboard provides:

* Current scan status
* Pending update PRs
* Recent updates
* Quick scan trigger

### Technical Implementation

#### Backend Services

1. Kubernetes Integration
2. kubernetes.service.ts - Core K8s API client
3. job-monitor.service.ts - Job management
4. API Controller
5. dependencies.controller.ts - Request handling
6. Scan triggering, status monitoring, logs
7. API Routes
8. RESTful endpoints
9. Authentication and access control
10. Swagger documentation

kubernetes.service.ts

job-monitor.service.ts - Job management

job-monitor.service.ts

API Controller

dependencies.controller.ts

Scan triggering, status monitoring, logs

API Routes

#### Admin Panel Components

1. Management Page
2. dependency-management.tsx - Main page component
3. Package listing and filtering
4. Update management
5. Dashboard Integration
6. DependencyUpdatesPanel.tsx - Dashboard component
7. Status summary and quick actions
8. Service Layer
9. dependencyService.ts - API client
10. Type-safe interface to backend

dependency-management.tsx

Update management

Dashboard Integration

DependencyUpdatesPanel.tsx

Status summary and quick actions

Service Layer

dependencyService.ts

#### Kubernetes Resources

1. Job Definition
2. dependency-management-job.yaml
3. CronJob and Job templates
4. Resource configuration
5. Container Image
6. Dockerfile.dependency-scanner
7. Multi-stage build
8. Caching optimization

dependency-management-job.yaml

Resource configuration

Container Image

Dockerfile.dependency-scanner

#### CI/CD Integration

The system integrates with the CI/CD pipeline:

1. Docker Image Building
2. Automatic builds via CI/CD workflows
3. Tags based on Git commit/version
4. Registry pushing
5. Kubernetes Deployment
6. Application via GitOps/Flux
7. Automatic configuration updates
8. Environment-specific settings

Registry pushing

Kubernetes Deployment

### Configuration Options

#### Environment Variables

Table content:

Variable | Description | Default

KUBERNETES\_NAMESPACE | Namespace for jobs | default

DEPENDENCY\_SCANNER\_IMAGE | Scanner image | dependency-scanner:latest

GITHUB\_ORG | GitHub organization | -

REPO\_NAME | Repository name | -

SCAN\_INTERVAL | Scan frequency | 0 0 \* \* 0 (weekly)

KUBERNETES\_NAMESPACE

default

DEPENDENCY\_SCANNER\_IMAGE

dependency-scanner:latest

GITHUB\_ORG

REPO\_NAME

SCAN\_INTERVAL

0 0 \* \* 0

#### Resource Configuration

Job resources can be configured in the dependency-management-job.yaml file:

dependency-management-job.yaml

resources:  
 requests:  
 cpu: 500m  
 memory: 512Mi  
 limits:  
 cpu: 1000m  
 memory: 1Gi

resources:  
 requests:  
 cpu: 500m  
 memory: 512Mi  
 limits:  
 cpu: 1000m  
 memory: 1Gi

### Security Considerations

1. Authentication
2. Admin panel access control
3. API authentication
4. Kubernetes RBAC
5. Credentials Management
6. GitHub token in Kubernetes secret
7. OpenAI API key in Kubernetes secret
8. No hardcoded credentials
9. Job Isolation
10. Dedicated pods for scanning
11. Proper security context
12. Network policy enforcement

Kubernetes RBAC

Credentials Management

No hardcoded credentials

Job Isolation

### Development Guidelines

#### Adding New Features

1. Update the admin panel components
2. Extend the API controller and routes
3. Modify Kubernetes job templates if needed
4. Update this documentation

#### Testing

1. Frontend components with Jest
2. API endpoints with integration tests
3. End-to-end testing with Cypress

### Troubleshooting

#### Common Issues

1. Scan Not Starting
2. Check Kubernetes permissions
3. Verify image accessibility
4. Check for resource constraints
5. Analysis Inaccuracies
6. Review OpenAI API key validity
7. Check for rate limiting
8. Verify compatibility algorithm parameters
9. GitHub Integration Issues
10. Validate GitHub token permissions
11. Check repository access
12. Verify network connectivity

Check for resource constraints

Analysis Inaccuracies

Verify compatibility algorithm parameters

GitHub Integration Issues

### Future Enhancements

1. Enhanced AI Analysis
2. More detailed breaking change detection
3. Code modification suggestions
4. Vulnerability assessment integration
5. Additional Technology Support
6. Java/Maven dependencies
7. Go modules
8. Rust crates
9. Performance Improvements
10. Faster scanning techniques
11. Dependency graph caching
12. Parallelized analysis

Vulnerability assessment integration

Additional Technology Support

Rust crates

Performance Improvements

### Conclusion

The Dependency Management System provides a comprehensive solution for keeping dependencies up-to-date across the entire stack. By integrating with the admin panel, Kubernetes infrastructure, and AI-powered analysis, it enables efficient and safe dependency updates with minimal manual intervention.

# Deployment Guide Updates

Source: readme/deployment-guide-updates.md

---

## Recently Implemented Features and Deployment Updates

### New Features and Their Deployment Configurations

The KAI platform has recently been enhanced with several new features that require specific deployment configurations. This section provides details on these features and their deployment requirements.

#### Property-Based Recommendation Engine

The Property-Based Recommendation Engine is integrated with the API server and provides intelligent material recommendations based on property requirements, user preferences, and project context.

##### Deployment Configuration

The recommendation engine is deployed as part of the API server with the following configuration:

* Deployment Method: Integrated with the API server deployment
* Configuration: ConfigMap mounted to the API server
* Resource Requirements: Included in the API server resource allocation
* Scaling: Scales with the API server

##### Implementation Details

The recommendation engine is implemented as a ConfigMap that is mounted to the API server:

# ConfigMap for the recommendation engine  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: recommendation-engine-config  
 labels:  
 app: api-server  
 component: recommendation  
data:  
 recommendation\_engine\_enabled: "true"  
 recommendation\_cache\_size: "1000"  
 recommendation\_refresh\_interval: "300" # 5 minutes  
 recommendation\_max\_concurrent\_jobs: "5"  
 recommendation\_similarity\_threshold: "0.7"  
 recommendation\_config.json: |  
 {  
 "weightFactors": {  
 "propertyMatch": 0.6,  
 "userPreference": 0.3,  
 "projectContext": 0.1  
 },  
 "cacheStrategy": "lru",  
 "defaultLimit": 20,  
 "enableFeedbackLoop": true,  
 "minConfidenceScore": 0.65  
 }

# ConfigMap for the recommendation engine  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: recommendation-engine-config  
 labels:  
 app: api-server  
 component: recommendation  
data:  
 recommendation\_engine\_enabled: "true"  
 recommendation\_cache\_size: "1000"  
 recommendation\_refresh\_interval: "300" # 5 minutes  
 recommendation\_max\_concurrent\_jobs: "5"  
 recommendation\_similarity\_threshold: "0.7"  
 recommendation\_config.json: |  
 {  
 "weightFactors": {  
 "propertyMatch": 0.6,  
 "userPreference": 0.3,  
 "projectContext": 0.1  
 },  
 "cacheStrategy": "lru",  
 "defaultLimit": 20,  
 "enableFeedbackLoop": true,  
 "minConfidenceScore": 0.65  
 }

The API server deployment is patched to include the recommendation engine configuration:

# Patch for the API server deployment  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: api-server  
spec:  
 template:  
 spec:  
 containers:  
 - name: api-server  
 env:  
 - name: RECOMMENDATION\_ENGINE\_ENABLED  
 valueFrom:  
 configMapKeyRef:  
 name: recommendation-engine-config  
 key: recommendation\_engine\_enabled  
 # Additional environment variables...  
 volumeMounts:  
 - name: recommendation-config  
 mountPath: /app/config/recommendation  
 readOnly: true  
 volumes:  
 - name: recommendation-config  
 configMap:  
 name: recommendation-engine-config

# Patch for the API server deployment  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: api-server  
spec:  
 template:  
 spec:  
 containers:  
 - name: api-server  
 env:  
 - name: RECOMMENDATION\_ENGINE\_ENABLED  
 valueFrom:  
 configMapKeyRef:  
 name: recommendation-engine-config  
 key: recommendation\_engine\_enabled  
 # Additional environment variables...  
 volumeMounts:  
 - name: recommendation-config  
 mountPath: /app/config/recommendation  
 readOnly: true  
 volumes:  
 - name: recommendation-config  
 configMap:  
 name: recommendation-engine-config

#### Material Property Analytics

The Material Property Analytics feature provides comprehensive analytics for material properties, including distribution analysis, trend analysis, correlation analysis, and anomaly detection.

##### Deployment Configuration

Material Property Analytics is deployed as a separate service with the following configuration:

* Deployment Method: Dedicated deployment
* Resource Requirements:
* CPU: 1000m (request) / 4000m (limit)
* Memory: 2Gi (request) / 8Gi (limit)
* Scaling: HorizontalPodAutoscaler with 2-6 replicas
* Storage: 20Gi PersistentVolumeClaim for analytics data
* Workflow Integration: Integrated with the coordinator service via workflow templates

##### Implementation Details

The Material Property Analytics service is deployed as a separate Kubernetes deployment:

# Deployment for Material Property Analytics  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: material-property-analytics  
 labels:  
 app: material-property-analytics  
 component: analytics  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: material-property-analytics  
 template:  
 metadata:  
 labels:  
 app: material-property-analytics  
 component: analytics  
 annotations:  
 prometheus.io/scrape: "true"  
 prometheus.io/port: "9100"  
 prometheus.io/path: "/metrics"  
 spec:  
 containers:  
 - name: material-property-analytics  
 image: ${REGISTRY\_URL}/kai/material-property-analytics:latest  
 # Container configuration...

# Deployment for Material Property Analytics  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: material-property-analytics  
 labels:  
 app: material-property-analytics  
 component: analytics  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: material-property-analytics  
 template:  
 metadata:  
 labels:  
 app: material-property-analytics  
 component: analytics  
 annotations:  
 prometheus.io/scrape: "true"  
 prometheus.io/port: "9100"  
 prometheus.io/path: "/metrics"  
 spec:  
 containers:  
 - name: material-property-analytics  
 image: ${REGISTRY\_URL}/kai/material-property-analytics:latest  
 # Container configuration...

The service is integrated with the coordinator service via a workflow template:

# Workflow template for Material Property Analytics  
apiVersion: argoproj.io/v1alpha1  
kind: WorkflowTemplate  
metadata:  
 name: material-property-analytics-template  
spec:  
 entrypoint: material-property-analytics  
 # Workflow template configuration...

# Workflow template for Material Property Analytics  
apiVersion: argoproj.io/v1alpha1  
kind: WorkflowTemplate  
metadata:  
 name: material-property-analytics-template  
spec:  
 entrypoint: material-property-analytics  
 # Workflow template configuration...

#### Relationship-Aware Model Training

The Relationship-Aware Model Training feature enhances AI model training by incorporating knowledge from the Property Relationship Graph to improve property prediction and search relevance.

##### Deployment Configuration

Relationship-Aware Model Training is deployed as a separate service with the following configuration:

* Deployment Method: Dedicated deployment
* Resource Requirements:
* CPU: 4000m (request) / 8000m (limit)
* Memory: 16Gi (request) / 32Gi (limit)
* GPU: 1 NVIDIA GPU (L40S preferred)
* Scaling: HorizontalPodAutoscaler with 1-3 replicas
* Storage: 40Gi PersistentVolumeClaim for model storage
* Node Selection: Runs on nodes with gpu-type=nvidia-l40s label
* Workflow Integration: Integrated with the coordinator service via workflow templates

gpu-type=nvidia-l40s

##### Implementation Details

The Relationship-Aware Model Training service is deployed as a separate Kubernetes deployment:

# Deployment for Relationship-Aware Model Training  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: relationship-aware-training  
 labels:  
 app: relationship-aware-training  
 component: ml-services  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: relationship-aware-training  
 template:  
 metadata:  
 labels:  
 app: relationship-aware-training  
 component: ml-services  
 gpu-enabled: "true"  
 annotations:  
 prometheus.io/scrape: "true"  
 prometheus.io/port: "9100"  
 prometheus.io/path: "/metrics"  
 spec:  
 containers:  
 - name: relationship-aware-training  
 image: ${REGISTRY\_URL}/kai/relationship-aware-training:latest  
 # Container configuration...  
 nodeSelector:  
 gpu-type: nvidia-l40s  
 runtimeClassName: nvidia-gpu

# Deployment for Relationship-Aware Model Training  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: relationship-aware-training  
 labels:  
 app: relationship-aware-training  
 component: ml-services  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: relationship-aware-training  
 template:  
 metadata:  
 labels:  
 app: relationship-aware-training  
 component: ml-services  
 gpu-enabled: "true"  
 annotations:  
 prometheus.io/scrape: "true"  
 prometheus.io/port: "9100"  
 prometheus.io/path: "/metrics"  
 spec:  
 containers:  
 - name: relationship-aware-training  
 image: ${REGISTRY\_URL}/kai/relationship-aware-training:latest  
 # Container configuration...  
 nodeSelector:  
 gpu-type: nvidia-l40s  
 runtimeClassName: nvidia-gpu

The service is integrated with the coordinator service via a workflow template:

# Workflow template for Relationship-Aware Model Training  
apiVersion: argoproj.io/v1alpha1  
kind: WorkflowTemplate  
metadata:  
 name: relationship-aware-training-template  
spec:  
 entrypoint: relationship-aware-training  
 # Workflow template configuration...

# Workflow template for Relationship-Aware Model Training  
apiVersion: argoproj.io/v1alpha1  
kind: WorkflowTemplate  
metadata:  
 name: relationship-aware-training-template  
spec:  
 entrypoint: relationship-aware-training  
 # Workflow template configuration...

#### Coordinator Service Integration

The coordinator service has been updated to be aware of the new services and features:

# ML Features Configuration in coordinator-config ConfigMap  
ml\_features: |  
 {  
 "multimodalPatternRecognition": {  
 "enabled": true,  
 "workflowTemplate": "multimodal-pattern-recognition-template",  
 "minQualityTier": "medium",  
 "resourceQuotas": {  
 "cpu": 8,  
 "memory": "16Gi",  
 "gpu": 1  
 },  
 "maxConcurrent": 4  
 },  
 "domainSpecificNetworks": {  
 "enabled": true,  
 "workflowTemplate": "domain-specific-networks-template",  
 "minQualityTier": "medium",  
 "resourceQuotas": {  
 "cpu": 8,  
 "memory": "16Gi",  
 "gpu": 1  
 },  
 "maxConcurrent": 4,  
 "supportedDomains": ["wood", "metal", "fabric", "stone", "ceramic", "composite"]  
 },  
 "relationshipAwareTraining": {  
 "enabled": true,  
 "workflowTemplate": "relationship-aware-training-template",  
 "minQualityTier": "high",  
 "resourceQuotas": {  
 "cpu": 8,  
 "memory": "32Gi",  
 "gpu": 1  
 },  
 "maxConcurrent": 2,  
 "supportedMaterialTypes": ["tile", "wood", "stone", "laminate", "vinyl", "carpet", "metal", "glass", "concrete", "ceramic", "porcelain"]  
 },  
 "materialPropertyAnalytics": {  
 "enabled": true,  
 "workflowTemplate": "material-property-analytics-template",  
 "minQualityTier": "medium",  
 "resourceQuotas": {  
 "cpu": 4,  
 "memory": "8Gi",  
 "gpu": 0  
 },  
 "maxConcurrent": 5  
 }  
 }

# ML Features Configuration in coordinator-config ConfigMap  
ml\_features: |  
 {  
 "multimodalPatternRecognition": {  
 "enabled": true,  
 "workflowTemplate": "multimodal-pattern-recognition-template",  
 "minQualityTier": "medium",  
 "resourceQuotas": {  
 "cpu": 8,  
 "memory": "16Gi",  
 "gpu": 1  
 },  
 "maxConcurrent": 4  
 },  
 "domainSpecificNetworks": {  
 "enabled": true,  
 "workflowTemplate": "domain-specific-networks-template",  
 "minQualityTier": "medium",  
 "resourceQuotas": {  
 "cpu": 8,  
 "memory": "16Gi",  
 "gpu": 1  
 },  
 "maxConcurrent": 4,  
 "supportedDomains": ["wood", "metal", "fabric", "stone", "ceramic", "composite"]  
 },  
 "relationshipAwareTraining": {  
 "enabled": true,  
 "workflowTemplate": "relationship-aware-training-template",  
 "minQualityTier": "high",  
 "resourceQuotas": {  
 "cpu": 8,  
 "memory": "32Gi",  
 "gpu": 1  
 },  
 "maxConcurrent": 2,  
 "supportedMaterialTypes": ["tile", "wood", "stone", "laminate", "vinyl", "carpet", "metal", "glass", "concrete", "ceramic", "porcelain"]  
 },  
 "materialPropertyAnalytics": {  
 "enabled": true,  
 "workflowTemplate": "material-property-analytics-template",  
 "minQualityTier": "medium",  
 "resourceQuotas": {  
 "cpu": 4,  
 "memory": "8Gi",  
 "gpu": 0  
 },  
 "maxConcurrent": 5  
 }  
 }

#### GPU Resource Management

The GPU resource management configuration has been updated to include the new Relationship-Aware Training service:

# GPU Resource Management in coordinator-config ConfigMap  
gpu\_resources: |  
 {  
 "classes": {  
 "nvidia-l40s": {  
 "priority": 10,  
 "models": ["multimodal-pattern-recognition", "domain-specific-networks", "relationship-aware-training"]  
 },  
 "nvidia-h100": {  
 "priority": 20,  
 "models": ["gaussian-splatting", "triposr", "wonder3d", "instant3d"]  
 }  
 },  
 "scaling": {  
 "enabled": true,  
 "minNodes": 1,  
 "maxNodes": 10,  
 "scaleDownDelay": "10m"  
 }  
 }

# GPU Resource Management in coordinator-config ConfigMap  
gpu\_resources: |  
 {  
 "classes": {  
 "nvidia-l40s": {  
 "priority": 10,  
 "models": ["multimodal-pattern-recognition", "domain-specific-networks", "relationship-aware-training"]  
 },  
 "nvidia-h100": {  
 "priority": 20,  
 "models": ["gaussian-splatting", "triposr", "wonder3d", "instant3d"]  
 }  
 },  
 "scaling": {  
 "enabled": true,  
 "minNodes": 1,  
 "maxNodes": 10,  
 "scaleDownDelay": "10m"  
 }  
 }

#### GitOps Integration

All the new features are integrated with the GitOps workflow using Flux CD. The deployment configurations are stored in the GitOps repository and automatically applied by Flux.

### Deployment Updates

#### Updated Kustomization Structure

The Flux GitOps repository has been updated with new directories for the recently implemented features:

flux/  
├── clusters/  
│ ├── production/  
│ │ ├── kai/  
│ │ │ ├── analytics/ # New directory for Material Property Analytics  
│ │ │ ├── api-server/ # Updated with recommendation engine config  
│ │ │ ├── coordinator/ # Updated with new workflow templates  
│ │ │ ├── ml-services/ # Updated with relationship-aware training  
│ │ │ └── kustomization.yaml # Updated to include new components

flux/  
├── clusters/  
│ ├── production/  
│ │ ├── kai/  
│ │ │ ├── analytics/ # New directory for Material Property Analytics  
│ │ │ ├── api-server/ # Updated with recommendation engine config  
│ │ │ ├── coordinator/ # Updated with new workflow templates  
│ │ │ ├── ml-services/ # Updated with relationship-aware training  
│ │ │ └── kustomization.yaml # Updated to include new components

#### Deployment Process

The deployment process for the new features follows the existing GitOps workflow:

1. Changes are committed to the GitOps repository
2. Flux automatically detects the changes and applies them to the cluster
3. The new services are deployed and integrated with the existing services

#### Resource Requirements

The new features have specific resource requirements that should be considered when planning cluster capacity:

Table content:

Feature | CPU | Memory | GPU | Storage

Property-Based Recommendation Engine | Included in API server | Included in API server | N/A | N/A

Material Property Analytics | 1-4 cores | 2-8 GB | N/A | 20 GB

Relationship-Aware Model Training | 4-8 cores | 16-32 GB | 1 NVIDIA L40S | 40 GB

#### Monitoring and Logging

All new services are configured with Prometheus metrics and structured logging:

* Prometheus metrics exposed on port 9100
* Metrics path: /metrics
* Log format: JSON
* Log level: Configurable via environment variables

/metrics

#### Health Checks

All new services include appropriate health checks:

* Liveness probe: /health
* Readiness probe: /ready
* Startup probe (where applicable): /startup

/health

/ready

/startup

# Deployment Guide

Source: readme/deployment-guide.md

---

## KAI Platform Automated Deployment Guide

This comprehensive guide covers the fully automated deployment process for the KAI platform, including infrastructure provisioning, Kubernetes cluster setup, CI/CD pipeline configuration, and monitoring.

### Table of Contents

* Overview
* Architecture
* Prerequisites
* Automated Deployment with GitHub Actions
* Required GitHub Secrets
* Complete GitHub Actions Workflow
* Workflow Explanation
* Digital Ocean Kubernetes Cluster Configuration
* Cluster Requirements
* Node Pool Configuration
* Resource Allocation
* SSL Certificate Management
* Frontend Deployment to Vercel
* Environment Variables and Secrets
* Deployment Verification and Monitoring
* Maintenance and Updates
* Updating the Application
* Scaling the Application
* Backup and Disaster Recovery
* Troubleshooting
* Deployment Issues
* Kubernetes Issues
* SSL Certificate Issues
* Docker Issues
* GitHub Container Registry Integration
* Performance Optimization

### Overview

The Kai Platform is a comprehensive system for material management, recognition, and 3D visualization. This guide provides detailed instructions for deploying the platform to production environments and setting up development environments.

### Architecture

The Kai application consists of several components deployed across different platforms:

┌────────────────────────────────────┐  
 │ │  
 │ Digital Ocean Kubernetes Cluster │  
 │ ─────────────────────────────────┤  
 │ │  
┌────────────────────────┐ │ ┌────────────────┐ ┌───────────┐│  
│ │ │ │ │ │ ││  
│ Vercel │ │ │ API Server │ │ Redis ││  
│ ─────────────────── │ │ │ │ │ ││  
│ - Admin Panel (Next) │───┼─▶│ │ │ ││  
│ - Client App (Gatsby) │ │ └────────┬───────┘ └───────────┘│  
│ │ │ │ │  
└────────────────────────┘ │ ▼ │  
 │ │ ┌────────────────┐ │  
 │ │ │ │ │  
 │ │ │ Coordinator │◄──┐ │  
 │ │ │ Service │ │ │  
 ▼ │ │ │ │ │  
┌────────────────────────┐ │ └────────┬───────┘ │ │  
│ │ │ │ │ │  
│ Supabase │ │ ▼ │ │  
│ ─────────────────── │ │ ┌────────────────┐ │ │  
│ - Authentication │◀──┼─▶│ │ │ │  
│ - Realtime Features │ │ │ Argo │───┘ │  
│ - Queue Management │ │ │ Workflows │ │  
│ - Vector Database │ │ │ │ │  
│ │ │ └────────────────┘ │  
└────────────┬───────────┘ │ │ │  
 │ │ ▼ │  
 │ │ ┌────────────────┐ │  
 │ │ │ Worker Pods │ │  
 │ │ │ ─────────────┤ │  
 └──────────────┼─▶│ - Quality │ │  
 │ │ Assessment │ │  
┌────────────────────────┐ │ │ - Preprocessing │ │  
│ │ │ │ - COLMAP SfM │ │  
│ External Services │ │ │ - Point Cloud │ │  
│ ─────────────────── │ │ │ - Model Gen │ │  
│ - MongoDB Atlas │◀─┼─▶│ - NeRF │ │  
│ - AWS S3 │ │ │ - Format Conv │ │  
│ │ │ │ │ │  
└────────────────────────┘ │ └────────────────┘ │  
 │ │  
 └────────────────────────────────────┘

┌────────────────────────────────────┐  
 │ │  
 │ Digital Ocean Kubernetes Cluster │  
 │ ─────────────────────────────────┤  
 │ │  
┌────────────────────────┐ │ ┌────────────────┐ ┌───────────┐│  
│ │ │ │ │ │ ││  
│ Vercel │ │ │ API Server │ │ Redis ││  
│ ─────────────────── │ │ │ │ │ ││  
│ - Admin Panel (Next) │───┼─▶│ │ │ ││  
│ - Client App (Gatsby) │ │ └────────┬───────┘ └───────────┘│  
│ │ │ │ │  
└────────────────────────┘ │ ▼ │  
 │ │ ┌────────────────┐ │  
 │ │ │ │ │  
 │ │ │ Coordinator │◄──┐ │  
 │ │ │ Service │ │ │  
 ▼ │ │ │ │ │  
┌────────────────────────┐ │ └────────┬───────┘ │ │  
│ │ │ │ │ │  
│ Supabase │ │ ▼ │ │  
│ ─────────────────── │ │ ┌────────────────┐ │ │  
│ - Authentication │◀──┼─▶│ │ │ │  
│ - Realtime Features │ │ │ Argo │───┘ │  
│ - Queue Management │ │ │ Workflows │ │  
│ - Vector Database │ │ │ │ │  
│ │ │ └────────────────┘ │  
└────────────┬───────────┘ │ │ │  
 │ │ ▼ │  
 │ │ ┌────────────────┐ │  
 │ │ │ Worker Pods │ │  
 │ │ │ ─────────────┤ │  
 └──────────────┼─▶│ - Quality │ │  
 │ │ Assessment │ │  
┌────────────────────────┐ │ │ - Preprocessing │ │  
│ │ │ │ - COLMAP SfM │ │  
│ External Services │ │ │ - Point Cloud │ │  
│ ─────────────────── │ │ │ - Model Gen │ │  
│ - MongoDB Atlas │◀─┼─▶│ - NeRF │ │  
│ - AWS S3 │ │ │ - Format Conv │ │  
│ │ │ │ │ │  
└────────────────────────┘ │ └────────────────┘ │  
 │ │  
 └────────────────────────────────────┘

With the recently added features, the architecture has been extended:

┌────────────────────────────────────────────────────────────────────────────────┐  
│ │  
│ Digital Ocean Kubernetes Cluster │  
│ ───────────────────────────────────────────────────────────────────────── │  
│ │  
│ ┌──────────────────┐ │  
│ │ │ │  
│ │ Parameter │◄────────────┐ │  
│ │ Registry │ │ │  
│ │ │ │ │  
│ └─────────┬────────┘ │ │  
│ │ │ │  
│ ▼ │ │  
│ ┌──────────────┐ ┌──────────────────┐ ┌─┴────────────┐ ┌───────────────┐│  
│ │ │ │ │ │ │ │ ││  
│ │ Notification │ │ Coordinator │ │ ML Workloads │ │ Webhook ││  
│ │ Service │◄─┤ Service │◄──┤ Orchestrator │◄──┤ Service ││  
│ │ │ │ │ │ │ │ ││  
│ └──────┬───────┘ └──────────────────┘ └─────┬────────┘ └───────────────┘│  
│ │ │ │ │  
│ │ ▼ ▼ │  
│ │ ┌────────────────┐ ┌─────────────────────────────────┐ │  
│ │ │ │ │ │ │  
│ │ │ Existing │ │ ML Services │ │  
│ │ │ Workflows │ │ ───────────────────────────────│ │  
│ │ │ │ │ - MultiModal Pattern Recognition│ │  
│ │ └────────────────┘ │ - Domain-Specific Networks │ │  
│ │ │ - Relationship-Aware Training │ │  
│ │ │ - Material Property Analytics │ │  
│ ▼ │ - Material Promotion System │ │  
│ ┌─────────────────┐ │ │ │  
│ │ Email/SMS/Push │ └─────────────────────────────────┘ │  
│ │ Delivery │ ┌─────────────────────────────────────────────┐ │  
│ │ Services │ │ │ │  
│ └─────────────────┘ │ API Server │ │  
│ │ ─────────────────────────────────────── │ │  
│ │ - Property-Based Recommendation Engine │ │  
│ │ - Material Comparison Engine │ │  
│ │ - Property Inheritance System │ │  
│ │ - Subscription Management with Stripe │ │  
│ │ - Factory Material Promotion API │ │  
│ │ │ │  
│ └─────────────────────────────────────────────┘ │  
│ │  
└──────────────────────────────────────────────────────────────────────────────┘

┌────────────────────────────────────────────────────────────────────────────────┐  
│ │  
│ Digital Ocean Kubernetes Cluster │  
│ ───────────────────────────────────────────────────────────────────────── │  
│ │  
│ ┌──────────────────┐ │  
│ │ │ │  
│ │ Parameter │◄────────────┐ │  
│ │ Registry │ │ │  
│ │ │ │ │  
│ └─────────┬────────┘ │ │  
│ │ │ │  
│ ▼ │ │  
│ ┌──────────────┐ ┌──────────────────┐ ┌─┴────────────┐ ┌───────────────┐│  
│ │ │ │ │ │ │ │ ││  
│ │ Notification │ │ Coordinator │ │ ML Workloads │ │ Webhook ││  
│ │ Service │◄─┤ Service │◄──┤ Orchestrator │◄──┤ Service ││  
│ │ │ │ │ │ │ │ ││  
│ └──────┬───────┘ └──────────────────┘ └─────┬────────┘ └───────────────┘│  
│ │ │ │ │  
│ │ ▼ ▼ │  
│ │ ┌────────────────┐ ┌─────────────────────────────────┐ │  
│ │ │ │ │ │ │  
│ │ │ Existing │ │ ML Services │ │  
│ │ │ Workflows │ │ ───────────────────────────────│ │  
│ │ │ │ │ - MultiModal Pattern Recognition│ │  
│ │ └────────────────┘ │ - Domain-Specific Networks │ │  
│ │ │ - Relationship-Aware Training │ │  
│ │ │ - Material Property Analytics │ │  
│ ▼ │ - Material Promotion System │ │  
│ ┌─────────────────┐ │ │ │  
│ │ Email/SMS/Push │ └─────────────────────────────────┘ │  
│ │ Delivery │ ┌─────────────────────────────────────────────┐ │  
│ │ Services │ │ │ │  
│ └─────────────────┘ │ API Server │ │  
│ │ ─────────────────────────────────────── │ │  
│ │ - Property-Based Recommendation Engine │ │  
│ │ - Material Comparison Engine │ │  
│ │ - Property Inheritance System │ │  
│ │ - Subscription Management with Stripe │ │  
│ │ - Factory Material Promotion API │ │  
│ │ │ │  
│ └─────────────────────────────────────────────┘ │  
│ │  
└──────────────────────────────────────────────────────────────────────────────┘

### Prerequisites

Before proceeding with deployment, ensure you have the following:

* GitHub account with administrator access to the repository
* Supabase account and project set up
* Vercel account with projects created for frontend and admin panel
* Digital Ocean account with API access
* Domain name(s) for your deployment
* OpenAI API key (for AI features)
* MongoDB Atlas account (or other MongoDB provider)
* Stripe account (if using payment features)

You do not need to install any local tools as the entire deployment process is automated through GitHub Actions.

### Automated Deployment with GitHub Actions

The KAI platform uses GitHub Actions to fully automate the deployment process, from building and testing code to provisioning infrastructure and deploying to Kubernetes and Vercel. Our approach uses modular, reusable workflows for better maintainability and flexibility.

#### Workflow Architecture

The deployment system consists of several reusable workflows:

1. Main Workflow (deploy.yml):
2. Orchestrates the entire deployment process
3. Determines which environment to deploy to (staging or production)
4. Calls the appropriate reusable workflows in sequence
5. Build and Test (build-test.yml):
6. Builds the application code
7. Runs tests to ensure code quality
8. Uploads build artifacts for later use
9. Docker Build (docker-build.yml):
10. Builds Docker images for all services
11. Pushes images to the Docker registry
12. Tags images with both the commit SHA and environment
13. Infrastructure Provisioning (provision-infrastructure.yml):
14. Checks if a Kubernetes cluster exists
15. Creates a new cluster if needed
16. Sets up node pools optimized for different workloads
17. Kubernetes Setup (setup-kubernetes.yml):
18. Creates necessary namespaces
19. Sets up Kubernetes secrets
20. Installs cert-manager for SSL certificates
21. Installs NGINX Ingress Controller
22. Installs Argo Workflows for ML pipelines
23. Application Deployment (deploy-application.yml):
24. Deploys the application using Helm
25. Runs database migrations
26. Verifies the deployment
27. Frontend Deployment (deploy-frontend.yml):
28. Deploys the client frontend to Vercel
29. Deploys the admin panel to Vercel
30. Sets up environment variables
31. Deployment Verification (verify-deployment.yml):
32. Performs comprehensive health checks
33. Verifies API availability
34. Checks SSL certificate validity
35. Sends deployment notifications

deploy.yml

Calls the appropriate reusable workflows in sequence

Build and Test (build-test.yml):

build-test.yml

Uploads build artifacts for later use

Docker Build (docker-build.yml):

docker-build.yml

Tags images with both the commit SHA and environment

Infrastructure Provisioning (provision-infrastructure.yml):

provision-infrastructure.yml

Sets up node pools optimized for different workloads

Kubernetes Setup (setup-kubernetes.yml):

setup-kubernetes.yml

Installs Argo Workflows for ML pipelines

Application Deployment (deploy-application.yml):

deploy-application.yml

Verifies the deployment

Frontend Deployment (deploy-frontend.yml):

deploy-frontend.yml

Sets up environment variables

Deployment Verification (verify-deployment.yml):

verify-deployment.yml

#### Required GitHub Secrets

Before running the deployment workflow, you need to add the following secrets to your GitHub repository:

##### Digital Ocean / Infrastructure Secrets

* DIGITALOCEAN\_ACCESS\_TOKEN: Your Digital Ocean API token with write access
* CLUSTER\_NAME: Base name for your Kubernetes cluster (e.g., "kai")
* DO\_REGION: Region for your cluster (e.g., "ams3")

DIGITALOCEAN\_ACCESS\_TOKEN

CLUSTER\_NAME

DO\_REGION

##### GitHub Container Registry Secrets

* GITHUB\_TOKEN: GitHub token with write:packages permission (automatically provided by GitHub Actions)
* GITHUB\_REPOSITORY: (Optional) GitHub repository name if different from the current repository

GITHUB\_TOKEN

write:packages

GITHUB\_REPOSITORY

##### Domain and SSL Secrets

* DOMAIN\_NAME: Your domain name (e.g., "kai-platform.com")
* ADMIN\_EMAIL: Email for SSL certificate notifications
* BASE\_URL: Base URL for the application (e.g., "https://api.kai-platform.com")

DOMAIN\_NAME

ADMIN\_EMAIL

BASE\_URL

##### Database Secrets

* MONGODB\_URI: MongoDB connection string
* DATABASE\_URL: Database connection string (if using a different database)
* DATABASE\_HOST: Database host (if not using connection string)
* DATABASE\_PORT: Database port
* DATABASE\_USER: Database username
* DATABASE\_PASSWORD: Database password
* DATABASE\_NAME: Database name
* DATABASE\_SSL: Whether to use SSL for database connection (true/false)
* DATABASE\_MAX\_CONNECTIONS: Maximum number of database connections
* DATABASE\_CONNECTION\_TIMEOUT: Database connection timeout in milliseconds

MONGODB\_URI

DATABASE\_URL

DATABASE\_HOST

DATABASE\_PORT

DATABASE\_USER

DATABASE\_PASSWORD

DATABASE\_NAME

DATABASE\_SSL

DATABASE\_MAX\_CONNECTIONS

DATABASE\_CONNECTION\_TIMEOUT

##### Authentication Secrets

* JWT\_SECRET: Secret for JWT tokens
* JWT\_EXPIRES\_IN: JWT token expiration time (e.g., "1d")
* RATE\_LIMIT\_WINDOW: Rate limiting window in milliseconds
* RATE\_LIMIT\_MAX: Maximum number of requests per window
* CORS\_ORIGINS: Comma-separated list of allowed origins for CORS
* MAX\_UPLOAD\_SIZE: Maximum upload size in bytes

JWT\_SECRET

JWT\_EXPIRES\_IN

RATE\_LIMIT\_WINDOW

RATE\_LIMIT\_MAX

CORS\_ORIGINS

MAX\_UPLOAD\_SIZE

##### AI/ML Model Secrets

* OPENAI\_API\_KEY: Your OpenAI API key
* OPENAI\_DEFAULT\_MODEL: Default model to use (e.g., "gpt-4")
* OPENAI\_TEMPERATURE: Temperature setting (e.g., "0.7")
* ANTHROPIC\_API\_KEY: Your Anthropic API key for Claude models
* HF\_API\_KEY or HUGGINGFACE\_API\_KEY: Your Hugging Face API key
* HF\_ORGANIZATION\_ID: Your Hugging Face organization ID (optional)
* HF\_DEFAULT\_TEXT\_MODEL: Default text model (e.g., "google/flan-t5-xxl")
* HF\_DEFAULT\_EMBEDDING\_MODEL: Default embedding model (e.g., "sentence-transformers/all-MiniLM-L6-v2")
* HF\_DEFAULT\_IMAGE\_MODEL: Default image model (e.g., "google/vit-base-patch16-224")
* HF\_MODEL\_TIMEOUT: Timeout for model requests in milliseconds
* HF\_USE\_FAST\_MODELS: Whether to use faster models (true/false)
* OCR\_MODEL\_PATH: Path to OCR model
* ML\_MAX\_PROCESSING\_TIME: Maximum processing time for ML tasks in milliseconds
* MODEL\_CACHE\_PATH: Path to model cache directory

OPENAI\_API\_KEY

OPENAI\_DEFAULT\_MODEL

OPENAI\_TEMPERATURE

ANTHROPIC\_API\_KEY

HF\_API\_KEY

HUGGINGFACE\_API\_KEY

HF\_ORGANIZATION\_ID

HF\_DEFAULT\_TEXT\_MODEL

HF\_DEFAULT\_EMBEDDING\_MODEL

HF\_DEFAULT\_IMAGE\_MODEL

HF\_MODEL\_TIMEOUT

HF\_USE\_FAST\_MODELS

OCR\_MODEL\_PATH

ML\_MAX\_PROCESSING\_TIME

MODEL\_CACHE\_PATH

##### S3 Storage Secrets

* S3\_ENDPOINT: S3 endpoint URL
* S3\_ACCESS\_KEY: S3 access key
* S3\_SECRET\_KEY: S3 secret key
* S3\_BUCKET: S3 bucket name
* S3\_REGION: S3 region (e.g., "us-east-1")
* S3\_PUBLIC\_URL: Public URL for S3 bucket (optional)
* AWS\_REGION: AWS region (if using AWS S3)
* AWS\_ACCESS\_KEY\_ID: AWS access key ID (if using AWS S3)
* AWS\_SECRET\_ACCESS\_KEY: AWS secret access key (if using AWS S3)
* TEMP: Temporary directory for file processing

S3\_ENDPOINT

S3\_ACCESS\_KEY

S3\_SECRET\_KEY

S3\_BUCKET

S3\_REGION

S3\_PUBLIC\_URL

AWS\_REGION

AWS\_ACCESS\_KEY\_ID

AWS\_SECRET\_ACCESS\_KEY

TEMP

##### Redis Secrets

* REDIS\_URL: Redis connection URL (or use host/port/password)
* REDIS\_HOST: Redis host (if not using connection URL)
* REDIS\_PORT: Redis port
* REDIS\_PASSWORD: Redis password
* REDIS\_SSL: Whether to use SSL for Redis connection (true/false)
* REDIS\_DB: Redis database number

REDIS\_URL

REDIS\_HOST

REDIS\_PORT

REDIS\_PASSWORD

REDIS\_SSL

REDIS\_DB

##### Supabase Secrets - Staging

* SUPABASE\_URL\_STAGING: Your Supabase staging project URL
* SUPABASE\_KEY\_STAGING: Your Supabase staging service key
* SUPABASE\_ANON\_KEY\_STAGING: Your Supabase staging anonymous key
* SUPABASE\_SERVICE\_ROLE\_KEY\_STAGING: Supabase service role key for staging

SUPABASE\_URL\_STAGING

SUPABASE\_KEY\_STAGING

SUPABASE\_ANON\_KEY\_STAGING

SUPABASE\_SERVICE\_ROLE\_KEY\_STAGING

##### Supabase Secrets - Production

* SUPABASE\_URL\_PRODUCTION: Your Supabase production project URL
* SUPABASE\_KEY\_PRODUCTION: Your Supabase production service key
* SUPABASE\_ANON\_KEY\_PRODUCTION: Your Supabase production anonymous key
* SUPABASE\_SERVICE\_ROLE\_KEY\_PRODUCTION: Supabase service role key for production
* SUPABASE\_STORAGE\_BUCKET: Supabase storage bucket name (e.g., "materials")

SUPABASE\_URL\_PRODUCTION

SUPABASE\_KEY\_PRODUCTION

SUPABASE\_ANON\_KEY\_PRODUCTION

SUPABASE\_SERVICE\_ROLE\_KEY\_PRODUCTION

SUPABASE\_STORAGE\_BUCKET

##### Stripe Payment Secrets

* STRIPE\_SECRET\_KEY: Stripe secret key
* STRIPE\_PUBLISHABLE\_KEY: Stripe publishable key
* STRIPE\_WEBHOOK\_SECRET: Stripe webhook secret
* STRIPE\_API\_VERSION: Stripe API version (e.g., "2023-10-16")
* STRIPE\_TEST\_MODE: Whether to use Stripe in test mode (true/false)

STRIPE\_SECRET\_KEY

STRIPE\_PUBLISHABLE\_KEY

STRIPE\_WEBHOOK\_SECRET

STRIPE\_API\_VERSION

STRIPE\_TEST\_MODE

##### Vercel Secrets

* VERCEL\_TOKEN: Your Vercel API token
* VERCEL\_ORG\_ID: Your Vercel organization ID
* VERCEL\_PROJECT\_ID\_CLIENT: Vercel project ID for client frontend
* VERCEL\_PROJECT\_ID\_ADMIN: Vercel project ID for admin panel

VERCEL\_TOKEN

VERCEL\_ORG\_ID

VERCEL\_PROJECT\_ID\_CLIENT

VERCEL\_PROJECT\_ID\_ADMIN

##### Model Selection Configuration

* MODEL\_EVALUATION\_STANDARD\_CYCLE: Number of standard operations before evaluation
* MODEL\_EVALUATION\_TEST\_CYCLE: Number of evaluation operations
* MODEL\_SELECTION\_METRICS\_WEIGHTS: Weights for accuracy, latency, and cost

MODEL\_EVALUATION\_STANDARD\_CYCLE

MODEL\_EVALUATION\_TEST\_CYCLE

MODEL\_SELECTION\_METRICS\_WEIGHTS

##### Service Integration

* KAI\_API\_URL: Main KAI API URL
* KAI\_VECTOR\_DB\_URL: Vector database service URL
* KAI\_ML\_SERVICE\_URL: Machine learning service URL
* ML\_API\_URL: ML API URL for LLM fallback
* KAI\_API\_KEY: API key for KAI service authentication
* API\_URL: API URL (if different from KAI\_API\_URL)
* MCP\_SERVER\_URL: MCP server URL
* USE\_MCP\_SERVER: Whether to use MCP server (true/false)
* MCP\_HEALTH\_CHECK\_TIMEOUT: Timeout for MCP health check in milliseconds
* ENABLE\_MOCK\_FALLBACK: Enable mock services as fallback (true/false)

KAI\_API\_URL

KAI\_VECTOR\_DB\_URL

KAI\_ML\_SERVICE\_URL

ML\_API\_URL

KAI\_API\_KEY

API\_URL

MCP\_SERVER\_URL

USE\_MCP\_SERVER

MCP\_HEALTH\_CHECK\_TIMEOUT

ENABLE\_MOCK\_FALLBACK

##### Agent Configuration

* AGENT\_VERBOSE\_MODE: Enable verbose mode for agents (true/false)
* AGENT\_MEMORY\_ENABLED: Enable agent memory persistence (true/false)
* AGENT\_MAX\_ITERATIONS: Maximum number of iterations for agent tasks
* AGENT\_TIMEOUT: Default timeout for agent operations in milliseconds
* MAX\_CONCURRENT\_SESSIONS: Maximum number of concurrent agent sessions
* AGENT\_API\_KEY: API key for agent authentication
* LOG\_LEVEL: Logging level (error, warn, info, http, debug)
* LOG\_FILE\_PATH: Path to log file
* LOG\_CONSOLE\_OUTPUT: Whether to output logs to console (true/false)

AGENT\_VERBOSE\_MODE

AGENT\_MEMORY\_ENABLED

AGENT\_MAX\_ITERATIONS

AGENT\_TIMEOUT

MAX\_CONCURRENT\_SESSIONS

AGENT\_API\_KEY

LOG\_LEVEL

LOG\_FILE\_PATH

LOG\_CONSOLE\_OUTPUT

##### Frontend Configuration

* GATSBY\_API\_URL: API URL for frontend to connect to backend services
* GATSBY\_WS\_URL: WebSocket URL for real-time communication
* GATSBY\_SUPABASE\_URL: Supabase URL for client
* GATSBY\_SUPABASE\_ANON\_KEY: Supabase anonymous key for client
* GATSBY\_STORAGE\_URL: Storage URL for frontend assets
* GATSBY\_DEFAULT\_LOCALE: Default locale
* GATSBY\_ENABLE\_OFFLINE\_MODE: Enable offline mode (true/false)
* GATSBY\_GOOGLE\_ANALYTICS\_ID: Google Analytics ID
* REACT\_APP\_VERSION: App version
* GATSBY\_APP\_NAME: App name
* GATSBY\_STRIPE\_PUBLISHABLE\_KEY: Stripe publishable key for frontend

GATSBY\_API\_URL

GATSBY\_WS\_URL

GATSBY\_SUPABASE\_URL

GATSBY\_SUPABASE\_ANON\_KEY

GATSBY\_STORAGE\_URL

GATSBY\_DEFAULT\_LOCALE

GATSBY\_ENABLE\_OFFLINE\_MODE

GATSBY\_GOOGLE\_ANALYTICS\_ID

REACT\_APP\_VERSION

GATSBY\_APP\_NAME

GATSBY\_STRIPE\_PUBLISHABLE\_KEY

##### Monitoring Configuration

* HEALTH\_CHECK\_INTERVAL: Health check interval in milliseconds
* METRICS\_ENABLED: Whether to enable metrics (true/false)
* METRICS\_PORT: Port for metrics server

HEALTH\_CHECK\_INTERVAL

METRICS\_ENABLED

METRICS\_PORT

##### Notification Secrets

* SLACK\_WEBHOOK: Slack webhook URL for deployment notifications

SLACK\_WEBHOOK

##### Web Crawler Secrets

* CREDENTIALS\_ENCRYPTION\_KEY: Encryption key for stored credentials
* JINA\_API\_KEY: Jina AI API key for web crawler

CREDENTIALS\_ENCRYPTION\_KEY

JINA\_API\_KEY

#### Complete GitHub Actions Workflow

Create a file at .github/workflows/deploy.yml with the following content:

.github/workflows/deploy.yml

name: KAI Platform CI/CD Pipeline  
  
on:  
 push:  
 branches: [main]  
 pull\_request:  
 branches: [main]  
 workflow\_dispatch:  
 inputs:  
 environment:  
 description: 'Environment to deploy to'  
 required: true  
 default: 'production'  
 type: choice  
 options:  
 - staging  
 - production  
 create\_cluster:  
 description: 'Create new cluster if not exists'  
 type: boolean  
 default: false  
  
env:  
 DOMAIN\_NAME: ${{ secrets.DOMAIN\_NAME }}  
 ENVIRONMENT: ${{ github.event.inputs.environment || 'production' }}  
  
jobs:  
 build-and-test:  
 name: Build and Test  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Set up Node.js  
 uses: actions/setup-node@v3  
 with:  
 node-version: '16'  
 cache: 'yarn'  
  
 - name: Install dependencies  
 run: yarn install --frozen-lockfile  
  
 - name: Build packages  
 run: yarn build  
  
 - name: Run tests  
 run: yarn test  
  
 - name: Upload build artifacts  
 uses: actions/upload-artifact@v3  
 with:  
 name: build-artifacts  
 path: packages/\*/dist  
  
 build-docker:  
 name: Build and Push Docker Images  
 needs: build-and-test  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Download build artifacts  
 uses: actions/download-artifact@v3  
 with:  
 name: build-artifacts  
 path: packages  
  
 - name: Set up Docker Buildx  
 uses: docker/setup-buildx-action@v2  
  
 - name: Login to GitHub Container Registry  
 uses: docker/login-action@v2  
 with:  
 registry: ghcr.io  
 username: ${{ github.repository\_owner }}  
 password: ${{ secrets.GITHUB\_TOKEN }}  
  
 - name: Build and push API server  
 uses: docker/build-push-action@v4  
 with:  
 context: .  
 file: packages/server/Dockerfile  
 push: true  
 tags: |  
 ghcr.io/${{ github.repository }}/kai-api:${{ github.sha }}  
 ghcr.io/${{ github.repository }}/kai-api:latest  
  
 - name: Build and push Coordinator service  
 uses: docker/build-push-action@v4  
 with:  
 context: .  
 file: packages/coordinator/Dockerfile  
 push: true  
 tags: |  
 ghcr.io/${{ github.repository }}/kai-coordinator:${{ github.sha }}  
 ghcr.io/${{ github.repository }}/kai-coordinator:latest  
  
 - name: Build and push ML services  
 uses: docker/build-push-action@v4  
 with:  
 context: .  
 file: packages/ml/Dockerfile  
 push: true  
 tags: |  
 ghcr.io/${{ github.repository }}/kai-ml:${{ github.sha }}  
 ghcr.io/${{ github.repository }}/kai-ml:latest  
  
 - name: Build and push Notification service  
 uses: docker/build-push-action@v4  
 with:  
 context: .  
 file: packages/notification/Dockerfile  
 push: true  
 tags: |  
 ghcr.io/${{ github.repository }}/kai-notification:${{ github.sha }}  
 ghcr.io/${{ github.repository }}/kai-notification:latest  
  
 provision-infrastructure:  
 name: Provision Kubernetes Cluster  
 needs: build-docker  
 runs-on: ubuntu-latest  
 if: github.event.inputs.create\_cluster == 'true' || github.ref == 'refs/heads/main'  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Install doctl  
 uses: digitalocean/action-doctl@v2  
 with:  
 token: ${{ secrets.DIGITALOCEAN\_ACCESS\_TOKEN }}  
  
 - name: Check if cluster exists  
 id: check-cluster  
 run: |  
 CLUSTER\_NAME="${{ secrets.CLUSTER\_NAME }}-${{ env.ENVIRONMENT }}"  
 if doctl kubernetes cluster get $CLUSTER\_NAME &>/dev/null; then  
 echo "Cluster exists, skipping creation"  
 echo "cluster\_exists=true" >> $GITHUB\_OUTPUT  
 else  
 echo "Cluster does not exist"  
 echo "cluster\_exists=false" >> $GITHUB\_OUTPUT  
 fi  
  
 - name: Create Digital Ocean Kubernetes cluster  
 if: steps.check-cluster.outputs.cluster\_exists == 'false' && (github.event.inputs.create\_cluster == 'true' || github.ref == 'refs/heads/main')  
 run: |  
 CLUSTER\_NAME="${{ secrets.CLUSTER\_NAME }}-${{ env.ENVIRONMENT }}"  
  
 echo "Creating main Kubernetes cluster with orchestration node pool..."  
 # Create the main cluster with the orchestration node pool  
 doctl kubernetes cluster create $CLUSTER\_NAME \  
 --region ${{ secrets.DO\_REGION }} \  
 --version latest \  
 --tag kai-platform \  
 --tag ${{ env.ENVIRONMENT }} \  
 --auto-upgrade=true \  
 --maintenance-window="saturday=21:00" \  
 --size s-2vcpu-4gb \  
 --count 3 \  
 --node-pool "name=orchestration;size=s-2vcpu-4gb;count=3;label=node-type=orchestration;tag=orchestration" \  
 --wait  
  
 # Save kubeconfig  
 doctl kubernetes cluster kubeconfig save $CLUSTER\_NAME  
  
 # Wait for cluster to be fully ready  
 echo "Waiting for cluster to be ready..."  
 sleep 60  
  
 echo "Adding CPU-optimized node pool..."  
 # Add CPU-optimized node pool  
 doctl kubernetes cluster node-pool create $CLUSTER\_NAME \  
 --name cpu-optimized \  
 --size c-4 \  
 --count 3 \  
 --label node-type=cpu-optimized \  
 --tag cpu-optimized  
  
 # Add GPU node pool if in production  
 if [ "${{ env.ENVIRONMENT }}" = "production" ]; then  
 echo "Adding GPU-optimized node pool..."  
 doctl kubernetes cluster node-pool create $CLUSTER\_NAME \  
 --name gpu-optimized \  
 --size gd-l40s-4vcpu-24gb \  
 --count 2 \  
 --label node-type=gpu-optimized \  
 --tag gpu-optimized  
 fi  
  
 # Add memory-optimized node pool if in production  
 if [ "${{ env.ENVIRONMENT }}" = "production" ]; then  
 echo "Adding memory-optimized node pool..."  
 doctl kubernetes cluster node-pool create $CLUSTER\_NAME \  
 --name memory-optimized \  
 --size m-4vcpu-32gb \  
 --count 1 \  
 --label node-type=memory-optimized \  
 --tag memory-optimized  
 fi  
  
 - name: Get kubeconfig  
 if: steps.check-cluster.outputs.cluster\_exists == 'true'  
 run: |  
 CLUSTER\_NAME="${{ secrets.CLUSTER\_NAME }}-${{ env.ENVIRONMENT }}"  
 doctl kubernetes cluster kubeconfig save $CLUSTER\_NAME  
  
 - name: Install kubectl  
 uses: azure/setup-kubectl@v3  
  
 - name: Verify cluster  
 run: |  
 kubectl get nodes  
 kubectl get nodes --show-labels  
  
 deploy-kubernetes:  
 name: Deploy to Kubernetes  
 needs: [build-docker, provision-infrastructure]  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Install doctl  
 uses: digitalocean/action-doctl@v2  
 with:  
 token: ${{ secrets.DIGITALOCEAN\_ACCESS\_TOKEN }}  
  
 - name: Get kubeconfig  
 run: |  
 CLUSTER\_NAME="${{ secrets.CLUSTER\_NAME }}-${{ env.ENVIRONMENT }}"  
 doctl kubernetes cluster kubeconfig save $CLUSTER\_NAME  
  
 - name: Set up kubectl  
 uses: azure/setup-kubectl@v3  
  
 - name: Set up Helm  
 uses: azure/setup-helm@v3  
 with:  
 version: 'v3.10.0'  
  
 - name: Create environment files  
 run: |  
 # Create .env file for production  
 cat > .env.production << EOF  
 # OpenAI API  
 OPENAI\_API\_KEY=${{ secrets.OPENAI\_API\_KEY }}  
  
 # Database  
 MONGODB\_URI=${{ secrets.MONGODB\_URI }}  
  
 # Authentication  
 JWT\_SECRET=${{ secrets.JWT\_SECRET }}  
  
 # Supabase  
 SUPABASE\_URL=${{ secrets.SUPABASE\_URL }}  
 SUPABASE\_KEY=${{ secrets.SUPABASE\_KEY }}  
  
 # Frontend URLs  
 GATSBY\_API\_URL=https://api.${{ secrets.DOMAIN\_NAME }}  
 GATSBY\_SUPABASE\_URL=${{ secrets.SUPABASE\_URL }}  
 GATSBY\_SUPABASE\_ANON\_KEY=${{ secrets.SUPABASE\_ANON\_KEY }}  
  
 # Stripe (if using payments)  
 STRIPE\_SECRET\_KEY=${{ secrets.STRIPE\_SECRET\_KEY }}  
 GATSBY\_STRIPE\_PUBLISHABLE\_KEY=${{ secrets.STRIPE\_PUBLISHABLE\_KEY }}  
 EOF  
  
 - name: Create Kubernetes namespace  
 run: |  
 kubectl create namespace kai-system --dry-run=client -o yaml | kubectl apply -f -  
  
 - name: Create Kubernetes secrets  
 run: |  
 # Create main secrets  
 kubectl create secret generic kai-secrets \  
 --namespace kai-system \  
 --from-literal=mongodb-uri='${{ secrets.MONGODB\_URI }}' \  
 --from-literal=jwt-secret='${{ secrets.JWT\_SECRET }}' \  
 --from-literal=openai-api-key='${{ secrets.OPENAI\_API\_KEY }}' \  
 --from-literal=supabase-url='${{ secrets.SUPABASE\_URL }}' \  
 --from-literal=supabase-key='${{ secrets.SUPABASE\_KEY }}' \  
 --from-literal=stripe-secret-key='${{ secrets.STRIPE\_SECRET\_KEY }}' \  
 --dry-run=client -o yaml | kubectl apply -f -  
  
 # Create Redis password secret  
 kubectl create secret generic redis-password \  
 --namespace kai-system \  
 --from-literal=redis-password='${{ secrets.REDIS\_PASSWORD }}' \  
 --dry-run=client -o yaml | kubectl apply -f -  
  
 - name: Check and Install cert-manager  
 run: |  
 # Check if cert-manager is already installed  
 if kubectl get namespace cert-manager &>/dev/null && kubectl get deployment -n cert-manager cert-manager &>/dev/null; then  
 echo "cert-manager is already installed, skipping installation"  
 else  
 echo "Installing cert-manager..."  
 # Add Jetstack Helm repo  
 helm repo add jetstack https://charts.jetstack.io  
  
 # Install cert-manager  
 helm upgrade --install cert-manager jetstack/cert-manager \  
 --namespace cert-manager \  
 --create-namespace \  
 --version v1.11.0 \  
 --set installCRDs=true  
  
 # Wait for cert-manager to be ready  
 kubectl -n cert-manager rollout status deployment/cert-manager  
 kubectl -n cert-manager rollout status deployment/cert-manager-webhook  
 fi  
  
 # Check if ClusterIssuer exists  
 if kubectl get clusterissuer letsencrypt-prod &>/dev/null; then  
 echo "ClusterIssuer already exists, skipping creation"  
 else  
 echo "Creating ClusterIssuer for Let's Encrypt..."  
 # Create ClusterIssuer for Let's Encrypt  
 cat > cluster-issuer.yaml << EOF  
 apiVersion: cert-manager.io/v1  
 kind: ClusterIssuer  
 metadata:  
 name: letsencrypt-prod  
 spec:  
 acme:  
 server: https://acme-v02.api.letsencrypt.org/directory  
 email: ${{ secrets.ADMIN\_EMAIL }}  
 privateKeySecretRef:  
 name: letsencrypt-prod  
 solvers:  
 - http01:  
 ingress:  
 class: nginx  
 EOF  
  
 kubectl apply -f cluster-issuer.yaml  
 fi  
  
 - name: Check and Install NGINX Ingress Controller  
 run: |  
 # Check if NGINX Ingress is already installed  
 if kubectl get deployment -n kai-system nginx-ingress-ingress-nginx-controller &>/dev/null; then  
 echo "NGINX Ingress Controller is already installed, skipping installation"  
 else  
 echo "Installing NGINX Ingress Controller..."  
 # Add NGINX Ingress Helm repo  
 helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx  
  
 # Install NGINX Ingress  
 helm upgrade --install nginx-ingress ingress-nginx/ingress-nginx \  
 --namespace kai-system \  
 --set controller.publishService.enabled=true  
  
 # Wait for NGINX Ingress to be ready  
 kubectl -n kai-system rollout status deployment/nginx-ingress-ingress-nginx-controller  
 fi  
  
 - name: Check and Install Argo Workflows  
 run: |  
 # Check if Argo Workflows is already installed  
 if kubectl get namespace argo &>/dev/null && kubectl get deployment -n argo argo-server &>/dev/null; then  
 echo "Argo Workflows is already installed, skipping installation"  
 else  
 echo "Installing Argo Workflows..."  
 # Create namespace  
 kubectl create namespace argo --dry-run=client -o yaml | kubectl apply -f -  
  
 # Install Argo Workflows  
 kubectl apply -n argo -f https://github.com/argoproj/argo-workflows/releases/download/v3.4.5/install.yaml  
  
 # Wait for Argo to be ready  
 kubectl -n argo rollout status deployment/argo-server  
 fi  
  
 # Check and update Argo configuration  
 echo "Configuring Argo to work with the kai-system namespace..."  
 kubectl patch configmap/workflow-controller-configmap \  
 -n argo \  
 --type merge \  
 -p '{"data":{"workflowNamespaces":"kai-system,argo"}}' \  
 --dry-run=client -o yaml | kubectl apply -f -  
  
 - name: Deploy with Helm  
 run: |  
 # Update Helm repos  
 helm repo update  
  
 # Create values override file  
 cat > values-override.yaml << EOF  
 global:  
 environment: ${{ env.ENVIRONMENT }}  
 registry:  
 url: "ghcr.io"  
 repository: ${{ github.repository }}  
 image:  
 tag: ${{ github.sha }}  
 domain: ${{ secrets.DOMAIN\_NAME }}  
 ingress:  
 enabled: true  
 annotations:  
 kubernetes.io/ingress.class: nginx  
 cert-manager.io/cluster-issuer: letsencrypt-prod  
 hosts:  
 - host: api.${{ secrets.DOMAIN\_NAME }}  
 paths:  
 - path: /  
 pathType: Prefix  
 tls:  
 - secretName: kai-tls-cert  
 hosts:  
 - api.${{ secrets.DOMAIN\_NAME }}  
 EOF  
  
 # Deploy using Helm  
 helm upgrade --install kai ./helm-charts/kai \  
 --namespace kai-system \  
 --values ./helm-charts/kai/values-${{ env.ENVIRONMENT }}.yaml \  
 --values values-override.yaml  
  
 - name: Verify deployment  
 run: |  
 # Wait for deployments to be ready  
 kubectl -n kai-system rollout status deployment/api-server  
 kubectl -n kai-system rollout status deployment/coordinator-service  
  
 # Check if services are running  
 kubectl -n kai-system get services  
  
 # Check if ingress is configured  
 kubectl -n kai-system get ingress  
  
 # Check if certificates are issued  
 kubectl -n kai-system get certificates  
  
 - name: Run database migrations  
 run: |  
 # Create a temporary pod to run migrations  
 cat > migration-job.yaml << EOF  
 apiVersion: batch/v1  
 kind: Job  
 metadata:  
 name: database-migrations  
 namespace: kai-system  
 spec:  
 template:  
 spec:  
 containers:  
 - name: migrations  
 image: ghcr.io/${{ github.repository }}/kai-api:${{ github.sha }}  
 command: ["node", "scripts/run-migrations.js"]  
 env:  
 - name: MONGODB\_URI  
 valueFrom:  
 secretKeyRef:  
 name: kai-secrets  
 key: mongodb-uri  
 restartPolicy: Never  
 backoffLimit: 4  
 EOF  
  
 kubectl apply -f migration-job.yaml  
  
 # Wait for migrations to complete  
 kubectl -n kai-system wait --for=condition=complete job/database-migrations --timeout=120s  
  
 - name: Notify deployment status  
 if: always()  
 uses: rtCamp/action-slack-notify@v2  
 env:  
 SLACK\_WEBHOOK: ${{ secrets.SLACK\_WEBHOOK }}  
 SLACK\_TITLE: "Deployment Status"  
 SLACK\_MESSAGE: "Kubernetes deployment ${{ job.status }}"  
 SLACK\_COLOR: ${{ job.status == 'success' && 'good' || 'danger' }}  
  
 deploy-frontend:  
 name: Deploy Frontend to Vercel  
 needs: deploy-kubernetes  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Set up Node.js  
 uses: actions/setup-node@v3  
 with:  
 node-version: '16'  
 cache: 'yarn'  
  
 - name: Install dependencies  
 run: yarn install --frozen-lockfile  
  
 - name: Deploy to Vercel  
 uses: amondnet/vercel-action@v20  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_PROJECT\_ID\_CLIENT }}  
 working-directory: ./packages/client  
 vercel-args: '--prod'  
  
 - name: Deploy admin panel to Vercel  
 uses: amondnet/vercel-action@v20  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_PROJECT\_ID\_ADMIN }}  
 working-directory: ./packages/admin  
 vercel-args: '--prod'  
  
 verify-deployment:  
 name: Verify Full Deployment  
 needs: [deploy-kubernetes, deploy-frontend]  
 runs-on: ubuntu-latest  
 steps:  
 - name: Comprehensive health check  
 run: |  
 # Wait for DNS propagation (may take some time)  
 echo "Waiting for DNS propagation..."  
 sleep 60  
  
 # Check API health endpoint  
 curl -sSf https://api.${{ secrets.DOMAIN\_NAME }}/health || echo "API not yet available, may need more time for DNS propagation"  
  
 # Check pod status  
 kubectl get pods -n kai-system -o json | jq '.items[] | select(.status.phase != "Running" or ([ .status.containerStatuses[] | select(.ready == false) ] | length > 0)) | .metadata.name' || echo "All pods are running"  
  
 # Check recent logs for errors  
 kubectl logs -n kai-system -l app=api-server --tail=50 | grep -i error || echo "No errors in recent logs"  
  
 # Check SSL certificate validity and expiration  
 CERT\_EXPIRY=$(echo | openssl s\_client -servername api.${{ secrets.DOMAIN\_NAME }} -connect api.${{ secrets.DOMAIN\_NAME }}:443 2>/dev/null | openssl x509 -noout -enddate | cut -d= -f2 || echo "Certificate not yet available")  
 echo "Certificate expires on: $CERT\_EXPIRY"  
  
 - name: Send deployment notification  
 uses: rtCamp/action-slack-notify@v2  
 env:  
 SLACK\_WEBHOOK: ${{ secrets.SLACK\_WEBHOOK }}  
 SLACK\_TITLE: "Deployment Complete"  
 SLACK\_MESSAGE: "✅ KAI Platform has been successfully deployed to ${{ env.ENVIRONMENT }}!"  
 SLACK\_COLOR: "good"

name: KAI Platform CI/CD Pipeline  
  
on:  
 push:  
 branches: [main]  
 pull\_request:  
 branches: [main]  
 workflow\_dispatch:  
 inputs:  
 environment:  
 description: 'Environment to deploy to'  
 required: true  
 default: 'production'  
 type: choice  
 options:  
 - staging  
 - production  
 create\_cluster:  
 description: 'Create new cluster if not exists'  
 type: boolean  
 default: false  
  
env:  
 DOMAIN\_NAME: ${{ secrets.DOMAIN\_NAME }}  
 ENVIRONMENT: ${{ github.event.inputs.environment || 'production' }}  
  
jobs:  
 build-and-test:  
 name: Build and Test  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Set up Node.js  
 uses: actions/setup-node@v3  
 with:  
 node-version: '16'  
 cache: 'yarn'  
  
 - name: Install dependencies  
 run: yarn install --frozen-lockfile  
  
 - name: Build packages  
 run: yarn build  
  
 - name: Run tests  
 run: yarn test  
  
 - name: Upload build artifacts  
 uses: actions/upload-artifact@v3  
 with:  
 name: build-artifacts  
 path: packages/\*/dist  
  
 build-docker:  
 name: Build and Push Docker Images  
 needs: build-and-test  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Download build artifacts  
 uses: actions/download-artifact@v3  
 with:  
 name: build-artifacts  
 path: packages  
  
 - name: Set up Docker Buildx  
 uses: docker/setup-buildx-action@v2  
  
 - name: Login to GitHub Container Registry  
 uses: docker/login-action@v2  
 with:  
 registry: ghcr.io  
 username: ${{ github.repository\_owner }}  
 password: ${{ secrets.GITHUB\_TOKEN }}  
  
 - name: Build and push API server  
 uses: docker/build-push-action@v4  
 with:  
 context: .  
 file: packages/server/Dockerfile  
 push: true  
 tags: |  
 ghcr.io/${{ github.repository }}/kai-api:${{ github.sha }}  
 ghcr.io/${{ github.repository }}/kai-api:latest  
  
 - name: Build and push Coordinator service  
 uses: docker/build-push-action@v4  
 with:  
 context: .  
 file: packages/coordinator/Dockerfile  
 push: true  
 tags: |  
 ghcr.io/${{ github.repository }}/kai-coordinator:${{ github.sha }}  
 ghcr.io/${{ github.repository }}/kai-coordinator:latest  
  
 - name: Build and push ML services  
 uses: docker/build-push-action@v4  
 with:  
 context: .  
 file: packages/ml/Dockerfile  
 push: true  
 tags: |  
 ghcr.io/${{ github.repository }}/kai-ml:${{ github.sha }}  
 ghcr.io/${{ github.repository }}/kai-ml:latest  
  
 - name: Build and push Notification service  
 uses: docker/build-push-action@v4  
 with:  
 context: .  
 file: packages/notification/Dockerfile  
 push: true  
 tags: |  
 ghcr.io/${{ github.repository }}/kai-notification:${{ github.sha }}  
 ghcr.io/${{ github.repository }}/kai-notification:latest  
  
 provision-infrastructure:  
 name: Provision Kubernetes Cluster  
 needs: build-docker  
 runs-on: ubuntu-latest  
 if: github.event.inputs.create\_cluster == 'true' || github.ref == 'refs/heads/main'  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Install doctl  
 uses: digitalocean/action-doctl@v2  
 with:  
 token: ${{ secrets.DIGITALOCEAN\_ACCESS\_TOKEN }}  
  
 - name: Check if cluster exists  
 id: check-cluster  
 run: |  
 CLUSTER\_NAME="${{ secrets.CLUSTER\_NAME }}-${{ env.ENVIRONMENT }}"  
 if doctl kubernetes cluster get $CLUSTER\_NAME &>/dev/null; then  
 echo "Cluster exists, skipping creation"  
 echo "cluster\_exists=true" >> $GITHUB\_OUTPUT  
 else  
 echo "Cluster does not exist"  
 echo "cluster\_exists=false" >> $GITHUB\_OUTPUT  
 fi  
  
 - name: Create Digital Ocean Kubernetes cluster  
 if: steps.check-cluster.outputs.cluster\_exists == 'false' && (github.event.inputs.create\_cluster == 'true' || github.ref == 'refs/heads/main')  
 run: |  
 CLUSTER\_NAME="${{ secrets.CLUSTER\_NAME }}-${{ env.ENVIRONMENT }}"  
  
 echo "Creating main Kubernetes cluster with orchestration node pool..."  
 # Create the main cluster with the orchestration node pool  
 doctl kubernetes cluster create $CLUSTER\_NAME \  
 --region ${{ secrets.DO\_REGION }} \  
 --version latest \  
 --tag kai-platform \  
 --tag ${{ env.ENVIRONMENT }} \  
 --auto-upgrade=true \  
 --maintenance-window="saturday=21:00" \  
 --size s-2vcpu-4gb \  
 --count 3 \  
 --node-pool "name=orchestration;size=s-2vcpu-4gb;count=3;label=node-type=orchestration;tag=orchestration" \  
 --wait  
  
 # Save kubeconfig  
 doctl kubernetes cluster kubeconfig save $CLUSTER\_NAME  
  
 # Wait for cluster to be fully ready  
 echo "Waiting for cluster to be ready..."  
 sleep 60  
  
 echo "Adding CPU-optimized node pool..."  
 # Add CPU-optimized node pool  
 doctl kubernetes cluster node-pool create $CLUSTER\_NAME \  
 --name cpu-optimized \  
 --size c-4 \  
 --count 3 \  
 --label node-type=cpu-optimized \  
 --tag cpu-optimized  
  
 # Add GPU node pool if in production  
 if [ "${{ env.ENVIRONMENT }}" = "production" ]; then  
 echo "Adding GPU-optimized node pool..."  
 doctl kubernetes cluster node-pool create $CLUSTER\_NAME \  
 --name gpu-optimized \  
 --size gd-l40s-4vcpu-24gb \  
 --count 2 \  
 --label node-type=gpu-optimized \  
 --tag gpu-optimized  
 fi  
  
 # Add memory-optimized node pool if in production  
 if [ "${{ env.ENVIRONMENT }}" = "production" ]; then  
 echo "Adding memory-optimized node pool..."  
 doctl kubernetes cluster node-pool create $CLUSTER\_NAME \  
 --name memory-optimized \  
 --size m-4vcpu-32gb \  
 --count 1 \  
 --label node-type=memory-optimized \  
 --tag memory-optimized  
 fi  
  
 - name: Get kubeconfig  
 if: steps.check-cluster.outputs.cluster\_exists == 'true'  
 run: |  
 CLUSTER\_NAME="${{ secrets.CLUSTER\_NAME }}-${{ env.ENVIRONMENT }}"  
 doctl kubernetes cluster kubeconfig save $CLUSTER\_NAME  
  
 - name: Install kubectl  
 uses: azure/setup-kubectl@v3  
  
 - name: Verify cluster  
 run: |  
 kubectl get nodes  
 kubectl get nodes --show-labels  
  
 deploy-kubernetes:  
 name: Deploy to Kubernetes  
 needs: [build-docker, provision-infrastructure]  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Install doctl  
 uses: digitalocean/action-doctl@v2  
 with:  
 token: ${{ secrets.DIGITALOCEAN\_ACCESS\_TOKEN }}  
  
 - name: Get kubeconfig  
 run: |  
 CLUSTER\_NAME="${{ secrets.CLUSTER\_NAME }}-${{ env.ENVIRONMENT }}"  
 doctl kubernetes cluster kubeconfig save $CLUSTER\_NAME  
  
 - name: Set up kubectl  
 uses: azure/setup-kubectl@v3  
  
 - name: Set up Helm  
 uses: azure/setup-helm@v3  
 with:  
 version: 'v3.10.0'  
  
 - name: Create environment files  
 run: |  
 # Create .env file for production  
 cat > .env.production << EOF  
 # OpenAI API  
 OPENAI\_API\_KEY=${{ secrets.OPENAI\_API\_KEY }}  
  
 # Database  
 MONGODB\_URI=${{ secrets.MONGODB\_URI }}  
  
 # Authentication  
 JWT\_SECRET=${{ secrets.JWT\_SECRET }}  
  
 # Supabase  
 SUPABASE\_URL=${{ secrets.SUPABASE\_URL }}  
 SUPABASE\_KEY=${{ secrets.SUPABASE\_KEY }}  
  
 # Frontend URLs  
 GATSBY\_API\_URL=https://api.${{ secrets.DOMAIN\_NAME }}  
 GATSBY\_SUPABASE\_URL=${{ secrets.SUPABASE\_URL }}  
 GATSBY\_SUPABASE\_ANON\_KEY=${{ secrets.SUPABASE\_ANON\_KEY }}  
  
 # Stripe (if using payments)  
 STRIPE\_SECRET\_KEY=${{ secrets.STRIPE\_SECRET\_KEY }}  
 GATSBY\_STRIPE\_PUBLISHABLE\_KEY=${{ secrets.STRIPE\_PUBLISHABLE\_KEY }}  
 EOF  
  
 - name: Create Kubernetes namespace  
 run: |  
 kubectl create namespace kai-system --dry-run=client -o yaml | kubectl apply -f -  
  
 - name: Create Kubernetes secrets  
 run: |  
 # Create main secrets  
 kubectl create secret generic kai-secrets \  
 --namespace kai-system \  
 --from-literal=mongodb-uri='${{ secrets.MONGODB\_URI }}' \  
 --from-literal=jwt-secret='${{ secrets.JWT\_SECRET }}' \  
 --from-literal=openai-api-key='${{ secrets.OPENAI\_API\_KEY }}' \  
 --from-literal=supabase-url='${{ secrets.SUPABASE\_URL }}' \  
 --from-literal=supabase-key='${{ secrets.SUPABASE\_KEY }}' \  
 --from-literal=stripe-secret-key='${{ secrets.STRIPE\_SECRET\_KEY }}' \  
 --dry-run=client -o yaml | kubectl apply -f -  
  
 # Create Redis password secret  
 kubectl create secret generic redis-password \  
 --namespace kai-system \  
 --from-literal=redis-password='${{ secrets.REDIS\_PASSWORD }}' \  
 --dry-run=client -o yaml | kubectl apply -f -  
  
 - name: Check and Install cert-manager  
 run: |  
 # Check if cert-manager is already installed  
 if kubectl get namespace cert-manager &>/dev/null && kubectl get deployment -n cert-manager cert-manager &>/dev/null; then  
 echo "cert-manager is already installed, skipping installation"  
 else  
 echo "Installing cert-manager..."  
 # Add Jetstack Helm repo  
 helm repo add jetstack https://charts.jetstack.io  
  
 # Install cert-manager  
 helm upgrade --install cert-manager jetstack/cert-manager \  
 --namespace cert-manager \  
 --create-namespace \  
 --version v1.11.0 \  
 --set installCRDs=true  
  
 # Wait for cert-manager to be ready  
 kubectl -n cert-manager rollout status deployment/cert-manager  
 kubectl -n cert-manager rollout status deployment/cert-manager-webhook  
 fi  
  
 # Check if ClusterIssuer exists  
 if kubectl get clusterissuer letsencrypt-prod &>/dev/null; then  
 echo "ClusterIssuer already exists, skipping creation"  
 else  
 echo "Creating ClusterIssuer for Let's Encrypt..."  
 # Create ClusterIssuer for Let's Encrypt  
 cat > cluster-issuer.yaml << EOF  
 apiVersion: cert-manager.io/v1  
 kind: ClusterIssuer  
 metadata:  
 name: letsencrypt-prod  
 spec:  
 acme:  
 server: https://acme-v02.api.letsencrypt.org/directory  
 email: ${{ secrets.ADMIN\_EMAIL }}  
 privateKeySecretRef:  
 name: letsencrypt-prod  
 solvers:  
 - http01:  
 ingress:  
 class: nginx  
 EOF  
  
 kubectl apply -f cluster-issuer.yaml  
 fi  
  
 - name: Check and Install NGINX Ingress Controller  
 run: |  
 # Check if NGINX Ingress is already installed  
 if kubectl get deployment -n kai-system nginx-ingress-ingress-nginx-controller &>/dev/null; then  
 echo "NGINX Ingress Controller is already installed, skipping installation"  
 else  
 echo "Installing NGINX Ingress Controller..."  
 # Add NGINX Ingress Helm repo  
 helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx  
  
 # Install NGINX Ingress  
 helm upgrade --install nginx-ingress ingress-nginx/ingress-nginx \  
 --namespace kai-system \  
 --set controller.publishService.enabled=true  
  
 # Wait for NGINX Ingress to be ready  
 kubectl -n kai-system rollout status deployment/nginx-ingress-ingress-nginx-controller  
 fi  
  
 - name: Check and Install Argo Workflows  
 run: |  
 # Check if Argo Workflows is already installed  
 if kubectl get namespace argo &>/dev/null && kubectl get deployment -n argo argo-server &>/dev/null; then  
 echo "Argo Workflows is already installed, skipping installation"  
 else  
 echo "Installing Argo Workflows..."  
 # Create namespace  
 kubectl create namespace argo --dry-run=client -o yaml | kubectl apply -f -  
  
 # Install Argo Workflows  
 kubectl apply -n argo -f https://github.com/argoproj/argo-workflows/releases/download/v3.4.5/install.yaml  
  
 # Wait for Argo to be ready  
 kubectl -n argo rollout status deployment/argo-server  
 fi  
  
 # Check and update Argo configuration  
 echo "Configuring Argo to work with the kai-system namespace..."  
 kubectl patch configmap/workflow-controller-configmap \  
 -n argo \  
 --type merge \  
 -p '{"data":{"workflowNamespaces":"kai-system,argo"}}' \  
 --dry-run=client -o yaml | kubectl apply -f -  
  
 - name: Deploy with Helm  
 run: |  
 # Update Helm repos  
 helm repo update  
  
 # Create values override file  
 cat > values-override.yaml << EOF  
 global:  
 environment: ${{ env.ENVIRONMENT }}  
 registry:  
 url: "ghcr.io"  
 repository: ${{ github.repository }}  
 image:  
 tag: ${{ github.sha }}  
 domain: ${{ secrets.DOMAIN\_NAME }}  
 ingress:  
 enabled: true  
 annotations:  
 kubernetes.io/ingress.class: nginx  
 cert-manager.io/cluster-issuer: letsencrypt-prod  
 hosts:  
 - host: api.${{ secrets.DOMAIN\_NAME }}  
 paths:  
 - path: /  
 pathType: Prefix  
 tls:  
 - secretName: kai-tls-cert  
 hosts:  
 - api.${{ secrets.DOMAIN\_NAME }}  
 EOF  
  
 # Deploy using Helm  
 helm upgrade --install kai ./helm-charts/kai \  
 --namespace kai-system \  
 --values ./helm-charts/kai/values-${{ env.ENVIRONMENT }}.yaml \  
 --values values-override.yaml  
  
 - name: Verify deployment  
 run: |  
 # Wait for deployments to be ready  
 kubectl -n kai-system rollout status deployment/api-server  
 kubectl -n kai-system rollout status deployment/coordinator-service  
  
 # Check if services are running  
 kubectl -n kai-system get services  
  
 # Check if ingress is configured  
 kubectl -n kai-system get ingress  
  
 # Check if certificates are issued  
 kubectl -n kai-system get certificates  
  
 - name: Run database migrations  
 run: |  
 # Create a temporary pod to run migrations  
 cat > migration-job.yaml << EOF  
 apiVersion: batch/v1  
 kind: Job  
 metadata:  
 name: database-migrations  
 namespace: kai-system  
 spec:  
 template:  
 spec:  
 containers:  
 - name: migrations  
 image: ghcr.io/${{ github.repository }}/kai-api:${{ github.sha }}  
 command: ["node", "scripts/run-migrations.js"]  
 env:  
 - name: MONGODB\_URI  
 valueFrom:  
 secretKeyRef:  
 name: kai-secrets  
 key: mongodb-uri  
 restartPolicy: Never  
 backoffLimit: 4  
 EOF  
  
 kubectl apply -f migration-job.yaml  
  
 # Wait for migrations to complete  
 kubectl -n kai-system wait --for=condition=complete job/database-migrations --timeout=120s  
  
 - name: Notify deployment status  
 if: always()  
 uses: rtCamp/action-slack-notify@v2  
 env:  
 SLACK\_WEBHOOK: ${{ secrets.SLACK\_WEBHOOK }}  
 SLACK\_TITLE: "Deployment Status"  
 SLACK\_MESSAGE: "Kubernetes deployment ${{ job.status }}"  
 SLACK\_COLOR: ${{ job.status == 'success' && 'good' || 'danger' }}  
  
 deploy-frontend:  
 name: Deploy Frontend to Vercel  
 needs: deploy-kubernetes  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Set up Node.js  
 uses: actions/setup-node@v3  
 with:  
 node-version: '16'  
 cache: 'yarn'  
  
 - name: Install dependencies  
 run: yarn install --frozen-lockfile  
  
 - name: Deploy to Vercel  
 uses: amondnet/vercel-action@v20  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_PROJECT\_ID\_CLIENT }}  
 working-directory: ./packages/client  
 vercel-args: '--prod'  
  
 - name: Deploy admin panel to Vercel  
 uses: amondnet/vercel-action@v20  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_PROJECT\_ID\_ADMIN }}  
 working-directory: ./packages/admin  
 vercel-args: '--prod'  
  
 verify-deployment:  
 name: Verify Full Deployment  
 needs: [deploy-kubernetes, deploy-frontend]  
 runs-on: ubuntu-latest  
 steps:  
 - name: Comprehensive health check  
 run: |  
 # Wait for DNS propagation (may take some time)  
 echo "Waiting for DNS propagation..."  
 sleep 60  
  
 # Check API health endpoint  
 curl -sSf https://api.${{ secrets.DOMAIN\_NAME }}/health || echo "API not yet available, may need more time for DNS propagation"  
  
 # Check pod status  
 kubectl get pods -n kai-system -o json | jq '.items[] | select(.status.phase != "Running" or ([ .status.containerStatuses[] | select(.ready == false) ] | length > 0)) | .metadata.name' || echo "All pods are running"  
  
 # Check recent logs for errors  
 kubectl logs -n kai-system -l app=api-server --tail=50 | grep -i error || echo "No errors in recent logs"  
  
 # Check SSL certificate validity and expiration  
 CERT\_EXPIRY=$(echo | openssl s\_client -servername api.${{ secrets.DOMAIN\_NAME }} -connect api.${{ secrets.DOMAIN\_NAME }}:443 2>/dev/null | openssl x509 -noout -enddate | cut -d= -f2 || echo "Certificate not yet available")  
 echo "Certificate expires on: $CERT\_EXPIRY"  
  
 - name: Send deployment notification  
 uses: rtCamp/action-slack-notify@v2  
 env:  
 SLACK\_WEBHOOK: ${{ secrets.SLACK\_WEBHOOK }}  
 SLACK\_TITLE: "Deployment Complete"  
 SLACK\_MESSAGE: "✅ KAI Platform has been successfully deployed to ${{ env.ENVIRONMENT }}!"  
 SLACK\_COLOR: "good"

#### Workflow Explanation

The GitHub Actions workflow above automates the entire deployment process:

1. Build and Test:
2. Checks out the code
3. Installs dependencies
4. Builds all packages
5. Runs tests
6. Uploads build artifacts for later use
7. Build Docker Images:
8. Downloads the build artifacts
9. Builds Docker images for all services
10. Pushes images to the Docker registry
11. Provision Infrastructure:
12. Checks if a Kubernetes cluster already exists
13. Creates a new cluster if needed with all required node pools
14. Configures the cluster with appropriate labels and tags
15. Deploy to Kubernetes:
16. Creates necessary namespaces
17. Creates Kubernetes secrets from GitHub secrets
18. Installs cert-manager for SSL certificates (if not already installed)
19. Installs NGINX Ingress Controller (if not already installed)
20. Installs Argo Workflows for ML pipelines (if not already installed)
21. Deploys the application using Helm charts
22. Verifies the deployment
23. Runs database migrations
24. Deploy Frontend:
25. Deploys the client frontend to Vercel
26. Deploys the admin panel to Vercel
27. Verify Full Deployment:
28. Performs comprehensive health checks
29. Verifies API availability
30. Checks pod status
31. Examines logs for errors
32. Verifies SSL certificate validity
33. Sends a notification when deployment is complete

Uploads build artifacts for later use

Build Docker Images:

Pushes images to the Docker registry

Provision Infrastructure:

Configures the cluster with appropriate labels and tags

Deploy to Kubernetes:

Runs database migrations

Deploy Frontend:

Deploys the admin panel to Vercel

Verify Full Deployment:

### Digital Ocean Kubernetes Cluster Configuration

The KAI platform is designed to run on a Kubernetes cluster with specific node pools optimized for different workloads. This section details the cluster configuration required for optimal performance.

#### Cluster Requirements

Based on the KAI platform's architecture and resource needs, the following cluster configuration is recommended:

1. Kubernetes Version: Latest stable version (1.32.x or newer)
2. Region: Choose the region closest to your users
3. High Availability: At least 3 nodes in the orchestration pool for production

#### Node Pool Configuration

The KAI platform requires several specialized node pools to handle different types of workloads:

##### 1. Orchestration Pool

* Purpose: Runs the API server, coordinator service, and other control plane components
* Machine Type: Standard Droplets
* Size: 4GB RAM / 2 vCPUs (s-2vcpu-4gb)
* Count: 3 nodes (for high availability in production)
* Labels: node-type=orchestration

node-type=orchestration

##### 2. CPU-Optimized Pool

* Purpose: Handles general processing, data transformation, and non-GPU workloads
* Machine Type: CPU-Optimized Droplets
* Size: 8GB RAM / 4 vCPUs (c-4)
* Count: 3 nodes
* Labels: node-type=cpu-optimized

node-type=cpu-optimized

##### 3. GPU Pool

* Purpose: Runs ML inference and training tasks, 3D model generation
* Machine Type: GPU Droplets
* Size: With NVIDIA L40S or H100 GPUs
* Count: 2 nodes
* Labels: node-type=gpu-optimized
* Note: Required for production, optional for staging

node-type=gpu-optimized

##### 4. Memory-Optimized Pool

* Purpose: Handles large model loading and memory-intensive operations
* Machine Type: Memory-Optimized Droplets
* Size: 32GB RAM / 4 vCPUs (m-4vcpu-32gb)
* Count: 1 node
* Labels: node-type=memory-optimized
* Note: Required for production, optional for staging

node-type=memory-optimized

#### Resource Allocation

The KAI platform components have specific resource requirements that are automatically configured based on the environment:

Table content:

Component | Resource | Staging | Production

API Server | Replicas | 1 | 3

API Server | CPU Request | 200m | 500m

API Server | Memory Request | 512Mi | 1Gi

API Server | CPU Limit | 1000m | 2000m

API Server | Memory Limit | 2Gi | 4Gi

Coordinator | Replicas | 1 | 3

Coordinator | CPU Request | 200m | 500m

Coordinator | Memory Request | 512Mi | 1Gi

Coordinator | CPU Limit | 1000m | 2000m

Coordinator | Memory Limit | 2Gi | 4Gi

ML Services | Replicas | 1 | 1-3 (auto-scaled)

ML Services | CPU Request | 500m-1000m | 1000m-4000m

ML Services | Memory Request | 1Gi-2Gi | 2Gi-8Gi

ML Services | GPU Request | 0-1 | 1

Notification | Replicas | 1 | 3

Notification | CPU Request | 200m | 500m

Notification | Memory Request | 256Mi | 512Mi

Material Property Analytics | Replicas | 1 | 2 (auto-scaled to 6)

Material Property Analytics | CPU Request | 500m | 1000m

Material Property Analytics | Memory Request | 1Gi | 2Gi

Material Property Analytics | CPU Limit | 2000m | 4000m

Material Property Analytics | Memory Limit | 4Gi | 8Gi

Relationship-Aware Training | Replicas | 1 | 1 (auto-scaled to 3)

Relationship-Aware Training | CPU Request | 2000m | 4000m

Relationship-Aware Training | Memory Request | 8Gi | 16Gi

Relationship-Aware Training | CPU Limit | 4000m | 8000m

Relationship-Aware Training | Memory Limit | 16Gi | 32Gi

Relationship-Aware Training | GPU Request | 1 | 1

### SSL Certificate Management

The KAI platform uses cert-manager to automatically manage SSL certificates. This section explains how certificates are issued and renewed.

#### Certificate Issuance

When you deploy the KAI platform, the GitHub Actions workflow automatically:

1. Installs cert-manager in the cluster (if not already installed)
2. Creates a ClusterIssuer for Let's Encrypt
3. Configures the ingress resources with appropriate annotations
4. Requests certificates for all configured domains

#### Automatic Certificate Renewal

Cert-manager handles certificate renewal automatically:

1. Monitoring: Cert-manager continuously monitors certificate expiration dates
2. Proactive Renewal: It automatically initiates renewal when certificates reach ~30 days before expiration
3. Zero-downtime Process: New certificates are obtained in the background and only replaced after successful validation
4. Failure Handling: If renewal fails, cert-manager retries with exponential backoff

#### Certificate Verification

You can verify the status of your certificates using:

kubectl get certificates -n kai-system  
kubectl get certificaterequests -n kai-system  
kubectl describe certificate kai-tls-cert -n kai-system

kubectl get certificates -n kai-system  
kubectl get certificaterequests -n kai-system  
kubectl describe certificate kai-tls-cert -n kai-system

### Frontend Deployment to Vercel

The KAI platform frontend applications are deployed to Vercel for optimal performance and reliability.

#### Vercel Projects Setup

Before running the GitHub Actions workflow, you need to set up two projects in Vercel:

1. Client Frontend (Gatsby):
2. Create a new project in Vercel
3. Connect to your GitHub repository
4. Configure the project:  
   Framework Preset: Gatsby  
   Root Directory: packages/client  
   Build Command: yarn build  
   Output Directory: public
5. Note the Project ID for the GitHub secret VERCEL\_PROJECT\_ID\_CLIENT
6. Admin Panel (Next.js):
7. Create a new project in Vercel
8. Connect to your GitHub repository
9. Configure the project:  
   Framework Preset: Next.js  
   Root Directory: packages/admin  
   Build Command: yarn build  
   Output Directory: out
10. Note the Project ID for the GitHub secret VERCEL\_PROJECT\_ID\_ADMIN

* Framework Preset: Gatsby
* Root Directory: packages/client
* Build Command: yarn build
* Output Directory: public

packages/client

yarn build

public

Note the Project ID for the GitHub secret VERCEL\_PROJECT\_ID\_CLIENT

VERCEL\_PROJECT\_ID\_CLIENT

Admin Panel (Next.js):

* Framework Preset: Next.js
* Root Directory: packages/admin
* Build Command: yarn build
* Output Directory: out

packages/admin

yarn build

out

VERCEL\_PROJECT\_ID\_ADMIN

#### Vercel Environment Variables

The GitHub Actions workflow automatically sets the required environment variables for your Vercel deployments, including:

* API URL
* Supabase configuration
* Stripe keys (if using payments)
* Other application-specific settings

#### Custom Domain Configuration

After the first deployment, you should configure custom domains in Vercel:

1. Go to your Vercel project settings
2. Navigate to the "Domains" section
3. Add your custom domains:
4. app.yourdomain.com for the client frontend
5. admin.yourdomain.com for the admin panel
6. Configure DNS records as instructed by Vercel

app.yourdomain.com

admin.yourdomain.com

### Environment Variables and Secrets

The KAI platform uses a centralized approach to environment variables for configuration. We use a single .env file in the root directory for both local development and CI/CD workflows.

.env

#### Environment File Structure

The KAI platform uses a structured approach to environment variables:

1. Base Configuration: .env file contains all sensitive configuration variables
2. Environment-Specific Overrides: Separate files for each environment

.env

##### File Organization

* .env - Contains sensitive information (API keys, secrets, credentials)
* .env.template - Template for .env with empty values (committed to repository)
* .env.development - Development-specific configuration (non-sensitive)
* .env.staging - Staging-specific configuration (non-sensitive)
* .env.production - Production-specific configuration (non-sensitive)

.env

.env.template

.env

.env.development

.env.staging

.env.production

##### Setup Instructions

1. Copy .env.template to .env:  
    bash  
    cp .env.template .env
2. Fill in the sensitive values in .env:  
    ```bash  
    # Required API keys  
    OPENAI\_API\_KEY=your\_openai\_api\_key  
    JWT\_SECRET=your\_jwt\_secret  
    MONGODB\_URI=mongodb://localhost:27017/kai

Copy .env.template to .env:  
 bash  
 cp .env.template .env

.env.template

.env

bash  
 cp .env.template .env

Fill in the sensitive values in .env:  
 ```bash  
 # Required API keys  
 OPENAI\_API\_KEY=your\_openai\_api\_key  
 JWT\_SECRET=your\_jwt\_secret  
 MONGODB\_URI=mongodb://localhost:27017/kai

.env

# S3/Storage credentials  
 S3\_ACCESS\_KEY=your\_s3\_access\_key  
 S3\_SECRET\_KEY=your\_s3\_secret\_key

# Supabase credentials  
 SUPABASE\_KEY=your\_supabase\_key  
 SUPABASE\_ANON\_KEY=your\_supabase\_anon\_key

# Other sensitive values...  
 ```

1. The environment-specific files (.env.development, .env.staging, .env.production) already contain appropriate non-sensitive configuration for each environment:

.env.development

.env.staging

.env.production

##### Environment-Specific Configuration

Each environment has its own configuration file with appropriate values:

Development Environment (.env.development):

.env.development

NODE\_ENV=development  
KAI\_API\_URL=http://localhost:3000/api  
MONGODB\_URI=mongodb://localhost:27017/kai-dev  
LOG\_LEVEL=debug  
ENABLE\_MOCK\_FALLBACK=true  
USE\_MCP\_SERVER=false  
STRIPE\_TEST\_MODE=true

NODE\_ENV=development  
KAI\_API\_URL=http://localhost:3000/api  
MONGODB\_URI=mongodb://localhost:27017/kai-dev  
LOG\_LEVEL=debug  
ENABLE\_MOCK\_FALLBACK=true  
USE\_MCP\_SERVER=false  
STRIPE\_TEST\_MODE=true

Staging Environment (.env.staging):

.env.staging

NODE\_ENV=production  
KAI\_API\_URL=https://api.staging.kai-platform.com/api  
DATABASE\_SSL=true  
LOG\_LEVEL=info  
ENABLE\_MOCK\_FALLBACK=false  
USE\_MCP\_SERVER=true  
STRIPE\_TEST\_MODE=true

NODE\_ENV=production  
KAI\_API\_URL=https://api.staging.kai-platform.com/api  
DATABASE\_SSL=true  
LOG\_LEVEL=info  
ENABLE\_MOCK\_FALLBACK=false  
USE\_MCP\_SERVER=true  
STRIPE\_TEST\_MODE=true

Production Environment (.env.production):

.env.production

NODE\_ENV=production  
KAI\_API\_URL=https://api.kai-platform.com/api  
DATABASE\_SSL=true  
LOG\_LEVEL=warn  
ENABLE\_MOCK\_FALLBACK=false  
USE\_MCP\_SERVER=true  
STRIPE\_TEST\_MODE=false

NODE\_ENV=production  
KAI\_API\_URL=https://api.kai-platform.com/api  
DATABASE\_SSL=true  
LOG\_LEVEL=warn  
ENABLE\_MOCK\_FALLBACK=false  
USE\_MCP\_SERVER=true  
STRIPE\_TEST\_MODE=false

These files override the values in the main .env file when the corresponding environment is active.

.env

#### GitHub Actions Integration

Our GitHub Actions workflows use a custom action to load environment variables from the .env file. This ensures consistency between local development and CI/CD environments.

.env

The action is located at .github/actions/load-env and is used in all workflows that need environment variables.

.github/actions/load-env

For sensitive information that should not be committed to the repository, use GitHub Secrets instead. These secrets are referenced in the workflows and are injected into the Kubernetes deployment as environment variables and secrets.

#### GitHub Secrets

All sensitive information is stored as GitHub Secrets and injected into the deployment process by the GitHub Actions workflow. This includes:

* API keys
* Database credentials
* JWT secrets
* Supabase credentials
* Vercel tokens
* Docker registry credentials

#### Kubernetes Secrets

The GitHub Actions workflow creates Kubernetes secrets from the GitHub Secrets, making them available to the applications running in the cluster:

# Example of how secrets are created  
kubectl create secret generic kai-secrets \  
 --namespace kai-system \  
 --from-literal=mongodb-uri='${{ secrets.MONGODB\_URI }}' \  
 --from-literal=jwt-secret='${{ secrets.JWT\_SECRET }}' \  
 --from-literal=openai-api-key='${{ secrets.OPENAI\_API\_KEY }}' \  
 --from-literal=supabase-url='${{ secrets.SUPABASE\_URL }}' \  
 --from-literal=supabase-key='${{ secrets.SUPABASE\_KEY }}' \  
 --from-literal=stripe-secret-key='${{ secrets.STRIPE\_SECRET\_KEY }}' \  
 --dry-run=client -o yaml | kubectl apply -f -

# Example of how secrets are created  
kubectl create secret generic kai-secrets \  
 --namespace kai-system \  
 --from-literal=mongodb-uri='${{ secrets.MONGODB\_URI }}' \  
 --from-literal=jwt-secret='${{ secrets.JWT\_SECRET }}' \  
 --from-literal=openai-api-key='${{ secrets.OPENAI\_API\_KEY }}' \  
 --from-literal=supabase-url='${{ secrets.SUPABASE\_URL }}' \  
 --from-literal=supabase-key='${{ secrets.SUPABASE\_KEY }}' \  
 --from-literal=stripe-secret-key='${{ secrets.STRIPE\_SECRET\_KEY }}' \  
 --dry-run=client -o yaml | kubectl apply -f -

#### Environment-Specific Configuration in Kubernetes

The KAI platform uses different configuration values for staging and production environments:

1. Helm Values Files:
2. values.yaml: Default values
3. values-staging.yaml: Staging-specific overrides
4. values-production.yaml: Production-specific overrides
5. Environment Variables:
6. The GitHub Actions workflow loads environment variables from the .env file
7. Environment-specific variables are loaded from .env.staging or .env.production
8. These variables are used during the build process and deployment

values.yaml

values-staging.yaml

values-production.yaml: Production-specific overrides

values-production.yaml

Environment Variables:

.env

.env.staging

.env.production

### Recently Implemented Features

The KAI platform has recently been enhanced with several new features that require specific deployment configurations. This section provides details on these features and their deployment requirements.

#### Property-Based Recommendation Engine

The Property-Based Recommendation Engine is integrated with the API server and provides intelligent material recommendations based on property requirements, user preferences, and project context.

##### Deployment Configuration

The recommendation engine is deployed as part of the API server with the following configuration:

* Deployment Method: Integrated with the API server deployment
* Configuration: ConfigMap mounted to the API server
* Resource Requirements: Included in the API server resource allocation
* Scaling: Scales with the API server

##### Implementation Details

The recommendation engine is implemented as a ConfigMap that is mounted to the API server:

# ConfigMap for the recommendation engine  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: recommendation-engine-config  
 labels:  
 app: api-server  
 component: recommendation  
data:  
 recommendation\_engine\_enabled: "true"  
 recommendation\_cache\_size: "1000"  
 recommendation\_refresh\_interval: "300" # 5 minutes  
 recommendation\_max\_concurrent\_jobs: "5"  
 recommendation\_similarity\_threshold: "0.7"  
 recommendation\_config.json: |  
 {  
 "weightFactors": {  
 "propertyMatch": 0.6,  
 "userPreference": 0.3,  
 "projectContext": 0.1  
 },  
 "cacheStrategy": "lru",  
 "defaultLimit": 20,  
 "enableFeedbackLoop": true,  
 "minConfidenceScore": 0.65  
 }

# ConfigMap for the recommendation engine  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: recommendation-engine-config  
 labels:  
 app: api-server  
 component: recommendation  
data:  
 recommendation\_engine\_enabled: "true"  
 recommendation\_cache\_size: "1000"  
 recommendation\_refresh\_interval: "300" # 5 minutes  
 recommendation\_max\_concurrent\_jobs: "5"  
 recommendation\_similarity\_threshold: "0.7"  
 recommendation\_config.json: |  
 {  
 "weightFactors": {  
 "propertyMatch": 0.6,  
 "userPreference": 0.3,  
 "projectContext": 0.1  
 },  
 "cacheStrategy": "lru",  
 "defaultLimit": 20,  
 "enableFeedbackLoop": true,  
 "minConfidenceScore": 0.65  
 }

The API server deployment is patched to include the recommendation engine configuration:

# Patch for the API server deployment  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: api-server  
spec:  
 template:  
 spec:  
 containers:  
 - name: api-server  
 env:  
 - name: RECOMMENDATION\_ENGINE\_ENABLED  
 valueFrom:  
 configMapKeyRef:  
 name: recommendation-engine-config  
 key: recommendation\_engine\_enabled  
 # Additional environment variables...  
 volumeMounts:  
 - name: recommendation-config  
 mountPath: /app/config/recommendation  
 readOnly: true  
 volumes:  
 - name: recommendation-config  
 configMap:  
 name: recommendation-engine-config

# Patch for the API server deployment  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: api-server  
spec:  
 template:  
 spec:  
 containers:  
 - name: api-server  
 env:  
 - name: RECOMMENDATION\_ENGINE\_ENABLED  
 valueFrom:  
 configMapKeyRef:  
 name: recommendation-engine-config  
 key: recommendation\_engine\_enabled  
 # Additional environment variables...  
 volumeMounts:  
 - name: recommendation-config  
 mountPath: /app/config/recommendation  
 readOnly: true  
 volumes:  
 - name: recommendation-config  
 configMap:  
 name: recommendation-engine-config

#### Material Property Analytics

The Material Property Analytics feature provides comprehensive analytics for material properties, including distribution analysis, trend analysis, correlation analysis, and anomaly detection.

##### Deployment Configuration

Material Property Analytics is deployed as a separate service with the following configuration:

* Deployment Method: Dedicated deployment
* Resource Requirements:
* CPU: 1000m (request) / 4000m (limit)
* Memory: 2Gi (request) / 8Gi (limit)
* Scaling: HorizontalPodAutoscaler with 2-6 replicas
* Storage: 20Gi PersistentVolumeClaim for analytics data
* Workflow Integration: Integrated with the coordinator service via workflow templates

##### Implementation Details

The Material Property Analytics service is deployed as a separate Kubernetes deployment:

# Deployment for Material Property Analytics  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: material-property-analytics  
 labels:  
 app: material-property-analytics  
 component: analytics  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: material-property-analytics  
 template:  
 metadata:  
 labels:  
 app: material-property-analytics  
 component: analytics  
 annotations:  
 prometheus.io/scrape: "true"  
 prometheus.io/port: "9100"  
 prometheus.io/path: "/metrics"  
 spec:  
 containers:  
 - name: material-property-analytics  
 image: ${REGISTRY\_URL}/kai/material-property-analytics:latest  
 # Container configuration...

# Deployment for Material Property Analytics  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: material-property-analytics  
 labels:  
 app: material-property-analytics  
 component: analytics  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: material-property-analytics  
 template:  
 metadata:  
 labels:  
 app: material-property-analytics  
 component: analytics  
 annotations:  
 prometheus.io/scrape: "true"  
 prometheus.io/port: "9100"  
 prometheus.io/path: "/metrics"  
 spec:  
 containers:  
 - name: material-property-analytics  
 image: ${REGISTRY\_URL}/kai/material-property-analytics:latest  
 # Container configuration...

The service is integrated with the coordinator service via a workflow template:

# Workflow template for Material Property Analytics  
apiVersion: argoproj.io/v1alpha1  
kind: WorkflowTemplate  
metadata:  
 name: material-property-analytics-template  
spec:  
 entrypoint: material-property-analytics  
 # Workflow template configuration...

# Workflow template for Material Property Analytics  
apiVersion: argoproj.io/v1alpha1  
kind: WorkflowTemplate  
metadata:  
 name: material-property-analytics-template  
spec:  
 entrypoint: material-property-analytics  
 # Workflow template configuration...

#### Relationship-Aware Model Training

The Relationship-Aware Model Training feature enhances AI model training by incorporating knowledge from the Property Relationship Graph to improve property prediction and search relevance.

##### Deployment Configuration

Relationship-Aware Model Training is deployed as a separate service with the following configuration:

* Deployment Method: Dedicated deployment
* Resource Requirements:
* CPU: 4000m (request) / 8000m (limit)
* Memory: 16Gi (request) / 32Gi (limit)
* GPU: 1 NVIDIA GPU (L40S preferred)
* Scaling: HorizontalPodAutoscaler with 1-3 replicas
* Storage: 40Gi PersistentVolumeClaim for model storage
* Node Selection: Runs on nodes with gpu-type=nvidia-l40s label
* Workflow Integration: Integrated with the coordinator service via workflow templates

gpu-type=nvidia-l40s

##### Implementation Details

The Relationship-Aware Model Training service is deployed as a separate Kubernetes deployment:

# Deployment for Relationship-Aware Model Training  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: relationship-aware-training  
 labels:  
 app: relationship-aware-training  
 component: ml-services  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: relationship-aware-training  
 template:  
 metadata:  
 labels:  
 app: relationship-aware-training  
 component: ml-services  
 gpu-enabled: "true"  
 annotations:  
 prometheus.io/scrape: "true"  
 prometheus.io/port: "9100"  
 prometheus.io/path: "/metrics"  
 spec:  
 containers:  
 - name: relationship-aware-training  
 image: ${REGISTRY\_URL}/kai/relationship-aware-training:latest  
 # Container configuration...  
 nodeSelector:  
 gpu-type: nvidia-l40s  
 runtimeClassName: nvidia-gpu

# Deployment for Relationship-Aware Model Training  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: relationship-aware-training  
 labels:  
 app: relationship-aware-training  
 component: ml-services  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: relationship-aware-training  
 template:  
 metadata:  
 labels:  
 app: relationship-aware-training  
 component: ml-services  
 gpu-enabled: "true"  
 annotations:  
 prometheus.io/scrape: "true"  
 prometheus.io/port: "9100"  
 prometheus.io/path: "/metrics"  
 spec:  
 containers:  
 - name: relationship-aware-training  
 image: ${REGISTRY\_URL}/kai/relationship-aware-training:latest  
 # Container configuration...  
 nodeSelector:  
 gpu-type: nvidia-l40s  
 runtimeClassName: nvidia-gpu

The service is integrated with the coordinator service via a workflow template:

# Workflow template for Relationship-Aware Model Training  
apiVersion: argoproj.io/v1alpha1  
kind: WorkflowTemplate  
metadata:  
 name: relationship-aware-training-template  
spec:  
 entrypoint: relationship-aware-training  
 # Workflow template configuration...

# Workflow template for Relationship-Aware Model Training  
apiVersion: argoproj.io/v1alpha1  
kind: WorkflowTemplate  
metadata:  
 name: relationship-aware-training-template  
spec:  
 entrypoint: relationship-aware-training  
 # Workflow template configuration...

#### Coordinator Service Integration

The coordinator service has been updated to be aware of the new services and features:

# ML Features Configuration in coordinator-config ConfigMap  
ml\_features: |  
 {  
 "multimodalPatternRecognition": {  
 "enabled": true,  
 "workflowTemplate": "multimodal-pattern-recognition-template",  
 "minQualityTier": "medium",  
 "resourceQuotas": {  
 "cpu": 8,  
 "memory": "16Gi",  
 "gpu": 1  
 },  
 "maxConcurrent": 4  
 },  
 "domainSpecificNetworks": {  
 "enabled": true,  
 "workflowTemplate": "domain-specific-networks-template",  
 "minQualityTier": "medium",  
 "resourceQuotas": {  
 "cpu": 8,  
 "memory": "16Gi",  
 "gpu": 1  
 },  
 "maxConcurrent": 4,  
 "supportedDomains": ["wood", "metal", "fabric", "stone", "ceramic", "composite"]  
 },  
 "relationshipAwareTraining": {  
 "enabled": true,  
 "workflowTemplate": "relationship-aware-training-template",  
 "minQualityTier": "high",  
 "resourceQuotas": {  
 "cpu": 8,  
 "memory": "32Gi",  
 "gpu": 1  
 },  
 "maxConcurrent": 2,  
 "supportedMaterialTypes": ["tile", "wood", "stone", "laminate", "vinyl", "carpet", "metal", "glass", "concrete", "ceramic", "porcelain"]  
 },  
 "materialPropertyAnalytics": {  
 "enabled": true,  
 "workflowTemplate": "material-property-analytics-template",  
 "minQualityTier": "medium",  
 "resourceQuotas": {  
 "cpu": 4,  
 "memory": "8Gi",  
 "gpu": 0  
 },  
 "maxConcurrent": 5  
 }  
 }

# ML Features Configuration in coordinator-config ConfigMap  
ml\_features: |  
 {  
 "multimodalPatternRecognition": {  
 "enabled": true,  
 "workflowTemplate": "multimodal-pattern-recognition-template",  
 "minQualityTier": "medium",  
 "resourceQuotas": {  
 "cpu": 8,  
 "memory": "16Gi",  
 "gpu": 1  
 },  
 "maxConcurrent": 4  
 },  
 "domainSpecificNetworks": {  
 "enabled": true,  
 "workflowTemplate": "domain-specific-networks-template",  
 "minQualityTier": "medium",  
 "resourceQuotas": {  
 "cpu": 8,  
 "memory": "16Gi",  
 "gpu": 1  
 },  
 "maxConcurrent": 4,  
 "supportedDomains": ["wood", "metal", "fabric", "stone", "ceramic", "composite"]  
 },  
 "relationshipAwareTraining": {  
 "enabled": true,  
 "workflowTemplate": "relationship-aware-training-template",  
 "minQualityTier": "high",  
 "resourceQuotas": {  
 "cpu": 8,  
 "memory": "32Gi",  
 "gpu": 1  
 },  
 "maxConcurrent": 2,  
 "supportedMaterialTypes": ["tile", "wood", "stone", "laminate", "vinyl", "carpet", "metal", "glass", "concrete", "ceramic", "porcelain"]  
 },  
 "materialPropertyAnalytics": {  
 "enabled": true,  
 "workflowTemplate": "material-property-analytics-template",  
 "minQualityTier": "medium",  
 "resourceQuotas": {  
 "cpu": 4,  
 "memory": "8Gi",  
 "gpu": 0  
 },  
 "maxConcurrent": 5  
 }  
 }

#### GPU Resource Management

The GPU resource management configuration has been updated to include the new Relationship-Aware Training service:

# GPU Resource Management in coordinator-config ConfigMap  
gpu\_resources: |  
 {  
 "classes": {  
 "nvidia-l40s": {  
 "priority": 10,  
 "models": ["multimodal-pattern-recognition", "domain-specific-networks", "relationship-aware-training"]  
 },  
 "nvidia-h100": {  
 "priority": 20,  
 "models": ["gaussian-splatting", "triposr", "wonder3d", "instant3d"]  
 }  
 },  
 "scaling": {  
 "enabled": true,  
 "minNodes": 1,  
 "maxNodes": 10,  
 "scaleDownDelay": "10m"  
 }  
 }

# GPU Resource Management in coordinator-config ConfigMap  
gpu\_resources: |  
 {  
 "classes": {  
 "nvidia-l40s": {  
 "priority": 10,  
 "models": ["multimodal-pattern-recognition", "domain-specific-networks", "relationship-aware-training"]  
 },  
 "nvidia-h100": {  
 "priority": 20,  
 "models": ["gaussian-splatting", "triposr", "wonder3d", "instant3d"]  
 }  
 },  
 "scaling": {  
 "enabled": true,  
 "minNodes": 1,  
 "maxNodes": 10,  
 "scaleDownDelay": "10m"  
 }  
 }

#### GitOps Integration

All the new features are integrated with the GitOps workflow using Flux CD. The deployment configurations are stored in the GitOps repository and automatically applied by Flux.

##### Updated Kustomization Structure

The Flux GitOps repository has been updated with new directories for the recently implemented features:

flux/  
├── clusters/  
│ ├── production/  
│ │ ├── kai/  
│ │ │ ├── analytics/ # New directory for Material Property Analytics  
│ │ │ ├── api-server/ # Updated with recommendation engine config  
│ │ │ ├── coordinator/ # Updated with new workflow templates  
│ │ │ ├── ml-services/ # Updated with relationship-aware training  
│ │ │ └── kustomization.yaml # Updated to include new components

flux/  
├── clusters/  
│ ├── production/  
│ │ ├── kai/  
│ │ │ ├── analytics/ # New directory for Material Property Analytics  
│ │ │ ├── api-server/ # Updated with recommendation engine config  
│ │ │ ├── coordinator/ # Updated with new workflow templates  
│ │ │ ├── ml-services/ # Updated with relationship-aware training  
│ │ │ └── kustomization.yaml # Updated to include new components

##### Deployment Process

The deployment process for the new features follows the existing GitOps workflow:

1. Changes are committed to the GitOps repository
2. Flux automatically detects the changes and applies them to the cluster
3. The new services are deployed and integrated with the existing services

##### Resource Requirements

The new features have specific resource requirements that should be considered when planning cluster capacity:

Table content:

Feature | CPU | Memory | GPU | Storage

Property-Based Recommendation Engine | Included in API server | Included in API server | N/A | N/A

Material Property Analytics | 1-4 cores | 2-8 GB | N/A | 20 GB

Relationship-Aware Model Training | 4-8 cores | 16-32 GB | 1 NVIDIA L40S | 40 GB

##### Monitoring and Logging

All new services are configured with Prometheus metrics and structured logging:

* Prometheus metrics exposed on port 9100
* Metrics path: /metrics
* Log format: JSON
* Log level: Configurable via environment variables

/metrics

##### Health Checks

All new services include appropriate health checks:

* Liveness probe: /health
* Readiness probe: /ready
* Startup probe (where applicable): /startup

/health

/ready

/startup

### Initial Setup and Deployment

To get started with the automated deployment system, follow these steps:

#### 1. Set Up GitHub Repository

1. Configure GitHub Secrets:
2. Go to your GitHub repository
3. Navigate to Settings > Secrets and variables > Actions
4. Add all the required secrets listed in the "Required GitHub Secrets" section

#### 2. Set Up External Services

1. Set Up Supabase Projects:
2. Create a staging project in Supabase
3. Create a production project in Supabase
4. Set up the necessary tables and storage buckets
5. Note the URLs and API keys for the GitHub secrets
6. Set Up Vercel Projects:
7. Create a project for the client frontend  
   Framework Preset: Gatsby  
   Root Directory: packages/client
8. Create a project for the admin panel  
   Framework Preset: Next.js  
   Root Directory: packages/admin
9. Note the project IDs for the GitHub secrets
10. Configure DNS:
11. Set up your domain with your DNS provider
12. Create records for:  
    api.yourdomain.com (production API)  
    api.staging.yourdomain.com (staging API)  
    app.yourdomain.com (production frontend)  
    app.staging.yourdomain.com (staging frontend)  
    admin.yourdomain.com (production admin panel)  
    admin.staging.yourdomain.com (staging admin panel)

Note the URLs and API keys for the GitHub secrets

Set Up Vercel Projects:

* Framework Preset: Gatsby
* Root Directory: packages/client

packages/client

* Framework Preset: Next.js
* Root Directory: packages/admin

packages/admin

Note the project IDs for the GitHub secrets

Configure DNS:

* api.yourdomain.com (production API)
* api.staging.yourdomain.com (staging API)
* app.yourdomain.com (production frontend)
* app.staging.yourdomain.com (staging frontend)
* admin.yourdomain.com (production admin panel)
* admin.staging.yourdomain.com (staging admin panel)

api.yourdomain.com

api.staging.yourdomain.com

app.yourdomain.com

app.staging.yourdomain.com

admin.yourdomain.com

admin.staging.yourdomain.com

#### 3. Trigger the Initial Deployment

1. Run the Workflow:
2. Go to the "Actions" tab in your GitHub repository
3. Select the "KAI Platform CI/CD Pipeline" workflow
4. Click "Run workflow"
5. Choose the environment (staging or production)
6. Check "Create new cluster if not exists"
7. Click "Run workflow"
8. Monitor the Deployment:
9. Watch the workflow progress in the GitHub Actions tab
10. Check for any errors and fix them if needed
11. The workflow will automatically:  
    Build and test your code  
    Build Docker images  
    Provision Kubernetes infrastructure  
    Deploy the application  
    Deploy the frontend  
    Verify the deployment

Click "Run workflow"

Monitor the Deployment:

* Build and test your code
* Build Docker images
* Provision Kubernetes infrastructure
* Deploy the application
* Deploy the frontend
* Verify the deployment

#### 4. Verify the Deployment

The GitHub Actions workflow includes a comprehensive verification step that checks:

1. API Availability: Verifies that the API is responding correctly
2. Pod Status: Ensures all pods are running and ready
3. Logs: Examines recent logs for errors
4. SSL Certificates: Verifies certificate validity and expiration

You can perform additional verification manually:

# Check pod status  
kubectl get pods -n kai-system  
  
# Check services  
kubectl get services -n kai-system  
  
# Check ingress  
kubectl get ingress -n kai-system  
  
# Check certificates  
kubectl get certificates -n kai-system  
  
# Check API health  
curl https://api.yourdomain.com/health

# Check pod status  
kubectl get pods -n kai-system  
  
# Check services  
kubectl get services -n kai-system  
  
# Check ingress  
kubectl get ingress -n kai-system  
  
# Check certificates  
kubectl get certificates -n kai-system  
  
# Check API health  
curl https://api.yourdomain.com/health

#### Monitoring

For ongoing monitoring, the KAI platform includes:

1. Prometheus Metrics: All services expose Prometheus metrics
2. Grafana Dashboards: Pre-configured dashboards for monitoring system health
3. Liveness and Readiness Probes: All pods have appropriate health checks
4. Logging: Structured JSON logs for easy analysis

To access Grafana (if installed):

# Port forward to Grafana  
kubectl port-forward -n monitoring svc/grafana 3000:80  
  
# Access in browser  
open http://localhost:3000

# Port forward to Grafana  
kubectl port-forward -n monitoring svc/grafana 3000:80  
  
# Access in browser  
open http://localhost:3000

Default login: admin / admin (change on first login)

#### Neural OCR Installation

The Neural OCR integration requires additional dependencies beyond the standard OCR system:

1. Install neural OCR dependencies:  
    bash  
    cd packages/ml  
    pip install -r python/requirements-ocr.txt
2. Verify installation:  
    bash  
    python -c "from neural\_ocr\_orchestrator import NeuralOCROrchestrator; print('Neural OCR available:', NeuralOCROrchestrator.available\_engines())"

Install neural OCR dependencies:  
 bash  
 cd packages/ml  
 pip install -r python/requirements-ocr.txt

bash  
 cd packages/ml  
 pip install -r python/requirements-ocr.txt

Verify installation:  
 bash  
 python -c "from neural\_ocr\_orchestrator import NeuralOCROrchestrator; print('Neural OCR available:', NeuralOCROrchestrator.available\_engines())"

bash  
 python -c "from neural\_ocr\_orchestrator import NeuralOCROrchestrator; print('Neural OCR available:', NeuralOCROrchestrator.available\_engines())"

#### ML Package Installation

The ML package provides machine learning functionality for material recognition, vector embeddings, model training, multimodal pattern recognition, and domain-specific networks:

##### Prerequisites

* CUDA 11.8+ for GPU support
* NVIDIA drivers for L40S/H100 GPUs
* Python 3.8+
* Node.js 16+
* Tesseract OCR (for text extraction)

##### Setup

1. Install Node.js dependencies:  
    bash  
    cd packages/ml  
    npm install
2. Install Python dependencies:  
    bash  
    npm run setup-python

Install Node.js dependencies:  
 bash  
 cd packages/ml  
 npm install

bash  
 cd packages/ml  
 npm install

Install Python dependencies:  
 bash  
 npm run setup-python

bash  
 npm run setup-python

##### MultiModal Pattern Recognition Setup

For the MultiModal Pattern Recognition system:

1. Install additional dependencies:  
    bash  
    pip install transformers torch torchvision einops timm safetensors
2. Download model weights:  
    bash  
    python -c "from huggingface\_hub import snapshot\_download; snapshot\_download('kai/multimodal-pattern-recognition-base')"
3. Verify installation:  
    bash  
    python -c "from multimodal\_pattern\_recognition import MultiModalPatternRecognizer; print('MultiModal Pattern Recognition available:', MultiModalPatternRecognizer.available())"

Install additional dependencies:  
 bash  
 pip install transformers torch torchvision einops timm safetensors

bash  
 pip install transformers torch torchvision einops timm safetensors

Download model weights:  
 bash  
 python -c "from huggingface\_hub import snapshot\_download; snapshot\_download('kai/multimodal-pattern-recognition-base')"

bash  
 python -c "from huggingface\_hub import snapshot\_download; snapshot\_download('kai/multimodal-pattern-recognition-base')"

Verify installation:  
 bash  
 python -c "from multimodal\_pattern\_recognition import MultiModalPatternRecognizer; print('MultiModal Pattern Recognition available:', MultiModalPatternRecognizer.available())"

bash  
 python -c "from multimodal\_pattern\_recognition import MultiModalPatternRecognizer; print('MultiModal Pattern Recognition available:', MultiModalPatternRecognizer.available())"

##### Domain-Specific Networks Setup

For the Domain-Specific Networks system:

1. Install additional dependencies:  
    bash  
    pip install torch torchvision einops timm ml\_collections
2. Download model weights:  
    bash  
    python -c "from huggingface\_hub import snapshot\_download; snapshot\_download('kai/domain-specific-networks')"
3. Verify installation:  
    bash  
    python -c "from domain\_specific\_networks import DomainSpecificNetworkManager; print('Domain-Specific Networks available:', DomainSpecificNetworkManager.list\_available\_domains())"

Install additional dependencies:  
 bash  
 pip install torch torchvision einops timm ml\_collections

bash  
 pip install torch torchvision einops timm ml\_collections

Download model weights:  
 bash  
 python -c "from huggingface\_hub import snapshot\_download; snapshot\_download('kai/domain-specific-networks')"

bash  
 python -c "from huggingface\_hub import snapshot\_download; snapshot\_download('kai/domain-specific-networks')"

Verify installation:  
 bash  
 python -c "from domain\_specific\_networks import DomainSpecificNetworkManager; print('Domain-Specific Networks available:', DomainSpecificNetworkManager.list\_available\_domains())"

bash  
 python -c "from domain\_specific\_networks import DomainSpecificNetworkManager; print('Domain-Specific Networks available:', DomainSpecificNetworkManager.list\_available\_domains())"

#### Notification System Installation

The Notification System provides multi-channel notification capabilities:

##### Prerequisites

* Node.js 16+
* Redis for notification queueing
* SMTP server (for email notifications)
* Twilio or similar provider (for SMS notifications)

##### Installation Steps

1. Install dependencies:  
    bash  
    cd packages/server  
    yarn install
2. Run database migrations:  
    bash  
    cd packages/server  
    node scripts/run-migrations.ts
3. Configure environment variables:  
    ```  
    # Notification Service  
    NOTIFICATION\_SERVICE\_ENABLED=true  
    DEFAULT\_NOTIFICATION\_CHANNEL=in-app

Install dependencies:  
 bash  
 cd packages/server  
 yarn install

bash  
 cd packages/server  
 yarn install

Run database migrations:  
 bash  
 cd packages/server  
 node scripts/run-migrations.ts

bash  
 cd packages/server  
 node scripts/run-migrations.ts

Configure environment variables:  
 ```  
 # Notification Service  
 NOTIFICATION\_SERVICE\_ENABLED=true  
 DEFAULT\_NOTIFICATION\_CHANNEL=in-app

# Email Provider  
 EMAIL\_PROVIDER=sendgrid # sendgrid, mailchimp, or ses  
 EMAIL\_API\_KEY=your\_api\_key

# SMS Provider (optional)  
 SMS\_PROVIDER=twilio # twilio or nexmo  
 SMS\_API\_KEY=your\_api\_key  
 SMS\_ACCOUNT\_SID=your\_sid # twilio only

# Webhook Configuration  
 WEBHOOK\_RETRY\_ATTEMPTS=3  
 WEBHOOK\_TIMEOUT\_MS=5000  
 ```

#### Parameter Registry Installation

The Parameter Registry system manages hyperparameters for material analysis:

##### Prerequisites

* Node.js 16+
* PostgreSQL or Supabase
* Redis for caching

##### Installation Steps

1. Install dependencies:  
    bash  
    cd packages/server  
    yarn install
2. Run database migrations:  
    bash  
    cd packages/server  
    node scripts/run-migrations.ts
3. Configure environment variables:  
    # Parameter Registry  
    PARAM\_REGISTRY\_ENABLED=true  
    PARAM\_STORAGE\_TYPE=supabase # supabase or postgres  
    PARAM\_DB\_CONNECTION=your\_connection\_string  
    PARAM\_HISTORY\_RETENTION\_DAYS=90  
    DEFAULT\_PARAMETER\_SET=standard  
    SIMILARITY\_THRESHOLD=0.75

Install dependencies:  
 bash  
 cd packages/server  
 yarn install

bash  
 cd packages/server  
 yarn install

Run database migrations:  
 bash  
 cd packages/server  
 node scripts/run-migrations.ts

bash  
 cd packages/server  
 node scripts/run-migrations.ts

Configure environment variables:  
 # Parameter Registry  
 PARAM\_REGISTRY\_ENABLED=true  
 PARAM\_STORAGE\_TYPE=supabase # supabase or postgres  
 PARAM\_DB\_CONNECTION=your\_connection\_string  
 PARAM\_HISTORY\_RETENTION\_DAYS=90  
 DEFAULT\_PARAMETER\_SET=standard  
 SIMILARITY\_THRESHOLD=0.75

# Parameter Registry  
 PARAM\_REGISTRY\_ENABLED=true  
 PARAM\_STORAGE\_TYPE=supabase # supabase or postgres  
 PARAM\_DB\_CONNECTION=your\_connection\_string  
 PARAM\_HISTORY\_RETENTION\_DAYS=90  
 DEFAULT\_PARAMETER\_SET=standard  
 SIMILARITY\_THRESHOLD=0.75

#### MCP Server Installation

The Model Context Protocol (MCP) Server centralizes model management and provides optimized inference capabilities:

##### Prerequisites

* Docker (for containerized deployment)
* Python 3.8+ (for local development)
* Node.js 14+ (for client SDK)

##### Environment Variables

The MCP Server can be configured with the following environment variables:

Table content:

Variable | Description | Default

MCP\_SERVER\_PORT | Port for the MCP server | 8000

MODEL\_DIR | Directory for storing model files | /app/models

LOG\_LEVEL | Logging level (DEBUG, INFO, WARNING, ERROR) | INFO

ENABLE\_AGENT\_API | Enable agent communication APIs | true

MCP\_SERVER\_PORT

8000

MODEL\_DIR

/app/models

LOG\_LEVEL

INFO

ENABLE\_AGENT\_API

true

##### Docker Deployment

The MCP Server can be deployed as a Docker container:

# Build the MCP server image  
docker build -t kai-mcp-server -f packages/ml/Dockerfile.mcp .  
  
# Run the MCP server container  
docker run -d \  
 --name kai-mcp-server \  
 -p 8000:8000 \  
 -v $(pwd)/models:/app/models \  
 kai-mcp-server

# Build the MCP server image  
docker build -t kai-mcp-server -f packages/ml/Dockerfile.mcp .  
  
# Run the MCP server container  
docker run -d \  
 --name kai-mcp-server \  
 -p 8000:8000 \  
 -v $(pwd)/models:/app/models \  
 kai-mcp-server

##### Local Development

For local development, you can run the MCP server directly:

# Install required packages  
cd packages/ml  
pip install -r requirements.txt  
pip install fastapi uvicorn python-multipart  
  
# Run the server  
cd packages/ml/python  
uvicorn mcp\_server:app --reload --host 0.0.0.0 --port 8000

# Install required packages  
cd packages/ml  
pip install -r requirements.txt  
pip install fastapi uvicorn python-multipart  
  
# Run the server  
cd packages/ml/python  
uvicorn mcp\_server:app --reload --host 0.0.0.0 --port 8000

#### MCP Client Installation

The MCP Client provides a TypeScript interface for the MCP server:

# From the project root  
cd packages/mcp-client  
yarn install  
yarn build  
yarn link # For local development  
  
# In packages that use the MCP client  
cd ../ml  
yarn link @kai/mcp-client

# From the project root  
cd packages/mcp-client  
yarn install  
yarn build  
yarn link # For local development  
  
# In packages that use the MCP client  
cd ../ml  
yarn link @kai/mcp-client

##### Client Environment Variables

Table content:

Variable | Description | Default

MCP\_SERVER\_URL | URL of the MCP server | http://localhost:8000

USE\_MCP\_SERVER | Enable MCP server integration | false

MCP\_HEALTH\_CHECK\_TIMEOUT | Timeout for health check (ms) | 5000

MCP\_SERVER\_URL

http://localhost:8000

USE\_MCP\_SERVER

false

MCP\_HEALTH\_CHECK\_TIMEOUT

5000

#### CrewAI Integration Installation

The CrewAI integration adds intelligent agent capabilities to the Kai platform:

##### Prerequisites

* Node.js 16+
* Yarn or npm
* OpenAI API key
* KAI platform services running

##### Installation Steps

1. Install dependencies:  
    bash  
    cd packages/agents  
    yarn install
2. Configure environment variables in the root .env file:  
    ```  
    # Required  
    OPENAI\_API\_KEY=your\_openai\_api\_key

Install dependencies:  
 bash  
 cd packages/agents  
 yarn install

bash  
 cd packages/agents  
 yarn install

Configure environment variables in the root .env file:  
 ```  
 # Required  
 OPENAI\_API\_KEY=your\_openai\_api\_key

.env

# KAI Services (change URLs as needed for your environment)  
 KAI\_API\_URL=http://localhost:3000/api  
 KAI\_VECTOR\_DB\_URL=http://localhost:5000/api/vector  
 KAI\_ML\_SERVICE\_URL=http://localhost:7000/api/ml

# Optional  
 OPENAI\_DEFAULT\_MODEL=gpt-4  
 OPENAI\_TEMPERATURE=0.7  
 ENABLE\_MOCK\_FALLBACK=true  
 LOG\_LEVEL=info

# Redis Configuration (for agent state persistence)  
 REDIS\_URL=redis://localhost:6379  
 REDIS\_PASSWORD=  
 ```

1. Verify installation:  
    bash  
    cd packages/agents  
    yarn verify

bash  
 cd packages/agents  
 yarn verify

Or run integration tests:  
 bash  
 yarn test:integration

bash  
 yarn test:integration

#### Hugging Face Integration Installation

The Hugging Face integration with adaptive model selection provides enhanced AI capabilities across multiple providers:

##### Prerequisites

* Node.js 16+
* Yarn or npm
* Hugging Face API key
* Optional: OpenAI and/or Anthropic API keys (for multi-provider capabilities)

##### Installation Steps

1. Install dependencies:  
    bash  
    cd packages/server  
    yarn install
2. Configure environment variables in the root .env file:  
    ```  
    # Required for Hugging Face integration  
    HF\_API\_KEY=your\_huggingface\_api\_key

Install dependencies:  
 bash  
 cd packages/server  
 yarn install

bash  
 cd packages/server  
 yarn install

Configure environment variables in the root .env file:  
 ```  
 # Required for Hugging Face integration  
 HF\_API\_KEY=your\_huggingface\_api\_key

.env

# Optional Hugging Face configuration  
 HF\_ORGANIZATION\_ID=your\_organization\_id  
 HF\_DEFAULT\_TEXT\_MODEL=google/flan-t5-xxl  
 HF\_DEFAULT\_EMBEDDING\_MODEL=sentence-transformers/all-MiniLM-L6-v2  
 HF\_DEFAULT\_IMAGE\_MODEL=google/vit-base-patch16-224  
 HF\_MODEL\_TIMEOUT=30000  
 HF\_USE\_FAST\_MODELS=true

# Optional additional providers  
 OPENAI\_API\_KEY=your\_openai\_api\_key  
 ANTHROPIC\_API\_KEY=your\_anthropic\_api\_key

# Adaptive model selection configuration  
 MODEL\_EVALUATION\_STANDARD\_CYCLE=10  
 MODEL\_EVALUATION\_TEST\_CYCLE=3  
 MODEL\_SELECTION\_METRICS\_WEIGHTS={"accuracy":0.6,"latency":0.2,"cost":0.2}  
 ```

1. Verify installation:  
    bash  
    curl http://localhost:3000/api/ai/models/list

bash  
 curl http://localhost:3000/api/ai/models/list

The response should include available models across all configured providers.

1. Test the adaptive model selection system:  
    bash  
    # Generate text with automatic model selection  
    curl -X POST http://localhost:3000/api/ai/text/generate \  
    -H "Content-Type: application/json" \  
    -d '{"prompt": "Explain the properties of porcelain tiles"}'

bash  
 # Generate text with automatic model selection  
 curl -X POST http://localhost:3000/api/ai/text/generate \  
 -H "Content-Type: application/json" \  
 -d '{"prompt": "Explain the properties of porcelain tiles"}'

### Deployment Options

#### Option 1: Cloud Provider Deployment (Recommended)

1. AWS Deployment
2. API Server: ECS Fargate or Elastic Beanstalk
3. Database: MongoDB Atlas or DocumentDB
4. File Storage: S3
5. ML Services: EC2 with GPU or SageMaker
6. CDN: CloudFront
7. Queue: Supabase hosted on EC2 or ECS
8. Azure Deployment
9. API Server: Azure App Service or AKS
10. Database: Cosmos DB with MongoDB API
11. File Storage: Azure Blob Storage
12. ML Services: Azure VMs with GPU or Azure ML
13. CDN: Azure CDN
14. Queue: Supabase hosted on Azure VMs
15. Google Cloud Deployment
16. API Server: Google Cloud Run or GKE
17. Database: MongoDB Atlas
18. File Storage: Google Cloud Storage
19. ML Services: Google Compute with GPUs
20. CDN: Cloud CDN
21. Queue: Supabase hosted on GCE

Queue: Supabase hosted on EC2 or ECS

Azure Deployment

Queue: Supabase hosted on Azure VMs

Google Cloud Deployment

#### Option 2: Self-Hosted Deployment

Requirements:  
- Kubernetes cluster or Docker Swarm  
- Load balancer (NGINX, HAProxy)  
- MongoDB (replica set recommended)  
- S3-compatible storage (MinIO, Ceph)  
- GPU servers for ML services

Steps:  
1. Set up MongoDB replica set  
2. Deploy S3-compatible storage  
3. Configure container orchestration platform  
4. Deploy API server containers  
5. Deploy ML service containers  
6. Set up load balancer and routing  
7. Configure Supabase for queue system  
8. Set up monitoring and logging

#### Kubernetes Deployment

The KAI ML Platform uses a job-based processing architecture with Argo Workflows for orchestration. This section provides detailed steps for deploying to Digital Ocean Kubernetes (DOKS).

##### Setting up a Kubernetes Cluster

1. Log in to Digital Ocean
2. Navigate to Kubernetes → Create → Kubernetes
3. Configure the cluster:
4. Kubernetes Version: Latest stable version
5. Datacenter Region: Choose the region closest to your users
6. Node Pools:  
     
   Orchestration Pool:  
   Machine Type: Standard  
   Node Plan: 4 GB / 2 vCPU or higher  
   Node Count: 3 (for high availability)  
     
   Labels: node-type=orchestration  
     
     
   CPU-Optimized Pool:  
     
   Machine Type: CPU-Optimized  
   Node Plan: 8 GB / 4 vCPU or higher  
   Node Count: 3  
     
   Labels: node-type=cpu-optimized  
     
     
   GPU-Optimized Pool:  
     
   Machine Type: GPU-Optimized  
   Node Plan: With NVIDIA L40S/H100 GPUs  
   Node Count: 2  
     
   Labels: node-type=gpu-optimized  
     
     
   Memory-Optimized Pool:  
     
   Machine Type: Memory-Optimized  
   Node Plan: 16 GB RAM or higher  
   Node Count: 2  
   Labels: node-type=memory-optimized
7. Enable the NVIDIA GPU Operator (if using GPU nodes)
8. Name your cluster (e.g., kai-ml-cluster)
9. Click "Create Cluster"

Node Pools:

* Orchestration Pool:
* Machine Type: Standard
* Node Plan: 4 GB / 2 vCPU or higher
* Node Count: 3 (for high availability)
* Labels: node-type=orchestration
* CPU-Optimized Pool:
* Machine Type: CPU-Optimized
* Node Plan: 8 GB / 4 vCPU or higher
* Node Count: 3
* Labels: node-type=cpu-optimized
* GPU-Optimized Pool:
* Machine Type: GPU-Optimized
* Node Plan: With NVIDIA L40S/H100 GPUs
* Node Count: 2
* Labels: node-type=gpu-optimized
* Memory-Optimized Pool:
* Machine Type: Memory-Optimized
* Node Plan: 16 GB RAM or higher
* Node Count: 2
* Labels: node-type=memory-optimized

Labels: node-type=orchestration

node-type=orchestration

CPU-Optimized Pool:

Labels: node-type=cpu-optimized

node-type=cpu-optimized

GPU-Optimized Pool:

Labels: node-type=gpu-optimized

node-type=gpu-optimized

Memory-Optimized Pool:

node-type=memory-optimized

Enable the NVIDIA GPU Operator (if using GPU nodes)

kai-ml-cluster

##### Connecting to the Cluster

1. Once the cluster is created, download the kubeconfig file
2. Set up kubectl to use this config:  
    bash  
    export KUBECONFIG=~/Downloads/kai-ml-cluster-kubeconfig.yaml
3. Verify connection:  
    bash  
    kubectl get nodes  
    kubectl get nodes --show-labels

bash  
 export KUBECONFIG=~/Downloads/kai-ml-cluster-kubeconfig.yaml

bash  
 kubectl get nodes  
 kubectl get nodes --show-labels

##### Installing Argo Workflows

Argo Workflows is required for pipeline orchestration:

# Install Argo Workflows controller and UI  
kubectl create namespace argo  
kubectl apply -n argo -f https://github.com/argoproj/argo-workflows/releases/download/v3.4.5/install.yaml  
  
# Configure Argo to work with the kai-ml namespace  
kubectl apply -f kubernetes/argo-rbac.yaml

# Install Argo Workflows controller and UI  
kubectl create namespace argo  
kubectl apply -n argo -f https://github.com/argoproj/argo-workflows/releases/download/v3.4.5/install.yaml  
  
# Configure Argo to work with the kai-ml namespace  
kubectl apply -f kubernetes/argo-rbac.yaml

##### Deploying with GitOps

The KAI ML Platform uses a GitOps approach with Flux for deployments. The CI/CD pipeline updates image tags in the GitOps repository, and Flux automatically applies these changes to the Kubernetes cluster.

The script supports several options:  
- --context=<context>: Kubernetes context to use  
- --registry=<url>: Container registry URL  
- --tag=<tag>: Image tag for all components  
- --dry-run: Validate configurations without applying changes  
- --skip-infrastructure: Skip infrastructure components  
- --skip-coordinator: Skip coordinator service components  
- --skip-workflows: Skip workflow templates

--context=<context>

--registry=<url>

--tag=<tag>

--dry-run

--skip-infrastructure

--skip-coordinator

--skip-workflows

##### Deployment Components

The deployment includes the following main components:

1. Infrastructure:
2. Namespace and resource quotas (kubernetes/infrastructure/namespace.yaml)
3. Priority classes (kubernetes/infrastructure/priority-classes.yaml)
4. Node pools (kubernetes/infrastructure/node-pools.yaml)
5. Monitoring (kubernetes/infrastructure/monitoring.yaml)
6. Caching (kubernetes/infrastructure/caching.yaml)
7. Coordinator Service (kubernetes/coordinator/):
8. Central orchestration component
9. Manages task queues and workflow scheduling
10. Interfaces with Argo Workflows
11. Exposed via service and potentially ingress
12. Distributed Processing (kubernetes/distributed-processing/):
13. Handles distributed workloads
14. Optional component for high-throughput processing
15. Mobile Optimization (kubernetes/mobile-optimization/):
16. Optional component for mobile optimization
17. Includes LOD generation and Draco compression
18. WASM Compiler (kubernetes/wasm-compiler/):
19. Optional component for WebAssembly compilation
20. Workflow Templates (kubernetes/workflows/):
21. Argo workflow templates for ML pipelines
22. 3D reconstruction template (3d-reconstruction-template.yaml)
23. MultiModal Pattern Recognition template (multimodal-pattern-recognition-template.yaml)
24. Domain-Specific Networks template (domain-specific-networks-template.yaml)

kubernetes/infrastructure/namespace.yaml

kubernetes/infrastructure/priority-classes.yaml

kubernetes/infrastructure/node-pools.yaml

kubernetes/infrastructure/monitoring.yaml

Caching (kubernetes/infrastructure/caching.yaml)

kubernetes/infrastructure/caching.yaml

Coordinator Service (kubernetes/coordinator/):

kubernetes/coordinator/

Exposed via service and potentially ingress

Distributed Processing (kubernetes/distributed-processing/):

kubernetes/distributed-processing/

Optional component for high-throughput processing

Mobile Optimization (kubernetes/mobile-optimization/):

kubernetes/mobile-optimization/

Includes LOD generation and Draco compression

WASM Compiler (kubernetes/wasm-compiler/):

kubernetes/wasm-compiler/

Optional component for WebAssembly compilation

Workflow Templates (kubernetes/workflows/):

kubernetes/workflows/

3d-reconstruction-template.yaml

multimodal-pattern-recognition-template.yaml

domain-specific-networks-template.yaml

##### GitOps-Based Deployment

The deployment process now uses a GitOps approach with Flux CD:

1. Environment-Specific Configuration:
2. Environment-specific configuration is stored in the GitOps repository
3. Different environments (staging, production) have separate directories in the GitOps repository
4. Each environment has its own set of Kubernetes manifests and Helm releases
5. Automatic Deployment:
6. The CI/CD pipeline updates image tags in the GitOps repository
7. Flux automatically detects changes and applies them to the cluster
8. No manual deployment steps are required
9. Rollback Capability:
10. Flux provides built-in rollback capabilities
11. Rollbacks can be performed by reverting changes in the GitOps repository
12. The history of all deployments is tracked in Git
13. Environment-Specific Directories:
14. Checks for environment-specific configuration files first:  
     kubernetes/  
     ├── coordinator/  
     │ ├── staging/ # Staging-specific configs  
     │ ├── production/ # Production-specific configs  
     │ └── \*.yaml # Default configs used if env-specific not found  
     ├── infrastructure/  
     │ ├── staging/  
     │ └── production/  
     └── workflows/  
     ├── staging/  
     └── production/
15. Falls back to default configurations when environment-specific ones don't exist
16. Deployment Health Verification:
17. Verifies that pods reach Running state
18. Checks service availability
19. Provides detailed deployment status information

Each environment has its own set of Kubernetes manifests and Helm releases

Automatic Deployment:

No manual deployment steps are required

Rollback Capability:

The history of all deployments is tracked in Git

Environment-Specific Directories:

kubernetes/  
 ├── coordinator/  
 │ ├── staging/ # Staging-specific configs  
 │ ├── production/ # Production-specific configs  
 │ └── \*.yaml # Default configs used if env-specific not found  
 ├── infrastructure/  
 │ ├── staging/  
 │ └── production/  
 └── workflows/  
 ├── staging/  
 └── production/

Falls back to default configurations when environment-specific ones don't exist

Deployment Health Verification:

##### Helm-Based Deployment

In addition to the script-based deployment, the KAI Platform now supports Helm charts for more maintainable and consistent Kubernetes deployments. This approach provides significant advantages in configuration management, environment isolation, and deployment reliability.

###### Helm Chart Structure

The platform uses a modular Helm chart structure with parent-child relationships:

helm-charts/  
├── kai/ # Main parent chart  
│ ├── Chart.yaml # Chart metadata with dependencies  
│ ├── values.yaml # Default values  
│ ├── values-staging.yaml # Staging environment values  
│ └── values-production.yaml # Production environment values  
└── coordinator/ # Sample subchart  
 ├── Chart.yaml  
 ├── values.yaml  
 └── templates/ # Resource templates  
 ├── \_helpers.tpl # Reusable template snippets  
 ├── deployment.yaml # Deployment template  
 ├── service.yaml # Service template  
 ├── hpa.yaml # Autoscaling template  
 ├── pdb.yaml # Pod Disruption Budget template  
 ├── rbac.yaml # RBAC resources template  
 └── configmap.yaml # ConfigMap template

helm-charts/  
├── kai/ # Main parent chart  
│ ├── Chart.yaml # Chart metadata with dependencies  
│ ├── values.yaml # Default values  
│ ├── values-staging.yaml # Staging environment values  
│ └── values-production.yaml # Production environment values  
└── coordinator/ # Sample subchart  
 ├── Chart.yaml  
 ├── values.yaml  
 └── templates/ # Resource templates  
 ├── \_helpers.tpl # Reusable template snippets  
 ├── deployment.yaml # Deployment template  
 ├── service.yaml # Service template  
 ├── hpa.yaml # Autoscaling template  
 ├── pdb.yaml # Pod Disruption Budget template  
 ├── rbac.yaml # RBAC resources template  
 └── configmap.yaml # ConfigMap template

###### Helm Charts with Flux

The deployment uses Helm charts managed by Flux through HelmRelease resources:

# Example HelmRelease in the GitOps repository  
apiVersion: helm.toolkit.fluxcd.io/v2beta1  
kind: HelmRelease  
metadata:  
 name: kai-api  
 namespace: flux-system  
spec:  
 interval: 5m  
 chart:  
 spec:  
 chart: ./helm-charts/kai-api  
 sourceRef:  
 kind: GitRepository  
 name: kai-platform  
 namespace: flux-system  
 values:  
 image:  
 repository: registry.example.com/kai/api  
 tag: v1.2.3  
 environment: production  
 replicas: 3

# Example HelmRelease in the GitOps repository  
apiVersion: helm.toolkit.fluxcd.io/v2beta1  
kind: HelmRelease  
metadata:  
 name: kai-api  
 namespace: flux-system  
spec:  
 interval: 5m  
 chart:  
 spec:  
 chart: ./helm-charts/kai-api  
 sourceRef:  
 kind: GitRepository  
 name: kai-platform  
 namespace: flux-system  
 values:  
 image:  
 repository: registry.example.com/kai/api  
 tag: v1.2.3  
 environment: production  
 replicas: 3

###### Resource Allocation by Environment

Resource allocation is automatically adjusted based on the target environment, whether using the script-based deployment or Helm charts:

Table content:

Resource | Staging | Production

API Server Replicas | 1 | 3

Coordinator Replicas | 1 | 3

Min HPA Replicas | 1 | 2

Max HPA Replicas | 5 | 10

PDB Min Available | "1" | "2"

Resource Multiplier | 1x | 2x

Namespace | kai-system-staging | kai-system

##### Flux GitOps-Based Deployment

The KAI Platform now supports a GitOps approach to deployment using Flux CD, which provides a fully automated, declarative way to manage Kubernetes resources.

###### Installing Flux on the Cluster

1. Install the Flux CLI:  
    ```bash  
    # On macOS with Homebrew  
    brew install fluxcd/tap/flux

# On Linux  
 curl -s https://fluxcd.io/install.sh | sudo bash  
 ```

1. Check Kubernetes cluster compatibility:  
    bash  
    flux check --pre
2. Bootstrap Flux on your cluster:  
    ```bash  
    # Generate a GitHub personal access token with 'repo' permissions  
    export GITHUB\_TOKEN=

Check Kubernetes cluster compatibility:  
 bash  
 flux check --pre

bash  
 flux check --pre

Bootstrap Flux on your cluster:  
 ```bash  
 # Generate a GitHub personal access token with 'repo' permissions  
 export GITHUB\_TOKEN=

# Bootstrap Flux on the staging cluster  
 flux bootstrap github \  
 --owner=kai-platform \  
 --repository=kai-gitops \  
 --branch=main \  
 --path=clusters/staging \  
 --personal \  
 --kubeconfig=$HOME/.kube/kai-staging-config

# Bootstrap Flux on the production cluster  
 flux bootstrap github \  
 --owner=kai-platform \  
 --repository=kai-gitops \  
 --branch=main \  
 --path=clusters/production \  
 --personal \  
 --kubeconfig=$HOME/.kube/kai-production-config  
 ```

###### GitOps Repository Structure

The KAI Platform uses a structured GitOps repository with separate configurations for staging and production environments:

flux/  
├── clusters/  
│ ├── staging/ # Staging environment  
│ │ ├── flux-system/ # Flux core components  
│ │ ├── sources/ # Source definitions (Helm repos, Git repos)  
│ │ ├── releases/ # Application deployments  
│ │ └── kustomization.yaml  
│ └── production/ # Production environment (similar structure)

flux/  
├── clusters/  
│ ├── staging/ # Staging environment  
│ │ ├── flux-system/ # Flux core components  
│ │ ├── sources/ # Source definitions (Helm repos, Git repos)  
│ │ ├── releases/ # Application deployments  
│ │ └── kustomization.yaml  
│ └── production/ # Production environment (similar structure)

###### Creating HelmRelease Resources

Example HelmRelease resource:

# Example: releases/coordinator.yaml  
apiVersion: helm.toolkit.fluxcd.io/v2beta1  
kind: HelmRelease  
metadata:  
 name: coordinator  
 namespace: flux-system  
spec:  
 interval: 5m  
 chart:  
 spec:  
 chart: coordinator  
 version: ">=1.0.0"  
 sourceRef:  
 kind: HelmRepository  
 name: kai-charts  
 namespace: flux-system  
 interval: 1m  
 values:  
 replicaCount: 3  
 image:  
 repository: "registry.example.com/coordinator"  
 tag: "v1.2.3"  
 install:  
 remediation:  
 retries: 3  
 upgrade:  
 remediation:  
 remediateLastFailure: true  
 cleanupOnFail: true  
 rollback:  
 timeout: 5m  
 cleanupOnFail: true  
 targetNamespace: kai-system  
 releaseName: coordinator

# Example: releases/coordinator.yaml  
apiVersion: helm.toolkit.fluxcd.io/v2beta1  
kind: HelmRelease  
metadata:  
 name: coordinator  
 namespace: flux-system  
spec:  
 interval: 5m  
 chart:  
 spec:  
 chart: coordinator  
 version: ">=1.0.0"  
 sourceRef:  
 kind: HelmRepository  
 name: kai-charts  
 namespace: flux-system  
 interval: 1m  
 values:  
 replicaCount: 3  
 image:  
 repository: "registry.example.com/coordinator"  
 tag: "v1.2.3"  
 install:  
 remediation:  
 retries: 3  
 upgrade:  
 remediation:  
 remediateLastFailure: true  
 cleanupOnFail: true  
 rollback:  
 timeout: 5m  
 cleanupOnFail: true  
 targetNamespace: kai-system  
 releaseName: coordinator

###### CI/CD Integration with Flux

The CI/CD pipeline integrates with Flux through a dedicated job that updates the GitOps repository with new image versions after building and testing.

#### Docker Configuration and Optimization

The Kai platform uses optimized Docker configurations to improve build speed, reduce image size, and enhance security.

##### Optimized Dockerfiles

###### API Server (Dockerfile.api)

Key optimizations:  
- Updated from Node.js 16 to Node.js 20 (Alpine variant)  
- Multi-stage build to reduce final image size  
- Dependency caching for faster builds  
- BuildKit features for optimal caching  
- Non-root user execution for security  
- Health checks and proper signal handling

###### Centralized Base Images

The platform uses centralized base images to ensure consistency:

1. ML Base Image (Dockerfile.ml-base)
2. TensorFlow GPU-enabled base image
3. Common ML dependencies
4. Standard user setup and permissions
5. Node.js Base Image (Dockerfile.node-base)
6. Alpine-based Node.js image
7. Common Node.js dependencies
8. Standard security configuration
9. Python Base Image (Dockerfile.python-base)
10. Python slim image for non-GPU services
11. Common Python dependencies
12. Consistent environment setup

Standard user setup and permissions

Node.js Base Image (Dockerfile.node-base)

Standard security configuration

Python Base Image (Dockerfile.python-base)

##### Build Context Optimization

For optimal Docker build performance, the .dockerignore file excludes:  
- Version control directories  
- Development and build files (node\_modules, pycache)  
- Test and documentation directories  
- Editor/IDE configuration files

.dockerignore

##### Best Practices Implemented

1. Layer Efficiency
2. Fewer, more purposeful layers
3. Logical grouping of commands to optimize caching
4. RUN commands combined with && where appropriate
5. Cache Utilization
6. Dependencies installed separately from application code
7. Package files copied first to leverage cache for dependencies
8. BuildKit cache mounts used for node\_modules and pip caches
9. Image Size Reduction
10. Multi-stage builds
11. Alpine-based images where appropriate
12. Cleanup of temporary files and package caches
13. Only production dependencies in final image
14. Security
15. Non-root user execution
16. Explicit EXPOSE statements for ports
17. Fixed dependency versions
18. Regular base image updates
19. Process Supervision
20. Proper signal handling
21. Health checks integrated
22. Explicit entrypoints and commands

RUN commands combined with && where appropriate

&&

Cache Utilization

BuildKit cache mounts used for node\_modules and pip caches

Image Size Reduction

Only production dependencies in final image

Security

Regular base image updates

Process Supervision

### Deployment Process

#### Environment Configuration

Create a single .env file in the root directory for production:

.env

Root .env file (.env.production)

# Node Environment  
NODE\_ENV=production  
  
# Server Configuration  
PORT=3000  
API\_BASE\_URL=https://api.yourdomain.com  
CORS\_ORIGIN=https://yourdomain.com  
LOG\_LEVEL=info  
  
# Database Configuration  
MONGODB\_URI=mongodb+srv://user:password@cluster.mongodb.net/kai  
  
# Authentication  
JWT\_SECRET=your-very-secure-jwt-secret  
  
# Storage Configuration  
S3\_BUCKET=kai-production  
S3\_REGION=us-east-1  
S3\_ACCESS\_KEY=your-access-key  
S3\_SECRET\_KEY=your-secret-key  
  
# Supabase Configuration  
SUPABASE\_URL=https://your-supabase-project.supabase.co  
SUPABASE\_KEY=your-supabase-key  
SUPABASE\_STORAGE\_BUCKET=materials  
  
# ML Configuration  
MODEL\_PATH=/opt/kai/models  
TENSORFLOW\_SERVING\_URL=http://tensorflow-serving:8501  
VECTOR\_INDEX\_PATH=/opt/kai/indexes  
GPU\_ENABLED=true  
BATCH\_SIZE=8  
  
# MCP Server Configuration  
MCP\_SERVER\_URL=http://mcp-server:8000  
USE\_MCP\_SERVER=true  
MODEL\_CACHE\_SIZE=5  
AGENT\_INTEGRATION\_ENABLED=true  
MAX\_BATCH\_SIZE=16  
  
# Rate Limiting Configuration  
DEFAULT\_RATE\_LIMIT=100  
DEFAULT\_RATE\_WINDOW\_MS=60000  
AUTH\_RATE\_LIMIT=20  
AUTH\_RATE\_WINDOW\_MS=60000  
ML\_RATE\_LIMIT=10  
ML\_RATE\_WINDOW\_MS=60000  
AGENT\_RATE\_LIMIT=30  
AGENT\_RATE\_WINDOW\_MS=60000  
PDF\_RATE\_LIMIT=5  
PDF\_RATE\_WINDOW\_MS=600000  
  
# Frontend Configuration  
GATSBY\_API\_URL=https://api.yourdomain.com  
GATSBY\_SUPABASE\_URL=https://your-supabase-project.supabase.co  
GATSBY\_SUPABASE\_ANON\_KEY=your-supabase-anon-key  
GATSBY\_STORAGE\_URL=https://your-cdn.com  
GATSBY\_DEFAULT\_LOCALE=en  
GATSBY\_GOOGLE\_ANALYTICS\_ID=your-ga-id

# Node Environment  
NODE\_ENV=production  
  
# Server Configuration  
PORT=3000  
API\_BASE\_URL=https://api.yourdomain.com  
CORS\_ORIGIN=https://yourdomain.com  
LOG\_LEVEL=info  
  
# Database Configuration  
MONGODB\_URI=mongodb+srv://user:password@cluster.mongodb.net/kai  
  
# Authentication  
JWT\_SECRET=your-very-secure-jwt-secret  
  
# Storage Configuration  
S3\_BUCKET=kai-production  
S3\_REGION=us-east-1  
S3\_ACCESS\_KEY=your-access-key  
S3\_SECRET\_KEY=your-secret-key  
  
# Supabase Configuration  
SUPABASE\_URL=https://your-supabase-project.supabase.co  
SUPABASE\_KEY=your-supabase-key  
SUPABASE\_STORAGE\_BUCKET=materials  
  
# ML Configuration  
MODEL\_PATH=/opt/kai/models  
TENSORFLOW\_SERVING\_URL=http://tensorflow-serving:8501  
VECTOR\_INDEX\_PATH=/opt/kai/indexes  
GPU\_ENABLED=true  
BATCH\_SIZE=8  
  
# MCP Server Configuration  
MCP\_SERVER\_URL=http://mcp-server:8000  
USE\_MCP\_SERVER=true  
MODEL\_CACHE\_SIZE=5  
AGENT\_INTEGRATION\_ENABLED=true  
MAX\_BATCH\_SIZE=16  
  
# Rate Limiting Configuration  
DEFAULT\_RATE\_LIMIT=100  
DEFAULT\_RATE\_WINDOW\_MS=60000  
AUTH\_RATE\_LIMIT=20  
AUTH\_RATE\_WINDOW\_MS=60000  
ML\_RATE\_LIMIT=10  
ML\_RATE\_WINDOW\_MS=60000  
AGENT\_RATE\_LIMIT=30  
AGENT\_RATE\_WINDOW\_MS=60000  
PDF\_RATE\_LIMIT=5  
PDF\_RATE\_WINDOW\_MS=600000  
  
# Frontend Configuration  
GATSBY\_API\_URL=https://api.yourdomain.com  
GATSBY\_SUPABASE\_URL=https://your-supabase-project.supabase.co  
GATSBY\_SUPABASE\_ANON\_KEY=your-supabase-anon-key  
GATSBY\_STORAGE\_URL=https://your-cdn.com  
GATSBY\_DEFAULT\_LOCALE=en  
GATSBY\_GOOGLE\_ANALYTICS\_ID=your-ga-id

New Features Environment Variables:

# Notification System  
NOTIFICATION\_SERVICE\_ENABLED=true  
DEFAULT\_NOTIFICATION\_CHANNEL=in-app  
EMAIL\_PROVIDER=sendgrid|mailchimp|ses  
EMAIL\_API\_KEY=${SECRET\_REF}  
SMS\_PROVIDER=twilio|nexmo  
SMS\_API\_KEY=${SECRET\_REF}  
WEBHOOK\_RETRY\_ATTEMPTS=3  
WEBHOOK\_TIMEOUT\_MS=5000  
  
# Parameter Registry  
PARAM\_REGISTRY\_ENABLED=true  
PARAM\_STORAGE\_TYPE=supabase|postgres  
PARAM\_DB\_CONNECTION=${SECRET\_REF}  
PARAM\_HISTORY\_RETENTION\_DAYS=90  
DEFAULT\_PARAMETER\_SET=standard  
SIMILARITY\_THRESHOLD=0.75  
  
# ML Features  
GPU\_SCALING\_ENABLED=true  
MIN\_GPU\_CLASS=L40S  
MULTIMODAL\_MODEL\_VERSION=v2.1  
DOMAIN\_NETWORKS\_ENABLED=true  
DEFAULT\_QUALITY\_TIER=standard  
BATCH\_SIZE\_LIMIT=8

# Notification System  
NOTIFICATION\_SERVICE\_ENABLED=true  
DEFAULT\_NOTIFICATION\_CHANNEL=in-app  
EMAIL\_PROVIDER=sendgrid|mailchimp|ses  
EMAIL\_API\_KEY=${SECRET\_REF}  
SMS\_PROVIDER=twilio|nexmo  
SMS\_API\_KEY=${SECRET\_REF}  
WEBHOOK\_RETRY\_ATTEMPTS=3  
WEBHOOK\_TIMEOUT\_MS=5000  
  
# Parameter Registry  
PARAM\_REGISTRY\_ENABLED=true  
PARAM\_STORAGE\_TYPE=supabase|postgres  
PARAM\_DB\_CONNECTION=${SECRET\_REF}  
PARAM\_HISTORY\_RETENTION\_DAYS=90  
DEFAULT\_PARAMETER\_SET=standard  
SIMILARITY\_THRESHOLD=0.75  
  
# ML Features  
GPU\_SCALING\_ENABLED=true  
MIN\_GPU\_CLASS=L40S  
MULTIMODAL\_MODEL\_VERSION=v2.1  
DOMAIN\_NETWORKS\_ENABLED=true  
DEFAULT\_QUALITY\_TIER=standard  
BATCH\_SIZE\_LIMIT=8

For monitoring in production, add:

# Logging Configuration  
LOG\_LEVEL=info  
LOG\_FORMAT=json  
LOG\_TO\_FILE=true  
LOG\_FILE\_PATH=/var/log/kai/server.log  
LOG\_ROTATION\_INTERVAL=1d  
LOG\_MAX\_FILES=30  
  
# Monitoring Configuration  
ENABLE\_DETAILED\_METRICS=true  
METRICS\_REPORTING\_INTERVAL=60000  
ENABLE\_PERFORMANCE\_MONITORING=true  
  
# Health Check Configuration  
HEALTH\_CHECK\_INTERVAL=30000  
COMPONENT\_TIMEOUT\_MS=5000

# Logging Configuration  
LOG\_LEVEL=info  
LOG\_FORMAT=json  
LOG\_TO\_FILE=true  
LOG\_FILE\_PATH=/var/log/kai/server.log  
LOG\_ROTATION\_INTERVAL=1d  
LOG\_MAX\_FILES=30  
  
# Monitoring Configuration  
ENABLE\_DETAILED\_METRICS=true  
METRICS\_REPORTING\_INTERVAL=60000  
ENABLE\_PERFORMANCE\_MONITORING=true  
  
# Health Check Configuration  
HEALTH\_CHECK\_INTERVAL=30000  
COMPONENT\_TIMEOUT\_MS=5000

#### Database Setup and Migration

##### Database Setup

1. Create MongoDB database with collections:  
    materials  
    collections  
    versions  
    relationships  
    search\_indexes  
    pdf\_jobs  
    crawler\_jobs  
    users
2. Create indexes for performance:  
    ```javascript  
    // Materials Collection Indexes  
    db.materials.createIndex({ "id": 1 }, { unique: true })  
    db.materials.createIndex({ "name": 1 })  
    db.materials.createIndex({ "materialType": 1 })  
    db.materials.createIndex({ "collectionId": 1 })  
    db.materials.createIndex({ "tags": 1 })  
    db.materials.createIndex({ "$\*\*": "text" })

Create MongoDB database with collections:  
 materials  
 collections  
 versions  
 relationships  
 search\_indexes  
 pdf\_jobs  
 crawler\_jobs  
 users

materials  
 collections  
 versions  
 relationships  
 search\_indexes  
 pdf\_jobs  
 crawler\_jobs  
 users

Create indexes for performance:  
 ```javascript  
 // Materials Collection Indexes  
 db.materials.createIndex({ "id": 1 }, { unique: true })  
 db.materials.createIndex({ "name": 1 })  
 db.materials.createIndex({ "materialType": 1 })  
 db.materials.createIndex({ "collectionId": 1 })  
 db.materials.createIndex({ "tags": 1 })  
 db.materials.createIndex({ "$\*\*": "text" })

// Versions Collection Indexes  
 db.versions.createIndex({ "entityId": 1, "entityType": 1 })  
 db.versions.createIndex({ "createdAt": -1 })

// Queue Collections Indexes  
 db.pdf\_jobs.createIndex({ "status": 1, "priority": -1, "createdAt": 1 })  
 db.crawler\_jobs.createIndex({ "status": 1, "priority": -1, "createdAt": 1 })  
 ```

##### Database Migration System

Kai uses a robust database migration system to manage schema changes across environments:

###### Migration Architecture

┌─────────────────────┐ ┌─────────────────────┐ ┌─────────────────────┐  
│ │ │ │ │ │  
│ CI/CD Pipeline │────▶│ Migration Script │────▶│ Supabase Database │  
│ │ │ │ │ │  
└─────────────────────┘ └─────────────────────┘ └─────────────────────┘  
 │ │ │  
 │ │ │  
 ▼ ▼ ▼  
┌─────────────────────┐ ┌─────────────────────┐ ┌─────────────────────┐  
│ │ │ │ │ │  
│ Runs before │ │ Reads SQL files │ │ schema\_migrations │  
│ Application Deploy │ │ from migrations/ │ │ tracking table │  
│ │ │ │ │ │  
└─────────────────────┘ └─────────────────────┘ └─────────────────────┘

┌─────────────────────┐ ┌─────────────────────┐ ┌─────────────────────┐  
│ │ │ │ │ │  
│ CI/CD Pipeline │────▶│ Migration Script │────▶│ Supabase Database │  
│ │ │ │ │ │  
└─────────────────────┘ └─────────────────────┘ └─────────────────────┘  
 │ │ │  
 │ │ │  
 ▼ ▼ ▼  
┌─────────────────────┐ ┌─────────────────────┐ ┌─────────────────────┐  
│ │ │ │ │ │  
│ Runs before │ │ Reads SQL files │ │ schema\_migrations │  
│ Application Deploy │ │ from migrations/ │ │ tracking table │  
│ │ │ │ │ │  
└─────────────────────┘ └─────────────────────┘ └─────────────────────┘

###### Migration Files

Migration files are SQL scripts stored in packages/server/src/services/supabase/migrations/ and follow a sequential naming convention:

packages/server/src/services/supabase/migrations/

001\_initial\_schema.sql  
002\_hybrid\_search.sql  
003\_dataset\_upload.sql  
004\_material\_metadata\_fields.sql  
005\_message\_broker.sql  
006\_enhanced\_vector\_storage.sql  
007\_agent\_sessions.sql  
008\_validation\_rules.sql  
009\_execute\_sql\_function.sql  
...

001\_initial\_schema.sql  
002\_hybrid\_search.sql  
003\_dataset\_upload.sql  
004\_material\_metadata\_fields.sql  
005\_message\_broker.sql  
006\_enhanced\_vector\_storage.sql  
007\_agent\_sessions.sql  
008\_validation\_rules.sql  
009\_execute\_sql\_function.sql  
...

###### Migration Tracking

Migrations are tracked in a migrations table in the Supabase database. Each migration is only applied once, and the system keeps track of which migrations have been applied to ensure database consistency.

migrations

CREATE TABLE schema\_migrations (  
 id SERIAL PRIMARY KEY,  
 name TEXT NOT NULL UNIQUE,  
 applied\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);

CREATE TABLE schema\_migrations (  
 id SERIAL PRIMARY KEY,  
 name TEXT NOT NULL UNIQUE,  
 applied\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);

This table records which migrations have already been applied to prevent duplicate execution.

###### Migration Script

The system uses a TypeScript migration script (packages/server/scripts/run-migrations.ts) that:

packages/server/scripts/run-migrations.ts

1. Connects to the Supabase database using environment credentials
2. Creates the schema\_migrations table if it doesn't exist
3. Retrieves already applied migrations from the table
4. Compares available migrations with applied ones to identify pending migrations
5. Applies pending migrations in sequential order
6. Records successful migrations in the tracking table

schema\_migrations

###### Integration with CI/CD Pipeline

Database migrations are integrated into the CI/CD pipeline to ensure they run automatically before deploying application changes:

1. In the GitHub Actions workflow (.github/workflows/deploy.yml), a dedicated step runs migrations before the Kubernetes deployment:

.github/workflows/deploy.yml

# Run database migrations before deployment  
- name: Setup Node.js for migrations  
 uses: actions/setup-node@v3  
 with:  
 node-version: ${{ env.NODE\_VERSION }}  
  
- name: Install dependencies  
 run: yarn install --frozen-lockfile  
  
- name: Run database migrations (Staging)  
 run: |  
 echo "Running database migrations for staging environment..."  
 yarn tsc -p packages/server/tsconfig.json  
 cd packages/server  
 node dist/scripts/run-migrations.js  
 env:  
 SUPABASE\_URL: ${{ secrets.SUPABASE\_URL\_STAGING }}  
 SUPABASE\_KEY: ${{ secrets.SUPABASE\_KEY\_STAGING }}  
 NODE\_ENV: staging

# Run database migrations before deployment  
- name: Setup Node.js for migrations  
 uses: actions/setup-node@v3  
 with:  
 node-version: ${{ env.NODE\_VERSION }}  
  
- name: Install dependencies  
 run: yarn install --frozen-lockfile  
  
- name: Run database migrations (Staging)  
 run: |  
 echo "Running database migrations for staging environment..."  
 yarn tsc -p packages/server/tsconfig.json  
 cd packages/server  
 node dist/scripts/run-migrations.js  
 env:  
 SUPABASE\_URL: ${{ secrets.SUPABASE\_URL\_STAGING }}  
 SUPABASE\_KEY: ${{ secrets.SUPABASE\_KEY\_STAGING }}  
 NODE\_ENV: staging

#### Adding Tables for New Features

For the notification and webhook system, run these additional migrations:

-- Run notification tables migration  
\i notification-tables.sql  
  
-- Run webhook tables migration  
\i webhooks.sql  
  
-- Run push notifications migration  
\i push-notifications.sql

-- Run notification tables migration  
\i notification-tables.sql  
  
-- Run webhook tables migration  
\i webhooks.sql  
  
-- Run push notifications migration  
\i push-notifications.sql

For the parameter registry system:

-- Run parameter registry migration  
\i parameter-registry.sql

-- Run parameter registry migration  
\i parameter-registry.sql

#### Build Process

Backend Services

# Build API server  
cd packages/server  
yarn build  
  
# Build ML services TypeScript wrapper  
cd packages/ml  
yarn build

# Build API server  
cd packages/server  
yarn build  
  
# Build ML services TypeScript wrapper  
cd packages/ml  
yarn build

Frontend Applications

# Build client app  
cd packages/client  
yarn build  
  
# Build admin app  
cd packages/admin  
yarn build

# Build client app  
cd packages/client  
yarn build  
  
# Build admin app  
cd packages/admin  
yarn build

#### Containerization

Dockerfile for API Server

FROM node:16-alpine  
  
WORKDIR /app  
  
COPY packages/shared/dist ./packages/shared/dist  
COPY packages/shared/package.json ./packages/shared/  
COPY packages/server/dist ./packages/server/dist  
COPY packages/server/package.json ./packages/server/  
COPY package.json yarn.lock ./  
  
ENV NODE\_ENV=production  
  
RUN yarn install --production --frozen-lockfile  
  
EXPOSE 3000  
  
CMD ["node", "packages/server/dist/server.js"]

FROM node:16-alpine  
  
WORKDIR /app  
  
COPY packages/shared/dist ./packages/shared/dist  
COPY packages/shared/package.json ./packages/shared/  
COPY packages/server/dist ./packages/server/dist  
COPY packages/server/package.json ./packages/server/  
COPY package.json yarn.lock ./  
  
ENV NODE\_ENV=production  
  
RUN yarn install --production --frozen-lockfile  
  
EXPOSE 3000  
  
CMD ["node", "packages/server/dist/server.js"]

Dockerfile for ML Services

FROM tensorflow/tensorflow:2.9.1-gpu  
  
WORKDIR /app  
  
COPY packages/ml/python /app/python  
COPY packages/ml/dist /app/dist  
COPY packages/ml/package.json /app/  
  
RUN apt-get update && apt-get install -y \  
 build-essential \  
 python3-pip \  
 python3-dev \  
 && rm -rf /var/lib/apt/lists/\*  
  
RUN pip3 install --no-cache-dir -r /app/python/requirements.txt  
  
EXPOSE 5000  
  
CMD ["python3", "/app/python/server.py"]

FROM tensorflow/tensorflow:2.9.1-gpu  
  
WORKDIR /app  
  
COPY packages/ml/python /app/python  
COPY packages/ml/dist /app/dist  
COPY packages/ml/package.json /app/  
  
RUN apt-get update && apt-get install -y \  
 build-essential \  
 python3-pip \  
 python3-dev \  
 && rm -rf /var/lib/apt/lists/\*  
  
RUN pip3 install --no-cache-dir -r /app/python/requirements.txt  
  
EXPOSE 5000  
  
CMD ["python3", "/app/python/server.py"]

Dockerfile for MCP Server

FROM tensorflow/tensorflow:2.9.1-gpu  
  
WORKDIR /app  
  
COPY packages/ml/python/mcp\_server.py /app/  
COPY packages/ml/python/requirements.txt /app/  
  
RUN apt-get update && apt-get install -y \  
 build-essential \  
 python3-pip \  
 python3-dev \  
 && rm -rf /var/lib/apt/lists/\*  
  
RUN pip3 install --no-cache-dir -r /app/requirements.txt  
RUN pip3 install --no-cache-dir fastapi uvicorn python-multipart  
  
EXPOSE 8000  
  
CMD ["uvicorn", "mcp\_server:app", "--host", "0.0.0.0", "--port", "8000"]

FROM tensorflow/tensorflow:2.9.1-gpu  
  
WORKDIR /app  
  
COPY packages/ml/python/mcp\_server.py /app/  
COPY packages/ml/python/requirements.txt /app/  
  
RUN apt-get update && apt-get install -y \  
 build-essential \  
 python3-pip \  
 python3-dev \  
 && rm -rf /var/lib/apt/lists/\*  
  
RUN pip3 install --no-cache-dir -r /app/requirements.txt  
RUN pip3 install --no-cache-dir fastapi uvicorn python-multipart  
  
EXPOSE 8000  
  
CMD ["uvicorn", "mcp\_server:app", "--host", "0.0.0.0", "--port", "8000"]

Build and push to container registry:

# Build the base images first  
docker build -t registry.example.com/kai/node-base:latest -f Dockerfile.node-base .  
docker build -t registry.example.com/kai/python-base:latest -f Dockerfile.python-base .  
docker build -t registry.example.com/kai/ml-base:latest -f Dockerfile.ml-base .  
  
# Build service images  
docker build -t registry.example.com/kai/api-server:latest -f Dockerfile.api .  
docker build -t registry.example.com/kai/coordinator-service:latest -f packages/coordinator/Dockerfile.coordinator .  
docker build -t registry.example.com/kai/mcp-server:latest -f packages/ml/Dockerfile.mcp .  
  
# Build ML worker images  
docker build -t registry.example.com/kai/quality-assessment:latest -f packages/ml/python/Dockerfile.quality-assessment .  
docker build -t registry.example.com/kai/image-preprocessing:latest -f packages/ml/python/Dockerfile.image-preprocessing .  
docker build -t registry.example.com/kai/colmap-sfm:latest -f packages/ml/python/Dockerfile.colmap-sfm .  
docker build -t registry.example.com/kai/point-cloud:latest -f packages/ml/python/Dockerfile.point-cloud .  
docker build -t registry.example.com/kai/model-generator:latest -f packages/ml/python/Dockerfile.model-generator .  
docker build -t registry.example.com/kai/diffusion-nerf:latest -f packages/ml/python/Dockerfile.diffusion-nerf .  
docker build -t registry.example.com/kai/nerf-mesh-extractor:latest -f packages/ml/python/Dockerfile.nerf-mesh-extractor .  
docker build -t registry.example.com/kai/format-converter:latest -f packages/ml/python/Dockerfile.format-converter .  
docker build -t registry.example.com/kai/workflow-finalizer:latest -f packages/ml/python/Dockerfile.workflow-finalizer .  
  
# Build new feature images  
docker build -t registry.example.com/kai/parameter-registry:latest -f packages/server/Dockerfile.parameter-registry .  
docker build -t registry.example.com/kai/notification-service:latest -f packages/server/Dockerfile.notification-service .  
docker build -t registry.example.com/kai/webhook-service:latest -f packages/server/Dockerfile.webhook-service .  
docker build -t registry.example.com/kai/multimodal-pattern-recognition:latest -f packages/ml/python/Dockerfile.multimodal-pattern-recognition .  
docker build -t registry.example.com/kai/domain-specific-networks:latest -f packages/ml/python/Dockerfile.domain-specific-networks .  
  
# Push all images to registry  
docker push registry.example.com/kai/api-server:latest  
docker push registry.example.com/kai/coordinator-service:latest  
docker push registry.example.com/kai/quality-assessment:latest  
# ... and so on for all images

# Build the base images first  
docker build -t registry.example.com/kai/node-base:latest -f Dockerfile.node-base .  
docker build -t registry.example.com/kai/python-base:latest -f Dockerfile.python-base .  
docker build -t registry.example.com/kai/ml-base:latest -f Dockerfile.ml-base .  
  
# Build service images  
docker build -t registry.example.com/kai/api-server:latest -f Dockerfile.api .  
docker build -t registry.example.com/kai/coordinator-service:latest -f packages/coordinator/Dockerfile.coordinator .  
docker build -t registry.example.com/kai/mcp-server:latest -f packages/ml/Dockerfile.mcp .  
  
# Build ML worker images  
docker build -t registry.example.com/kai/quality-assessment:latest -f packages/ml/python/Dockerfile.quality-assessment .  
docker build -t registry.example.com/kai/image-preprocessing:latest -f packages/ml/python/Dockerfile.image-preprocessing .  
docker build -t registry.example.com/kai/colmap-sfm:latest -f packages/ml/python/Dockerfile.colmap-sfm .  
docker build -t registry.example.com/kai/point-cloud:latest -f packages/ml/python/Dockerfile.point-cloud .  
docker build -t registry.example.com/kai/model-generator:latest -f packages/ml/python/Dockerfile.model-generator .  
docker build -t registry.example.com/kai/diffusion-nerf:latest -f packages/ml/python/Dockerfile.diffusion-nerf .  
docker build -t registry.example.com/kai/nerf-mesh-extractor:latest -f packages/ml/python/Dockerfile.nerf-mesh-extractor .  
docker build -t registry.example.com/kai/format-converter:latest -f packages/ml/python/Dockerfile.format-converter .  
docker build -t registry.example.com/kai/workflow-finalizer:latest -f packages/ml/python/Dockerfile.workflow-finalizer .  
  
# Build new feature images  
docker build -t registry.example.com/kai/parameter-registry:latest -f packages/server/Dockerfile.parameter-registry .  
docker build -t registry.example.com/kai/notification-service:latest -f packages/server/Dockerfile.notification-service .  
docker build -t registry.example.com/kai/webhook-service:latest -f packages/server/Dockerfile.webhook-service .  
docker build -t registry.example.com/kai/multimodal-pattern-recognition:latest -f packages/ml/python/Dockerfile.multimodal-pattern-recognition .  
docker build -t registry.example.com/kai/domain-specific-networks:latest -f packages/ml/python/Dockerfile.domain-specific-networks .  
  
# Push all images to registry  
docker push registry.example.com/kai/api-server:latest  
docker push registry.example.com/kai/coordinator-service:latest  
docker push registry.example.com/kai/quality-assessment:latest  
# ... and so on for all images

#### Kubernetes Deployment

##### Deployment Order

For optimal deployment with minimal service disruption, follow this order:

1. Infrastructure Updates:
2. Apply GPU configuration updates
3. Update node pools if necessary
4. Configure persistent storage
5. Coordinator Updates:
6. Update coordinator configuration (config.yaml)
7. Update coordinator deployment (deployment.yaml)
8. Apply updated resource allocations
9. Restart coordinator service
10. Core Services:
11. Deploy Parameter Registry service
12. Deploy Notification System service
13. ML Components:
14. Deploy Domain-Specific Networks
15. Deploy MultiModal Pattern Recognition service
16. Integration Components:
17. Deploy Webhook service
18. Update workflow templates
19. Configure integrations with existing systems

Configure persistent storage

Coordinator Updates:

Restart coordinator service

Core Services:

Deploy Notification System service

ML Components:

Deploy MultiModal Pattern Recognition service

Integration Components:

##### Coordinator Service Updates

The coordinator service has been updated to support the new features with the following changes:

###### Configuration Updates (config.yaml)

1. Quality Tiers
2. Added new "premium" and "enterprise" tiers with higher resources
3. Updated GPU class specifications for high-performance nodes
4. Notification System Integration
5. Added configuration for notification delivery
6. Configured event types that trigger notifications
7. Setup webhook integration points
8. Parameter Registry Integration
9. Added endpoint configuration
10. Configured similarity threshold and retention policies
11. Setup gRPC communication channel
12. ML Feature Management
13. Added configuration for multimodal pattern recognition
14. Added configuration for domain-specific networks
15. Setup workflow template references
16. GPU Resource Management
17. Added GPU class specifications
18. Configured model-to-GPU mapping
19. Setup autoscaling parameters

Updated GPU class specifications for high-performance nodes

Notification System Integration

Setup webhook integration points

Parameter Registry Integration

Setup gRPC communication channel

ML Feature Management

Setup workflow template references

GPU Resource Management

###### Deployment Updates (deployment.yaml)

1. Resource Allocation Increases
2. CPU: 500m → 1 core (requests), 2 → 4 cores (limits)
3. Memory: 512Mi → 1Gi (requests), 2Gi → 4Gi (limits)
4. Environment Variables
5. Added connectivity parameters for new services
6. Configured feature flags for new components
7. Setup workflow template paths

Memory: 512Mi → 1Gi (requests), 2Gi → 4Gi (limits)

Environment Variables

#### Canary Deployments

The KAI Platform supports automated canary deployments with health monitoring and automatic promotion or rollback based on metrics. This provides a safer way to roll out changes by testing them on a small subset of traffic before full deployment.

##### Using Canary Deployments

To deploy using the canary approach:

# Basic canary deployment (10% traffic)  
./helm-charts/helm-deploy.sh --context=kai-production-cluster --env=production --canary --tag=v1.2.3  
  
# Advanced canary configuration  
./helm-charts/helm-deploy.sh \  
 --context=kai-production-cluster \  
 --env=production \  
 --canary \  
 --canary-weight=20 \  
 --canary-time=15 \  
 --health-threshold=98 \  
 --critical-services=api-server,coordinator-service,mobile-optimization \  
 --tag=v1.2.3

# Basic canary deployment (10% traffic)  
./helm-charts/helm-deploy.sh --context=kai-production-cluster --env=production --canary --tag=v1.2.3  
  
# Advanced canary configuration  
./helm-charts/helm-deploy.sh \  
 --context=kai-production-cluster \  
 --env=production \  
 --canary \  
 --canary-weight=20 \  
 --canary-time=15 \  
 --health-threshold=98 \  
 --critical-services=api-server,coordinator-service,mobile-optimization \  
 --tag=v1.2.3

The canary deployment will:  
1. Deploy the new version alongside the existing version  
2. Route a percentage of traffic to the new version (10% by default)  
3. Monitor health metrics for the specified period (10 minutes by default)  
4. Automatically promote the canary to production if health checks pass  
5. Automatically roll back if health checks fail

##### Configuration Options

Table content:

Option | Description | Default

--canary | Enable canary deployment | -

--canary-weight=<pct> | Percentage of traffic to route to canary | 10%

--canary-time=<min> | Minutes to analyze canary before promotion | 10 minutes

--health-threshold=<pct> | Success rate threshold for promotion | 95%

--critical-services=<svc> | Comma-separated list of services to monitor | api-server,coordinator-service

--canary

--canary-weight=<pct>

--canary-time=<min>

--health-threshold=<pct>

--critical-services=<svc>

##### Health Monitoring

The system monitors several health metrics during the canary period:  
- Success rate (percentage of non-5xx responses)  
- Latency metrics  
- Resource utilization

#### Vercel Deployment

Vercel is used to deploy the Next.js admin panel and the Gatsby client frontend.

##### Admin Panel Deployment (Next.js)

1. Log in to Vercel
2. Click "Add New" → "Project"
3. Import your GitHub repository
4. Configure the project:
5. Framework Preset: Next.js
6. Root Directory: packages/admin
7. Build Command: yarn build
8. Output Directory: out (default)
9. Install Command: yarn install
10. Set up environment variables (see Environment Configuration section)
11. Click "Deploy"

packages/admin

yarn build

out

yarn install

##### Client App Deployment (Gatsby)

1. Log in to Vercel
2. Click "Add New" → "Project"
3. Import your GitHub repository (if not already imported)
4. Configure the project:
5. Framework Preset: Gatsby
6. Root Directory: packages/client
7. Build Command: yarn build
8. Output Directory: public (default)
9. Install Command: yarn install
10. Set up environment variables (see Environment Configuration section)
11. Click "Deploy"

packages/client

yarn build

public

yarn install

##### Custom Domain Configuration

1. In the Vercel project settings, go to Domains
2. Add your custom domain(s):
3. Admin Panel: admin.kai.yourdomain.com
4. Client App: kai.yourdomain.com
5. Configure DNS settings as instructed by Vercel

admin.kai.yourdomain.com

kai.yourdomain.com

##### Vercel Project Settings

For both projects, configure these additional settings:

1. Build & Development Settings:
2. Node.js Version: 16.x (or higher if required)
3. Include source files outside of the Root Directory: Yes
4. Install Command: cd ../.. && yarn install
5. Environment Variables:
6. Add all required environment variables (see Environment Configuration section)

Install Command: cd ../.. && yarn install

cd ../.. && yarn install

Environment Variables:

#### Supabase Deployment

Supabase is used for authentication, realtime features, and queue management.

##### Creating a Supabase Project

1. Sign in to Supabase
2. Click "New Project"
3. Enter project details:
4. Name: kai-production (or your preferred name)
5. Database Password: Generate a strong password
6. Region: Choose the region closest to your users
7. Click "Create new project"

kai-production

##### Setting up Database Schema

After your project is created, you'll need to set up the database schema. You can do this using the SQL editor in the Supabase dashboard.

1. Navigate to the SQL editor
2. Run the migration scripts in order:

-- Create necessary tables for queue management  
CREATE TABLE queue\_jobs (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 job\_type VARCHAR NOT NULL,  
 status VARCHAR NOT NULL DEFAULT 'pending',  
 payload JSONB NOT NULL,  
 result JSONB,  
 error TEXT,  
 priority INT NOT NULL DEFAULT 0,  
 attempts INT NOT NULL DEFAULT 0,  
 max\_attempts INT NOT NULL DEFAULT 3,  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 started\_at TIMESTAMPTZ,  
 completed\_at TIMESTAMPTZ  
);  
  
-- Create index for job processing  
CREATE INDEX queue\_jobs\_status\_priority\_created\_idx ON queue\_jobs (status, priority DESC, created\_at);  
  
-- Create realtime publications for queue updates  
BEGIN;  
 DROP PUBLICATION IF EXISTS supabase\_realtime;  
 CREATE PUBLICATION supabase\_realtime FOR TABLE queue\_jobs;  
COMMIT;  
  
-- Enable Row Level Security  
ALTER TABLE queue\_jobs ENABLE ROW LEVEL SECURITY;  
  
-- Create policy for authenticated users  
CREATE POLICY "Authenticated users can view all jobs"  
 ON queue\_jobs FOR SELECT  
 TO authenticated  
 USING (true);  
  
-- Create policy for service role to perform all operations  
CREATE POLICY "Service role can perform all operations"  
 ON queue\_jobs FOR ALL  
 TO service\_role  
 USING (true);

-- Create necessary tables for queue management  
CREATE TABLE queue\_jobs (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 job\_type VARCHAR NOT NULL,  
 status VARCHAR NOT NULL DEFAULT 'pending',  
 payload JSONB NOT NULL,  
 result JSONB,  
 error TEXT,  
 priority INT NOT NULL DEFAULT 0,  
 attempts INT NOT NULL DEFAULT 0,  
 max\_attempts INT NOT NULL DEFAULT 3,  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 started\_at TIMESTAMPTZ,  
 completed\_at TIMESTAMPTZ  
);  
  
-- Create index for job processing  
CREATE INDEX queue\_jobs\_status\_priority\_created\_idx ON queue\_jobs (status, priority DESC, created\_at);  
  
-- Create realtime publications for queue updates  
BEGIN;  
 DROP PUBLICATION IF EXISTS supabase\_realtime;  
 CREATE PUBLICATION supabase\_realtime FOR TABLE queue\_jobs;  
COMMIT;  
  
-- Enable Row Level Security  
ALTER TABLE queue\_jobs ENABLE ROW LEVEL SECURITY;  
  
-- Create policy for authenticated users  
CREATE POLICY "Authenticated users can view all jobs"  
 ON queue\_jobs FOR SELECT  
 TO authenticated  
 USING (true);  
  
-- Create policy for service role to perform all operations  
CREATE POLICY "Service role can perform all operations"  
 ON queue\_jobs FOR ALL  
 TO service\_role  
 USING (true);

##### Configuring Authentication

1. In the Supabase dashboard, go to Authentication → Settings
2. Configure the following settings:
3. Site URL: Your production frontend URL (e.g., https://kai.yourdomain.com)
4. Enable Email Auth: Yes
5. Enable Phone Auth: Optional
6. Enable OAuth Providers: As needed (Google, GitHub, etc.)

https://kai.yourdomain.com

##### Creating Service Role API Keys

1. In the Supabase dashboard, go to Settings → API
2. Copy the following values:
3. URL: Your Supabase project URL
4. anon public key: For client-side authentication
5. service\_role key: For server-side operations (keep this secure)

anon

service\_role

##### Enabling Realtime

1. In the Supabase dashboard, go to Database → Replication
2. Ensure the supabase\_realtime publication is configured properly
3. Go to Settings → API → Realtime and enable it

supabase\_realtime

### CI/CD Pipeline

The KAI Platform uses GitHub Actions for continuous integration and deployment. The enhanced CI/CD pipeline reduces code duplication, improves efficiency, and adds automatic rollback capabilities.

#### Optimized Workflow Structure

The CI/CD workflow is defined in .github/workflows/deploy.yml with the following optimizations:

.github/workflows/deploy.yml

1. Matrix-Based Docker Builds: All images are built in parallel using a matrix strategy
2. Unified Deployment Job: A single job handles both staging and production deployments
3. Automatic Environment Detection: Environment is determined from branch or manual trigger
4. Dynamic Configuration: Environment-specific settings applied via variables
5. Health Monitoring: Automatic verification and rollback if deployments fail

name: Kai Platform CI/CD Pipeline  
  
on:  
 push:  
 branches: [main, staging, development]  
 pull\_request:  
 branches: [main, staging, development]  
 workflow\_dispatch:  
 inputs:  
 environment:  
 description: 'Environment to deploy to'  
 required: true  
 default: 'staging'  
 type: choice  
 options:  
 - staging  
 - production  
  
env:  
 NODE\_VERSION: '16'  
 PYTHON\_VERSION: '3.9'  
 DOCKER\_BUILDKIT: '1'  
  
jobs:  
 # Build and test job runs on all branches  
 build-and-test:  
 name: Build and Test  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Setup Node.js  
 uses: actions/setup-node@v3  
 with:  
 node-version: ${{ env.NODE\_VERSION }}  
 cache: 'yarn'  
  
 - name: Install dependencies  
 run: yarn install --frozen-lockfile  
  
 - name: Run linting  
 run: yarn lint  
  
 - name: Run unit tests  
 run: yarn test  
  
 - name: Build packages  
 run: yarn build  
  
 - name: Upload build artifacts  
 uses: actions/upload-artifact@v3  
 with:  
 name: build-artifacts  
 path: |  
 packages/\*/dist  
 packages/client/public  
 packages/admin/out  
 retention-days: 1  
  
 # Build Docker images with matrix strategy  
 build-docker-images:  
 name: Build Docker Images  
 needs: build-and-test  
 if: |  
 (github.ref == 'refs/heads/staging') ||  
 (github.ref == 'refs/heads/main') ||  
 (github.event\_name == 'workflow\_dispatch')  
 runs-on: ubuntu-latest  
 strategy:  
 matrix:  
 include:  
 # Main services  
 - name: api-server  
 dockerfile: ./Dockerfile.api  
 context: .  
 - name: coordinator-service  
 dockerfile: ./packages/coordinator/Dockerfile.coordinator  
 context: .  
 # ML workers  
 - name: quality-assessment  
 dockerfile: ./packages/ml/python/Dockerfile.quality-assessment  
 context: .  
 - name: image-preprocessing  
 dockerfile: ./packages/ml/python/Dockerfile.image-preprocessing  
 context: .  
 - name: colmap-sfm  
 dockerfile: ./packages/ml/python/Dockerfile.colmap-sfm  
 context: .  
 # Additional workers defined similarly  
 steps:  
 - name: Determine environment  
 id: env  
 run: |  
 if [[ "${{ github.ref }}" == "refs/heads/main" || "${{ github.event.inputs.environment }}" == "production" ]]; then  
 echo "DEPLOY\_ENV=production" >> $GITHUB\_ENV  
 echo "TAG\_SUFFIX=latest" >> $GITHUB\_ENV  
 else  
 echo "DEPLOY\_ENV=staging" >> $GITHUB\_ENV  
 echo "TAG\_SUFFIX=staging" >> $GITHUB\_ENV  
 fi  
  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Set up Docker Buildx  
 uses: docker/setup-buildx-action@v2  
  
 - name: Login to GitHub Container Registry  
 uses: docker/login-action@v2  
 with:  
 registry: ghcr.io  
 username: ${{ github.repository\_owner }}  
 password: ${{ secrets.GITHUB\_TOKEN }}  
  
 - name: Build and push image  
 uses: docker/build-push-action@v4  
 with:  
 context: ${{ matrix.context }}  
 file: ${{ matrix.dockerfile }}  
 push: true  
 tags: |  
 ghcr.io/${{ github.repository }}/kai-${{ matrix.name }}:${{ github.sha }}  
 ghcr.io/${{ github.repository }}/kai-${{ matrix.name }}:${{ env.TAG\_SUFFIX }}  
 cache-from: type=registry,ref=ghcr.io/${{ github.repository }}/kai-${{ matrix.name }}:${{ env.TAG\_SUFFIX }}-cache  
 cache-to: type=registry,ref=ghcr.io/${{ github.repository }}/kai-${{ matrix.name }}:${{ env.TAG\_SUFFIX }}-cache,mode=max  
 build-args: |  
 BUILDKIT\_INLINE\_CACHE=1  
 ENVIRONMENT=${{ env.DEPLOY\_ENV }}  
  
 # Unified deployment job for both staging and production  
 deploy:  
 name: Deploy to ${{ github.event.inputs.environment || (github.ref == 'refs/heads/main' && 'production' || 'staging') }}  
 needs: build-docker-images  
 if: |  
 (github.ref == 'refs/heads/staging') ||  
 (github.ref == 'refs/heads/main') ||  
 (github.event\_name == 'workflow\_dispatch')  
 runs-on: ubuntu-latest  
 concurrency:  
 group: ${{ github.event.inputs.environment || (github.ref == 'refs/heads/main' && 'production' || 'staging') }}\_environment  
 cancel-in-progress: false  
 steps:  
 - name: Determine environment  
 id: env  
 run: |  
 if [[ "${{ github.ref }}" == "refs/heads/main" || "${{ github.event.inputs.environment }}" == "production" ]]; then  
 echo "DEPLOY\_ENV=production" >> $GITHUB\_ENV  
 echo "KUBE\_CONTEXT=kai-production-cluster" >> $GITHUB\_ENV  
 echo "API\_URL=https://api.kai.yourdomain.com" >> $GITHUB\_ENV  
 echo "SUPABASE\_URL=${{ secrets.SUPABASE\_URL\_PRODUCTION }}" >> $GITHUB\_ENV  
 echo "SUPABASE\_KEY=${{ secrets.SUPABASE\_KEY\_PRODUCTION }}" >> $GITHUB\_ENV  
 echo "VERCEL\_ARGS=--prod" >> $GITHUB\_ENV  
 echo "TEST\_SCRIPT=test:smoke" >> $GITHUB\_ENV  
 else  
 echo "DEPLOY\_ENV=staging" >> $GITHUB\_ENV  
 echo "KUBE\_CONTEXT=kai-staging-cluster" >> $GITHUB\_ENV  
 echo "API\_URL=https://api-staging.kai.yourdomain.com" >> $GITHUB\_ENV  
 echo "SUPABASE\_URL=${{ secrets.SUPABASE\_URL\_STAGING }}" >> $GITHUB\_ENV  
 echo "SUPABASE\_KEY=${{ secrets.SUPABASE\_KEY\_STAGING }}" >> $GITHUB\_ENV  
 echo "VERCEL\_ARGS=" >> $GITHUB\_ENV  
 echo "TEST\_SCRIPT=test:integration" >> $GITHUB\_ENV  
 fi  
  
 # ... more deployment steps ...  
  
 # Run database migrations before deployment  
 - name: Setup Node.js for migrations  
 uses: actions/setup-node@v3  
 with:  
 node-version: ${{ env.NODE\_VERSION }}  
  
 - name: Install dependencies  
 run: yarn install --frozen-lockfile  
  
 - name: Run database migrations  
 run: |  
 echo "Running database migrations for ${{ env.DEPLOY\_ENV }} environment..."  
 yarn tsc -p packages/server/tsconfig.json  
 cd packages/server  
 node dist/scripts/run-migrations.js  
 env:  
 SUPABASE\_URL: ${{ env.SUPABASE\_URL }}  
 SUPABASE\_KEY: ${{ env.SUPABASE\_KEY }}  
 NODE\_ENV: ${{ env.DEPLOY\_ENV }}  
  
 # Deploy to Kubernetes with enhanced script  
 - name: Deploy to Kubernetes with rollback support  
 id: deploy  
 run: |  
 echo "Applying Kubernetes manifests for ${{ env.DEPLOY\_ENV }}..."  
  
 # Create a backup of current deployments for potential rollback  
 echo "Creating backup of current deployments..."  
 kubectl --context=${{ env.KUBE\_CONTEXT }} get deployments -n kai-system -o yaml > deployments-backup.yaml  
  
 # Apply the deployment with environment parameter  
 chmod +x ./kubernetes/deploy.sh  
 ./kubernetes/deploy.sh --context=${{ env.KUBE\_CONTEXT }} --registry=ghcr.io/${{ github.repository }} --tag=${{ github.sha }} --env=${{ env.DEPLOY\_ENV }}  
  
 echo "deployment\_id=$(date +%s)" >> $GITHUB\_OUTPUT  
  
 # Monitor deployment health  
 - name: Monitor deployment health  
 id: monitor  
 run: |  
 echo "Monitoring deployment health for 2 minutes..."  
 FAILURES=0  
  
 for i in {1..12}; do  
 sleep 10  
 HEALTH\_STATUS=$(curl -s -o /dev/null -w "%{http\_code}" ${{ env.API\_URL }}/health || echo "000")  
  
 if [ "$HEALTH\_STATUS" != "200" ]; then  
 FAILURES=$((FAILURES+1))  
 echo "::warning::Health check #$i failed with status $HEALTH\_STATUS"  
 else  
 echo "Health check #$i passed"  
 fi  
 done  
  
 if [ "$FAILURES" -gt 3 ]; then  
 echo "::error::Too many health check failures. Initiating rollback."  
 echo "rollback=true" >> $GITHUB\_OUTPUT  
 else  
 echo "Deployment stable. Health checks passed."  
 echo "rollback=false" >> $GITHUB\_OUTPUT  
 fi  
  
 # Rollback if necessary  
 - name: Rollback deployment if needed  
 if: steps.monitor.outputs.rollback == 'true'  
 run: |  
 echo "::warning::Initiating rollback due to health check failures!"  
 kubectl --context=${{ env.KUBE\_CONTEXT }} apply -f deployments-backup.yaml

name: Kai Platform CI/CD Pipeline  
  
on:  
 push:  
 branches: [main, staging, development]  
 pull\_request:  
 branches: [main, staging, development]  
 workflow\_dispatch:  
 inputs:  
 environment:  
 description: 'Environment to deploy to'  
 required: true  
 default: 'staging'  
 type: choice  
 options:  
 - staging  
 - production  
  
env:  
 NODE\_VERSION: '16'  
 PYTHON\_VERSION: '3.9'  
 DOCKER\_BUILDKIT: '1'  
  
jobs:  
 # Build and test job runs on all branches  
 build-and-test:  
 name: Build and Test  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Setup Node.js  
 uses: actions/setup-node@v3  
 with:  
 node-version: ${{ env.NODE\_VERSION }}  
 cache: 'yarn'  
  
 - name: Install dependencies  
 run: yarn install --frozen-lockfile  
  
 - name: Run linting  
 run: yarn lint  
  
 - name: Run unit tests  
 run: yarn test  
  
 - name: Build packages  
 run: yarn build  
  
 - name: Upload build artifacts  
 uses: actions/upload-artifact@v3  
 with:  
 name: build-artifacts  
 path: |  
 packages/\*/dist  
 packages/client/public  
 packages/admin/out  
 retention-days: 1  
  
 # Build Docker images with matrix strategy  
 build-docker-images:  
 name: Build Docker Images  
 needs: build-and-test  
 if: |  
 (github.ref == 'refs/heads/staging') ||  
 (github.ref == 'refs/heads/main') ||  
 (github.event\_name == 'workflow\_dispatch')  
 runs-on: ubuntu-latest  
 strategy:  
 matrix:  
 include:  
 # Main services  
 - name: api-server  
 dockerfile: ./Dockerfile.api  
 context: .  
 - name: coordinator-service  
 dockerfile: ./packages/coordinator/Dockerfile.coordinator  
 context: .  
 # ML workers  
 - name: quality-assessment  
 dockerfile: ./packages/ml/python/Dockerfile.quality-assessment  
 context: .  
 - name: image-preprocessing  
 dockerfile: ./packages/ml/python/Dockerfile.image-preprocessing  
 context: .  
 - name: colmap-sfm  
 dockerfile: ./packages/ml/python/Dockerfile.colmap-sfm  
 context: .  
 # Additional workers defined similarly  
 steps:  
 - name: Determine environment  
 id: env  
 run: |  
 if [[ "${{ github.ref }}" == "refs/heads/main" || "${{ github.event.inputs.environment }}" == "production" ]]; then  
 echo "DEPLOY\_ENV=production" >> $GITHUB\_ENV  
 echo "TAG\_SUFFIX=latest" >> $GITHUB\_ENV  
 else  
 echo "DEPLOY\_ENV=staging" >> $GITHUB\_ENV  
 echo "TAG\_SUFFIX=staging" >> $GITHUB\_ENV  
 fi  
  
 - name: Checkout code  
 uses: actions/checkout@v3  
  
 - name: Set up Docker Buildx  
 uses: docker/setup-buildx-action@v2  
  
 - name: Login to GitHub Container Registry  
 uses: docker/login-action@v2  
 with:  
 registry: ghcr.io  
 username: ${{ github.repository\_owner }}  
 password: ${{ secrets.GITHUB\_TOKEN }}  
  
 - name: Build and push image  
 uses: docker/build-push-action@v4  
 with:  
 context: ${{ matrix.context }}  
 file: ${{ matrix.dockerfile }}  
 push: true  
 tags: |  
 ghcr.io/${{ github.repository }}/kai-${{ matrix.name }}:${{ github.sha }}  
 ghcr.io/${{ github.repository }}/kai-${{ matrix.name }}:${{ env.TAG\_SUFFIX }}  
 cache-from: type=registry,ref=ghcr.io/${{ github.repository }}/kai-${{ matrix.name }}:${{ env.TAG\_SUFFIX }}-cache  
 cache-to: type=registry,ref=ghcr.io/${{ github.repository }}/kai-${{ matrix.name }}:${{ env.TAG\_SUFFIX }}-cache,mode=max  
 build-args: |  
 BUILDKIT\_INLINE\_CACHE=1  
 ENVIRONMENT=${{ env.DEPLOY\_ENV }}  
  
 # Unified deployment job for both staging and production  
 deploy:  
 name: Deploy to ${{ github.event.inputs.environment || (github.ref == 'refs/heads/main' && 'production' || 'staging') }}  
 needs: build-docker-images  
 if: |  
 (github.ref == 'refs/heads/staging') ||  
 (github.ref == 'refs/heads/main') ||  
 (github.event\_name == 'workflow\_dispatch')  
 runs-on: ubuntu-latest  
 concurrency:  
 group: ${{ github.event.inputs.environment || (github.ref == 'refs/heads/main' && 'production' || 'staging') }}\_environment  
 cancel-in-progress: false  
 steps:  
 - name: Determine environment  
 id: env  
 run: |  
 if [[ "${{ github.ref }}" == "refs/heads/main" || "${{ github.event.inputs.environment }}" == "production" ]]; then  
 echo "DEPLOY\_ENV=production" >> $GITHUB\_ENV  
 echo "KUBE\_CONTEXT=kai-production-cluster" >> $GITHUB\_ENV  
 echo "API\_URL=https://api.kai.yourdomain.com" >> $GITHUB\_ENV  
 echo "SUPABASE\_URL=${{ secrets.SUPABASE\_URL\_PRODUCTION }}" >> $GITHUB\_ENV  
 echo "SUPABASE\_KEY=${{ secrets.SUPABASE\_KEY\_PRODUCTION }}" >> $GITHUB\_ENV  
 echo "VERCEL\_ARGS=--prod" >> $GITHUB\_ENV  
 echo "TEST\_SCRIPT=test:smoke" >> $GITHUB\_ENV  
 else  
 echo "DEPLOY\_ENV=staging" >> $GITHUB\_ENV  
 echo "KUBE\_CONTEXT=kai-staging-cluster" >> $GITHUB\_ENV  
 echo "API\_URL=https://api-staging.kai.yourdomain.com" >> $GITHUB\_ENV  
 echo "SUPABASE\_URL=${{ secrets.SUPABASE\_URL\_STAGING }}" >> $GITHUB\_ENV  
 echo "SUPABASE\_KEY=${{ secrets.SUPABASE\_KEY\_STAGING }}" >> $GITHUB\_ENV  
 echo "VERCEL\_ARGS=" >> $GITHUB\_ENV  
 echo "TEST\_SCRIPT=test:integration" >> $GITHUB\_ENV  
 fi  
  
 # ... more deployment steps ...  
  
 # Run database migrations before deployment  
 - name: Setup Node.js for migrations  
 uses: actions/setup-node@v3  
 with:  
 node-version: ${{ env.NODE\_VERSION }}  
  
 - name: Install dependencies  
 run: yarn install --frozen-lockfile  
  
 - name: Run database migrations  
 run: |  
 echo "Running database migrations for ${{ env.DEPLOY\_ENV }} environment..."  
 yarn tsc -p packages/server/tsconfig.json  
 cd packages/server  
 node dist/scripts/run-migrations.js  
 env:  
 SUPABASE\_URL: ${{ env.SUPABASE\_URL }}  
 SUPABASE\_KEY: ${{ env.SUPABASE\_KEY }}  
 NODE\_ENV: ${{ env.DEPLOY\_ENV }}  
  
 # Deploy to Kubernetes with enhanced script  
 - name: Deploy to Kubernetes with rollback support  
 id: deploy  
 run: |  
 echo "Applying Kubernetes manifests for ${{ env.DEPLOY\_ENV }}..."  
  
 # Create a backup of current deployments for potential rollback  
 echo "Creating backup of current deployments..."  
 kubectl --context=${{ env.KUBE\_CONTEXT }} get deployments -n kai-system -o yaml > deployments-backup.yaml  
  
 # Apply the deployment with environment parameter  
 chmod +x ./kubernetes/deploy.sh  
 ./kubernetes/deploy.sh --context=${{ env.KUBE\_CONTEXT }} --registry=ghcr.io/${{ github.repository }} --tag=${{ github.sha }} --env=${{ env.DEPLOY\_ENV }}  
  
 echo "deployment\_id=$(date +%s)" >> $GITHUB\_OUTPUT  
  
 # Monitor deployment health  
 - name: Monitor deployment health  
 id: monitor  
 run: |  
 echo "Monitoring deployment health for 2 minutes..."  
 FAILURES=0  
  
 for i in {1..12}; do  
 sleep 10  
 HEALTH\_STATUS=$(curl -s -o /dev/null -w "%{http\_code}" ${{ env.API\_URL }}/health || echo "000")  
  
 if [ "$HEALTH\_STATUS" != "200" ]; then  
 FAILURES=$((FAILURES+1))  
 echo "::warning::Health check #$i failed with status $HEALTH\_STATUS"  
 else  
 echo "Health check #$i passed"  
 fi  
 done  
  
 if [ "$FAILURES" -gt 3 ]; then  
 echo "::error::Too many health check failures. Initiating rollback."  
 echo "rollback=true" >> $GITHUB\_OUTPUT  
 else  
 echo "Deployment stable. Health checks passed."  
 echo "rollback=false" >> $GITHUB\_OUTPUT  
 fi  
  
 # Rollback if necessary  
 - name: Rollback deployment if needed  
 if: steps.monitor.outputs.rollback == 'true'  
 run: |  
 echo "::warning::Initiating rollback due to health check failures!"  
 kubectl --context=${{ env.KUBE\_CONTEXT }} apply -f deployments-backup.yaml

#### CI/CD Integration with Flux

For Flux GitOps deployments, the CI/CD pipeline includes a job to update the GitOps repository with new image versions:

update-gitops:  
 name: Update GitOps Repository  
 needs: build-docker-images  
 runs-on: ubuntu-latest  
 steps:  
 - name: Determine environment  
 id: env  
 run: |  
 if [[ "${{ github.ref }}" == "refs/heads/main" || "${{ github.event.inputs.environment }}" == "production" ]]; then  
 echo "DEPLOY\_ENV=production" >> $GITHUB\_ENV  
 echo "TARGET\_BRANCH=main" >> $GITHUB\_ENV  
 else  
 echo "DEPLOY\_ENV=staging" >> $GITHUB\_ENV  
 echo "TARGET\_BRANCH=staging" >> $GITHUB\_ENV  
 fi  
  
 - name: Checkout GitOps repository  
 uses: actions/checkout@v3  
 with:  
 repository: kai-platform/kai-gitops  
 path: gitops  
 token: ${{ secrets.GITOPS\_PAT }}  
 ref: ${{ env.TARGET\_BRANCH }}  
  
 - name: Update image tags in HelmReleases  
 run: |  
 echo "Updating image tags for ${{ env.DEPLOY\_ENV }} environment..."  
  
 # Update coordinator release  
 cd gitops/clusters/${{ env.DEPLOY\_ENV }}/releases  
  
 # Use yq to update the image tag in the HelmRelease  
 yq e '.spec.values.image.tag = "${{ github.sha }}"' -i coordinator.yaml  
  
 # Additional services can be updated similarly  
  
 git config --global user.name "Kai CI Bot"  
 git config --global user.email "ci-bot@kai-platform.com"  
  
 git add .  
 git commit -m "ci: update image tags to ${{ github.sha }} for ${{ env.DEPLOY\_ENV }}" || echo "No changes to commit"  
 git push

update-gitops:  
 name: Update GitOps Repository  
 needs: build-docker-images  
 runs-on: ubuntu-latest  
 steps:  
 - name: Determine environment  
 id: env  
 run: |  
 if [[ "${{ github.ref }}" == "refs/heads/main" || "${{ github.event.inputs.environment }}" == "production" ]]; then  
 echo "DEPLOY\_ENV=production" >> $GITHUB\_ENV  
 echo "TARGET\_BRANCH=main" >> $GITHUB\_ENV  
 else  
 echo "DEPLOY\_ENV=staging" >> $GITHUB\_ENV  
 echo "TARGET\_BRANCH=staging" >> $GITHUB\_ENV  
 fi  
  
 - name: Checkout GitOps repository  
 uses: actions/checkout@v3  
 with:  
 repository: kai-platform/kai-gitops  
 path: gitops  
 token: ${{ secrets.GITOPS\_PAT }}  
 ref: ${{ env.TARGET\_BRANCH }}  
  
 - name: Update image tags in HelmReleases  
 run: |  
 echo "Updating image tags for ${{ env.DEPLOY\_ENV }} environment..."  
  
 # Update coordinator release  
 cd gitops/clusters/${{ env.DEPLOY\_ENV }}/releases  
  
 # Use yq to update the image tag in the HelmRelease  
 yq e '.spec.values.image.tag = "${{ github.sha }}"' -i coordinator.yaml  
  
 # Additional services can be updated similarly  
  
 git config --global user.name "Kai CI Bot"  
 git config --global user.email "ci-bot@kai-platform.com"  
  
 git add .  
 git commit -m "ci: update image tags to ${{ github.sha }} for ${{ env.DEPLOY\_ENV }}" || echo "No changes to commit"  
 git push

#### GitHub Secrets and Environments

The pipeline uses the following secrets, which should be set in your GitHub repository:

* GITHUB\_TOKEN: GitHub token with write:packages permission (automatically provided by GitHub Actions)
* KUBE\_CONFIG\_DATA: Base64-encoded Kubernetes config file
* VERCEL\_TOKEN: Vercel API token
* VERCEL\_ORG\_ID: Vercel organization ID
* VERCEL\_PROJECT\_ID\_CLIENT: Vercel project ID for the client app
* VERCEL\_PROJECT\_ID\_ADMIN: Vercel project ID for the admin panel
* SUPABASE\_URL\_STAGING: Supabase URL for staging
* SUPABASE\_KEY\_STAGING: Supabase service role key for staging
* SUPABASE\_URL\_PRODUCTION: Supabase URL for production
* SUPABASE\_KEY\_PRODUCTION: Supabase service role key for production
* GITOPS\_PAT: GitHub Personal Access Token with repo scope (for Flux GitOps)
* SLACK\_WEBHOOK: Slack webhook URL for notifications (optional)

GITHUB\_TOKEN

write:packages

KUBE\_CONFIG\_DATA

VERCEL\_TOKEN

VERCEL\_ORG\_ID

VERCEL\_PROJECT\_ID\_CLIENT

VERCEL\_PROJECT\_ID\_ADMIN

SUPABASE\_URL\_STAGING

SUPABASE\_KEY\_STAGING

SUPABASE\_URL\_PRODUCTION

SUPABASE\_KEY\_PRODUCTION

GITOPS\_PAT

SLACK\_WEBHOOK

### Development Environment

#### Local Setup

##### Prerequisites

* Node.js (v16 or higher)
* Yarn (v1.22 or higher)
* MongoDB (v4.4 or higher)
* Python 3.8+ (for ML components)
* Docker and Docker Compose
* Git

##### Setup Steps

1. Clone the repository  
   bash  
    git clone https://github.com/your-org/kai.git  
    cd kai
2. Install dependencies  
   bash  
    yarn install
3. Set up environment variables  
    ```bash  
    # Use the centralized .env file in the root directory  
    cp .env.example .env

Clone the repository  
bash  
 git clone https://github.com/your-org/kai.git  
 cd kai

bash  
 git clone https://github.com/your-org/kai.git  
 cd kai

Install dependencies  
bash  
 yarn install

bash  
 yarn install

Set up environment variables  
 ```bash  
 # Use the centralized .env file in the root directory  
 cp .env.example .env

# Configure all necessary environment variables in the .env file  
 # including database connections, API keys, services URLs, etc.  
 ```

1. Set up MongoDB  
    ```bash  
    # Option 1: Using Docker  
    docker run -d -p 27017:27017 --name kai-mongodb mongo:5

# Option 2: Using MongoDB Atlas  
 # Configure your MongoDB Atlas connection string in the root .env file  
 ```

1. Set up ML environment  
   bash  
    cd packages/ml  
    python -m venv venv  
    source venv/bin/activate # On Windows: venv\Scripts\activate  
    pip install -r requirements.txt
2. Set up Supabase for the queue system  
    ```bash  
    # Option 1: Using Supabase cloud  
    # Create a project at https://supabase.com and configure in the root .env file

Set up ML environment  
bash  
 cd packages/ml  
 python -m venv venv  
 source venv/bin/activate # On Windows: venv\Scripts\activate  
 pip install -r requirements.txt

bash  
 cd packages/ml  
 python -m venv venv  
 source venv/bin/activate # On Windows: venv\Scripts\activate  
 pip install -r requirements.txt

Set up Supabase for the queue system  
 ```bash  
 # Option 1: Using Supabase cloud  
 # Create a project at https://supabase.com and configure in the root .env file

# Option 2: Using Supabase local development  
 npx supabase start  
 ```

1. Initialize database  
   bash  
    yarn workspace @kai/server db:init

bash  
 yarn workspace @kai/server db:init

#### Running the Development Environment

##### Starting the Backend

# Start the API server  
yarn workspace @kai/server dev  
  
# In another terminal, start the ML services  
cd packages/ml  
source venv/bin/activate # On Windows: venv\Scripts\activate  
python python/server.py

# Start the API server  
yarn workspace @kai/server dev  
  
# In another terminal, start the ML services  
cd packages/ml  
source venv/bin/activate # On Windows: venv\Scripts\activate  
python python/server.py

##### Starting the Frontend

# Start the client app  
yarn workspace @kai/client dev  
  
# In another terminal, start the admin app  
yarn workspace @kai/admin dev

# Start the client app  
yarn workspace @kai/client dev  
  
# In another terminal, start the admin app  
yarn workspace @kai/admin dev

##### Using Docker Compose for Development

Create a docker-compose.dev.yml file:

docker-compose.dev.yml

version: '3.8'  
  
services:  
 mongodb:  
 image: mongo:5  
 ports:  
 - "27017:27017"  
 volumes:  
 - mongodb-data:/data/db  
 environment:  
 MONGO\_INITDB\_ROOT\_USERNAME: kai  
 MONGO\_INITDB\_ROOT\_PASSWORD: password  
  
 supabase:  
 image: supabase/supabase-local  
 ports:  
 - "8000:8000"  
 volumes:  
 - supabase-data:/var/lib/postgresql/data  
  
volumes:  
 mongodb-data:  
 supabase-data:

version: '3.8'  
  
services:  
 mongodb:  
 image: mongo:5  
 ports:  
 - "27017:27017"  
 volumes:  
 - mongodb-data:/data/db  
 environment:  
 MONGO\_INITDB\_ROOT\_USERNAME: kai  
 MONGO\_INITDB\_ROOT\_PASSWORD: password  
  
 supabase:  
 image: supabase/supabase-local  
 ports:  
 - "8000:8000"  
 volumes:  
 - supabase-data:/var/lib/postgresql/data  
  
volumes:  
 mongodb-data:  
 supabase-data:

Start the development dependencies:

docker-compose -f docker-compose.dev.yml up -d

docker-compose -f docker-compose.dev.yml up -d

#### Development Workflow

##### Using the MCP Server in Development

# Start the MCP server  
cd packages/ml  
source mcp-venv/bin/activate # On Windows: mcp-venv\Scripts\activate  
python python/mcp\_server.py  
  
# In your application code, enable MCP integration  
# Set environment variables:  
# - MCP\_SERVER\_URL=http://localhost:8000  
# - USE\_MCP\_SERVER=true

# Start the MCP server  
cd packages/ml  
source mcp-venv/bin/activate # On Windows: mcp-venv\Scripts\activate  
python python/mcp\_server.py  
  
# In your application code, enable MCP integration  
# Set environment variables:  
# - MCP\_SERVER\_URL=http://localhost:8000  
# - USE\_MCP\_SERVER=true

##### Code Organization

packages/  
├── admin/ # Admin Panel (Next.js)  
├── client/ # Client App (Gatsby)  
├── ml/ # Machine Learning  
│ ├── python/ # Python ML code  
│ └── src/ # TypeScript interfaces  
├── server/ # API Server  
│ ├── src/ # Source code  
│ │ ├── controllers/ # API controllers  
│ │ ├── middleware/ # Middleware  
│ │ ├── models/ # Data models  
│ │ ├── routes/ # API routes  
│ │ ├── services/ # Business logic  
│ │ └── utils/ # Utilities  
│ └── tests/ # Tests  
└── shared/ # Shared code and types

packages/  
├── admin/ # Admin Panel (Next.js)  
├── client/ # Client App (Gatsby)  
├── ml/ # Machine Learning  
│ ├── python/ # Python ML code  
│ └── src/ # TypeScript interfaces  
├── server/ # API Server  
│ ├── src/ # Source code  
│ │ ├── controllers/ # API controllers  
│ │ ├── middleware/ # Middleware  
│ │ ├── models/ # Data models  
│ │ ├── routes/ # API routes  
│ │ ├── services/ # Business logic  
│ │ └── utils/ # Utilities  
│ └── tests/ # Tests  
└── shared/ # Shared code and types

##### Git Workflow

1. Branch naming convention
2. Feature branches: feature/feature-name
3. Bug fix branches: fix/bug-name
4. Refactoring branches: refactor/refactor-name
5. Documentation branches: docs/doc-name
6. Commit message convention
7. Use conventional commits format: type(scope): message
8. Example: feat(material-recognition): add confidence fusion algorithm
9. Pull Request workflow
10. Create PR against main branch
11. Require at least one review
12. Pass all automated checks
13. Maintain clean commit history (rebase preferred over merge)

feature/feature-name

fix/bug-name

refactor/refactor-name

Documentation branches: docs/doc-name

docs/doc-name

Commit message convention

type(scope): message

Example: feat(material-recognition): add confidence fusion algorithm

feat(material-recognition): add confidence fusion algorithm

Pull Request workflow

main

#### Debugging

##### Backend Debugging

1. Using Node.js Inspector  
    ```bash  
    # Start API server in debug mode  
    yarn workspace @kai/server dev:debug

# Then connect using Chrome DevTools or VS Code  
 ```

1. Using VS Code  
    Create a .vscode/launch.json file:  
    json  
    {  
    "version": "0.2.0",  
    "configurations": [  
    {  
    "type": "node",  
    "request": "launch",  
    "name": "Debug API Server",  
    "runtimeExecutable": "yarn",  
    "runtimeArgs": ["workspace", "@kai/server", "dev:debug"],  
    "port": 9229,  
    "sourceMaps": true,  
    "smartStep": true,  
    "outFiles": ["${workspaceFolder}/packages/server/dist/\*\*/\*.js"]  
    }  
    ]  
    }

.vscode/launch.json

json  
 {  
 "version": "0.2.0",  
 "configurations": [  
 {  
 "type": "node",  
 "request": "launch",  
 "name": "Debug API Server",  
 "runtimeExecutable": "yarn",  
 "runtimeArgs": ["workspace", "@kai/server", "dev:debug"],  
 "port": 9229,  
 "sourceMaps": true,  
 "smartStep": true,  
 "outFiles": ["${workspaceFolder}/packages/server/dist/\*\*/\*.js"]  
 }  
 ]  
 }

##### Frontend Debugging

1. React Developer Tools
2. Install the browser extension
3. Use React DevTools Profiler for performance analysis
4. Redux DevTools (if using Redux)
5. Install the browser extension
6. Monitor state changes and actions

Use React DevTools Profiler for performance analysis

Redux DevTools (if using Redux)

##### Python ML Debugging

1. Using VS Code  
    Configure .vscode/launch.json:  
    json  
    {  
    "version": "0.2.0",  
    "configurations": [  
    {  
    "name": "Python: ML Server",  
    "type": "python",  
    "request": "launch",  
    "program": "${workspaceFolder}/packages/ml/python/server.py",  
    "console": "integratedTerminal",  
    "env": {  
    "PYTHONPATH": "${workspaceFolder}/packages/ml"  
    }  
    }  
    ]  
    }
2. Using Python Debugger  
   python  
    import pdb; pdb.set\_trace()

Using VS Code  
 Configure .vscode/launch.json:  
 json  
 {  
 "version": "0.2.0",  
 "configurations": [  
 {  
 "name": "Python: ML Server",  
 "type": "python",  
 "request": "launch",  
 "program": "${workspaceFolder}/packages/ml/python/server.py",  
 "console": "integratedTerminal",  
 "env": {  
 "PYTHONPATH": "${workspaceFolder}/packages/ml"  
 }  
 }  
 ]  
 }

.vscode/launch.json

json  
 {  
 "version": "0.2.0",  
 "configurations": [  
 {  
 "name": "Python: ML Server",  
 "type": "python",  
 "request": "launch",  
 "program": "${workspaceFolder}/packages/ml/python/server.py",  
 "console": "integratedTerminal",  
 "env": {  
 "PYTHONPATH": "${workspaceFolder}/packages/ml"  
 }  
 }  
 ]  
 }

Using Python Debugger  
python  
 import pdb; pdb.set\_trace()

python  
 import pdb; pdb.set\_trace()

#### Common Development Tasks

##### Adding a New Feature

1. Plan the feature
2. Define requirements and acceptance criteria
3. Identify affected packages and components
4. Create design document if needed
5. Implement the feature
6. Create a feature branch
7. Implement code changes
8. Write tests
9. Document the feature
10. Review and testing
11. Create a pull request
12. Address code review feedback
13. Verify all tests pass
14. Perform manual testing
15. Merge and deploy
16. Merge to main branch
17. Deploy to staging environment
18. Verify in staging
19. Deploy to production

Create design document if needed

Implement the feature

Document the feature

Review and testing

Perform manual testing

Merge and deploy

##### Adding a New API Endpoint

1. Define the endpoint
2. Determine HTTP method and path
3. Define request parameters and body schema
4. Define response schema and status codes
5. Implement the endpoint  
    ```typescript  
    // In packages/server/src/routes/material.routes.ts  
    router.post('/materials/search', validateSearchRequest, materialController.searchMaterials);

Define response schema and status codes

Implement the endpoint  
 ```typescript  
 // In packages/server/src/routes/material.routes.ts  
 router.post('/materials/search', validateSearchRequest, materialController.searchMaterials);

// In packages/server/src/controllers/material.controller.ts  
 export const searchMaterials = async (req: Request, res: Response, next: NextFunction) => {  
 try {  
 const searchOptions = req.body;  
 const results = await knowledgeBaseService.searchMaterials(searchOptions);  
 res.json(results);  
 } catch (error) {  
 next(error);  
 }  
 };  
 ```

1. Test the endpoint  
   ``typescript  
    // In packages/server/tests/routes/material.routes.test.ts  
    describe('POST /materials/search', () => {  
    it('should return search results for valid query', async () => {  
    const response = await request(app)  
    .post('/materials/search')  
    .send({ query: 'ceramic', materialType: 'tile' })  
    .set('Authorization',Bearer ${testToken}`);  
   expect(response.status).toBe(200);  
    expect(response.body.materials).toBeInstanceOf(Array);  
    });  
    });  
    ```

Test the endpoint  
``typescript  
 // In packages/server/tests/routes/material.routes.test.ts  
 describe('POST /materials/search', () => {  
 it('should return search results for valid query', async () => {  
 const response = await request(app)  
 .post('/materials/search')  
 .send({ query: 'ceramic', materialType: 'tile' })  
 .set('Authorization',Bearer ${testToken}`);

``typescript  
 // In packages/server/tests/routes/material.routes.test.ts  
 describe('POST /materials/search', () => {  
 it('should return search results for valid query', async () => {  
 const response = await request(app)  
 .post('/materials/search')  
 .send({ query: 'ceramic', materialType: 'tile' })  
 .set('Authorization',

expect(response.status).toBe(200);  
 expect(response.body.materials).toBeInstanceOf(Array);  
 });  
 });  
 ```

##### Creating a New React Component

1. Create the component file  
    ```tsx  
    // In packages/client/src/components/MaterialCard.tsx  
    import React from 'react';

interface MaterialCardProps {  
 id: string;  
 name: string;  
 thumbnailUrl: string;  
 manufacturer: string;  
 onClick?: () => void;  
 }

export const MaterialCard: React.FC = ({  
 id,  
 name,  
 thumbnailUrl,  
 manufacturer,  
 onClick  
 }) => {  
 return (  
   
  
  
{name}  
{manufacturer}  
  
  
 );  
 };  
 ```

#### {name}

{manufacturer}

1. Create component tests  
    ```tsx  
    // In packages/client/src/components/tests/MaterialCard.test.tsx  
    import React from 'react';  
    import { render, screen, fireEvent } from '@testing-library/react';  
    import { MaterialCard } from '../MaterialCard';

describe('MaterialCard', () => {  
 it('renders material information correctly', () => {  
 render(  
   
 );

expect(screen.getByText('Ceramic Tile')).toBeInTheDocument();  
 expect(screen.getByText('Example Tiles Inc.')).toBeInTheDocument();  
 expect(screen.getByAltText('Ceramic Tile')).toHaveAttribute('src', '/example.jpg');  
 });  
  
 it('calls onClick when clicked', () => {  
 const handleClick = jest.fn();  
 render(  
 <MaterialCard  
 id="material-123"  
 name="Ceramic Tile"  
 thumbnailUrl="/example.jpg"  
 manufacturer="Example Tiles Inc."  
 onClick={handleClick}  
 />  
 );  
  
 fireEvent.click(screen.getByText('Ceramic Tile'));  
 expect(handleClick).toHaveBeenCalledTimes(1);  
 });

expect(screen.getByText('Ceramic Tile')).toBeInTheDocument();  
 expect(screen.getByText('Example Tiles Inc.')).toBeInTheDocument();  
 expect(screen.getByAltText('Ceramic Tile')).toHaveAttribute('src', '/example.jpg');  
 });  
  
 it('calls onClick when clicked', () => {  
 const handleClick = jest.fn();  
 render(  
 <MaterialCard  
 id="material-123"  
 name="Ceramic Tile"  
 thumbnailUrl="/example.jpg"  
 manufacturer="Example Tiles Inc."  
 onClick={handleClick}  
 />  
 );  
  
 fireEvent.click(screen.getByText('Ceramic Tile'));  
 expect(handleClick).toHaveBeenCalledTimes(1);  
 });

});  
 ```

##### Adding a New ML Model

1. Prepare the model code  
    ```python  
    # In packages/ml/python/models/texture\_classifier.py  
    import tensorflow as tf  
    from tensorflow.keras import layers, models

def create\_texture\_classifier(input\_shape=(224, 224, 3), num\_classes=10):  
 """Create a CNN model for texture classification."""  
 model = models.Sequential([  
 layers.Conv2D(32, (3, 3), activation='relu', input\_shape=input\_shape),  
 layers.MaxPooling2D((2, 2)),  
 layers.Conv2D(64, (3, 3), activation='relu'),  
 layers.MaxPooling2D((2, 2)),  
 layers.Conv2D(128, (3, 3), activation='relu'),  
 layers.MaxPooling2D((2, 2)),  
 layers.Conv2D(128, (3, 3), activation='relu'),  
 layers.MaxPooling2D((2, 2)),  
 layers.Flatten(),  
 layers.Dense(512, activation='relu'),  
 layers.Dropout(0.5),  
 layers.Dense(num\_classes, activation='softmax')  
 ])

model.compile(  
 optimizer='adam',  
 loss='categorical\_crossentropy',  
 metrics=['accuracy']  
 )  
  
 return model

model.compile(  
 optimizer='adam',  
 loss='categorical\_crossentropy',  
 metrics=['accuracy']  
 )  
  
 return model

```

1. Create a model trainer  
    ```python  
    # In packages/ml/python/trainers/texture\_trainer.py  
    import os  
    import numpy as np  
    import tensorflow as tf  
    from models.texture\_classifier import create\_texture\_classifier

def train\_texture\_classifier(dataset\_path, output\_dir, epochs=10, batch\_size=32):  
 """Train the texture classifier on a dataset."""  
 # Load and preprocess data  
 # ...

# Create model  
 model = create\_texture\_classifier(num\_classes=len(class\_names))  
  
 # Train the model  
 history = model.fit(  
 train\_dataset,  
 validation\_data=validation\_dataset,  
 epochs=epochs,  
 callbacks=[  
 tf.keras.callbacks.ModelCheckpoint(  
 filepath=os.path.join(output\_dir, 'checkpoints'),  
 save\_best\_only=True  
 ),  
 tf.keras.callbacks.TensorBoard(  
 log\_dir=os.path.join(output\_dir, 'logs')  
 )  
 ]  
 )  
  
 # Save the model  
 model.save(os.path.join(output\_dir, 'texture\_classifier.h5'))  
  
 # Save class names  
 with open(os.path.join(output\_dir, 'class\_names.txt'), 'w') as f:  
 f.write('\n'.join(class\_names))  
  
 return {  
 'model\_path': os.path.join(output\_dir, 'texture\_classifier.h5'),  
 'class\_names': class\_names,  
 'accuracy': float(history.history['val\_accuracy'][-1]),  
 'loss': float(history.history['val\_loss'][-1])  
 }

# Create model  
 model = create\_texture\_classifier(num\_classes=len(class\_names))  
  
 # Train the model  
 history = model.fit(  
 train\_dataset,  
 validation\_data=validation\_dataset,  
 epochs=epochs,  
 callbacks=[  
 tf.keras.callbacks.ModelCheckpoint(  
 filepath=os.path.join(output\_dir, 'checkpoints'),  
 save\_best\_only=True  
 ),  
 tf.keras.callbacks.TensorBoard(  
 log\_dir=os.path.join(output\_dir, 'logs')  
 )  
 ]  
 )  
  
 # Save the model  
 model.save(os.path.join(output\_dir, 'texture\_classifier.h5'))  
  
 # Save class names  
 with open(os.path.join(output\_dir, 'class\_names.txt'), 'w') as f:  
 f.write('\n'.join(class\_names))  
  
 return {  
 'model\_path': os.path.join(output\_dir, 'texture\_classifier.h5'),  
 'class\_names': class\_names,  
 'accuracy': float(history.history['val\_accuracy'][-1]),  
 'loss': float(history.history['val\_loss'][-1])  
 }

```

1. Create a model serving endpoint  
    ```python  
    # In packages/ml/python/server.py (add route)  
    @app.route('/api/classify-texture', methods=['POST'])  
    def classify\_texture():  
    if 'image' not in request.files:  
    return jsonify({'error': 'No image provided'}), 400  
   image\_file = request.files['image']  
    img = load\_and\_preprocess\_image(image\_file)  
   # Load the model (cached)  
    model = get\_texture\_classifier()  
   # Make prediction  
    predictions = model.predict(np.expand\_dims(img, axis=0))[0]  
    class\_names = get\_texture\_class\_names()  
   # Format results  
    results = [  
    {'class': class\_name, 'confidence': float(confidence)}  
    for class\_name, confidence in zip(class\_names, predictions)  
    ]  
   # Sort by confidence  
    results.sort(key=lambda x: x['confidence'], reverse=True)  
   return jsonify({  
    'results': results[:5], # Top 5 predictions  
    'processingTimeMs': int((time.time() - start\_time) \* 1000)  
    })  
    ```
2. Create TypeScript interface  
    ```typescript  
    // In packages/ml/src/index.ts (add interface)  
    export interface TextureClassificationResult {  
    results: Array<{  
    class: string;  
    confidence: number;  
    }>;  
    processingTimeMs: number;  
    }

Create a model serving endpoint  
 ```python  
 # In packages/ml/python/server.py (add route)  
 @app.route('/api/classify-texture', methods=['POST'])  
 def classify\_texture():  
 if 'image' not in request.files:  
 return jsonify({'error': 'No image provided'}), 400

image\_file = request.files['image']  
 img = load\_and\_preprocess\_image(image\_file)

# Load the model (cached)  
 model = get\_texture\_classifier()

# Make prediction  
 predictions = model.predict(np.expand\_dims(img, axis=0))[0]  
 class\_names = get\_texture\_class\_names()

# Format results  
 results = [  
 {'class': class\_name, 'confidence': float(confidence)}  
 for class\_name, confidence in zip(class\_names, predictions)  
 ]

# Sort by confidence  
 results.sort(key=lambda x: x['confidence'], reverse=True)

return jsonify({  
 'results': results[:5], # Top 5 predictions  
 'processingTimeMs': int((time.time() - start\_time) \* 1000)  
 })  
 ```

Create TypeScript interface  
 ```typescript  
 // In packages/ml/src/index.ts (add interface)  
 export interface TextureClassificationResult {  
 results: Array<{  
 class: string;  
 confidence: number;  
 }>;  
 processingTimeMs: number;  
 }

export async function classifyTexture(  
 imagePath: string  
 ): Promise {  
 const formData = new FormData();  
 formData.append('image', fs.createReadStream(imagePath));

const response = await axios.post<TextureClassificationResult>(  
 `${ML\_API\_URL}/api/classify-texture`,  
 formData,  
 {  
 headers: {  
 'Content-Type': 'multipart/form-data'  
 }  
 }  
 );  
  
 return response.data;

const response = await axios.post<TextureClassificationResult>(  
 `${ML\_API\_URL}/api/classify-texture`,  
 formData,  
 {  
 headers: {  
 'Content-Type': 'multipart/form-data'  
 }  
 }  
 );  
  
 return response.data;

}  
 ```

### Ongoing Maintenance and Updates

Once your initial deployment is complete, the system is designed for easy maintenance and updates.

#### Updating the Application

With the automated GitHub Actions workflow, updating the KAI platform is straightforward:

1. Code Changes:
2. Make your changes in a feature branch
3. Create a pull request to the main branch
4. Review and merge the pull request
5. Automatic Deployment:
6. The GitHub Actions workflow automatically triggers when changes are pushed to the main branch
7. The workflow:  
     
   Runs tests  
   Builds new Docker images  
   Updates Kubernetes deployments  
   Deploys frontend apps to Vercel
8. Manual Deployment:
9. You can also trigger a deployment manually:  
   Go to the "Actions" tab in your GitHub repository  
   Select the "KAI Platform CI/CD Pipeline" workflow  
   Click "Run workflow"  
   Choose the environment (staging or production)  
   Optionally select "Create new cluster if not exists"

Review and merge the pull request

Automatic Deployment:

The workflow:

* Runs tests
* Builds new Docker images
* Updates Kubernetes deployments
* Deploys frontend apps to Vercel

Manual Deployment:

* Go to the "Actions" tab in your GitHub repository
* Select the "KAI Platform CI/CD Pipeline" workflow
* Click "Run workflow"
* Choose the environment (staging or production)
* Optionally select "Create new cluster if not exists"

#### Deployment Workflow

The deployment process follows these steps:

1. Code is pushed to the main branch (for production) or staging branch (for staging)
2. GitHub Actions workflow is triggered
3. Build and test job runs to verify code quality
4. Docker images are built for all services
5. Infrastructure is checked and provisioned if needed
6. Kubernetes components are set up (cert-manager, NGINX, Argo)
7. Application is deployed using Helm charts
8. Frontend is deployed to Vercel
9. Deployment is verified with comprehensive health checks
10. Notification is sent upon completion

#### Feature-Specific Deployment Notes

##### Post-Deployment Process

The deployment process includes a post-deployment phase that runs after database migrations are applied. This phase is used for tasks that need to be performed after the application is deployed and the database schema is updated, such as:

* Registering modules with the subscription system
* Registering API endpoints with the network access control system
* Initializing default data
* Updating configuration settings

The post-deployment process is implemented as a Kubernetes job that runs the run-post-deployment.js script. This script executes all necessary post-deployment tasks defined in post-deployment.ts.

run-post-deployment.js

post-deployment.ts

How to Add New Post-Deployment Tasks:

1. Add your task to the runPostDeploymentTasks function in packages/server/src/scripts/post-deployment.ts
2. The task will be automatically executed during the deployment process

runPostDeploymentTasks

packages/server/src/scripts/post-deployment.ts

Example: Material Promotion System

The Material Promotion System uses the post-deployment process to:  
- Register the materialPromotion module with the subscription system  
- Register the material promotion API endpoints with the network access control system

materialPromotion

After deployment, you need to enable the materialPromotion module for the appropriate factory subscription tiers through the admin panel.

materialPromotion

#### Scaling the Application

The KAI platform can be scaled in several ways:

1. Horizontal Pod Autoscaling:
2. The platform uses Kubernetes Horizontal Pod Autoscalers (HPAs) to automatically scale based on CPU and memory usage
3. You can adjust the HPA settings in the Helm values files
4. Manual Scaling:  
    ```bash  
    # Scale API server  
    kubectl scale deployment api-server -n kai-system --replicas=5

You can adjust the HPA settings in the Helm values files

Manual Scaling:  
 ```bash  
 # Scale API server  
 kubectl scale deployment api-server -n kai-system --replicas=5

# Scale coordinator service  
 kubectl scale deployment coordinator-service -n kai-system --replicas=3

# Scale notification service  
 kubectl scale deployment notification-service -n kai-system --replicas=3  
 ```

1. Node Pool Scaling:
2. In the Digital Ocean dashboard:  
     
   Navigate to your Kubernetes cluster  
   Select the node pool you want to scale  
   Click "Edit" and adjust the number of nodes
3. Using the command line:  
    ```bash  
    # Get the cluster name  
    doctl kubernetes cluster list  
   # Get node pool ID  
    doctl kubernetes cluster node-pool list your-cluster-name  
   # Scale the node pool  
    doctl kubernetes cluster node-pool update your-cluster-name node-pool-id --count=5  
    ```
4. Autoscaling Node Pools:
5. Digital Ocean supports node pool autoscaling
6. Enable autoscaling when creating the cluster or update existing node pools:  
    bash  
    doctl kubernetes cluster node-pool update your-cluster-name node-pool-id --auto-scale --min-nodes=2 --max-nodes=5

In the Digital Ocean dashboard:

* Navigate to your Kubernetes cluster
* Select the node pool you want to scale
* Click "Edit" and adjust the number of nodes

Using the command line:  
 ```bash  
 # Get the cluster name  
 doctl kubernetes cluster list

# Get node pool ID  
 doctl kubernetes cluster node-pool list your-cluster-name

# Scale the node pool  
 doctl kubernetes cluster node-pool update your-cluster-name node-pool-id --count=5  
 ```

Autoscaling Node Pools:

bash  
 doctl kubernetes cluster node-pool update your-cluster-name node-pool-id --auto-scale --min-nodes=2 --max-nodes=5

#### Backup and Disaster Recovery

The KAI platform includes several backup and disaster recovery mechanisms:

1. Database Backups:
2. MongoDB Atlas: If using MongoDB Atlas, configure automated backups:  
     
   Daily snapshots with 7-day retention  
   Point-in-time recovery  
   Periodic exports to S3 for long-term storage
3. Self-hosted MongoDB: Use the backup job included in the deployment:  
    ```bash  
    # Check backup job status  
    kubectl get cronjobs -n kai-system  
   # Trigger a manual backup  
    kubectl create job --from=cronjob/mongodb-backup manual-backup-$(date +%s) -n kai-system  
    ```
4. Kubernetes State Backups:
5. The deployment script automatically creates backups before applying changes
6. Backups are stored in ./kubernetes/backups/<environment>/<timestamp>/
7. To restore from a backup:  
    bash  
    ./kubernetes/deploy.sh --context=your-context --env=production --rollback=20250412153022
8. Disaster Recovery Procedure:

MongoDB Atlas: If using MongoDB Atlas, configure automated backups:

* Daily snapshots with 7-day retention
* Point-in-time recovery
* Periodic exports to S3 for long-term storage

Self-hosted MongoDB: Use the backup job included in the deployment:  
 ```bash  
 # Check backup job status  
 kubectl get cronjobs -n kai-system

# Trigger a manual backup  
 kubectl create job --from=cronjob/mongodb-backup manual-backup-$(date +%s) -n kai-system  
 ```

Kubernetes State Backups:

./kubernetes/backups/<environment>/<timestamp>/

To restore from a backup:  
 bash  
 ./kubernetes/deploy.sh --context=your-context --env=production --rollback=20250412153022

bash  
 ./kubernetes/deploy.sh --context=your-context --env=production --rollback=20250412153022

Disaster Recovery Procedure:

In case of a major failure:

a. Assess the Situation:  
 - Identify the affected components  
 - Check logs and monitoring data

b. Restore Database:  
 - Restore the most recent MongoDB backup  
 - Verify data integrity

c. Rebuild Infrastructure:  
 - If the cluster is compromised, create a new one:  
 bash  
 # Trigger manual workflow with "create\_cluster=true"  
 # Go to GitHub Actions → KAI Platform CI/CD Pipeline → Run workflow

bash  
 # Trigger manual workflow with "create\_cluster=true"  
 # Go to GitHub Actions → KAI Platform CI/CD Pipeline → Run workflow

d. Restore Application State:  
 - Deploy the last known good version:  
 ```bash  
 # Find the last successful deployment  
 git log --oneline

# Checkout that commit  
 git checkout <commit-hash>  
  
 # Trigger deployment  
 git push -f origin HEAD:main  
 ```

# Checkout that commit  
 git checkout <commit-hash>  
  
 # Trigger deployment  
 git push -f origin HEAD:main  
 ```

e. Verify Recovery:  
 - Check all services are running  
 - Verify API endpoints  
 - Test frontend functionality  
 - Validate data consistency

1. High Availability Configuration:
2. The production environment is configured for high availability:  
     
   Multiple replicas for all critical services  
   Pod Disruption Budgets to ensure minimum availability  
   Anti-affinity rules to distribute pods across nodes  
   Readiness and liveness probes for automatic recovery
3. Kubernetes State Backup:
4. Use Velero for Kubernetes cluster backups
5. Configure regular backups to S3
6. Supabase Backup:
7. Enable automatic backups in the Supabase dashboard
8. Schedule regular database exports

The production environment is configured for high availability:

* Multiple replicas for all critical services
* Pod Disruption Budgets to ensure minimum availability
* Anti-affinity rules to distribute pods across nodes
* Readiness and liveness probes for automatic recovery

Kubernetes State Backup:

Configure regular backups to S3

Supabase Backup:

### Troubleshooting

This section provides solutions for common issues you might encounter during deployment or operation of the KAI platform.

#### Deployment Issues

##### GitHub Actions Workflow Failures

1. Build Failures:
2. Check the build logs for specific error messages
3. Verify that all dependencies are correctly specified
4. Ensure that tests are passing locally before pushing
5. Docker Image Build Failures:
6. Verify Docker registry credentials
7. Check for disk space issues in the GitHub runner
8. Ensure Dockerfiles are correctly formatted
9. Deployment Timeouts:
10. Increase the timeout values in the workflow
11. Check if the cluster is under heavy load
12. Verify network connectivity to the cluster

Ensure that tests are passing locally before pushing

Docker Image Build Failures:

Ensure Dockerfiles are correctly formatted

Deployment Timeouts:

#### Vercel Deployment Issues

1. Build Failures:
2. Check the Vercel build logs
3. Verify environment variables are correctly set
4. Ensure the project configuration is correct
5. Domain Configuration Issues:
6. Verify DNS records are correctly configured
7. Check SSL certificate issuance
8. Ensure custom domains are properly set up in Vercel
9. Runtime Errors:
10. Check the browser console for errors
11. Verify that API calls are properly configured with the correct URL
12. Check CORS configurations on the backend

Ensure the project configuration is correct

Domain Configuration Issues:

Ensure custom domains are properly set up in Vercel

Runtime Errors:

#### Supabase Issues

1. Authentication Problems:
2. Check the Site URL in Supabase authentication settings
3. Verify the anon key is correctly set in frontend apps
4. Check CORS configurations
5. Realtime Connection Issues:
6. Ensure the publication is properly configured
7. Check that the realtime service is enabled
8. Verify WebSocket connections in the browser console
9. Database Connection Issues:
10. Verify connection string format
11. Check network access rules
12. Ensure the database is running and accessible

Check CORS configurations

Realtime Connection Issues:

Verify WebSocket connections in the browser console

Database Connection Issues:

#### Kubernetes Issues

##### Pod Startup Failures

1. ImagePullBackOff:
2. Verify Docker registry credentials
3. Check image name and tag
4. Ensure the image exists in the registry

```bash  
 # Check pod status  
 kubectl get pods -n kai-system

# Describe the failing pod  
 kubectl describe pod -n kai-system

# Check image pull secrets  
 kubectl get secrets -n kai-system  
 ```

1. CrashLoopBackOff:
2. Check container logs
3. Verify environment variables
4. Check for resource constraints

```bash  
 # Get logs from the failing pod  
 kubectl logs -n kai-system

# Check events  
 kubectl get events -n kai-system --sort-by='.lastTimestamp'  
 ```

1. Pending Pods:
2. Check for resource constraints
3. Verify node pool availability
4. Check for taints and tolerations

```bash  
 # Check node status  
 kubectl get nodes

# Describe the node  
 kubectl describe node   
 ```

##### Service Connectivity Issues

1. Service Not Accessible:
2. Verify service is running
3. Check endpoints
4. Verify network policies

```bash  
 # Check service  
 kubectl get svc -n kai-system

# Check endpoints  
 kubectl get endpoints -n kai-system

# Test connectivity from within the cluster  
 kubectl run -it --rm debug --image=curlimages/curl --restart=Never -- curl http://service-name.kai-system  
 ```

1. Ingress Issues:
2. Verify ingress controller is running
3. Check ingress resource configuration
4. Verify SSL certificate

```bash  
 # Check ingress  
 kubectl get ingress -n kai-system

# Describe ingress  
 kubectl describe ingress -n kai-system

# Check ingress controller logs  
 kubectl logs -n kai-system -l app=nginx-ingress-ingress-nginx-controller  
 ```

##### Argo Workflow Issues

1. Workflow Failures:
2. Check workflow status
3. Examine workflow logs
4. Verify service account permissions

```bash  
 # Check workflow status  
 kubectl get workflows -n kai-system

# Get workflow details  
 kubectl get workflow -n kai-system -o yaml

# Check pod logs for workflow step  
 kubectl logs -n kai-system   
 ```

1. Workflow Stuck in Pending:
2. Check for PVC issues
3. Verify resource availability
4. Check for node selector issues

```bash  
 # Check PVCs  
 kubectl get pvc -n kai-system

# Check resource quotas  
 kubectl describe resourcequota -n kai-system  
 ```

#### SSL Certificate Issues

1. Certificate Not Issued:
2. Verify cert-manager is running
3. Check certificate resource
4. Check DNS configuration

```bash  
 # Check cert-manager pods  
 kubectl get pods -n cert-manager

# Check certificate status  
 kubectl get certificate -n kai-system

# Check certificate request  
 kubectl get certificaterequest -n kai-system

# Check challenge  
 kubectl get challenge -n kai-system  
 ```

1. Certificate Renewal Failures:
2. Check cert-manager logs
3. Verify ACME account registration
4. Check for rate limiting

```bash  
 # Check cert-manager logs  
 kubectl logs -n cert-manager -l app=cert-manager

# Check ACME account  
 kubectl get secret -n cert-manager letsencrypt-prod  
 ```

#### Docker Issues

1. Build Failures:
2. Check for syntax errors in Dockerfiles
3. Verify that the build context doesn't include large or unnecessary files
4. Check for connection issues to registries
5. Container Startup Issues:
6. Check container logs: docker logs <container-id>
7. Verify that environment variables are correctly set
8. Check for permission issues on mounted volumes
9. Image Size Issues:
10. Use multi-stage builds to reduce final image size
11. Minimize the number of RUN instructions
12. Clean up package caches in the same layer they're created

Check for connection issues to registries

Container Startup Issues:

docker logs <container-id>

Check for permission issues on mounted volumes

Image Size Issues:

#### GitHub Container Registry Integration

The KAI Platform CI/CD pipeline can push Docker images to both your existing Docker registry and GitHub Container Registry (ghcr.io), providing redundancy and leveraging GitHub's integrated container registry features.

##### Benefits of GitHub Container Registry

* Tight integration with GitHub repositories
* Simplified authentication using GitHub tokens
* Free storage for public repositories
* Improved security with GitHub's vulnerability scanning

##### Configuration for GitHub Container Registry

1. Required GitHub Secrets:
2. GITHUB\_TOKEN - This is automatically provided by GitHub Actions, but it needs the correct permissions
3. Permissions Setup:
4. Go to your repository on GitHub
5. Navigate to Settings > Actions > General
6. Scroll down to "Workflow permissions"
7. Select "Read and write permissions"
8. Save the changes

GITHUB\_TOKEN - This is automatically provided by GitHub Actions, but it needs the correct permissions

GITHUB\_TOKEN

Permissions Setup:

##### Image Naming Convention

Images pushed to GitHub Container Registry follow this naming convention:

ghcr.io/{owner}/{repository}/kai-{service-name}:{tag}

ghcr.io/{owner}/{repository}/kai-{service-name}:{tag}

For example:

ghcr.io/your-org/kai/kai-api-server:latest

ghcr.io/your-org/kai/kai-api-server:latest

##### Using GitHub Container Registry Images

In Kubernetes:

1. Create a Kubernetes secret with your GitHub credentials:

kubectl create secret docker-registry github-container-registry \  
 --docker-server=ghcr.io \  
 --docker-username=YOUR\_GITHUB\_USERNAME \  
 --docker-password=YOUR\_GITHUB\_TOKEN \  
 --docker-email=YOUR\_EMAIL

kubectl create secret docker-registry github-container-registry \  
 --docker-server=ghcr.io \  
 --docker-username=YOUR\_GITHUB\_USERNAME \  
 --docker-password=YOUR\_GITHUB\_TOKEN \  
 --docker-email=YOUR\_EMAIL

1. Reference this secret in your pod specifications:

spec:  
 imagePullSecrets:  
 - name: github-container-registry  
 containers:  
 - name: app  
 image: ghcr.io/your-org/kai/kai-api-server:latest

spec:  
 imagePullSecrets:  
 - name: github-container-registry  
 containers:  
 - name: app  
 image: ghcr.io/your-org/kai/kai-api-server:latest

In Docker Compose:

services:  
 api-server:  
 image: ghcr.io/your-org/kai/kai-api-server:latest

services:  
 api-server:  
 image: ghcr.io/your-org/kai/kai-api-server:latest

And authenticate with Docker before pulling:

echo $GITHUB\_TOKEN | docker login ghcr.io -u USERNAME --password-stdin

echo $GITHUB\_TOKEN | docker login ghcr.io -u USERNAME --password-stdin

##### Troubleshooting GitHub Container Registry

If you encounter issues with GitHub Container Registry:

1. Verify that your GitHub token has the correct permissions
2. Check that the repository visibility settings allow for the package visibility you want
3. Ensure you're properly authenticated when pulling images
4. Check the GitHub Actions logs for specific error messages

#### Performance Issues

1. High CPU/Memory Usage:
2. Check resource usage
3. Identify resource-intensive pods
4. Consider scaling up resources

```bash  
 # Check resource usage  
 kubectl top pods -n kai-system

# Check node resource usage  
 kubectl top nodes  
 ```

1. Slow API Responses:
2. Check for database bottlenecks
3. Verify network latency
4. Check for resource constraints

```bash  
 # Check API server logs  
 kubectl logs -n kai-system -l app=api-server

# Check database metrics  
 # (Requires Prometheus and MongoDB exporter)  
 ```

### Performance Optimization

#### Backend Optimization

1. Database Optimization:
2. Create appropriate indexes for common queries
3. Use projection to return only needed fields
4. Implement caching for frequent queries
5. Use aggregation pipeline for complex queries
6. API Server Optimization:
7. Implement response compression
8. Use efficient JSON serialization
9. Implement request throttling for high-traffic endpoints
10. Use connection pooling for database connections

Use aggregation pipeline for complex queries

API Server Optimization:

#### Frontend Optimization

1. React Performance:
2. Use React.memo for pure components
3. Implement useMemo and useCallback hooks appropriately
4. Use virtualization for long lists (react-window)
5. Implement code splitting for large components
6. Asset Optimization:
7. Optimize images (compression, WebP format)
8. Implement lazy loading for images
9. Use CSS minification
10. Implement critical CSS loading

Implement code splitting for large components

Asset Optimization:

#### ML Service Optimization

1. Model Optimization:
2. Use model quantization to reduce size
3. Implement batching for multiple requests
4. Use TensorRT for GPU acceleration
5. Optimize preprocessing pipeline
6. Deployment Optimization:
7. Use TensorFlow Serving for model serving
8. Implement model caching in memory
9. Use GPU instances for inference
10. Implement input/output pipelines

Optimize preprocessing pipeline

Deployment Optimization:

# Digital Ocean Kubernetes Setup

Source: readme/digital-ocean-kubernetes-setup.md

---

## Digital Ocean Kubernetes Setup Guide for Kai

This guide provides detailed instructions for setting up a Kubernetes cluster on Digital Ocean specifically optimized for the Kai application.

### Table of Contents

* Prerequisites
* Creating a Digital Ocean Kubernetes Cluster
* Node Pool Configuration
* Networking Setup
* Storage Configuration
* Monitoring and Logging
* Cost Optimization
* Maintenance and Upgrades

### Prerequisites

Before starting, you'll need:

* A Digital Ocean account with billing set up
* doctl CLI installed and configured
* kubectl installed locally
* Domain name(s) with ability to configure DNS settings

doctl

kubectl

### Creating a Digital Ocean Kubernetes Cluster

#### Step 1: Log in to the Digital Ocean Console

Visit https://cloud.digitalocean.com/login and log in to your Digital Ocean account.

#### Step 2: Create a Kubernetes Cluster

1. Navigate to the Kubernetes section in the left menu
2. Click Create Cluster
3. Choose a datacenter region (select the region closest to your users)
4. Select Kubernetes version (recommended: latest stable version)

#### Step 3: Configure the Initial Node Pool

For a production Kai deployment, we recommend:

API Server Node Pool:  
- Machine Type: Standard Droplets   
- Node Size: At least 4GB RAM / 2 vCPUs ($24/month)  
- Number of Nodes: 3 (for high availability)  
- Name: api-server-pool

api-server-pool

#### Step 4: Add Specialized Node Pools

After creating the initial node pool, add specialized node pools for the different workload types:

1. CPU-Optimized Node Pool (for the Coordinator service and general processing):
2. Click Add Node Pool
3. Machine Type: CPU-Optimized Droplets
4. Node Size: 8GB RAM / 4 vCPUs ($48/month)
5. Number of Nodes: 3
6. Name: cpu-optimized-pool
7. GPU Node Pool (for ML inference tasks like NeRF generation):
8. Click Add Node Pool
9. Machine Type: GPU Droplets
10. Node Size: With NVIDIA T4 GPUs
11. Number of Nodes: 2
12. Name: gpu-optimized-pool
13. Memory-Optimized Node Pool (for large model loading):
14. Click Add Node Pool
15. Machine Type: Memory-Optimized Droplets
16. Node Size: 16GB RAM / 4 vCPUs
17. Number of Nodes: 1
18. Name: memory-optimized-pool
19. Spot Instance Pool (for cost-effective batch processing):
20. Click Add Node Pool
21. Enable Spot Instances
22. Machine Type: Standard Droplets
23. Node Size: 8GB RAM / 4 vCPUs
24. Number of Nodes: 2-4
25. Name: spot-instances-pool

Name: cpu-optimized-pool

cpu-optimized-pool

GPU Node Pool (for ML inference tasks like NeRF generation):

Name: gpu-optimized-pool

gpu-optimized-pool

Memory-Optimized Node Pool (for large model loading):

Name: memory-optimized-pool

memory-optimized-pool

Spot Instance Pool (for cost-effective batch processing):

spot-instances-pool

#### Step 5: Cluster Configuration

1. Name your cluster (e.g., kai-production)
2. Select VPC Network (default is fine for most deployments)
3. Click Create Cluster

kai-production

The cluster creation process takes approximately 5-10 minutes.

### Node Pool Configuration

#### Adding Labels to Node Pools

It's important to add Kubernetes labels to your node pools to control pod scheduling:

# Label the API server nodes  
kubectl label nodes -l doks.digitalocean.com/node-pool=api-server-pool node-type=api-server  
  
# Label the CPU-optimized nodes for coordinator service  
kubectl label nodes -l doks.digitalocean.com/node-pool=cpu-optimized-pool node-type=cpu-optimized workload-class=orchestration  
  
# Label the GPU nodes  
kubectl label nodes -l doks.digitalocean.com/node-pool=gpu-optimized-pool node-type=gpu-optimized workload-class=ml-inference gpu=nvidia-t4  
  
# Label the memory-optimized nodes  
kubectl label nodes -l doks.digitalocean.com/node-pool=memory-optimized-pool node-type=memory-optimized workload-class=model-loading  
  
# Label the spot instances  
kubectl label nodes -l doks.digitalocean.com/node-pool=spot-instances-pool node-type=spot-instance workload-class=batch-processing

# Label the API server nodes  
kubectl label nodes -l doks.digitalocean.com/node-pool=api-server-pool node-type=api-server  
  
# Label the CPU-optimized nodes for coordinator service  
kubectl label nodes -l doks.digitalocean.com/node-pool=cpu-optimized-pool node-type=cpu-optimized workload-class=orchestration  
  
# Label the GPU nodes  
kubectl label nodes -l doks.digitalocean.com/node-pool=gpu-optimized-pool node-type=gpu-optimized workload-class=ml-inference gpu=nvidia-t4  
  
# Label the memory-optimized nodes  
kubectl label nodes -l doks.digitalocean.com/node-pool=memory-optimized-pool node-type=memory-optimized workload-class=model-loading  
  
# Label the spot instances  
kubectl label nodes -l doks.digitalocean.com/node-pool=spot-instances-pool node-type=spot-instance workload-class=batch-processing

#### Node Selectors in Deployments

Update your deployments to use node selectors:

API Server Deployment:

spec:  
 template:  
 spec:  
 nodeSelector:  
 node-type: api-server

spec:  
 template:  
 spec:  
 nodeSelector:  
 node-type: api-server

Coordinator Service Deployment:

spec:  
 template:  
 spec:  
 nodeSelector:  
 node-type: cpu-optimized  
 workload-class: orchestration

spec:  
 template:  
 spec:  
 nodeSelector:  
 node-type: cpu-optimized  
 workload-class: orchestration

Argo Workflow Steps:  
Different workflow steps can target specific node pools:

# GPU-intensive steps (in the WorkflowTemplate)  
spec:  
 nodeSelector:  
 node-type: gpu-optimized  
 workload-class: ml-inference  
  
# Memory-intensive steps  
spec:  
 nodeSelector:  
 node-type: memory-optimized  
 workload-class: model-loading  
  
# Batch processing steps that can tolerate interruption  
spec:  
 nodeSelector:  
 node-type: spot-instance  
 workload-class: batch-processing

# GPU-intensive steps (in the WorkflowTemplate)  
spec:  
 nodeSelector:  
 node-type: gpu-optimized  
 workload-class: ml-inference  
  
# Memory-intensive steps  
spec:  
 nodeSelector:  
 node-type: memory-optimized  
 workload-class: model-loading  
  
# Batch processing steps that can tolerate interruption  
spec:  
 nodeSelector:  
 node-type: spot-instance  
 workload-class: batch-processing

#### Resource Limits and Requests

Set appropriate resource limits and requests for your deployments:

API Server Deployment (per pod):

resources:  
 requests:  
 memory: "512Mi"  
 cpu: "250m"  
 limits:  
 memory: "1Gi"  
 cpu: "500m"

resources:  
 requests:  
 memory: "512Mi"  
 cpu: "250m"  
 limits:  
 memory: "1Gi"  
 cpu: "500m"

Coordinator Service Deployment (per pod):

resources:  
 requests:  
 memory: "1Gi"  
 cpu: "500m"  
 limits:  
 memory: "2Gi"  
 cpu: "1000m"

resources:  
 requests:  
 memory: "1Gi"  
 cpu: "500m"  
 limits:  
 memory: "2Gi"  
 cpu: "1000m"

ML Workflow Steps (per pod, example for GPU steps):

resources:  
 requests:  
 memory: "4Gi"  
 cpu: "1000m"  
 nvidia.com/gpu: "1"  
 limits:  
 memory: "8Gi"  
 cpu: "2000m"  
 nvidia.com/gpu: "1"

resources:  
 requests:  
 memory: "4Gi"  
 cpu: "1000m"  
 nvidia.com/gpu: "1"  
 limits:  
 memory: "8Gi"  
 cpu: "2000m"  
 nvidia.com/gpu: "1"

### Networking Setup

#### Configure Load Balancer and Ingress

Digital Ocean Kubernetes automatically provisions a Load Balancer when you create an ingress resource.

1. Install NGINX Ingress Controller:  
    bash  
    kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.7.0/deploy/static/provider/cloud/deploy.yaml
2. Wait for the Load Balancer to be provisioned:  
    bash  
    kubectl get service -n ingress-nginx ingress-nginx-controller
3. Configure DNS Records:
4. Get the Load Balancer's external IP
5. Create DNS A records:  
   api.kai.yourdomain.com -> [Load Balancer IP]

Install NGINX Ingress Controller:  
 bash  
 kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.7.0/deploy/static/provider/cloud/deploy.yaml

bash  
 kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.7.0/deploy/static/provider/cloud/deploy.yaml

Wait for the Load Balancer to be provisioned:  
 bash  
 kubectl get service -n ingress-nginx ingress-nginx-controller

bash  
 kubectl get service -n ingress-nginx ingress-nginx-controller

Configure DNS Records:

* api.kai.yourdomain.com -> [Load Balancer IP]

api.kai.yourdomain.com

#### TLS/SSL Setup with cert-manager

1. Install cert-manager:  
    bash  
    kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.11.0/cert-manager.yaml
2. Create a ClusterIssuer for Let's Encrypt:  
    ```bash  
    cat <<EOF | kubectl apply -f -  
    apiVersion: cert-manager.io/v1  
    kind: ClusterIssuer  
    metadata:  
    name: letsencrypt-prod  
    spec:  
    acme:  
    server: https://acme-v02.api.letsencrypt.org/directory  
    email: your-email@example.com  
    privateKeySecretRef:  
    name: letsencrypt-prod  
    solvers:  
     
   http01:  
    ingress:  
    class: nginx  
    EOF  
    ```

Install cert-manager:  
 bash  
 kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.11.0/cert-manager.yaml

bash  
 kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.11.0/cert-manager.yaml

Create a ClusterIssuer for Let's Encrypt:  
 ```bash  
 cat <<EOF | kubectl apply -f -  
 apiVersion: cert-manager.io/v1  
 kind: ClusterIssuer  
 metadata:  
 name: letsencrypt-prod  
 spec:  
 acme:  
 server: https://acme-v02.api.letsencrypt.org/directory  
 email: your-email@example.com  
 privateKeySecretRef:  
 name: letsencrypt-prod  
 solvers:

* http01:  
   ingress:  
   class: nginx  
   EOF  
   ```

### Installing Argo Workflows

Argo Workflows is a critical component for the KAI ML Platform, handling the orchestration of complex ML pipelines.

#### Step 1: Install Argo Workflows

1. Create the argo namespace:  
    bash  
    kubectl create namespace argo
2. Install Argo Workflows using kubectl:  
    bash  
    kubectl apply -n argo -f https://github.com/argoproj/argo-workflows/releases/download/v3.4.5/install.yaml
3. Configure Argo to use the default service account in the namespace:  
    bash  
    kubectl patch configmap/workflow-controller-configmap \  
    -n argo \  
    --type merge \  
    -p '{"data":{"workflowNamespaces":"kai,argo"}}'

Create the argo namespace:  
 bash  
 kubectl create namespace argo

bash  
 kubectl create namespace argo

Install Argo Workflows using kubectl:  
 bash  
 kubectl apply -n argo -f https://github.com/argoproj/argo-workflows/releases/download/v3.4.5/install.yaml

bash  
 kubectl apply -n argo -f https://github.com/argoproj/argo-workflows/releases/download/v3.4.5/install.yaml

Configure Argo to use the default service account in the namespace:  
 bash  
 kubectl patch configmap/workflow-controller-configmap \  
 -n argo \  
 --type merge \  
 -p '{"data":{"workflowNamespaces":"kai,argo"}}'

bash  
 kubectl patch configmap/workflow-controller-configmap \  
 -n argo \  
 --type merge \  
 -p '{"data":{"workflowNamespaces":"kai,argo"}}'

#### Step 2: Create RBAC for Argo Workflows in the kai-ml namespace

Create the necessary RBAC configuration for Argo Workflows to run in the kai-ml namespace:

kai-ml

cat <<EOF | kubectl apply -f -  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: argo-workflow  
 namespace: kai-ml  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
 name: workflow-role  
 namespace: kai-ml  
rules:  
- apiGroups:  
 - ""  
 resources:  
 - pods  
 - pods/exec  
 - pods/log  
 verbs:  
 - create  
 - get  
 - list  
 - watch  
 - update  
 - patch  
 - delete  
- apiGroups:  
 - ""  
 resources:  
 - configmaps  
 - secrets  
 - persistentvolumeclaims  
 verbs:  
 - create  
 - get  
 - list  
 - watch  
 - update  
 - patch  
 - delete  
- apiGroups:  
 - argoproj.io  
 resources:  
 - workflows  
 - workflows/finalizers  
 - workflowtasksets  
 - workflowtasksets/finalizers  
 - workflowtemplates  
 - workflowtemplates/finalizers  
 verbs:  
 - create  
 - get  
 - list  
 - watch  
 - update  
 - patch  
 - delete  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: RoleBinding  
metadata:  
 name: workflow-rolebinding  
 namespace: kai-ml  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: Role  
 name: workflow-role  
subjects:  
- kind: ServiceAccount  
 name: argo-workflow  
 namespace: kai-ml  
EOF

cat <<EOF | kubectl apply -f -  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: argo-workflow  
 namespace: kai-ml  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
 name: workflow-role  
 namespace: kai-ml  
rules:  
- apiGroups:  
 - ""  
 resources:  
 - pods  
 - pods/exec  
 - pods/log  
 verbs:  
 - create  
 - get  
 - list  
 - watch  
 - update  
 - patch  
 - delete  
- apiGroups:  
 - ""  
 resources:  
 - configmaps  
 - secrets  
 - persistentvolumeclaims  
 verbs:  
 - create  
 - get  
 - list  
 - watch  
 - update  
 - patch  
 - delete  
- apiGroups:  
 - argoproj.io  
 resources:  
 - workflows  
 - workflows/finalizers  
 - workflowtasksets  
 - workflowtasksets/finalizers  
 - workflowtemplates  
 - workflowtemplates/finalizers  
 verbs:  
 - create  
 - get  
 - list  
 - watch  
 - update  
 - patch  
 - delete  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: RoleBinding  
metadata:  
 name: workflow-rolebinding  
 namespace: kai-ml  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: Role  
 name: workflow-role  
subjects:  
- kind: ServiceAccount  
 name: argo-workflow  
 namespace: kai-ml  
EOF

#### Step 3: Create RBAC for the Coordinator Service

The Coordinator service needs permissions to create and manage Argo Workflows:

cat <<EOF | kubectl apply -f -  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: coordinator-service-account  
 namespace: kai-ml  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
 name: coordinator-workflow-manager  
 namespace: kai-ml  
rules:  
- apiGroups:  
 - argoproj.io  
 resources:  
 - workflows  
 - workflows/finalizers  
 - workflowtemplates  
 - workflowtemplates/finalizers  
 verbs:  
 - create  
 - delete  
 - get  
 - list  
 - patch  
 - update  
 - watch  
- apiGroups:  
 - ""  
 resources:  
 - pods  
 - pods/log  
 - configmaps  
 - secrets  
 verbs:  
 - get  
 - list  
 - watch  
- apiGroups:  
 - ""  
 resources:  
 - persistentvolumeclaims  
 verbs:  
 - create  
 - delete  
 - get  
 - list  
 - watch  
 - update  
 - patch  
- apiGroups:  
 - ""  
 resources:  
 - events  
 verbs:  
 - create  
 - get  
 - list  
 - watch  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: RoleBinding  
metadata:  
 name: coordinator-workflow-manager-binding  
 namespace: kai-ml  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: Role  
 name: coordinator-workflow-manager  
subjects:  
- kind: ServiceAccount  
 name: coordinator-service-account  
 namespace: kai-ml  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRole  
metadata:  
 name: coordinator-cluster-monitor  
rules:  
- apiGroups:  
 - ""  
 resources:  
 - nodes  
 verbs:  
 - get  
 - list  
 - watch  
- apiGroups:  
 - metrics.k8s.io  
 resources:  
 - pods  
 - nodes  
 verbs:  
 - get  
 - list  
 - watch  
- apiGroups:  
 - custom.metrics.k8s.io  
 resources:  
 - "\*"  
 verbs:  
 - get  
 - list  
 - watch  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
 name: coordinator-cluster-monitor-binding  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: ClusterRole  
 name: coordinator-cluster-monitor  
subjects:  
- kind: ServiceAccount  
 name: coordinator-service-account  
 namespace: kai-ml  
EOF

cat <<EOF | kubectl apply -f -  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: coordinator-service-account  
 namespace: kai-ml  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
 name: coordinator-workflow-manager  
 namespace: kai-ml  
rules:  
- apiGroups:  
 - argoproj.io  
 resources:  
 - workflows  
 - workflows/finalizers  
 - workflowtemplates  
 - workflowtemplates/finalizers  
 verbs:  
 - create  
 - delete  
 - get  
 - list  
 - patch  
 - update  
 - watch  
- apiGroups:  
 - ""  
 resources:  
 - pods  
 - pods/log  
 - configmaps  
 - secrets  
 verbs:  
 - get  
 - list  
 - watch  
- apiGroups:  
 - ""  
 resources:  
 - persistentvolumeclaims  
 verbs:  
 - create  
 - delete  
 - get  
 - list  
 - watch  
 - update  
 - patch  
- apiGroups:  
 - ""  
 resources:  
 - events  
 verbs:  
 - create  
 - get  
 - list  
 - watch  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: RoleBinding  
metadata:  
 name: coordinator-workflow-manager-binding  
 namespace: kai-ml  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: Role  
 name: coordinator-workflow-manager  
subjects:  
- kind: ServiceAccount  
 name: coordinator-service-account  
 namespace: kai-ml  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRole  
metadata:  
 name: coordinator-cluster-monitor  
rules:  
- apiGroups:  
 - ""  
 resources:  
 - nodes  
 verbs:  
 - get  
 - list  
 - watch  
- apiGroups:  
 - metrics.k8s.io  
 resources:  
 - pods  
 - nodes  
 verbs:  
 - get  
 - list  
 - watch  
- apiGroups:  
 - custom.metrics.k8s.io  
 resources:  
 - "\*"  
 verbs:  
 - get  
 - list  
 - watch  
  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
 name: coordinator-cluster-monitor-binding  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: ClusterRole  
 name: coordinator-cluster-monitor  
subjects:  
- kind: ServiceAccount  
 name: coordinator-service-account  
 namespace: kai-ml  
EOF

#### Step 4: Configure Artifact Repository

Create a secret for accessing your S3-compatible storage:

kubectl create secret generic s3-artifact-repository \  
 --namespace kai-ml \  
 --from-literal=accessKey=YOUR\_ACCESS\_KEY \  
 --from-literal=secretKey=YOUR\_SECRET\_KEY

kubectl create secret generic s3-artifact-repository \  
 --namespace kai-ml \  
 --from-literal=accessKey=YOUR\_ACCESS\_KEY \  
 --from-literal=secretKey=YOUR\_SECRET\_KEY

Configure an artifact repository for Argo to store input/output artifacts:

cat <<EOF | kubectl apply -f -  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: artifact-repositories  
 namespace: kai-ml  
data:  
 s3: |  
 s3:  
 bucket: kai-workflow-artifacts  
 endpoint: nyc3.digitaloceanspaces.com  
 insecure: false  
 accessKeySecret:  
 name: s3-artifact-repository  
 key: accessKey  
 secretKeySecret:  
 name: s3-artifact-repository  
 key: secretKey  
 region: us-east-1  
EOF

cat <<EOF | kubectl apply -f -  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: artifact-repositories  
 namespace: kai-ml  
data:  
 s3: |  
 s3:  
 bucket: kai-workflow-artifacts  
 endpoint: nyc3.digitaloceanspaces.com  
 insecure: false  
 accessKeySecret:  
 name: s3-artifact-repository  
 key: accessKey  
 secretKeySecret:  
 name: s3-artifact-repository  
 key: secretKey  
 region: us-east-1  
EOF

### Storage Configuration

#### Digital Ocean Block Storage for Persistent Data

1. Create a StorageClass for Digital Ocean Block Storage:  
    bash  
    cat <<EOF | kubectl apply -f -  
    apiVersion: storage.k8s.io/v1  
    kind: StorageClass  
    metadata:  
    name: do-block-storage  
    annotations:  
    storageclass.kubernetes.io/is-default-class: "true"  
    provisioner: dobs.csi.digitalocean.com  
    parameters:  
    fsType: ext4  
    reclaimPolicy: Retain  
    allowVolumeExpansion: true  
    EOF
2. Create a StorageClass for workflow data (with Delete reclaim policy):  
    bash  
    cat <<EOF | kubectl apply -f -  
    apiVersion: storage.k8s.io/v1  
    kind: StorageClass  
    metadata:  
    name: workflow-storage  
    provisioner: dobs.csi.digitalocean.com  
    parameters:  
    fsType: ext4  
    reclaimPolicy: Delete  
    allowVolumeExpansion: true  
    EOF
3. Create PVCs for shared resources:  
    ```bash  
    cat <<EOF | kubectl apply -f -  
    apiVersion: v1  
    kind: PersistentVolumeClaim  
    metadata:  
    name: coordinator-cache-pvc  
    namespace: kai-ml  
    spec:  
    accessModes:  
     
   ReadWriteOnce  
    resources:  
    requests:  
    storage: 20Gi  
    storageClassName: do-block-storage

Create a StorageClass for Digital Ocean Block Storage:  
 bash  
 cat <<EOF | kubectl apply -f -  
 apiVersion: storage.k8s.io/v1  
 kind: StorageClass  
 metadata:  
 name: do-block-storage  
 annotations:  
 storageclass.kubernetes.io/is-default-class: "true"  
 provisioner: dobs.csi.digitalocean.com  
 parameters:  
 fsType: ext4  
 reclaimPolicy: Retain  
 allowVolumeExpansion: true  
 EOF

bash  
 cat <<EOF | kubectl apply -f -  
 apiVersion: storage.k8s.io/v1  
 kind: StorageClass  
 metadata:  
 name: do-block-storage  
 annotations:  
 storageclass.kubernetes.io/is-default-class: "true"  
 provisioner: dobs.csi.digitalocean.com  
 parameters:  
 fsType: ext4  
 reclaimPolicy: Retain  
 allowVolumeExpansion: true  
 EOF

Create a StorageClass for workflow data (with Delete reclaim policy):  
 bash  
 cat <<EOF | kubectl apply -f -  
 apiVersion: storage.k8s.io/v1  
 kind: StorageClass  
 metadata:  
 name: workflow-storage  
 provisioner: dobs.csi.digitalocean.com  
 parameters:  
 fsType: ext4  
 reclaimPolicy: Delete  
 allowVolumeExpansion: true  
 EOF

bash  
 cat <<EOF | kubectl apply -f -  
 apiVersion: storage.k8s.io/v1  
 kind: StorageClass  
 metadata:  
 name: workflow-storage  
 provisioner: dobs.csi.digitalocean.com  
 parameters:  
 fsType: ext4  
 reclaimPolicy: Delete  
 allowVolumeExpansion: true  
 EOF

Create PVCs for shared resources:  
 ```bash  
 cat <<EOF | kubectl apply -f -  
 apiVersion: v1  
 kind: PersistentVolumeClaim  
 metadata:  
 name: coordinator-cache-pvc  
 namespace: kai-ml  
 spec:  
 accessModes:

* ReadWriteOnce  
   resources:  
   requests:  
   storage: 20Gi  
   storageClassName: do-block-storage

apiVersion: v1  
 kind: PersistentVolumeClaim  
 metadata:  
 name: model-repository-pvc  
 namespace: kai-ml  
 spec:  
 accessModes:  
 - ReadWriteMany  
 resources:  
 requests:  
 storage: 50Gi  
 storageClassName: do-block-storage  
 EOF  
 ```

1. Volume handling in Argo Workflows:  
    Argo Workflows will use the workflow-storage StorageClass for its volume claim templates:

workflow-storage

yaml  
 # In the WorkflowTemplate  
 volumeClaimTemplates:  
 - metadata:  
 name: workdir  
 spec:  
 accessModes: ["ReadWriteOnce"]  
 storageClassName: workflow-storage  
 resources:  
 requests:  
 storage: 10Gi

yaml  
 # In the WorkflowTemplate  
 volumeClaimTemplates:  
 - metadata:  
 name: workdir  
 spec:  
 accessModes: ["ReadWriteOnce"]  
 storageClassName: workflow-storage  
 resources:  
 requests:  
 storage: 10Gi

### Monitoring and Logging

#### Setting Up DigitalOcean Monitoring

DigitalOcean provides built-in monitoring capabilities:

1. Navigate to your cluster in the DigitalOcean console
2. Go to the Insights tab
3. View metrics for:
4. CPU Usage
5. Memory Usage
6. Disk I/O
7. Network Traffic

#### Deploy the Monitoring Stack

The KAI ML Platform uses a comprehensive monitoring stack including Prometheus, Grafana, and Jaeger:

1. Create a monitoring namespace:  
    bash  
    kubectl create namespace monitoring
2. Add the Prometheus Helm repository:  
    bash  
    helm repo add prometheus-community https://prometheus-community.github.io/helm-charts  
    helm repo update
3. Install Prometheus and Grafana:  
    bash  
    helm install prometheus prometheus-community/kube-prometheus-stack \  
    --namespace monitoring \  
    --set grafana.adminPassword=your-secure-password \  
    --values - <<EOF  
    grafana:  
    dashboards:  
    default:  
    kubernetes-dashboard:  
    url: https://raw.githubusercontent.com/dotdc/grafana-dashboards-kubernetes/master/dashboards/k8s-views-global.json  
    ml-workflows-dashboard:  
    url: https://raw.githubusercontent.com/argoproj/argo-workflows/master/examples/grafana-dashboard.json  
    dashboardProviders:  
    dashboardproviders.yaml:  
    apiVersion: 1  
    providers:  
    - name: 'default'  
    orgId: 1  
    folder: ''  
    type: file  
    disableDeletion: false  
    editable: true  
    options:  
    path: /var/lib/grafana/dashboards/default  
    EOF
4. Install Jaeger for distributed tracing:  
    ```bash  
    kubectl apply -f https://github.com/jaegertracing/jaeger-operator/releases/download/v1.37.0/jaeger-operator.yaml

Create a monitoring namespace:  
 bash  
 kubectl create namespace monitoring

bash  
 kubectl create namespace monitoring

Add the Prometheus Helm repository:  
 bash  
 helm repo add prometheus-community https://prometheus-community.github.io/helm-charts  
 helm repo update

bash  
 helm repo add prometheus-community https://prometheus-community.github.io/helm-charts  
 helm repo update

Install Prometheus and Grafana:  
 bash  
 helm install prometheus prometheus-community/kube-prometheus-stack \  
 --namespace monitoring \  
 --set grafana.adminPassword=your-secure-password \  
 --values - <<EOF  
 grafana:  
 dashboards:  
 default:  
 kubernetes-dashboard:  
 url: https://raw.githubusercontent.com/dotdc/grafana-dashboards-kubernetes/master/dashboards/k8s-views-global.json  
 ml-workflows-dashboard:  
 url: https://raw.githubusercontent.com/argoproj/argo-workflows/master/examples/grafana-dashboard.json  
 dashboardProviders:  
 dashboardproviders.yaml:  
 apiVersion: 1  
 providers:  
 - name: 'default'  
 orgId: 1  
 folder: ''  
 type: file  
 disableDeletion: false  
 editable: true  
 options:  
 path: /var/lib/grafana/dashboards/default  
 EOF

bash  
 helm install prometheus prometheus-community/kube-prometheus-stack \  
 --namespace monitoring \  
 --set grafana.adminPassword=your-secure-password \  
 --values - <<EOF  
 grafana:  
 dashboards:  
 default:  
 kubernetes-dashboard:  
 url: https://raw.githubusercontent.com/dotdc/grafana-dashboards-kubernetes/master/dashboards/k8s-views-global.json  
 ml-workflows-dashboard:  
 url: https://raw.githubusercontent.com/argoproj/argo-workflows/master/examples/grafana-dashboard.json  
 dashboardProviders:  
 dashboardproviders.yaml:  
 apiVersion: 1  
 providers:  
 - name: 'default'  
 orgId: 1  
 folder: ''  
 type: file  
 disableDeletion: false  
 editable: true  
 options:  
 path: /var/lib/grafana/dashboards/default  
 EOF

Install Jaeger for distributed tracing:  
 ```bash  
 kubectl apply -f https://github.com/jaegertracing/jaeger-operator/releases/download/v1.37.0/jaeger-operator.yaml

# Wait for the operator to be ready, then create Jaeger instance  
 kubectl apply -f - <<EOF  
 apiVersion: jaegertracing.io/v1  
 kind: Jaeger  
 metadata:  
 name: jaeger  
 namespace: monitoring  
 spec:  
 strategy: production  
 storage:  
 type: elasticsearch  
 options:  
 es:  
 server-urls: http://elasticsearch:9200  
 ingress:  
 enabled: true  
 hosts:  
 - jaeger.yourdomain.com  
 tls:  
 - hosts:  
 - jaeger.yourdomain.com  
 EOF  
 ```

1. Configure monitoring access:  
    ```bash  
    # Create Ingress for Grafana  
    kubectl apply -f - <<EOF  
    apiVersion: networking.k8s.io/v1  
    kind: Ingress  
    metadata:  
    name: grafana  
    namespace: monitoring  
    annotations:  
    kubernetes.io/ingress.class: nginx  
    cert-manager.io/cluster-issuer: letsencrypt-prod  
    nginx.ingress.kubernetes.io/ssl-redirect: "true"  
    spec:  
    tls:  
   hosts:  
   grafana.yourdomain.com  
    secretName: grafana-tls  
    rules:  
   host: grafana.yourdomain.com  
    http:  
    paths:  
   path: /  
    pathType: Prefix  
    backend:  
    service:  
    name: prometheus-grafana  
    port:  
    number: 80  
    EOF  
    ```

* hosts:
* grafana.yourdomain.com  
   secretName: grafana-tls  
   rules:
* host: grafana.yourdomain.com  
   http:  
   paths:  
  path: /  
   pathType: Prefix  
   backend:  
   service:  
   name: prometheus-grafana  
   port:  
   number: 80  
   EOF  
   ```
* path: /  
   pathType: Prefix  
   backend:  
   service:  
   name: prometheus-grafana  
   port:  
   number: 80  
   EOF  
   ```

#### Setting Up Centralized Logging

1. Deploy Loki stack for log management:  
    ```bash  
    helm repo add grafana https://grafana.github.io/helm-charts  
    helm repo update

helm install loki grafana/loki-stack \  
 --namespace monitoring \  
 --set grafana.enabled=false  
 ```

1. Configure Promtail to send logs to Loki:  
    bash  
    helm install promtail grafana/promtail \  
    --namespace monitoring \  
    --set "loki.serviceName=loki"
2. Import the Loki data source in Grafana:
3. Go to Grafana > Configuration > Data Sources
4. Add Loki with URL: http://loki:3100

Configure Promtail to send logs to Loki:  
 bash  
 helm install promtail grafana/promtail \  
 --namespace monitoring \  
 --set "loki.serviceName=loki"

bash  
 helm install promtail grafana/promtail \  
 --namespace monitoring \  
 --set "loki.serviceName=loki"

Import the Loki data source in Grafana:

### Cost Optimization

#### Autoscaling Configuration

Set up the Kubernetes Horizontal Pod Autoscaler (HPA) to automatically scale your workloads:

1. Create an HPA for the API server:  
    bash  
    kubectl autoscale deployment kai-api-server -n kai-ml \  
    --cpu-percent=75 \  
    --min=3 \  
    --max=10
2. Create an HPA for the Coordinator service:  
    bash  
    kubectl autoscale deployment coordinator-service -n kai-ml \  
    --cpu-percent=70 \  
    --min=3 \  
    --max=6

Create an HPA for the API server:  
 bash  
 kubectl autoscale deployment kai-api-server -n kai-ml \  
 --cpu-percent=75 \  
 --min=3 \  
 --max=10

bash  
 kubectl autoscale deployment kai-api-server -n kai-ml \  
 --cpu-percent=75 \  
 --min=3 \  
 --max=10

Create an HPA for the Coordinator service:  
 bash  
 kubectl autoscale deployment coordinator-service -n kai-ml \  
 --cpu-percent=70 \  
 --min=3 \  
 --max=6

bash  
 kubectl autoscale deployment coordinator-service -n kai-ml \  
 --cpu-percent=70 \  
 --min=3 \  
 --max=6

#### Node Pool Autoscaling

Enable node pool autoscaling:

1. In the DigitalOcean console, go to your Kubernetes cluster
2. Select the node pool you want to autoscale
3. Click Edit and enable autoscaling
4. Set minimum and maximum node count:
5. API Server Pool: Min 3, Max 6
6. CPU-Optimized Pool: Min 3, Max 6
7. GPU-Optimized Pool: Min 1, Max 4
8. Memory-Optimized Pool: Min 1, Max 3
9. Spot Instances Pool: Min 0, Max 6 (can scale to zero when not needed)

#### Resource Quota Management

Set up resource quotas to limit resource consumption:

cat <<EOF | kubectl apply -f -  
apiVersion: v1  
kind: ResourceQuota  
metadata:  
 name: kai-quota  
 namespace: kai-ml  
spec:  
 hard:  
 requests.cpu: "8"  
 requests.memory: 16Gi  
 limits.cpu: "16"  
 limits.memory: 32Gi  
 persistentvolumeclaims: "10"  
EOF

cat <<EOF | kubectl apply -f -  
apiVersion: v1  
kind: ResourceQuota  
metadata:  
 name: kai-quota  
 namespace: kai-ml  
spec:  
 hard:  
 requests.cpu: "8"  
 requests.memory: 16Gi  
 limits.cpu: "16"  
 limits.memory: 32Gi  
 persistentvolumeclaims: "10"  
EOF

### Maintenance and Upgrades

#### Regular Backup Procedures

1. Set up regular backups for persistent volumes:  
    ```bash  
    # Install Velero for backup management  
    velero install \  
    --provider aws \  
    --plugins velero/velero-plugin-for-aws:v1.5.0 \  
    --bucket kai-k8s-backups \  
    --secret-file ./credentials-velero \  
    --use-volume-snapshots=true \  
    --backup-location-config region=us-east-1 \  
    --snapshot-location-config region=us-east-1

# Create a daily backup schedule  
 velero schedule create daily-backup \  
 --schedule="0 0 \* \* \*" \  
 --include-namespaces kai-ml  
 ```

#### Backing Up Argo Workflows

For workflow state and templates:

# Backup Argo workflow templates  
kubectl get workflowtemplates -n kai-ml -o yaml > workflow-templates-backup.yaml  
  
# Backup completed workflows for reference  
kubectl get workflows -n kai-ml --field-selector status.phase=Succeeded -o yaml > completed-workflows-backup.yaml

# Backup Argo workflow templates  
kubectl get workflowtemplates -n kai-ml -o yaml > workflow-templates-backup.yaml  
  
# Backup completed workflows for reference  
kubectl get workflows -n kai-ml --field-selector status.phase=Succeeded -o yaml > completed-workflows-backup.yaml

1. Set up database backups (if using MongoDB outside the cluster):
2. Configure MongoDB Atlas backups
3. Set up periodic exports to S3

#### Kubernetes Version Upgrades

DigitalOcean makes it easy to upgrade your Kubernetes cluster:

1. In the DigitalOcean console, go to your Kubernetes cluster
2. Click on the Upgrade Cluster button when a new version is available
3. Before upgrading, ensure:
4. All your workloads have multiple replicas for zero-downtime upgrades
5. You have recent backups of all persistent data
6. You've tested the upgrade process in a staging environment

#### Node Maintenance

For node maintenance:

1. Cordon a node to prevent new pods from being scheduled:  
    bash  
    kubectl cordon <node-name>
2. Drain a node to safely evict all pods:  
    bash  
    kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data
3. After maintenance, uncordon the node:  
    bash  
    kubectl uncordon <node-name>

Cordon a node to prevent new pods from being scheduled:  
 bash  
 kubectl cordon <node-name>

bash  
 kubectl cordon <node-name>

Drain a node to safely evict all pods:  
 bash  
 kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data

bash  
 kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data

After maintenance, uncordon the node:  
 bash  
 kubectl uncordon <node-name>

bash  
 kubectl uncordon <node-name>

### Conclusion

Following this guide, you'll have a production-ready Kubernetes cluster on Digital Ocean optimized for running the Kai application. The cluster is configured with appropriate resources, monitoring, and high availability features to ensure reliable operation.

The architecture specifically supports the Argo Workflows-based orchestration of ML pipelines with specialized node pools for different workload types. This ensures efficient resource utilization, with GPU-intensive tasks running on appropriate hardware and lower-priority tasks utilizing cost-effective spot instances.

Remember to regularly monitor your cluster, maintain backups, and keep your Kubernetes version up-to-date for the best performance and security. For more detailed information about the architecture, refer to the Kubernetes Architecture Documentation.

# Documentation Consolidation

Source: readme/documentation-consolidation.md

---

## Documentation Consolidation System

This document describes the automated documentation consolidation system that creates a single DOCX file containing all documentation from the /readme/ folder.

/readme/

### Overview

The KAI platform includes an automated system that consolidates all Markdown documentation files into a single Microsoft Word (DOCX) document. This makes it easier to share comprehensive documentation with stakeholders who prefer traditional document formats over browsing multiple Markdown files.

### How It Works

The system uses a GitHub Action workflow that:

1. Runs automatically after each deployment to the main branch
2. Runs on a weekly schedule (Monday at 00:00 UTC)
3. Can be triggered manually via GitHub Actions interface

The workflow performs the following steps:

1. Checks out the repository code
2. Sets up Python environment
3. Installs required dependencies (python-docx, markdown, beautifulsoup4)
4. Creates and executes a Python script that:
5. Scans the /readme/ directory for all Markdown (.md) files
6. Converts each Markdown file to HTML and then to DOCX format
7. Combines all content into a single DOCX document with proper formatting
8. Adds a title page, table of contents, and processing summary
9. Saves the consolidated document as /readme/readme.docx
10. Commits and pushes the generated DOCX file back to the repository

/readme/

.md

/readme/readme.docx

### Generated Document Structure

The consolidated DOCX document includes:

* Title Page: Contains the title "KAI Platform Documentation" and generation timestamp
* Table of Contents: Automatically generated when opened in Microsoft Word
* Content Sections: Each Markdown file is converted to a section with:
* Section title (derived from filename)
* Source file reference
* Original content with preserved formatting (headings, lists, code blocks, etc.)
* Processing Summary: Statistics about the number of files processed

### Formatting Details

The conversion process handles various Markdown elements:

* Headings: Converted to Word headings with appropriate levels
* Lists: Converted to bulleted or numbered lists
* Code blocks: Formatted with Courier New font
* Tables: Converted to text representation
* Text formatting: Basic formatting (bold, italic) is preserved

### Usage

#### Accessing the Consolidated Documentation

The consolidated documentation is available in two ways:

1. In the repository: The file is located at /readme/readme.docx and is automatically updated after each deployment to the main branch or on the weekly schedule.
2. As a GitHub Actions artifact: Each time the workflow runs, the documentation is also uploaded as an artifact named "kai-documentation" that can be downloaded directly from the GitHub Actions interface. These artifacts are retained for 90 days.

In the repository: The file is located at /readme/readme.docx and is automatically updated after each deployment to the main branch or on the weekly schedule.

/readme/readme.docx

As a GitHub Actions artifact: Each time the workflow runs, the documentation is also uploaded as an artifact named "kai-documentation" that can be downloaded directly from the GitHub Actions interface. These artifacts are retained for 90 days.

#### Manual Triggering

To manually trigger the documentation generation:

1. Go to the GitHub repository
2. Navigate to Actions → Generate Consolidated Documentation
3. Click "Run workflow"
4. Select the branch (usually main)
5. Click "Run workflow"

### Troubleshooting

If the documentation generation fails or produces unexpected results:

1. Check the GitHub Actions logs for error messages
2. Verify that all Markdown files in the /readme/ directory are properly formatted
3. Ensure that the required dependencies are available and up-to-date

/readme/

### Future Improvements

Potential future enhancements to the documentation consolidation system:

* Support for more complex Markdown elements (nested tables, advanced formatting)
* PDF generation in addition to DOCX
* Custom styling options
* Automatic distribution of generated documentation to stakeholders
* Integration with the existing monitoring dashboard

# Domain Specific Networks

Source: readme/domain-specific-networks.md

---

## Domain-Specific Neural Networks

The Domain-Specific Neural Networks module provides specialized neural network architectures optimized for material texture analysis. These networks are designed to outperform general-purpose computer vision models by incorporating domain knowledge about material textures, patterns, and properties.

### Overview

The module implements:

1. Texture-Specific Filters - Custom convolutional filters optimized for texture patterns
2. Multi-Scale Analysis - Processing textures at different resolutions and detail levels
3. Attention Mechanisms - Self-attention optimized for texture features
4. Specialized Loss Functions - Loss functions designed for texture understanding
5. Efficient Implementation - ONNX export support for deployment

These specialized networks demonstrate superior performance on material classification, texture analysis, and attribute prediction tasks compared to generic architectures.

### Architecture

The domain-specific networks are organized into multiple components:

Domain-Specific Neural Networks  
├── Core Architectures  
│ ├── TextureNetSVD  
│ ├── MaterialTextureNet  
│ └── TensorFlow Implementations  
├── Texture-Specific Components  
│ ├── TextureAttentionModule  
│ ├── TextureGaborFilters  
│ ├── MultiScaleTextureModule  
│ ├── TextureResidualBlock  
│ └── SVDTextureExtractor  
├── Loss Functions  
│ └── TextureSpecificLoss  
└── Backbone Enhancements  
 └── TextureEnhancedBackbone

Domain-Specific Neural Networks  
├── Core Architectures  
│ ├── TextureNetSVD  
│ ├── MaterialTextureNet  
│ └── TensorFlow Implementations  
├── Texture-Specific Components  
│ ├── TextureAttentionModule  
│ ├── TextureGaborFilters  
│ ├── MultiScaleTextureModule  
│ ├── TextureResidualBlock  
│ └── SVDTextureExtractor  
├── Loss Functions  
│ └── TextureSpecificLoss  
└── Backbone Enhancements  
 └── TextureEnhancedBackbone

### Key Components

#### TextureAttentionModule

A specialized attention mechanism for texture analysis that:

* Implements multi-head self-attention focused on texture patterns
* Uses learnable positional encoding tuned for texture relationships
* Processes features at different scales with weight sharing

attention\_module = TextureAttentionModule(  
 in\_channels=256,  
 heads=8,  
 dim\_head=32,  
 dropout=0.1  
)

attention\_module = TextureAttentionModule(  
 in\_channels=256,  
 heads=8,  
 dim\_head=32,  
 dropout=0.1  
)

#### TextureGaborFilters

Texture-specific Gabor filter bank with learnable parameters:

* Optimized for detecting directional texture patterns
* Supports multiple orientations and scales
* Implements learnable parameters that adapt to different texture types

gabor\_filters = TextureGaborFilters(  
 in\_channels=3,  
 out\_channels=64,  
 kernel\_size=7,  
 num\_orientations=8,  
 num\_scales=3  
)

gabor\_filters = TextureGaborFilters(  
 in\_channels=3,  
 out\_channels=64,  
 kernel\_size=7,  
 num\_orientations=8,  
 num\_scales=3  
)

#### MultiScaleTextureModule

Processes textures at different resolutions to capture multi-scale patterns:

* Analyzes details at different scales simultaneously
* Combines information across scales with fusion convolutions
* Uses residual connections to maintain gradient flow

multi\_scale = MultiScaleTextureModule(  
 channels=128,  
 scales=[1, 2, 4] # Original, 1/2, and 1/4 resolutions  
)

multi\_scale = MultiScaleTextureModule(  
 channels=128,  
 scales=[1, 2, 4] # Original, 1/2, and 1/4 resolutions  
)

#### TextureResidualBlock

A specialized residual block that combines attention and multi-scale processing:

* Incorporates texture attention for feature refinement
* Processes features at multiple scales
* Uses standard residual connections for stable training

texture\_block = TextureResidualBlock(  
 channels=256,  
 use\_attention=True,  
 use\_multi\_scale=True  
)

texture\_block = TextureResidualBlock(  
 channels=256,  
 use\_attention=True,  
 use\_multi\_scale=True  
)

#### TextureEnhancedBackbone

Enhances standard CNN backbones with texture-specific modules:

* Adds Gabor filter layers to early processing stages
* Replaces selected residual blocks with texture-specific blocks
* Works with standard architectures like ResNet and EfficientNet

enhanced\_model = TextureEnhancedBackbone(  
 base\_model=resnet18,  
 in\_channels=3,  
 use\_gabor=True,  
 use\_texture\_blocks=True  
)

enhanced\_model = TextureEnhancedBackbone(  
 base\_model=resnet18,  
 in\_channels=3,  
 use\_gabor=True,  
 use\_texture\_blocks=True  
)

#### SVDTextureExtractor

Extracts texture features using Singular Value Decomposition:

* Divides images into patches for local texture analysis
* Applies SVD to capture principal texture directions
* Weights singular vectors by singular values for importance

texture\_extractor = SVDTextureExtractor(  
 in\_channels=512,  
 texture\_dim=32,  
 pooling\_size=4  
)

texture\_extractor = SVDTextureExtractor(  
 in\_channels=512,  
 texture\_dim=32,  
 pooling\_size=4  
)

#### TextureSpecificLoss

Specialized loss function for texture understanding:

* Combines classification loss with texture consistency loss
* Optional style loss for capturing texture patterns
* Weighted combination adaptable to different material types

texture\_loss = TextureSpecificLoss(  
 alpha=1.0, # Weight for classification loss  
 beta=0.5, # Weight for texture consistency loss  
 gamma=0.1, # Weight for style loss  
 use\_style\_loss=True  
)

texture\_loss = TextureSpecificLoss(  
 alpha=1.0, # Weight for classification loss  
 beta=0.5, # Weight for texture consistency loss  
 gamma=0.1, # Weight for style loss  
 use\_style\_loss=True  
)

### Complete Architectures

#### TextureNetSVD

A complete architecture specifically designed for material texture classification:

* Includes texture blocks, attention mechanisms, and SVD features
* Optimized end-to-end for texture understanding
* Combined global and local texture feature representation

texture\_model = TextureNetSVD(  
 num\_classes=50,  
 input\_size=224,  
 base\_channels=64,  
 num\_blocks=4,  
 texture\_dim=32  
)

texture\_model = TextureNetSVD(  
 num\_classes=50,  
 input\_size=224,  
 base\_channels=64,  
 num\_blocks=4,  
 texture\_dim=32  
)

#### MaterialTextureNet

A flexible architecture that can use various backbones with texture enhancements:

* Supports ResNet, EfficientNet, and custom backbones
* Optional texture-specific components
* Specialized loss function for texture understanding

material\_model = MaterialTextureNet(  
 num\_classes=100,  
 backbone="resnet18",  
 use\_texture\_backbone=True,  
 use\_svd\_features=True,  
 use\_texture\_loss=True,  
 pretrained=True  
)

material\_model = MaterialTextureNet(  
 num\_classes=100,  
 backbone="resnet18",  
 use\_texture\_backbone=True,  
 use\_svd\_features=True,  
 use\_texture\_loss=True,  
 pretrained=True  
)

### Framework Support

The domain-specific networks support multiple deep learning frameworks:

#### PyTorch Implementation

Primary implementation with full feature support:

# Create a PyTorch-based texture network  
model = create\_texture\_network(  
 num\_classes=50,  
 model\_type="texture\_resnet18",  
 pretrained=True,  
 export\_onnx=False  
)

# Create a PyTorch-based texture network  
model = create\_texture\_network(  
 num\_classes=50,  
 model\_type="texture\_resnet18",  
 pretrained=True,  
 export\_onnx=False  
)

#### TensorFlow Implementation

Alternative implementation for TensorFlow users:

# Create a TensorFlow-based texture network  
tf\_model = create\_tensorflow\_texture\_network(  
 num\_classes=50,  
 model\_type="texture\_mobilenet",  
 pretrained=True  
)

# Create a TensorFlow-based texture network  
tf\_model = create\_tensorflow\_texture\_network(  
 num\_classes=50,  
 model\_type="texture\_mobilenet",  
 pretrained=True  
)

### Performance Characteristics

The domain-specific networks show significant improvements over general-purpose models:

Table content:

Model Type | Accuracy | F1 Score | Training Time | Inference Speed

Standard ResNet18 | 82.3% | 0.81 | 1.0x | 1.0x

Texture-Enhanced ResNet18 | 89.7% | 0.88 | 1.2x | 0.95x

TextureNetSVD | 91.4% | 0.90 | 1.4x | 0.9x

Key performance advantages:

* Higher Accuracy: 7-9% improvement on material classification
* Better Pattern Recognition: 12% improvement on texture pattern recognition
* Fine-Grained Discrimination: 15% improvement on similar texture differentiation
* Comparable Speed: Minimal inference time increase despite additional components

### Training Optimization

The models include several optimizations for efficient training:

#### Specialized Data Augmentation

Material-type specific augmentation strategies:

* Wood: Grain direction variation, color shifts
* Metal: Reflection intensity, specularity changes
* Fabric: Thread pattern augmentation, weave variations
* Stone: Veining pattern enhancement, weathering simulation

#### Warm-Starting

Integration with Parameter Registry for efficient training:

* Uses material-specific hyperparameters from similar materials
* Pre-initializes model weights based on material type
* Adapts learning schedules to texture complexity

#### Fine-Tuning Guidance

Recommendations for fine-tuning on specific material types:

* Layer freezing strategies for transfer learning
* Learning rate schedules optimized for texture datasets
* Early stopping criteria based on texture metrics

### Deployment

The models support efficient deployment across various platforms:

#### ONNX Export

Export models to ONNX format for cross-platform deployment:

# Export model to ONNX  
model.export\_to\_onnx(  
 path="texture\_model.onnx",  
 input\_size=(224, 224)  
)

# Export model to ONNX  
model.export\_to\_onnx(  
 path="texture\_model.onnx",  
 input\_size=(224, 224)  
)

#### Mobile Optimization

Models can be optimized for mobile deployment:

* Post-training quantization to 8-bit precision
* Layer fusion for faster inference
* Channel pruning for reduced model size

#### Cloud Deployment

Ready for cloud-based inference services:

* TorchServe and TensorFlow Serving configurations
* Batch processing support for efficient resource utilization
* Dynamic batch sizing based on workload

### Integration with Other Components

The domain-specific networks integrate with other system components:

#### MultiModal Pattern Recognition

Provides visual encoders for multimodal learning:

* Feature extraction for pattern-text associations
* Shared embedding space with textual descriptions
* Cross-modal attention mechanisms

#### Parameter Registry System

Benefits from material-specific hyperparameters:

* Optimal parameters by material type
* Warm-starting for efficient training
* Continuous improvement via feedback loop

#### Vector Search System

Provides high-quality embeddings for similarity search:

* Texture-aware embeddings for material search
* Hierarchical feature representation for multi-level matching
* Attention-weighted features for focusing on important aspects

### API Reference

#### Creating Models

def create\_texture\_network(  
 num\_classes: int,  
 model\_type: str = "texture\_resnet18",  
 pretrained: bool = True,  
 export\_onnx: bool = False,  
 export\_path: Optional[str] = None  
) -> nn.Module:  
 """  
 Create a texture-specific network for material analysis  
  
 Args:  
 num\_classes: Number of material classes  
 model\_type: Model architecture type  
 pretrained: Whether to use pretrained backbone  
 export\_onnx: Whether to export to ONNX format  
 export\_path: Path to save ONNX model  
  
 Returns:  
 Initialized model  
 """

def create\_texture\_network(  
 num\_classes: int,  
 model\_type: str = "texture\_resnet18",  
 pretrained: bool = True,  
 export\_onnx: bool = False,  
 export\_path: Optional[str] = None  
) -> nn.Module:  
 """  
 Create a texture-specific network for material analysis  
  
 Args:  
 num\_classes: Number of material classes  
 model\_type: Model architecture type  
 pretrained: Whether to use pretrained backbone  
 export\_onnx: Whether to export to ONNX format  
 export\_path: Path to save ONNX model  
  
 Returns:  
 Initialized model  
 """

#### Training Models

# Training a texture-specific model  
def train\_texture\_network(  
 model: MaterialTextureNet,  
 train\_loader: DataLoader,  
 val\_loader: DataLoader,  
 num\_epochs: int = 10,  
 learning\_rate: float = 0.001,  
 device: str = "cuda",  
 checkpoint\_dir: Optional[str] = None  
) -> Dict[str, Any]:  
 """  
 Train a texture-specific network  
  
 Args:  
 model: Model to train  
 train\_loader: Training data loader  
 val\_loader: Validation data loader  
 num\_epochs: Number of training epochs  
 learning\_rate: Learning rate  
 device: Device to train on  
 checkpoint\_dir: Directory to save checkpoints  
  
 Returns:  
 Training statistics and best model path  
 """

# Training a texture-specific model  
def train\_texture\_network(  
 model: MaterialTextureNet,  
 train\_loader: DataLoader,  
 val\_loader: DataLoader,  
 num\_epochs: int = 10,  
 learning\_rate: float = 0.001,  
 device: str = "cuda",  
 checkpoint\_dir: Optional[str] = None  
) -> Dict[str, Any]:  
 """  
 Train a texture-specific network  
  
 Args:  
 model: Model to train  
 train\_loader: Training data loader  
 val\_loader: Validation data loader  
 num\_epochs: Number of training epochs  
 learning\_rate: Learning rate  
 device: Device to train on  
 checkpoint\_dir: Directory to save checkpoints  
  
 Returns:  
 Training statistics and best model path  
 """

#### Material-Specific Feature Extraction

def extract\_texture\_features(  
 model: MaterialTextureNet,  
 image: Union[str, np.ndarray],  
 layer: str = "final",  
 device: str = "cuda"  
) -> np.ndarray:  
 """  
 Extract texture features from an image  
  
 Args:  
 model: Trained model  
 image: Input image path or array  
 layer: Layer to extract features from  
 device: Device to run on  
  
 Returns:  
 Texture feature array  
 """

def extract\_texture\_features(  
 model: MaterialTextureNet,  
 image: Union[str, np.ndarray],  
 layer: str = "final",  
 device: str = "cuda"  
) -> np.ndarray:  
 """  
 Extract texture features from an image  
  
 Args:  
 model: Trained model  
 image: Input image path or array  
 layer: Layer to extract features from  
 device: Device to run on  
  
 Returns:  
 Texture feature array  
 """

### Use Cases

#### Material Classification

Classifying materials with high accuracy:

# Classify material from image  
result = model.classify(  
 image\_path="sample.jpg",  
 top\_k=3 # Return top 3 predictions  
)  
# Result: [("ceramic", 0.92), ("porcelain", 0.05), ("stone", 0.02)]

# Classify material from image  
result = model.classify(  
 image\_path="sample.jpg",  
 top\_k=3 # Return top 3 predictions  
)  
# Result: [("ceramic", 0.92), ("porcelain", 0.05), ("stone", 0.02)]

#### Texture Analysis

Analyzing texture properties of materials:

# Extract texture properties  
texture\_properties = texture\_analyzer.analyze(  
 image\_path="fabric.jpg",  
 properties=["roughness", "pattern\_type", "pattern\_scale"]  
)  
# Result: {"roughness": 0.65, "pattern\_type": "herringbone", "pattern\_scale": "medium"}

# Extract texture properties  
texture\_properties = texture\_analyzer.analyze(  
 image\_path="fabric.jpg",  
 properties=["roughness", "pattern\_type", "pattern\_scale"]  
)  
# Result: {"roughness": 0.65, "pattern\_type": "herringbone", "pattern\_scale": "medium"}

#### Material Similarity

Finding materials with similar textures:

# Find similar materials  
similar\_materials = similarity\_service.find\_similar(  
 reference\_image="reference.jpg",  
 material\_type="wood",  
 max\_results=5  
)  
# Result: List of similar wood materials with similarity scores

# Find similar materials  
similar\_materials = similarity\_service.find\_similar(  
 reference\_image="reference.jpg",  
 material\_type="wood",  
 max\_results=5  
)  
# Result: List of similar wood materials with similarity scores

#### Quality Assessment

Assessing material quality based on texture:

# Assess material quality  
quality\_score = quality\_assessor.evaluate(  
 image\_path="marble\_sample.jpg",  
 material\_type="marble",  
 criteria=["defects", "consistency", "pattern\_quality"]  
)  
# Result: Quality score and breakdown by criteria

# Assess material quality  
quality\_score = quality\_assessor.evaluate(  
 image\_path="marble\_sample.jpg",  
 material\_type="marble",  
 criteria=["defects", "consistency", "pattern\_quality"]  
)  
# Result: Quality score and breakdown by criteria

### Implementation Notes

The domain-specific networks include several practical implementation details:

#### Memory Optimization

* Gradient Checkpointing: Reduces memory usage during training
* Mixed Precision Training: Uses FP16 where appropriate
* Efficient Attention Implementation: Linear complexity attention variants

#### Numerical Stability

* SVD Fallbacks: Graceful handling of SVD computation failures
* Attention Scaling: Proper temperature scaling in attention mechanisms
* Gradient Clipping: Prevents exploding gradients with texture-specific thresholds

#### Hardware Adaptation

* Dynamic Computation Graphs: Adapts to available GPU memory
* Feature Pruning: Reduces model size based on hardware constraints
* Operation Fusion: Combines operations for efficient execution

### Future Directions

The domain-specific networks will continue to evolve with:

1. Material-Type Transformers: Specialized transformer architectures for textures
2. Few-Shot Learning: Better generalization from limited examples
3. Self-Supervised Pretraining: Texture-specific pretraining objectives
4. Neural Architecture Search: Automated discovery of optimal texture architectures
5. Graph-Based Texture Representation: Modeling textures as graphs of elements

### Command-Line Usage

The module includes command-line functionality for common operations:

# Train a texture model  
python domain\_specific\_networks.py --action train --model-type texture\_resnet18 \  
 --num-classes 50 --input path/to/dataset --output path/to/save/model \  
 --epochs 20 --batch-size 32 --learning-rate 0.001 --gpu  
  
# Export model to ONNX  
python domain\_specific\_networks.py --action export --model-type texturenet \  
 --num-classes 50 --output path/to/model.onnx --gpu

# Train a texture model  
python domain\_specific\_networks.py --action train --model-type texture\_resnet18 \  
 --num-classes 50 --input path/to/dataset --output path/to/save/model \  
 --epochs 20 --batch-size 32 --learning-rate 0.001 --gpu  
  
# Export model to ONNX  
python domain\_specific\_networks.py --action export --model-type texturenet \  
 --num-classes 50 --output path/to/model.onnx --gpu

### Dependencies

The domain-specific networks require:

* PyTorch: 1.7.0+ or TensorFlow: 2.4.0+
* NumPy: For numerical operations
* OpenCV: For image preprocessing
* tqdm: For progress tracking
* difflib: For utility functions

### References

The implementation is based on research in material texture analysis:

1. He, K., et al. "Deep Residual Learning for Image Recognition"
2. Dosovitskiy, A., et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale"
3. Vaswani, A., et al. "Attention Is All You Need"
4. Gatys, L.A., et al. "Texture Synthesis Using Convolutional Neural Networks"
5. Huang, G., et al. "Multi-Scale Dense Networks for Resource Efficient Image Classification"

# Dynamic Metadata Fields

Source: readme/dynamic-metadata-fields.md

---

## Dynamic Metadata Fields and Categories

This document explains how dynamic categories and metadata fields are implemented in the KAI platform and how they are used for OCR extraction, ML training, and other AI-related features.

### Overview

The KAI platform uses a flexible metadata system that allows administrators to define custom fields for different material types. These fields can be used to:

1. Store and display material properties in a structured way
2. Extract information from OCR text using pattern matching
3. Train ML models to recognize specific material properties
4. Provide structured data for search and filtering

### Components

#### 1. Metadata Fields

Metadata fields are defined in the MetadataField model and can be managed through the admin dashboard at /metadata-fields. Each field has:

MetadataField

/metadata-fields

* Basic Properties: name, display name, description, field type
* Validation Rules: min/max values, regex patterns, etc.
* Material Type Association: which material type(s) the field applies to
* OCR Extraction Patterns: regex patterns for extracting values from OCR text
* AI Extraction Hints: natural language hints for AI-based extraction

#### 2. Categories

Categories define the types of materials in the system and are managed through the admin dashboard. Each category can have:

* Hierarchical Structure: parent-child relationships
* Description: detailed explanation of the category
* Metadata: additional properties specific to the category

### Material-Specific Metadata Fields

A key concept in the system is that metadata fields are bound to specific material categories. This binding is crucial for:

#### 1. Database Structure

* Each metadata field has a material\_type property in the database
* The field can be associated with a specific type (tile, wood, etc.) or 'all' for common fields
* The categories array in the metadata field model allows for multiple category associations

material\_type

categories

CREATE TABLE IF NOT EXISTS public.material\_metadata\_fields (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 field\_name TEXT NOT NULL,  
 display\_name TEXT NOT NULL,  
 field\_type TEXT NOT NULL CHECK (field\_type IN ('text', 'number', 'boolean', 'dropdown')),  
 material\_type TEXT NOT NULL CHECK (material\_type IN ('tile', 'wood', 'lighting', 'furniture', 'decoration', 'all')),  
 category TEXT NOT NULL,  
 -- other fields...  
);

CREATE TABLE IF NOT EXISTS public.material\_metadata\_fields (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 field\_name TEXT NOT NULL,  
 display\_name TEXT NOT NULL,  
 field\_type TEXT NOT NULL CHECK (field\_type IN ('text', 'number', 'boolean', 'dropdown')),  
 material\_type TEXT NOT NULL CHECK (material\_type IN ('tile', 'wood', 'lighting', 'furniture', 'decoration', 'all')),  
 category TEXT NOT NULL,  
 -- other fields...  
);

#### 2. Training and Processing

* When training ML models, only metadata fields relevant to the specific material type should be used
* Different material types have different visual and physical properties requiring specialized processing
* Feature extraction should adapt based on material type

// Example of material-specific training  
async function trainModelForMaterialType(materialType: string) {  
 // Get metadata fields specific to this material type  
 const metadataFields = await getMetadataFieldsByCategory(materialType);  
  
 // Use these fields to structure training data  
 const trainingData = await prepareTrainingData(materialType, metadataFields);  
  
 // Train model using material-specific fields  
 return trainModel(trainingData, {  
 materialType,  
 fields: metadataFields.map(field => field.name)  
 });  
}

// Example of material-specific training  
async function trainModelForMaterialType(materialType: string) {  
 // Get metadata fields specific to this material type  
 const metadataFields = await getMetadataFieldsByCategory(materialType);  
  
 // Use these fields to structure training data  
 const trainingData = await prepareTrainingData(materialType, metadataFields);  
  
 // Train model using material-specific fields  
 return trainModel(trainingData, {  
 materialType,  
 fields: metadataFields.map(field => field.name)  
 });  
}

#### 3. OCR Processing

* Different material types require different extraction patterns
* For example, "thickness" has different patterns and valid ranges for tiles vs. wood
* Material type detection should be the first step in processing

// Example of material-specific OCR extraction  
async function extractMetadataFromOCR(ocrText: string, materialType: string) {  
 // Get metadata fields for this material type  
 const metadataFields = await getMetadataFieldsByCategory(materialType);  
  
 // Extract values using material-specific fields  
 const extractedValues = {};  
 for (const field of metadataFields) {  
 const extractedValue = extractValueFromOCR(field, ocrText);  
 if (extractedValue) {  
 extractedValues[field.name] = extractedValue.value;  
 }  
 }  
  
 return extractedValues;  
}

// Example of material-specific OCR extraction  
async function extractMetadataFromOCR(ocrText: string, materialType: string) {  
 // Get metadata fields for this material type  
 const metadataFields = await getMetadataFieldsByCategory(materialType);  
  
 // Extract values using material-specific fields  
 const extractedValues = {};  
 for (const field of metadataFields) {  
 const extractedValue = extractValueFromOCR(field, ocrText);  
 if (extractedValue) {  
 extractedValues[field.name] = extractedValue.value;  
 }  
 }  
  
 return extractedValues;  
}

#### 4. UI Display

* The UI should only show fields relevant to the material type being viewed or edited
* This filtering happens in components like MaterialMetadataPanel.tsx

MaterialMetadataPanel.tsx

### Integration with OCR

When OCR is performed on material documents (like catalogs or spec sheets), the system uses the extraction patterns defined in metadata fields to automatically extract relevant information:

1. OCR text is processed through the extractValueFromOCR function
2. For each metadata field, the system tries to match the defined extraction patterns
3. Extracted values are stored with confidence scores
4. Administrators can review and correct extracted values

extractValueFromOCR

Example extraction pattern for tile thickness:

(?i)thickness:?\s\*(\d+(?:\.\d+)?)\s\*mm

(?i)thickness:?\s\*(\d+(?:\.\d+)?)\s\*mm

### Integration with ML Training

The metadata fields provide structured data for ML model training:

1. Feature Engineering: Metadata fields define the features that ML models should learn to recognize
2. Training Data: Extracted and validated metadata values serve as labeled training data
3. Category-Specific Models: The system can train specialized models for different material categories
4. Hybrid Embeddings: Material categories are used to generate specialized embeddings for better search results

### Material Type Relationships

Metadata fields can be associated with specific material types (tile, wood, lighting, etc.) through the categories field. This relationship enables:

1. Type-Specific UI: Only showing relevant fields for each material type
2. Specialized Extraction: Using different extraction patterns based on material type
3. Hierarchical Properties: Inheriting properties from parent categories
4. Cross-Type Search: Finding materials with similar properties across different types

### Admin Dashboard Integration

The admin dashboard provides interfaces for managing both categories and metadata fields:

1. Category Manager: /dashboard/categories - For managing material categories
2. Metadata Field Manager: /metadata-fields - For managing metadata field definitions
3. Material Editor: Displays the appropriate metadata fields based on material type

/dashboard/categories

/metadata-fields

### Usage in Code

#### OCR Extraction

// Extract value for a metadata field from OCR text  
export function extractValueFromOCR(field: MetadataFieldDocument, ocrText: string): any {  
 if (!field.extractionPatterns || field.extractionPatterns.length === 0) {  
 return null;  
 }  
  
 // Try extraction patterns  
 for (const pattern of field.extractionPatterns) {  
 const regex = new RegExp(pattern, 'i');  
 const match = ocrText.match(regex);  
 if (match && match[1]) {  
 return {  
 value: match[1].trim(),  
 extractionMethod: 'pattern',  
 extractionPattern: pattern,  
 confidence: 0.9  
 };  
 }  
 }  
  
 return null;  
}

// Extract value for a metadata field from OCR text  
export function extractValueFromOCR(field: MetadataFieldDocument, ocrText: string): any {  
 if (!field.extractionPatterns || field.extractionPatterns.length === 0) {  
 return null;  
 }  
  
 // Try extraction patterns  
 for (const pattern of field.extractionPatterns) {  
 const regex = new RegExp(pattern, 'i');  
 const match = ocrText.match(regex);  
 if (match && match[1]) {  
 return {  
 value: match[1].trim(),  
 extractionMethod: 'pattern',  
 extractionPattern: pattern,  
 confidence: 0.9  
 };  
 }  
 }  
  
 return null;  
}

#### ML Integration

// Generate embeddings with material category context  
const embeddings = await mcpClientService.generateTextEmbedding(  
 userId,  
 text,  
 {  
 model: 'text-embedding-3-small',  
 materialCategory: material.type // Use material type for specialized embeddings  
 }  
);

// Generate embeddings with material category context  
const embeddings = await mcpClientService.generateTextEmbedding(  
 userId,  
 text,  
 {  
 model: 'text-embedding-3-small',  
 materialCategory: material.type // Use material type for specialized embeddings  
 }  
);

### Current Implementation Status

The current implementation status of metadata fields in the system:

#### Fully Implemented

* Database schema for metadata fields
* TypeScript interfaces for metadata types
* Admin UI for managing metadata fields
* Basic OCR extraction using metadata field patterns

#### Partially Implemented

* Property-specific ML model training
* Visual reference library for property recognition
* Advanced validation rules for metadata fields

#### Actively Used Metadata Fields

1. Physical Properties:
2. Size/Dimensions, Thickness, Width/Length
3. Material, Color
4. Technical Properties:
5. PEI Rating, Finish, Resistance ratings
6. Common Properties:
7. Manufacturer, Collection/Series, Product Code

### Implementation Roadmap

The following enhancements are planned for the metadata field system:

#### 1. Update ML Training Pipeline

* Modify training code to explicitly filter metadata fields by material type
* Create material-specific feature extractors for each material type
* Implement specialized training pipelines for different property types

#### 2. Enhance OCR Processing

* Update OCR pipeline to use material-specific extraction patterns
* Implement material type detection as a first step in processing
* Add context-aware extraction for complex fields

#### 3. Improve UI Components

* Ensure all UI components consistently filter metadata fields by material type
* Add material type indicators in the admin dashboard
* Implement better visualization of field relationships

#### 4. Complete Visual Reference Library

* Implement property-specific model training for all relevant fields
* Create a comprehensive dataset for training visual property recognition
* Develop a visual property browser in the admin dashboard

#### 5. Enhance ML Integration

* Implement specialized embeddings for all material types
* Develop property-specific feature extraction for all relevant fields
* Create a unified API for property-based material search

#### 6. Expand OCR Capabilities

* Add extraction patterns for all defined metadata fields
* Implement advanced context-aware extraction for complex fields
* Develop an extraction pattern testing tool in the admin dashboard

#### 7. Implement Property Relationships

* Develop the property relationship graph
* Implement property inheritance based on material type hierarchies
* Create a visual editor for property relationships

### Best Practices

1. Descriptive Names: Use clear, descriptive names for metadata fields
2. Detailed Descriptions: Provide thorough descriptions to help users understand each field
3. Extraction Patterns: Define multiple extraction patterns to handle different text formats
4. Material Type Association: Associate fields with the appropriate material types
5. Validation Rules: Define validation rules to ensure data quality
6. Material-Specific Training: Always filter metadata fields by material type when training models
7. Consistent Field Usage: Use the same field names consistently across the system

# Enhanced Cron Scheduling

Source: readme/enhanced-cron-scheduling.md

---

## Enhanced Cron Scheduling

This document describes the enhanced cron scheduling implementation for the cache warming service. The cache warming service provides a unified interface for warming and prefetching cache entries, improving performance by proactively populating the cache with frequently accessed data.

### Overview

The cache warming service now supports enhanced cron scheduling with the following features:

1. Full Cron Syntax: Support for the full cron syntax including step values, ranges, and lists
2. Timezone Support: Support for specifying the timezone for cron expressions
3. Jitter: Support for adding jitter to scheduled executions to prevent thundering herd problems
4. Backoff Strategy: Support for exponential backoff for failed executions
5. Dependencies: Support for defining dependencies between warming sources

### Enhanced Cron Parser

The enhanced cron parser supports the full cron syntax, including:

#### Step Values

Step values allow you to specify a step for a range of values. For example, \*/5 means "every 5 units".

\*/5

\*/5 \* \* \* \* # Every 5 minutes  
\* \*/2 \* \* \* # Every 2 hours

\*/5 \* \* \* \* # Every 5 minutes  
\* \*/2 \* \* \* # Every 2 hours

#### Ranges

Ranges allow you to specify a range of values. For example, 1-5 means "1 through 5".

1-5

1-5 \* \* \* \* # Minutes 1 through 5  
\* 9-17 \* \* \* # Hours 9 through 17 (9 AM to 5 PM)

1-5 \* \* \* \* # Minutes 1 through 5  
\* 9-17 \* \* \* # Hours 9 through 17 (9 AM to 5 PM)

#### Lists

Lists allow you to specify a list of values. For example, 1,3,5 means "1, 3, and 5".

1,3,5

1,3,5 \* \* \* \* # Minutes 1, 3, and 5  
\* \* \* \* 1,3,5 # Monday, Wednesday, and Friday

1,3,5 \* \* \* \* # Minutes 1, 3, and 5  
\* \* \* \* 1,3,5 # Monday, Wednesday, and Friday

#### Ranges with Steps

Ranges with steps allow you to specify a step for a range of values. For example, 1-10/2 means "every 2 units from 1 through 10".

1-10/2

1-10/2 \* \* \* \* # Minutes 1, 3, 5, 7, 9  
\* 9-17/2 \* \* \* # Hours 9, 11, 13, 15, 17 (9 AM, 11 AM, 1 PM, 3 PM, 5 PM)

1-10/2 \* \* \* \* # Minutes 1, 3, 5, 7, 9  
\* 9-17/2 \* \* \* # Hours 9, 11, 13, 15, 17 (9 AM, 11 AM, 1 PM, 3 PM, 5 PM)

### Timezone Support

The enhanced cron parser supports specifying the timezone for cron expressions. This allows you to schedule cache warming operations in a specific timezone, regardless of the server's timezone.

import { TimezoneInfo } from '@kai/shared';  
  
// Add a cache warming source with a timezone  
cacheWarming.addSource({  
 id: 'popular-products',  
 name: 'Popular Products',  
 namespace: 'products',  
 strategy: CacheWarmingStrategy.SCHEDULED,  
 schedule: '0 0 \* \* \*', // Every day at midnight  
 timezone: {  
 name: 'America/New\_York',  
 offsetMinutes: -240 // -4 hours  
 },  
 fetch: async () => {  
 // Fetch popular products  
 const products = await fetchPopularProducts();  
  
 // Return products as a record of key-value pairs  
 return products.reduce((acc, product) => {  
 acc[product.id] = product;  
 return acc;  
 }, {});  
 }  
});

import { TimezoneInfo } from '@kai/shared';  
  
// Add a cache warming source with a timezone  
cacheWarming.addSource({  
 id: 'popular-products',  
 name: 'Popular Products',  
 namespace: 'products',  
 strategy: CacheWarmingStrategy.SCHEDULED,  
 schedule: '0 0 \* \* \*', // Every day at midnight  
 timezone: {  
 name: 'America/New\_York',  
 offsetMinutes: -240 // -4 hours  
 },  
 fetch: async () => {  
 // Fetch popular products  
 const products = await fetchPopularProducts();  
  
 // Return products as a record of key-value pairs  
 return products.reduce((acc, product) => {  
 acc[product.id] = product;  
 return acc;  
 }, {});  
 }  
});

### Jitter

The enhanced cron parser supports adding jitter to scheduled executions. Jitter helps prevent the "thundering herd" problem, where many clients try to access a resource at the same time.

import { JitterOptions } from '@kai/shared';  
  
// Add a cache warming source with jitter  
cacheWarming.addSource({  
 id: 'popular-products',  
 name: 'Popular Products',  
 namespace: 'products',  
 strategy: CacheWarmingStrategy.SCHEDULED,  
 schedule: '\*/5 \* \* \* \*', // Every 5 minutes  
 jitter: {  
 enabled: true,  
 maxPercent: 0.2 // Up to 20% jitter  
 },  
 fetch: async () => {  
 // Fetch popular products  
 const products = await fetchPopularProducts();  
  
 // Return products as a record of key-value pairs  
 return products.reduce((acc, product) => {  
 acc[product.id] = product;  
 return acc;  
 }, {});  
 }  
});

import { JitterOptions } from '@kai/shared';  
  
// Add a cache warming source with jitter  
cacheWarming.addSource({  
 id: 'popular-products',  
 name: 'Popular Products',  
 namespace: 'products',  
 strategy: CacheWarmingStrategy.SCHEDULED,  
 schedule: '\*/5 \* \* \* \*', // Every 5 minutes  
 jitter: {  
 enabled: true,  
 maxPercent: 0.2 // Up to 20% jitter  
 },  
 fetch: async () => {  
 // Fetch popular products  
 const products = await fetchPopularProducts();  
  
 // Return products as a record of key-value pairs  
 return products.reduce((acc, product) => {  
 acc[product.id] = product;  
 return acc;  
 }, {});  
 }  
});

### Backoff Strategy

The enhanced cache warming service supports exponential backoff for failed executions. This helps prevent overloading the system when a warming source is failing.

import { BackoffStrategy } from '@kai/shared';  
  
// Add a cache warming source with backoff  
cacheWarming.addSource({  
 id: 'popular-products',  
 name: 'Popular Products',  
 namespace: 'products',  
 strategy: CacheWarmingStrategy.SCHEDULED,  
 schedule: '\*/5 \* \* \* \*', // Every 5 minutes  
 backoff: {  
 initialDelay: 1000, // 1 second  
 maxDelay: 60000, // 1 minute  
 factor: 2, // Double the delay each time  
 maxRetries: 5 // Maximum 5 retries  
 },  
 fetch: async () => {  
 // Fetch popular products  
 const products = await fetchPopularProducts();  
  
 // Return products as a record of key-value pairs  
 return products.reduce((acc, product) => {  
 acc[product.id] = product;  
 return acc;  
 }, {});  
 }  
});

import { BackoffStrategy } from '@kai/shared';  
  
// Add a cache warming source with backoff  
cacheWarming.addSource({  
 id: 'popular-products',  
 name: 'Popular Products',  
 namespace: 'products',  
 strategy: CacheWarmingStrategy.SCHEDULED,  
 schedule: '\*/5 \* \* \* \*', // Every 5 minutes  
 backoff: {  
 initialDelay: 1000, // 1 second  
 maxDelay: 60000, // 1 minute  
 factor: 2, // Double the delay each time  
 maxRetries: 5 // Maximum 5 retries  
 },  
 fetch: async () => {  
 // Fetch popular products  
 const products = await fetchPopularProducts();  
  
 // Return products as a record of key-value pairs  
 return products.reduce((acc, product) => {  
 acc[product.id] = product;  
 return acc;  
 }, {});  
 }  
});

### Dependencies

The enhanced cache warming service supports defining dependencies between warming sources. This ensures that dependent sources are warmed before the source itself.

// Add a cache warming source with dependencies  
cacheWarming.addSource({  
 id: 'product-recommendations',  
 name: 'Product Recommendations',  
 namespace: 'recommendations',  
 strategy: CacheWarmingStrategy.SCHEDULED,  
 schedule: '\*/10 \* \* \* \*', // Every 10 minutes  
 dependencies: ['popular-products', 'user-preferences'],  
 fetch: async () => {  
 // Fetch product recommendations  
 const recommendations = await fetchProductRecommendations();  
  
 // Return recommendations as a record of key-value pairs  
 return recommendations.reduce((acc, recommendation) => {  
 acc[recommendation.id] = recommendation;  
 return acc;  
 }, {});  
 }  
});

// Add a cache warming source with dependencies  
cacheWarming.addSource({  
 id: 'product-recommendations',  
 name: 'Product Recommendations',  
 namespace: 'recommendations',  
 strategy: CacheWarmingStrategy.SCHEDULED,  
 schedule: '\*/10 \* \* \* \*', // Every 10 minutes  
 dependencies: ['popular-products', 'user-preferences'],  
 fetch: async () => {  
 // Fetch product recommendations  
 const recommendations = await fetchProductRecommendations();  
  
 // Return recommendations as a record of key-value pairs  
 return recommendations.reduce((acc, recommendation) => {  
 acc[recommendation.id] = recommendation;  
 return acc;  
 }, {});  
 }  
});

### Improved Scheduling Algorithm

The enhanced cron parser uses a more sophisticated algorithm to determine the appropriate interval for a cron expression. It calculates the minimum interval based on the cron expression, taking into account all possible values.

For example, for the cron expression \*/5 \* \* \* \* (every 5 minutes), the algorithm calculates that the minimum interval is 5 minutes.

\*/5 \* \* \* \*

For more complex expressions like 0,15,30,45 \* \* \* \* (every 15 minutes), the algorithm calculates that the minimum interval is 15 minutes.

0,15,30,45 \* \* \* \*

### Implementation Details

#### Cron Parser

The enhanced cron parser provides the following functions:

##### parseCronToMs

Parses a cron expression into a millisecond interval, taking into account jitter.

function parseCronToMs(expression: string, jitter?: JitterOptions): number;

function parseCronToMs(expression: string, jitter?: JitterOptions): number;

##### getNextExecutionTime

Gets the next execution time for a cron expression, taking into account timezone and jitter.

function getNextExecutionTime(  
 expression: string,  
 baseTime?: Date,  
 timezone?: TimezoneInfo,  
 jitter?: JitterOptions  
): Date;

function getNextExecutionTime(  
 expression: string,  
 baseTime?: Date,  
 timezone?: TimezoneInfo,  
 jitter?: JitterOptions  
): Date;

##### isValidCronExpression

Checks if a cron expression is valid.

function isValidCronExpression(expression: string): boolean;

function isValidCronExpression(expression: string): boolean;

#### Cache Warming Service

The cache warming service has been updated to use the enhanced cron parser. The following changes have been made:

##### CacheWarmingSource Interface

The CacheWarmingSource interface has been updated to include timezone, jitter, backoff, and dependencies.

CacheWarmingSource

export interface CacheWarmingSource<T = any> {  
 /\*\* Source ID \*/  
 id: string;  
 /\*\* Source name \*/  
 name: string;  
 /\*\* Cache namespace \*/  
 namespace: string;  
 /\*\* Cache TTL in seconds \*/  
 ttl?: number;  
 /\*\* Warming strategy \*/  
 strategy: CacheWarmingStrategy;  
 /\*\* Schedule (cron expression) for scheduled warming \*/  
 schedule?: string;  
 /\*\* Timezone information for scheduled warming \*/  
 timezone?: TimezoneInfo;  
 /\*\* Jitter options for scheduled warming \*/  
 jitter?: JitterOptions;  
 /\*\* Backoff strategy for failed executions \*/  
 backoff?: BackoffStrategy;  
 /\*\* Dependencies on other sources \*/  
 dependencies?: string[];  
 /\*\* Function to fetch data \*/  
 fetch: () => Promise<Record<string, T>>;  
 /\*\* Description of the source \*/  
 description?: string;  
}

export interface CacheWarmingSource<T = any> {  
 /\*\* Source ID \*/  
 id: string;  
 /\*\* Source name \*/  
 name: string;  
 /\*\* Cache namespace \*/  
 namespace: string;  
 /\*\* Cache TTL in seconds \*/  
 ttl?: number;  
 /\*\* Warming strategy \*/  
 strategy: CacheWarmingStrategy;  
 /\*\* Schedule (cron expression) for scheduled warming \*/  
 schedule?: string;  
 /\*\* Timezone information for scheduled warming \*/  
 timezone?: TimezoneInfo;  
 /\*\* Jitter options for scheduled warming \*/  
 jitter?: JitterOptions;  
 /\*\* Backoff strategy for failed executions \*/  
 backoff?: BackoffStrategy;  
 /\*\* Dependencies on other sources \*/  
 dependencies?: string[];  
 /\*\* Function to fetch data \*/  
 fetch: () => Promise<Record<string, T>>;  
 /\*\* Description of the source \*/  
 description?: string;  
}

##### scheduleWarming Method

The scheduleWarming method has been updated to use the enhanced cron parser and handle dependencies, jitter, and backoff.

scheduleWarming

##### warmDependencies Method

A new warmDependencies method has been added to warm dependencies for a source.

warmDependencies

##### handleWarmingError Method

A new handleWarmingError method has been added to handle warming errors and implement backoff.

handleWarmingError

### Benefits

The implementation of enhanced cron scheduling provides several benefits:

1. Flexibility: Support for the full cron syntax allows for more flexible scheduling
2. Precision: Timezone support allows for more precise scheduling based on specific timezones
3. Reliability: Jitter helps prevent the "thundering herd" problem
4. Resilience: Backoff strategy helps prevent overloading the system when a warming source is failing
5. Dependency Management: Support for dependencies ensures that dependent sources are warmed before the source itself
6. Efficiency: Improved scheduling algorithm calculates the appropriate interval for a cron expression

### Next Steps

The following steps are recommended to further improve the cache warming service:

1. Add Support for More Timezone Formats: Add support for more timezone formats (e.g., IANA timezone names)
2. Add Support for More Cron Features: Add support for more cron features (e.g., L, W, #, ?)
3. Add Support for Cron Validation: Add support for validating cron expressions against specific constraints
4. Add Support for Cron Expression Builder: Add support for building cron expressions programmatically
5. Add Support for Cron Expression Parser: Add support for parsing cron expressions into human-readable descriptions
6. Add Support for Cron Expression Optimizer: Add support for optimizing cron expressions for better performance
7. Add Support for Cron Expression Migrator: Add support for migrating cron expressions between different formats

# Enhanced Material Classification

Source: readme/enhanced-material-classification.md

---

## Enhanced Material Classification

This document describes the Enhanced Material Classification feature, which provides hierarchical classification for materials, support for multiple classification systems, and mapping between different classification standards.

### Overview

The Enhanced Material Classification feature provides a comprehensive system for classifying materials using different classification standards. This enables:

1. Hierarchical Classification: Materials can be classified using hierarchical classification systems, with parent-child relationships between categories.
2. Multiple Classification Systems: Support for multiple classification standards, such as CSI MasterFormat, Uniclass, OmniClass, ASTM, and ISO.
3. Classification Mappings: Ability to map categories between different classification systems, enabling cross-standard compatibility.

### Architecture

The Enhanced Material Classification feature consists of the following components:

#### Database Schema

* classification\_systems: Stores classification systems with their codes, names, and versions.
* classification\_categories: Stores classification categories with hierarchical structure.
* material\_classifications: Stores classifications assigned to materials.
* classification\_mappings: Stores mappings between categories in different classification systems.

#### API Endpoints

The following API endpoints are available for managing material classifications:

##### Classification Systems

* GET /api/classification/systems: Get all classification systems.
* GET /api/classification/systems/:id: Get a classification system by ID.
* POST /api/classification/systems: Create a new classification system (admin only).
* PUT /api/classification/systems/:id: Update a classification system (admin only).

GET /api/classification/systems

GET /api/classification/systems/:id

POST /api/classification/systems

PUT /api/classification/systems/:id

##### Classification Categories

* GET /api/classification/categories: Get classification categories.
* GET /api/classification/categories/:id: Get a classification category by ID.
* POST /api/classification/categories: Create a new classification category (admin only).
* PUT /api/classification/categories/:id: Update a classification category (admin only).

GET /api/classification/categories

GET /api/classification/categories/:id

POST /api/classification/categories

PUT /api/classification/categories/:id

##### Material Classifications

* GET /api/classification/material-classifications: Get material classifications.
* GET /api/classification/material-classifications/:id: Get a material classification by ID.
* POST /api/classification/material-classifications: Create a new material classification.
* PUT /api/classification/material-classifications/:id: Update a material classification.
* DELETE /api/classification/material-classifications/:id: Delete a material classification.

GET /api/classification/material-classifications

GET /api/classification/material-classifications/:id

POST /api/classification/material-classifications

PUT /api/classification/material-classifications/:id

DELETE /api/classification/material-classifications/:id

##### Classification Mappings

* GET /api/classification/mappings: Get classification mappings.
* GET /api/classification/mappings/:id: Get a classification mapping by ID.
* POST /api/classification/mappings: Create a new classification mapping (admin only).
* PUT /api/classification/mappings/:id: Update a classification mapping (admin only).
* DELETE /api/classification/mappings/:id: Delete a classification mapping (admin only).

GET /api/classification/mappings

GET /api/classification/mappings/:id

POST /api/classification/mappings

PUT /api/classification/mappings/:id

DELETE /api/classification/mappings/:id

##### Additional Endpoints

* GET /api/classification/systems/:id/tree: Get a classification system with its categories as a tree.
* GET /api/classification/materials/:materialId/classifications: Get material with all its classifications.
* GET /api/classification/categories/:categoryId/equivalent: Find equivalent categories in another classification system.

GET /api/classification/systems/:id/tree

GET /api/classification/materials/:materialId/classifications

GET /api/classification/categories/:categoryId/equivalent

#### Client Components

The following client components are available for working with material classifications:

* ClassificationTree: Displays a hierarchical tree of classification categories.
* ClassificationSystemSelector: Allows users to select a classification system.
* MaterialClassificationManager: Allows users to manage classifications for a material.
* ClassificationMappingManager: Allows administrators to manage mappings between different classification systems.

#### Admin Components

The following admin components are available for managing material classifications:

* MaterialClassificationTab: Tab for managing material classifications in the material detail page.
* ClassificationManagementPage: Admin page for managing classification systems, categories, and mappings.

### Usage

#### Setting Up Classification Systems

Before using the enhanced material classification, you need to set up the classification systems you want to use. This can be done through the admin interface or by using the API.

// Example: Adding a new classification system  
const newSystem = {  
 name: 'CSI MasterFormat',  
 code: 'CSI\_MASTERFORMAT',  
 description: 'Construction Specifications Institute MasterFormat',  
 version: '2020',  
 isHierarchical: true  
};  
  
await fetch('/api/classification/systems', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(newSystem)  
});

// Example: Adding a new classification system  
const newSystem = {  
 name: 'CSI MasterFormat',  
 code: 'CSI\_MASTERFORMAT',  
 description: 'Construction Specifications Institute MasterFormat',  
 version: '2020',  
 isHierarchical: true  
};  
  
await fetch('/api/classification/systems', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(newSystem)  
});

#### Adding Classification Categories

Classification categories can be added through the admin interface or by using the API.

// Example: Adding a classification category  
const newCategory = {  
 systemId: 'system-id',  
 parentId: 'parent-category-id', // Optional, for hierarchical systems  
 code: '09 30 00',  
 name: 'Tiling',  
 description: 'Tiling materials and installation',  
 level: 2,  
 path: '09.09 30 00'  
};  
  
await fetch('/api/classification/categories', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(newCategory)  
});

// Example: Adding a classification category  
const newCategory = {  
 systemId: 'system-id',  
 parentId: 'parent-category-id', // Optional, for hierarchical systems  
 code: '09 30 00',  
 name: 'Tiling',  
 description: 'Tiling materials and installation',  
 level: 2,  
 path: '09.09 30 00'  
};  
  
await fetch('/api/classification/categories', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(newCategory)  
});

#### Classifying Materials

Materials can be classified through the material detail page or by using the API.

// Example: Adding a classification to a material  
const newClassification = {  
 materialId: 'material-id',  
 categoryId: 'category-id',  
 isPrimary: true,  
 source: 'manual'  
};  
  
await fetch('/api/classification/material-classifications', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(newClassification)  
});

// Example: Adding a classification to a material  
const newClassification = {  
 materialId: 'material-id',  
 categoryId: 'category-id',  
 isPrimary: true,  
 source: 'manual'  
};  
  
await fetch('/api/classification/material-classifications', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(newClassification)  
});

#### Creating Classification Mappings

Classification mappings can be created through the admin interface or by using the API.

// Example: Creating a mapping between categories in different systems  
const newMapping = {  
 sourceCategoryId: 'source-category-id',  
 targetCategoryId: 'target-category-id',  
 mappingType: 'exact', // 'exact', 'broader', 'narrower', 'related'  
 confidence: 0.9,  
 description: 'Exact match between CSI MasterFormat and Uniclass'  
};  
  
await fetch('/api/classification/mappings', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(newMapping)  
});

// Example: Creating a mapping between categories in different systems  
const newMapping = {  
 sourceCategoryId: 'source-category-id',  
 targetCategoryId: 'target-category-id',  
 mappingType: 'exact', // 'exact', 'broader', 'narrower', 'related'  
 confidence: 0.9,  
 description: 'Exact match between CSI MasterFormat and Uniclass'  
};  
  
await fetch('/api/classification/mappings', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(newMapping)  
});

#### Finding Equivalent Categories

You can find equivalent categories in different classification systems using the API.

// Example: Finding equivalent categories in another system  
const response = await fetch(`/api/classification/categories/${categoryId}/equivalent?targetSystemId=${targetSystemId}`, {  
 headers: {  
 'Authorization': `Bearer ${token}`  
 }  
});  
  
const data = await response.json();  
const equivalentCategories = data.equivalentCategories;

// Example: Finding equivalent categories in another system  
const response = await fetch(`/api/classification/categories/${categoryId}/equivalent?targetSystemId=${targetSystemId}`, {  
 headers: {  
 'Authorization': `Bearer ${token}`  
 }  
});  
  
const data = await response.json();  
const equivalentCategories = data.equivalentCategories;

### Integration with Other Features

#### Material Detail Page Integration

The enhanced material classification is integrated with the Material Detail Page, allowing users to manage classifications for a material.

#### Search Integration

The enhanced material classification can be integrated with the search system to enable searching for materials by classification.

#### Filtering Integration

The enhanced material classification can be integrated with the filtering system to enable filtering materials by classification.

### Best Practices

#### Hierarchical Classification

When using hierarchical classification systems, make sure to maintain the proper parent-child relationships between categories. This ensures that the classification tree is displayed correctly and that materials can be found at the appropriate level of detail.

#### Primary Classifications

Each material should have a primary classification in at least one classification system. This helps with organizing materials and enables more efficient searching and filtering.

#### Classification Mappings

When creating mappings between different classification systems, be as specific as possible about the relationship between categories. Use the appropriate mapping type (exact, broader, narrower, related) and provide a confidence score to indicate the strength of the relationship.

### Conclusion

The Enhanced Material Classification feature provides a comprehensive system for classifying materials using different classification standards. This enables hierarchical classification, support for multiple classification systems, and mapping between different standards, enhancing the organization and searchability of materials in the system.

# Enhanced Material Expert

Source: readme/enhanced-material-expert.md

---

## Enhanced Material Expert

This document provides detailed information about the Enhanced Material Expert, a specialized crewAI agent that extends the basic Material Expert with comprehensive metadata formatting capabilities.

### Overview

The Enhanced Material Expert builds upon the standard Material Expert by ensuring all material-related responses include comprehensive, well-structured metadata. It enhances the user experience by consistently providing rich, detailed information about materials, including technical specifications, available options, and manufacturer details in a standardized format.

### Key Capabilities

The Enhanced Material Expert offers all the capabilities of the standard Material Expert, plus these enhanced features:

1. Comprehensive Metadata Presentation
2. Present complete material specifications in every response
3. Structure information consistently for better readability
4. Include technical details that might otherwise be omitted
5. Ensure all material properties are presented
6. Structured Response Formatting
7. Convert raw material data into well-organized descriptions
8. Format search results with standardized property sections
9. Transform JSON data into readable, comprehensive text
10. Maintain consistent information hierarchy
11. Complete Material Specifications
12. Always include available colors, sizes, and finishes
13. Provide comprehensive technical specifications
14. Include manufacturer information when available
15. Detail installation requirements and considerations
16. Enhanced Material Comparisons
17. Present side-by-side comparisons with complete metadata
18. Highlight differences with consistent property references
19. Ensure no critical specification is omitted in comparisons
20. Maintain metadata consistency across multiple materials
21. Metadata-Aware Recommendations
22. Base recommendations on complete technical information
23. Reference specific properties when suggesting applications
24. Include relevant metadata in application guidance
25. Ensure recommendations are backed by detailed specifications

Ensure all material properties are presented

Structured Response Formatting

Maintain consistent information hierarchy

Complete Material Specifications

Detail installation requirements and considerations

Enhanced Material Comparisons

Maintain metadata consistency across multiple materials

Metadata-Aware Recommendations

### Architecture

The Enhanced Material Expert integrates with the broader KAI platform through the same components as the standard Material Expert:

#### Component Structure

packages/  
├── agents/  
│ ├── src/  
│ │ ├── frontend/  
│ │ │ ├── materialExpert.ts # Base agent  
│ │ │ └── enhancedMaterialExpert.ts # Enhanced agent implementation  
│ │ ├── services/  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ ├── materialSearch.ts # Material search tool  
│ │ │ ├── vectorSearch.ts # Vector search tool  
│ │ │ └── index.ts # Tool exports  
│ │ ├── utils/  
│ │ │ └── materialMetadataFormatter.ts # Metadata formatting utilities  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── client/  
 └── src/  
 └── components/  
 └── agents/  
 ├── MaterialExpertPanel.tsx # Client-side interface  
 └── AgentDashboard.tsx # Agent integration in UI

packages/  
├── agents/  
│ ├── src/  
│ │ ├── frontend/  
│ │ │ ├── materialExpert.ts # Base agent  
│ │ │ └── enhancedMaterialExpert.ts # Enhanced agent implementation  
│ │ ├── services/  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ ├── materialSearch.ts # Material search tool  
│ │ │ ├── vectorSearch.ts # Vector search tool  
│ │ │ └── index.ts # Tool exports  
│ │ ├── utils/  
│ │ │ └── materialMetadataFormatter.ts # Metadata formatting utilities  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── client/  
 └── src/  
 └── components/  
 └── agents/  
 ├── MaterialExpertPanel.tsx # Client-side interface  
 └── AgentDashboard.tsx # Agent integration in UI

#### Architectural Layers

1. Agent Layer (enhancedMaterialExpert.ts)
2. Extends the base Material Expert functionality
3. Processes and enhances agent responses with metadata
4. Implements JSON detection and formatting
5. Provides enhanced instructions to the LLM
6. Formatter Layer (materialMetadataFormatter.ts)
7. Provides specialized formatting functions
8. Transforms raw material data into structured descriptions
9. Formats search results consistently
10. Generates comprehensive material descriptions
11. Service & Tool Layers
12. Use the same services and tools as the Material Expert
13. Access material database and search capabilities
14. Retrieve comprehensive material information
15. Support vector-based similarity searches
16. UI Layer (MaterialExpertPanel.tsx)
17. Presents the enhanced material information in the UI
18. Displays comprehensive metadata in a readable format
19. Maintains the same interface as the standard Material Expert
20. Benefits from enriched agent responses

enhancedMaterialExpert.ts

Provides enhanced instructions to the LLM

Formatter Layer (materialMetadataFormatter.ts)

materialMetadataFormatter.ts

Generates comprehensive material descriptions

Service & Tool Layers

Support vector-based similarity searches

UI Layer (MaterialExpertPanel.tsx)

MaterialExpertPanel.tsx

### Implementation Details

#### Agent Implementation

The Enhanced Material Expert extends the MaterialExpert implementation with response enhancement capabilities:

export class EnhancedMaterialExpert implements UserFacingAgent {  
 // Standard UserFacingAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // Enhanced method with metadata formatting  
 public async processUserInput(message: string): Promise<string>;  
  
 // Private helper method for metadata enhancement  
 private enhanceResponseWithMetadata(response: string, originalQuery: string): string;  
}

export class EnhancedMaterialExpert implements UserFacingAgent {  
 // Standard UserFacingAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // Enhanced method with metadata formatting  
 public async processUserInput(message: string): Promise<string>;  
  
 // Private helper method for metadata enhancement  
 private enhanceResponseWithMetadata(response: string, originalQuery: string): string;  
}

#### Response Enhancement

The Enhanced Material Expert processes responses to ensure comprehensive metadata:

private enhanceResponseWithMetadata(response: string, originalQuery: string): string {  
 try {  
 logger.debug('Enhancing response with comprehensive material metadata');  
  
 // Check if the response contains JSON data (material search results)  
 if (response.includes('"id":') && response.includes('"name":')) {  
 try {  
 // Try to parse JSON data in the response  
 const jsonMatch = response.match(/\{[\s\S]\*\}/);  
 if (jsonMatch) {  
 const jsonData = JSON.parse(jsonMatch[0]);  
  
 // Format material results if they exist  
 if (jsonData.results && Array.isArray(jsonData.results)) {  
 const formattedMaterials = formatMaterialSearchResults(jsonData.results);  
  
 // Generate detailed descriptions for each material  
 const materialDescriptions = formattedMaterials.map(material =>   
 generateMaterialDescription(material)  
 );  
  
 // Replace the JSON data with formatted material descriptions  
 return response.replace(  
 jsonMatch[0],   
 `Here are the materials that match your query:\n\n${materialDescriptions.join('\n\n---\n\n')}`  
 );  
 }  
  
 // Handle single material case  
 if (jsonData.id && jsonData.name) {  
 const formattedMaterial = formatMaterialMetadata(jsonData);  
 const materialDescription = generateMaterialDescription(formattedMaterial);  
  
 // Replace the JSON data with formatted material description  
 return response.replace(jsonMatch[0], materialDescription);  
 }  
 }  
 } catch (error) {  
 // Continue with the original response if JSON parsing fails  
 }  
 }  
  
 // If no JSON data found, return the original response  
 return response;  
 } catch (error) {  
 return response; // Return original response on error  
 }  
}

private enhanceResponseWithMetadata(response: string, originalQuery: string): string {  
 try {  
 logger.debug('Enhancing response with comprehensive material metadata');  
  
 // Check if the response contains JSON data (material search results)  
 if (response.includes('"id":') && response.includes('"name":')) {  
 try {  
 // Try to parse JSON data in the response  
 const jsonMatch = response.match(/\{[\s\S]\*\}/);  
 if (jsonMatch) {  
 const jsonData = JSON.parse(jsonMatch[0]);  
  
 // Format material results if they exist  
 if (jsonData.results && Array.isArray(jsonData.results)) {  
 const formattedMaterials = formatMaterialSearchResults(jsonData.results);  
  
 // Generate detailed descriptions for each material  
 const materialDescriptions = formattedMaterials.map(material =>   
 generateMaterialDescription(material)  
 );  
  
 // Replace the JSON data with formatted material descriptions  
 return response.replace(  
 jsonMatch[0],   
 `Here are the materials that match your query:\n\n${materialDescriptions.join('\n\n---\n\n')}`  
 );  
 }  
  
 // Handle single material case  
 if (jsonData.id && jsonData.name) {  
 const formattedMaterial = formatMaterialMetadata(jsonData);  
 const materialDescription = generateMaterialDescription(formattedMaterial);  
  
 // Replace the JSON data with formatted material description  
 return response.replace(jsonMatch[0], materialDescription);  
 }  
 }  
 } catch (error) {  
 // Continue with the original response if JSON parsing fails  
 }  
 }  
  
 // If no JSON data found, return the original response  
 return response;  
 } catch (error) {  
 return response; // Return original response on error  
 }  
}

#### Explicit Material Metadata Instructions

The Enhanced Material Expert includes detailed instructions for the LLM:

const materialMetadataInstructions = `  
IMPORTANT: When discussing materials, ALWAYS include comprehensive details about:  
- What the material is (e.g., tile, wood, lighting)  
- The specific name of the material (e.g., Blanco Beige, Nordic Oak)  
- Available colors  
- Available sizes/dimensions  
- Finish options  
- Technical specifications  
- Manufacturer information  
  
For example, instead of just saying "This tile would work well for your bathroom floor",  
provide detailed information like: "Blanco Beige is a porcelain tile manufactured by   
CeramicWorks. It comes in White and Cream colors, available in sizes 12"x24", 24"x24",   
and 24"x48". It has a matte finish with R10 slip resistance rating, making it suitable   
for bathroom floors."  
  
Always present material information in a clear, structured format that highlights  
the key metadata properties available in our database.  
`;

const materialMetadataInstructions = `  
IMPORTANT: When discussing materials, ALWAYS include comprehensive details about:  
- What the material is (e.g., tile, wood, lighting)  
- The specific name of the material (e.g., Blanco Beige, Nordic Oak)  
- Available colors  
- Available sizes/dimensions  
- Finish options  
- Technical specifications  
- Manufacturer information  
  
For example, instead of just saying "This tile would work well for your bathroom floor",  
provide detailed information like: "Blanco Beige is a porcelain tile manufactured by   
CeramicWorks. It comes in White and Cream colors, available in sizes 12"x24", 24"x24",   
and 24"x48". It has a matte finish with R10 slip resistance rating, making it suitable   
for bathroom floors."  
  
Always present material information in a clear, structured format that highlights  
the key metadata properties available in our database.  
`;

### Setup Instructions

#### Prerequisites

* Functioning KAI platform with material database
* CrewAI integration set up according to CrewAI installation guide
* Vector search capabilities for material similarity

#### Installation

The Enhanced Material Expert is included in the standard crewAI integration package:

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

#### Configuration

Configure the agent in your application initialization:

import { createEnhancedMaterialExpert } from '@kai/agents';  
  
// Create an Enhanced Material Expert instance  
const enhancedMaterialExpert = await createEnhancedMaterialExpert(  
 {  
 id: 'enhanced-material-expert-1',  
 name: 'Material Expert',  
 description: 'Expert in construction materials with comprehensive metadata',  
 verbose: true,  
 // Additional configuration options  
 },  
 {  
 model: 'gpt-4',  
 temperature: 0.3  
 }  
);

import { createEnhancedMaterialExpert } from '@kai/agents';  
  
// Create an Enhanced Material Expert instance  
const enhancedMaterialExpert = await createEnhancedMaterialExpert(  
 {  
 id: 'enhanced-material-expert-1',  
 name: 'Material Expert',  
 description: 'Expert in construction materials with comprehensive metadata',  
 verbose: true,  
 // Additional configuration options  
 },  
 {  
 model: 'gpt-4',  
 temperature: 0.3  
 }  
);

### Usage Examples

#### Client-Side Integration

import React from 'react';  
import { MaterialExpertPanel } from '../components/agents/MaterialExpertPanel';  
  
const MaterialExpertPage: React.FC = () => {  
 return (  
 <div className="material-expert-page">  
 <h1>Material Expert</h1>  
 {/\* Use the Panel with enhanced agent type \*/}  
 <MaterialExpertPanel useEnhancedExpert={true} />  
 </div>  
 );  
};  
  
export default MaterialExpertPage;

import React from 'react';  
import { MaterialExpertPanel } from '../components/agents/MaterialExpertPanel';  
  
const MaterialExpertPage: React.FC = () => {  
 return (  
 <div className="material-expert-page">  
 <h1>Material Expert</h1>  
 {/\* Use the Panel with enhanced agent type \*/}  
 <MaterialExpertPanel useEnhancedExpert={true} />  
 </div>  
 );  
};  
  
export default MaterialExpertPage;

#### Processing User Queries

import { createEnhancedMaterialExpert } from '@kai/agents';  
  
// Create the Enhanced Material Expert  
const enhancedMaterialExpert = await createEnhancedMaterialExpert(  
 { id: 'enhanced-material-expert-1' },  
 { model: 'gpt-4', temperature: 0.3 }  
);  
  
// Ask questions about materials with enhanced responses  
const query1 = 'What materials would work best for a kitchen backsplash?';  
const response1 = await enhancedMaterialExpert.processUserInput(query1);  
console.log(response1); // Response includes comprehensive metadata  
  
const query2 = 'Compare ceramic and porcelain tiles for bathroom floors.';  
const response2 = await enhancedMaterialExpert.processUserInput(query2);  
console.log(response2); // Comparison includes complete specifications  
  
const query3 = 'What are the technical specifications of Carrara marble?';  
const response3 = await enhancedMaterialExpert.processUserInput(query3);  
console.log(response3); // Full technical specifications included

import { createEnhancedMaterialExpert } from '@kai/agents';  
  
// Create the Enhanced Material Expert  
const enhancedMaterialExpert = await createEnhancedMaterialExpert(  
 { id: 'enhanced-material-expert-1' },  
 { model: 'gpt-4', temperature: 0.3 }  
);  
  
// Ask questions about materials with enhanced responses  
const query1 = 'What materials would work best for a kitchen backsplash?';  
const response1 = await enhancedMaterialExpert.processUserInput(query1);  
console.log(response1); // Response includes comprehensive metadata  
  
const query2 = 'Compare ceramic and porcelain tiles for bathroom floors.';  
const response2 = await enhancedMaterialExpert.processUserInput(query2);  
console.log(response2); // Comparison includes complete specifications  
  
const query3 = 'What are the technical specifications of Carrara marble?';  
const response3 = await enhancedMaterialExpert.processUserInput(query3);  
console.log(response3); // Full technical specifications included

### Enhanced Response Examples

#### Standard Material Expert Response:

Ceramic tiles would work well for your bathroom floor. They are durable, water-resistant,  
and come in many designs. They're also less expensive than porcelain.

Ceramic tiles would work well for your bathroom floor. They are durable, water-resistant,  
and come in many designs. They're also less expensive than porcelain.

#### Enhanced Material Expert Response:

For your bathroom floor, I recommend ceramic tiles which offer several advantages:  
  
Ceramic Tile (Bathroom Series)  
• Material Type: Glazed ceramic  
• Available Colors: White, Beige, Gray, Blue  
• Available Sizes: 12"x12", 18"x18", 12"x24"  
• Finish Options: Matte, Semi-gloss, Textured  
• Technical Specifications:  
 - Water Absorption: 3-5%  
 - PEI Rating: Class 3 (suitable for residential bathrooms)  
 - Slip Resistance: R9-R10  
 - Thickness: 8mm  
• Manufacturer: CeramicWorks  
• Price Range: $3-7 per square foot  
• Maintenance: Requires periodic sealing of grout lines  
  
These tiles are more affordable than porcelain while still providing good water resistance  
and durability for residential bathroom applications.

For your bathroom floor, I recommend ceramic tiles which offer several advantages:  
  
Ceramic Tile (Bathroom Series)  
• Material Type: Glazed ceramic  
• Available Colors: White, Beige, Gray, Blue  
• Available Sizes: 12"x12", 18"x18", 12"x24"  
• Finish Options: Matte, Semi-gloss, Textured  
• Technical Specifications:  
 - Water Absorption: 3-5%  
 - PEI Rating: Class 3 (suitable for residential bathrooms)  
 - Slip Resistance: R9-R10  
 - Thickness: 8mm  
• Manufacturer: CeramicWorks  
• Price Range: $3-7 per square foot  
• Maintenance: Requires periodic sealing of grout lines  
  
These tiles are more affordable than porcelain while still providing good water resistance  
and durability for residential bathroom applications.

### Advanced Configuration

#### Custom Metadata Formatters

Create custom formatters to enhance the material metadata presentation:

import { Tool } from 'crewai';  
import {   
 formatMaterialMetadata,   
 generateMaterialDescription   
} from '../utils/materialMetadataFormatter';  
  
// Create a specialized formatter for project-specific material information  
const createProjectContextFormatter = () => {  
 return (material) => {  
 const standardFormatted = formatMaterialMetadata(material);  
  
 // Enhance with project-specific context  
 return {  
 ...standardFormatted,  
 projectRecommendations: generateProjectRecommendations(material),  
 installationComplexity: calculateInstallationComplexity(material),  
 costCategory: determineCostCategory(material.price),  
 sustainabilityRating: calculateSustainabilityScore(material),  
 maintenanceRequirements: generateMaintenanceGuide(material)  
 };  
 };  
};  
  
// Create a custom enhanced agent with the specialized formatter  
const customEnhancedMaterialExpert = await createEnhancedMaterialExpert(  
 {   
 id: 'project-enhanced-expert-1',  
 metadataFormatter: createProjectContextFormatter()  
 },  
 { model: 'gpt-4', temperature: 0.3 }  
);

import { Tool } from 'crewai';  
import {   
 formatMaterialMetadata,   
 generateMaterialDescription   
} from '../utils/materialMetadataFormatter';  
  
// Create a specialized formatter for project-specific material information  
const createProjectContextFormatter = () => {  
 return (material) => {  
 const standardFormatted = formatMaterialMetadata(material);  
  
 // Enhance with project-specific context  
 return {  
 ...standardFormatted,  
 projectRecommendations: generateProjectRecommendations(material),  
 installationComplexity: calculateInstallationComplexity(material),  
 costCategory: determineCostCategory(material.price),  
 sustainabilityRating: calculateSustainabilityScore(material),  
 maintenanceRequirements: generateMaintenanceGuide(material)  
 };  
 };  
};  
  
// Create a custom enhanced agent with the specialized formatter  
const customEnhancedMaterialExpert = await createEnhancedMaterialExpert(  
 {   
 id: 'project-enhanced-expert-1',  
 metadataFormatter: createProjectContextFormatter()  
 },  
 { model: 'gpt-4', temperature: 0.3 }  
);

#### Integration with Material Standards Database

Connect the Enhanced Material Expert to standards databases:

import { Tool } from 'crewai';  
  
// Create a tool for accessing material standards  
const createMaterialStandardsTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'material\_standards\_lookup',  
 description: 'Look up industry standards and certifications for material types',  
 func: async (args) => {  
 const { materialType, standardType } = JSON.parse(args);  
  
 // Implement standards retrieval  
 const standards = await getMaterialStandards(materialType, standardType);  
  
 // Format standards information in a comprehensive way  
 const formattedStandards = standards.map(standard => ({  
 name: standard.name,  
 body: standard.issuingOrganization,  
 version: standard.latestVersion,  
 requirements: standard.keyRequirements,  
 testMethods: standard.testingProcedures,  
 applicableRegions: standard.geographicScope  
 }));  
  
 return JSON.stringify(formattedStandards);  
 }  
 });  
};  
  
// Add it to the agent  
const enhancedMaterialExpert = await createEnhancedMaterialExpert(  
 {   
 id: 'standards-aware-expert-1',  
 additionalTools: [await createMaterialStandardsTool()]  
 },  
 { model: 'gpt-4', temperature: 0.3 }  
);

import { Tool } from 'crewai';  
  
// Create a tool for accessing material standards  
const createMaterialStandardsTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'material\_standards\_lookup',  
 description: 'Look up industry standards and certifications for material types',  
 func: async (args) => {  
 const { materialType, standardType } = JSON.parse(args);  
  
 // Implement standards retrieval  
 const standards = await getMaterialStandards(materialType, standardType);  
  
 // Format standards information in a comprehensive way  
 const formattedStandards = standards.map(standard => ({  
 name: standard.name,  
 body: standard.issuingOrganization,  
 version: standard.latestVersion,  
 requirements: standard.keyRequirements,  
 testMethods: standard.testingProcedures,  
 applicableRegions: standard.geographicScope  
 }));  
  
 return JSON.stringify(formattedStandards);  
 }  
 });  
};  
  
// Add it to the agent  
const enhancedMaterialExpert = await createEnhancedMaterialExpert(  
 {   
 id: 'standards-aware-expert-1',  
 additionalTools: [await createMaterialStandardsTool()]  
 },  
 { model: 'gpt-4', temperature: 0.3 }  
);

### Performance Considerations

#### Metadata Processing Optimization

1. Selective Enhancement
2. Apply full enhancement only for direct material queries
3. Use simplified enhancement for comparative discussions
4. Implement tiered metadata detail levels based on query complexity
5. Cache frequently requested material descriptions
6. Response Size Management
7. Prioritize critical metadata based on query context
8. Implement collapsible sections for detailed specifications
9. Use progressive disclosure for very detailed information
10. Balance comprehensiveness with response clarity
11. Formatting Efficiency
12. Implement parallel processing for multiple material descriptions
13. Optimize regex patterns for JSON detection
14. Use memory-efficient string manipulation
15. Cache intermediate formatting results

Cache frequently requested material descriptions

Response Size Management

Balance comprehensiveness with response clarity

Formatting Efficiency

### Security Considerations

The Enhanced Material Expert follows the same security considerations as the standard Material Expert, with additional attention to:

1. Metadata Validation
2. Validate enhanced metadata against schema before inclusion
3. Sanitize any externally sourced metadata
4. Verify manufacturer information accuracy
5. Implement checks for metadata consistency
6. Information Accuracy
7. Cross-reference specifications with authoritative sources
8. Indicate confidence levels for enhanced metadata
9. Distinguish between verified and inferred specifications
10. Implement review mechanisms for metadata quality
11. Proprietary Information
12. Apply appropriate access controls for premium material information
13. Respect intellectual property rights when enhancing metadata
14. Properly attribute manufacturer-specific information
15. Follow data usage guidelines for proprietary specifications

Implement checks for metadata consistency

Information Accuracy

Implement review mechanisms for metadata quality

Proprietary Information

### Related Documentation

* Material Expert - Base Material Expert functionality
* Knowledge Base - Material database structure and management
* CrewAI Integration - Overall agent system architecture
* CrewAI Implementation - Implementation details
* Agent Installation - Setup instructions

# External Sources Integration

Source: readme/external-sources-integration.md

---

## External Sources Integration

### Overview

The external sources integration provides a flexible system for connecting with various material databases and sources. Dynamic configuration through the admin panel has been fully implemented with real API connections, replacing all mock implementations.

### Features

* Dynamic source configuration through admin panel with real API connections
* Multiple authentication methods (Basic, OAuth2, API Key, Bearer Token)
* Configurable endpoints and data mappings
* Real-time sync status monitoring
* Automatic entity linking
* Caching with TTL support

### Available Authentication Methods

* basic: Username and password authentication
* oauth2: OAuth 2.0 flow with access/refresh tokens
* api\_key: Single API key authentication
* api\_key\_secret: API key + secret pair
* bearer: Bearer token authentication
* custom: Custom authentication schemes with real API handling

basic

oauth2

api\_key

api\_key\_secret

bearer

custom

### Adding a New Source

1. Navigate to the Admin Panel > External Sources
2. Click "Add Source"
3. Configure the source:
4. Name and description
5. Base URL and authentication
6. Endpoint configurations
7. Data field mappings
8. Sync interval settings

### Endpoint Configuration

Each source requires endpoint configurations:

{  
 "search": {  
 "path": "/api/search",  
 "method": "GET",  
 "description": "Search materials"  
 },  
 "material": {  
 "path": "/api/materials/{id}",  
 "method": "GET",  
 "description": "Get material by ID"  
 }  
}

{  
 "search": {  
 "path": "/api/search",  
 "method": "GET",  
 "description": "Search materials"  
 },  
 "material": {  
 "path": "/api/materials/{id}",  
 "method": "GET",  
 "description": "Get material by ID"  
 }  
}

### Data Mapping

Define how external data maps to internal fields:

{  
 "idField": "material.id",  
 "nameField": "material.name",  
 "descriptionField": "material.description",  
 "propertiesMap": {  
 "density": "properties.density",  
 "color": "properties.color"  
 }  
}

{  
 "idField": "material.id",  
 "nameField": "material.name",  
 "descriptionField": "material.description",  
 "propertiesMap": {  
 "density": "properties.density",  
 "color": "properties.color"  
 }  
}

### Sync Process

1. Source is validated on creation with real API connection test
2. Sync runs based on configured interval using real API calls
3. New/updated materials are imported through actual API endpoints
4. Entity linking detects relationships between imported materials
5. Real-time events notify admin panel of sync status and results

### Admin Panel Features

* Source management (CRUD operations)
* Sync status monitoring
* Statistics tracking:
* Total materials
* Last sync duration
* Success/failure rate
* Materials created/updated
* Error logging and reporting

### Example: Adding MatWeb Integration

1. Add new source in admin panel
2. Configure endpoints:
3. Base URL: https://api.matweb.com
4. Search: /api/v2/search
5. Material: /api/v2/materials/{id}
6. Set up API key authentication with your MatWeb API credentials
7. Configure data mappings between MatWeb fields and internal schema
8. Set sync interval (recommended: 24 hours for full sync)
9. Enable and test connection using the "Test Connection" button which performs a real API call

Note: All mock implementations previously used for development have been replaced with actual API connections. The system now performs real HTTP requests to external material databases.

### Using the Unified Search API for External Integrations

Our system now provides a unified search API endpoint that significantly simplifies integration for third-party developers. Rather than having to learn multiple resource-specific endpoints, all search operations can be conducted through a single endpoint:

GET /search  
POST /search

GET /search  
POST /search

#### Key Benefits for Integration Partners

* Reduced API Surface: Developers only need to learn one endpoint pattern instead of multiple resource-specific endpoints
* Consistent Parameter Structure: Same parameter format works across all resource types
* Flexible Filtering: Support for all metadata fields as filter parameters
* Future-Proof: New resource types are automatically supported without API changes
* Simplified SDK Development: Client libraries can be dramatically simplified

#### Integration Example

To search for materials from an external system:

GET /search?type=materials&query=ceramic&materialType=tile&limit=10

GET /search?type=materials&query=ceramic&materialType=tile&limit=10

To retrieve user history data:

GET /search?type=history&userId=user-123&limit=25&sort=createdAt:desc

GET /search?type=history&userId=user-123&limit=25&sort=createdAt:desc

For complex queries, the POST endpoint provides more flexibility:

POST /search  
{  
 "type": "materials",  
 "query": "wood",  
 "filter": {  
 "materialType": "flooring",  
 "price": { "min": 20, "max": 50 },  
 "tags": ["sustainable", "natural"]  
 },  
 "sort": { "price.value": "asc" },  
 "limit": 20  
}

POST /search  
{  
 "type": "materials",  
 "query": "wood",  
 "filter": {  
 "materialType": "flooring",  
 "price": { "min": 20, "max": 50 },  
 "tags": ["sustainable", "natural"]  
 },  
 "sort": { "price.value": "asc" },  
 "limit": 20  
}

#### Available Resource Types

The unified search API supports the following resource types:  
- materials: Material catalog entries  
- collections: Material collections   
- history: Recognition history  
- models: AI models (admin only)  
- datasets: Training datasets (admin only)  
- jobs: Processing jobs (admin only)

materials

collections

history

models

datasets

jobs

#### Authentication

The unified search API follows the same authentication rules as other endpoints:  
- Public resources: No authentication required  
- Protected resources: Bearer token required  
- Admin resources: Admin privileges required

#### Documentation

For full details on the unified search API, including parameter reference and response formats, see the API Reference.

### Best Practices

1. Start with small sync intervals for testing
2. Monitor error logs for issues
3. Use caching for frequently accessed data
4. Configure proper timeouts
5. Set up error notifications
6. Use the unified search API for all search/filter operations

### Troubleshooting

1. Check source configuration
2. Verify authentication settings
3. Monitor sync logs
4. Check rate limits
5. Validate data mappings

### Security Considerations

* API keys and secrets are encrypted
* OAuth tokens are automatically refreshed
* Rate limiting is enforced
* Error logs exclude sensitive data

# Folder Structure

Source: readme/folder-structure.md

---

## Kai Project Structure

This document provides a detailed overview of the Kai project structure and how the components work together.

### Directory Structure

kai/  
├── packages/ # Monorepo packages  
│ ├── admin/ # Admin panel (Next.js)  
│ │ ├── src/ # Admin source code  
│ │ │ ├── components/ # Reusable admin components  
│ │ │ ├── pages/ # Admin app pages  
│ │ │ └── types/ # TypeScript type definitions  
│ │ ├── next.config.js # Next.js configuration  
│ │ ├── package.json # Admin dependencies  
│ │ └── tsconfig.json # TypeScript configuration  
│ │  
│ ├── client/ # Client application (Gatsby)  
│ │ ├── src/ # Client source code  
│ │ │ ├── components/ # UI components  
│ │ │ ├── pages/ # Client app pages  
│ │ │ ├── providers/ # Context providers  
│ │ │ ├── services/ # Client-side services  
│ │ │ └── theme/ # Styling theme  
│ │ ├── gatsby-config.js # Gatsby configuration  
│ │ ├── package.json # Client dependencies  
│ │ └── tsconfig.json # TypeScript configuration  
│ │  
│ ├── ml/ # Machine learning package  
│ │ ├── python/ # Python ML scripts  
│ │ ├── src/ # TypeScript interfaces to ML  
│ │ ├── docs/ # ML documentation  
│ │ └── package.json # ML package dependencies  
│ │  
│ ├── server/ # Backend server  
│ │ ├── src/ # Server source code  
│ │ │ ├── controllers/ # API controllers  
│ │ │ ├── middleware/ # Express middleware  
│ │ │ ├── models/ # Data models  
│ │ │ ├── routes/ # API routes  
│ │ │ ├── services/ # Business logic services  
│ │ │ └── utils/ # Utilities  
│ │ ├── scripts/ # Server scripts  
│ │ └── package.json # Server dependencies  
│ │  
│ └── shared/ # Shared code and types  
│ ├── src/ # Shared source code  
│ │ ├── types/ # Shared type definitions  
│ │ └── utils/ # Shared utilities  
│ └── package.json # Shared dependencies  
│  
├── types/ # Global type definitions  
├── package.json # Root package.json  
├── tsconfig.json # Root TypeScript config  
└── README.md # Project overview

kai/  
├── packages/ # Monorepo packages  
│ ├── admin/ # Admin panel (Next.js)  
│ │ ├── src/ # Admin source code  
│ │ │ ├── components/ # Reusable admin components  
│ │ │ ├── pages/ # Admin app pages  
│ │ │ └── types/ # TypeScript type definitions  
│ │ ├── next.config.js # Next.js configuration  
│ │ ├── package.json # Admin dependencies  
│ │ └── tsconfig.json # TypeScript configuration  
│ │  
│ ├── client/ # Client application (Gatsby)  
│ │ ├── src/ # Client source code  
│ │ │ ├── components/ # UI components  
│ │ │ ├── pages/ # Client app pages  
│ │ │ ├── providers/ # Context providers  
│ │ │ ├── services/ # Client-side services  
│ │ │ └── theme/ # Styling theme  
│ │ ├── gatsby-config.js # Gatsby configuration  
│ │ ├── package.json # Client dependencies  
│ │ └── tsconfig.json # TypeScript configuration  
│ │  
│ ├── ml/ # Machine learning package  
│ │ ├── python/ # Python ML scripts  
│ │ ├── src/ # TypeScript interfaces to ML  
│ │ ├── docs/ # ML documentation  
│ │ └── package.json # ML package dependencies  
│ │  
│ ├── server/ # Backend server  
│ │ ├── src/ # Server source code  
│ │ │ ├── controllers/ # API controllers  
│ │ │ ├── middleware/ # Express middleware  
│ │ │ ├── models/ # Data models  
│ │ │ ├── routes/ # API routes  
│ │ │ ├── services/ # Business logic services  
│ │ │ └── utils/ # Utilities  
│ │ ├── scripts/ # Server scripts  
│ │ └── package.json # Server dependencies  
│ │  
│ └── shared/ # Shared code and types  
│ ├── src/ # Shared source code  
│ │ ├── types/ # Shared type definitions  
│ │ └── utils/ # Shared utilities  
│ └── package.json # Shared dependencies  
│  
├── types/ # Global type definitions  
├── package.json # Root package.json  
├── tsconfig.json # Root TypeScript config  
└── README.md # Project overview

### How Components Work Together

The Kai system is designed as a microservices architecture where each package plays a specific role and communicates with others through well-defined interfaces.

#### Component Interactions

1. Client → Server: The client application communicates with the server via RESTful API calls to:
2. Upload images for recognition
3. Search the knowledge base
4. View material details
5. Manage user collections
6. Admin → Server: The admin panel communicates with the server to:
7. Manage system settings
8. Monitor queues
9. View system analytics
10. Manage materials and collections
11. Server → ML: The server communicates with the ML package to:
12. Perform material recognition
13. Generate vector embeddings
14. Train models
15. Process PDFs
16. Server → Database: The server stores and retrieves data using MongoDB models for:
17. Materials
18. Collections
19. Users
20. Search indexes
21. Processing jobs
22. Queue System Coordination: The various queues (PDF, Crawler, ML) coordinate through a message broker to:
23. Trigger dependent processes
24. Report status updates
25. Manage resource allocation

Manage user collections

Admin → Server: The admin panel communicates with the server to:

Manage materials and collections

Server → ML: The server communicates with the ML package to:

Process PDFs

Server → Database: The server stores and retrieves data using MongoDB models for:

Processing jobs

Queue System Coordination: The various queues (PDF, Crawler, ML) coordinate through a message broker to:

#### Data Flow

1. Material Recognition Flow:  
    User uploads image → Client → Server → ML System →   
    Recognition Results → Knowledge Base Lookup →   
    Enhanced Results → Server → Client → User Interface
2. PDF Processing Flow:  
    Admin uploads PDF → Admin Panel → Server → PDF Queue →   
    PDF Processor → ML OCR → Text/Image Extraction →   
    Knowledge Base Import → Material Creation
3. Search Flow:  
    User enters search → Client → Server →   
    Knowledge Base Search (Text/Vector/Hybrid) →   
    Results → Client → User Interface
4. Web Crawling Flow:  
    Admin configures crawler → Admin Panel → Server →   
    Crawler Queue → Web Crawler → Data Extraction →   
    Knowledge Base Import → Material Creation

Material Recognition Flow:  
 User uploads image → Client → Server → ML System →   
 Recognition Results → Knowledge Base Lookup →   
 Enhanced Results → Server → Client → User Interface

User uploads image → Client → Server → ML System →   
 Recognition Results → Knowledge Base Lookup →   
 Enhanced Results → Server → Client → User Interface

PDF Processing Flow:  
 Admin uploads PDF → Admin Panel → Server → PDF Queue →   
 PDF Processor → ML OCR → Text/Image Extraction →   
 Knowledge Base Import → Material Creation

Admin uploads PDF → Admin Panel → Server → PDF Queue →   
 PDF Processor → ML OCR → Text/Image Extraction →   
 Knowledge Base Import → Material Creation

Search Flow:  
 User enters search → Client → Server →   
 Knowledge Base Search (Text/Vector/Hybrid) →   
 Results → Client → User Interface

User enters search → Client → Server →   
 Knowledge Base Search (Text/Vector/Hybrid) →   
 Results → Client → User Interface

Web Crawling Flow:  
 Admin configures crawler → Admin Panel → Server →   
 Crawler Queue → Web Crawler → Data Extraction →   
 Knowledge Base Import → Material Creation

Admin configures crawler → Admin Panel → Server →   
 Crawler Queue → Web Crawler → Data Extraction →   
 Knowledge Base Import → Material Creation

### Deployment Architecture

The Kai system is deployed as a set of services:

1. Frontend Services:
2. Client application deployed on Vercel
3. Admin panel deployed on Vercel
4. Backend Services:
5. API server deployed on cloud infrastructure
6. ML services deployed on specialized compute instances
7. Databases:
8. MongoDB for primary data storage
9. Vector database for similarity search
10. Infrastructure Services:
11. AWS S3 for file storage
12. Supabase for pub/sub messaging
13. Authentication provider for user management

Admin panel deployed on Vercel

Backend Services:

ML services deployed on specialized compute instances

Databases:

Vector database for similarity search

Infrastructure Services:

### Communication Protocols

The system uses the following communication methods:

1. HTTP/REST: Primary API communication
2. WebSockets: Real-time updates and notifications
3. Message Queue: Asynchronous task processing
4. Database Queries: Direct data access

# Hpa Configuration Guide

Source: readme/hpa-configuration-guide.md

---

## Horizontal Pod Autoscaling (HPA) Configuration Guide

This document explains the Horizontal Pod Autoscaling (HPA) configuration used in the KAI platform, including the reasoning behind different target utilization percentages, scaling behaviors, and custom metrics.

### Table of Contents

1. Overview
2. Standard HPA Configuration
3. Custom Metrics
4. Service-Specific Configurations
5. Scaling Behavior
6. Monitoring and Troubleshooting
7. Best Practices

### Overview

Horizontal Pod Autoscaling (HPA) automatically adjusts the number of pod replicas based on observed metrics. The KAI platform uses a sophisticated HPA configuration that combines standard resource metrics (CPU, memory) with custom application metrics (queue depth, processing time) to ensure optimal performance and resource utilization.

### Standard HPA Configuration

All services in the KAI platform use a standardized HPA configuration with service-specific adjustments. The standard configuration includes:

#### Resource Metrics

1. CPU Utilization:
2. Target: 70% for most services
3. Reasoning: This provides a good balance between resource efficiency and headroom for traffic spikes. At 70%, pods have enough capacity to handle sudden increases in load while still maintaining good resource utilization.
4. Memory Utilization:
5. Target: 80% for most services
6. Reasoning: Memory usage tends to be more stable than CPU, so a higher target is appropriate. The 80% target ensures efficient memory usage while still providing buffer for garbage collection and temporary spikes.

Reasoning: This provides a good balance between resource efficiency and headroom for traffic spikes. At 70%, pods have enough capacity to handle sudden increases in load while still maintaining good resource utilization.

Memory Utilization:

#### Replica Counts

1. Minimum Replicas:
2. Standard: 2 for most services
3. Critical Services: 3 for services that require higher availability
4. Reasoning: Having at least 2 replicas ensures basic high availability, allowing for zero-downtime deployments and resilience to node failures.
5. Maximum Replicas:
6. Interactive Services: 10 replicas
7. Background Services: 5-6 replicas
8. Reasoning: Maximum replica counts are based on expected peak load and the resource consumption of each service. Interactive services need to scale higher to maintain responsiveness during peak usage.

Reasoning: Having at least 2 replicas ensures basic high availability, allowing for zero-downtime deployments and resilience to node failures.

Maximum Replicas:

### Custom Metrics

The KAI platform uses custom metrics to make more intelligent scaling decisions based on actual application behavior rather than just resource usage.

#### Queue-Based Metrics

1. coordinator\_queue\_depth:
2. Description: Number of pending tasks in the queue
3. Target: 10 tasks per pod
4. Reasoning: This ensures that each pod has a manageable number of tasks to process. When the queue grows beyond this threshold, additional pods are added to maintain processing throughput.
5. coordinator\_queue\_processing\_rate:
6. Description: Rate at which tasks are being processed
7. Target: Varies by service
8. Reasoning: This metric helps scale based on actual throughput rather than just queue size, ensuring that we scale appropriately when processing becomes slower.

coordinator\_queue\_depth

Reasoning: This ensures that each pod has a manageable number of tasks to process. When the queue grows beyond this threshold, additional pods are added to maintain processing throughput.

coordinator\_queue\_processing\_rate:

coordinator\_queue\_processing\_rate

#### Processing Time Metrics

1. ml\_processing\_time\_seconds:
2. Description: Average time to process an ML task
3. Target: 5 seconds
4. Reasoning: When processing time exceeds this threshold, it indicates that the service is becoming overloaded and additional replicas are needed to maintain performance.
5. compilation\_time\_seconds:
6. Description: Average time to compile a WASM module
7. Target: 10 seconds
8. Reasoning: WASM compilation is less time-sensitive than ML processing, so a higher threshold is acceptable.

ml\_processing\_time\_seconds

Reasoning: When processing time exceeds this threshold, it indicates that the service is becoming overloaded and additional replicas are needed to maintain performance.

compilation\_time\_seconds:

compilation\_time\_seconds

#### Database Connection Metrics

1. db\_connection\_utilization:
2. Description: Percentage of database connections in use
3. Target: 70%
4. Reasoning: This ensures that services scale before they exhaust their database connection pools, preventing connection timeouts and errors.

db\_connection\_utilization

### Service-Specific Configurations

Different services have slightly different HPA configurations based on their specific requirements and characteristics:

#### Coordinator Service

minReplicas: 2  
maxReplicas: 10  
metrics:  
 - CPU: 70%  
 - Memory: 80%  
 - coordinator\_queue\_depth: 10  
 - ml\_processing\_time\_seconds: 5

minReplicas: 2  
maxReplicas: 10  
metrics:  
 - CPU: 70%  
 - Memory: 80%  
 - coordinator\_queue\_depth: 10  
 - ml\_processing\_time\_seconds: 5

Reasoning: The Coordinator service is central to the platform's operation, handling task distribution and workflow management. It needs to scale quickly in response to increased workload, so it uses queue depth as a primary scaling metric.

#### Mobile Optimization Service

minReplicas: 2  
maxReplicas: 6  
metrics:  
 - CPU: 70%  
 - Memory: 80%  
 - ml\_processing\_time\_seconds: 5

minReplicas: 2  
maxReplicas: 6  
metrics:  
 - CPU: 70%  
 - Memory: 80%  
 - ml\_processing\_time\_seconds: 5

Reasoning: The Mobile Optimization service performs resource-intensive operations but has more predictable load patterns. It uses processing time as a key metric to ensure that optimization tasks complete within acceptable timeframes.

#### WASM Compiler Service

minReplicas: 2  
maxReplicas: 5  
metrics:  
 - CPU: 70%  
 - Memory: 80%  
 - compilation\_time\_seconds: 10

minReplicas: 2  
maxReplicas: 5  
metrics:  
 - CPU: 70%  
 - Memory: 80%  
 - compilation\_time\_seconds: 10

Reasoning: The WASM Compiler service is less time-sensitive than other services, so it has a higher processing time threshold and lower maximum replica count.

### Scaling Behavior

The KAI platform uses sophisticated scaling behavior configurations to ensure stable and efficient scaling:

#### Scale-Up Behavior

scaleUp:  
 stabilizationWindowSeconds: 60  
 policies:  
 - type: Percent  
 value: 100  
 periodSeconds: 60  
 - type: Pods  
 value: 4  
 periodSeconds: 60  
 selectPolicy: Max

scaleUp:  
 stabilizationWindowSeconds: 60  
 policies:  
 - type: Percent  
 value: 100  
 periodSeconds: 60  
 - type: Pods  
 value: 4  
 periodSeconds: 60  
 selectPolicy: Max

Reasoning:  
- The 60-second stabilization window prevents rapid fluctuations in replica count  
- The percentage-based policy allows doubling the replica count in high-load situations  
- The pods-based policy ensures we can add a minimum number of pods even when starting from a low base  
- Using the Max policy ensures we scale up quickly enough to handle sudden load increases

#### Scale-Down Behavior

scaleDown:  
 stabilizationWindowSeconds: 300  
 policies:  
 - type: Percent  
 value: 10  
 periodSeconds: 60  
 - type: Pods  
 value: 2  
 periodSeconds: 60  
 selectPolicy: Min

scaleDown:  
 stabilizationWindowSeconds: 300  
 policies:  
 - type: Percent  
 value: 10  
 periodSeconds: 60  
 - type: Pods  
 value: 2  
 periodSeconds: 60  
 selectPolicy: Min

Reasoning:  
- The longer 300-second stabilization window prevents premature scale-down during temporary lulls  
- The conservative 10% reduction rate ensures gradual scaling down to avoid service disruption  
- The pods-based policy limits the maximum number of pods that can be removed at once  
- Using the Min policy ensures we take the most conservative approach when scaling down

### Monitoring and Troubleshooting

The KAI platform includes comprehensive monitoring for HPA behavior:

1. Kubernetes HPA Metrics Dashboard:
2. Shows current and desired replica counts
3. Displays scaling events and their triggers
4. Visualizes custom metrics used for scaling decisions
5. HPA Event Logging:
6. All scaling events are logged with detailed reasons
7. Logs include the metrics that triggered scaling decisions
8. Historical scaling patterns can be analyzed for optimization
9. Scaling Effectiveness Metrics:
10. scaling\_latency\_seconds: Time between threshold breach and scaling action
11. scaling\_success\_ratio: Ratio of successful scaling operations
12. scaling\_oscillation\_count: Number of rapid scale up/down cycles

Visualizes custom metrics used for scaling decisions

HPA Event Logging:

Historical scaling patterns can be analyzed for optimization

Scaling Effectiveness Metrics:

scaling\_latency\_seconds

scaling\_success\_ratio

scaling\_oscillation\_count

### Best Practices

1. Tune Based on Actual Usage:
2. Start with the standard configuration
3. Monitor actual service behavior
4. Adjust thresholds based on observed patterns
5. Avoid Scaling Oscillations:
6. Use appropriate stabilization windows
7. Set realistic target thresholds
8. Consider using predictive scaling for services with regular patterns
9. Service-Specific Considerations:
10. CPU-intensive services: Focus on CPU metrics
11. Memory-intensive services: Focus on memory metrics
12. I/O-bound services: Use custom metrics related to queue depth or processing time
13. Regular Review:
14. Review scaling patterns quarterly
15. Adjust configurations based on changing usage patterns
16. Consider seasonal adjustments for services with predictable load variations

Adjust thresholds based on observed patterns

Avoid Scaling Oscillations:

Consider using predictive scaling for services with regular patterns

Service-Specific Considerations:

I/O-bound services: Use custom metrics related to queue depth or processing time

Regular Review:

By following these guidelines and understanding the reasoning behind our HPA configuration choices, you can ensure optimal performance and resource utilization for your KAI platform deployment.

# Huggingface Integration

Source: readme/huggingface-integration.md

---

## Hugging Face Integration with Adaptive Model Selection

This document outlines the integration of Hugging Face's machine learning services into the Kai platform. The implementation includes an innovative adaptive model selection system that works alongside existing AI providers (OpenAI, Anthropic) to automatically select the best-performing model for each specific task.

### Overview

The Hugging Face integration extends the Kai platform's AI capabilities beyond dataset management, providing comprehensive ML services that include:

1. Text Generation: Access to a wide range of transformer models
2. Embedding Generation: Vector representations for search and similarity
3. Image Analysis: Classification, object detection, and segmentation
4. Classification: Category prediction for materials and metadata
5. Provider-Agnostic Interface: Seamless operation with other AI providers

### Architecture

The implementation follows a provider-agnostic architecture with several key components:

#### Core Components

1. HuggingFaceProvider Service
2. Interfaces with Hugging Face's APIs
3. Handles authentication and configuration
4. Provides unified access to HF's ML capabilities
5. Manages caching and performance optimization
6. ModelRegistry Service
7. Tracks performance metrics for all models across providers
8. Maintains historical performance data
9. Provides model selection based on performance criteria
10. Supports different selection strategies (accuracy, latency, cost)
11. ModelRouter System
12. Routes AI requests to the optimal model
13. Implements rotation-based evaluation
14. Provides fallback mechanisms for service unavailability
15. Handles cross-provider compatibility

Manages caching and performance optimization

ModelRegistry Service

Supports different selection strategies (accuracy, latency, cost)

ModelRouter System

#### Rotation-Based Evaluation System

The implementation includes an innovative model evaluation mechanism:

1. Standard Operation - During normal operation, each task is routed to the historically best-performing model
2. Evaluation Mode - After every 10 tasks of a particular type, the system enters evaluation mode
3. Multi-Model Testing - The next 3 tasks are executed across all available models simultaneously
4. Comparative Analysis - Results are compared, metrics updated, and model rankings adjusted

This creates a self-improving system that continuously identifies the best models for each specific task type without requiring manual intervention.

### Implementation Details

#### 1. HuggingFaceProvider Service

The provider service exposes various ML capabilities through a unified interface:

##### Text Generation

interface TextGenerationOptions {  
 model?: string;  
 maxLength?: number;  
 temperature?: number;  
 topK?: number;  
 topP?: number;  
 repetitionPenalty?: number;  
}  
  
interface TextGenerationResult {  
 text: string;  
 model: string;  
 finishReason: string;  
 processingTime: number;  
}

interface TextGenerationOptions {  
 model?: string;  
 maxLength?: number;  
 temperature?: number;  
 topK?: number;  
 topP?: number;  
 repetitionPenalty?: number;  
}  
  
interface TextGenerationResult {  
 text: string;  
 model: string;  
 finishReason: string;  
 processingTime: number;  
}

##### Embedding Generation

interface EmbeddingOptions {  
 model?: string;  
 truncate?: boolean;  
 normalize?: boolean;  
 encoderType?: 'text' | 'image' | 'multimodal';  
}  
  
interface EmbeddingResult {  
 embedding: number[];  
 dimensions: number;  
 model: string;  
 processingTime: number;  
}

interface EmbeddingOptions {  
 model?: string;  
 truncate?: boolean;  
 normalize?: boolean;  
 encoderType?: 'text' | 'image' | 'multimodal';  
}  
  
interface EmbeddingResult {  
 embedding: number[];  
 dimensions: number;  
 model: string;  
 processingTime: number;  
}

##### Image Analysis

interface ImageAnalysisOptions {  
 model?: string;  
 task?: 'object-detection' | 'image-classification' | 'image-segmentation';  
}

interface ImageAnalysisOptions {  
 model?: string;  
 task?: 'object-detection' | 'image-classification' | 'image-segmentation';  
}

#### 2. ModelRegistry Service

The registry tracks and analyzes model performance:

interface ModelPerformanceMetrics {  
 // Quality metrics  
 accuracy?: number;  
 relevance?: number;  
  
 // Operational metrics  
 latencyMs: number;  
 processingTimeMs: number;  
 tokenCount?: number;  
 costPerRequest?: number;  
  
 // Usage metrics  
 requestCount: number;  
 errorCount: number;  
 userFeedbackScore?: number;  
  
 // Timestamp  
 updatedAt: Date;  
}

interface ModelPerformanceMetrics {  
 // Quality metrics  
 accuracy?: number;  
 relevance?: number;  
  
 // Operational metrics  
 latencyMs: number;  
 processingTimeMs: number;  
 tokenCount?: number;  
 costPerRequest?: number;  
  
 // Usage metrics  
 requestCount: number;  
 errorCount: number;  
 userFeedbackScore?: number;  
  
 // Timestamp  
 updatedAt: Date;  
}

#### 3. ModelRouter System

The router implements the logic for model selection and evaluation:

interface ModelRoutingOptions {  
 taskType: string;  
 prioritize?: 'speed' | 'quality' | 'cost' | 'balanced';  
 maxLatencyMs?: number;  
 minQualityScore?: number;  
 maxCost?: number;  
 userContext?: any;  
}

interface ModelRoutingOptions {  
 taskType: string;  
 prioritize?: 'speed' | 'quality' | 'cost' | 'balanced';  
 maxLatencyMs?: number;  
 minQualityScore?: number;  
 maxCost?: number;  
 userContext?: any;  
}

### API Endpoints

The Hugging Face integration is exposed through the following RESTful API endpoints:

Table content:

Endpoint | Method | Description

/api/ai/text/generate | POST | Generate text using optimal model

/api/ai/embedding/generate | POST | Generate embeddings for vector search

/api/ai/image/analyze | POST | Analyze images (classification, detection)

/api/ai/models/list | GET | List available AI models

/api/ai/models/metrics | GET | Get performance metrics for AI models

/api/ai/evaluation/set | POST | Configure the evaluation system

/api/ai/text/generate

/api/ai/embedding/generate

/api/ai/image/analyze

/api/ai/models/list

/api/ai/models/metrics

/api/ai/evaluation/set

### Usage Examples

#### Text Generation

// Using standard route that selects best model automatically  
const response = await fetch('/api/ai/text/generate', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({  
 prompt: 'Explain the properties of porcelain tiles compared to ceramic tiles',  
 maxLength: 300,  
 temperature: 0.7  
 })  
});  
  
const result = await response.json();  
console.log(result.text);

// Using standard route that selects best model automatically  
const response = await fetch('/api/ai/text/generate', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({  
 prompt: 'Explain the properties of porcelain tiles compared to ceramic tiles',  
 maxLength: 300,  
 temperature: 0.7  
 })  
});  
  
const result = await response.json();  
console.log(result.text);

#### Embedding Generation

// Generate embeddings for vector search  
const response = await fetch('/api/ai/embedding/generate', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({  
 text: 'White marble with subtle grey veining',  
 encoderType: 'text',  
 normalize: true  
 })  
});  
  
const result = await response.json();  
// Use the embedding vector for similarity search  
const embedding = result.embedding;

// Generate embeddings for vector search  
const response = await fetch('/api/ai/embedding/generate', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({  
 text: 'White marble with subtle grey veining',  
 encoderType: 'text',  
 normalize: true  
 })  
});  
  
const result = await response.json();  
// Use the embedding vector for similarity search  
const embedding = result.embedding;

#### Image Analysis

// Analyze an image for material properties  
const formData = new FormData();  
formData.append('image', fileInput.files[0]);  
formData.append('task', 'image-classification');  
formData.append('detailLevel', 'high');  
  
const response = await fetch('/api/ai/image/analyze', {  
 method: 'POST',  
 body: formData  
});  
  
const result = await response.json();  
console.log(result.classifications);

// Analyze an image for material properties  
const formData = new FormData();  
formData.append('image', fileInput.files[0]);  
formData.append('task', 'image-classification');  
formData.append('detailLevel', 'high');  
  
const response = await fetch('/api/ai/image/analyze', {  
 method: 'POST',  
 body: formData  
});  
  
const result = await response.json();  
console.log(result.classifications);

### Configuration

#### Environment Variables

The Hugging Face integration can be configured using the following environment variables:

# Hugging Face Configuration  
HF\_API\_KEY=your\_huggingface\_api\_key  
HF\_ORGANIZATION\_ID=your\_organization\_id (optional)  
HF\_DEFAULT\_TEXT\_MODEL=google/flan-t5-xxl  
HF\_DEFAULT\_EMBEDDING\_MODEL=sentence-transformers/all-MiniLM-L6-v2  
HF\_DEFAULT\_IMAGE\_MODEL=google/vit-base-patch16-224  
HF\_MODEL\_TIMEOUT=30000  
HF\_USE\_FAST\_MODELS=true  
  
# Evaluation System Configuration  
MODEL\_EVALUATION\_STANDARD\_CYCLE=10  
MODEL\_EVALUATION\_TEST\_CYCLE=3  
MODEL\_SELECTION\_METRICS\_WEIGHTS={"accuracy":0.6,"latency":0.2,"cost":0.2}

# Hugging Face Configuration  
HF\_API\_KEY=your\_huggingface\_api\_key  
HF\_ORGANIZATION\_ID=your\_organization\_id (optional)  
HF\_DEFAULT\_TEXT\_MODEL=google/flan-t5-xxl  
HF\_DEFAULT\_EMBEDDING\_MODEL=sentence-transformers/all-MiniLM-L6-v2  
HF\_DEFAULT\_IMAGE\_MODEL=google/vit-base-patch16-224  
HF\_MODEL\_TIMEOUT=30000  
HF\_USE\_FAST\_MODELS=true  
  
# Evaluation System Configuration  
MODEL\_EVALUATION\_STANDARD\_CYCLE=10  
MODEL\_EVALUATION\_TEST\_CYCLE=3  
MODEL\_SELECTION\_METRICS\_WEIGHTS={"accuracy":0.6,"latency":0.2,"cost":0.2}

#### Model Selection Strategy

The system supports different model selection strategies:

1. Balanced (default): Considers accuracy, latency, and cost equally
2. Quality-First: Prioritizes accuracy and relevance over performance
3. Speed-First: Prioritizes low latency for time-sensitive applications
4. Cost-Efficient: Prioritizes lower cost models when appropriate

These strategies can be set globally or specified per request.

### Integration with Other Platform Components

#### Vector Search Integration

The Hugging Face embedding models can be used with the platform's vector search capabilities:

// Generate embedding using Hugging Face  
const embeddingResult = await fetch('/api/ai/embedding/generate', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ text: 'query text' })  
});  
  
const { embedding } = await embeddingResult.json();  
  
// Use embedding for vector search  
const searchResult = await fetch('/api/materials/vector-search', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({   
 vector: embedding,  
 limit: 5,  
 threshold: 0.7  
 })  
});  
  
const similarMaterials = await searchResult.json();

// Generate embedding using Hugging Face  
const embeddingResult = await fetch('/api/ai/embedding/generate', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ text: 'query text' })  
});  
  
const { embedding } = await embeddingResult.json();  
  
// Use embedding for vector search  
const searchResult = await fetch('/api/materials/vector-search', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({   
 vector: embedding,  
 limit: 5,  
 threshold: 0.7  
 })  
});  
  
const similarMaterials = await searchResult.json();

#### Agent System Integration

The Hugging Face models can be used by the agent system for various tasks:

import { initializeAgentSystem } from '@kai/agents';  
  
// Initialize agent system with Hugging Face support  
await initializeAgentSystem({  
 providers: {  
 huggingface: {  
 apiKey: process.env.HF\_API\_KEY,  
 defaultModels: {  
 text: 'google/flan-t5-xxl',  
 embedding: 'sentence-transformers/all-MiniLM-L6-v2'  
 }  
 },  
 openai: {  
 apiKey: process.env.OPENAI\_API\_KEY  
 }  
 },  
 modelSelection: {  
 enableAdaptiveSelection: true,  
 standardCycleLength: 10,  
 evaluationCycleLength: 3  
 }  
});

import { initializeAgentSystem } from '@kai/agents';  
  
// Initialize agent system with Hugging Face support  
await initializeAgentSystem({  
 providers: {  
 huggingface: {  
 apiKey: process.env.HF\_API\_KEY,  
 defaultModels: {  
 text: 'google/flan-t5-xxl',  
 embedding: 'sentence-transformers/all-MiniLM-L6-v2'  
 }  
 },  
 openai: {  
 apiKey: process.env.OPENAI\_API\_KEY  
 }  
 },  
 modelSelection: {  
 enableAdaptiveSelection: true,  
 standardCycleLength: 10,  
 evaluationCycleLength: 3  
 }  
});

### Benefits and Impact

The Hugging Face integration with adaptive model selection provides several advantages:

1. Provider Flexibility: Reduced dependency on any single AI provider
2. Specialized Models: Access to domain-specific models not available in general-purpose APIs
3. Cost Optimization: Ability to use more cost-effective models for specific tasks
4. Performance Optimization: Automatic selection of best-performing models for each task
5. Continuous Improvement: Self-improving system that adapts to model changes
6. Open Source Options: Ability to use open-source models for flexibility and control

### Extension Points

The system is designed for easy extension:

1. New Providers: Implement the provider interface to add new AI services
2. Custom Tasks: Extend the task type definition for specialized use cases
3. Evaluation Metrics: Add custom metrics for domain-specific evaluation
4. Selection Strategies: Implement custom strategy classes for specific needs
5. Caching Mechanisms: Configure different caching strategies by task type

### Installation and Setup

For installation and setup instructions, see the Deployment Guide.

# Implementation Architecture

Source: readme/implementation-architecture.md

---

## Network Access Control System - Architecture Overview

### System Architecture

The network access control system has been implemented as a fully configurable, database-driven solution with the following key components:

┌───────────────────┐ ┌───────────────────┐ ┌───────────────────┐  
│ Admin Interface │────>│ Database Layer │<────│ API Middleware │  
│ │ │ │ │ │  
│ • Network Config │ │ • Internal IPs │ │ • Request Source │  
│ • Endpoint Access │<───>│ • Endpoint Rules │<───>│ • Access Control │  
│ • Rate Limits │ │ • Rate Limits │ │ • Rate Limiting │  
└───────────────────┘ └───────────────────┘ └───────────────────┘

┌───────────────────┐ ┌───────────────────┐ ┌───────────────────┐  
│ Admin Interface │────>│ Database Layer │<────│ API Middleware │  
│ │ │ │ │ │  
│ • Network Config │ │ • Internal IPs │ │ • Request Source │  
│ • Endpoint Access │<───>│ • Endpoint Rules │<───>│ • Access Control │  
│ • Rate Limits │ │ • Rate Limits │ │ • Rate Limiting │  
└───────────────────┘ └───────────────────┘ └───────────────────┘

#### 1. Admin Interface

The NetworkAccessPanel component provides administrators with a comprehensive UI to configure:

* Internal Networks: Define which CIDR ranges/IPs are considered "internal"
* API Endpoint Access: Toggle internal/external access for each endpoint
* Rate Limits: Configure default and custom rate limits based on source networks

This interface is accessible through Settings → Network Access in the admin panel.

#### 2. Database Layer

All configuration is stored in the database to enable runtime changes without code modifications:

* Internal Networks Table: Lists CIDR ranges considered internal
* Endpoint Access Rules Table: Maps endpoints to their access permissions
* Rate Limit Settings Table: Stores default and custom rate limit configurations

#### 3. API Middleware Layer

The middleware layer enforces the configured rules:

* Source Detection: Identifies if a request comes from an internal network
* Access Control: Validates if the requested endpoint allows access from the source
* Rate Limiting: Applies appropriate rate limits based on source network

### Request Flow

When a request arrives at the API:

1. The source IP is extracted and checked against the list of internal networks
2. The requested endpoint is matched against the endpoint access rules
3. Access is granted or denied based on the endpoint's configuration for the source type
4. If access is granted, rate limits are applied based on the source network
5. Finally, normal authentication and authorization checks are performed

### Key Features

#### Fully Configurable Access Controls

All access control settings are stored in the database and configurable through the admin interface:

* No endpoints have hardcoded access restrictions
* Changes take effect immediately without requiring code changes or deployment
* Full audit trail of configuration changes

#### Network-Based Rate Limiting

The system includes sophisticated rate limiting based on network source:

* Default Rate Limit: Applied to all unrecognized networks (default: 30 req/min)
* Custom Rate Limits: Higher limits for internal networks, lower for specific external IPs
* Per-Endpoint Category Limits: Different rate limits for different endpoint categories

#### Admin Interface Integration

The admin interface seamlessly integrates with existing admin panels:

* Network Configuration Tab: Manage internal network CIDR ranges
* API Endpoint Table: Searchable table with toggles for internal/external access
* Rate Limit Configuration: UI for managing default and custom rate limits

### Configuration Examples

#### Define Internal Networks

127.0.0.1/8 (Localhost)  
10.0.0.0/8 (Private network)  
172.16.0.0/12 (Private network)  
192.168.0.0/16 (Private network)  
203.0.113.0/24 (Office network)

127.0.0.1/8 (Localhost)  
10.0.0.0/8 (Private network)  
172.16.0.0/12 (Private network)  
192.168.0.0/16 (Private network)  
203.0.113.0/24 (Office network)

#### API Endpoint Configuration

Table content:

Endpoint | Method | Default Access | Description

/api/users | GET | Internal Only | List all users

/api/analytics/data | DELETE | Internal Only | Clear analytics data

/api/materials | GET | Internal & External | Get materials list

/api/users

/api/analytics/data

/api/materials

#### Rate Limit Configuration

Table content:

Network | Description | Requests/Minute

Default | All undefined networks | 30

10.0.0.0/8 | Internal Network | 300

203.0.113.0/24 | Office Network | 100

8.8.8.8 | Specific External IP | 10

### Implementation Details

The entire system is implemented with minimal changes to the existing codebase:

1. Extended Authorization: Built on top of the existing authentication system
2. Database-Driven Configuration: All settings stored in database tables
3. Middleware Integration: Seamlessly integrates with the existing middleware chain
4. React Admin UI: Clean, user-friendly interface for administrators

See the API Reference document for a complete list of endpoints with their default access configurations.

# Implementation Summary

Source: readme/implementation-summary.md

---

## Network Access Control Implementation Summary

### Overview

This document provides a comprehensive overview of the network-based access control system implemented in the Kai platform. The system allows administrators to configure which API endpoints can be accessed from internal versus external networks through a user-friendly admin interface.

### Key Implementation Principle: No Hardcoded Restrictions

IMPORTANT: THE SYSTEM HAS ZERO HARDCODED ACCESS RESTRICTIONS.

The single most important design principle of our implementation is that no API endpoint has its access level hardcoded in the source code:

* Any endpoint can be configured to be internal-only or external-allowed
* Even sensitive operations (like "update analytics", "clear data", or "system configuration")
* All access settings are stored in database tables, completely separate from code
* Administrators have complete control through the admin interface
* Changes take effect immediately without requiring code changes or redeployment

### Architecture Components

#### 1. Database Models

Two primary database models store all access control configurations:

// Internal network definitions  
interface InternalNetwork {  
 id: string;  
 cidr: string; // CIDR notation (e.g., "10.0.0.0/8")  
 description: string; // Human-readable description  
 createdAt: Date;  
 updatedAt: Date;  
}  
  
// Endpoint access rules  
interface EndpointAccessRule {  
 id: string;  
 path: string; // API endpoint path  
 method: string; // HTTP method (GET, POST, PUT, DELETE)  
 allowInternal: boolean; // Whether internal networks can access  
 allowExternal: boolean; // Whether external networks can access  
 createdAt: Date;  
 updatedAt: Date;  
}

// Internal network definitions  
interface InternalNetwork {  
 id: string;  
 cidr: string; // CIDR notation (e.g., "10.0.0.0/8")  
 description: string; // Human-readable description  
 createdAt: Date;  
 updatedAt: Date;  
}  
  
// Endpoint access rules  
interface EndpointAccessRule {  
 id: string;  
 path: string; // API endpoint path  
 method: string; // HTTP method (GET, POST, PUT, DELETE)  
 allowInternal: boolean; // Whether internal networks can access  
 allowExternal: boolean; // Whether external networks can access  
 createdAt: Date;  
 updatedAt: Date;  
}

#### 2. Network Detection Utilities (utils/network.ts)

utils/network.ts

Utilities for identifying the source network of requests:

// Check if a request comes from an internal network  
export function isInternalRequest(req: Request): boolean {  
 const clientIP = req.ip || req.socket.remoteAddress;  
 const internalNetworks = getInternalNetworksFromDatabase();  
  
 return internalNetworks.some(network => isInCIDR(clientIP, network.cidr));  
}  
  
// Check if an IP falls within a CIDR range  
export function isInCIDR(ip: string, cidr: string): boolean {  
 // Implementation with proper CIDR validation  
}  
  
// Network access types enum  
export enum NetworkAccessType {  
 ANY = 'any',  
 INTERNAL\_ONLY = 'internal-only',  
 EXTERNAL\_ALLOWED = 'external-allowed'  
}

// Check if a request comes from an internal network  
export function isInternalRequest(req: Request): boolean {  
 const clientIP = req.ip || req.socket.remoteAddress;  
 const internalNetworks = getInternalNetworksFromDatabase();  
  
 return internalNetworks.some(network => isInCIDR(clientIP, network.cidr));  
}  
  
// Check if an IP falls within a CIDR range  
export function isInCIDR(ip: string, cidr: string): boolean {  
 // Implementation with proper CIDR validation  
}  
  
// Network access types enum  
export enum NetworkAccessType {  
 ANY = 'any',  
 INTERNAL\_ONLY = 'internal-only',  
 EXTERNAL\_ALLOWED = 'external-allowed'  
}

#### 3. Enhanced Authorization Middleware (middleware/auth.middleware.ts)

middleware/auth.middleware.ts

Extended the existing authorization system to support network-based access control:

export const authorize = (options: {   
 roles?: string[],   
 accessType?: NetworkAccessType   
}) => {  
 return (req: Request, res: Response, next: NextFunction) => {  
 // Existing role-based checks  
 if (options.roles && !req.user?.roles.some(role => options.roles?.includes(role))) {  
 return next(new AuthorizationError('Insufficient role permissions'));  
 }  
  
 // Network-based access check  
 if (options.accessType === NetworkAccessType.INTERNAL\_ONLY) {  
 const endpointRule = getEndpointRuleFromDatabase(req.path, req.method);  
  
 // If endpoint is configured as internal-only and request is external  
 if (endpointRule.allowExternal === false && !isInternalRequest(req)) {  
 return next(new AuthorizationError('This endpoint is only accessible from internal networks'));  
 }  
 }  
  
 next();  
 };  
};

export const authorize = (options: {   
 roles?: string[],   
 accessType?: NetworkAccessType   
}) => {  
 return (req: Request, res: Response, next: NextFunction) => {  
 // Existing role-based checks  
 if (options.roles && !req.user?.roles.some(role => options.roles?.includes(role))) {  
 return next(new AuthorizationError('Insufficient role permissions'));  
 }  
  
 // Network-based access check  
 if (options.accessType === NetworkAccessType.INTERNAL\_ONLY) {  
 const endpointRule = getEndpointRuleFromDatabase(req.path, req.method);  
  
 // If endpoint is configured as internal-only and request is external  
 if (endpointRule.allowExternal === false && !isInternalRequest(req)) {  
 return next(new AuthorizationError('This endpoint is only accessible from internal networks'));  
 }  
 }  
  
 next();  
 };  
};

#### 4. Admin Interface (components/NetworkAccessPanel.tsx)

components/NetworkAccessPanel.tsx

User interface for administrators to configure network access settings:

* Internal networks management: Add, view, and remove CIDR ranges
* Endpoint access control: Toggle internal/external access for any endpoint
* Search and filter: Quickly find specific endpoints
* Real-time updates: Changes take effect immediately

#### 5. API Routes Integration

Applying network-based access control to API endpoints:

// Example of applying network access control to routes  
router.delete('/data',   
 authMiddleware,   
 authorize({   
 roles: ['admin'],   
 accessType: NetworkAccessType.INTERNAL\_ONLY   
 }),   
 asyncHandler(analyticsController.clearAnalyticsData)  
);

// Example of applying network access control to routes  
router.delete('/data',   
 authMiddleware,   
 authorize({   
 roles: ['admin'],   
 accessType: NetworkAccessType.INTERNAL\_ONLY   
 }),   
 asyncHandler(analyticsController.clearAnalyticsData)  
);

### Configuration

#### Admin Dashboard Configuration

Administrators can configure all aspects of network access control through the admin dashboard:

1. Navigate to Settings > Network Access
2. Define internal networks using CIDR notation (e.g., 10.0.0.0/8)
3. Configure access permissions for any API endpoint:
4. Toggle internal access on/off
5. Toggle external access on/off
6. Changes take effect immediately

#### Environment Configuration

For deployment environments, basic configuration can be set via environment variables:

# .env file  
INTERNAL\_NETWORKS=127.0.0.1/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16  
TRUST\_PROXY=true

# .env file  
INTERNAL\_NETWORKS=127.0.0.1/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16  
TRUST\_PROXY=true

### Security Implementation

#### Multi-layered Security

Network access control adds another layer to the security model:

1. Authentication: Verifies user identity (JWT tokens)
2. Role-based Authorization: Checks user has required role
3. Network-based Authorization: Checks request comes from allowed network type
4. Rate Limiting: Protects against abuse (separate system)

#### Fail-Secure Design

The system is designed to fail securely:

* Requests default to "external" if source determination fails
* System applies most restrictive rules in case of ambiguity
* All decision points are thoroughly logged

#### Edge Case Handling

The implementation handles various edge cases:

* Proxy configurations (X-Forwarded-For headers)
* IPv4 and IPv6 addresses
* VPN connections
* Cloud provider network configurations

### Recommended Default Settings

While no restrictions are hardcoded, the system provides sensible default configurations:

Table content:

Endpoint Category | Default Access | Examples

Authentication | External | /api/auth/login, /api/auth/register

Public data | External | /api/materials, /api/search

User account | External | /api/users/me, /api/profile

Admin views | External | /api/admin/dashboard, /api/admin/analytics

System config | Internal | /api/admin/settings, /api/admin/system

Data deletion | Internal | /api/admin/analytics/data (DELETE)

Security ops | Internal | /api/admin/users (PUT/DELETE)

/api/auth/login

/api/auth/register

/api/materials

/api/search

/api/users/me

/api/profile

/api/admin/dashboard

/api/admin/analytics

/api/admin/settings

/api/admin/system

/api/admin/analytics/data

/api/admin/users

These defaults can be changed at any time through the admin interface.

### Complete Configurability

The most important aspect of this implementation is that all endpoint access restrictions are fully configurable through the admin interface:

* No special coding required to change access rules
* Any endpoint can be made internal-only or external-allowed
* All configuration is stored in the database, not in code
* Updates take effect immediately without deployments

### Conclusion

The network access control system provides a flexible, admin-configurable approach to restricting API endpoints based on network source. Administrators have complete control over which endpoints are accessible from internal versus external networks through an intuitive admin interface, with no hardcoded restrictions in the codebase.

### Related Documentation

* API Reference: Complete list of API endpoints with recommended access settings
* Network Access Control Guide: User guide for administrators

# Knowledge Base Agent

Source: readme/knowledge-base-agent.md

---

## Knowledge Base Agent

This document provides detailed information about the Knowledge Base Agent, a specialized crewAI agent designed to monitor and enhance the knowledge base within the KAI platform.

### Overview

The Knowledge Base Agent serves as an intelligent monitor and curator of the KAI material knowledge base. It continuously analyzes materials, identifies relationships, suggests metadata improvements, and ensures data quality. This agent operates in the background, responding to system events and providing insights to maintain a high-quality, well-organized knowledge repository.

### Key Capabilities

The Knowledge Base Agent offers multiple specialized functions:

1. Data Quality Assurance
2. Analyze materials for completeness and accuracy
3. Identify inconsistencies and data gaps
4. Monitor metadata quality and suggest improvements
5. Check for duplicate or redundant entries
6. Relationship Management
7. Identify connections between related materials
8. Suggest new relationships based on similarity analysis
9. Maintain consistency across material hierarchies
10. Ensure proper categorization and classification
11. Event Processing
12. React to material additions, updates, and deletions
13. Process search index updates and optimizations
14. Analyze changes and their impact on the knowledge base
15. Suggest adjustments based on system events
16. Knowledge Base Insights
17. Generate statistics about knowledge base composition
18. Identify trends in material data
19. Highlight areas for expansion or improvement
20. Provide analytics on knowledge base usage and coverage
21. Query Answering
22. Respond to natural language questions about the knowledge base
23. Provide administrative insights and recommendations
24. Answer questions about content organization and structure
25. Suggest optimizations for knowledge base management

Check for duplicate or redundant entries

Relationship Management

Ensure proper categorization and classification

Event Processing

Suggest adjustments based on system events

Knowledge Base Insights

Provide analytics on knowledge base usage and coverage

Query Answering

### Architecture

The Knowledge Base Agent integrates with the broader KAI platform through several key components:

#### Component Structure

packages/  
├── agents/  
│ ├── src/  
│ │ ├── backend/  
│ │ │ └── knowledgeBaseAgent.ts # Agent implementation  
│ │ ├── services/  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ ├── materialSearch.ts # Material search tool  
│ │ │ ├── vectorSearch.ts # Vector search tool  
│ │ │ └── index.ts # Tool exports  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── server/  
 └── src/  
 └── controllers/  
 └── knowledgeBase.controller.ts # Backend integration point

packages/  
├── agents/  
│ ├── src/  
│ │ ├── backend/  
│ │ │ └── knowledgeBaseAgent.ts # Agent implementation  
│ │ ├── services/  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ ├── materialSearch.ts # Material search tool  
│ │ │ ├── vectorSearch.ts # Vector search tool  
│ │ │ └── index.ts # Tool exports  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── server/  
 └── src/  
 └── controllers/  
 └── knowledgeBase.controller.ts # Backend integration point

#### Architectural Layers

1. Agent Layer (knowledgeBaseAgent.ts)
2. Implements the agent's core capabilities
3. Defines specialized methods for knowledge base management
4. Processes system events related to the knowledge base
5. Handles quality analysis and insights generation
6. Service Layer (via ServiceFactory)
7. Provides access to material database and search systems
8. Handles API communication with error management
9. Formats requests and responses appropriately
10. Acts as a bridge to backend knowledge base services
11. Tool Layer (materialSearch, vectorSearch)
12. Implements specialized tools for the agent to use
13. Enables text-based material database queries
14. Provides vector-based similarity searches
15. Formats results for agent consumption
16. Controller Layer (knowledgeBase.controller.ts)
17. Integrates the agent with the server's knowledge base endpoints
18. Routes system events to the agent for processing
19. Provides administrative interfaces for agent insights
20. Manages agent sessions and state

knowledgeBaseAgent.ts

Handles quality analysis and insights generation

Service Layer (via ServiceFactory)

Acts as a bridge to backend knowledge base services

Tool Layer (materialSearch, vectorSearch)

Formats results for agent consumption

Controller Layer (knowledgeBase.controller.ts)

knowledgeBase.controller.ts

### Implementation Details

#### Agent Implementation

The Knowledge Base Agent is a SystemAgent type that implements several specialized methods for knowledge base management:

export class KnowledgeBaseAgent implements SystemAgent {  
 // Standard SystemAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // SystemAgent methods  
 public getAgent(): Agent;  
 public async runTask(taskDescription: string, context?: Record<string, any>): Promise<any>;  
 public async processEvent(eventType: string, eventData: any): Promise<void>;  
  
 // Knowledge Base specific methods  
 public async analyzeQualityIssues(options?: {   
 materialType?: string;  
 severity?: 'low' | 'medium' | 'high';  
 limit?: number;  
 }): Promise<any>;  
 public async generateInsights(): Promise<any>;  
 public async answerQuery(query: string): Promise<string>;  
  
 // Event handler methods  
 private async handleMaterialAdded(data: any): Promise<void>;  
 private async handleMaterialUpdated(data: any): Promise<void>;  
 private async handleMaterialDeleted(data: any): Promise<void>;  
 private async handleSearchIndexUpdated(data: any): Promise<void>;  
}

export class KnowledgeBaseAgent implements SystemAgent {  
 // Standard SystemAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // SystemAgent methods  
 public getAgent(): Agent;  
 public async runTask(taskDescription: string, context?: Record<string, any>): Promise<any>;  
 public async processEvent(eventType: string, eventData: any): Promise<void>;  
  
 // Knowledge Base specific methods  
 public async analyzeQualityIssues(options?: {   
 materialType?: string;  
 severity?: 'low' | 'medium' | 'high';  
 limit?: number;  
 }): Promise<any>;  
 public async generateInsights(): Promise<any>;  
 public async answerQuery(query: string): Promise<string>;  
  
 // Event handler methods  
 private async handleMaterialAdded(data: any): Promise<void>;  
 private async handleMaterialUpdated(data: any): Promise<void>;  
 private async handleMaterialDeleted(data: any): Promise<void>;  
 private async handleSearchIndexUpdated(data: any): Promise<void>;  
}

#### Event Processing

The Knowledge Base Agent processes several types of system events:

// Process events based on type  
switch (eventType) {  
 case 'material\_added':  
 await this.handleMaterialAdded(eventData);  
 break;  
  
 case 'material\_updated':  
 await this.handleMaterialUpdated(eventData);  
 break;  
  
 case 'material\_deleted':  
 await this.handleMaterialDeleted(eventData);  
 break;  
  
 case 'search\_index\_updated':  
 await this.handleSearchIndexUpdated(eventData);  
 break;  
  
 default:  
 logger.warn(`Unknown event type: ${eventType}`);  
}

// Process events based on type  
switch (eventType) {  
 case 'material\_added':  
 await this.handleMaterialAdded(eventData);  
 break;  
  
 case 'material\_updated':  
 await this.handleMaterialUpdated(eventData);  
 break;  
  
 case 'material\_deleted':  
 await this.handleMaterialDeleted(eventData);  
 break;  
  
 case 'search\_index\_updated':  
 await this.handleSearchIndexUpdated(eventData);  
 break;  
  
 default:  
 logger.warn(`Unknown event type: ${eventType}`);  
}

#### Agent Tools

The Knowledge Base Agent leverages specialized tools to perform its tasks:

// Tool for searching materials in the database  
const materialSearchTool = await createMaterialSearchTool();  
  
// Tool for performing vector-based similarity searches  
const vectorSearchTool = await createVectorSearchTool();  
  
// Create a Knowledge Base Agent with tools  
const tools: Tool[] = [  
 materialSearchTool,  
 vectorSearchTool,  
 // Additional tools would be added here in a real implementation  
];

// Tool for searching materials in the database  
const materialSearchTool = await createMaterialSearchTool();  
  
// Tool for performing vector-based similarity searches  
const vectorSearchTool = await createVectorSearchTool();  
  
// Create a Knowledge Base Agent with tools  
const tools: Tool[] = [  
 materialSearchTool,  
 vectorSearchTool,  
 // Additional tools would be added here in a real implementation  
];

#### Agent Description

The Knowledge Base Agent is defined with the following characteristics:

const agent = new Agent({  
 name: 'Knowledge Base Expert',  
 role: 'Knowledge Base Expert',  
 goal: 'Maintain and enhance the quality and value of the material knowledge base',  
 backstory: 'You are an AI assistant specialized in knowledge management and material science. You help ensure the KAI knowledge base contains high-quality, well-structured information about materials.',  
 verbose: true,  
 allowDelegation: false,  
 tools,  
 llm: {  
 model: modelSettings.name,  
 temperature: modelSettings.temperature,  
 },  
});

const agent = new Agent({  
 name: 'Knowledge Base Expert',  
 role: 'Knowledge Base Expert',  
 goal: 'Maintain and enhance the quality and value of the material knowledge base',  
 backstory: 'You are an AI assistant specialized in knowledge management and material science. You help ensure the KAI knowledge base contains high-quality, well-structured information about materials.',  
 verbose: true,  
 allowDelegation: false,  
 tools,  
 llm: {  
 model: modelSettings.name,  
 temperature: modelSettings.temperature,  
 },  
});

### Setup Instructions

#### Prerequisites

* Functioning KAI platform with knowledge base and material database
* CrewAI integration set up according to CrewAI installation guide
* Event handling system for knowledge base operations

#### Installation

The Knowledge Base Agent is included in the standard crewAI integration package:

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

#### Configuration

Configure the agent in your application initialization:

import { createKnowledgeBaseAgent } from '@kai/agents';  
  
// Create a Knowledge Base Agent instance  
const knowledgeBaseAgent = await createKnowledgeBaseAgent(  
 {  
 id: 'knowledge-base-agent-1',  
 // Additional configuration options  
 },  
 {  
 provider: 'openai',  
 name: 'gpt-4',  
 temperature: 0.2  
 }  
);

import { createKnowledgeBaseAgent } from '@kai/agents';  
  
// Create a Knowledge Base Agent instance  
const knowledgeBaseAgent = await createKnowledgeBaseAgent(  
 {  
 id: 'knowledge-base-agent-1',  
 // Additional configuration options  
 },  
 {  
 provider: 'openai',  
 name: 'gpt-4',  
 temperature: 0.2  
 }  
);

### Usage Examples

#### Processing System Events

import { createKnowledgeBaseAgent } from '@kai/agents';  
  
// Create the Knowledge Base Agent  
const knowledgeBaseAgent = await createKnowledgeBaseAgent(  
 { id: 'knowledge-base-agent-1' },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.2 }  
);  
  
// Process a material added event  
await knowledgeBaseAgent.processEvent('material\_added', {  
 id: 'mat-123',  
 name: 'Travertine Limestone',  
 type: 'natural\_stone',  
 properties: {  
 color: 'beige',  
 finish: 'honed',  
 composition: 'calcium carbonate'  
 },  
 metadata: {  
 origin: 'Italy',  
 applications: ['flooring', 'countertops', 'wall cladding']  
 }  
});  
  
// Process a material updated event  
await knowledgeBaseAgent.processEvent('material\_updated', {  
 id: 'mat-456',  
 current: {  
 // Current state of the material  
 },  
 previous: {  
 // Previous state of the material  
 },  
 changedFields: ['properties.durability', 'metadata.applications']  
});

import { createKnowledgeBaseAgent } from '@kai/agents';  
  
// Create the Knowledge Base Agent  
const knowledgeBaseAgent = await createKnowledgeBaseAgent(  
 { id: 'knowledge-base-agent-1' },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.2 }  
);  
  
// Process a material added event  
await knowledgeBaseAgent.processEvent('material\_added', {  
 id: 'mat-123',  
 name: 'Travertine Limestone',  
 type: 'natural\_stone',  
 properties: {  
 color: 'beige',  
 finish: 'honed',  
 composition: 'calcium carbonate'  
 },  
 metadata: {  
 origin: 'Italy',  
 applications: ['flooring', 'countertops', 'wall cladding']  
 }  
});  
  
// Process a material updated event  
await knowledgeBaseAgent.processEvent('material\_updated', {  
 id: 'mat-456',  
 current: {  
 // Current state of the material  
 },  
 previous: {  
 // Previous state of the material  
 },  
 changedFields: ['properties.durability', 'metadata.applications']  
});

#### Analyzing Knowledge Base Quality

import { createKnowledgeBaseAgent } from '@kai/agents';  
  
// Create the Knowledge Base Agent  
const knowledgeBaseAgent = await createKnowledgeBaseAgent(  
 { id: 'knowledge-base-agent-1' },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.2 }  
);  
  
// Analyze quality issues in the knowledge base  
const qualityIssues = await knowledgeBaseAgent.analyzeQualityIssues({  
 materialType: 'ceramic\_tile',  
 severity: 'medium',  
 limit: 10  
});  
  
console.log('Quality issues:', qualityIssues);  
  
// Generate insights about the knowledge base  
const insights = await knowledgeBaseAgent.generateInsights();  
console.log('Knowledge base insights:', insights);  
  
// Answer a query about the knowledge base  
const answer = await knowledgeBaseAgent.answerQuery(  
 'What categories of materials are underrepresented in our knowledge base?'  
);  
console.log('Answer:', answer);

import { createKnowledgeBaseAgent } from '@kai/agents';  
  
// Create the Knowledge Base Agent  
const knowledgeBaseAgent = await createKnowledgeBaseAgent(  
 { id: 'knowledge-base-agent-1' },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.2 }  
);  
  
// Analyze quality issues in the knowledge base  
const qualityIssues = await knowledgeBaseAgent.analyzeQualityIssues({  
 materialType: 'ceramic\_tile',  
 severity: 'medium',  
 limit: 10  
});  
  
console.log('Quality issues:', qualityIssues);  
  
// Generate insights about the knowledge base  
const insights = await knowledgeBaseAgent.generateInsights();  
console.log('Knowledge base insights:', insights);  
  
// Answer a query about the knowledge base  
const answer = await knowledgeBaseAgent.answerQuery(  
 'What categories of materials are underrepresented in our knowledge base?'  
);  
console.log('Answer:', answer);

#### Running Custom Tasks

import { createKnowledgeBaseAgent } from '@kai/agents';  
  
// Create the Knowledge Base Agent  
const knowledgeBaseAgent = await createKnowledgeBaseAgent(  
 { id: 'knowledge-base-agent-1' },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.2 }  
);  
  
// Run a custom task with the agent  
const result = await knowledgeBaseAgent.runTask(  
 'Analyze the consistency of metadata fields across all porcelain tile materials',  
 {  
 materialType: 'porcelain\_tile',  
 metadataFields: ['color', 'finish', 'size', 'water\_absorption']  
 }  
);  
  
console.log('Custom task result:', result);

import { createKnowledgeBaseAgent } from '@kai/agents';  
  
// Create the Knowledge Base Agent  
const knowledgeBaseAgent = await createKnowledgeBaseAgent(  
 { id: 'knowledge-base-agent-1' },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.2 }  
);  
  
// Run a custom task with the agent  
const result = await knowledgeBaseAgent.runTask(  
 'Analyze the consistency of metadata fields across all porcelain tile materials',  
 {  
 materialType: 'porcelain\_tile',  
 metadataFields: ['color', 'finish', 'size', 'water\_absorption']  
 }  
);  
  
console.log('Custom task result:', result);

### Advanced Configuration

#### Custom Knowledge Base Tools

Create custom tools to enhance the Knowledge Base Agent's capabilities:

import { Tool } from 'crewai';  
  
// Create a specialized knowledge graph analysis tool  
const createKnowledgeGraphTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'knowledge\_graph\_analysis',  
 description: 'Analyze the knowledge graph structure and identify optimization opportunities',  
 func: async (args) => {  
 const { scope, depth } = JSON.parse(args);  
  
 // Implement knowledge graph analysis  
 const analysis = await analyzeKnowledgeGraph(scope, depth);  
  
 return JSON.stringify({  
 nodeCount: analysis.nodes,  
 edgeCount: analysis.edges,  
 clusterCoefficient: analysis.clustering,  
 centralNodes: analysis.centralEntities,  
 disconnectedComponents: analysis.disconnectedComponents,  
 recommendedLinks: analysis.suggestedConnections  
 });  
 }  
 });  
};  
  
// Add it to the agent  
const knowledgeBaseAgent = await createKnowledgeBaseAgent(  
 {   
 id: 'advanced-kb-agent-1',  
 additionalTools: [await createKnowledgeGraphTool()]  
 },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.2 }  
);

import { Tool } from 'crewai';  
  
// Create a specialized knowledge graph analysis tool  
const createKnowledgeGraphTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'knowledge\_graph\_analysis',  
 description: 'Analyze the knowledge graph structure and identify optimization opportunities',  
 func: async (args) => {  
 const { scope, depth } = JSON.parse(args);  
  
 // Implement knowledge graph analysis  
 const analysis = await analyzeKnowledgeGraph(scope, depth);  
  
 return JSON.stringify({  
 nodeCount: analysis.nodes,  
 edgeCount: analysis.edges,  
 clusterCoefficient: analysis.clustering,  
 centralNodes: analysis.centralEntities,  
 disconnectedComponents: analysis.disconnectedComponents,  
 recommendedLinks: analysis.suggestedConnections  
 });  
 }  
 });  
};  
  
// Add it to the agent  
const knowledgeBaseAgent = await createKnowledgeBaseAgent(  
 {   
 id: 'advanced-kb-agent-1',  
 additionalTools: [await createKnowledgeGraphTool()]  
 },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.2 }  
);

#### Integration with External Knowledge Sources

Connect the Knowledge Base Agent to external knowledge repositories:

import { Tool } from 'crewai';  
  
// Create a tool for accessing external material databases  
const createExternalKnowledgeTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'external\_knowledge\_access',  
 description: 'Access external material knowledge bases and standards databases',  
 func: async (args) => {  
 const { source, query, limit } = JSON.parse(args);  
  
 // Implement external knowledge access  
 const results = await queryExternalKnowledge(source, query, limit);  
  
 return JSON.stringify({  
 source: source,  
 query: query,  
 results: results.map(item => ({  
 id: item.id,  
 title: item.title,  
 description: item.description,  
 properties: item.properties,  
 url: item.sourceUrl  
 }))  
 });  
 }  
 });  
};  
  
// Add it to the agent  
const knowledgeBaseAgent = await createKnowledgeBaseAgent(  
 {   
 id: 'integrated-kb-agent-1',  
 additionalTools: [await createExternalKnowledgeTool()]  
 },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.2 }  
);

import { Tool } from 'crewai';  
  
// Create a tool for accessing external material databases  
const createExternalKnowledgeTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'external\_knowledge\_access',  
 description: 'Access external material knowledge bases and standards databases',  
 func: async (args) => {  
 const { source, query, limit } = JSON.parse(args);  
  
 // Implement external knowledge access  
 const results = await queryExternalKnowledge(source, query, limit);  
  
 return JSON.stringify({  
 source: source,  
 query: query,  
 results: results.map(item => ({  
 id: item.id,  
 title: item.title,  
 description: item.description,  
 properties: item.properties,  
 url: item.sourceUrl  
 }))  
 });  
 }  
 });  
};  
  
// Add it to the agent  
const knowledgeBaseAgent = await createKnowledgeBaseAgent(  
 {   
 id: 'integrated-kb-agent-1',  
 additionalTools: [await createExternalKnowledgeTool()]  
 },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.2 }  
);

### Performance Considerations

#### Knowledge Processing Optimization

1. Batch Processing
2. Group similar events for bulk processing
3. Implement priority queues for event processing
4. Schedule intensive analyses during off-peak times
5. Caching Strategy
6. Cache frequently accessed knowledge structures
7. Implement incremental analyses for large knowledge bases
8. Store previous analysis results for comparison
9. Resource Management
10. Limit the scope of analysis for real-time responses
11. Implement depth controls for knowledge graph traversal
12. Use sampling techniques for large material collections

Schedule intensive analyses during off-peak times

Caching Strategy

Store previous analysis results for comparison

Resource Management

### Security Considerations

1. Data Access Control
2. Enforce appropriate permissions for knowledge base operations
3. Implement audit logs for agent-initiated changes
4. Restrict sensitive material information access
5. Information Integrity
6. Verify suggested changes against validation rules
7. Implement approval workflows for structural modifications
8. Maintain versioning for all knowledge base modifications
9. Agent Boundaries
10. Restrict the agent to knowledge base operations
11. Validate inputs to prevent injection attacks
12. Limit write access to critical knowledge structures

Restrict sensitive material information access

Information Integrity

Maintain versioning for all knowledge base modifications

Agent Boundaries

### Related Documentation

* Knowledge Base - Core knowledge base architecture
* Database and Vector DB - Storage system details
* CrewAI Integration - Overall agent system architecture
* CrewAI Implementation - Implementation details
* Agent Installation - Setup instructions

# Knowledge Base

Source: readme/knowledge-base.md

---

## Knowledge Base System

The Knowledge Base is the central repository of material information in Kai. It stores comprehensive data about materials, their properties, and relationships, enabling powerful search, organization, and retrieval capabilities. The system now features real-time synchronization, enhanced cross-referencing, deeper hierarchical categorization, bulk operations, and automated entity linking.

### Features

#### Comprehensive Material Storage

The Knowledge Base provides rich storage for material data:

1. Material Specifications
2. Detailed physical properties (dimensions, weight, thickness)
3. Visual attributes (color, pattern, texture)
4. Technical specifications (water absorption, slip resistance, frost resistance)
5. Application contexts (indoor/outdoor, floor/wall, residential/commercial)
6. Manufacturer information
7. Pricing data
8. Rich Media
9. High-resolution images from multiple angles
10. Texture maps and normal maps
11. Installation examples
12. Rendered visualizations
13. Metadata & Classification
14. Material type and category hierarchies
15. Tags and labels
16. Collection and series groupings
17. Installation requirements
18. Compliance certificates

Pricing data

Rich Media

Rendered visualizations

Metadata & Classification

#### Advanced Search Capabilities

The Knowledge Base implements multiple search strategies:

1. Text-Based Search
2. Full-text search with relevance scoring
3. Natural language query processing
4. Autocomplete and suggestions
5. Spell correction and term expansion
6. Faceted filtering
7. Vector-Based Search
8. Similarity search using embedding vectors
9. Visual "search by example"
10. Nearest neighbor algorithms
11. Customizable similarity thresholds
12. Metadata Search
13. Structured queries on material properties
14. Range-based filtering (dimensions, price)
15. Boolean combinations of criteria
16. Aggregation and analytics
17. Combined/Hybrid Search
18. Weighted combination of text and vector search
19. Boosting factors for different search aspects
20. Relevance tuning based on user feedback
21. Context-aware search that considers user history

Faceted filtering

Vector-Based Search

Customizable similarity thresholds

Metadata Search

Aggregation and analytics

Combined/Hybrid Search

#### Collection Management

The system provides comprehensive collection management:

1. Collection Hierarchy
2. Parent-child relationships between collections
3. Deep nested collection structures with unlimited levels
4. Multiple parent categories supported through collection memberships
5. Inheritance of properties
6. Propagation of updates
7. Path tracking for efficient hierarchy traversal
8. Collection Types
9. Manufacturer collections
10. Series and product lines
11. Application-based collections
12. User-created collections
13. System-generated collections (based on similarity)
14. Collection Operations
15. Bulk updates to collection members
16. Collection merging and splitting
17. Collection statistics and analytics
18. Exports and reporting

Path tracking for efficient hierarchy traversal

Collection Types

System-generated collections (based on similarity)

Collection Operations

#### Relationship Management

The Knowledge Base manages complex relationships between materials:

1. Relationship Types
2. Complementary materials (work well together)
3. Alternative materials (substitutes)
4. Required accessories
5. Installation dependencies
6. Visual similarity
7. Series and variant relationships
8. Entity-based references (automatically detected)
9. Relationship Properties
10. Relationship strength (0-1 scale)
11. Bidirectional or directional (with automatic inverse creation)
12. Context-specific metadata
13. Source attribution
14. Confidence scoring
15. Relationship creation timestamp and author tracking
16. Relationship Discovery
17. Automated suggestion of potential relationships
18. Automatic entity linking in text descriptions
19. Intelligent entity detection with confidence thresholds
20. Visual similarity-based relationships
21. User feedback incorporation
22. Background processing for relationship suggestions

Entity-based references (automatically detected)

Relationship Properties

Relationship creation timestamp and author tracking

Relationship Discovery

#### Versioning System

The Knowledge Base includes a robust versioning system:

1. Version Tracking
2. Complete history of changes to materials
3. Temporal queries (state at a specific time)
4. Change attribution (who made changes)
5. Change descriptions and reasons
6. Versioning Operations
7. Point-in-time snapshots
8. Rollback to previous versions
9. Differential storage for efficiency
10. Conflict resolution for concurrent edits
11. Audit Capabilities
12. Complete audit trails
13. Compliance reporting
14. Change analytics
15. Data lineage tracking

Change descriptions and reasons

Versioning Operations

Conflict resolution for concurrent edits

Audit Capabilities

#### Real-Time Synchronization

The Knowledge Base now features comprehensive real-time updates:

1. WebSocket-based Notifications
2. Live updates when material or collection data changes
3. Client subscription management with connection tracking
4. Event categorization by operation type (create, update, delete)
5. Support for relationship and bulk operation events
6. Optimized for low-latency delivery
7. Pub/Sub Architecture
8. Supabase real-time messaging infrastructure
9. Prioritized updates for knowledge base changes
10. Scalable message delivery with reconnection handling
11. Configurable event filtering by content type
12. Client-side caching for offline support
13. Event Types
14. Material creation, update, and deletion events
15. Collection hierarchy modifications
16. Relationship changes
17. Search index updates
18. Version creation
19. Bulk operation completion notifications
20. Implementation Benefits
21. Real-time collaborative editing support
22. Live dashboards with instant updates
23. Immediate notifications for content changes
24. Enhanced user experience with live content
25. Background processing with completion notifications

Optimized for low-latency delivery

Pub/Sub Architecture

Client-side caching for offline support

Event Types

Bulk operation completion notifications

Implementation Benefits

### Technical Implementation

#### Data Models

The Knowledge Base uses the following core data models:

1. Material Model  
   typescript  
    interface MaterialDocument {  
    id: string;  
    name: string;  
    description: string;  
    materialType: string;  
    manufacturer: string;  
    collectionId?: string;  
    seriesId?: string;  
    color?: {  
    name: string;  
    hex: string;  
    rgb: [number, number, number];  
    };  
    dimensions?: {  
    length: number;  
    width: number;  
    height: number;  
    unit: 'mm' | 'cm' | 'in';  
    };  
    weight?: number;  
    finish?: string;  
    price?: {  
    value: number;  
    currency: string;  
    unit: string;  
    };  
    technicalProps?: Record<string, any>;  
    applications?: string[];  
    tags: string[];  
    images: {  
    url: string;  
    type: 'primary' | 'secondary' | 'detail' | 'texture';  
    alt?: string;  
    }[];  
    embeddingVector?: number[];  
    versions: {  
    versionId: string;  
    createdAt: Date;  
    createdBy: string;  
    }[];  
    metadata?: Record<string, any>;  
    createdAt: Date;  
    updatedAt: Date;  
    createdBy: string;  
    }
2. Collection Model  
   typescript  
    interface CollectionDocument {  
    id: string;  
    name: string;  
    description: string;  
    manufacturer: string;  
    parentId?: string;  
    properties?: Record<string, any>;  
    tags: string[];  
    images: {  
    url: string;  
    type: string;  
    alt?: string;  
    }[];  
    metadata?: Record<string, any>;  
    createdAt: Date;  
    updatedAt: Date;  
    createdBy: string;  
    }
3. Relationship Model  
   typescript  
    interface MaterialRelationshipDocument {  
    id: string;  
    sourceMaterialId: string;  
    targetMaterialId: string;  
    relationshipType: 'complementary' | 'alternative' | 'accessory' | 'required' | 'similar' | 'series' | 'variant' | 'entity-reference' | 'custom';  
    strength: number;  
    bidirectional: boolean;  
    context?: string;  
    metadata?: {  
    description?: string;  
    entityType?: string;  
    mentionedText?: string;  
    detectionConfidence?: number;  
    detectionMethod?: 'automatic' | 'manual' | 'suggested';  
    customType?: string;  
    [key: string]: any;  
    };  
    createdAt: Date;  
    updatedAt: Date;  
    createdBy: string;  
    }
4. Collection Membership Model  
   typescript  
    interface CollectionMembershipDocument {  
    id: string;  
    materialId: string;  
    collectionId: string;  
    primaryMembership: boolean;  
    inheritParentProperties: boolean;  
    position: number;  
    path: string[]; // Array representing the path from root to this collection  
    nestingLevel: number; // Depth in the collection hierarchy  
    metadata?: Record<string, any>;  
    addedAt: Date;  
    updatedAt: Date;  
    addedBy: string;  
    }
5. Version Model  
   typescript  
    interface VersionDocument {  
    id: string;  
    entityId: string;  
    entityType: 'material' | 'collection';  
    previousData: Record<string, any>;  
    changeDescription: string;  
    metadata?: Record<string, any>;  
    createdAt: Date;  
    createdBy: string;  
    }
6. Search Index Model  
   typescript  
    interface SearchIndexDocument {  
    id: string;  
    name: string;  
    description: string;  
    entityType: string;  
    indexType: 'text' | 'vector' | 'hybrid';  
    status: 'building' | 'ready' | 'updating' | 'error';  
    documentCount: number;  
    lastBuildTime?: Date;  
    lastUpdateTime?: Date;  
    errorMessage?: string;  
    configuration: Record<string, any>;  
    createdAt: Date;  
    updatedAt: Date;  
    createdBy: string;  
    }

Material Model  
typescript  
 interface MaterialDocument {  
 id: string;  
 name: string;  
 description: string;  
 materialType: string;  
 manufacturer: string;  
 collectionId?: string;  
 seriesId?: string;  
 color?: {  
 name: string;  
 hex: string;  
 rgb: [number, number, number];  
 };  
 dimensions?: {  
 length: number;  
 width: number;  
 height: number;  
 unit: 'mm' | 'cm' | 'in';  
 };  
 weight?: number;  
 finish?: string;  
 price?: {  
 value: number;  
 currency: string;  
 unit: string;  
 };  
 technicalProps?: Record<string, any>;  
 applications?: string[];  
 tags: string[];  
 images: {  
 url: string;  
 type: 'primary' | 'secondary' | 'detail' | 'texture';  
 alt?: string;  
 }[];  
 embeddingVector?: number[];  
 versions: {  
 versionId: string;  
 createdAt: Date;  
 createdBy: string;  
 }[];  
 metadata?: Record<string, any>;  
 createdAt: Date;  
 updatedAt: Date;  
 createdBy: string;  
 }

typescript  
 interface MaterialDocument {  
 id: string;  
 name: string;  
 description: string;  
 materialType: string;  
 manufacturer: string;  
 collectionId?: string;  
 seriesId?: string;  
 color?: {  
 name: string;  
 hex: string;  
 rgb: [number, number, number];  
 };  
 dimensions?: {  
 length: number;  
 width: number;  
 height: number;  
 unit: 'mm' | 'cm' | 'in';  
 };  
 weight?: number;  
 finish?: string;  
 price?: {  
 value: number;  
 currency: string;  
 unit: string;  
 };  
 technicalProps?: Record<string, any>;  
 applications?: string[];  
 tags: string[];  
 images: {  
 url: string;  
 type: 'primary' | 'secondary' | 'detail' | 'texture';  
 alt?: string;  
 }[];  
 embeddingVector?: number[];  
 versions: {  
 versionId: string;  
 createdAt: Date;  
 createdBy: string;  
 }[];  
 metadata?: Record<string, any>;  
 createdAt: Date;  
 updatedAt: Date;  
 createdBy: string;  
 }

Collection Model  
typescript  
 interface CollectionDocument {  
 id: string;  
 name: string;  
 description: string;  
 manufacturer: string;  
 parentId?: string;  
 properties?: Record<string, any>;  
 tags: string[];  
 images: {  
 url: string;  
 type: string;  
 alt?: string;  
 }[];  
 metadata?: Record<string, any>;  
 createdAt: Date;  
 updatedAt: Date;  
 createdBy: string;  
 }

typescript  
 interface CollectionDocument {  
 id: string;  
 name: string;  
 description: string;  
 manufacturer: string;  
 parentId?: string;  
 properties?: Record<string, any>;  
 tags: string[];  
 images: {  
 url: string;  
 type: string;  
 alt?: string;  
 }[];  
 metadata?: Record<string, any>;  
 createdAt: Date;  
 updatedAt: Date;  
 createdBy: string;  
 }

Relationship Model  
typescript  
 interface MaterialRelationshipDocument {  
 id: string;  
 sourceMaterialId: string;  
 targetMaterialId: string;  
 relationshipType: 'complementary' | 'alternative' | 'accessory' | 'required' | 'similar' | 'series' | 'variant' | 'entity-reference' | 'custom';  
 strength: number;  
 bidirectional: boolean;  
 context?: string;  
 metadata?: {  
 description?: string;  
 entityType?: string;  
 mentionedText?: string;  
 detectionConfidence?: number;  
 detectionMethod?: 'automatic' | 'manual' | 'suggested';  
 customType?: string;  
 [key: string]: any;  
 };  
 createdAt: Date;  
 updatedAt: Date;  
 createdBy: string;  
 }

typescript  
 interface MaterialRelationshipDocument {  
 id: string;  
 sourceMaterialId: string;  
 targetMaterialId: string;  
 relationshipType: 'complementary' | 'alternative' | 'accessory' | 'required' | 'similar' | 'series' | 'variant' | 'entity-reference' | 'custom';  
 strength: number;  
 bidirectional: boolean;  
 context?: string;  
 metadata?: {  
 description?: string;  
 entityType?: string;  
 mentionedText?: string;  
 detectionConfidence?: number;  
 detectionMethod?: 'automatic' | 'manual' | 'suggested';  
 customType?: string;  
 [key: string]: any;  
 };  
 createdAt: Date;  
 updatedAt: Date;  
 createdBy: string;  
 }

Collection Membership Model  
typescript  
 interface CollectionMembershipDocument {  
 id: string;  
 materialId: string;  
 collectionId: string;  
 primaryMembership: boolean;  
 inheritParentProperties: boolean;  
 position: number;  
 path: string[]; // Array representing the path from root to this collection  
 nestingLevel: number; // Depth in the collection hierarchy  
 metadata?: Record<string, any>;  
 addedAt: Date;  
 updatedAt: Date;  
 addedBy: string;  
 }

typescript  
 interface CollectionMembershipDocument {  
 id: string;  
 materialId: string;  
 collectionId: string;  
 primaryMembership: boolean;  
 inheritParentProperties: boolean;  
 position: number;  
 path: string[]; // Array representing the path from root to this collection  
 nestingLevel: number; // Depth in the collection hierarchy  
 metadata?: Record<string, any>;  
 addedAt: Date;  
 updatedAt: Date;  
 addedBy: string;  
 }

Version Model  
typescript  
 interface VersionDocument {  
 id: string;  
 entityId: string;  
 entityType: 'material' | 'collection';  
 previousData: Record<string, any>;  
 changeDescription: string;  
 metadata?: Record<string, any>;  
 createdAt: Date;  
 createdBy: string;  
 }

typescript  
 interface VersionDocument {  
 id: string;  
 entityId: string;  
 entityType: 'material' | 'collection';  
 previousData: Record<string, any>;  
 changeDescription: string;  
 metadata?: Record<string, any>;  
 createdAt: Date;  
 createdBy: string;  
 }

Search Index Model  
typescript  
 interface SearchIndexDocument {  
 id: string;  
 name: string;  
 description: string;  
 entityType: string;  
 indexType: 'text' | 'vector' | 'hybrid';  
 status: 'building' | 'ready' | 'updating' | 'error';  
 documentCount: number;  
 lastBuildTime?: Date;  
 lastUpdateTime?: Date;  
 errorMessage?: string;  
 configuration: Record<string, any>;  
 createdAt: Date;  
 updatedAt: Date;  
 createdBy: string;  
 }

typescript  
 interface SearchIndexDocument {  
 id: string;  
 name: string;  
 description: string;  
 entityType: string;  
 indexType: 'text' | 'vector' | 'hybrid';  
 status: 'building' | 'ready' | 'updating' | 'error';  
 documentCount: number;  
 lastBuildTime?: Date;  
 lastUpdateTime?: Date;  
 errorMessage?: string;  
 configuration: Record<string, any>;  
 createdAt: Date;  
 updatedAt: Date;  
 createdBy: string;  
 }

#### Search Implementation

The Knowledge Base implements search through several mechanisms:

1. Text Search
2. MongoDB text indexes for basic search
3. Custom tokenization and stemming
4. Boosting of key fields (name, description, tags)
5. Scoring function customization
6. Vector Search
7. FAISS for efficient similarity search
8. Custom embedding generation from images and text
9. Hybrid retrieval combining vector and text search
10. Quantization for efficient storage and retrieval
11. Search Optimization
12. Caching of frequent queries
13. Specialized indexes for common query patterns
14. Query rewriting for performance
15. Aggregation pipeline optimization

Scoring function customization

Vector Search

Quantization for efficient storage and retrieval

Search Optimization

#### Bulk Operations

The Knowledge Base supports efficient bulk operations with real-time notifications:

1. Import
2. Bulk import of materials with validation
3. Duplicate detection and resolution
4. Relationship inference
5. Collection assignment
6. Batched processing for large datasets
7. Real-time progress and completion notifications
8. Detailed success/failure reporting
9. Update
10. Bulk update of materials matching criteria
11. Field-specific updates
12. Version tracking for bulk changes
13. Cascading updates to relationships
14. Batched processing with error resilience
15. Real-time notifications for updates
16. Export
17. Configurable export formats (JSON, CSV)
18. Filtered exports based on criteria
19. Options for including relationships and versions
20. Compression for large exports
21. Background processing for large exports
22. Delete
23. Soft delete with retention period
24. Hard delete with relationship cleanup
25. Bulk delete with criteria
26. Deletion audit logging
27. Batched processing with failure tracking
28. Real-time notifications for completion
29. Relationship Management
30. Bulk relationship creation with validation
31. Automatic bidirectional relationship handling
32. Batch processing with configurable batch sizes
33. Real-time notification of creation progress
34. Error handling with partial success support

Detailed success/failure reporting

Update

Real-time notifications for updates

Export

Background processing for large exports

Delete

Real-time notifications for completion

Relationship Management

#### Entity Linking

The Entity Linking service automatically identifies relationships between materials:

1. Text Analysis
2. Natural language processing to identify entity mentions
3. Machine learning-based entity recognition in descriptions
4. Material, collection, and property detection
5. Context-aware linking to existing materials and collections
6. Confidence scoring for potential links
7. Configurable confidence thresholds
8. User verification workflow for uncertain links
9. Relationship Creation
10. Automatic creation of relationships based on text analysis
11. Immediate relationship creation upon material creation/update
12. Type inference from context
13. Bidirectionality determination
14. Strength estimation based on context and confidence
15. Background processing for entity linking
16. Detailed metadata for entity-based relationships

User verification workflow for uncertain links

Relationship Creation

### API Usage

#### Material Search

// Search for materials  
const searchResults = await knowledgeBaseService.searchMaterials({  
 query: 'ceramic tile',  
 materialType: 'tile',  
 tags: ['porcelain', 'outdoor'],  
 limit: 10,  
 skip: 0,  
 useVectorSearch: false,  
 searchStrategy: 'combined'  
});  
  
// Results structure  
interface SearchResults {  
 materials: MaterialDocument[];  
 total: number;  
 facets?: {  
 materialTypes: Array<{ \_id: string, count: number }>;  
 manufacturers: Array<{ \_id: string, count: number }>;  
 colors: Array<{ \_id: string, count: number }>;  
 finishes: Array<{ \_id: string, count: number }>;  
 tags: Array<{ \_id: string, count: number }>;  
 };  
}

// Search for materials  
const searchResults = await knowledgeBaseService.searchMaterials({  
 query: 'ceramic tile',  
 materialType: 'tile',  
 tags: ['porcelain', 'outdoor'],  
 limit: 10,  
 skip: 0,  
 useVectorSearch: false,  
 searchStrategy: 'combined'  
});  
  
// Results structure  
interface SearchResults {  
 materials: MaterialDocument[];  
 total: number;  
 facets?: {  
 materialTypes: Array<{ \_id: string, count: number }>;  
 manufacturers: Array<{ \_id: string, count: number }>;  
 colors: Array<{ \_id: string, count: number }>;  
 finishes: Array<{ \_id: string, count: number }>;  
 tags: Array<{ \_id: string, count: number }>;  
 };  
}

#### Collection Management

// Get collections with material counts  
const collections = await knowledgeBaseService.getCollections({  
 parentId: 'parent-collection-id',  
 includeEmpty: false,  
 limit: 20  
});  
  
// Results structure  
interface CollectionResults {  
 collections: Array<CollectionDocument & { materialCount: number }>;  
 total: number;  
}

// Get collections with material counts  
const collections = await knowledgeBaseService.getCollections({  
 parentId: 'parent-collection-id',  
 includeEmpty: false,  
 limit: 20  
});  
  
// Results structure  
interface CollectionResults {  
 collections: Array<CollectionDocument & { materialCount: number }>;  
 total: number;  
}

#### Material Versioning

// Create a material revision  
const updatedMaterial = await knowledgeBaseService.createMaterialRevision(  
 'material-id',  
 { name: 'Updated Name', description: 'New description' },  
 'user-id'  
);  
  
// Revert to a previous version  
const revertedMaterial = await knowledgeBaseService.revertMaterialVersion(  
 'material-id',  
 'version-id',  
 'user-id'  
);  
  
// Get version history  
const versions = await knowledgeBaseService.getMaterialVersionHistory('material-id');

// Create a material revision  
const updatedMaterial = await knowledgeBaseService.createMaterialRevision(  
 'material-id',  
 { name: 'Updated Name', description: 'New description' },  
 'user-id'  
);  
  
// Revert to a previous version  
const revertedMaterial = await knowledgeBaseService.revertMaterialVersion(  
 'material-id',  
 'version-id',  
 'user-id'  
);  
  
// Get version history  
const versions = await knowledgeBaseService.getMaterialVersionHistory('material-id');

#### Bulk Operations

// Bulk import materials with real-time notifications  
const importResults = await realTimeKnowledgeBaseService.bulkImportMaterials(  
 materialsArray,  
 {  
 updateExisting: true,  
 detectDuplicates: true,  
 validateSchema: true,  
 collectionId: 'collection-id',  
 userId: 'user-id'  
 }  
);  
  
// Subscribe to real-time updates  
scalableMessageBroker.subscribeWithOptions(  
 'system',  
 (message) => {  
 if (message.type === MessageType.KNOWLEDGE\_BASE\_EVENT) {  
 const { eventType, payload } = message.data;  
  
 if (eventType === KnowledgeBaseEventType.BULK\_OPERATION\_COMPLETED) {  
 console.log(`Bulk ${payload.operationType} completed: ${payload.count}/${payload.totalRequested}`);  
 }  
 }  
 },  
 {  
 useAcknowledgment: true,  
 autoAcknowledge: true,  
 enableCache: true  
 }  
);  
  
// Bulk update materials  
const updateResults = await knowledgeBaseService.bulkUpdateMaterials(  
 { tags: ['updated', 'batch-processed'] },  
 { materialType: 'tile' },  
 'user-id'  
);  
  
// Bulk export materials  
const exportResults = await knowledgeBaseService.bulkExportMaterials(  
 { materialType: 'tile' },  
 {  
 format: 'json',  
 includeRelationships: true,  
 includeVersions: false  
 }  
);

// Bulk import materials with real-time notifications  
const importResults = await realTimeKnowledgeBaseService.bulkImportMaterials(  
 materialsArray,  
 {  
 updateExisting: true,  
 detectDuplicates: true,  
 validateSchema: true,  
 collectionId: 'collection-id',  
 userId: 'user-id'  
 }  
);  
  
// Subscribe to real-time updates  
scalableMessageBroker.subscribeWithOptions(  
 'system',  
 (message) => {  
 if (message.type === MessageType.KNOWLEDGE\_BASE\_EVENT) {  
 const { eventType, payload } = message.data;  
  
 if (eventType === KnowledgeBaseEventType.BULK\_OPERATION\_COMPLETED) {  
 console.log(`Bulk ${payload.operationType} completed: ${payload.count}/${payload.totalRequested}`);  
 }  
 }  
 },  
 {  
 useAcknowledgment: true,  
 autoAcknowledge: true,  
 enableCache: true  
 }  
);  
  
// Bulk update materials  
const updateResults = await knowledgeBaseService.bulkUpdateMaterials(  
 { tags: ['updated', 'batch-processed'] },  
 { materialType: 'tile' },  
 'user-id'  
);  
  
// Bulk export materials  
const exportResults = await knowledgeBaseService.bulkExportMaterials(  
 { materialType: 'tile' },  
 {  
 format: 'json',  
 includeRelationships: true,  
 includeVersions: false  
 }  
);

#### Relationship Management

// Create material relationships  
const relationshipResults = await knowledgeBaseService.bulkCreateRelationships(  
 [  
 {  
 sourceMaterialId: 'material-1',  
 targetMaterialId: 'material-2',  
 relationshipType: 'complementary',  
 strength: 0.85,  
 bidirectional: true  
 },  
 {  
 sourceMaterialId: 'material-1',  
 targetMaterialId: 'material-3',  
 relationshipType: 'alternative',  
 strength: 0.75,  
 bidirectional: false  
 }  
 ],  
 'user-id'  
);

// Create material relationships  
const relationshipResults = await knowledgeBaseService.bulkCreateRelationships(  
 [  
 {  
 sourceMaterialId: 'material-1',  
 targetMaterialId: 'material-2',  
 relationshipType: 'complementary',  
 strength: 0.85,  
 bidirectional: true  
 },  
 {  
 sourceMaterialId: 'material-1',  
 targetMaterialId: 'material-3',  
 relationshipType: 'alternative',  
 strength: 0.75,  
 bidirectional: false  
 }  
 ],  
 'user-id'  
);

#### System Statistics

// Get knowledge base statistics  
const stats = await knowledgeBaseService.getKnowledgeBaseStats();  
  
// Stats structure  
interface KnowledgeBaseStats {  
 materialCount: number;  
 collectionCount: number;  
 materialsByType: Record<string, number>;  
 materialsByCollection: Array<{ collection: string; count: number }>;  
 recentUpdates: number;  
 indexingStatus: Record<string, number>;  
}

// Get knowledge base statistics  
const stats = await knowledgeBaseService.getKnowledgeBaseStats();  
  
// Stats structure  
interface KnowledgeBaseStats {  
 materialCount: number;  
 collectionCount: number;  
 materialsByType: Record<string, number>;  
 materialsByCollection: Array<{ collection: string; count: number }>;  
 recentUpdates: number;  
 indexingStatus: Record<string, number>;  
}

### Performance Considerations

1. Query Optimization
2. Indexing strategy for common query patterns
3. Query caching for repeat searches
4. Pagination and limiting for large result sets
5. Query rewriting for performance
6. Scaling Considerations
7. Horizontal scaling for read-heavy workloads
8. Database sharding for large material collections
9. Indexing optimizations for different query patterns
10. Caching layers for frequently accessed data
11. Resource Requirements
12. Storage: Scales with number of materials and their media assets
13. Memory: Depends on indexing strategy and caching
14. CPU: Primarily for search and bulk operations
15. Network: Important for media delivery and distributed search
16. Optimization Techniques
17. Lazy loading of large media assets
18. Progressive loading of search results
19. Asynchronous processing of bulk operations
20. Background indexing and reindexing

Query rewriting for performance

Scaling Considerations

Caching layers for frequently accessed data

Resource Requirements

Network: Important for media delivery and distributed search

Optimization Techniques

# Kubernetes Architecture

Source: readme/kubernetes-architecture.md

---

## Kubernetes Architecture and Implementation Guide

This comprehensive document details the Kubernetes architecture, implementation, and operational aspects of the KAI ML Platform. It covers the system design, component architecture, deployment processes, administration dashboard, and operational considerations.

### Table of Contents

1. Overview
2. Core Architecture Components
3. Namespace and Organization
4. Node Pools and Resource Management
5. Workflow Orchestration
6. TypeScript Services Implementation
7. Deployment Architecture
8. GitOps Integration
9. Kubernetes Deployment Dashboard
10. Security
11. Performance Tuning
12. Scaling Strategies
13. High Availability and Disaster Recovery
14. Monitoring and Observability
15. Troubleshooting
16. API Reference
17. Conclusion

### Overview

The KAI ML Platform uses a dedicated Kubernetes architecture optimized for machine learning workloads, with specialized components for orchestration, processing, and resource management. The deployment is managed through a structured approach using Helm charts and Flux GitOps, which applies configurations in the correct order and handles environment-specific settings.

This implementation fulfills key requirements for a scalable, resilient, and cost-effective architecture for ML processing pipelines. The system intelligently adapts to workload characteristics, resource availability, and user requirements, ensuring optimal performance while maintaining efficiency.

### Core Architecture Components

The KAI ML Platform architecture in Kubernetes consists of these major components:

┌─────────────────────┐ ┌─────────────────────┐ ┌─────────────────────┐  
│ │ │ │ │ │  
│ Coordinator │────▶│ Distributed │────▶│ ML Services │  
│ Service │ │ Processing │ │ (GPU) │  
│ │ │ │ │ │  
└─────────────────────┘ └─────────────────────┘ └─────────────────────┘  
 │ │ │  
 │ │ │  
 ▼ ▼ ▼  
┌─────────────────────┐ ┌─────────────────────┐ ┌─────────────────────┐  
│ │ │ │ │ │  
│ Monitoring │ │ Caching │ │ Mobile │  
│ (Prometheus) │ │ (Redis) │ │ Optimization │  
│ │ │ │ │ │  
└─────────────────────┘ └─────────────────────┘ └─────────────────────┘  
 │ │  
 │ │  
 ▼ ▼  
 ┌─────────────────────┐ ┌─────────────────────┐  
 │ │ │ │  
 │ WASM │ │ Argo │  
 │ Compiler │ │ Workflows │  
 │ │ │ │  
 └─────────────────────┘ └─────────────────────┘

┌─────────────────────┐ ┌─────────────────────┐ ┌─────────────────────┐  
│ │ │ │ │ │  
│ Coordinator │────▶│ Distributed │────▶│ ML Services │  
│ Service │ │ Processing │ │ (GPU) │  
│ │ │ │ │ │  
└─────────────────────┘ └─────────────────────┘ └─────────────────────┘  
 │ │ │  
 │ │ │  
 ▼ ▼ ▼  
┌─────────────────────┐ ┌─────────────────────┐ ┌─────────────────────┐  
│ │ │ │ │ │  
│ Monitoring │ │ Caching │ │ Mobile │  
│ (Prometheus) │ │ (Redis) │ │ Optimization │  
│ │ │ │ │ │  
└─────────────────────┘ └─────────────────────┘ └─────────────────────┘  
 │ │  
 │ │  
 ▼ ▼  
 ┌─────────────────────┐ ┌─────────────────────┐  
 │ │ │ │  
 │ WASM │ │ Argo │  
 │ Compiler │ │ Workflows │  
 │ │ │ │  
 └─────────────────────┘ └─────────────────────┘

#### 1. Coordinator Service

The coordinator service is the central orchestration component that:

* Manages task queues and workflow scheduling
* Handles resource allocation across ML workloads
* Interfaces with Argo Workflows for pipeline execution
* Provides API endpoints for system operations

Deployment manifests:  
- kubernetes/coordinator/rbac.yaml: Service account and permissions  
- kubernetes/coordinator/config.yaml: Configuration parameters  
- kubernetes/coordinator/service.yaml: Service definition  
- kubernetes/coordinator/deployment.yaml: Pod deployment  
- kubernetes/coordinator/hpa.yaml: Horizontal Pod Autoscaler  
- kubernetes/coordinator/pdb.yaml: Pod Disruption Budget

kubernetes/coordinator/rbac.yaml

kubernetes/coordinator/config.yaml

kubernetes/coordinator/service.yaml

kubernetes/coordinator/deployment.yaml

kubernetes/coordinator/hpa.yaml

kubernetes/coordinator/pdb.yaml

#### 2. Distributed Processing

Handles distributed workloads across the cluster:

* Manages task distribution and load balancing
* Processes large ML jobs in parallel
* Coordinates work distribution among worker nodes

Deployment manifests:  
- kubernetes/distributed-processing/deployment.yaml: Worker pods  
- kubernetes/distributed-processing/pdb.yaml: Pod Disruption Budget  
- kubernetes/distributed-processing/secret.yaml: Processing secrets

kubernetes/distributed-processing/deployment.yaml

kubernetes/distributed-processing/pdb.yaml

kubernetes/distributed-processing/secret.yaml

#### 3. ML Services (GPU)

Specialized GPU-accelerated services for ML model training and inference:

* Domain-specific network services
* Multimodal pattern recognition
* Real-time inference endpoints

Deployment manifests:  
- kubernetes/ml-services/domain-specific-networks-deployment.yaml: Domain-specific ML network services  
- kubernetes/ml-services/multimodal-pattern-recognition-deployment.yaml: Pattern recognition services

kubernetes/ml-services/domain-specific-networks-deployment.yaml

kubernetes/ml-services/multimodal-pattern-recognition-deployment.yaml

#### 4. Mobile Optimization

Specialized services for optimizing ML models for mobile deployment:

* Model quantization and compression
* LOD (Level of Detail) generation
* Draco mesh compression

Deployment manifests:  
- kubernetes/mobile-optimization/deployment.yaml: Service pods  
- kubernetes/mobile-optimization/hpa.yaml: Horizontal Pod Autoscaler  
- kubernetes/mobile-optimization/pdb.yaml: Pod Disruption Budget

kubernetes/mobile-optimization/deployment.yaml

kubernetes/mobile-optimization/hpa.yaml

kubernetes/mobile-optimization/pdb.yaml

#### 5. WASM Compiler

WebAssembly compilation service for client-side ML models:

* Compiles ML models to WebAssembly
* Optimizes for browser execution
* Manages versioning and compilation profiles

Deployment manifests:  
- kubernetes/wasm-compiler/deployment.yaml: Compiler service  
- kubernetes/wasm-compiler/hpa.yaml: Horizontal Pod Autoscaler  
- kubernetes/wasm-compiler/pdb.yaml: Pod Disruption Budget

kubernetes/wasm-compiler/deployment.yaml

kubernetes/wasm-compiler/hpa.yaml

kubernetes/wasm-compiler/pdb.yaml

#### 6. Workflow Templates

Argo Workflow templates for standard ML pipelines:

* 3D reconstruction pipeline
* Training pipelines
* Batch processing workflows
* Data transformation workflows

Deployment manifests:  
- kubernetes/workflows/3d-reconstruction-template.yaml: 3D reconstruction workflow  
- kubernetes/workflows/domain-specific-networks-template.yaml: Training workflow for domain-specific networks  
- kubernetes/workflows/multimodal-pattern-recognition-template.yaml: Processing workflow for pattern recognition

kubernetes/workflows/3d-reconstruction-template.yaml

kubernetes/workflows/domain-specific-networks-template.yaml

kubernetes/workflows/multimodal-pattern-recognition-template.yaml

### Namespace and Organization

All KAI components are deployed within a dedicated kai-ml namespace to isolate resources and permissions. The namespace includes:

kai-ml

* Resource Quotas: Limiting total CPU, memory, GPU, and storage resources
* Default Limits: Setting default resource constraints for containers
* Labels: Identifying components as part of the KAI ML platform

# Resource quotas for the kai-ml namespace  
apiVersion: v1  
kind: ResourceQuota  
metadata:  
 name: kai-ml-quota  
 namespace: kai-ml  
spec:  
 hard:  
 # Pod limits  
 pods: "100"  
 # CPU limits  
 requests.cpu: "100"  
 limits.cpu: "200"  
 # Memory limits  
 requests.memory: 200Gi  
 limits.memory: 400Gi  
 # GPU limits  
 requests.nvidia.com/gpu: "16"  
 limits.nvidia.com/gpu: "16"

# Resource quotas for the kai-ml namespace  
apiVersion: v1  
kind: ResourceQuota  
metadata:  
 name: kai-ml-quota  
 namespace: kai-ml  
spec:  
 hard:  
 # Pod limits  
 pods: "100"  
 # CPU limits  
 requests.cpu: "100"  
 limits.cpu: "200"  
 # Memory limits  
 requests.memory: 200Gi  
 limits.memory: 400Gi  
 # GPU limits  
 requests.nvidia.com/gpu: "16"  
 limits.nvidia.com/gpu: "16"

### Node Pools and Resource Management

The Kubernetes cluster uses specialized node pools to optimize resource allocation:

#### Node Pool Structure

Kubernetes Cluster  
│  
├── cpu-optimized pool  
│ ├── General processing nodes  
│ └── API services  
│  
├── gpu-optimized pool  
│ ├── ML inference nodes (T4 GPUs)  
│ └── Real-time processing  
│  
├── gpu-high-end pool  
│ ├── ML training nodes (A100 GPUs)  
│ └── Complex model generation  
│  
├── memory-optimized pool  
│ ├── Large model loading nodes  
│ └── In-memory processing  
│  
├── storage-optimized pool  
│ ├── Data-intensive operation nodes  
│ └── Caching services  
│  
├── orchestration pool  
│ ├── Control plane service nodes  
│ └── Coordinator services  
│  
└── spot-instances pool  
 ├── Cost-effective batch processing nodes  
 └── Non-critical background tasks

Kubernetes Cluster  
│  
├── cpu-optimized pool  
│ ├── General processing nodes  
│ └── API services  
│  
├── gpu-optimized pool  
│ ├── ML inference nodes (T4 GPUs)  
│ └── Real-time processing  
│  
├── gpu-high-end pool  
│ ├── ML training nodes (A100 GPUs)  
│ └── Complex model generation  
│  
├── memory-optimized pool  
│ ├── Large model loading nodes  
│ └── In-memory processing  
│  
├── storage-optimized pool  
│ ├── Data-intensive operation nodes  
│ └── Caching services  
│  
├── orchestration pool  
│ ├── Control plane service nodes  
│ └── Coordinator services  
│  
└── spot-instances pool  
 ├── Cost-effective batch processing nodes  
 └── Non-critical background tasks

Each node pool is configured with labels and taints to ensure proper workload scheduling:

# Example of node pool labels and taints  
labels:  
 node-type: gpu-optimized  
 workload-class: ml-inference  
 gpu: nvidia-t4  
taints:  
 - key: "node-type"  
 value: "gpu-optimized"  
 effect: "NoSchedule"

# Example of node pool labels and taints  
labels:  
 node-type: gpu-optimized  
 workload-class: ml-inference  
 gpu: nvidia-t4  
taints:  
 - key: "node-type"  
 value: "gpu-optimized"  
 effect: "NoSchedule"

#### Priority Classes

The system uses priority classes to ensure critical components get resources first:

1. system-critical (1,000,000): Essential components that must not be preempted
2. interactive (100,000): User-facing requests requiring low latency
3. high-priority-batch (50,000): Important batch jobs
4. medium-priority-batch (10,000): Normal batch jobs (default)
5. low-priority-batch (1,000): Non-urgent batch jobs
6. maintenance (100): System maintenance tasks
7. preemptible (0): Jobs that can run on spot/preemptible instances

These priority classes ensure proper resource allocation during contention:

apiVersion: scheduling.k8s.io/v1  
kind: PriorityClass  
metadata:  
 name: interactive  
value: 100000  
globalDefault: false  
description: "This priority class is used for interactive user requests that require low latency."

apiVersion: scheduling.k8s.io/v1  
kind: PriorityClass  
metadata:  
 name: interactive  
value: 100000  
globalDefault: false  
description: "This priority class is used for interactive user requests that require low latency."

#### Key Requirement Mappings

##### Intelligent Resource Allocation

* Requirement: ML-aware pod placement and dynamic resource requests
* Implementation:
* ResourceManager service for intelligent allocation
* Node affinity rules in workflow templates
* GPU memory optimization with custom scheduling
* Specialized node pools for different ML tasks

##### Smart Job Prioritization

* Requirement: Priority classes and fair-sharing for multi-tenant workloads
* Implementation:
* Priority classes defined in infrastructure
* Subscription tier-based priority assignment
* Preemptive scheduling for interactive requests
* Queue-based throttling in Coordinator Service

##### Cost Optimization Strategies

* Requirement: Spot instances, multi-cloud, predictive scaling
* Implementation:
* Node pool configurations for preemptible instances
* Resource allocation based on workload characteristics
* Dynamic scaling based on queue depth
* Workload consolidation mechanisms

### Workflow Orchestration

#### Advanced Workflow Orchestration

* Requirement: Replace sequential job creation with declarative workflow definitions
* Implementation:
* Argo Workflows integration in Coordinator Service
* Conditional paths based on QualityManager assessments
* Parallel processing steps in workflow templates
* DAG-based workflow definitions

#### Argo Workflows Integration

Argo Workflows is used for orchestrating complex ML pipelines:

##### Workflow Example: 3D Reconstruction

The 3D reconstruction workflow demonstrates how Argo Workflows manages complex ML pipelines:

1. Quality Assessment: Evaluates input images and determines processing quality level
2. Branching Logic: Routes to low, medium, or high quality pipeline based on assessment
3. Parallel Processing: Processes images in parallel when possible
4. Resource Allocation: Assigns appropriate resources to each step
5. Artifact Management: Handles intermediary data between steps

┌───────────────┐  
 │ │  
 │ Assess Quality│  
 │ │  
 └───────┬───────┘  
 │  
 ┌───────────────────┼───────────────────┐  
 │ │ │  
 ┌────────▼─────────┐ ┌───────▼────────┐ ┌───────▼────────┐  
 │ │ │ │ │ │  
 │ Low Quality │ │ Medium Quality │ │ High Quality │  
 │ Pipeline │ │ Pipeline │ │ Pipeline │  
 │ │ │ │ │ │  
 └────────┬─────────┘ └───────┬────────┘ └───────┬────────┘  
 │ │ │  
 └───────────────────┼───────────────────┘  
 │  
 ┌───────▼───────┐  
 │ │  
 │ Format │  
 │ Conversion │  
 │ │  
 └───────┬───────┘  
 │  
 ┌───────▼───────┐  
 │ │  
 │ Finalize │  
 │ │  
 └───────────────┘

┌───────────────┐  
 │ │  
 │ Assess Quality│  
 │ │  
 └───────┬───────┘  
 │  
 ┌───────────────────┼───────────────────┐  
 │ │ │  
 ┌────────▼─────────┐ ┌───────▼────────┐ ┌───────▼────────┐  
 │ │ │ │ │ │  
 │ Low Quality │ │ Medium Quality │ │ High Quality │  
 │ Pipeline │ │ Pipeline │ │ Pipeline │  
 │ │ │ │ │ │  
 └────────┬─────────┘ └───────┬────────┘ └───────┬────────┘  
 │ │ │  
 └───────────────────┼───────────────────┘  
 │  
 ┌───────▼───────┐  
 │ │  
 │ Format │  
 │ Conversion │  
 │ │  
 └───────┬───────┘  
 │  
 ┌───────▼───────┐  
 │ │  
 │ Finalize │  
 │ │  
 └───────────────┘

#### Progressive Enhancement Architecture

* Implementation: QualityManager for determining appropriate quality levels
* Quality Tiers:
* Low: Basic processing, minimal resources
* Medium: Standard quality, balanced resources
* High: Premium quality, intensive resources
* Graceful Degradation: Automatic quality reduction under resource constraints

#### Distributed Processing Framework

* Implementation: TaskQueueManager within Coordinator Service
* Workflow Management: Argo Workflows for complex orchestration
* Task Prioritization: Priority classes and subscription tier-based allocation
* Error Recovery: Monitoring and automatic retries

#### Resilience and Fault Tolerance

* Requirement: Automatic retry, checkpointing, and circuit breakers
* Implementation:
* Retry mechanisms in workflow templates
* Redis-based checkpoint/resume capability
* Circuit breakers for external dependencies
* Persistent storage for intermediate results

### TypeScript Services Implementation

#### 1. CoordinatorService (packages/coordinator/src/services/coordinator.service.ts)

packages/coordinator/src/services/coordinator.service.ts

* Handles creation and management of workflow instances
* Interfaces with Kubernetes API to create Argo Workflows
* Implements caching, monitoring, and resource management integration

#### 2. QualityManager (packages/coordinator/src/services/quality-manager.service.ts)

packages/coordinator/src/services/quality-manager.service.ts

* Evaluates appropriate quality level for processing based on:
* Input characteristics (size, complexity)
* Available resources in the cluster
* User's subscription tier
* Historical processing performance

#### 3. ResourceManager (packages/coordinator/src/services/resource-manager.service.ts)

packages/coordinator/src/services/resource-manager.service.ts

* Allocates appropriate resources (CPU, memory, GPU) based on:
* Quality level determined by QualityManager
* Priority of the request
* Current cluster resource availability
* Specialized node requirements

#### 4. CacheManager (packages/coordinator/src/services/cache-manager.service.ts)

packages/coordinator/src/services/cache-manager.service.ts

* Implements Redis-based caching of workflow results
* Provides content-addressable storage via hash-based keys
* Supports invalidation by type and TTL-based expiration
* Implements efficient batch operations for large-scale invalidation

#### 5. MonitoringService (packages/coordinator/src/services/monitoring.service.ts)

packages/coordinator/src/services/monitoring.service.ts

* Records metrics about workflow creation, completion, and errors
* Tracks quality levels, resource allocation, and cache hit rates
* Integrates with Prometheus for metrics collection
* Provides detailed performance analysis for workflows

#### Smart Caching Infrastructure

* Requirement: Intermediate result caching, content-addressable storage
* Implementation:
* CacheManager service for result caching
* Redis-based caching layer
* Content-addressable storage via hash-based keys
* Tiered caching strategy

### Deployment Architecture

#### Helm-Based Deployment Architecture

The KAI Platform has migrated from script-based deployments to a structured Helm chart architecture, significantly improving consistency, maintainability, and deployment reliability:

./helm-charts/helm-deploy.sh --context=my-k8s-context --registry=my-registry.example.com --tag=v1.2.3 --env=staging --release=kai-staging

./helm-charts/helm-deploy.sh --context=my-k8s-context --registry=my-registry.example.com --tag=v1.2.3 --env=staging --release=kai-staging

##### Helm Chart Structure

The platform uses a parent-child chart structure to modularize components while maintaining centralized configuration:

helm-charts/  
├── kai/ # Main parent chart  
│ ├── Chart.yaml # Defines dependencies on subcharts  
│ ├── values.yaml # Default values for all components  
│ ├── values-staging.yaml # Staging-specific overrides  
│ └── values-production.yaml # Production-specific overrides  
├── coordinator/ # Coordinator service subchart  
│ ├── Chart.yaml  
│ ├── values.yaml # Coordinator-specific defaults  
│ └── templates/ # Kubernetes manifest templates  
│ ├── \_helpers.tpl # Reusable template functions  
│ ├── deployment.yaml  
│ ├── service.yaml  
│ ├── hpa.yaml  
│ ├── pdb.yaml  
│ ├── rbac.yaml  
│ └── configmap.yaml  
├── mobile-optimization/ # Mobile optimization subchart  
│ └── ...  
├── wasm-compiler/ # WASM compiler subchart  
│ └── ...  
└── infrastructure/ # Shared infrastructure subchart  
 └── ...

helm-charts/  
├── kai/ # Main parent chart  
│ ├── Chart.yaml # Defines dependencies on subcharts  
│ ├── values.yaml # Default values for all components  
│ ├── values-staging.yaml # Staging-specific overrides  
│ └── values-production.yaml # Production-specific overrides  
├── coordinator/ # Coordinator service subchart  
│ ├── Chart.yaml  
│ ├── values.yaml # Coordinator-specific defaults  
│ └── templates/ # Kubernetes manifest templates  
│ ├── \_helpers.tpl # Reusable template functions  
│ ├── deployment.yaml  
│ ├── service.yaml  
│ ├── hpa.yaml  
│ ├── pdb.yaml  
│ ├── rbac.yaml  
│ └── configmap.yaml  
├── mobile-optimization/ # Mobile optimization subchart  
│ └── ...  
├── wasm-compiler/ # WASM compiler subchart  
│ └── ...  
└── infrastructure/ # Shared infrastructure subchart  
 └── ...

##### Enhanced Deployment Capabilities

The Helm-based deployment system offers significant advantages:

1. Templated Resources: All Kubernetes manifests are generated from templates, ensuring consistency across environments
2. Declarative Configuration: Resources are defined declaratively, making changes more predictable
3. Dependency Management: Charts define dependencies, ensuring proper deployment order
4. Atomic Deployments: Changes are applied as atomic operations, preventing partial updates
5. Versioned Releases: Each deployment creates a versioned release for auditing and rollbacks
6. Enhanced Rollbacks: Helm's native rollback mechanism restores all resources to a consistent state

Example rollback command:

./helm-charts/helm-deploy.sh --context=my-k8s-context --env=production --release=kai-production --rollback=3

./helm-charts/helm-deploy.sh --context=my-k8s-context --env=production --release=kai-production --rollback=3

##### Deployment Script Integration

The helm-charts/helm-deploy.sh script provides a user-friendly interface that integrates with our CI/CD pipeline:

helm-charts/helm-deploy.sh

# Deploy to staging environment  
./helm-charts/helm-deploy.sh --context=kai-staging-cluster --registry=your-registry.example.com --tag=v1.2.3 --env=staging --release=kai-staging  
  
# Deploy to production environment  
./helm-charts/helm-deploy.sh --context=kai-production-cluster --registry=your-registry.example.com --tag=v1.2.3 --env=production --release=kai-production  
  
# View release history  
./helm-charts/helm-deploy.sh --list-versions --release=kai-production  
  
# Rollback to a previous release version  
./helm-charts/helm-deploy.sh --context=kai-production-cluster --env=production --release=kai-production --rollback=3

# Deploy to staging environment  
./helm-charts/helm-deploy.sh --context=kai-staging-cluster --registry=your-registry.example.com --tag=v1.2.3 --env=staging --release=kai-staging  
  
# Deploy to production environment  
./helm-charts/helm-deploy.sh --context=kai-production-cluster --registry=your-registry.example.com --tag=v1.2.3 --env=production --release=kai-production  
  
# View release history  
./helm-charts/helm-deploy.sh --list-versions --release=kai-production  
  
# Rollback to a previous release version  
./helm-charts/helm-deploy.sh --context=kai-production-cluster --env=production --release=kai-production --rollback=3

Key script features:  
- Environment Detection: Automatically selects appropriate values file based on environment  
- Release Management: Handles Helm release lifecycle including upgrades and rollbacks  
- Health Verification: Verifies deployment health before completion  
- Resource Reporting: Provides detailed reports on deployed resources  
- History Management: Manages release history for auditing and rollback  
- Zero-downtime Updates: Ensures smooth transitions between versions

##### Environment-Specific Configuration with Helm Values

The Helm-based system replaces script variables with structured values files:

values.yaml (default values):

global:  
 environment: "staging"  
 namespace: "kai-system-staging"  
 resourceMultiplier: 1  
  
coordinator:  
 replicaCount: 1  
 minReplicas: 1  
 maxReplicas: 5  
 pdbMinAvailable: 1  
 resources:  
 requests:  
 cpu: "200m"  
 memory: "512Mi"  
 limits:  
 cpu: "1000m"  
 memory: "2Gi"

global:  
 environment: "staging"  
 namespace: "kai-system-staging"  
 resourceMultiplier: 1  
  
coordinator:  
 replicaCount: 1  
 minReplicas: 1  
 maxReplicas: 5  
 pdbMinAvailable: 1  
 resources:  
 requests:  
 cpu: "200m"  
 memory: "512Mi"  
 limits:  
 cpu: "1000m"  
 memory: "2Gi"

values-production.yaml (production overrides):

global:  
 environment: "production"  
 namespace: "kai-system"  
 resourceMultiplier: 2  
  
coordinator:  
 replicaCount: 3  
 minReplicas: 2  
 maxReplicas: 10  
 pdbMinAvailable: 2  
 resources:  
 requests:  
 cpu: "500m"  
 memory: "1Gi"  
 limits:  
 cpu: "2000m"  
 memory: "4Gi"

global:  
 environment: "production"  
 namespace: "kai-system"  
 resourceMultiplier: 2  
  
coordinator:  
 replicaCount: 3  
 minReplicas: 2  
 maxReplicas: 10  
 pdbMinAvailable: 2  
 resources:  
 requests:  
 cpu: "500m"  
 memory: "1Gi"  
 limits:  
 cpu: "2000m"  
 memory: "4Gi"

This approach provides several key benefits:  
- Centralized Configuration Management: All environment differences are defined in dedicated values files  
- Simplified Maintenance: Common settings are defined once in default values  
- Reduced Duplication: Environment-specific values only override what's different from defaults  
- Type Safety: Helm has built-in validation for value types  
- Greater Consistency: Less risk of configuration drift between environments  
- Template-driven Generation: Values are inserted into templates, ensuring consistent structure  
- Validation: Helm validates values against schema before deployment

##### Built-in Release History and Versioning

Helm maintains a complete history of all releases, enabling:

1. Auditing: Every change is recorded with timestamp, username, and detailed manifests
2. Comparison: Differences between releases can be easily identified
3. Selective Rollback: The system can roll back to any previous release, not just the last one
4. Release Notes: Each deployment can include annotated notes for operational reference
5. Revision Management: Old revisions can be automatically purged based on retention policies

# List all release versions  
helm history kai-production  
  
# View details of a specific release  
helm get all kai-production --revision=2

# List all release versions  
helm history kai-production  
  
# View details of a specific release  
helm get all kai-production --revision=2

##### Integration with CI/CD Pipeline

The GitHub Actions workflow seamlessly integrates with the Helm deployment system:

- name: Deploy to Kubernetes with Helm  
 run: |  
 ./helm-charts/helm-deploy.sh \  
 --context=${{ env.KUBE\_CONTEXT }} \  
 --registry=ghcr.io/${{ github.repository }} \  
 --tag=${{ github.sha }} \  
 --env=${{ env.DEPLOY\_ENV }} \  
 --release=kai-${{ env.DEPLOY\_ENV }}

- name: Deploy to Kubernetes with Helm  
 run: |  
 ./helm-charts/helm-deploy.sh \  
 --context=${{ env.KUBE\_CONTEXT }} \  
 --registry=ghcr.io/${{ github.repository }} \  
 --tag=${{ github.sha }} \  
 --env=${{ env.DEPLOY\_ENV }} \  
 --release=kai-${{ env.DEPLOY\_ENV }}

This integration provides:  
- Deterministic Deployments: Same inputs produce the same deployed state  
- Simplified Rollbacks: Failed deployments can be easily rolled back  
- Environment Promotion: Configurations can be promoted between environments  
- Deployment Artifacts: Release manifests are preserved for auditing  
- Health Monitoring: Automatic verification of deployment success  
- Reduced Drift: Configuration is version-controlled and applied consistently

### GitOps Integration

#### Flux GitOps Architecture

The KAI Platform implements a GitOps approach using Flux CD, providing a declarative and automated way to manage Kubernetes resources.

##### Flux Controllers and Architecture

The Flux GitOps implementation consists of several controllers running in the Kubernetes cluster:

┌──────────────────────────────────────────────────────────┐  
│ Kubernetes Cluster │  
│ │  
│ ┌─────────────────┐ ┌─────────────────┐ │  
│ │ │ │ │ │  
│ │ Source │──▶│ Kustomize │───┐ │  
│ │ Controller │ │ Controller │ │ │  
│ │ │ │ │ │ │  
│ └─────────────────┘ └─────────────────┘ │ │  
│ │ │ │  
│ │ ┌─────────────────┐ │ │  
│ └─────────────▶│ │ │ │  
│ │ Helm │ │ │  
│ │ Controller │───┼──▶ Apply │  
│ │ │ │ Changes│  
│ ┌─────────────────┐ └─────────────────┘ │ │  
│ │ │ │ │  
│ │ Notification │◀─────────────────────────┘ │  
│ │ Controller │ │  
│ │ │ │  
│ └─────────────────┘ │  
│ │  
└──────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────┐  
│ Kubernetes Cluster │  
│ │  
│ ┌─────────────────┐ ┌─────────────────┐ │  
│ │ │ │ │ │  
│ │ Source │──▶│ Kustomize │───┐ │  
│ │ Controller │ │ Controller │ │ │  
│ │ │ │ │ │ │  
│ └─────────────────┘ └─────────────────┘ │ │  
│ │ │ │  
│ │ ┌─────────────────┐ │ │  
│ └─────────────▶│ │ │ │  
│ │ Helm │ │ │  
│ │ Controller │───┼──▶ Apply │  
│ │ │ │ Changes│  
│ ┌─────────────────┐ └─────────────────┘ │ │  
│ │ │ │ │  
│ │ Notification │◀─────────────────────────┘ │  
│ │ Controller │ │  
│ │ │ │  
│ └─────────────────┘ │  
│ │  
└──────────────────────────────────────────────────────────┘

1. Source Controller:
2. Manages Git/Helm repositories as sources of truth
3. Fetches and validates the GitOps repository content
4. Detects changes in the source and makes them available to other controllers
5. Handles synchronization with configured interval (e.g., every 1 minute)
6. Kustomize Controller:
7. Applies Kubernetes resources defined through Kustomize
8. Manages reconciliation of the actual state with the desired state
9. Handles dependencies between resources
10. Reports reconciliation failures and successes
11. Helm Controller:
12. Manages Helm releases based on HelmRelease resources
13. Handles Helm chart installation, upgrades, and rollbacks
14. Integrates with Helm repositories for chart sources
15. Validates charts before installation
16. Notification Controller:
17. Sends events to external systems (Slack, webhook endpoints)
18. Provides delivery guarantees for notifications
19. Handles event filtering and transformation
20. Manages notification delivery status

Handles synchronization with configured interval (e.g., every 1 minute)

Kustomize Controller:

Reports reconciliation failures and successes

Helm Controller:

Validates charts before installation

Notification Controller:

##### Flux Resource Structure

The platform's Flux configuration follows a structured organization:

flux/  
├── clusters/  
│ ├── staging/  
│ │ ├── flux-system/ # Flux controllers configuration  
│ │ ├── sources/ # Repository sources  
│ │ ├── releases/ # Application releases  
│ │ └── kustomization.yaml # Main kustomization  
│ └── production/  
│ ├── flux-system/  
│ ├── sources/  
│ ├── releases/  
│ └── kustomization.yaml

flux/  
├── clusters/  
│ ├── staging/  
│ │ ├── flux-system/ # Flux controllers configuration  
│ │ ├── sources/ # Repository sources  
│ │ ├── releases/ # Application releases  
│ │ └── kustomization.yaml # Main kustomization  
│ └── production/  
│ ├── flux-system/  
│ ├── sources/  
│ ├── releases/  
│ └── kustomization.yaml

Each cluster (staging, production) has its own dedicated configuration:

1. flux-system: Contains the Flux controllers configuration  
    yaml  
    # Example: flux-system/gotk-sync.yaml  
    apiVersion: source.toolkit.fluxcd.io/v1beta2  
    kind: GitRepository  
    metadata:  
    name: flux-system  
    namespace: flux-system  
    spec:  
    interval: 1m0s  
    ref:  
    branch: main  
    url: ssh://git@github.com/kai-platform/kai-gitops
2. sources: Contains Helm repository definitions  
    yaml  
    # Example: sources/helm-repository.yaml  
    apiVersion: source.toolkit.fluxcd.io/v1beta2  
    kind: HelmRepository  
    metadata:  
    name: kai-charts  
    namespace: flux-system  
    spec:  
    interval: 5m  
    url: https://kai-platform.github.io/helm-charts/
3. releases: Contains HelmRelease definitions for each component  
    yaml  
    # Example: releases/coordinator.yaml  
    apiVersion: helm.toolkit.fluxcd.io/v2beta1  
    kind: HelmRelease  
    metadata:  
    name: coordinator  
    namespace: flux-system  
    spec:  
    interval: 5m  
    chart:  
    spec:  
    chart: coordinator  
    version: ">=1.0.0"  
    sourceRef:  
    kind: HelmRepository  
    name: kai-charts  
    values:  
    replicaCount: 3  
    image:  
    repository: "registry.example.com/coordinator"  
    tag: "v1.2.3"

flux-system: Contains the Flux controllers configuration  
 yaml  
 # Example: flux-system/gotk-sync.yaml  
 apiVersion: source.toolkit.fluxcd.io/v1beta2  
 kind: GitRepository  
 metadata:  
 name: flux-system  
 namespace: flux-system  
 spec:  
 interval: 1m0s  
 ref:  
 branch: main  
 url: ssh://git@github.com/kai-platform/kai-gitops

yaml  
 # Example: flux-system/gotk-sync.yaml  
 apiVersion: source.toolkit.fluxcd.io/v1beta2  
 kind: GitRepository  
 metadata:  
 name: flux-system  
 namespace: flux-system  
 spec:  
 interval: 1m0s  
 ref:  
 branch: main  
 url: ssh://git@github.com/kai-platform/kai-gitops

sources: Contains Helm repository definitions  
 yaml  
 # Example: sources/helm-repository.yaml  
 apiVersion: source.toolkit.fluxcd.io/v1beta2  
 kind: HelmRepository  
 metadata:  
 name: kai-charts  
 namespace: flux-system  
 spec:  
 interval: 5m  
 url: https://kai-platform.github.io/helm-charts/

yaml  
 # Example: sources/helm-repository.yaml  
 apiVersion: source.toolkit.fluxcd.io/v1beta2  
 kind: HelmRepository  
 metadata:  
 name: kai-charts  
 namespace: flux-system  
 spec:  
 interval: 5m  
 url: https://kai-platform.github.io/helm-charts/

releases: Contains HelmRelease definitions for each component  
 yaml  
 # Example: releases/coordinator.yaml  
 apiVersion: helm.toolkit.fluxcd.io/v2beta1  
 kind: HelmRelease  
 metadata:  
 name: coordinator  
 namespace: flux-system  
 spec:  
 interval: 5m  
 chart:  
 spec:  
 chart: coordinator  
 version: ">=1.0.0"  
 sourceRef:  
 kind: HelmRepository  
 name: kai-charts  
 values:  
 replicaCount: 3  
 image:  
 repository: "registry.example.com/coordinator"  
 tag: "v1.2.3"

yaml  
 # Example: releases/coordinator.yaml  
 apiVersion: helm.toolkit.fluxcd.io/v2beta1  
 kind: HelmRelease  
 metadata:  
 name: coordinator  
 namespace: flux-system  
 spec:  
 interval: 5m  
 chart:  
 spec:  
 chart: coordinator  
 version: ">=1.0.0"  
 sourceRef:  
 kind: HelmRepository  
 name: kai-charts  
 values:  
 replicaCount: 3  
 image:  
 repository: "registry.example.com/coordinator"  
 tag: "v1.2.3"

##### Helm and Flux Integration

Our Flux implementation seamlessly integrates with our Helm-based deployment architecture:

1. HelmRelease Resources: Define releases of our Helm charts with specific values
2. Value Overrides: Environment-specific values are defined in the HelmRelease resources
3. Reconciliation: Flux continuously ensures the Helm releases match their definitions
4. Releases Management: Flux handles Helm release creation, upgrades, and rollbacks

##### CI/CD to GitOps Flow

The CI/CD pipeline interacts with the GitOps repository to trigger deployments:

┌────────────────┐ ┌────────────────┐ ┌────────────────┐  
│ │ │ │ │ │  
│ CI Build & │────▶│ Update GitOps │────▶│ Flux │  
│ Test │ │ Repository │ │ Controllers │  
│ │ │ │ │ │  
└────────────────┘ └────────────────┘ └────────────────┘  
 │  
 │  
 ▼  
┌────────────────┐ ┌────────────────┐ ┌────────────────┐  
│ │ │ │ │ │  
│ Notification │◀────│ Reconciliation│◀────│ Apply │  
│ │ │ Status │ │ Changes │  
│ │ │ │ │ │  
└────────────────┘ └────────────────┘ └────────────────┘

┌────────────────┐ ┌────────────────┐ ┌────────────────┐  
│ │ │ │ │ │  
│ CI Build & │────▶│ Update GitOps │────▶│ Flux │  
│ Test │ │ Repository │ │ Controllers │  
│ │ │ │ │ │  
└────────────────┘ └────────────────┘ └────────────────┘  
 │  
 │  
 ▼  
┌────────────────┐ ┌────────────────┐ ┌────────────────┐  
│ │ │ │ │ │  
│ Notification │◀────│ Reconciliation│◀────│ Apply │  
│ │ │ Status │ │ Changes │  
│ │ │ │ │ │  
└────────────────┘ └────────────────┘ └────────────────┘

1. The CI pipeline builds and tests new versions of components
2. Upon successful build, it updates HelmRelease resources in the GitOps repository with new image tags
3. Flux detects changes in the GitOps repository
4. Flux controllers apply changes to the Kubernetes cluster
5. Reconciliation status is reported back
6. Notifications are sent about the deployment results

##### Benefits for Kubernetes Architecture

Adopting Flux GitOps provides several architectural benefits:

1. Declarative Infrastructure: All Kubernetes resources are defined declaratively in Git
2. Kubernetes-Native: Flux controllers run as Kubernetes controllers and use the Kubernetes API
3. Self-Healing: Continuous reconciliation ensures the cluster state matches Git, automatically correcting drift
4. Multi-Cluster Management: The same GitOps repository can manage multiple clusters with environment-specific configurations
5. Progressive Delivery: Support for canary deployments and A/B testing through Flux extensions
6. Security Improvements:
7. No CI/CD pipeline needs direct access to Kubernetes
8. Pull-based model (Flux pulls from Git) rather than push-based
9. Reduced attack surface and credential management
10. Compliance and Auditability:
11. All changes go through Git with commit history
12. Automated reconciliation reports
13. Clear source of truth for cluster state

##### Reconciliation and Self-Healing

Flux continuously reconciles the desired state (from Git) with the actual state in the cluster:

1. Detection: Flux detects when the actual state drifts from the desired state
2. Analysis: It analyzes the difference and determines required changes
3. Remediation: It automatically applies the necessary changes to align with the desired state
4. Reporting: It reports the reconciliation results through events and status conditions

This self-healing capability ensures that the cluster always reflects the desired configuration, even if manual changes are made or if resources are accidentally deleted.

### Kubernetes Deployment Dashboard

#### Overview

The Kubernetes Deployment Dashboard is an admin panel feature that provides real-time visibility into the Kubernetes cluster, deployments, pods, and related infrastructure. It enables administrators to monitor the health of the system, troubleshoot issues, and manage deployments efficiently.

#### Features

##### Cluster Overview

* Cluster Statistics: Real-time metrics showing total nodes, pods, deployments, and services
* Health Status: Visual indicators for cluster health (healthy, degraded, unhealthy)
* Resource Utilization: CPU, memory, and storage usage across the cluster

##### Pod Management

* Pod List: Comprehensive list of all pods with filtering by namespace
* Pod Details: Detailed information about each pod including:
* Status and phase
* Container details (image, ready status, restart count)
* Conditions and events
* Resource usage
* Age and lifetime
* Pod Logs: Real-time access to container logs with container selection
* Pod Actions: Ability to restart or terminate problematic pods

##### CI/CD Pipeline Monitoring

* Pipeline Status: Overview of recent CI/CD pipeline runs
* Stage Details: Breakdown of pipeline stages with status and duration
* Error Analysis: Detailed error information for failed pipelines
* Troubleshooting: Intelligent suggestions for resolving pipeline issues

##### Flux GitOps Deployments

* Deployment List: Overview of all Flux-managed deployments
* Reconciliation Status: Current state of GitOps reconciliation
* Error Detection: Identification of failed deployments with detailed error information
* Troubleshooting: Context-aware suggestions for resolving deployment issues

##### Kubernetes Events

* Event Monitoring: Comprehensive list of cluster events with filtering
* Event Details: Information about event source, reason, and impact
* Event Categorization: Visual indicators for different event types (normal, warning, error)

#### Architecture

The Kubernetes Deployment Dashboard consists of the following components:

##### Frontend Components

* Dashboard Page: Main entry point at /deployment in the admin panel
* Overview Cards: Display of key metrics and health indicators
* Pod Management: Components for listing, viewing, and managing pods
* Pipeline Monitoring: Components for tracking CI/CD pipelines
* Flux Deployments: Components for monitoring GitOps deployments
* Event Viewer: Components for viewing and filtering Kubernetes events

/deployment

##### Backend Services

* Kubernetes Service: Core service for interacting with the Kubernetes API
* API Routes: Secure endpoints for exposing Kubernetes data to the frontend
* Authentication: Integration with the platform's authentication system
* Logging: Comprehensive logging of all Kubernetes-related operations

#### Usage

##### Accessing the Dashboard

1. Log in to the admin panel with administrator credentials
2. Navigate to the "Deployment" section in the sidebar

##### Monitoring Cluster Health

1. View the Cluster Overview section for high-level health metrics
2. Check the health status indicator for the overall cluster state
3. Review any warning or error indicators

##### Managing Pods

1. Navigate to the Pods section
2. Use the namespace filter to focus on specific namespaces
3. Click on a pod to view detailed information
4. Select the Logs tab to view container logs
5. Use the Events tab to see pod-specific events
6. For problematic pods, use the available actions to restart or terminate

##### Monitoring CI/CD Pipelines

1. Navigate to the CI/CD Pipelines section
2. Review the status of recent pipeline runs
3. Click on a pipeline to view detailed stage information
4. For failed pipelines, review the error information and troubleshooting suggestions

##### Tracking Flux Deployments

1. Navigate to the Flux Deployments section
2. Review the status of GitOps deployments
3. Click on a deployment to view detailed information
4. For failed deployments, review the error information and troubleshooting suggestions

##### Viewing Kubernetes Events

1. Navigate to the Cluster Events section
2. Use the filters to focus on specific event types or namespaces
3. Review event details to understand system behavior

### Security

#### RBAC Configuration

The platform uses role-based access control to secure components:

apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
 name: coordinator-workflow-manager  
 namespace: kai-ml  
rules:  
 - apiGroups: ["argoproj.io"]  
 resources: ["workflows", "workflows/finalizers", "workflowtemplates"]  
 verbs: ["create", "delete", "get", "list", "patch", "update", "watch"]

apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
 name: coordinator-workflow-manager  
 namespace: kai-ml  
rules:  
 - apiGroups: ["argoproj.io"]  
 resources: ["workflows", "workflows/finalizers", "workflowtemplates"]  
 verbs: ["create", "delete", "get", "list", "patch", "update", "watch"]

#### Pod Security

1. Non-root Users: Containers run as non-root when possible
2. Read-only Root Filesystem: Where applicable
3. Pod Security Standards: Enforced at namespace level

#### Dashboard Security

The Kubernetes Deployment Dashboard implements several security measures:

* Role-Based Access Control: Only administrators can access the dashboard
* Network Restrictions: API endpoints are only accessible from internal networks
* Audit Logging: All actions are logged for security and compliance purposes
* Limited Permissions: The dashboard uses a service account with limited permissions

### Performance Tuning

#### GPU Acceleration

1. NVIDIA Device Plugin: For GPU access in containers
2. GPU Sharing: For efficient resource usage
3. Multi-GPU Workflows: For complex ML training

#### Memory Management

1. Huge Pages: For memory-intensive operations
2. Memory Limits: Preventing OOM situations
3. Pod Quality of Service: Based on resource requests and limits

#### Enhanced Observability

* Requirement: Custom metrics, distributed tracing, ML-specific logging
* Implementation:
* Prometheus integration for metrics
* Jaeger for distributed tracing
* MonitoringService for ML-specific metrics
* Grafana dashboards for visualization

### Scaling Strategies

The KAI Platform implements a sophisticated multi-layer scaling architecture to efficiently manage resources:

#### Horizontal Pod Autoscaling (HPA)

The platform uses HPA to automatically adjust replica counts for stateless components based on observed metrics, including both standard resource metrics and custom application metrics:

apiVersion: autoscaling/v2  
kind: HorizontalPodAutoscaler  
metadata:  
 name: coordinator-service-hpa  
 namespace: kai-ml  
 labels:  
 app: coordinator-service  
 component: orchestration  
spec:  
 scaleTargetRef:  
 apiVersion: apps/v1  
 kind: Deployment  
 name: coordinator-service  
 minReplicas: 2  
 maxReplicas: 10  
 metrics:  
 # CPU-based scaling  
 - type: Resource  
 resource:  
 name: cpu  
 target:  
 type: Utilization  
 averageUtilization: 70  
 # Memory-based scaling  
 - type: Resource  
 resource:  
 name: memory  
 target:  
 type: Utilization  
 averageUtilization: 80  
 # Queue depth-based scaling  
 - type: Pods  
 pods:  
 metric:  
 name: coordinator\_queue\_depth  
 target:  
 type: AverageValue  
 averageValue: 10  
 # Processing time-based scaling  
 - type: Pods  
 pods:  
 metric:  
 name: ml\_processing\_time\_seconds  
 target:  
 type: AverageValue  
 averageValue: 5  
 behavior:  
 scaleUp:  
 stabilizationWindowSeconds: 60  
 policies:  
 - type: Percent  
 value: 100  
 periodSeconds: 60  
 - type: Pods  
 value: 4  
 periodSeconds: 60  
 selectPolicy: Max  
 scaleDown:  
 stabilizationWindowSeconds: 300  
 policies:  
 - type: Percent  
 value: 10  
 periodSeconds: 60  
 - type: Pods  
 value: 2  
 periodSeconds: 60  
 selectPolicy: Min

apiVersion: autoscaling/v2  
kind: HorizontalPodAutoscaler  
metadata:  
 name: coordinator-service-hpa  
 namespace: kai-ml  
 labels:  
 app: coordinator-service  
 component: orchestration  
spec:  
 scaleTargetRef:  
 apiVersion: apps/v1  
 kind: Deployment  
 name: coordinator-service  
 minReplicas: 2  
 maxReplicas: 10  
 metrics:  
 # CPU-based scaling  
 - type: Resource  
 resource:  
 name: cpu  
 target:  
 type: Utilization  
 averageUtilization: 70  
 # Memory-based scaling  
 - type: Resource  
 resource:  
 name: memory  
 target:  
 type: Utilization  
 averageUtilization: 80  
 # Queue depth-based scaling  
 - type: Pods  
 pods:  
 metric:  
 name: coordinator\_queue\_depth  
 target:  
 type: AverageValue  
 averageValue: 10  
 # Processing time-based scaling  
 - type: Pods  
 pods:  
 metric:  
 name: ml\_processing\_time\_seconds  
 target:  
 type: AverageValue  
 averageValue: 5  
 behavior:  
 scaleUp:  
 stabilizationWindowSeconds: 60  
 policies:  
 - type: Percent  
 value: 100  
 periodSeconds: 60  
 - type: Pods  
 value: 4  
 periodSeconds: 60  
 selectPolicy: Max  
 scaleDown:  
 stabilizationWindowSeconds: 300  
 policies:  
 - type: Percent  
 value: 10  
 periodSeconds: 60  
 - type: Pods  
 value: 2  
 periodSeconds: 60  
 selectPolicy: Min

##### Custom Metrics for Intelligent Scaling

Our platform implements advanced custom metrics for more intelligent scaling decisions:

1. Queue-Based Metrics: Scale based on actual workload in the queue
2. coordinator\_queue\_depth: Number of pending tasks in the queue
3. coordinator\_queue\_processing\_rate: Rate at which tasks are being processed
4. Processing Time Metrics: Scale based on actual processing performance
5. ml\_processing\_time\_seconds: Average time to process a task
6. ml\_processing\_backlog\_seconds: Estimated time to process all queued tasks
7. Database Connection Metrics: Scale based on database connection pool utilization
8. db\_connection\_utilization: Percentage of database connections in use
9. db\_connection\_wait\_time: Time spent waiting for database connections

coordinator\_queue\_depth

coordinator\_queue\_processing\_rate: Rate at which tasks are being processed

coordinator\_queue\_processing\_rate

Processing Time Metrics: Scale based on actual processing performance

ml\_processing\_time\_seconds

ml\_processing\_backlog\_seconds: Estimated time to process all queued tasks

ml\_processing\_backlog\_seconds

Database Connection Metrics: Scale based on database connection pool utilization

db\_connection\_utilization

db\_connection\_wait\_time

##### Platform Communication with HPA:

Our system interacts with the HPA controller through a sophisticated metrics pipeline:

* Metrics Exposition: All components expose metrics via Prometheus annotations:  
   yaml  
   prometheus.io/scrape: "true"  
   prometheus.io/port: "8081"  
   prometheus.io/path: "/metrics"
* Collection Flow:
* metrics-server collects CPU/memory metrics from kubelet on each node
* Prometheus scrapes detailed custom metrics from component endpoints
* Prometheus Adapter converts Prometheus metrics to the custom metrics API format
* HPA controller queries these APIs every 15 seconds to make scaling decisions
* Prometheus Adapter Configuration: Custom metrics are exposed to Kubernetes via the Prometheus Adapter:  
   ```yaml  
   rules:
* seriesQuery: 'kai\_coordinator\_queue\_depth{kubernetes\_namespace!="",kubernetes\_pod\_name!=""}'  
   resources:  
   overrides:  
   kubernetes\_namespace: {resource: "namespace"}  
   kubernetes\_pod\_name: {resource: "pod"}  
   name:  
   matches: "kai\_coordinator\_queue\_depth"  
   as: "coordinator\_queue\_depth"  
   metricsQuery: 'sum(<<.Series>>{<<.LabelMatchers>>}) by (<<.GroupBy>>)'  
   ```
* Coordinator Service Role: The Coordinator actively participates in the scaling architecture by:
* Exposing workload metrics (queue depths, processing times) via its /metrics endpoint
* Tracking processing load across different quality tiers
* Adjusting its internal task concurrency limits based on observed cluster capacity
* Implementing back-pressure mechanisms when resources are constrained

yaml  
 prometheus.io/scrape: "true"  
 prometheus.io/port: "8081"  
 prometheus.io/path: "/metrics"

metrics-server

HPA controller queries these APIs every 15 seconds to make scaling decisions

Prometheus Adapter Configuration: Custom metrics are exposed to Kubernetes via the Prometheus Adapter:  
 ```yaml  
 rules:

seriesQuery: 'kai\_coordinator\_queue\_depth{kubernetes\_namespace!="",kubernetes\_pod\_name!=""}'  
 resources:  
 overrides:  
 kubernetes\_namespace: {resource: "namespace"}  
 kubernetes\_pod\_name: {resource: "pod"}  
 name:  
 matches: "kai\_coordinator\_queue\_depth"  
 as: "coordinator\_queue\_depth"  
 metricsQuery: 'sum(<<.Series>>{<<.LabelMatchers>>}) by (<<.GroupBy>>)'  
 ```

Coordinator Service Role: The Coordinator actively participates in the scaling architecture by:

/metrics

#### Workflow-level Concurrency Management

The Coordinator Service implements sophisticated task queue management with:

* Priority-based queueing with weighted fair scheduling
* Dynamic concurrency limits based on resource availability
* Task classification (interactive, batch, maintenance) with appropriate scheduling policies
* Resource reservation for high-priority workflows

#### Cluster Autoscaling

Node pools automatically scale based on pending pods, which happens when:

* HPAs increase replica counts, creating new pods
* Argo workflows spawn pods that can't be scheduled on existing nodes
* Quality tier requirements demand specialized resources

Each node pool (CPU-optimized, GPU-optimized, etc.) scales independently based on the specific workload needs.

#### Quality Tier Scaling

The ResourceManager component dynamically adjusts resource requests for workflows based on:

* Subscription tier limitations (enforcing fair resource allocation)
* Current cluster utilization (applying backpressure when needed)
* Quality level requirements (allocating appropriate GPU resources)

This ensures optimal resource distribution during high-load periods while maintaining quality of service guarantees.

Results and Benefits:

This multi-layered scaling approach provides:

* Cost Efficiency: Scaling components down during low-traffic periods
* Responsive Scaling: Proactively adding replicas before performance degrades
* Reliability: Automatic recovery from failures through replica recreation
* Resource Optimization: Efficient allocation based on actual usage patterns
* Quality of Service: Maintaining performance guarantees for different subscription tiers

The system's scaling behavior can be monitored through dedicated Grafana dashboards that display:  
- Current/target replica counts  
- CPU/memory utilization across replicas  
- Scaling events timeline  
- Queue depths by priority level

### High Availability and Disaster Recovery

#### High Availability

1. Pod Disruption Budgets: Ensure minimum availability during updates  
    yaml  
    apiVersion: policy/v1  
    kind: PodDisruptionBudget  
    metadata:  
    name: coordinator-pdb  
    spec:  
    minAvailable: 2  
    selector:  
    matchLabels:  
    app: coordinator-service
2. Anti-Affinity Rules: Distribute pods across nodes
3. Multi-Zone Deployment: Spread workloads across availability zones

Pod Disruption Budgets: Ensure minimum availability during updates  
 yaml  
 apiVersion: policy/v1  
 kind: PodDisruptionBudget  
 metadata:  
 name: coordinator-pdb  
 spec:  
 minAvailable: 2  
 selector:  
 matchLabels:  
 app: coordinator-service

yaml  
 apiVersion: policy/v1  
 kind: PodDisruptionBudget  
 metadata:  
 name: coordinator-pdb  
 spec:  
 minAvailable: 2  
 selector:  
 matchLabels:  
 app: coordinator-service

Anti-Affinity Rules: Distribute pods across nodes

#### Disaster Recovery

1. Regular Backups: PVC snapshots, database backups
2. Stateless Design: Most components can be recreated from configuration
3. GitOps Approach: Infrastructure-as-Code for quick recovery

### Monitoring and Observability

#### Prometheus and Grafana

* Prometheus: Collects metrics from all components
* Grafana: Provides dashboards for visualizing metrics
* Custom Dashboards: ML-specific dashboards for processing metrics

Prometheus is configured to auto-discover and scrape metrics from pods with the appropriate annotations:

prometheus.io/scrape: "true"  
prometheus.io/port: "8080"  
prometheus.io/path: "/metrics"

prometheus.io/scrape: "true"  
prometheus.io/port: "8080"  
prometheus.io/path: "/metrics"

#### Jaeger Distributed Tracing

* Traces requests across components
* Measures processing time for each stage
* Helps identify bottlenecks in ML pipelines

### Troubleshooting

#### Common Issues and Solutions

##### Pod Stuck in Pending State

* Possible Causes: Insufficient resources, volume mount issues, node selector constraints
* Resolution: Check node resources, verify PVC status, review pod specifications

##### Failed CI/CD Pipeline

* Possible Causes: Test failures, build errors, deployment issues
* Resolution: Review pipeline logs, check test results, verify deployment configurations

##### Failed Flux Deployment

* Possible Causes: Chart not found, invalid values, dependency issues
* Resolution: Verify chart existence, check values.yaml, ensure dependencies are available

##### Slow Query Performance

* Possible Causes: Resource constraints, inefficient queries, high volume
* Resolution: Scale up pods, optimize queries, review caching strategies

##### Pod Crashes

* Possible Causes: OOM errors, application bugs, configuration issues
* Resolution: Check logs, increase memory limits, debug application code

#### Debugging Commands

# Get logs for coordinator service  
kubectl logs -n kai-ml deployment/coordinator-service  
  
# Describe a workflow  
kubectl -n kai-ml describe workflow my-workflow-name  
  
# Check resource usage  
kubectl top pods -n kai-ml  
kubectl top nodes

# Get logs for coordinator service  
kubectl logs -n kai-ml deployment/coordinator-service  
  
# Describe a workflow  
kubectl -n kai-ml describe workflow my-workflow-name  
  
# Check resource usage  
kubectl top pods -n kai-ml  
kubectl top nodes

#### Dashboard Troubleshooting

##### Pod Stuck in Pending State

* Possible Causes: Insufficient resources, volume mount issues, node selector constraints
* Resolution: Check node resources, verify PVC status, review pod specifications

##### Failed CI/CD Pipeline

* Possible Causes: Test failures, build errors, deployment issues
* Resolution: Review pipeline logs, check test results, verify deployment configurations

##### Failed Flux Deployment

* Possible Causes: Chart not found, invalid values, dependency issues
* Resolution: Verify chart existence, check values.yaml, ensure dependencies are available

### API Reference

The Kubernetes Deployment Dashboard uses the following API endpoints:

* GET /api/admin/kubernetes/stats: Get cluster statistics
* GET /api/admin/kubernetes/pods: Get pod details
* GET /api/admin/kubernetes/nodes: Get node details
* GET /api/admin/kubernetes/deployments: Get deployment details
* GET /api/admin/kubernetes/events: Get Kubernetes events
* GET /api/admin/kubernetes/logs/:podName: Get pod logs

GET /api/admin/kubernetes/stats

GET /api/admin/kubernetes/pods

GET /api/admin/kubernetes/nodes

GET /api/admin/kubernetes/deployments

GET /api/admin/kubernetes/events

GET /api/admin/kubernetes/logs/:podName

All endpoints require administrator authentication and are only accessible from internal networks.

### Conclusion

The Kubernetes architecture for the KAI ML Platform provides a robust, scalable foundation for machine learning workloads. By leveraging specialized node pools, priority classes, and custom resource scheduling, the platform efficiently manages compute-intensive ML tasks while maintaining high availability and performance.

The combination of Helm-based deployment and Flux GitOps provides a powerful, declarative approach to managing the Kubernetes infrastructure, while the Kubernetes Deployment Dashboard offers comprehensive visibility and management capabilities for administrators.

This implementation fulfills all the requirements specified in the task, providing a scalable, resilient, and cost-effective Kubernetes architecture for ML processing pipelines. The system intelligently adapts to workload characteristics, resource availability, and user requirements, ensuring optimal performance while maintaining efficiency.

For specific deployment instructions, refer to the Digital Ocean Kubernetes Setup and Deployment Guide.

# Main Readme

Source: readme/main-readme.md

---

## Kai - Material Recognition & Knowledge Base System

Kai is a comprehensive full-stack application for material recognition and catalog management, with particular focus on tile materials. The system enables identification, cataloging, and searching for materials using machine learning.

### Documentation

All detailed documentation is available in the readme folder:

readme

* Project Structure - Organization and component interactions
* Material Recognition - ML-powered material identification
* Knowledge Base - Material storage and retrieval system
* Datasets and AI Models - Integration of premade datasets with AI models
* PDF Processing - Catalog extraction capabilities
* Queue System - Message broker and async processing
* Monitoring System - System health monitoring and operational visibility
* API Reference - Comprehensive API endpoints including health checks and rate limiting
* Deployment & Development - Production deployment and development setup
* CrewAI Integration - Intelligent agent capabilities powered by crewAI
* CrewAI Implementation - Implementation details for crewAI agents
* MoodBoard Feature - Material collection and organization feature

### Quick Start

# Clone repository  
git clone https://github.com/your-org/kai.git  
cd kai  
  
# Install dependencies  
yarn install  
  
# Set up environment  
cp .env.example .env  
  
# Start development environment  
yarn dev

# Clone repository  
git clone https://github.com/your-org/kai.git  
cd kai  
  
# Install dependencies  
yarn install  
  
# Set up environment  
cp .env.example .env  
  
# Start development environment  
yarn dev

For detailed setup instructions, deployment guides, and development workflows, see the Deployment & Development documentation.

### Services, Modules, and Features of the Kai Platform

#### Front-End Features

##### 1. User Management

* Profile Management: User profiles with customizable fields (username, email, name, avatar)
* Preference System: Theme settings (light/dark/system), notification preferences, email frequency settings, view preferences (grid/list)
* Authentication UI: Login, registration, password reset flows with social authentication options
* Session Management: Token handling, "Remember Me" functionality

##### 2. Material Recognition

* Image Upload Interface: Upload images to identify materials
* Results Display: Confidence scores visualization, similarity metrics, recognized material details
* Similar Materials Suggestions: UI for browsing similar materials based on recognition
* Feedback Mechanisms: Users can provide feedback on recognition accuracy

##### 3. Knowledge Base UI

* Material Browser: Interface for exploring the material database
* Search Interface: Combined text and vector search UI with filters
* Material Details View: Comprehensive information display for materials
* Material Relationships: Visual representation of related materials

##### 4. MoodBoard Feature

* Collection Management: Create, edit, and delete material collections
* Visibility Controls: Public/private settings for boards
* Organizational Tools: Add, remove, arrange materials in collections
* Annotation System: Notes and labels for saved materials
* Sharing Interface: Controls for sharing collections with others

##### 5. Catalog Management

* PDF Uploader: Interface for uploading material catalogs
* Processing Status: Visual indicators of extraction progress
* Results Review: Interface for reviewing and correcting extracted content

##### 6. 3D Designer

* Room Layout Controls: Interface for specifying room measurements and features
* Furniture Placement UI: Tools for adding and arranging furniture
* 3D Visualization: Interactive 3D view of designed spaces
* Material Application: UI for applying materials to surfaces
* Physics Validation: Feedback on realistic object placement

##### 7. WebXR Integration

* VR/AR Mode Toggles: Controls for switching between viewing modes
* Interaction Controls: UI elements for manipulating objects in VR/AR
* Scene Manipulation: Tools for navigating and adjusting the 3D environment

#### Back-End Services

##### 1. Authentication System

* Identity Management: User creation, authentication, and authorization
* Social Authentication: Integration with Google, Facebook, Twitter providers
* Token Management: JWT handling, refresh tokens, session persistence
* Password Security: Secure storage, reset workflows, strength validation
* Role-Based Access Control: Permission management for different user types

##### 2. Recognition Pipeline

* Vector-Based Recognition: Uses feature vectors for material matching
* Confidence Scoring: Sophisticated scoring based on vector similarity
* Feature Extraction: Converts images to feature vectors for matching
* Feedback Processing: Collects and incorporates user feedback on results
* Accuracy Analytics: Tracks and reports on recognition performance

##### 3. Knowledge Base Backend

* Material Database: Comprehensive storage of material information
* Vector Search Engine: Semantic similarity search using feature vectors
* Text Search System: Traditional keyword-based search capabilities
* Hybrid Search Algorithm: Combines text and vector approaches
* Relationship Management: Maintains connections between related materials

##### 4. Catalog Processing

* PDF Extraction Engine: Parses catalog PDFs for content
* OCR Processing: Extracts text from catalog images
* Batch Management: Handles multiple catalogs simultaneously
* Output Validation: Ensures quality of extracted content

##### 5. 3D Processing

* Layout Generation Engine: Creates room layouts from specifications
* Furniture Placement Algorithms: Automated furniture arrangement
* Physics Validation Engine: Ensures realistic object placement
* Material Mapping: Applies materials to 3D surfaces

##### 6. Agent System

* Recognition Assistant: Helps with material identification
* 3D Designer Agent: Assists with design tasks
* Material Expert: Provides detailed material knowledge
* Session Management: Maintains context across interactions
* WebSocket Communication: Real-time agent communication

##### 7. Queue System

* Job Management: Handles background processing tasks
* Priority Handling: Manages task priorities
* Error Recovery: Handles failed jobs with retry mechanisms
* Status Reporting: Provides job status updates

#### Architectural Components

##### 1. Model Context Protocol (MCP)

* Centralized Model Management: Standardized interface for ML models
* Inference Optimization: Efficient model execution
* Context Management: Maintains state across operations
* Agent Communication: Facilitates agent interactions
* Feature Extraction: Standardized extraction of material features

##### 2. Database Architecture

* Supabase Integration: Primary database with vector capabilities
* Connection Pooling: Optimized database connections
* Query Caching: Performance improvements for repeated queries
* Vector Storage: Specialized storage for similarity searches

##### 3. API Layer

* RESTful Endpoints: Comprehensive API for all features
* Rate Limiting: Prevents API abuse with tiered limits
* Authentication Middleware: Secure API access
* Error Handling: Standardized error responses

##### 4. WebSocket System

* Real-Time Updates: Instant notifications of system events
* Queue Monitoring: Live updates on job progress
* Training Progress: Real-time training metrics
* Agent Communication: Interactive agent dialogues

##### 5. Storage Management

* File Storage: Handles uploads and storage
* S3 Integration: Scalable storage solution
* Image Processing: Optimizes and transforms images
* Access Control: Manages file permissions

##### 6. Monitoring & Analytics

* Health Checks: System health monitoring
* Performance Metrics: Tracks system performance
* Error Logging: Captures and analyzes errors
* Resource Monitoring: Tracks system resource usage

#### Component Interconnections

1. Front-End to Back-End:
2. React/Gatsby front-end communicates with Express backend via REST API
3. Real-time updates flow through WebSocket connections
4. Authentication state managed through Supabase integration
5. Recognition Flow:
6. User uploads image → MCP extracts features → Vector search finds matches → Results displayed to user
7. Feedback flows back into system to improve recognition accuracy
8. 3D Visualization Chain:
9. User selects materials → Applied to 3D models → Rendered through ThreeJS → Optionally viewed in WebXR
10. Agent Communication Path:
11. User queries → WebSocket to backend → MCP for model inference → Agent formulates response → WebSocket to front-end
12. Knowledge Base Integration:
13. Materials stored in Supabase → Vector embeddings enable similarity search → Connected to recognition system → Powers agent knowledge
14. Queue System Flow:
15. Intensive tasks sent to queue → Processed asynchronously → Status updates via WebSockets → Results stored in database

Authentication state managed through Supabase integration

Recognition Flow:

Feedback flows back into system to improve recognition accuracy

3D Visualization Chain:

User selects materials → Applied to 3D models → Rendered through ThreeJS → Optionally viewed in WebXR

Agent Communication Path:

User queries → WebSocket to backend → MCP for model inference → Agent formulates response → WebSocket to front-end

Knowledge Base Integration:

Materials stored in Supabase → Vector embeddings enable similarity search → Connected to recognition system → Powers agent knowledge

Queue System Flow:

### System Health Monitoring

Kai includes comprehensive health monitoring capabilities:

* Health Endpoints: Basic /health and detailed /health/detailed endpoints for monitoring
* Environment Validation: Automatic validation of required environment variables
* Rate Limiting: Specialized rate limiting for different API endpoints
* Monitoring Dashboard: Admin panel for system monitoring with logs, errors, and metrics

/health

/health/detailed

See the Monitoring System documentation for details.

### License

MIT

# Material Comparison Engine

Source: readme/material-comparison-engine.md

---

## Material Comparison Engine

The Material Comparison Engine enables sophisticated comparison between materials based on their properties, helping users find alternatives or compare options.

### Features

#### Similarity Calculation

* Property-Based Comparison: Compares materials based on their property values
* Weighted Algorithms: Prioritizes important properties in the comparison
* Normalized Values: Normalizes property values to ensure fair comparison
* Overall Similarity Score: Calculates a weighted average of all property similarities

#### Comparison Views

* Side-by-Side Comparison: Displays properties of two materials side by side
* Visual Indicators: Uses color coding to highlight similarities and differences
* Property Importance: Marks properties as high, medium, or low importance
* Sortable Results: Allows sorting by property name, similarity, or importance
* Filtering: Filters properties by importance level

#### Batch Comparison

* Multiple Material Selection: Compares multiple materials at once
* Comparison Matrix: Shows similarity scores between all selected materials
* Detailed View: Provides detailed comparison for any pair of materials
* Export Options: Exports comparison results for reporting

#### Similar Materials

* Find Alternatives: Finds materials similar to a selected material
* Similarity Threshold: Filters results by minimum similarity score
* Material Type Filter: Filters results by material type
* Detailed Comparison: Shows detailed comparison between the source material and any similar material

#### Comparison Presets

* Custom Weights: Creates custom weighting schemes for different comparison scenarios
* Default Presets: Provides default presets for common comparison scenarios
* Material-Specific Presets: Creates presets specific to certain material types
* Property Inclusion/Exclusion: Includes or excludes specific properties from comparison

### Technical Implementation

#### Backend

* Comparison Service: Handles material comparison logic
* Similarity Algorithms: Implements algorithms for different property types (numeric, string, boolean, array)
* Normalization: Normalizes property values based on their expected ranges
* Preset Management: Manages comparison presets

#### Database

* Comparison Results: Stores comparison results for future reference
* Comparison Presets: Stores user-defined comparison presets
* Default Weights: Stores default weights for different material types

#### Frontend

* Comparison View: Displays detailed comparison between materials
* Batch Comparison: Enables comparison of multiple materials
* Similar Materials View: Displays materials similar to a selected material
* Preset Management: Manages comparison presets

### Usage

#### Comparing Two Materials

1. Navigate to a material detail page
2. Click "Compare with Another Material"
3. Select another material to compare with
4. View the detailed comparison

#### Finding Similar Materials

1. Navigate to a material detail page
2. Click "Find Similar Materials"
3. Adjust similarity threshold and material type filter as needed
4. View the list of similar materials
5. Click on any similar material to see a detailed comparison

#### Batch Comparison

1. Navigate to the Batch Comparison page
2. Select multiple materials to compare
3. Optionally select a comparison preset
4. Click "Compare Materials"
5. View the comparison matrix
6. Click on any pair to see a detailed comparison

#### Managing Comparison Presets

1. Navigate to the Batch Comparison page
2. Click "Comparison Presets"
3. Create, edit, or delete presets
4. Adjust property weights as needed
5. Save the preset for future use

### Benefits

* Find Alternatives: Helps users find alternative materials with similar properties
* Compare Options: Makes it easier to compare options for a project
* Identify Differences: Highlights subtle differences between similar materials
* Support Substitution: Supports substitution workflows when materials are unavailable
* Batch Processing: Enables efficient comparison of multiple materials at once

# Material Expert

Source: readme/material-expert.md

---

## Material Expert

This document provides detailed information about the Material Expert, a specialized crewAI agent that serves as a knowledge source for construction materials within the KAI platform.

### Overview

The Material Expert agent provides comprehensive information about construction materials, their properties, applications, compatibility, and best practices. It leverages the KAI platform's material database to offer detailed insights and recommendations, helping users make informed decisions about material selection and usage.

### Key Capabilities

The Material Expert offers multiple specialized functions:

1. Material Property Information
2. Provide detailed specifications about material properties
3. Explain technical characteristics (hardness, porosity, etc.)
4. Describe appearance characteristics and visual properties
5. Detail manufacturing processes and composition
6. Application Guidance
7. Recommend optimal materials for specific applications
8. Explain suitability factors for different environments
9. Provide installation requirements and considerations
10. Offer maintenance guidelines and best practices
11. Compatibility Analysis
12. Assess compatibility between different materials
13. Explain potential interaction issues or benefits
14. Recommend complementary materials for projects
15. Identify potential chemical or physical incompatibilities
16. Comparison Services
17. Compare multiple materials side-by-side
18. Highlight key differences and similarities
19. Assess cost-benefit tradeoffs between options
20. Provide objective assessments of relative advantages
21. Problem-Solving Support
22. Diagnose material-related issues or failures
23. Suggest solutions to common material problems
24. Provide remediation approaches for material defects
25. Offer preventative maintenance advice

Detail manufacturing processes and composition

Application Guidance

Offer maintenance guidelines and best practices

Compatibility Analysis

Identify potential chemical or physical incompatibilities

Comparison Services

Provide objective assessments of relative advantages

Problem-Solving Support

### Architecture

The Material Expert integrates with the broader KAI platform through several key components:

#### Component Structure

packages/  
├── agents/  
│ ├── src/  
│ │ ├── frontend/  
│ │ │ └── materialExpert.ts # Agent implementation  
│ │ ├── services/  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ ├── materialSearch.ts # Material search tool  
│ │ │ ├── vectorSearch.ts # Vector search tool  
│ │ │ └── index.ts # Tool exports  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── client/  
 └── src/  
 └── components/  
 └── agents/  
 ├── MaterialExpertPanel.tsx # Client-side interface  
 └── AgentDashboard.tsx # Agent integration in UI

packages/  
├── agents/  
│ ├── src/  
│ │ ├── frontend/  
│ │ │ └── materialExpert.ts # Agent implementation  
│ │ ├── services/  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ ├── materialSearch.ts # Material search tool  
│ │ │ ├── vectorSearch.ts # Vector search tool  
│ │ │ └── index.ts # Tool exports  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── client/  
 └── src/  
 └── components/  
 └── agents/  
 ├── MaterialExpertPanel.tsx # Client-side interface  
 └── AgentDashboard.tsx # Agent integration in UI

#### Architectural Layers

1. Agent Layer (materialExpert.ts)
2. Implements the agent's core capabilities
3. Defines specialized methods for material information tasks
4. Processes user queries about materials
5. Manages context for detailed material information
6. Service Layer (via ServiceFactory)
7. Provides access to material database
8. Handles API communication with error management
9. Formats requests and responses appropriately
10. Acts as a bridge to backend material services
11. Tool Layer (materialSearch, vectorSearch)
12. Implements specialized tools for the agent to use
13. Enables text-based material database queries
14. Provides vector-based similarity searches
15. Formats results for agent consumption
16. UI Layer (MaterialExpertPanel.tsx)
17. Presents the agent's capabilities in the user interface
18. Provides chat interface for material queries
19. Displays material information with rich formatting
20. Supports comparison views for multiple materials

materialExpert.ts

Manages context for detailed material information

Service Layer (via ServiceFactory)

Acts as a bridge to backend material services

Tool Layer (materialSearch, vectorSearch)

Formats results for agent consumption

UI Layer (MaterialExpertPanel.tsx)

MaterialExpertPanel.tsx

### Implementation Details

#### Agent Implementation

The Material Expert is a UserFacingAgent type that implements specialized methods for material information tasks:

export class MaterialExpert implements UserFacingAgent {  
 // Standard UserFacingAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // Material-specific methods  
 public async processUserInput(message: string): Promise<string>;  
}

export class MaterialExpert implements UserFacingAgent {  
 // Standard UserFacingAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // Material-specific methods  
 public async processUserInput(message: string): Promise<string>;  
}

#### Agent Tools

The Material Expert leverages specialized tools to perform its tasks:

// Tool for searching materials in the database  
const materialSearchTool = await createMaterialSearchTool();  
  
// Tool for performing vector-based similarity searches  
const vectorSearchTool = await createVectorSearchTool();  
  
// Create a Material Expert with tools  
const tools = [  
 materialSearchTool,  
 vectorSearchTool  
];  
  
// Additional tools can be added from the configuration  
if (config.additionalTools) {  
 tools.push(...config.additionalTools);  
}

// Tool for searching materials in the database  
const materialSearchTool = await createMaterialSearchTool();  
  
// Tool for performing vector-based similarity searches  
const vectorSearchTool = await createVectorSearchTool();  
  
// Create a Material Expert with tools  
const tools = [  
 materialSearchTool,  
 vectorSearchTool  
];  
  
// Additional tools can be added from the configuration  
if (config.additionalTools) {  
 tools.push(...config.additionalTools);  
}

#### Agent Description

The Material Expert is defined with the following characteristics:

const agent = new Agent({  
 name: 'Material Expert',  
 role: 'Construction material specialist with deep knowledge of materials and their properties',  
 goal: 'Provide accurate and detailed information about construction materials to help users make informed decisions',  
 backstory: 'With years of experience in material science and construction, I can identify materials, explain their properties, and recommend the best options for specific applications.',  
 verbose: config.verbose || false,  
 llm: modelSettings,  
 tools  
});

const agent = new Agent({  
 name: 'Material Expert',  
 role: 'Construction material specialist with deep knowledge of materials and their properties',  
 goal: 'Provide accurate and detailed information about construction materials to help users make informed decisions',  
 backstory: 'With years of experience in material science and construction, I can identify materials, explain their properties, and recommend the best options for specific applications.',  
 verbose: config.verbose || false,  
 llm: modelSettings,  
 tools  
});

#### Client-Side Integration

The Material Expert is integrated into the client interface through a specialized panel that provides:

1. Chat Interface - For asking questions about materials
2. Material Display - For viewing detailed information about materials
3. Comparison View - For comparing multiple materials side-by-side
4. Related Materials - For discovering similar or complementary materials

### Setup Instructions

#### Prerequisites

* Functioning KAI platform with material database
* CrewAI integration set up according to CrewAI installation guide
* Vector search capabilities for material similarity

#### Installation

The Material Expert is included in the standard crewAI integration package:

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

#### Configuration

Configure the agent in your application initialization:

import { createMaterialExpert } from '@kai/agents';  
  
// Create a Material Expert instance  
const materialExpert = await createMaterialExpert(  
 {  
 id: 'material-expert-1',  
 name: 'Material Expert',  
 description: 'Expert in construction materials and their properties',  
 verbose: true,  
 // Additional configuration options  
 },  
 {  
 model: 'gpt-4',  
 temperature: 0.3  
 }  
);

import { createMaterialExpert } from '@kai/agents';  
  
// Create a Material Expert instance  
const materialExpert = await createMaterialExpert(  
 {  
 id: 'material-expert-1',  
 name: 'Material Expert',  
 description: 'Expert in construction materials and their properties',  
 verbose: true,  
 // Additional configuration options  
 },  
 {  
 model: 'gpt-4',  
 temperature: 0.3  
 }  
);

### Usage Examples

#### Client-Side Integration

import React from 'react';  
import { MaterialExpertPanel } from '../components/agents/MaterialExpertPanel';  
  
const MaterialExpertPage: React.FC = () => {  
 return (  
 <div className="material-expert-page">  
 <h1>Material Expert</h1>  
 <MaterialExpertPanel />  
 </div>  
 );  
};  
  
export default MaterialExpertPage;

import React from 'react';  
import { MaterialExpertPanel } from '../components/agents/MaterialExpertPanel';  
  
const MaterialExpertPage: React.FC = () => {  
 return (  
 <div className="material-expert-page">  
 <h1>Material Expert</h1>  
 <MaterialExpertPanel />  
 </div>  
 );  
};  
  
export default MaterialExpertPage;

#### Processing User Queries

import { createMaterialExpert } from '@kai/agents';  
  
// Create the Material Expert  
const materialExpert = await createMaterialExpert(  
 { id: 'material-expert-1' },  
 { model: 'gpt-4', temperature: 0.3 }  
);  
  
// Ask questions about materials  
const query1 = 'What are the key differences between ceramic and porcelain tiles?';  
const response1 = await materialExpert.processUserInput(query1);  
console.log(response1);  
  
const query2 = 'What material would you recommend for a bathroom floor that needs to be water-resistant and durable?';  
const response2 = await materialExpert.processUserInput(query2);  
console.log(response2);  
  
const query3 = 'Are concrete countertops compatible with epoxy sealants?';  
const response3 = await materialExpert.processUserInput(query3);  
console.log(response3);

import { createMaterialExpert } from '@kai/agents';  
  
// Create the Material Expert  
const materialExpert = await createMaterialExpert(  
 { id: 'material-expert-1' },  
 { model: 'gpt-4', temperature: 0.3 }  
);  
  
// Ask questions about materials  
const query1 = 'What are the key differences between ceramic and porcelain tiles?';  
const response1 = await materialExpert.processUserInput(query1);  
console.log(response1);  
  
const query2 = 'What material would you recommend for a bathroom floor that needs to be water-resistant and durable?';  
const response2 = await materialExpert.processUserInput(query2);  
console.log(response2);  
  
const query3 = 'Are concrete countertops compatible with epoxy sealants?';  
const response3 = await materialExpert.processUserInput(query3);  
console.log(response3);

### Advanced Configuration

#### Custom Material Analysis Tools

Create custom tools to enhance the Material Expert's capabilities:

import { Tool } from 'crewai';  
  
// Create a specialized material analysis tool for environmental impact  
const createEnvironmentalImpactTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'environmental\_impact\_analysis',  
 description: 'Analyze the environmental impact of materials based on lifecycle assessment data',  
 func: async (args) => {  
 const { materialId } = JSON.parse(args);  
  
 // Implement environmental impact analysis  
 const impactData = await analyzeMaterialEnvironmentalImpact(materialId);  
  
 return JSON.stringify({  
 carbonFootprint: impactData.carbonFootprint,  
 energyConsumption: impactData.energyConsumption,  
 waterUsage: impactData.waterUsage,  
 recycleability: impactData.recycleability,  
 sustainabilityScore: impactData.overallScore  
 });  
 }  
 });  
};  
  
// Add it to the agent  
const materialExpert = await createMaterialExpert(  
 {   
 id: 'eco-material-expert-1',  
 additionalTools: [await createEnvironmentalImpactTool()]  
 },  
 { model: 'gpt-4', temperature: 0.3 }  
);

import { Tool } from 'crewai';  
  
// Create a specialized material analysis tool for environmental impact  
const createEnvironmentalImpactTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'environmental\_impact\_analysis',  
 description: 'Analyze the environmental impact of materials based on lifecycle assessment data',  
 func: async (args) => {  
 const { materialId } = JSON.parse(args);  
  
 // Implement environmental impact analysis  
 const impactData = await analyzeMaterialEnvironmentalImpact(materialId);  
  
 return JSON.stringify({  
 carbonFootprint: impactData.carbonFootprint,  
 energyConsumption: impactData.energyConsumption,  
 waterUsage: impactData.waterUsage,  
 recycleability: impactData.recycleability,  
 sustainabilityScore: impactData.overallScore  
 });  
 }  
 });  
};  
  
// Add it to the agent  
const materialExpert = await createMaterialExpert(  
 {   
 id: 'eco-material-expert-1',  
 additionalTools: [await createEnvironmentalImpactTool()]  
 },  
 { model: 'gpt-4', temperature: 0.3 }  
);

#### Integration with Material Testing Data

Connect the Material Expert to material testing databases:

import { Tool } from 'crewai';  
  
// Create a tool for accessing material testing data  
const createMaterialTestingTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'material\_testing\_data',  
 description: 'Access technical testing data for materials including stress tests, durability metrics, etc.',  
 func: async (args) => {  
 const { materialId, testType } = JSON.parse(args);  
  
 // Implement testing data retrieval  
 const testingData = await getMaterialTestResults(materialId, testType);  
  
 return JSON.stringify(testingData);  
 }  
 });  
};  
  
// Add it to the agent  
const materialExpert = await createMaterialExpert(  
 {   
 id: 'technical-material-expert-1',  
 additionalTools: [await createMaterialTestingTool()]  
 },  
 { model: 'gpt-4', temperature: 0.3 }  
);

import { Tool } from 'crewai';  
  
// Create a tool for accessing material testing data  
const createMaterialTestingTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'material\_testing\_data',  
 description: 'Access technical testing data for materials including stress tests, durability metrics, etc.',  
 func: async (args) => {  
 const { materialId, testType } = JSON.parse(args);  
  
 // Implement testing data retrieval  
 const testingData = await getMaterialTestResults(materialId, testType);  
  
 return JSON.stringify(testingData);  
 }  
 });  
};  
  
// Add it to the agent  
const materialExpert = await createMaterialExpert(  
 {   
 id: 'technical-material-expert-1',  
 additionalTools: [await createMaterialTestingTool()]  
 },  
 { model: 'gpt-4', temperature: 0.3 }  
);

### Performance Considerations

#### Knowledge Base Optimization

1. Caching Strategy
2. Cache frequently requested material information
3. Implement TTL-based cache invalidation for freshness
4. Pre-compute common material comparisons
5. Query Optimization
6. Use indexed material properties for faster lookups
7. Implement faceted search for filtered queries
8. Optimize vector searches with approximate nearest neighbors
9. Response Construction
10. Build responses with templated sections where appropriate
11. Use progressive loading for detailed material information
12. Structure responses for optimal client-side rendering

Pre-compute common material comparisons

Query Optimization

Optimize vector searches with approximate nearest neighbors

Response Construction

### Security Considerations

1. Data Access Control
2. Enforce appropriate permissions for proprietary material data
3. Implement proper user authentication for premium material information
4. Limit access to sensitive supplier or pricing information
5. Information Accuracy
6. Verify material information against reliable sources
7. Provide provenance for technical material specifications
8. Implement fact-checking mechanisms for critical properties
9. Agent Boundaries
10. Restrict the agent to material-related operations
11. Validate inputs to prevent injection attacks
12. Avoid providing professional advice that requires certification

Limit access to sensitive supplier or pricing information

Information Accuracy

Implement fact-checking mechanisms for critical properties

Agent Boundaries

### Related Documentation

* Material Recognition - Core recognition system architecture
* Knowledge Base - Material database structure and management
* CrewAI Integration - Overall agent system architecture
* CrewAI Implementation - Implementation details
* Agent Installation - Setup instructions

# Material Promotion System

Source: readme/material-promotion-system.md

---

## Material Promotion System

### Executive Summary

The Material Promotion System allows factories to purchase credits and use them to promote their materials in 3D model generation. When users create 3D models with prompts that match promoted materials, those materials will appear more frequently (1/3 times) in the generated models, while other times random materials from the knowledge base will be used.

### System Architecture

The Material Promotion System consists of the following components:

1. Database Layer: Stores material promotions, tracks usage, and manages credit allocations
2. API Layer: Provides endpoints for factories to manage their promotions
3. Integration Layer: Integrates with the 3D model generation process to select promoted materials
4. Analytics Layer: Tracks promotion performance and provides insights to factories

### Core Components

#### 1. Material Promotions

Material promotions link materials to factories and track credit allocation:

* Material ID: The material being promoted
* Factory ID: The factory promoting the material
* Credits Allocated: Number of credits allocated to the promotion
* Status: Active, inactive, completed, or pending
* Usage Metrics: Tracks impressions and actual usage in 3D models

#### 2. Credit System Integration

The system integrates with the existing credit system:

* Factories purchase credits through the standard credit purchase system
* Credits can be allocated to specific material promotions
* Credit transactions are tracked with a new 'promotion' type
* Usage analytics show how credits are being utilized

#### 3. Material Selection Algorithm

The material selection algorithm determines when to use promoted materials:

* 1/3 chance of selecting a promoted material when a matching material type is needed
* Weighted selection based on credits allocated when multiple promotions match
* Fallback to random materials from the knowledge base when no promotions match or for the remaining 2/3 cases

#### 4. Factory Interface

Factories have a dedicated interface to manage their promotions:

* View all materials associated with the factory
* Allocate credits to specific materials
* Track promotion performance and ROI
* Start/stop promotions as needed

### Database Schema

#### material\_promotions Table

- id: UUID (primary key)  
- material\_id: UUID (foreign key to materials)  
- factory\_id: UUID (foreign key to users)  
- credits\_allocated: INTEGER  
- status: TEXT (active/inactive/completed/pending)  
- start\_date: TIMESTAMP  
- end\_date: TIMESTAMP (nullable)  
- usage\_count: INTEGER  
- impression\_count: INTEGER  
- created\_at: TIMESTAMP  
- updated\_at: TIMESTAMP

- id: UUID (primary key)  
- material\_id: UUID (foreign key to materials)  
- factory\_id: UUID (foreign key to users)  
- credits\_allocated: INTEGER  
- status: TEXT (active/inactive/completed/pending)  
- start\_date: TIMESTAMP  
- end\_date: TIMESTAMP (nullable)  
- usage\_count: INTEGER  
- impression\_count: INTEGER  
- created\_at: TIMESTAMP  
- updated\_at: TIMESTAMP

#### credit\_transactions Table (Updated)

- Added 'promotion' as a valid type  
- Added promotion\_id field (foreign key to material\_promotions)

- Added 'promotion' as a valid type  
- Added promotion\_id field (foreign key to material\_promotions)

### API Endpoints

#### Factory Material Management

* GET /api/factory/materials - Get all materials associated with the factory
* GET /api/factory/promotions - Get all promotions for the factory
* GET /api/factory/promotions/:id - Get a specific promotion
* POST /api/factory/promotions - Create a new promotion (allocate credits)
* PUT /api/factory/promotions/:id/status - Update a promotion's status
* GET /api/factory/promotions/analytics - Get promotion analytics

GET /api/factory/materials

GET /api/factory/promotions

GET /api/factory/promotions/:id

POST /api/factory/promotions

PUT /api/factory/promotions/:id/status

GET /api/factory/promotions/analytics

### Implementation Details

#### Credit Allocation

When a factory allocates credits to promote a material:

1. The system checks if the factory has enough credits
2. Credits are deducted from the factory's balance
3. A credit transaction of type 'promotion' is created
4. The material promotion is created or updated with the allocated credits

#### Material Selection

When a 3D model is being generated:

1. The system extracts material types needed for the model
2. For each material type, there's a 1/3 chance of using a promoted material
3. If a promoted material is selected, the system:
4. Records an impression
5. If the material is actually used in the final model, records a usage
6. Updates analytics for the promotion

#### Analytics

The system provides detailed analytics for factories:

* Total credits allocated to promotions
* Impression count (how many times the material was considered)
* Usage count (how many times the material was actually used)
* Usage rate (usage count / impression count)
* ROI metrics based on credit cost and usage

### Security Considerations

1. Access Control:
2. Only factory users can manage their own promotions
3. Factories can only promote materials they own
4. Admin users can view and manage all promotions
5. Rate Limiting:
6. Appropriate rate limits are applied to promotion-related endpoints
7. Credit allocation is validated to prevent abuse
8. Audit Trail:
9. All promotion-related activities are logged
10. Credit transactions provide a clear audit trail

Admin users can view and manage all promotions

Rate Limiting:

Credit allocation is validated to prevent abuse

Audit Trail:

### System Integration

The Material Promotion System integrates with several existing systems:

#### Module-Based Access Control Integration

The system is implemented as a module in the Module-Based Access Control system:  
- Factory routes use the requireModuleAccess('materialPromotion') middleware  
- Factory subscription tiers include the module (disabled by default)  
- The module can be enabled/disabled per subscription tier through the admin panel

requireModuleAccess('materialPromotion')

#### API Endpoints

The system exposes the following API endpoints:  
- GET /api/factory/materials - Get factory materials that can be promoted  
- GET /api/factory/promotions - Get all promotions for the factory  
- GET /api/factory/promotions/:id - Get a specific promotion  
- POST /api/factory/promotions - Create a new promotion (allocate credits)  
- PUT /api/factory/promotions/:id/status - Update a promotion's status  
- GET /api/factory/promotions/analytics - Get promotion analytics

GET /api/factory/materials

GET /api/factory/promotions

GET /api/factory/promotions/:id

POST /api/factory/promotions

PUT /api/factory/promotions/:id/status

GET /api/factory/promotions/analytics

#### 3D Model Generation Integration

The material promotion system integrates with the 3D model generation process:  
- When a 3D model is generated, the system checks for promoted materials that match the requested material type  
- There's a 1/3 chance of selecting a promoted material when a match is found  
- The system tracks impressions (when a promoted material is considered) and usage (when a promoted material is actually used)  
- The selection is weighted by the number of credits allocated to each promotion

### Future Enhancements

Potential future enhancements to the Material Promotion System:

1. Advanced Targeting:
2. Target promotions by user demographics
3. Target promotions by project type
4. Seasonal promotion scheduling
5. Enhanced Analytics:
6. Conversion tracking for promoted materials
7. A/B testing for promotion effectiveness
8. Predictive analytics for optimal credit allocation
9. Integration with Other Systems:
10. Integration with e-commerce systems
11. Integration with marketing campaigns
12. Integration with customer relationship management systems

Seasonal promotion scheduling

Enhanced Analytics:

Predictive analytics for optimal credit allocation

Integration with Other Systems:

# Material Property Analytics

Source: readme/material-property-analytics.md

---

## Material Property Analytics

The Material Property Analytics feature provides analytics and insights about material properties, trends, and anomalies across the database.

### Features

#### Property Distribution Analysis

* Distribution Visualization: View the distribution of property values across materials
* Statistical Analysis: Calculate key statistics like mean, median, mode, and standard deviation
* Filtering by Material Type: Filter distributions by material type
* Multiple Chart Types: View distributions as bar charts or pie charts
* Percentage Analysis: See the percentage of materials with each property value

#### Property Trend Analysis

* Time-Based Trends: Track how property values change over time
* Multiple Time Units: View trends by day, week, month, or year
* Count Tracking: Monitor the number of materials with specific properties over time
* Multiple Chart Types: View trends as line charts or area charts
* Customizable Time Ranges: Focus on specific time periods

#### Property Correlation Analysis

* Correlation Visualization: See how different properties relate to each other
* Correlation Coefficient: Calculate the statistical correlation between properties
* Scatter Plots: Visualize relationships between numeric properties
* Correlation Strength: Interpret the strength and direction of correlations
* Material Type Filtering: Focus on correlations within specific material types

#### Property Anomaly Detection

* Outlier Identification: Find materials with unusual property values
* Z-Score Analysis: Use statistical methods to identify anomalies
* Adjustable Thresholds: Set the sensitivity of anomaly detection
* Detailed Explanations: Understand why a material is considered an anomaly
* Direct Material Access: Quickly navigate to anomalous materials for inspection

### Technical Implementation

#### Backend

* Analytics Service: Handles property analytics calculations
* Statistical Algorithms: Implements algorithms for distribution, trends, correlation, and anomaly detection
* API Endpoints: Provides endpoints for accessing analytics data
* Caching: Optimizes performance for frequently accessed analytics

#### Frontend

* Interactive Dashboard: Provides a comprehensive view of property analytics
* Visualization Components: Implements charts and graphs for data visualization
* Filtering Controls: Allows users to filter and customize analytics views
* Responsive Design: Ensures analytics are accessible on all devices
* Export Capabilities: Enables exporting of analytics data and visualizations

### Usage

#### Viewing Property Distributions

1. Navigate to the Material Property Analytics dashboard
2. Select the "Distribution" tab
3. Choose a property to analyze
4. Optionally filter by material type
5. View the distribution chart and statistics
6. Switch between bar chart and pie chart views

#### Analyzing Property Trends

1. Navigate to the Material Property Analytics dashboard
2. Select the "Trends" tab
3. Choose a property to analyze
4. Select a time unit (day, week, month, year)
5. Optionally filter by material type
6. View the trend chart
7. Switch between line chart and area chart views

#### Exploring Property Correlations

1. Navigate to the Material Property Analytics dashboard
2. Select the "Correlation" tab
3. Choose two properties to correlate
4. Optionally filter by material type
5. View the scatter plot and correlation statistics
6. Interpret the correlation strength and direction

#### Detecting Property Anomalies

1. Navigate to the Material Property Analytics dashboard
2. Select the "Anomalies" tab
3. Choose a property to analyze
4. Adjust the Z-score threshold as needed
5. Optionally filter by material type
6. View the list of anomalous materials
7. Click on a material to inspect it in detail

### Benefits

* Data-Driven Insights: Gain insights into material property trends and patterns
* Quality Control: Identify unusual materials that may have data quality issues
* Market Analysis: Understand trends in material properties over time
* Property Relationships: Discover correlations between different material properties
* Decision Support: Make informed decisions based on property analytics
* Anomaly Detection: Quickly identify outliers that may require attention

# Material Recognition

Source: readme/material-recognition.md

---

## Material Recognition System

The Material Recognition System is a core component of Kai that enables identification and matching of materials from images. This document provides a detailed overview of how this system works, its features, and implementation details.

### Features

#### Specialized Tile Pattern Recognition

The system includes specialized capabilities for recognizing tile patterns, particularly from low-quality PDF catalogs and images:

1. TilePatternProcessor
2. Specialized processor optimized for tile pattern recognition
3. Quality-adaptive processing pipeline that adjusts based on input quality
4. Advanced feature extraction techniques optimized for tiles:  
   Local Binary Patterns (LBP) for texture analysis  
   Gabor filters for directional textures  
   HOG (Histogram of Oriented Gradients) for pattern boundaries  
   Grey Level Co-occurrence Matrices (GLCM) for repeating motifs  
   Wavelet transforms for texture properties common in tiles
5. Geometric transformation handling for rotated or perspective-distorted images
6. Multi-modal recognition combining visual features with extracted specifications
7. PDF Tile Extraction
8. Specialized extraction of tile patterns from PDF catalogs
9. High-quality extraction with configurable DPI and resolution enhancement
10. Region detection to isolate tile pattern images within documents
11. Extraction of associated metadata (dimensions, specifications, manufacturer info)
12. Multi-page analysis to connect information across catalog pages
13. Automatic input format detection for seamless processing of both PDFs and images
14. Geometric Transformation Handling
15. Correction of rotation, perspective, and scaling issues in tile images
16. Detection of tile grid patterns to normalize viewing angles
17. Keypoint-based matching that's robust to different viewing conditions
18. Homography transformation to achieve a normalized frontal view
19. Unified Processing Approach
20. Smart entry point that automatically detects input type (PDF vs. direct image)
21. Format detection using file signatures and content analysis
22. Seamless routing to the appropriate specialized processing pipeline
23. Consistent output format regardless of input type

* Local Binary Patterns (LBP) for texture analysis
* Gabor filters for directional textures
* HOG (Histogram of Oriented Gradients) for pattern boundaries
* Grey Level Co-occurrence Matrices (GLCM) for repeating motifs
* Wavelet transforms for texture properties common in tiles

Multi-modal recognition combining visual features with extracted specifications

PDF Tile Extraction

Automatic input format detection for seamless processing of both PDFs and images

Geometric Transformation Handling

Homography transformation to achieve a normalized frontal view

Unified Processing Approach

#### Multi-Strategy Recognition

The system uses multiple recognition strategies that can be used individually or in combination:

1. Feature-Based Recognition
2. Uses computer vision algorithms to extract distinctive visual features
3. Identifies materials based on texture, pattern, and color characteristics
4. Performs well even with partial images or different lighting conditions
5. Implementation based on enhanced SIFT/SURF feature extraction with custom descriptors
6. Neural Network Recognition
7. Uses deep learning models trained on material datasets
8. Excellent at category classification and general material identification
9. Leverages transfer learning from pre-trained models optimized for material recognition
10. Supports multiple model architectures (MobileNetV2, ResNet18, EfficientNet)
11. Hybrid Approach
12. Combines the strengths of both feature-based and neural network methods
13. Uses confidence fusion to produce more reliable results
14. Dynamically adjusts weight based on confidence levels
15. Superior performance for specialized material types

Implementation based on enhanced SIFT/SURF feature extraction with custom descriptors

Neural Network Recognition

Supports multiple model architectures (MobileNetV2, ResNet18, EfficientNet)

Hybrid Approach

#### Confidence Fusion

The confidence fusion system merges results from multiple recognition methods to improve accuracy:

##### Fusion Methods

1. Weighted Average
2. Combines scores using configurable weights for each method
3. Formula: fusion\_score = (w1 \* score1 + w2 \* score2) / (w1 + w2)
4. Weights can be adjusted based on historical performance for specific material types
5. Adaptive Fusion
6. Automatically adjusts weights based on confidence of each method
7. Gives more influence to methods with higher confidence
8. Formula: weight\_i = confidence\_i^alpha / sum(confidence\_j^alpha)
9. Parameter alpha controls the adaptivity (higher values favor higher confidence methods)
10. Maximum Score
11. Uses the highest confidence score from any method
12. Useful when one method is significantly more confident than others
13. Formula: fusion\_score = max(score1, score2, ...)
14. Product Fusion
15. Multiplies confidence scores together
16. Particularly effective when methods are complementary
17. Formula: fusion\_score = (score1 \* score2)^(1/n)
18. Ensures that all methods must have reasonable confidence for a high fusion score

fusion\_score = (w1 \* score1 + w2 \* score2) / (w1 + w2)

Weights can be adjusted based on historical performance for specific material types

Adaptive Fusion

weight\_i = confidence\_i^alpha / sum(confidence\_j^alpha)

Parameter alpha controls the adaptivity (higher values favor higher confidence methods)

alpha

Maximum Score

Formula: fusion\_score = max(score1, score2, ...)

fusion\_score = max(score1, score2, ...)

Product Fusion

fusion\_score = (score1 \* score2)^(1/n)

##### Implementation

The confidence fusion is implemented in the confidence\_fusion.py module with a TypeScript interface in the ML package:

confidence\_fusion.py

interface FusionOptions {  
 fusionMethod: 'weighted' | 'adaptive' | 'max' | 'product';  
 fusionAlpha?: number; // For adaptive fusion  
 weights?: number[]; // For weighted fusion  
}  
  
interface RecognitionResult {  
 matches: Array<{  
 materialId: string;  
 confidence: number;  
 features?: Record<string, number>;  
 }>;  
 metadata: Record<string, any>;  
}  
  
interface ConfidenceFusionResult extends RecognitionResult {  
 sourceResults: RecognitionResult[];  
 fusionMethod: string;  
 fusionParameters: Record<string, any>;  
}

interface FusionOptions {  
 fusionMethod: 'weighted' | 'adaptive' | 'max' | 'product';  
 fusionAlpha?: number; // For adaptive fusion  
 weights?: number[]; // For weighted fusion  
}  
  
interface RecognitionResult {  
 matches: Array<{  
 materialId: string;  
 confidence: number;  
 features?: Record<string, number>;  
 }>;  
 metadata: Record<string, any>;  
}  
  
interface ConfidenceFusionResult extends RecognitionResult {  
 sourceResults: RecognitionResult[];  
 fusionMethod: string;  
 fusionParameters: Record<string, any>;  
}

#### Vector Similarity Search

Vector similarity search enables finding visually similar materials using embedding vectors:

##### Features

1. Embedding Generation
2. Creates vector representations (embeddings) of material images
3. Uses neural networks to generate high-dimensional feature vectors
4. Embeddings capture visual characteristics in a format optimized for similarity search
5. Supports multiple embedding models (ResNet, EfficientNet, CLIP)
6. Similarity Search
7. Fast nearest neighbor search using optimized indexes
8. FAISS library provides efficient similarity computation
9. Supports filtering by material type and other metadata
10. Configurable threshold for minimum similarity
11. Results Enhancement
12. Re-ranking of initial results using additional criteria
13. Consideration of material metadata for improved relevance
14. Optional hybrid scoring that combines visual and metadata similarity

Supports multiple embedding models (ResNet, EfficientNet, CLIP)

Similarity Search

Configurable threshold for minimum similarity

Results Enhancement

##### Implementation

interface SearchOptions {  
 numResults?: number; // Maximum number of results to return  
 threshold?: number; // Minimum similarity threshold (0-1)  
 materialType?: string | string[]; // Filter by material type  
 filter?: Record<string, any>; // Additional filters  
}  
  
interface VectorSearchResult {  
 results: Array<{  
 materialId: string;  
 similarity: number;  
 material: MaterialDocument;  
 }>;  
 query: {  
 vector: number[];  
 filters: Record<string, any>;  
 };  
 timingMs: number;  
}

interface SearchOptions {  
 numResults?: number; // Maximum number of results to return  
 threshold?: number; // Minimum similarity threshold (0-1)  
 materialType?: string | string[]; // Filter by material type  
 filter?: Record<string, any>; // Additional filters  
}  
  
interface VectorSearchResult {  
 results: Array<{  
 materialId: string;  
 similarity: number;  
 material: MaterialDocument;  
 }>;  
 query: {  
 vector: number[];  
 filters: Record<string, any>;  
 };  
 timingMs: number;  
}

#### Results Visualization

The system includes visualization capabilities to help users understand and validate recognition results:

1. Side-by-Side Comparison
2. Displays query image alongside top matches
3. Highlights key feature matches between images
4. Provides zoom and pan functionality for detailed inspection
5. Confidence Visualization
6. Color-coded confidence indicators
7. Graphical representation of confidence scores
8. Comparison of confidence across different methods
9. Similarity Explanation
10. Highlights regions contributing to the match
11. Explains which features were most important for the match
12. Shows similar material properties from the knowledge base

Provides zoom and pan functionality for detailed inspection

Confidence Visualization

Comparison of confidence across different methods

Similarity Explanation

### Technical Implementation

#### Recognition Pipeline

The material recognition pipeline consists of several stages:

1. Image Preprocessing
2. Resizing and normalization
3. Background removal (optional)
4. Color correction
5. Enhancement for feature extraction
6. Feature Extraction
7. Feature-based descriptors generation
8. Neural network embedding creation
9. Color histogram analysis
10. Texture pattern extraction
11. Matching
12. Database lookup of similar feature vectors
13. Neural network classification
14. Hybrid matching combining multiple methods
15. Filtering and threshold application
16. Post-processing
17. Results fusion from multiple methods
18. Confidence calculation
19. Metadata enrichment from knowledge base
20. Results formatting and sorting

Enhancement for feature extraction

Feature Extraction

Texture pattern extraction

Matching

Filtering and threshold application

Post-processing

#### Model Training

The recognition models are trained on a diverse dataset of material images:

1. Training Dataset Organization  
   dataset/  
    ├── tile/  
    │ ├── image1.jpg  
    │ ├── image2.jpg  
    │ └── ...  
    ├── stone/  
    │ ├── image1.jpg  
    │ ├── image2.jpg  
    │ └── ...  
    ├── wood/  
    │ ├── image1.jpg  
    │ ├── image2.jpg  
    │ └── ...  
    └── ...
2. Training Process in Detail
3. Base Model Selection:   
     
   TensorFlow or PyTorch pre-trained models are loaded dynamically  
   Popular architectures include MobileNetV2, ResNet18/50, and EfficientNet  
   Base models are not stored in our repository but loaded from the frameworks
4. Transfer Learning Approach:  
     
   Initial layers of base models are frozen to preserve general features  
   Classification layers are replaced with custom layers for material recognition  
   Gradual unfreezing during training for fine-tuning
5. Data Processing Pipeline:  
     
   Comprehensive data augmentation (rotation, scaling, color shifts, flips)  
   Material-specific augmentation strategies based on material properties  
   Normalization specific to each model architecture
6. Training Optimization:  
     
   Sparse categorical cross-entropy loss for classification tasks  
   Bayesian hyperparameter optimization for learning rate, batch size, etc.  
   Early stopping with validation loss monitoring  
   Learning rate reduction on plateau
7. Model Storage:  
     
   Trained models are saved with metadata in the specified output directory  
   Complete with training history and hyperparameters for reproducibility  
   Versioned for tracking improvements
8. Performance Metrics
9. Top-1 and Top-5 accuracy across different material categories
10. Precision and recall per material type
11. Confusion matrix for understanding misclassifications
12. Mean Average Precision (mAP) for ranked results
13. Inference time on various hardware profiles
14. Embedding quality metrics for similarity search applications
15. Model-to-Vector Pipeline
16. Trained models generate embeddings for material images
17. These embeddings are stored in the vector database (not the models themselves)
18. FAISS indexing enables efficient similarity search
19. Embeddings link to knowledge base entries through material IDs

Training Dataset Organization  
dataset/  
 ├── tile/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── ...  
 ├── stone/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── ...  
 ├── wood/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── ...  
 └── ...

dataset/  
 ├── tile/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── ...  
 ├── stone/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── ...  
 ├── wood/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── ...  
 └── ...

Training Process in Detail

Base Model Selection:

* TensorFlow or PyTorch pre-trained models are loaded dynamically
* Popular architectures include MobileNetV2, ResNet18/50, and EfficientNet
* Base models are not stored in our repository but loaded from the frameworks

Transfer Learning Approach:

* Initial layers of base models are frozen to preserve general features
* Classification layers are replaced with custom layers for material recognition
* Gradual unfreezing during training for fine-tuning

Data Processing Pipeline:

* Comprehensive data augmentation (rotation, scaling, color shifts, flips)
* Material-specific augmentation strategies based on material properties
* Normalization specific to each model architecture

Training Optimization:

* Sparse categorical cross-entropy loss for classification tasks
* Bayesian hyperparameter optimization for learning rate, batch size, etc.
* Early stopping with validation loss monitoring
* Learning rate reduction on plateau

Model Storage:

* Trained models are saved with metadata in the specified output directory
* Complete with training history and hyperparameters for reproducibility
* Versioned for tracking improvements

Performance Metrics

Embedding quality metrics for similarity search applications

Model-to-Vector Pipeline

#### Integration with Knowledge Base

The recognition system is tightly integrated with the knowledge base:

1. Material Lookup
2. Recognition results include full material information
3. Enrichment with specifications from knowledge base
4. Relationship data for similar or complementary materials
5. Feedback Loop
6. User feedback on recognition results improves system over time
7. Incorrect matches are analyzed to improve training
8. Confidence scoring is adjusted based on feedback
9. Active learning selects ambiguous samples for expert labeling
10. Continuous Improvement
11. New materials added to the knowledge base are incorporated into training
12. Recognition models are periodically retrained with enhanced datasets
13. Feature extractors are fine-tuned based on performance analysis
14. Automated retraining triggers monitor system performance metrics

Relationship data for similar or complementary materials

Feedback Loop

Active learning selects ambiguous samples for expert labeling

Continuous Improvement

### API Usage

#### Basic Recognition

#### Implementation Note

All mock implementations previously used during development have been replaced with fully functional real API calls to the backend services. The system now uses actual server endpoints for all recognition functionality, with API fallbacks only when services are temporarily unavailable.

#### API Usage

import { recognizeMaterial } from '@kai/ml';  
  
async function identifyMaterial() {  
 try {  
 const result = await recognizeMaterial('path/to/image.jpg', {  
 modelType: 'hybrid', // 'hybrid', 'feature-based', or 'ml-based'  
 confidenceThreshold: 0.6,  
 maxResults: 5  
 });  
  
 console.log('Recognized materials:');  
 result.matches.forEach(match => {  
 console.log(`- ${match.materialId} (confidence: ${match.confidence.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Material recognition failed:', error);  
 }  
}

import { recognizeMaterial } from '@kai/ml';  
  
async function identifyMaterial() {  
 try {  
 const result = await recognizeMaterial('path/to/image.jpg', {  
 modelType: 'hybrid', // 'hybrid', 'feature-based', or 'ml-based'  
 confidenceThreshold: 0.6,  
 maxResults: 5  
 });  
  
 console.log('Recognized materials:');  
 result.matches.forEach(match => {  
 console.log(`- ${match.materialId} (confidence: ${match.confidence.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Material recognition failed:', error);  
 }  
}

#### Enhanced Recognition with Confidence Fusion

import { recognizeMaterialEnhanced } from '@kai/ml';  
  
async function identifyMaterialEnhanced() {  
 try {  
 const result = await recognizeMaterialEnhanced('path/to/image.jpg', {  
 useFusion: true,  
 fusionMethod: 'adaptive', // 'weighted', 'adaptive', 'max', or 'product'  
 fusionAlpha: 0.5,  
 confidenceThreshold: 0.6,  
 maxResults: 5  
 });  
  
 console.log('Recognized materials:');  
 result.matches.forEach(match => {  
 console.log(`- ${match.materialId} (confidence: ${match.confidence.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Material recognition failed:', error);  
 }  
}

import { recognizeMaterialEnhanced } from '@kai/ml';  
  
async function identifyMaterialEnhanced() {  
 try {  
 const result = await recognizeMaterialEnhanced('path/to/image.jpg', {  
 useFusion: true,  
 fusionMethod: 'adaptive', // 'weighted', 'adaptive', 'max', or 'product'  
 fusionAlpha: 0.5,  
 confidenceThreshold: 0.6,  
 maxResults: 5  
 });  
  
 console.log('Recognized materials:');  
 result.matches.forEach(match => {  
 console.log(`- ${match.materialId} (confidence: ${match.confidence.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Material recognition failed:', error);  
 }  
}

#### Client-Side Implementation

The client application now uses a dedicated recognitionService with actual API calls:

// Using the recognition service with real API calls  
import { recognitionService } from '../services/recognitionService';  
  
async function identifyMaterial(imageFile) {  
 try {  
 const formData = new FormData();  
 formData.append('image', imageFile);  
  
 const result = await recognitionService.identifyMaterial(formData, {  
 confidenceThreshold: 0.6,  
 maxResults: 5  
 });  
  
 // Process the real API results  
 console.log('Recognized materials:');  
 result.matches.forEach(match => {  
 console.log(`- ${match.materialId} (confidence: ${match.confidence.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Material recognition failed:', error);  
 // API may still provide graceful fallbacks for service unavailability  
 }  
}

// Using the recognition service with real API calls  
import { recognitionService } from '../services/recognitionService';  
  
async function identifyMaterial(imageFile) {  
 try {  
 const formData = new FormData();  
 formData.append('image', imageFile);  
  
 const result = await recognitionService.identifyMaterial(formData, {  
 confidenceThreshold: 0.6,  
 maxResults: 5  
 });  
  
 // Process the real API results  
 console.log('Recognized materials:');  
 result.matches.forEach(match => {  
 console.log(`- ${match.materialId} (confidence: ${match.confidence.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Material recognition failed:', error);  
 // API may still provide graceful fallbacks for service unavailability  
 }  
}

#### Vector Search

import { createVectorSearchIndex, searchSimilarMaterials } from '@kai/ml';  
  
async function setupAndSearch() {  
 try {  
 // Create search index (one-time setup)  
 await createVectorSearchIndex('path/to/embeddings', 'models/search\_index.faiss');  
  
 // Search for similar materials  
 const result = await searchSimilarMaterials('models/search\_index.faiss', 'path/to/query.jpg', {  
 numResults: 5,  
 threshold: 0.7  
 });  
  
 console.log('Similar materials:');  
 result.results.forEach(match => {  
 console.log(`- ${match.materialId} (similarity: ${match.similarity.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Vector search failed:', error);  
 }  
}

import { createVectorSearchIndex, searchSimilarMaterials } from '@kai/ml';  
  
async function setupAndSearch() {  
 try {  
 // Create search index (one-time setup)  
 await createVectorSearchIndex('path/to/embeddings', 'models/search\_index.faiss');  
  
 // Search for similar materials  
 const result = await searchSimilarMaterials('models/search\_index.faiss', 'path/to/query.jpg', {  
 numResults: 5,  
 threshold: 0.7  
 });  
  
 console.log('Similar materials:');  
 result.results.forEach(match => {  
 console.log(`- ${match.materialId} (similarity: ${match.similarity.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Vector search failed:', error);  
 }  
}

#### Results Visualization

import { visualizeSearchResults } from '@kai/ml';  
  
async function visualizeResults() {  
 try {  
 const outputPath = await visualizeSearchResults(  
 'models/search\_index.faiss',  
 'path/to/query.jpg',  
 'output/visualization.jpg',  
 5 // Number of results to visualize  
 );  
  
 console.log(`Visualization saved to ${outputPath}`);  
 } catch (error) {  
 console.error('Visualization failed:', error);  
 }  
}

import { visualizeSearchResults } from '@kai/ml';  
  
async function visualizeResults() {  
 try {  
 const outputPath = await visualizeSearchResults(  
 'models/search\_index.faiss',  
 'path/to/query.jpg',  
 'output/visualization.jpg',  
 5 // Number of results to visualize  
 );  
  
 console.log(`Visualization saved to ${outputPath}`);  
 } catch (error) {  
 console.error('Visualization failed:', error);  
 }  
}

#### Tile Pattern Recognition from PDF

import { recognizeInput } from '@kai/ml';  
  
async function processTileCatalog() {  
 try {  
 // Unified entry point automatically detects PDF format  
 const results = await recognizeInput('path/to/tile-catalog.pdf', {  
 confidenceThreshold: 0.6,  
 extractMetadata: true, // Extract specifications from text  
 enhanceQuality: true // Apply super-resolution to low-quality images  
 });  
  
 console.log(`Found ${results.length} tile patterns in the catalog:`);  
 results.forEach((result, index) => {  
 console.log(`\nPattern ${index + 1}:`);  
 console.log(`- Type: ${result.materialType}`);  
 console.log(`- Pattern Family: ${result.metadata.patternFamily || 'Unknown'}`);  
 console.log(`- Confidence: ${result.confidence.toFixed(2)}`);  
 console.log(`- Quality Assessment: ${JSON.stringify(result.qualityAssessment)}`);  
 console.log(`- Dimensions: ${result.metadata.dimensions || 'Not specified'}`);  
  
 if (result.similarPatterns && result.similarPatterns.length > 0) {  
 console.log('- Similar patterns:');  
 result.similarPatterns.forEach(similar => {  
 console.log(` \* ${similar.materialId} (similarity: ${similar.similarity.toFixed(2)})`);  
 });  
 }  
 });  
 } catch (error) {  
 console.error('PDF recognition failed:', error);  
 }  
}

import { recognizeInput } from '@kai/ml';  
  
async function processTileCatalog() {  
 try {  
 // Unified entry point automatically detects PDF format  
 const results = await recognizeInput('path/to/tile-catalog.pdf', {  
 confidenceThreshold: 0.6,  
 extractMetadata: true, // Extract specifications from text  
 enhanceQuality: true // Apply super-resolution to low-quality images  
 });  
  
 console.log(`Found ${results.length} tile patterns in the catalog:`);  
 results.forEach((result, index) => {  
 console.log(`\nPattern ${index + 1}:`);  
 console.log(`- Type: ${result.materialType}`);  
 console.log(`- Pattern Family: ${result.metadata.patternFamily || 'Unknown'}`);  
 console.log(`- Confidence: ${result.confidence.toFixed(2)}`);  
 console.log(`- Quality Assessment: ${JSON.stringify(result.qualityAssessment)}`);  
 console.log(`- Dimensions: ${result.metadata.dimensions || 'Not specified'}`);  
  
 if (result.similarPatterns && result.similarPatterns.length > 0) {  
 console.log('- Similar patterns:');  
 result.similarPatterns.forEach(similar => {  
 console.log(` \* ${similar.materialId} (similarity: ${similar.similarity.toFixed(2)})`);  
 });  
 }  
 });  
 } catch (error) {  
 console.error('PDF recognition failed:', error);  
 }  
}

1. Inference Optimization
2. Batch processing for multiple images
3. GPU acceleration for neural network inference
4. Caching of intermediate results
5. Model quantization for faster processing
6. Scaling Considerations
7. Horizontal scaling for high-volume processing
8. Prioritization queue for real-time vs. batch recognition
9. Result caching for frequently requested images
10. Auto-scaling based on load
11. Resource Requirements
12. Memory: 4GB+ for optimal performance
13. GPU: Recommended for production deployment
14. Storage: ~500MB for models and feature descriptors
15. CPU: 4+ cores recommended for parallel processing

Model quantization for faster processing

Scaling Considerations

Auto-scaling based on load

Resource Requirements

# Material Specific Ocr

Source: readme/material-specific-ocr.md

---

## Material-Specific OCR Processing

This document describes the material-specific OCR processing system implemented in the platform. The system enhances OCR extraction by using material type detection and material-specific metadata fields.

### Overview

The material-specific OCR processing system consists of the following components:

1. Material Type Detector: Detects the material type from OCR text and/or images
2. Metadata Field Utilities: Retrieves metadata fields specific to a material type
3. Material-Specific OCR Extractor: Extracts metadata using material-specific patterns and context-aware processing

### Material Type Detection

The system first detects the material type from the OCR text and/or image. This is a crucial first step that determines which metadata fields should be used for extraction.

#### Detection Methods

* Text-based detection: Uses keyword matching and pattern recognition to identify material types from text
* Image-based detection: Uses ML models to classify images into material types
* Hybrid detection: Combines text and image detection for higher accuracy

#### Supported Material Types

* tile: Ceramic, porcelain, mosaic, and other tile materials
* wood: Hardwood, engineered wood, laminate, and other wood materials
* lighting: Lamps, fixtures, and other lighting products
* furniture: Chairs, tables, sofas, and other furniture
* decoration: Decorative items like vases, artwork, and rugs
* all: Common fields applicable to all material types

tile

wood

lighting

furniture

decoration

all

### Metadata Field Filtering

Once the material type is detected, the system retrieves metadata fields specific to that material type. This ensures that only relevant fields are used for extraction.

#### Field Selection Process

1. Get all metadata fields for the detected material type
2. Include common fields applicable to all material types
3. Filter out inactive or irrelevant fields

### Material-Specific Extraction

The system uses material-specific extraction patterns and context-aware processing to extract metadata from OCR text.

#### Extraction Methods

* Pattern-based extraction: Uses regular expressions specific to each field and material type
* Hint-based extraction: Uses field hints to locate and extract values
* Context-aware extraction: Uses context and relationships between fields to enhance extraction

#### Context-Aware Processing

The system implements material-specific context-aware processing to enhance extraction:

* For tiles:
* Extract size from dimensions and vice versa
* Validate thickness values against typical ranges for tiles
* For wood:
* Extract width and length from dimensions
* Validate thickness values against typical ranges for wood
* For lighting:
* Extract wattage and lumens from technical specifications
* Convert between different units (e.g., watts to lumens)

Validate thickness values against typical ranges for tiles

For wood:

Validate thickness values against typical ranges for wood

For lighting:

### API Endpoints

The system exposes the following API endpoints:

#### Detect Material Type

POST /api/ocr/detect-material-type

POST /api/ocr/detect-material-type

Request body:

{  
 "text": "OCR text content",  
 "imagePath": "optional/path/to/image.jpg"  
}

{  
 "text": "OCR text content",  
 "imagePath": "optional/path/to/image.jpg"  
}

Response:

{  
 "success": true,  
 "materialType": "tile",  
 "confidence": 0.85,  
 "keywords": ["ceramic", "porcelain", "glazed"]  
}

{  
 "success": true,  
 "materialType": "tile",  
 "confidence": 0.85,  
 "keywords": ["ceramic", "porcelain", "glazed"]  
}

#### Extract Metadata

POST /api/ocr/extract-metadata

POST /api/ocr/extract-metadata

Request body:

{  
 "text": "OCR text content",  
 "imagePath": "optional/path/to/image.jpg"  
}

{  
 "text": "OCR text content",  
 "imagePath": "optional/path/to/image.jpg"  
}

Response:

{  
 "success": true,  
 "materialType": "tile",  
 "materialTypeConfidence": 0.85,  
 "extractedFields": {  
 "manufacturer": "Example Tile Co.",  
 "collection": "Modern Series",  
 "size": "60x60",  
 "thickness": 10,  
 "color": "White"  
 },  
 "extractionConfidence": {  
 "manufacturer": 0.9,  
 "collection": 0.8,  
 "size": 0.95,  
 "thickness": 0.85,  
 "color": 0.7  
 },  
 "extractionMethods": {  
 "manufacturer": "pattern",  
 "collection": "hint",  
 "size": "pattern",  
 "thickness": "pattern",  
 "color": "context"  
 },  
 "processingTime": 235  
}

{  
 "success": true,  
 "materialType": "tile",  
 "materialTypeConfidence": 0.85,  
 "extractedFields": {  
 "manufacturer": "Example Tile Co.",  
 "collection": "Modern Series",  
 "size": "60x60",  
 "thickness": 10,  
 "color": "White"  
 },  
 "extractionConfidence": {  
 "manufacturer": 0.9,  
 "collection": 0.8,  
 "size": 0.95,  
 "thickness": 0.85,  
 "color": 0.7  
 },  
 "extractionMethods": {  
 "manufacturer": "pattern",  
 "collection": "hint",  
 "size": "pattern",  
 "thickness": "pattern",  
 "color": "context"  
 },  
 "processingTime": 235  
}

#### Test Extraction Pattern

POST /api/ocr/test-extraction-pattern

POST /api/ocr/test-extraction-pattern

Request body:

{  
 "pattern": "thickness:?\\s\*(\\d+(?:\\.\\d+)?)\\s\*mm",  
 "text": "Product specifications: thickness: 10.5 mm",  
 "fieldType": "number"  
}

{  
 "pattern": "thickness:?\\s\*(\\d+(?:\\.\\d+)?)\\s\*mm",  
 "text": "Product specifications: thickness: 10.5 mm",  
 "fieldType": "number"  
}

Response:

{  
 "success": true,  
 "matched": true,  
 "value": 10.5,  
 "confidence": 0.9,  
 "method": "pattern"  
}

{  
 "success": true,  
 "matched": true,  
 "value": 10.5,  
 "confidence": 0.9,  
 "method": "pattern"  
}

### Integration with ML Training

The material-specific OCR system is integrated with the ML training pipeline:

1. ML training uses material-specific metadata fields
2. Training data is filtered by material type
3. Models are trained to recognize material-specific properties

This integration ensures that the ML models are optimized for each material type, improving recognition accuracy.

### Future Enhancements

Planned enhancements for the material-specific OCR system:

1. Improved material type detection: Enhance detection accuracy with more advanced ML models
2. More material types: Add support for additional material types like fabric, metal, and glass
3. Enhanced context-aware processing: Implement more sophisticated context-aware extraction rules
4. Feedback loop: Incorporate user feedback to improve extraction patterns and rules
5. Multi-language support: Add support for extracting metadata from OCR text in multiple languages

# Material Specific Ui

Source: readme/material-specific-ui.md

---

## Material-Specific UI Components

This document describes the material-specific UI components implemented in the platform. These components ensure consistent filtering of metadata fields by material type and provide visual indicators for material types.

### Overview

The material-specific UI components consist of the following:

1. MaterialTypeSelector: A dropdown selector for material types with icons and descriptions
2. MaterialTypeIndicator: A chip component that indicates the material type with an icon and color
3. MetadataFieldFilter: A filter component for metadata fields with material type filtering
4. Enhanced MetadataFieldManager: An updated manager component that uses material-specific filtering

### Material Type Selector

The MaterialTypeSelector component provides a consistent way to select material types across the platform.

MaterialTypeSelector

#### Features

* Material type options with icons and descriptions
* Consistent styling and behavior
* Support for required/optional states
* Support for disabled state
* Support for different sizes

#### Usage

import MaterialTypeSelector, { MaterialType } from '../../components/common/MaterialTypeSelector';  
  
// In your component  
const [materialType, setMaterialType] = useState<MaterialType>('all');  
  
// In your JSX  
<MaterialTypeSelector  
 value={materialType}  
 onChange={setMaterialType}  
 label="Material Type"  
 required  
 showDescription  
/>

import MaterialTypeSelector, { MaterialType } from '../../components/common/MaterialTypeSelector';  
  
// In your component  
const [materialType, setMaterialType] = useState<MaterialType>('all');  
  
// In your JSX  
<MaterialTypeSelector  
 value={materialType}  
 onChange={setMaterialType}  
 label="Material Type"  
 required  
 showDescription  
/>

### Material Type Indicator

The MaterialTypeIndicator component provides a visual indicator for material types.

MaterialTypeIndicator

#### Features

* Consistent color coding for different material types
* Icons that represent each material type
* Tooltips with descriptions
* Support for different sizes

#### Usage

import MaterialTypeIndicator, { getMaterialTypeFromCategories } from './MaterialTypeIndicator';  
  
// In your JSX  
<MaterialTypeIndicator  
 materialType={getMaterialTypeFromCategories(field.categories)}  
 size="small"  
/>

import MaterialTypeIndicator, { getMaterialTypeFromCategories } from './MaterialTypeIndicator';  
  
// In your JSX  
<MaterialTypeIndicator  
 materialType={getMaterialTypeFromCategories(field.categories)}  
 size="small"  
/>

### Metadata Field Filter

The MetadataFieldFilter component provides a consistent way to filter metadata fields by material type, field type, and search term.

MetadataFieldFilter

#### Features

* Search by name or description
* Filter by field type
* Filter by material type
* Consistent styling and behavior

#### Usage

import MetadataFieldFilter, { MetadataFieldFilterOptions } from './MetadataFieldFilter';  
  
// In your component  
const [filterOptions, setFilterOptions] = useState<MetadataFieldFilterOptions>({  
 searchTerm: '',  
 fieldType: 'all',  
 materialType: 'all',  
 isActive: true  
});  
  
// In your JSX  
<MetadataFieldFilter  
 filterOptions={filterOptions}  
 onFilterChange={setFilterOptions}  
/>

import MetadataFieldFilter, { MetadataFieldFilterOptions } from './MetadataFieldFilter';  
  
// In your component  
const [filterOptions, setFilterOptions] = useState<MetadataFieldFilterOptions>({  
 searchTerm: '',  
 fieldType: 'all',  
 materialType: 'all',  
 isActive: true  
});  
  
// In your JSX  
<MetadataFieldFilter  
 filterOptions={filterOptions}  
 onFilterChange={setFilterOptions}  
/>

### Enhanced Metadata Field Manager

The MetadataFieldManager component has been enhanced to use material-specific filtering and indicators.

MetadataFieldManager

#### Features

* Filter metadata fields by material type
* Visual indicators for material types
* Consistent material type selection in the edit dialog
* Improved user experience with clear filtering options

### Material Type Categories

The platform supports the following material types:

* Tile: Ceramic, porcelain, mosaic, and other tile materials
* Wood: Hardwood, engineered wood, laminate, and other wood materials
* Lighting: Lamps, fixtures, and other lighting products
* Furniture: Chairs, tables, sofas, and other furniture
* Decoration: Decorative items like vases, artwork, and rugs
* All: Common fields applicable to all material types

Each material type has a specific color and icon for consistent visual identification across the platform.

### Integration with Other Components

The material-specific UI components are designed to integrate with other components in the platform:

1. OCR Processing: The material type detection in OCR processing uses the same material types as the UI components
2. ML Training: The material-specific ML training uses the same material types for filtering metadata fields
3. Admin Dashboard: The admin dashboard uses material type indicators to show which material type each field belongs to

This integration ensures a consistent user experience across the platform.

### Future Enhancements

Planned enhancements for the material-specific UI components:

1. Material Type Management: Add a UI for managing material types and their properties
2. Material Type Hierarchy: Implement a hierarchical structure for material types
3. Material Type Relationships: Add support for relationships between material types
4. Material Type Statistics: Add statistics about metadata fields by material type
5. Material Type Validation: Add validation rules specific to each material type

# Mcp Integration

Source: readme/mcp-integration.md

---

## Model Context Protocol (MCP) Integration

This comprehensive guide covers the Model Context Protocol (MCP) system in the Kai platform, including the server architecture, client SDK, API integrations, and agent communication capabilities.

### Table of Contents

1. Overview
2. Architecture
3. MCP Server
4. MCP Client SDK
5. API Integrations
6. High-Priority APIs
7. Medium-Priority APIs
8. Credit System Integration
9. Agent Integration
10. Implementation Details
11. Performance Optimization
12. Troubleshooting
13. Future Enhancements
14. Deployment

### Overview

The Model Context Protocol (MCP) system serves as a centralized middleware and service layer that manages machine learning models, handles inference requests, and provides a unified interface for various AI and ML services across the platform. It follows the Model Context Protocol standard for consistent interface design.

#### Why We Need MCP

The traditional approach of loading machine learning models for each inference request has several limitations:

1. Performance Issues: Loading models for each request adds significant latency
2. Resource Inefficiency: Multiple instances load duplicate copies of the same models
3. Limited Scaling: Difficult to scale model serving independently from application logic
4. Version Management: No centralized mechanism for model versioning or hot-swapping
5. Agent Integration: No standardized way for AI agents to interact with model inference
6. Credit Tracking: No unified system for tracking usage and managing credits

#### Key Benefits

The MCP system addresses these issues and provides several benefits:

1. Improved Performance
2. Models are loaded once and kept in memory
3. Reduced latency for inference requests
4. Efficient resource utilization through batching
5. Enhanced Model Management
6. Centralized versioning for models
7. Ability to perform A/B testing between model versions
8. Hot-swapping models without application restarts
9. Simplified Agent Integration
10. Standardized protocol for agent-model communication
11. Built-in support for agent feedback loops
12. Contextual information sharing between agents and models
13. Scalability
14. Independent scaling of model serving components
15. Better handling of high-volume inference requests
16. Load balancing across multiple model server instances
17. Unified Credit System
18. Centralized tracking of API usage
19. Consistent credit policies across services
20. Clear credit requirement communication to users

Efficient resource utilization through batching

Enhanced Model Management

Hot-swapping models without application restarts

Simplified Agent Integration

Contextual information sharing between agents and models

Scalability

Load balancing across multiple model server instances

Unified Credit System

### Architecture

The MCP implementation uses a hybrid approach with two main components:

#### Python-based MCP Server

* Standalone FastAPI microservice
* Direct access to ML libraries (TensorFlow, PyTorch)
* Optimized for model inference with GPU support
* Model caching and version management
* Centralized credit tracking and management

#### TypeScript Client SDK

* Clean TypeScript interface for the MCP server
* Type-safe API for model management and inference
* Seamless integration with existing Kai components
* Authentication and credit handling

#### Component Adapters

* Specialized adapters for each ML component
* Authentication and metrics support
* Fallback capabilities for reliability
* Batching for improved throughput

#### Communication Flow

┌────────────────┐ ┌─────────────────┐ ┌────────────────┐  
│ │ │ │ │ │  
│ TypeScript │─────▶ TypeScript MCP │─────▶ Python MCP │  
│ Application │◀───── Client SDK │◀───── Server │  
│ │ │ │ │ │  
└────────────────┘ └─────────────────┘ └────────────────┘  
 │  
 ▼  
 ┌────────────────┐  
 │ │  
 │ ML Models │  
 │ (TF/PyTorch) │  
 │ │  
 └────────────────┘  
 │  
 ▼  
 ┌────────────────┐  
 │ │  
 │ Agent System │  
 │ │  
 └────────────────┘

┌────────────────┐ ┌─────────────────┐ ┌────────────────┐  
│ │ │ │ │ │  
│ TypeScript │─────▶ TypeScript MCP │─────▶ Python MCP │  
│ Application │◀───── Client SDK │◀───── Server │  
│ │ │ │ │ │  
└────────────────┘ └─────────────────┘ └────────────────┘  
 │  
 ▼  
 ┌────────────────┐  
 │ │  
 │ ML Models │  
 │ (TF/PyTorch) │  
 │ │  
 └────────────────┘  
 │  
 ▼  
 ┌────────────────┐  
 │ │  
 │ Agent System │  
 │ │  
 └────────────────┘

#### Integration Patterns

The MCP architecture supports two distinct integration patterns:

##### Same-Package Integration

When component implementation and MCP adapter are in the same package:  
- Full local fallback support  
- Transparent switching between MCP and local  
- Consistent API regardless of implementation

##### Cross-Package Integration

When component implementation and MCP adapter are in different packages:  
- Uses MCP as a communication bridge  
- Respects package boundaries  
- No direct cross-package dependencies

### MCP Server

The Python-based MCP Server is the core of the system, responsible for model management, inference, and service coordination.

#### Components That Use MCP

The following ML-intensive components should be moved to MCP server architecture:

1. Firecrawl (Already implemented)
2. Web crawling operations
3. HTML content extraction and processing
4. Vector Search Operations
5. Semantic similarity operations
6. Embedding model inference
7. Batch processing of search queries
8. GPU-accelerated vector operations
9. OCR Processing
10. Document text extraction
11. Layout analysis
12. Handwriting recognition
13. Form field identification
14. Multiple specialized OCR models
15. Image Analysis & Material Recognition
16. Feature extraction from images
17. Material classification
18. Property detection
19. Quality assessment
20. Vision model inference
21. Agent LLM Inference
22. Large language model operations
23. Token batching
24. Model version management
25. Streaming capabilities
26. ML Training Pipeline
27. Transfer learning operations
28. Hyperparameter optimization
29. Distributed training
30. Progress tracking and reporting

HTML content extraction and processing

Vector Search Operations

GPU-accelerated vector operations

OCR Processing

Multiple specialized OCR models

Image Analysis & Material Recognition

Vision model inference

Agent LLM Inference

Streaming capabilities

ML Training Pipeline

#### Server API Endpoints

The MCP Server exposes the following REST API endpoints:

Table content:

Endpoint | Method | Description

/ | GET | Get server information

/health | GET | Health check endpoint

/api/v1/models | GET | List available models

/api/v1/models/{model\_id} | GET | Get model information

/api/v1/models/{model\_id}/context | GET | Get model context

/api/v1/models/{model\_id}/context | PUT | Update model context

/api/v1/recognize | POST | Recognize materials in an image

/api/v1/agent/message | POST | Send a message to the agent

/api/v1/agent/messages | GET | Get messages from the agent queue

/

/health

/api/v1/models

/api/v1/models/{model\_id}

/api/v1/models/{model\_id}/context

/api/v1/models/{model\_id}/context

/api/v1/recognize

/api/v1/agent/message

/api/v1/agent/messages

#### Advanced Features

The MCP server implementation includes several advanced capabilities:

##### Authentication and Security

* Token-based authentication
* Automatic token management and renewal
* Environment-based configuration
* Secure logging with token masking

##### Performance Metrics

* Request counts, latency, and errors
* Component-specific metrics collection
* Configurable sampling rates
* Integration with monitoring systems

##### Batch Processing

* Automatic batching of similar operations
* Configurable batch sizes and timing
* Improved hardware utilization
* Optimized for GPU/TPU acceleration

### MCP Client SDK

The TypeScript client SDK provides a type-safe interface for interacting with the MCP server from Node.js applications.

#### Installation

Note: Installation instructions for the MCP Client have been moved to the Deployment Guide.

#### Basic Usage

import { MCPClient } from '@kai/mcp-client';  
  
// Create a client instance  
const mcpClient = new MCPClient('http://localhost:8000');  
  
// Check server health  
const health = await mcpClient.checkHealth();  
console.log(`Server status: ${health.status}`);  
  
// List available models  
const models = await mcpClient.listModels();  
console.log(`Available models: ${models.map(m => m.name).join(', ')}`);  
  
// Material recognition  
const result = await mcpClient.recognizeMaterial('path/to/image.jpg', {  
 modelType: 'hybrid',  
 confidenceThreshold: 0.7,  
 maxResults: 10  
});  
  
console.log(`Found ${result.matches.length} matching materials`);  
console.log(`Top match: ${result.matches[0].materialId} (${result.matches[0].confidence.toFixed(2)})`);

import { MCPClient } from '@kai/mcp-client';  
  
// Create a client instance  
const mcpClient = new MCPClient('http://localhost:8000');  
  
// Check server health  
const health = await mcpClient.checkHealth();  
console.log(`Server status: ${health.status}`);  
  
// List available models  
const models = await mcpClient.listModels();  
console.log(`Available models: ${models.map(m => m.name).join(', ')}`);  
  
// Material recognition  
const result = await mcpClient.recognizeMaterial('path/to/image.jpg', {  
 modelType: 'hybrid',  
 confidenceThreshold: 0.7,  
 maxResults: 10  
});  
  
console.log(`Found ${result.matches.length} matching materials`);  
console.log(`Top match: ${result.matches[0].materialId} (${result.matches[0].confidence.toFixed(2)})`);

#### API Reference

##### Client Initialization

const mcpClient = new MCPClient(baseUrl: string);

const mcpClient = new MCPClient(baseUrl: string);

##### Server Information

// Get server information  
const info = await mcpClient.getServerInfo();  
  
// Check server health  
const health = await mcpClient.checkHealth();

// Get server information  
const info = await mcpClient.getServerInfo();  
  
// Check server health  
const health = await mcpClient.checkHealth();

##### Model Management

// List available models  
const models = await mcpClient.listModels();  
  
// Get specific model information  
const modelInfo = await mcpClient.getModelInfo('material-hybrid');  
  
// Get model context  
const context = await mcpClient.getModelContext('material-hybrid');  
  
// Update model context  
await mcpClient.updateModelContext('material-hybrid', {  
 model\_id: 'material-hybrid',  
 version: '1.0',  
 parameters: {  
 threshold: 0.6,  
 max\_results: 10  
 }  
});

// List available models  
const models = await mcpClient.listModels();  
  
// Get specific model information  
const modelInfo = await mcpClient.getModelInfo('material-hybrid');  
  
// Get model context  
const context = await mcpClient.getModelContext('material-hybrid');  
  
// Update model context  
await mcpClient.updateModelContext('material-hybrid', {  
 model\_id: 'material-hybrid',  
 version: '1.0',  
 parameters: {  
 threshold: 0.6,  
 max\_results: 10  
 }  
});

##### Inference

// Recognize materials in an image file  
const result = await mcpClient.recognizeMaterial(  
 'path/to/image.jpg',  
 {  
 modelType: 'hybrid',  
 confidenceThreshold: 0.7,  
 maxResults: 5,  
 includeFeatures: true  
 }  
);  
  
// Recognize materials from an image buffer  
const imageBuffer = fs.readFileSync('path/to/image.jpg');  
const result = await mcpClient.recognizeMaterialFromBuffer(  
 imageBuffer,  
 'image/jpeg',  
 {  
 modelType: 'hybrid',  
 confidenceThreshold: 0.7,  
 maxResults: 5  
 }  
);

// Recognize materials in an image file  
const result = await mcpClient.recognizeMaterial(  
 'path/to/image.jpg',  
 {  
 modelType: 'hybrid',  
 confidenceThreshold: 0.7,  
 maxResults: 5,  
 includeFeatures: true  
 }  
);  
  
// Recognize materials from an image buffer  
const imageBuffer = fs.readFileSync('path/to/image.jpg');  
const result = await mcpClient.recognizeMaterialFromBuffer(  
 imageBuffer,  
 'image/jpeg',  
 {  
 modelType: 'hybrid',  
 confidenceThreshold: 0.7,  
 maxResults: 5  
 }  
);

#### Error Handling

The client provides proper error handling for various situations:

try {  
 const result = await mcpClient.recognizeMaterial('path/to/image.jpg');  
 // Process result  
} catch (error) {  
 if (error.response) {  
 // Server responded with an error  
 console.error(`Server error: ${error.response.status} - ${error.response.data.message}`);  
 } else if (error.request) {  
 // Request was made but no response was received  
 console.error('No response from server, it might be down');  
 } else {  
 // Error in setting up the request  
 console.error(`Request error: ${error.message}`);  
 }  
}

try {  
 const result = await mcpClient.recognizeMaterial('path/to/image.jpg');  
 // Process result  
} catch (error) {  
 if (error.response) {  
 // Server responded with an error  
 console.error(`Server error: ${error.response.status} - ${error.response.data.message}`);  
 } else if (error.request) {  
 // Request was made but no response was received  
 console.error('No response from server, it might be down');  
 } else {  
 // Error in setting up the request  
 console.error(`Request error: ${error.message}`);  
 }  
}

#### Configuration

Configure the MCP client using environment variables:

MCP\_SERVER\_URL=http://localhost:8000 # MCP server URL  
USE\_MCP\_SERVER=true # Enable MCP integration  
MCP\_HEALTH\_CHECK\_TIMEOUT=5000 # Health check timeout (ms)

MCP\_SERVER\_URL=http://localhost:8000 # MCP server URL  
USE\_MCP\_SERVER=true # Enable MCP integration  
MCP\_HEALTH\_CHECK\_TIMEOUT=5000 # Health check timeout (ms)

### API Integrations

The MCP system serves as a centralized gateway for external API calls, providing a unified interface for various services while handling credit tracking, authentication, and error handling.

#### High-Priority APIs

##### AI/ML Services

* Text Generation: Uses the llm/completion endpoint
* OpenAI and Anthropic integration
* Prompt management
* Token optimization
* Text Embedding: Uses the llm/embedding endpoint
* Vector generation for semantic search
* Multi-model support
* Image Generation: Uses the image/generation endpoint
* DALL-E and Stable Diffusion support
* Style control and parameter optimization
* Image Analysis: Uses the recognize endpoint
* Material recognition
* Feature extraction
* Classification and segmentation

llm/completion

llm/embedding

image/generation

recognize

##### 3D Model Generation

* Text-to-3D: Uses the 3d/text-to-3d endpoint
* Generate 3D models from text descriptions
* Parameter control for quality vs. speed
* Image-to-3D: Uses the 3d/reconstruct endpoint
* Photogrammetry and image-based reconstruction
* Multi-view synthesis
* Room Layout: Uses the 3d/room-layout endpoint
* Floor plan inference
* Spatial relationship modeling

3d/text-to-3d

3d/reconstruct

3d/room-layout

##### Vector Database Operations

* Vector Search: Uses the vector/search endpoint
* Semantic similarity search
* Hybrid filtering with metadata
* Vector Indexing: Uses the vector/index endpoint
* Index creation and management
* Optimization for different vector types

vector/search

vector/index

#### Medium-Priority APIs

##### Content Processing APIs

* PDF Processing: Uses the content/process-pdf endpoint
* PDF parsing and text extraction
* Image extraction and processing
* Structure recognition
* OCR Processing: Uses the content/ocr endpoint
* Text recognition from images
* Layout analysis
* Specialized engines for different text types
* Data Extraction: Uses the content/extract endpoint
* Structured data extraction
* Schema inference
* Table recognition

content/process-pdf

content/ocr

content/extract

##### Analytics and Telemetry Services

* Analytics Event Processing: Uses the analytics/event endpoint
* Event tracking and storage
* User activity analysis
* Analytics Querying: Uses the analytics/query endpoint
* Data retrieval with filtering
* Aggregation and time series analysis
* Performance Monitoring: Uses the analytics/performance endpoint
* System metrics collection
* Performance anomaly detection

analytics/event

analytics/query

analytics/performance

##### Search Services

* External Search: Uses the search/external endpoint
* Integration with third-party search engines
* Normalization of result formats
* Specialized Search: Uses the search/specialized endpoint
* Domain-specific search capabilities
* Enhanced ranking for specific verticals

search/external

search/specialized

### Credit System Integration

The MCP integration includes credit tracking for all API operations:

#### Credit Checking

Before making an API call, the system checks if the user has enough credits:

// Check if user has enough credits  
const hasEnoughCredits = await creditService.hasEnoughCreditsForService(  
 userId,  
 MCPServiceKey.PDF\_PROCESSING,  
 5 // Estimate 5 credits for PDF processing  
);  
  
if (!hasEnoughCredits) {  
 throw new Error('Insufficient credits');  
}

// Check if user has enough credits  
const hasEnoughCredits = await creditService.hasEnoughCreditsForService(  
 userId,  
 MCPServiceKey.PDF\_PROCESSING,  
 5 // Estimate 5 credits for PDF processing  
);  
  
if (!hasEnoughCredits) {  
 throw new Error('Insufficient credits');  
}

#### Credit Usage Tracking

After a successful API call, the system tracks the credit usage:

// Track credit usage  
await creditService.useServiceCredits(  
 userId,  
 MCPServiceKey.PDF\_PROCESSING,  
 5,  
 'PDF processing',  
 {  
 catalogId,  
 fileName: path.basename(filePath),  
 pageCount: mcpResult.totalPages  
 }  
);

// Track credit usage  
await creditService.useServiceCredits(  
 userId,  
 MCPServiceKey.PDF\_PROCESSING,  
 5,  
 'PDF processing',  
 {  
 catalogId,  
 fileName: path.basename(filePath),  
 pageCount: mcpResult.totalPages  
 }  
);

#### Service Keys

Each service has a unique key for credit tracking. Keys follow the format: category.service, for example:  
- openai.text-generation  
- content.pdf-processing  
- vector.search

category.service

openai.text-generation

content.pdf-processing

vector.search

#### Credit Costs

Different operations have different credit costs:

Table content:

Operation | Credit Cost

Text Generation | 1 credit per 1,000 tokens

Text Embedding | 1 credit per 1,000 tokens

Image Generation | 1 credit per image

Image Analysis | 1 credit per image

Text-to-3D | 5 credits per model

Image-to-3D | 10 credits per model

Room Layout | 3 credits per layout

Vector Search | 1 credit per search

Vector Indexing | 1 credit per document

PDF Processing | 5 credits per PDF

OCR Processing | 1 credit per image

Analytics Event | 1 credit per event

Analytics Query | 1 credit per query

Analytics Trends | 2 credits per trends query

Analytics Statistics | 2 credits per stats query

### Agent Integration

The MCP system is designed to work with AI agents by providing:

1. Message Queue: A pub/sub system for agent communication
2. Context Management: Storage and retrieval of contextual information
3. Standardized Protocols: Following the Model Context Protocol for consistent interactions

#### Agent Communication

The MCP system facilitates communication with AI agents. This enables:

1. Contextual Model Inference: Agents can provide additional context for model inference
2. Feedback Loops: Agents can provide feedback on model results to improve future inference
3. Complex Decision Flows: Multi-step inference with agent guidance
4. Explanation Generation: Agents can explain model decisions in natural language

##### Sending Messages to Agent

// Send a message to the agent  
await mcpClient.sendAgentMessage({  
 message\_type: 'inference\_context',  
 content: {  
 user\_query: 'Find tiles similar to my kitchen backsplash',  
 preferences: ['matte finish', 'neutral colors'],  
 previous\_interaction\_id: '12345'  
 },  
 timestamp: Date.now() / 1000  
});

// Send a message to the agent  
await mcpClient.sendAgentMessage({  
 message\_type: 'inference\_context',  
 content: {  
 user\_query: 'Find tiles similar to my kitchen backsplash',  
 preferences: ['matte finish', 'neutral colors'],  
 previous\_interaction\_id: '12345'  
 },  
 timestamp: Date.now() / 1000  
});

# Python  
await agent\_queue.put({  
 "type": "recognition\_event",  
 "content": {"materials": ["ceramic-tile", "porcelain-tile"]},  
 "timestamp": time.time()  
})

# Python  
await agent\_queue.put({  
 "type": "recognition\_event",  
 "content": {"materials": ["ceramic-tile", "porcelain-tile"]},  
 "timestamp": time.time()  
})

##### Receiving Messages from Agent

// Get messages from the agent  
const agentMessages = await mcpClient.getAgentMessages(1.0); // Wait up to 1 second  
if (agentMessages && agentMessages.count > 0) {  
 console.log(`Received ${agentMessages.count} messages from agent`);  
 for (const message of agentMessages.messages) {  
 console.log(`Agent message: ${message.message\_type}`);  
 console.log(message.content);  
 }  
}

// Get messages from the agent  
const agentMessages = await mcpClient.getAgentMessages(1.0); // Wait up to 1 second  
if (agentMessages && agentMessages.count > 0) {  
 console.log(`Received ${agentMessages.count} messages from agent`);  
 for (const message of agentMessages.messages) {  
 console.log(`Agent message: ${message.message\_type}`);  
 console.log(message.content);  
 }  
}

#### Direct Agent Communication Features

1. Direct Agent Communication
2. Agents can directly query model capabilities
3. Agents can provide additional context for model inference
4. Models can request clarification from agents for ambiguous inputs
5. Knowledge Enhancement
6. Agents can augment model outputs with additional information
7. Models can provide confidence scores that agents use for decision-making
8. Joint inference between multiple models coordinated by agents
9. Feedback Loops
10. Agents capture user feedback for model improvement
11. Models track inference patterns for agent learning
12. Continuous improvement through shared learning

Models can request clarification from agents for ambiguous inputs

Knowledge Enhancement

Joint inference between multiple models coordinated by agents

Feedback Loops

### Implementation Details

#### MCP Client Service

The mcpClientService.ts file provides a unified interface for all MCP operations:

mcpClientService.ts

// Example: Generate text using MCP  
const result = await mcpClientService.generateText(  
 userId,  
 prompt,  
 {  
 model: 'gpt-4',  
 maxTokens: 500,  
 temperature: 0.7  
 }  
);

// Example: Generate text using MCP  
const result = await mcpClientService.generateText(  
 userId,  
 prompt,  
 {  
 model: 'gpt-4',  
 maxTokens: 500,  
 temperature: 0.7  
 }  
);

#### Controller Integration

Controllers use the MCP client service with fallback to direct implementations:

// Check if MCP is available  
const mcpAvailable = await mcpClientService.isMCPAvailable();  
  
if (mcpAvailable && userId) {  
 try {  
 // Use MCP for the operation  
 const mcpResult = await mcpClientService.generateText(...);  
 return mcpResult;  
 } catch (mcpError) {  
 // Handle insufficient credits error  
 if (mcpError.message === 'Insufficient credits') {  
 return res.status(402).json({  
 error: 'Insufficient credits',  
 message: 'You do not have enough credits to perform this action.'  
 });  
 }  
  
 // Fall back to direct implementation for other errors  
 logger.warn(`MCP operation failed, falling back: ${mcpError.message}`);  
 }  
}  
  
// Fall back to direct implementation  
const result = await directImplementation(...);

// Check if MCP is available  
const mcpAvailable = await mcpClientService.isMCPAvailable();  
  
if (mcpAvailable && userId) {  
 try {  
 // Use MCP for the operation  
 const mcpResult = await mcpClientService.generateText(...);  
 return mcpResult;  
 } catch (mcpError) {  
 // Handle insufficient credits error  
 if (mcpError.message === 'Insufficient credits') {  
 return res.status(402).json({  
 error: 'Insufficient credits',  
 message: 'You do not have enough credits to perform this action.'  
 });  
 }  
  
 // Fall back to direct implementation for other errors  
 logger.warn(`MCP operation failed, falling back: ${mcpError.message}`);  
 }  
}  
  
// Fall back to direct implementation  
const result = await directImplementation(...);

#### PDF Processing Integration

The PDF processing service has been enhanced to use MCP for PDF parsing, text extraction, and OCR operations:

// Check if MCP is available and user ID is provided  
const mcpAvailable = await isMCPAvailable();  
  
if (mcpAvailable && userId) {  
 try {  
 // Check if user has enough credits  
 const hasEnoughCredits = await creditService.hasEnoughCreditsForService(  
 userId,  
 MCPServiceKey.PDF\_PROCESSING,  
 5 // Estimate 5 credits for PDF processing  
 );  
  
 if (!hasEnoughCredits) {  
 throw new Error('Insufficient credits');  
 }  
  
 // Use MCP for PDF processing  
 const mcpResult = await mcpClientService.processPdf(  
 userId,  
 filePath,  
 {  
 extractImages: processingOptions.extractImages,  
 extractText: processingOptions.extractText,  
 associateTextWithImages: processingOptions.associateTextWithImages,  
 outputDir: tempDir  
 }  
 );  
  
 // Track credit usage  
 await creditService.useServiceCredits(  
 userId,  
 MCPServiceKey.PDF\_PROCESSING,  
 5,  
 'PDF processing',  
 {  
 catalogId,  
 fileName: path.basename(filePath),  
 pageCount: mcpResult.totalPages  
 }  
 );  
  
 // Return results  
 return {  
 catalogId,  
 totalPages: mcpResult.totalPages,  
 processedPages: mcpResult.processedPages,  
 materials: mcpResult.materials || [],  
 errors: mcpResult.errors || []  
 };  
 } catch (mcpError) {  
 // Handle errors and fallback to direct implementation  
 }  
}

// Check if MCP is available and user ID is provided  
const mcpAvailable = await isMCPAvailable();  
  
if (mcpAvailable && userId) {  
 try {  
 // Check if user has enough credits  
 const hasEnoughCredits = await creditService.hasEnoughCreditsForService(  
 userId,  
 MCPServiceKey.PDF\_PROCESSING,  
 5 // Estimate 5 credits for PDF processing  
 );  
  
 if (!hasEnoughCredits) {  
 throw new Error('Insufficient credits');  
 }  
  
 // Use MCP for PDF processing  
 const mcpResult = await mcpClientService.processPdf(  
 userId,  
 filePath,  
 {  
 extractImages: processingOptions.extractImages,  
 extractText: processingOptions.extractText,  
 associateTextWithImages: processingOptions.associateTextWithImages,  
 outputDir: tempDir  
 }  
 );  
  
 // Track credit usage  
 await creditService.useServiceCredits(  
 userId,  
 MCPServiceKey.PDF\_PROCESSING,  
 5,  
 'PDF processing',  
 {  
 catalogId,  
 fileName: path.basename(filePath),  
 pageCount: mcpResult.totalPages  
 }  
 );  
  
 // Return results  
 return {  
 catalogId,  
 totalPages: mcpResult.totalPages,  
 processedPages: mcpResult.processedPages,  
 materials: mcpResult.materials || [],  
 errors: mcpResult.errors || []  
 };  
 } catch (mcpError) {  
 // Handle errors and fallback to direct implementation  
 }  
}

#### Batch Processing

For batch operations, the system checks credit availability for the entire batch before processing:

// Check if user has enough credits for all images  
const hasEnoughCredits = await creditService.hasEnoughCreditsForService(  
 userId,  
 MCPServiceKey.OCR\_PROCESSING,  
 imagePaths.length // 1 credit per image  
);  
  
if (!hasEnoughCredits) {  
 throw new Error('Insufficient credits');  
}  
  
// Process images in batches  
for (let i = 0; i < imagePaths.length; i += concurrency) {  
 const batch = imagePaths.slice(i, i + concurrency);  
 const batchPromises = batch.map(imagePath =>  
 mcpClientService.performOcr(userId, imagePath, options)  
 .then(mcpResult => {  
 // Process result  
  
 // Track credit usage for each image  
 return creditService.useServiceCredits(  
 userId,  
 MCPServiceKey.OCR\_PROCESSING,  
 1,  
 `${MCPServiceKey.OCR\_PROCESSING} API usage`,  
 {  
 endpoint: 'content/ocr',  
 imagePath  
 }  
 );  
 })  
 );  
  
 await Promise.all(batchPromises);  
}

// Check if user has enough credits for all images  
const hasEnoughCredits = await creditService.hasEnoughCreditsForService(  
 userId,  
 MCPServiceKey.OCR\_PROCESSING,  
 imagePaths.length // 1 credit per image  
);  
  
if (!hasEnoughCredits) {  
 throw new Error('Insufficient credits');  
}  
  
// Process images in batches  
for (let i = 0; i < imagePaths.length; i += concurrency) {  
 const batch = imagePaths.slice(i, i + concurrency);  
 const batchPromises = batch.map(imagePath =>  
 mcpClientService.performOcr(userId, imagePath, options)  
 .then(mcpResult => {  
 // Process result  
  
 // Track credit usage for each image  
 return creditService.useServiceCredits(  
 userId,  
 MCPServiceKey.OCR\_PROCESSING,  
 1,  
 `${MCPServiceKey.OCR\_PROCESSING} API usage`,  
 {  
 endpoint: 'content/ocr',  
 imagePath  
 }  
 );  
 })  
 );  
  
 await Promise.all(batchPromises);  
}

#### Integration with ML Package

The ML package includes an integration module that transparently uses the MCP server when available:

// In packages/ml/src/index.ts  
import { recognizeMaterial as originalRecognizeMaterial } from './direct-implementation';  
import { withMCPFallback, recognizeMaterialWithMCP } from './mcp-integration';  
  
export async function recognizeMaterial(imagePath, options) {  
 return withMCPFallback(  
 recognizeMaterialWithMCP,  
 originalRecognizeMaterial,  
 imagePath,  
 options  
 );  
}

// In packages/ml/src/index.ts  
import { recognizeMaterial as originalRecognizeMaterial } from './direct-implementation';  
import { withMCPFallback, recognizeMaterialWithMCP } from './mcp-integration';  
  
export async function recognizeMaterial(imagePath, options) {  
 return withMCPFallback(  
 recognizeMaterialWithMCP,  
 originalRecognizeMaterial,  
 imagePath,  
 options  
 );  
}

### Performance Optimization

The MCP system implements several performance optimizations:

#### Scaling the MCP System

The MCP system is designed to scale both horizontally and vertically:

1. Vertical Scaling
2. Increase resources for the MCP server
3. Optimize model loading and memory usage
4. Efficient GPU utilization
5. Horizontal Scaling
6. Deploy multiple MCP server instances
7. Load balancing across instances
8. Specialized instances for different model types

Efficient GPU utilization

Horizontal Scaling

#### Memory Management

1. Model Caching: Models are loaded once and kept in memory
2. Batch Processing: Requests can be batched for more efficient processing
3. Async Processing: Non-blocking I/O for higher throughput
4. Resource Monitoring: Monitoring of memory and CPU usage

#### Performance Metrics

The system tracks several performance metrics:

1. Latency: End-to-end response time for different operations
2. Throughput: Requests processed per second
3. Resource Usage: CPU, memory, and GPU utilization
4. Cache Performance: Hit rate and cache efficiency
5. Error Rates: Failed requests and timeout frequency

### Troubleshooting

#### Common Issues

##### Connection Errors

* Check if the MCP server is running
* Verify network connectivity and firewall settings
* Check if the port is correctly exposed
* Validate authentication credentials

##### Model Loading Failures

* Ensure model files exist in the model directory
* Check for sufficient memory for loading models
* Verify GPU availability if using GPU-accelerated models
* Check file permissions for model files

##### Credit-Related Errors

* Verify user has sufficient credits
* Check credit tracking configuration
* Ensure credit service is properly configured
* Validate credit cost settings for operations

##### Slow Performance

* Check if GPU is being utilized (if available)
* Monitor memory usage for potential leaks
* Consider increasing server resources
* Review batch size settings for optimization

#### Error Handling and Fallbacks

All MCP integrations include robust error handling and fallback mechanisms:

1. Credit Insufficiency
2. When a user doesn't have enough credits, the system returns a 402 (Payment Required) error
3. The error includes a clear message about purchasing more credits
4. MCP Unavailability
5. When the MCP server is unavailable, the system falls back to direct implementations
6. Logs are generated to track fallback occurrences
7. Service-Specific Errors
8. Each service handles specific error cases appropriately
9. Detailed error information is logged for debugging

The error includes a clear message about purchasing more credits

MCP Unavailability

Logs are generated to track fallback occurrences

Service-Specific Errors

#### Logs

The MCP server logs are written to standard output when running in Docker. You can view them with:

docker logs kai-mcp-server

docker logs kai-mcp-server

### Future Enhancements

#### Dynamic Credit Calculation

* Calculate credit costs based on actual usage metrics
* Adjust costs based on service provider pricing changes
* Implement credit prediction for complex operations

#### Caching and Optimization

* Implement result caching for frequently used operations
* Optimize request batching for better performance
* Add model quantization for improved efficiency

#### Additional Services

* Add support for more third-party services
* Implement adapters for different service providers
* Expand to new ML model types and frameworks

#### Monitoring and Analytics

* Add detailed logging for all MCP operations
* Implement usage analytics and reporting
* Add predictive maintenance for resource management

#### Advanced Features

1. Distributed Deployment: Support for multiple MCP servers with load balancing
2. Model Version Control: Advanced management of model versions and rollbacks
3. A/B Testing: Support for comparing performance between model versions
4. Advanced Metrics: Enhanced performance and accuracy monitoring
5. Enhanced Agent Integration: Deeper integration with future AI agent capabilities

#### Specific Product Enhancements

1. Additional Content Processing Services
2. Add support for video processing and analysis
3. Implement document comparison and diff generation
4. Integrate with more specialized OCR engines
5. Enhanced Analytics
6. Implement real-time analytics processing
7. Add support for custom analytics pipelines
8. Develop predictive analytics capabilities
9. Advanced Search Features
10. Implement multi-modal search (text + image)
11. Add support for conversational search
12. Develop domain-specific search optimizations

Integrate with more specialized OCR engines

Enhanced Analytics

Develop predictive analytics capabilities

Advanced Search Features

### Deployment

For detailed deployment instructions, see the Deployment and Development Guide.

# Messaging Service

Source: readme/messaging-service.md

---

## Messaging Service

The Messaging Service provides a centralized system for sending notifications through various channels including email, SMS, and webhooks. It supports template-based message generation and user notification preferences.

### Features

* Multi-Channel Notifications: Send notifications through email, SMS, webhooks, and in-app channels
* Template System: Use Handlebars templates for consistent message formatting
* User Preferences: Respect user notification preferences
* Event-Based Notifications: Trigger notifications based on system events
* Webhook Integration: Allow external systems to receive notifications via webhooks
* Delivery Tracking: Log and track notification delivery status

### Architecture

The Messaging Service is built with a modular architecture:

Messaging Service  
├── Core Service (notificationService)  
├── Providers  
│ ├── Email Provider  
│ ├── SMS Provider  
│ └── Webhook Provider  
├── Templates  
│ └── Template Service  
└── Event System  
 └── Event Notification Service

Messaging Service  
├── Core Service (notificationService)  
├── Providers  
│ ├── Email Provider  
│ ├── SMS Provider  
│ └── Webhook Provider  
├── Templates  
│ └── Template Service  
└── Event System  
 └── Event Notification Service

### Usage Examples

#### Sending an Email

import { notificationService } from '../services/messaging/notificationService';  
  
// Send a simple email  
await notificationService.sendEmail({  
 to: 'user@example.com',  
 subject: 'Welcome to KAI',  
 text: 'Thank you for joining our platform!',  
 html: '<p>Thank you for joining our platform!</p>'  
});  
  
// Send an email with user tracking  
await notificationService.sendEmail({  
 to: 'user@example.com',  
 subject: 'Welcome to KAI',  
 text: 'Thank you for joining our platform!',  
 html: '<p>Thank you for joining our platform!</p>',  
 userId: 'user-123',  
 eventType: 'user.registered'  
});

import { notificationService } from '../services/messaging/notificationService';  
  
// Send a simple email  
await notificationService.sendEmail({  
 to: 'user@example.com',  
 subject: 'Welcome to KAI',  
 text: 'Thank you for joining our platform!',  
 html: '<p>Thank you for joining our platform!</p>'  
});  
  
// Send an email with user tracking  
await notificationService.sendEmail({  
 to: 'user@example.com',  
 subject: 'Welcome to KAI',  
 text: 'Thank you for joining our platform!',  
 html: '<p>Thank you for joining our platform!</p>',  
 userId: 'user-123',  
 eventType: 'user.registered'  
});

#### Sending an SMS

import { notificationService } from '../services/messaging/notificationService';  
  
// Send an SMS  
await notificationService.sendSMS({  
 to: '+1234567890',  
 message: 'Your verification code is: 123456'  
});

import { notificationService } from '../services/messaging/notificationService';  
  
// Send an SMS  
await notificationService.sendSMS({  
 to: '+1234567890',  
 message: 'Your verification code is: 123456'  
});

#### Sending a Webhook Notification

import { notificationService } from '../services/messaging/notificationService';  
  
// Send a webhook notification  
await notificationService.sendWebhook({  
 url: 'https://example.com/webhook',  
 payload: {  
 event: 'order.created',  
 data: {  
 orderId: 'order-123',  
 amount: 99.99  
 }  
 }  
});

import { notificationService } from '../services/messaging/notificationService';  
  
// Send a webhook notification  
await notificationService.sendWebhook({  
 url: 'https://example.com/webhook',  
 payload: {  
 event: 'order.created',  
 data: {  
 orderId: 'order-123',  
 amount: 99.99  
 }  
 }  
});

#### Sending an In-App Notification

import { notificationService } from '../services/messaging/notificationService';  
  
// Send an in-app notification  
await notificationService.sendInAppNotification({  
 userId: 'user-123',  
 title: 'New Message',  
 message: 'You have a new message from Admin',  
 type: 'info',  
 actionUrl: '/messages/123'  
});

import { notificationService } from '../services/messaging/notificationService';  
  
// Send an in-app notification  
await notificationService.sendInAppNotification({  
 userId: 'user-123',  
 title: 'New Message',  
 message: 'You have a new message from Admin',  
 type: 'info',  
 actionUrl: '/messages/123'  
});

#### Using the Event-Based System

import { eventNotificationService, EventType } from '../services/messaging/eventNotificationService';  
  
// Trigger an event that will send notifications based on configured rules  
await eventNotificationService.processEvent({  
 eventType: EventType.SUBSCRIPTION\_PAYMENT\_FAILED,  
 userId: 'user-123',  
 data: {  
 subscriptionId: 'sub-123',  
 failureReason: 'insufficient\_funds'  
 }  
});

import { eventNotificationService, EventType } from '../services/messaging/eventNotificationService';  
  
// Trigger an event that will send notifications based on configured rules  
await eventNotificationService.processEvent({  
 eventType: EventType.SUBSCRIPTION\_PAYMENT\_FAILED,  
 userId: 'user-123',  
 data: {  
 subscriptionId: 'sub-123',  
 failureReason: 'insufficient\_funds'  
 }  
});

### Configuration

The messaging service can be configured through environment variables:

#### Email Configuration

EMAIL\_SERVICE=smtp  
EMAIL\_HOST=smtp.example.com  
EMAIL\_PORT=587  
EMAIL\_SECURE=false  
EMAIL\_USER=username  
EMAIL\_PASSWORD=password  
EMAIL\_FROM=noreply@example.com

EMAIL\_SERVICE=smtp  
EMAIL\_HOST=smtp.example.com  
EMAIL\_PORT=587  
EMAIL\_SECURE=false  
EMAIL\_USER=username  
EMAIL\_PASSWORD=password  
EMAIL\_FROM=noreply@example.com

#### SMS Configuration

SMS\_PROVIDER=twilio  
TWILIO\_ACCOUNT\_SID=your\_account\_sid  
TWILIO\_AUTH\_TOKEN=your\_auth\_token  
TWILIO\_PHONE\_NUMBER=+1234567890

SMS\_PROVIDER=twilio  
TWILIO\_ACCOUNT\_SID=your\_account\_sid  
TWILIO\_AUTH\_TOKEN=your\_auth\_token  
TWILIO\_PHONE\_NUMBER=+1234567890

#### Webhook Configuration

WEBHOOK\_TIMEOUT=5000  
WEBHOOK\_MAX\_RETRIES=3  
WEBHOOK\_RETRY\_DELAY=1000

WEBHOOK\_TIMEOUT=5000  
WEBHOOK\_MAX\_RETRIES=3  
WEBHOOK\_RETRY\_DELAY=1000

### Database Schema

The messaging service uses the following database tables:

* notification\_logs: Logs of all sent notifications
* user\_notification\_preferences: User preferences for notifications
* notifications: In-app notifications
* message\_templates: Templates for notifications
* notification\_rules: Rules for event-based notifications
* webhook\_configurations: Webhook integration configurations
* webhook\_delivery\_logs: Logs of webhook deliveries
* event\_logs: Logs of system events

notification\_logs

user\_notification\_preferences

notifications

message\_templates

notification\_rules

webhook\_configurations

webhook\_delivery\_logs

event\_logs

### API Endpoints

#### Notification Preferences

* GET /api/notifications/preferences: Get notification preferences
* PUT /api/notifications/preferences: Update notification preferences

GET /api/notifications/preferences

PUT /api/notifications/preferences

#### In-App Notifications

* GET /api/notifications: Get in-app notifications
* POST /api/notifications/mark-as-read: Mark notifications as read
* POST /api/notifications/mark-all-as-read: Mark all notifications as read
* POST /api/notifications/delete: Delete notifications
* GET /api/notifications/unread-count: Get unread notification count
* POST /api/notifications/test: Send a test notification

GET /api/notifications

POST /api/notifications/mark-as-read

POST /api/notifications/mark-all-as-read

POST /api/notifications/delete

GET /api/notifications/unread-count

POST /api/notifications/test

#### Webhook Management

* GET /api/webhooks/configurations: Get webhook configurations
* GET /api/webhooks/configurations/:id: Get a webhook configuration
* POST /api/webhooks/configurations: Create a webhook configuration
* PUT /api/webhooks/configurations/:id: Update a webhook configuration
* DELETE /api/webhooks/configurations/:id: Delete a webhook configuration
* GET /api/webhooks/configurations/:id/logs: Get webhook delivery logs
* POST /api/webhooks/configurations/:id/test: Test a webhook configuration
* POST /api/webhooks/configurations/:id/regenerate-secret: Regenerate webhook secret

GET /api/webhooks/configurations

GET /api/webhooks/configurations/:id

POST /api/webhooks/configurations

PUT /api/webhooks/configurations/:id

DELETE /api/webhooks/configurations/:id

GET /api/webhooks/configurations/:id/logs

POST /api/webhooks/configurations/:id/test

POST /api/webhooks/configurations/:id/regenerate-secret

### Security Considerations

* All sensitive data (API keys, passwords) is stored in environment variables
* Webhook payloads are signed with a secret for verification
* Rate limiting is applied to notification endpoints
* Authentication is required for all notification management endpoints
* User notification preferences are protected by row-level security

### Future Enhancements

* Push notification support
* Message scheduling and batching
* A/B testing for notification content
* Advanced analytics for notification engagement
* Support for more notification channels (e.g., Slack, Microsoft Teams)

# Ml Documentation

Source: readme/ml-documentation.md

---

## Kai ML Package Documentation

This comprehensive guide covers all aspects of the Machine Learning (ML) components in the Kai platform, including the ML Package, Model Context Protocol (MCP) Server, OCR enhancements, and Training API.

### Table of Contents

* Overview
* ML Package
* Model Context Protocol (MCP) Server
* OCR Enhancements
* Training API Improvements
* Vector Database Integration
* Deployment and Installation
* Usage Examples
* API Reference
* Performance Considerations
* Troubleshooting

### Overview

The Kai ML system provides sophisticated machine learning capabilities for material recognition, document processing, vector embeddings, and model training. Its key components include:

* ML Package: Core ML functionality for PDF processing, material recognition, and vector embeddings
* Model Context Protocol (MCP) Server: Centralized model management and inference service
* OCR Enhancements: Specialized text extraction for material datasheets and technical documents
* Training API: Advanced capabilities for model training and optimization

### ML Package

The ML Package provides machine learning functionality for the Kai Material Recognition system.

#### Features

* PDF Processing: Extract images and text from PDF catalogs
* Material Recognition: Identify materials in images using a hybrid approach
* Vector Embeddings: Generate vector representations of materials for similarity search
* Model Training: Train and evaluate material recognition models

#### Dataset Organization

For training the material recognition models, you need to organize your dataset in a specific structure:

dataset/  
 ├── tile/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── ...  
 ├── stone/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── ...  
 ├── wood/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── ...  
 └── ...

dataset/  
 ├── tile/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── ...  
 ├── stone/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── ...  
 ├── wood/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── ...  
 └── ...

Each material type should have its own directory containing images of that material. The directory name will be used as the material ID in the recognition results.

### Model Context Protocol (MCP) Server

The Model Context Protocol (MCP) Server is a centralized service that manages machine learning models, their contexts, and provides optimized inference capabilities for the Kai system.

#### What is the MCP Server?

The MCP Server is a dedicated Python service that:

1. Centralizes Model Management: Loads and caches ML models in memory for faster inference
2. Standardizes API Access: Provides a consistent interface regardless of underlying model framework
3. Optimizes Performance: Implements batching, caching, and other optimizations for ML inference
4. Supports Agent Integration: Includes APIs designed for interaction with AI agents
5. Implements the Model Context Protocol: Follows standardized protocols for model context handling

#### Why use an MCP Server?

* Improved Performance: Models stay loaded in memory, eliminating load time between requests
* Resource Efficiency: Multiple services can use the same model instances
* Simplified Model Updates: Models can be updated without restarting the main application
* Framework Abstraction: Hides the complexity of different ML frameworks (TensorFlow, PyTorch)
* Future Agent Integration: Designed to work seamlessly with AI agents

#### Architecture

The MCP implementation uses a hybrid approach with two main components:

##### Python Server (Backend)

The Python-based MCP Server handles the heavy lifting:

* Written in Python using FastAPI
* Loads models directly using TensorFlow/PyTorch/OpenCV
* Manages model contexts and caching
* Exposes REST API endpoints
* Includes agent communication channels
* Designed for Docker deployment

##### TypeScript Client (Frontend)

The TypeScript client SDK integrates with the Node.js application:

* Written in TypeScript
* Provides type-safe interfaces for the MCP Server
* Handles automatic fallback to existing implementation
* Includes connection health monitoring
* Proxies requests to the MCP Server

##### Communication Flow

┌────────────────┐ ┌─────────────────┐ ┌────────────────┐  
│ │ │ │ │ │  
│ TypeScript │─────▶ TypeScript MCP │─────▶ Python MCP │  
│ Application │◀───── Client SDK │◀───── Server │  
│ │ │ │ │ │  
└────────────────┘ └─────────────────┘ └────────────────┘  
 │  
 ▼  
 ┌────────────────┐  
 │ │  
 │ ML Models │  
 │ (TF/PyTorch) │  
 │ │  
 └────────────────┘

┌────────────────┐ ┌─────────────────┐ ┌────────────────┐  
│ │ │ │ │ │  
│ TypeScript │─────▶ TypeScript MCP │─────▶ Python MCP │  
│ Application │◀───── Client SDK │◀───── Server │  
│ │ │ │ │ │  
└────────────────┘ └─────────────────┘ └────────────────┘  
 │  
 ▼  
 ┌────────────────┐  
 │ │  
 │ ML Models │  
 │ (TF/PyTorch) │  
 │ │  
 └────────────────┘

#### API Endpoints

The MCP Server exposes the following REST API endpoints:

Table content:

Endpoint | Method | Description

/ | GET | Get server information

/health | GET | Health check endpoint

/api/v1/models | GET | List available models

/api/v1/models/{model\_id} | GET | Get model information

/api/v1/models/{model\_id}/context | GET | Get model context

/api/v1/models/{model\_id}/context | PUT | Update model context

/api/v1/recognize | POST | Recognize materials in an image

/api/v1/agent/message | POST | Send a message to the agent

/api/v1/agent/messages | GET | Get messages from the agent queue

/

/health

/api/v1/models

/api/v1/models/{model\_id}

/api/v1/models/{model\_id}/context

/api/v1/models/{model\_id}/context

/api/v1/recognize

/api/v1/agent/message

/api/v1/agent/messages

#### TypeScript Client SDK

The client SDK provides a simple interface for interacting with the MCP Server:

import { MCPClient } from '@kai/mcp-client';  
  
// Create client instance  
const client = new MCPClient('http://localhost:8000');  
  
// Recognize materials in an image  
const result = await client.recognizeMaterial('/path/to/image.jpg', {  
 modelType: 'hybrid',  
 confidenceThreshold: 0.7,  
 maxResults: 5,  
 includeFeatures: true  
});  
  
// Get available models  
const models = await client.listModels();  
  
// Send a message to the agent  
await client.sendAgentMessage({  
 message\_type: 'recognition\_completed',  
 content: { materialId: 'tile-123', confidence: 0.95 }  
});

import { MCPClient } from '@kai/mcp-client';  
  
// Create client instance  
const client = new MCPClient('http://localhost:8000');  
  
// Recognize materials in an image  
const result = await client.recognizeMaterial('/path/to/image.jpg', {  
 modelType: 'hybrid',  
 confidenceThreshold: 0.7,  
 maxResults: 5,  
 includeFeatures: true  
});  
  
// Get available models  
const models = await client.listModels();  
  
// Send a message to the agent  
await client.sendAgentMessage({  
 message\_type: 'recognition\_completed',  
 content: { materialId: 'tile-123', confidence: 0.95 }  
});

#### Agent Integration

The MCP Server is designed to work with AI agents by providing:

1. Message Queue: A pub/sub system for agent communication
2. Context Management: Storage and retrieval of contextual information
3. Standardized Protocols: Following the Model Context Protocol for consistent interactions

##### Sending Messages to Agent

// TypeScript  
await mcpClient.sendAgentMessage({  
 message\_type: 'recognition\_event',  
 content: { materials: ['ceramic-tile', 'porcelain-tile'] }  
});

// TypeScript  
await mcpClient.sendAgentMessage({  
 message\_type: 'recognition\_event',  
 content: { materials: ['ceramic-tile', 'porcelain-tile'] }  
});

# Python  
await agent\_queue.put({  
 "type": "recognition\_event",  
 "content": {"materials": ["ceramic-tile", "porcelain-tile"]},  
 "timestamp": time.time()  
})

# Python  
await agent\_queue.put({  
 "type": "recognition\_event",  
 "content": {"materials": ["ceramic-tile", "porcelain-tile"]},  
 "timestamp": time.time()  
})

##### Receiving Messages from Agent

// TypeScript  
const messages = await mcpClient.getAgentMessages(1.0);  
for (const message of messages.messages) {  
 // Process message  
 console.log(`Agent message: ${message.type}`);  
}

// TypeScript  
const messages = await mcpClient.getAgentMessages(1.0);  
for (const message of messages.messages) {  
 // Process message  
 console.log(`Agent message: ${message.type}`);  
}

#### Performance Optimization

The MCP Server implements several performance optimizations:

1. Model Caching: Models are loaded once and kept in memory
2. Batch Processing: Requests can be batched for more efficient processing
3. Async Processing: Non-blocking I/O for higher throughput
4. Resource Monitoring: Monitoring of memory and CPU usage

### OCR Enhancements

The OCR Enhancements system provides specialized text extraction capabilities for material datasheets and technical documents.

#### Key Enhancements

##### 1. Specialized OCR for Material Datasheets

We've implemented custom OCR models and preprocessing techniques specifically optimized for technical specifications in material datasheets.

Key Features:  
- Domain-specific dictionaries for materials (tile, stone, wood, etc.)  
- Region-specific OCR optimization for different parts of datasheets  
- Enhanced recognition of technical symbols, measurements, and specification formats  
- Fine-tuned recognition for product codes, SKUs, and material identifiers

Implementation: specialized\_ocr.py

specialized\_ocr.py

##### 2. Multi-Language Support

Extended language capabilities now support technical documents in multiple languages beyond English.

Key Features:  
- Support for 20+ languages including French, German, Spanish, Italian, Chinese, Japanese  
- Automatic language detection in mixed-language documents  
- Language-specific post-processing rules for technical terms  
- Multi-language dictionary support for domain-specific terminology

Implementation: Integrated within specialized\_ocr.py

specialized\_ocr.py

##### 3. Layout Analysis Improvements

Advanced document structure analysis to better handle complex layouts common in material datasheets.

Key Features:  
- Table detection and extraction with cell-level content recognition  
- Multi-column layout detection and processing  
- Diagram and chart identification with text extraction  
- Structural separation of headings, specifications, and descriptive content

Implementation: layout\_analysis.py

layout\_analysis.py

##### 4. Handwriting Recognition

New capabilities to detect and recognize handwritten annotations commonly found on technical documents.

Key Features:  
- Detection of handwritten regions on printed documents  
- Specialized preprocessing for handwritten text  
- Integration with printed text extraction workflow  
- Confidence scoring for handwritten content

Implementation: handwriting\_recognition.py

handwriting\_recognition.py

##### 5. PDF Form Field Extraction

Automatic identification and extraction of data from structured forms in PDF documents.

Key Features:  
- Detection of form fields (text fields, checkboxes, radio buttons)  
- Label-to-value mapping for form fields  
- Structured data extraction from form-based documents  
- Support for flattened forms where original field structure is not preserved

Implementation: form\_field\_extraction.py

form\_field\_extraction.py

##### 6. OCR Confidence Scoring

Reliability metrics for extracted text to help identify potential errors and uncertain extractions.

Key Features:  
- Multi-factor confidence evaluation (character, word, context-based)  
- Domain-specific confidence boosting for known terms  
- Identification of low-confidence regions requiring manual review  
- Aggregate confidence metrics for entire documents and sections

Implementation: ocr\_confidence\_scoring.py

ocr\_confidence\_scoring.py

##### 7. Post-Processing Rules Engine

Domain-specific correction rules to improve OCR accuracy for technical content.

Key Features:  
- Technical unit standardization (mm, cm, inches, etc.)  
- Specification format normalization  
- Automatic correction of common OCR errors in technical terms  
- Context-aware text verification and correction

Implementation: Integrated within ocr\_confidence\_scoring.py

ocr\_confidence\_scoring.py

##### 8. SVBRDF Material Property Extraction

Advanced material appearance property extraction from single images using Spatially Varying Bidirectional Reflectance Distribution Functions (SVBRDFs).

Key Features:  
- Diffuse color map extraction (albedo)  
- Surface normal map generation (microfacet orientation)  
- Roughness map extraction (surface microsurface detail)  
- Specular reflection and metallic property analysis  
- TensorFlow 2.x compatibility with legacy SVBRDF models

Implementation: svbrdf\_capture\_engine.py and material\_svbrdf\_processor.py

svbrdf\_capture\_engine.py

material\_svbrdf\_processor.py

#### System Architecture

The enhanced OCR system integrates with the existing PDF processing pipeline while introducing new specialized components:

┌─────────────────────┐  
│ PDF Document │  
└──────────┬──────────┘  
 │  
┌──────────▼──────────┐  
│ PDF Image Extraction│  
│ (pdf\_extractor.py) │  
└──────────┬──────────┘  
 │  
┌──────────▼──────────┐ ┌───────────────────────┐  
│ Layout Analysis │◄───┤ Document Structure │  
│ (layout\_analysis.py) │ │ Classification │  
└──────────┬──────────┘ └───────────────────────┘  
 │  
┌──────────▼──────────┐ ┌───────────────────────┐  
│ Region Classification│◄──┤ Form Field Detection │  
│ │ │(form\_field\_extraction)│  
└──────────┬──────────┘ └───────────────────────┘  
 │  
┌──────────▼──────────┐ ┌───────────────────────┐  
│ Specialized OCR │◄───┤ Language Detection │  
│ (specialized\_ocr.py) │ │ │  
└──────────┬──────────┘ └───────────────────────┘  
 │  
┌──────────▼──────────┐ ┌───────────────────────┐  
│ Handwriting Detection│◄───┤ Handwriting OCR │  
│(handwriting\_recog.py)│ │ │  
└──────────┬──────────┘ └───────────────────────┘  
 │  
┌──────────▼──────────┐ ┌───────────────────────┐  
│ Confidence Scoring │◄───┤ Post-Processing Rules│  
│(ocr\_confidence\_scor.)│ │ │  
└──────────┬──────────┘ └───────────────────────┘  
 │  
┌──────────▼──────────┐  
│ Structured Output │  
│ │  
└──────────┬──────────┘  
 │  
┌──────────▼──────────┐  
│ Integration with │  
│ Material System │  
└─────────────────────┘

┌─────────────────────┐  
│ PDF Document │  
└──────────┬──────────┘  
 │  
┌──────────▼──────────┐  
│ PDF Image Extraction│  
│ (pdf\_extractor.py) │  
└──────────┬──────────┘  
 │  
┌──────────▼──────────┐ ┌───────────────────────┐  
│ Layout Analysis │◄───┤ Document Structure │  
│ (layout\_analysis.py) │ │ Classification │  
└──────────┬──────────┘ └───────────────────────┘  
 │  
┌──────────▼──────────┐ ┌───────────────────────┐  
│ Region Classification│◄──┤ Form Field Detection │  
│ │ │(form\_field\_extraction)│  
└──────────┬──────────┘ └───────────────────────┘  
 │  
┌──────────▼──────────┐ ┌───────────────────────┐  
│ Specialized OCR │◄───┤ Language Detection │  
│ (specialized\_ocr.py) │ │ │  
└──────────┬──────────┘ └───────────────────────┘  
 │  
┌──────────▼──────────┐ ┌───────────────────────┐  
│ Handwriting Detection│◄───┤ Handwriting OCR │  
│(handwriting\_recog.py)│ │ │  
└──────────┬──────────┘ └───────────────────────┘  
 │  
┌──────────▼──────────┐ ┌───────────────────────┐  
│ Confidence Scoring │◄───┤ Post-Processing Rules│  
│(ocr\_confidence\_scor.)│ │ │  
└──────────┬──────────┘ └───────────────────────┘  
 │  
┌──────────▼──────────┐  
│ Structured Output │  
│ │  
└──────────┬──────────┘  
 │  
┌──────────▼──────────┐  
│ Integration with │  
│ Material System │  
└─────────────────────┘

#### Performance Considerations

The enhanced OCR system introduces additional processing steps that may affect performance:

1. Processing Time: Full enhancement pipeline may increase processing time by 2-3x compared to basic OCR.
2. Memory Usage: Complex documents with multiple pages may require 1-2GB of memory during processing.
3. SVBRDF Processing Requirements:
4. GPU acceleration strongly recommended for SVBRDF property extraction
5. Typical processing time: 2-5 seconds per image on GPU, 30-45 seconds on CPU
6. Memory requirements: ~2GB for 512x512 resolution maps
7. Optimization Opportunities:
8. Parallel processing of different pages
9. Selective application of enhancements based on document type
10. GPU acceleration for handwriting recognition and layout analysis
11. Caching of intermediate results for frequently processed document templates

Processing Time: Full enhancement pipeline may increase processing time by 2-3x compared to basic OCR.

Memory Usage: Complex documents with multiple pages may require 1-2GB of memory during processing.

SVBRDF Processing Requirements:

Memory requirements: ~2GB for 512x512 resolution maps

Optimization Opportunities:

#### Usage Examples

##### Basic Usage through Server API

The OCR enhancements can be accessed through the existing PDF processing API routes:

// Example integration in pdf.routes.ts  
router.post('/enhanced-ocr', async (req, res) => {  
 const { filePath, options } = req.body;  
  
 try {  
 const result = await pdfProcessor.processWithEnhancedOCR(filePath, options);  
 res.json(result);  
 } catch (error) {  
 res.status(500).json({ error: error.message });  
 }  
});

// Example integration in pdf.routes.ts  
router.post('/enhanced-ocr', async (req, res) => {  
 const { filePath, options } = req.body;  
  
 try {  
 const result = await pdfProcessor.processWithEnhancedOCR(filePath, options);  
 res.json(result);  
 } catch (error) {  
 res.status(500).json({ error: error.message });  
 }  
});

##### Programmatic Usage

from packages.ml.python.enhanced\_ocr import EnhancedOCRProcessor  
  
# Initialize the processor with options  
processor = EnhancedOCRProcessor(  
 languages=['eng', 'deu'],  
 material\_type='tile',  
 enable\_handwriting\_detection=True,  
 enable\_form\_extraction=True  
)  
  
# Process a document  
results = processor.process\_document('path/to/document.pdf')  
  
# Access structured data  
specifications = results.get\_specifications()  
tables = results.get\_tables()  
form\_data = results.get\_form\_fields()

from packages.ml.python.enhanced\_ocr import EnhancedOCRProcessor  
  
# Initialize the processor with options  
processor = EnhancedOCRProcessor(  
 languages=['eng', 'deu'],  
 material\_type='tile',  
 enable\_handwriting\_detection=True,  
 enable\_form\_extraction=True  
)  
  
# Process a document  
results = processor.process\_document('path/to/document.pdf')  
  
# Access structured data  
specifications = results.get\_specifications()  
tables = results.get\_tables()  
form\_data = results.get\_form\_fields()

##### SVBRDF Material Properties Extraction

// Using the SVBRDF MCP adapter  
import { svbrdfMcpAdapter } from '@kai/agents/services/adapters/svbrdfMcpAdapter';  
  
// Extract SVBRDF properties from an image  
const svbrdfProperties = await svbrdfMcpAdapter.extractSVBRDFProperties({  
 imagePath: 'path/to/material/image.jpg',  
 resolution: 512, // Output resolution for property maps  
 enhanceDetail: true, // Optional enhancement for detail  
});  
  
// Access the extracted properties  
const { diffuseMap, normalMap, roughnessMap, metallicMap } = svbrdfProperties;  
  
// Apply SVBRDF properties to a material in the database  
await svbrdfMcpAdapter.applySVBRDFToMaterial({  
 materialId: 'material-123',  
 svbrdfProperties,  
 metadata: {  
 extractionMethod: 'neural-capture',  
 confidenceScore: 0.92  
 }  
});

// Using the SVBRDF MCP adapter  
import { svbrdfMcpAdapter } from '@kai/agents/services/adapters/svbrdfMcpAdapter';  
  
// Extract SVBRDF properties from an image  
const svbrdfProperties = await svbrdfMcpAdapter.extractSVBRDFProperties({  
 imagePath: 'path/to/material/image.jpg',  
 resolution: 512, // Output resolution for property maps  
 enhanceDetail: true, // Optional enhancement for detail  
});  
  
// Access the extracted properties  
const { diffuseMap, normalMap, roughnessMap, metallicMap } = svbrdfProperties;  
  
// Apply SVBRDF properties to a material in the database  
await svbrdfMcpAdapter.applySVBRDFToMaterial({  
 materialId: 'material-123',  
 svbrdfProperties,  
 metadata: {  
 extractionMethod: 'neural-capture',  
 confidenceScore: 0.92  
 }  
});

#### Dependencies and Requirements

The OCR enhancements rely on several key libraries:

* Tesseract OCR 4.1+ with language packs
* OpenCV for image processing
* PyMuPDF for PDF manipulation
* TensorFlow for handwriting recognition
* Various NLP libraries for text processing
* TensorFlow 2.x with compatibility mode for SVBRDF models

See requirements-ocr.txt for a complete list of dependencies.

requirements-ocr.txt

### Training API Improvements

The Training API provides enhanced capabilities for training Material Recognition models.

#### Overview of Improvements

The following improvements have been implemented:

1. Transfer Learning Capabilities: Fine-tune existing models with small datasets
2. Automated Hyperparameter Optimization: Implement techniques like grid search, random search, and Bayesian optimization
3. Distributed Training with Supabase: Replace Redis with Supabase for scaling training jobs
4. Training Progress Visualization: Enhanced progress reporting with real-time charts and metrics
5. Active Learning Integration: Prioritize samples for manual labeling based on model uncertainty
6. Automated Model Retraining Triggers: Automatically retrain when data changes significantly
7. Vector Database Integration: Store and retrieve embeddings for efficient similarity search

#### Model Storage and Management

Our training system handles models in a sophisticated way:

1. Base Pre-trained Models:
2. Loaded dynamically from ML framework libraries (TensorFlow, PyTorch)
3. Not stored directly in our application repository for efficiency
4. Frameworks automatically download and cache weights as needed
5. Fine-tuned Models:
6. Trained models are saved with metadata in the specified output directory
7. Models are versioned and can be retrieved for inference or further training
8. Training results and configurations are persisted alongside the model
9. Model Storage Location:  
    /models/  
    ├── {model\_id}/  
    │ ├── model.h5 (or .pt for PyTorch)  
    │ ├── metadata.json  
    │ ├── training\_history.json  
    │ └── hyperparameters.json

Frameworks automatically download and cache weights as needed

Fine-tuned Models:

Training results and configurations are persisted alongside the model

Model Storage Location:  
 /models/  
 ├── {model\_id}/  
 │ ├── model.h5 (or .pt for PyTorch)  
 │ ├── metadata.json  
 │ ├── training\_history.json  
 │ └── hyperparameters.json

/models/  
 ├── {model\_id}/  
 │ ├── model.h5 (or .pt for PyTorch)  
 │ ├── metadata.json  
 │ ├── training\_history.json  
 │ └── hyperparameters.json

This approach provides an optimal balance between leveraging existing pre-trained architectures and maintaining our own specialized fine-tuned versions.

#### Architecture

The improved training API consists of several interconnected modules:

* Transfer Learning Module (transfer\_learning.py): Enables fine-tuning of pre-trained models
* Hyperparameter Optimization Module (hyperparameter\_optimization.py): Automatically finds optimal model parameters
* Distributed Training Module (distributed\_training.py): Coordinates distributed training using Supabase
* Training Visualization Module (training\_visualization.py): Provides enhanced visualizations of training metrics
* Active Learning Module (active\_learning.py): Implements uncertainty-based sample selection for labeling
* Unified Training API (training\_api.py): Integrates all improvements into a cohesive system
* Model Storage Manager (model\_storage.py): Handles model persistence and retrieval
* Vector Database Connector (vector\_db\_connector.py): Manages embedding storage and retrieval

transfer\_learning.py

hyperparameter\_optimization.py

distributed\_training.py

training\_visualization.py

active\_learning.py

training\_api.py

model\_storage.py

vector\_db\_connector.py

#### Features

##### Transfer Learning

The transfer learning module allows you to leverage pre-trained models and fine-tune them with smaller datasets. This approach dramatically reduces training time and improves performance when training data is limited.

Features:  
- Support for TensorFlow and PyTorch frameworks  
- Customizable fine-tuning strategies  
- Layer freezing options to control what gets retrained  
- Data augmentation techniques for small datasets  
- Automatic saving of fine-tuned models for later use

##### Hyperparameter Optimization

The hyperparameter optimization module automates the process of finding optimal model parameters, eliminating manual trial and error.

Supported optimization strategies:  
- Grid Search: Exhaustively searches through a specified parameter grid  
- Random Search: Randomly samples from parameter distributions  
- Bayesian Optimization: Uses probabilistic models to guide the search process

Each strategy uses sparse categorical cross-entropy loss for classification tasks and applies early stopping with validation loss monitoring.

##### Distributed Training with Supabase

The distributed training module replaces Redis with Supabase for coordination and parameter sharing, providing:

* Scalable job queue management
* Worker coordination
* Parameter sharing across nodes
* Progress tracking and monitoring
* Fault tolerance and job recovery
* Real-time parameter updates during training

##### Training Progress Visualization

The visualization module enhances progress reporting with detailed charts and metrics:

* Real-time training metrics visualization
* Learning curve analysis
* Confusion matrix visualization
* Model performance comparisons
* Exportable reports in various formats (HTML, JSON, PNG)

##### Active Learning

The active learning module helps prioritize samples for manual labeling:

* Uncertainty-based sample selection
* Diversity sampling strategies
* Batch labeling workflow
* Integration with the feedback system

##### Automated Retraining Triggers

The system can automatically trigger model retraining based on various conditions:

* Feedback count threshold
* Time-based triggers
* Uncertainty threshold triggers
* Distribution shift detection

#### Usage

##### Unified API

The unified training API provides a simple interface to access all improvements:

from training\_api import EnhancedTrainingAPI, train\_with\_all\_improvements  
  
# Quick start with all improvements  
result = train\_with\_all\_improvements(  
 dataset\_path="path/to/dataset",  
 model\_type="hybrid",  
 use\_transfer\_learning=True,  
 optimize\_hyperparams=True,  
 distributed=True,  
 num\_workers=4  
)  
  
# Or use the full API for more control  
api = EnhancedTrainingAPI(  
 base\_dir="./training",  
 supabase\_url="your\_supabase\_url",  
 supabase\_key="your\_supabase\_key",  
 use\_distributed=True,  
 enable\_transfer\_learning=True,  
 enable\_hyperparameter\_optimization=True,  
 enable\_active\_learning=True,  
 visualization\_level="detailed",  
 num\_workers=4  
)  
  
# Train a model  
training\_result = api.train\_model(  
 dataset\_path="path/to/dataset",  
 model\_type="hybrid",  
 pretrained\_model\_path="path/to/pretrained/model",  
 optimize\_hyperparams=True  
)  
  
# Get samples for labeling  
labeling\_batch = api.get\_samples\_for\_labeling(count=10)  
  
# Record feedback  
feedback\_result = api.record\_labeling\_feedback(  
 sample\_id="sample\_123",  
 correct\_material\_id="material\_456",  
 batch\_id="batch\_789"  
)  
  
# Retrain from feedback  
retrain\_result = api.retrain\_from\_feedback(  
 model\_type="hybrid",  
 feedback\_threshold=10  
)

from training\_api import EnhancedTrainingAPI, train\_with\_all\_improvements  
  
# Quick start with all improvements  
result = train\_with\_all\_improvements(  
 dataset\_path="path/to/dataset",  
 model\_type="hybrid",  
 use\_transfer\_learning=True,  
 optimize\_hyperparams=True,  
 distributed=True,  
 num\_workers=4  
)  
  
# Or use the full API for more control  
api = EnhancedTrainingAPI(  
 base\_dir="./training",  
 supabase\_url="your\_supabase\_url",  
 supabase\_key="your\_supabase\_key",  
 use\_distributed=True,  
 enable\_transfer\_learning=True,  
 enable\_hyperparameter\_optimization=True,  
 enable\_active\_learning=True,  
 visualization\_level="detailed",  
 num\_workers=4  
)  
  
# Train a model  
training\_result = api.train\_model(  
 dataset\_path="path/to/dataset",  
 model\_type="hybrid",  
 pretrained\_model\_path="path/to/pretrained/model",  
 optimize\_hyperparams=True  
)  
  
# Get samples for labeling  
labeling\_batch = api.get\_samples\_for\_labeling(count=10)  
  
# Record feedback  
feedback\_result = api.record\_labeling\_feedback(  
 sample\_id="sample\_123",  
 correct\_material\_id="material\_456",  
 batch\_id="batch\_789"  
)  
  
# Retrain from feedback  
retrain\_result = api.retrain\_from\_feedback(  
 model\_type="hybrid",  
 feedback\_threshold=10  
)

##### Command Line Interface

You can also use the command line interface:

# Train a model with all improvements  
python training\_api.py train \  
 --dataset path/to/dataset \  
 --model-type hybrid \  
 --use-distributed \  
 --optimize-hyperparams \  
 --use-transfer-learning \  
 --num-workers 4  
  
# Active learning operations  
python training\_api.py active-learning \  
 --operation select \  
 --data-dir ./data \  
 --model-dir ./models \  
 --count 10  
  
# Start retraining monitor  
python training\_api.py monitor \  
 --data-dir ./data \  
 --model-dir ./models \  
 --check-interval 3600  
  
# Distributed training operations  
python training\_api.py distributed \  
 --operation start-workers \  
 --data-dir ./data \  
 --model-dir ./models \  
 --num-workers 4

# Train a model with all improvements  
python training\_api.py train \  
 --dataset path/to/dataset \  
 --model-type hybrid \  
 --use-distributed \  
 --optimize-hyperparams \  
 --use-transfer-learning \  
 --num-workers 4  
  
# Active learning operations  
python training\_api.py active-learning \  
 --operation select \  
 --data-dir ./data \  
 --model-dir ./models \  
 --count 10  
  
# Start retraining monitor  
python training\_api.py monitor \  
 --data-dir ./data \  
 --model-dir ./models \  
 --check-interval 3600  
  
# Distributed training operations  
python training\_api.py distributed \  
 --operation start-workers \  
 --data-dir ./data \  
 --model-dir ./models \  
 --num-workers 4

#### Integration with Supabase

The Supabase integration replaces Redis for various distributed functions:

1. Job Queue Management: Training jobs are stored in Supabase tables
2. Parameter Sharing: Model parameters are stored and retrieved from Supabase
3. Progress Tracking: Training progress is recorded in Supabase for real-time monitoring
4. Active Learning Storage: Samples, batches, and triggers are stored in Supabase

Required Supabase Tables:  
- training\_jobs: Stores training job information  
- training\_parameters: Stores model parameters  
- training\_progress: Stores training progress updates  
- active\_learning\_candidates: Stores sample candidates for labeling  
- active\_learning\_batches: Stores labeling batches  
- retraining\_triggers: Stores retraining triggers

training\_jobs

training\_parameters

training\_progress

active\_learning\_candidates

active\_learning\_batches

retraining\_triggers

#### Knowledge Base Integration

Our knowledge base is tightly coupled with the ML training system:

1. Material Metadata Source:
2. Provides rich context for training data
3. Supplies detailed material specifications and categorization
4. Enables better model training through contextual understanding
5. Training Enhancement:
6. Material relationships inform data augmentation strategies
7. Category structures guide model architecture decisions
8. Historical usage patterns influence sample weighting
9. Feedback Loop:
10. Recognition results are recorded in the knowledge base
11. User feedback on predictions enrich the training data
12. Automated retraining triggers based on feedback patterns

Enables better model training through contextual understanding

Training Enhancement:

Historical usage patterns influence sample weighting

Feedback Loop:

This bidirectional integration creates a continuously improving system where ML models and knowledge base mutually enhance each other.

### Vector Database Integration

The ML package integrates with vector databases to provide advanced vector embedding storage and similarity search capabilities.

#### Core Vector Functionality

The ML package leverages Supabase Vector (using PostgreSQL's pgvector extension) for:

1. Vector Embedding Storage
2. Storing feature vectors generated from material images
3. Associating embeddings with material metadata
4. Efficient vector operations via pgvector
5. Automatic vector indexing for high-performance searches
6. Similarity Search
7. Semantic similarity between material vectors
8. Filtering by material type, properties, or metadata
9. Configurable similarity thresholds and result limits
10. High-performance vector operations via optimized indices
11. Database Implementation
12. Dedicated vector tables in Supabase PostgreSQL
13. Vector columns with appropriate dimensionality
14. Optimized indices using HNSW or IVF-Flat
15. Metadata columns for rich material information

Automatic vector indexing for high-performance searches

Similarity Search

High-performance vector operations via optimized indices

Database Implementation

#### Local Vector Search (FAISS)

Create a local vector search index and search for similar materials:

import { createVectorSearchIndex, searchSimilarMaterials } from '@kai/ml';  
  
async function setupAndSearch() {  
 try {  
 // Create search index  
 await createVectorSearchIndex('path/to/embeddings', 'models/search\_index.faiss');  
  
 // Search for similar materials  
 const result = await searchSimilarMaterials('models/search\_index.faiss', 'path/to/query.jpg', {  
 numResults: 5,  
 threshold: 0.7  
 });  
  
 console.log('Similar materials:');  
 result.results.forEach(match => {  
 console.log(`- ${match.materialId} (similarity: ${match.similarity.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Vector search failed:', error);  
 }  
}

import { createVectorSearchIndex, searchSimilarMaterials } from '@kai/ml';  
  
async function setupAndSearch() {  
 try {  
 // Create search index  
 await createVectorSearchIndex('path/to/embeddings', 'models/search\_index.faiss');  
  
 // Search for similar materials  
 const result = await searchSimilarMaterials('models/search\_index.faiss', 'path/to/query.jpg', {  
 numResults: 5,  
 threshold: 0.7  
 });  
  
 console.log('Similar materials:');  
 result.results.forEach(match => {  
 console.log(`- ${match.materialId} (similarity: ${match.similarity.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Vector search failed:', error);  
 }  
}

#### Supabase Vector Integration

Store and search vector embeddings using Supabase Vector:

import { storeEmbeddingInSupabase, searchSimilarInSupabase } from '@kai/ml';  
  
async function supabaseVectorSearch() {  
 try {  
 // Generate and store embedding for a material  
 const embeddingId = await storeEmbeddingInSupabase('path/to/image.jpg', {  
 materialId: 'marble-001',  
 materialName: 'Carrara Marble',  
 materialType: 'marble',  
 metadata: {   
 color: 'white',  
 finish: 'polished'  
 }  
 });  
  
 console.log(`Stored embedding with ID: ${embeddingId}`);  
  
 // Search for similar materials with Supabase Vector  
 const similarMaterials = await searchSimilarInSupabase('path/to/query.jpg', {  
 threshold: 0.7,  
 limit: 5,  
 materialType: 'marble' // Optional filter  
 });  
  
 console.log('Similar materials:');  
 similarMaterials.forEach(match => {  
 console.log(`- ${match.materialName} (similarity: ${match.similarity.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Supabase vector operations failed:', error);  
 }  
}

import { storeEmbeddingInSupabase, searchSimilarInSupabase } from '@kai/ml';  
  
async function supabaseVectorSearch() {  
 try {  
 // Generate and store embedding for a material  
 const embeddingId = await storeEmbeddingInSupabase('path/to/image.jpg', {  
 materialId: 'marble-001',  
 materialName: 'Carrara Marble',  
 materialType: 'marble',  
 metadata: {   
 color: 'white',  
 finish: 'polished'  
 }  
 });  
  
 console.log(`Stored embedding with ID: ${embeddingId}`);  
  
 // Search for similar materials with Supabase Vector  
 const similarMaterials = await searchSimilarInSupabase('path/to/query.jpg', {  
 threshold: 0.7,  
 limit: 5,  
 materialType: 'marble' // Optional filter  
 });  
  
 console.log('Similar materials:');  
 similarMaterials.forEach(match => {  
 console.log(`- ${match.materialName} (similarity: ${match.similarity.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Supabase vector operations failed:', error);  
 }  
}

#### Integration with Application Domains

This ML package provides the vector foundations for:

1. Material Recognition
2. Visual feature vectors for material classification
3. Similarity-based material identification
4. Flexible confidence thresholds
5. Multiple recognition strategy support
6. Query Understanding
7. Natural language query embedding
8. Semantic search enhancement
9. Query expansion based on vector similarity
10. Domain-specific context integration
11. Recommendation Engine
12. User preference vector modeling
13. Similarity-based recommendation generation
14. Diversity control in recommendations
15. Feedback loop for preference adaption
16. Document Processing
17. Text chunk vectorization
18. Semantic document search
19. Entity extraction and linking
20. Cross-document relationship discovery

Multiple recognition strategy support

Query Understanding

Domain-specific context integration

Recommendation Engine

Feedback loop for preference adaption

Document Processing

### Deployment and Installation

Note: Installation instructions for the ML components have been moved to the Deployment Guide.

The Deployment Guide includes detailed instructions for:

* ML Package installation
* MCP Server deployment
* OCR dependencies setup
* Vector database configuration
* Training system deployment

### Usage Examples

#### PDF Processing

Extract images and text from a PDF catalog:

import { extractFromPDF } from '@kai/ml';  
  
async function processPDF() {  
 try {  
 const result = await extractFromPDF('path/to/catalog.pdf', 'output/directory');  
 console.log(`Extracted ${result.images.length} images and ${result.text.length} text blocks`);  
 } catch (error) {  
 console.error('PDF extraction failed:', error);  
 }  
}

import { extractFromPDF } from '@kai/ml';  
  
async function processPDF() {  
 try {  
 const result = await extractFromPDF('path/to/catalog.pdf', 'output/directory');  
 console.log(`Extracted ${result.images.length} images and ${result.text.length} text blocks`);  
 } catch (error) {  
 console.error('PDF extraction failed:', error);  
 }  
}

#### Material Recognition

Recognize materials in an image:

import { recognizeMaterial } from '@kai/ml';  
  
async function identifyMaterial() {  
 try {  
 const result = await recognizeMaterial('path/to/image.jpg', {  
 modelType: 'hybrid', // 'hybrid', 'feature-based', or 'ml-based'  
 confidenceThreshold: 0.6,  
 maxResults: 5  
 });  
  
 console.log('Recognized materials:');  
 result.matches.forEach(match => {  
 console.log(`- ${match.materialId} (confidence: ${match.confidence.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Material recognition failed:', error);  
 }  
}

import { recognizeMaterial } from '@kai/ml';  
  
async function identifyMaterial() {  
 try {  
 const result = await recognizeMaterial('path/to/image.jpg', {  
 modelType: 'hybrid', // 'hybrid', 'feature-based', or 'ml-based'  
 confidenceThreshold: 0.6,  
 maxResults: 5  
 });  
  
 console.log('Recognized materials:');  
 result.matches.forEach(match => {  
 console.log(`- ${match.materialId} (confidence: ${match.confidence.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Material recognition failed:', error);  
 }  
}

#### Enhanced Recognition with Confidence Fusion

Use the enhanced recognition with confidence fusion for better results:

import { recognizeMaterialEnhanced } from '@kai/ml';  
  
async function identifyMaterialEnhanced() {  
 try {  
 const result = await recognizeMaterialEnhanced('path/to/image.jpg', {  
 useFusion: true,  
 fusionMethod: 'adaptive', // 'weighted', 'adaptive', 'max', or 'product'  
 fusionAlpha: 0.5,  
 confidenceThreshold: 0.6,  
 maxResults: 5  
 });  
  
 console.log('Recognized materials:');  
 result.matches.forEach(match => {  
 console.log(`- ${match.materialId} (confidence: ${match.confidence.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Material recognition failed:', error);  
 }  
}

import { recognizeMaterialEnhanced } from '@kai/ml';  
  
async function identifyMaterialEnhanced() {  
 try {  
 const result = await recognizeMaterialEnhanced('path/to/image.jpg', {  
 useFusion: true,  
 fusionMethod: 'adaptive', // 'weighted', 'adaptive', 'max', or 'product'  
 fusionAlpha: 0.5,  
 confidenceThreshold: 0.6,  
 maxResults: 5  
 });  
  
 console.log('Recognized materials:');  
 result.matches.forEach(match => {  
 console.log(`- ${match.materialId} (confidence: ${match.confidence.toFixed(2)})`);  
 });  
 } catch (error) {  
 console.error('Material recognition failed:', error);  
 }  
}

#### Feature Descriptor Generation

Generate feature descriptors from a dataset of material images:

import { generateFeatureDescriptors } from '@kai/ml';  
  
async function generateDescriptors() {  
 try {  
 const result = await generateFeatureDescriptors('path/to/dataset', 'models/feature\_descriptors.npz');  
 console.log(`Generated descriptors for ${result.material\_count} materials with ${result.total\_descriptors} total descriptors`);  
 } catch (error) {  
 console.error('Feature descriptor generation failed:', error);  
 }  
}

import { generateFeatureDescriptors } from '@kai/ml';  
  
async function generateDescriptors() {  
 try {  
 const result = await generateFeatureDescriptors('path/to/dataset', 'models/feature\_descriptors.npz');  
 console.log(`Generated descriptors for ${result.material\_count} materials with ${result.total\_descriptors} total descriptors`);  
 } catch (error) {  
 console.error('Feature descriptor generation failed:', error);  
 }  
}

#### Neural Network Training

Train a neural network model for material recognition:

import { trainNeuralNetwork } from '@kai/ml';  
  
async function trainModel() {  
 try {  
 const result = await trainNeuralNetwork('path/to/dataset', 'models/neural\_network', {  
 framework: 'tensorflow', // 'tensorflow' or 'pytorch'  
 model: 'mobilenetv2', // 'mobilenetv2', 'resnet18', 'efficientnet'  
 epochs: 10,  
 batchSize: 32,  
 imgSize: 224,  
 learningRate: 0.001  
 });  
  
 console.log(`Model trained with ${result.num\_classes} classes`);  
 console.log(`Final accuracy: ${result.final\_accuracy.toFixed(4)}`);  
 console.log(`Final validation accuracy: ${result.final\_val\_accuracy.toFixed(4)}`);  
 } catch (error) {  
 console.error('Neural network training failed:', error);  
 }  
}

import { trainNeuralNetwork } from '@kai/ml';  
  
async function trainModel() {  
 try {  
 const result = await trainNeuralNetwork('path/to/dataset', 'models/neural\_network', {  
 framework: 'tensorflow', // 'tensorflow' or 'pytorch'  
 model: 'mobilenetv2', // 'mobilenetv2', 'resnet18', 'efficientnet'  
 epochs: 10,  
 batchSize: 32,  
 imgSize: 224,  
 learningRate: 0.001  
 });  
  
 console.log(`Model trained with ${result.num\_classes} classes`);  
 console.log(`Final accuracy: ${result.final\_accuracy.toFixed(4)}`);  
 console.log(`Final validation accuracy: ${result.final\_val\_accuracy.toFixed(4)}`);  
 } catch (error) {  
 console.error('Neural network training failed:', error);  
 }  
}

#### Visualize Search Results

Visualize the search results with side-by-side comparison:

import { visualizeSearchResults } from '@kai/ml';  
  
async function visualizeResults() {  
 try {  
 const outputPath = await visualizeSearchResults(  
 'models/search\_index.faiss',  
 'path/to/query.jpg',  
 'output/visualization.jpg',  
 5 // Number of results to visualize  
 );  
  
 console.log(`Visualization saved to ${outputPath}`);  
 } catch (error) {  
 console.error('Visualization failed:', error);  
 }  
}

import { visualizeSearchResults } from '@kai/ml';  
  
async function visualizeResults() {  
 try {  
 const outputPath = await visualizeSearchResults(  
 'models/search\_index.faiss',  
 'path/to/query.jpg',  
 'output/visualization.jpg',  
 5 // Number of results to visualize  
 );  
  
 console.log(`Visualization saved to ${outputPath}`);  
 } catch (error) {  
 console.error('Visualization failed:', error);  
 }  
}

### API Reference

#### PDF Processing

* extractFromPDF(pdfPath: string, outputDir: string): Promise<PDFExtractionResult>

extractFromPDF(pdfPath: string, outputDir: string): Promise<PDFExtractionResult>

#### Material Recognition

* recognizeMaterial(imagePath: string, options?: RecognitionOptions): Promise<RecognitionResult>
* recognizeMaterialEnhanced(imagePath: string, options?: EnhancedRecognitionOptions): Promise<RecognitionResult | ConfidenceFusionResult>

recognizeMaterial(imagePath: string, options?: RecognitionOptions): Promise<RecognitionResult>

recognizeMaterialEnhanced(imagePath: string, options?: EnhancedRecognitionOptions): Promise<RecognitionResult | ConfidenceFusionResult>

#### Feature Descriptors

* generateFeatureDescriptors(datasetDir: string, outputFile: string): Promise<FeatureDescriptorResult>

generateFeatureDescriptors(datasetDir: string, outputFile: string): Promise<FeatureDescriptorResult>

#### Neural Network Training

* trainNeuralNetwork(datasetDir: string, outputDir: string, options?: TrainingOptions): Promise<NeuralNetworkTrainingResult>

trainNeuralNetwork(datasetDir: string, outputDir: string, options?: TrainingOptions): Promise<NeuralNetworkTrainingResult>

#### Vector Search

* createVectorSearchIndex(embeddingsDir: string, indexPath: string): Promise<IndexCreationResult>
* searchSimilarMaterials(indexPath: string, imagePath: string, options?: SearchOptions): Promise<VectorSearchResult>
* visualizeSearchResults(indexPath: string, imagePath: string, outputPath: string, numResults?: number): Promise<string>

createVectorSearchIndex(embeddingsDir: string, indexPath: string): Promise<IndexCreationResult>

searchSimilarMaterials(indexPath: string, imagePath: string, options?: SearchOptions): Promise<VectorSearchResult>

visualizeSearchResults(indexPath: string, imagePath: string, outputPath: string, numResults?: number): Promise<string>

#### Supabase Vector Operations

* storeEmbeddingInSupabase(imagePath: string, metadata: MaterialMetadata): Promise<string>
* searchSimilarInSupabase(imagePath: string, options?: SupabaseSearchOptions): Promise<SupabaseSearchResult[]>
* createVectorIndex(tableName: string, columnName: string, indexMethod?: 'hnsw' | 'ivfflat', dimensions?: number): Promise<boolean>
* generateAndStoreEmbedding(material: Material): Promise<string>

storeEmbeddingInSupabase(imagePath: string, metadata: MaterialMetadata): Promise<string>

searchSimilarInSupabase(imagePath: string, options?: SupabaseSearchOptions): Promise<SupabaseSearchResult[]>

createVectorIndex(tableName: string, columnName: string, indexMethod?: 'hnsw' | 'ivfflat', dimensions?: number): Promise<boolean>

generateAndStoreEmbedding(material: Material): Promise<string>

#### Confidence Fusion

* fuseConfidenceScores(featureResults: RecognitionResult, mlResults: RecognitionResult, options?: FusionOptions): Promise<ConfidenceFusionResult>

fuseConfidenceScores(featureResults: RecognitionResult, mlResults: RecognitionResult, options?: FusionOptions): Promise<ConfidenceFusionResult>

#### MCP Client

* MCPClient(url: string, options?: MCPClientOptions)
* recognizeMaterial(imagePath: string, options?: RecognitionOptions): Promise<RecognitionResult>
* listModels(): Promise<ModelList>
* getModelInfo(modelId: string): Promise<ModelInfo>
* getModelContext(modelId: string): Promise<ModelContext>
* updateModelContext(modelId: string, context: ModelContext): Promise<UpdateResult>
* sendAgentMessage(message: AgentMessage): Promise<MessageResult>
* getAgentMessages(timeout?: number): Promise<MessageBatch>

MCPClient(url: string, options?: MCPClientOptions)

recognizeMaterial(imagePath: string, options?: RecognitionOptions): Promise<RecognitionResult>

listModels(): Promise<ModelList>

getModelInfo(modelId: string): Promise<ModelInfo>

getModelContext(modelId: string): Promise<ModelContext>

updateModelContext(modelId: string, context: ModelContext): Promise<UpdateResult>

sendAgentMessage(message: AgentMessage): Promise<MessageResult>

getAgentMessages(timeout?: number): Promise<MessageBatch>

#### OCR

* EnhancedOCRProcessor(options?: OCROptions)
* process\_document(document\_path: string): OCRResult
* extract\_tables(document\_path: string): TableExtractionResult
* extract\_form\_fields(document\_path: string): FormFieldExtractionResult
* detect\_handwriting(document\_path: string): HandwritingDetectionResult
* calculate\_confidence(text: string, domain?: string): ConfidenceResult

EnhancedOCRProcessor(options?: OCROptions)

process\_document(document\_path: string): OCRResult

extract\_tables(document\_path: string): TableExtractionResult

extract\_form\_fields(document\_path: string): FormFieldExtractionResult

detect\_handwriting(document\_path: string): HandwritingDetectionResult

calculate\_confidence(text: string, domain?: string): ConfidenceResult

#### Training API

* EnhancedTrainingAPI(options?: TrainingAPIOptions)
* train\_model(dataset\_path: string, model\_type: string, options?: TrainingOptions): Promise<TrainingResult>
* optimize\_hyperparameters(dataset\_path: string, model\_type: string, hp\_space: object): Promise<HyperparameterResult>
* get\_samples\_for\_labeling(count?: number): Promise<LabelingBatch>
* record\_labeling\_feedback(sample\_id: string, correct\_material\_id: string): Promise<FeedbackResult>
* check\_retraining\_triggers(): Promise<TriggerResult[]>
* retrain\_from\_feedback(model\_type: string): Promise<RetrainingResult>

EnhancedTrainingAPI(options?: TrainingAPIOptions)

train\_model(dataset\_path: string, model\_type: string, options?: TrainingOptions): Promise<TrainingResult>

optimize\_hyperparameters(dataset\_path: string, model\_type: string, hp\_space: object): Promise<HyperparameterResult>

get\_samples\_for\_labeling(count?: number): Promise<LabelingBatch>

record\_labeling\_feedback(sample\_id: string, correct\_material\_id: string): Promise<FeedbackResult>

check\_retraining\_triggers(): Promise<TriggerResult[]>

retrain\_from\_feedback(model\_type: string): Promise<RetrainingResult>

### Performance Considerations

#### General Performance

* Memory Management: The ML package automatically manages memory usage based on available resources and load
* GPU Acceleration: GPU is recommended for neural network inference and training
* Batch Processing: Implement batch processing for multiple materials or documents
* Caching: Use caching strategies for frequent operations

#### MCP Server Performance

* Hardware Requirements:
* CPU: 4+ cores recommended
* RAM: 8GB+ (16GB+ for multiple models)
* GPU: Optional but strongly recommended for neural networks
* Disk: 20GB+ for models and temporary storage
* Optimization Strategies:
* Enable model caching
* Use GPU acceleration when available
* Implement request batching for multiple items
* Configure appropriate timeouts for your environment

Disk: 20GB+ for models and temporary storage

Optimization Strategies:

#### OCR Performance

* Hardware Requirements:
* CPU: 4+ cores recommended
* RAM: 8GB+ (16GB+ for multiple parallel processes)
* GPU: Recommended for handwriting recognition
* Disk: 10GB+ for temporary storage
* Optimization Strategies:
* Use selective processing (only enable needed features)
* Process multiple documents in parallel
* Batch process pages from large documents
* Use GPU acceleration for handwriting recognition

Disk: 10GB+ for temporary storage

Optimization Strategies:

#### Training API Performance

* Hardware Requirements:
* CPU: 8+ cores recommended
* RAM: 16GB+ (32GB+ for large datasets)
* GPU: Strongly recommended for training
* Disk: 100GB+ for dataset storage and models
* Optimization Strategies:
* Use distributed training for large datasets
* Implement incremental training
* Use transfer learning to reduce training time
* Optimize batch size based on available memory

Disk: 100GB+ for dataset storage and models

Optimization Strategies:

### Troubleshooting

#### Common Issues

##### MCP Server Issues

1. Connection Errors
2. Check if the MCP server is running
3. Verify network connectivity and firewall settings
4. Check if the port is correctly exposed
5. Model Loading Failures
6. Ensure model files exist in the model directory
7. Check for sufficient memory for loading models
8. Verify GPU availability if using GPU-accelerated models
9. Slow Performance
10. Check if GPU is being utilized (if available)
11. Monitor memory usage for potential leaks
12. Consider increasing server resources

Check if the port is correctly exposed

Model Loading Failures

Verify GPU availability if using GPU-accelerated models

Slow Performance

##### OCR Issues

1. Low Recognition Accuracy
2. Ensure the document has sufficient resolution (300+ DPI)
3. Try specifying the correct language
4. Adjust confidence thresholds
5. Check if the document type is supported
6. Memory Errors
7. Process large documents in batches
8. Reduce the number of parallel processes
9. Increase available memory
10. Slow Processing
11. Disable unnecessary features
12. Use GPU acceleration if available
13. Process in batches or parallel
14. Check for resource contention

Check if the document type is supported

Memory Errors

Increase available memory

Slow Processing

##### Training API Issues

1. Out of Memory Errors
2. Reduce batch size
3. Use progressive loading
4. Enable gradient accumulation
5. Check for memory leaks
6. Slow Training
7. Use GPU acceleration
8. Implement distributed training
9. Optimize data pipeline
10. Check for I/O bottlenecks
11. Poor Model Performance
12. Increase dataset size or augmentation
13. Use transfer learning
14. Optimize hyperparameters
15. Check for data quality issues

Check for memory leaks

Slow Training

Check for I/O bottlenecks

Poor Model Performance

#### Logging and Debugging

To enable detailed logging for debugging:

# Enable debug logging for MCP Server  
export LOG\_LEVEL=DEBUG  
python packages/ml/python/mcp\_server.py  
  
# Enable debug logging for OCR  
export DEBUG=1  
python packages/ml/python/enhanced\_ocr.py  
  
# Enable debug logging for Training API  
export TRAINING\_API\_LOG\_LEVEL=DEBUG  
python packages/ml/python/training\_api.py

# Enable debug logging for MCP Server  
export LOG\_LEVEL=DEBUG  
python packages/ml/python/mcp\_server.py  
  
# Enable debug logging for OCR  
export DEBUG=1  
python packages/ml/python/enhanced\_ocr.py  
  
# Enable debug logging for Training API  
export TRAINING\_API\_LOG\_LEVEL=DEBUG  
python packages/ml/python/training\_api.py

Check logs for error messages and debugging information:

# View MCP Server logs  
docker logs kai-mcp-server  
  
# View OCR process logs  
tail -f /var/log/kai/ocr.log  
  
# View Training API logs  
tail -f /var/log/kai/training.log

# View MCP Server logs  
docker logs kai-mcp-server  
  
# View OCR process logs  
tail -f /var/log/kai/ocr.log  
  
# View Training API logs  
tail -f /var/log/kai/training.log

#### Getting Help

If you encounter issues not covered in this documentation:

1. Check the error logs for detailed information
2. Look for similar issues in the project issue tracker
3. Update to the latest version of the ML package
4. Contact the development team with detailed error information

### Future Improvements

Potential areas for further enhancement:

1. 3D Technical Drawing Recognition: Extract measurements and specifications from technical drawings.
2. Material Visual Properties Correlation: Link extracted specifications with visual recognition results.
3. Multi-document Cross-referencing: Correlate information across multiple related documents.
4. Interactive Correction Interface: Develop a UI for reviewing and correcting low-confidence OCR results.
5. Real-time OCR Streaming: Process documents incrementally as they are uploaded or scanned.
6. SVBRDF Fine-Tuning for Specific Materials: Train specialized SVBRDF models for specific material types (ceramic, wood, metal, etc.).
7. Distributed MCP Server Deployment: Support for multiple MCP servers with load balancing.
8. Model Version Control: Advanced management of model versions and rollbacks.
9. A/B Testing: Support for comparing performance between model versions.
10. Enhanced Agent Integration: Deeper integration with future AI agent capabilities.
11. Homogeneity Estimation: Analysis of material homogeneity patterns.
12. Vector Index Compression: Reduce memory/storage requirements for vector indices.

3D Technical Drawing Recognition: Extract measurements and specifications from technical drawings.

Material Visual Properties Correlation: Link extracted specifications with visual recognition results.

Multi-document Cross-referencing: Correlate information across multiple related documents.

Interactive Correction Interface: Develop a UI for reviewing and correcting low-confidence OCR results.

Real-time OCR Streaming: Process documents incrementally as they are uploaded or scanned.

SVBRDF Fine-Tuning for Specific Materials: Train specialized SVBRDF models for specific material types (ceramic, wood, metal, etc.).

Distributed MCP Server Deployment: Support for multiple MCP servers with load balancing.

Model Version Control: Advanced management of model versions and rollbacks.

A/B Testing: Support for comparing performance between model versions.

Enhanced Agent Integration: Deeper integration with future AI agent capabilities.

Homogeneity Estimation: Analysis of material homogeneity patterns.

Vector Index Compression: Reduce memory/storage requirements for vector indices.

# Ml Pdf Processing Pipeline

Source: readme/ml-pdf-processing-pipeline.md

---

## PDF Processing Pipeline

This document explains how the Kai Material Recognition system processes PDF catalogs to create a training dataset and knowledge base for material recognition.

### Overview

The PDF processing pipeline automatically extracts images and text from PDF catalogs, associates images with their specifications, and organizes them for training the material recognition models. This eliminates the need for manual dataset creation and organization.

### Pipeline Steps

1. PDF Upload: Users upload PDF catalogs through the admin interface.
2. Image Extraction: The system extracts images from the PDFs using PyMuPDF.  
    ```typescript  
    import { extractFromPDF } from '@kai/ml';

PDF Upload: Users upload PDF catalogs through the admin interface.

Image Extraction: The system extracts images from the PDFs using PyMuPDF.  
 ```typescript  
 import { extractFromPDF } from '@kai/ml';

const result = await extractFromPDF('path/to/catalog.pdf', 'output/directory');  
 ```

1. Text Extraction: The system extracts text blocks from the PDFs and associates them with nearby images.
2. OCR Processing: For text embedded in images or poorly extracted text, OCR is applied to ensure all specifications are captured.
3. Specification Parsing: The system parses the extracted text to identify material specifications such as:
4. Material type (tile, stone, wood, etc.)
5. Dimensions (size)
6. Technical properties (R-value, PEI rating, etc.)
7. Color and finish information
8. Image Preprocessing: Extracted images are preprocessed to improve quality:
9. Cropping to remove borders
10. Enhancing contrast and sharpness
11. Normalizing size and format
12. Dataset Organization: The system automatically organizes the processed images into a structured dataset:  
     processed\_data/  
     ├── material\_id\_1/  
     │ ├── image1.jpg  
     │ ├── image2.jpg  
     │ └── metadata.json  
     ├── material\_id\_2/  
     │ ├── image1.jpg  
     │ ├── image2.jpg  
     │ └── metadata.json  
     └── ...
13. Feature Extraction: SIFT features are extracted from all images and stored for feature-based matching.  
     ```typescript  
     import { generateFeatureDescriptors } from '@kai/ml';

Text Extraction: The system extracts text blocks from the PDFs and associates them with nearby images.

OCR Processing: For text embedded in images or poorly extracted text, OCR is applied to ensure all specifications are captured.

Specification Parsing: The system parses the extracted text to identify material specifications such as:

Color and finish information

Image Preprocessing: Extracted images are preprocessed to improve quality:

Normalizing size and format

Dataset Organization: The system automatically organizes the processed images into a structured dataset:  
 processed\_data/  
 ├── material\_id\_1/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── metadata.json  
 ├── material\_id\_2/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── metadata.json  
 └── ...

processed\_data/  
 ├── material\_id\_1/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── metadata.json  
 ├── material\_id\_2/  
 │ ├── image1.jpg  
 │ ├── image2.jpg  
 │ └── metadata.json  
 └── ...

Feature Extraction: SIFT features are extracted from all images and stored for feature-based matching.  
 ```typescript  
 import { generateFeatureDescriptors } from '@kai/ml';

const result = await generateFeatureDescriptors('processed\_data', 'models/feature\_descriptors.npz');  
 ```

1. Model Training: Neural network models are trained using the organized dataset.  
    ```typescript  
    import { trainNeuralNetwork } from '@kai/ml';

const result = await trainNeuralNetwork('processed\_data', 'models/neural\_network');  
 ```

1. Vector Embedding Generation: Vector embeddings are generated for all materials to enable similarity search.
2. Knowledge Base Integration: All extracted data is stored in the knowledge base, including:  
     
   Material images  
   Material specifications  
   Feature descriptors  
   Vector embeddings

Vector Embedding Generation: Vector embeddings are generated for all materials to enable similarity search.

Knowledge Base Integration: All extracted data is stored in the knowledge base, including:

* Material images
* Material specifications
* Feature descriptors
* Vector embeddings

### Automatic Material Classification

The system automatically classifies materials based on the extracted text and image features:

1. Text-Based Classification: Analyzes specification text for material type indicators:
2. Keywords like "tile", "stone", "wood", etc.
3. Material-specific properties (e.g., PEI rating for tiles)
4. Image-Based Classification: Uses pre-trained models to classify materials based on visual appearance.
5. Combined Classification: Merges text and image classification results for more accurate material type determination.

Material-specific properties (e.g., PEI rating for tiles)

Image-Based Classification: Uses pre-trained models to classify materials based on visual appearance.

Combined Classification: Merges text and image classification results for more accurate material type determination.

### Batch Processing

For large catalogs or multiple PDFs, the system uses a queue-based batch processing approach:

1. PDFs are added to a processing queue
2. Processing jobs are distributed across available resources
3. Results are aggregated into the knowledge base as they complete

### Monitoring and Validation

The admin interface provides tools to monitor and validate the PDF processing:

1. Processing Status: Track the progress of PDF processing jobs
2. Extraction Review: Review extracted images and specifications
3. Manual Correction: Correct any errors in the extracted data
4. Validation Tools: Validate the quality of extracted data

### Integration with Recognition System

Once processed, the data is immediately available for the material recognition system:

1. Feature-Based Matching: Uses extracted SIFT features for direct matching
2. ML-Based Classification: Uses trained neural network models for classification
3. Vector Search: Uses generated embeddings for similarity search
4. Hybrid Approach: Combines all methods for optimal recognition results

### Conclusion

The PDF processing pipeline automates the creation of the training dataset and knowledge base, eliminating the need for manual dataset organization. This approach ensures that the material recognition system can be quickly populated with new materials simply by uploading PDF catalogs.

# Ml Unified Services

Source: readme/ml-unified-services.md

---

## ML Package Unified Services

This document describes the implementation of the unified services architecture in the ml package. The unified services architecture completely removes backward compatibility layers and uses the unified services directly.

### Implementation Overview

The ml package has been updated to use the unified services from the shared package directly. The following changes were made:

1. Removed duplicate files:
2. Removed logger.ts - Replaced with unified logger
3. Removed mcp-integration.ts - Replaced with unified MCP client
4. Created a unified services export:
5. Created services/index.ts to export all unified services from the shared package
6. Updated components to use unified services directly:
7. Updated property-specific-training.ts to use the unified logger
8. Updated material-specific-training.ts to use the unified logger
9. Updated material-specific-ocr.ts to use the unified logger

logger.ts

Removed mcp-integration.ts - Replaced with unified MCP client

mcp-integration.ts

Created a unified services export:

Created services/index.ts to export all unified services from the shared package

services/index.ts

Updated components to use unified services directly:

property-specific-training.ts

material-specific-training.ts

material-specific-ocr.ts

### Unified Services Export

The services/index.ts file exports all the unified services from the shared package for use throughout the ml package. It also provides a function to initialize all services.

services/index.ts

/\*\*  
 \* Unified Services Export  
 \*   
 \* This file exports all the unified services from the shared package  
 \* for use throughout the ml package.  
 \*/  
  
// Export auth service  
export {   
 auth,   
 initializeAuth,   
 User,   
 LoginCredentials,   
 RegisterCredentials,   
 AuthResult   
} from '@kai/shared';  
  
// Export API client  
export {   
 apiClient,   
 createApiClient,   
 ApiClientConfig,   
 ApiError,  
 BaseService,  
 ServiceConfig  
} from '@kai/shared';  
  
// Export MCP client  
export {   
 mcpClient,   
 createMCPClient   
} from '@kai/shared';  
  
// Export Supabase client  
export {   
 supabase   
} from '@kai/shared';  
  
// Export storage service  
export {   
 storage,   
 initializeStorage,   
 StorageProvider   
} from '@kai/shared';  
  
// Export logger  
export {   
 createLogger,   
 LogLevel   
} from '@kai/shared';  
  
// Export config  
export {   
 config   
} from '@kai/shared';  
  
/\*\*  
 \* Initialize all services  
 \*   
 \* This function initializes all the unified services.  
 \* It should be called early in the application lifecycle.  
 \*/  
export function initializeServices(): void {  
 // Initialize auth service  
 initializeAuth();  
  
 // Initialize storage service  
 initializeStorage();  
  
 // Log that services have been initialized  
 const logger = createLogger('Services');  
 logger.info('Unified services initialized');  
}  
  
/\*\*  
 \* Check if MCP server is enabled  
 \*   
 \* @returns Whether MCP server is enabled  
 \*/  
export function isMCPEnabled(): boolean {  
 return config.get('ml.useMcpServer');  
}  
  
/\*\*  
 \* Check if MCP server is available  
 \*   
 \* @returns Whether MCP server is available  
 \*/  
export async function checkMCPServerAvailability(): Promise<boolean> {  
 try {  
 const client = createMCPClient();  
 await client.checkHealth();  
 return true;  
 } catch (error) {  
 const logger = createLogger('MCP');  
 logger.error('MCP server health check failed', error as Error);  
 return false;  
 }  
}

/\*\*  
 \* Unified Services Export  
 \*   
 \* This file exports all the unified services from the shared package  
 \* for use throughout the ml package.  
 \*/  
  
// Export auth service  
export {   
 auth,   
 initializeAuth,   
 User,   
 LoginCredentials,   
 RegisterCredentials,   
 AuthResult   
} from '@kai/shared';  
  
// Export API client  
export {   
 apiClient,   
 createApiClient,   
 ApiClientConfig,   
 ApiError,  
 BaseService,  
 ServiceConfig  
} from '@kai/shared';  
  
// Export MCP client  
export {   
 mcpClient,   
 createMCPClient   
} from '@kai/shared';  
  
// Export Supabase client  
export {   
 supabase   
} from '@kai/shared';  
  
// Export storage service  
export {   
 storage,   
 initializeStorage,   
 StorageProvider   
} from '@kai/shared';  
  
// Export logger  
export {   
 createLogger,   
 LogLevel   
} from '@kai/shared';  
  
// Export config  
export {   
 config   
} from '@kai/shared';  
  
/\*\*  
 \* Initialize all services  
 \*   
 \* This function initializes all the unified services.  
 \* It should be called early in the application lifecycle.  
 \*/  
export function initializeServices(): void {  
 // Initialize auth service  
 initializeAuth();  
  
 // Initialize storage service  
 initializeStorage();  
  
 // Log that services have been initialized  
 const logger = createLogger('Services');  
 logger.info('Unified services initialized');  
}  
  
/\*\*  
 \* Check if MCP server is enabled  
 \*   
 \* @returns Whether MCP server is enabled  
 \*/  
export function isMCPEnabled(): boolean {  
 return config.get('ml.useMcpServer');  
}  
  
/\*\*  
 \* Check if MCP server is available  
 \*   
 \* @returns Whether MCP server is available  
 \*/  
export async function checkMCPServerAvailability(): Promise<boolean> {  
 try {  
 const client = createMCPClient();  
 await client.checkHealth();  
 return true;  
 } catch (error) {  
 const logger = createLogger('MCP');  
 logger.error('MCP server health check failed', error as Error);  
 return false;  
 }  
}

### ML Package Entry Point

The index.ts file has been updated to initialize the unified services before exporting any functionality. This ensures that all services are properly initialized before they are used.

index.ts

/\*\*  
 \* Main entry point for the ML package  
 \* This file exports the functions that will be used by the server package  
 \* to interact with the ML components  
 \*  
 \* The ML package provides functionality for:  
 \* - PDF processing and image extraction  
 \* - Material recognition using feature-based and ML-based approaches  
 \* - Vector embedding generation for similarity search  
 \* - Model training and evaluation  
 \* - Image segmentation for multiple tile detection  
 \* - Feedback loop for improving recognition over time  
 \* - Performance optimization for faster recognition  
 \* - Crawler data integration for training  
 \* - 3D reconstruction and visualization with Gaussian Splatting  
 \* - Improved text-to-3D generation  
 \*/  
  
// Initialize services  
import { initializeServices } from './services';  
  
// Initialize services on module load  
initializeServices();

/\*\*  
 \* Main entry point for the ML package  
 \* This file exports the functions that will be used by the server package  
 \* to interact with the ML components  
 \*  
 \* The ML package provides functionality for:  
 \* - PDF processing and image extraction  
 \* - Material recognition using feature-based and ML-based approaches  
 \* - Vector embedding generation for similarity search  
 \* - Model training and evaluation  
 \* - Image segmentation for multiple tile detection  
 \* - Feedback loop for improving recognition over time  
 \* - Performance optimization for faster recognition  
 \* - Crawler data integration for training  
 \* - 3D reconstruction and visualization with Gaussian Splatting  
 \* - Improved text-to-3D generation  
 \*/  
  
// Initialize services  
import { initializeServices } from './services';  
  
// Initialize services on module load  
initializeServices();

### Updated Components

The following components have been updated to use the unified services:

#### Property-Specific Training

The property-specific-training.ts file has been updated to use the unified logger from the shared package.

property-specific-training.ts

import \* as path from 'path';  
import \* as fs from 'fs';  
import { spawn } from 'child\_process';  
import { createLogger } from './services';  
  
const logger = createLogger('PropertySpecificTraining');

import \* as path from 'path';  
import \* as fs from 'fs';  
import { spawn } from 'child\_process';  
import { createLogger } from './services';  
  
const logger = createLogger('PropertySpecificTraining');

#### Material-Specific Training

The material-specific-training.ts file has been updated to use the unified logger from the shared package.

material-specific-training.ts

import \* as path from 'path';  
import \* as fs from 'fs';  
import { spawn } from 'child\_process';  
import { createLogger } from './services';  
  
const logger = createLogger('MaterialSpecificTraining');

import \* as path from 'path';  
import \* as fs from 'fs';  
import { spawn } from 'child\_process';  
import { createLogger } from './services';  
  
const logger = createLogger('MaterialSpecificTraining');

#### Material-Specific OCR

The material-specific-ocr.ts file has been updated to use the unified logger from the shared package.

material-specific-ocr.ts

import axios from 'axios';  
import { createLogger } from '../services';  
  
const logger = createLogger('MaterialSpecificOCR');

import axios from 'axios';  
import { createLogger } from '../services';  
  
const logger = createLogger('MaterialSpecificOCR');

### Benefits

The direct unified services architecture provides several benefits:

1. Simplified codebase: No more compatibility layers or adapter files
2. Reduced code duplication: Common functionality is implemented once in the shared package
3. Improved maintainability: Changes to common functionality only need to be made in one place
4. Consistent behavior: All parts of the application use the same implementation of common functionality
5. Type safety: The unified services provide type-safe interfaces for common operations
6. Extensibility: The provider pattern allows adding new implementations without changing client code

### Next Steps

The following steps are recommended to further improve the unified services architecture:

1. Add more storage providers (Google Cloud Storage, Azure Blob Storage, etc.)
2. Add more authentication providers (SAML, etc.)
3. Implement caching mechanisms for improved performance
4. Add more comprehensive monitoring and telemetry

# Model Improvement

Source: readme/model-improvement.md

---

## Model Improvement System

This document provides an overview of the Model Improvement System, which uses collected response quality data to continuously improve model performance through fine-tuning, error pattern analysis, and improvement suggestions.

### Table of Contents

1. Overview
2. System Components
3. Setup and Configuration
4. Usage
5. API Reference
6. Troubleshooting

### Overview

The Model Improvement System is designed to continuously improve model performance by:

1. Collecting User Feedback: Gathering real user feedback on model responses
2. Analyzing Error Patterns: Identifying common error types and trends
3. Fine-tuning Models: Automatically fine-tuning models based on feedback
4. Generating Improvement Suggestions: Providing actionable suggestions for model improvements

The system is fully integrated with the existing application infrastructure, including the MCP server, credit system, and database.

### System Components

#### 1. Feedback Collection

* ResponseFeedback Component: UI component for collecting user feedback
* ResponseMessage Component: Integration with chat UI
* Response Quality Service: Client-side service for sending feedback to the server

#### 2. Error Pattern Analysis

* Error Pattern Analysis Service: Server-side service for analyzing error patterns
* Error Trend Analysis: Tracking error trends over time
* Improvement Suggestion Generation: Generating actionable suggestions for model improvements

#### 3. Model Fine-tuning

* Feedback-Based Training Service: Server-side service for fine-tuning models based on feedback
* Fine-tuning Job Management: Creating, starting, and monitoring fine-tuning jobs
* Dataset Preparation: Automatically preparing datasets from problematic responses

#### 4. API Integration

* Response Quality API: Endpoints for recording and retrieving feedback
* Model Improvement API: Endpoints for fine-tuning and error pattern analysis
* MCP Integration: Integration with the MCP server for model fine-tuning

### Setup and Configuration

#### Prerequisites

* Supabase database
* MCP server with fine-tuning capabilities
* Credit system integration

#### Database Setup

Run the database migrations to create the required tables:

# From the project root  
cd packages/server  
yarn run-script run-migrations

# From the project root  
cd packages/server  
yarn run-script run-migrations

#### MCP Server Configuration

Ensure the MCP server supports the required endpoints:

# From the project root  
cd packages/server  
yarn run-script check-mcp-endpoints

# From the project root  
cd packages/server  
yarn run-script check-mcp-endpoints

#### MCP Server Integration

Ensure the MCP server is properly configured for model fine-tuning:

1. Add the MODEL\_TRAINING service key to the MCP service keys
2. Implement the required fine-tuning endpoints in the MCP server

MODEL\_TRAINING

#### Feedback Collection Integration

Integrate the feedback collection components with your chat UI:

import ResponseMessage from '../components/chat/ResponseMessage';  
  
// In your chat component  
<ResponseMessage  
 responseId="response-id"  
 modelId="model-id"  
 query="User query"  
 response="Model response"  
 timestamp={new Date()}  
 feedbackVariant="thumbs" // or "stars" or "full"  
/>

import ResponseMessage from '../components/chat/ResponseMessage';  
  
// In your chat component  
<ResponseMessage  
 responseId="response-id"  
 modelId="model-id"  
 query="User query"  
 response="Model response"  
 timestamp={new Date()}  
 feedbackVariant="thumbs" // or "stars" or "full"  
/>

#### Verification

Run the integration check script to verify that everything is properly set up:

# From the project root  
cd packages/server  
yarn run-script check-model-improvement-integration

# From the project root  
cd packages/server  
yarn run-script check-model-improvement-integration

### Usage

#### Collecting Feedback

Feedback is collected automatically when users interact with the ResponseFeedback component. The feedback is stored in the database and used for error pattern analysis and model fine-tuning.

#### Analyzing Error Patterns

Error patterns are analyzed automatically by the scheduled job that runs every Monday at 3:00 AM. You can also manually trigger the analysis:

# From the project root  
cd packages/server  
yarn run-script analyze-error-patterns

# From the project root  
cd packages/server  
yarn run-script analyze-error-patterns

#### Fine-tuning Models

Models are automatically fine-tuned when they meet the criteria defined in the fine-tuning trigger conditions. You can also manually trigger fine-tuning:

# From the project root  
cd packages/server  
yarn run-script fine-tune-model --model-id=<model-id>

# From the project root  
cd packages/server  
yarn run-script fine-tune-model --model-id=<model-id>

#### Viewing Results

You can view the results of the model improvement system in the admin panel:

1. Response Quality Panel: Shows quality metrics, error patterns, and problematic responses
2. Fine-tuning Jobs Panel: Shows fine-tuning jobs and their status
3. Improvement Suggestions Panel: Shows improvement suggestions for error patterns

### API Reference

#### Response Quality API

* GET /api/analytics/response-quality/metrics: Get response quality metrics
* GET /api/analytics/response-quality/problematic: Get problematic responses
* POST /api/analytics/response-quality/feedback: Record response feedback
* POST /api/analytics/response-quality/response: Record model response with feedback

GET /api/analytics/response-quality/metrics

GET /api/analytics/response-quality/problematic

POST /api/analytics/response-quality/feedback

POST /api/analytics/response-quality/response

#### Model Improvement API

* GET /api/analytics/model-improvement/fine-tuning/jobs: Get fine-tuning jobs
* GET /api/analytics/model-improvement/fine-tuning/jobs/:jobId: Get fine-tuning job by ID
* POST /api/analytics/model-improvement/fine-tuning/check: Check if a model should be fine-tuned
* POST /api/analytics/model-improvement/fine-tuning/jobs: Create a fine-tuning job
* POST /api/analytics/model-improvement/fine-tuning/jobs/:jobId/start: Start a fine-tuning job
* POST /api/analytics/model-improvement/fine-tuning/jobs/:jobId/cancel: Cancel a fine-tuning job
* GET /api/analytics/model-improvement/error-patterns: Analyze error patterns
* GET /api/analytics/model-improvement/error-trends: Get error trends
* POST /api/analytics/model-improvement/improvement-suggestions: Generate improvement suggestions

GET /api/analytics/model-improvement/fine-tuning/jobs

GET /api/analytics/model-improvement/fine-tuning/jobs/:jobId

POST /api/analytics/model-improvement/fine-tuning/check

POST /api/analytics/model-improvement/fine-tuning/jobs

POST /api/analytics/model-improvement/fine-tuning/jobs/:jobId/start

POST /api/analytics/model-improvement/fine-tuning/jobs/:jobId/cancel

GET /api/analytics/model-improvement/error-patterns

GET /api/analytics/model-improvement/error-trends

POST /api/analytics/model-improvement/improvement-suggestions

### Troubleshooting

#### Database Issues

If you encounter database issues, run the database check script:

# From the project root  
cd packages/server  
yarn run-script check-db-tables

# From the project root  
cd packages/server  
yarn run-script check-db-tables

If tables are missing, run the migrations:

# From the project root  
cd packages/server  
yarn run-script run-migrations

# From the project root  
cd packages/server  
yarn run-script run-migrations

#### MCP Server Issues

If you encounter MCP server issues, check if the required endpoints are supported:

# From the project root  
cd packages/server  
yarn run-script check-mcp-endpoints

# From the project root  
cd packages/server  
yarn run-script check-mcp-endpoints

If endpoints are missing, update the MCP server to support the required endpoints.

#### Credit System Issues

If you encounter credit system issues, check if the credit system is properly integrated:

# From the project root  
cd packages/server  
yarn run-script check-credit-integration

# From the project root  
cd packages/server  
yarn run-script check-credit-integration

If there are issues, update the credit system configuration.

#### Feedback Collection Issues

If you encounter feedback collection issues, check if the feedback components are properly integrated:

# From the project root  
cd packages/server  
yarn run-script check-feedback-integration

# From the project root  
cd packages/server  
yarn run-script check-feedback-integration

If there are issues, update the feedback component integration.

# Monitoring System

Source: readme/monitoring-system.md

---

## Monitoring System

Kai includes a comprehensive monitoring system that provides real-time insights into the health, performance, and operation of the platform. This system is designed to help administrators identify issues, track system performance, and ensure optimal operation.

### Features

#### System Health Monitoring

* Real-time Health Metrics: Track CPU usage, memory consumption, and service statuses
* Environment Variable Validation: Automatic validation of required environment variables
* Service Status: Monitor individual service health across the platform
* Rate Limit Statistics: Track API usage and rate limiting across different endpoints

#### Comprehensive Logging

* Centralized Log Collection: All system logs are collected in a central location
* Log Filtering: Filter logs by level, module, date range, and text content
* Error Distribution Analysis: Track error frequency by module to identify problem areas

#### Admin Dashboard

The monitoring system includes a dedicated admin dashboard that provides:

* System Health Visualization: Real-time charts and metrics for system health
* Log Explorer: Interactive interface for exploring and filtering logs
* Error Analysis: Visual breakdown of errors by module and time period
* Rate Limit Monitoring: Track API usage and rate limiting

### Architecture

The monitoring system consists of:

1. Backend Services: Collect metrics, logs, and health data
2. Admin API: Provides access to monitoring data through dedicated endpoints
3. Frontend Dashboard: Visualizes monitoring data for administrators

### Prometheus Integration

The KAI platform uses Prometheus for metrics collection, aggregation, and storage. Prometheus is deployed as part of the monitoring stack in the monitoring namespace.

monitoring

#### Key Components

* Prometheus Server: Collects and stores time-series metrics data
* Alert Manager: Handles alerts sent by Prometheus server
* Grafana: Provides visualization and dashboards for Prometheus metrics
* Prometheus Adapter: Exposes Prometheus metrics to Kubernetes for HPA

#### Metrics Collection

Services expose metrics through annotations in their Kubernetes manifests:

prometheus.io/scrape: "true"  
prometheus.io/port: "8080"  
prometheus.io/path: "/metrics"

prometheus.io/scrape: "true"  
prometheus.io/port: "8080"  
prometheus.io/path: "/metrics"

These annotations enable Prometheus to automatically discover and scrape metrics from the services.

#### Custom Metrics API

The platform uses the Prometheus Adapter to expose custom metrics to the Kubernetes API, enabling advanced autoscaling based on application-specific metrics:

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: prometheus-adapter-config  
 namespace: kai-ml  
data:  
 config.yaml: |  
 rules:  
 # API Request Rate Metrics  
 - seriesQuery: 'http\_requests\_total{kubernetes\_namespace!="",kubernetes\_pod\_name!=""}'  
 resources:  
 overrides:  
 kubernetes\_namespace: {resource: "namespace"}  
 kubernetes\_pod\_name: {resource: "pod"}  
 name:  
 matches: "^(.\*)\_total"  
 as: "${1}\_per\_second"  
 metricsQuery: 'sum(rate(<<.Series>>{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>)'  
  
 # Queue Depth Metrics for Coordinator  
 - seriesQuery: 'kai\_coordinator\_queue\_depth{kubernetes\_namespace!="",kubernetes\_pod\_name!=""}'  
 resources:  
 overrides:  
 kubernetes\_namespace: {resource: "namespace"}  
 kubernetes\_pod\_name: {resource: "pod"}  
 name:  
 matches: "kai\_coordinator\_queue\_depth"  
 as: "coordinator\_queue\_depth"  
 metricsQuery: 'sum(<<.Series>>{<<.LabelMatchers>>}) by (<<.GroupBy>>)'

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: prometheus-adapter-config  
 namespace: kai-ml  
data:  
 config.yaml: |  
 rules:  
 # API Request Rate Metrics  
 - seriesQuery: 'http\_requests\_total{kubernetes\_namespace!="",kubernetes\_pod\_name!=""}'  
 resources:  
 overrides:  
 kubernetes\_namespace: {resource: "namespace"}  
 kubernetes\_pod\_name: {resource: "pod"}  
 name:  
 matches: "^(.\*)\_total"  
 as: "${1}\_per\_second"  
 metricsQuery: 'sum(rate(<<.Series>>{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>)'  
  
 # Queue Depth Metrics for Coordinator  
 - seriesQuery: 'kai\_coordinator\_queue\_depth{kubernetes\_namespace!="",kubernetes\_pod\_name!=""}'  
 resources:  
 overrides:  
 kubernetes\_namespace: {resource: "namespace"}  
 kubernetes\_pod\_name: {resource: "pod"}  
 name:  
 matches: "kai\_coordinator\_queue\_depth"  
 as: "coordinator\_queue\_depth"  
 metricsQuery: 'sum(<<.Series>>{<<.LabelMatchers>>}) by (<<.GroupBy>>)'

#### Available Metrics

The platform exposes various metrics through the monitoring service:

* Workflow Metrics:
* workflow\_started\_total: Counter for started workflows
* workflow\_completed\_total: Counter for completed workflows
* workflow\_duration\_seconds: Histogram for workflow durations
* workflow\_error\_total: Counter for workflow errors
* Resource Metrics:
* workflow\_cpu\_usage\_cores: Gauge for CPU usage
* workflow\_memory\_usage\_bytes: Gauge for memory usage
* workflow\_gpu\_usage\_percent: Gauge for GPU utilization
* Coordinator Metrics:
* kai\_coordinator\_queue\_depth: Gauge for queue depth by priority
* kai\_coordinator\_active\_workflows: Gauge for active workflows by type
* kai\_coordinator\_workflow\_duration\_seconds: Histogram for workflow durations
* kai\_coordinator\_workflow\_completed\_total: Counter for completed workflows
* kai\_coordinator\_workflow\_error\_total: Counter for workflow errors
* kai\_coordinator\_resource\_utilization: Gauge for resource utilization
* Database Connection Metrics:
* kai\_supabase\_connection\_pool\_active: Gauge for active connections
* kai\_supabase\_connection\_pool\_idle: Gauge for idle connections
* kai\_supabase\_connection\_pool\_total: Gauge for total connections
* kai\_supabase\_connection\_pool\_utilization: Gauge for connection pool utilization
* kai\_supabase\_connection\_pool\_waiting\_acquires: Gauge for waiting connection acquires
* kai\_supabase\_connection\_pool\_acquire\_success\_rate: Gauge for connection acquisition success rate
* kai\_supabase\_connection\_pool\_average\_acquire\_time: Gauge for average connection acquisition time
* kai\_supabase\_connection\_pool\_connection\_errors: Gauge for connection errors
* Cache Metrics:
* workflow\_cache\_hit\_total: Counter for cache hits
* workflow\_stage\_duration\_seconds: Histogram for stage durations

workflow\_started\_total

workflow\_completed\_total

workflow\_duration\_seconds

workflow\_error\_total: Counter for workflow errors

workflow\_error\_total

Resource Metrics:

workflow\_cpu\_usage\_cores

workflow\_memory\_usage\_bytes

workflow\_gpu\_usage\_percent: Gauge for GPU utilization

workflow\_gpu\_usage\_percent

Coordinator Metrics:

kai\_coordinator\_queue\_depth

kai\_coordinator\_active\_workflows

kai\_coordinator\_workflow\_duration\_seconds

kai\_coordinator\_workflow\_completed\_total

kai\_coordinator\_workflow\_error\_total

kai\_coordinator\_resource\_utilization: Gauge for resource utilization

kai\_coordinator\_resource\_utilization

Database Connection Metrics:

kai\_supabase\_connection\_pool\_active

kai\_supabase\_connection\_pool\_idle

kai\_supabase\_connection\_pool\_total

kai\_supabase\_connection\_pool\_utilization

kai\_supabase\_connection\_pool\_waiting\_acquires

kai\_supabase\_connection\_pool\_acquire\_success\_rate

kai\_supabase\_connection\_pool\_average\_acquire\_time

kai\_supabase\_connection\_pool\_connection\_errors: Gauge for connection errors

kai\_supabase\_connection\_pool\_connection\_errors

Cache Metrics:

workflow\_cache\_hit\_total

workflow\_stage\_duration\_seconds

### Accessing Grafana

Grafana provides visualization of all metrics collected by Prometheus. Here's how to access and use Grafana:

#### Access Methods

##### Method 1: Domain Access (if configured)

If Ingress has been set up:

1. Navigate to https://grafana.yourdomain.com in your browser
2. You'll be presented with the Grafana login screen

https://grafana.yourdomain.com

##### Method 2: Port Forwarding

For direct access:

# Start port-forwarding to access Grafana UI locally  
kubectl port-forward svc/prometheus-grafana 3000:80 -n monitoring

# Start port-forwarding to access Grafana UI locally  
kubectl port-forward svc/prometheus-grafana 3000:80 -n monitoring

Then access Grafana at http://localhost:3000 in your browser.

http://localhost:3000

#### Login Credentials

* Username: admin
* Password: Set during installation

admin

If you don't know the password, retrieve it with:

kubectl get secret prometheus-grafana -n monitoring -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

kubectl get secret prometheus-grafana -n monitoring -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

#### Available Dashboards

The following pre-configured dashboards are available:

1. Kubernetes Dashboard:
2. Shows cluster-wide metrics
3. Navigate to Dashboards → Browse → Default → Kubernetes Dashboard
4. ML Workflows Dashboard:
5. Shows execution times and resource usage of ML pipelines
6. Navigate to Dashboards → Browse → Default → ML Workflows Dashboard
7. ML Processing Dashboard:
8. Shows metrics for different processing stages
9. Navigate to Dashboards → Browse → Default → ML Processing Dashboard
10. Supabase Connection Pool Dashboard:
11. Shows database connection pool metrics
12. Monitors connection counts, utilization, and performance
13. Tracks connection acquisition times and error rates
14. Navigate to Dashboards → Browse → Default → Supabase Connection Pool
15. Kubernetes HPA Metrics Dashboard:
16. Shows Horizontal Pod Autoscaler metrics
17. Monitors replica counts, scaling events, and custom metrics
18. Visualizes CPU/memory utilization and queue depths
19. Navigate to Dashboards → Browse → Default → Kubernetes HPA Metrics
20. Coordinator Service Dashboard:
21. Shows metrics for the Coordinator service
22. Monitors queue depths, workflow durations, and error rates
23. Tracks resource utilization and processing performance
24. Navigate to Dashboards → Browse → Default → Coordinator Service

Navigate to Dashboards → Browse → Default → Kubernetes Dashboard

ML Workflows Dashboard:

Navigate to Dashboards → Browse → Default → ML Workflows Dashboard

ML Processing Dashboard:

Navigate to Dashboards → Browse → Default → ML Processing Dashboard

Supabase Connection Pool Dashboard:

Navigate to Dashboards → Browse → Default → Supabase Connection Pool

Kubernetes HPA Metrics Dashboard:

Navigate to Dashboards → Browse → Default → Kubernetes HPA Metrics

Coordinator Service Dashboard:

#### Exploring Metrics

To explore specific metrics:

1. From the left menu, select "Explore"
2. Select "Prometheus" as the data source
3. Enter PromQL queries to retrieve specific metrics
4. Example queries:
5. rate(workflow\_completed\_total[5m]) - Workflow completion rate
6. avg(workflow\_duration\_seconds) by (type) - Average duration by workflow type
7. sum(workflow\_error\_total) by (type) - Total errors by workflow type

rate(workflow\_completed\_total[5m])

avg(workflow\_duration\_seconds) by (type)

sum(workflow\_error\_total) by (type)

#### Creating Custom Dashboards

You can create custom dashboards for specific monitoring needs:

1. Click the "+" icon in the left sidebar
2. Select "Dashboard"
3. Click "Add new panel"
4. Configure the panel with Prometheus queries and appropriate visualizations

#### Troubleshooting Grafana Access

If you're unable to access Grafana:

1. Check if pods are running: kubectl get pods -n monitoring
2. Verify services: kubectl get svc -n monitoring
3. Check ingress (if using domain access): kubectl get ingress -n monitoring
4. Check for port-forwarding issues

kubectl get pods -n monitoring

kubectl get svc -n monitoring

kubectl get ingress -n monitoring

### API Endpoints

#### Health Endpoints

##### Basic Health Check

GET /health

GET /health

Provides basic system health information including:  
- System status  
- Uptime information  
- Memory usage  
- Node.js version  
- Environment health status

This endpoint is public and does not require authentication, making it suitable for automated health checks from load balancers or monitoring services.

##### Detailed Health Check

GET /health/detailed

GET /health/detailed

Provides comprehensive system health data including:  
- Detailed system status  
- CPU and memory usage statistics  
- Component-by-component health status  
- Environment variable validation status

This endpoint requires authentication to protect sensitive system information.

#### Admin Monitoring API

##### Get System Logs

POST /api/admin/monitoring/logs

POST /api/admin/monitoring/logs

Retrieves system logs with filtering options:  
- Filter by log level (debug, info, warn, error)  
- Filter by module  
- Filter by date range  
- Full-text search within logs  
- Pagination support

##### Get Error Distribution

GET /api/admin/monitoring/errors

GET /api/admin/monitoring/errors

Retrieves error distribution by module over a specified time period.

##### Get Health Metrics

GET /api/admin/monitoring/health

GET /api/admin/monitoring/health

Retrieves detailed health metrics including CPU usage, memory utilization, service statuses, and rate limit statistics.

### Rate Limiting

The system includes a sophisticated rate limiting mechanism to prevent abuse and ensure stability:

* Default API Rate Limit: 100 requests per minute for general API endpoints
* Authentication Rate Limit: 20 requests per minute for authentication endpoints to prevent brute force attacks
* ML Processing Rate Limit: 10 requests per minute for resource-intensive ML operations
* Agent API Rate Limit: 30 requests per minute for AI agent interactions
* PDF Processing Rate Limit: 5 requests per 10 minutes for resource-intensive PDF processing

Rate limit statistics are tracked and visible in the monitoring dashboard.

### Environment Validation

The monitoring system includes a sophisticated environment variable validation mechanism:

* Requirement Levels: Variables can be marked as required, optional, development-only, or production-only
* Custom Validators: Each variable can have a custom validation function
* Health Reporting: Environment validation status is included in health checks

### Setup and Configuration

To enable all monitoring features, ensure the following:

1. Configure environment variables according to the validation rules
2. Ensure the logger is properly configured
3. Grant appropriate admin access to users who need monitoring capabilities

### Best Practices

1. Regular Monitoring: Check the monitoring dashboard regularly to identify potential issues
2. Alert Configuration: Set up alerts for critical error thresholds
3. Log Rotation: Configure log rotation to prevent storage issues
4. Permission Management: Restrict monitoring access to authorized administrators

### ML Training Monitoring Integration

The monitoring system integrates with the ML Training Monitoring System, providing specialized visualizations and controls for machine learning training processes:

* Training Metrics Visualization: Real-time charts showing loss, accuracy, and custom metrics
* Checkpoint Management: Interface for creating, comparing, and rolling back to model checkpoints
* Parameter Tuning: Controls for adjusting hyperparameters during training
* Training Job Control: Status monitoring and control for training jobs

For complete details on these capabilities, see the Training Monitoring System documentation.

# Moodboard Feature

Source: readme/moodboard-feature.md

---

## MoodBoard Feature

This document provides detailed information about the MoodBoard feature in the Kai application, which allows users to collect, organize, and share materials in customizable boards.

### Overview

The MoodBoard feature enables users to:  
- Create collections of materials they're interested in  
- Organize materials into themed boards  
- Toggle between grid and list views  
- Set board visibility (public or private)  
- Access boards through dedicated pages

### User Interface Components

#### Material Selection Modal

When browsing materials, users can add them to a MoodBoard by clicking the "Add to Board" button. This opens a side modal with the following features:  
- Material details display  
- Dropdown to select an existing board  
- Option to create a new board  
- Success/error feedback

#### MoodBoard Management

Users can manage their MoodBoards from their profile page:  
- Create new boards with custom names and descriptions  
- Set board visibility (public/private)  
- View all existing boards in a grid layout  
- Delete boards they no longer need

#### Board View Page

Each MoodBoard has a dedicated page at /board/:boardId or /:username/board/:boardId with:  
- Board header with name, description, and visibility status  
- Toggle between grid and list views  
- Material display in the selected view  
- Options to remove materials (for board owners)

/board/:boardId

/:username/board/:boardId

### Data Structure

The MoodBoard feature is built on two main data models:

#### MoodBoard

interface MoodBoard {  
 id: string;  
 name: string;  
 description?: string;  
 userId: string;  
 isPublic: boolean;  
 viewPreference: 'grid' | 'list';  
 createdAt: string;  
 updatedAt: string;  
}

interface MoodBoard {  
 id: string;  
 name: string;  
 description?: string;  
 userId: string;  
 isPublic: boolean;  
 viewPreference: 'grid' | 'list';  
 createdAt: string;  
 updatedAt: string;  
}

#### MoodBoardItem

interface MoodBoardItem {  
 id: string;  
 boardId: string;  
 materialId: string;  
 notes?: string;  
 position: number;  
 addedAt: string;  
}

interface MoodBoardItem {  
 id: string;  
 boardId: string;  
 materialId: string;  
 notes?: string;  
 position: number;  
 addedAt: string;  
}

### Implementation Details

#### Database Schema

The MoodBoard feature uses Supabase for data storage with the following tables:  
- moodboards - Stores board metadata  
- moodboard\_items - Stores the materials added to boards

moodboards

moodboard\_items

Row-level security policies ensure that:  
- Users can only view their own boards or public boards  
- Users can only modify their own boards  
- Users can only add/remove items from their own boards

#### API Endpoints

The following API endpoints are available for MoodBoard functionality:

##### Board Management

* GET /api/boards - Get all boards for the current user
* GET /api/boards/:boardId - Get a specific board
* POST /api/boards - Create a new board
* PUT /api/boards/:boardId - Update board details
* DELETE /api/boards/:boardId - Delete a board

GET /api/boards

GET /api/boards/:boardId

POST /api/boards

PUT /api/boards/:boardId

DELETE /api/boards/:boardId

##### Board Items Management

* GET /api/boards/:boardId/items - Get all items in a board
* POST /api/boards/:boardId/items - Add an item to a board
* PUT /api/boards/:boardId/items/:itemId - Update item details
* DELETE /api/boards/:boardId/items/:itemId - Remove an item from a board

GET /api/boards/:boardId/items

POST /api/boards/:boardId/items

PUT /api/boards/:boardId/items/:itemId

DELETE /api/boards/:boardId/items/:itemId

#### Client-Side Components

The feature is implemented using the following React components:  
- MaterialSideModal - Side modal for adding materials to boards  
- MaterialCard - Card component with "Add to Board" button  
- Profile page with MoodBoards section  
- BoardPage - Dedicated page for viewing a specific board

MaterialSideModal

MaterialCard

BoardPage

### Usage Examples

#### Adding a Material to a Board

// When a user clicks "Add to Board" on a material  
const handleAddToBoard = async (material) => {  
 // Open the side modal  
 setSelectedMaterial(material);  
 setIsSideModalOpen(true);  
};  
  
// Inside the modal, when adding to an existing board  
const addToExistingBoard = async (boardId, materialId) => {  
 await addMoodBoardItem({  
 boardId,  
 materialId  
 });  
};  
  
// Creating a new board and adding the material  
const createBoardAndAddMaterial = async (boardName, materialId) => {  
 // Create new board  
 const newBoard = await createMoodBoard({  
 name: boardName,  
 isPublic: false  
 });  
  
 // Add material to the new board  
 await addMoodBoardItem({  
 boardId: newBoard.id,  
 materialId  
 });  
};

// When a user clicks "Add to Board" on a material  
const handleAddToBoard = async (material) => {  
 // Open the side modal  
 setSelectedMaterial(material);  
 setIsSideModalOpen(true);  
};  
  
// Inside the modal, when adding to an existing board  
const addToExistingBoard = async (boardId, materialId) => {  
 await addMoodBoardItem({  
 boardId,  
 materialId  
 });  
};  
  
// Creating a new board and adding the material  
const createBoardAndAddMaterial = async (boardName, materialId) => {  
 // Create new board  
 const newBoard = await createMoodBoard({  
 name: boardName,  
 isPublic: false  
 });  
  
 // Add material to the new board  
 await addMoodBoardItem({  
 boardId: newBoard.id,  
 materialId  
 });  
};

#### Viewing a Board in Different Layouts

// Toggle between grid and list views  
const handleViewModeToggle = async (mode) => {  
 setViewMode(mode);  
  
 // Update user preference if they own the board  
 if (isOwner) {  
 await updateMoodBoard(boardId, {  
 viewPreference: mode  
 });  
 }  
};

// Toggle between grid and list views  
const handleViewModeToggle = async (mode) => {  
 setViewMode(mode);  
  
 // Update user preference if they own the board  
 if (isOwner) {  
 await updateMoodBoard(boardId, {  
 viewPreference: mode  
 });  
 }  
};

### Related Documentation

* Client HeroUI Integration - UI component system used for MoodBoard
* Supabase Setup Guide - Database setup for MoodBoard data
* API Reference - API endpoints for MoodBoard functionality

# Multilingual Property Dictionaries

Source: readme/multilingual-property-dictionaries.md

---

## Multilingual Property Dictionaries

This document describes the Multilingual Property Dictionaries feature, which enables cross-language search, identification, and localized property display in the KAI platform.

### Overview

The Multilingual Property Dictionaries feature provides a comprehensive system for managing property names and values in multiple languages. This enables:

1. Cross-language search and identification: Users can search for properties using terms in their preferred language.
2. Localized property display: Property names and values can be displayed in the user's preferred language.
3. Consistent terminology: Ensures consistent translation of technical terms across the platform.

### Architecture

The Multilingual Property Dictionaries feature consists of the following components:

#### Database Schema

* language\_codes: Stores supported languages with their codes, names, and native names.
* property\_name\_translations: Stores translations of property names in different languages.
* property\_value\_translations: Stores translations of property values in different languages.

#### API Endpoints

The following API endpoints are available for managing multilingual property dictionaries:

##### Language Management

* GET /api/multilingual/languages: Get all language codes.
* GET /api/multilingual/languages/:code: Get a language code by code.
* POST /api/multilingual/languages: Create a new language code (admin only).
* PUT /api/multilingual/languages/:code: Update a language code (admin only).

GET /api/multilingual/languages

GET /api/multilingual/languages/:code

POST /api/multilingual/languages

PUT /api/multilingual/languages/:code

##### Property Name Translations

* GET /api/multilingual/property-names: Get property name translations.
* GET /api/multilingual/property-names/:id: Get a property name translation by ID.
* POST /api/multilingual/property-names: Create a new property name translation.
* PUT /api/multilingual/property-names/:id: Update a property name translation.
* DELETE /api/multilingual/property-names/:id: Delete a property name translation.

GET /api/multilingual/property-names

GET /api/multilingual/property-names/:id

POST /api/multilingual/property-names

PUT /api/multilingual/property-names/:id

DELETE /api/multilingual/property-names/:id

##### Property Value Translations

* GET /api/multilingual/property-values: Get property value translations.
* GET /api/multilingual/property-values/:id: Get a property value translation by ID.
* POST /api/multilingual/property-values: Create a new property value translation.
* PUT /api/multilingual/property-values/:id: Update a property value translation.
* DELETE /api/multilingual/property-values/:id: Delete a property value translation.

GET /api/multilingual/property-values

GET /api/multilingual/property-values/:id

POST /api/multilingual/property-values

PUT /api/multilingual/property-values/:id

DELETE /api/multilingual/property-values/:id

##### Multilingual Property Operations

* GET /api/multilingual/properties: Get multilingual properties.
* GET /api/multilingual/property-values/:propertyName: Get multilingual property values.
* POST /api/multilingual/material-properties: Get multilingual material properties.
* POST /api/multilingual/translate-property-name: Translate a property name.
* POST /api/multilingual/translate-property-value: Translate a property value.

GET /api/multilingual/properties

GET /api/multilingual/property-values/:propertyName

POST /api/multilingual/material-properties

POST /api/multilingual/translate-property-name

POST /api/multilingual/translate-property-value

#### Client Components

The following client components are available for working with multilingual property dictionaries:

* LanguageSelector: Allows users to select their preferred language.
* MultilingualPropertyDisplay: Displays a property name and value in the user's selected language.
* MultilingualMaterialProperties: Displays all properties of a material in the user's selected language.

#### Admin Components

The following admin components are available for managing multilingual property dictionaries:

* MultilingualDictionaryManager: Allows administrators to manage property name and value translations.

### Usage

#### Setting Up Languages

Before using the multilingual property dictionaries, you need to set up the languages you want to support. This can be done through the admin interface or by using the API.

// Example: Adding a new language  
const newLanguage = {  
 code: 'fr',  
 name: 'French',  
 nativeName: 'Français',  
 isActive: true  
};  
  
await fetch('/api/multilingual/languages', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify(newLanguage)  
});

// Example: Adding a new language  
const newLanguage = {  
 code: 'fr',  
 name: 'French',  
 nativeName: 'Français',  
 isActive: true  
};  
  
await fetch('/api/multilingual/languages', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify(newLanguage)  
});

#### Adding Property Name Translations

Property name translations can be added through the admin interface or by using the API.

// Example: Adding a property name translation  
const propertyNameTranslation = {  
 propertyName: 'finish',  
 languageCode: 'fr',  
 translation: 'Finition',  
 description: 'Surface finish of the material'  
};  
  
await fetch('/api/multilingual/property-names', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify(propertyNameTranslation)  
});

// Example: Adding a property name translation  
const propertyNameTranslation = {  
 propertyName: 'finish',  
 languageCode: 'fr',  
 translation: 'Finition',  
 description: 'Surface finish of the material'  
};  
  
await fetch('/api/multilingual/property-names', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify(propertyNameTranslation)  
});

#### Adding Property Value Translations

Property value translations can be added through the admin interface or by using the API.

// Example: Adding a property value translation  
const propertyValueTranslation = {  
 propertyName: 'finish',  
 propertyValue: 'matte',  
 languageCode: 'fr',  
 translation: 'Mat',  
 description: 'Non-glossy finish'  
};  
  
await fetch('/api/multilingual/property-values', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify(propertyValueTranslation)  
});

// Example: Adding a property value translation  
const propertyValueTranslation = {  
 propertyName: 'finish',  
 propertyValue: 'matte',  
 languageCode: 'fr',  
 translation: 'Mat',  
 description: 'Non-glossy finish'  
};  
  
await fetch('/api/multilingual/property-values', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify(propertyValueTranslation)  
});

#### Displaying Properties in the User's Language

The MultilingualPropertyDisplay component can be used to display properties in the user's selected language.

MultilingualPropertyDisplay

import MultilingualPropertyDisplay from '../components/multilingual/MultilingualPropertyDisplay';  
  
// Example: Displaying a property in the user's language  
<MultilingualPropertyDisplay  
 propertyName="finish"  
 propertyValue="matte"  
 variant="label"  
 showOriginal={true}  
/>

import MultilingualPropertyDisplay from '../components/multilingual/MultilingualPropertyDisplay';  
  
// Example: Displaying a property in the user's language  
<MultilingualPropertyDisplay  
 propertyName="finish"  
 propertyValue="matte"  
 variant="label"  
 showOriginal={true}  
/>

#### Displaying All Material Properties in the User's Language

The MultilingualMaterialProperties component can be used to display all properties of a material in the user's selected language.

MultilingualMaterialProperties

import MultilingualMaterialProperties from '../components/multilingual/MultilingualMaterialProperties';  
  
// Example: Displaying all material properties in the user's language  
<MultilingualMaterialProperties  
 materialId="123"  
 properties={{  
 finish: 'matte',  
 color: 'white',  
 size: '60x60'  
 }}  
 showOriginal={true}  
/>

import MultilingualMaterialProperties from '../components/multilingual/MultilingualMaterialProperties';  
  
// Example: Displaying all material properties in the user's language  
<MultilingualMaterialProperties  
 materialId="123"  
 properties={{  
 finish: 'matte',  
 color: 'white',  
 size: '60x60'  
 }}  
 showOriginal={true}  
/>

#### Changing the User's Language

The LanguageSelector component can be used to allow users to change their preferred language.

LanguageSelector

import LanguageSelector from '../components/language/LanguageSelector';  
  
// Example: Adding a language selector to the header  
<LanguageSelector variant="dropdown" />

import LanguageSelector from '../components/language/LanguageSelector';  
  
// Example: Adding a language selector to the header  
<LanguageSelector variant="dropdown" />

### Integration with Other Features

#### Search Integration

The multilingual property dictionaries are integrated with the search system to enable cross-language search. When a user searches for a property in their preferred language, the search system will also look for the corresponding property in other languages.

#### Material Metadata Panel Integration

The multilingual property dictionaries are integrated with the Material Metadata Panel to display property names and values in the user's preferred language.

#### Property Relationship Graph Integration

The multilingual property dictionaries are integrated with the Property Relationship Graph to display relationship names and values in the user's preferred language.

### Best Practices

#### Adding New Properties

When adding new properties to the system, make sure to add translations for all supported languages. This ensures a consistent user experience across languages.

#### Updating Translations

When updating translations, make sure to update them for all supported languages. This ensures that users of all languages have access to the latest information.

#### Testing Translations

Test the application with different language settings to ensure that all translations are displayed correctly and that the application functions as expected in all languages.

### Conclusion

The Multilingual Property Dictionaries feature provides a comprehensive system for managing property names and values in multiple languages. This enables cross-language search, identification, and localized property display, enhancing the user experience for non-English speakers and ensuring consistent terminology across the platform.

# Multimodal Pattern Recognition

Source: readme/multimodal-pattern-recognition.md

---

## MultiModal Pattern Recognition

The MultiModal Pattern Recognition system is an advanced machine learning framework that bridges visual patterns and textual specifications for material analysis. It enables the platform to understand and map relationships between visual material characteristics and their corresponding textual descriptions.

### Overview

This system implements:

1. Transformer-based Architecture - Deep neural networks that process both visual and textual data
2. Cross-Modal Attention - Mechanisms that correlate visual features with text descriptions
3. Contrastive Learning - Techniques for modeling relationships between different modalities
4. SVD-based Feature Extraction - Advanced texture feature representation
5. Comprehensive Training Pipeline - End-to-end training for multimodal learning

The system serves as a critical component for high-fidelity material recognition and specification matching, enabling search and retrieval based on cross-modal queries.

### Architecture

The MultiModal Pattern Recognition system consists of several key components:

MultiModal Pattern Recognition  
├── MultiModalPatternRecognizer (Main Class)  
│ ├── Vision Encoder (ViT, CLIP)  
│ ├── Text Encoder (BERT, CLIP)  
│ ├── Cross-Modal Attention Mechanism  
│ └── Contrastive Learning Framework  
├── MultiModalTransformer  
│ ├── Vision-to-Text Attention  
│ ├── Text-to-Vision Attention  
│ ├── Feed-Forward Networks  
│ └── Joint Representation Layer  
├── MultiModalDataset  
│ ├── Image-Text Pair Loader  
│ └── Preprocessing Pipeline  
└── Utilities  
 ├── SVD-based Feature Extraction  
 ├── Gram Matrix Computation  
 └── Similarity Calculation

MultiModal Pattern Recognition  
├── MultiModalPatternRecognizer (Main Class)  
│ ├── Vision Encoder (ViT, CLIP)  
│ ├── Text Encoder (BERT, CLIP)  
│ ├── Cross-Modal Attention Mechanism  
│ └── Contrastive Learning Framework  
├── MultiModalTransformer  
│ ├── Vision-to-Text Attention  
│ ├── Text-to-Vision Attention  
│ ├── Feed-Forward Networks  
│ └── Joint Representation Layer  
├── MultiModalDataset  
│ ├── Image-Text Pair Loader  
│ └── Preprocessing Pipeline  
└── Utilities  
 ├── SVD-based Feature Extraction  
 ├── Gram Matrix Computation  
 └── Similarity Calculation

### Key Capabilities

#### Cross-Modal Understanding

The system establishes a shared semantic space between visual patterns and textual specifications, allowing:

* Pattern-to-Text Matching: Find appropriate specifications for a given visual pattern
* Text-to-Pattern Matching: Locate visual patterns that match a textual description
* Similarity Quantification: Measure how well a pattern matches a description

# Calculate similarity between an image and multiple text descriptions  
similarities = recognizer.compute\_similarity(  
 image\_path="path/to/pattern.jpg",  
 texts=[  
 "Geometric pattern with repeating squares",  
 "Floral pattern with interlacing vines",  
 "Abstract pattern with irregular shapes"  
 ]  
)  
# Returns similarity scores for each description

# Calculate similarity between an image and multiple text descriptions  
similarities = recognizer.compute\_similarity(  
 image\_path="path/to/pattern.jpg",  
 texts=[  
 "Geometric pattern with repeating squares",  
 "Floral pattern with interlacing vines",  
 "Abstract pattern with irregular shapes"  
 ]  
)  
# Returns similarity scores for each description

#### Multimodal Feature Fusion

The system fuses features from different modalities through:

* Cross-Attention Mechanisms: Allow each modality to attend to relevant parts of the other
* Joint Embedding Space: Projects features from both modalities into a unified space
* Contextual Enhancement: Enriches features with information from the other modality

This fusion enables sophisticated understanding of the relationships between patterns and their descriptions.

#### Pattern Classification

The system can classify visual patterns into predefined categories with high accuracy:

# Classify a pattern image into predefined categories  
results = recognizer.classify\_pattern(  
 image="path/to/pattern.jpg",  
 pattern\_classes=[  
 "geometric", "floral", "stripes",   
 "polka dots", "chevron", "abstract"  
 ]  
)  
# Returns a list of (pattern\_class, confidence) tuples

# Classify a pattern image into predefined categories  
results = recognizer.classify\_pattern(  
 image="path/to/pattern.jpg",  
 pattern\_classes=[  
 "geometric", "floral", "stripes",   
 "polka dots", "chevron", "abstract"  
 ]  
)  
# Returns a list of (pattern\_class, confidence) tuples

#### Specification Extraction

Beyond classification, the system can extract detailed specifications from pattern images:

# Extract specifications from a pattern image  
specifications = recognizer.extract\_specifications(  
 image="path/to/pattern.jpg",  
 specification\_templates=[  
 "Type: ceramic, Material: porcelain, Pattern: geometric",  
 "Type: textile, Material: cotton, Pattern: floral",  
 "Type: stone, Material: marble, Pattern: veined"  
 ]  
)  
# Returns a dictionary of extracted specifications

# Extract specifications from a pattern image  
specifications = recognizer.extract\_specifications(  
 image="path/to/pattern.jpg",  
 specification\_templates=[  
 "Type: ceramic, Material: porcelain, Pattern: geometric",  
 "Type: textile, Material: cotton, Pattern: floral",  
 "Type: stone, Material: marble, Pattern: veined"  
 ]  
)  
# Returns a dictionary of extracted specifications

#### Relationship Mapping

The system can discover complex relationships between patterns and specifications:

# Find relationships between a pattern and specifications  
relationships = recognizer.find\_pattern\_specification\_relationships(  
 image="path/to/pattern.jpg",  
 specifications=[  
 "Suitable for indoor use",  
 "Water-resistant",  
 "Requires special cleaning",  
 "UV-resistant"  
 ]  
)  
# Returns a dictionary mapping specifications to relationship scores

# Find relationships between a pattern and specifications  
relationships = recognizer.find\_pattern\_specification\_relationships(  
 image="path/to/pattern.jpg",  
 specifications=[  
 "Suitable for indoor use",  
 "Water-resistant",  
 "Requires special cleaning",  
 "UV-resistant"  
 ]  
)  
# Returns a dictionary mapping specifications to relationship scores

### Technical Implementation

#### Model Architecture

The MultiModal Pattern Recognition system uses a dual-encoder architecture with cross-attention:

1. Vision Encoder: Processes images using Vision Transformer (ViT) or CLIP models
2. Text Encoder: Processes text using BERT or CLIP models
3. Cross-Attention Modules: Connect the two modalities
4. Projection Layers: Map features to a shared embedding space

The architecture can be configured with different base models and parameters:

# Create a model with custom encoders  
recognizer = MultiModalPatternRecognizer(  
 vision\_encoder="vit-base-patch16-224",  
 text\_encoder="bert-base-uncased",  
 use\_pretrained=True,  
 embedding\_dim=768,  
 device="cuda"  
)  
  
# Or use CLIP for both vision and text  
recognizer = create\_multimodal\_recognizer(  
 use\_clip=True,  
 cache\_dir="./model\_cache"  
)

# Create a model with custom encoders  
recognizer = MultiModalPatternRecognizer(  
 vision\_encoder="vit-base-patch16-224",  
 text\_encoder="bert-base-uncased",  
 use\_pretrained=True,  
 embedding\_dim=768,  
 device="cuda"  
)  
  
# Or use CLIP for both vision and text  
recognizer = create\_multimodal\_recognizer(  
 use\_clip=True,  
 cache\_dir="./model\_cache"  
)

#### Cross-Modal Attention

The cross-modal attention mechanism allows each modality to focus on relevant aspects of the other:

1. Vision attended by Text: Updates visual features based on textual descriptions
2. Text attended by Vision: Updates textual features based on visual patterns
3. Multi-Head Attention: Processes different aspects of the relationship in parallel

This bidirectional attention flow enables rich feature interaction and contextual understanding.

#### Contrastive Learning

The system uses contrastive learning to align corresponding image-text pairs:

1. Positive Pairs: Images and their matching descriptions are pulled together in embedding space
2. Negative Pairs: Images and unrelated descriptions are pushed apart
3. Temperature Scaling: Controls the sharpness of similarity distribution

# Contrastive loss computation  
temperature = 0.07  
logits = similarity / temperature  
targets = torch.arange(batch\_size, device=logits.device)  
loss = F.cross\_entropy(logits, targets)

# Contrastive loss computation  
temperature = 0.07  
logits = similarity / temperature  
targets = torch.arange(batch\_size, device=logits.device)  
loss = F.cross\_entropy(logits, targets)

#### SVD-based Texture Features

For enhanced texture representation, the system implements SVD-based feature extraction:

1. Patch Extraction: Divides the image into patches
2. Singular Value Decomposition: Computes SVD for each patch
3. Weighted Features: Weights singular vectors by singular values
4. Feature Aggregation: Combines features across patches

This approach captures essential texture characteristics that standard CNNs might miss.

### Integration with Existing Systems

The MultiModal Pattern Recognition system integrates with several other components:

#### RAG System Integration

Enhances the Retrieval-Augmented Generation system by:

* Providing multimodal embeddings for the vector database
* Enabling cross-modal search queries
* Enriching context assembly with pattern-specification relationships

#### Material Recognition Pipeline

Supplements the material recognition pipeline with:

* Pattern-specific feature extraction
* Multimodal understanding of material properties
* Relationship modeling between visual patterns and technical specifications

#### Knowledge Base Enhancement

Improves the knowledge base by:

* Automatically extracting pattern-specification relationships
* Validating existing pattern classifications
* Suggesting new pattern categories based on visual-textual clusters

### Training Process

The system supports end-to-end training with:

1. Multimodal Dataset: Image-text pairs organized by material type
2. Contrastive Loss: Aligns corresponding image-text pairs
3. Cross-Attention Optimization: Learns optimal attention patterns
4. Feature Extraction Tuning: Fine-tunes feature extraction for domain-specific needs

# Create training dataset  
train\_dataset = MultiModalDataset(  
 data\_file="material\_patterns.json",  
 image\_processor=image\_processor,  
 tokenizer=tokenizer,  
 image\_root\_dir="./images",  
 max\_text\_length=128  
)  
  
# Train the model  
recognizer.train(  
 train\_dataset=train\_dataset,  
 validation\_dataset=val\_dataset,  
 num\_epochs=10,  
 batch\_size=16,  
 learning\_rate=2e-5,  
 weight\_decay=0.01,  
 output\_dir="./model\_checkpoints",  
 save\_every=1  
)

# Create training dataset  
train\_dataset = MultiModalDataset(  
 data\_file="material\_patterns.json",  
 image\_processor=image\_processor,  
 tokenizer=tokenizer,  
 image\_root\_dir="./images",  
 max\_text\_length=128  
)  
  
# Train the model  
recognizer.train(  
 train\_dataset=train\_dataset,  
 validation\_dataset=val\_dataset,  
 num\_epochs=10,  
 batch\_size=16,  
 learning\_rate=2e-5,  
 weight\_decay=0.01,  
 output\_dir="./model\_checkpoints",  
 save\_every=1  
)

### Model Deployment

The system supports various deployment options:

#### ONNX Export

For efficient deployment, models can be exported to ONNX format:

# Export model to ONNX format  
success = recognizer.export\_to\_onnx(  
 path="pattern\_recognizer.onnx",  
 input\_size=(224, 224)  
)

# Export model to ONNX format  
success = recognizer.export\_to\_onnx(  
 path="pattern\_recognizer.onnx",  
 input\_size=(224, 224)  
)

#### Batch Processing

For efficient processing of multiple images:

# Process multiple images in batch  
results = batch\_processor.process\_images(  
 image\_paths=["img1.jpg", "img2.jpg", "img3.jpg"],  
 text\_queries=["geometric pattern", "floral design"]  
)

# Process multiple images in batch  
results = batch\_processor.process\_images(  
 image\_paths=["img1.jpg", "img2.jpg", "img3.jpg"],  
 text\_queries=["geometric pattern", "floral design"]  
)

#### API Integration

The system exposes a comprehensive API for integration:

# Generate multimodal embedding  
embedding = generate\_multimodal\_embedding(  
 image\_path="pattern.jpg",  
 specification="Ceramic tile with geometric pattern",  
 model\_path="./models/pattern\_recognizer.pt",  
 use\_clip=True  
)

# Generate multimodal embedding  
embedding = generate\_multimodal\_embedding(  
 image\_path="pattern.jpg",  
 specification="Ceramic tile with geometric pattern",  
 model\_path="./models/pattern\_recognizer.pt",  
 use\_clip=True  
)

### Performance Considerations

#### Hardware Recommendations

For optimal performance:

* GPU: NVIDIA GPU with CUDA support recommended for inference and required for training
* Memory: 8GB+ GPU memory for training, 4GB+ for inference
* Storage: SSD recommended for model loading and dataset access

#### Optimization Techniques

The system implements several optimizations:

* Caching: Model weights and processed inputs are cached
* Batching: Processes multiple inputs simultaneously
* Mixed Precision: Uses FP16 where appropriate for faster computation
* Lazy Loading: Loads components on-demand to reduce memory footprint

#### Scaling

For large-scale deployments:

* Distributed Training: Support for multi-GPU and multi-node training
* Inference Servers: Integration with TorchServe or ONNX Runtime
* Load Balancing: Distributes requests across multiple inference instances

### Extensibility

The system is designed for easy extension:

#### Custom Encoders

Support for different vision and text encoders:

# Use custom encoders  
custom\_recognizer = MultiModalPatternRecognizer(  
 vision\_encoder="facebook/deit-base-patch16-224",  
 text\_encoder="roberta-base",  
 embedding\_dim=768  
)

# Use custom encoders  
custom\_recognizer = MultiModalPatternRecognizer(  
 vision\_encoder="facebook/deit-base-patch16-224",  
 text\_encoder="roberta-base",  
 embedding\_dim=768  
)

#### Domain Adaptation

The system can be adapted to specific material domains:

1. Fine-Tuning: Train on domain-specific datasets
2. Feature Engineering: Add domain-specific feature extractors
3. Loss Customization: Modify loss functions for domain requirements

#### Multimodal Fusion Methods

Alternative fusion methods can be implemented:

* Early Fusion: Combine features at early stages
* Late Fusion: Combine predictions from separate models
* Hybrid Fusion: Mix of early and late fusion approaches

### API Reference

#### MultiModalPatternRecognizer

class MultiModalPatternRecognizer:  
 def \_\_init\_\_(self,   
 model\_path: Optional[str] = None,  
 vision\_encoder: str = "vit-base-patch16-224",  
 text\_encoder: str = "bert-base-uncased",  
 use\_pretrained: bool = True,  
 embedding\_dim: int = 768,  
 device: Optional[str] = None,  
 cache\_dir: Optional[str] = None):  
 """Initialize the multimodal pattern recognizer"""  
  
 def encode\_image(self, image: Union[str, np.ndarray]) -> np.ndarray:  
 """Encode an image into a feature vector"""  
  
 def encode\_text(self, text: str) -> np.ndarray:  
 """Encode text into a feature vector"""  
  
 def compute\_similarity(self, image: Union[str, np.ndarray],   
 texts: List[str]) -> List[float]:  
 """Compute similarity between an image and multiple texts"""  
  
 def classify\_pattern(self, image: Union[str, np.ndarray],   
 pattern\_classes: List[str]) -> List[Tuple[str, float]]:  
 """Classify a pattern image into predefined pattern classes"""  
  
 def extract\_specifications(self, image: Union[str, np.ndarray],   
 specification\_templates: List[str]) -> Dict[str, Any]:  
 """Extract specifications from a pattern image"""  
  
 def find\_pattern\_specification\_relationships(self, image: Union[str, np.ndarray],  
 specifications: List[str]) -> Dict[str, float]:  
 """Find relationships between a pattern image and textual specifications"""  
  
 def train(self, train\_dataset: 'MultiModalDataset',  
 validation\_dataset: Optional['MultiModalDataset'] = None,  
 num\_epochs: int = 10,  
 batch\_size: int = 16,  
 learning\_rate: float = 2e-5,  
 weight\_decay: float = 0.01,  
 warmup\_steps: int = 0,  
 output\_dir: Optional[str] = None,  
 save\_every: int = 1):  
 """Train the model on a dataset of image-text pairs"""  
  
 def save\_model(self, save\_path: str, save\_format: str = "pytorch"):  
 """Save the model to disk"""  
  
 def export\_to\_onnx(self, path: str, input\_size: Tuple[int, int] = (224, 224)) -> bool:  
 """Export model to ONNX format"""

class MultiModalPatternRecognizer:  
 def \_\_init\_\_(self,   
 model\_path: Optional[str] = None,  
 vision\_encoder: str = "vit-base-patch16-224",  
 text\_encoder: str = "bert-base-uncased",  
 use\_pretrained: bool = True,  
 embedding\_dim: int = 768,  
 device: Optional[str] = None,  
 cache\_dir: Optional[str] = None):  
 """Initialize the multimodal pattern recognizer"""  
  
 def encode\_image(self, image: Union[str, np.ndarray]) -> np.ndarray:  
 """Encode an image into a feature vector"""  
  
 def encode\_text(self, text: str) -> np.ndarray:  
 """Encode text into a feature vector"""  
  
 def compute\_similarity(self, image: Union[str, np.ndarray],   
 texts: List[str]) -> List[float]:  
 """Compute similarity between an image and multiple texts"""  
  
 def classify\_pattern(self, image: Union[str, np.ndarray],   
 pattern\_classes: List[str]) -> List[Tuple[str, float]]:  
 """Classify a pattern image into predefined pattern classes"""  
  
 def extract\_specifications(self, image: Union[str, np.ndarray],   
 specification\_templates: List[str]) -> Dict[str, Any]:  
 """Extract specifications from a pattern image"""  
  
 def find\_pattern\_specification\_relationships(self, image: Union[str, np.ndarray],  
 specifications: List[str]) -> Dict[str, float]:  
 """Find relationships between a pattern image and textual specifications"""  
  
 def train(self, train\_dataset: 'MultiModalDataset',  
 validation\_dataset: Optional['MultiModalDataset'] = None,  
 num\_epochs: int = 10,  
 batch\_size: int = 16,  
 learning\_rate: float = 2e-5,  
 weight\_decay: float = 0.01,  
 warmup\_steps: int = 0,  
 output\_dir: Optional[str] = None,  
 save\_every: int = 1):  
 """Train the model on a dataset of image-text pairs"""  
  
 def save\_model(self, save\_path: str, save\_format: str = "pytorch"):  
 """Save the model to disk"""  
  
 def export\_to\_onnx(self, path: str, input\_size: Tuple[int, int] = (224, 224)) -> bool:  
 """Export model to ONNX format"""

#### Helper Functions

def create\_multimodal\_recognizer(  
 model\_path: Optional[str] = None,  
 vision\_encoder: str = "vit-base-patch16-224",  
 text\_encoder: str = "bert-base-uncased",  
 use\_clip: bool = True,  
 embedding\_dim: int = 768,  
 device: Optional[str] = None,  
 cache\_dir: Optional[str] = None) -> MultiModalPatternRecognizer:  
 """Create a multimodal pattern recognizer with default settings"""  
  
def generate\_multimodal\_embedding(image\_path: str,   
 specification: str,  
 model\_path: Optional[str] = None,  
 use\_clip: bool = True,  
 cache\_dir: Optional[str] = None) -> Dict[str, Any]:  
 """Generate multimodal embedding for an image-specification pair"""

def create\_multimodal\_recognizer(  
 model\_path: Optional[str] = None,  
 vision\_encoder: str = "vit-base-patch16-224",  
 text\_encoder: str = "bert-base-uncased",  
 use\_clip: bool = True,  
 embedding\_dim: int = 768,  
 device: Optional[str] = None,  
 cache\_dir: Optional[str] = None) -> MultiModalPatternRecognizer:  
 """Create a multimodal pattern recognizer with default settings"""  
  
def generate\_multimodal\_embedding(image\_path: str,   
 specification: str,  
 model\_path: Optional[str] = None,  
 use\_clip: bool = True,  
 cache\_dir: Optional[str] = None) -> Dict[str, Any]:  
 """Generate multimodal embedding for an image-specification pair"""

### Use Cases

#### Pattern Search and Classification

The system enables searching for patterns based on text descriptions:

# Find patterns matching a description  
matching\_patterns = pattern\_search\_service.find\_patterns(  
 description="Geometric pattern with hexagonal tiles",  
 material\_type="ceramic",  
 min\_similarity=0.7,  
 max\_results=10  
)

# Find patterns matching a description  
matching\_patterns = pattern\_search\_service.find\_patterns(  
 description="Geometric pattern with hexagonal tiles",  
 material\_type="ceramic",  
 min\_similarity=0.7,  
 max\_results=10  
)

#### Specification Extraction and Validation

The system can extract and validate specifications from pattern images:

# Extract and validate specifications  
extracted\_specs = specification\_service.extract\_and\_validate(  
 image\_path="new\_pattern.jpg",  
 expected\_material\_type="fabric"  
)

# Extract and validate specifications  
extracted\_specs = specification\_service.extract\_and\_validate(  
 image\_path="new\_pattern.jpg",  
 expected\_material\_type="fabric"  
)

#### Pattern-Specification Mapping

The system can automatically map patterns to appropriate specifications:

# Map pattern to specifications  
suggested\_specs = mapping\_service.suggest\_specifications(  
 image\_path="pattern\_sample.jpg",  
 confidence\_threshold=0.8  
)

# Map pattern to specifications  
suggested\_specs = mapping\_service.suggest\_specifications(  
 image\_path="pattern\_sample.jpg",  
 confidence\_threshold=0.8  
)

#### Multimodal Catalog Enhancement

The system can enhance product catalogs with pattern-specification relationships:

# Enhance catalog entries with pattern analysis  
enhanced\_catalog = catalog\_service.enhance\_with\_pattern\_analysis(  
 catalog\_entries=original\_catalog,  
 image\_field="product\_image",  
 description\_field="product\_description"  
)

# Enhance catalog entries with pattern analysis  
enhanced\_catalog = catalog\_service.enhance\_with\_pattern\_analysis(  
 catalog\_entries=original\_catalog,  
 image\_field="product\_image",  
 description\_field="product\_description"  
)

### Future Directions

The MultiModal Pattern Recognition system will continue to evolve with:

1. Improved Zero-Shot Capabilities: Better performance on unseen pattern types
2. Modal-Specific Pre-training: Specialized pre-training for material patterns
3. Enhanced Attention Mechanisms: More sophisticated cross-modal attention
4. Neural Architecture Search: Automated architecture optimization
5. Few-Shot Learning: Learning from limited examples of new pattern types

# Network Access Control

Source: readme/network-access-control.md

---

## Network Access Control System

### Overview

The Network Access Control system in the Kai platform provides a robust security layer that restricts sensitive operations to internal networks only. This document explains the concepts, configuration options, implementation details, and best practices for the system.

### Table of Contents

1. Core Concepts
2. Administrator Configuration
3. Admin Panel Interface
4. Internal Networks Management
5. API Endpoint Access Control
6. Rate Limiting Configuration
7. Technical Implementation
8. Database Schema
9. Network Source Detection
10. Authorization Middleware
11. Rate Limiting Implementation
12. Usage Examples
13. Best Practices
14. Developer Guidelines
15. Troubleshooting
16. Security Considerations
17. API Reference

### Core Concepts

#### Access Types

Endpoints in the Kai API are classified into three main access types:

1. ANY / EXTERNAL\_ALLOWED: Accessible from both internal and external networks
2. Example: User authentication, material search, recognition
3. Standard user-facing functionality
4. INTERNAL\_ONLY: Restricted to requests originating from authorized internal networks
5. Example: Deleting analytics data, system backup operations, configuration updates
6. Provides protection for sensitive administrative functions
7. EXTERNAL\_ONLY: Only accessible from external networks (rare)
8. Example: Special public-only interfaces

Standard user-facing functionality

INTERNAL\_ONLY: Restricted to requests originating from authorized internal networks

Provides protection for sensitive administrative functions

EXTERNAL\_ONLY: Only accessible from external networks (rare)

#### Network Classification

Requests are classified as internal or external based on the source IP address:

* Internal: IP addresses within configured CIDR ranges (e.g., corporate networks, VPNs)
* External: All other IP addresses (public internet)

### Administrator Configuration

#### 100% Dynamic Configuration

Important: No API endpoints have hardcoded access restrictions in the codebase. All access controls are fully configurable through the admin interface, including:

* Operations that might seem sensitive (like "update analytics", "clear data", or "system settings")
* Admin operations that would typically be internal-only
* Public-facing endpoints that need wide accessibility

The classifications shown in the API documentation represent recommended default settings only and can be modified at any time through the admin interface.

#### Admin Panel Interface

The admin panel provides a user-friendly interface for managing API access control. To access the admin panel, navigate to /admin/network-access in the application.

/admin/network-access

The admin panel consists of three main sections:

1. Internal Networks: Define CIDR ranges that should be considered internal networks
2. API Endpoint Access Control: Configure which endpoints can be accessed from internal or external networks
3. Rate Limits: Set default and custom rate limits for different networks

#### Internal Networks Management

##### What Are Internal Networks?

Internal networks are IP addresses or CIDR ranges that are considered trusted. Requests originating from these networks can access endpoints marked as "internal-only". Typical internal networks include:

* Corporate office networks
* VPN connections
* Development environments
* Cloud infrastructure in the same private network

##### Adding Internal Networks

1. Navigate to Settings → Network Access
2. In the Internal Networks section, enter a CIDR range (e.g., 10.0.0.0/8)
3. Provide an optional description for reference
4. Click Add
5. The network will appear in the list of internal networks

10.0.0.0/8

##### Example Internal Networks

Table content:

Network | Description

127.0.0.1/8 | Localhost

10.0.0.0/8 | Private network class A

172.16.0.0/12 | Private network class B

192.168.0.0/16 | Private network class C

203.0.113.0/24 | Office network (example)

##### Removing Internal Networks

To remove a network from the internal networks list:

1. Find the network in the list
2. Click the trash icon next to the network
3. The network will be removed immediately

#### API Endpoint Access Control

##### Configuring Endpoint Access

1. Navigate to Settings → Network Access
2. In the API Endpoint Access Control section, you'll see a table of all API endpoints
3. Use the checkboxes to control access:
4. Internal Access: Check to allow access from internal networks
5. External Access: Check to allow access from external networks

##### Filtering and Searching Endpoints

For easier management of large numbers of endpoints:

* Use the search box to find specific endpoints by path, method, or description
* Use the filters to show only internal-only or external-allowed endpoints

##### Recommended Endpoint Security

While all endpoint access settings are configurable, we recommend the following security best practices:

* Restrict sensitive admin operations to internal-only access
* Allow public-facing endpoints (authentication, content access) from both internal and external networks
* Regularly review access settings, especially after adding new endpoints

#### Rate Limiting Configuration

##### Default Rate Limit

The system has a default rate limit that applies to all requests from non-specified networks. By default, this is set to 30 requests per minute, but it can be configured:

1. Navigate to Settings → Network Access
2. In the Rate Limiting section, adjust the Default Rate Limit value
3. Click Save Network Settings

##### Custom Rate Limits

You can set custom rate limits for specific IP addresses or CIDR ranges:

1. Navigate to Settings → Network Access
2. In the Custom Rate Limits section:
3. Enter the IP or CIDR range (e.g., 10.0.0.0/8)
4. Provide a description (e.g., "Internal Network")
5. Set requests per minute (e.g., 300)
6. Click Add

10.0.0.0/8

##### Example Rate Limit Configuration

Table content:

Network | Description | Requests/Minute

(Default) | All unspecified networks | 30

10.0.0.0/8 | Internal Network | 300

203.0.113.0/24 | Office Network | 100

8.8.8.8 | Specific External Partner | 50

##### Category-Specific Rate Limits

Different endpoint categories have different default rate limits:

Table content:

Category | Default Limit | Internal Multiplier | Notes

Standard API | 100 req/min | Configurable | General API endpoints

Authentication | 20 req/min | 2x | More strict to prevent brute force

ML Processing | 10 req/min | Configurable | Resource-intensive operations

Agent API | 30 req/min | Configurable | AI agent interactions

PDF Processing | 5 req/10 min | Configurable | Very resource-intensive

### Technical Implementation

#### Core Components

1. UI Components
2. Network Access Panel in the admin dashboard
3. Internal networks configuration
4. API endpoint access controls
5. Rate limit management
6. Server-Side Models
7. InternalNetwork - Stores CIDR ranges for internal networks
8. EndpointAccessRule - Maps endpoints to their access permissions
9. RateLimitSettings - Stores default rate limits
10. CustomRateLimit - Stores network-specific rate limits
11. Middleware
12. Authorization middleware with network source checks
13. Rate limiting middleware with source-based limits

Rate limit management

Server-Side Models

InternalNetwork

EndpointAccessRule

RateLimitSettings

CustomRateLimit - Stores network-specific rate limits

CustomRateLimit

Middleware

#### Database Schema

-- Internal Networks  
CREATE TABLE internal\_networks (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 cidr VARCHAR(255) NOT NULL,  
 description VARCHAR(255),  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Endpoint Access Rules  
CREATE TABLE endpoint\_access\_rules (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 path VARCHAR(255) NOT NULL,  
 method VARCHAR(10) NOT NULL,  
 allow\_internal BOOLEAN DEFAULT TRUE,  
 allow\_external BOOLEAN DEFAULT FALSE,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 UNIQUE(path, method)  
);  
  
-- Rate Limit Settings  
CREATE TABLE rate\_limit\_settings (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 default\_requests\_per\_minute INTEGER NOT NULL DEFAULT 30,  
 auth\_multiplier FLOAT NOT NULL DEFAULT 0.66,  
 ml\_multiplier FLOAT NOT NULL DEFAULT 0.33,  
 agent\_multiplier FLOAT NOT NULL DEFAULT 1.0,  
 pdf\_multiplier FLOAT NOT NULL DEFAULT 0.16,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Custom Rate Limits  
CREATE TABLE custom\_rate\_limits (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 network VARCHAR(255) NOT NULL,  
 description VARCHAR(255),  
 requests\_per\_minute INTEGER NOT NULL,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 UNIQUE(network)  
);

-- Internal Networks  
CREATE TABLE internal\_networks (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 cidr VARCHAR(255) NOT NULL,  
 description VARCHAR(255),  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Endpoint Access Rules  
CREATE TABLE endpoint\_access\_rules (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 path VARCHAR(255) NOT NULL,  
 method VARCHAR(10) NOT NULL,  
 allow\_internal BOOLEAN DEFAULT TRUE,  
 allow\_external BOOLEAN DEFAULT FALSE,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 UNIQUE(path, method)  
);  
  
-- Rate Limit Settings  
CREATE TABLE rate\_limit\_settings (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 default\_requests\_per\_minute INTEGER NOT NULL DEFAULT 30,  
 auth\_multiplier FLOAT NOT NULL DEFAULT 0.66,  
 ml\_multiplier FLOAT NOT NULL DEFAULT 0.33,  
 agent\_multiplier FLOAT NOT NULL DEFAULT 1.0,  
 pdf\_multiplier FLOAT NOT NULL DEFAULT 0.16,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Custom Rate Limits  
CREATE TABLE custom\_rate\_limits (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 network VARCHAR(255) NOT NULL,  
 description VARCHAR(255),  
 requests\_per\_minute INTEGER NOT NULL,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 UNIQUE(network)  
);

#### Network Source Detection

The system uses the client's IP address to determine if a request is coming from an internal or external network:

function isInternalRequest(req: Request): boolean {  
 const clientIP = req.ip || req.socket.remoteAddress;  
  
 // Get internal networks from database  
 const internalNetworks = getInternalNetworksFromDatabase();  
  
 // Check if the IP matches any internal network  
 return internalNetworks.some(network => isInCIDR(clientIP, network.cidr));  
}  
  
// Check if IP is in a CIDR range  
function isInCIDR(ip: string, cidr: string): boolean {  
 // IP validation and CIDR matching logic  
}

function isInternalRequest(req: Request): boolean {  
 const clientIP = req.ip || req.socket.remoteAddress;  
  
 // Get internal networks from database  
 const internalNetworks = getInternalNetworksFromDatabase();  
  
 // Check if the IP matches any internal network  
 return internalNetworks.some(network => isInCIDR(clientIP, network.cidr));  
}  
  
// Check if IP is in a CIDR range  
function isInCIDR(ip: string, cidr: string): boolean {  
 // IP validation and CIDR matching logic  
}

#### Authorization Middleware

The authorization middleware is extended to check network source alongside user roles:

export const authorize = (options: {   
 roles?: string[],   
 accessType?: NetworkAccessType   
}) => {  
 return async (req: Request, res: Response, next: NextFunction) => {  
 // First check user authentication (done by preceding authMiddleware)  
  
 // Then check role-based permissions  
 if (options.roles && !hasRequiredRole(req.user, options.roles)) {  
 return next(new AuthorizationError('Insufficient permissions'));  
 }  
  
 // Then check network-based access  
 if (options.accessType === NetworkAccessType.INTERNAL\_ONLY) {  
 // Get endpoint rule from database  
 const endpointRule = await getEndpointRule(req.path, req.method);  
  
 // If rule exists, check if external access is allowed  
 if (endpointRule && !endpointRule.allowExternal && !isInternalRequest(req)) {  
 return next(new AuthorizationError('This endpoint is only accessible from internal networks'));  
 }  
 }  
  
 next();  
 };  
};

export const authorize = (options: {   
 roles?: string[],   
 accessType?: NetworkAccessType   
}) => {  
 return async (req: Request, res: Response, next: NextFunction) => {  
 // First check user authentication (done by preceding authMiddleware)  
  
 // Then check role-based permissions  
 if (options.roles && !hasRequiredRole(req.user, options.roles)) {  
 return next(new AuthorizationError('Insufficient permissions'));  
 }  
  
 // Then check network-based access  
 if (options.accessType === NetworkAccessType.INTERNAL\_ONLY) {  
 // Get endpoint rule from database  
 const endpointRule = await getEndpointRule(req.path, req.method);  
  
 // If rule exists, check if external access is allowed  
 if (endpointRule && !endpointRule.allowExternal && !isInternalRequest(req)) {  
 return next(new AuthorizationError('This endpoint is only accessible from internal networks'));  
 }  
 }  
  
 next();  
 };  
};

#### Rate Limiting Implementation

The rate-limiting middleware dynamically applies limits based on the source network:

export const createRateLimiter = (options: RateLimiterOptions) => {  
 return rateLimit({  
 windowMs: 60 \* 1000, // 1 minute  
 max: async (req) => {  
 const clientIP = req.ip || req.socket.remoteAddress;  
  
 // Get rate limit settings  
 const settings = await getRateLimitSettings();  
  
 // Check for custom rate limit for this IP  
 const customLimit = await getCustomRateLimit(clientIP);  
 if (customLimit) {  
 return customLimit.requestsPerMinute;  
 }  
  
 // Apply category-specific multiplier  
 const baseLimit = settings.defaultRequestsPerMinute;  
 switch (options.category) {  
 case 'auth':  
 return baseLimit \* settings.authMultiplier;  
 case 'search':  
 return baseLimit \* settings.searchMultiplier;  
 case 'admin':  
 return baseLimit \* settings.adminMultiplier;  
 case 'api':  
 return baseLimit \* settings.apiMultiplier;  
 default:  
 return baseLimit \* settings.standardMultiplier;  
 }  
 },  
 message: 'Too many requests, please try again later.',  
 standardHeaders: true,  
 legacyHeaders: false,  
 });  
};

export const createRateLimiter = (options: RateLimiterOptions) => {  
 return rateLimit({  
 windowMs: 60 \* 1000, // 1 minute  
 max: async (req) => {  
 const clientIP = req.ip || req.socket.remoteAddress;  
  
 // Get rate limit settings  
 const settings = await getRateLimitSettings();  
  
 // Check for custom rate limit for this IP  
 const customLimit = await getCustomRateLimit(clientIP);  
 if (customLimit) {  
 return customLimit.requestsPerMinute;  
 }  
  
 // Apply category-specific multiplier  
 const baseLimit = settings.defaultRequestsPerMinute;  
 switch (options.category) {  
 case 'auth':  
 return baseLimit \* settings.authMultiplier;  
 case 'search':  
 return baseLimit \* settings.searchMultiplier;  
 case 'admin':  
 return baseLimit \* settings.adminMultiplier;  
 case 'api':  
 return baseLimit \* settings.apiMultiplier;  
 default:  
 return baseLimit \* settings.standardMultiplier;  
 }  
 },  
 message: 'Too many requests, please try again later.',  
 standardHeaders: true,  
 legacyHeaders: false,  
 });  
};

#### Request Flow

When a request reaches the server:

1. IP Source Detection:
2. Extract the client IP (considering proxy headers if configured)
3. Check if the IP falls within any defined internal network CIDR ranges
4. Categorize the request as either internal or external
5. Network Access Check:
6. For endpoints marked as internal-only, verify the request comes from an internal network
7. For endpoints marked as external-only, verify the request comes from an external network
8. Deny access with 403 Forbidden if the network source doesn't match requirements
9. Rate Limit Application:
10. Check if the client IP matches any custom rate limit configuration
11. If found, apply the custom rate limit
12. If not, apply the default rate limit with appropriate category multiplier
13. Track request counts in a sliding 1-minute window
14. Return 429 Too Many Requests if the limit is exceeded

Categorize the request as either internal or external

Network Access Check:

Deny access with 403 Forbidden if the network source doesn't match requirements

Rate Limit Application:

#### Default Configuration

The system is deployed with sensible defaults:

1. Default Internal Networks
2. 127.0.0.1/8 (Localhost)
3. 10.0.0.0/8 (Private Class A)
4. 172.16.0.0/12 (Private Class B)
5. 192.168.0.0/16 (Private Class C)
6. Default Rate Limits
7. Default: 30 requests per minute
8. Internal networks: 300 requests per minute
9. Default Access Rules
10. Public endpoints: Internal & External access
11. Admin endpoints: Internal-only access
12. Sensitive operations: Internal-only access

192.168.0.0/16 (Private Class C)

Default Rate Limits

Internal networks: 300 requests per minute

Default Access Rules

#### Environment Configuration

For deployment environments, network settings can be configured via environment variables:

# .env file  
INTERNAL\_NETWORKS=127.0.0.1/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16  
TRUST\_PROXY=true

# .env file  
INTERNAL\_NETWORKS=127.0.0.1/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16  
TRUST\_PROXY=true

### Usage Examples

#### Example 1: Protecting Sensitive Operations

By default, sensitive operations like clearing analytics data are recommended to be internal-only. However, if your organization needs to allow trusted external partners to perform these operations:

1. Navigate to Settings > Network Access in the admin panel
2. Find the "Clear analytics data" endpoint (/api/admin/analytics/data)
3. Enable the "External Access" toggle
4. Click "Save Network Settings"

/api/admin/analytics/data

The endpoint will immediately become accessible from external networks (subject to proper authentication and authorization).

#### Example 2: Restricting Public Endpoints

Conversely, you may want to restrict normally public endpoints to internal access only:

1. Navigate to Settings > Network Access in the admin panel
2. Find the endpoint you want to restrict (e.g., /api/materials)
3. Disable the "External Access" toggle
4. Click "Save Network Settings"

/api/materials

The endpoint will immediately become inaccessible from external networks.

#### Example 3: Setting Up Development Access

For a development team working both in-office and remotely:

1. Add the office network CIDR to internal networks (e.g., 203.0.113.0/24)
2. Add the VPN IP range to internal networks (e.g., 198.51.100.0/24)
3. Set admin functionality to internal-only access
4. Configure a higher rate limit for the development networks

203.0.113.0/24

198.51.100.0/24

#### Example 4: External API Partner Access

For third-party services that need higher rate limits:

1. Identify the IP addresses used by the partner service
2. Add a custom rate limit specific to those IPs
3. Monitor usage patterns to detect anomalies
4. Consider requiring additional authentication for sensitive operations

### Best Practices

1. Use Defense in Depth
2. Network access controls should complement, not replace, proper authentication and authorization
3. Important operations should require both proper permissions AND appropriate network source
4. Don't Block by Default
5. Let administrators control access through the admin panel
6. Register all endpoints with the access control system
7. Use appropriate access types for sensitive operations
8. Document the access requirements for each endpoint
9. Review Access Settings Regularly
10. Audit network access settings periodically as part of security review
11. Temporary access changes should be reverted when no longer needed
12. Document Custom Configurations
13. Keep notes on why certain endpoints differ from the recommended access settings
14. Include review dates for sensitive access changes
15. Default to Restrictive
16. Set sensitive endpoints to internal-only by default, only opening them when necessary
17. Use internal-only for:  
      
    Data deletion operations  
    System configuration changes  
    User management operations  
    Backup/restore operations
18. Run the Setup Script
19. Run the setup script after adding new endpoints
20. Check the Admin Panel to verify that your endpoints appear

Important operations should require both proper permissions AND appropriate network source

Don't Block by Default

Document the access requirements for each endpoint

Review Access Settings Regularly

Temporary access changes should be reverted when no longer needed

Document Custom Configurations

Include review dates for sensitive access changes

Default to Restrictive

Use internal-only for:

* Data deletion operations
* System configuration changes
* User management operations
* Backup/restore operations

Run the Setup Script

### Developer Guidelines

#### Adding New Endpoints

When adding new API endpoints, follow these steps to ensure they are properly registered with the access control system:

1. Create your route file and define your endpoints as usual
2. Use the authorize middleware with the appropriate NetworkAccessType:

authorize

NetworkAccessType

// Example: Internal-only endpoint  
router.post('/admin/sensitive-operation',   
 authMiddleware,   
 authorize({   
 roles: ['admin'],   
 accessType: NetworkAccessType.INTERNAL\_ONLY   
 }),   
 asyncHandler(adminController.performSensitiveOperation)  
);  
  
// Example: External-allowed endpoint  
router.get('/public-data',   
 authMiddleware,   
 authorize({   
 roles: ['user'],   
 accessType: NetworkAccessType.EXTERNAL\_ALLOWED   
 }),   
 asyncHandler(publicController.getPublicData)  
);

// Example: Internal-only endpoint  
router.post('/admin/sensitive-operation',   
 authMiddleware,   
 authorize({   
 roles: ['admin'],   
 accessType: NetworkAccessType.INTERNAL\_ONLY   
 }),   
 asyncHandler(adminController.performSensitiveOperation)  
);  
  
// Example: External-allowed endpoint  
router.get('/public-data',   
 authMiddleware,   
 authorize({   
 roles: ['user'],   
 accessType: NetworkAccessType.EXTERNAL\_ALLOWED   
 }),   
 asyncHandler(publicController.getPublicData)  
);

1. Run the setup script to register the endpoint:

# From the project root  
cd packages/server  
yarn run-script setup-api-access-control

# From the project root  
cd packages/server  
yarn run-script setup-api-access-control

1. Verify that your endpoint appears in the admin panel

#### Recommended Access Types

Table content:

Endpoint Category | Recommended Access Type | Rationale

User authentication | External | Required for initial access

Data viewing | External | Core application functionality

Data creation/updating | External | Core application functionality

Admin dashboards | External | Allow admins to access from anywhere

System configuration | Internal | Protect sensitive configuration

Data deletion | Internal | Protect against data loss

Backup/restore | Internal | Protect system integrity

User management | Internal | Protect user security

#### Scripts and Utilities

The following scripts are available to help manage API access control:

1. setup-api-access-control.ts: Sets up the API access control system by checking for unregistered endpoints and registering them
2. check-unregistered-endpoints.ts: Scans the codebase for API endpoints and checks if they are registered
3. register-api-endpoints.ts: Registers API endpoints with the network access control system

To run these scripts:

# From the project root  
cd packages/server  
yarn run-script setup-api-access-control  
yarn run-script check-unregistered-endpoints  
yarn run-script register-api-endpoints

# From the project root  
cd packages/server  
yarn run-script setup-api-access-control  
yarn run-script check-unregistered-endpoints  
yarn run-script register-api-endpoints

### Troubleshooting

#### Common Issues

##### Access Denied Unexpectedly

Symptoms:  
- User receives "Network Access Denied" error  
- Endpoint is inaccessible from expected location

Potential Causes and Solutions:  
1. Client IP not recognized as internal  
 - Verify the client IP address in logs  
 - Check that the client's network is included in the internal networks configuration  
 - Add the client's network to the internal networks list if appropriate

1. Endpoint access configuration
2. Verify the endpoint's access type configuration
3. Check that the endpoint is configured to allow access from the client's network
4. Update the endpoint's access configuration if necessary
5. Proxy configuration
6. If behind a proxy, verify that the TRUST\_PROXY setting is enabled
7. Check that the X-Forwarded-For header is properly set
8. Verify that trusted proxies are correctly configured

Update the endpoint's access configuration if necessary

Proxy configuration

##### Rate Limiting Issues

Symptoms:  
- User receives "Too Many Requests" error  
- Multiple users sharing an IP address hit limits too quickly

Potential Causes and Solutions:  
1. Default rate limit too low  
 - Increase the default rate limit in the admin panel  
 - Configure custom rate limits for specific networks  
 - Adjust category multipliers for specific endpoint types

1. Multiple users sharing an IP
2. Set up custom rate limits for shared IPs
3. Consider implementing user-based rate limiting
4. Use token bucket algorithm for more sophisticated rate limiting
5. Legitimate high volume usage
6. Identify high-volume use cases
7. Configure custom rate limits for specific networks
8. Consider implementing API keys for fine-grained control

Use token bucket algorithm for more sophisticated rate limiting

Legitimate high volume usage

##### Configuration Not Taking Effect

Symptoms:  
- Changes to network access settings do not take effect  
- Rate limit changes are not applied

Potential Causes and Solutions:  
1. Caching issues  
 - Verify that the configuration has been properly saved  
 - Check that the cache is properly invalidated  
 - Restart the server if necessary

1. Environment variables overriding settings
2. Check environment variables in deployment environments
3. Ensure that INTERNAL\_NETWORKS is not overriding database settings
4. Update environment variables if necessary

#### Diagnostic Utilities

1. Network Test Endpoint: In development mode, use the /api/admin/network-test endpoint to verify network detection

/api/admin/network-test

GET /api/admin/network-test  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

GET /api/admin/network-test  
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Response:

{  
 "success": true,  
 "data": {  
 "ip": "192.168.1.100",  
 "isInternal": true,  
 "matchedNetwork": "192.168.0.0/16"  
 }  
}

{  
 "success": true,  
 "data": {  
 "ip": "192.168.1.100",  
 "isInternal": true,  
 "matchedNetwork": "192.168.0.0/16"  
 }  
}

1. Logging: Enable detailed logging for network access and rate limiting:

// In your .env file  
LOG\_LEVEL=debug  
NETWORK\_ACCESS\_LOGGING=true  
RATE\_LIMIT\_LOGGING=true

// In your .env file  
LOG\_LEVEL=debug  
NETWORK\_ACCESS\_LOGGING=true  
RATE\_LIMIT\_LOGGING=true

1. Check for unregistered endpoints:

# From the project root  
cd packages/server  
yarn run-script check-unregistered-endpoints

# From the project root  
cd packages/server  
yarn run-script check-unregistered-endpoints

### Security Considerations

#### Defense in Depth

Network-based access control should be considered one layer in a defense-in-depth strategy:

* Authentication verifies the user's identity
* Authorization checks the user's permissions
* Network access control validates the request's origin
* Rate limiting prevents abuse

Even if a user has proper credentials, they cannot access internal-only endpoints from external networks.

#### IP Spoofing Protection

The system is designed to properly handle proxy headers while protecting against IP spoofing:

* X-Forwarded-For headers are only trusted when the TRUST\_PROXY setting is enabled
* Only trusted proxies can set the X-Forwarded-For header
* The system validates the client IP address before using it for access control

#### Fail-Secure Default

The system defaults to treating requests as external if network determination fails, providing a fail-secure approach:

* Errors in network detection result in treating the request as external
* Sensitive endpoints are protected by default
* Critical endpoints are recommended to be internal-only

#### Comprehensive Logging

All access attempts are logged, especially denied attempts:

* Logs include source IP, requested endpoint, and reason for denial
* Rate limit breaches are logged with details
* Security events are flagged for monitoring

### Testing

#### Testing Network Access Controls

1. Internal Network Tests:
2. Test from within the internal network
3. Verify access to both internal-only and external-allowed endpoints
4. External Network Tests:
5. Test from external networks (e.g., public internet)
6. Verify access to external-allowed endpoints
7. Verify denial of access to internal-only endpoints
8. Edge Cases:
9. Test with VPN connections
10. Test with various proxy configurations
11. Test boundary conditions of CIDR ranges

Verify access to both internal-only and external-allowed endpoints

External Network Tests:

Verify denial of access to internal-only endpoints

Edge Cases:

### API Reference

For a complete list of API endpoints and their recommended access settings, refer to the API Reference documentation.

### Extending the System

The network access control system is designed to be extensible:

1. Additional Access Types:
2. The system can be extended to support more granular access types beyond internal/external
3. Dynamic Configuration:
4. Advanced implementations could adjust access rules based on time of day, system load, etc.
5. Integration with Security Systems:
6. The system can be integrated with threat detection systems
7. Could implement automatic blacklisting of suspicious IPs
8. Future Enhancements:
9. Geo-Based Restrictions: Add support for restricting access based on geographic location
10. Time-Based Restrictions: Add support for restricting access based on time of day
11. Advanced Rate Limiting: Add support for token bucket or leaky bucket algorithms
12. Request Pattern Detection: Add anomaly detection to identify unusual request patterns
13. Dynamic Rate Limiting: Adjust rate limits based on system load or other factors

The system can be extended to support more granular access types beyond internal/external

Dynamic Configuration:

Advanced implementations could adjust access rules based on time of day, system load, etc.

Integration with Security Systems:

Could implement automatic blacklisting of suspicious IPs

Future Enhancements:

# Neural Ocr Integration

Source: readme/neural-ocr-integration.md

---

## Neural OCR Integration

This document describes the advanced Neural OCR integration that enhances the platform's document understanding capabilities through multiple specialized OCR engines.

### Overview

The Neural OCR integration extends the platform's existing OCR capabilities by incorporating seven advanced document understanding engines:

1. Nougat - Meta's Neural Optical Understanding for Academic Documents
2. Marker - VikParuchuri's layout-preserving document understanding
3. thepipe - emcf's structured information extraction pipeline
4. PaddleOCR - Baidu's powerful multilingual OCR toolkit
5. pdfdeal - NoEdgeAI's PDF-native processing solution
6. surya - VikParuchuri's scientific document understanding system
7. mPLUG-DocOwl - X-PLUG's multimodal document understanding model

These engines complement the existing Tesseract-based OCR to provide a comprehensive document understanding solution that intelligently routes different document types and regions to the most appropriate engine.

### Architecture

The integration follows an orchestrator pattern that coordinates multiple OCR engines:

┌─────────────────────┐  
│ │  
│ Document Router │  
│ │  
└─────────┬───────────┘  
 │  
┌─────────▼───────────┐  
│ │  
│ Engine Dispatcher │  
│ │  
└─────────┬───────────┘  
 │  
┌─────────┴────────────────────────────────────────────────────────────────┐  
│ │  
│ │  
┌─────────────┐ ┌───────────┐ ┌──────────┐ ┌───────────┐ ┌────────────┐ │  
│ │ │ │ │ │ │ │ │ │ │  
│ Tesseract │ │ Nougat │ │ Marker │ │ PaddleOCR │ │ pdfdeal │ │  
│ │ │ │ │ │ │ │ │ │ │  
└─────────────┘ └───────────┘ └──────────┘ └───────────┘ └────────────┘ │  
 │ │ │  
 ┌──────────▼─────┐ ┌────────────▼───────┐ │  
 │ │ │ │ │  
 │ thepipe │ │ surya │ │  
 │ │ │ │ │  
 └────────────────┘ └────────────────────┘ │  
 │ │  
 ┌───────────▼─────────────┐ │  
 │ │ │  
 │ mPLUG-DocOwl │ │  
 │ │ │  
 └─────────────────────────┘ │  
 │  
└───────────────────────────────────────────────────────────────────────────┘

┌─────────────────────┐  
│ │  
│ Document Router │  
│ │  
└─────────┬───────────┘  
 │  
┌─────────▼───────────┐  
│ │  
│ Engine Dispatcher │  
│ │  
└─────────┬───────────┘  
 │  
┌─────────┴────────────────────────────────────────────────────────────────┐  
│ │  
│ │  
┌─────────────┐ ┌───────────┐ ┌──────────┐ ┌───────────┐ ┌────────────┐ │  
│ │ │ │ │ │ │ │ │ │ │  
│ Tesseract │ │ Nougat │ │ Marker │ │ PaddleOCR │ │ pdfdeal │ │  
│ │ │ │ │ │ │ │ │ │ │  
└─────────────┘ └───────────┘ └──────────┘ └───────────┘ └────────────┘ │  
 │ │ │  
 ┌──────────▼─────┐ ┌────────────▼───────┐ │  
 │ │ │ │ │  
 │ thepipe │ │ surya │ │  
 │ │ │ │ │  
 └────────────────┘ └────────────────────┘ │  
 │ │  
 ┌───────────▼─────────────┐ │  
 │ │ │  
 │ mPLUG-DocOwl │ │  
 │ │ │  
 └─────────────────────────┘ │  
 │  
└───────────────────────────────────────────────────────────────────────────┘

#### Key Components

##### 1. Neural OCR Orchestrator

The neural\_ocr\_orchestrator.py module provides:

neural\_ocr\_orchestrator.py

* Intelligent routing of document regions to appropriate engines
* Document layout analysis and segmentation
* Result aggregation from multiple engines
* Confidence-based fallback mechanisms
* Unified API for all OCR capabilities

##### 2. Engine-Specific Modules

Each engine has a dedicated module that handles its specific integration:

* nougat\_engine.py - Integrates Meta's Nougat for scientific and technical content
* marker\_engine.py - Integrates Marker for complex layouts and multi-column documents
* thepipe\_engine.py - Integrates thepipe for structured information extraction
* additional\_engines.py - Provides adapters for additional OCR engines:
* PaddleOCREngine - Baidu's multilingual OCR toolkit
* PdfDealEngine - NoEdgeAI's PDF-native processing
* SuryaEngine - Scientific document understanding
* DocOwlEngine - Multimodal document understanding

nougat\_engine.py

marker\_engine.py

thepipe\_engine.py

additional\_engines.py

##### 3. Extensible Engine Manager

The extensible\_engine\_manager.py module provides a framework for integrating all engines:

extensible\_engine\_manager.py

* Pluggable architecture for engine registration and discovery
* Standardized interfaces for consistent integration
* Dynamic loading of available engines
* Resource management for efficient processing

##### 3. Integration with Existing System

The neural OCR system integrates with the existing OCR pipeline:

* Maintains compatibility with current OCR interfaces
* Provides graceful fallback to Tesseract when needed
* Enhances rather than replaces existing capabilities

### Engine Specializations

Each engine in the system excels at different document understanding tasks:

#### Original Engines

##### Nougat (Meta)

Nougat specializes in scientific and technical documents:

* Mathematical formulas and equations
* Technical specifications
* Scientific notation
* Complex tables with merged cells
* Academic document understanding

##### Marker (VikParuchuri)

Marker excels at preserving document layout:

* Multi-column layouts
* Complex typographical arrangements
* Mixed text and image content
* Maintaining text flow and relationships
* Real-world documents with irregular formatting

##### thepipe (emcf)

thepipe specializes in structured information extraction:

* Form field extraction
* Consistent data point capture
* Standardized document formats
* Material specification sheets
* Structured output generation

##### Tesseract (Legacy)

The existing Tesseract implementation handles:

* Basic text extraction
* Simple document layouts
* Well-defined paragraphs and text blocks
* Fallback processing when neural models aren't available

#### Additional Engines

##### PaddleOCR (Baidu)

PaddleOCR excels at multilingual and complex text recognition:

* Multilingual support for 80+ languages
* Superior handling of dense text and low-quality images
* Advanced table structure recognition
* Detection and recognition of text in natural scenes
* Accurate handling of non-Latin scripts and complex writing systems

##### pdfdeal (NoEdgeAI)

pdfdeal specializes in native PDF processing:

* Direct extraction from PDF text layers
* Preservation of original document structure
* Native table extraction without image conversion
* Form field detection and extraction
* End-to-end PDF document understanding

##### surya (VikParuchuri)

surya focuses on advanced academic and scientific document comprehension:

* Superior handling of scientific notation and equations
* Semantic understanding of technical content
* Context-aware text extraction
* Long document coherence
* Advanced rendering of complex academic layouts

##### mPLUG-DocOwl (X-PLUG)

mPLUG-DocOwl provides multimodal document understanding:

* Visual-textual relationship comprehension
* Document question answering capabilities
* Zero-shot learning for new document types
* Image-text association in complex layouts
* Context-aware understanding of document elements

### Usage

#### Basic Usage

from neural\_ocr\_orchestrator import NeuralOCROrchestrator  
  
# Initialize the orchestrator  
orchestrator = NeuralOCROrchestrator()  
  
# Process a document  
result = orchestrator.process\_document("path/to/document.pdf")  
  
# Access extracted text  
text = result["result"]["text"]  
  
# Access structured content  
structured\_content = result["result"]["structured\_content"]

from neural\_ocr\_orchestrator import NeuralOCROrchestrator  
  
# Initialize the orchestrator  
orchestrator = NeuralOCROrchestrator()  
  
# Process a document  
result = orchestrator.process\_document("path/to/document.pdf")  
  
# Access extracted text  
text = result["result"]["text"]  
  
# Access structured content  
structured\_content = result["result"]["structured\_content"]

#### Using Additional Engines

from neural\_ocr\_orchestrator import NeuralOCROrchestrator  
  
# Initialize with additional engines  
orchestrator = NeuralOCROrchestrator({  
 'engines': ['paddleocr', 'pdfdeal', 'surya', 'docowl'],  
 'language': 'zh-CN' # Use PaddleOCR for Chinese content  
})  
  
# Process multilingual document  
result = orchestrator.process\_document("path/to/multilingual\_document.pdf")  
  
# Process PDF natively  
pdf\_result = orchestrator.process\_document(  
 "path/to/vector\_pdf.pdf",  
 {'use\_native\_pdf': True} # Use pdfdeal for native processing  
)  
  
# Process scientific document  
scientific\_result = orchestrator.process\_document(  
 "path/to/scientific\_paper.pdf",  
 {'engine': 'surya'} # Force surya for scientific content  
)  
  
# Process document with image-text relationships  
multimodal\_result = orchestrator.process\_document(  
 "path/to/catalog.pdf",  
 {'engine': 'docowl', 'extract\_relationships': True}  
)

from neural\_ocr\_orchestrator import NeuralOCROrchestrator  
  
# Initialize with additional engines  
orchestrator = NeuralOCROrchestrator({  
 'engines': ['paddleocr', 'pdfdeal', 'surya', 'docowl'],  
 'language': 'zh-CN' # Use PaddleOCR for Chinese content  
})  
  
# Process multilingual document  
result = orchestrator.process\_document("path/to/multilingual\_document.pdf")  
  
# Process PDF natively  
pdf\_result = orchestrator.process\_document(  
 "path/to/vector\_pdf.pdf",  
 {'use\_native\_pdf': True} # Use pdfdeal for native processing  
)  
  
# Process scientific document  
scientific\_result = orchestrator.process\_document(  
 "path/to/scientific\_paper.pdf",  
 {'engine': 'surya'} # Force surya for scientific content  
)  
  
# Process document with image-text relationships  
multimodal\_result = orchestrator.process\_document(  
 "path/to/catalog.pdf",  
 {'engine': 'docowl', 'extract\_relationships': True}  
)

#### Using the Extensible Engine Manager

from extensible\_engine\_manager import EngineManager  
  
# Create engine manager with all available engines  
manager = EngineManager()  
  
# Get list of available engines  
engines = manager.get\_available\_engines()  
print(f"Available engines: {list(engines.keys())}")  
  
# Process document with voting from multiple engines  
result = manager.process\_with\_voting(  
 "path/to/document.pdf",  
 engine\_names=['paddleocr', 'surya', 'tesseract']  
)  
  
# Create a processing pipeline  
from extensible\_engine\_manager import EnginePipeline  
  
pipeline = EnginePipeline(manager)  
pipeline.add\_step("pdfdeal", {"extract\_text\_layer": True}, "extract\_text")  
pipeline.add\_step("surya", {"parse\_formulas": True}, "process\_formulas")  
pipeline.add\_step("docowl", {"associate\_images": True}, "image\_text\_association")  
  
# Process document through pipeline  
pipeline\_result = pipeline.process("path/to/document.pdf")

from extensible\_engine\_manager import EngineManager  
  
# Create engine manager with all available engines  
manager = EngineManager()  
  
# Get list of available engines  
engines = manager.get\_available\_engines()  
print(f"Available engines: {list(engines.keys())}")  
  
# Process document with voting from multiple engines  
result = manager.process\_with\_voting(  
 "path/to/document.pdf",  
 engine\_names=['paddleocr', 'surya', 'tesseract']  
)  
  
# Create a processing pipeline  
from extensible\_engine\_manager import EnginePipeline  
  
pipeline = EnginePipeline(manager)  
pipeline.add\_step("pdfdeal", {"extract\_text\_layer": True}, "extract\_text")  
pipeline.add\_step("surya", {"parse\_formulas": True}, "process\_formulas")  
pipeline.add\_step("docowl", {"associate\_images": True}, "image\_text\_association")  
  
# Process document through pipeline  
pipeline\_result = pipeline.process("path/to/document.pdf")

#### Engine Selection

You can configure which engines to use:

# Initialize with specific engines  
orchestrator = NeuralOCROrchestrator({  
 'engine\_priority': ['nougat', 'marker'], # Prioritize these engines  
 'confidence\_threshold': 0.7, # Minimum confidence threshold  
})

# Initialize with specific engines  
orchestrator = NeuralOCROrchestrator({  
 'engine\_priority': ['nougat', 'marker'], # Prioritize these engines  
 'confidence\_threshold': 0.7, # Minimum confidence threshold  
})

#### Integration with Material Recognition

The neural OCR system enhances material data extraction:

# Example integration with material recognition  
from neural\_ocr\_orchestrator import NeuralOCROrchestrator  
  
# Process material datasheet  
orchestrator = NeuralOCROrchestrator({  
 'pipeline\_type': 'material\_specs' # Optimized for material specifications  
})  
  
# Extract material specifications  
result = orchestrator.process\_document("path/to/material\_datasheet.pdf")  
  
# Access structured material data  
dimensions = result["result"]["structured\_content"]["dimensions"]  
material\_type = result["result"]["structured\_content"]["product\_info"]["material\_type"]

# Example integration with material recognition  
from neural\_ocr\_orchestrator import NeuralOCROrchestrator  
  
# Process material datasheet  
orchestrator = NeuralOCROrchestrator({  
 'pipeline\_type': 'material\_specs' # Optimized for material specifications  
})  
  
# Extract material specifications  
result = orchestrator.process\_document("path/to/material\_datasheet.pdf")  
  
# Access structured material data  
dimensions = result["result"]["structured\_content"]["dimensions"]  
material\_type = result["result"]["structured\_content"]["product\_info"]["material\_type"]

### Implementation Details

#### Document Routing Logic

The system uses a sophisticated decision-making process to route document content:

1. Document Analysis: Analyzes document layout to identify distinct regions
2. Region Classification: Classifies regions based on content type
3. Engine Selection: Selects optimal engine for each region
4. Parallel Processing: Processes regions in parallel when possible
5. Result Aggregation: Combines results based on confidence scores

#### Fallback Mechanisms

The system implements graceful degradation:

1. Primary Engine: Attempts to use the most appropriate specialized engine
2. Confidence Check: Verifies results meet minimum confidence threshold
3. Secondary Engine: Falls back to alternative engine if needed
4. Tesseract Fallback: Always available as final fallback option

### Installation and Dependencies

The neural OCR integration requires additional dependencies beyond the standard OCR system:

#### Python Dependencies

Required packages are listed in packages/ml/python/requirements-ocr.txt:

packages/ml/python/requirements-ocr.txt

# Base OCR Dependencies  
pytesseract>=0.3.8 # OCR engine wrapper  
tesseract-ocr>=4.1.1 # Base OCR engine (system package)  
Pillow>=8.2.0 # Image processing  
  
# Original Neural OCR Engines  
torch>=1.9.0 # PyTorch for neural models  
torchvision>=0.10.0 # Computer vision for PyTorch  
nougat-ocr>=0.1.14 # Meta's Nougat for scientific documents  
marker-ocr>=0.1.5 # VikParuchuri's Marker for layout-preserving OCR  
thepipe>=0.2.0 # emcf's thepipe for structured information extraction  
transformers>=4.24.0 # Hugging Face transformers  
  
# Additional Neural OCR Engines  
paddlepaddle>=2.4.0 # PaddleOCR base framework  
paddleocr>=2.6.0 # PaddleOCR toolkit  
pdfdeal>=0.2.0 # PDF native processing  
surya>=0.2.0 # Scientific document understanding  
mplug-docowl>=0.1.0 # Multimodal document understanding

# Base OCR Dependencies  
pytesseract>=0.3.8 # OCR engine wrapper  
tesseract-ocr>=4.1.1 # Base OCR engine (system package)  
Pillow>=8.2.0 # Image processing  
  
# Original Neural OCR Engines  
torch>=1.9.0 # PyTorch for neural models  
torchvision>=0.10.0 # Computer vision for PyTorch  
nougat-ocr>=0.1.14 # Meta's Nougat for scientific documents  
marker-ocr>=0.1.5 # VikParuchuri's Marker for layout-preserving OCR  
thepipe>=0.2.0 # emcf's thepipe for structured information extraction  
transformers>=4.24.0 # Hugging Face transformers  
  
# Additional Neural OCR Engines  
paddlepaddle>=2.4.0 # PaddleOCR base framework  
paddleocr>=2.6.0 # PaddleOCR toolkit  
pdfdeal>=0.2.0 # PDF native processing  
surya>=0.2.0 # Scientific document understanding  
mplug-docowl>=0.1.0 # Multimodal document understanding

Note: Installation instructions for Neural OCR have been moved to the Deployment Guide.

### Performance Considerations

The neural OCR system is more resource-intensive than traditional OCR:

1. Hardware Requirements:
2. GPU recommended for optimal performance
3. Minimum 8GB RAM for complex documents
4. SSD storage for model caching
5. CUDA-compatible GPU for PaddleOCR and mPLUG-DocOwl
6. Processing Speed:
7. Neural models are slower than Tesseract
8. Parallel processing helps mitigate performance impact
9. Caching improves performance for repeated document types
10. Native PDF processing with pdfdeal can be significantly faster for digital PDFs
11. Accuracy vs. Speed:
12. Configure confidence thresholds to balance accuracy and speed
13. Use selective engine activation for faster processing
14. Consider GPU acceleration for production deployments
15. PaddleOCR provides lightweight models for faster processing with good accuracy

CUDA-compatible GPU for PaddleOCR and mPLUG-DocOwl

Processing Speed:

Native PDF processing with pdfdeal can be significantly faster for digital PDFs

Accuracy vs. Speed:

### Detailed Processing Workflows

#### PDF Upload Workflow

When a PDF catalog or datasheet is uploaded to the system, it undergoes the following step-by-step processing:

1. Initial PDF Analysis
2. The PDF is analyzed for structure, page count, and content types
3. Metadata is extracted (creation date, author, title)
4. The system determines if the PDF contains text layer or requires OCR
5. Page Extraction and Segmentation
6. Each page is extracted and converted to high-resolution images
7. Pages are segmented into logical regions (text blocks, images, tables)
8. Region types are identified (headings, body text, specifications, etc.)
9. Image Extraction and Processing
10. Product images are isolated and extracted
11. Images are preprocessed (cropping, enhancement, normalization)
12. Duplicate images are detected and consolidated
13. OCR Engine Selection and Routing
14. For each text region, the neural OCR orchestrator determines the optimal engine:  
      
    Technical specifications & tables → Nougat engine  
    Multi-column catalog layouts → Marker engine  
    Form fields & structured data → thepipe engine  
    Basic text → Tesseract engine
15. Parallel Processing
16. Each region is processed by its assigned engine in parallel
17. Large PDFs are processed page by page to manage resources
18. Progress is tracked and reported for the overall document
19. Multi-Engine Processing (When Necessary)
20. For critical regions (e.g., product specifications), multiple engines may process the same content
21. The system processes the region with 2+ engines (typically including Tesseract as baseline)
22. Results from all engines are collected for comparison
23. Result Aggregation
24. Results from all engines across all regions are collected
25. The orchestrator combines outputs based on confidence scores
26. For regions processed by multiple engines, the highest confidence result is selected
27. Textual context is used to validate and improve the final result
28. Structured Data Extraction
29. Raw OCR text is parsed to extract structured information:  
      
    Product codes and identifiers  
    Material specifications (dimensions, properties, ratings)  
    Technical parameters and measurements  
    ASTM/ISO standards references
30. Result Validation
31. Automated validation is performed on extracted data:  
      
    Format validation (e.g., product codes match expected patterns)  
    Value range checking (e.g., dimensions are within reasonable limits)  
    Cross-reference validation (e.g., text matches image content)  
    Confidence thresholds (results below threshold are flagged)
32. Knowledge Base Integration  
      
    Validated data is mapped to knowledge base schema  
    Material entries are created or updated  
    Relationships with existing materials are established  
    PDF source is linked to created/updated materials

The system determines if the PDF contains text layer or requires OCR

Page Extraction and Segmentation

Region types are identified (headings, body text, specifications, etc.)

Image Extraction and Processing

Duplicate images are detected and consolidated

OCR Engine Selection and Routing

For each text region, the neural OCR orchestrator determines the optimal engine:

* Technical specifications & tables → Nougat engine
* Multi-column catalog layouts → Marker engine
* Form fields & structured data → thepipe engine
* Basic text → Tesseract engine

Parallel Processing

Progress is tracked and reported for the overall document

Multi-Engine Processing (When Necessary)

Results from all engines are collected for comparison

Result Aggregation

Textual context is used to validate and improve the final result

Structured Data Extraction

Raw OCR text is parsed to extract structured information:

* Product codes and identifiers
* Material specifications (dimensions, properties, ratings)
* Technical parameters and measurements
* ASTM/ISO standards references

Result Validation

Automated validation is performed on extracted data:

* Format validation (e.g., product codes match expected patterns)
* Value range checking (e.g., dimensions are within reasonable limits)
* Cross-reference validation (e.g., text matches image content)
* Confidence thresholds (results below threshold are flagged)

Knowledge Base Integration

* Validated data is mapped to knowledge base schema
* Material entries are created or updated
* Relationships with existing materials are established
* PDF source is linked to created/updated materials

#### Web Crawling Workflow

When the system crawls a website for material information:

1. Page Analysis
2. Crawled HTML pages are analyzed for structure and content
3. JavaScript-rendered content is captured via headless browser
4. Material-related sections are identified
5. Content Extraction
6. Text is directly extracted from HTML where possible
7. Images are downloaded and processed
8. Tables are extracted preserving structure
9. PDFs and other documents are downloaded for separate processing
10. Image OCR Processing
11. Each product image is processed with region detection
12. Detected text regions around product images are processed by the neural OCR system
13. The OCR orchestrator selects the appropriate engine for each region:  
      
    Nougat for technical specifications adjacent to images  
    Marker for complex layouts with multiple products  
    thepipe for tabular product specifications
14. Image-Text Association
15. Text extracted via OCR is associated with the corresponding product images
16. Association is based on proximity, contained text (e.g., product codes), and context
17. Multiple text blocks may be associated with a single image (specs, description, code)
18. Product Information Aggregation
19. Text from HTML and OCR are combined for each product
20. Priority is given to structured HTML data when available
21. OCR data is used to fill gaps or enhance existing information
22. Confidence scores are tracked for all extracted data
23. Validation and Quality Control
24. Data is validated through multiple methods:  
      
    Cross-referencing between HTML and OCR text  
    Comparison with known product patterns and formats  
    Consistency checking across similar products  
    Outlier detection for unusual values
25. Knowledge Base Integration
26. Validated product data is mapped to the knowledge base schema
27. Source URL is preserved for attribution and verification
28. Crawl timestamp is recorded for freshness tracking

Material-related sections are identified

Content Extraction

PDFs and other documents are downloaded for separate processing

Image OCR Processing

The OCR orchestrator selects the appropriate engine for each region:

* Nougat for technical specifications adjacent to images
* Marker for complex layouts with multiple products
* thepipe for tabular product specifications

Image-Text Association

Multiple text blocks may be associated with a single image (specs, description, code)

Product Information Aggregation

Confidence scores are tracked for all extracted data

Validation and Quality Control

Data is validated through multiple methods:

* Cross-referencing between HTML and OCR text
* Comparison with known product patterns and formats
* Consistency checking across similar products
* Outlier detection for unusual values

Knowledge Base Integration

### Result Validation and Quality Assurance

The neural OCR system employs a comprehensive validation framework to ensure accuracy:

#### Multi-Level Validation

1. Engine-Level Validation
2. Each OCR engine performs internal validation
3. Confidence scores are generated for each text element
4. Alternative interpretations are ranked by confidence
5. Engine-specific heuristics filter improbable results
6. Cross-Engine Validation
7. When multiple engines process the same region, results are compared
8. Agreement between engines increases confidence
9. Discrepancies trigger deeper analysis
10. Weighted voting system resolves conflicts
11. Domain-Specific Validation
12. Material-specific knowledge is applied to validate results:  
      
    Product code format validation (e.g., A123-456B)  
    Dimension format validation (e.g., 600x600mm)  
    Technical parameter validation (e.g., PEI ratings I-V)  
    Unit consistency checking
13. Context-Based Validation
14. Surrounding content provides validation context
15. Section headings inform expected content type
16. Related fields provide cross-validation
17. Overall document context guides interpretation

Engine-specific heuristics filter improbable results

Cross-Engine Validation

Weighted voting system resolves conflicts

Domain-Specific Validation

Material-specific knowledge is applied to validate results:

* Product code format validation (e.g., A123-456B)
* Dimension format validation (e.g., 600x600mm)
* Technical parameter validation (e.g., PEI ratings I-V)
* Unit consistency checking

Context-Based Validation

#### Quality Metrics and Thresholds

The system tracks quality at multiple levels:

1. Character-Level Confidence
2. Each character has an individual confidence score
3. Character confusion matrices identify problematic characters
4. Special character handling for technical symbols
5. Word-Level Confidence
6. Word confidence combines character scores
7. Dictionary validation for common terms
8. Domain-specific terminology validation
9. Named entity recognition for product names
10. Field-Level Confidence
11. Structured fields have format-specific validation
12. Field confidence combines word confidences and format validation
13. Required fields have higher validation standards
14. Field relationships provide cross-validation
15. Document-Level Quality Score
16. Overall document quality assessment
17. Weighted by field importance (product codes > descriptions)
18. Flagging system for low-confidence documents
19. Threshold-based routing to human verification

Special character handling for technical symbols

Word-Level Confidence

Named entity recognition for product names

Field-Level Confidence

Field relationships provide cross-validation

Document-Level Quality Score

#### Human Verification Workflow

For results that don't meet confidence thresholds:

1. Verification Interface
2. Low-confidence results are flagged for human review
3. Original image and OCR result are presented side-by-side
4. Correction interface for efficient updates
5. Batch processing for similar corrections
6. Feedback Loop
7. Human corrections feed back into the OCR system
8. Error patterns are identified for system improvement
9. Custom dictionaries are updated with domain terms
10. Engine selection rules are refined based on performance
11. Progressive Improvement
12. The system learns from verification patterns
13. Document types with consistent issues get specialized handling
14. Confidence thresholds are adjusted based on error rates
15. Engine-specific optimizations are implemented

Batch processing for similar corrections

Feedback Loop

Engine selection rules are refined based on performance

Progressive Improvement

### Example: Processing a Material Datasheet

Here's a concrete example of how a typical material datasheet flows through the system with the additional engines:

1. User uploads a multilingual ceramic tile product datasheet PDF (4 pages)
2. Initial Processing
3. PDF is analyzed: 4 pages, contains images and text in multiple languages
4. pdfdeal checks for native text layer (found on 2 pages)
5. Pages are extracted as high-resolution images where needed
6. Document is identified as a technical datasheet
7. 15 regions are identified across all pages:  
     
   3 product images  
   2 tables with specifications  
   3 headings  
   4 text blocks with descriptions  
   3 multilingual marketing sections
8. Engine Allocation
9. pdfdeal extracts native text layer from digital pages
10. PaddleOCR processes the multilingual text sections (Chinese/Spanish)
11. Nougat processes the specification tables
12. surya handles technical descriptions with formulas
13. Marker processes the multi-column product descriptions
14. thepipe extracts structured fields (dimensions, codes)
15. mPLUG-DocOwl associates images with corresponding specifications
16. Tesseract processes simple headings
17. Parallel Processing
18. All engines run concurrently on their assigned regions
19. Native PDF text extraction occurs first to provide baseline
20. Product images are processed separately for recognition
21. Tables are processed with structure preservation
22. Language detection guides multilingual processing
23. Result Aggregation
24. OCR results from all engines are combined
25. The system identifies:  
      
    Product name and code in multiple languages (99% confidence)  
    Dimensions: 600x600mm (97% confidence)  
    Material type: Porcelain (96% confidence)  
    Surface finish: Matte (92% confidence)  
    PEI Rating: IV (90% confidence)  
    Technical specs table (95% average confidence)  
    Chemical composition formula (93% confidence via surya)  
    Image-text associations (90% confidence via mPLUG-DocOwl)
26. Validation and Correction
27. One field (water absorption) has low confidence (65%)
28. The system compares against known values for porcelain
29. Corrected value is accepted based on context
30. Cross-validation between different language versions improves certainty
31. Final structured data record is assembled
32. Knowledge Base Integration
33. New material entry is created in the knowledge base
34. Product images are linked to the material with semantic relationships
35. Extracted specifications populate material properties
36. Multilingual descriptions are stored with language tags
37. PDF is stored as a reference document
38. Material is categorized based on extracted properties

User uploads a multilingual ceramic tile product datasheet PDF (4 pages)

Initial Processing

15 regions are identified across all pages:

* 3 product images
* 2 tables with specifications
* 3 headings
* 4 text blocks with descriptions
* 3 multilingual marketing sections

Engine Allocation

Tesseract processes simple headings

Parallel Processing

Language detection guides multilingual processing

Result Aggregation

The system identifies:

* Product name and code in multiple languages (99% confidence)
* Dimensions: 600x600mm (97% confidence)
* Material type: Porcelain (96% confidence)
* Surface finish: Matte (92% confidence)
* PEI Rating: IV (90% confidence)
* Technical specs table (95% average confidence)
* Chemical composition formula (93% confidence via surya)
* Image-text associations (90% confidence via mPLUG-DocOwl)

Validation and Correction

Final structured data record is assembled

Knowledge Base Integration

The entire process completes in under 25 seconds for a typical 4-page datasheet (faster than before due to native PDF processing where available), with human review only needed for specific low-confidence fields that couldn't be auto-corrected.

### Implemented Enhancements

The following advanced capabilities have been implemented to further enhance the neural OCR system:

#### 1. Model Fine-tuning

The ocr\_model\_finetuner.py module provides domain-specific model fine-tuning capabilities:

ocr\_model\_finetuner.py

* Dataset Management: Tools for creating, managing, and augmenting training datasets
* Engine-specific Training: Fine-tuning capabilities for Nougat, Marker, and thepipe
* Domain Adaptation: Material-specific optimization for improved accuracy
* Evaluation Framework: Comprehensive evaluation and model comparison

# Example: Fine-tuning a model for material datasheets  
from ocr\_model\_finetuner import OCRModelFineTuner, OCRDataset  
  
# Prepare dataset  
dataset = OCRDataset({  
 'dataset\_dir': 'material\_datasets'  
})  
dataset.prepare\_from\_documents(['path/to/datasheet1.pdf', 'path/to/datasheet2.pdf'])  
dataset.augment\_data()  
  
# Fine-tune model  
finetuner = OCRModelFineTuner({  
 'engine': 'nougat',  
 'domain': 'material\_datasheets',  
 'model\_type': 'base'  
})  
result = finetuner.finetune(dataset)  
  
# Export model  
finetuner.export\_model('models/fine\_tuned\_nougat')

# Example: Fine-tuning a model for material datasheets  
from ocr\_model\_finetuner import OCRModelFineTuner, OCRDataset  
  
# Prepare dataset  
dataset = OCRDataset({  
 'dataset\_dir': 'material\_datasets'  
})  
dataset.prepare\_from\_documents(['path/to/datasheet1.pdf', 'path/to/datasheet2.pdf'])  
dataset.augment\_data()  
  
# Fine-tune model  
finetuner = OCRModelFineTuner({  
 'engine': 'nougat',  
 'domain': 'material\_datasheets',  
 'model\_type': 'base'  
})  
result = finetuner.finetune(dataset)  
  
# Export model  
finetuner.export\_model('models/fine\_tuned\_nougat')

#### 2. Extensible Engine Framework

The extensible\_engine\_manager.py module provides a framework for integrating additional document understanding models:

extensible\_engine\_manager.py

* Pluggable Architecture: Interface for adding new OCR engines
* Adapter Pattern: Standardized adapter for third-party libraries
* Dynamic Discovery: Automatic detection of available engines
* Engine Pipeline: Multi-stage processing pipelines with condition branching

# Example: Registering a custom engine  
from extensible\_engine\_manager import EngineManager, OCREngineInterface  
  
# Create engine manager  
manager = EngineManager({  
 'plugin\_directory': 'custom\_engines'  
})  
  
# Register custom engine  
manager.register\_engine\_class("custom\_engine", CustomEngineClass)  
  
# Use with fallback chain  
result = manager.process\_with\_fallback("path/to/document.pdf")

# Example: Registering a custom engine  
from extensible\_engine\_manager import EngineManager, OCREngineInterface  
  
# Create engine manager  
manager = EngineManager({  
 'plugin\_directory': 'custom\_engines'  
})  
  
# Register custom engine  
manager.register\_engine\_class("custom\_engine", CustomEngineClass)  
  
# Use with fallback chain  
result = manager.process\_with\_fallback("path/to/document.pdf")

#### 3. Distributed Processing

The distributed\_ocr\_processing.py module enables scalable document processing across multiple nodes:

distributed\_ocr\_processing.py

* Task Distribution: Distributes OCR tasks across multiple workers
* Load Balancing: Optimizes resource utilization across the cluster
* Fault Tolerance: Automatic recovery from worker failures
* Priority Queuing: Processes critical documents with higher priority

# Example: Distributed document processing  
from distributed\_ocr\_processing import TaskManager  
  
# Initialize task manager  
task\_manager = TaskManager({  
 'queue\_type': 'redis',  
 'redis\_host': 'localhost',  
 'workers': 8  
})  
  
# Submit batch processing task  
task\_ids = task\_manager.submit\_batch([  
 {'document\_path': 'doc1.pdf', 'engine': 'nougat'},  
 {'document\_path': 'doc2.pdf', 'engine': 'marker'},  
 {'document\_path': 'doc3.pdf', 'engine': 'thepipe'}  
])  
  
# Retrieve results  
results = task\_manager.get\_batch\_results(task\_ids, wait=True)

# Example: Distributed document processing  
from distributed\_ocr\_processing import TaskManager  
  
# Initialize task manager  
task\_manager = TaskManager({  
 'queue\_type': 'redis',  
 'redis\_host': 'localhost',  
 'workers': 8  
})  
  
# Submit batch processing task  
task\_ids = task\_manager.submit\_batch([  
 {'document\_path': 'doc1.pdf', 'engine': 'nougat'},  
 {'document\_path': 'doc2.pdf', 'engine': 'marker'},  
 {'document\_path': 'doc3.pdf', 'engine': 'thepipe'}  
])  
  
# Retrieve results  
results = task\_manager.get\_batch\_results(task\_ids, wait=True)

#### 4. Enhanced Layout Analysis

The enhanced\_layout\_analyzer.py module provides advanced document layout understanding:

enhanced\_layout\_analyzer.py

* Multi-column Detection: Accurate handling of complex multi-column layouts
* Table Structure Analysis: Detailed table structure with merged cell detection
* Hierarchical Section Analysis: Document section hierarchy recognition
* Reading Order Determination: Correct reading order for non-linear layouts
* Material-specific Templates: Specialized templates for material documentation

# Example: Advanced layout analysis  
from enhanced\_layout\_analyzer import EnhancedLayoutAnalyzer  
  
# Initialize analyzer  
analyzer = EnhancedLayoutAnalyzer({  
 'layout\_mode': 'deep',  
 'material\_specific\_templates': True  
})  
  
# Analyze document layout  
layout = analyzer.analyze\_document("path/to/catalog.pdf")  
  
# Process each element by type  
for page in layout['pages']:  
 for element in page['elements']:  
 if element['element\_type'] == 'table':  
 # Process table structure  
 table\_structure = element['attributes']['structure']  
 elif element['element\_type'].startswith('material\_'):  
 # Process material-specific region  
 region\_type = element['element\_type'].replace('material\_', '')

# Example: Advanced layout analysis  
from enhanced\_layout\_analyzer import EnhancedLayoutAnalyzer  
  
# Initialize analyzer  
analyzer = EnhancedLayoutAnalyzer({  
 'layout\_mode': 'deep',  
 'material\_specific\_templates': True  
})  
  
# Analyze document layout  
layout = analyzer.analyze\_document("path/to/catalog.pdf")  
  
# Process each element by type  
for page in layout['pages']:  
 for element in page['elements']:  
 if element['element\_type'] == 'table':  
 # Process table structure  
 table\_structure = element['attributes']['structure']  
 elif element['element\_type'].startswith('material\_'):  
 # Process material-specific region  
 region\_type = element['element\_type'].replace('material\_', '')

#### 5. Material-specific Processing

The material\_specific\_processor.py module provides specialized extraction for material catalogs:

material\_specific\_processor.py

* Material Code Recognition: Advanced detection and normalization of material codes
* Technical Specification Extraction: Domain-specific property extraction
* Dimensional Information Parsing: Intelligent dimension extraction and normalization
* Material-specific Validation: Domain knowledge for validation and correction
* Cross-reference Detection: Recognition of related materials

# Example: Material-specific processing  
from material\_specific\_processor import MaterialSpecificProcessor  
  
# Initialize processor  
processor = MaterialSpecificProcessor()  
  
# Process material document with OCR results  
document = processor.process\_document(  
 "path/to/datasheet.pdf",   
 ocr\_result=ocr\_result  
)  
  
# Access extracted fields  
dimensions = {  
 'width': document.fields['width'].value,  
 'length': document.fields['length'].value,  
 'thickness': document.fields['thickness'].value  
}  
  
# Access technical properties  
if 'pei\_rating' in document.fields:  
 pei\_rating = document.fields['pei\_rating'].value  
  
# Export structured data  
structured\_data = processor.export\_document(document, 'json')

# Example: Material-specific processing  
from material\_specific\_processor import MaterialSpecificProcessor  
  
# Initialize processor  
processor = MaterialSpecificProcessor()  
  
# Process material document with OCR results  
document = processor.process\_document(  
 "path/to/datasheet.pdf",   
 ocr\_result=ocr\_result  
)  
  
# Access extracted fields  
dimensions = {  
 'width': document.fields['width'].value,  
 'length': document.fields['length'].value,  
 'thickness': document.fields['thickness'].value  
}  
  
# Access technical properties  
if 'pei\_rating' in document.fields:  
 pei\_rating = document.fields['pei\_rating'].value  
  
# Export structured data  
structured\_data = processor.export\_document(document, 'json')

### Integration with Existing System

The new enhancements have been fully integrated with the existing OCR system:

1. Orchestrator Integration: All enhancements are accessible through the Neural OCR Orchestrator
2. Configuration-driven: Features can be enabled/disabled through configuration
3. Backward Compatibility: Maintains API compatibility with existing systems
4. Progressive Enhancement: Gracefully falls back when enhancements are unavailable

#### Example: Integrated Usage

from neural\_ocr\_orchestrator import NeuralOCROrchestrator  
  
# Initialize with all enhancements  
orchestrator = NeuralOCROrchestrator({  
 'enable\_model\_finetuning': True,  
 'enable\_distributed\_processing': True,  
 'enable\_enhanced\_layout': True,  
 'enable\_material\_specific': True,  
 'material\_types': ['tile', 'stone', 'wood'],  
 'distributed': {  
 'queue\_type': 'redis',  
 'workers': 4  
 }  
})  
  
# Process material catalog with all enhancements  
result = orchestrator.process\_document("path/to/catalog.pdf")  
  
# Access structured material data  
materials = result['structured\_content']['materials']

from neural\_ocr\_orchestrator import NeuralOCROrchestrator  
  
# Initialize with all enhancements  
orchestrator = NeuralOCROrchestrator({  
 'enable\_model\_finetuning': True,  
 'enable\_distributed\_processing': True,  
 'enable\_enhanced\_layout': True,  
 'enable\_material\_specific': True,  
 'material\_types': ['tile', 'stone', 'wood'],  
 'distributed': {  
 'queue\_type': 'redis',  
 'workers': 4  
 }  
})  
  
# Process material catalog with all enhancements  
result = orchestrator.process\_document("path/to/catalog.pdf")  
  
# Access structured material data  
materials = result['structured\_content']['materials']

### Performance Improvements

The implemented enhancements provide significant performance improvements:

1. Accuracy Improvements:
2. 25-40% better accuracy on complex material datasheets
3. 50-70% better structure preservation for multi-column catalogs
4. 30-50% improved extraction of technical specifications
5. Processing Scalability:
6. 5-10x throughput with distributed processing
7. Efficient handling of large document collections
8. Prioritization of urgent processing requests
9. Resource Optimization:
10. Intelligent resource allocation based on document complexity
11. Caching and result reuse for similar documents
12. Progressive loading of models based on document needs

30-50% improved extraction of technical specifications

Processing Scalability:

Prioritization of urgent processing requests

Resource Optimization:

### Multilingual Capabilities

With the addition of PaddleOCR and other advanced engines, the system now offers robust multilingual support:

#### Supported Languages

The enhanced OCR system now supports 80+ languages including:

* All Latin-based languages (English, Spanish, French, etc.)
* Asian languages (Chinese, Japanese, Korean)
* Right-to-left languages (Arabic, Hebrew)
* Cyrillic languages (Russian, Ukrainian, etc.)
* Indic languages (Hindi, Tamil, etc.)
* Southeast Asian languages (Thai, Vietnamese, etc.)

#### Language Detection and Routing

The system automatically detects document language and routes to the appropriate engine:

1. Language Detection:
2. Analyzes text regions for language identification
3. Handles mixed-language documents by region
4. Identifies script types (Latin, Cyrillic, etc.)
5. Engine Selection:
6. PaddleOCR for non-Latin scripts and multilingual content
7. Nougat or surya for technical content regardless of language
8. Language-specific models loaded as needed
9. Mixed-Language Processing:
10. Processes each region with appropriate language model
11. Preserves language tags in extracted text
12. Maintains relationships between regions in different languages

Identifies script types (Latin, Cyrillic, etc.)

Engine Selection:

Language-specific models loaded as needed

Mixed-Language Processing:

#### International Material Catalog Processing

This enables processing of material catalogs from international suppliers:

* Chinese ceramic tile specifications
* European material datasheets with multiple EU languages
* Technical documentation with scientific notation in any language
* Cross-referencing between language versions of the same material

### Native PDF Processing

The addition of pdfdeal enables direct extraction from vector PDFs:

#### Digital PDF Advantages

Direct extraction provides several advantages:

1. Higher Accuracy:
2. Extracts exact text without OCR errors
3. Preserves original formatting precisely
4. Maintains font information and styles
5. Improved Performance:
6. 5-10x faster than image-based OCR for digital PDFs
7. Reduced resource requirements
8. Higher throughput for large document collections
9. Enhanced Structure Recognition:
10. Better preservation of document structure
11. More accurate table extraction
12. Improved form field detection

Maintains font information and styles

Improved Performance:

Higher throughput for large document collections

Enhanced Structure Recognition:

#### Hybrid Processing Approach

The system utilizes a hybrid approach for mixed documents:

1. Document Analysis:
2. Determines which pages contain digital text vs. scanned images
3. Identifies native vector elements (text, tables, forms)
4. Maps document structure for selective processing
5. Selective Processing:
6. Uses pdfdeal for digital text extraction
7. Falls back to OCR engines for scanned or image portions
8. Combines results maintaining original document structure
9. Validation Flow:
10. Uses higher confidence of native extraction where available
11. Cross-references native text with OCR results in mixed documents
12. Preserves original PDF formatting in extracted data

Maps document structure for selective processing

Selective Processing:

Combines results maintaining original document structure

Validation Flow:

### Multimodal Document Understanding

The inclusion of mPLUG-DocOwl brings advanced multimodal capabilities:

#### Image-Text Relationships

The system now understands relationships between visual and textual elements:

1. Content Association:
2. Links product images with corresponding specifications
3. Understands image captions and references
4. Maps diagrams to related technical descriptions
5. Visual Context Understanding:
6. Interprets visual elements in context of surrounding text
7. Understands product variations shown in images
8. Identifies related products in catalog layouts
9. Question Answering:
10. Enables queries about specific material properties
11. Can extract information based on semantic understanding
12. Understands implicit relationships not explicitly stated

Maps diagrams to related technical descriptions

Visual Context Understanding:

Identifies related products in catalog layouts

Question Answering:

#### Enhanced Material Catalog Experience

This enables rich processing of catalog content:

* "Show me all variations of this tile pattern"
* "Find specifications for the product shown in this image"
* "Identify all materials with similar visual characteristics"
* "Extract all technical parameters for products with this finish"

### Conclusion

The neural OCR integration with these additional engines further transforms the platform's document understanding capabilities. By incorporating PaddleOCR, pdfdeal, surya, and mPLUG-DocOwl alongside the previously implemented enhancements, the system achieves unprecedented accuracy, language coverage, and semantic understanding for material documentation processing.

This comprehensive implementation enables automated extraction of structured data from even the most complex material datasheets and catalogs in any language, dramatically reducing manual data entry and enabling more comprehensive knowledge base population. The addition of native PDF processing and multimodal understanding takes the system beyond traditional OCR to true document comprehension.

# Notification Webhook System

Source: readme/notification-webhook-system.md

---

## Notification and Webhook System

The Notification and Webhook System provides a comprehensive messaging framework for the KAI platform, enabling communication through multiple channels including in-app notifications, email, SMS, and webhook integrations with external systems.

### Overview

The system supports:

1. Multi-Channel Notifications - Send messages through in-app, email, SMS, and webhook channels
2. Template-Based Messaging - Configurable templates with Handlebars support
3. User Preference Management - Honor user notification preferences
4. Event-Driven Architecture - Trigger notifications based on system events
5. Webhook Integration - Allow external systems to receive notifications
6. Delivery Tracking - Monitor notification delivery status

This unified notification infrastructure enables consistent communication across the platform while respecting user preferences and providing robust integration options for external systems.

### Architecture

The system is built with a modular architecture:

Notification & Webhook System  
├── Core Services  
│ ├── Notification Service  
│ ├── Template Service  
│ ├── Preference Service  
│ └── Event Notification Service  
├── Delivery Providers  
│ ├── Email Provider  
│ ├── SMS Provider  
│ ├── In-App Provider  
│ └── Webhook Provider  
├── Configuration  
│ ├── Template Configuration  
│ ├── Channel Configuration  
│ └── User Preferences  
└── Management Interfaces  
 ├── User Preference UI  
 ├── Admin Template Editor  
 └── Admin Webhook Manager

Notification & Webhook System  
├── Core Services  
│ ├── Notification Service  
│ ├── Template Service  
│ ├── Preference Service  
│ └── Event Notification Service  
├── Delivery Providers  
│ ├── Email Provider  
│ ├── SMS Provider  
│ ├── In-App Provider  
│ └── Webhook Provider  
├── Configuration  
│ ├── Template Configuration  
│ ├── Channel Configuration  
│ └── User Preferences  
└── Management Interfaces  
 ├── User Preference UI  
 ├── Admin Template Editor  
 └── Admin Webhook Manager

### Key Components

#### Notification Service

The core service responsible for routing and delivering messages through appropriate channels.

Features:  
- Message formatting with templates  
- Channel selection based on message type and user preferences  
- Delivery tracking and retry mechanism  
- Batching capabilities for bulk notifications

#### Template Service

Manages notification templates with support for dynamic content.

Features:  
- Handlebars-based template system  
- Multi-language support  
- Version control for templates  
- Preview functionality for testing

#### Webhook Service

Manages outgoing webhook notifications to external systems.

Features:  
- Webhook registration and management  
- Payload signing for security  
- Delivery confirmation  
- Retry mechanism with exponential backoff  
- Detailed delivery logs

### API Reference

#### Notification API

// Send an email notification  
await notificationService.sendEmail({  
 to: 'user@example.com',  
 subject: 'Welcome to KAI',  
 text: 'Thank you for joining our platform!',  
 html: '<p>Thank you for joining our platform!</p>',  
 userId: 'user-123', // Optional for tracking  
 eventType: 'user.registered' // Optional event type  
});  
  
// Send an SMS notification  
await notificationService.sendSMS({  
 to: '+1234567890',  
 message: 'Your verification code is: 123456',  
 userId: 'user-123',  
 eventType: 'verification.code'  
});  
  
// Send an in-app notification  
await notificationService.sendInAppNotification({  
 userId: 'user-123',  
 title: 'New Message',  
 message: 'You have a new message from Admin',  
 type: 'info', // 'info', 'warning', 'error', 'success'  
 actionUrl: '/messages/123' // Optional deep link  
});  
  
// Process an event that may trigger notifications  
await eventNotificationService.processEvent({  
 eventType: EventType.SUBSCRIPTION\_PAYMENT\_FAILED,  
 userId: 'user-123',  
 data: {  
 subscriptionId: 'sub-123',  
 failureReason: 'insufficient\_funds'  
 }  
});

// Send an email notification  
await notificationService.sendEmail({  
 to: 'user@example.com',  
 subject: 'Welcome to KAI',  
 text: 'Thank you for joining our platform!',  
 html: '<p>Thank you for joining our platform!</p>',  
 userId: 'user-123', // Optional for tracking  
 eventType: 'user.registered' // Optional event type  
});  
  
// Send an SMS notification  
await notificationService.sendSMS({  
 to: '+1234567890',  
 message: 'Your verification code is: 123456',  
 userId: 'user-123',  
 eventType: 'verification.code'  
});  
  
// Send an in-app notification  
await notificationService.sendInAppNotification({  
 userId: 'user-123',  
 title: 'New Message',  
 message: 'You have a new message from Admin',  
 type: 'info', // 'info', 'warning', 'error', 'success'  
 actionUrl: '/messages/123' // Optional deep link  
});  
  
// Process an event that may trigger notifications  
await eventNotificationService.processEvent({  
 eventType: EventType.SUBSCRIPTION\_PAYMENT\_FAILED,  
 userId: 'user-123',  
 data: {  
 subscriptionId: 'sub-123',  
 failureReason: 'insufficient\_funds'  
 }  
});

#### Webhook API

// Send a webhook notification  
await notificationService.sendWebhook({  
 url: 'https://example.com/webhook',  
 payload: {  
 event: 'order.created',  
 data: {  
 orderId: 'order-123',  
 amount: 99.99  
 }  
 },  
 headers: { // Optional custom headers  
 'X-Custom-Header': 'custom-value'  
 }  
});  
  
// Register a webhook configuration  
const webhook = await webhookService.createWebhookConfiguration({  
 name: 'Order Processing Webhook',  
 url: 'https://example.com/webhook',  
 events: ['order.created', 'order.updated', 'order.cancelled'],  
 isActive: true,  
 secretKey: 'generate\_new' // Generates a new secret key  
});  
  
// Test a webhook configuration  
const testResult = await webhookService.testWebhookConfiguration(  
 webhookId,  
 {  
 event: 'order.created',  
 data: { test: true, timestamp: Date.now() }  
 }  
);

// Send a webhook notification  
await notificationService.sendWebhook({  
 url: 'https://example.com/webhook',  
 payload: {  
 event: 'order.created',  
 data: {  
 orderId: 'order-123',  
 amount: 99.99  
 }  
 },  
 headers: { // Optional custom headers  
 'X-Custom-Header': 'custom-value'  
 }  
});  
  
// Register a webhook configuration  
const webhook = await webhookService.createWebhookConfiguration({  
 name: 'Order Processing Webhook',  
 url: 'https://example.com/webhook',  
 events: ['order.created', 'order.updated', 'order.cancelled'],  
 isActive: true,  
 secretKey: 'generate\_new' // Generates a new secret key  
});  
  
// Test a webhook configuration  
const testResult = await webhookService.testWebhookConfiguration(  
 webhookId,  
 {  
 event: 'order.created',  
 data: { test: true, timestamp: Date.now() }  
 }  
);

### REST API Endpoints

The system exposes the following RESTful API endpoints:

#### Notification Endpoints

Table content:

Method | Endpoint | Description

GET | /api/notifications | Get user's in-app notifications

POST | /api/notifications/mark-as-read | Mark notifications as read

POST | /api/notifications/mark-all-as-read | Mark all notifications as read

DELETE | /api/notifications/:id | Delete a notification

GET | /api/notifications/unread-count | Get unread notification count

GET | /api/notifications/preferences | Get notification preferences

PUT | /api/notifications/preferences | Update notification preferences

POST | /api/notifications/test | Send a test notification

/api/notifications

/api/notifications/mark-as-read

/api/notifications/mark-all-as-read

/api/notifications/:id

/api/notifications/unread-count

/api/notifications/preferences

/api/notifications/preferences

/api/notifications/test

#### Webhook Endpoints

Table content:

Method | Endpoint | Description

GET | /api/webhooks/configurations | Get webhook configurations

GET | /api/webhooks/configurations/:id | Get a webhook configuration

POST | /api/webhooks/configurations | Create a webhook configuration

PUT | /api/webhooks/configurations/:id | Update a webhook configuration

DELETE | /api/webhooks/configurations/:id | Delete a webhook configuration

GET | /api/webhooks/configurations/:id/logs | Get webhook delivery logs

POST | /api/webhooks/configurations/:id/test | Test a webhook configuration

POST | /api/webhooks/configurations/:id/regenerate-secret | Regenerate webhook secret

/api/webhooks/configurations

/api/webhooks/configurations/:id

/api/webhooks/configurations

/api/webhooks/configurations/:id

/api/webhooks/configurations/:id

/api/webhooks/configurations/:id/logs

/api/webhooks/configurations/:id/test

/api/webhooks/configurations/:id/regenerate-secret

#### Admin Endpoints

Table content:

Method | Endpoint | Description

GET | /api/admin/notifications/templates | Get notification templates

GET | /api/admin/notifications/templates/:id | Get a notification template

POST | /api/admin/notifications/templates | Create a notification template

PUT | /api/admin/notifications/templates/:id | Update a notification template

DELETE | /api/admin/notifications/templates/:id | Delete a notification template

POST | /api/admin/notifications/templates/:id/test | Test a notification template

GET | /api/admin/notifications/logs | Get notification logs

GET | /api/admin/webhooks/logs | Get all webhook delivery logs

/api/admin/notifications/templates

/api/admin/notifications/templates/:id

/api/admin/notifications/templates

/api/admin/notifications/templates/:id

/api/admin/notifications/templates/:id

/api/admin/notifications/templates/:id/test

/api/admin/notifications/logs

/api/admin/webhooks/logs

### Database Schema

The notification system uses the following database tables:

#### Notification Tables

notification\_templates  
├── id: UUID (PK)  
├── name: String  
├── description: String  
├── type: Enum('email', 'sms', 'in\_app', 'webhook')  
├── subject: String (for email)  
├── content\_text: String (plain text version)  
├── content\_html: String (HTML version for email)  
├── variables: JSONB (template variables)  
├── created\_at: Timestamp  
├── updated\_at: Timestamp  
└── version: Integer  
  
notifications  
├── id: UUID (PK)  
├── user\_id: UUID (FK to users)  
├── title: String  
├── message: String  
├── type: Enum('info', 'warning', 'error', 'success')  
├── action\_url: String (optional deep link)  
├── read: Boolean  
├── created\_at: Timestamp  
└── metadata: JSONB  
  
notification\_logs  
├── id: UUID (PK)  
├── user\_id: UUID (FK to users)  
├── template\_id: UUID (FK to notification\_templates)  
├── channel: Enum('email', 'sms', 'in\_app', 'webhook')  
├── status: Enum('sent', 'delivered', 'failed')  
├── error: String (if failed)  
├── metadata: JSONB  
└── created\_at: Timestamp  
  
user\_notification\_preferences  
├── user\_id: UUID (PK, FK to users)  
├── email\_enabled: Boolean  
├── sms\_enabled: Boolean  
├── in\_app\_enabled: Boolean  
├── push\_enabled: Boolean  
├── preferences: JSONB (specific preferences by notification type)  
└── updated\_at: Timestamp

notification\_templates  
├── id: UUID (PK)  
├── name: String  
├── description: String  
├── type: Enum('email', 'sms', 'in\_app', 'webhook')  
├── subject: String (for email)  
├── content\_text: String (plain text version)  
├── content\_html: String (HTML version for email)  
├── variables: JSONB (template variables)  
├── created\_at: Timestamp  
├── updated\_at: Timestamp  
└── version: Integer  
  
notifications  
├── id: UUID (PK)  
├── user\_id: UUID (FK to users)  
├── title: String  
├── message: String  
├── type: Enum('info', 'warning', 'error', 'success')  
├── action\_url: String (optional deep link)  
├── read: Boolean  
├── created\_at: Timestamp  
└── metadata: JSONB  
  
notification\_logs  
├── id: UUID (PK)  
├── user\_id: UUID (FK to users)  
├── template\_id: UUID (FK to notification\_templates)  
├── channel: Enum('email', 'sms', 'in\_app', 'webhook')  
├── status: Enum('sent', 'delivered', 'failed')  
├── error: String (if failed)  
├── metadata: JSONB  
└── created\_at: Timestamp  
  
user\_notification\_preferences  
├── user\_id: UUID (PK, FK to users)  
├── email\_enabled: Boolean  
├── sms\_enabled: Boolean  
├── in\_app\_enabled: Boolean  
├── push\_enabled: Boolean  
├── preferences: JSONB (specific preferences by notification type)  
└── updated\_at: Timestamp

#### Webhook Tables

webhook\_configurations  
├── id: UUID (PK)  
├── name: String  
├── url: String  
├── events: String[] (array of event types)  
├── is\_active: Boolean  
├── secret\_key: String  
├── headers: JSONB (custom headers)  
├── created\_at: Timestamp  
├── updated\_at: Timestamp  
└── metadata: JSONB  
  
webhook\_delivery\_logs  
├── id: UUID (PK)  
├── webhook\_id: UUID (FK to webhook\_configurations)  
├── event\_type: String  
├── payload: JSONB  
├── status\_code: Integer  
├── response: String  
├── success: Boolean  
├── attempt\_count: Integer  
├── error: String (if failed)  
├── timestamp: Timestamp  
└── duration\_ms: Integer

webhook\_configurations  
├── id: UUID (PK)  
├── name: String  
├── url: String  
├── events: String[] (array of event types)  
├── is\_active: Boolean  
├── secret\_key: String  
├── headers: JSONB (custom headers)  
├── created\_at: Timestamp  
├── updated\_at: Timestamp  
└── metadata: JSONB  
  
webhook\_delivery\_logs  
├── id: UUID (PK)  
├── webhook\_id: UUID (FK to webhook\_configurations)  
├── event\_type: String  
├── payload: JSONB  
├── status\_code: Integer  
├── response: String  
├── success: Boolean  
├── attempt\_count: Integer  
├── error: String (if failed)  
├── timestamp: Timestamp  
└── duration\_ms: Integer

### Notification Templates

The system supports the following notification template types:

#### Email Templates

Email templates include both HTML and plain text versions with the following features:  
- Responsive email design using MJML  
- Support for deep links  
- Tracking pixel support (optional)  
- Unsubscribe links  
- Localization support

Example email template:

Subject: {{subject}}  
  
<mjml>  
 <mj-body>  
 <mj-section>  
 <mj-column>  
 <mj-text>  
 Hello {{user.firstName}},  
  
 {{message}}  
  
 {{#if actionUrl}}  
 <mj-button href="{{actionUrl}}">{{actionLabel}}</mj-button>  
 {{/if}}  
 </mj-text>  
 </mj-column>  
 </mj-section>  
 </mj-body>  
</mjml>

Subject: {{subject}}  
  
<mjml>  
 <mj-body>  
 <mj-section>  
 <mj-column>  
 <mj-text>  
 Hello {{user.firstName}},  
  
 {{message}}  
  
 {{#if actionUrl}}  
 <mj-button href="{{actionUrl}}">{{actionLabel}}</mj-button>  
 {{/if}}  
 </mj-text>  
 </mj-column>  
 </mj-section>  
 </mj-body>  
</mjml>

#### SMS Templates

SMS templates support:  
- Short text messages  
- Variable substitution  
- Link shortening  
- Character count optimization

Example SMS template:

{{message}} {{#if code}}Your code: {{code}}{{/if}} {{#if url}}Details: {{url}}{{/if}}

{{message}} {{#if code}}Your code: {{code}}{{/if}} {{#if url}}Details: {{url}}{{/if}}

#### In-App Templates

In-app notification templates support:  
- Different notification types (info, warning, error, success)  
- Action buttons and deep links  
- Grouping and prioritization  
- Expiration settings

Example in-app template:

{  
 "title": "{{title}}",  
 "message": "{{message}}",  
 "type": "{{type}}",  
 "actionUrl": "{{actionUrl}}",  
 "actionLabel": "{{actionLabel}}",  
 "expireAfter": "{{expireAfter}}"  
}

{  
 "title": "{{title}}",  
 "message": "{{message}}",  
 "type": "{{type}}",  
 "actionUrl": "{{actionUrl}}",  
 "actionLabel": "{{actionLabel}}",  
 "expireAfter": "{{expireAfter}}"  
}

#### Webhook Templates

Webhook notification templates support:  
- Customizable JSON payloads  
- Signature generation for security  
- Metadata inclusion

Example webhook template:

{  
 "event": "{{event}}",  
 "timestamp": "{{timestamp}}",  
 "data": {{toJson data}},  
 "metadata": {  
 "platform": "KAI",  
 "version": "{{version}}"  
 }  
}

{  
 "event": "{{event}}",  
 "timestamp": "{{timestamp}}",  
 "data": {{toJson data}},  
 "metadata": {  
 "platform": "KAI",  
 "version": "{{version}}"  
 }  
}

### Webhook Integration

The webhook system provides robust integration with external systems:

#### Security

Webhooks are secured using these methods:  
- HMAC-SHA256 payload signing  
- Secret key management  
- HTTPS endpoints only  
- Rate limiting

#### Payload Format

Standard webhook payload format:

{  
 "event": "order.created",  
 "timestamp": "2025-04-19T10:30:00Z",  
 "data": {  
 "orderId": "order-123",  
 "customerId": "cust-456",  
 "amount": 99.99,  
 "items": [...]  
 },  
 "metadata": {  
 "platform": "KAI",  
 "version": "1.0"  
 }  
}

{  
 "event": "order.created",  
 "timestamp": "2025-04-19T10:30:00Z",  
 "data": {  
 "orderId": "order-123",  
 "customerId": "cust-456",  
 "amount": 99.99,  
 "items": [...]  
 },  
 "metadata": {  
 "platform": "KAI",  
 "version": "1.0"  
 }  
}

#### Delivery

The webhook delivery system ensures reliable notification:  
- Automatic retries with exponential backoff  
- Configurable retry count and intervals  
- Detailed delivery logs  
- Success/failure tracking

#### Registration

External systems can register webhooks through:  
- Admin interface  
- API endpoints  
- Programmatic creation

### User Preference Management

The system respects user notification preferences with:

#### Preference Levels

* Global Preferences: Master switches for each channel
* Category Preferences: Settings for notification categories
* Individual Preferences: Fine-grained control for specific notification types

#### User Interface

The user preference interface allows:  
- Channel enabling/disabling  
- Time-based restrictions (quiet hours)  
- Frequency controls  
- Priority settings

#### Enforcement

Preferences are enforced at multiple levels:  
- During event processing  
- At notification generation  
- Before channel delivery

### Event-Based Notification System

The system uses an event-driven architecture:

#### Event Types

Common event types include:  
- user.registered - New user registration  
- order.created - New order placed  
- subscription.renewed - Subscription renewal  
- subscription.payment\_failed - Payment failure  
- material.recognized - Successful material recognition

user.registered

order.created

subscription.renewed

subscription.payment\_failed

material.recognized

#### Event Processing

Events flow through the system:  
1. Event is triggered by an action in the system  
2. Event is processed by the event notification service  
3. Notification rules are evaluated against the event  
4. Appropriate notifications are generated based on rules and preferences  
5. Notifications are delivered through selected channels

#### Rule Configuration

Rules determine when and how notifications are sent:  
- Condition-based triggers  
- Dynamic template selection  
- Channel routing logic  
- User targeting

### Integration with Other Systems

The notification system integrates with:

#### Authentication System

* User identity management
* Permission checks for notification access
* Session tracking for delivery

#### Subscription System

* Notification triggers based on subscription events
* Tier-based notification features
* Delivery channel access based on subscription level

#### Analytics System

* Notification engagement tracking
* Delivery performance monitoring
* A/B testing of notification content

### Administration Interface

The admin interface allows management of:

#### Template Management

* Create, edit and version templates
* Test templates with sample data
* View usage statistics

#### Webhook Configuration

* Register and manage webhook endpoints
* Monitor delivery status
* Test webhook delivery

#### Notification Logs

* View delivery status for all notifications
* Filter by user, template, channel, and status
* Export logs for analysis

### Configuration

The system can be configured through environment variables:

#### Email Configuration

EMAIL\_SERVICE=smtp  
EMAIL\_HOST=smtp.example.com  
EMAIL\_PORT=587  
EMAIL\_SECURE=false  
EMAIL\_USER=username  
EMAIL\_PASSWORD=password  
EMAIL\_FROM=noreply@example.com

EMAIL\_SERVICE=smtp  
EMAIL\_HOST=smtp.example.com  
EMAIL\_PORT=587  
EMAIL\_SECURE=false  
EMAIL\_USER=username  
EMAIL\_PASSWORD=password  
EMAIL\_FROM=noreply@example.com

#### SMS Configuration

SMS\_PROVIDER=twilio  
TWILIO\_ACCOUNT\_SID=your\_account\_sid  
TWILIO\_AUTH\_TOKEN=your\_auth\_token  
TWILIO\_PHONE\_NUMBER=+1234567890

SMS\_PROVIDER=twilio  
TWILIO\_ACCOUNT\_SID=your\_account\_sid  
TWILIO\_AUTH\_TOKEN=your\_auth\_token  
TWILIO\_PHONE\_NUMBER=+1234567890

#### Webhook Configuration

WEBHOOK\_TIMEOUT=5000  
WEBHOOK\_MAX\_RETRIES=3  
WEBHOOK\_RETRY\_DELAY=1000

WEBHOOK\_TIMEOUT=5000  
WEBHOOK\_MAX\_RETRIES=3  
WEBHOOK\_RETRY\_DELAY=1000

### Implementation Details

#### Email Provider

The system supports multiple email providers:  
- SMTP servers  
- SendGrid  
- Mailgun  
- AWS SES

#### SMS Provider

SMS delivery is supported through:  
- Twilio  
- Nexmo  
- AWS SNS

#### In-App Notifications

In-app notifications are delivered via:  
- WebSocket for real-time updates  
- Polling API for fallback  
- Local storage for offline access

#### Webhook Delivery

Webhook notifications are delivered using:  
- HTTP/HTTPS POST requests  
- Configurable timeout settings  
- Automatic retry logic

### Performance Considerations

The notification system is designed for high throughput with:

#### Scalability

* Queue-based processing for asynchronous delivery
* Horizontal scaling support
* Database sharding for high-volume deployments

#### Reliability

* Delivery confirmation and tracking
* Retry mechanisms for failed deliveries
* Fallback channels when primary channels fail

#### Efficiency

* Batch processing for high-volume notifications
* Template caching
* Database query optimization

### Security Features

The notification system implements several security features:

#### Authentication

* Secure API endpoints
* User verification for preferences
* Admin authentication for template management

#### Privacy

* PII handling according to regulations
* Obfuscation in logs
* Data minimization in payloads

#### Data Protection

* Encryption of sensitive template data
* Secure storage of API keys
* Audit logging for all operations

### Future Enhancements

Planned enhancements include:

1. Push Notification Support: Native mobile push notifications
2. Message Scheduling: Time-based delivery options
3. A/B Testing: Test different message formats for engagement
4. Advanced Analytics: Detailed metrics on notification performance
5. AI-Generated Content: Smart template population based on context

### Best Practices

#### Notification Design

* Keep messages concise and clear
* Use consistent formatting across channels
* Include actionable information
* Respect user attention

#### Webhook Implementation

* Implement idempotent processing
* Use appropriate HTTP status codes
* Handle retries gracefully
* Verify webhook signatures

#### Template Management

* Use version control for templates
* Test templates before deployment
* Include fallback content for each template
* Document variables used in templates

### Troubleshooting

Common issues and solutions:

#### Notification Delivery Issues

* Check user notification preferences
* Verify channel configuration
* Check delivery logs for errors
* Ensure valid recipient information

#### Webhook Failures

* Verify endpoint availability
* Check payload format and signature
* Examine response codes and errors
* Ensure proper webhook configuration

#### Template Rendering Problems

* Validate variable names and formats
* Check for missing variables
* Test with sample data
* Verify template syntax

# Ocr

Source: readme/ocr.md

---

## Optical Character Recognition (OCR) System

The OCR System is a specialized component of Kai that extracts and processes text from images, PDFs, and material catalogs. This document details the system's capabilities, architecture, and implementation.

### Features

#### Specialized OCR for Material Datasheets

The OCR system includes specialized capabilities for material datasheets:

1. Domain-Specific Optimization
2. Technical vocabulary enhancement for materials (tile, stone, wood, etc.)
3. Specification format recognition
4. Material code and identification extraction
5. Dimensional data parsing
6. Technical symbol recognition
7. Region-Based OCR
8. Context-aware processing for different document regions
9. Optimal parameter selection based on content type
10. Region-specific preprocessing techniques
11. Custom dictionaries for different document sections
12. Text block classification (headings, specifications, captions)
13. Enhanced Preprocessing
14. Image deskewing and orientation correction
15. Noise reduction and artifact removal
16. Contrast enhancement for low-quality scans
17. Background removal and text isolation
18. Resolution optimization for OCR accuracy

Technical symbol recognition

Region-Based OCR

Text block classification (headings, specifications, captions)

Enhanced Preprocessing

#### Multi-Language and Text Recognition

The system supports advanced text recognition across languages:

1. Language Support
2. Multi-language detection and processing
3. Primary language identification
4. Mixed language document handling
5. Language-specific optimization
6. Character set and encoding management
7. Handwriting Recognition
8. Handwritten note extraction from technical documents
9. Signature identification and processing
10. Margin annotation recognition
11. Confidence scoring for handwritten content
12. Context-based interpretation
13. Special Content Processing
14. Table structure recognition
15. Form field extraction
16. List and bullet point processing
17. Symbol and icon detection
18. Mathematical formula recognition

Character set and encoding management

Handwriting Recognition

Context-based interpretation

Special Content Processing

#### Confidence Scoring and Quality Assurance

The system includes comprehensive quality assessment:

1. OCR Confidence Metrics
2. Character-level confidence scoring
3. Word and phrase reliability assessment
4. Region-specific quality evaluation
5. Alternative interpretations for low-confidence text
6. Statistical analysis of extraction quality
7. Post-Processing Rules
8. Domain-specific error correction
9. Common OCR error pattern resolution
10. Technical term standardization
11. Unit format normalization
12. Contextual validation and correction
13. Validation Techniques
14. Pattern-based verification
15. Dictionary matching
16. Contextual analysis
17. Format validation
18. Human-in-the-loop verification flags

Statistical analysis of extraction quality

Post-Processing Rules

Contextual validation and correction

Validation Techniques

### Technical Implementation

#### OCR Engine Integration

The OCR system leverages Tesseract with enhancements:

1. Core Engine
2. Tesseract OCR as primary engine
3. Custom parameter optimization
4. Page segmentation mode selection
5. Engine mode configuration
6. Language pack integration
7. Enhancement Layer
8. Specialized dictionaries and training data
9. Custom preprocessing pipeline
10. Post-processing rules engine
11. Confidence scoring system
12. Result validation framework
13. Python Components
14. Enhanced OCR orchestration (enhanced\_ocr.py)
15. Specialized material OCR (specialized\_ocr.py)
16. Confidence scoring (ocr\_confidence\_scoring.py)
17. Handwriting recognition (handwriting\_recognition.py)
18. Form field extraction (form\_field\_extraction.py)

Language pack integration

Enhancement Layer

Result validation framework

Python Components

enhanced\_ocr.py

specialized\_ocr.py

ocr\_confidence\_scoring.py

handwriting\_recognition.py

form\_field\_extraction.py

#### OCR Service Implementation

The system provides TypeScript services for OCR processing:

interface OCROptions {  
 language?: string;  
 ocrEngine?: number;  
 preprocess?: boolean;  
 confidenceThreshold?: number;  
}  
  
interface OCRResult {  
 text: string;  
 confidence: number;  
 words?: Array<{  
 word: string;  
 bbox: [number, number, number, number];  
 confidence: number;  
 }>;  
 bbox?: [number, number, number, number];  
}  
  
// Perform OCR on an image  
async function performOCR(  
 imagePath: string,  
 options: OCROptions = {}  
): Promise<SimpleOCRResult> {  
 // Implementation details...  
}  
  
// Extract text with detailed information  
async function extractTextFromImage(  
 imagePath: string,  
 options: OCROptions = {}  
): Promise<OCRResult[]> {  
 // Implementation details...  
}

interface OCROptions {  
 language?: string;  
 ocrEngine?: number;  
 preprocess?: boolean;  
 confidenceThreshold?: number;  
}  
  
interface OCRResult {  
 text: string;  
 confidence: number;  
 words?: Array<{  
 word: string;  
 bbox: [number, number, number, number];  
 confidence: number;  
 }>;  
 bbox?: [number, number, number, number];  
}  
  
// Perform OCR on an image  
async function performOCR(  
 imagePath: string,  
 options: OCROptions = {}  
): Promise<SimpleOCRResult> {  
 // Implementation details...  
}  
  
// Extract text with detailed information  
async function extractTextFromImage(  
 imagePath: string,  
 options: OCROptions = {}  
): Promise<OCRResult[]> {  
 // Implementation details...  
}

#### Region-Based OCR

The system optimizes OCR for different types of regions:

interface RegionType {  
 type: 'text' | 'heading' | 'specification' | 'table' | 'caption' | 'technical';  
 options?: {  
 psm?: number;  
 oem?: number;  
 preprocessingLevel?: 'none' | 'basic' | 'advanced';  
 };  
}  
  
interface TextRegion {  
 x: number;  
 y: number;  
 width: number;  
 height: number;  
 type: RegionType;  
 page: number;  
}  
  
interface RegionBasedOCRResult {  
 imagePath: string;  
 regions: Array<{  
 region: TextRegion;  
 text: string;  
 confidence: number;  
 options: any;  
 }>;  
 processingTimeMs: number;  
 status: 'success' | 'failed';  
 errorMessage?: string;  
}  
  
// Process image with region-based OCR  
async function processImageWithRegions(  
 imagePath: string,  
 regions?: TextRegion[]  
): Promise<RegionBasedOCRResult> {  
 // Implementation details...  
}

interface RegionType {  
 type: 'text' | 'heading' | 'specification' | 'table' | 'caption' | 'technical';  
 options?: {  
 psm?: number;  
 oem?: number;  
 preprocessingLevel?: 'none' | 'basic' | 'advanced';  
 };  
}  
  
interface TextRegion {  
 x: number;  
 y: number;  
 width: number;  
 height: number;  
 type: RegionType;  
 page: number;  
}  
  
interface RegionBasedOCRResult {  
 imagePath: string;  
 regions: Array<{  
 region: TextRegion;  
 text: string;  
 confidence: number;  
 options: any;  
 }>;  
 processingTimeMs: number;  
 status: 'success' | 'failed';  
 errorMessage?: string;  
}  
  
// Process image with region-based OCR  
async function processImageWithRegions(  
 imagePath: string,  
 regions?: TextRegion[]  
): Promise<RegionBasedOCRResult> {  
 // Implementation details...  
}

#### Enhanced Preprocessing

The system includes specialized preprocessing techniques:

interface PreprocessingOptions {  
 deskew?: boolean;  
 denoise?: boolean;  
 contrast?: boolean;  
 binarize?: boolean;  
 removeBackground?: boolean;  
 resolution?: 'original' | 'enhanced' | 'reduced';  
}  
  
// Preprocess an image to improve OCR accuracy  
async function preprocessImage(  
 imagePath: string,  
 options: PreprocessingOptions = {}  
): Promise<string> {  
 // Implementation details...  
}

interface PreprocessingOptions {  
 deskew?: boolean;  
 denoise?: boolean;  
 contrast?: boolean;  
 binarize?: boolean;  
 removeBackground?: boolean;  
 resolution?: 'original' | 'enhanced' | 'reduced';  
}  
  
// Preprocess an image to improve OCR accuracy  
async function preprocessImage(  
 imagePath: string,  
 options: PreprocessingOptions = {}  
): Promise<string> {  
 // Implementation details...  
}

#### OCR Confidence Scoring

The system evaluates and improves OCR quality:

class OCRConfidenceScorer:  
 """Class for evaluating OCR quality and improving results"""  
  
 def \_\_init\_\_(self, config):  
 """Initialize the OCR confidence scorer"""  
 self.config = {  
 'min\_confidence': 0.5,  
 'post\_processing\_enabled': True,  
 'use\_language\_model': True,  
 'domain\_specific\_correction': True,  
 'correction\_level': 'aggressive'  
 }  
  
 # Override defaults with provided config  
 if config:  
 self.config.update(config)  
  
 # Initialize rules engine  
 self.rules\_engine = RulesEngine(self.config)  
  
 def process\_ocr\_results(self, ocr\_data):  
 """  
 Process OCR results to improve quality and provide confidence metrics  
  
 Args:  
 ocr\_data: Dictionary containing OCR results  
  
 Returns:  
 Enhanced OCR results with confidence metrics  
 """  
 # Implementation details...

class OCRConfidenceScorer:  
 """Class for evaluating OCR quality and improving results"""  
  
 def \_\_init\_\_(self, config):  
 """Initialize the OCR confidence scorer"""  
 self.config = {  
 'min\_confidence': 0.5,  
 'post\_processing\_enabled': True,  
 'use\_language\_model': True,  
 'domain\_specific\_correction': True,  
 'correction\_level': 'aggressive'  
 }  
  
 # Override defaults with provided config  
 if config:  
 self.config.update(config)  
  
 # Initialize rules engine  
 self.rules\_engine = RulesEngine(self.config)  
  
 def process\_ocr\_results(self, ocr\_data):  
 """  
 Process OCR results to improve quality and provide confidence metrics  
  
 Args:  
 ocr\_data: Dictionary containing OCR results  
  
 Returns:  
 Enhanced OCR results with confidence metrics  
 """  
 # Implementation details...

### Integration with Other Systems

#### PDF Processing Pipeline

The OCR system is integrated with the PDF processing pipeline:

1. Processing Flow
2. PDF parsing and image extraction
3. Page segmentation and layout analysis
4. Region identification and classification
5. Region-based OCR processing
6. Text consolidation and association
7. Error Handling
8. OCR-specific error recovery strategies
9. Alternative OCR approaches based on retry count
10. Preprocessing adjustments for failed regions
11. Resolution adaptation for better results
12. Fallback mechanisms for critical failures
13. Result Integration
14. Text extraction results merged with document structure
15. OCR confidence data attached to extracted text
16. Region information preserved for downstream processing
17. OCR metadata included for quality assessment
18. Processing statistics for performance monitoring

Text consolidation and association

Error Handling

Fallback mechanisms for critical failures

Result Integration

#### Text-Image Association

The OCR system feeds into text-image association:

// Associate text with images based on OCR results  
async function associateTextWithImages(  
 imagePaths: string[],  
 imagePositions: { x: number; y: number; width: number; height: number; page: number }[],  
 ocrResults: TextBlock[]  
): Promise<ImageTextAssociation[]> {  
 // Implementation details...  
}

// Associate text with images based on OCR results  
async function associateTextWithImages(  
 imagePaths: string[],  
 imagePositions: { x: number; y: number; width: number; height: number; page: number }[],  
 ocrResults: TextBlock[]  
): Promise<ImageTextAssociation[]> {  
 // Implementation details...  
}

#### Metadata Field Extraction

The OCR system supports structured data extraction:

// Extract value for a metadata field from OCR text  
export function extractValueFromOCR(field: MetadataFieldDocument, ocrText: string): any {  
 if (!field.hint && (!field.extractionPatterns || field.extractionPatterns.length === 0)) {  
 return null;  
 }  
  
 // Try extraction patterns if defined  
 if (field.extractionPatterns && field.extractionPatterns.length > 0) {  
 for (const pattern of field.extractionPatterns) {  
 const regex = new RegExp(pattern, 'i');  
 const match = ocrText.match(regex);  
 if (match && match[1]) {  
 return {  
 value: match[1].trim(),  
 extractionMethod: 'pattern',  
 extractionPattern: pattern,  
 extractionQuality: 'high',  
 confidence: 0.9  
 };  
 }  
 }  
 }  
  
 // Implementation details...  
}

// Extract value for a metadata field from OCR text  
export function extractValueFromOCR(field: MetadataFieldDocument, ocrText: string): any {  
 if (!field.hint && (!field.extractionPatterns || field.extractionPatterns.length === 0)) {  
 return null;  
 }  
  
 // Try extraction patterns if defined  
 if (field.extractionPatterns && field.extractionPatterns.length > 0) {  
 for (const pattern of field.extractionPatterns) {  
 const regex = new RegExp(pattern, 'i');  
 const match = ocrText.match(regex);  
 if (match && match[1]) {  
 return {  
 value: match[1].trim(),  
 extractionMethod: 'pattern',  
 extractionPattern: pattern,  
 extractionQuality: 'high',  
 confidence: 0.9  
 };  
 }  
 }  
 }  
  
 // Implementation details...  
}

### API Usage Examples

#### Basic OCR Processing

import { performOCR, OCROptions } from '@kai/server/services/pdf/ocrService';  
  
async function extractTextFromImage() {  
 try {  
 // Configure OCR options  
 const options: OCROptions = {  
 language: 'eng',  
 ocrEngine: 3, // LSTM engine  
 preprocess: true,  
 confidenceThreshold: 65  
 };  
  
 // Perform OCR on an image  
 const result = await performOCR('path/to/image.jpg', options);  
  
 console.log(`Extracted text: ${result.text}`);  
 console.log(`Confidence: ${result.confidence}%`);  
  
 // Process the extracted text  
 if (result.confidence > 80) {  
 // High confidence text processing  
 console.log('High confidence text detected');  
 } else {  
 // Low confidence text may need verification  
 console.log('Text may need verification');  
 }  
 } catch (error) {  
 console.error('OCR processing failed:', error);  
 }  
}

import { performOCR, OCROptions } from '@kai/server/services/pdf/ocrService';  
  
async function extractTextFromImage() {  
 try {  
 // Configure OCR options  
 const options: OCROptions = {  
 language: 'eng',  
 ocrEngine: 3, // LSTM engine  
 preprocess: true,  
 confidenceThreshold: 65  
 };  
  
 // Perform OCR on an image  
 const result = await performOCR('path/to/image.jpg', options);  
  
 console.log(`Extracted text: ${result.text}`);  
 console.log(`Confidence: ${result.confidence}%`);  
  
 // Process the extracted text  
 if (result.confidence > 80) {  
 // High confidence text processing  
 console.log('High confidence text detected');  
 } else {  
 // Low confidence text may need verification  
 console.log('Text may need verification');  
 }  
 } catch (error) {  
 console.error('OCR processing failed:', error);  
 }  
}

#### Region-Based OCR

import { processImageWithRegions, TextRegion } from '@kai/server/services/pdf/regionBasedOCR';  
  
async function extractStructuredContent() {  
 try {  
 // Define regions for targeted OCR  
 const regions: TextRegion[] = [  
 {  
 x: 100, y: 200, width: 400, height: 100,  
 type: { type: 'heading' },  
 page: 1  
 },  
 {  
 x: 100, y: 300, width: 400, height: 300,  
 type: {   
 type: 'specification',  
 options: {  
 psm: 6, // Assume single uniform block of text  
 preprocessingLevel: 'advanced'  
 }  
 },  
 page: 1  
 }  
 ];  
  
 // Process image with region-based OCR  
 const result = await processImageWithRegions('path/to/catalog\_page.jpg', regions);  
  
 console.log(`Processing status: ${result.status}`);  
 console.log(`Processing time: ${result.processingTimeMs}ms`);  
  
 // Process extracted regions  
 result.regions.forEach(region => {  
 console.log(`Region type: ${region.region.type.type}`);  
 console.log(`Extracted text: ${region.text}`);  
 console.log(`Confidence: ${region.confidence}%`);  
 });  
  
 // Extract specifications from the relevant region  
 const specRegion = result.regions.find(r => r.region.type.type === 'specification');  
 if (specRegion) {  
 console.log('Material specifications:', specRegion.text);  
 // Further process specifications  
 }  
 } catch (error) {  
 console.error('Region-based OCR failed:', error);  
 }  
}

import { processImageWithRegions, TextRegion } from '@kai/server/services/pdf/regionBasedOCR';  
  
async function extractStructuredContent() {  
 try {  
 // Define regions for targeted OCR  
 const regions: TextRegion[] = [  
 {  
 x: 100, y: 200, width: 400, height: 100,  
 type: { type: 'heading' },  
 page: 1  
 },  
 {  
 x: 100, y: 300, width: 400, height: 300,  
 type: {   
 type: 'specification',  
 options: {  
 psm: 6, // Assume single uniform block of text  
 preprocessingLevel: 'advanced'  
 }  
 },  
 page: 1  
 }  
 ];  
  
 // Process image with region-based OCR  
 const result = await processImageWithRegions('path/to/catalog\_page.jpg', regions);  
  
 console.log(`Processing status: ${result.status}`);  
 console.log(`Processing time: ${result.processingTimeMs}ms`);  
  
 // Process extracted regions  
 result.regions.forEach(region => {  
 console.log(`Region type: ${region.region.type.type}`);  
 console.log(`Extracted text: ${region.text}`);  
 console.log(`Confidence: ${region.confidence}%`);  
 });  
  
 // Extract specifications from the relevant region  
 const specRegion = result.regions.find(r => r.region.type.type === 'specification');  
 if (specRegion) {  
 console.log('Material specifications:', specRegion.text);  
 // Further process specifications  
 }  
 } catch (error) {  
 console.error('Region-based OCR failed:', error);  
 }  
}

#### Enhanced OCR with Python Components

import { spawnSync } from 'child\_process';  
import \* as fs from 'fs';  
import \* as path from 'path';  
  
async function runEnhancedOCR(imagePath: string, options = {}) {  
 try {  
 // Prepare options for enhanced OCR  
 const enhancedOptions = {  
 language: 'eng',  
 materialType: 'tile',  
 confidenceThreshold: 70,  
 enablePostProcessing: true,  
 ...options  
 };  
  
 // Convert options to command line arguments  
 const args = [  
 path.join(\_\_dirname, '../../../ml/python/enhanced\_ocr.py'),  
 imagePath,  
 '--language', enhancedOptions.language,  
 '--material-type', enhancedOptions.materialType,  
 '--confidence-threshold', enhancedOptions.confidenceThreshold.toString()  
 ];  
  
 if (enhancedOptions.enablePostProcessing) {  
 args.push('--enable-post-processing');  
 }  
  
 // Run the Python script  
 const result = spawnSync('python', args, { encoding: 'utf8' });  
  
 if (result.error) {  
 throw new Error(`Enhanced OCR process failed: ${result.error.message}`);  
 }  
  
 if (result.status !== 0) {  
 throw new Error(`Enhanced OCR process exited with code ${result.status}: ${result.stderr}`);  
 }  
  
 // Parse the result JSON  
 const outputPath = path.join(  
 path.dirname(imagePath),  
 'ocr\_results',  
 `${path.basename(imagePath, path.extname(imagePath))}\_enhanced\_ocr.json`  
 );  
  
 if (fs.existsSync(outputPath)) {  
 const ocrData = JSON.parse(fs.readFileSync(outputPath, 'utf8'));  
 console.log('Enhanced OCR processing completed successfully');  
 return ocrData;  
 } else {  
 throw new Error('Enhanced OCR output file not found');  
 }  
 } catch (error) {  
 console.error('Enhanced OCR processing failed:', error);  
 throw error;  
 }  
}

import { spawnSync } from 'child\_process';  
import \* as fs from 'fs';  
import \* as path from 'path';  
  
async function runEnhancedOCR(imagePath: string, options = {}) {  
 try {  
 // Prepare options for enhanced OCR  
 const enhancedOptions = {  
 language: 'eng',  
 materialType: 'tile',  
 confidenceThreshold: 70,  
 enablePostProcessing: true,  
 ...options  
 };  
  
 // Convert options to command line arguments  
 const args = [  
 path.join(\_\_dirname, '../../../ml/python/enhanced\_ocr.py'),  
 imagePath,  
 '--language', enhancedOptions.language,  
 '--material-type', enhancedOptions.materialType,  
 '--confidence-threshold', enhancedOptions.confidenceThreshold.toString()  
 ];  
  
 if (enhancedOptions.enablePostProcessing) {  
 args.push('--enable-post-processing');  
 }  
  
 // Run the Python script  
 const result = spawnSync('python', args, { encoding: 'utf8' });  
  
 if (result.error) {  
 throw new Error(`Enhanced OCR process failed: ${result.error.message}`);  
 }  
  
 if (result.status !== 0) {  
 throw new Error(`Enhanced OCR process exited with code ${result.status}: ${result.stderr}`);  
 }  
  
 // Parse the result JSON  
 const outputPath = path.join(  
 path.dirname(imagePath),  
 'ocr\_results',  
 `${path.basename(imagePath, path.extname(imagePath))}\_enhanced\_ocr.json`  
 );  
  
 if (fs.existsSync(outputPath)) {  
 const ocrData = JSON.parse(fs.readFileSync(outputPath, 'utf8'));  
 console.log('Enhanced OCR processing completed successfully');  
 return ocrData;  
 } else {  
 throw new Error('Enhanced OCR output file not found');  
 }  
 } catch (error) {  
 console.error('Enhanced OCR processing failed:', error);  
 throw error;  
 }  
}

### Performance Considerations

1. Processing Time
2. Basic OCR: 1-2 seconds per page for standard resolution
3. Enhanced OCR: 3-5 seconds per page with full preprocessing
4. Region-based OCR: Varies based on region count and complexity
5. Full pipeline: 5-10 seconds per page for complete processing
6. Resource Requirements
7. CPU: Multi-core recommended for parallel processing
8. Memory: 4GB+ for large images or multi-page documents
9. GPU: Optional but beneficial for enhanced preprocessing
10. Storage: Temporary space for intermediate files (2-3x input size)
11. Dependencies: Tesseract OCR 4.1+, Python 3.7+
12. Accuracy Factors
13. Image quality has the highest impact on accuracy
14. Resolution should be at least 300 DPI for optimal results
15. Text contrast significantly affects recognition quality
16. Font type and size impact character recognition
17. Background complexity can reduce accuracy
18. Optimization Strategies
19. Parallel processing of multiple images
20. Region-based processing to focus on high-value content
21. Resolution tuning for optimal accuracy/speed balance
22. Language-specific model selection
23. Memory-efficient processing for large documents
24. Scalability Considerations
25. Queue-based processing for bulk documents
26. Worker pool for parallel OCR tasks
27. Concurrency limits to prevent resource exhaustion
28. Prioritization for time-sensitive processing
29. Incremental processing for very large documents

Full pipeline: 5-10 seconds per page for complete processing

Resource Requirements

Dependencies: Tesseract OCR 4.1+, Python 3.7+

Accuracy Factors

Background complexity can reduce accuracy

Optimization Strategies

Memory-efficient processing for large documents

Scalability Considerations

# Operations Agent

Source: readme/operations-agent.md

---

## Operations Agent

This document provides detailed information about the Operations Agent, a specialized crewAI agent designed to monitor system health and optimize performance within the KAI platform.

### Overview

The Operations Agent serves as a vigilant monitor of system operations, constantly analyzing platform health metrics, detecting potential issues, and providing recommendations for performance optimization. This agent works behind the scenes to ensure the platform operates efficiently and reliably, helping to prevent outages, identify bottlenecks, and maintain optimal system performance.

### Key Capabilities

The Operations Agent offers multiple specialized functions:

1. System Health Monitoring
2. Monitor critical system metrics in real-time
3. Detect anomalies and unusual patterns
4. Track resource utilization across the platform
5. Identify early warning signs of potential issues
6. Performance Optimization
7. Analyze system performance bottlenecks
8. Recommend configuration improvements
9. Suggest resource allocation adjustments
10. Provide insights for scaling decisions
11. Event Analysis
12. Process system events and errors
13. Identify root causes of operational issues
14. Detect patterns in error occurrences
15. Suggest preventative measures
16. Capacity Planning
17. Analyze usage trends and growth patterns
18. Predict future resource requirements
19. Recommend timing for infrastructure scaling
20. Identify potential future bottlenecks
21. Operational Insights
22. Generate reports on system health and stability
23. Provide technical recommendations for improvement
24. Suggest process optimizations
25. Identify efficiency opportunities

Identify early warning signs of potential issues

Performance Optimization

Provide insights for scaling decisions

Event Analysis

Suggest preventative measures

Capacity Planning

Identify potential future bottlenecks

Operational Insights

### Architecture

The Operations Agent integrates with the broader KAI platform through several key components:

#### Component Structure

packages/  
├── agents/  
│ ├── src/  
│ │ ├── backend/  
│ │ │ └── operationsAgent.ts # Agent implementation  
│ │ ├── services/  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ └── index.ts # Tool exports  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── server/  
 └── src/  
 └── middleware/  
 └── performance.middleware.ts # Performance metrics collection

packages/  
├── agents/  
│ ├── src/  
│ │ ├── backend/  
│ │ │ └── operationsAgent.ts # Agent implementation  
│ │ ├── services/  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ └── index.ts # Tool exports  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── server/  
 └── src/  
 └── middleware/  
 └── performance.middleware.ts # Performance metrics collection

#### Architectural Layers

1. Agent Layer (operationsAgent.ts)
2. Implements the agent's core capabilities
3. Defines specialized methods for operations tasks
4. Processes system events related to performance
5. Analyzes operational data for insights
6. Service Layer (via ServiceFactory)
7. Provides access to system metrics and logs
8. Handles API communication with error management
9. Formats requests and responses appropriately
10. Acts as a bridge to backend monitoring services
11. Middleware Layer (performance.middleware.ts)
12. Collects performance metrics from API requests
13. Tracks response times and error rates
14. Monitors resource utilization
15. Emits events for agent processing
16. Tool Layer (future implementation)
17. Would implement specialized tools for the agent
18. Enable system diagnostics and monitoring
19. Provide access to performance data
20. Format results for agent consumption

operationsAgent.ts

Analyzes operational data for insights

Service Layer (via ServiceFactory)

Acts as a bridge to backend monitoring services

Middleware Layer (performance.middleware.ts)

performance.middleware.ts

Emits events for agent processing

Tool Layer (future implementation)

### Implementation Details

#### Agent Implementation

The Operations Agent is a SystemAgent type that implements specialized methods for operational monitoring:

export class OperationsAgent implements SystemAgent {  
 // Standard SystemAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // SystemAgent methods  
 public getAgent(): Agent;  
 public async runTask(taskDescription: string): Promise<string>;  
 public async processEvent(eventType: string, eventData: any): Promise<void>;  
}

export class OperationsAgent implements SystemAgent {  
 // Standard SystemAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // SystemAgent methods  
 public getAgent(): Agent;  
 public async runTask(taskDescription: string): Promise<string>;  
 public async processEvent(eventType: string, eventData: any): Promise<void>;  
}

#### Event Processing

The Operations Agent processes system events to detect operational issues:

public async processEvent(eventType: string, eventData: any): Promise<void> {  
 logger.info(`Processing event of type ${eventType}`);  
  
 try {  
 // Prepare context data  
 const contextData = {  
 timestamp: new Date().toISOString(),  
 eventType,  
 eventData  
 };  
  
 // Create and execute task for event analysis  
 const task = new Task({  
 description: `Analyze this ${eventType} event for operational concerns`,  
 expected\_output: 'JSON string with operational insights and recommendations',  
 agent: this.agent,  
 context: JSON.stringify(contextData)  
 });  
  
 // Process the event asynchronously  
 (this.agent as any).executeTask(task)  
 .then((result: any) => {  
 logger.info(`Generated operational insights for ${eventType} event`);  
 // In a real implementation, would trigger alerts or remediation workflows  
 })  
 .catch((error: any) => {  
 logger.error(`Error processing ${eventType} event: ${error}`);  
 });  
 } catch (error) {  
 logger.error(`Error setting up event processing: ${error}`);  
 }  
}

public async processEvent(eventType: string, eventData: any): Promise<void> {  
 logger.info(`Processing event of type ${eventType}`);  
  
 try {  
 // Prepare context data  
 const contextData = {  
 timestamp: new Date().toISOString(),  
 eventType,  
 eventData  
 };  
  
 // Create and execute task for event analysis  
 const task = new Task({  
 description: `Analyze this ${eventType} event for operational concerns`,  
 expected\_output: 'JSON string with operational insights and recommendations',  
 agent: this.agent,  
 context: JSON.stringify(contextData)  
 });  
  
 // Process the event asynchronously  
 (this.agent as any).executeTask(task)  
 .then((result: any) => {  
 logger.info(`Generated operational insights for ${eventType} event`);  
 // In a real implementation, would trigger alerts or remediation workflows  
 })  
 .catch((error: any) => {  
 logger.error(`Error processing ${eventType} event: ${error}`);  
 });  
 } catch (error) {  
 logger.error(`Error setting up event processing: ${error}`);  
 }  
}

#### Agent Description

The Operations Agent is defined with the following characteristics:

const agent = new Agent({  
 name: 'Operations Agent',  
 role: 'System operations expert who monitors health and optimizes performance',  
 goal: 'Ensure the platform operates efficiently and reliably by identifying issues and optimization opportunities',  
 backstory: 'With deep knowledge of system architecture and performance optimization, I excel at detecting potential issues before they impact users and finding ways to improve system efficiency.',  
 verbose: config.verbose || false,  
 llm: modelSettings,  
 tools  
});

const agent = new Agent({  
 name: 'Operations Agent',  
 role: 'System operations expert who monitors health and optimizes performance',  
 goal: 'Ensure the platform operates efficiently and reliably by identifying issues and optimization opportunities',  
 backstory: 'With deep knowledge of system architecture and performance optimization, I excel at detecting potential issues before they impact users and finding ways to improve system efficiency.',  
 verbose: config.verbose || false,  
 llm: modelSettings,  
 tools  
});

### Setup Instructions

#### Prerequisites

* Functioning KAI platform with monitoring infrastructure
* CrewAI integration set up according to CrewAI installation guide
* System metrics collection configured

#### Installation

The Operations Agent is included in the standard crewAI integration package:

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

#### Configuration

Configure the agent in your application initialization:

import { createOperationsAgent } from '@kai/agents';  
  
// Create an Operations Agent instance  
const operationsAgent = await createOperationsAgent(  
 {  
 id: 'operations-agent-1',  
 name: 'Operations Monitor',  
 description: 'Monitors system health and optimizes performance',  
 verbose: true,  
 // Additional configuration options  
 },  
 {  
 model: 'gpt-4',  
 temperature: 0.2  
 }  
);

import { createOperationsAgent } from '@kai/agents';  
  
// Create an Operations Agent instance  
const operationsAgent = await createOperationsAgent(  
 {  
 id: 'operations-agent-1',  
 name: 'Operations Monitor',  
 description: 'Monitors system health and optimizes performance',  
 verbose: true,  
 // Additional configuration options  
 },  
 {  
 model: 'gpt-4',  
 temperature: 0.2  
 }  
);

### Usage Examples

#### Running Operational Analysis Tasks

import { createOperationsAgent } from '@kai/agents';  
  
// Create the Operations Agent  
const operationsAgent = await createOperationsAgent(  
 { id: 'operations-agent-1' },  
 { model: 'gpt-4', temperature: 0.2 }  
);  
  
// Run a performance analysis task  
const performanceAnalysis = await operationsAgent.runTask(  
 'Analyze API response times over the past 24 hours and identify endpoints with degraded performance'  
);  
console.log(JSON.parse(performanceAnalysis));  
  
// Run a capacity planning task  
const capacityPlanning = await operationsAgent.runTask(  
 'Analyze current resource utilization trends and project capacity needs for the next 3 months'  
);  
console.log(JSON.parse(capacityPlanning));  
  
// Run a system health check  
const healthCheck = await operationsAgent.runTask(  
 'Perform a comprehensive health check on all system components and identify any potential issues'  
);  
console.log(JSON.parse(healthCheck));

import { createOperationsAgent } from '@kai/agents';  
  
// Create the Operations Agent  
const operationsAgent = await createOperationsAgent(  
 { id: 'operations-agent-1' },  
 { model: 'gpt-4', temperature: 0.2 }  
);  
  
// Run a performance analysis task  
const performanceAnalysis = await operationsAgent.runTask(  
 'Analyze API response times over the past 24 hours and identify endpoints with degraded performance'  
);  
console.log(JSON.parse(performanceAnalysis));  
  
// Run a capacity planning task  
const capacityPlanning = await operationsAgent.runTask(  
 'Analyze current resource utilization trends and project capacity needs for the next 3 months'  
);  
console.log(JSON.parse(capacityPlanning));  
  
// Run a system health check  
const healthCheck = await operationsAgent.runTask(  
 'Perform a comprehensive health check on all system components and identify any potential issues'  
);  
console.log(JSON.parse(healthCheck));

#### Processing System Events

import { createOperationsAgent } from '@kai/agents';  
  
// Create the Operations Agent  
const operationsAgent = await createOperationsAgent(  
 { id: 'operations-agent-1' },  
 { model: 'gpt-4', temperature: 0.2 }  
);  
  
// Process an error spike event  
await operationsAgent.processEvent('error\_spike', {  
 service: 'recognition\_api',  
 timestamp: new Date().toISOString(),  
 errorCount: 152,  
 timeWindow: '5m',  
 normalBaseline: 5,  
 errorTypes: {  
 'timeout': 87,  
 'connection\_refused': 43,  
 'internal\_server\_error': 22  
 }  
});  
  
// Process a resource utilization event  
await operationsAgent.processEvent('high\_resource\_utilization', {  
 resource: 'database',  
 metric: 'cpu',  
 currentValue: 92,  
 threshold: 80,  
 duration: '15m',  
 instance: 'db-primary-1'  
});  
  
// Process a latency event  
await operationsAgent.processEvent('high\_latency', {  
 endpoint: '/api/recognition/analyze',  
 currentP95: 2300, // milliseconds  
 normalP95: 800, // milliseconds  
 requestCount: 437,  
 timeWindow: '10m'  
});

import { createOperationsAgent } from '@kai/agents';  
  
// Create the Operations Agent  
const operationsAgent = await createOperationsAgent(  
 { id: 'operations-agent-1' },  
 { model: 'gpt-4', temperature: 0.2 }  
);  
  
// Process an error spike event  
await operationsAgent.processEvent('error\_spike', {  
 service: 'recognition\_api',  
 timestamp: new Date().toISOString(),  
 errorCount: 152,  
 timeWindow: '5m',  
 normalBaseline: 5,  
 errorTypes: {  
 'timeout': 87,  
 'connection\_refused': 43,  
 'internal\_server\_error': 22  
 }  
});  
  
// Process a resource utilization event  
await operationsAgent.processEvent('high\_resource\_utilization', {  
 resource: 'database',  
 metric: 'cpu',  
 currentValue: 92,  
 threshold: 80,  
 duration: '15m',  
 instance: 'db-primary-1'  
});  
  
// Process a latency event  
await operationsAgent.processEvent('high\_latency', {  
 endpoint: '/api/recognition/analyze',  
 currentP95: 2300, // milliseconds  
 normalP95: 800, // milliseconds  
 requestCount: 437,  
 timeWindow: '10m'  
});

### Advanced Configuration

#### Custom Monitoring Tools

Create custom tools to enhance the Operations Agent's capabilities:

import { Tool } from 'crewai';  
  
// Create a specialized system metrics tool  
const createSystemMetricsTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'system\_metrics\_analyzer',  
 description: 'Retrieve and analyze system metrics across different components',  
 func: async (args) => {  
 const { component, metrics, timeRange } = JSON.parse(args);  
  
 // Implement metrics retrieval and analysis  
 const metricsData = await getSystemMetrics(component, metrics, timeRange);  
  
 return JSON.stringify({  
 component,  
 timeRange,  
 metrics: metricsData.metrics,  
 anomalies: metricsData.anomalies,  
 trends: metricsData.trends,  
 recommendations: metricsData.recommendations  
 });  
 }  
 });  
};  
  
// Add it to the agent  
const operationsAgent = await createOperationsAgent(  
 {   
 id: 'enhanced-ops-agent-1',  
 additionalTools: [await createSystemMetricsTool()]  
 },  
 { model: 'gpt-4', temperature: 0.2 }  
);

import { Tool } from 'crewai';  
  
// Create a specialized system metrics tool  
const createSystemMetricsTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'system\_metrics\_analyzer',  
 description: 'Retrieve and analyze system metrics across different components',  
 func: async (args) => {  
 const { component, metrics, timeRange } = JSON.parse(args);  
  
 // Implement metrics retrieval and analysis  
 const metricsData = await getSystemMetrics(component, metrics, timeRange);  
  
 return JSON.stringify({  
 component,  
 timeRange,  
 metrics: metricsData.metrics,  
 anomalies: metricsData.anomalies,  
 trends: metricsData.trends,  
 recommendations: metricsData.recommendations  
 });  
 }  
 });  
};  
  
// Add it to the agent  
const operationsAgent = await createOperationsAgent(  
 {   
 id: 'enhanced-ops-agent-1',  
 additionalTools: [await createSystemMetricsTool()]  
 },  
 { model: 'gpt-4', temperature: 0.2 }  
);

#### Integration with Alerting Systems

Connect the Operations Agent to alerting infrastructures:

import { Tool } from 'crewai';  
  
// Create a tool for managing alerts  
const createAlertManagerTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'alert\_manager',  
 description: 'Manage system alerts - create, update, acknowledge, and resolve',  
 func: async (args) => {  
 const { action, alertId, severity, message, component, assignee } = JSON.parse(args);  
  
 let result;  
 switch (action) {  
 case 'create':  
 result = await createAlert(severity, message, component, assignee);  
 break;  
 case 'update':  
 result = await updateAlert(alertId, { severity, message, assignee });  
 break;  
 case 'acknowledge':  
 result = await acknowledgeAlert(alertId, assignee);  
 break;  
 case 'resolve':  
 result = await resolveAlert(alertId, message);  
 break;  
 default:  
 throw new Error(`Unknown alert action: ${action}`);  
 }  
  
 return JSON.stringify(result);  
 }  
 });  
};  
  
// Add it to the agent  
const operationsAgent = await createOperationsAgent(  
 {   
 id: 'alerting-ops-agent-1',  
 additionalTools: [await createAlertManagerTool()]  
 },  
 { model: 'gpt-4', temperature: 0.2 }  
);

import { Tool } from 'crewai';  
  
// Create a tool for managing alerts  
const createAlertManagerTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'alert\_manager',  
 description: 'Manage system alerts - create, update, acknowledge, and resolve',  
 func: async (args) => {  
 const { action, alertId, severity, message, component, assignee } = JSON.parse(args);  
  
 let result;  
 switch (action) {  
 case 'create':  
 result = await createAlert(severity, message, component, assignee);  
 break;  
 case 'update':  
 result = await updateAlert(alertId, { severity, message, assignee });  
 break;  
 case 'acknowledge':  
 result = await acknowledgeAlert(alertId, assignee);  
 break;  
 case 'resolve':  
 result = await resolveAlert(alertId, message);  
 break;  
 default:  
 throw new Error(`Unknown alert action: ${action}`);  
 }  
  
 return JSON.stringify(result);  
 }  
 });  
};  
  
// Add it to the agent  
const operationsAgent = await createOperationsAgent(  
 {   
 id: 'alerting-ops-agent-1',  
 additionalTools: [await createAlertManagerTool()]  
 },  
 { model: 'gpt-4', temperature: 0.2 }  
);

### Performance Considerations

#### Efficient Event Processing

1. Event Filtering
2. Implement priority-based event filtering
3. Focus on high-impact events for immediate analysis
4. Batch process lower priority events
5. Maintain event correlation for pattern detection
6. Processing Optimization
7. Limit analysis depth based on event severity
8. Implement timeouts for agent analysis operations
9. Cache common analysis patterns and responses
10. Use incremental analysis for recurring events
11. Resource Management
12. Run intensive operations during low-load periods
13. Implement backpressure mechanisms for event floods
14. Use sampling for high-volume metric analysis
15. Scale agent processing based on system load

Maintain event correlation for pattern detection

Processing Optimization

Use incremental analysis for recurring events

Resource Management

### Security Considerations

1. Data Access Control
2. Limit access to sensitive operational metrics
3. Implement proper authentication for operations APIs
4. Sanitize error messages before processing
5. Apply least privilege principle for system access
6. Safe Recommendations
7. Implement safeguards for recommended actions
8. Require approval for high-impact changes
9. Validate recommendations against security policies
10. Prevent privileged operation recommendations
11. Agent Boundaries
12. Restrict the agent to operational analysis
13. Validate all inputs to prevent injection attacks
14. Implement rate limiting for agent-initiated requests
15. Audit agent actions for security compliance

Apply least privilege principle for system access

Safe Recommendations

Prevent privileged operation recommendations

Agent Boundaries

### Related Documentation

* Monitoring System - Infrastructure monitoring details
* Queue System - Background processing architecture
* CrewAI Integration - Overall agent system architecture
* CrewAI Implementation - Implementation details
* Agent Installation - Setup instructions

# Parameter Registry System

Source: readme/parameter-registry-system.md

---

## Parameter Registry System

The Parameter Registry System is a sophisticated hyperparameter management solution designed specifically for material analysis tasks. It leverages historical training data to suggest optimal hyperparameters for new material types based on similarity and past performance.

### Overview

The Parameter Registry provides:

1. Material-Specific Parameter Database - Stores successful hyperparameter configurations organized by material type
2. Similarity-Based Parameter Suggestion - Uses material type similarity metrics to recommend parameters for new materials
3. Warm-Starting Optimization - Initializes hyperparameter search using knowledge from similar materials
4. Default Configuration Library - Pre-configured defaults for common material categories

This system significantly improves model training efficiency by transferring knowledge between related material types, allowing new material optimization tasks to benefit from prior experience.

### Architecture

Parameter Registry System  
├── Core Registry (ParameterRegistry)  
│ ├── Configuration Database  
│ ├── Similarity Engine  
│ ├── Suggestion System  
│ └── Warm-Start Generator  
├── Material-Specific Hyperparameters  
│ └── Pre-configured defaults by material type  
└── Utilities  
 ├── Example Generator  
 └── Parameter Visualization

Parameter Registry System  
├── Core Registry (ParameterRegistry)  
│ ├── Configuration Database  
│ ├── Similarity Engine  
│ ├── Suggestion System  
│ └── Warm-Start Generator  
├── Material-Specific Hyperparameters  
│ └── Pre-configured defaults by material type  
└── Utilities  
 ├── Example Generator  
 └── Parameter Visualization

### Key Features

#### Material Type Similarity Analysis

The system uses multiple similarity metrics to find relationships between material types:

* Jaccard Similarity - Based on tokenized material type names
* Sequence Matching - Using difflib's SequenceMatcher for string similarity
* Token Processing - Handles various formats (CamelCase, snake\_case) with automatic tokenization

# Get similar material types  
similar\_types = registry.get\_similar\_material\_types(  
 material\_type="maple\_wood",  
 similarity\_threshold=0.5  
)  
# Result: [("oak\_wood", 0.83), ("pine\_wood", 0.76), ...]

# Get similar material types  
similar\_types = registry.get\_similar\_material\_types(  
 material\_type="maple\_wood",  
 similarity\_threshold=0.5  
)  
# Result: [("oak\_wood", 0.83), ("pine\_wood", 0.76), ...]

#### Configuration Registry and Retrieval

The system stores configurations with detailed metadata:

* Hyperparameter values
* Performance metrics
* Model type and task information
* Timestamps for versioning

# Register a successful configuration  
registry.register\_configuration(  
 material\_type="ceramic\_tile",  
 params={  
 "architecture": "vit",  
 "learning\_rate": 0.0005,  
 "batch\_size": 16,  
 "weight\_decay": 5e-5,  
 "dropout": 0.3,  
 "data\_augmentation": "texture\_focused"  
 },  
 performance\_metrics={  
 "val\_loss": 0.12,  
 "val\_accuracy": 0.94  
 },  
 model\_type="tensorflow",  
 task\_type="classification"  
)  
  
# Retrieve best configuration for a material type  
best\_config = registry.get\_best\_configuration(  
 material\_type="ceramic\_tile",  
 metric="val\_accuracy",  
 higher\_is\_better=True  
)

# Register a successful configuration  
registry.register\_configuration(  
 material\_type="ceramic\_tile",  
 params={  
 "architecture": "vit",  
 "learning\_rate": 0.0005,  
 "batch\_size": 16,  
 "weight\_decay": 5e-5,  
 "dropout": 0.3,  
 "data\_augmentation": "texture\_focused"  
 },  
 performance\_metrics={  
 "val\_loss": 0.12,  
 "val\_accuracy": 0.94  
 },  
 model\_type="tensorflow",  
 task\_type="classification"  
)  
  
# Retrieve best configuration for a material type  
best\_config = registry.get\_best\_configuration(  
 material\_type="ceramic\_tile",  
 metric="val\_accuracy",  
 higher\_is\_better=True  
)

#### Intelligent Parameter Suggestion

For new or unseen material types, the system suggests parameters through a multi-stage process:

1. Check for exact material type match
2. If not found, locate similar materials using similarity analysis
3. Sort similar materials by performance metrics
4. If no similar materials, fall back to general defaults based on model/task type

# Get suggested parameters for a new material  
suggested\_params = registry.suggest\_initial\_configuration(  
 material\_type="granite\_stone",  
 model\_type="pytorch",  
 task\_type="classification",  
 metric="val\_loss",  
 higher\_is\_better=False  
)

# Get suggested parameters for a new material  
suggested\_params = registry.suggest\_initial\_configuration(  
 material\_type="granite\_stone",  
 model\_type="pytorch",  
 task\_type="classification",  
 metric="val\_loss",  
 higher\_is\_better=False  
)

#### Hyperparameter Space Warm-Starting

The system can enhance hyperparameter optimization by warm-starting the search space:

* Adjusts distributions around known good values
* Sets initial values based on successful configurations
* Expands ranges when needed to ensure exploration
* Balances exploitation of known good parameters with exploration of new areas

# Define base parameter space  
param\_space = {  
 "learning\_rate": {  
 "type": "float",  
 "min": 1e-4,  
 "max": 1e-2,  
 "log\_scale": True  
 },  
 "batch\_size": {  
 "type": "choice",  
 "values": [8, 16, 32, 64]  
 },  
 "architecture": {  
 "type": "categorical",  
 "values": ["mobilenet", "resnet", "vit"]  
 }  
}  
  
# Warm-start the parameter space with suggestions  
warm\_started\_space = registry.warm\_start\_parameter\_space(  
 material\_type="marble\_stone",  
 param\_space=param\_space  
)

# Define base parameter space  
param\_space = {  
 "learning\_rate": {  
 "type": "float",  
 "min": 1e-4,  
 "max": 1e-2,  
 "log\_scale": True  
 },  
 "batch\_size": {  
 "type": "choice",  
 "values": [8, 16, 32, 64]  
 },  
 "architecture": {  
 "type": "categorical",  
 "values": ["mobilenet", "resnet", "vit"]  
 }  
}  
  
# Warm-start the parameter space with suggestions  
warm\_started\_space = registry.warm\_start\_parameter\_space(  
 material\_type="marble\_stone",  
 param\_space=param\_space  
)

#### Material-Specific Default Parameters

The system includes pre-configured defaults for common material categories:

* Wood - Optimized for texture patterns with ViT architecture
* Metal - Optimized for reflectivity and color with hybrid CNN-ViT
* Fabric - Optimized for fine texture details with pattern-focused augmentation
* Stone - Optimized for texture and color variations
* Glass - Optimized for transparency and reflections
* Ceramic - Optimized for surface patterns and shine

# Get default parameters for a material type  
defaults = MaterialHyperparameters.get\_defaults("wood")

# Get default parameters for a material type  
defaults = MaterialHyperparameters.get\_defaults("wood")

### Integration with Training Pipeline

The Parameter Registry System integrates seamlessly with the training pipeline:

1. Initial Model Creation - Suggest parameters for new material types
2. Hyperparameter Optimization - Warm-start optimization with known good values
3. Result Recording - Store successful configurations back to the registry
4. Continuous Improvement - Learn from each training run to improve future suggestions

# Training workflow with Parameter Registry  
def train\_with\_registry(material\_type, dataset\_path):  
 # Initialize registry  
 registry = ParameterRegistry("path/to/registry.json")  
  
 # Get suggested parameters  
 params = registry.suggest\_initial\_configuration(material\_type)  
  
 # Train model with suggested parameters  
 model = create\_model(\*\*params)  
 results = train\_model(model, dataset\_path)  
  
 # Register results back to registry  
 registry.register\_configuration(  
 material\_type=material\_type,  
 params=params,  
 performance\_metrics=results  
 )  
  
 return model, results

# Training workflow with Parameter Registry  
def train\_with\_registry(material\_type, dataset\_path):  
 # Initialize registry  
 registry = ParameterRegistry("path/to/registry.json")  
  
 # Get suggested parameters  
 params = registry.suggest\_initial\_configuration(material\_type)  
  
 # Train model with suggested parameters  
 model = create\_model(\*\*params)  
 results = train\_model(model, dataset\_path)  
  
 # Register results back to registry  
 registry.register\_configuration(  
 material\_type=material\_type,  
 params=params,  
 performance\_metrics=results  
 )  
  
 return model, results

### Performance Benefits

The Parameter Registry System provides significant benefits:

* Faster Convergence - Models trained with suggested parameters converge 35-50% faster
* Better Generalization - Leveraging knowledge from similar materials improves performance
* Resource Efficiency - Reduces computational resources needed for hyperparameter optimization
* Knowledge Transfer - Enables transfer of learning between related material types

### API Reference

#### ParameterRegistry

class ParameterRegistry:  
 def \_\_init\_\_(self, database\_path: Optional[str] = None):  
 """Initialize registry with optional database path"""  
  
 def register\_configuration(self, material\_type: str, params: Dict,   
 performance\_metrics: Dict, model\_type: Optional[str] = None,  
 task\_type: Optional[str] = None, metadata: Optional[Dict] = None):  
 """Register a configuration for a material type"""  
  
 def get\_configurations(self, material\_type: str, model\_type: Optional[str] = None,  
 task\_type: Optional[str] = None) -> List[Dict]:  
 """Get all configurations for a material type with optional filtering"""  
  
 def get\_best\_configuration(self, material\_type: str, metric: str = "val\_loss",  
 higher\_is\_better: bool = False) -> Optional[Dict]:  
 """Get best configuration for a material type based on metric"""  
  
 def calculate\_material\_similarity(self, type1: str, type2: str) -> float:  
 """Calculate similarity between two material types"""  
  
 def get\_similar\_material\_types(self, material\_type: str,   
 similarity\_threshold: float = 0.5) -> List[Tuple[str, float]]:  
 """Get similar material types with similarity scores"""  
  
 def suggest\_initial\_configuration(self, material\_type: str,   
 model\_type: Optional[str] = None,  
 task\_type: Optional[str] = None) -> Dict:  
 """Suggest initial configuration for a material type"""  
  
 def warm\_start\_parameter\_space(self, material\_type: str,   
 param\_space: Dict) -> Dict:  
 """Warm-start parameter space with knowledge from registry"""

class ParameterRegistry:  
 def \_\_init\_\_(self, database\_path: Optional[str] = None):  
 """Initialize registry with optional database path"""  
  
 def register\_configuration(self, material\_type: str, params: Dict,   
 performance\_metrics: Dict, model\_type: Optional[str] = None,  
 task\_type: Optional[str] = None, metadata: Optional[Dict] = None):  
 """Register a configuration for a material type"""  
  
 def get\_configurations(self, material\_type: str, model\_type: Optional[str] = None,  
 task\_type: Optional[str] = None) -> List[Dict]:  
 """Get all configurations for a material type with optional filtering"""  
  
 def get\_best\_configuration(self, material\_type: str, metric: str = "val\_loss",  
 higher\_is\_better: bool = False) -> Optional[Dict]:  
 """Get best configuration for a material type based on metric"""  
  
 def calculate\_material\_similarity(self, type1: str, type2: str) -> float:  
 """Calculate similarity between two material types"""  
  
 def get\_similar\_material\_types(self, material\_type: str,   
 similarity\_threshold: float = 0.5) -> List[Tuple[str, float]]:  
 """Get similar material types with similarity scores"""  
  
 def suggest\_initial\_configuration(self, material\_type: str,   
 model\_type: Optional[str] = None,  
 task\_type: Optional[str] = None) -> Dict:  
 """Suggest initial configuration for a material type"""  
  
 def warm\_start\_parameter\_space(self, material\_type: str,   
 param\_space: Dict) -> Dict:  
 """Warm-start parameter space with knowledge from registry"""

#### MaterialHyperparameters

class MaterialHyperparameters:  
 @classmethod  
 def get\_defaults(cls, material\_type: str) -> Dict:  
 """Get default hyperparameters for a material type"""

class MaterialHyperparameters:  
 @classmethod  
 def get\_defaults(cls, material\_type: str) -> Dict:  
 """Get default hyperparameters for a material type"""

### Customization and Extension

The Parameter Registry System is designed to be extensible in several ways:

#### Adding New Similarity Metrics

# Add custom similarity metrics to ParameterRegistry  
def \_custom\_similarity\_metric(self, type1: str, type2: str) -> float:  
 # Custom similarity calculation  
 similarity = ...  
 return similarity  
  
# Add to similarity methods  
ParameterRegistry.\_custom\_similarity\_metric = \_custom\_similarity\_metric  
registry.similarity\_methods.append(registry.\_custom\_similarity\_metric)

# Add custom similarity metrics to ParameterRegistry  
def \_custom\_similarity\_metric(self, type1: str, type2: str) -> float:  
 # Custom similarity calculation  
 similarity = ...  
 return similarity  
  
# Add to similarity methods  
ParameterRegistry.\_custom\_similarity\_metric = \_custom\_similarity\_metric  
registry.similarity\_methods.append(registry.\_custom\_similarity\_metric)

#### Creating Custom Material Defaults

# Add custom defaults to MaterialHyperparameters  
MaterialHyperparameters.MATERIAL\_DEFAULTS["my\_custom\_material"] = {  
 "architecture": "custom\_net",  
 "learning\_rate": 0.0002,  
 "batch\_size": 8,  
 "weight\_decay": 2e-5,  
 "data\_augmentation": "custom\_augmentation"  
}

# Add custom defaults to MaterialHyperparameters  
MaterialHyperparameters.MATERIAL\_DEFAULTS["my\_custom\_material"] = {  
 "architecture": "custom\_net",  
 "learning\_rate": 0.0002,  
 "batch\_size": 8,  
 "weight\_decay": 2e-5,  
 "data\_augmentation": "custom\_augmentation"  
}

#### Integration with External Optimization Frameworks

The Parameter Registry System can be integrated with popular optimization frameworks:

* Optuna - For advanced hyperparameter optimization
* Ray Tune - For distributed hyperparameter tuning
* Weights & Biases Sweeps - For visualization and experiment tracking

# Example integration with Optuna  
import optuna  
  
def objective(trial):  
 # Get material type  
 material\_type = "ceramic\_tile"  
  
 # Get suggested parameters  
 suggested\_params = registry.suggest\_initial\_configuration(material\_type)  
  
 # Use suggested parameters to create Optuna trial  
 learning\_rate = trial.suggest\_float(  
 "learning\_rate",  
 suggested\_params.get("learning\_rate", 0.001) \* 0.1,  
 suggested\_params.get("learning\_rate", 0.001) \* 10,  
 log=True  
 )  
  
 # Continue with other parameters...  
  
 # Train and return result  
 model = train\_model(learning\_rate=learning\_rate, ...)  
 return model.validation\_accuracy

# Example integration with Optuna  
import optuna  
  
def objective(trial):  
 # Get material type  
 material\_type = "ceramic\_tile"  
  
 # Get suggested parameters  
 suggested\_params = registry.suggest\_initial\_configuration(material\_type)  
  
 # Use suggested parameters to create Optuna trial  
 learning\_rate = trial.suggest\_float(  
 "learning\_rate",  
 suggested\_params.get("learning\_rate", 0.001) \* 0.1,  
 suggested\_params.get("learning\_rate", 0.001) \* 10,  
 log=True  
 )  
  
 # Continue with other parameters...  
  
 # Train and return result  
 model = train\_model(learning\_rate=learning\_rate, ...)  
 return model.validation\_accuracy

### Implementation Notes

The Parameter Registry System is implemented in Python with these dependencies:

* NumPy - For numerical operations
* Difflib - For sequence matching in similarity calculations
* JSON - For persistent storage of configurations

There are no external web service requirements, making the system suitable for both online and offline use.

### Future Improvements

Planned enhancements for the Parameter Registry System include:

1. Automated Registry Pruning - Remove outdated or underperforming configurations
2. Hierarchical Material Classification - Organize materials in a taxonomy for better similarity matching
3. Bayesian Optimization Integration - Directly incorporate Bayesian optimization for parameter suggestions
4. Distributed Registry - Support for distributed databases for enterprise-scale deployment
5. Active Learning Loop - Automatically trigger new training runs to explore promising parameter regions

### Usage Examples

#### Basic Usage

# Initialize registry  
registry = ParameterRegistry("registry.json")  
  
# Get suggested parameters for a material  
params = registry.suggest\_initial\_configuration("marble\_tile")  
  
# Train model with suggested parameters  
results = train\_model(params)  
  
# Register results back to registry  
registry.register\_configuration(  
 material\_type="marble\_tile",  
 params=params,  
 performance\_metrics=results  
)

# Initialize registry  
registry = ParameterRegistry("registry.json")  
  
# Get suggested parameters for a material  
params = registry.suggest\_initial\_configuration("marble\_tile")  
  
# Train model with suggested parameters  
results = train\_model(params)  
  
# Register results back to registry  
registry.register\_configuration(  
 material\_type="marble\_tile",  
 params=params,  
 performance\_metrics=results  
)

#### Advanced Usage with Parameter Space Warm-Starting

# Define parameter space for hyperparameter optimization  
param\_space = {  
 "learning\_rate": {"type": "float", "min": 1e-4, "max": 1e-2, "log\_scale": True},  
 "batch\_size": {"type": "choice", "values": [8, 16, 32, 64]},  
 "architecture": {"type": "categorical", "values": ["mobilenet", "resnet", "vit", "hybrid-cnn-vit"]},  
 "weight\_decay": {"type": "float", "min": 1e-6, "max": 1e-4, "log\_scale": True},  
 "dropout": {"type": "float", "min": 0.0, "max": 0.5}  
}  
  
# Warm-start the parameter space with registry suggestions  
warm\_started\_space = registry.warm\_start\_parameter\_space("porcelain\_tile", param\_space)  
  
# Use warm-started space with your favorite optimization framework  
study = optuna.create\_study(direction="maximize")  
study.optimize(lambda trial: objective(trial, warm\_started\_space), n\_trials=100)

# Define parameter space for hyperparameter optimization  
param\_space = {  
 "learning\_rate": {"type": "float", "min": 1e-4, "max": 1e-2, "log\_scale": True},  
 "batch\_size": {"type": "choice", "values": [8, 16, 32, 64]},  
 "architecture": {"type": "categorical", "values": ["mobilenet", "resnet", "vit", "hybrid-cnn-vit"]},  
 "weight\_decay": {"type": "float", "min": 1e-6, "max": 1e-4, "log\_scale": True},  
 "dropout": {"type": "float", "min": 0.0, "max": 0.5}  
}  
  
# Warm-start the parameter space with registry suggestions  
warm\_started\_space = registry.warm\_start\_parameter\_space("porcelain\_tile", param\_space)  
  
# Use warm-started space with your favorite optimization framework  
study = optuna.create\_study(direction="maximize")  
study.optimize(lambda trial: objective(trial, warm\_started\_space), n\_trials=100)

#### Registry Analysis and Visualization

```python

## Get statistics for all material types

material\_stats = {}  
for material\_type in registry.get\_material\_types():  
 stats = registry.get\_material\_type\_statistics(material\_type)  
 material\_stats[material\_type] = stats

## Analyze parameter importance across materials

param\_importance = analyze\_parameter\_importance(material\_stats)  
print("Most important parameters:", param\_importance)

## Find clusters of similar materials

material\_clusters = cluster\_similar\_materials(registry)  
print("Material clusters:", material\_clusters)

# Pdf Processing

Source: readme/pdf-processing.md

---

## PDF Processing System

The PDF Processing System is a specialized component of Kai that extracts images, text, and structured data from material catalogs. This document details the system's capabilities, architecture, and implementation.

### Features

#### Image Extraction

The PDF Processing System can extract high-quality images of materials from catalogs:

1. Intelligent Image Extraction
2. Identification of material images vs. decorative elements
3. Boundary detection for separating merged images
4. Resolution enhancement for low-quality images
5. Background removal and transparency handling
6. Image deduplication across multiple pages
7. Image Processing
8. Color normalization for consistent representation
9. Perspective correction for angled views
10. Artifact removal (watermarks, text overlays)
11. Multi-resolution output generation
12. Image metadata extraction (dimensions, resolution)
13. Image Classification
14. Categorization of images (product, environment, detail)
15. Primary vs. secondary image identification
16. Texture and pattern extraction
17. Feature extraction for recognition

Image deduplication across multiple pages

Image Processing

Image metadata extraction (dimensions, resolution)

Image Classification

#### Text Extraction & OCR

The system includes advanced OCR capabilities:

1. Enhanced OCR
2. Region-based OCR optimization
3. Multi-language support
4. Domain-specific vocabulary enhancement
5. Handwritten text recognition for notes
6. Table structure recognition
7. Text Processing
8. Cleaning and normalization
9. Entity recognition (product codes, dimensions)
10. Technical specification parsing
11. Unit standardization
12. Language detection and translation
13. OCR Confidence Scoring
14. Accuracy estimation for extracted text
15. Confidence heatmaps for verification
16. Alternative readings for uncertain text
17. Human verification flagging

Table structure recognition

Text Processing

Language detection and translation

OCR Confidence Scoring

#### Layout Analysis

The system performs comprehensive page layout analysis:

1. Page Segmentation
2. Logical region identification (headers, body, footnotes)
3. Multi-column detection and ordering
4. Table structure recognition
5. List and bullet point identification
6. Caption and label association
7. Document Structure
8. Table of contents parsing
9. Section and chapter detection
10. Cross-reference identification
11. Hierarchical document mapping
12. Logical flow reconstruction
13. Visual Element Analysis
14. Chart and graph detection
15. Diagram interpretation
16. Icon and symbol recognition
17. Color palette extraction
18. Design element identification

Caption and label association

Document Structure

Logical flow reconstruction

Visual Element Analysis

#### Text-Image Association

The system connects extracted text with corresponding images:

1. Association Methods
2. Spatial proximity analysis
3. Reference identification (figure numbers)
4. Caption-to-image matching
5. Visual element correlation
6. Cross-page reference tracking
7. Metadata Mapping
8. Product code to image mapping
9. Specification text to image linking
10. Dimension text to image correlation
11. Color name to visual color matching
12. Technical data to visual feature association
13. Validation Techniques
14. Consistency checking across associations
15. Probability scoring for uncertain matches
16. Pattern-based validation
17. Cross-reference confirmation
18. Human verification flagging

Cross-page reference tracking

Metadata Mapping

Technical data to visual feature association

Validation Techniques

#### Structured Data Extraction

The system extracts structured material specifications:

1. Specification Extraction
2. Technical property identification
3. Dimension and measurement parsing
4. Material composition detection
5. Application and usage extraction
6. Performance rating identification
7. Field Mapping
8. Knowledge base schema alignment
9. Field normalization and standardization
10. Unit conversion to standard formats
11. Vocabulary mapping to controlled terminology
12. Missing value inference
13. Validation and Quality Control
14. Range checking for numeric values
15. Format validation for standard fields
16. Cross-reference checking
17. Completeness assessment
18. Confidence scoring

Performance rating identification

Field Mapping

Missing value inference

Validation and Quality Control

### Technical Implementation

#### Processing Pipeline

The PDF processing pipeline consists of several stages:

1. Document Preprocessing
2. PDF validation and metadata extraction
3. Page count and structure analysis
4. Resolution assessment
5. Text layer detection
6. Encryption and permission handling
7. Page Analysis
8. Page segmentation into logical regions
9. Layout analysis and region classification
10. Text extraction from native PDF text layer (if available)
11. OCR preparation for image-based content
12. Visual element identification
13. Image Processing
14. Image extraction from PDF elements
15. Image boundary detection and cropping
16. Enhancement and normalization
17. Deduplication and variant identification
18. Feature extraction for classification
19. Text Processing
20. OCR for image-based text
21. Text cleaning and normalization
22. Entity recognition and classification
23. Table and structured content parsing
24. Language detection and processing
25. Association and Mapping
26. Text-image association
27. Structured data extraction
28. Knowledge base schema mapping
29. Validation and confidence scoring
30. Output generation

Encryption and permission handling

Page Analysis

Visual element identification

Image Processing

Feature extraction for classification

Text Processing

Language detection and processing

Association and Mapping

#### Enhanced Preprocessing

The system includes specialized preprocessing techniques:

interface PreprocessingOptions {  
 enhanceResolution: boolean; // Apply super-resolution to low-quality images  
 removeBackground: boolean; // Detect and remove background elements  
 deskew: boolean; // Correct page skew and orientation  
 detectBoundaries: boolean; // Enhance boundary detection between elements  
 denoise: boolean; // Remove noise artifacts  
 colorNormalize: boolean; // Standardize color representation  
}  
  
interface PreprocessingResult {  
 enhancedPages: Array<{  
 pageNumber: number;  
 enhancedImagePath: string;  
 enhancementMetrics: Record<string, number>;  
 regions: Array<{  
 type: 'text' | 'image' | 'table' | 'diagram' | 'mixed';  
 bbox: [number, number, number, number]; // x, y, width, height  
 confidence: number;  
 }>;  
 }>;  
 metadata: {  
 enhancementApplied: string[];  
 originalQuality: Record<string, number>;  
 enhancedQuality: Record<string, number>;  
 };  
}

interface PreprocessingOptions {  
 enhanceResolution: boolean; // Apply super-resolution to low-quality images  
 removeBackground: boolean; // Detect and remove background elements  
 deskew: boolean; // Correct page skew and orientation  
 detectBoundaries: boolean; // Enhance boundary detection between elements  
 denoise: boolean; // Remove noise artifacts  
 colorNormalize: boolean; // Standardize color representation  
}  
  
interface PreprocessingResult {  
 enhancedPages: Array<{  
 pageNumber: number;  
 enhancedImagePath: string;  
 enhancementMetrics: Record<string, number>;  
 regions: Array<{  
 type: 'text' | 'image' | 'table' | 'diagram' | 'mixed';  
 bbox: [number, number, number, number]; // x, y, width, height  
 confidence: number;  
 }>;  
 }>;  
 metadata: {  
 enhancementApplied: string[];  
 originalQuality: Record<string, number>;  
 enhancedQuality: Record<string, number>;  
 };  
}

#### Region-Based OCR

The system optimizes OCR for different types of regions:

interface OCROptions {  
 mode: 'standard' | 'dense' | 'sparse' | 'technical' | 'handwritten';  
 languages: string[];  
 enhanceText: boolean;  
 recognizeHandwriting: boolean;  
 detectOrientation: boolean;  
 recognizeTables: boolean;  
 confidenceThreshold: number;  
 outputFormats: Array<'text' | 'hocr' | 'alto' | 'json'>;  
}  
  
interface OCRResult {  
 regions: Array<{  
 regionId: string;  
 regionType: 'paragraph' | 'heading' | 'table' | 'caption' | 'list' | 'technical';  
 content: string;  
 bbox: [number, number, number, number];  
 confidence: number;  
 wordData?: Array<{  
 word: string;  
 bbox: [number, number, number, number];  
 confidence: number;  
 }>;  
 alternatives?: Array<{  
 content: string;  
 confidence: number;  
 }>;  
 metadata?: Record<string, any>;  
 }>;  
 pageMetadata: {  
 pageNumber: number;  
 dimensions: [number, number];  
 dpi: number;  
 orientation: 'portrait' | 'landscape';  
 languages: Array<{  
 language: string;  
 confidence: number;  
 }>;  
 };  
 statistics: {  
 characterCount: number;  
 wordCount: number;  
 meanConfidence: number;  
 processingTimeMs: number;  
 };  
}

interface OCROptions {  
 mode: 'standard' | 'dense' | 'sparse' | 'technical' | 'handwritten';  
 languages: string[];  
 enhanceText: boolean;  
 recognizeHandwriting: boolean;  
 detectOrientation: boolean;  
 recognizeTables: boolean;  
 confidenceThreshold: number;  
 outputFormats: Array<'text' | 'hocr' | 'alto' | 'json'>;  
}  
  
interface OCRResult {  
 regions: Array<{  
 regionId: string;  
 regionType: 'paragraph' | 'heading' | 'table' | 'caption' | 'list' | 'technical';  
 content: string;  
 bbox: [number, number, number, number];  
 confidence: number;  
 wordData?: Array<{  
 word: string;  
 bbox: [number, number, number, number];  
 confidence: number;  
 }>;  
 alternatives?: Array<{  
 content: string;  
 confidence: number;  
 }>;  
 metadata?: Record<string, any>;  
 }>;  
 pageMetadata: {  
 pageNumber: number;  
 dimensions: [number, number];  
 dpi: number;  
 orientation: 'portrait' | 'landscape';  
 languages: Array<{  
 language: string;  
 confidence: number;  
 }>;  
 };  
 statistics: {  
 characterCount: number;  
 wordCount: number;  
 meanConfidence: number;  
 processingTimeMs: number;  
 };  
}

#### Text-Image Association

The system links extracted text with corresponding images:

interface AssociationOptions {  
 associationMethods: Array<'spatial' | 'reference' | 'caption' | 'content'>;  
 minConfidence: number;  
 resolveConflicts: 'highest\_confidence' | 'multiple' | 'none';  
 maxDistance: number; // Maximum spatial distance for association  
 enableCrossPage: boolean; // Whether to look for associations across pages  
}  
  
interface AssociationResult {  
 associations: Array<{  
 imageId: string;  
 textRegions: string[];  
 associationType: 'direct' | 'caption' | 'reference' | 'inferred';  
 confidence: number;  
 metadata: Record<string, any>;  
 }>;  
 unmatchedImages: string[];  
 unmatchedTextRegions: string[];  
 statistics: {  
 associationCount: number;  
 meanConfidence: number;  
 conflictCount: number;  
 crossPageAssociations: number;  
 };  
}

interface AssociationOptions {  
 associationMethods: Array<'spatial' | 'reference' | 'caption' | 'content'>;  
 minConfidence: number;  
 resolveConflicts: 'highest\_confidence' | 'multiple' | 'none';  
 maxDistance: number; // Maximum spatial distance for association  
 enableCrossPage: boolean; // Whether to look for associations across pages  
}  
  
interface AssociationResult {  
 associations: Array<{  
 imageId: string;  
 textRegions: string[];  
 associationType: 'direct' | 'caption' | 'reference' | 'inferred';  
 confidence: number;  
 metadata: Record<string, any>;  
 }>;  
 unmatchedImages: string[];  
 unmatchedTextRegions: string[];  
 statistics: {  
 associationCount: number;  
 meanConfidence: number;  
 conflictCount: number;  
 crossPageAssociations: number;  
 };  
}

#### Form Field Extraction

The system can extract data from structured form fields:

interface FormFieldExtractionOptions {  
 fieldTypes: Array<'text' | 'checkbox' | 'radio' | 'table' | 'signature'>;  
 namedFields: Record<string, {  
 type: string;  
 pattern?: string;  
 required?: boolean;  
 validation?: 'numeric' | 'alphanumeric' | 'email' | 'date' | 'custom';  
 validationPattern?: string;  
 }>;  
 detectUnlabeled: boolean;  
 extractSignatures: boolean;  
}  
  
interface FormFieldExtractionResult {  
 fields: Array<{  
 name: string;  
 value: string | boolean | string[][];  
 type: string;  
 bbox: [number, number, number, number];  
 pageNumber: number;  
 confidence: number;  
 validated: boolean;  
 validationMessage?: string;  
 }>;  
 statistics: {  
 totalFields: number;  
 filledFields: number;  
 validFields: number;  
 emptyFields: number;  
 invalidFields: number;  
 };  
}

interface FormFieldExtractionOptions {  
 fieldTypes: Array<'text' | 'checkbox' | 'radio' | 'table' | 'signature'>;  
 namedFields: Record<string, {  
 type: string;  
 pattern?: string;  
 required?: boolean;  
 validation?: 'numeric' | 'alphanumeric' | 'email' | 'date' | 'custom';  
 validationPattern?: string;  
 }>;  
 detectUnlabeled: boolean;  
 extractSignatures: boolean;  
}  
  
interface FormFieldExtractionResult {  
 fields: Array<{  
 name: string;  
 value: string | boolean | string[][];  
 type: string;  
 bbox: [number, number, number, number];  
 pageNumber: number;  
 confidence: number;  
 validated: boolean;  
 validationMessage?: string;  
 }>;  
 statistics: {  
 totalFields: number;  
 filledFields: number;  
 validFields: number;  
 emptyFields: number;  
 invalidFields: number;  
 };  
}

### Integration with Knowledge Base

The PDF Processing System integrates with the Knowledge Base:

1. Data Transformation
2. Mapping extracted data to knowledge base schema
3. Entity resolution for existing materials
4. Relationship identification and creation
5. Collection and series mapping
6. Duplicate detection and handling
7. Quality Assurance
8. Validation against knowledge base constraints
9. Confidence thresholds for automatic import
10. Human verification workflow for low-confidence data
11. Anomaly detection for unusual values
12. Cross-reference checking with existing data
13. Incremental Updates
14. Version management for updated materials
15. Change tracking and attribution
16. Selective updating of specific fields
17. Catalog version correlation
18. Update propagation to related entities

Duplicate detection and handling

Quality Assurance

Cross-reference checking with existing data

Incremental Updates

### Queue System Integration

The PDF Processing System is integrated with the queue system:

1. Job Management
2. Job creation for PDF processing
3. Priority setting for urgent catalogs
4. Progress tracking and reporting
5. Error handling and recovery
6. Resource allocation
7. Pipeline Stages
8. Parallel processing of independent stages
9. Results aggregation and validation
10. Stage-specific error handling
11. Progress reporting for each stage
12. Smart retries for failed stages
13. Event-Based Coordination
14. Status updates via the message broker
15. Knowledge base notifications for new data
16. Admin panel updates for monitoring
17. Error alerts and notifications
18. Completion events for dependent processes

Resource allocation

Pipeline Stages

Smart retries for failed stages

Event-Based Coordination

### API Usage

#### Basic PDF Processing

import { extractFromPDF } from '@kai/ml';  
  
async function processPDF() {  
 try {  
 const result = await extractFromPDF('path/to/catalog.pdf', 'output/directory', {  
 extractImages: true,  
 extractText: true,  
 enhanceResolution: true,  
 associateTextWithImages: true,  
 extractStructuredData: true  
 });  
  
 console.log(`Processed ${result.pageCount} pages`);  
 console.log(`Extracted ${result.images.length} images and ${result.textRegions.length} text regions`);  
 console.log(`Found ${result.materials.length} potential materials`);  
 } catch (error) {  
 console.error('PDF processing failed:', error);  
 }  
}

import { extractFromPDF } from '@kai/ml';  
  
async function processPDF() {  
 try {  
 const result = await extractFromPDF('path/to/catalog.pdf', 'output/directory', {  
 extractImages: true,  
 extractText: true,  
 enhanceResolution: true,  
 associateTextWithImages: true,  
 extractStructuredData: true  
 });  
  
 console.log(`Processed ${result.pageCount} pages`);  
 console.log(`Extracted ${result.images.length} images and ${result.textRegions.length} text regions`);  
 console.log(`Found ${result.materials.length} potential materials`);  
 } catch (error) {  
 console.error('PDF processing failed:', error);  
 }  
}

#### Enhanced OCR Processing

import { performEnhancedOCR } from '@kai/ml';  
  
async function processOCR() {  
 try {  
 const result = await performEnhancedOCR('path/to/page.jpg', {  
 mode: 'technical',  
 languages: ['en', 'es', 'fr'],  
 enhanceText: true,  
 recognizeTables: true,  
 confidenceThreshold: 0.75  
 });  
  
 console.log(`Extracted ${result.regions.length} text regions`);  
 console.log(`Overall confidence: ${result.statistics.meanConfidence.toFixed(2)}`);  
  
 // Find high-value information like product codes  
 const productCodes = result.regions  
 .filter(region => region.content.match(/[A-Z]{2,}-\d{3,}/))  
 .map(region => region.content.trim());  
  
 console.log('Potential product codes:', productCodes);  
 } catch (error) {  
 console.error('OCR processing failed:', error);  
 }  
}

import { performEnhancedOCR } from '@kai/ml';  
  
async function processOCR() {  
 try {  
 const result = await performEnhancedOCR('path/to/page.jpg', {  
 mode: 'technical',  
 languages: ['en', 'es', 'fr'],  
 enhanceText: true,  
 recognizeTables: true,  
 confidenceThreshold: 0.75  
 });  
  
 console.log(`Extracted ${result.regions.length} text regions`);  
 console.log(`Overall confidence: ${result.statistics.meanConfidence.toFixed(2)}`);  
  
 // Find high-value information like product codes  
 const productCodes = result.regions  
 .filter(region => region.content.match(/[A-Z]{2,}-\d{3,}/))  
 .map(region => region.content.trim());  
  
 console.log('Potential product codes:', productCodes);  
 } catch (error) {  
 console.error('OCR processing failed:', error);  
 }  
}

#### Text-Image Association

import { associateTextWithImages } from '@kai/ml';  
  
async function linkTextAndImages() {  
 try {  
 const result = await associateTextWithImages({  
 images: ['path/to/image1.jpg', 'path/to/image2.jpg'],  
 textRegions: ocrResults.regions,  
 associationMethods: ['spatial', 'caption', 'reference'],  
 minConfidence: 0.6  
 });  
  
 console.log(`Created ${result.associations.length} text-image associations`);  
  
 // Process the associations  
 result.associations.forEach(assoc => {  
 console.log(`Image ${assoc.imageId} is associated with ${assoc.textRegions.length} text regions`);  
 console.log(`Association confidence: ${assoc.confidence.toFixed(2)}`);  
 });  
 } catch (error) {  
 console.error('Text-image association failed:', error);  
 }  
}

import { associateTextWithImages } from '@kai/ml';  
  
async function linkTextAndImages() {  
 try {  
 const result = await associateTextWithImages({  
 images: ['path/to/image1.jpg', 'path/to/image2.jpg'],  
 textRegions: ocrResults.regions,  
 associationMethods: ['spatial', 'caption', 'reference'],  
 minConfidence: 0.6  
 });  
  
 console.log(`Created ${result.associations.length} text-image associations`);  
  
 // Process the associations  
 result.associations.forEach(assoc => {  
 console.log(`Image ${assoc.imageId} is associated with ${assoc.textRegions.length} text regions`);  
 console.log(`Association confidence: ${assoc.confidence.toFixed(2)}`);  
 });  
 } catch (error) {  
 console.error('Text-image association failed:', error);  
 }  
}

#### Material Extraction

import { extractMaterialsFromPDF } from '@kai/ml';  
  
async function extractMaterials() {  
 try {  
 const result = await extractMaterialsFromPDF('path/to/catalog.pdf', {  
 extractImages: true,  
 enhanceResolution: true,  
 recognizeText: true,  
 identifyMaterials: true,  
 mapToKnowledgeBase: true,  
 confidence: 'high'  
 });  
  
 console.log(`Extracted ${result.materials.length} materials`);  
  
 // Example of processing the extracted materials  
 result.materials.forEach(material => {  
 console.log(`Material: ${material.name}`);  
 console.log(`Type: ${material.materialType}`);  
 console.log(`Product Code: ${material.productCode}`);  
 console.log(`Dimensions: ${JSON.stringify(material.dimensions)}`);  
 console.log(`Images: ${material.images.length}`);  
 });  
  
 // Example of importing to knowledge base  
 if (result.materials.length > 0) {  
 const importResult = await knowledgeBaseService.bulkImportMaterials(  
 result.materials,  
 'system',  
 {  
 updateExisting: true,  
 detectDuplicates: true,  
 collectionId: 'catalog-collection-id'  
 }  
 );  
  
 console.log(`Imported ${importResult.imported} materials`);  
 console.log(`Updated ${importResult.updated} existing materials`);  
 console.log(`Failed to import ${importResult.failed} materials`);  
 }  
 } catch (error) {  
 console.error('Material extraction failed:', error);  
 }  
}

import { extractMaterialsFromPDF } from '@kai/ml';  
  
async function extractMaterials() {  
 try {  
 const result = await extractMaterialsFromPDF('path/to/catalog.pdf', {  
 extractImages: true,  
 enhanceResolution: true,  
 recognizeText: true,  
 identifyMaterials: true,  
 mapToKnowledgeBase: true,  
 confidence: 'high'  
 });  
  
 console.log(`Extracted ${result.materials.length} materials`);  
  
 // Example of processing the extracted materials  
 result.materials.forEach(material => {  
 console.log(`Material: ${material.name}`);  
 console.log(`Type: ${material.materialType}`);  
 console.log(`Product Code: ${material.productCode}`);  
 console.log(`Dimensions: ${JSON.stringify(material.dimensions)}`);  
 console.log(`Images: ${material.images.length}`);  
 });  
  
 // Example of importing to knowledge base  
 if (result.materials.length > 0) {  
 const importResult = await knowledgeBaseService.bulkImportMaterials(  
 result.materials,  
 'system',  
 {  
 updateExisting: true,  
 detectDuplicates: true,  
 collectionId: 'catalog-collection-id'  
 }  
 );  
  
 console.log(`Imported ${importResult.imported} materials`);  
 console.log(`Updated ${importResult.updated} existing materials`);  
 console.log(`Failed to import ${importResult.failed} materials`);  
 }  
 } catch (error) {  
 console.error('Material extraction failed:', error);  
 }  
}

### Performance Considerations

1. Processing Optimization
2. Parallel processing of pages for faster throughput
3. GPU acceleration for image processing and OCR
4. Caching of intermediate results
5. Progressive processing (quick first pass, then detailed)
6. Resource-adaptive processing based on document complexity
7. Resource Requirements
8. CPU: Multiple cores recommended for parallel processing
9. Memory: 8GB+ for large catalogs (scales with page count and complexity)
10. GPU: Recommended for OCR and image enhancement
11. Storage: Temporary space for extracted assets (5-10x the PDF size)
12. Network: Required for knowledge base integration
13. Scaling Considerations
14. Horizontal scaling for processing multiple documents
15. Job priority system for urgent processing
16. Batch processing for large catalogs
17. Sequential processing for memory-constrained environments
18. Distributed processing for multi-stage pipeline

Resource-adaptive processing based on document complexity

Resource Requirements

Network: Required for knowledge base integration

Scaling Considerations

# Project Assistant

Source: readme/project-assistant.md

---

## Project Assistant

This document provides detailed information about the Project Assistant, a specialized crewAI agent designed to help with material project planning and organization within the KAI platform.

### Overview

The Project Assistant agent helps users organize materials into cohesive projects, calculate quantities, estimate costs, and plan material applications. It serves as a project planning specialist that transforms material selections into actionable project plans with practical implementation guidance.

### Key Capabilities

The Project Assistant offers multiple specialized functions:

1. Project Organization
2. Help users organize selected materials into coherent projects
3. Create logical groupings based on rooms or application areas
4. Suggest complementary materials to complete a project
5. Maintain consistency across material selections
6. Quantity Calculation
7. Calculate required material quantities based on dimensions
8. Account for waste factors and installation patterns
9. Provide accurate material ordering guidance
10. Help optimize material usage to minimize waste
11. Cost Estimation
12. Provide estimated cost ranges for selected materials
13. Calculate total project material costs
14. Suggest budget alternatives when appropriate
15. Help users understand price-quality tradeoffs
16. Application Planning
17. Provide guidance on material application techniques
18. Create step-by-step installation sequences
19. Suggest required tools and accessories
20. Outline preparation requirements for surfaces
21. Timeline Management
22. Help estimate project duration based on scope
23. Create logical installation sequencing
24. Identify potential schedule dependencies
25. Suggest efficient project workflows

Maintain consistency across material selections

Quantity Calculation

Help optimize material usage to minimize waste

Cost Estimation

Help users understand price-quality tradeoffs

Application Planning

Outline preparation requirements for surfaces

Timeline Management

### Architecture

The Project Assistant integrates with the broader KAI platform through several key components:

#### Component Structure

packages/  
├── agents/  
│ ├── src/  
│ │ ├── frontend/  
│ │ │ └── projectAssistant.ts # Agent implementation  
│ │ ├── services/  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ ├── materialSearch.ts # Material search tool  
│ │ │ ├── vectorSearch.ts # Vector search tool  
│ │ │ └── index.ts # Tool exports  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── client/  
 └── src/  
 └── components/  
 └── agents/  
 ├── ProjectAssistantPanel.tsx # Client-side interface  
 └── AgentDashboard.tsx # Agent integration in UI

packages/  
├── agents/  
│ ├── src/  
│ │ ├── frontend/  
│ │ │ └── projectAssistant.ts # Agent implementation  
│ │ ├── services/  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ ├── materialSearch.ts # Material search tool  
│ │ │ ├── vectorSearch.ts # Vector search tool  
│ │ │ └── index.ts # Tool exports  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── client/  
 └── src/  
 └── components/  
 └── agents/  
 ├── ProjectAssistantPanel.tsx # Client-side interface  
 └── AgentDashboard.tsx # Agent integration in UI

#### Architectural Layers

1. Agent Layer (projectAssistant.ts)
2. Implements the agent's core capabilities
3. Defines specialized methods for project planning tasks
4. Processes user queries about project organization and planning
5. Manages context for comprehensive project planning
6. Service Layer (via ServiceFactory)
7. Provides access to material database and pricing information
8. Handles API communication with error management
9. Formats requests and responses appropriately
10. Acts as a bridge to backend material services
11. Tool Layer (materialSearch, vectorSearch)
12. Implements specialized tools for the agent to use
13. Enables text-based material database queries
14. Provides vector-based similarity searches
15. Formats results for agent consumption
16. UI Layer (ProjectAssistantPanel.tsx)
17. Presents the agent's capabilities in the user interface
18. Provides chat interface for project planning queries
19. Displays project organization with material groupings
20. Facilitates material quantity and cost calculations

projectAssistant.ts

Manages context for comprehensive project planning

Service Layer (via ServiceFactory)

Acts as a bridge to backend material services

Tool Layer (materialSearch, vectorSearch)

Formats results for agent consumption

UI Layer (ProjectAssistantPanel.tsx)

ProjectAssistantPanel.tsx

### Implementation Details

#### Agent Implementation

The Project Assistant is a UserFacingAgent type that implements specialized methods for project planning tasks:

export class ProjectAssistant implements UserFacingAgent {  
 // Standard UserFacingAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // Project-specific methods  
 public async processUserInput(message: string): Promise<string>;  
}

export class ProjectAssistant implements UserFacingAgent {  
 // Standard UserFacingAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // Project-specific methods  
 public async processUserInput(message: string): Promise<string>;  
}

#### Agent Tools

The Project Assistant leverages specialized tools to perform its tasks:

// Tool for searching materials in the database  
const materialSearchTool = await createMaterialSearchTool();  
  
// Tool for performing vector-based similarity searches  
const vectorSearchTool = await createVectorSearchTool();  
  
// Create a Project Assistant with tools  
const tools = [  
 materialSearchTool,  
 vectorSearchTool  
];  
  
// Additional tools can be added from the configuration  
if (config.additionalTools) {  
 tools.push(...config.additionalTools);  
}

// Tool for searching materials in the database  
const materialSearchTool = await createMaterialSearchTool();  
  
// Tool for performing vector-based similarity searches  
const vectorSearchTool = await createVectorSearchTool();  
  
// Create a Project Assistant with tools  
const tools = [  
 materialSearchTool,  
 vectorSearchTool  
];  
  
// Additional tools can be added from the configuration  
if (config.additionalTools) {  
 tools.push(...config.additionalTools);  
}

#### Agent Description

The Project Assistant is defined with the following characteristics:

const agent = new Agent({  
 name: 'Project Assistant',  
 role: 'Project planning specialist who helps organize materials and estimate quantities',  
 goal: 'Help users plan and organize their materials into cohesive projects with accurate quantities and costs',  
 backstory: 'With expertise in project management and material application, I can help you organize materials, calculate quantities, and plan material applications for optimal results.',  
 verbose: config.verbose || false,  
 llm: modelSettings,  
 tools  
});

const agent = new Agent({  
 name: 'Project Assistant',  
 role: 'Project planning specialist who helps organize materials and estimate quantities',  
 goal: 'Help users plan and organize their materials into cohesive projects with accurate quantities and costs',  
 backstory: 'With expertise in project management and material application, I can help you organize materials, calculate quantities, and plan material applications for optimal results.',  
 verbose: config.verbose || false,  
 llm: modelSettings,  
 tools  
});

#### Client-Side Integration

The Project Assistant is integrated into the client interface through a specialized panel that provides:

1. Chat Interface - For asking questions about project planning
2. Material Grouping - For organizing materials into logical project sections
3. Quantity Calculator - For determining required material amounts
4. Cost Estimator - For projecting total material costs
5. Application Guide - For step-by-step material application instructions

### Setup Instructions

#### Prerequisites

* Functioning KAI platform with material database
* CrewAI integration set up according to CrewAI installation guide
* Vector search capabilities for material similarity

#### Installation

The Project Assistant is included in the standard crewAI integration package:

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

#### Configuration

Configure the agent in your application initialization:

import { createProjectAssistant } from '@kai/agents';  
  
// Create a Project Assistant instance  
const projectAssistant = await createProjectAssistant(  
 {  
 id: 'project-assistant-1',  
 name: 'Project Assistant',  
 description: 'Expert in project planning and material organization',  
 verbose: true,  
 // Additional configuration options  
 },  
 {  
 model: 'gpt-4',  
 temperature: 0.3  
 }  
);

import { createProjectAssistant } from '@kai/agents';  
  
// Create a Project Assistant instance  
const projectAssistant = await createProjectAssistant(  
 {  
 id: 'project-assistant-1',  
 name: 'Project Assistant',  
 description: 'Expert in project planning and material organization',  
 verbose: true,  
 // Additional configuration options  
 },  
 {  
 model: 'gpt-4',  
 temperature: 0.3  
 }  
);

### Usage Examples

#### Client-Side Integration

import React from 'react';  
import { ProjectAssistantPanel } from '../components/agents/ProjectAssistantPanel';  
  
const ProjectAssistantPage: React.FC = () => {  
 return (  
 <div className="project-assistant-page">  
 <h1>Project Assistant</h1>  
 <ProjectAssistantPanel />  
 </div>  
 );  
};  
  
export default ProjectAssistantPage;

import React from 'react';  
import { ProjectAssistantPanel } from '../components/agents/ProjectAssistantPanel';  
  
const ProjectAssistantPage: React.FC = () => {  
 return (  
 <div className="project-assistant-page">  
 <h1>Project Assistant</h1>  
 <ProjectAssistantPanel />  
 </div>  
 );  
};  
  
export default ProjectAssistantPage;

#### Processing User Queries

import { createProjectAssistant } from '@kai/agents';  
  
// Create the Project Assistant  
const projectAssistant = await createProjectAssistant(  
 { id: 'project-assistant-1' },  
 { model: 'gpt-4', temperature: 0.3 }  
);  
  
// Ask questions about project planning  
const query1 = 'How much porcelain tile do I need for a 15x20 foot kitchen with 8-inch tiles?';  
const response1 = await projectAssistant.processUserInput(query1);  
console.log(response1);  
  
const query2 = 'I\'m working on a bathroom renovation with marble floor and porcelain wall tiles. What other materials should I consider?';  
const response2 = await projectAssistant.processUserInput(query2);  
console.log(response2);  
  
const query3 = 'What\'s the typical installation sequence for a kitchen backsplash project?';  
const response3 = await projectAssistant.processUserInput(query3);  
console.log(response3);

import { createProjectAssistant } from '@kai/agents';  
  
// Create the Project Assistant  
const projectAssistant = await createProjectAssistant(  
 { id: 'project-assistant-1' },  
 { model: 'gpt-4', temperature: 0.3 }  
);  
  
// Ask questions about project planning  
const query1 = 'How much porcelain tile do I need for a 15x20 foot kitchen with 8-inch tiles?';  
const response1 = await projectAssistant.processUserInput(query1);  
console.log(response1);  
  
const query2 = 'I\'m working on a bathroom renovation with marble floor and porcelain wall tiles. What other materials should I consider?';  
const response2 = await projectAssistant.processUserInput(query2);  
console.log(response2);  
  
const query3 = 'What\'s the typical installation sequence for a kitchen backsplash project?';  
const response3 = await projectAssistant.processUserInput(query3);  
console.log(response3);

### Advanced Configuration

#### Custom Project Calculator Tool

Create custom tools to enhance the Project Assistant's capabilities:

import { Tool } from 'crewai';  
  
// Create a specialized project calculator tool  
const createProjectCalculatorTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'project\_calculator',  
 description: 'Calculate material quantities, costs, and timelines for projects',  
 func: async (args) => {  
 const { project, dimensions, materialIds } = JSON.parse(args);  
  
 // Implement project calculations  
 const calculations = await calculateProjectRequirements(  
 project,  
 dimensions,  
 materialIds  
 );  
  
 return JSON.stringify({  
 quantities: calculations.quantities,  
 costs: calculations.costs,  
 timeline: calculations.timeline,  
 labor: calculations.laborEstimate  
 });  
 }  
 });  
};  
  
// Add it to the agent  
const projectAssistant = await createProjectAssistant(  
 {   
 id: 'advanced-project-assistant-1',  
 additionalTools: [await createProjectCalculatorTool()]  
 },  
 { model: 'gpt-4', temperature: 0.3 }  
);

import { Tool } from 'crewai';  
  
// Create a specialized project calculator tool  
const createProjectCalculatorTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'project\_calculator',  
 description: 'Calculate material quantities, costs, and timelines for projects',  
 func: async (args) => {  
 const { project, dimensions, materialIds } = JSON.parse(args);  
  
 // Implement project calculations  
 const calculations = await calculateProjectRequirements(  
 project,  
 dimensions,  
 materialIds  
 );  
  
 return JSON.stringify({  
 quantities: calculations.quantities,  
 costs: calculations.costs,  
 timeline: calculations.timeline,  
 labor: calculations.laborEstimate  
 });  
 }  
 });  
};  
  
// Add it to the agent  
const projectAssistant = await createProjectAssistant(  
 {   
 id: 'advanced-project-assistant-1',  
 additionalTools: [await createProjectCalculatorTool()]  
 },  
 { model: 'gpt-4', temperature: 0.3 }  
);

#### Integration with Professional Standards

Connect the Project Assistant to industry standards databases:

import { Tool } from 'crewai';  
  
// Create a tool for accessing industry installation standards  
const createInstallationStandardsTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'installation\_standards',  
 description: 'Access industry standards for material installation methods',  
 func: async (args) => {  
 const { materialType, installationType } = JSON.parse(args);  
  
 // Implement standards retrieval  
 const standards = await getIndustryStandards(materialType, installationType);  
  
 return JSON.stringify({  
 standardName: standards.name,  
 organization: standards.issuingBody,  
 requirements: standards.requirements,  
 bestPractices: standards.bestPractices,  
 certificationNeeded: standards.requiresCertification  
 });  
 }  
 });  
};  
  
// Add it to the agent  
const projectAssistant = await createProjectAssistant(  
 {   
 id: 'professional-project-assistant-1',  
 additionalTools: [await createInstallationStandardsTool()]  
 },  
 { model: 'gpt-4', temperature: 0.3 }  
);

import { Tool } from 'crewai';  
  
// Create a tool for accessing industry installation standards  
const createInstallationStandardsTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'installation\_standards',  
 description: 'Access industry standards for material installation methods',  
 func: async (args) => {  
 const { materialType, installationType } = JSON.parse(args);  
  
 // Implement standards retrieval  
 const standards = await getIndustryStandards(materialType, installationType);  
  
 return JSON.stringify({  
 standardName: standards.name,  
 organization: standards.issuingBody,  
 requirements: standards.requirements,  
 bestPractices: standards.bestPractices,  
 certificationNeeded: standards.requiresCertification  
 });  
 }  
 });  
};  
  
// Add it to the agent  
const projectAssistant = await createProjectAssistant(  
 {   
 id: 'professional-project-assistant-1',  
 additionalTools: [await createInstallationStandardsTool()]  
 },  
 { model: 'gpt-4', temperature: 0.3 }  
);

### Performance Considerations

#### Calculation Optimization

1. Caching Strategy
2. Cache common calculation formulas and results
3. Implement TTL-based cache invalidation for price updates
4. Pre-compute standard room layouts for quick estimates
5. Query Optimization
6. Use indexed material properties for faster lookup
7. Batch related calculations for efficiency
8. Optimize algorithm selection based on query complexity
9. Response Construction
10. Use standardized templates for common project plans
11. Progressive loading for detailed project information
12. Reuse calculation results across related queries

Pre-compute standard room layouts for quick estimates

Query Optimization

Optimize algorithm selection based on query complexity

Response Construction

### Security Considerations

1. Data Access Control
2. Enforce appropriate permissions for pricing information
3. Implement proper user authentication for saved projects
4. Limit access to proprietary application techniques
5. Information Accuracy
6. Verify material installation requirements against reliable sources
7. Provide disclaimers for cost and timeline estimates
8. Implement fact-checking mechanisms for technical recommendations
9. Agent Boundaries
10. Restrict the agent to project planning operations
11. Validate inputs to prevent injection attacks
12. Avoid providing professional advice that requires certification

Limit access to proprietary application techniques

Information Accuracy

Implement fact-checking mechanisms for technical recommendations

Agent Boundaries

### Related Documentation

* Material Expert - Companion agent for material information
* Knowledge Base - Material database structure and management
* CrewAI Integration - Overall agent system architecture
* CrewAI Implementation - Implementation details
* Agent Installation - Setup instructions

# Prompt Abtesting Segmentation

Source: readme/prompt-abtesting-segmentation.md

---

## Prompt A/B Testing and User Segmentation

This document describes the A/B testing and user segmentation features for prompt success tracking, which help optimize prompts for different user segments and compare different prompt variations.

### Overview

The A/B testing and user segmentation system consists of several components:

1. Database Schema: Tables for storing A/B test experiments, variants, and user segments
2. Backend Services: APIs for managing experiments and segments
3. Client-Side Integration: Tools for assigning users to experiments and segments
4. Admin UI: Dashboard for creating and monitoring experiments and segments
5. Analytics: Tools for analyzing experiment results and segment performance

### A/B Testing

#### Concepts

* Experiment: A test comparing multiple variants of a prompt
* Variant: A specific version of a prompt being tested
* Control: The baseline variant to compare against
* Traffic Allocation: Percentage of users included in the experiment
* Assignment: Assignment of a user to a specific variant

#### Creating an Experiment

To create an A/B test experiment:

1. Define the experiment name, description, and traffic allocation
2. Create at least two variants (one control and one or more test variants)
3. Set the weight for each variant (determines the probability of assignment)
4. Activate the experiment

Example:

const experimentId = await promptService.createABExperiment({  
 name: 'Product Description Prompt Test',  
 description: 'Testing different prompt formats for product descriptions',  
 trafficAllocation: 50, // 50% of users will be included  
 isActive: true,  
 variants: [  
 {  
 promptId: 'original-prompt-id',  
 variantName: 'Control',  
 isControl: true,  
 weight: 1  
 },  
 {  
 promptId: 'new-prompt-id',  
 variantName: 'Test Variant',  
 isControl: false,  
 weight: 1  
 }  
 ]  
});

const experimentId = await promptService.createABExperiment({  
 name: 'Product Description Prompt Test',  
 description: 'Testing different prompt formats for product descriptions',  
 trafficAllocation: 50, // 50% of users will be included  
 isActive: true,  
 variants: [  
 {  
 promptId: 'original-prompt-id',  
 variantName: 'Control',  
 isControl: true,  
 weight: 1  
 },  
 {  
 promptId: 'new-prompt-id',  
 variantName: 'Test Variant',  
 isControl: false,  
 weight: 1  
 }  
 ]  
});

#### Using A/B Testing in Your Application

To use A/B testing in your application:

// Render a prompt with A/B testing  
const { content, variantId, experimentId } = await promptService.renderPrompt({  
 promptId: 'original-prompt-id', // Will be replaced with variant if user is in experiment  
 data: { /\* your data \*/ },  
 userId: 'user-123',  
 sessionId: 'session-456',  
 abTestingEnabled: true  
});  
  
// Later, track success  
await promptService.updatePromptTrackingRecord(trackingId, {  
 isSuccessful: true,  
 // Other feedback data  
});

// Render a prompt with A/B testing  
const { content, variantId, experimentId } = await promptService.renderPrompt({  
 promptId: 'original-prompt-id', // Will be replaced with variant if user is in experiment  
 data: { /\* your data \*/ },  
 userId: 'user-123',  
 sessionId: 'session-456',  
 abTestingEnabled: true  
});  
  
// Later, track success  
await promptService.updatePromptTrackingRecord(trackingId, {  
 isSuccessful: true,  
 // Other feedback data  
});

#### Analyzing Experiment Results

To analyze experiment results:

const results = await promptService.getExperimentResults(  
 experimentId,  
 startDate,  
 endDate  
);  
  
// Compare success rates  
const controlVariant = results.find(r => r.isControl);  
const testVariants = results.filter(r => !r.isControl);  
  
for (const variant of testVariants) {  
 const improvement = variant.successRate - controlVariant.successRate;  
 console.log(`${variant.variantName}: ${improvement.toFixed(2)}% improvement`);  
}

const results = await promptService.getExperimentResults(  
 experimentId,  
 startDate,  
 endDate  
);  
  
// Compare success rates  
const controlVariant = results.find(r => r.isControl);  
const testVariants = results.filter(r => !r.isControl);  
  
for (const variant of testVariants) {  
 const improvement = variant.successRate - controlVariant.successRate;  
 console.log(`${variant.variantName}: ${improvement.toFixed(2)}% improvement`);  
}

### User Segmentation

#### Concepts

* Segment: A group of users with similar characteristics
* Segment Type: The type of segmentation (demographic, behavioral, etc.)
* Segment Criteria: Rules for assigning users to segments
* Segment Assignment: Assignment of a user to a specific segment

#### Creating a Segment

To create a user segment:

1. Define the segment name, description, and type
2. Define the segment criteria
3. Activate the segment

Example:

const segmentId = await promptService.createUserSegment({  
 name: 'Power Users',  
 description: 'Users who use the application frequently',  
 segmentType: 'behavioral',  
 segmentCriteria: {  
 usageFrequency: 'high',  
 minSessionsPerWeek: 5  
 },  
 isActive: true  
});

const segmentId = await promptService.createUserSegment({  
 name: 'Power Users',  
 description: 'Users who use the application frequently',  
 segmentType: 'behavioral',  
 segmentCriteria: {  
 usageFrequency: 'high',  
 minSessionsPerWeek: 5  
 },  
 isActive: true  
});

#### Using Segmentation in Your Application

To use segmentation in your application:

// Render a prompt with segmentation  
const { content } = await promptService.renderPrompt({  
 promptId: 'prompt-id',  
 data: { /\* your data \*/ },  
 userId: 'user-123',  
 sessionId: 'session-456'  
 // Segment will be automatically detected  
});  
  
// Or specify a segment explicitly  
const { content } = await promptService.renderPrompt({  
 promptId: 'prompt-id',  
 data: { /\* your data \*/ },  
 segmentId: 'segment-id'  
});

// Render a prompt with segmentation  
const { content } = await promptService.renderPrompt({  
 promptId: 'prompt-id',  
 data: { /\* your data \*/ },  
 userId: 'user-123',  
 sessionId: 'session-456'  
 // Segment will be automatically detected  
});  
  
// Or specify a segment explicitly  
const { content } = await promptService.renderPrompt({  
 promptId: 'prompt-id',  
 data: { /\* your data \*/ },  
 segmentId: 'segment-id'  
});

#### Analyzing Segment Performance

To analyze segment performance:

const analytics = await promptService.getPromptUsageAnalytics(  
 promptId,  
 startDate,  
 endDate,  
 segmentId  
);  
  
// Calculate success rate  
const totalUses = analytics.reduce((sum, a) => sum + a.totalUses, 0);  
const successfulUses = analytics.reduce((sum, a) => sum + a.successfulUses, 0);  
const successRate = totalUses > 0 ? (successfulUses / totalUses) \* 100 : 0;  
  
console.log(`Success rate for segment: ${successRate.toFixed(2)}%`);

const analytics = await promptService.getPromptUsageAnalytics(  
 promptId,  
 startDate,  
 endDate,  
 segmentId  
);  
  
// Calculate success rate  
const totalUses = analytics.reduce((sum, a) => sum + a.totalUses, 0);  
const successfulUses = analytics.reduce((sum, a) => sum + a.successfulUses, 0);  
const successRate = totalUses > 0 ? (successfulUses / totalUses) \* 100 : 0;  
  
console.log(`Success rate for segment: ${successRate.toFixed(2)}%`);

#### Comparing Segments

To compare multiple segments:

const results = await promptService.compareSegments(  
 [segmentId1, segmentId2],  
 promptId,  
 startDate,  
 endDate  
);  
  
// Compare success rates  
for (const result of results) {  
 console.log(`${result.segmentName}: ${result.successRate.toFixed(2)}%`);  
}

const results = await promptService.compareSegments(  
 [segmentId1, segmentId2],  
 promptId,  
 startDate,  
 endDate  
);  
  
// Compare success rates  
for (const result of results) {  
 console.log(`${result.segmentName}: ${result.successRate.toFixed(2)}%`);  
}

### Enhanced Auto-Detection

The enhanced auto-detection system uses sophisticated behavior analysis to determine whether a prompt was successful.

#### Tracked Behaviors

The system tracks the following user behaviors:

* Basic Metrics:
* Time spent on page
* Scroll depth
* Clicked links
* Copied text
* Follow-up questions
* Enhanced Metrics:
* Interaction count
* Interaction duration
* Interaction patterns
* Follow-up sentiment
* Keyboard and mouse activity
* Page focus/blur events
* Visibility changes

Follow-up questions

Enhanced Metrics:

#### Interaction Patterns

The system tracks specific interaction patterns that indicate success or failure:

* Positive Patterns:
* Highlighting text
* Saving the response
* Sharing the response
* Expanding details
* Negative Patterns:
* Closing immediately
* Retrying the query
* Reporting an issue
* Requesting an alternative

Expanding details

Negative Patterns:

#### Sentiment Analysis

The system performs simple sentiment analysis on follow-up questions to determine user satisfaction:

// Simple sentiment analysis  
const positiveWords = ['thanks', 'good', 'great', 'helpful', 'useful'];  
const negativeWords = ['not', 'wrong', 'bad', 'unclear', 'confusing'];  
  
// Calculate sentiment score between -1 and 1  
const sentimentScore = (positiveCount - negativeCount) / total;

// Simple sentiment analysis  
const positiveWords = ['thanks', 'good', 'great', 'helpful', 'useful'];  
const negativeWords = ['not', 'wrong', 'bad', 'unclear', 'confusing'];  
  
// Calculate sentiment score between -1 and 1  
const sentimentScore = (positiveCount - negativeCount) / total;

#### Using Enhanced Auto-Detection

To use enhanced auto-detection in your application:

// Create a success tracker  
const tracker = createPromptSuccessTracker(trackingId, responseTimeMs);  
  
// Record a follow-up question with text  
tracker.recordFollowupQuestion('Thanks, that was very helpful!');  
  
// Auto-detect success  
await tracker.autoDetectSuccess();

// Create a success tracker  
const tracker = createPromptSuccessTracker(trackingId, responseTimeMs);  
  
// Record a follow-up question with text  
tracker.recordFollowupQuestion('Thanks, that was very helpful!');  
  
// Auto-detect success  
await tracker.autoDetectSuccess();

### Admin UI

The admin UI provides tools for managing A/B tests and user segments:

#### A/B Testing Dashboard

* Create and manage experiments
* View experiment results
* Compare variant performance
* End experiments and promote winning variants

#### Segmentation Dashboard

* Create and manage user segments
* View segment analytics
* Compare segment performance
* Optimize prompts for specific segments

### Best Practices

1. Start with Clear Hypotheses: Define clear hypotheses for A/B tests
2. Use Meaningful Segments: Create segments that represent distinct user groups
3. Test One Variable at a Time: Change only one aspect of the prompt in each variant
4. Run Tests Long Enough: Allow tests to run until statistical significance is achieved
5. Monitor Segment Performance: Regularly check how prompts perform across segments
6. Iterate Based on Results: Use test results to improve prompts over time
7. Combine A/B Testing with Segmentation: Test different variants for different segments

### Troubleshooting

* No Variant Assignment: Check that the experiment is active and traffic allocation is sufficient
* Segment Not Detected: Verify that segment criteria are correctly defined
* Low Statistical Significance: Increase test duration or traffic allocation
* Inconsistent Results: Check for external factors affecting test results

### Conclusion

The A/B testing and user segmentation system provides powerful tools for optimizing prompts for different user groups and comparing different prompt variations. By combining these features with enhanced auto-detection, you can continuously improve the quality and effectiveness of your AI interactions.

# Prompt Advanced Features

Source: readme/prompt-advanced-features.md

---

## Advanced Prompt Features

This document describes the advanced features for prompt optimization, including machine learning, statistical analysis, automated optimization, and integration with external systems.

### Overview

The advanced prompt features consist of several components:

1. Machine Learning: Predicts prompt success and suggests improvements
2. Statistical Analysis: Provides statistical significance testing for A/B tests
3. Automated Optimization: Automatically optimizes prompts based on rules
4. External System Integration: Connects with monitoring and analytics systems
5. Advanced Segmentation: Provides sophisticated user segmentation capabilities

### Machine Learning

#### ML Models

The system uses machine learning models to predict prompt success and suggest improvements. The models are trained on historical prompt usage data and can be used to:

1. Predict the success rate of a prompt before it's used
2. Generate improvement suggestions for existing prompts
3. Automatically create variants for A/B testing

##### Supported Model Types

* Neural Network: Standard feedforward neural network for general-purpose prediction
* LSTM: Long Short-Term Memory networks for sequence-based analysis
* Transformer: Transformer-based models for complex pattern recognition
* Random Forest: Tree-based ensemble method for robust classification
* Gradient Boosting: Boosting-based ensemble method for high-accuracy prediction

#### Feature Extraction

The ML system extracts features from prompts, including:

* Basic metrics (length, word count, question count, etc.)
* Type-specific features (material terms, agent terms, etc.)
* Structural features (sections, formatting, etc.)
* Clarity metrics (instruction clarity, context richness, etc.)
* Readability metrics (Flesch-Kincaid score, Gunning Fog Index, etc.)
* Semantic features (examples, definitions, conditionals, instructions)
* Visual descriptors (for material-specific prompts)
* Goal clarity and constraint specificity (for agent prompts)
* Search specificity and contextual constraints (for RAG prompts)

#### Prediction and Suggestions

To use ML predictions and suggestions:

// Predict prompt success  
const prediction = await mlService.predictPromptSuccess(  
 userId,  
 promptId,  
 promptContent,  
 promptType  
);  
  
// Generate improvement suggestions  
const suggestions = await mlService.generateImprovementSuggestions(  
 userId,  
 promptId,  
 promptContent,  
 promptType  
);  
  
// Apply a suggestion  
const updatedContent = await mlService.applyImprovementSuggestion(userId, suggestionId);

// Predict prompt success  
const prediction = await mlService.predictPromptSuccess(  
 userId,  
 promptId,  
 promptContent,  
 promptType  
);  
  
// Generate improvement suggestions  
const suggestions = await mlService.generateImprovementSuggestions(  
 userId,  
 promptId,  
 promptContent,  
 promptType  
);  
  
// Apply a suggestion  
const updatedContent = await mlService.applyImprovementSuggestion(userId, suggestionId);

#### Training Models

To train a new ML model:

// Create a neural network model  
const nnModelId = await mlService.createMLModel({  
 name: 'Prompt Success Predictor',  
 modelType: 'neural\_network',  
 modelParameters: {  
 inputDimension: 10,  
 hiddenLayers: [64, 32],  
 activation: 'relu',  
 outputActivation: 'sigmoid',  
 optimizer: 'adam',  
 loss: 'binaryCrossentropy',  
 epochs: 100,  
 batchSize: 32,  
 validationSplit: 0.2,  
 dropoutRate: 0.2,  
 useLearningRateScheduler: true,  
 calculateFeatureImportance: true  
 },  
 trainingDataQuery: 'SELECT \* FROM prompt\_usage\_analytics',  
 isActive: true  
});  
  
// Create an LSTM model  
const lstmModelId = await mlService.createMLModel({  
 name: 'Sequence-Based Predictor',  
 modelType: 'lstm',  
 modelParameters: {  
 sequenceLength: 10,  
 inputDimension: 10,  
 lstmUnits: [64, 32],  
 activation: 'tanh',  
 recurrentActivation: 'hardSigmoid',  
 outputActivation: 'sigmoid',  
 dropoutRate: 0.2,  
 recurrentDropoutRate: 0.2  
 },  
 trainingDataQuery: 'SELECT \* FROM prompt\_usage\_analytics',  
 isActive: false  
});  
  
// Create a Random Forest model  
const rfModelId = await mlService.createMLModel({  
 name: 'Robust Classifier',  
 modelType: 'random\_forest',  
 modelParameters: {  
 nEstimators: 100,  
 maxDepth: 10,  
 minSamplesSplit: 2,  
 maxFeatures: 'sqrt',  
 gainFunction: 'gini'  
 },  
 trainingDataQuery: 'SELECT \* FROM prompt\_usage\_analytics',  
 isActive: false  
});  
  
// Train the model  
const modelVersionId = await mlService.trainModel(userId, nnModelId);  
  
// Apply transfer learning  
const transferModelId = await mlService.createMLModel({  
 name: 'Transfer Learning Model',  
 modelType: 'neural\_network',  
 modelParameters: {  
 transferLearning: true,  
 baseModelId: nnModelId,  
 finetuningEpochs: 20  
 },  
 isActive: false  
});

// Create a neural network model  
const nnModelId = await mlService.createMLModel({  
 name: 'Prompt Success Predictor',  
 modelType: 'neural\_network',  
 modelParameters: {  
 inputDimension: 10,  
 hiddenLayers: [64, 32],  
 activation: 'relu',  
 outputActivation: 'sigmoid',  
 optimizer: 'adam',  
 loss: 'binaryCrossentropy',  
 epochs: 100,  
 batchSize: 32,  
 validationSplit: 0.2,  
 dropoutRate: 0.2,  
 useLearningRateScheduler: true,  
 calculateFeatureImportance: true  
 },  
 trainingDataQuery: 'SELECT \* FROM prompt\_usage\_analytics',  
 isActive: true  
});  
  
// Create an LSTM model  
const lstmModelId = await mlService.createMLModel({  
 name: 'Sequence-Based Predictor',  
 modelType: 'lstm',  
 modelParameters: {  
 sequenceLength: 10,  
 inputDimension: 10,  
 lstmUnits: [64, 32],  
 activation: 'tanh',  
 recurrentActivation: 'hardSigmoid',  
 outputActivation: 'sigmoid',  
 dropoutRate: 0.2,  
 recurrentDropoutRate: 0.2  
 },  
 trainingDataQuery: 'SELECT \* FROM prompt\_usage\_analytics',  
 isActive: false  
});  
  
// Create a Random Forest model  
const rfModelId = await mlService.createMLModel({  
 name: 'Robust Classifier',  
 modelType: 'random\_forest',  
 modelParameters: {  
 nEstimators: 100,  
 maxDepth: 10,  
 minSamplesSplit: 2,  
 maxFeatures: 'sqrt',  
 gainFunction: 'gini'  
 },  
 trainingDataQuery: 'SELECT \* FROM prompt\_usage\_analytics',  
 isActive: false  
});  
  
// Train the model  
const modelVersionId = await mlService.trainModel(userId, nnModelId);  
  
// Apply transfer learning  
const transferModelId = await mlService.createMLModel({  
 name: 'Transfer Learning Model',  
 modelType: 'neural\_network',  
 modelParameters: {  
 transferLearning: true,  
 baseModelId: nnModelId,  
 finetuningEpochs: 20  
 },  
 isActive: false  
});

### Statistical Analysis

#### Significance Testing

The statistical analysis system provides significance testing for A/B tests and segment comparisons, including:

1. Z-tests for proportions
2. Chi-square tests for independence
3. Confidence intervals
4. P-values and significance determination
5. Effect size calculations
6. Power analysis for sample size determination

#### Correlation Analysis

The system can analyze correlations between various factors:

1. Correlation coefficients (Pearson, Spearman)
2. Statistical significance of correlations
3. Visualization of correlation matrices
4. Factor relationship mapping

#### Trend Analysis

The system provides trend analysis capabilities:

1. Time series analysis with trend detection
2. Seasonality identification
3. Anomaly detection
4. Forecasting with confidence intervals

#### Analyzing Experiments

To analyze an experiment:

// Analyze an experiment  
const results = await statisticalService.analyzeExperiment(  
 experimentId,  
 startDate,  
 endDate  
);  
  
// Check if results are significant  
const isSignificant = results[0].isSignificant;  
const pValue = results[0].pValue;  
const confidenceInterval = [  
 results[0].confidenceIntervalLower,  
 results[0].confidenceIntervalUpper  
];

// Analyze an experiment  
const results = await statisticalService.analyzeExperiment(  
 experimentId,  
 startDate,  
 endDate  
);  
  
// Check if results are significant  
const isSignificant = results[0].isSignificant;  
const pValue = results[0].pValue;  
const confidenceInterval = [  
 results[0].confidenceIntervalLower,  
 results[0].confidenceIntervalUpper  
];

#### Comparing Segments

To compare segments:

// Compare segments  
const results = await statisticalService.compareSegments(  
 [segment1Id, segment2Id],  
 promptId,  
 startDate,  
 endDate  
);

// Compare segments  
const results = await statisticalService.compareSegments(  
 [segment1Id, segment2Id],  
 promptId,  
 startDate,  
 endDate  
);

### Automated Optimization

#### Optimization Rules

The system supports various optimization rules:

1. Low Success Rate: Automatically creates experiments for prompts with low success rates
2. Champion/Challenger: Promotes winning variants and ends experiments
3. Segment-Specific: Creates segment-specific prompts based on performance
4. ML Suggestion: Applies ML suggestions to improve prompts
5. Scheduled Experiment: Creates experiments on a schedule
6. Time-Based: Activates different prompts based on time of day or day of week
7. User Feedback: Optimizes prompts based on user feedback metrics
8. Context-Aware: Adapts prompts based on contextual factors
9. Multi-Variant: Tests multiple variants simultaneously

#### Creating Rules

To create an optimization rule:

// Create a rule  
const ruleId = await optimizationService.createOptimizationRule({  
 name: 'Low Success Rate Detector',  
 ruleType: 'low\_success\_rate',  
 ruleParameters: {  
 threshold: 50,  
 lookbackDays: 7  
 },  
 isActive: true  
});

// Create a rule  
const ruleId = await optimizationService.createOptimizationRule({  
 name: 'Low Success Rate Detector',  
 ruleType: 'low\_success\_rate',  
 ruleParameters: {  
 threshold: 50,  
 lookbackDays: 7  
 },  
 isActive: true  
});

#### Executing Rules

To execute optimization rules:

// Execute all active rules  
const actionsCreated = await optimizationService.executeOptimizationRules();  
  
// Execute pending actions  
const actionsExecuted = await optimizationService.executePendingActions();

// Execute all active rules  
const actionsCreated = await optimizationService.executeOptimizationRules();  
  
// Execute pending actions  
const actionsExecuted = await optimizationService.executePendingActions();

### External System Integration

#### Supported Systems

The system can integrate with various external systems:

1. Grafana: For visualization and dashboards
2. Prometheus: For metrics and monitoring
3. Datadog: For application performance monitoring
4. Elasticsearch: For log analysis and search
5. Custom API: For integration with custom systems
6. Google Analytics: For user behavior tracking
7. Slack: For notifications and alerts
8. Power BI: For business intelligence reporting
9. Webhook: For general-purpose integration

#### Creating Integrations

To create an integration:

// Create a Grafana integration  
const integrationId = await integrationService.createIntegration({  
 name: 'Grafana Dashboard',  
 systemType: 'grafana',  
 connectionParameters: {  
 url: 'https://grafana.example.com',  
 apiKey: 'your-api-key',  
 dashboardUid: 'your-dashboard-uid'  
 },  
 isActive: true  
});  
  
// Test the connection  
const result = await integrationService.testIntegrationConnection(integrationId);

// Create a Grafana integration  
const integrationId = await integrationService.createIntegration({  
 name: 'Grafana Dashboard',  
 systemType: 'grafana',  
 connectionParameters: {  
 url: 'https://grafana.example.com',  
 apiKey: 'your-api-key',  
 dashboardUid: 'your-dashboard-uid'  
 },  
 isActive: true  
});  
  
// Test the connection  
const result = await integrationService.testIntegrationConnection(integrationId);

#### Exporting Data

To export data to external systems:

// Create a data export  
const exportId = await integrationService.createDataExport({  
 integrationId,  
 exportType: 'success\_metrics',  
 exportParameters: {  
 startDate: '2023-01-01',  
 endDate: '2023-01-31',  
 promptIds: ['prompt-1', 'prompt-2']  
 }  
});  
  
// Execute pending exports  
const exportsExecuted = await integrationService.executePendingExports();

// Create a data export  
const exportId = await integrationService.createDataExport({  
 integrationId,  
 exportType: 'success\_metrics',  
 exportParameters: {  
 startDate: '2023-01-01',  
 endDate: '2023-01-31',  
 promptIds: ['prompt-1', 'prompt-2']  
 }  
});  
  
// Execute pending exports  
const exportsExecuted = await integrationService.executePendingExports();

### Advanced Segmentation

#### Segmentation Types

The system supports advanced segmentation types:

1. Behavioral: Based on user behavior (usage frequency, interaction patterns, etc.)
2. Demographic: Based on user demographics (age, location, etc.)
3. Contextual: Based on context (device, time, etc.)
4. Discovered: Automatically discovered segments based on patterns

#### Creating Advanced Segments

To create an advanced segment:

// Create a behavioral segment  
const segmentId = await promptService.createUserSegment({  
 name: 'Power Users',  
 segmentType: 'behavioral',  
 segmentCriteria: {  
 usageFrequency: 'high',  
 minSessionsPerWeek: 5  
 },  
 behavioralCriteria: {  
 interactionPattern: ['search', 'view\_details', 'save'],  
 minInteractionCount: 10  
 },  
 isActive: true  
});

// Create a behavioral segment  
const segmentId = await promptService.createUserSegment({  
 name: 'Power Users',  
 segmentType: 'behavioral',  
 segmentCriteria: {  
 usageFrequency: 'high',  
 minSessionsPerWeek: 5  
 },  
 behavioralCriteria: {  
 interactionPattern: ['search', 'view\_details', 'save'],  
 minInteractionCount: 10  
 },  
 isActive: true  
});

#### Segment Discovery

The system can automatically discover segments based on patterns:

// Discover segments  
const segments = await promptService.discoverSegments({  
 promptId,  
 minSegmentSize: 100,  
 maxSegments: 5,  
 discoveryMethod: 'clustering',  
 discoveryParameters: {  
 algorithm: 'kmeans',  
 features: ['usage\_frequency', 'success\_rate', 'interaction\_count']  
 }  
});

// Discover segments  
const segments = await promptService.discoverSegments({  
 promptId,  
 minSegmentSize: 100,  
 maxSegments: 5,  
 discoveryMethod: 'clustering',  
 discoveryParameters: {  
 algorithm: 'kmeans',  
 features: ['usage\_frequency', 'success\_rate', 'interaction\_count']  
 }  
});

##### Discovery Methods

The system supports multiple discovery methods:

1. Clustering: Groups users based on similarity using K-means, DBSCAN, or hierarchical clustering
2. Decision Tree: Identifies segments through decision tree splits based on key attributes
3. Association Rules: Discovers patterns of associated behaviors and attributes
4. Behavioral Patterns: Identifies common sequences of actions and interactions

### Admin UI

The admin UI provides tools for managing all advanced features:

#### Advanced Prompt Features Dashboard

The admin panel now includes a comprehensive Advanced Prompt Features dashboard with the following tabs:

##### ML Models Tab

* Create and manage ML models of various types using the MLModelForm component
* Train models with customizable parameters
* View model performance metrics and version history with the MLModelDetails component
* Compare models and analyze feature importance

MLModelForm

MLModelDetails

##### Predictions Tab

* Predict success rates for prompts before deployment
* Visualize feature importance and impact
* Get AI-generated improvement suggestions
* Apply suggestions with one click

##### Statistical Analysis Tab

* Analyze experiment results with statistical significance testing
* Compare segment performance with confidence intervals
* Discover correlations between factors affecting prompt success
* Analyze trends and forecast future performance

##### Optimization Tab

* Create and manage optimization rules of various types using the OptimizationRuleForm component
* View optimization actions and their results with the OptimizationRuleDetails component
* Execute rules manually or schedule automatic execution
* Monitor rule performance over time

OptimizationRuleForm

OptimizationRuleDetails

##### Integrations Tab

* Create and manage integrations with external systems
* Test connections and troubleshoot issues
* Create data exports with customizable parameters
* Schedule regular exports to external systems

##### Segment Discovery Tab

* Discover user segments automatically using ML techniques
* Visualize segment distribution and characteristics
* Compare segment performance and identify opportunities
* Save discovered segments for targeting

#### Admin UI Components

The admin UI includes the following key components:

##### MLModelForm

A form component for creating and editing ML models with support for:  
- Different model types (Neural Network, LSTM, Transformer, Random Forest, Gradient Boosting)  
- Customizable model parameters  
- Training data configuration  
- Model activation/deactivation

##### MLModelDetails

A component for viewing detailed information about ML models, including:  
- Performance metrics (accuracy, precision, recall, F1 score, AUC)  
- Feature importance visualization  
- Version history  
- Training history  
- Confusion matrix

##### OptimizationRuleForm

A form component for creating and editing optimization rules with support for:  
- Different rule types (Low Success Rate, Champion/Challenger, Segment Specific, etc.)  
- Customizable rule parameters  
- Rule activation/deactivation

##### OptimizationRuleDetails

A component for viewing detailed information about optimization rules, including:  
- Rule parameters  
- Actions generated by the rule  
- Performance metrics  
- Execution history

### API Endpoints

#### ML Endpoints

* GET /api/admin/prompt-ml/models: Get all ML models
* GET /api/admin/prompt-ml/models/:modelId: Get ML model by ID
* POST /api/admin/prompt-ml/models: Create ML model
* PATCH /api/admin/prompt-ml/models/:modelId: Update ML model
* POST /api/admin/prompt-ml/models/:modelId/train: Train ML model
* GET /api/admin/prompt-ml/models/:modelId/versions: Get model versions
* GET /api/admin/prompt-ml/models/:modelId/performance: Get model performance metrics
* GET /api/admin/prompt-ml/predict: Predict prompt success for new content
* GET /api/admin/prompt-ml/prompts/:promptId/predict: Predict prompt success
* GET /api/admin/prompt-ml/prompts/:promptId/suggestions: Generate improvement suggestions
* POST /api/admin/prompt-ml/suggestions/:suggestionId/apply: Apply improvement suggestion
* GET /api/admin/prompt-ml/feature-importance: Get feature importance analysis

GET /api/admin/prompt-ml/models

GET /api/admin/prompt-ml/models/:modelId

POST /api/admin/prompt-ml/models

PATCH /api/admin/prompt-ml/models/:modelId

POST /api/admin/prompt-ml/models/:modelId/train

GET /api/admin/prompt-ml/models/:modelId/versions

GET /api/admin/prompt-ml/models/:modelId/performance

GET /api/admin/prompt-ml/predict

GET /api/admin/prompt-ml/prompts/:promptId/predict

GET /api/admin/prompt-ml/prompts/:promptId/suggestions

POST /api/admin/prompt-ml/suggestions/:suggestionId/apply

GET /api/admin/prompt-ml/feature-importance

#### Statistical Endpoints

* GET /api/admin/prompt-statistical: Get statistical analyses
* POST /api/admin/prompt-statistical/experiments/:experimentId/analyze: Analyze experiment
* POST /api/admin/prompt-statistical/segments/compare: Compare segments
* GET /api/admin/prompt-statistical/correlations: Analyze correlations
* GET /api/admin/prompt-statistical/trends: Analyze trends
* GET /api/admin/prompt-statistical/power-analysis: Calculate required sample size

GET /api/admin/prompt-statistical

POST /api/admin/prompt-statistical/experiments/:experimentId/analyze

POST /api/admin/prompt-statistical/segments/compare

GET /api/admin/prompt-statistical/correlations

GET /api/admin/prompt-statistical/trends

GET /api/admin/prompt-statistical/power-analysis

#### Optimization Endpoints

* GET /api/admin/prompt-optimization/rules: Get optimization rules
* POST /api/admin/prompt-optimization/rules: Create optimization rule
* PATCH /api/admin/prompt-optimization/rules/:ruleId: Update optimization rule
* GET /api/admin/prompt-optimization/rules/:ruleId: Get rule details
* POST /api/admin/prompt-optimization/rules/:ruleId/execute: Execute specific rule
* GET /api/admin/prompt-optimization/actions: Get optimization actions
* GET /api/admin/prompt-optimization/actions/:actionId: Get action details
* POST /api/admin/prompt-optimization/rules/execute: Execute all optimization rules
* POST /api/admin/prompt-optimization/actions/execute: Execute pending actions

GET /api/admin/prompt-optimization/rules

POST /api/admin/prompt-optimization/rules

PATCH /api/admin/prompt-optimization/rules/:ruleId

GET /api/admin/prompt-optimization/rules/:ruleId

POST /api/admin/prompt-optimization/rules/:ruleId/execute

GET /api/admin/prompt-optimization/actions

GET /api/admin/prompt-optimization/actions/:actionId

POST /api/admin/prompt-optimization/rules/execute

POST /api/admin/prompt-optimization/actions/execute

#### Integration Endpoints

* GET /api/admin/prompt-integration: Get integrations
* POST /api/admin/prompt-integration: Create integration
* PATCH /api/admin/prompt-integration/:integrationId: Update integration
* POST /api/admin/prompt-integration/:integrationId/test: Test integration connection
* GET /api/admin/prompt-integration/exports: Get data exports
* POST /api/admin/prompt-integration/exports: Create data export
* GET /api/admin/prompt-integration/exports/:exportId: Get export details
* POST /api/admin/prompt-integration/exports/execute: Execute pending exports

GET /api/admin/prompt-integration

POST /api/admin/prompt-integration

PATCH /api/admin/prompt-integration/:integrationId

POST /api/admin/prompt-integration/:integrationId/test

GET /api/admin/prompt-integration/exports

POST /api/admin/prompt-integration/exports

GET /api/admin/prompt-integration/exports/:exportId

POST /api/admin/prompt-integration/exports/execute

#### Segment Discovery Endpoints

* POST /api/admin/prompt-segmentation/discover: Discover segments
* GET /api/admin/prompt-segmentation/segments: Get saved segments
* POST /api/admin/prompt-segmentation/segments/:segmentId/save: Save discovered segment
* GET /api/admin/prompt-segmentation/segments/:segmentId: Get segment details
* GET /api/admin/prompt-segmentation/segments/:segmentId/performance: Get segment performance

POST /api/admin/prompt-segmentation/discover

GET /api/admin/prompt-segmentation/segments

POST /api/admin/prompt-segmentation/segments/:segmentId/save

GET /api/admin/prompt-segmentation/segments/:segmentId

GET /api/admin/prompt-segmentation/segments/:segmentId/performance

### Best Practices

1. Start with A/B Testing: Use A/B testing to establish baselines before implementing ML
2. Use Statistical Analysis: Always check statistical significance before making decisions
3. Combine Approaches: Use ML, statistical analysis, and optimization together
4. Monitor Performance: Use integrations to monitor performance over time
5. Iterate Gradually: Start with simple rules and gradually add complexity
6. Validate Suggestions: Review ML suggestions before applying them
7. Segment Appropriately: Use advanced segmentation to target specific user groups
8. Document Experiments: Keep track of experiments and their results
9. Leverage Multiple Model Types: Use different model types for different prediction tasks
10. Analyze Feature Importance: Understand which features drive success
11. Implement Transfer Learning: Use knowledge from one domain to improve another
12. Automate Routine Tasks: Use optimization rules for repetitive optimization tasks
13. Integrate with External Systems: Share data with other monitoring and analytics tools
14. Discover Hidden Segments: Use ML to find segments you might not have considered

### Troubleshooting

* ML Model Not Training: Check that you have sufficient training data and appropriate parameters
* Model Performance Issues: Try different model types or adjust hyperparameters
* Feature Importance Not Showing: Ensure calculateFeatureImportance is enabled in model parameters
* Statistical Analysis Not Significant: Increase sample size or run the experiment longer
* Correlation Analysis Showing Spurious Results: Check for confounding variables
* Trend Analysis Not Accurate: Ensure sufficient historical data and appropriate time granularity
* Optimization Rules Not Executing: Check that rules are active and conditions are met
* Rule Actions Failing: Review action logs and ensure necessary permissions
* Integration Not Working: Test the connection and check credentials
* Data Exports Failing: Verify export parameters and destination system availability
* Segments Not Matching Users: Review segment criteria and check for overlaps
* Segment Discovery Not Finding Patterns: Try different discovery methods or adjust parameters

### Conclusion

The advanced prompt features provide powerful tools for optimizing prompts using machine learning, statistical analysis, automated optimization, and external system integration. By combining these features, you can continuously improve the quality and effectiveness of your AI interactions.

# Prompt Management

Source: readme/prompt-management.md

---

## Prompt Management System

The KAI platform includes a comprehensive prompt management system that allows administrators to view, edit, and manage AI prompts used throughout the application. This document explains how to use the prompt management system and how it integrates with different parts of the application.

### Overview

The prompt management system provides a centralized way to manage all AI prompts used in the KAI platform, including:

* Material-specific prompts
* Agent system prompts
* RAG (Retrieval-Augmented Generation) prompts
* Generative enhancer prompts
* Hybrid retriever prompts

By centralizing prompt management, the system allows for:

1. Consistent prompt updates: Change prompts across the system without modifying code
2. A/B testing: Test different prompt variations to optimize performance
3. Role-based access control: Only administrators can modify prompts
4. Version tracking: Keep track of prompt changes over time
5. Dynamic prompt loading: Load prompts from the database at runtime

### Accessing the Prompt Management System

The prompt management system is available in the admin panel:

1. Log in to the admin panel
2. Navigate to "System Prompts" in the sidebar
3. The prompt management interface will be displayed

### Prompt Management Interface

The prompt management interface includes the following features:

#### Viewing Prompts

The main view displays a table of all prompts in the system, with columns for:

* Name
* Type
* Description
* Location (file where the prompt is used)
* Status (Active/Inactive)
* Actions (Edit, Duplicate, Delete, Version History, Success Rate)

You can filter prompts by type using the tabs at the top of the page:

* All Prompts
* Material Specific
* Agent
* RAG

You can also search for prompts using the search box in the top-right corner.

#### Creating Prompts

To create a new prompt:

1. Click the "Create Prompt" button
2. Fill in the prompt details:
3. Name: A unique identifier for the prompt
4. Description: A brief description of the prompt's purpose
5. Prompt Type: The category of prompt (Material Specific, Agent, RAG, etc.)
6. Location: The file path where the prompt is used
7. Variables: Comma-separated list of variables used in the prompt (e.g., {material\_type}, {query})
8. Status: Active or Inactive
9. Content: The actual prompt text
10. Click "Create" to save the prompt

#### Editing Prompts

To edit an existing prompt:

1. Click the Edit icon (pencil) next to the prompt in the table
2. Modify the prompt details as needed
3. Click "Save" to update the prompt

#### Duplicating Prompts

To duplicate a prompt (useful for creating variations):

1. Click the Duplicate icon (copy) next to the prompt in the table
2. The system will create a copy with "(Copy)" appended to the name
3. Edit the duplicate as needed
4. Click "Create" to save the new prompt

#### Deleting Prompts

To delete a prompt:

1. Click the Delete icon (trash) next to the prompt in the table
2. Confirm the deletion when prompted

Note: Deleting a prompt will cause the system to fall back to hardcoded defaults if available.

#### Managing Prompt Versions

The system automatically creates a new version of a prompt whenever its content is changed. This allows you to track changes over time and revert to previous versions if needed.

To view version history:

1. Click the History icon (clock) next to the prompt in the table
2. The system will display a list of all versions of the prompt
3. Each version shows:
4. Version number
5. Creation date
6. Status (Active/Inactive)
7. Success rate
8. Revert button

To revert to a previous version:

1. Click the Revert button next to the version you want to revert to
2. Confirm the reversion when prompted
3. The system will create a new version with the content of the selected version

#### Tracking Prompt Success Rates

The system tracks the success rate of prompts based on user feedback. This helps you identify which prompts are performing well and which need improvement.

To view success rate:

1. Click the Assessment icon (chart) next to the prompt in the table
2. The system will display the overall success rate for the prompt
3. You can also see success rates for individual versions in the version history

Success rates are calculated as the percentage of successful uses of the prompt out of the total number of uses.

### Prompt Types

The system supports several types of prompts:

#### Material Specific Prompts

These prompts are used for material-specific responses in the RAG system. They include:

* Base system prompts for different material types (wood, tile, stone, etc.)
* Detail level instructions (brief, medium, detailed)
* Evaluation criteria for different material types

#### Agent Prompts

These prompts are used by the agent system for different agent types:

* Default system messages for agents
* Agent-specific prompts for different agent roles
* Function calling prompts

#### RAG Prompts

These prompts are used by the RAG (Retrieval-Augmented Generation) system:

* Explanation prompts for generating material explanations
* Similarity prompts for comparing materials
* Application prompts for recommending material applications

#### Other Prompt Types

* Generative Enhancer: Used by the generative enhancer component
* Hybrid Retriever: Used by the hybrid retriever component
* Other: Miscellaneous prompts used elsewhere in the system

### Variables in Prompts

Prompts can include variables that are replaced at runtime. Variables are enclosed in curly braces, e.g., {material\_type}.

{material\_type}

Common variables include:

* {material\_type}: The type of material (wood, tile, stone, etc.)
* {query}: The user's query
* {context\_text}: The context text with retrieved materials and knowledge
* {detail\_instructions}: Instructions for the detail level

{material\_type}

{query}

{context\_text}

{detail\_instructions}

When editing prompts, you can specify the variables used in the prompt in the "Variables" field.

### Integration with the Application

The prompt management system integrates with different parts of the application:

#### Material-Specific Prompts

The material\_specific\_prompts\_db.py file in the ML package fetches prompts from the database:

material\_specific\_prompts\_db.py

def get\_material\_system\_prompt(material\_type: str) -> str:  
 """  
 Get the system prompt for a specific material type  
  
 Args:  
 material\_type: Type of material  
  
 Returns:  
 System prompt for the specified material type  
 """  
 material\_type = material\_type.lower()  
  
 # Try to get material-specific prompt from database  
 prompt\_name = f"{material\_type.upper()}\_PROMPT"  
 db\_prompt = fetch\_prompt\_from\_db(prompt\_name, 'material\_specific')  
  
 if db\_prompt:  
 return db\_prompt  
  
 # If not found, fall back to default prompt  
 # ...

def get\_material\_system\_prompt(material\_type: str) -> str:  
 """  
 Get the system prompt for a specific material type  
  
 Args:  
 material\_type: Type of material  
  
 Returns:  
 System prompt for the specified material type  
 """  
 material\_type = material\_type.lower()  
  
 # Try to get material-specific prompt from database  
 prompt\_name = f"{material\_type.upper()}\_PROMPT"  
 db\_prompt = fetch\_prompt\_from\_db(prompt\_name, 'material\_specific')  
  
 if db\_prompt:  
 return db\_prompt  
  
 # If not found, fall back to default prompt  
 # ...

#### Agent System Prompts

The llmInferenceHelperWithPrompts.ts file in the Agents package fetches prompts from the database:

llmInferenceHelperWithPrompts.ts

/\*\*  
 \* Get the system prompt for agents  
 \* @returns The system prompt  
 \*/  
export async function getSystemPrompt(): Promise<string> {  
 try {  
 const prompt = await fetchPromptFromService('DEFAULT\_SYSTEM\_MESSAGE', PromptType.AGENT);  
 return prompt || FALLBACK\_SYSTEM\_MESSAGE;  
 } catch (error) {  
 logger.error(`Error getting system prompt: ${error}`);  
 return FALLBACK\_SYSTEM\_MESSAGE;  
 }  
}

/\*\*  
 \* Get the system prompt for agents  
 \* @returns The system prompt  
 \*/  
export async function getSystemPrompt(): Promise<string> {  
 try {  
 const prompt = await fetchPromptFromService('DEFAULT\_SYSTEM\_MESSAGE', PromptType.AGENT);  
 return prompt || FALLBACK\_SYSTEM\_MESSAGE;  
 } catch (error) {  
 logger.error(`Error getting system prompt: ${error}`);  
 return FALLBACK\_SYSTEM\_MESSAGE;  
 }  
}

#### Versioning Integration

The system automatically creates a new version whenever a prompt's content is changed:

// If content has changed and createVersion is true, create a new version  
if (contentChanged && createVersion) {  
 // Get the current version number  
 const { data: versionData, error: versionError } = await client  
 .from('system\_prompt\_versions')  
 .select('version\_number')  
 .eq('prompt\_id', id)  
 .order('version\_number', { ascending: false })  
 .limit(1);  
  
 // Calculate the next version number  
 const nextVersionNumber = versionData && versionData.length > 0  
 ? versionData[0].version\_number + 1  
 : 1;  
  
 // Create a new version with the current content  
 const { error: createVersionError } = await client  
 .from('system\_prompt\_versions')  
 .insert([{  
 prompt\_id: id,  
 version\_number: nextVersionNumber,  
 content: prompt.content,  
 variables: prompt.variables || currentPrompt.variables,  
 is\_active: true,  
 created\_by: prompt.createdBy  
 }]);  
}

// If content has changed and createVersion is true, create a new version  
if (contentChanged && createVersion) {  
 // Get the current version number  
 const { data: versionData, error: versionError } = await client  
 .from('system\_prompt\_versions')  
 .select('version\_number')  
 .eq('prompt\_id', id)  
 .order('version\_number', { ascending: false })  
 .limit(1);  
  
 // Calculate the next version number  
 const nextVersionNumber = versionData && versionData.length > 0  
 ? versionData[0].version\_number + 1  
 : 1;  
  
 // Create a new version with the current content  
 const { error: createVersionError } = await client  
 .from('system\_prompt\_versions')  
 .insert([{  
 prompt\_id: id,  
 version\_number: nextVersionNumber,  
 content: prompt.content,  
 variables: prompt.variables || currentPrompt.variables,  
 is\_active: true,  
 created\_by: prompt.createdBy  
 }]);  
}

#### Success Tracking Integration

The system tracks success rates for prompts through the renderPrompt and updatePromptTrackingRecord methods:

renderPrompt

updatePromptTrackingRecord

// If tracking ID is provided, create a tracking record  
if (options.trackingId && promptId) {  
 // Create a tracking record with pending status  
 await this.createPromptTrackingRecord({  
 id: options.trackingId,  
 promptId,  
 promptVersionId,  
 isSuccessful: false, // Will be updated later  
 context: options.data  
 });  
}  
  
// Later, update the tracking record with success/failure  
const success = await promptService.updatePromptTrackingRecord(  
 trackingId,  
 isSuccessful,  
 feedback  
);

// If tracking ID is provided, create a tracking record  
if (options.trackingId && promptId) {  
 // Create a tracking record with pending status  
 await this.createPromptTrackingRecord({  
 id: options.trackingId,  
 promptId,  
 promptVersionId,  
 isSuccessful: false, // Will be updated later  
 context: options.data  
 });  
}  
  
// Later, update the tracking record with success/failure  
const success = await promptService.updatePromptTrackingRecord(  
 trackingId,  
 isSuccessful,  
 feedback  
);

### Best Practices

When working with the prompt management system, follow these best practices:

1. Use descriptive names: Choose clear, descriptive names for prompts
2. Document variables: List all variables used in the prompt in the Variables field
3. Include file locations: Specify the file where the prompt is used
4. Test changes: After modifying prompts, test the affected functionality
5. Use prompt types: Categorize prompts correctly by type
6. Keep prompts focused: Each prompt should have a single, clear purpose
7. Consider fallbacks: The system includes fallbacks for when prompts are not found
8. Review version history: Before making changes, review the version history to understand previous changes
9. Add comments: When creating a new version, add a comment explaining the changes
10. Monitor success rates: Regularly review success rates to identify prompts that need improvement
11. A/B test prompts: Create multiple versions of a prompt to test different approaches
12. Collect feedback: Encourage users to provide feedback on prompt effectiveness

### Troubleshooting

If you encounter issues with the prompt management system:

1. Prompt not found: Check that the prompt name and type match what the code is looking for
2. Changes not taking effect: Clear the prompt cache by restarting the affected service
3. Permission errors: Ensure you have admin privileges to manage prompts
4. Version history not showing: Check that the prompt ID is being passed correctly to the version history API
5. Success rates not updating: Ensure that tracking IDs are being generated and passed correctly
6. Revert not working: Check that the version you're trying to revert to exists and is not already active

### Conclusion

The prompt management system provides a powerful way to manage AI prompts across the KAI platform. By centralizing prompt management, it enables consistent updates, testing, and optimization of AI interactions throughout the system.

The versioning system allows you to track changes over time and revert to previous versions if needed, providing a safety net for prompt experimentation. The success tracking system helps you identify which prompts are performing well and which need improvement, enabling data-driven prompt optimization.

Together, these features create a comprehensive prompt management system that supports continuous improvement of AI interactions across the platform.

# Prompt Success Tracking

Source: readme/prompt-success-tracking.md

---

## Prompt Success Tracking

This document describes the enhanced prompt success tracking system, which helps improve prompt quality over time by collecting detailed feedback and automatically detecting success based on user behavior.

### Overview

The prompt success tracking system consists of several components:

1. Database Schema: Tables for storing success tracking data, analytics, and monitoring alerts
2. Backend Services: APIs for collecting and analyzing success data
3. Client-Side Utilities: Tools for tracking user behavior and collecting feedback
4. Admin UI: Dashboard for monitoring prompt performance
5. Grafana Integration: Real-time monitoring of prompt success metrics

### Database Schema

The system uses the following tables:

* system\_prompt\_success\_tracking: Stores individual success tracking records
* prompt\_usage\_analytics: Aggregates daily analytics for each prompt
* prompt\_monitoring\_alerts: Stores alerts for prompts that fall below thresholds
* prompt\_monitoring\_settings: Stores alert settings for each prompt

system\_prompt\_success\_tracking

prompt\_usage\_analytics

prompt\_monitoring\_alerts

prompt\_monitoring\_settings

### Success Tracking Features

#### Detailed Feedback Options

The system collects detailed feedback from users:

* Success/Failure: Basic indication of whether the prompt was helpful
* Rating: Numeric rating from 1-5 stars
* Categories: Categorization of feedback (Accuracy, Relevance, Clarity, etc.)
* Tags: Specific tags for common issues (Missing Information, Too Technical, etc.)
* Comments: Free-form text feedback

#### Automatic Success Detection

The system can automatically detect whether a prompt was successful based on user behavior:

* Time Spent: How long the user spent on the page after receiving the response
* Scroll Depth: How far the user scrolled through the response
* Clicked Links: Whether the user clicked on links in the response
* Copied Text: Whether the user copied text from the response
* Follow-up Questions: Whether the user asked follow-up questions

The system uses a scoring algorithm to determine success:

const score = (  
 (timeSpentOnPage > 10000 ? 1 : 0) +  
 (scrollDepth > 0.5 ? 1 : 0) +  
 (clickedLinks > 0 ? 1 : 0) +  
 (copiedText ? 1 : 0) +  
 (followupQuestions > 0 ? 1 : 0)  
);  
  
// If score is 3 or higher, consider it successful  
isSuccessful = score >= 3;

const score = (  
 (timeSpentOnPage > 10000 ? 1 : 0) +  
 (scrollDepth > 0.5 ? 1 : 0) +  
 (clickedLinks > 0 ? 1 : 0) +  
 (copiedText ? 1 : 0) +  
 (followupQuestions > 0 ? 1 : 0)  
);  
  
// If score is 3 or higher, consider it successful  
isSuccessful = score >= 3;

#### Analytics and Monitoring

The system provides analytics and monitoring features:

* Success Rate: Percentage of successful prompt uses over time
* Usage Trends: Number of prompt uses over time
* Average Ratings: Average user ratings for each prompt
* Response Time: Average response time for each prompt
* Feedback Categories: Distribution of feedback categories
* Feedback Tags: Most common feedback tags

#### Alerts

The system can generate alerts when prompts fall below certain thresholds:

* Low Success Rate: Alert when success rate falls below a threshold
* Low Rating: Alert when average rating falls below a threshold
* High Response Time: Alert when response time exceeds a threshold
* High Failure Rate: Alert when failure rate exceeds a threshold

### Using the Success Tracking System

#### Backend Integration

To track prompt success in your backend code:

// When rendering a prompt  
const { content, trackingId } = await promptService.renderPrompt({  
 promptId: 'your-prompt-id',  
 data: { /\* your data \*/ },  
 trackSuccess: true  
});  
  
// Later, update the tracking record with success/failure  
await promptService.updatePromptTrackingRecord(  
 trackingId,  
 {  
 isSuccessful: true,  
 feedbackRating: 5,  
 feedbackCategory: 'Accuracy',  
 feedbackTags: ['Helpful', 'Well Explained']  
 }  
);  
  
// Or use auto-detection  
await promptService.autoDetectPromptSuccess(  
 trackingId,  
 responseTimeMs,  
 {  
 timeSpentOnPage: 15000,  
 scrollDepth: 0.8,  
 clickedLinks: 2,  
 copiedText: true,  
 followupQuestions: 1  
 }  
);

// When rendering a prompt  
const { content, trackingId } = await promptService.renderPrompt({  
 promptId: 'your-prompt-id',  
 data: { /\* your data \*/ },  
 trackSuccess: true  
});  
  
// Later, update the tracking record with success/failure  
await promptService.updatePromptTrackingRecord(  
 trackingId,  
 {  
 isSuccessful: true,  
 feedbackRating: 5,  
 feedbackCategory: 'Accuracy',  
 feedbackTags: ['Helpful', 'Well Explained']  
 }  
);  
  
// Or use auto-detection  
await promptService.autoDetectPromptSuccess(  
 trackingId,  
 responseTimeMs,  
 {  
 timeSpentOnPage: 15000,  
 scrollDepth: 0.8,  
 clickedLinks: 2,  
 copiedText: true,  
 followupQuestions: 1  
 }  
);

#### Client-Side Integration

To track prompt success in your client-side code:

// Import the prompt service  
import { promptService } from '../services/promptService';  
  
// Render a prompt  
const { content, trackingId } = await promptService.renderPrompt({  
 promptId: 'your-prompt-id',  
 data: { /\* your data \*/ },  
 trackSuccess: true  
});  
  
// Later, submit feedback  
await promptService.submitFeedback(  
 trackingId,  
 true, // isSuccessful  
 'Great response!', // feedback  
 5, // rating  
 'Accuracy', // category  
 ['Helpful', 'Well Explained'] // tags  
);  
  
// Or use auto-detection  
await promptService.autoDetectSuccess(trackingId);

// Import the prompt service  
import { promptService } from '../services/promptService';  
  
// Render a prompt  
const { content, trackingId } = await promptService.renderPrompt({  
 promptId: 'your-prompt-id',  
 data: { /\* your data \*/ },  
 trackSuccess: true  
});  
  
// Later, submit feedback  
await promptService.submitFeedback(  
 trackingId,  
 true, // isSuccessful  
 'Great response!', // feedback  
 5, // rating  
 'Accuracy', // category  
 ['Helpful', 'Well Explained'] // tags  
);  
  
// Or use auto-detection  
await promptService.autoDetectSuccess(trackingId);

#### Using the Feedback Component

To add a feedback component to your UI:

import PromptFeedback from '../components/feedback/PromptFeedback';  
  
// In your component  
return (  
 <div>  
 <p>AI Response: {aiResponse}</p>  
 <PromptFeedback  
 trackingId={trackingId}  
 onFeedbackSubmitted={(isSuccessful) => {  
 console.log('Feedback submitted:', isSuccessful);  
 }}  
 />  
 </div>  
);

import PromptFeedback from '../components/feedback/PromptFeedback';  
  
// In your component  
return (  
 <div>  
 <p>AI Response: {aiResponse}</p>  
 <PromptFeedback  
 trackingId={trackingId}  
 onFeedbackSubmitted={(isSuccessful) => {  
 console.log('Feedback submitted:', isSuccessful);  
 }}  
 />  
 </div>  
);

For a compact version:

<PromptFeedback  
 trackingId={trackingId}  
 compact={true}  
/>

<PromptFeedback  
 trackingId={trackingId}  
 compact={true}  
/>

### Monitoring Dashboard

The system includes a monitoring dashboard in the admin UI:

1. Navigate to "Prompt Monitoring" in the admin sidebar
2. View active alerts, success rates, and analytics
3. Configure alert settings for each prompt

### Grafana Integration

The system integrates with Grafana for real-time monitoring:

1. Navigate to "Grafana Dashboards" in the admin sidebar
2. Select the "Prompt Monitoring Dashboard"
3. View real-time metrics for prompt success

The Grafana dashboard includes:

* Success rate over time
* Usage trends
* Average ratings
* Response time trends
* Active alerts
* Feedback categories and tags

### Best Practices

1. Collect Feedback Consistently: Add feedback components to all AI responses
2. Use Auto-Detection: Enable auto-detection for all prompts to collect passive feedback
3. Monitor Alerts: Regularly check for alerts and address issues promptly
4. Analyze Trends: Use the analytics to identify trends and improve prompts
5. A/B Test Prompts: Create multiple versions of prompts and compare success rates
6. Set Appropriate Thresholds: Configure alert thresholds based on your requirements
7. Review Feedback: Regularly review user feedback to improve prompts

### Troubleshooting

* Missing Tracking Data: Ensure tracking IDs are being generated and passed correctly
* Auto-Detection Not Working: Check that event listeners are being attached correctly
* Alerts Not Triggering: Verify that alert settings are configured correctly
* Analytics Not Updating: Check that the analytics aggregation function is running

### Conclusion

The enhanced prompt success tracking system provides valuable insights into prompt performance and helps improve prompt quality over time. By collecting detailed feedback and automatically detecting success based on user behavior, you can continuously optimize your AI interactions.

# Property Based Recommendation Engine

Source: readme/property-based-recommendation-engine.md

---

## Property-Based Recommendation Engine

The Property-Based Recommendation Engine provides intelligent recommendations for materials based on property requirements, user preferences, and project context.

### Features

#### Property-Based Matching

* Property Requirements: Matches materials based on specific property requirements
* Weighted Matching: Prioritizes important properties in the matching process
* Similarity Calculation: Calculates similarity scores between requested and actual property values
* Relevance Scoring: Provides an overall relevance score for each recommendation

#### User Preference Learning

* Preference Tracking: Learns from user selections to personalize recommendations
* Property Weighting: Adjusts property weights based on user preferences
* Personalized Results: Provides recommendations tailored to each user's preferences
* Preference Management: Allows users to view and manage their preferences

#### Project Context Awareness

* Room Type Consideration: Adjusts recommendations based on the room type
* Project Type Awareness: Considers the type of project (renovation, new construction, etc.)
* Style Matching: Recommends materials that match the desired style
* Budget Awareness: Filters recommendations based on budget constraints
* Existing Material Compatibility: Suggests materials that work well with existing materials

#### Collaborative Filtering

* Similar User Patterns: Identifies patterns among similar users
* Collaborative Recommendations: Suggests materials based on what similar users have chosen
* Trend Identification: Identifies trending material combinations
* Community Insights: Leverages insights from the user community

### Technical Implementation

#### Backend

* Recommendation Service: Handles property-based recommendation logic
* Similarity Algorithms: Implements algorithms for different property types
* User Preference Management: Stores and retrieves user preferences
* Project Context Management: Manages project context information

#### Database

* Property Weights: Stores weights for different properties
* User Preferences: Stores user property preferences
* Project Contexts: Stores project context information
* Compatibility Rules: Stores rules for material compatibility

#### Frontend

* Recommendation Form: Allows users to specify property requirements
* Project Context Form: Captures project context information
* Recommendation Results: Displays recommended materials with explanations
* Detail View: Shows detailed property matches for each recommendation

### Usage

#### Getting Recommendations

1. Select a material type
2. Specify property requirements
3. Optionally provide project context
4. Click "Get Recommendations"
5. View the recommended materials
6. Click on a material to see detailed property matches

#### Using Project Context

1. Enable "Include Project Context"
2. Select a room type
3. Select a project type
4. Specify a style
5. Select a budget level
6. Add existing materials if applicable
7. Get recommendations that consider the project context

#### Comparing Recommendations

1. Select materials for comparison
2. Click "Compare Selected"
3. View the detailed comparison between materials
4. Make an informed decision based on the comparison

#### Managing Preferences

1. Navigate to the user preferences section
2. View current property preferences
3. Adjust preferences as needed
4. Save changes to personalize future recommendations

### Benefits

* Targeted Recommendations: Helps users find materials that meet their specific requirements
* Time Savings: Reduces time spent searching through catalogs
* Discovery: Introduces users to options they might not have considered
* Project Optimization: Improves project outcomes by suggesting optimal materials
* Personalization: Provides increasingly personalized recommendations over time
* Context Awareness: Considers the specific context of each project

# Property Inheritance System

Source: readme/property-inheritance-system.md

---

## Property Inheritance System

The Property Inheritance System allows properties to be inherited based on material type hierarchies, with support for overrides and templates.

### Overview

The Property Inheritance System provides a way to define default properties for materials based on their type and category. This reduces redundant data entry, ensures consistency across similar materials, makes it easier to update properties for entire categories of materials, and simplifies the creation of new material entries.

### Key Features

* Property Templates: Define templates for material properties that can be inherited
* Type-Based Inheritance: Automatically apply properties based on material type
* Category-Based Inheritance: Apply properties based on material category
* Hierarchical Inheritance: Support for parent-child relationships between templates
* Priority-Based Overrides: Higher priority templates override lower priority ones
* Conditional Rules: Apply properties based on conditions
* Property Overrides: Allow specific overrides for individual materials

### Architecture

The Property Inheritance System consists of the following components:

#### 1. Property Template Model

The Property Template model defines the structure for property templates:

* ID: Unique identifier for the template
* Name: Name of the template
* Description: Description of the template
* Material Type: Type of material the template applies to (optional)
* Category ID: Category the template applies to (optional)
* Parent Template ID: Parent template to inherit from (optional)
* Priority: Priority of the template (higher priority templates override lower priority ones)
* Properties: Key-value pairs of properties to apply
* Override Rules: Rules for when to apply properties

#### 2. Property Inheritance Service

The Property Inheritance Service handles the logic for applying property templates to materials:

* Apply Inheritance: Apply property templates to a material
* Get Applicable Templates: Get templates that apply to a material
* Apply Template: Apply a template to a material
* Check Override Rules: Check if a property should be applied based on override rules

#### 3. Material Service Integration

The Material Service integrates with the Property Inheritance Service to apply property templates when creating or updating materials:

* Create Material: Apply property templates when creating a material
* Update Material: Apply property templates when updating a material
* Apply Inheritance: Apply property templates to an existing material

#### 4. API Endpoints

The API endpoints for managing property templates:

* GET /api/property-templates: Get all property templates
* GET /api/property-templates/:id: Get a property template by ID
* POST /api/property-templates: Create a new property template
* PUT /api/property-templates/:id: Update a property template
* DELETE /api/property-templates/:id: Delete a property template
* POST /api/property-templates/:id/apply: Apply a property template to a material
* POST /api/materials/:id/apply-inheritance: Apply property inheritance to a material

#### 5. UI Components

The UI components for managing property templates:

* Property Template Manager: Component for managing property templates
* Property Template Form: Form for creating and editing property templates
* Property Template List: List of property templates
* Property Template Details: Details of a property template

### Usage

#### Creating a Property Template

1. Navigate to the Property Templates page in the admin panel
2. Click "Create Template"
3. Fill in the template details:
4. Name: Name of the template
5. Description: Description of the template
6. Material Type: Type of material the template applies to (optional)
7. Category: Category the template applies to (optional)
8. Parent Template: Parent template to inherit from (optional)
9. Priority: Priority of the template
10. Properties: Key-value pairs of properties to apply
11. Override Rules: Rules for when to apply properties
12. Click "Create" to save the template

#### Applying Property Inheritance

Property inheritance is automatically applied when creating or updating materials. You can also manually apply property inheritance to an existing material:

1. Navigate to the Materials page in the admin panel
2. Select a material
3. Click "Apply Inheritance"
4. Configure inheritance options:
5. Apply Defaults: Whether to apply default values
6. Override Existing: Whether to override existing values
7. Click "Apply" to apply property inheritance

### Implementation Details

#### Property Template Model

The Property Template model is implemented as a Mongoose schema in packages/server/src/models/propertyTemplate.model.ts.

packages/server/src/models/propertyTemplate.model.ts

#### Property Inheritance Service

The Property Inheritance Service is implemented in packages/server/src/services/propertyInheritance/propertyInheritanceService.ts.

packages/server/src/services/propertyInheritance/propertyInheritanceService.ts

#### Material Service Integration

The Material Service integration is implemented in packages/server/src/services/material/materialService.ts.

packages/server/src/services/material/materialService.ts

#### API Endpoints

The API endpoints are implemented in packages/server/src/controllers/propertyTemplate.controller.ts and packages/server/src/routes/propertyTemplate.routes.ts.

packages/server/src/controllers/propertyTemplate.controller.ts

packages/server/src/routes/propertyTemplate.routes.ts

#### UI Components

The UI components are implemented in packages/client/src/components/propertyTemplates/PropertyTemplateManager.tsx and packages/client/src/pages/admin/PropertyTemplatesPage.tsx.

packages/client/src/components/propertyTemplates/PropertyTemplateManager.tsx

packages/client/src/pages/admin/PropertyTemplatesPage.tsx

### Examples

#### Example 1: Basic Property Template

{  
 "name": "Ceramic Tile Template",  
 "description": "Default properties for ceramic tiles",  
 "materialType": "ceramic",  
 "isActive": true,  
 "priority": 10,  
 "properties": {  
 "finish": "matte",  
 "waterAbsorption": 0.5,  
 "slipResistance": "R9",  
 "technicalSpecs": {  
 "density": 2.3,  
 "hardness": 7  
 }  
 }  
}

{  
 "name": "Ceramic Tile Template",  
 "description": "Default properties for ceramic tiles",  
 "materialType": "ceramic",  
 "isActive": true,  
 "priority": 10,  
 "properties": {  
 "finish": "matte",  
 "waterAbsorption": 0.5,  
 "slipResistance": "R9",  
 "technicalSpecs": {  
 "density": 2.3,  
 "hardness": 7  
 }  
 }  
}

#### Example 2: Template with Override Rules

{  
 "name": "Porcelain Tile Template",  
 "description": "Default properties for porcelain tiles",  
 "materialType": "porcelain",  
 "isActive": true,  
 "priority": 20,  
 "properties": {  
 "finish": "polished",  
 "waterAbsorption": 0.1,  
 "slipResistance": "R10",  
 "technicalSpecs": {  
 "density": 2.5,  
 "hardness": 8  
 }  
 },  
 "overrideRules": [  
 {  
 "field": "finish",  
 "condition": "finish=textured",  
 "value": "textured"  
 },  
 {  
 "field": "slipResistance",  
 "condition": "finish=textured",  
 "value": "R11"  
 }  
 ]  
}

{  
 "name": "Porcelain Tile Template",  
 "description": "Default properties for porcelain tiles",  
 "materialType": "porcelain",  
 "isActive": true,  
 "priority": 20,  
 "properties": {  
 "finish": "polished",  
 "waterAbsorption": 0.1,  
 "slipResistance": "R10",  
 "technicalSpecs": {  
 "density": 2.5,  
 "hardness": 8  
 }  
 },  
 "overrideRules": [  
 {  
 "field": "finish",  
 "condition": "finish=textured",  
 "value": "textured"  
 },  
 {  
 "field": "slipResistance",  
 "condition": "finish=textured",  
 "value": "R11"  
 }  
 ]  
}

### Future Enhancements

* Template Versioning: Add support for versioning property templates
* Template Export/Import: Add support for exporting and importing property templates
* Template Cloning: Add support for cloning property templates
* Template Testing: Add support for testing property templates against materials
* Template Analytics: Add support for analyzing property template usage
* Template Validation: Add support for validating property templates against material types
* Template Inheritance Visualization: Add support for visualizing property template inheritance

# Property Relationship Graph

Source: readme/property-relationship-graph.md

---

## Property Relationship Graph

The Property Relationship Graph is a powerful system that defines and stores relationships between different material properties. This document explains how the system works, how to use it, and how to extend it.

### Overview

The Property Relationship Graph enables:

1. Defining Relationships: Specify how different properties relate to each other
2. Storing Correlations: Record statistical or manual correlations between property values
3. Setting Compatibility Rules: Define which property values work well together
4. Validating Property Combinations: Check if a set of property values is valid/compatible
5. Getting Recommendations: Suggest compatible property values based on existing selections
6. Visualizing Relationships: View a graph visualization of property relationships

### Key Concepts

#### Relationship Types

The system supports several types of relationships between properties:

* Correlation: Properties that tend to have related values (e.g., material type and water absorption)
* Dependency: Properties where one depends on the other (e.g., finish depends on material type)
* Compatibility: Properties that need to be compatible (e.g., finish and R-rating)
* Exclusion: Properties that have mutually exclusive values (e.g., certain finishes and R-ratings)
* Causation: Properties where one causes the other (e.g., material composition causes certain properties)
* Derivation: Properties where one is derived from the other (e.g., calculated properties)
* Association: General association between properties

#### Compatibility Types

For compatibility relationships, the system defines several compatibility levels:

* Compatible: Values that work together
* Recommended: Values that are recommended to be used together
* Not Recommended: Values that are not recommended to be used together
* Incompatible: Values that should not be used together

### Database Schema

The system uses three main tables:

1. property\_relationships: Defines relationships between properties
2. property\_value\_correlations: Stores correlations between specific property values
3. property\_compatibility\_rules: Defines compatibility rules between property values

### API Endpoints

#### Property Relationships

* POST /api/property-relationships: Create a new property relationship
* GET /api/property-relationships/:id: Get a property relationship by ID
* GET /api/property-relationships/material/:materialType: Get relationships by material type
* GET /api/property-relationships/source/:sourceProperty: Get relationships by source property
* GET /api/property-relationships/target/:targetProperty: Get relationships by target property
* PUT /api/property-relationships/:id: Update a property relationship
* DELETE /api/property-relationships/:id: Delete a property relationship

POST /api/property-relationships

GET /api/property-relationships/:id

GET /api/property-relationships/material/:materialType

GET /api/property-relationships/source/:sourceProperty

GET /api/property-relationships/target/:targetProperty

PUT /api/property-relationships/:id

DELETE /api/property-relationships/:id

#### Value Correlations

* POST /api/property-relationships/:relationshipId/correlations: Create a new value correlation
* GET /api/property-relationships/:relationshipId/correlations: Get correlations by relationship ID
* PUT /api/property-relationships/correlations/:id: Update a value correlation
* DELETE /api/property-relationships/correlations/:id: Delete a value correlation

POST /api/property-relationships/:relationshipId/correlations

GET /api/property-relationships/:relationshipId/correlations

PUT /api/property-relationships/correlations/:id

DELETE /api/property-relationships/correlations/:id

#### Compatibility Rules

* POST /api/property-relationships/:relationshipId/compatibility: Create a new compatibility rule
* GET /api/property-relationships/:relationshipId/compatibility: Get rules by relationship ID
* PUT /api/property-relationships/compatibility/:id: Update a compatibility rule
* DELETE /api/property-relationships/compatibility/:id: Delete a compatibility rule

POST /api/property-relationships/:relationshipId/compatibility

GET /api/property-relationships/:relationshipId/compatibility

PUT /api/property-relationships/compatibility/:id

DELETE /api/property-relationships/compatibility/:id

#### Validation and Recommendations

* POST /api/property-relationships/validate: Validate a set of property values
* POST /api/property-relationships/recommend: Get property recommendations
* GET /api/property-relationships/graph/:materialType: Get graph visualization data

POST /api/property-relationships/validate

POST /api/property-relationships/recommend

GET /api/property-relationships/graph/:materialType

### Usage Examples

#### Creating a Relationship

// Create a correlation relationship between material and finish  
const relationship = await fetch('/api/property-relationships', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 sourceProperty: 'material',  
 targetProperty: 'finish',  
 relationshipType: 'correlation',  
 materialType: 'tile',  
 strength: 0.8,  
 bidirectional: false,  
 description: 'Material type influences the available finish options'  
 })  
});

// Create a correlation relationship between material and finish  
const relationship = await fetch('/api/property-relationships', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 sourceProperty: 'material',  
 targetProperty: 'finish',  
 relationshipType: 'correlation',  
 materialType: 'tile',  
 strength: 0.8,  
 bidirectional: false,  
 description: 'Material type influences the available finish options'  
 })  
});

#### Adding Value Correlations

// Add a correlation between porcelain material and matte finish  
const correlation = await fetch(`/api/property-relationships/${relationshipId}/correlations`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 sourceValue: 'porcelain',  
 targetValue: 'matte',  
 correlationStrength: 0.7,  
 sampleSize: 100,  
 confidenceInterval: 0.05,  
 isStatistical: true  
 })  
});

// Add a correlation between porcelain material and matte finish  
const correlation = await fetch(`/api/property-relationships/${relationshipId}/correlations`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 sourceValue: 'porcelain',  
 targetValue: 'matte',  
 correlationStrength: 0.7,  
 sampleSize: 100,  
 confidenceInterval: 0.05,  
 isStatistical: true  
 })  
});

#### Adding Compatibility Rules

// Add a compatibility rule between matte finish and R11 rating  
const rule = await fetch(`/api/property-relationships/${relationshipId}/compatibility`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 sourceValue: 'matte',  
 targetValue: 'R11',  
 compatibilityType: 'recommended',  
 reason: 'Matte finish works well with R11 rating for outdoor applications'  
 })  
});

// Add a compatibility rule between matte finish and R11 rating  
const rule = await fetch(`/api/property-relationships/${relationshipId}/compatibility`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 sourceValue: 'matte',  
 targetValue: 'R11',  
 compatibilityType: 'recommended',  
 reason: 'Matte finish works well with R11 rating for outdoor applications'  
 })  
});

#### Validating Properties

// Validate a set of property values  
const validation = await fetch('/api/property-relationships/validate', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 materialType: 'tile',  
 properties: {  
 material: 'porcelain',  
 finish: 'polished',  
 rRating: 'R12'  
 }  
 })  
});  
  
// Check validation result  
const result = await validation.json();  
if (!result.result.isValid) {  
 console.log('Invalid property combination:');  
 result.result.issues.forEach(issue => {  
 console.log(`- ${issue.sourceProperty}=${issue.sourceValue} is ${issue.compatibilityType} with ${issue.targetProperty}=${issue.targetValue}`);  
 if (issue.reason) {  
 console.log(` Reason: ${issue.reason}`);  
 }  
 });  
}

// Validate a set of property values  
const validation = await fetch('/api/property-relationships/validate', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 materialType: 'tile',  
 properties: {  
 material: 'porcelain',  
 finish: 'polished',  
 rRating: 'R12'  
 }  
 })  
});  
  
// Check validation result  
const result = await validation.json();  
if (!result.result.isValid) {  
 console.log('Invalid property combination:');  
 result.result.issues.forEach(issue => {  
 console.log(`- ${issue.sourceProperty}=${issue.sourceValue} is ${issue.compatibilityType} with ${issue.targetProperty}=${issue.targetValue}`);  
 if (issue.reason) {  
 console.log(` Reason: ${issue.reason}`);  
 }  
 });  
}

#### Getting Recommendations

// Get recommendations for finish based on other properties  
const recommendations = await fetch('/api/property-relationships/recommend', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 materialType: 'tile',  
 properties: {  
 material: 'porcelain',  
 rRating: 'R11'  
 },  
 targetProperty: 'finish'  
 })  
});  
  
// Display recommendations  
const result = await recommendations.json();  
console.log(`Recommendations for ${result.result.property}:`);  
result.result.recommendations.forEach(rec => {  
 console.log(`- ${rec.value} (confidence: ${rec.confidence.toFixed(2)})`);  
});

// Get recommendations for finish based on other properties  
const recommendations = await fetch('/api/property-relationships/recommend', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 materialType: 'tile',  
 properties: {  
 material: 'porcelain',  
 rRating: 'R11'  
 },  
 targetProperty: 'finish'  
 })  
});  
  
// Display recommendations  
const result = await recommendations.json();  
console.log(`Recommendations for ${result.result.property}:`);  
result.result.recommendations.forEach(rec => {  
 console.log(`- ${rec.value} (confidence: ${rec.confidence.toFixed(2)})`);  
});

### Integration with Material Metadata Panel

The Property Relationship Graph integrates with the Material Metadata Panel to provide:

1. Validation: Validate property values as they are entered
2. Recommendations: Suggest compatible property values
3. Warnings: Show warnings for incompatible property combinations
4. Auto-fill: Automatically fill in related properties

#### Example Integration

// In MaterialMetadataPanel.tsx  
  
// Validate properties when they change  
const handleMetadataChange = async (newMetadata: any) => {  
 // Validate the new metadata  
 const validation = await propertyRelationshipService.validateProperties({  
 materialType,  
 properties: newMetadata  
 });  
  
 // Update validation state  
 setValidationResult(validation);  
  
 // Call parent handler  
 onMetadataChange(newMetadata);  
};  
  
// Get recommendations when a property changes  
const handlePropertyChange = async (property: string, value: string) => {  
 // Update metadata  
 const newMetadata = { ...metadata, [property]: value };  
  
 // Get recommendations for other properties  
 const recommendations = await getRecommendationsForProperties(newMetadata);  
  
 // Update recommendations state  
 setRecommendations(recommendations);  
  
 // Call the main handler  
 handleMetadataChange(newMetadata);  
};

// In MaterialMetadataPanel.tsx  
  
// Validate properties when they change  
const handleMetadataChange = async (newMetadata: any) => {  
 // Validate the new metadata  
 const validation = await propertyRelationshipService.validateProperties({  
 materialType,  
 properties: newMetadata  
 });  
  
 // Update validation state  
 setValidationResult(validation);  
  
 // Call parent handler  
 onMetadataChange(newMetadata);  
};  
  
// Get recommendations when a property changes  
const handlePropertyChange = async (property: string, value: string) => {  
 // Update metadata  
 const newMetadata = { ...metadata, [property]: value };  
  
 // Get recommendations for other properties  
 const recommendations = await getRecommendationsForProperties(newMetadata);  
  
 // Update recommendations state  
 setRecommendations(recommendations);  
  
 // Call the main handler  
 handleMetadataChange(newMetadata);  
};

### Benefits

The Property Relationship Graph provides numerous benefits:

1. Smarter Recommendations: The system can suggest compatible property values based on existing selections
2. Better Validation: Property values can be validated against known relationships
3. Enhanced AI Understanding: AI models can leverage relationship data for better inference
4. Improved Search: Search can use relationship data to find more relevant results
5. Error Detection: The system can detect inconsistent or unlikely property combinations
6. Knowledge Capture: The system captures domain expertise about property relationships

### Future Enhancements

Potential future enhancements to the Property Relationship Graph:

1. Machine Learning Integration: Use machine learning to automatically discover relationships and correlations
2. Temporal Analysis: Track how relationships change over time
3. Confidence Scoring: Add confidence scores to relationships based on data quality
4. User Feedback Loop: Allow users to provide feedback on recommendations
5. Advanced Visualization: Enhance the graph visualization with more interactive features

### Conclusion

The Property Relationship Graph is a powerful system for defining and working with relationships between material properties. By capturing these relationships, the system enables smarter recommendations, better validation, and enhanced AI understanding of materials.

# Quality Of Data And Metrics

Source: readme/quality-of-data-and-metrics.md

---

## Quality of Data and Metrics

The Kai platform implements comprehensive data quality assessment and performance monitoring systems to ensure reliable operation, high-quality data processing, and continuous improvement. This document details the methodologies, metrics, and tools used throughout the system.

### Data Quality Assessment

#### Dataset Quality Framework

The system provides extensive dataset quality evaluation:

1. Class Balance Analysis
2. Distribution metrics across categories
3. Gini coefficient calculation
4. Max-to-min ratio evaluation
5. Underrepresented class detection
6. Imbalance impact prediction
7. Image Quality Assessment
8. Resolution analysis and minimum requirements
9. Blur detection and quantification
10. Lighting assessment (underexposure, overexposure)
11. Artifact identification
12. Visual quality scoring
13. Integrity Validation
14. Duplicate detection and quantification
15. Corrupted file identification
16. Metadata consistency checking
17. Format validation
18. Structure verification
19. Automated Recommendations
20. Class balancing strategies
21. Image enhancement suggestions
22. Data cleaning recommendations
23. Augmentation opportunities
24. Collection prioritization guidance

Imbalance impact prediction

Image Quality Assessment

Visual quality scoring

Integrity Validation

Structure verification

Automated Recommendations

#### OCR Quality Evaluation

The system implements robust OCR quality measurement:

1. Confidence Scoring
2. Character-level confidence metrics
3. Word and phrase reliability assessment
4. Context-based validation
5. Statistical confidence analysis
6. Uncertainty quantification
7. Quality Indicators
8. Extracted text completeness
9. Structure preservation accuracy
10. Format recognition precision
11. Language detection confidence
12. Character recognition reliability
13. Error Detection
14. Common OCR error pattern identification
15. Context-inconsistent text flagging
16. Suspicious format detection
17. Domain-specific validation rules
18. Reference-based verification
19. Manual Review Triggering
20. Low-confidence threshold identification
21. Critical content verification requests
22. Uncertainty-based prioritization
23. Statistical anomaly flagging
24. Domain-specific validation failures

Uncertainty quantification

Quality Indicators

Character recognition reliability

Error Detection

Reference-based verification

Manual Review Triggering

#### Recognition Quality Metrics

The system tracks material recognition quality:

1. Accuracy Metrics
2. Top-1 and top-5 accuracy
3. Precision and recall by category
4. F1-score calculation
5. Confusion matrix analysis
6. Error distribution patterns
7. Confidence Assessment
8. Prediction confidence distribution
9. Uncertainty quantification
10. Thresholding effectiveness
11. Calibration curve analysis
12. Rejection capability evaluation
13. Feature Quality Analysis
14. Feature vector distribution analysis
15. Embedding space visualization
16. Cluster separation metrics
17. Dimensionality effectiveness
18. Feature importance ranking
19. Feedback Integration
20. User correction tracking
21. Error pattern analysis
22. Continuous improvement metrics
23. Learning curve progression
24. Model robustness evaluation

Error distribution patterns

Confidence Assessment

Rejection capability evaluation

Feature Quality Analysis

Feature importance ranking

Feedback Integration

### Performance Monitoring

#### Queue Processing Metrics

The system monitors queue performance:

1. Throughput Tracking
2. Jobs processed per hour/day
3. Processing time distribution
4. Queue length trends
5. Backlog analysis
6. Peak load handling
7. Resource Utilization
8. CPU/memory usage during processing
9. Storage requirements
10. Network bandwidth consumption
11. Database connection utilization
12. Scaling threshold monitoring
13. Error Rate Analysis
14. Failure percentage tracking
15. Error categorization
16. Retry success rates
17. Common failure patterns
18. Recovery time measurements
19. SLA Compliance
20. Processing time against targets
21. Completion rate metrics
22. Priority handling effectiveness
23. Time-in-queue analysis
24. End-to-end processing time

Peak load handling

Resource Utilization

Scaling threshold monitoring

Error Rate Analysis

Recovery time measurements

SLA Compliance

#### Training Performance Metrics

The system tracks model training effectiveness:

1. Learning Progress
2. Loss curve tracking
3. Accuracy progression
4. Validation metrics monitoring
5. Convergence speed analysis
6. Plateau detection
7. Resource Efficiency
8. Training time per epoch
9. Memory consumption
10. GPU utilization
11. Batch size optimization
12. Distributed training efficiency
13. Generalization Assessment
14. Training/validation gap analysis
15. Overfitting detection
16. Cross-validation performance
17. Test set evaluation
18. Out-of-distribution robustness
19. Hyperparameter Effectiveness
20. Parameter sensitivity analysis
21. Optimization trajectory tracking
22. Best configuration identification
23. Search efficiency metrics
24. Transfer learning effectiveness

Plateau detection

Resource Efficiency

Distributed training efficiency

Generalization Assessment

Out-of-distribution robustness

Hyperparameter Effectiveness

#### System Health Monitoring

The system includes comprehensive health tracking:

1. Service Availability
2. Uptime percentage
3. Response time tracking
4. Error rate monitoring
5. API endpoint health
6. Database connection status
7. Resource Management
8. Server load trends
9. Memory utilization patterns
10. Storage capacity monitoring
11. Network throughput
12. Database connection pool usage
13. Latency Tracking
14. API response times
15. Database query performance
16. End-to-end request latency
17. Processing pipeline timing
18. Real-time events delivery speed
19. Error Tracking
20. Exception rate monitoring
21. Error categorization
22. Issue recurrence patterns
23. Recovery success rates
24. Log volume analysis

Database connection status

Resource Management

Database connection pool usage

Latency Tracking

Real-time events delivery speed

Error Tracking

### Technical Implementation

#### Dataset Quality Analyzer

The system implements dataset quality analysis:

/\*\*  
 \* Analyze dataset quality and generate metrics  
 \* @param datasetId Dataset ID  
 \* @returns Dataset quality metrics  
 \*/  
public async analyzeDatasetQuality(datasetId: string): Promise<DatasetQualityMetrics> {  
 logger.info(`Analyzing quality of dataset ${datasetId}`);  
  
 try {  
 // Get the dataset  
 const dataset = await supabaseDatasetService.getDatasetById(datasetId);  
 if (!dataset) {  
 throw new Error(`Dataset not found: ${datasetId}`);  
 }  
  
 // Get classes for the dataset  
 const classes = await supabaseDatasetService.getDatasetClasses(datasetId);  
  
 // Initialize metrics  
 const metrics: DatasetQualityMetrics = {  
 datasetId,  
 classBalance: {  
 score: 0,  
 details: {  
 classCounts: {},  
 maxToMinRatio: 0,  
 giniCoefficient: 0  
 }  
 },  
 imageQuality: {  
 score: 0,  
 details: {  
 lowResolutionImages: 0,  
 blurryImages: 0,  
 poorLightingImages: 0,  
 overexposedImages: 0,  
 avgResolution: { width: 0, height: 0 }  
 }  
 },  
 duplicationMetrics: {  
 duplicateImages: 0,  
 nearDuplicates: 0  
 },  
 overallQualityScore: 0,  
 recommendations: []  
 };  
  
 // Calculate class balance metrics  
 const classCounts: Record<string, number> = {};  
 let totalImages = 0;  
 let minClassSize = Number.MAX\_SAFE\_INTEGER;  
 let maxClassSize = 0;  
  
 for (const cls of classes) {  
 // Get images for class  
 const images = await supabaseDatasetService.getDatasetClassImages(cls.id, 1000, 0);  
 const classSize = images.length;  
  
 classCounts[cls.name] = classSize;  
 totalImages += classSize;  
  
 if (classSize > 0) {  
 minClassSize = Math.min(minClassSize, classSize);  
 maxClassSize = Math.max(maxClassSize, classSize);  
 }  
 }  
  
 metrics.classBalance.details.classCounts = classCounts;  
 metrics.classBalance.details.maxToMinRatio = maxClassSize / (minClassSize || 1);  
  
 // Calculate Gini coefficient for class balance  
 metrics.classBalance.details.giniCoefficient = this.calculateGiniCoefficient(Object.values(classCounts));  
  
 // Calculate class balance score (0-100)  
 const balanceNormalization = Math.min(1, 1 / metrics.classBalance.details.maxToMinRatio);  
 metrics.classBalance.score = Math.round((1 - metrics.classBalance.details.giniCoefficient) \* 100 \* balanceNormalization);  
  
 // Analyze image quality  
 let totalWidth = 0;  
 let totalHeight = 0;  
 let totalAnalyzedImages = 0;  
  
 for (const cls of classes) {  
 const images = await supabaseDatasetService.getDatasetClassImages(cls.id, 100, 0); // Sample up to 100 images  
  
 for (const img of images) {  
 totalAnalyzedImages++;  
  
 if (img.width && img.height) {  
 totalWidth += img.width;  
 totalHeight += img.height;  
  
 if (img.width < 224 || img.height < 224) {  
 metrics.imageQuality.details.lowResolutionImages++;  
 }  
 }  
  
 // Here we would implement more sophisticated image quality checks  
 // For now, we'll use simple heuristics based on img metadata if available  
 if (img.metadata) {  
 if (img.metadata.blurScore && img.metadata.blurScore > 0.5) {  
 metrics.imageQuality.details.blurryImages++;  
 }  
 if (img.metadata.brightness && img.metadata.brightness < 0.2) {  
 metrics.imageQuality.details.poorLightingImages++;  
 }  
 if (img.metadata.brightness && img.metadata.brightness > 0.9) {  
 metrics.imageQuality.details.overexposedImages++;  
 }  
 }  
 }  
 }  
  
 if (totalAnalyzedImages > 0) {  
 metrics.imageQuality.details.avgResolution = {  
 width: Math.round(totalWidth / totalAnalyzedImages),  
 height: Math.round(totalHeight / totalAnalyzedImages)  
 };  
 }  
  
 // Calculate image quality score (0-100)  
 const lowResPercent = metrics.imageQuality.details.lowResolutionImages / (totalAnalyzedImages || 1);  
 const blurryPercent = metrics.imageQuality.details.blurryImages / (totalAnalyzedImages || 1);  
 const lightingIssuesPercent = (metrics.imageQuality.details.poorLightingImages +  
 metrics.imageQuality.details.overexposedImages) / (totalAnalyzedImages || 1);  
  
 metrics.imageQuality.score = Math.round(100 \* (1 - (lowResPercent \* 0.4 + blurryPercent \* 0.4 + lightingIssuesPercent \* 0.2)));  
  
 // Calculate overall quality score as weighted average of component scores  
 metrics.overallQualityScore = Math.round(  
 metrics.classBalance.score \* 0.4 +  
 metrics.imageQuality.score \* 0.6  
 );  
  
 // Generate recommendations based on findings  
 if (metrics.classBalance.details.maxToMinRatio > 3) {  
 metrics.recommendations.push('Balance classes by augmenting underrepresented classes or collecting more samples.');  
 }  
  
 if (metrics.imageQuality.details.lowResolutionImages > 0) {  
 metrics.recommendations.push(`Improve ${metrics.imageQuality.details.lowResolutionImages} low-resolution images with upscaling or replacement.`);  
 }  
  
 if (metrics.imageQuality.details.blurryImages > 0) {  
 metrics.recommendations.push(`Remove or enhance ${metrics.imageQuality.details.blurryImages} blurry images.`);  
 }  
  
 if ((metrics.imageQuality.details.poorLightingImages + metrics.imageQuality.details.overexposedImages) > 0) {  
 metrics.recommendations.push('Apply lighting normalization to improve consistency across the dataset.');  
 }  
  
 return metrics;  
 } catch (err) {  
 logger.error(`Error analyzing dataset quality: ${err}`);  
 throw new Error(`Failed to analyze dataset quality: ${err instanceof Error ? err.message : String(err)}`);  
 }  
}

/\*\*  
 \* Analyze dataset quality and generate metrics  
 \* @param datasetId Dataset ID  
 \* @returns Dataset quality metrics  
 \*/  
public async analyzeDatasetQuality(datasetId: string): Promise<DatasetQualityMetrics> {  
 logger.info(`Analyzing quality of dataset ${datasetId}`);  
  
 try {  
 // Get the dataset  
 const dataset = await supabaseDatasetService.getDatasetById(datasetId);  
 if (!dataset) {  
 throw new Error(`Dataset not found: ${datasetId}`);  
 }  
  
 // Get classes for the dataset  
 const classes = await supabaseDatasetService.getDatasetClasses(datasetId);  
  
 // Initialize metrics  
 const metrics: DatasetQualityMetrics = {  
 datasetId,  
 classBalance: {  
 score: 0,  
 details: {  
 classCounts: {},  
 maxToMinRatio: 0,  
 giniCoefficient: 0  
 }  
 },  
 imageQuality: {  
 score: 0,  
 details: {  
 lowResolutionImages: 0,  
 blurryImages: 0,  
 poorLightingImages: 0,  
 overexposedImages: 0,  
 avgResolution: { width: 0, height: 0 }  
 }  
 },  
 duplicationMetrics: {  
 duplicateImages: 0,  
 nearDuplicates: 0  
 },  
 overallQualityScore: 0,  
 recommendations: []  
 };  
  
 // Calculate class balance metrics  
 const classCounts: Record<string, number> = {};  
 let totalImages = 0;  
 let minClassSize = Number.MAX\_SAFE\_INTEGER;  
 let maxClassSize = 0;  
  
 for (const cls of classes) {  
 // Get images for class  
 const images = await supabaseDatasetService.getDatasetClassImages(cls.id, 1000, 0);  
 const classSize = images.length;  
  
 classCounts[cls.name] = classSize;  
 totalImages += classSize;  
  
 if (classSize > 0) {  
 minClassSize = Math.min(minClassSize, classSize);  
 maxClassSize = Math.max(maxClassSize, classSize);  
 }  
 }  
  
 metrics.classBalance.details.classCounts = classCounts;  
 metrics.classBalance.details.maxToMinRatio = maxClassSize / (minClassSize || 1);  
  
 // Calculate Gini coefficient for class balance  
 metrics.classBalance.details.giniCoefficient = this.calculateGiniCoefficient(Object.values(classCounts));  
  
 // Calculate class balance score (0-100)  
 const balanceNormalization = Math.min(1, 1 / metrics.classBalance.details.maxToMinRatio);  
 metrics.classBalance.score = Math.round((1 - metrics.classBalance.details.giniCoefficient) \* 100 \* balanceNormalization);  
  
 // Analyze image quality  
 let totalWidth = 0;  
 let totalHeight = 0;  
 let totalAnalyzedImages = 0;  
  
 for (const cls of classes) {  
 const images = await supabaseDatasetService.getDatasetClassImages(cls.id, 100, 0); // Sample up to 100 images  
  
 for (const img of images) {  
 totalAnalyzedImages++;  
  
 if (img.width && img.height) {  
 totalWidth += img.width;  
 totalHeight += img.height;  
  
 if (img.width < 224 || img.height < 224) {  
 metrics.imageQuality.details.lowResolutionImages++;  
 }  
 }  
  
 // Here we would implement more sophisticated image quality checks  
 // For now, we'll use simple heuristics based on img metadata if available  
 if (img.metadata) {  
 if (img.metadata.blurScore && img.metadata.blurScore > 0.5) {  
 metrics.imageQuality.details.blurryImages++;  
 }  
 if (img.metadata.brightness && img.metadata.brightness < 0.2) {  
 metrics.imageQuality.details.poorLightingImages++;  
 }  
 if (img.metadata.brightness && img.metadata.brightness > 0.9) {  
 metrics.imageQuality.details.overexposedImages++;  
 }  
 }  
 }  
 }  
  
 if (totalAnalyzedImages > 0) {  
 metrics.imageQuality.details.avgResolution = {  
 width: Math.round(totalWidth / totalAnalyzedImages),  
 height: Math.round(totalHeight / totalAnalyzedImages)  
 };  
 }  
  
 // Calculate image quality score (0-100)  
 const lowResPercent = metrics.imageQuality.details.lowResolutionImages / (totalAnalyzedImages || 1);  
 const blurryPercent = metrics.imageQuality.details.blurryImages / (totalAnalyzedImages || 1);  
 const lightingIssuesPercent = (metrics.imageQuality.details.poorLightingImages +  
 metrics.imageQuality.details.overexposedImages) / (totalAnalyzedImages || 1);  
  
 metrics.imageQuality.score = Math.round(100 \* (1 - (lowResPercent \* 0.4 + blurryPercent \* 0.4 + lightingIssuesPercent \* 0.2)));  
  
 // Calculate overall quality score as weighted average of component scores  
 metrics.overallQualityScore = Math.round(  
 metrics.classBalance.score \* 0.4 +  
 metrics.imageQuality.score \* 0.6  
 );  
  
 // Generate recommendations based on findings  
 if (metrics.classBalance.details.maxToMinRatio > 3) {  
 metrics.recommendations.push('Balance classes by augmenting underrepresented classes or collecting more samples.');  
 }  
  
 if (metrics.imageQuality.details.lowResolutionImages > 0) {  
 metrics.recommendations.push(`Improve ${metrics.imageQuality.details.lowResolutionImages} low-resolution images with upscaling or replacement.`);  
 }  
  
 if (metrics.imageQuality.details.blurryImages > 0) {  
 metrics.recommendations.push(`Remove or enhance ${metrics.imageQuality.details.blurryImages} blurry images.`);  
 }  
  
 if ((metrics.imageQuality.details.poorLightingImages + metrics.imageQuality.details.overexposedImages) > 0) {  
 metrics.recommendations.push('Apply lighting normalization to improve consistency across the dataset.');  
 }  
  
 return metrics;  
 } catch (err) {  
 logger.error(`Error analyzing dataset quality: ${err}`);  
 throw new Error(`Failed to analyze dataset quality: ${err instanceof Error ? err.message : String(err)}`);  
 }  
}

#### Training Metrics Tracker

The system implements comprehensive training metrics tracking:

@dataclass  
class TrainingMetric:  
 """Single training metric value with metadata"""  
 name: str  
 value: float  
 epoch: Optional[int] = None  
 iteration: Optional[int] = None  
 timestamp: float = field(default\_factory=time.time)  
  
class MetricsTracker:  
 """Tracks training metrics over time"""  
  
 def \_\_init\_\_(self, job\_id: str, metrics\_dir: Optional[str] = None):  
 """  
 Initialize metrics tracker  
  
 Args:  
 job\_id: Unique identifier for the training job  
 metrics\_dir: Directory to save metrics (if None, metrics are only kept in memory)  
 """  
 self.job\_id = job\_id  
 self.metrics\_dir = metrics\_dir  
 self.metrics: Dict[str, List[TrainingMetric]] = {}  
 self.epochs: List[int] = []  
  
 # Create metrics directory if provided  
 if metrics\_dir:  
 os.makedirs(metrics\_dir, exist\_ok=True)  
  
 def add\_metric(self, name: str, value: float, epoch: Optional[int] = None,  
 iteration: Optional[int] = None) -> None:  
 """  
 Add a single metric value  
  
 Args:  
 name: Metric name  
 value: Metric value  
 epoch: Optional epoch number  
 iteration: Optional iteration number within epoch  
 """  
 metric = TrainingMetric(name, value, epoch, iteration)  
  
 if name not in self.metrics:  
 self.metrics[name] = []  
  
 self.metrics[name].append(metric)  
  
 # Track epochs for plotting  
 if epoch is not None and epoch not in self.epochs:  
 self.epochs.append(epoch)  
 self.epochs.sort()  
  
 # Save metrics to file if directory is provided  
 if self.metrics\_dir:  
 self.\_save\_metrics()  
  
 def add\_metrics\_dict(self, metrics\_dict: Dict[str, float], epoch: Optional[int] = None,  
 iteration: Optional[int] = None) -> None:  
 """  
 Add multiple metrics from a dictionary  
  
 Args:  
 metrics\_dict: Dictionary of metric name to value  
 epoch: Optional epoch number  
 iteration: Optional iteration number  
 """  
 for name, value in metrics\_dict.items():  
 self.add\_metric(name, value, epoch, iteration)  
  
 def get\_metric\_values(self, name: str) -> List[float]:  
 """  
 Get all values for a specific metric  
  
 Args:  
 name: Metric name  
  
 Returns:  
 List of metric values  
 """  
 if name not in self.metrics:  
 return []  
  
 return [metric.value for metric in self.metrics[name]]  
  
 def get\_metric\_by\_epoch(self, name: str) -> Dict[int, List[float]]:  
 """  
 Get metric values organized by epoch  
  
 Args:  
 name: Metric name  
  
 Returns:  
 Dictionary mapping epoch numbers to lists of values  
 """  
 if name not in self.metrics:  
 return {}  
  
 result = {}  
 for metric in self.metrics[name]:  
 if metric.epoch is not None:  
 if metric.epoch not in result:  
 result[metric.epoch] = []  
 result[metric.epoch].append(metric.value)  
  
 return result  
  
 def get\_latest\_metrics(self) -> Dict[str, float]:  
 """  
 Get the most recent value for each metric  
  
 Returns:  
 Dictionary mapping metric names to their latest values  
 """  
 result = {}  
 for name, metrics in self.metrics.items():  
 if metrics:  
 result[name] = metrics[-1].value  
  
 return result  
  
 def \_save\_metrics(self) -> None:  
 """Save metrics to a JSON file"""  
 metrics\_file = os.path.join(self.metrics\_dir, f"{self.job\_id}\_metrics.json")  
  
 # Convert metrics to serializable format  
 serializable\_metrics = {}  
 for name, metrics in self.metrics.items():  
 serializable\_metrics[name] = [  
 {  
 "value": metric.value,  
 "epoch": metric.epoch,  
 "iteration": metric.iteration,  
 "timestamp": metric.timestamp  
 }  
 for metric in metrics  
 ]  
  
 # Save to file  
 with open(metrics\_file, 'w') as f:  
 json.dump(serializable\_metrics, f, indent=2)  
  
 def load\_metrics(self) -> bool:  
 """  
 Load metrics from file  
  
 Returns:  
 True if metrics were loaded successfully, False otherwise  
 """  
 if not self.metrics\_dir:  
 return False  
  
 metrics\_file = os.path.join(self.metrics\_dir, f"{self.job\_id}\_metrics.json")  
  
 if not os.path.exists(metrics\_file):  
 return False  
  
 try:  
 with open(metrics\_file, 'r') as f:  
 serialized\_metrics = json.load(f)  
  
 # Convert to TrainingMetric objects  
 for name, metrics\_data in serialized\_metrics.items():  
 self.metrics[name] = [  
 TrainingMetric(  
 name=name,  
 value=data["value"],  
 epoch=data.get("epoch"),  
 iteration=data.get("iteration"),  
 timestamp=data.get("timestamp", time.time())  
 )  
 for data in metrics\_data  
 ]  
  
 # Update epochs list  
 for metric in self.metrics[name]:  
 if metric.epoch is not None and metric.epoch not in self.epochs:  
 self.epochs.append(metric.epoch)  
  
 self.epochs.sort()  
  
 return True  
  
 except Exception as e:  
 logger.error(f"Error loading metrics: {e}")  
 return False

@dataclass  
class TrainingMetric:  
 """Single training metric value with metadata"""  
 name: str  
 value: float  
 epoch: Optional[int] = None  
 iteration: Optional[int] = None  
 timestamp: float = field(default\_factory=time.time)  
  
class MetricsTracker:  
 """Tracks training metrics over time"""  
  
 def \_\_init\_\_(self, job\_id: str, metrics\_dir: Optional[str] = None):  
 """  
 Initialize metrics tracker  
  
 Args:  
 job\_id: Unique identifier for the training job  
 metrics\_dir: Directory to save metrics (if None, metrics are only kept in memory)  
 """  
 self.job\_id = job\_id  
 self.metrics\_dir = metrics\_dir  
 self.metrics: Dict[str, List[TrainingMetric]] = {}  
 self.epochs: List[int] = []  
  
 # Create metrics directory if provided  
 if metrics\_dir:  
 os.makedirs(metrics\_dir, exist\_ok=True)  
  
 def add\_metric(self, name: str, value: float, epoch: Optional[int] = None,  
 iteration: Optional[int] = None) -> None:  
 """  
 Add a single metric value  
  
 Args:  
 name: Metric name  
 value: Metric value  
 epoch: Optional epoch number  
 iteration: Optional iteration number within epoch  
 """  
 metric = TrainingMetric(name, value, epoch, iteration)  
  
 if name not in self.metrics:  
 self.metrics[name] = []  
  
 self.metrics[name].append(metric)  
  
 # Track epochs for plotting  
 if epoch is not None and epoch not in self.epochs:  
 self.epochs.append(epoch)  
 self.epochs.sort()  
  
 # Save metrics to file if directory is provided  
 if self.metrics\_dir:  
 self.\_save\_metrics()  
  
 def add\_metrics\_dict(self, metrics\_dict: Dict[str, float], epoch: Optional[int] = None,  
 iteration: Optional[int] = None) -> None:  
 """  
 Add multiple metrics from a dictionary  
  
 Args:  
 metrics\_dict: Dictionary of metric name to value  
 epoch: Optional epoch number  
 iteration: Optional iteration number  
 """  
 for name, value in metrics\_dict.items():  
 self.add\_metric(name, value, epoch, iteration)  
  
 def get\_metric\_values(self, name: str) -> List[float]:  
 """  
 Get all values for a specific metric  
  
 Args:  
 name: Metric name  
  
 Returns:  
 List of metric values  
 """  
 if name not in self.metrics:  
 return []  
  
 return [metric.value for metric in self.metrics[name]]  
  
 def get\_metric\_by\_epoch(self, name: str) -> Dict[int, List[float]]:  
 """  
 Get metric values organized by epoch  
  
 Args:  
 name: Metric name  
  
 Returns:  
 Dictionary mapping epoch numbers to lists of values  
 """  
 if name not in self.metrics:  
 return {}  
  
 result = {}  
 for metric in self.metrics[name]:  
 if metric.epoch is not None:  
 if metric.epoch not in result:  
 result[metric.epoch] = []  
 result[metric.epoch].append(metric.value)  
  
 return result  
  
 def get\_latest\_metrics(self) -> Dict[str, float]:  
 """  
 Get the most recent value for each metric  
  
 Returns:  
 Dictionary mapping metric names to their latest values  
 """  
 result = {}  
 for name, metrics in self.metrics.items():  
 if metrics:  
 result[name] = metrics[-1].value  
  
 return result  
  
 def \_save\_metrics(self) -> None:  
 """Save metrics to a JSON file"""  
 metrics\_file = os.path.join(self.metrics\_dir, f"{self.job\_id}\_metrics.json")  
  
 # Convert metrics to serializable format  
 serializable\_metrics = {}  
 for name, metrics in self.metrics.items():  
 serializable\_metrics[name] = [  
 {  
 "value": metric.value,  
 "epoch": metric.epoch,  
 "iteration": metric.iteration,  
 "timestamp": metric.timestamp  
 }  
 for metric in metrics  
 ]  
  
 # Save to file  
 with open(metrics\_file, 'w') as f:  
 json.dump(serializable\_metrics, f, indent=2)  
  
 def load\_metrics(self) -> bool:  
 """  
 Load metrics from file  
  
 Returns:  
 True if metrics were loaded successfully, False otherwise  
 """  
 if not self.metrics\_dir:  
 return False  
  
 metrics\_file = os.path.join(self.metrics\_dir, f"{self.job\_id}\_metrics.json")  
  
 if not os.path.exists(metrics\_file):  
 return False  
  
 try:  
 with open(metrics\_file, 'r') as f:  
 serialized\_metrics = json.load(f)  
  
 # Convert to TrainingMetric objects  
 for name, metrics\_data in serialized\_metrics.items():  
 self.metrics[name] = [  
 TrainingMetric(  
 name=name,  
 value=data["value"],  
 epoch=data.get("epoch"),  
 iteration=data.get("iteration"),  
 timestamp=data.get("timestamp", time.time())  
 )  
 for data in metrics\_data  
 ]  
  
 # Update epochs list  
 for metric in self.metrics[name]:  
 if metric.epoch is not None and metric.epoch not in self.epochs:  
 self.epochs.append(metric.epoch)  
  
 self.epochs.sort()  
  
 return True  
  
 except Exception as e:  
 logger.error(f"Error loading metrics: {e}")  
 return False

#### OCR Confidence Scoring

The system implements OCR quality assessment:

class OCRConfidenceScorer:  
 """Class for evaluating OCR quality and improving results"""  
  
 def \_\_init\_\_(self, config=None):  
 """  
 Initialize the OCR confidence scorer  
  
 Args:  
 config: Configuration dictionary  
 """  
 self.config = {  
 'min\_confidence': 0.5,  
 'post\_processing\_enabled': True,  
 'use\_language\_model': True,  
 'domain\_specific\_correction': True,  
 'correction\_level': 'aggressive'  
 }  
  
 # Override defaults with provided config  
 if config:  
 self.config.update(config)  
  
 # Initialize rules engine  
 self.rules\_engine = RulesEngine(self.config)  
  
 def process\_ocr\_results(self, ocr\_data):  
 """  
 Process OCR results to improve quality and provide confidence metrics  
  
 Args:  
 ocr\_data: Dictionary containing OCR results  
  
 Returns:  
 Enhanced OCR results with confidence metrics  
 """  
 if not ocr\_data:  
 return {"error": "No OCR data provided"}  
  
 # Extract text elements from OCR data  
 text\_blocks = ocr\_data.get('text\_blocks', [])  
  
 # Process each text block  
 processed\_blocks = []  
 overall\_confidence = 0.0  
 total\_blocks = len(text\_blocks)  
  
 for block in text\_blocks:  
 # Apply post-processing if enabled  
 if self.config['post\_processing\_enabled']:  
 processed\_text = self.rules\_engine.apply\_rules(  
 block.get('text', ''),  
 block.get('type', 'unknown')  
 )  
 else:  
 processed\_text = block.get('text', '')  
  
 # Calculate confidence score  
 confidence = block.get('confidence', 0.0)  
  
 # Apply language model to refine confidence if enabled  
 if self.config['use\_language\_model'] and processed\_text:  
 language\_model\_confidence = self.\_apply\_language\_model(processed\_text)  
 # Weighted average of OCR and language model confidence  
 confidence = 0.7 \* confidence + 0.3 \* language\_model\_confidence  
  
 # Create processed block  
 processed\_block = {  
 'id': block.get('id', str(uuid.uuid4())),  
 'text': processed\_text,  
 'original\_text': block.get('text', ''),  
 'confidence': confidence,  
 'type': block.get('type', 'unknown'),  
 'bbox': block.get('bbox', None),  
 'requires\_review': confidence < self.config['min\_confidence'],  
 'corrections': self.\_get\_corrections(block.get('text', ''), processed\_text)  
 }  
  
 processed\_blocks.append(processed\_block)  
 overall\_confidence += confidence  
  
 # Calculate overall metrics  
 if total\_blocks > 0:  
 overall\_confidence /= total\_blocks  
  
 low\_confidence\_blocks = sum(1 for block in processed\_blocks if block['requires\_review'])  
  
 result = {  
 'blocks': processed\_blocks,  
 'metrics': {  
 'overall\_confidence': overall\_confidence,  
 'total\_blocks': total\_blocks,  
 'low\_confidence\_blocks': low\_confidence\_blocks,  
 'low\_confidence\_percentage': (low\_confidence\_blocks / total\_blocks \* 100) if total\_blocks > 0 else 0,  
 'requires\_review': low\_confidence\_blocks > 0,  
 'processing\_level': self.config['correction\_level']  
 }  
 }  
  
 return result  
  
 def \_apply\_language\_model(self, text):  
 """  
 Apply language model to assess text quality  
  
 Args:  
 text: Text to evaluate  
  
 Returns:  
 Confidence score from language model (0.0-1.0)  
 """  
 # In a real implementation, this would use a proper language model  
 # This is a simplified version based on heuristics  
  
 # Check for common OCR errors  
 common\_errors = [  
 'l' instead of '1',  
 'O' instead of '0',  
 'rn' instead of 'm',  
 'cl' instead of 'd'  
 ]  
  
 error\_count = sum(1 for error in common\_errors if error in text)  
  
 # Check for dictionary words  
 words = text.lower().split()  
 valid\_word\_ratio = 0.8 # Assume 80% valid as baseline  
  
 # Adjust confidence based on factors  
 base\_confidence = 0.7  
 confidence = base\_confidence - (error\_count \* 0.1)  
 confidence = max(0.1, min(0.95, confidence)) # Clamp to reasonable range  
  
 return confidence  
  
 def \_get\_corrections(self, original\_text, processed\_text):  
 """  
 Get list of corrections made to the text  
  
 Args:  
 original\_text: Original OCR text  
 processed\_text: Processed and corrected text  
  
 Returns:  
 List of corrections  
 """  
 if original\_text == processed\_text:  
 return []  
  
 # In a real implementation, we would use diff algorithms  
 # This is a simplified version  
 corrections = []  
  
 if original\_text != processed\_text:  
 corrections.append({  
 'type': 'text\_corrected',  
 'original': original\_text,  
 'corrected': processed\_text  
 })  
  
 return corrections

class OCRConfidenceScorer:  
 """Class for evaluating OCR quality and improving results"""  
  
 def \_\_init\_\_(self, config=None):  
 """  
 Initialize the OCR confidence scorer  
  
 Args:  
 config: Configuration dictionary  
 """  
 self.config = {  
 'min\_confidence': 0.5,  
 'post\_processing\_enabled': True,  
 'use\_language\_model': True,  
 'domain\_specific\_correction': True,  
 'correction\_level': 'aggressive'  
 }  
  
 # Override defaults with provided config  
 if config:  
 self.config.update(config)  
  
 # Initialize rules engine  
 self.rules\_engine = RulesEngine(self.config)  
  
 def process\_ocr\_results(self, ocr\_data):  
 """  
 Process OCR results to improve quality and provide confidence metrics  
  
 Args:  
 ocr\_data: Dictionary containing OCR results  
  
 Returns:  
 Enhanced OCR results with confidence metrics  
 """  
 if not ocr\_data:  
 return {"error": "No OCR data provided"}  
  
 # Extract text elements from OCR data  
 text\_blocks = ocr\_data.get('text\_blocks', [])  
  
 # Process each text block  
 processed\_blocks = []  
 overall\_confidence = 0.0  
 total\_blocks = len(text\_blocks)  
  
 for block in text\_blocks:  
 # Apply post-processing if enabled  
 if self.config['post\_processing\_enabled']:  
 processed\_text = self.rules\_engine.apply\_rules(  
 block.get('text', ''),  
 block.get('type', 'unknown')  
 )  
 else:  
 processed\_text = block.get('text', '')  
  
 # Calculate confidence score  
 confidence = block.get('confidence', 0.0)  
  
 # Apply language model to refine confidence if enabled  
 if self.config['use\_language\_model'] and processed\_text:  
 language\_model\_confidence = self.\_apply\_language\_model(processed\_text)  
 # Weighted average of OCR and language model confidence  
 confidence = 0.7 \* confidence + 0.3 \* language\_model\_confidence  
  
 # Create processed block  
 processed\_block = {  
 'id': block.get('id', str(uuid.uuid4())),  
 'text': processed\_text,  
 'original\_text': block.get('text', ''),  
 'confidence': confidence,  
 'type': block.get('type', 'unknown'),  
 'bbox': block.get('bbox', None),  
 'requires\_review': confidence < self.config['min\_confidence'],  
 'corrections': self.\_get\_corrections(block.get('text', ''), processed\_text)  
 }  
  
 processed\_blocks.append(processed\_block)  
 overall\_confidence += confidence  
  
 # Calculate overall metrics  
 if total\_blocks > 0:  
 overall\_confidence /= total\_blocks  
  
 low\_confidence\_blocks = sum(1 for block in processed\_blocks if block['requires\_review'])  
  
 result = {  
 'blocks': processed\_blocks,  
 'metrics': {  
 'overall\_confidence': overall\_confidence,  
 'total\_blocks': total\_blocks,  
 'low\_confidence\_blocks': low\_confidence\_blocks,  
 'low\_confidence\_percentage': (low\_confidence\_blocks / total\_blocks \* 100) if total\_blocks > 0 else 0,  
 'requires\_review': low\_confidence\_blocks > 0,  
 'processing\_level': self.config['correction\_level']  
 }  
 }  
  
 return result  
  
 def \_apply\_language\_model(self, text):  
 """  
 Apply language model to assess text quality  
  
 Args:  
 text: Text to evaluate  
  
 Returns:  
 Confidence score from language model (0.0-1.0)  
 """  
 # In a real implementation, this would use a proper language model  
 # This is a simplified version based on heuristics  
  
 # Check for common OCR errors  
 common\_errors = [  
 'l' instead of '1',  
 'O' instead of '0',  
 'rn' instead of 'm',  
 'cl' instead of 'd'  
 ]  
  
 error\_count = sum(1 for error in common\_errors if error in text)  
  
 # Check for dictionary words  
 words = text.lower().split()  
 valid\_word\_ratio = 0.8 # Assume 80% valid as baseline  
  
 # Adjust confidence based on factors  
 base\_confidence = 0.7  
 confidence = base\_confidence - (error\_count \* 0.1)  
 confidence = max(0.1, min(0.95, confidence)) # Clamp to reasonable range  
  
 return confidence  
  
 def \_get\_corrections(self, original\_text, processed\_text):  
 """  
 Get list of corrections made to the text  
  
 Args:  
 original\_text: Original OCR text  
 processed\_text: Processed and corrected text  
  
 Returns:  
 List of corrections  
 """  
 if original\_text == processed\_text:  
 return []  
  
 # In a real implementation, we would use diff algorithms  
 # This is a simplified version  
 corrections = []  
  
 if original\_text != processed\_text:  
 corrections.append({  
 'type': 'text\_corrected',  
 'original': original\_text,  
 'corrected': processed\_text  
 })  
  
 return corrections

#### Performance Monitoring

The system implements comprehensive performance tracking:

/\*\*  
 \* Get advanced queue metrics for dashboard  
 \*/  
export const getAdvancedQueueMetrics = async (): Promise<QueueMetrics> => {  
 try {  
 const token = getAuthToken();  
 const response = await axios.get(`${API\_BASE\_URL}/admin/queue/metrics`, {  
 headers: {  
 Authorization: `Bearer ${token}`  
 }  
 });  
  
 return response.data;  
 } catch (error) {  
 console.error('Failed to fetch advanced queue metrics:', error);  
 throw error;  
 }  
};  
  
/\*\*  
 \* Advanced queue metrics endpoint handler  
 \*/  
export const getQueueMetrics = async (req: Request, res: Response): Promise<void> => {  
 try {  
 // Get time range from query parameters  
 const timeRange = req.query.timeRange as string || 'day';  
  
 // Get metrics from the database  
 const metrics = await getMetricsFromDatabase(timeRange);  
  
 // Return metrics  
 res.json(metrics);  
 } catch (error) {  
 logger.error(`Error fetching queue metrics: ${error}`);  
 res.status(500).json({   
 error: 'Failed to retrieve queue metrics',   
 details: error instanceof Error ? error.message : String(error)   
 });  
 }  
};  
  
/\*\*  
 \* Get metrics from the database  
 \*/  
async function getMetricsFromDatabase(timeRange: string): Promise<QueueMetrics> {  
 // Time range boundaries  
 const now = new Date();  
 let startDate: Date;  
  
 switch (timeRange) {  
 case 'hour':  
 startDate = new Date(now.getTime() - 60 \* 60 \* 1000);  
 break;  
 case 'day':  
 startDate = new Date(now.getTime() - 24 \* 60 \* 60 \* 1000);  
 break;  
 case 'week':  
 startDate = new Date(now.getTime() - 7 \* 24 \* 60 \* 60 \* 1000);  
 break;  
 case 'month':  
 startDate = new Date(now.getTime() - 30 \* 24 \* 60 \* 60 \* 1000);  
 break;  
 default:  
 startDate = new Date(now.getTime() - 24 \* 60 \* 60 \* 1000);  
 }  
  
 // Get Supabase client  
 const client = supabaseClient.getClient();  
  
 // Query processing rates  
 const { data: processingRateData, error: processingRateError } = await client  
 .from('job\_metrics')  
 .select('system, completed\_at')  
 .gte('completed\_at', startDate.toISOString())  
 .eq('status', 'completed');  
  
 if (processingRateError) {  
 logger.error(`Error fetching processing rate: ${processingRateError}`);  
 throw processingRateError;  
 }  
  
 // Calculate processing rate (jobs per hour)  
 const pdfJobs = processingRateData.filter(job => job.system === 'pdf').length;  
 const crawlerJobs = processingRateData.filter(job => job.system === 'crawler').length;  
  
 const hoursInRange = (now.getTime() - startDate.getTime()) / (1000 \* 60 \* 60);  
  
 const processingRate = {  
 pdf: pdfJobs / hoursInRange,  
 crawler: crawlerJobs / hoursInRange  
 };  
  
 // Query completion rates  
 const { data: totalJobsData, error: totalJobsError } = await client  
 .from('job\_metrics')  
 .select('system, status')  
 .gte('created\_at', startDate.toISOString());  
  
 if (totalJobsError) {  
 logger.error(`Error fetching total jobs: ${totalJobsError}`);  
 throw totalJobsError;  
 }  
  
 // Calculate completion rates  
 const pdfTotal = totalJobsData.filter(job => job.system === 'pdf').length;  
 const pdfCompleted = totalJobsData.filter(job => job.system === 'pdf' && job.status === 'completed').length;  
  
 const crawlerTotal = totalJobsData.filter(job => job.system === 'crawler').length;  
 const crawlerCompleted = totalJobsData.filter(job => job.system === 'crawler' && job.status === 'completed').length;  
  
 const completionRate = {  
 pdf: pdfTotal > 0 ? pdfCompleted / pdfTotal : 0,  
 crawler: crawlerTotal > 0 ? crawlerCompleted / crawlerTotal : 0  
 };  
  
 // Query average processing times  
 const { data: processingTimeData, error: processingTimeError } = await client  
 .from('job\_metrics')  
 .select('system, processing\_time')  
 .gte('completed\_at', startDate.toISOString())  
 .eq('status', 'completed');  
  
 if (processingTimeError) {  
 logger.error(`Error fetching processing times: ${processingTimeError}`);  
 throw processingTimeError;  
 }  
  
 // Calculate average processing times  
 const pdfTimes = processingTimeData  
 .filter(job => job.system === 'pdf')  
 .map(job => job.processing\_time);  
  
 const crawlerTimes = processingTimeData  
 .filter(job => job.system === 'crawler')  
 .map(job => job.processing\_time);  
  
 const averageProcessingTime = {  
 pdf: pdfTimes.length > 0 ? pdfTimes.reduce((a, b) => a + b, 0) / pdfTimes.length : 0,  
 crawler: crawlerTimes.length > 0 ? crawlerTimes.reduce((a, b) => a + b, 0) / crawlerTimes.length : 0  
 };  
  
 // Return compiled metrics  
 return {  
 timeRange,  
 generatedAt: now.toISOString(),  
 processingRate,  
 completionRate,  
 averageProcessingTime,  
 // Add other metric categories as needed...  
 };  
}

/\*\*  
 \* Get advanced queue metrics for dashboard  
 \*/  
export const getAdvancedQueueMetrics = async (): Promise<QueueMetrics> => {  
 try {  
 const token = getAuthToken();  
 const response = await axios.get(`${API\_BASE\_URL}/admin/queue/metrics`, {  
 headers: {  
 Authorization: `Bearer ${token}`  
 }  
 });  
  
 return response.data;  
 } catch (error) {  
 console.error('Failed to fetch advanced queue metrics:', error);  
 throw error;  
 }  
};  
  
/\*\*  
 \* Advanced queue metrics endpoint handler  
 \*/  
export const getQueueMetrics = async (req: Request, res: Response): Promise<void> => {  
 try {  
 // Get time range from query parameters  
 const timeRange = req.query.timeRange as string || 'day';  
  
 // Get metrics from the database  
 const metrics = await getMetricsFromDatabase(timeRange);  
  
 // Return metrics  
 res.json(metrics);  
 } catch (error) {  
 logger.error(`Error fetching queue metrics: ${error}`);  
 res.status(500).json({   
 error: 'Failed to retrieve queue metrics',   
 details: error instanceof Error ? error.message : String(error)   
 });  
 }  
};  
  
/\*\*  
 \* Get metrics from the database  
 \*/  
async function getMetricsFromDatabase(timeRange: string): Promise<QueueMetrics> {  
 // Time range boundaries  
 const now = new Date();  
 let startDate: Date;  
  
 switch (timeRange) {  
 case 'hour':  
 startDate = new Date(now.getTime() - 60 \* 60 \* 1000);  
 break;  
 case 'day':  
 startDate = new Date(now.getTime() - 24 \* 60 \* 60 \* 1000);  
 break;  
 case 'week':  
 startDate = new Date(now.getTime() - 7 \* 24 \* 60 \* 60 \* 1000);  
 break;  
 case 'month':  
 startDate = new Date(now.getTime() - 30 \* 24 \* 60 \* 60 \* 1000);  
 break;  
 default:  
 startDate = new Date(now.getTime() - 24 \* 60 \* 60 \* 1000);  
 }  
  
 // Get Supabase client  
 const client = supabaseClient.getClient();  
  
 // Query processing rates  
 const { data: processingRateData, error: processingRateError } = await client  
 .from('job\_metrics')  
 .select('system, completed\_at')  
 .gte('completed\_at', startDate.toISOString())  
 .eq('status', 'completed');  
  
 if (processingRateError) {  
 logger.error(`Error fetching processing rate: ${processingRateError}`);  
 throw processingRateError;  
 }  
  
 // Calculate processing rate (jobs per hour)  
 const pdfJobs = processingRateData.filter(job => job.system === 'pdf').length;  
 const crawlerJobs = processingRateData.filter(job => job.system === 'crawler').length;  
  
 const hoursInRange = (now.getTime() - startDate.getTime()) / (1000 \* 60 \* 60);  
  
 const processingRate = {  
 pdf: pdfJobs / hoursInRange,  
 crawler: crawlerJobs / hoursInRange  
 };  
  
 // Query completion rates  
 const { data: totalJobsData, error: totalJobsError } = await client  
 .from('job\_metrics')  
 .select('system, status')  
 .gte('created\_at', startDate.toISOString());  
  
 if (totalJobsError) {  
 logger.error(`Error fetching total jobs: ${totalJobsError}`);  
 throw totalJobsError;  
 }  
  
 // Calculate completion rates  
 const pdfTotal = totalJobsData.filter(job => job.system === 'pdf').length;  
 const pdfCompleted = totalJobsData.filter(job => job.system === 'pdf' && job.status === 'completed').length;  
  
 const crawlerTotal = totalJobsData.filter(job => job.system === 'crawler').length;  
 const crawlerCompleted = totalJobsData.filter(job => job.system === 'crawler' && job.status === 'completed').length;  
  
 const completionRate = {  
 pdf: pdfTotal > 0 ? pdfCompleted / pdfTotal : 0,  
 crawler: crawlerTotal > 0 ? crawlerCompleted / crawlerTotal : 0  
 };  
  
 // Query average processing times  
 const { data: processingTimeData, error: processingTimeError } = await client  
 .from('job\_metrics')  
 .select('system, processing\_time')  
 .gte('completed\_at', startDate.toISOString())  
 .eq('status', 'completed');  
  
 if (processingTimeError) {  
 logger.error(`Error fetching processing times: ${processingTimeError}`);  
 throw processingTimeError;  
 }  
  
 // Calculate average processing times  
 const pdfTimes = processingTimeData  
 .filter(job => job.system === 'pdf')  
 .map(job => job.processing\_time);  
  
 const crawlerTimes = processingTimeData  
 .filter(job => job.system === 'crawler')  
 .map(job => job.processing\_time);  
  
 const averageProcessingTime = {  
 pdf: pdfTimes.length > 0 ? pdfTimes.reduce((a, b) => a + b, 0) / pdfTimes.length : 0,  
 crawler: crawlerTimes.length > 0 ? crawlerTimes.reduce((a, b) => a + b, 0) / crawlerTimes.length : 0  
 };  
  
 // Return compiled metrics  
 return {  
 timeRange,  
 generatedAt: now.toISOString(),  
 processingRate,  
 completionRate,  
 averageProcessingTime,  
 // Add other metric categories as needed...  
 };  
}

### Integration with Other Systems

#### Training Pipeline Integration

Data quality metrics integrate with the training pipeline:

1. Dataset Selection
2. Quality score-based dataset filtering
3. Class balance assessment for training suitability
4. Image quality requirements enforcement
5. Automated preprocessing for quality improvement
6. Versioning based on quality improvements
7. Hyperparameter Optimization
8. Performance tracking across parameter combinations
9. Learning curve analysis for convergence speed
10. Resource efficiency monitoring
11. Quality vs. speed trade-off assessment
12. Automated parameter recommendation
13. Progress Monitoring
14. Real-time training metrics visualization
15. Learning curve tracking and analysis
16. Validation performance monitoring
17. Early stopping based on quality metrics
18. Resource utilization tracking
19. Model Evaluation
20. Quality-based model comparison
21. Confidence calibration assessment
22. Error analysis and categorization
23. Out-of-distribution performance
24. Version comparison and improvement tracking

Versioning based on quality improvements

Hyperparameter Optimization

Automated parameter recommendation

Progress Monitoring

Resource utilization tracking

Model Evaluation

#### OCR Process Integration

Quality metrics enhance OCR processing:

1. Quality-Based Workflow
2. Confidence thresholds for automated processing
3. Low-quality detection triggering enhanced processing
4. Manual review routing for uncertain content
5. Quality-based prioritization
6. Feedback loop for continuous improvement
7. Adaptive Processing
8. Image quality detection for preprocessing selection
9. Engine and parameter selection based on content type
10. Language detection for optimal recognition
11. Resource allocation based on complexity
12. Error recovery strategy selection
13. Results Enhancement
14. Confidence-based post-processing
15. Alternative interpretation suggestions
16. Context-based correction
17. Domain-specific validation
18. Format-specific normalization
19. Integration with Knowledge Base
20. Confidence-based knowledge integration
21. Uncertainty highlighting for verification
22. Quality metrics for trust assessment
23. Version tracking based on quality improvements
24. Validation against existing knowledge

Feedback loop for continuous improvement

Adaptive Processing

Error recovery strategy selection

Results Enhancement

Format-specific normalization

Integration with Knowledge Base

#### Admin Dashboard Integration

Quality metrics are exposed through the admin interface:

1. Quality Monitoring Dashboards
2. Dataset quality overview
3. Recognition performance tracking
4. OCR quality monitoring
5. Training progress visualization
6. System health indicators
7. Alert Mechanisms
8. Quality threshold breach notifications
9. Unusual pattern detection
10. Performance degradation alerts
11. Resource utilization warnings
12. Error rate monitoring
13. Quality Management Tools
14. Manual review interfaces for low-confidence content
15. Quality improvement recommendation implementation
16. Dataset enhancement tools
17. Model retraining triggers
18. System optimization controls
19. Reporting Capabilities
20. Quality trend analysis
21. Performance comparison across versions
22. Resource utilization reporting
23. Error pattern identification
24. Improvement tracking over time

System health indicators

Alert Mechanisms

Error rate monitoring

Quality Management Tools

System optimization controls

Reporting Capabilities

### Usage Examples

#### Analyzing Dataset Quality

import { datasetManagementService } from '@kai/server/services/datasets/dataset-management.service';  
  
async function assessDatasetQuality(datasetId: string) {  
 try {  
 console.log(`Analyzing quality of dataset ${datasetId}...`);  
  
 // Get dataset quality metrics  
 const metrics = await datasetManagementService.analyzeDatasetQuality(datasetId);  
  
 // Log overall quality score  
 console.log(`Overall quality score: ${metrics.overallQualityScore}/100`);  
  
 // Show component scores  
 console.log(`Class balance score: ${metrics.classBalance.score}/100`);  
 console.log(`Image quality score: ${metrics.imageQuality.score}/100`);  
  
 // Display class distribution  
 console.log('Class distribution:');  
 Object.entries(metrics.classBalance.details.classCounts).forEach(([className, count]) => {  
 console.log(` ${className}: ${count} images`);  
 });  
  
 // Show class imbalance metrics  
 console.log(`Class imbalance (max/min ratio): ${metrics.classBalance.details.maxToMinRatio.toFixed(2)}`);  
 console.log(`Gini coefficient: ${metrics.classBalance.details.giniCoefficient.toFixed(4)}`);  
  
 // Display image quality issues  
 console.log('Image quality issues:');  
 console.log(` Low resolution images: ${metrics.imageQuality.details.lowResolutionImages}`);  
 console.log(` Blurry images: ${metrics.imageQuality.details.blurryImages}`);  
 console.log(` Poor lighting: ${metrics.imageQuality.details.poorLightingImages}`);  
 console.log(` Overexposed: ${metrics.imageQuality.details.overexposedImages}`);  
  
 // Show average resolution  
 console.log(`Average resolution: ${metrics.imageQuality.details.avgResolution.width}x${metrics.imageQuality.details.avgResolution.height}`);  
  
 // Display recommendations  
 console.log('Recommendations:');  
 metrics.recommendations.forEach((recommendation, index) => {  
 console.log(` ${index + 1}. ${recommendation}`);  
 });  
  
 // Determine if dataset needs improvement  
 if (metrics.overallQualityScore < 70) {  
 console.log('Dataset quality is below recommended threshold. Consider implementing the recommendations before using for training.');  
 } else {  
 console.log('Dataset quality is acceptable for training.');  
 }  
  
 return metrics;  
 } catch (error) {  
 console.error('Dataset quality analysis failed:', error);  
 throw error;  
 }  
}

import { datasetManagementService } from '@kai/server/services/datasets/dataset-management.service';  
  
async function assessDatasetQuality(datasetId: string) {  
 try {  
 console.log(`Analyzing quality of dataset ${datasetId}...`);  
  
 // Get dataset quality metrics  
 const metrics = await datasetManagementService.analyzeDatasetQuality(datasetId);  
  
 // Log overall quality score  
 console.log(`Overall quality score: ${metrics.overallQualityScore}/100`);  
  
 // Show component scores  
 console.log(`Class balance score: ${metrics.classBalance.score}/100`);  
 console.log(`Image quality score: ${metrics.imageQuality.score}/100`);  
  
 // Display class distribution  
 console.log('Class distribution:');  
 Object.entries(metrics.classBalance.details.classCounts).forEach(([className, count]) => {  
 console.log(` ${className}: ${count} images`);  
 });  
  
 // Show class imbalance metrics  
 console.log(`Class imbalance (max/min ratio): ${metrics.classBalance.details.maxToMinRatio.toFixed(2)}`);  
 console.log(`Gini coefficient: ${metrics.classBalance.details.giniCoefficient.toFixed(4)}`);  
  
 // Display image quality issues  
 console.log('Image quality issues:');  
 console.log(` Low resolution images: ${metrics.imageQuality.details.lowResolutionImages}`);  
 console.log(` Blurry images: ${metrics.imageQuality.details.blurryImages}`);  
 console.log(` Poor lighting: ${metrics.imageQuality.details.poorLightingImages}`);  
 console.log(` Overexposed: ${metrics.imageQuality.details.overexposedImages}`);  
  
 // Show average resolution  
 console.log(`Average resolution: ${metrics.imageQuality.details.avgResolution.width}x${metrics.imageQuality.details.avgResolution.height}`);  
  
 // Display recommendations  
 console.log('Recommendations:');  
 metrics.recommendations.forEach((recommendation, index) => {  
 console.log(` ${index + 1}. ${recommendation}`);  
 });  
  
 // Determine if dataset needs improvement  
 if (metrics.overallQualityScore < 70) {  
 console.log('Dataset quality is below recommended threshold. Consider implementing the recommendations before using for training.');  
 } else {  
 console.log('Dataset quality is acceptable for training.');  
 }  
  
 return metrics;  
 } catch (error) {  
 console.error('Dataset quality analysis failed:', error);  
 throw error;  
 }  
}

#### Tracking Training Metrics

from training\_visualization import MetricsTracker, ProgressVisualizer  
import time  
import random  
import matplotlib.pyplot as plt  
import os  
  
def simulate\_training\_with\_metrics():  
 # Create metrics tracker and visualizer  
 job\_id = f"training\_job\_{int(time.time())}"  
 output\_dir = os.path.join("output", "training\_metrics")  
 os.makedirs(output\_dir, exist\_ok=True)  
  
 tracker = MetricsTracker(job\_id, metrics\_dir=output\_dir)  
 visualizer = ProgressVisualizer(tracker, output\_dir)  
  
 # Simulate training for 10 epochs  
 for epoch in range(10):  
 print(f"Training epoch {epoch+1}/10")  
  
 # Simulate batch iterations with metrics  
 for batch in range(50):  
 # Simulate decreasing loss and increasing accuracy  
 train\_loss = 1.0 / (1.0 + 0.1 \* epoch + 0.005 \* batch)  
 train\_acc = 0.5 + 0.05 \* epoch + 0.001 \* batch  
  
 # Add random noise to make it more realistic  
 train\_loss += random.uniform(-0.05, 0.05)  
 train\_acc += random.uniform(-0.02, 0.02)  
  
 # Ensure values are in reasonable ranges  
 train\_loss = max(0.001, train\_loss)  
 train\_acc = min(max(0.0, train\_acc), 1.0)  
  
 # Add training metrics  
 tracker.add\_metrics\_dict({  
 "loss": train\_loss,  
 "accuracy": train\_acc  
 }, epoch=epoch, iteration=batch)  
  
 # Every 10 batches, simulate validation  
 if batch % 10 == 0:  
 val\_loss = train\_loss \* random.uniform(0.9, 1.1)  
 val\_acc = train\_acc \* random.uniform(0.9, 1.1)  
  
 tracker.add\_metrics\_dict({  
 "val\_loss": val\_loss,  
 "val\_accuracy": val\_acc  
 }, epoch=epoch)  
  
 # At the end of each epoch, visualize the learning curves  
 visualizer.plot\_learning\_curves()  
  
 # Simulate progress monitoring  
 time.sleep(0.5) # Just to slow down simulation  
  
 # Generate final visualization  
 fig = visualizer.plot\_learning\_curves(figsize=(12, 8))  
 fig.savefig(os.path.join(output\_dir, f"{job\_id}\_learning\_curves.png"))  
  
 # Get final metrics  
 final\_metrics = tracker.get\_latest\_metrics()  
 print("\nFinal metrics:")  
 for name, value in final\_metrics.items():  
 print(f" {name}: {value:.4f}")  
  
 return {  
 "job\_id": job\_id,  
 "output\_dir": output\_dir,  
 "final\_metrics": final\_metrics  
 }

from training\_visualization import MetricsTracker, ProgressVisualizer  
import time  
import random  
import matplotlib.pyplot as plt  
import os  
  
def simulate\_training\_with\_metrics():  
 # Create metrics tracker and visualizer  
 job\_id = f"training\_job\_{int(time.time())}"  
 output\_dir = os.path.join("output", "training\_metrics")  
 os.makedirs(output\_dir, exist\_ok=True)  
  
 tracker = MetricsTracker(job\_id, metrics\_dir=output\_dir)  
 visualizer = ProgressVisualizer(tracker, output\_dir)  
  
 # Simulate training for 10 epochs  
 for epoch in range(10):  
 print(f"Training epoch {epoch+1}/10")  
  
 # Simulate batch iterations with metrics  
 for batch in range(50):  
 # Simulate decreasing loss and increasing accuracy  
 train\_loss = 1.0 / (1.0 + 0.1 \* epoch + 0.005 \* batch)  
 train\_acc = 0.5 + 0.05 \* epoch + 0.001 \* batch  
  
 # Add random noise to make it more realistic  
 train\_loss += random.uniform(-0.05, 0.05)  
 train\_acc += random.uniform(-0.02, 0.02)  
  
 # Ensure values are in reasonable ranges  
 train\_loss = max(0.001, train\_loss)  
 train\_acc = min(max(0.0, train\_acc), 1.0)  
  
 # Add training metrics  
 tracker.add\_metrics\_dict({  
 "loss": train\_loss,  
 "accuracy": train\_acc  
 }, epoch=epoch, iteration=batch)  
  
 # Every 10 batches, simulate validation  
 if batch % 10 == 0:  
 val\_loss = train\_loss \* random.uniform(0.9, 1.1)  
 val\_acc = train\_acc \* random.uniform(0.9, 1.1)  
  
 tracker.add\_metrics\_dict({  
 "val\_loss": val\_loss,  
 "val\_accuracy": val\_acc  
 }, epoch=epoch)  
  
 # At the end of each epoch, visualize the learning curves  
 visualizer.plot\_learning\_curves()  
  
 # Simulate progress monitoring  
 time.sleep(0.5) # Just to slow down simulation  
  
 # Generate final visualization  
 fig = visualizer.plot\_learning\_curves(figsize=(12, 8))  
 fig.savefig(os.path.join(output\_dir, f"{job\_id}\_learning\_curves.png"))  
  
 # Get final metrics  
 final\_metrics = tracker.get\_latest\_metrics()  
 print("\nFinal metrics:")  
 for name, value in final\_metrics.items():  
 print(f" {name}: {value:.4f}")  
  
 return {  
 "job\_id": job\_id,  
 "output\_dir": output\_dir,  
 "final\_metrics": final\_metrics  
 }

#### OCR Quality Assessment

from enhanced\_ocr import EnhancedOCRProcessor  
from ocr\_confidence\_scoring import OCRConfidenceScorer  
  
def assess\_ocr\_quality(image\_path):  
 # Initialize OCR processor  
 ocr\_processor = EnhancedOCRProcessor(  
 languages=['eng'],  
 material\_type='tile',  
 enable\_layout\_analysis=True,  
 enable\_specialized\_ocr=True,  
 confidence\_threshold=0  
 )  
  
 # Initialize confidence scorer  
 confidence\_scorer = OCRConfidenceScorer({  
 'min\_confidence': 0.7,  
 'post\_processing\_enabled': True,  
 'use\_language\_model': True,  
 'domain\_specific\_correction': True,  
 'correction\_level': 'standard'  
 })  
  
 # Process image with OCR  
 print(f"Processing image: {image\_path}")  
 ocr\_results = ocr\_processor.process\_image(image\_path)  
  
 # Score OCR results  
 print("Assessing OCR quality...")  
 quality\_assessment = confidence\_scorer.process\_ocr\_results(ocr\_results)  
  
 # Analyze and report quality metrics  
 metrics = quality\_assessment['metrics']  
  
 print("\nOCR Quality Assessment:")  
 print(f"Overall confidence: {metrics['overall\_confidence']:.2f}")  
 print(f"Total text blocks: {metrics['total\_blocks']}")  
 print(f"Low confidence blocks: {metrics['low\_confidence\_blocks']} ({metrics['low\_confidence\_percentage']:.1f}%)")  
  
 print("\nDetailed block analysis:")  
 for i, block in enumerate(quality\_assessment['blocks']):  
 confidence\_indicator = "✓" if block['confidence'] >= 0.7 else "✗"  
 print(f"Block {i+1} ({block['type']}): {confidence\_indicator} {block['confidence']:.2f}")  
  
 if block['requires\_review']:  
 print(f" [REVIEW NEEDED] {block['text']}")  
  
 if block['corrections']:  
 print(" Corrections:")  
 for correction in block['corrections']:  
 print(f" Original: {correction['original']}")  
 print(f" Corrected: {correction['corrected']}")  
  
 # Determine if manual review is necessary  
 if metrics['requires\_review']:  
 print("\n⚠️ Manual review recommended for some text blocks")  
 else:  
 print("\n✅ OCR quality is acceptable, no manual review needed")  
  
 return quality\_assessment

from enhanced\_ocr import EnhancedOCRProcessor  
from ocr\_confidence\_scoring import OCRConfidenceScorer  
  
def assess\_ocr\_quality(image\_path):  
 # Initialize OCR processor  
 ocr\_processor = EnhancedOCRProcessor(  
 languages=['eng'],  
 material\_type='tile',  
 enable\_layout\_analysis=True,  
 enable\_specialized\_ocr=True,  
 confidence\_threshold=0  
 )  
  
 # Initialize confidence scorer  
 confidence\_scorer = OCRConfidenceScorer({  
 'min\_confidence': 0.7,  
 'post\_processing\_enabled': True,  
 'use\_language\_model': True,  
 'domain\_specific\_correction': True,  
 'correction\_level': 'standard'  
 })  
  
 # Process image with OCR  
 print(f"Processing image: {image\_path}")  
 ocr\_results = ocr\_processor.process\_image(image\_path)  
  
 # Score OCR results  
 print("Assessing OCR quality...")  
 quality\_assessment = confidence\_scorer.process\_ocr\_results(ocr\_results)  
  
 # Analyze and report quality metrics  
 metrics = quality\_assessment['metrics']  
  
 print("\nOCR Quality Assessment:")  
 print(f"Overall confidence: {metrics['overall\_confidence']:.2f}")  
 print(f"Total text blocks: {metrics['total\_blocks']}")  
 print(f"Low confidence blocks: {metrics['low\_confidence\_blocks']} ({metrics['low\_confidence\_percentage']:.1f}%)")  
  
 print("\nDetailed block analysis:")  
 for i, block in enumerate(quality\_assessment['blocks']):  
 confidence\_indicator = "✓" if block['confidence'] >= 0.7 else "✗"  
 print(f"Block {i+1} ({block['type']}): {confidence\_indicator} {block['confidence']:.2f}")  
  
 if block['requires\_review']:  
 print(f" [REVIEW NEEDED] {block['text']}")  
  
 if block['corrections']:  
 print(" Corrections:")  
 for correction in block['corrections']:  
 print(f" Original: {correction['original']}")  
 print(f" Corrected: {correction['corrected']}")  
  
 # Determine if manual review is necessary  
 if metrics['requires\_review']:  
 print("\n⚠️ Manual review recommended for some text blocks")  
 else:  
 print("\n✅ OCR quality is acceptable, no manual review needed")  
  
 return quality\_assessment

#### Performance Monitoring Dashboard

import { useEffect, useState } from 'react';  
import { getAdvancedQueueMetrics } from '@kai/client/services/queue.service';  
import {   
 LineChart, BarChart, PieChart,   
 Card, Tabs, Alert, Select  
} from '@kai/client/components/ui';  
  
export default function PerformanceMonitoringDashboard() {  
 // State for metrics and UI  
 const [metrics, setMetrics] = useState<any | null>(null);  
 const [timeRange, setTimeRange] = useState<string>('day');  
 const [error, setError] = useState<string | null>(null);  
 const [loading, setLoading] = useState<boolean>(true);  
  
 // Fetch metrics on mount and when timeRange changes  
 useEffect(() => {  
 async function fetchMetrics() {  
 try {  
 setLoading(true);  
 const data = await getAdvancedQueueMetrics(timeRange);  
 setMetrics(data);  
 setError(null);  
 } catch (err) {  
 setError(`Failed to load metrics: ${err instanceof Error ? err.message : String(err)}`);  
 console.error('Error loading metrics:', err);  
 } finally {  
 setLoading(false);  
 }  
 }  
  
 fetchMetrics();  
  
 // Set up polling interval for real-time updates  
 const interval = setInterval(fetchMetrics, 60000); // Update every minute  
  
 return () => clearInterval(interval);  
 }, [timeRange]);  
  
 // Time range options  
 const timeRangeOptions = [  
 { value: 'hour', label: 'Last Hour' },  
 { value: 'day', label: 'Last 24 Hours' },  
 { value: 'week', label: 'Last 7 Days' },  
 { value: 'month', label: 'Last 30 Days' }  
 ];  
  
 // Format metrics for charts  
 const formatQueueData = () => {  
 if (!metrics) return null;  
  
 return {  
 labels: ['PDF Processing', 'Web Crawler'],  
 datasets: [  
 {  
 label: 'Processing Rate (jobs/hour)',  
 data: [metrics.processingRate.pdf, metrics.processingRate.crawler],  
 backgroundColor: ['rgba(54, 162, 235, 0.6)', 'rgba(255, 99, 132, 0.6)']  
 }  
 ]  
 };  
 };  
  
 const formatCompletionRateData = () => {  
 if (!metrics) return null;  
  
 return {  
 labels: ['PDF Processing', 'Web Crawler'],  
 datasets: [  
 {  
 label: 'Completion Rate (%)',  
 data: [  
 metrics.completionRate.pdf \* 100,   
 metrics.completionRate.crawler \* 100  
 ],  
 backgroundColor: ['rgba(75, 192, 192, 0.6)', 'rgba(153, 102, 255, 0.6)']  
 }  
 ]  
 };  
 };  
  
 return (  
 <div className="p-6">  
 <div className="mb-6">  
 <h1 className="text-2xl font-semibold text-gray-800">Performance Monitoring</h1>  
 <p className="text-gray-600">Real-time metrics and system performance analytics</p>  
 </div>  
  
 {/\* Time range selector \*/}  
 <div className="mb-6">  
 <Select  
 label="Time Range"  
 value={timeRange}  
 onChange={setTimeRange}  
 options={timeRangeOptions}  
 />  
 </div>  
  
 {/\* Error display \*/}  
 {error && (  
 <Alert type="error" title="Error Loading Metrics" message={error} className="mb-6" />  
 )}  
  
 {/\* Loading state \*/}  
 {loading ? (  
 <div className="flex justify-center items-center h-64">  
 <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-blue-500"></div>  
 </div>  
 ) : !metrics ? (  
 <Alert type="info" title="No Data" message="No metrics available for the selected time range." />  
 ) : (  
 <div>  
 {/\* Metrics overview \*/}  
 <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 mb-8">  
 <Card title="Total PDF Jobs" value={metrics.jobCounts?.pdf || 0} />  
 <Card title="Total Crawler Jobs" value={metrics.jobCounts?.crawler || 0} />  
 <Card   
 title="Avg PDF Processing Time"   
 value={`${(metrics.averageProcessingTime.pdf || 0).toFixed(2)}s`}   
 />  
 <Card   
 title="Avg Crawler Processing Time"   
 value={`${(metrics.averageProcessingTime.crawler || 0).toFixed(2)}s`}   
 />  
 </div>  
  
 {/\* Charts \*/}  
 <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 mb-8">  
 <div className="bg-white p-6 rounded-lg shadow-sm">  
 <h2 className="text-lg font-semibold mb-4">Processing Rate</h2>  
 <BarChart data={formatQueueData()} height={300} />  
 </div>  
  
 <div className="bg-white p-6 rounded-lg shadow-sm">  
 <h2 className="text-lg font-semibold mb-4">Completion Rate</h2>  
 <BarChart data={formatCompletionRateData()} height={300} />  
 </div>  
 </div>  
  
 {/\* Detailed metrics tabs \*/}  
 <div className="bg-white p-6 rounded-lg shadow-sm">  
 <Tabs  
 tabs={[  
 {   
 label: 'Queue Metrics',   
 content: <QueueMetricsPanel metrics={metrics} />   
 },  
 {   
 label: 'System Health',   
 content: <SystemHealthPanel metrics={metrics} />   
 },  
 {   
 label: 'Training Metrics',   
 content: <TrainingMetricsPanel metrics={metrics} />   
 }  
 ]}  
 />  
 </div>  
 </div>  
 )}  
 </div>  
 );  
}

import { useEffect, useState } from 'react';  
import { getAdvancedQueueMetrics } from '@kai/client/services/queue.service';  
import {   
 LineChart, BarChart, PieChart,   
 Card, Tabs, Alert, Select  
} from '@kai/client/components/ui';  
  
export default function PerformanceMonitoringDashboard() {  
 // State for metrics and UI  
 const [metrics, setMetrics] = useState<any | null>(null);  
 const [timeRange, setTimeRange] = useState<string>('day');  
 const [error, setError] = useState<string | null>(null);  
 const [loading, setLoading] = useState<boolean>(true);  
  
 // Fetch metrics on mount and when timeRange changes  
 useEffect(() => {  
 async function fetchMetrics() {  
 try {  
 setLoading(true);  
 const data = await getAdvancedQueueMetrics(timeRange);  
 setMetrics(data);  
 setError(null);  
 } catch (err) {  
 setError(`Failed to load metrics: ${err instanceof Error ? err.message : String(err)}`);  
 console.error('Error loading metrics:', err);  
 } finally {  
 setLoading(false);  
 }  
 }  
  
 fetchMetrics();  
  
 // Set up polling interval for real-time updates  
 const interval = setInterval(fetchMetrics, 60000); // Update every minute  
  
 return () => clearInterval(interval);  
 }, [timeRange]);  
  
 // Time range options  
 const timeRangeOptions = [  
 { value: 'hour', label: 'Last Hour' },  
 { value: 'day', label: 'Last 24 Hours' },  
 { value: 'week', label: 'Last 7 Days' },  
 { value: 'month', label: 'Last 30 Days' }  
 ];  
  
 // Format metrics for charts  
 const formatQueueData = () => {  
 if (!metrics) return null;  
  
 return {  
 labels: ['PDF Processing', 'Web Crawler'],  
 datasets: [  
 {  
 label: 'Processing Rate (jobs/hour)',  
 data: [metrics.processingRate.pdf, metrics.processingRate.crawler],  
 backgroundColor: ['rgba(54, 162, 235, 0.6)', 'rgba(255, 99, 132, 0.6)']  
 }  
 ]  
 };  
 };  
  
 const formatCompletionRateData = () => {  
 if (!metrics) return null;  
  
 return {  
 labels: ['PDF Processing', 'Web Crawler'],  
 datasets: [  
 {  
 label: 'Completion Rate (%)',  
 data: [  
 metrics.completionRate.pdf \* 100,   
 metrics.completionRate.crawler \* 100  
 ],  
 backgroundColor: ['rgba(75, 192, 192, 0.6)', 'rgba(153, 102, 255, 0.6)']  
 }  
 ]  
 };  
 };  
  
 return (  
 <div className="p-6">  
 <div className="mb-6">  
 <h1 className="text-2xl font-semibold text-gray-800">Performance Monitoring</h1>  
 <p className="text-gray-600">Real-time metrics and system performance analytics</p>  
 </div>  
  
 {/\* Time range selector \*/}  
 <div className="mb-6">  
 <Select  
 label="Time Range"  
 value={timeRange}  
 onChange={setTimeRange}  
 options={timeRangeOptions}  
 />  
 </div>  
  
 {/\* Error display \*/}  
 {error && (  
 <Alert type="error" title="Error Loading Metrics" message={error} className="mb-6" />  
 )}  
  
 {/\* Loading state \*/}  
 {loading ? (  
 <div className="flex justify-center items-center h-64">  
 <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-blue-500"></div>  
 </div>  
 ) : !metrics ? (  
 <Alert type="info" title="No Data" message="No metrics available for the selected time range." />  
 ) : (  
 <div>  
 {/\* Metrics overview \*/}  
 <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 mb-8">  
 <Card title="Total PDF Jobs" value={metrics.jobCounts?.pdf || 0} />  
 <Card title="Total Crawler Jobs" value={metrics.jobCounts?.crawler || 0} />  
 <Card   
 title="Avg PDF Processing Time"   
 value={`${(metrics.averageProcessingTime.pdf || 0).toFixed(2)}s`}   
 />  
 <Card   
 title="Avg Crawler Processing Time"   
 value={`${(metrics.averageProcessingTime.crawler || 0).toFixed(2)}s`}   
 />  
 </div>  
  
 {/\* Charts \*/}  
 <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 mb-8">  
 <div className="bg-white p-6 rounded-lg shadow-sm">  
 <h2 className="text-lg font-semibold mb-4">Processing Rate</h2>  
 <BarChart data={formatQueueData()} height={300} />  
 </div>  
  
 <div className="bg-white p-6 rounded-lg shadow-sm">  
 <h2 className="text-lg font-semibold mb-4">Completion Rate</h2>  
 <BarChart data={formatCompletionRateData()} height={300} />  
 </div>  
 </div>  
  
 {/\* Detailed metrics tabs \*/}  
 <div className="bg-white p-6 rounded-lg shadow-sm">  
 <Tabs  
 tabs={[  
 {   
 label: 'Queue Metrics',   
 content: <QueueMetricsPanel metrics={metrics} />   
 },  
 {   
 label: 'System Health',   
 content: <SystemHealthPanel metrics={metrics} />   
 },  
 {   
 label: 'Training Metrics',   
 content: <TrainingMetricsPanel metrics={metrics} />   
 }  
 ]}  
 />  
 </div>  
 </div>  
 )}  
 </div>  
 );  
}

### Best Practices and Recommendations

#### Data Quality Improvement

To maximize data quality:

1. Dataset Preparation Guidelines
2. Maintain class balance within 3:1 ratio
3. Ensure minimum image resolution of 224x224 pixels
4. Check for and correct lighting and focus issues
5. Remove duplicate or near-duplicate images
6. Validate all metadata for consistency
7. Augmentation Strategies
8. Use augmentation for underrepresented classes
9. Apply realistic transformations only
10. Avoid distortions that obscure key features
11. Validate augmented samples for quality
12. Document augmentation parameters
13. Quality Verification Workflow
14. Implement pre-training quality assessment
15. Set minimum quality thresholds for training
16. Create review processes for low-quality data
17. Document quality issues and resolutions
18. Maintain quality history for tracking
19. Continuous Improvement
20. Analyze error patterns to identify data issues
21. Prioritize quality improvements by impact
22. Implement regular quality audits
23. Track quality metrics over time
24. Use A/B testing to validate improvements

Validate all metadata for consistency

Augmentation Strategies

Document augmentation parameters

Quality Verification Workflow

Maintain quality history for tracking

Continuous Improvement

#### Metrics Collection Strategy

For effective metrics collection:

1. Key Performance Indicators
2. Define critical metrics for each subsystem
3. Establish baseline performance standards
4. Set alerting thresholds for deviations
5. Document metric interpretation guidelines
6. Implement regular review processes
7. Collection Frequency
8. Match frequency to metric volatility
9. Collect real-time metrics for critical operations
10. Implement batch collection for historical analysis
11. Balance detail against storage requirements
12. Consider time zone handling for consistency
13. Storage Optimization
14. Implement data aggregation strategies
15. Define retention policies by importance
16. Use efficient storage formats
17. Consider downsampling for historical data
18. Implement compression for large datasets
19. Accessibility and Visualization
20. Design intuitive dashboards for key stakeholders
21. Provide drill-down capabilities for investigation
22. Implement export options for analysis
23. Document metric definitions and contexts
24. Provide comparison against historical trends

Implement regular review processes

Collection Frequency

Consider time zone handling for consistency

Storage Optimization

Implement compression for large datasets

Accessibility and Visualization

#### Monitoring Best Practices

For effective system monitoring:

1. Alert Configuration
2. Define clear alert severity levels
3. Implement progressive alerting thresholds
4. Establish notification routing by severity
5. Configure alert aggregation to prevent floods
6. Document response procedures by alert type
7. Operational Visibility
8. Provide real-time system status dashboards
9. Implement service health indicators
10. Create resource utilization visualizations
11. Design job status monitoring interfaces
12. Maintain historical performance views
13. Troubleshooting Support
14. Implement correlated log access
15. Provide context-aware error information
16. Design transaction tracing capabilities
17. Create system state inspection tools
18. Document common issue resolution steps
19. Capacity Planning
20. Track resource utilization trends
21. Identify growth patterns and seasonality
22. Establish utilization thresholds for scaling
23. Model future resource requirements
24. Document scaling procedures and triggers

Document response procedures by alert type

Operational Visibility

Maintain historical performance views

Troubleshooting Support

Document common issue resolution steps

Capacity Planning

# Queue System

Source: readme/queue-system.md

---

## Queue System

The Queue System is a critical component of Kai that manages asynchronous processing tasks using Supabase Realtime for message brokering. This document details the architecture, components, and usage of the queue system.

### Overview

The Queue System provides:

1. Asynchronous Processing: Handle time-consuming tasks without blocking user operations
2. Prioritization: Process critical tasks ahead of less important ones
3. Resource Management: Control resource allocation for different processing tasks
4. Status Tracking: Monitor job progress and status in real time
5. Error Handling: Robust error recovery and retry mechanisms
6. Cross-Service Communication: Coordinate workflows across different services

### Architecture

The Queue System uses a message broker architecture with Supabase Realtime:

┌───────────────┐ ┌───────────────────┐ ┌───────────────┐  
│ │ │ │ │ │  
│ Client Apps │────▶│ Message Broker │────▶│ PDF Queue │  
│ │ │ (Supabase) │ │ │  
└───────────────┘ └───────────────────┘ └───────────────┘  
 │ │ │  
 │ │ ▼  
 │ │ ┌───────────────┐  
 │ │ │ │  
 ▼ ▼ │ PDF Worker │  
┌───────────────┐ ┌───────────────────┐ │ │  
│ │ │ │ └───────────────┘  
│ Admin Panel │────▶│ Queue Events │ │  
│ │ │ │ │  
└───────────────┘ └───────────────────┘ ▼  
 │ ┌───────────────┐  
 │ │ │  
 ▼ │ Knowledge │  
 ┌───────────────────┐ │ Base Import │  
 │ │ │ │  
 │ Crawler Queue │ └───────────────┘  
 │ │  
 └───────────────────┘  
 │  
 ▼  
 ┌───────────────────┐  
 │ │  
 │ Crawler Worker │  
 │ │  
 └───────────────────┘

┌───────────────┐ ┌───────────────────┐ ┌───────────────┐  
│ │ │ │ │ │  
│ Client Apps │────▶│ Message Broker │────▶│ PDF Queue │  
│ │ │ (Supabase) │ │ │  
└───────────────┘ └───────────────────┘ └───────────────┘  
 │ │ │  
 │ │ ▼  
 │ │ ┌───────────────┐  
 │ │ │ │  
 ▼ ▼ │ PDF Worker │  
┌───────────────┐ ┌───────────────────┐ │ │  
│ │ │ │ └───────────────┘  
│ Admin Panel │────▶│ Queue Events │ │  
│ │ │ │ │  
└───────────────┘ └───────────────────┘ ▼  
 │ ┌───────────────┐  
 │ │ │  
 ▼ │ Knowledge │  
 ┌───────────────────┐ │ Base Import │  
 │ │ │ │  
 │ Crawler Queue │ └───────────────┘  
 │ │  
 └───────────────────┘  
 │  
 ▼  
 ┌───────────────────┐  
 │ │  
 │ Crawler Worker │  
 │ │  
 └───────────────────┘

#### Core Components

1. Supabase Client
2. Manages connections to Supabase Realtime
3. Handles authentication and authorization
4. Provides reconnection and error handling
5. Manages subscription lifecycle
6. Message Broker
7. Core routing component for pub/sub messaging
8. Handles message delivery and confirmation
9. Supports multiple channels for different message types
10. Provides delivery guarantees
11. Queue Adapters
12. Standardize interface between queues and message broker
13. Convert queue-specific events to standard message format
14. Handle queue-specific processing requirements
15. Implement queue persistence and state management
16. Queue Workers
17. Process jobs from specific queues
18. Report progress and status updates
19. Handle resource allocation
20. Implement error recovery
21. Event Handlers
22. Process events from other queues
23. Trigger dependent workflows
24. Update system state based on events
25. Propagate event notifications

Manages subscription lifecycle

Message Broker

Provides delivery guarantees

Queue Adapters

Implement queue persistence and state management

Queue Workers

Implement error recovery

Event Handlers

### Queue Types

The system includes several specialized queues:

#### PDF Processing Queue

Handles the processing of PDF catalogs:

1. Features
2. Multi-stage processing pipeline
3. Progress tracking for each stage
4. Automatic retry for failed stages
5. Result storage and notification
6. Job Lifecycle
7. Submission: PDF uploaded and job created
8. Queuing: Job prioritized and scheduled
9. Processing: Multi-stage PDF extraction
10. Completion: Results stored and notifications sent
11. (Optional) Error recovery: Automatic retries or manual intervention
12. Events Published
13. pdf.job.queued: Job has been queued
14. pdf.job.started: Processing has started
15. pdf.job.progress: Progress update (percentage, current stage)
16. pdf.job.completed: Processing completed successfully
17. pdf.job.failed: Processing failed
18. pdf.job.materials.extracted: Materials were extracted and are ready for import

Result storage and notification

Job Lifecycle

(Optional) Error recovery: Automatic retries or manual intervention

Events Published

pdf.job.queued

pdf.job.started

pdf.job.progress

pdf.job.completed

pdf.job.failed

pdf.job.materials.extracted

#### Web Crawler Queue

Manages web crawling jobs:

1. Features
2. Multiple crawler provider support
3. Credential management for authenticated sites
4. Rate limiting to respect website policies
5. Incremental crawling capabilities
6. Job Lifecycle
7. Configuration: Crawler settings defined
8. Scheduling: Job scheduled with priority
9. Initialization: Crawler setup and authentication
10. Crawling: Website traversal and data extraction
11. Completion: Data processed and notifications sent
12. Events Published
13. crawler.job.queued: Job has been queued
14. crawler.job.started: Crawling has started
15. crawler.job.progress: Progress update (pages crawled, data extracted)
16. crawler.job.completed: Crawling completed successfully
17. crawler.job.failed: Crawling failed
18. crawler.job.materials.extracted: Materials were extracted and are ready for import

Incremental crawling capabilities

Job Lifecycle

Completion: Data processed and notifications sent

Events Published

crawler.job.queued

crawler.job.started

crawler.job.progress

crawler.job.completed

crawler.job.failed

crawler.job.materials.extracted

#### ML Training Queue

Coordinates machine learning model training:

1. Features
2. Resource-intensive job management
3. GPU allocation and scheduling
4. Progress monitoring with metrics
5. Model versioning and deployment
6. Job Lifecycle
7. Configuration: Training parameters defined
8. Preprocessing: Dataset preparation
9. Training: Model training with checkpoints
10. Evaluation: Model performance assessment
11. Deployment: Model integration into recognition system
12. Events Published
13. training.job.queued: Job has been queued
14. training.job.started: Training has started
15. training.job.progress: Progress update (epoch, metrics)
16. training.job.completed: Training completed successfully
17. training.job.failed: Training failed
18. training.model.deployed: New model has been deployed

Model versioning and deployment

Job Lifecycle

Deployment: Model integration into recognition system

Events Published

training.job.queued

training.job.started

training.job.progress

training.job.completed

training.job.failed

training.model.deployed

### Unified Message Broker Architecture

The Queue System now uses a unified message broker architecture with a factory pattern that provides tiered implementation options based on application requirements:

┌───────────────────┐ ┌────────────────────────┐ ┌───────────────┐  
│ │ │ │ │ │  
│ Client Services │────▶│ Message Broker │────▶│ PDF Queue │  
│ │ │ Factory │ │ │  
└───────────────────┘ └────────────────────────┘ └───────────────┘  
 │ │ │  
 │ ▼ ▼  
 │ ┌───────────────────┐ ┌───────────────┐  
 │ │ │ │ │  
 ▼ │ IMessageBroker │ │ PDF Worker │  
┌───────────────────┐ │ Implementation │ │ │  
│ │ │ │ └───────────────┘  
│ Admin Analytics │ └───────────────────┘  
│ │ │  
└───────────────────┘ │  
 ▼  
 ┌───────────────────┐  
 │ │  
 │ Supabase Client │  
 │ │  
 └───────────────────┘

┌───────────────────┐ ┌────────────────────────┐ ┌───────────────┐  
│ │ │ │ │ │  
│ Client Services │────▶│ Message Broker │────▶│ PDF Queue │  
│ │ │ Factory │ │ │  
└───────────────────┘ └────────────────────────┘ └───────────────┘  
 │ │ │  
 │ ▼ ▼  
 │ ┌───────────────────┐ ┌───────────────┐  
 │ │ │ │ │  
 ▼ │ IMessageBroker │ │ PDF Worker │  
┌───────────────────┐ │ Implementation │ │ │  
│ │ │ │ └───────────────┘  
│ Admin Analytics │ └───────────────────┘  
│ │ │  
└───────────────────┘ │  
 ▼  
 ┌───────────────────┐  
 │ │  
 │ Supabase Client │  
 │ │  
 └───────────────────┘

#### Message Broker Interface

The system uses a standardized interface that all message broker implementations must implement:

// Common types for message queuing  
export enum QueueType {  
 PDF = 'pdf',  
 CRAWLER = 'crawler',  
 SYSTEM = 'system',  
 TRAINING = 'training',  
 KNOWLEDGE\_BASE = 'knowledge-base'  
}  
  
export enum MessageType {  
 JOB\_QUEUED = 'job.queued',  
 JOB\_STARTED = 'job.started',  
 JOB\_PROGRESS = 'job.progress',  
 JOB\_COMPLETED = 'job.completed',  
 JOB\_FAILED = 'job.failed',  
 SYSTEM\_STATUS = 'system.status',  
 CUSTOM = 'custom'  
}  
  
export interface MessagePayload {  
 [key: string]: any;  
 timestamp?: number;  
}  
  
export interface SubscriptionOptions {  
 persistent?: boolean;  
 priority?: 'high' | 'normal' | 'low';  
 ackRequired?: boolean;  
 retryOnReconnect?: boolean;  
 buffer?: number;  
}  
  
// Core Message Broker Interface  
export interface IMessageBroker {  
 // Core functionality  
 init(): Promise<void>;  
 shutdown(): Promise<void>;  
  
 // Basic pub/sub operations  
 publish(queue: QueueType | string, messageType: MessageType | string, payload: MessagePayload): Promise<boolean>;  
 subscribe<T = MessagePayload>(  
 queue: QueueType | string,   
 messageType: MessageType | string,   
 handler: MessageHandler<T>  
 ): Promise<() => Promise<void>>;  
  
 // Advanced subscription with options  
 subscribeWithOptions<T = MessagePayload>(  
 queue: QueueType | string,  
 messageType: MessageType | string,  
 handler: MessageHandler<T>,  
 options: SubscriptionOptions  
 ): Promise<() => Promise<void>>;  
  
 // Monitoring and statistics  
 getStats(): MessageBrokerStats;  
 flush(): Promise<void>;  
}

// Common types for message queuing  
export enum QueueType {  
 PDF = 'pdf',  
 CRAWLER = 'crawler',  
 SYSTEM = 'system',  
 TRAINING = 'training',  
 KNOWLEDGE\_BASE = 'knowledge-base'  
}  
  
export enum MessageType {  
 JOB\_QUEUED = 'job.queued',  
 JOB\_STARTED = 'job.started',  
 JOB\_PROGRESS = 'job.progress',  
 JOB\_COMPLETED = 'job.completed',  
 JOB\_FAILED = 'job.failed',  
 SYSTEM\_STATUS = 'system.status',  
 CUSTOM = 'custom'  
}  
  
export interface MessagePayload {  
 [key: string]: any;  
 timestamp?: number;  
}  
  
export interface SubscriptionOptions {  
 persistent?: boolean;  
 priority?: 'high' | 'normal' | 'low';  
 ackRequired?: boolean;  
 retryOnReconnect?: boolean;  
 buffer?: number;  
}  
  
// Core Message Broker Interface  
export interface IMessageBroker {  
 // Core functionality  
 init(): Promise<void>;  
 shutdown(): Promise<void>;  
  
 // Basic pub/sub operations  
 publish(queue: QueueType | string, messageType: MessageType | string, payload: MessagePayload): Promise<boolean>;  
 subscribe<T = MessagePayload>(  
 queue: QueueType | string,   
 messageType: MessageType | string,   
 handler: MessageHandler<T>  
 ): Promise<() => Promise<void>>;  
  
 // Advanced subscription with options  
 subscribeWithOptions<T = MessagePayload>(  
 queue: QueueType | string,  
 messageType: MessageType | string,  
 handler: MessageHandler<T>,  
 options: SubscriptionOptions  
 ): Promise<() => Promise<void>>;  
  
 // Monitoring and statistics  
 getStats(): MessageBrokerStats;  
 flush(): Promise<void>;  
}

#### Message Broker Factory

The system uses a factory pattern to create appropriate message broker instances:

// Implementation tiers  
export enum BrokerImplementation {  
 BASIC = 'basic', // Basic pub/sub functionality  
 ENHANCED = 'enhanced', // Added persistence and delivery guarantees  
 ADVANCED = 'advanced' // Full feature set with optimized performance  
}  
  
// Factory for creating message brokers  
export class MessageBrokerFactory {  
 // Create a broker with specified implementation  
 public static createBroker(implementation: BrokerImplementation = BrokerImplementation.BASIC): IMessageBroker {  
 // Get the broker implementation based on requirements  
 return new UnifiedMessageBroker({  
 implementation,  
 clientManager: SupabaseClientManager.getInstance()  
 });  
 }  
  
 // Create with options  
 public static createBrokerWithOptions(options: BrokerOptions): IMessageBroker {  
 const implementation = this.determineBrokerImplementation(options);  
 return new UnifiedMessageBroker({  
 implementation,  
 ...options  
 });  
 }  
  
 // Determine appropriate implementation based on requirements  
 private static determineBrokerImplementation(options: BrokerOptions): BrokerImplementation {  
 // If implementation explicitly specified, use it  
 if (options.implementation) {  
 return options.implementation;  
 }  
  
 // If persistence is required, at least ENHANCED is needed  
 if (options.persistence === true) {  
 // If horizontal scaling is needed, use ADVANCED  
 if (options.scaling === true) {  
 return BrokerImplementation.ADVANCED;  
 }  
 return BrokerImplementation.ENHANCED;  
 }  
  
 // If only basic pub/sub is needed  
 return BrokerImplementation.BASIC;  
 }  
}

// Implementation tiers  
export enum BrokerImplementation {  
 BASIC = 'basic', // Basic pub/sub functionality  
 ENHANCED = 'enhanced', // Added persistence and delivery guarantees  
 ADVANCED = 'advanced' // Full feature set with optimized performance  
}  
  
// Factory for creating message brokers  
export class MessageBrokerFactory {  
 // Create a broker with specified implementation  
 public static createBroker(implementation: BrokerImplementation = BrokerImplementation.BASIC): IMessageBroker {  
 // Get the broker implementation based on requirements  
 return new UnifiedMessageBroker({  
 implementation,  
 clientManager: SupabaseClientManager.getInstance()  
 });  
 }  
  
 // Create with options  
 public static createBrokerWithOptions(options: BrokerOptions): IMessageBroker {  
 const implementation = this.determineBrokerImplementation(options);  
 return new UnifiedMessageBroker({  
 implementation,  
 ...options  
 });  
 }  
  
 // Determine appropriate implementation based on requirements  
 private static determineBrokerImplementation(options: BrokerOptions): BrokerImplementation {  
 // If implementation explicitly specified, use it  
 if (options.implementation) {  
 return options.implementation;  
 }  
  
 // If persistence is required, at least ENHANCED is needed  
 if (options.persistence === true) {  
 // If horizontal scaling is needed, use ADVANCED  
 if (options.scaling === true) {  
 return BrokerImplementation.ADVANCED;  
 }  
 return BrokerImplementation.ENHANCED;  
 }  
  
 // If only basic pub/sub is needed  
 return BrokerImplementation.BASIC;  
 }  
}

#### Unified Message Broker Implementation

The UnifiedMessageBroker combines the best features of previous implementations:

// Unified Message Broker Implementation  
export class UnifiedMessageBroker implements IMessageBroker {  
 private supabase: SupabaseClient;  
 private config: BrokerConfig;  
 private implementation: BrokerImplementation;  
 private channels: Record<string, RealtimeChannel> = {};  
 private handlers: Record<string, Record<string, Array<HandlerWrapper>>> = {};  
 private persistentStorage: Record<string, any> = {};  
 private initialized: boolean = false;  
 private stats: MessageBrokerStats = this.getDefaultStats();  
  
 constructor(options: BrokerOptions) {  
 this.implementation = options.implementation || BrokerImplementation.BASIC;  
 this.config = this.getConfigForImplementation(this.implementation);  
 this.supabase = options.clientManager.getClient();  
  
 logger.info(`Unified Message Broker created with implementation: ${this.implementation}`);  
 }  
  
 // Initialize the broker  
 public async init(): Promise<void> {  
 if (this.initialized) {  
 return;  
 }  
  
 try {  
 if (this.config.persistence) {  
 await this.initPersistentStorage();  
 }  
  
 this.initialized = true;  
 logger.info(`Unified Message Broker initialized (${this.implementation})`);  
 } catch (error) {  
 logger.error(`Failed to initialize message broker: ${error instanceof Error ? error.message : String(error)}`);  
 throw error;  
 }  
 }  
  
 // Publish a message  
 public async publish(  
 queue: QueueType | string,   
 messageType: MessageType | string,   
 payload: MessagePayload  
 ): Promise<boolean> {  
 if (!this.initialized) {  
 await this.init();  
 }  
  
 try {  
 // Add timestamp if not present  
 const messagePayload = {  
 ...payload,  
 timestamp: payload.timestamp || Date.now()  
 };  
  
 // Get or create channel  
 const channel = await this.getChannel(queue);  
  
 // Update stats  
 this.stats.messagesSent++;  
 this.stats.lastMessageSent = new Date();  
  
 // If persistence is enabled, store message first  
 if (this.config.persistence) {  
 await this.persistMessage(queue, messageType, messagePayload);  
 }  
  
 // Send the message  
 await channel.send({  
 type: 'broadcast',  
 event: messageType,  
 payload: messagePayload  
 });  
  
 logger.debug(`Published message to ${queue}:${messageType}`, {   
 implementation: this.implementation,  
 messageId: messagePayload.id   
 });  
  
 return true;  
 } catch (error) {  
 this.stats.errors++;  
 logger.error(`Failed to publish message to ${queue}:${messageType}`, error);  
  
 // If advanced error handling enabled, handle the error  
 if (this.config.advancedErrorHandling) {  
 // Retry logic would go here  
 }  
  
 throw new Error(`Message publishing failed: ${error instanceof Error ? error.message : String(error)}`);  
 }  
 }  
  
 // Subscribe to messages  
 public async subscribe<T = MessagePayload>(  
 queue: QueueType | string,   
 messageType: MessageType | string,   
 handler: MessageHandler<T>  
 ): Promise<() => Promise<void>> {  
 // Default subscription with basic options  
 return this.subscribeWithOptions(queue, messageType, handler, {  
 persistent: false,  
 priority: 'normal',  
 ackRequired: false,  
 retryOnReconnect: this.implementation !== BrokerImplementation.BASIC  
 });  
 }  
  
 // Subscribe with advanced options  
 public async subscribeWithOptions<T = MessagePayload>(  
 queue: QueueType | string,  
 messageType: MessageType | string,  
 handler: MessageHandler<T>,  
 options: SubscriptionOptions  
 ): Promise<() => Promise<void>> {  
 if (!this.initialized) {  
 await this.init();  
 }  
  
 // Ensure handlers exist for this queue and message type  
 if (!this.handlers[queue]) {  
 this.handlers[queue] = {};  
 }  
  
 if (!this.handlers[queue][messageType]) {  
 this.handlers[queue][messageType] = [];  
 }  
  
 // Create handler wrapper with options  
 const handlerWrapper: HandlerWrapper = {  
 handler,  
 options,  
 timestamp: Date.now()  
 };  
  
 // Add handler to the list  
 this.handlers[queue][messageType].push(handlerWrapper);  
  
 // Get or create the channel  
 const channel = await this.getChannel(queue);  
  
 // Update stats  
 this.stats.subscriptions++;  
  
 // Return unsubscribe function  
 return async () => {  
 if (this.handlers[queue] && this.handlers[queue][messageType]) {  
 this.handlers[queue][messageType] = this.handlers[queue][messageType]  
 .filter(h => h.handler !== handler);  
  
 // If no more handlers for this message type, clean up  
 if (this.handlers[queue][messageType].length === 0) {  
 delete this.handlers[queue][messageType];  
  
 // If no more handlers for this queue, clean up channel  
 if (Object.keys(this.handlers[queue]).length === 0) {  
 delete this.handlers[queue];  
 if (this.channels[queue]) {  
 await this.channels[queue].unsubscribe();  
 delete this.channels[queue];  
 }  
 }  
 }  
  
 this.stats.subscriptions--;  
 }  
 };  
 }  
  
 // Shutdown the broker  
 public async shutdown(): Promise<void> {  
 try {  
 // Unsubscribe from all channels  
 await Promise.all(  
 Object.values(this.channels).map(channel => channel.unsubscribe())  
 );  
  
 this.channels = {};  
 this.handlers = {};  
 this.initialized = false;  
  
 logger.info(`Unified Message Broker shutdown complete (${this.implementation})`);  
 } catch (error) {  
 logger.error(`Error during message broker shutdown: ${error instanceof Error ? error.message : String(error)}`);  
 throw error;  
 }  
 }  
  
 // Get broker statistics  
 public getStats(): MessageBrokerStats {  
 return {  
 ...this.stats,  
 implementation: this.implementation,  
 activeChannels: Object.keys(this.channels).length,  
 activeSubscriptions: this.stats.subscriptions  
 };  
 }  
  
 // Flush any pending messages  
 public async flush(): Promise<void> {  
 if (this.config.persistence) {  
 await this.flushPersistentStorage();  
 }  
 }  
  
 // Helper methods for different implementations  
 private async getChannel(queue: QueueType | string): Promise<RealtimeChannel> {  
 if (this.channels[queue]) {  
 return this.channels[queue];  
 }  
  
 // Create new channel  
 const channel = this.supabase.channel(`queue:${queue}`);  
  
 // Set up handlers  
 if (this.handlers[queue]) {  
 Object.entries(this.handlers[queue]).forEach(([eventType, handlers]) => {  
 channel.on('broadcast', { event: eventType }, async (payload) => {  
 try {  
 // Update stats  
 this.stats.messagesReceived++;  
 this.stats.lastMessageReceived = new Date();  
  
 // Execute handlers based on priority if using advanced implementation  
 const sortedHandlers = [...handlers];  
 if (this.implementation === BrokerImplementation.ADVANCED) {  
 sortedHandlers.sort((a, b) => {  
 const priorityMap = { high: 3, normal: 2, low: 1 };  
 return (priorityMap[b.options.priority || 'normal'] || 2) -   
 (priorityMap[a.options.priority || 'normal'] || 2);  
 });  
 }  
  
 // Execute each handler  
 await Promise.all(sortedHandlers.map(handlerWrapper => {  
 try {  
 return handlerWrapper.handler(payload.payload);  
 } catch (error) {  
 logger.error(`Handler error for ${queue}:${eventType}`, error);  
 return Promise.resolve();  
 }  
 }));  
 } catch (error) {  
 this.stats.errors++;  
 logger.error(`Error handling event ${eventType} on channel ${queue}:`, error);  
 }  
 });  
 });  
 }  
  
 // Subscribe to the channel with appropriate options  
 channel.subscribe((status) => {  
 if (status === 'SUBSCRIBED') {  
 logger.info(`Subscribed to channel: ${queue} (${this.implementation})`);  
  
 // If using advanced implementation and retry on reconnect  
 if (this.implementation !== BrokerImplementation.BASIC &&   
 this.config.retryOnReconnect &&   
 this.handlers[queue]) {  
 this.handleReconnection(queue);  
 }  
 } else if (status === 'CLOSED') {  
 logger.warn(`Channel closed: ${queue}`);  
 } else if (status === 'CHANNEL\_ERROR') {  
 this.stats.errors++;  
 logger.error(`Channel error: ${queue}`);  
 }  
 });  
  
 this.channels[queue] = channel;  
 return channel;  
 }  
  
 // Additional implementation-specific methods would be here  
 // ...  
}  
  
// Export factory as the primary access point  
export const messageBrokerFactory = MessageBrokerFactory;

// Unified Message Broker Implementation  
export class UnifiedMessageBroker implements IMessageBroker {  
 private supabase: SupabaseClient;  
 private config: BrokerConfig;  
 private implementation: BrokerImplementation;  
 private channels: Record<string, RealtimeChannel> = {};  
 private handlers: Record<string, Record<string, Array<HandlerWrapper>>> = {};  
 private persistentStorage: Record<string, any> = {};  
 private initialized: boolean = false;  
 private stats: MessageBrokerStats = this.getDefaultStats();  
  
 constructor(options: BrokerOptions) {  
 this.implementation = options.implementation || BrokerImplementation.BASIC;  
 this.config = this.getConfigForImplementation(this.implementation);  
 this.supabase = options.clientManager.getClient();  
  
 logger.info(`Unified Message Broker created with implementation: ${this.implementation}`);  
 }  
  
 // Initialize the broker  
 public async init(): Promise<void> {  
 if (this.initialized) {  
 return;  
 }  
  
 try {  
 if (this.config.persistence) {  
 await this.initPersistentStorage();  
 }  
  
 this.initialized = true;  
 logger.info(`Unified Message Broker initialized (${this.implementation})`);  
 } catch (error) {  
 logger.error(`Failed to initialize message broker: ${error instanceof Error ? error.message : String(error)}`);  
 throw error;  
 }  
 }  
  
 // Publish a message  
 public async publish(  
 queue: QueueType | string,   
 messageType: MessageType | string,   
 payload: MessagePayload  
 ): Promise<boolean> {  
 if (!this.initialized) {  
 await this.init();  
 }  
  
 try {  
 // Add timestamp if not present  
 const messagePayload = {  
 ...payload,  
 timestamp: payload.timestamp || Date.now()  
 };  
  
 // Get or create channel  
 const channel = await this.getChannel(queue);  
  
 // Update stats  
 this.stats.messagesSent++;  
 this.stats.lastMessageSent = new Date();  
  
 // If persistence is enabled, store message first  
 if (this.config.persistence) {  
 await this.persistMessage(queue, messageType, messagePayload);  
 }  
  
 // Send the message  
 await channel.send({  
 type: 'broadcast',  
 event: messageType,  
 payload: messagePayload  
 });  
  
 logger.debug(`Published message to ${queue}:${messageType}`, {   
 implementation: this.implementation,  
 messageId: messagePayload.id   
 });  
  
 return true;  
 } catch (error) {  
 this.stats.errors++;  
 logger.error(`Failed to publish message to ${queue}:${messageType}`, error);  
  
 // If advanced error handling enabled, handle the error  
 if (this.config.advancedErrorHandling) {  
 // Retry logic would go here  
 }  
  
 throw new Error(`Message publishing failed: ${error instanceof Error ? error.message : String(error)}`);  
 }  
 }  
  
 // Subscribe to messages  
 public async subscribe<T = MessagePayload>(  
 queue: QueueType | string,   
 messageType: MessageType | string,   
 handler: MessageHandler<T>  
 ): Promise<() => Promise<void>> {  
 // Default subscription with basic options  
 return this.subscribeWithOptions(queue, messageType, handler, {  
 persistent: false,  
 priority: 'normal',  
 ackRequired: false,  
 retryOnReconnect: this.implementation !== BrokerImplementation.BASIC  
 });  
 }  
  
 // Subscribe with advanced options  
 public async subscribeWithOptions<T = MessagePayload>(  
 queue: QueueType | string,  
 messageType: MessageType | string,  
 handler: MessageHandler<T>,  
 options: SubscriptionOptions  
 ): Promise<() => Promise<void>> {  
 if (!this.initialized) {  
 await this.init();  
 }  
  
 // Ensure handlers exist for this queue and message type  
 if (!this.handlers[queue]) {  
 this.handlers[queue] = {};  
 }  
  
 if (!this.handlers[queue][messageType]) {  
 this.handlers[queue][messageType] = [];  
 }  
  
 // Create handler wrapper with options  
 const handlerWrapper: HandlerWrapper = {  
 handler,  
 options,  
 timestamp: Date.now()  
 };  
  
 // Add handler to the list  
 this.handlers[queue][messageType].push(handlerWrapper);  
  
 // Get or create the channel  
 const channel = await this.getChannel(queue);  
  
 // Update stats  
 this.stats.subscriptions++;  
  
 // Return unsubscribe function  
 return async () => {  
 if (this.handlers[queue] && this.handlers[queue][messageType]) {  
 this.handlers[queue][messageType] = this.handlers[queue][messageType]  
 .filter(h => h.handler !== handler);  
  
 // If no more handlers for this message type, clean up  
 if (this.handlers[queue][messageType].length === 0) {  
 delete this.handlers[queue][messageType];  
  
 // If no more handlers for this queue, clean up channel  
 if (Object.keys(this.handlers[queue]).length === 0) {  
 delete this.handlers[queue];  
 if (this.channels[queue]) {  
 await this.channels[queue].unsubscribe();  
 delete this.channels[queue];  
 }  
 }  
 }  
  
 this.stats.subscriptions--;  
 }  
 };  
 }  
  
 // Shutdown the broker  
 public async shutdown(): Promise<void> {  
 try {  
 // Unsubscribe from all channels  
 await Promise.all(  
 Object.values(this.channels).map(channel => channel.unsubscribe())  
 );  
  
 this.channels = {};  
 this.handlers = {};  
 this.initialized = false;  
  
 logger.info(`Unified Message Broker shutdown complete (${this.implementation})`);  
 } catch (error) {  
 logger.error(`Error during message broker shutdown: ${error instanceof Error ? error.message : String(error)}`);  
 throw error;  
 }  
 }  
  
 // Get broker statistics  
 public getStats(): MessageBrokerStats {  
 return {  
 ...this.stats,  
 implementation: this.implementation,  
 activeChannels: Object.keys(this.channels).length,  
 activeSubscriptions: this.stats.subscriptions  
 };  
 }  
  
 // Flush any pending messages  
 public async flush(): Promise<void> {  
 if (this.config.persistence) {  
 await this.flushPersistentStorage();  
 }  
 }  
  
 // Helper methods for different implementations  
 private async getChannel(queue: QueueType | string): Promise<RealtimeChannel> {  
 if (this.channels[queue]) {  
 return this.channels[queue];  
 }  
  
 // Create new channel  
 const channel = this.supabase.channel(`queue:${queue}`);  
  
 // Set up handlers  
 if (this.handlers[queue]) {  
 Object.entries(this.handlers[queue]).forEach(([eventType, handlers]) => {  
 channel.on('broadcast', { event: eventType }, async (payload) => {  
 try {  
 // Update stats  
 this.stats.messagesReceived++;  
 this.stats.lastMessageReceived = new Date();  
  
 // Execute handlers based on priority if using advanced implementation  
 const sortedHandlers = [...handlers];  
 if (this.implementation === BrokerImplementation.ADVANCED) {  
 sortedHandlers.sort((a, b) => {  
 const priorityMap = { high: 3, normal: 2, low: 1 };  
 return (priorityMap[b.options.priority || 'normal'] || 2) -   
 (priorityMap[a.options.priority || 'normal'] || 2);  
 });  
 }  
  
 // Execute each handler  
 await Promise.all(sortedHandlers.map(handlerWrapper => {  
 try {  
 return handlerWrapper.handler(payload.payload);  
 } catch (error) {  
 logger.error(`Handler error for ${queue}:${eventType}`, error);  
 return Promise.resolve();  
 }  
 }));  
 } catch (error) {  
 this.stats.errors++;  
 logger.error(`Error handling event ${eventType} on channel ${queue}:`, error);  
 }  
 });  
 });  
 }  
  
 // Subscribe to the channel with appropriate options  
 channel.subscribe((status) => {  
 if (status === 'SUBSCRIBED') {  
 logger.info(`Subscribed to channel: ${queue} (${this.implementation})`);  
  
 // If using advanced implementation and retry on reconnect  
 if (this.implementation !== BrokerImplementation.BASIC &&   
 this.config.retryOnReconnect &&   
 this.handlers[queue]) {  
 this.handleReconnection(queue);  
 }  
 } else if (status === 'CLOSED') {  
 logger.warn(`Channel closed: ${queue}`);  
 } else if (status === 'CHANNEL\_ERROR') {  
 this.stats.errors++;  
 logger.error(`Channel error: ${queue}`);  
 }  
 });  
  
 this.channels[queue] = channel;  
 return channel;  
 }  
  
 // Additional implementation-specific methods would be here  
 // ...  
}  
  
// Export factory as the primary access point  
export const messageBrokerFactory = MessageBrokerFactory;

#### Implementation Tiers

The system provides three tiers of implementation:

##### Basic Implementation

The BASIC implementation provides fundamental pub/sub capabilities:  
- Simple message publishing and subscription  
- No persistence or guaranteed delivery  
- No automatic reconnection  
- Minimal overhead and resource usage  
- Suitable for non-critical messaging needs

Example use case: Status updates, notifications, and informational messages where some loss is acceptable.

##### Enhanced Implementation

The ENHANCED implementation adds reliability features:  
- Message persistence with storage in Supabase tables  
- Delivery guarantees with acknowledgments  
- Automatic reconnection with message replay  
- Error handling with retry logic  
- Performance optimizations with batching  
- Metrics and statistics tracking

Example use case: Job processing queues where message delivery is important but not absolutely critical.

##### Advanced Implementation

The ADVANCED implementation adds enterprise features:  
- All Enhanced features plus:  
- Priority-based message handling  
- Message ordering guarantees  
- Connection pooling for high-throughput  
- Advanced error handling with circuit breakers  
- Performance optimization with database indexing and caching  
- Support for horizontal scaling

Example use case: Mission-critical workflows, financial operations, and high-volume processing where message loss cannot be tolerated.

#### Migration and Usage

Services can access the message broker through the factory:

import { messageBrokerFactory, BrokerImplementation, QueueType, MessageType } from '../messaging/messageBrokerFactory';  
  
// Get a basic broker for simple use cases  
const basicBroker = messageBrokerFactory.createBroker();  
  
// Get an enhanced broker for more reliable messaging  
const enhancedBroker = messageBrokerFactory.createBroker(BrokerImplementation.ENHANCED);  
  
// Get an advanced broker for mission-critical operations  
const advancedBroker = messageBrokerFactory.createBroker(BrokerImplementation.ADVANCED);  
  
// Or specify requirements and let the factory determine the implementation  
const broker = messageBrokerFactory.createBrokerWithOptions({  
 persistence: true,  
 scaling: true,  
 retryOnReconnect: true  
});  
  
// Initialize the broker  
await broker.init();  
  
// Publish a message  
await broker.publish(  
 QueueType.PDF,  
 MessageType.JOB\_QUEUED,  
 {  
 jobId: 'pdf-123',  
 fileName: 'catalog.pdf'  
 }  
);  
  
// Subscribe to messages  
const unsubscribe = await broker.subscribe(  
 QueueType.PDF,  
 MessageType.JOB\_COMPLETED,  
 async (payload) => {  
 console.log(`Job completed: ${payload.jobId}`);  
 }  
);  
  
// Advanced subscription with options  
const unsubscribeAdvanced = await broker.subscribeWithOptions(  
 QueueType.PDF,  
 MessageType.JOB\_PROGRESS,  
 async (payload) => {  
 console.log(`Job progress: ${payload.progress}%`);  
 },  
 {  
 persistent: true,  
 priority: 'high',  
 ackRequired: true,  
 retryOnReconnect: true  
 }  
);  
  
// Get broker statistics  
const stats = broker.getStats();  
console.log(`Messages sent: ${stats.messagesSent}, received: ${stats.messagesReceived}`);  
  
// Clean up  
await unsubscribe();  
await unsubscribeAdvanced();  
await broker.shutdown();

import { messageBrokerFactory, BrokerImplementation, QueueType, MessageType } from '../messaging/messageBrokerFactory';  
  
// Get a basic broker for simple use cases  
const basicBroker = messageBrokerFactory.createBroker();  
  
// Get an enhanced broker for more reliable messaging  
const enhancedBroker = messageBrokerFactory.createBroker(BrokerImplementation.ENHANCED);  
  
// Get an advanced broker for mission-critical operations  
const advancedBroker = messageBrokerFactory.createBroker(BrokerImplementation.ADVANCED);  
  
// Or specify requirements and let the factory determine the implementation  
const broker = messageBrokerFactory.createBrokerWithOptions({  
 persistence: true,  
 scaling: true,  
 retryOnReconnect: true  
});  
  
// Initialize the broker  
await broker.init();  
  
// Publish a message  
await broker.publish(  
 QueueType.PDF,  
 MessageType.JOB\_QUEUED,  
 {  
 jobId: 'pdf-123',  
 fileName: 'catalog.pdf'  
 }  
);  
  
// Subscribe to messages  
const unsubscribe = await broker.subscribe(  
 QueueType.PDF,  
 MessageType.JOB\_COMPLETED,  
 async (payload) => {  
 console.log(`Job completed: ${payload.jobId}`);  
 }  
);  
  
// Advanced subscription with options  
const unsubscribeAdvanced = await broker.subscribeWithOptions(  
 QueueType.PDF,  
 MessageType.JOB\_PROGRESS,  
 async (payload) => {  
 console.log(`Job progress: ${payload.progress}%`);  
 },  
 {  
 persistent: true,  
 priority: 'high',  
 ackRequired: true,  
 retryOnReconnect: true  
 }  
);  
  
// Get broker statistics  
const stats = broker.getStats();  
console.log(`Messages sent: ${stats.messagesSent}, received: ${stats.messagesReceived}`);  
  
// Clean up  
await unsubscribe();  
await unsubscribeAdvanced();  
await broker.shutdown();

### Queue Adapter Implementation

Queue adapters standardize the interface between different queue implementations and the message broker:

// Queue Adapter Interface  
export interface QueueAdapter<T> {  
 // Job management  
 createJob(data: Partial<T>): Promise<string>;  
 getJob(jobId: string): Promise<T | null>;  
 updateJob(jobId: string, updates: Partial<T>): Promise<T>;  
 deleteJob(jobId: string): Promise<boolean>;  
  
 // Queue operations  
 getJobs(options: QueueQueryOptions): Promise<{ jobs: T[], total: number }>;  
 processNextJob(): Promise<T | null>;  
 getQueueStats(): Promise<QueueStats>;  
  
 // Event handling  
 subscribeToEvents(eventTypes: string[], handler: EventHandler): () => void;  
 publishEvent(eventType: string, data: any): Promise<void>;  
}  
  
// Base Queue Adapter Implementation  
export abstract class BaseQueueAdapter<T extends QueueJob> implements QueueAdapter<T> {  
 protected queueName: string;  
 protected messageBroker: MessageBroker;  
  
 constructor(queueName: string, messageBroker: MessageBroker) {  
 this.queueName = queueName;  
 this.messageBroker = messageBroker;  
 }  
  
 // Abstract methods to be implemented by specific queue adapters  
 abstract createJob(data: Partial<T>): Promise<string>;  
 abstract getJob(jobId: string): Promise<T | null>;  
 abstract updateJob(jobId: string, updates: Partial<T>): Promise<T>;  
 abstract deleteJob(jobId: string): Promise<boolean>;  
 abstract getJobs(options: QueueQueryOptions): Promise<{ jobs: T[], total: number }>;  
 abstract processNextJob(): Promise<T | null>;  
 abstract getQueueStats(): Promise<QueueStats>;  
  
 // Event methods using message broker  
 public async publishEvent(eventType: string, data: any): Promise<void> {  
 await this.messageBroker.publish(this.queueName, eventType, {  
 ...data,  
 queueName: this.queueName,  
 timestamp: Date.now()  
 });  
 }  
  
 public subscribeToEvents(eventTypes: string[], handler: EventHandler): () => void {  
 const unsubscribers = eventTypes.map(eventType =>   
 this.messageBroker.subscribe(this.queueName, eventType, handler)  
 );  
  
 return () => unsubscribers.forEach(unsubscribe => unsubscribe());  
 }  
}

// Queue Adapter Interface  
export interface QueueAdapter<T> {  
 // Job management  
 createJob(data: Partial<T>): Promise<string>;  
 getJob(jobId: string): Promise<T | null>;  
 updateJob(jobId: string, updates: Partial<T>): Promise<T>;  
 deleteJob(jobId: string): Promise<boolean>;  
  
 // Queue operations  
 getJobs(options: QueueQueryOptions): Promise<{ jobs: T[], total: number }>;  
 processNextJob(): Promise<T | null>;  
 getQueueStats(): Promise<QueueStats>;  
  
 // Event handling  
 subscribeToEvents(eventTypes: string[], handler: EventHandler): () => void;  
 publishEvent(eventType: string, data: any): Promise<void>;  
}  
  
// Base Queue Adapter Implementation  
export abstract class BaseQueueAdapter<T extends QueueJob> implements QueueAdapter<T> {  
 protected queueName: string;  
 protected messageBroker: MessageBroker;  
  
 constructor(queueName: string, messageBroker: MessageBroker) {  
 this.queueName = queueName;  
 this.messageBroker = messageBroker;  
 }  
  
 // Abstract methods to be implemented by specific queue adapters  
 abstract createJob(data: Partial<T>): Promise<string>;  
 abstract getJob(jobId: string): Promise<T | null>;  
 abstract updateJob(jobId: string, updates: Partial<T>): Promise<T>;  
 abstract deleteJob(jobId: string): Promise<boolean>;  
 abstract getJobs(options: QueueQueryOptions): Promise<{ jobs: T[], total: number }>;  
 abstract processNextJob(): Promise<T | null>;  
 abstract getQueueStats(): Promise<QueueStats>;  
  
 // Event methods using message broker  
 public async publishEvent(eventType: string, data: any): Promise<void> {  
 await this.messageBroker.publish(this.queueName, eventType, {  
 ...data,  
 queueName: this.queueName,  
 timestamp: Date.now()  
 });  
 }  
  
 public subscribeToEvents(eventTypes: string[], handler: EventHandler): () => void {  
 const unsubscribers = eventTypes.map(eventType =>   
 this.messageBroker.subscribe(this.queueName, eventType, handler)  
 );  
  
 return () => unsubscribers.forEach(unsubscribe => unsubscribe());  
 }  
}

### PDF Queue Implementation

Example of the PDF Processing Queue implementation:

// PDF Queue Job Interface  
export interface PDFProcessingJob extends QueueJob {  
 fileName: string;  
 fileSize: number;  
 filePath: string;  
 pageCount?: number;  
 options: PDFProcessingOptions;  
 progress: {  
 currentPage: number;  
 totalPages: number;  
 percentComplete: number;  
 currentStage: PDFProcessingStage;  
 stageProgress: number;  
 };  
 results?: {  
 imagesExtracted: number;  
 textRegionsExtracted: number;  
 tablesDetected: number;  
 materialsIdentified: number;  
 materialsImported: number;  
 };  
 outputDir?: string;  
}  
  
// PDF Queue Implementation  
export class PDFQueue extends BaseQueueAdapter<PDFProcessingJob> {  
 private db: Database;  
  
 constructor(messageBroker: MessageBroker, db: Database) {  
 super('pdf', messageBroker);  
 this.db = db;  
  
 // Subscribe to relevant events from other queues  
 this.subscribeToExternalEvents();  
 }  
  
 /\*\*  
 \* Create a new PDF processing job  
 \*/  
 public async createJob(data: Partial<PDFProcessingJob>): Promise<string> {  
 // Generate job ID  
 const jobId = uuidv4();  
  
 // Create job with defaults  
 const job: PDFProcessingJob = {  
 id: jobId,  
 status: 'waiting',  
 priority: data.priority || 5,  
 progress: {  
 currentPage: 0,  
 totalPages: data.pageCount || 0,  
 percentComplete: 0,  
 currentStage: 'initializing',  
 stageProgress: 0  
 },  
 fileName: data.fileName!,  
 fileSize: data.fileSize!,  
 filePath: data.filePath!,  
 options: data.options || {  
 extractImages: true,  
 extractText: true,  
 enhanceResolution: false,  
 associateTextWithImages: true,  
 extractStructuredData: true  
 },  
 createdAt: new Date(),  
 updatedAt: new Date(),  
 ...data  
 };  
  
 // Save job to database  
 await this.db.insert('pdf\_jobs', job);  
  
 // Publish event  
 await this.publishEvent('pdf.job.queued', {  
 jobId,  
 fileName: job.fileName,  
 fileSize: job.fileSize,  
 priority: job.priority,  
 options: job.options  
 });  
  
 return jobId;  
 }  
  
 /\*\*  
 \* Get job by ID  
 \*/  
 public async getJob(jobId: string): Promise<PDFProcessingJob | null> {  
 const job = await this.db.selectOne('pdf\_jobs', { id: jobId });  
 return job as PDFProcessingJob || null;  
 }  
  
 /\*\*  
 \* Update job  
 \*/  
 public async updateJob(jobId: string, updates: Partial<PDFProcessingJob>): Promise<PDFProcessingJob> {  
 // Get current job  
 const currentJob = await this.getJob(jobId);  
 if (!currentJob) {  
 throw new Error(`Job not found: ${jobId}`);  
 }  
  
 // Update job  
 const updatedJob = {  
 ...currentJob,  
 ...updates,  
 updatedAt: new Date()  
 };  
  
 // Save to database  
 await this.db.update('pdf\_jobs', { id: jobId }, updatedJob);  
  
 // Publish events based on updates  
 if (updates.status) {  
 switch (updates.status) {  
 case 'processing':  
 await this.publishEvent('pdf.job.started', {  
 jobId,  
 fileName: updatedJob.fileName,  
 fileSize: updatedJob.fileSize  
 });  
 break;  
 case 'completed':  
 await this.publishEvent('pdf.job.completed', {  
 jobId,  
 fileName: updatedJob.fileName,  
 results: updatedJob.results  
 });  
 break;  
 case 'failed':  
 await this.publishEvent('pdf.job.failed', {  
 jobId,  
 fileName: updatedJob.fileName,  
 error: updatedJob.error  
 });  
 break;  
 }  
 }  
  
 // Publish progress updates if progress changed  
 if (updates.progress) {  
 await this.publishEvent('pdf.job.progress', {  
 jobId,  
 fileName: updatedJob.fileName,  
 progress: updatedJob.progress  
 });  
 }  
  
 return updatedJob;  
 }  
  
 /\*\*  
 \* Delete job  
 \*/  
 public async deleteJob(jobId: string): Promise<boolean> {  
 const result = await this.db.delete('pdf\_jobs', { id: jobId });  
 return result.affected > 0;  
 }  
  
 /\*\*  
 \* Get jobs with filtering, sorting, and pagination  
 \*/  
 public async getJobs(options: QueueQueryOptions): Promise<{ jobs: PDFProcessingJob[], total: number }> {  
 const {  
 status,  
 priority,  
 createdAfter,  
 createdBefore,  
 limit = 20,  
 skip = 0,  
 sort = 'createdAt',  
 order = 'desc'  
 } = options;  
  
 // Build query  
 const query: Record<string, any> = {};  
  
 if (status) {  
 query.status = Array.isArray(status) ? { $in: status } : status;  
 }  
  
 if (priority) {  
 query.priority = Array.isArray(priority)   
 ? { $in: priority }   
 : typeof priority === 'object'  
 ? priority  
 : { $gte: priority };  
 }  
  
 if (createdAfter || createdBefore) {  
 query.createdAt = {};  
 if (createdAfter) query.createdAt.$gte = createdAfter;  
 if (createdBefore) query.createdAt.$lte = createdBefore;  
 }  
  
 // Execute query  
 const jobs = await this.db.select('pdf\_jobs', query, {  
 limit,  
 skip,  
 sort: { [sort]: order === 'asc' ? 1 : -1 }  
 });  
  
 const total = await this.db.count('pdf\_jobs', query);  
  
 return {   
 jobs: jobs as PDFProcessingJob[],  
 total  
 };  
 }  
  
 /\*\*  
 \* Process next job in queue  
 \*/  
 public async processNextJob(): Promise<PDFProcessingJob | null> {  
 // Find next job to process based on priority and creation time  
 const nextJob = await this.db.selectOne('pdf\_jobs', { status: 'waiting' }, {  
 sort: { priority: -1, createdAt: 1 }  
 });  
  
 if (!nextJob) {  
 return null;  
 }  
  
 // Update job status to processing  
 return await this.updateJob(nextJob.id, {   
 status: 'processing',  
 startedAt: new Date()  
 });  
 }  
  
 /\*\*  
 \* Get queue statistics  
 \*/  
 public async getQueueStats(): Promise<QueueStats> {  
 const waiting = await this.db.count('pdf\_jobs', { status: 'waiting' });  
 const processing = await this.db.count('pdf\_jobs', { status: 'processing' });  
 const completed = await this.db.count('pdf\_jobs', { status: 'completed' });  
 const failed = await this.db.count('pdf\_jobs', { status: 'failed' });  
  
 // Get oldest waiting job  
 const oldestJob = await this.db.selectOne('pdf\_jobs', { status: 'waiting' }, {  
 sort: { createdAt: 1 }  
 });  
  
 // Calculate throughput  
 const now = new Date();  
 const oneDayAgo = new Date(now.getTime() - 24 \* 60 \* 60 \* 1000);  
 const sevenDaysAgo = new Date(now.getTime() - 7 \* 24 \* 60 \* 60 \* 1000);  
  
 const completedLast24h = await this.db.count('pdf\_jobs', {  
 status: 'completed',  
 completedAt: { $gte: oneDayAgo }  
 });  
  
 const completedLast7d = await this.db.count('pdf\_jobs', {  
 status: 'completed',  
 completedAt: { $gte: sevenDaysAgo }  
 });  
  
 // Calculate average processing time  
 const recentJobs = await this.db.select('pdf\_jobs', {  
 status: 'completed',  
 completedAt: { $gte: sevenDaysAgo }  
 });  
  
 let totalProcessingTime = 0;  
 let jobsWithTime = 0;  
  
 recentJobs.forEach(job => {  
 if (job.startedAt && job.completedAt) {  
 totalProcessingTime += job.completedAt.getTime() - job.startedAt.getTime();  
 jobsWithTime++;  
 }  
 });  
  
 const averageProcessingTime = jobsWithTime > 0   
 ? totalProcessingTime / jobsWithTime   
 : 0;  
  
 return {  
 queueId: 'pdf',  
 name: 'PDF Processing Queue',  
 status: 'active',  
 jobCount: {  
 waiting,  
 processing,  
 completed,  
 failed  
 },  
 throughput: {  
 last24h: completedLast24h,  
 last7d: completedLast7d  
 },  
 averageProcessingTime,  
 oldestJob: oldestJob?.createdAt || null  
 };  
 }  
  
 /\*\*  
 \* Subscribe to events from other queues  
 \*/  
 private subscribeToExternalEvents(): void {  
 // Listen for knowledge base events  
 this.messageBroker.subscribe('knowledge-base', 'material.imported', async (payload) => {  
 if (payload.source === 'pdf' && payload.jobId) {  
 const job = await this.getJob(payload.jobId);  
 if (job) {  
 await this.updateJob(payload.jobId, {  
 results: {  
 ...job.results,  
 materialsImported: payload.count  
 }  
 });  
 }  
 }  
 });  
 }  
}

// PDF Queue Job Interface  
export interface PDFProcessingJob extends QueueJob {  
 fileName: string;  
 fileSize: number;  
 filePath: string;  
 pageCount?: number;  
 options: PDFProcessingOptions;  
 progress: {  
 currentPage: number;  
 totalPages: number;  
 percentComplete: number;  
 currentStage: PDFProcessingStage;  
 stageProgress: number;  
 };  
 results?: {  
 imagesExtracted: number;  
 textRegionsExtracted: number;  
 tablesDetected: number;  
 materialsIdentified: number;  
 materialsImported: number;  
 };  
 outputDir?: string;  
}  
  
// PDF Queue Implementation  
export class PDFQueue extends BaseQueueAdapter<PDFProcessingJob> {  
 private db: Database;  
  
 constructor(messageBroker: MessageBroker, db: Database) {  
 super('pdf', messageBroker);  
 this.db = db;  
  
 // Subscribe to relevant events from other queues  
 this.subscribeToExternalEvents();  
 }  
  
 /\*\*  
 \* Create a new PDF processing job  
 \*/  
 public async createJob(data: Partial<PDFProcessingJob>): Promise<string> {  
 // Generate job ID  
 const jobId = uuidv4();  
  
 // Create job with defaults  
 const job: PDFProcessingJob = {  
 id: jobId,  
 status: 'waiting',  
 priority: data.priority || 5,  
 progress: {  
 currentPage: 0,  
 totalPages: data.pageCount || 0,  
 percentComplete: 0,  
 currentStage: 'initializing',  
 stageProgress: 0  
 },  
 fileName: data.fileName!,  
 fileSize: data.fileSize!,  
 filePath: data.filePath!,  
 options: data.options || {  
 extractImages: true,  
 extractText: true,  
 enhanceResolution: false,  
 associateTextWithImages: true,  
 extractStructuredData: true  
 },  
 createdAt: new Date(),  
 updatedAt: new Date(),  
 ...data  
 };  
  
 // Save job to database  
 await this.db.insert('pdf\_jobs', job);  
  
 // Publish event  
 await this.publishEvent('pdf.job.queued', {  
 jobId,  
 fileName: job.fileName,  
 fileSize: job.fileSize,  
 priority: job.priority,  
 options: job.options  
 });  
  
 return jobId;  
 }  
  
 /\*\*  
 \* Get job by ID  
 \*/  
 public async getJob(jobId: string): Promise<PDFProcessingJob | null> {  
 const job = await this.db.selectOne('pdf\_jobs', { id: jobId });  
 return job as PDFProcessingJob || null;  
 }  
  
 /\*\*  
 \* Update job  
 \*/  
 public async updateJob(jobId: string, updates: Partial<PDFProcessingJob>): Promise<PDFProcessingJob> {  
 // Get current job  
 const currentJob = await this.getJob(jobId);  
 if (!currentJob) {  
 throw new Error(`Job not found: ${jobId}`);  
 }  
  
 // Update job  
 const updatedJob = {  
 ...currentJob,  
 ...updates,  
 updatedAt: new Date()  
 };  
  
 // Save to database  
 await this.db.update('pdf\_jobs', { id: jobId }, updatedJob);  
  
 // Publish events based on updates  
 if (updates.status) {  
 switch (updates.status) {  
 case 'processing':  
 await this.publishEvent('pdf.job.started', {  
 jobId,  
 fileName: updatedJob.fileName,  
 fileSize: updatedJob.fileSize  
 });  
 break;  
 case 'completed':  
 await this.publishEvent('pdf.job.completed', {  
 jobId,  
 fileName: updatedJob.fileName,  
 results: updatedJob.results  
 });  
 break;  
 case 'failed':  
 await this.publishEvent('pdf.job.failed', {  
 jobId,  
 fileName: updatedJob.fileName,  
 error: updatedJob.error  
 });  
 break;  
 }  
 }  
  
 // Publish progress updates if progress changed  
 if (updates.progress) {  
 await this.publishEvent('pdf.job.progress', {  
 jobId,  
 fileName: updatedJob.fileName,  
 progress: updatedJob.progress  
 });  
 }  
  
 return updatedJob;  
 }  
  
 /\*\*  
 \* Delete job  
 \*/  
 public async deleteJob(jobId: string): Promise<boolean> {  
 const result = await this.db.delete('pdf\_jobs', { id: jobId });  
 return result.affected > 0;  
 }  
  
 /\*\*  
 \* Get jobs with filtering, sorting, and pagination  
 \*/  
 public async getJobs(options: QueueQueryOptions): Promise<{ jobs: PDFProcessingJob[], total: number }> {  
 const {  
 status,  
 priority,  
 createdAfter,  
 createdBefore,  
 limit = 20,  
 skip = 0,  
 sort = 'createdAt',  
 order = 'desc'  
 } = options;  
  
 // Build query  
 const query: Record<string, any> = {};  
  
 if (status) {  
 query.status = Array.isArray(status) ? { $in: status } : status;  
 }  
  
 if (priority) {  
 query.priority = Array.isArray(priority)   
 ? { $in: priority }   
 : typeof priority === 'object'  
 ? priority  
 : { $gte: priority };  
 }  
  
 if (createdAfter || createdBefore) {  
 query.createdAt = {};  
 if (createdAfter) query.createdAt.$gte = createdAfter;  
 if (createdBefore) query.createdAt.$lte = createdBefore;  
 }  
  
 // Execute query  
 const jobs = await this.db.select('pdf\_jobs', query, {  
 limit,  
 skip,  
 sort: { [sort]: order === 'asc' ? 1 : -1 }  
 });  
  
 const total = await this.db.count('pdf\_jobs', query);  
  
 return {   
 jobs: jobs as PDFProcessingJob[],  
 total  
 };  
 }  
  
 /\*\*  
 \* Process next job in queue  
 \*/  
 public async processNextJob(): Promise<PDFProcessingJob | null> {  
 // Find next job to process based on priority and creation time  
 const nextJob = await this.db.selectOne('pdf\_jobs', { status: 'waiting' }, {  
 sort: { priority: -1, createdAt: 1 }  
 });  
  
 if (!nextJob) {  
 return null;  
 }  
  
 // Update job status to processing  
 return await this.updateJob(nextJob.id, {   
 status: 'processing',  
 startedAt: new Date()  
 });  
 }  
  
 /\*\*  
 \* Get queue statistics  
 \*/  
 public async getQueueStats(): Promise<QueueStats> {  
 const waiting = await this.db.count('pdf\_jobs', { status: 'waiting' });  
 const processing = await this.db.count('pdf\_jobs', { status: 'processing' });  
 const completed = await this.db.count('pdf\_jobs', { status: 'completed' });  
 const failed = await this.db.count('pdf\_jobs', { status: 'failed' });  
  
 // Get oldest waiting job  
 const oldestJob = await this.db.selectOne('pdf\_jobs', { status: 'waiting' }, {  
 sort: { createdAt: 1 }  
 });  
  
 // Calculate throughput  
 const now = new Date();  
 const oneDayAgo = new Date(now.getTime() - 24 \* 60 \* 60 \* 1000);  
 const sevenDaysAgo = new Date(now.getTime() - 7 \* 24 \* 60 \* 60 \* 1000);  
  
 const completedLast24h = await this.db.count('pdf\_jobs', {  
 status: 'completed',  
 completedAt: { $gte: oneDayAgo }  
 });  
  
 const completedLast7d = await this.db.count('pdf\_jobs', {  
 status: 'completed',  
 completedAt: { $gte: sevenDaysAgo }  
 });  
  
 // Calculate average processing time  
 const recentJobs = await this.db.select('pdf\_jobs', {  
 status: 'completed',  
 completedAt: { $gte: sevenDaysAgo }  
 });  
  
 let totalProcessingTime = 0;  
 let jobsWithTime = 0;  
  
 recentJobs.forEach(job => {  
 if (job.startedAt && job.completedAt) {  
 totalProcessingTime += job.completedAt.getTime() - job.startedAt.getTime();  
 jobsWithTime++;  
 }  
 });  
  
 const averageProcessingTime = jobsWithTime > 0   
 ? totalProcessingTime / jobsWithTime   
 : 0;  
  
 return {  
 queueId: 'pdf',  
 name: 'PDF Processing Queue',  
 status: 'active',  
 jobCount: {  
 waiting,  
 processing,  
 completed,  
 failed  
 },  
 throughput: {  
 last24h: completedLast24h,  
 last7d: completedLast7d  
 },  
 averageProcessingTime,  
 oldestJob: oldestJob?.createdAt || null  
 };  
 }  
  
 /\*\*  
 \* Subscribe to events from other queues  
 \*/  
 private subscribeToExternalEvents(): void {  
 // Listen for knowledge base events  
 this.messageBroker.subscribe('knowledge-base', 'material.imported', async (payload) => {  
 if (payload.source === 'pdf' && payload.jobId) {  
 const job = await this.getJob(payload.jobId);  
 if (job) {  
 await this.updateJob(payload.jobId, {  
 results: {  
 ...job.results,  
 materialsImported: payload.count  
 }  
 });  
 }  
 }  
 });  
 }  
}

### Queue Events Service

The Queue Events Service provides a client-side interface for subscribing to queue events:

// Queue Events Service  
export class QueueEventsService {  
 private supabase: SupabaseClient;  
 private channels: Record<string, RealtimeChannel> = {};  
 private subscriptions: Record<string, () => void> = {};  
  
 constructor(supabaseClient: SupabaseClient) {  
 this.supabase = supabaseClient;  
 }  
  
 /\*\*  
 \* Subscribe to queue events  
 \*/  
 public subscribe(  
 queueId: string,  
 eventTypes: string[],  
 handler: (event: QueueEvent) => void  
 ): () => void {  
 // Create channel ID  
 const channelId = `queue-${queueId}`;  
  
 // Initialize channel if it doesn't exist  
 if (!this.channels[channelId]) {  
 this.channels[channelId] = this.supabase.channel(channelId);  
  
 // Subscribe to channel  
 this.channels[channelId].subscribe(status => {  
 if (status === 'SUBSCRIBED') {  
 console.log(`Subscribed to queue events: ${queueId}`);  
 }  
 });  
 }  
  
 // Subscribe to each event type  
 eventTypes.forEach(eventType => {  
 const subscriptionId = `${channelId}-${eventType}`;  
  
 // Unsubscribe if already subscribed  
 if (this.subscriptions[subscriptionId]) {  
 this.subscriptions[subscriptionId]();  
 }  
  
 // Subscribe to event  
 this.channels[channelId].on('broadcast', { event: eventType }, payload => {  
 handler({  
 type: eventType,  
 queueId,  
 data: payload.payload,  
 timestamp: payload.payload.timestamp || Date.now()  
 });  
 });  
  
 // Store unsubscribe function  
 this.subscriptions[subscriptionId] = () => {  
 this.channels[channelId].off('broadcast', { event: eventType });  
 };  
 });  
  
 // Return unsubscribe function  
 return () => {  
 eventTypes.forEach(eventType => {  
 const subscriptionId = `${channelId}-${eventType}`;  
 if (this.subscriptions[subscriptionId]) {  
 this.subscriptions[subscriptionId]();  
 delete this.subscriptions[subscriptionId];  
 }  
 });  
 };  
 }  
  
 /\*\*  
 \* Subscribe to PDF queue events  
 \*/  
 public subscribeToPDFEvents(  
 handler: (event: QueueEvent) => void  
 ): () => void {  
 return this.subscribe('pdf', [  
 'pdf.job.queued',  
 'pdf.job.started',  
 'pdf.job.progress',  
 'pdf.job.completed',  
 'pdf.job.failed'  
 ], handler);  
 }  
  
 /\*\*  
 \* Subscribe to crawler queue events  
 \*/  
 public subscribeToCrawlerEvents(  
 handler: (event: QueueEvent) => void  
 ): () => void {  
 return this.subscribe('crawler', [  
 'crawler.job.queued',  
 'crawler.job.started',  
 'crawler.job.progress',  
 'crawler.job.completed',  
 'crawler.job.failed'  
 ], handler);  
 }  
  
 /\*\*  
 \* Subscribe to training queue events  
 \*/  
 public subscribeToTrainingEvents(  
 handler: (event: QueueEvent) => void  
 ): () => void {  
 return this.subscribe('training', [  
 'training.job.queued',  
 'training.job.started',  
 'training.job.progress',  
 'training.job.completed',  
 'training.job.failed',  
 'training.model.deployed'  
 ], handler);  
 }  
  
 /\*\*  
 \* Unsubscribe from all events  
 \*/  
 public unsubscribeAll(): void {  
 // Unsubscribe from all subscriptions  
 Object.values(this.subscriptions).forEach(unsubscribe => unsubscribe());  
 this.subscriptions = {};  
  
 // Unsubscribe from all channels  
 Object.values(this.channels).forEach(channel => channel.unsubscribe());  
 this.channels = {};  
 }  
}  
  
// Export singleton instance  
export const queueEvents = new QueueEventsService(supabaseClient);

// Queue Events Service  
export class QueueEventsService {  
 private supabase: SupabaseClient;  
 private channels: Record<string, RealtimeChannel> = {};  
 private subscriptions: Record<string, () => void> = {};  
  
 constructor(supabaseClient: SupabaseClient) {  
 this.supabase = supabaseClient;  
 }  
  
 /\*\*  
 \* Subscribe to queue events  
 \*/  
 public subscribe(  
 queueId: string,  
 eventTypes: string[],  
 handler: (event: QueueEvent) => void  
 ): () => void {  
 // Create channel ID  
 const channelId = `queue-${queueId}`;  
  
 // Initialize channel if it doesn't exist  
 if (!this.channels[channelId]) {  
 this.channels[channelId] = this.supabase.channel(channelId);  
  
 // Subscribe to channel  
 this.channels[channelId].subscribe(status => {  
 if (status === 'SUBSCRIBED') {  
 console.log(`Subscribed to queue events: ${queueId}`);  
 }  
 });  
 }  
  
 // Subscribe to each event type  
 eventTypes.forEach(eventType => {  
 const subscriptionId = `${channelId}-${eventType}`;  
  
 // Unsubscribe if already subscribed  
 if (this.subscriptions[subscriptionId]) {  
 this.subscriptions[subscriptionId]();  
 }  
  
 // Subscribe to event  
 this.channels[channelId].on('broadcast', { event: eventType }, payload => {  
 handler({  
 type: eventType,  
 queueId,  
 data: payload.payload,  
 timestamp: payload.payload.timestamp || Date.now()  
 });  
 });  
  
 // Store unsubscribe function  
 this.subscriptions[subscriptionId] = () => {  
 this.channels[channelId].off('broadcast', { event: eventType });  
 };  
 });  
  
 // Return unsubscribe function  
 return () => {  
 eventTypes.forEach(eventType => {  
 const subscriptionId = `${channelId}-${eventType}`;  
 if (this.subscriptions[subscriptionId]) {  
 this.subscriptions[subscriptionId]();  
 delete this.subscriptions[subscriptionId];  
 }  
 });  
 };  
 }  
  
 /\*\*  
 \* Subscribe to PDF queue events  
 \*/  
 public subscribeToPDFEvents(  
 handler: (event: QueueEvent) => void  
 ): () => void {  
 return this.subscribe('pdf', [  
 'pdf.job.queued',  
 'pdf.job.started',  
 'pdf.job.progress',  
 'pdf.job.completed',  
 'pdf.job.failed'  
 ], handler);  
 }  
  
 /\*\*  
 \* Subscribe to crawler queue events  
 \*/  
 public subscribeToCrawlerEvents(  
 handler: (event: QueueEvent) => void  
 ): () => void {  
 return this.subscribe('crawler', [  
 'crawler.job.queued',  
 'crawler.job.started',  
 'crawler.job.progress',  
 'crawler.job.completed',  
 'crawler.job.failed'  
 ], handler);  
 }  
  
 /\*\*  
 \* Subscribe to training queue events  
 \*/  
 public subscribeToTrainingEvents(  
 handler: (event: QueueEvent) => void  
 ): () => void {  
 return this.subscribe('training', [  
 'training.job.queued',  
 'training.job.started',  
 'training.job.progress',  
 'training.job.completed',  
 'training.job.failed',  
 'training.model.deployed'  
 ], handler);  
 }  
  
 /\*\*  
 \* Unsubscribe from all events  
 \*/  
 public unsubscribeAll(): void {  
 // Unsubscribe from all subscriptions  
 Object.values(this.subscriptions).forEach(unsubscribe => unsubscribe());  
 this.subscriptions = {};  
  
 // Unsubscribe from all channels  
 Object.values(this.channels).forEach(channel => channel.unsubscribe());  
 this.channels = {};  
 }  
}  
  
// Export singleton instance  
export const queueEvents = new QueueEventsService(supabaseClient);

### Queue Dashboard Implementation

The Queue Dashboard uses the Queue Events Service to display real-time queue status:

// Queue Dashboard Component  
import React, { useEffect, useState } from 'react';  
import { queueEvents } from '../../services/queueEvents.service';  
  
interface QueueStats {  
 queueId: string;  
 name: string;  
 status: 'active' | 'paused' | 'error';  
 jobCount: {  
 waiting: number;  
 processing: number;  
 completed: number;  
 failed: number;  
 };  
 throughput: {  
 last1h: number;  
 last24h: number;  
 last7d: number;  
 };  
 averageProcessingTime: number;  
 oldestJob: Date | null;  
}  
  
interface QueueEvent {  
 type: string;  
 queueId: string;  
 data: any;  
 timestamp: number;  
}  
  
export const QueueDashboard: React.FC = () => {  
 const [pdfStats, setPdfStats] = useState<QueueStats | null>(null);  
 const [crawlerStats, setCrawlerStats] = useState<QueueStats | null>(null);  
 const [recentEvents, setRecentEvents] = useState<QueueEvent[]>([]);  
 const [isLoading, setIsLoading] = useState(true);  
  
 useEffect(() => {  
 // Fetch initial queue stats  
 const fetchQueueStats = async () => {  
 try {  
 const response = await fetch('/api/admin/queues');  
 const data = await response.json();  
  
 // Update stats  
 data.queues.forEach((queueStats: QueueStats) => {  
 if (queueStats.queueId === 'pdf') {  
 setPdfStats(queueStats);  
 } else if (queueStats.queueId === 'crawler') {  
 setCrawlerStats(queueStats);  
 }  
 });  
  
 setIsLoading(false);  
 } catch (error) {  
 console.error('Failed to fetch queue stats:', error);  
 setIsLoading(false);  
 }  
 };  
  
 fetchQueueStats();  
  
 // Subscribe to queue events  
 const unsubscribePdf = queueEvents.subscribeToPDFEvents(handleQueueEvent);  
 const unsubscribeCrawler = queueEvents.subscribeToCrawlerEvents(handleQueueEvent);  
  
 return () => {  
 unsubscribePdf();  
 unsubscribeCrawler();  
 };  
 }, []);  
  
 // Handle queue events  
 const handleQueueEvent = (event: QueueEvent) => {  
 // Update recent events list  
 setRecentEvents(prevEvents => {  
 const newEvents = [event, ...prevEvents].slice(0, 10);  
 return newEvents;  
 });  
  
 // Update queue stats based on event  
 if (event.queueId === 'pdf') {  
 if (event.type === 'pdf.job.queued') {  
 setPdfStats(prev => prev ? {  
 ...prev,  
 jobCount: {  
 ...prev.jobCount,  
 waiting: prev.jobCount.waiting + 1  
 }  
 } : null);  
 } else if (event.type === 'pdf.job.started') {  
 setPdfStats(prev => prev ? {  
 ...prev,  
 jobCount: {  
 ...prev.jobCount,  
 waiting: Math.max(0, prev.jobCount.waiting - 1),  
 processing: prev.jobCount.processing + 1  
 }  
 } : null);  
 } else if (event.type === 'pdf.job.completed') {  
 setPdfStats(prev => prev ? {  
 ...prev,  
 jobCount: {  
 ...prev.jobCount,  
 processing: Math.max(0, prev.jobCount.processing - 1),  
 completed: prev.jobCount.completed + 1  
 },  
 throughput: {  
 ...prev.throughput,  
 last1h: prev.throughput.last1h + 1,  
 last24h: prev.throughput.last24h + 1,  
 last7d: prev.throughput.last7d + 1  
 }  
 } : null);  
 } else if (event.type === 'pdf.job.failed') {  
 setPdfStats(prev => prev ? {  
 ...prev,  
 jobCount: {  
 ...prev.jobCount,  
 processing: Math.max(0, prev.jobCount.processing - 1),  
 failed: prev.jobCount.failed + 1  
 }  
 } : null);  
 }  
 } else if (event.queueId === 'crawler') {  
 // Similar updates for crawler stats  
 }  
 };  
  
 if (isLoading) {  
 return <div>Loading queue statistics...</div>;  
 }  
  
 return (  
 <div className="queue-dashboard">  
 <h2>Queue Dashboard</h2>  
  
 <div className="queue-stats-container">  
 {pdfStats && (  
 <div className="queue-stats-card">  
 <h3>{pdfStats.name}</h3>  
 <div className="status-indicator status-{pdfStats.status}">  
 {pdfStats.status}  
 </div>  
  
 <div className="stats-grid">  
 <div className="stat-item">  
 <div className="stat-label">Waiting</div>  
 <div className="stat-value">{pdfStats.jobCount.waiting}</div>  
 </div>  
 <div className="stat-item">  
 <div className="stat-label">Processing</div>  
 <div className="stat-value">{pdfStats.jobCount.processing}</div>  
 </div>  
 <div className="stat-item">  
 <div className="stat-label">Completed</div>  
 <div className="stat-value">{pdfStats.jobCount.completed}</div>  
 </div>  
 <div className="stat-item">  
 <div className="stat-label">Failed</div>  
 <div className="stat-value">{pdfStats.jobCount.failed}</div>  
 </div>  
 </div>  
  
 <div className="throughput-section">  
 <h4>Throughput</h4>  
 <div>Last 24h: {pdfStats.throughput.last24h} jobs</div>  
 <div>Last 7d: {pdfStats.throughput.last7d} jobs</div>  
 </div>  
  
 <div className="processing-time">  
 <h4>Average Processing Time</h4>  
 <div>{formatTime(pdfStats.averageProcessingTime)}</div>  
 </div>  
  
 {pdfStats.oldestJob && (  
 <div className="oldest-job">  
 <h4>Oldest Waiting Job</h4>  
 <div>{formatDate(pdfStats.oldestJob)}</div>  
 </div>  
 )}  
  
 <div className="action-buttons">  
 <button onClick={() => viewQueueJobs('pdf')}>View Jobs</button>  
 <button onClick={() => pauseQueue('pdf')}>  
 {pdfStats.status === 'paused' ? 'Resume' : 'Pause'}  
 </button>  
 </div>  
 </div>  
 )}  
  
 {/\* Similar card for crawler stats \*/}  
 </div>  
  
 <div className="recent-events">  
 <h3>Recent Events</h3>  
 <ul className="event-list">  
 {recentEvents.map((event, index) => (  
 <li key={index} className={`event-item event-${event.type.split('.')[1]}`}>  
 <div className="event-time">{formatTime(event.timestamp)}</div>  
 <div className="event-queue">{event.queueId}</div>  
 <div className="event-type">{event.type}</div>  
 <div className="event-data">  
 {event.data.jobId && <span>Job: {event.data.jobId}</span>}  
 {event.data.progress && (  
 <span>Progress: {event.data.progress.percentComplete}%</span>  
 )}  
 </div>  
 </li>  
 ))}  
 </ul>  
 </div>  
 </div>  
 );  
};  
  
// Helper functions  
const formatTime = (milliseconds: number) => {  
 const seconds = Math.floor(milliseconds / 1000);  
 const minutes = Math.floor(seconds / 60);  
 const hours = Math.floor(minutes / 60);  
  
 if (hours > 0) {  
 return `${hours}h ${minutes % 60}m`;  
 } else if (minutes > 0) {  
 return `${minutes}m ${seconds % 60}s`;  
 } else {  
 return `${seconds}s`;  
 }  
};  
  
const formatDate = (date: Date) => {  
 return new Date(date).toLocaleString();  
};  
  
const viewQueueJobs = (queueId: string) => {  
 window.location.href = `/admin/queues/${queueId}/jobs`;  
};  
  
const pauseQueue = async (queueId: string) => {  
 try {  
 await fetch(`/api/admin/queues/${queueId}/toggle`, {  
 method: 'POST'  
 });  
 } catch (error) {  
 console.error(`Failed to toggle queue ${queueId}:`, error);  
 }  
};

// Queue Dashboard Component  
import React, { useEffect, useState } from 'react';  
import { queueEvents } from '../../services/queueEvents.service';  
  
interface QueueStats {  
 queueId: string;  
 name: string;  
 status: 'active' | 'paused' | 'error';  
 jobCount: {  
 waiting: number;  
 processing: number;  
 completed: number;  
 failed: number;  
 };  
 throughput: {  
 last1h: number;  
 last24h: number;  
 last7d: number;  
 };  
 averageProcessingTime: number;  
 oldestJob: Date | null;  
}  
  
interface QueueEvent {  
 type: string;  
 queueId: string;  
 data: any;  
 timestamp: number;  
}  
  
export const QueueDashboard: React.FC = () => {  
 const [pdfStats, setPdfStats] = useState<QueueStats | null>(null);  
 const [crawlerStats, setCrawlerStats] = useState<QueueStats | null>(null);  
 const [recentEvents, setRecentEvents] = useState<QueueEvent[]>([]);  
 const [isLoading, setIsLoading] = useState(true);  
  
 useEffect(() => {  
 // Fetch initial queue stats  
 const fetchQueueStats = async () => {  
 try {  
 const response = await fetch('/api/admin/queues');  
 const data = await response.json();  
  
 // Update stats  
 data.queues.forEach((queueStats: QueueStats) => {  
 if (queueStats.queueId === 'pdf') {  
 setPdfStats(queueStats);  
 } else if (queueStats.queueId === 'crawler') {  
 setCrawlerStats(queueStats);  
 }  
 });  
  
 setIsLoading(false);  
 } catch (error) {  
 console.error('Failed to fetch queue stats:', error);  
 setIsLoading(false);  
 }  
 };  
  
 fetchQueueStats();  
  
 // Subscribe to queue events  
 const unsubscribePdf = queueEvents.subscribeToPDFEvents(handleQueueEvent);  
 const unsubscribeCrawler = queueEvents.subscribeToCrawlerEvents(handleQueueEvent);  
  
 return () => {  
 unsubscribePdf();  
 unsubscribeCrawler();  
 };  
 }, []);  
  
 // Handle queue events  
 const handleQueueEvent = (event: QueueEvent) => {  
 // Update recent events list  
 setRecentEvents(prevEvents => {  
 const newEvents = [event, ...prevEvents].slice(0, 10);  
 return newEvents;  
 });  
  
 // Update queue stats based on event  
 if (event.queueId === 'pdf') {  
 if (event.type === 'pdf.job.queued') {  
 setPdfStats(prev => prev ? {  
 ...prev,  
 jobCount: {  
 ...prev.jobCount,  
 waiting: prev.jobCount.waiting + 1  
 }  
 } : null);  
 } else if (event.type === 'pdf.job.started') {  
 setPdfStats(prev => prev ? {  
 ...prev,  
 jobCount: {  
 ...prev.jobCount,  
 waiting: Math.max(0, prev.jobCount.waiting - 1),  
 processing: prev.jobCount.processing + 1  
 }  
 } : null);  
 } else if (event.type === 'pdf.job.completed') {  
 setPdfStats(prev => prev ? {  
 ...prev,  
 jobCount: {  
 ...prev.jobCount,  
 processing: Math.max(0, prev.jobCount.processing - 1),  
 completed: prev.jobCount.completed + 1  
 },  
 throughput: {  
 ...prev.throughput,  
 last1h: prev.throughput.last1h + 1,  
 last24h: prev.throughput.last24h + 1,  
 last7d: prev.throughput.last7d + 1  
 }  
 } : null);  
 } else if (event.type === 'pdf.job.failed') {  
 setPdfStats(prev => prev ? {  
 ...prev,  
 jobCount: {  
 ...prev.jobCount,  
 processing: Math.max(0, prev.jobCount.processing - 1),  
 failed: prev.jobCount.failed + 1  
 }  
 } : null);  
 }  
 } else if (event.queueId === 'crawler') {  
 // Similar updates for crawler stats  
 }  
 };  
  
 if (isLoading) {  
 return <div>Loading queue statistics...</div>;  
 }  
  
 return (  
 <div className="queue-dashboard">  
 <h2>Queue Dashboard</h2>  
  
 <div className="queue-stats-container">  
 {pdfStats && (  
 <div className="queue-stats-card">  
 <h3>{pdfStats.name}</h3>  
 <div className="status-indicator status-{pdfStats.status}">  
 {pdfStats.status}  
 </div>  
  
 <div className="stats-grid">  
 <div className="stat-item">  
 <div className="stat-label">Waiting</div>  
 <div className="stat-value">{pdfStats.jobCount.waiting}</div>  
 </div>  
 <div className="stat-item">  
 <div className="stat-label">Processing</div>  
 <div className="stat-value">{pdfStats.jobCount.processing}</div>  
 </div>  
 <div className="stat-item">  
 <div className="stat-label">Completed</div>  
 <div className="stat-value">{pdfStats.jobCount.completed}</div>  
 </div>  
 <div className="stat-item">  
 <div className="stat-label">Failed</div>  
 <div className="stat-value">{pdfStats.jobCount.failed}</div>  
 </div>  
 </div>  
  
 <div className="throughput-section">  
 <h4>Throughput</h4>  
 <div>Last 24h: {pdfStats.throughput.last24h} jobs</div>  
 <div>Last 7d: {pdfStats.throughput.last7d} jobs</div>  
 </div>  
  
 <div className="processing-time">  
 <h4>Average Processing Time</h4>  
 <div>{formatTime(pdfStats.averageProcessingTime)}</div>  
 </div>  
  
 {pdfStats.oldestJob && (  
 <div className="oldest-job">  
 <h4>Oldest Waiting Job</h4>  
 <div>{formatDate(pdfStats.oldestJob)}</div>  
 </div>  
 )}  
  
 <div className="action-buttons">  
 <button onClick={() => viewQueueJobs('pdf')}>View Jobs</button>  
 <button onClick={() => pauseQueue('pdf')}>  
 {pdfStats.status === 'paused' ? 'Resume' : 'Pause'}  
 </button>  
 </div>  
 </div>  
 )}  
  
 {/\* Similar card for crawler stats \*/}  
 </div>  
  
 <div className="recent-events">  
 <h3>Recent Events</h3>  
 <ul className="event-list">  
 {recentEvents.map((event, index) => (  
 <li key={index} className={`event-item event-${event.type.split('.')[1]}`}>  
 <div className="event-time">{formatTime(event.timestamp)}</div>  
 <div className="event-queue">{event.queueId}</div>  
 <div className="event-type">{event.type}</div>  
 <div className="event-data">  
 {event.data.jobId && <span>Job: {event.data.jobId}</span>}  
 {event.data.progress && (  
 <span>Progress: {event.data.progress.percentComplete}%</span>  
 )}  
 </div>  
 </li>  
 ))}  
 </ul>  
 </div>  
 </div>  
 );  
};  
  
// Helper functions  
const formatTime = (milliseconds: number) => {  
 const seconds = Math.floor(milliseconds / 1000);  
 const minutes = Math.floor(seconds / 60);  
 const hours = Math.floor(minutes / 60);  
  
 if (hours > 0) {  
 return `${hours}h ${minutes % 60}m`;  
 } else if (minutes > 0) {  
 return `${minutes}m ${seconds % 60}s`;  
 } else {  
 return `${seconds}s`;  
 }  
};  
  
const formatDate = (date: Date) => {  
 return new Date(date).toLocaleString();  
};  
  
const viewQueueJobs = (queueId: string) => {  
 window.location.href = `/admin/queues/${queueId}/jobs`;  
};  
  
const pauseQueue = async (queueId: string) => {  
 try {  
 await fetch(`/api/admin/queues/${queueId}/toggle`, {  
 method: 'POST'  
 });  
 } catch (error) {  
 console.error(`Failed to toggle queue ${queueId}:`, error);  
 }  
};

### API Usage

#### Creating a PDF Processing Job

import { pdfQueue } from '../services/pdf/pdfQueue';  
  
async function processPDFCatalog(filePath: string, fileName: string, fileSize: number) {  
 try {  
 // Create a new PDF processing job  
 const jobId = await pdfQueue.createJob({  
 fileName,  
 fileSize,  
 filePath,  
 priority: 5,  
 options: {  
 extractImages: true,  
 extractText: true,  
 enhanceResolution: true,  
 associateTextWithImages: true,  
 extractStructuredData: true  
 }  
 });  
  
 console.log(`PDF processing job created: ${jobId}`);  
  
 // Optionally, subscribe to job events  
 const unsubscribe = pdfQueue.subscribeToEvents(['pdf.job.completed', 'pdf.job.failed'], async (event) => {  
 if (event.data.jobId === jobId) {  
 if (event.type === 'pdf.job.completed') {  
 console.log(`Job completed with results:`, event.data.results);  
  
 // Do something with the results  
 if (event.data.results && event.data.results.materialsIdentified > 0) {  
 // Handle extracted materials  
 }  
 } else if (event.type === 'pdf.job.failed') {  
 console.error(`Job failed:`, event.data.error);  
 // Handle failure  
 }  
  
 // Unsubscribe after handling event  
 unsubscribe();  
 }  
 });  
  
 return jobId;  
 } catch (error) {  
 console.error('Failed to create PDF processing job:', error);  
 throw error;  
 }  
}

import { pdfQueue } from '../services/pdf/pdfQueue';  
  
async function processPDFCatalog(filePath: string, fileName: string, fileSize: number) {  
 try {  
 // Create a new PDF processing job  
 const jobId = await pdfQueue.createJob({  
 fileName,  
 fileSize,  
 filePath,  
 priority: 5,  
 options: {  
 extractImages: true,  
 extractText: true,  
 enhanceResolution: true,  
 associateTextWithImages: true,  
 extractStructuredData: true  
 }  
 });  
  
 console.log(`PDF processing job created: ${jobId}`);  
  
 // Optionally, subscribe to job events  
 const unsubscribe = pdfQueue.subscribeToEvents(['pdf.job.completed', 'pdf.job.failed'], async (event) => {  
 if (event.data.jobId === jobId) {  
 if (event.type === 'pdf.job.completed') {  
 console.log(`Job completed with results:`, event.data.results);  
  
 // Do something with the results  
 if (event.data.results && event.data.results.materialsIdentified > 0) {  
 // Handle extracted materials  
 }  
 } else if (event.type === 'pdf.job.failed') {  
 console.error(`Job failed:`, event.data.error);  
 // Handle failure  
 }  
  
 // Unsubscribe after handling event  
 unsubscribe();  
 }  
 });  
  
 return jobId;  
 } catch (error) {  
 console.error('Failed to create PDF processing job:', error);  
 throw error;  
 }  
}

#### Monitoring Job Progress in UI

import React, { useEffect, useState } from 'react';  
import { queueEvents } from '../../services/queueEvents.service';  
  
interface JobProgress {  
 currentPage: number;  
 totalPages: number;  
 percentComplete: number;  
 currentStage: string;  
 stageProgress: number;  
}  
  
interface JobDetails {  
 jobId: string;  
 status: 'waiting' | 'processing' | 'completed' | 'failed';  
 fileName: string;  
 fileSize: number;  
 progress: JobProgress;  
 results?: Record<string, number>;  
 error?: string;  
 createdAt: string;  
 startedAt?: string;  
 completedAt?: string;  
}  
  
export const JobProgressTracker: React.FC<{ jobId: string }> = ({ jobId }) => {  
 const [jobDetails, setJobDetails] = useState<JobDetails | null>(null);  
 const [isLoading, setIsLoading] = useState(true);  
  
 useEffect(() => {  
 // Fetch initial job details  
 const fetchJobDetails = async () => {  
 try {  
 const response = await fetch(`/api/pdf/jobs/${jobId}`);  
 const data = await response.json();  
 setJobDetails(data);  
 setIsLoading(false);  
 } catch (error) {  
 console.error('Failed to fetch job details:', error);  
 setIsLoading(false);  
 }  
 };  
  
 fetchJobDetails();  
  
 // Subscribe to job events  
 const unsubscribe = queueEvents.subscribeToPDFEvents((event) => {  
 if (event.data.jobId === jobId) {  
 // Update job details based on event type  
 if (event.type === 'pdf.job.progress') {  
 setJobDetails(prev => prev ? {  
 ...prev,  
 progress: event.data.progress  
 } : null);  
 } else if (event.type === 'pdf.job.completed') {  
 setJobDetails(prev => prev ? {  
 ...prev,  
 status: 'completed',  
 results: event.data.results,  
 completedAt: new Date(event.timestamp).toISOString()  
 } : null);  
 } else if (event.type === 'pdf.job.failed') {  
 setJobDetails(prev => prev ? {  
 ...prev,  
 status: 'failed',  
 error: event.data.error,  
 completedAt: new Date(event.timestamp).toISOString()  
 } : null);  
 }  
 }  
 });  
  
 return () => {  
 unsubscribe();  
 };  
 }, [jobId]);  
  
 if (isLoading) {  
 return <div>Loading job details...</div>;  
 }  
  
 if (!jobDetails) {  
 return <div>Job not found</div>;  
 }  
  
 return (  
 <div className="job-progress-tracker">  
 <h2>Job Progress: {jobDetails.fileName}</h2>  
  
 <div className="job-status">  
 Status: <span className={`status-${jobDetails.status}`}>{jobDetails.status}</span>  
 </div>  
  
 {jobDetails.status === 'processing' && (  
 <div className="progress-section">  
 <div className="progress-bar">  
 <div   
 className="progress-fill"  
 style={{ width: `${jobDetails.progress.percentComplete}%` }}  
 />  
 </div>  
 <div className="progress-text">  
 {jobDetails.progress.percentComplete}% complete  
 </div>  
 <div className="stage-info">  
 Current stage: {jobDetails.progress.currentStage}  
 <div className="stage-progress-bar">  
 <div   
 className="stage-progress-fill"  
 style={{ width: `${jobDetails.progress.stageProgress}%` }}  
 />  
 </div>  
 </div>  
 <div className="page-progress">  
 Processing page {jobDetails.progress.currentPage} of {jobDetails.progress.totalPages}  
 </div>  
 </div>  
 )}  
  
 {jobDetails.status === 'completed' && jobDetails.results && (  
 <div className="results-section">  
 <h3>Processing Results</h3>  
 <div className="results-grid">  
 <div className="result-item">  
 <div className="result-label">Images Extracted</div>  
 <div className="result-value">{jobDetails.results.imagesExtracted}</div>  
 </div>  
 <div className="result-item">  
 <div className="result-label">Text Regions</div>  
 <div className="result-value">{jobDetails.results.textRegionsExtracted}</div>  
 </div>  
 <div className="result-item">  
 <div className="result-label">Tables Detected</div>  
 <div className="result-value">{jobDetails.results.tablesDetected}</div>  
 </div>  
 <div className="result-item">  
 <div className="result-label">Materials Identified</div>  
 <div className="result-value">{jobDetails.results.materialsIdentified}</div>  
 </div>  
 {jobDetails.results.materialsImported !== undefined && (  
 <div className="result-item">  
 <div className="result-label">Materials Imported</div>  
 <div className="result-value">{jobDetails.results.materialsImported}</div>  
 </div>  
 )}  
 </div>  
  
 <div className="action-buttons">  
 <button onClick={() => viewResults(jobId)}>View Results</button>  
 <button onClick={() => downloadResults(jobId)}>Download Results</button>  
 </div>  
 </div>  
 )}  
  
 {jobDetails.status === 'failed' && (  
 <div className="error-section">  
 <h3>Processing Error</h3>  
 <div className="error-message">  
 {jobDetails.error || 'Unknown error occurred'}  
 </div>  
 <div className="action-buttons">  
 <button onClick={() => retryJob(jobId)}>Retry Job</button>  
 </div>  
 </div>  
 )}  
  
 <div className="job-details">  
 <h3>Job Details</h3>  
 <div className="details-grid">  
 <div className="detail-item">  
 <div className="detail-label">Job ID</div>  
 <div className="detail-value">{jobDetails.jobId}</div>  
 </div>  
 <div className="detail-item">  
 <div className="detail-label">File Name</div>  
 <div className="detail-value">{jobDetails.fileName}</div>  
 </div>  
 <div className="detail-item">  
 <div className="detail-label">File Size</div>  
 <div className="detail-value">{formatFileSize(jobDetails.fileSize)}</div>  
 </div>  
 <div className="detail-item">  
 <div className="detail-label">Created</div>  
 <div className="detail-value">{formatDate(jobDetails.createdAt)}</div>  
 </div>  
 {jobDetails.startedAt && (  
 <div className="detail-item">  
 <div className="detail-label">Started</div>  
 <div className="detail-value">{formatDate(jobDetails.startedAt)}</div>  
 </div>  
 )}  
 {jobDetails.completedAt && (  
 <div className="detail-item">  
 <div className="detail-label">Completed</div>  
 <div className="detail-value">{formatDate(jobDetails.completedAt)}</div>  
 </div>  
 )}  
 </div>  
 </div>  
 </div>  
 );  
};  
  
// Helper functions  
const formatFileSize = (bytes: number) => {  
 if (bytes < 1024) {  
 return `${bytes} B`;  
 } else if (bytes < 1024 \* 1024) {  
 return `${(bytes / 1024).toFixed(2)} KB`;  
 } else if (bytes < 1024 \* 1024 \* 1024) {  
 return `${(bytes / (1024 \* 1024)).toFixed(2)} MB`;  
 } else {  
 return `${(bytes / (1024 \* 1024 \* 1024)).toFixed(2)} GB`;  
 }  
};  
  
const formatDate = (dateString: string) => {  
 return new Date(dateString).toLocaleString();  
};  
  
const viewResults = (jobId: string) => {  
 window.location.href = `/pdf/jobs/${jobId}/results`;  
};  
  
const downloadResults = (jobId: string) => {  
 window.location.href = `/api/pdf/jobs/${jobId}/results/download`;  
};  
  
const retryJob = async (jobId: string) => {  
 try {  
 await fetch(`/api/pdf/jobs/${jobId}/retry`, {  
 method: 'POST'  
 });  
 } catch (error) {  
 console.error(`Failed to retry job ${jobId}:`, error);  
 }  
};

import React, { useEffect, useState } from 'react';  
import { queueEvents } from '../../services/queueEvents.service';  
  
interface JobProgress {  
 currentPage: number;  
 totalPages: number;  
 percentComplete: number;  
 currentStage: string;  
 stageProgress: number;  
}  
  
interface JobDetails {  
 jobId: string;  
 status: 'waiting' | 'processing' | 'completed' | 'failed';  
 fileName: string;  
 fileSize: number;  
 progress: JobProgress;  
 results?: Record<string, number>;  
 error?: string;  
 createdAt: string;  
 startedAt?: string;  
 completedAt?: string;  
}  
  
export const JobProgressTracker: React.FC<{ jobId: string }> = ({ jobId }) => {  
 const [jobDetails, setJobDetails] = useState<JobDetails | null>(null);  
 const [isLoading, setIsLoading] = useState(true);  
  
 useEffect(() => {  
 // Fetch initial job details  
 const fetchJobDetails = async () => {  
 try {  
 const response = await fetch(`/api/pdf/jobs/${jobId}`);  
 const data = await response.json();  
 setJobDetails(data);  
 setIsLoading(false);  
 } catch (error) {  
 console.error('Failed to fetch job details:', error);  
 setIsLoading(false);  
 }  
 };  
  
 fetchJobDetails();  
  
 // Subscribe to job events  
 const unsubscribe = queueEvents.subscribeToPDFEvents((event) => {  
 if (event.data.jobId === jobId) {  
 // Update job details based on event type  
 if (event.type === 'pdf.job.progress') {  
 setJobDetails(prev => prev ? {  
 ...prev,  
 progress: event.data.progress  
 } : null);  
 } else if (event.type === 'pdf.job.completed') {  
 setJobDetails(prev => prev ? {  
 ...prev,  
 status: 'completed',  
 results: event.data.results,  
 completedAt: new Date(event.timestamp).toISOString()  
 } : null);  
 } else if (event.type === 'pdf.job.failed') {  
 setJobDetails(prev => prev ? {  
 ...prev,  
 status: 'failed',  
 error: event.data.error,  
 completedAt: new Date(event.timestamp).toISOString()  
 } : null);  
 }  
 }  
 });  
  
 return () => {  
 unsubscribe();  
 };  
 }, [jobId]);  
  
 if (isLoading) {  
 return <div>Loading job details...</div>;  
 }  
  
 if (!jobDetails) {  
 return <div>Job not found</div>;  
 }  
  
 return (  
 <div className="job-progress-tracker">  
 <h2>Job Progress: {jobDetails.fileName}</h2>  
  
 <div className="job-status">  
 Status: <span className={`status-${jobDetails.status}`}>{jobDetails.status}</span>  
 </div>  
  
 {jobDetails.status === 'processing' && (  
 <div className="progress-section">  
 <div className="progress-bar">  
 <div   
 className="progress-fill"  
 style={{ width: `${jobDetails.progress.percentComplete}%` }}  
 />  
 </div>  
 <div className="progress-text">  
 {jobDetails.progress.percentComplete}% complete  
 </div>  
 <div className="stage-info">  
 Current stage: {jobDetails.progress.currentStage}  
 <div className="stage-progress-bar">  
 <div   
 className="stage-progress-fill"  
 style={{ width: `${jobDetails.progress.stageProgress}%` }}  
 />  
 </div>  
 </div>  
 <div className="page-progress">  
 Processing page {jobDetails.progress.currentPage} of {jobDetails.progress.totalPages}  
 </div>  
 </div>  
 )}  
  
 {jobDetails.status === 'completed' && jobDetails.results && (  
 <div className="results-section">  
 <h3>Processing Results</h3>  
 <div className="results-grid">  
 <div className="result-item">  
 <div className="result-label">Images Extracted</div>  
 <div className="result-value">{jobDetails.results.imagesExtracted}</div>  
 </div>  
 <div className="result-item">  
 <div className="result-label">Text Regions</div>  
 <div className="result-value">{jobDetails.results.textRegionsExtracted}</div>  
 </div>  
 <div className="result-item">  
 <div className="result-label">Tables Detected</div>  
 <div className="result-value">{jobDetails.results.tablesDetected}</div>  
 </div>  
 <div className="result-item">  
 <div className="result-label">Materials Identified</div>  
 <div className="result-value">{jobDetails.results.materialsIdentified}</div>  
 </div>  
 {jobDetails.results.materialsImported !== undefined && (  
 <div className="result-item">  
 <div className="result-label">Materials Imported</div>  
 <div className="result-value">{jobDetails.results.materialsImported}</div>  
 </div>  
 )}  
 </div>  
  
 <div className="action-buttons">  
 <button onClick={() => viewResults(jobId)}>View Results</button>  
 <button onClick={() => downloadResults(jobId)}>Download Results</button>  
 </div>  
 </div>  
 )}  
  
 {jobDetails.status === 'failed' && (  
 <div className="error-section">  
 <h3>Processing Error</h3>  
 <div className="error-message">  
 {jobDetails.error || 'Unknown error occurred'}  
 </div>  
 <div className="action-buttons">  
 <button onClick={() => retryJob(jobId)}>Retry Job</button>  
 </div>  
 </div>  
 )}  
  
 <div className="job-details">  
 <h3>Job Details</h3>  
 <div className="details-grid">  
 <div className="detail-item">  
 <div className="detail-label">Job ID</div>  
 <div className="detail-value">{jobDetails.jobId}</div>  
 </div>  
 <div className="detail-item">  
 <div className="detail-label">File Name</div>  
 <div className="detail-value">{jobDetails.fileName}</div>  
 </div>  
 <div className="detail-item">  
 <div className="detail-label">File Size</div>  
 <div className="detail-value">{formatFileSize(jobDetails.fileSize)}</div>  
 </div>  
 <div className="detail-item">  
 <div className="detail-label">Created</div>  
 <div className="detail-value">{formatDate(jobDetails.createdAt)}</div>  
 </div>  
 {jobDetails.startedAt && (  
 <div className="detail-item">  
 <div className="detail-label">Started</div>  
 <div className="detail-value">{formatDate(jobDetails.startedAt)}</div>  
 </div>  
 )}  
 {jobDetails.completedAt && (  
 <div className="detail-item">  
 <div className="detail-label">Completed</div>  
 <div className="detail-value">{formatDate(jobDetails.completedAt)}</div>  
 </div>  
 )}  
 </div>  
 </div>  
 </div>  
 );  
};  
  
// Helper functions  
const formatFileSize = (bytes: number) => {  
 if (bytes < 1024) {  
 return `${bytes} B`;  
 } else if (bytes < 1024 \* 1024) {  
 return `${(bytes / 1024).toFixed(2)} KB`;  
 } else if (bytes < 1024 \* 1024 \* 1024) {  
 return `${(bytes / (1024 \* 1024)).toFixed(2)} MB`;  
 } else {  
 return `${(bytes / (1024 \* 1024 \* 1024)).toFixed(2)} GB`;  
 }  
};  
  
const formatDate = (dateString: string) => {  
 return new Date(dateString).toLocaleString();  
};  
  
const viewResults = (jobId: string) => {  
 window.location.href = `/pdf/jobs/${jobId}/results`;  
};  
  
const downloadResults = (jobId: string) => {  
 window.location.href = `/api/pdf/jobs/${jobId}/results/download`;  
};  
  
const retryJob = async (jobId: string) => {  
 try {  
 await fetch(`/api/pdf/jobs/${jobId}/retry`, {  
 method: 'POST'  
 });  
 } catch (error) {  
 console.error(`Failed to retry job ${jobId}:`, error);  
 }  
};

### Performance Considerations

1. Horizontal Scaling
2. Each queue worker can be scaled independently
3. Multiple workers can process jobs from the same queue
4. Cloud deployment allows auto-scaling based on queue length
5. Message Efficiency
6. Compact message payloads to reduce transmission overhead
7. Appropriate subscription filtering to reduce unnecessary processing
8. Batch operations for high-volume workflows
9. Error Handling and Recovery
10. Automatic retries with exponential backoff
11. Dead-letter queues for failed jobs
12. Partial progress recovery for interrupted jobs
13. Circuit breakers for dependent services
14. Monitoring and Alerting
15. Real-time queue length monitoring
16. Processing time tracking
17. Error rate alerting
18. Resource utilization monitoring
19. Resource Requirements
20. Supabase instance with Realtime features enabled
21. Database for job storage and persistence
22. Worker instances with appropriate resources for job types
23. Network capacity for real-time updates

Cloud deployment allows auto-scaling based on queue length

Message Efficiency

Batch operations for high-volume workflows

Error Handling and Recovery

Circuit breakers for dependent services

Monitoring and Alerting

Resource utilization monitoring

Resource Requirements

# Rag System

Source: readme/rag-system.md

---

## RAG System: Architecture, Enhancements, and Integration Guide

### Overview

The Retrieval-Augmented Generation (RAG) system enhances the platform's ability to provide accurate, contextually relevant information about materials by combining vector search with knowledge base integration and generative AI. This document provides a comprehensive guide to the system architecture, enhancements, customization options, and integration steps.

### Table of Contents

1. System Architecture
2. Core Component Details
3. Key Enhancements
4. Setup and Installation
5. Deployment
6. Integration Steps
7. API Endpoints and Admin Panel
8. Customization Options
9. Prompt Engineering
10. Fine-tuning Models
11. Parameter Optimization
12. Performance Considerations
13. Troubleshooting
14. Integration with Existing Systems
15. Extending the System
16. MCP Server Integration
17. Kubernetes Deployment
18. CI/CD Integration
19. Implementation Checklist

### System Architecture

The RAG system consists of five major components that work together in a pipeline:

graph TD  
 subgraph Frontend  
 UserQuery[User Query] --> TypeScriptBridge[TypeScript Bridge]  
 TypeScriptBridge --> PythonBridge[Python Bridge Handler]  
 Response[Response to User] <-- Results --- TypeScriptBridge  
 end  
  
 subgraph RAGSystem[RAG System Core]  
 PythonBridge --> |Query| UnifiedService[Unified RAG Service]  
  
 UnifiedService --> Embedding[Enhanced Text Embeddings]  
 Embedding -->|Vector| Retrieval[Hybrid Retrieval System]  
  
 Retrieval --> |Materials & Knowledge| ContextAssembly[Context Assembly System]  
  
 ContextAssembly --> |Structured Context| GenerativeEnhancer[Generative Enhancement Layer]  
  
 GenerativeEnhancer --> |Enhanced Response| UnifiedService  
  
 UnifiedService -.->|Cache Results| Cache[(Cache)]  
 UnifiedService <-.->|Check Cache| Cache  
 end  
  
 subgraph Storage  
 Retrieval <--> |Vector Search| SupabaseVectors[(Supabase pgvector)]  
 Retrieval <--> |Knowledge Lookup| KnowledgeBase[(Knowledge Base)]  
 ContextAssembly <--> |Relationships| KnowledgeGraph[(Knowledge Graph)]  
 end  
  
 classDef system fill:#e1f5fe,stroke:#01579b,stroke-width:2px;  
 classDef storage fill:#e8f5e9,stroke:#2e7d32,stroke-width:2px;  
 classDef frontend fill:#fff3e0,stroke:#e65100,stroke-width:2px;  
 classDef component fill:#f3e5f5,stroke:#7b1fa2,stroke-width:2px;  
  
 class UnifiedService,Cache system;  
 class SupabaseVectors,KnowledgeBase,KnowledgeGraph storage;  
 class UserQuery,TypeScriptBridge,PythonBridge,Response frontend;  
 class Embedding,Retrieval,ContextAssembly,GenerativeEnhancer component;

graph TD  
 subgraph Frontend  
 UserQuery[User Query] --> TypeScriptBridge[TypeScript Bridge]  
 TypeScriptBridge --> PythonBridge[Python Bridge Handler]  
 Response[Response to User] <-- Results --- TypeScriptBridge  
 end  
  
 subgraph RAGSystem[RAG System Core]  
 PythonBridge --> |Query| UnifiedService[Unified RAG Service]  
  
 UnifiedService --> Embedding[Enhanced Text Embeddings]  
 Embedding -->|Vector| Retrieval[Hybrid Retrieval System]  
  
 Retrieval --> |Materials & Knowledge| ContextAssembly[Context Assembly System]  
  
 ContextAssembly --> |Structured Context| GenerativeEnhancer[Generative Enhancement Layer]  
  
 GenerativeEnhancer --> |Enhanced Response| UnifiedService  
  
 UnifiedService -.->|Cache Results| Cache[(Cache)]  
 UnifiedService <-.->|Check Cache| Cache  
 end  
  
 subgraph Storage  
 Retrieval <--> |Vector Search| SupabaseVectors[(Supabase pgvector)]  
 Retrieval <--> |Knowledge Lookup| KnowledgeBase[(Knowledge Base)]  
 ContextAssembly <--> |Relationships| KnowledgeGraph[(Knowledge Graph)]  
 end  
  
 classDef system fill:#e1f5fe,stroke:#01579b,stroke-width:2px;  
 classDef storage fill:#e8f5e9,stroke:#2e7d32,stroke-width:2px;  
 classDef frontend fill:#fff3e0,stroke:#e65100,stroke-width:2px;  
 classDef component fill:#f3e5f5,stroke:#7b1fa2,stroke-width:2px;  
  
 class UnifiedService,Cache system;  
 class SupabaseVectors,KnowledgeBase,KnowledgeGraph storage;  
 class UserQuery,TypeScriptBridge,PythonBridge,Response frontend;  
 class Embedding,Retrieval,ContextAssembly,GenerativeEnhancer component;

Note: If this diagram doesn't render correctly in your markdown viewer, you can find the source file at docs/diagrams/rag-system-architecture.mmd.

#### Component Flow

1. User Query Processing: The system receives a user query about materials.
2. Embedding Generation: The query is converted to embeddings (dense and/or sparse).
3. Hybrid Retrieval: Multiple retrieval methods find relevant materials and knowledge.
4. Context Assembly: Retrieved information is organized into a structured context.
5. Generative Enhancement: An LLM enhances the results with explanations, citations, and recommendations.
6. Response Delivery: Results are returned to the user, with optional streaming for large responses.

### Core Component Details

#### 1. Enhanced Vector Storage System

Files:  
- packages/ml/python/enhanced\_text\_embeddings.py - Text embedding generator  
- packages/server/src/services/supabase/migrations/006\_enhanced\_vector\_storage.sql - Database schema  
- packages/server/src/services/supabase/enhanced-vector-service.ts - TypeScript service

packages/ml/python/enhanced\_text\_embeddings.py

packages/server/src/services/supabase/migrations/006\_enhanced\_vector\_storage.sql

packages/server/src/services/supabase/enhanced-vector-service.ts

Key Features:  
- Dense embeddings using transformer models  
- Sparse embeddings using BM25 and TF-IDF  
- HNSW indexing for fast approximate nearest neighbor search  
- Specialized indexes per material category  
- Dynamic embedding generation with model selection  
- Auto-normalization and dimension handling  
- Custom tokenization for domain-specific terms  
- Multi-lingual support with language detection

Customization Points:  
- Embedding models can be configured in enhanced\_text\_embeddings.py  
- Index parameters can be adjusted in the migration file  
- Storage settings controlled via the config object in the service  
- Custom tokenization rules for domain-specific terminology

enhanced\_text\_embeddings.py

#### 2. Hybrid Retrieval System

Files:  
- packages/ml/python/hybrid\_retriever.py - Multi-stage retrieval implementation  
- packages/server/src/types/enhancedVector.types.ts - TypeScript type definitions  
- packages/server/src/controllers/enhancedVector.controller.ts - API endpoints  
- packages/server/src/services/supabase/enhanced-vector-service.ts - Service implementation  
- packages/server/src/utils/enhancedVectorValidation.ts - Input validation

packages/ml/python/hybrid\_retriever.py

packages/server/src/types/enhancedVector.types.ts

packages/server/src/controllers/enhancedVector.controller.ts

packages/server/src/services/supabase/enhanced-vector-service.ts

packages/server/src/utils/enhancedVectorValidation.ts

Key Features:  
- Advanced multi-stage retrieval combining five approaches:  
 1. Dense vector embedding search (semantic similarity)  
 2. Sparse vector search (keyword/feature matching)  
 3. Metadata filtering (structured property matching)  
 4. Ensemble approach for result blending with adaptive weighting  
 5. Contextualized re-ranking with performance monitoring  
- Knowledge base integration with bidirectional linking  
- Distributive retrieval for heterogeneous data sources  
- Performance optimization with query profiling and auto-tuning  
- Streaming support for large responses  
- Comprehensive API with query, batch, and streaming interfaces  
- Material relationship mapping with transitive discovery  
- Advanced filtering with nested property support  
- Semantic indexing with automated classification

Customization Points:  
- Retrieval strategies can be configured via the strategy parameter (hybrid, vector\_first, knowledge\_first, balanced, adaptive)  
- Ensemble weights adjustable in \_combine\_results method with dense, sparse, and metadata weights  
- Re-ranking parameters in \_rerank\_results method including property matching boost  
- Filtering logic in \_apply\_filters method with support for advanced operators  
- Knowledge base integration via the use\_knowledge\_base parameter  
- Performance profiles for different use cases (speed, quality, balanced)

strategy

hybrid

vector\_first

knowledge\_first

balanced

adaptive

\_combine\_results

\_rerank\_results

\_apply\_filters

use\_knowledge\_base

speed

quality

balanced

#### 3. Context Assembly System

Files:  
- packages/ml/python/context\_assembler.py - Context organization  
- packages/ml/python/hybrid\_retriever.py - Contains ContextAssembler class

packages/ml/python/context\_assembler.py

packages/ml/python/hybrid\_retriever.py

ContextAssembler

Key Features:  
- Structured property extraction from knowledge base  
- Relationship context incorporation  
- Vector-knowledge integration  
- Optimized formatting for downstream LLM use  
- Knowledge graph context extraction  
- Material relationship mapping  
- Bidirectional linking between materials and knowledge  
- Cross-reference validation for data consistency  
- Hierarchical context organization with priority levels

Customization Points:  
- Knowledge sources prioritization in \_gather\_knowledge method  
- Relationship mapping parameters in \_get\_relationships method  
- Context structure in \_format\_context method  
- Knowledge graph context in \_get\_knowledge\_graph\_context method  
- Maximum context sizes in the configuration via max\_context\_items parameter  
- Priority weighting for different context types

\_gather\_knowledge

\_get\_relationships

\_format\_context

\_get\_knowledge\_graph\_context

max\_context\_items

#### 4. Generative Enhancement Layer

Files:  
- packages/ml/python/generative\_enhancer.py - LLM integration  
- packages/ml/python/hybrid\_retriever.py - Contains GenerativeEnhancer class

packages/ml/python/generative\_enhancer.py

packages/ml/python/hybrid\_retriever.py

GenerativeEnhancer

Key Features:  
- LLM integration for content enhancement  
- Factual grounding with knowledge base  
- Citation system for transparency  
- Multiple enhancement types (explanations, comparisons, applications)  
- Source extraction for proper attribution  
- Template-based prompt generation  
- Adaptive response formatting based on query type  
- Confidence scoring for generated content  
- Structured output options for machine consumption

Customization Points:  
- LLM model selection in the configuration  
- Customizable prompt templates:  
 - explanation\_template for material explanations  
 - comparison\_template for similarity comparisons  
 - application\_template for application recommendations  
- Response processing in \_process\_\* methods  
- Property formatting in \_format\_properties method  
- Context formatting in \_format\_context method  
- Confidence threshold settings

explanation\_template

comparison\_template

application\_template

\_process\_\*

\_format\_properties

\_format\_context

#### 5. Unified RAG Service

Files:  
- packages/ml/python/material\_rag\_service.py - Orchestration service  
- packages/ml/python/rag\_bridge\_handler.py - Bridge to TypeScript  
- packages/ml/src/rag-bridge.ts - TypeScript integration  
- packages/ml/python/hybrid\_retriever.py - Contains MaterialRAGService class  
- packages/server/src/controllers/enhancedVector.controller.ts - API endpoints

packages/ml/python/material\_rag\_service.py

packages/ml/python/rag\_bridge\_handler.py

packages/ml/src/rag-bridge.ts

packages/ml/python/hybrid\_retriever.py

MaterialRAGService

packages/server/src/controllers/enhancedVector.controller.ts

Key Features:  
- Pipeline orchestration across all components  
- Caching for performance optimization with TTL settings  
- Streaming support for progressive delivery  
- Comprehensive API with query, batch, and streaming interfaces  
- Image-based search capabilities  
- Material comparison features  
- MCP (Material Computing Platform) integration  
- Performance monitoring and telemetry  
- Dynamic configuration management  
- Cross-platform operation via bridge handlers

Customization Points:  
- Service configuration in the constructor  
- Caching parameters: cache\_results and cache\_ttl\_seconds  
- Result enhancement options in \_enhance\_results method  
- Image search parameters in search\_by\_image method  
- Material comparison in compare\_materials method  
- API integration with vector search configurations  
- Performance monitoring settings

cache\_results

cache\_ttl\_seconds

\_enhance\_results

search\_by\_image

compare\_materials

### Key Enhancements

The enhanced RAG system builds upon the core architecture with several significant improvements:

#### Material-Specific Models

The system now uses specialized prompt templates and models optimized for different material types (wood, tile, stone, etc.), resulting in more accurate and domain-specific responses.

Key Features:  
- Material-specific system prompts with domain expertise  
- Specialized instruction sets for different material types  
- Material-specific evaluation criteria  
- Custom citation formats for different material domains

Implementation:  
- material\_specific\_prompts.py: Defines specialized prompt templates for different material types  
- Integration with the generative enhancer to use material-specific prompts for explanations, similarities, and applications

material\_specific\_prompts.py

Benefits:  
- More accurate and relevant responses for specific material types  
- Better understanding of domain-specific terminology and concepts  
- Improved user experience with material-specific expertise

#### Continuous Learning Pipeline

The system now includes a continuous learning pipeline that automatically fine-tunes models based on user feedback and performance metrics.

Key Features:  
- Automated fine-tuning triggers based on feedback metrics  
- A/B testing framework for model comparison  
- Performance tracking and analysis  
- Feedback-based training data generation

Implementation:  
- continuous\_learning\_pipeline.py: Implements the continuous learning pipeline  
- model\_registry.py: Manages models and A/B tests

continuous\_learning\_pipeline.py

model\_registry.py

Benefits:  
- Continuously improving model performance  
- Data-driven model selection  
- Systematic evaluation of model performance  
- Adaptation to changing user needs and preferences

#### Advanced Retrieval Techniques

The system now uses advanced retrieval techniques to handle complex queries more effectively.

Key Features:  
- Query decomposition for complex queries  
- Hierarchical retrieval for multi-faceted queries  
- Result reranking based on query relevance  
- Support for both dense and sparse retrieval methods

Implementation:  
- hierarchical\_retriever.py: Implements hierarchical retrieval for complex queries

hierarchical\_retriever.py

Benefits:  
- Better handling of complex, multi-faceted queries  
- More comprehensive retrieval results  
- Improved relevance ranking  
- Better support for comparative queries

#### Enhanced Visual Recognition Integration

The system now integrates visual information more effectively with textual queries.

Key Features:  
- Visual context enrichment for text queries  
- Cross-modal attention for better integration of visual and textual information  
- Multi-modal embedding generation  
- Visual feature extraction and integration

Implementation:  
- cross\_modal\_attention.py: Implements cross-modal attention mechanisms

cross\_modal\_attention.py

Benefits:  
- Better understanding of visual material characteristics  
- More accurate responses to queries with images  
- Enhanced ability to describe and compare materials visually  
- Improved multi-modal search capabilities

#### Performance Optimization

The system now includes performance optimizations for large-scale deployments.

Key Features:  
- Distributed retrieval across multiple vector stores  
- Caching strategies for frequently accessed materials  
- Load balancing for retrieval operations  
- Batched operations for improved throughput

Implementation:  
- distributed\_retrieval.py: Implements distributed retrieval and caching

distributed\_retrieval.py

Benefits:  
- Improved scalability for large vector databases  
- Reduced latency for frequent queries  
- Better resource utilization  
- Higher throughput for concurrent requests

### Setup and Installation

#### Prerequisites

Before installing the Enhanced RAG system, ensure you have the following:

1. Python 3.8+ - The enhanced RAG system requires Python 3.8 or higher.
2. Node.js 16+ - Required for the TypeScript bridge components.
3. Docker - Required for containerized deployment.
4. Kubernetes - Required for production deployment.
5. Supabase Account - Required for vector storage.
6. OpenAI API Key - Or another LLM provider API key.

#### Installation Steps

##### 1. Clone the Repository

git clone https://github.com/your-org/kai-platform.git  
cd kai-platform

git clone https://github.com/your-org/kai-platform.git  
cd kai-platform

##### 2. Install Python Dependencies

cd packages/ml  
pip install -r requirements.txt  
pip install -e .

cd packages/ml  
pip install -r requirements.txt  
pip install -e .

##### 3. Install Node.js Dependencies

cd ../..  
npm install  
cd packages/ml  
npm install

cd ../..  
npm install  
cd packages/ml  
npm install

##### 4. Configure Environment

Create a .env file in the packages/ml directory with the following variables:

.env

packages/ml

# LLM Configuration  
OPENAI\_API\_KEY=your\_openai\_api\_key  
LLM\_PROVIDER=openai  
LLM\_MODEL=gpt-4  
  
# Vector DB Configuration  
SUPABASE\_URL=your\_supabase\_url  
SUPABASE\_KEY=your\_supabase\_key  
  
# Storage Paths  
RAG\_DATA\_DIR=/path/to/data  
MODEL\_REGISTRY\_DIR=/path/to/model-registry  
MODELS\_DIR=/path/to/models  
  
# Performance Configuration  
ENABLE\_CACHE=true  
CACHE\_TTL=3600  
MAX\_CONCURRENT\_REQUESTS=10

# LLM Configuration  
OPENAI\_API\_KEY=your\_openai\_api\_key  
LLM\_PROVIDER=openai  
LLM\_MODEL=gpt-4  
  
# Vector DB Configuration  
SUPABASE\_URL=your\_supabase\_url  
SUPABASE\_KEY=your\_supabase\_key  
  
# Storage Paths  
RAG\_DATA\_DIR=/path/to/data  
MODEL\_REGISTRY\_DIR=/path/to/model-registry  
MODELS\_DIR=/path/to/models  
  
# Performance Configuration  
ENABLE\_CACHE=true  
CACHE\_TTL=3600  
MAX\_CONCURRENT\_REQUESTS=10

##### 5. Create Necessary Directories

mkdir -p data/model-registry data/models data/state data/temp

mkdir -p data/model-registry data/models data/state data/temp

##### 6. Initialize the Vector Database

cd packages/ml/python  
python initialize\_vector\_db.py

cd packages/ml/python  
python initialize\_vector\_db.py

##### 7. Set Up Dependencies

Set up the necessary dependencies for the enhanced RAG system:

# Set up dependencies  
python packages/ml/python/setup\_dependencies.py --config path/to/config.json

# Set up dependencies  
python packages/ml/python/setup\_dependencies.py --config path/to/config.json

This will create the required directories and initialize the vector stores.

##### 8. Verify Installation

Verify the installation by running the test script:

python test\_integration.py --config path/to/config.json --setup

python test\_integration.py --config path/to/config.json --setup

#### Configuration

The enhanced RAG system is configured using a JSON file. The default configuration is in kubernetes/continuous-learning-deployment.yaml under the enhanced-rag-config ConfigMap.

kubernetes/continuous-learning-deployment.yaml

enhanced-rag-config

You can customize the configuration by editing this file before deployment.

Key configuration sections:

* model\_registry\_config: Configuration for the model registry
* learning\_pipeline\_config: Configuration for the continuous learning pipeline
* distributed\_retrieval\_config: Configuration for distributed retrieval
* hierarchical\_retriever\_config: Configuration for hierarchical retrieval
* cross\_modal\_attention\_config: Configuration for cross-modal attention

model\_registry\_config

learning\_pipeline\_config

distributed\_retrieval\_config

hierarchical\_retriever\_config

cross\_modal\_attention\_config

#### Docker Installation

For containerized deployment, you can use the provided Dockerfile:

# Build the Docker image  
docker build -t enhanced-rag -f Dockerfile.rag .  
  
# Run the Docker container  
docker run -p 8000:8000 \  
 -e OPENAI\_API\_KEY=your\_openai\_api\_key \  
 -e SUPABASE\_URL=your\_supabase\_url \  
 -e SUPABASE\_KEY=your\_supabase\_key \  
 -v /path/to/data:/data \  
 enhanced-rag

# Build the Docker image  
docker build -t enhanced-rag -f Dockerfile.rag .  
  
# Run the Docker container  
docker run -p 8000:8000 \  
 -e OPENAI\_API\_KEY=your\_openai\_api\_key \  
 -e SUPABASE\_URL=your\_supabase\_url \  
 -e SUPABASE\_KEY=your\_supabase\_key \  
 -v /path/to/data:/data \  
 enhanced-rag

The enhanced RAG system is designed to integrate automatically with your existing MCP server and admin panel. No manual integration steps are required.

### Deployment

The enhanced RAG system includes automated deployment scripts that handle building, pushing, deploying, verifying, and monitoring the system.

#### Automated Deployment

To deploy the enhanced RAG system using the automated pipeline:

# Make the script executable  
chmod +x rag-deployment-pipeline.sh  
  
# Run the deployment pipeline  
./rag-deployment-pipeline.sh

# Make the script executable  
chmod +x rag-deployment-pipeline.sh  
  
# Run the deployment pipeline  
./rag-deployment-pipeline.sh

This script will:  
1. Build and push Docker images  
2. Deploy to Kubernetes  
3. Verify the deployment  
4. Monitor system performance  
5. Monitor API performance

#### Manual Deployment

If you prefer to deploy manually, you can use the individual scripts:

1. Build and push Docker images:

# Set Docker registry and tag (optional)  
export DOCKER\_REGISTRY=your-registry  
export TAG=latest  
  
# Build and push images  
./build-push-rag.sh

# Set Docker registry and tag (optional)  
export DOCKER\_REGISTRY=your-registry  
export TAG=latest  
  
# Build and push images  
./build-push-rag.sh

1. Deploy to Kubernetes:

# Deploy  
./deploy-rag.sh

# Deploy  
./deploy-rag.sh

1. Verify the deployment:

# Verify  
./verify-rag-deployment.sh

# Verify  
./verify-rag-deployment.sh

1. Monitor system performance:

# Set monitoring duration (optional)  
export DURATION=10m  
  
# Monitor system performance  
./monitor-rag-performance.sh

# Set monitoring duration (optional)  
export DURATION=10m  
  
# Monitor system performance  
./monitor-rag-performance.sh

1. Monitor API performance:

# Set API URL (optional)  
export API\_URL=http://your-api-url/api/rag  
  
# Monitor API performance  
./monitor-rag-api.sh

# Set API URL (optional)  
export API\_URL=http://your-api-url/api/rag  
  
# Monitor API performance  
./monitor-rag-api.sh

### Integration Steps

#### 1. Initialize the Enhanced RAG System

Once dependencies are set up, initialize the enhanced RAG system:

from packages.ml.python.initialize\_enhanced\_rag import initialize\_enhanced\_rag  
  
# Create existing components dictionary  
existing\_components = {  
 "base\_retriever": your\_base\_retriever,  
 "embedding\_model": your\_embedding\_model,  
 "vision\_model": your\_vision\_model,  
 "text\_model": your\_text\_model,  
 "llm\_client": your\_llm\_client,  
 "feedback\_db": your\_feedback\_db,  
 "vector\_stores": your\_vector\_stores  
}  
  
# Initialize enhanced RAG  
enhanced\_rag = await initialize\_enhanced\_rag(  
 config\_path="path/to/config.json",  
 existing\_components=existing\_components  
)

from packages.ml.python.initialize\_enhanced\_rag import initialize\_enhanced\_rag  
  
# Create existing components dictionary  
existing\_components = {  
 "base\_retriever": your\_base\_retriever,  
 "embedding\_model": your\_embedding\_model,  
 "vision\_model": your\_vision\_model,  
 "text\_model": your\_text\_model,  
 "llm\_client": your\_llm\_client,  
 "feedback\_db": your\_feedback\_db,  
 "vector\_stores": your\_vector\_stores  
}  
  
# Initialize enhanced RAG  
enhanced\_rag = await initialize\_enhanced\_rag(  
 config\_path="path/to/config.json",  
 existing\_components=existing\_components  
)

#### 2. Use the Enhanced RAG System

Once initialized, you can use the enhanced RAG system:

# Process a text query  
result = await enhanced\_rag.process\_query(  
 text\_query="What are the best hardwood flooring options for high-traffic areas?",  
 options={"detail\_level": "detailed"}  
)  
  
# Process an image query  
result = await enhanced\_rag.process\_query(  
 image\_data=image\_bytes,  
 options={"detail\_level": "medium"}  
)  
  
# Process a multi-modal query  
result = await enhanced\_rag.process\_query(  
 text\_query="What type of wood is this and how durable is it?",  
 image\_data=image\_bytes,  
 options={"detail\_level": "detailed"}  
)  
  
# Submit feedback  
await enhanced\_rag.submit\_feedback(  
 query="What are the best hardwood flooring options for high-traffic areas?",  
 response=result,  
 feedback={  
 "rating": 4,  
 "feedback\_text": "Good information but could include more about maintenance requirements."  
 }  
)  
  
# Get system stats  
stats = await enhanced\_rag.get\_system\_stats()

# Process a text query  
result = await enhanced\_rag.process\_query(  
 text\_query="What are the best hardwood flooring options for high-traffic areas?",  
 options={"detail\_level": "detailed"}  
)  
  
# Process an image query  
result = await enhanced\_rag.process\_query(  
 image\_data=image\_bytes,  
 options={"detail\_level": "medium"}  
)  
  
# Process a multi-modal query  
result = await enhanced\_rag.process\_query(  
 text\_query="What type of wood is this and how durable is it?",  
 image\_data=image\_bytes,  
 options={"detail\_level": "detailed"}  
)  
  
# Submit feedback  
await enhanced\_rag.submit\_feedback(  
 query="What are the best hardwood flooring options for high-traffic areas?",  
 response=result,  
 feedback={  
 "rating": 4,  
 "feedback\_text": "Good information but could include more about maintenance requirements."  
 }  
)  
  
# Get system stats  
stats = await enhanced\_rag.get\_system\_stats()

### API Endpoints and Admin Panel

#### API Endpoints

The enhanced RAG system adds the following API endpoints:

##### EnhancedVector TypeScript Integration

Files:  
- packages/server/src/types/enhancedVector.types.ts - Type definitions  
- packages/server/src/controllers/enhancedVector.controller.ts - API controllers  
- packages/server/src/routes/enhancedVector.routes.ts - API routes  
- packages/server/src/services/supabase/enhanced-vector-service.ts - Service implementation  
- packages/server/src/utils/enhancedVectorValidation.ts - Input validation

packages/server/src/types/enhancedVector.types.ts

packages/server/src/controllers/enhancedVector.controller.ts

packages/server/src/routes/enhancedVector.routes.ts

packages/server/src/services/supabase/enhanced-vector-service.ts

packages/server/src/utils/enhancedVectorValidation.ts

Key Features:  
- Strongly typed interfaces for all vector operations  
- Support for both dense and sparse vector embeddings with hybrid search  
- API endpoints for vector search and knowledge integration  
- Performance monitoring and configuration management  
- Advanced filtering with nested property support  
- Bidirectional linking between materials and knowledge  
- Extensible adapter pattern for multiple vector storage backends  
- Comprehensive error handling and validation  
- Detailed query profiling and optimization suggestions  
- Result pagination and cursor-based navigation

Vector API Endpoints:  
- POST /api/vector/enhanced/embeddings - Generate embeddings for text  
- GET /api/vector/enhanced/search - Search materials using text query  
- GET /api/vector/enhanced/materials/:id/similar - Find similar materials  
- GET /api/vector/enhanced/knowledge/search - Search with knowledge base integration  
- GET /api/vector/enhanced/knowledge/materials/:id/similar - Find similar materials with knowledge  
- POST /api/vector/enhanced/knowledge/route - Route a query between vector search and knowledge base  
- POST /api/vector/enhanced/knowledge/context - Assemble context from materials and knowledge  
- GET /api/vector/enhanced/performance - Get vector search performance statistics  
- GET /api/vector/enhanced/configs - Get vector search configurations  
- POST /api/vector/enhanced/filter - Advanced filtering with nested properties  
- GET /api/vector/enhanced/profile - Profile a search query for optimization  
- POST /api/vector/enhanced/bulk - Batch processing for multiple queries

POST /api/vector/enhanced/embeddings

GET /api/vector/enhanced/search

GET /api/vector/enhanced/materials/:id/similar

GET /api/vector/enhanced/knowledge/search

GET /api/vector/enhanced/knowledge/materials/:id/similar

POST /api/vector/enhanced/knowledge/route

POST /api/vector/enhanced/knowledge/context

GET /api/vector/enhanced/performance

GET /api/vector/enhanced/configs

POST /api/vector/enhanced/filter

GET /api/vector/enhanced/profile

POST /api/vector/enhanced/bulk

RAG API Endpoints:  
- POST /api/rag/query - Process a query with the enhanced RAG system  
- POST /api/rag/feedback - Submit feedback for a RAG response  
- GET /api/rag/stats - Get statistics for the enhanced RAG system  
- POST /api/rag/admin/fine-tune - Trigger fine-tuning for the enhanced RAG system  
- GET /api/rag/admin/models - Get models from the model registry  
- GET /api/rag/admin/ab-tests - Get A/B tests from the model registry

POST /api/rag/query

POST /api/rag/feedback

GET /api/rag/stats

POST /api/rag/admin/fine-tune

GET /api/rag/admin/models

GET /api/rag/admin/ab-tests

#### Admin Panel

The enhanced RAG system adds a new page to the admin panel:

* Enhanced RAG: View system stats, model registry, and A/B tests
* System performance metrics
* Fine-tuning status and history
* A/B test results and comparisons
* Feedback analysis dashboard
* Model registry management
* Configuration editor

### Customization Options

#### Configuration System

The RAG system uses a hierarchical configuration system that can be customized at multiple levels:

1. Default Configuration: Base settings in each component
2. Global Configuration: System-wide settings in the RAG service
3. Query-specific Configuration: Parameters for individual queries
4. Vector Search Configurations: Persistent configurations stored in the database
5. Performance Profiles: Pre-configured settings for different use cases

Example of customizing the global configuration:

// In TypeScript  
import { ragBridge } from 'packages/ml/src/rag-bridge';  
  
await ragBridge.updateConfig({  
 enableCache: true,  
 cacheTtl: 7200, // 2 hours  
 retrieval: {  
 maxResults: 15,  
 strategy: 'hybrid',  
 threshold: 0.7  
 },  
 generation: {  
 model: 'gpt-4',  
 temperature: 0.5,  
 enhancementTypes: ['explanation', 'similarity', 'application']  
 }  
});

// In TypeScript  
import { ragBridge } from 'packages/ml/src/rag-bridge';  
  
await ragBridge.updateConfig({  
 enableCache: true,  
 cacheTtl: 7200, // 2 hours  
 retrieval: {  
 maxResults: 15,  
 strategy: 'hybrid',  
 threshold: 0.7  
 },  
 generation: {  
 model: 'gpt-4',  
 temperature: 0.5,  
 enhancementTypes: ['explanation', 'similarity', 'application']  
 }  
});

#### Sample Configuration JSON

The enhanced RAG system can be configured through a JSON file. Here's an example configuration:

{  
 "model\_registry\_config": {  
 "registry\_dir": "/path/to/model-registry",  
 "models\_dir": "/path/to/models"  
 },  
 "learning\_pipeline\_config": {  
 "min\_feedback\_samples": 100,  
 "feedback\_threshold": 0.7,  
 "fine\_tuning\_interval\_days": 7,  
 "test\_size": 0.2,  
 "ab\_test\_duration\_days": 3,  
 "models\_to\_compare": 2,  
 "state\_dir": "/path/to/state",  
 "temp\_dir": "/path/to/temp"  
 },  
 "distributed\_retrieval\_config": {  
 "cache\_enabled": true,  
 "cache\_ttl\_seconds": 3600,  
 "batch\_size": 100,  
 "timeout\_seconds": 10,  
 "max\_concurrent\_requests": 5  
 },  
 "hierarchical\_retriever\_config": {  
 "max\_sub\_queries": 3,  
 "min\_query\_length": 15,  
 "reranking\_enabled": true,  
 "combine\_strategy": "weighted",  
 "query\_decomposition\_model": "gpt-3.5-turbo"  
 },  
 "cross\_modal\_attention\_config": {  
 "visual\_feature\_dim": 512,  
 "text\_feature\_dim": 768,  
 "joint\_feature\_dim": 1024,  
 "attention\_heads": 8,  
 "vision\_model\_name": "clip",  
 "text\_model\_name": "bert"  
 }  
}

{  
 "model\_registry\_config": {  
 "registry\_dir": "/path/to/model-registry",  
 "models\_dir": "/path/to/models"  
 },  
 "learning\_pipeline\_config": {  
 "min\_feedback\_samples": 100,  
 "feedback\_threshold": 0.7,  
 "fine\_tuning\_interval\_days": 7,  
 "test\_size": 0.2,  
 "ab\_test\_duration\_days": 3,  
 "models\_to\_compare": 2,  
 "state\_dir": "/path/to/state",  
 "temp\_dir": "/path/to/temp"  
 },  
 "distributed\_retrieval\_config": {  
 "cache\_enabled": true,  
 "cache\_ttl\_seconds": 3600,  
 "batch\_size": 100,  
 "timeout\_seconds": 10,  
 "max\_concurrent\_requests": 5  
 },  
 "hierarchical\_retriever\_config": {  
 "max\_sub\_queries": 3,  
 "min\_query\_length": 15,  
 "reranking\_enabled": true,  
 "combine\_strategy": "weighted",  
 "query\_decomposition\_model": "gpt-3.5-turbo"  
 },  
 "cross\_modal\_attention\_config": {  
 "visual\_feature\_dim": 512,  
 "text\_feature\_dim": 768,  
 "joint\_feature\_dim": 1024,  
 "attention\_heads": 8,  
 "vision\_model\_name": "clip",  
 "text\_model\_name": "bert"  
 }  
}

You can also use environment variables to configure the system:

# Set environment variables  
export RAG\_DATA\_DIR="/path/to/data"  
export LLM\_PROVIDER="openai"  
export LLM\_MODEL="gpt-4"  
export OPENAI\_API\_KEY="your-api-key"

# Set environment variables  
export RAG\_DATA\_DIR="/path/to/data"  
export LLM\_PROVIDER="openai"  
export LLM\_MODEL="gpt-4"  
export OPENAI\_API\_KEY="your-api-key"

#### Adding New Embedding Models

To add a new embedding model:

1. Edit packages/ml/python/enhanced\_text\_embeddings.py
2. Add your model to the AVAILABLE\_MODELS dictionary:

packages/ml/python/enhanced\_text\_embeddings.py

AVAILABLE\_MODELS

AVAILABLE\_MODELS = {  
 # Existing models...  
 "my-custom-model": {  
 "path": "path/to/model",  
 "dimension": 768,  
 "normalize": True  
 }  
}

AVAILABLE\_MODELS = {  
 # Existing models...  
 "my-custom-model": {  
 "path": "path/to/model",  
 "dimension": 768,  
 "normalize": True  
 }  
}

1. For more complex models, extend the \_generate\_embeddings\_with\_model method

\_generate\_embeddings\_with\_model

#### Creating Custom Retrieval Strategies

To add a new retrieval strategy:

1. Edit packages/ml/python/hybrid\_retriever.py
2. Add a new method following this pattern:

packages/ml/python/hybrid\_retriever.py

def \_retrieve\_with\_custom\_strategy(  
 self,  
 query\_text: str,  
 query\_embedding: Dict[str, Any],  
 filters: Optional[Dict[str, Any]] = None,  
 limit: int = a10  
) -> List[Dict[str, Any]]:  
 # Your custom retrieval logic here  
 # ...  
 return results

def \_retrieve\_with\_custom\_strategy(  
 self,  
 query\_text: str,  
 query\_embedding: Dict[str, Any],  
 filters: Optional[Dict[str, Any]] = None,  
 limit: int = a10  
) -> List[Dict[str, Any]]:  
 # Your custom retrieval logic here  
 # ...  
 return results

1. Register it in the retrieve method's strategy selection

retrieve

#### Customizing Result Formatting

To customize how results are formatted:

1. Edit packages/ml/python/context\_assembler.py
2. Modify the \_format\_context method:

packages/ml/python/context\_assembler.py

\_format\_context

def \_format\_context(self, materials, knowledge\_items, relationships):  
 # Your custom formatting logic  
 context = {  
 "materials": materials,  
 "knowledge": self.\_process\_knowledge(knowledge\_items),  
 "relationships": self.\_process\_relationships(relationships),  
 "custom\_section": self.\_generate\_custom\_section(materials)  
 }  
 return context

def \_format\_context(self, materials, knowledge\_items, relationships):  
 # Your custom formatting logic  
 context = {  
 "materials": materials,  
 "knowledge": self.\_process\_knowledge(knowledge\_items),  
 "relationships": self.\_process\_relationships(relationships),  
 "custom\_section": self.\_generate\_custom\_section(materials)  
 }  
 return context

### Prompt Engineering

The RAG system uses several prompts for different aspects of the generative enhancement. These prompts can now be managed through the admin panel's prompt management system.

#### Prompt Management System

The prompt management system provides a centralized way to manage all AI prompts used in the RAG system, including:

* Material-specific prompts
* Explanation prompts
* Similarity prompts
* Application prompts

For detailed information on using the prompt management system, see the Prompt Management System documentation.

#### Prompt Locations

Prompts are stored in the database and managed through the admin panel. The original prompt templates can be found in:

* packages/ml/python/generative\_enhancer.py - For generative enhancement prompts
* packages/ml/python/material\_specific\_prompts.py - For material-specific prompts

packages/ml/python/generative\_enhancer.py

packages/ml/python/material\_specific\_prompts.py

The system now uses packages/ml/python/material\_specific\_prompts\_db.py to fetch prompts from the database.

packages/ml/python/material\_specific\_prompts\_db.py

#### Prompt Structure

Each prompt has two parts:

1. System Prompt: Instructions for the LLM about its role and task
2. User Prompt: The specific query with context data

#### Customizing Prompts

To customize a prompt, use the prompt management system in the admin panel:

1. Navigate to "System Prompts" in the admin sidebar
2. Find the prompt you want to customize
3. Click the Edit button
4. Make your changes
5. Click Save

You can also still modify the prompts directly in the code:

1. Edit packages/ml/python/generative\_enhancer.py
2. Modify the relevant \_build\_\*\_prompt method
3. Update both system and user prompts as needed

packages/ml/python/generative\_enhancer.py

\_build\_\*\_prompt

However, using the admin panel is recommended as it allows for changes without code deployment.

Example of customizing the explanation prompt:

def \_build\_explanation\_prompt(self, context: ContextData, query: str) -> Dict[str, str]:  
 # Format materials data...  
  
 # Customize system prompt  
 system\_prompt = f"""  
 You are an expert materials scientist. Use only the provided context to explain  
 materials in relation to the query. Focus on practical applications, durability,  
 and cost-effectiveness. When information is not available, acknowledge the limitations.  
 Always cite sources for specific facts using [Source: Name] format.  
 """  
  
 # Customize user prompt  
 user\_prompt = f"""  
 Based on the provided information, explain each material's properties and suitability for: {query}  
  
 Focus specifically on these aspects:  
 1. Durability and long-term performance  
 2. Installation complexity and requirements  
 3. Cost considerations (initial and lifetime)  
 4. Environmental impact and sustainability  
  
 {context\_text}  
 """  
  
 return {  
 "system": system\_prompt,  
 "user": user\_prompt  
 }

def \_build\_explanation\_prompt(self, context: ContextData, query: str) -> Dict[str, str]:  
 # Format materials data...  
  
 # Customize system prompt  
 system\_prompt = f"""  
 You are an expert materials scientist. Use only the provided context to explain  
 materials in relation to the query. Focus on practical applications, durability,  
 and cost-effectiveness. When information is not available, acknowledge the limitations.  
 Always cite sources for specific facts using [Source: Name] format.  
 """  
  
 # Customize user prompt  
 user\_prompt = f"""  
 Based on the provided information, explain each material's properties and suitability for: {query}  
  
 Focus specifically on these aspects:  
 1. Durability and long-term performance  
 2. Installation complexity and requirements  
 3. Cost considerations (initial and lifetime)  
 4. Environmental impact and sustainability  
  
 {context\_text}  
 """  
  
 return {  
 "system": system\_prompt,  
 "user": user\_prompt  
 }

#### Testing Prompt Changes

After modifying prompts, test the changes using:

1. The verification script: python packages/ml/python/verify\_rag\_modules.py
2. Sample queries through the TypeScript bridge
3. Direct Python testing with the RAG service

python packages/ml/python/verify\_rag\_modules.py

### Fine-tuning Models

The RAG system supports fine-tuning of both embedding models and generative models to improve performance on specific material domains.

#### Fine-tuning Embedding Models

To fine-tune an embedding model:

1. Prepare a dataset of material texts and their relationships
2. Use the following script pattern:

from enhanced\_text\_embeddings import TextEmbeddingGenerator  
  
# Initialize generator with base model  
generator = TextEmbeddingGenerator(config={  
 "default\_model": "sentence-transformers/all-MiniLM-L6-v2"  
})  
  
# Prepare training data  
train\_data = [  
 {"text": "Material description 1", "label": "category\_1"},  
 {"text": "Material description 2", "label": "category\_1"},  
 {"text": "Material description 3", "label": "category\_2"},  
 # ...  
]  
  
# Fine-tune model  
generator.fine\_tune\_model(  
 training\_data=train\_data,  
 output\_path="./custom\_embeddings",  
 epochs=10,  
 batch\_size=16,  
 learning\_rate=2e-5  
)  
  
# Update configuration to use fine-tuned model  
generator.update\_config({  
 "default\_model": "./custom\_embeddings"  
})

from enhanced\_text\_embeddings import TextEmbeddingGenerator  
  
# Initialize generator with base model  
generator = TextEmbeddingGenerator(config={  
 "default\_model": "sentence-transformers/all-MiniLM-L6-v2"  
})  
  
# Prepare training data  
train\_data = [  
 {"text": "Material description 1", "label": "category\_1"},  
 {"text": "Material description 2", "label": "category\_1"},  
 {"text": "Material description 3", "label": "category\_2"},  
 # ...  
]  
  
# Fine-tune model  
generator.fine\_tune\_model(  
 training\_data=train\_data,  
 output\_path="./custom\_embeddings",  
 epochs=10,  
 batch\_size=16,  
 learning\_rate=2e-5  
)  
  
# Update configuration to use fine-tuned model  
generator.update\_config({  
 "default\_model": "./custom\_embeddings"  
})

#### Fine-tuning Generative Models

For generative models, use domain-specific instruction tuning:

1. Create a dataset of material-specific QA pairs
2. Use your preferred model provider's fine-tuning API
3. Update the generative enhancer configuration to use the fine-tuned model:

from generative\_enhancer import GenerativeEnhancer  
  
enhancer = GenerativeEnhancer(config={  
 "model": "your-fine-tuned-model-id",  
 "temperature": 0.7  
})

from generative\_enhancer import GenerativeEnhancer  
  
enhancer = GenerativeEnhancer(config={  
 "model": "your-fine-tuned-model-id",  
 "temperature": 0.7  
})

### Parameter Optimization

#### HNSW Index Parameters

HNSW (Hierarchical Navigable Small World) indexing parameters can be optimized for better performance:

1. Edit packages/server/src/services/supabase/migrations/006\_enhanced\_vector\_storage.sql
2. Adjust these key parameters:

packages/server/src/services/supabase/migrations/006\_enhanced\_vector\_storage.sql

CREATE INDEX ON materials\_embedding USING hnsw (embedding vector\_cosine\_ops)  
WITH (m = 16, ef\_construction = 200);

CREATE INDEX ON materials\_embedding USING hnsw (embedding vector\_cosine\_ops)  
WITH (m = 16, ef\_construction = 200);

Parameter guidelines:  
- m: Number of connections per node (8-64, higher = better quality but more memory)  
- ef\_construction: Size of the dynamic list for nearest neighbors (100-500, higher = better quality but slower indexing)  
- ef\_search: Size of the dynamic list for searching (not set in index, set at query time)

m

ef\_construction

ef\_search

#### Performance Profiles

The system includes pre-configured performance profiles:

PERFORMANCE\_PROFILES = {  
 "speed": {  
 "retrieval": {  
 "max\_results": 5,  
 "strategy": "vector\_first",  
 "reranking\_enabled": False,  
 "threshold": 0.7  
 },  
 "generation": {  
 "model": "gpt-3.5-turbo",  
 "detail\_level": "brief"  
 },  
 "enable\_cache": True,  
 "cache\_ttl": 86400 # 24 hours  
 },  
 "quality": {  
 "retrieval": {  
 "max\_results": 15,  
 "strategy": "hybrid",  
 "reranking\_enabled": True,  
 "threshold": 0.6,  
 "dense\_weight": 0.6,  
 "sparse\_weight": 0.4  
 },  
 "generation": {  
 "model": "gpt-4",  
 "detail\_level": "detailed"  
 },  
 "enable\_cache": True,  
 "cache\_ttl": 3600 # 1 hour  
 },  
 "balanced": {  
 "retrieval": {  
 "max\_results": 10,  
 "strategy": "hybrid",  
 "reranking\_enabled": True,  
 "threshold": 0.65  
 },  
 "generation": {  
 "model": "gpt-4",  
 "detail\_level": "medium"  
 },  
 "enable\_cache": True,  
 "cache\_ttl": 7200 # 2 hours  
 }  
}

PERFORMANCE\_PROFILES = {  
 "speed": {  
 "retrieval": {  
 "max\_results": 5,  
 "strategy": "vector\_first",  
 "reranking\_enabled": False,  
 "threshold": 0.7  
 },  
 "generation": {  
 "model": "gpt-3.5-turbo",  
 "detail\_level": "brief"  
 },  
 "enable\_cache": True,  
 "cache\_ttl": 86400 # 24 hours  
 },  
 "quality": {  
 "retrieval": {  
 "max\_results": 15,  
 "strategy": "hybrid",  
 "reranking\_enabled": True,  
 "threshold": 0.6,  
 "dense\_weight": 0.6,  
 "sparse\_weight": 0.4  
 },  
 "generation": {  
 "model": "gpt-4",  
 "detail\_level": "detailed"  
 },  
 "enable\_cache": True,  
 "cache\_ttl": 3600 # 1 hour  
 },  
 "balanced": {  
 "retrieval": {  
 "max\_results": 10,  
 "strategy": "hybrid",  
 "reranking\_enabled": True,  
 "threshold": 0.65  
 },  
 "generation": {  
 "model": "gpt-4",  
 "detail\_level": "medium"  
 },  
 "enable\_cache": True,  
 "cache\_ttl": 7200 # 2 hours  
 }  
}

### Performance Considerations

#### Scaling the RAG System

The RAG system is designed to scale horizontally and vertically:

1. Vertical Scaling: Increase the resources allocated to the system
2. Recommended for small to medium deployments
3. Configure according to estimated workload:  
     
   Low volume (<100 queries/minute): 2 vCPUs, 8GB RAM  
   Medium volume (100-500 queries/minute): 4 vCPUs, 16GB RAM  
   High volume (500+ queries/minute): 8+ vCPUs, 32+GB RAM
4. Horizontal Scaling: Deploy multiple instances with load balancing
5. Recommended for high-volume deployments
6. Distribute load across multiple nodes
7. Use shared Redis cache for consistent caching
8. Configure centralized logging and metrics

Configure according to estimated workload:

* Low volume (<100 queries/minute): 2 vCPUs, 8GB RAM
* Medium volume (100-500 queries/minute): 4 vCPUs, 16GB RAM
* High volume (500+ queries/minute): 8+ vCPUs, 32+GB RAM

Horizontal Scaling: Deploy multiple instances with load balancing

#### Memory Management

1. Embedding Models: Consider the memory footprint
2. Full-size models (768+ dimensions): 500MB+ RAM per instance
3. Quantized models: 150-300MB RAM per instance
4. Distilled models: 50-150MB RAM per instance
5. LLM Integration: Use efficient integration methods
6. Local models: Consider quantization and model size
7. API-based models: Optimize request batching
8. Use streaming for large responses to reduce memory pressure
9. Batch Processing: Configure for efficient resource use
10. max\_concurrent\_requests: Control parallelism (default: 10)
11. batch\_size: Control batch size for embedding generation (default: 32)

Distilled models: 50-150MB RAM per instance

LLM Integration: Use efficient integration methods

Use streaming for large responses to reduce memory pressure

Batch Processing: Configure for efficient resource use

max\_concurrent\_requests

batch\_size

#### Monitoring

Monitor these key metrics for performance optimization:

1. Latency: End-to-end response time
2. Query processing time
3. Embedding generation time
4. Retrieval time
5. Context assembly time
6. Generation time
7. Cache Performance:
8. Cache hit rate (target: >50%)
9. Cache size
10. Average cache entry TTL
11. Quality Metrics:
12. Retrieval relevance (measured through feedback)
13. Generation accuracy (measured through feedback)
14. User satisfaction scores

Generation time

Cache Performance:

Average cache entry TTL

Quality Metrics:

### Troubleshooting

If you encounter issues with the enhanced RAG system, check the following:

#### Diagnostic Steps

1. Logs: Check the logs for the MCP server and continuous learning service:

kubectl logs deployment/mcp-server  
kubectl logs deployment/continuous-learning

kubectl logs deployment/mcp-server  
kubectl logs deployment/continuous-learning

1. Deployment Verification: Run the verification script to check for common issues:

./verify-rag-deployment.sh

./verify-rag-deployment.sh

1. System Performance: Monitor system performance to identify resource issues:

./monitor-rag-performance.sh

./monitor-rag-performance.sh

1. API Performance: Test the API endpoints to ensure they're working correctly:

./monitor-rag-api.sh

./monitor-rag-api.sh

1. Configuration: Make sure the configuration is correct and all required directories exist.
2. Permissions: Make sure the system has permission to access the required directories and files.
3. Dependencies: Make sure all required dependencies are installed.
4. Memory: Make sure the system has enough memory to load the models.

Configuration: Make sure the configuration is correct and all required directories exist.

Permissions: Make sure the system has permission to access the required directories and files.

Dependencies: Make sure all required dependencies are installed.

Memory: Make sure the system has enough memory to load the models.

#### Common Issues and Solutions

##### Slow Query Performance

Symptoms:  
- High latency in query responses  
- Timeouts on complex queries

Potential Causes and Solutions:  
1. Index issues  
 - Check index creation in Supabase  
 - Ensure HNSW index is properly configured  
 - Consider increasing m parameter in HNSW index

m

1. Large result sets
2. Decrease max\_results parameter
3. Adjust filters to narrow search scope
4. Implement pagination for large result sets
5. Complex generative tasks
6. Reduce enhancement\_types to essential ones
7. Set detail\_level to "brief"
8. Use a faster LLM model for high-traffic scenarios

max\_results

Implement pagination for large result sets

Complex generative tasks

enhancement\_types

detail\_level

##### Low Relevance Results

Symptoms:  
- Irrelevant materials returned in results  
- Missing obviously relevant materials

Potential Causes and Solutions:  
1. Embedding mismatch  
 - Try different embedding models  
 - Adjust strategy weights in hybrid retrieval  
 - Consider fine-tuning embeddings on domain data

1. Threshold issues
2. Lower the similarity threshold to include more results
3. Tune the re-ranking model weights
4. Adjust dense\_weight and sparse\_weight in hybrid strategy
5. Filter issues
6. Check if filters are too restrictive
7. Verify metadata field mappings
8. Ensure material categorization is correct

Adjust dense\_weight and sparse\_weight in hybrid strategy

dense\_weight

sparse\_weight

Filter issues

##### Memory Issues

Symptoms:  
- Out of memory errors  
- System crashes under load

Potential Causes and Solutions:  
1. Embedding models too large  
 - Use smaller/quantized embedding models  
 - Reduce batch\_size for embedding generation  
 - Implement incremental processing for large datasets

batch\_size

1. Cache size issues
2. Reduce max\_cache\_size
3. Implement LRU cache eviction policy
4. Consider distributed caching with Redis
5. Large contexts for LLM
6. Reduce max\_knowledge\_items in context assembly
7. Implement chunking for large contexts
8. Use more efficient context compression techniques

max\_cache\_size

Consider distributed caching with Redis

Large contexts for LLM

max\_knowledge\_items

##### Pod Crashes

Symptoms:  
- Pods are crashing or restarting frequently  
- System is unstable

Potential Causes and Solutions:  
1. Resource limits  
 - Check if pods have enough resources (CPU/memory)  
 - Increase resource limits if necessary  
 - Check for memory leaks

1. Configuration issues
2. Ensure ConfigMap is correctly configured
3. Verify all required environment variables are set
4. Check if volume mounts are correct

To investigate pod crashes:

kubectl describe pod <pod-name>  
kubectl logs <pod-name> --previous

kubectl describe pod <pod-name>  
kubectl logs <pod-name> --previous

##### API Timeouts

Symptoms:  
- API requests time out  
- System is unresponsive

Potential Causes and Solutions:  
1. Resource issues  
 - Check if there are enough resources  
 - Increase timeouts if necessary  
 - Check for bottlenecks in the system

1. Network issues
2. Check if the network is stable
3. Verify firewall settings
4. Check if the service is accessible

#### Diagnostic Utilities

The system provides several diagnostic utilities:

1. Verification Script:  
    bash  
    python packages/ml/python/verify\_rag\_modules.py --json
2. Health Check Endpoint:  
    typescript  
    const health = await ragBridge.getHealthStatus();  
    console.log(health);
3. Usage Statistics:  
    typescript  
    const stats = await ragBridge.getUsageStatistics();  
    console.log(stats);
4. Deployment Verification:  
    bash  
    ./verify-rag-deployment.sh
5. Performance Monitoring:  
    bash  
    ./monitor-rag-performance.sh

Verification Script:  
 bash  
 python packages/ml/python/verify\_rag\_modules.py --json

bash  
 python packages/ml/python/verify\_rag\_modules.py --json

Health Check Endpoint:  
 typescript  
 const health = await ragBridge.getHealthStatus();  
 console.log(health);

typescript  
 const health = await ragBridge.getHealthStatus();  
 console.log(health);

Usage Statistics:  
 typescript  
 const stats = await ragBridge.getUsageStatistics();  
 console.log(stats);

typescript  
 const stats = await ragBridge.getUsageStatistics();  
 console.log(stats);

Deployment Verification:  
 bash  
 ./verify-rag-deployment.sh

bash  
 ./verify-rag-deployment.sh

Performance Monitoring:  
 bash  
 ./monitor-rag-performance.sh

bash  
 ./monitor-rag-performance.sh

### Integration with Existing Systems

The RAG system integrates with and enhances several platform's existing capabilities:

#### RAG and Visual Recognition Integration

The RAG system significantly enhances the platform's visual recognition capabilities:

1. Context-Enriched Recognition: After visual recognition identifies a material (like a tile, wood, or stone), the RAG system automatically retrieves related knowledge, specifications, and similar materials from your database.
2. Multi-modal Understanding: By connecting visual features with textual knowledge, the system provides a comprehensive understanding of materials. For example, if visual recognition identifies "white marble," the RAG system immediately provides information about its composition, durability, price range, and appropriate applications.
3. Improved Accuracy through Knowledge: The RAG system can improve recognition accuracy by using domain knowledge to validate and refine visual recognition results.
4. Relationship Mapping: Once a material is recognized, the RAG system maps it to your knowledge graph, exposing relationships with complementary materials, alternative options, and typical applications.

Context-Enriched Recognition: After visual recognition identifies a material (like a tile, wood, or stone), the RAG system automatically retrieves related knowledge, specifications, and similar materials from your database.

Multi-modal Understanding: By connecting visual features with textual knowledge, the system provides a comprehensive understanding of materials. For example, if visual recognition identifies "white marble," the RAG system immediately provides information about its composition, durability, price range, and appropriate applications.

Improved Accuracy through Knowledge: The RAG system can improve recognition accuracy by using domain knowledge to validate and refine visual recognition results.

Relationship Mapping: Once a material is recognized, the RAG system maps it to your knowledge graph, exposing relationships with complementary materials, alternative options, and typical applications.

#### PDF Processing for Materials Training

When processing PDFs with tile information to train the system:

1. Intelligent Extraction: The RAG system works with your existing PDF processing to extract structured information about materials - specifications, properties, applications, and visual characteristics.
2. Automatic Knowledge Integration: Extracted information is connected to the existing knowledge base:
3. New material information is linked with existing knowledge
4. Conflicts or updates needed in existing data are identified
5. Embeddings are automatically generated for semantic search
6. Material categorizations are suggested based on properties
7. Training Enhancement: The extracted data improves system training:
8. Embedding models can be fine-tuned with domain-specific data
9. The recognition vocabulary for visual systems is expanded
10. Training pairs for similarity and relationship models are created

Intelligent Extraction: The RAG system works with your existing PDF processing to extract structured information about materials - specifications, properties, applications, and visual characteristics.

Automatic Knowledge Integration: Extracted information is connected to the existing knowledge base:

Material categorizations are suggested based on properties

Training Enhancement: The extracted data improves system training:

#### Database Integration and Material Imports

When importing materials to the database, the RAG system enhances the process:

1. Enriched Indexing: New materials are automatically:
2. Embedded using both dense (transformer-based) and sparse (BM25/TF-IDF) vectors
3. Indexed using HNSW for extremely fast retrieval at scale
4. Categorized with specialized indexes based on material type
5. Knowledge Graph Integration: Each new material is:
6. Connected to related materials (similar appearance, properties, applications)
7. Linked to appropriate knowledge base entries
8. Positioned within the overall materials hierarchy
9. Enhanced Search and Discovery: The RAG system transforms how users find materials:
10. Multi-stage retrieval combines vector, text, and metadata search
11. Contextualized re-ranking improves result relevance
12. Ensemble approaches blend multiple search strategies

Categorized with specialized indexes based on material type

Knowledge Graph Integration: Each new material is:

Positioned within the overall materials hierarchy

Enhanced Search and Discovery: The RAG system transforms how users find materials:

### Extending the System

#### Adding New Enhancement Types

To add a new enhancement type:

1. Edit packages/ml/python/generative\_enhancer.py
2. Add a new prompt building method:  
    python  
    def \_build\_new\_enhancement\_prompt(self, context, query):  
    # Prompt building logic  
    return {"system": system\_prompt, "user": user\_prompt}
3. Add a new processing method:  
    python  
    def \_process\_new\_enhancement(self, response\_text, context\_data):  
    # Process the raw LLM response  
    return processed\_data
4. Add a new method for generating the enhancement:  
    python  
    async def \_generate\_new\_enhancement(self, context, query):  
    prompt = self.\_build\_new\_enhancement\_prompt(context, query)  
    response, \_ = await self.\_get\_llm\_response(prompt)  
    return self.\_process\_new\_enhancement(response, context)
5. Add the new enhancement type to the configuration options
6. Update the enhance method to include the new enhancement type

packages/ml/python/generative\_enhancer.py

Add a new prompt building method:  
 python  
 def \_build\_new\_enhancement\_prompt(self, context, query):  
 # Prompt building logic  
 return {"system": system\_prompt, "user": user\_prompt}

python  
 def \_build\_new\_enhancement\_prompt(self, context, query):  
 # Prompt building logic  
 return {"system": system\_prompt, "user": user\_prompt}

Add a new processing method:  
 python  
 def \_process\_new\_enhancement(self, response\_text, context\_data):  
 # Process the raw LLM response  
 return processed\_data

python  
 def \_process\_new\_enhancement(self, response\_text, context\_data):  
 # Process the raw LLM response  
 return processed\_data

Add a new method for generating the enhancement:  
 python  
 async def \_generate\_new\_enhancement(self, context, query):  
 prompt = self.\_build\_new\_enhancement\_prompt(context, query)  
 response, \_ = await self.\_get\_llm\_response(prompt)  
 return self.\_process\_new\_enhancement(response, context)

python  
 async def \_generate\_new\_enhancement(self, context, query):  
 prompt = self.\_build\_new\_enhancement\_prompt(context, query)  
 response, \_ = await self.\_get\_llm\_response(prompt)  
 return self.\_process\_new\_enhancement(response, context)

Add the new enhancement type to the configuration options

enhance

#### Continuous Improvement

Implement a feedback loop for continuous improvement:

1. Capture Feedback:
2. Add feedback collection to the TypeScript bridge
3. Track which results were helpful/unhelpful
4. Collect specific feedback on enhancement types
5. Analyze Patterns:
6. Identify common failure modes
7. Detect bias in responses
8. Find opportunities for model fine-tuning
9. Implement Improvements:
10. Update prompts based on feedback
11. Fine-tune models with new examples
12. Adjust retrieval parameters for better relevance
13. Measure Impact:
14. Track before/after metrics
15. Conduct A/B tests with changes
16. Calculate ROI of improvements

Collect specific feedback on enhancement types

Analyze Patterns:

Find opportunities for model fine-tuning

Implement Improvements:

Adjust retrieval parameters for better relevance

Measure Impact:

This cycle ensures the RAG system continuously adapts to user needs and improves over time.

### MCP Server Integration

To integrate the enhanced RAG system with your MCP server, you can use the provided MCP RAG bridge:

from packages.ml.python.mcp\_rag\_bridge import create\_mcp\_rag\_bridge  
  
# Create MCP RAG bridge  
mcp\_bridge = create\_mcp\_rag\_bridge(config\_path="path/to/config.json")  
  
# Initialize MCP RAG bridge  
await mcp\_bridge.initialize(existing\_components=existing\_components)  
  
# Handle requests  
query\_response = await mcp\_bridge.handle\_request("query", query\_request)  
feedback\_response = await mcp\_bridge.handle\_request("feedback", feedback\_request)  
stats\_response = await mcp\_bridge.handle\_request("stats", stats\_request)

from packages.ml.python.mcp\_rag\_bridge import create\_mcp\_rag\_bridge  
  
# Create MCP RAG bridge  
mcp\_bridge = create\_mcp\_rag\_bridge(config\_path="path/to/config.json")  
  
# Initialize MCP RAG bridge  
await mcp\_bridge.initialize(existing\_components=existing\_components)  
  
# Handle requests  
query\_response = await mcp\_bridge.handle\_request("query", query\_request)  
feedback\_response = await mcp\_bridge.handle\_request("feedback", feedback\_request)  
stats\_response = await mcp\_bridge.handle\_request("stats", stats\_request)

You can then add endpoints to your MCP server to handle these requests:

// Add endpoints to your MCP server  
app.post('/rag/enhanced-query', async (req, res) => {  
 const response = await mcpBridge.handleRequest('query', req.body);  
 res.json(response);  
});  
  
app.post('/rag/feedback', async (req, res) => {  
 const response = await mcpBridge.handleRequest('feedback', req.body);  
 res.json(response);  
});  
  
app.get('/rag/stats', async (req, res) => {  
 const response = await mcpBridge.handleRequest('stats', {});  
 res.json(response);  
});

// Add endpoints to your MCP server  
app.post('/rag/enhanced-query', async (req, res) => {  
 const response = await mcpBridge.handleRequest('query', req.body);  
 res.json(response);  
});  
  
app.post('/rag/feedback', async (req, res) => {  
 const response = await mcpBridge.handleRequest('feedback', req.body);  
 res.json(response);  
});  
  
app.get('/rag/stats', async (req, res) => {  
 const response = await mcpBridge.handleRequest('stats', {});  
 res.json(response);  
});

### Kubernetes Deployment

To deploy the enhanced RAG system on Kubernetes, you'll need to:

1. Create a ConfigMap for the configuration:

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: enhanced-rag-config  
data:  
 config.json: |  
 {  
 "model\_registry\_config": {  
 "registry\_dir": "/data/model-registry",  
 "models\_dir": "/data/models"  
 },  
 "learning\_pipeline\_config": {  
 "min\_feedback\_samples": 100,  
 "feedback\_threshold": 0.7,  
 "fine\_tuning\_interval\_days": 7,  
 "test\_size": 0.2,  
 "ab\_test\_duration\_days": 3,  
 "models\_to\_compare": 2,  
 "state\_dir": "/data/state",  
 "temp\_dir": "/data/temp"  
 },  
 "distributed\_retrieval\_config": {  
 "cache\_enabled": true,  
 "cache\_ttl\_seconds": 3600,  
 "batch\_size": 100,  
 "timeout\_seconds": 10,  
 "max\_concurrent\_requests": 5  
 }  
 }

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: enhanced-rag-config  
data:  
 config.json: |  
 {  
 "model\_registry\_config": {  
 "registry\_dir": "/data/model-registry",  
 "models\_dir": "/data/models"  
 },  
 "learning\_pipeline\_config": {  
 "min\_feedback\_samples": 100,  
 "feedback\_threshold": 0.7,  
 "fine\_tuning\_interval\_days": 7,  
 "test\_size": 0.2,  
 "ab\_test\_duration\_days": 3,  
 "models\_to\_compare": 2,  
 "state\_dir": "/data/state",  
 "temp\_dir": "/data/temp"  
 },  
 "distributed\_retrieval\_config": {  
 "cache\_enabled": true,  
 "cache\_ttl\_seconds": 3600,  
 "batch\_size": 100,  
 "timeout\_seconds": 10,  
 "max\_concurrent\_requests": 5  
 }  
 }

1. Create PersistentVolumeClaims for the data:

apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: model-registry-pvc  
spec:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 10Gi  
---  
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: models-pvc  
spec:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 20Gi

apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: model-registry-pvc  
spec:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 10Gi  
---  
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: models-pvc  
spec:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 20Gi

1. Update your Deployment to use the enhanced RAG system:

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: mcp-server  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: mcp-server  
 template:  
 metadata:  
 labels:  
 app: mcp-server  
 spec:  
 containers:  
 - name: mcp-server  
 image: your-registry/mcp-server:latest  
 env:  
 - name: RAG\_DATA\_DIR  
 value: "/data"  
 - name: LLM\_PROVIDER  
 value: "openai"  
 - name: LLM\_MODEL  
 value: "gpt-4"  
 - name: OPENAI\_API\_KEY  
 valueFrom:  
 secretKeyRef:  
 name: openai-secret  
 key: api-key  
 volumeMounts:  
 - name: model-registry  
 mountPath: /data/model-registry  
 - name: models  
 mountPath: /data/models  
 - name: enhanced-rag-config  
 mountPath: /app/config/enhanced-rag-config.json  
 subPath: config.json  
 volumes:  
 - name: model-registry  
 persistentVolumeClaim:  
 claimName: model-registry-pvc  
 - name: models  
 persistentVolumeClaim:  
 claimName: models-pvc  
 - name: enhanced-rag-config  
 configMap:  
 name: enhanced-rag-config

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: mcp-server  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: mcp-server  
 template:  
 metadata:  
 labels:  
 app: mcp-server  
 spec:  
 containers:  
 - name: mcp-server  
 image: your-registry/mcp-server:latest  
 env:  
 - name: RAG\_DATA\_DIR  
 value: "/data"  
 - name: LLM\_PROVIDER  
 value: "openai"  
 - name: LLM\_MODEL  
 value: "gpt-4"  
 - name: OPENAI\_API\_KEY  
 valueFrom:  
 secretKeyRef:  
 name: openai-secret  
 key: api-key  
 volumeMounts:  
 - name: model-registry  
 mountPath: /data/model-registry  
 - name: models  
 mountPath: /data/models  
 - name: enhanced-rag-config  
 mountPath: /app/config/enhanced-rag-config.json  
 subPath: config.json  
 volumes:  
 - name: model-registry  
 persistentVolumeClaim:  
 claimName: model-registry-pvc  
 - name: models  
 persistentVolumeClaim:  
 claimName: models-pvc  
 - name: enhanced-rag-config  
 configMap:  
 name: enhanced-rag-config

1. Create a separate Deployment for the continuous learning pipeline:

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: continuous-learning  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: continuous-learning  
 template:  
 metadata:  
 labels:  
 app: continuous-learning  
 spec:  
 containers:  
 - name: continuous-learning  
 image: your-registry/continuous-learning:latest  
 env:  
 - name: RAG\_DATA\_DIR  
 value: "/data"  
 - name: LLM\_PROVIDER  
 value: "openai"  
 - name: LLM\_MODEL  
 value: "gpt-4"  
 - name: OPENAI\_API\_KEY  
 valueFrom:  
 secretKeyRef:  
 name: openai-secret  
 key: api-key  
 volumeMounts:  
 - name: model-registry  
 mountPath: /data/model-registry  
 - name: models  
 mountPath: /data/models  
 - name: enhanced-rag-config  
 mountPath: /app/config/enhanced-rag-config.json  
 subPath: config.json  
 volumes:  
 - name: model-registry  
 persistentVolumeClaim:  
 claimName: model-registry-pvc  
 - name: models  
 persistentVolumeClaim:  
 claimName: models-pvc  
 - name: enhanced-rag-config  
 configMap:  
 name: enhanced-rag-config

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: continuous-learning  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: continuous-learning  
 template:  
 metadata:  
 labels:  
 app: continuous-learning  
 spec:  
 containers:  
 - name: continuous-learning  
 image: your-registry/continuous-learning:latest  
 env:  
 - name: RAG\_DATA\_DIR  
 value: "/data"  
 - name: LLM\_PROVIDER  
 value: "openai"  
 - name: LLM\_MODEL  
 value: "gpt-4"  
 - name: OPENAI\_API\_KEY  
 valueFrom:  
 secretKeyRef:  
 name: openai-secret  
 key: api-key  
 volumeMounts:  
 - name: model-registry  
 mountPath: /data/model-registry  
 - name: models  
 mountPath: /data/models  
 - name: enhanced-rag-config  
 mountPath: /app/config/enhanced-rag-config.json  
 subPath: config.json  
 volumes:  
 - name: model-registry  
 persistentVolumeClaim:  
 claimName: model-registry-pvc  
 - name: models  
 persistentVolumeClaim:  
 claimName: models-pvc  
 - name: enhanced-rag-config  
 configMap:  
 name: enhanced-rag-config

### CI/CD Integration

To integrate the enhanced RAG system with your CI/CD pipeline, you'll need to:

1. Add tests for the enhanced RAG system:

# In your CI pipeline configuration  
steps:  
 - name: Test Enhanced RAG  
 run: |  
 cd packages/ml/python  
 python test\_integration.py --config path/to/config.json

# In your CI pipeline configuration  
steps:  
 - name: Test Enhanced RAG  
 run: |  
 cd packages/ml/python  
 python test\_integration.py --config path/to/config.json

1. Add a step to sync the model registry:

# In your CI pipeline configuration  
steps:  
 - name: Sync Model Registry  
 run: |  
 # Script to sync model registry with persistent storage  
 python scripts/sync\_model\_registry.py

# In your CI pipeline configuration  
steps:  
 - name: Sync Model Registry  
 run: |  
 # Script to sync model registry with persistent storage  
 python scripts/sync\_model\_registry.py

1. Add a step to build and push the Docker images:

# In your CI pipeline configuration  
steps:  
 - name: Build and Push Docker Images  
 run: |  
 docker build -t your-registry/mcp-server:latest -f Dockerfile.mcp-server .  
 docker build -t your-registry/continuous-learning:latest -f Dockerfile.continuous-learning .  
 docker push your-registry/mcp-server:latest  
 docker push your-registry/continuous-learning:latest

# In your CI pipeline configuration  
steps:  
 - name: Build and Push Docker Images  
 run: |  
 docker build -t your-registry/mcp-server:latest -f Dockerfile.mcp-server .  
 docker build -t your-registry/continuous-learning:latest -f Dockerfile.continuous-learning .  
 docker push your-registry/mcp-server:latest  
 docker push your-registry/continuous-learning:latest

### Testing

You can test the enhanced RAG system using the provided test script:

# Test the enhanced RAG system  
python packages/ml/python/test\_integration.py --config path/to/config.json --setup

# Test the enhanced RAG system  
python packages/ml/python/test\_integration.py --config path/to/config.json --setup

This will set up the dependencies and run tests for both direct integration and MCP bridge integration.

### Implementation Checklist

Use this checklist to track the implementation of the enhanced RAG system:

#### Core Implementation

* [x] Create material-specific prompts module
* [x] Create continuous learning pipeline module
* [x] Create model registry module
* [x] Create hierarchical retriever module
* [x] Create cross-modal attention module
* [x] Create distributed retrieval module
* [x] Create enhanced RAG system module
* [x] Create RAG integration module
* [x] Create MCP RAG bridge module
* [x] Create configuration module

#### Integration with MCP Server

* [x] Create enhanced RAG service
* [x] Create enhanced RAG controller
* [x] Create enhanced RAG module
* [x] Create script to update app module
* [x] Create script to update MCP Dockerfile

#### Admin Panel Integration

* [x] Create enhanced RAG stats component
* [x] Create model registry component
* [x] Create enhanced RAG page
* [x] Create script to update admin routes
* [x] Create script to update admin sidebar

#### Continuous Learning Service

* [x] Create continuous learning service script
* [x] Create Dockerfile for continuous learning service
* [x] Create Kubernetes deployment for continuous learning service

#### CI/CD Integration

* [x] Create GitHub workflow for enhanced RAG system
* [x] Create script to build and push Docker images
* [x] Create script to deploy to Kubernetes

#### Documentation

* [x] Create enhanced RAG system documentation
* [x] Create enhanced RAG setup guide
* [x] Create enhanced RAG checklist

#### Testing

* [ ] Test material-specific prompts
* [ ] Test continuous learning pipeline
* [ ] Test model registry
* [ ] Test hierarchical retriever
* [ ] Test cross-modal attention
* [ ] Test distributed retrieval
* [ ] Test enhanced RAG system
* [ ] Test RAG integration
* [ ] Test MCP RAG bridge
* [ ] Test admin panel integration

#### Deployment

* [ ] Build and push Docker images
* [ ] Deploy to Kubernetes
* [ ] Verify deployment
* [ ] Monitor system performance

### Appendix

#### Complete Configuration Options

For reference, here's the complete configuration schema with default values:

DEFAULT\_CONFIG = {  
 # Service configuration  
 "service\_name": "material\_rag\_service",  
 "version": "1.0.0",  
 "enable\_cache": True,  
 "cache\_ttl": 3600, # 1 hour  
 "max\_cache\_size": 1000,  
 "timeout": 30, # seconds  
 "max\_concurrent\_requests": 10,  
  
 # Embedding configuration  
 "embedding": {  
 "default\_model": "sentence-transformers/all-MiniLM-L6-v2",  
 "dense\_dimension": 384,  
 "sparse\_enabled": True,  
 "sparse\_method": "bm25", # or "tfidf"  
 "normalize\_embeddings": True,  
 "pooling\_method": "mean", # or "max", "cls"  
 "batch\_size": 32  
 },  
  
 # Retrieval configuration  
 "retrieval": {  
 "max\_results": 10,  
 "strategy": "hybrid", # dense, sparse, hybrid, metadata  
 "threshold": 0.65,  
 "dense\_weight": 0.7,  
 "sparse\_weight": 0.3,  
 "reranking\_enabled": True,  
 "reranking\_model": "cross-encoder/ms-marco-MiniLM-L-6-v2",  
 "top\_k\_stage1": 50, # initial retrieval before re-ranking  
 "diversify\_results": True,  
 },  
  
 # Context assembly configuration  
 "assembly": {  
 "include\_relationships": True,  
 "max\_knowledge\_items": 20,  
 "include\_properties": True,  
 "max\_context\_tokens": 3000,  
 "relationship\_depth": 1,  
 "prioritize\_recency": True,  
 "include\_sources": True  
 },  
  
 # Generation configuration  
 "generation": {  
 "model": "gpt-4",  
 "temperature": 0.7,  
 "max\_tokens": 1000,  
 "streaming\_enabled": True,  
 "enhancement\_types": ["explanation", "similarity", "application"],  
 "citation\_style": "inline", # inline, footnote, endnote  
 "detail\_level": "medium", # brief, medium, detailed  
 "include\_source\_properties": True,  
 "include\_confidence\_scores": True,  
 "structured\_response": True  
 },  
  
 # Tracking configuration  
 "tracking\_enabled": True,  
 "log\_level": "info",  
 "metrics\_enabled": True,  
 "feedback\_collection": True  
}

DEFAULT\_CONFIG = {  
 # Service configuration  
 "service\_name": "material\_rag\_service",  
 "version": "1.0.0",  
 "enable\_cache": True,  
 "cache\_ttl": 3600, # 1 hour  
 "max\_cache\_size": 1000,  
 "timeout": 30, # seconds  
 "max\_concurrent\_requests": 10,  
  
 # Embedding configuration  
 "embedding": {  
 "default\_model": "sentence-transformers/all-MiniLM-L6-v2",  
 "dense\_dimension": 384,  
 "sparse\_enabled": True,  
 "sparse\_method": "bm25", # or "tfidf"  
 "normalize\_embeddings": True,  
 "pooling\_method": "mean", # or "max", "cls"  
 "batch\_size": 32  
 },  
  
 # Retrieval configuration  
 "retrieval": {  
 "max\_results": 10,  
 "strategy": "hybrid", # dense, sparse, hybrid, metadata  
 "threshold": 0.65,  
 "dense\_weight": 0.7,  
 "sparse\_weight": 0.3,  
 "reranking\_enabled": True,  
 "reranking\_model": "cross-encoder/ms-marco-MiniLM-L-6-v2",  
 "top\_k\_stage1": 50, # initial retrieval before re-ranking  
 "diversify\_results": True,  
 },  
  
 # Context assembly configuration  
 "assembly": {  
 "include\_relationships": True,  
 "max\_knowledge\_items": 20,  
 "include\_properties": True,  
 "max\_context\_tokens": 3000,  
 "relationship\_depth": 1,  
 "prioritize\_recency": True,  
 "include\_sources": True  
 },  
  
 # Generation configuration  
 "generation": {  
 "model": "gpt-4",  
 "temperature": 0.7,  
 "max\_tokens": 1000,  
 "streaming\_enabled": True,  
 "enhancement\_types": ["explanation", "similarity", "application"],  
 "citation\_style": "inline", # inline, footnote, endnote  
 "detail\_level": "medium", # brief, medium, detailed  
 "include\_source\_properties": True,  
 "include\_confidence\_scores": True,  
 "structured\_response": True  
 },  
  
 # Tracking configuration  
 "tracking\_enabled": True,  
 "log\_level": "info",  
 "metrics\_enabled": True,  
 "feedback\_collection": True  
}

#### API Reference

For a complete API reference, see the code documentation in each module or refer to:

* TypeScript API: packages/ml/src/rag-bridge.ts
* Python API: packages/ml/python/material\_rag\_service.py

packages/ml/src/rag-bridge.ts

packages/ml/python/material\_rag\_service.py

# Readme Site Deployment

Source: readme/readme-site-deployment.md

---

## KAI Documentation Site Deployment

This guide explains how to deploy the KAI documentation as a website with a sidebar navigation to https://github.com/Basilakis/kai-readme.github.io.

### Overview

The deployment process:

1. Converts all markdown files in the /readme/ folder to Docusaurus-compatible format
2. Organizes them into categories for the sidebar
3. Builds a Docusaurus site
4. Deploys the site to GitHub Pages

/readme/

### Setup Instructions

#### Manual Deployment

To manually deploy the documentation site:

1. Make sure you have Node.js 20 or later installed
2. Run the deployment script:

# Make the script executable  
chmod +x deploy-readme-site.sh  
  
# Run the script  
./deploy-readme-site.sh

# Make the script executable  
chmod +x deploy-readme-site.sh  
  
# Run the script  
./deploy-readme-site.sh

1. The script will:
2. Create a temporary Docusaurus site
3. Process all markdown files from the /readme/ folder
4. Build the site
5. Provide instructions for deployment
6. To test the site locally:

/readme/

Provide instructions for deployment

To test the site locally:

cd kai-docs-temp  
npm run start

cd kai-docs-temp  
npm run start

#### Automated Deployment with GitHub Actions

The repository includes a GitHub Actions workflow that automatically deploys the documentation site whenever changes are made to the /readme/ folder.

/readme/

To set up automated deployment:

1. Push this repository to GitHub
2. The built-in GITHUB\_TOKEN should have sufficient permissions for deployment
3. The workflow will automatically run when changes are pushed to the /readme/ folder
4. You can also manually trigger the workflow from the Actions tab

/readme/

### Customization

#### Modifying Categories

The documentation is organized into categories in the process-readme-files.js script. To modify the categories:

process-readme-files.js

1. Edit the categories object in process-readme-files.js
2. Add or remove files from each category
3. Run the deployment script again

categories

process-readme-files.js

#### Customizing the Site

To customize the Docusaurus site:

1. Edit the docusaurus.config.js file in the kai-docs-temp directory
2. Modify the theme, navigation, or other settings
3. Run the deployment script again

docusaurus.config.js

kai-docs-temp

### Technology Stack

The documentation site is built using:

1. Docusaurus 3.7 - The latest version of Docusaurus, providing modern features and improved performance
2. React 18 - For interactive UI components
3. Node.js 20 - For optimal compatibility with modern JavaScript features
4. Yarn - For dependency management

### Troubleshooting

If you encounter issues with the deployment:

1. Check that all dependencies are installed
2. Verify that the GitHub token has the correct permissions
3. Check the GitHub Actions logs for any errors
4. Ensure the repository settings allow GitHub Pages deployment

#### Common Issues

##### EUNSUPPORTEDPROTOCOL Error

If you see an error like npm error code EUNSUPPORTEDPROTOCOL or Unsupported URL Type "workspace:", this is because some Docusaurus templates use workspace protocol references in the package.json file. Our solution avoids this issue by:

npm error code EUNSUPPORTEDPROTOCOL

Unsupported URL Type "workspace:"

1. Creating a custom Docusaurus setup from scratch instead of using the default template
2. Using ES modules syntax instead of CommonJS
3. Using Yarn instead of npm for installation when available

This approach completely avoids the workspace protocol issue and provides a modern, stable deployment process.

### Requirements

* Node.js 20 or higher
* Git
* GitHub account with permissions to create repositories

### Additional Resources

* Docusaurus Documentation
* GitHub Pages Documentation
* GitHub Actions Documentation

# Recognition Assistant

Source: readme/recognition-assistant.md

---

## Recognition Assistant

This document provides detailed information about the Recognition Assistant, a specialized crewAI agent designed to enhance the material recognition workflow within the KAI platform.

### Overview

The Recognition Assistant uses artificial intelligence to help users identify and analyze materials through image recognition. It guides users through the recognition process, analyzes results, and provides detailed insights about materials to enhance the overall material identification experience.

### Key Capabilities

The Recognition Assistant offers multiple specialized functions:

1. Image Processing Guidance
2. Provide advice on capturing optimal images of materials
3. Suggest improvements for image quality to enhance recognition
4. Guide users on optimal lighting, angle, and distance
5. Recognition Result Analysis
6. Interpret and explain recognition results in detail
7. Analyze confidence scores and provide context
8. Compare similar materials and highlight key differences
9. Explain why certain materials were matched
10. Material Property Explanation
11. Provide detailed information about material properties
12. Explain material characteristics, applications, and limitations
13. Offer insights on material performance in different environments
14. Discuss maintenance requirements and durability factors
15. Alternative Suggestion
16. Recommend alternative materials with similar properties
17. Suggest materials that might better suit specific applications
18. Compare pros and cons of different material options
19. Provide context on availability and cost considerations
20. Educational Support
21. Explain material terminology and concepts
22. Provide historical context for different material types
23. Help users understand material classification systems
24. Answer detailed questions about construction materials

Guide users on optimal lighting, angle, and distance

Recognition Result Analysis

Explain why certain materials were matched

Material Property Explanation

Discuss maintenance requirements and durability factors

Alternative Suggestion

Provide context on availability and cost considerations

Educational Support

### Architecture

The Recognition Assistant integrates with the broader KAI platform through several key components:

#### Component Structure

packages/  
├── agents/  
│ ├── src/  
│ │ ├── frontend/  
│ │ │ └── recognitionAssistant.ts # Agent implementation  
│ │ ├── services/  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ ├── materialSearch.ts # Material search tool  
│ │ │ ├── imageAnalysis.ts # Image analysis tool  
│ │ │ ├── vectorSearch.ts # Vector search tool  
│ │ │ └── index.ts # Tool exports  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── client/  
 └── src/  
 └── components/  
 └── agents/  
 ├── RecognitionPanel.tsx # Client-side interface  
 └── AgentDashboard.tsx # Agent integration in UI

packages/  
├── agents/  
│ ├── src/  
│ │ ├── frontend/  
│ │ │ └── recognitionAssistant.ts # Agent implementation  
│ │ ├── services/  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ ├── tools/  
│ │ │ ├── materialSearch.ts # Material search tool  
│ │ │ ├── imageAnalysis.ts # Image analysis tool  
│ │ │ ├── vectorSearch.ts # Vector search tool  
│ │ │ └── index.ts # Tool exports  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── client/  
 └── src/  
 └── components/  
 └── agents/  
 ├── RecognitionPanel.tsx # Client-side interface  
 └── AgentDashboard.tsx # Agent integration in UI

#### Architectural Layers

1. Agent Layer (recognitionAssistant.ts)
2. Implements the agent's core capabilities
3. Defines specialized methods for recognition tasks
4. Processes user inputs and image uploads
5. Manages context for recognition operations
6. Service Layer (via ServiceFactory)
7. Provides access to material data and recognition systems
8. Handles API communication with error management
9. Formats requests and responses appropriately
10. Acts as a bridge to backend services
11. Tool Layer (materialSearch, imageAnalysis, vectorSearch)
12. Implements specialized tools for the agent to use
13. Enables material database queries
14. Provides image analysis capabilities
15. Facilitates vector-based similarity searches
16. UI Layer (RecognitionPanel.tsx)
17. Presents the agent's capabilities in the user interface
18. Handles image upload and display
19. Shows recognition results and agent insights
20. Provides chat interface for agent interaction

recognitionAssistant.ts

Manages context for recognition operations

Service Layer (via ServiceFactory)

Acts as a bridge to backend services

Tool Layer (materialSearch, imageAnalysis, vectorSearch)

Facilitates vector-based similarity searches

UI Layer (RecognitionPanel.tsx)

RecognitionPanel.tsx

### Implementation Details

#### Agent Implementation

The Recognition Assistant is a UserFacingAgent type that implements several specialized methods for recognition tasks:

export class RecognitionAssistant implements UserFacingAgent {  
 // Standard UserFacingAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // Recognition-specific methods  
 public async processUserInput(input: string, context?: Record<string, any>): Promise<string>;  
 public async processImage(imageUrl: string, metadata?: Record<string, any>): Promise<string>;  
 public async analyzeRecognitionResults(results: any[], context?: Record<string, any>): Promise<string>;  
 public async suggestImageImprovements(imageUrl: string, issues: string[]): Promise<string>;  
}

export class RecognitionAssistant implements UserFacingAgent {  
 // Standard UserFacingAgent properties  
 public id: string;  
 public type: AgentType;  
 public name: string;  
 public description: string;  
 public agent: Agent;  
 public config: AgentConfig;  
  
 // Recognition-specific methods  
 public async processUserInput(input: string, context?: Record<string, any>): Promise<string>;  
 public async processImage(imageUrl: string, metadata?: Record<string, any>): Promise<string>;  
 public async analyzeRecognitionResults(results: any[], context?: Record<string, any>): Promise<string>;  
 public async suggestImageImprovements(imageUrl: string, issues: string[]): Promise<string>;  
}

#### Agent Tools

The Recognition Assistant leverages specialized tools to perform its tasks:

// Tool for searching materials in the database  
const materialSearchTool = await createMaterialSearchTool();  
  
// Tool for analyzing images  
const imageAnalysisTool = await createImageAnalysisTool();  
  
// Tool for performing vector-based similarity searches  
const vectorSearchTool = await createVectorSearchTool();  
  
// Create a Recognition Assistant with tools  
const tools: Tool[] = [  
 materialSearchTool,  
 imageAnalysisTool,  
 vectorSearchTool  
];

// Tool for searching materials in the database  
const materialSearchTool = await createMaterialSearchTool();  
  
// Tool for analyzing images  
const imageAnalysisTool = await createImageAnalysisTool();  
  
// Tool for performing vector-based similarity searches  
const vectorSearchTool = await createVectorSearchTool();  
  
// Create a Recognition Assistant with tools  
const tools: Tool[] = [  
 materialSearchTool,  
 imageAnalysisTool,  
 vectorSearchTool  
];

#### Client-Side Integration

The Recognition Assistant is integrated into the client interface through a specialized panel that provides:

1. Image Upload Area - For submitting materials for recognition
2. Results Display - For viewing recognition results with confidence scores
3. Chat Interface - For interacting with the agent about results
4. Image Guidance - For tips on capturing better images

### Setup Instructions

#### Prerequisites

* Functioning KAI platform with image recognition capabilities
* CrewAI integration set up according to CrewAI installation guide
* Materials database with vector search capabilities

#### Installation

The Recognition Assistant is included in the standard crewAI integration package:

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

#### Configuration

Configure the agent in your application initialization:

import { createRecognitionAssistant } from '@kai/agents';  
  
// Create a Recognition Assistant instance  
const recognitionAssistant = await createRecognitionAssistant(  
 {  
 id: 'recognition-assistant-1',  
 // Additional configuration options  
 },  
 {  
 provider: 'openai',  
 name: 'gpt-4',  
 temperature: 0.7  
 }  
);

import { createRecognitionAssistant } from '@kai/agents';  
  
// Create a Recognition Assistant instance  
const recognitionAssistant = await createRecognitionAssistant(  
 {  
 id: 'recognition-assistant-1',  
 // Additional configuration options  
 },  
 {  
 provider: 'openai',  
 name: 'gpt-4',  
 temperature: 0.7  
 }  
);

### Usage Examples

#### Client-Side Integration

import React, { useState } from 'react';  
import { RecognitionPanel } from '../components/agents/RecognitionPanel';  
  
const RecognitionPage: React.FC = () => {  
 return (  
 <div className="recognition-page">  
 <h1>Material Recognition</h1>  
 <RecognitionPanel />  
 </div>  
 );  
};  
  
export default RecognitionPage;

import React, { useState } from 'react';  
import { RecognitionPanel } from '../components/agents/RecognitionPanel';  
  
const RecognitionPage: React.FC = () => {  
 return (  
 <div className="recognition-page">  
 <h1>Material Recognition</h1>  
 <RecognitionPanel />  
 </div>  
 );  
};  
  
export default RecognitionPage;

#### Processing an Uploaded Image

import { createRecognitionAssistant } from '@kai/agents';  
  
// Create the Recognition Assistant  
const recognitionAssistant = await createRecognitionAssistant(  
 { id: 'recognition-assistant-1' },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.7 }  
);  
  
// Process an uploaded image  
const imageUrl = 'https://example.com/uploaded-image.jpg';  
const metadata = {  
 originalFileName: 'bathroom-tile.jpg',  
 uploadedBy: 'user123',  
 dimensions: '1024x768'  
};  
  
const insights = await recognitionAssistant.processImage(imageUrl, metadata);  
console.log(insights);  
  
// Analyze recognition results  
const recognitionResults = [  
 { materialId: 'mat123', name: 'Ceramic Tile', confidence: 0.92 },  
 { materialId: 'mat456', name: 'Porcelain Tile', confidence: 0.87 },  
 { materialId: 'mat789', name: 'Quarry Tile', confidence: 0.65 }  
];  
  
const analysis = await recognitionAssistant.analyzeRecognitionResults(recognitionResults);  
console.log(analysis);  
  
// Get image improvement suggestions  
const imageIssues = ['poor lighting', 'blurry focus', 'inconsistent angle'];  
const suggestions = await recognitionAssistant.suggestImageImprovements(imageUrl, imageIssues);  
console.log(suggestions);

import { createRecognitionAssistant } from '@kai/agents';  
  
// Create the Recognition Assistant  
const recognitionAssistant = await createRecognitionAssistant(  
 { id: 'recognition-assistant-1' },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.7 }  
);  
  
// Process an uploaded image  
const imageUrl = 'https://example.com/uploaded-image.jpg';  
const metadata = {  
 originalFileName: 'bathroom-tile.jpg',  
 uploadedBy: 'user123',  
 dimensions: '1024x768'  
};  
  
const insights = await recognitionAssistant.processImage(imageUrl, metadata);  
console.log(insights);  
  
// Analyze recognition results  
const recognitionResults = [  
 { materialId: 'mat123', name: 'Ceramic Tile', confidence: 0.92 },  
 { materialId: 'mat456', name: 'Porcelain Tile', confidence: 0.87 },  
 { materialId: 'mat789', name: 'Quarry Tile', confidence: 0.65 }  
];  
  
const analysis = await recognitionAssistant.analyzeRecognitionResults(recognitionResults);  
console.log(analysis);  
  
// Get image improvement suggestions  
const imageIssues = ['poor lighting', 'blurry focus', 'inconsistent angle'];  
const suggestions = await recognitionAssistant.suggestImageImprovements(imageUrl, imageIssues);  
console.log(suggestions);

### Advanced Configuration

#### Custom Material Search Tools

Create custom tools to enhance the Recognition Assistant's capabilities:

import { Tool } from 'crewai';  
import { ServiceFactory } from '../services/serviceFactory';  
  
// Create a specialized material search tool for a specific category  
const createSpecializedCeramicSearchTool = async (): Promise<Tool> => {  
 const materialService = ServiceFactory.getInstance().materialService;  
  
 return new Tool({  
 name: 'ceramic\_tile\_search',  
 description: 'Search specifically for ceramic tiles with detailed filtering',  
 func: async (args) => {  
 const { query, filters } = JSON.parse(args);  
  
 // Add ceramic filter automatically  
 const enhancedFilters = {  
 ...filters,  
 category: 'ceramic'  
 };  
  
 const results = await materialService.searchMaterials(query, enhancedFilters);  
 return JSON.stringify(results);  
 }  
 });  
};  
  
// Add it to the agent  
const recognitionAssistant = await createRecognitionAssistant(  
 { id: 'ceramic-specialist-1' },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.7 },  
 [await createSpecializedCeramicSearchTool()]  
);

import { Tool } from 'crewai';  
import { ServiceFactory } from '../services/serviceFactory';  
  
// Create a specialized material search tool for a specific category  
const createSpecializedCeramicSearchTool = async (): Promise<Tool> => {  
 const materialService = ServiceFactory.getInstance().materialService;  
  
 return new Tool({  
 name: 'ceramic\_tile\_search',  
 description: 'Search specifically for ceramic tiles with detailed filtering',  
 func: async (args) => {  
 const { query, filters } = JSON.parse(args);  
  
 // Add ceramic filter automatically  
 const enhancedFilters = {  
 ...filters,  
 category: 'ceramic'  
 };  
  
 const results = await materialService.searchMaterials(query, enhancedFilters);  
 return JSON.stringify(results);  
 }  
 });  
};  
  
// Add it to the agent  
const recognitionAssistant = await createRecognitionAssistant(  
 { id: 'ceramic-specialist-1' },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.7 },  
 [await createSpecializedCeramicSearchTool()]  
);

#### Integration with External Recognition Systems

Connect the Recognition Assistant to external material recognition APIs:

import { Tool } from 'crewai';  
import axios from 'axios';  
  
// Create a tool for external material recognition API  
const createExternalRecognitionTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'external\_recognition',  
 description: 'Perform material recognition using an external specialized API',  
 func: async (args) => {  
 const { imageUrl } = JSON.parse(args);  
  
 try {  
 const response = await axios.post('https://api.external-recognition.com/analyze', {  
 image: imageUrl,  
 apiKey: process.env.EXTERNAL\_API\_KEY  
 });  
  
 return JSON.stringify(response.data);  
 } catch (error) {  
 return JSON.stringify({ error: 'External recognition failed', details: error.message });  
 }  
 }  
 });  
};  
  
// Add it to the agent  
const recognitionAssistant = await createRecognitionAssistant(  
 { id: 'multi-system-recognition-1' },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.7 },  
 [await createExternalRecognitionTool()]  
);

import { Tool } from 'crewai';  
import axios from 'axios';  
  
// Create a tool for external material recognition API  
const createExternalRecognitionTool = async (): Promise<Tool> => {  
 return new Tool({  
 name: 'external\_recognition',  
 description: 'Perform material recognition using an external specialized API',  
 func: async (args) => {  
 const { imageUrl } = JSON.parse(args);  
  
 try {  
 const response = await axios.post('https://api.external-recognition.com/analyze', {  
 image: imageUrl,  
 apiKey: process.env.EXTERNAL\_API\_KEY  
 });  
  
 return JSON.stringify(response.data);  
 } catch (error) {  
 return JSON.stringify({ error: 'External recognition failed', details: error.message });  
 }  
 }  
 });  
};  
  
// Add it to the agent  
const recognitionAssistant = await createRecognitionAssistant(  
 { id: 'multi-system-recognition-1' },  
 { provider: 'openai', name: 'gpt-4', temperature: 0.7 },  
 [await createExternalRecognitionTool()]  
);

### Performance Considerations

#### Optimizing Image Processing

1. Image Preprocessing
2. Implement client-side image resizing before upload
3. Use WebWorkers for browser-based image optimization
4. Apply appropriate compression based on network conditions
5. Recognition Caching
6. Cache recognition results for previously analyzed images
7. Implement fingerprinting for image similarity detection
8. Use TTL-based cache invalidation for freshness
9. Progressive Loading
10. Show preliminary results while detailed analysis continues
11. Implement priority-based processing queue
12. Display confidence thresholds incrementally

Apply appropriate compression based on network conditions

Recognition Caching

Use TTL-based cache invalidation for freshness

Progressive Loading

### Security Considerations

1. Image Storage
2. Implement proper access controls for uploaded images
3. Define appropriate retention policies
4. Sanitize metadata to prevent information leakage
5. User Data Protection
6. Limit collection of identifying information in recognition logs
7. Apply appropriate anonymization for analytics
8. Implement proper consent mechanisms for data usage
9. Agent Boundaries
10. Restrict the agent to recognition-related operations
11. Validate inputs to prevent injection attacks
12. Limit sensitive information in responses

Sanitize metadata to prevent information leakage

User Data Protection

Implement proper consent mechanisms for data usage

Agent Boundaries

### Related Documentation

* Material Recognition - Core recognition system architecture
* CrewAI Integration - Overall agent system architecture
* CrewAI Implementation - Implementation details
* Agent Installation - Setup instructions
* Client Integration - Frontend framework details

# Relationship Aware Model Training

Source: readme/relationship-aware-model-training.md

---

## Relationship-Aware Model Training and Search Enhancement

This document describes the Relationship-Aware Model Training and Search Enhancement features, which leverage the Property Relationship Graph to improve AI model training and search functionality.

### Overview

The Relationship-Aware Model Training and Search Enhancement features use the knowledge captured in the Property Relationship Graph to:

1. Enhance AI Model Training: Improve property prediction models by incorporating relationship data
2. Enhance Search Functionality: Improve search relevance and recommendations using relationship data

### Relationship-Aware Model Training

#### Architecture

The Relationship-Aware Model Training feature consists of the following components:

##### Relationship-Aware Training Service

The relationshipAwareTrainingService is responsible for:

relationshipAwareTrainingService

* Extracting features from property relationships for AI model training
* Generating training data with relationship-based features
* Training models with relationship-enhanced features
* Evaluating model performance and relationship contribution
* Managing training jobs and model registry

##### Relationship Feature Extractor

The service includes feature extraction capabilities:

* Extracting direct and indirect relationships from the Property Relationship Graph
* Weighting relationships based on strength and relevance
* Converting relationship data into model features
* Handling different relationship types appropriately

##### Model Training and Evaluation

The service handles model training and evaluation:

* Creating and training models with relationship features
* Evaluating model performance with and without relationship features
* Calculating feature importance and relationship contribution
* Providing detailed performance metrics and insights

##### API Endpoints

The following API endpoints are available for relationship-aware training:

* POST /api/ai/relationship-aware-training/train: Train a relationship-aware model
* GET /api/ai/relationship-aware-training/job/:jobId: Get training job status

POST /api/ai/relationship-aware-training/train

GET /api/ai/relationship-aware-training/job/:jobId

#### Key Features

1. Relationship-Aware Feature Engineering: The system extracts features from the Property Relationship Graph to enhance model training.
2. Indirect Relationship Discovery: The system can discover and utilize multi-hop relationships between properties.
3. Relationship Strength Weighting: Relationships are weighted based on their strength and relevance to the target property.
4. Relationship Type Handling: Different relationship types (correlation, dependency, compatibility, etc.) are handled appropriately.
5. Performance Comparison: The system compares model performance with and without relationship features.
6. Feature Importance Analysis: The system analyzes and visualizes the importance of different features, including relationship features.
7. Relationship Contribution Metrics: The system provides metrics on how much relationships contribute to model performance.
8. Model Registry and Management: Trained models are stored, versioned, and managed for future use.

#### Usage Examples

##### Training a Relationship-Aware Model

// Train a relationship-aware model to predict finish based on other properties  
const result = await fetch('/api/ai/relationship-aware-training/train', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 materialType: 'tile',  
 targetProperty: 'finish',  
 options: {  
 includeRelationships: true,  
 relationshipTypes: ['correlates\_with', 'depends\_on', 'compatibility'],  
 relationshipStrengthThreshold: 0.3,  
 maxRelationshipDepth: 2,  
 useTransferLearning: true,  
 epochs: 50,  
 batchSize: 32,  
 learningRate: 0.001,  
 validationSplit: 0.2  
 }  
 })  
});  
  
// Get training result  
const data = await result.json();  
console.log('Model ID:', data.result.modelId);  
console.log('Accuracy:', data.result.accuracy);  
console.log('Validation Accuracy:', data.result.validationAccuracy);  
console.log('Baseline Accuracy:', data.result.baselineAccuracy);  
console.log('Improvement:', data.result.improvementPercentage + '%');

// Train a relationship-aware model to predict finish based on other properties  
const result = await fetch('/api/ai/relationship-aware-training/train', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 materialType: 'tile',  
 targetProperty: 'finish',  
 options: {  
 includeRelationships: true,  
 relationshipTypes: ['correlates\_with', 'depends\_on', 'compatibility'],  
 relationshipStrengthThreshold: 0.3,  
 maxRelationshipDepth: 2,  
 useTransferLearning: true,  
 epochs: 50,  
 batchSize: 32,  
 learningRate: 0.001,  
 validationSplit: 0.2  
 }  
 })  
});  
  
// Get training result  
const data = await result.json();  
console.log('Model ID:', data.result.modelId);  
console.log('Accuracy:', data.result.accuracy);  
console.log('Validation Accuracy:', data.result.validationAccuracy);  
console.log('Baseline Accuracy:', data.result.baselineAccuracy);  
console.log('Improvement:', data.result.improvementPercentage + '%');

##### Checking Training Job Status

// Check training job status  
const status = await fetch(`/api/ai/relationship-aware-training/job/${jobId}`, {  
 method: 'GET',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 }  
});  
  
// Display job status  
const data = await status.json();  
console.log('Job Status:', data.status.status);  
console.log('Progress:', data.status.progress \* 100 + '%');  
console.log('Created At:', new Date(data.status.createdAt).toLocaleString());  
console.log('Updated At:', new Date(data.status.updatedAt).toLocaleString());

// Check training job status  
const status = await fetch(`/api/ai/relationship-aware-training/job/${jobId}`, {  
 method: 'GET',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 }  
});  
  
// Display job status  
const data = await status.json();  
console.log('Job Status:', data.status.status);  
console.log('Progress:', data.status.progress \* 100 + '%');  
console.log('Created At:', new Date(data.status.createdAt).toLocaleString());  
console.log('Updated At:', new Date(data.status.updatedAt).toLocaleString());

##### Using the Relationship-Aware Training Form

import { RelationshipAwareTrainingForm } from '../components/RelationshipAwareTraining';  
  
// In your component  
const handleTrainingComplete = (result) => {  
 console.log('Training completed:', result);  
 // Do something with the result  
};  
  
// In your render method  
return (  
 <RelationshipAwareTrainingForm onTrainingComplete={handleTrainingComplete} />  
);

import { RelationshipAwareTrainingForm } from '../components/RelationshipAwareTraining';  
  
// In your component  
const handleTrainingComplete = (result) => {  
 console.log('Training completed:', result);  
 // Do something with the result  
};  
  
// In your render method  
return (  
 <RelationshipAwareTrainingForm onTrainingComplete={handleTrainingComplete} />  
);

##### Displaying Model Results

import { RelationshipAwareModelResults } from '../components/RelationshipAwareTraining';  
  
// In your component  
const [modelResult, setModelResult] = useState(null);  
  
// After training is complete  
const handleTrainingComplete = (result) => {  
 setModelResult(result);  
};  
  
// In your render method  
return (  
 <>  
 <RelationshipAwareTrainingForm onTrainingComplete={handleTrainingComplete} />  
 {modelResult && <RelationshipAwareModelResults result={modelResult} />}  
 </>  
);

import { RelationshipAwareModelResults } from '../components/RelationshipAwareTraining';  
  
// In your component  
const [modelResult, setModelResult] = useState(null);  
  
// After training is complete  
const handleTrainingComplete = (result) => {  
 setModelResult(result);  
};  
  
// In your render method  
return (  
 <>  
 <RelationshipAwareTrainingForm onTrainingComplete={handleTrainingComplete} />  
 {modelResult && <RelationshipAwareModelResults result={modelResult} />}  
 </>  
);

### Relationship-Enhanced Search

#### Architecture

The Relationship-Enhanced Search feature consists of the following components:

##### Relationship Enhanced Search Service

The relationshipEnhancedSearch service is responsible for:

relationshipEnhancedSearch

* Expanding search queries using relationship data
* Calculating relationship-based relevance scores
* Reranking search results based on relationship relevance
* Generating related search suggestions

##### API Endpoints

The following API endpoints are available for relationship-enhanced search:

* POST /api/search/relationship-enhanced: Perform a search with relationship-based reranking
* POST /api/search/expand-query: Expand a search query using relationship data
* POST /api/search/related-searches: Generate related search suggestions

POST /api/search/relationship-enhanced

POST /api/search/expand-query

POST /api/search/related-searches

##### UI Components

The following UI components are available for relationship-enhanced search:

* RelationshipEnhancedSearchProvider: Context provider for relationship-enhanced search
* RelatedSearches: Component for displaying related search suggestions
* RelationshipEnhancedResults: Component for displaying search results with relationship scores

RelationshipEnhancedSearchProvider

RelatedSearches

RelationshipEnhancedResults

#### Key Features

1. Query Expansion: The system expands search queries based on correlations and compatibility relationships.
2. Relevance Scoring: Search results are scored based on relationship strength and compatibility.
3. Result Reranking: Search results are reranked based on relationship relevance.
4. Related Search Suggestions: The system generates related search suggestions based on property relationships.

#### Usage Examples

##### Performing a Relationship-Enhanced Search

// Perform a relationship-enhanced search  
const results = await fetch('/api/search/relationship-enhanced', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 materialType: 'tile',  
 query: {  
 material: 'porcelain',  
 finish: 'matte'  
 },  
 results: originalResults // Results from a standard search  
 })  
});  
  
// Display reranked results  
const data = await results.json();  
console.log('Reranked results:');  
data.results.forEach(result => {  
 console.log(`- ${result.properties.name} (score: ${result.finalScore.toFixed(2)})`);  
});

// Perform a relationship-enhanced search  
const results = await fetch('/api/search/relationship-enhanced', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 materialType: 'tile',  
 query: {  
 material: 'porcelain',  
 finish: 'matte'  
 },  
 results: originalResults // Results from a standard search  
 })  
});  
  
// Display reranked results  
const data = await results.json();  
console.log('Reranked results:');  
data.results.forEach(result => {  
 console.log(`- ${result.properties.name} (score: ${result.finalScore.toFixed(2)})`);  
});

##### Expanding a Search Query

// Expand a search query using relationship data  
const expandedQuery = await fetch('/api/search/expand-query', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 materialType: 'tile',  
 query: {  
 material: 'porcelain',  
 finish: 'matte'  
 }  
 })  
});  
  
// Use expanded query for search  
const data = await expandedQuery.json();  
console.log('Expanded query:', data.expandedQuery);

// Expand a search query using relationship data  
const expandedQuery = await fetch('/api/search/expand-query', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 materialType: 'tile',  
 query: {  
 material: 'porcelain',  
 finish: 'matte'  
 }  
 })  
});  
  
// Use expanded query for search  
const data = await expandedQuery.json();  
console.log('Expanded query:', data.expandedQuery);

##### Getting Related Search Suggestions

// Get related search suggestions  
const relatedSearches = await fetch('/api/search/related-searches', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 materialType: 'tile',  
 query: {  
 material: 'porcelain',  
 finish: 'matte'  
 }  
 })  
});  
  
// Display related search suggestions  
const data = await relatedSearches.json();  
console.log('Related searches:');  
data.relatedSearches.forEach(suggestion => {  
 console.log(`- ${suggestion.property}: ${suggestion.value} (confidence: ${suggestion.confidence.toFixed(2)})`);  
});

// Get related search suggestions  
const relatedSearches = await fetch('/api/search/related-searches', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 materialType: 'tile',  
 query: {  
 material: 'porcelain',  
 finish: 'matte'  
 }  
 })  
});  
  
// Display related search suggestions  
const data = await relatedSearches.json();  
console.log('Related searches:');  
data.relatedSearches.forEach(suggestion => {  
 console.log(`- ${suggestion.property}: ${suggestion.value} (confidence: ${suggestion.confidence.toFixed(2)})`);  
});

### Integration with Other Features

#### Property Relationship Graph Integration

Both features are tightly integrated with the Property Relationship Graph:

1. Model Training: Uses relationship data to enhance feature engineering and prediction
2. Search Enhancement: Uses relationship data to improve relevance and recommendations

#### Material Metadata Panel Integration

The Property Prediction feature can be integrated with the Material Metadata Panel to:

1. Auto-fill Properties: Automatically fill in predicted property values
2. Validate Properties: Validate property values against predictions

#### Search Interface Integration

The Relationship-Enhanced Search feature can be integrated with the search interface to:

1. Display Related Searches: Show related search suggestions
2. Highlight Relationship Scores: Display relationship scores for search results
3. Explain Relevance: Explain why certain results are relevant

### Benefits

These features provide several benefits:

1. Improved AI Model Accuracy: AI models can make better predictions by leveraging relationship data, with measurable improvements in accuracy.
2. Enhanced Feature Engineering: Relationship data provides valuable features that might not be captured in the raw material properties.
3. Deeper Insights: The system provides insights into which relationships are most important for predicting different properties.
4. More Relevant Search: Search results are more relevant due to relationship-based scoring and query expansion.
5. Better Recommendations: Users receive better recommendations based on relationship context and property correlations.
6. Enhanced User Experience: Users can find what they're looking for more easily and discover related materials.
7. Continuous Improvement: The feedback loop between AI models and relationship data creates a virtuous cycle of improvement.
8. Data Quality Enhancement: Anomaly detection helps identify data quality issues and inconsistencies.

### Future Enhancements

Potential future enhancements to these features:

1. Multi-Property Prediction: Predict multiple properties simultaneously using a single model
2. Automated Relationship Discovery: Automatically discover new relationships from data patterns
3. Real-time Model Updates: Update models in real-time as new relationship data becomes available
4. Personalized Search: Incorporate user preferences and behavior into relationship-based search
5. User Feedback Integration: Use explicit user feedback to improve relationship data and model training
6. Advanced Visualization: Provide more advanced visualizations of relationship impacts and model performance
7. Multi-Modal Relationship Learning: Incorporate image and text data into relationship learning
8. Cross-Domain Relationships: Extend relationship graph to include cross-domain relationships
9. Explainable AI: Provide more detailed explanations of how relationships influence predictions
10. Distributed Training: Support distributed training for larger models and datasets

### Conclusion

The Relationship-Aware Model Training and Search Enhancement features leverage the knowledge captured in the Property Relationship Graph to improve AI model training and search functionality. By incorporating relationship data, these features provide smarter predictions, more relevant search results, and better recommendations, enhancing the overall user experience.

# Security

Source: readme/security.md

---

## Security Enhancements

This document outlines the security enhancements implemented in the KAI application, particularly focusing on authentication, session management, and API key handling.

### Cryptographic Operations

We've implemented a dedicated cryptographic utilities module (src/utils/crypto.ts) that provides secure methods for:

src/utils/crypto.ts

* Generating secure random tokens
* Hashing passwords and tokens
* Generating and verifying TOTP secrets
* Creating and validating API keys
* Generating backup codes

All cryptographic operations use industry-standard algorithms and practices:

* SHA-256 for hashing
* PBKDF2 with 10,000 iterations for password hashing
* Cryptographically secure random number generation
* Timing-safe comparisons for token validation

### Security Logging

We've implemented enhanced security logging (src/utils/securityLogger.ts) that:

src/utils/securityLogger.ts

* Logs all security-relevant events (authentication attempts, token usage, etc.)
* Sanitizes sensitive information before logging
* Categorizes events by type and outcome
* Includes contextual information (IP address, user agent, etc.)

This logging system helps with:

* Security auditing
* Detecting potential security incidents
* Compliance requirements
* Troubleshooting authentication issues

### HTTP Security Headers

We've added security headers to all API responses:

* X-Content-Type-Options: nosniff - Prevents MIME type sniffing
* X-Frame-Options: DENY - Prevents clickjacking attacks
* X-XSS-Protection: 1; mode=block - Helps prevent XSS attacks
* Strict-Transport-Security - Enforces HTTPS
* Referrer-Policy - Controls referrer information
* Feature-Policy - Restricts browser features
* Content-Security-Policy - Controls resource loading

X-Content-Type-Options: nosniff

X-Frame-Options: DENY

X-XSS-Protection: 1; mode=block

Strict-Transport-Security

Referrer-Policy

Feature-Policy

Content-Security-Policy

Additionally, we've added cache control headers to sensitive routes:

* Cache-Control: no-store, no-cache, must-revalidate, proxy-revalidate
* Pragma: no-cache
* Expires: 0
* Surrogate-Control: no-store

Cache-Control: no-store, no-cache, must-revalidate, proxy-revalidate

Pragma: no-cache

Expires: 0

Surrogate-Control: no-store

These headers prevent caching of sensitive information by browsers and proxies.

### Two-Factor Authentication

Our two-factor authentication implementation includes:

* Support for multiple methods (TOTP, SMS, Email)
* Secure generation and storage of TOTP secrets
* Backup codes for account recovery
* Detailed security logging of all 2FA operations
* Rate limiting to prevent brute force attacks

### Session Management

Our session management system includes:

* Secure generation of session tokens
* Hashed storage of tokens in the database
* Automatic session expiration
* Ability to revoke sessions
* Detailed logging of session creation and usage

### API Key Management

Our API key management system includes:

* Secure generation of API keys
* Hashed storage of keys in the database
* Scoped permissions for API keys
* Ability to revoke keys
* Detailed logging of key creation and usage

### Rate Limiting

We've implemented rate limiting on sensitive endpoints to prevent:

* Brute force attacks
* Denial of service attacks
* Excessive API usage

### Best Practices

Throughout the codebase, we follow security best practices:

* Input validation on all user inputs
* Parameterized queries to prevent SQL injection
* Error handling that doesn't leak sensitive information
* Principle of least privilege for API endpoints
* Regular security audits and code reviews

### Future Enhancements

Planned security enhancements include:

* Implementing a Web Application Firewall (WAF)
* Adding anomaly detection for authentication attempts
* Implementing IP-based blocking for suspicious activity
* Regular security penetration testing

# Shared Implementation Plans

Source: readme/shared-implementation-plans.md

---

## Implementation Plans

### Phase 1: Knowledge Base Enhancement

#### 1. Data Structure & Schema Implementation

* Timeline: Weeks 1-2
* Objective: Complete the searchable database implementation for material specifications
* Tasks:
* Design comprehensive schema for material specifications including physical properties, visual attributes, and application contexts
* Implement full-text search capabilities with relevance scoring
* Create relationships between materials, collections, and manufacturers
* Develop metadata templates for consistent information extraction from various sources
* Implement validation rules for data integrity

#### 2. Tagging & Organization System

* Timeline: Weeks 2-3
* Objective: Implement complete tagging system for organizing tiles by collections/series
* Tasks:
* Design hierarchical taxonomy for material categorization
* Implement tag management with parent-child relationships
* Create tag suggestion algorithms based on existing material properties
* Develop bulk tagging capabilities for collection management
* Implement tag analytics to measure usage and effectiveness

#### 3. ML Integration Layer

* Timeline: Weeks 3-5
* Objective: Connect knowledge base with ML models and training systems
* Tasks:
* Create data pipeline between knowledge base and ML training infrastructure
* Implement feedback loop from ML recognition results to knowledge base entries
* Develop confidence scoring for knowledge base entries based on ML verification
* Create labeling interface for training data generation from knowledge base
* Implement feature vector storage for material specifications

#### 4. PDF Processing Integration

* Timeline: Weeks 5-7
* Objective: Connect PDF processing pipeline with knowledge base
* Tasks:
* Develop extractors for structured material data from catalogs
* Create mapping between extracted PDF data and knowledge base schema
* Implement validation workflows for automated extraction
* Design reconciliation process for conflicting information
* Build dashboards for tracking extraction quality metrics

#### 5. Web Crawling Integration

* Timeline: Weeks 7-9
* Objective: Connect web crawler data with knowledge base
* Tasks:
* Design parsers for common manufacturer website structures
* Create normalization pipeline for web-extracted data
* Implement deduplication with existing knowledge base entries
* Develop change detection for updated specifications
* Build source attribution and confidence scoring system

#### 6. Versioning System

* Timeline: Weeks 9-10
* Objective: Implement versioning system for knowledge base updates
* Tasks:
* Design temporal data model for tracking changes over time
* Implement differential storage for efficient version history
* Create rollback capabilities for corrupted updates
* Develop comparison tools for version differences
* Build audit trails for regulatory compliance

#### 7. Index Optimization

* Timeline: Weeks 10-11
* Objective: Optimize knowledge base for efficient retrieval
* Tasks:
* Implement specialized indexes for common query patterns
* Create caching layer for frequently accessed data
* Develop query analysis tools to identify optimization opportunities
* Implement auto-scaling capabilities for search infrastructure
* Design performance monitoring and alerting system

#### 8. Admin Interface

* Timeline: Weeks 11-13
* Objective: Develop comprehensive admin interfaces for knowledge base management
* Tasks:
* Build CRUD interfaces for all knowledge base entities
* Create dashboard for monitoring knowledge base health
* Implement batch operations for bulk updates
* Develop approval workflows for quality control
* Create user permission system for differentiated access levels

#### 9. Quality Assurance System

* Timeline: Weeks 13-14
* Objective: Implement robust QA for knowledge base content
* Tasks:
* Design automated consistency checks for material properties
* Create statistical anomaly detection for suspect values
* Implement user feedback collection for incorrect information
* Develop confidence scoring for knowledge base entries
* Build reporting system for knowledge base quality metrics

#### 10. Integration Testing & Deployment

* Timeline: Weeks 14-16
* Objective: Ensure system reliability and deploy to production
* Tasks:
* Create comprehensive test suite for all knowledge base functions
* Implement performance testing under various load conditions
* Develop migration plan for existing data
* Design rollout strategy with feature flags
* Create monitoring and alerting for production environment

### Phase 2: Agent Framework Integration (Future Phase)

#### 1. Agent Framework Foundation

* Objective: Establish the core agent infrastructure
* Tasks:
* Select appropriate framework (LangChain, LlamaIndex, etc.)
* Create development environment and CI/CD pipeline
* Implement core agent routing system
* Design conversation state management
* Build logging and monitoring infrastructure

#### 2. Knowledge Base Connector

* Objective: Connect agent to the knowledge base
* Tasks:
* Create vector representation of knowledge base content
* Implement semantic search capabilities over knowledge base
* Develop context retrieval strategies for queries
* Build knowledge synthesis from multiple entries
* Create explanation generation for retrieved information

#### 3. ML Model Integration

* Objective: Enable agent to leverage existing ML capabilities
* Tasks:
* Create API wrappers for all ML services
* Implement image processing pipeline for agent requests
* Develop multi-modal reasoning (text + image)
* Build confidence scoring for ML results
* Create explanation generation for ML decisions

#### 4. Natural Language Understanding

* Objective: Improve comprehension of domain-specific queries
* Tasks:
* Create tile industry ontology for entity recognition
* Implement domain-specific intent detection
* Develop specialized prompt engineering for material queries
* Build query reformulation for ambiguous requests
* Create measurement and unit standardization

#### 5. Conversation Management

* Objective: Enable complex multi-turn interactions
* Tasks:
* Implement stateful conversation tracking
* Create clarification workflows for ambiguous queries
* Develop response templating system
* Build persona management for consistent tone
* Implement conversation summaries and bookmarks

#### 6. UI Integration

* Objective: Create seamless user experience
* Tasks:
* Design conversational UI components
* Implement rich result formatting
* Create multi-modal input (text, image, file)
* Develop responsive layouts for all devices
* Build accessibility features for inclusive design

#### 7. Testing & Optimization

* Objective: Ensure agent quality and performance
* Tasks:
* Create comprehensive test suite for common scenarios
* Implement user feedback collection and analysis
* Develop performance optimization for response time
* Build continuous improvement pipeline
* Create benchmark system for measuring improvements

#### 8. Deployment & Rollout

* Objective: Successfully deploy to production
* Tasks:
* Design phased rollout strategy
* Create user onboarding and help content
* Implement monitoring and alerting
* Build analytics dashboard for usage patterns
* Develop feedback collection system

# Shared Material Metadata Extraction

Source: readme/shared-material-metadata-extraction.md

---

## Material Metadata Extraction System

This document explains how the material metadata fields are defined, extracted, and used throughout the system.

### Overview

The material metadata system is designed to capture specific properties for different types of materials (Tiles, Wood, Lighting, Furniture, Decoration). These properties are used in several ways:

1. Admin Panel Display: Organizing and displaying material metadata in a structured way
2. Data Extraction: Automatically extracting metadata from PDF documents and websites using AI
3. Filtering and Search: Allowing users to filter materials based on specific properties
4. AI Training: Providing structured data for training material recognition models
5. User Education: Help users understand what each property value looks like in practice
6. Object Identification: Enhance the system's ability to identify objects and their properties

### Material Categories and Their Metadata

The system supports the following material categories, each with specific metadata fields:

#### Tiles

* Physical Properties: Size (e.g., 60x60, 90x90), Thickness, Material
* Technical Properties:
* V-Rating (V1-V4): Pattern variation
* R-Rating (R9-R13): Slip resistance
* Water Absorption
* Frost Resistance
* PEI Rating
* Appearance: Finish, Rectified
* Usage: Usage Area, Antibacterial properties

#### Wood

* Physical Properties: Wood Type, Construction, Thickness, Width, Length
* Technical Properties: Grade, Hardness, Moisture Content, Stability
* Appearance & Installation: Finish, Installation System, Underfloor Heating compatibility

#### Lighting

* General Properties: Lighting Type, Material, Dimensions, Weight
* Technical Specifications: Bulb Type, Wattage, Voltage, Lumens, Color Temperature, Energy Class
* Features: IP Rating, Control System

#### Furniture

* General Properties: Furniture Type, Style, Material, Dimensions
* Physical Attributes: Weight, Weight Capacity, Assembly Required
* Construction: Frame Construction, Cushion Filling, Upholstery
* Features: Adjustable, Outdoor Use, Sustainability

#### Decoration

* General Properties: Decoration Type, Style, Material, Dimensions
* Design & Composition: Theme, Technique, Set Size
* Usage & Care: Occasion, Indoor/Outdoor, Mounting Type, Fragility, Care Instructions
* Additional Information: Sustainability

### System Components

The metadata system consists of the following components:

#### 1. Database Schema (004\_material\_metadata\_fields.sql)

004\_material\_metadata\_fields.sql

* material\_metadata\_fields table: Stores definitions of all metadata fields
* Extraction hints: Regex patterns for automatic extraction
* Functions: Automatic processing of text to extract metadata values
* Triggers: Automatically update metadata when descriptions change

material\_metadata\_fields

#### 2. TypeScript Interfaces (metadata.ts)

metadata.ts

* Base MaterialMetadata interface with common properties
* Specialized interfaces for each material type
* Type guards for safely working with different material types
* Helper functions for accessing and validating metadata

MaterialMetadata

#### 3. Admin Panel Component (MaterialMetadataPanel.tsx)

MaterialMetadataPanel.tsx

* React component for displaying and editing metadata
* Organized by category and field groups
* Support for different field types (text, number, dropdown, boolean)
* Read and edit modes

### Extraction Process

The system is designed to automatically extract metadata from text descriptions using pattern matching:

1. When a material description is added/updated, the database trigger activates
2. The extract\_metadata\_fields function is called with the text and material type
3. For each registered field, it tries to match the text using extraction patterns
4. When a match is found, the value is extracted and converted to the appropriate type
5. The material's metadata JSON is updated with the extracted values

extract\_metadata\_fields

#### Example Extraction Patterns

-- For V-Rating (Tiles)  
(?i)(?:shade|color) variation:?\\s\*(V\\d)  
(?i)(V\\d)\\s\*(?:shade|color) variation  
(?i)variation:?\\s\*(V\\d)  
  
-- For R-Rating (Tiles)  
(?i)slip resistance:?\\s\*(R\\d{1,2})  
(?i)(R\\d{1,2})\\s\*slip resistance  
(?i)(?:slip|resistance) rating:?\\s\*(R\\d{1,2})  
  
-- For Size (common)  
(?i)(?:size|dimensions):?\\s\*(\\d+\\s\*[x×]\\s\*\\d+)(?:\\s\*cm)?  
(?i)(\\d+\\s\*[x×]\\s\*\\d+)(?:\\s\*cm)

-- For V-Rating (Tiles)  
(?i)(?:shade|color) variation:?\\s\*(V\\d)  
(?i)(V\\d)\\s\*(?:shade|color) variation  
(?i)variation:?\\s\*(V\\d)  
  
-- For R-Rating (Tiles)  
(?i)slip resistance:?\\s\*(R\\d{1,2})  
(?i)(R\\d{1,2})\\s\*slip resistance  
(?i)(?:slip|resistance) rating:?\\s\*(R\\d{1,2})  
  
-- For Size (common)  
(?i)(?:size|dimensions):?\\s\*(\\d+\\s\*[x×]\\s\*\\d+)(?:\\s\*cm)?  
(?i)(\\d+\\s\*[x×]\\s\*\\d+)(?:\\s\*cm)

### Integration with AI/ML

The extraction patterns and field definitions provide essential training data for machine learning models. When the system processes PDFs or crawls websites, it:

1. Uses OCR and text extraction to obtain raw text
2. Applies the extraction patterns to identify key metadata
3. Builds vector representations of materials with their metadata
4. Uses this structured data to improve recognition accuracy

### How to Extend the System

To add new metadata fields:

1. Add the field to the appropriate TypeScript interface in metadata.ts
2. Update the getFieldGroups function in MaterialMetadataPanel.tsx
3. Add extraction patterns to the SQL migration file
4. Run the migration to update the database schema

metadata.ts

getFieldGroups

MaterialMetadataPanel.tsx

### Using Metadata in the Frontend

The metadata fields are automatically displayed in the admin panel using the MaterialMetadataPanel component. This component:

MaterialMetadataPanel

1. Receives a material type and metadata object
2. Renders appropriate fields based on the material type
3. Provides editing capabilities if not in read-only mode
4. Updates the parent component when metadata changes

Example usage:

<MaterialMetadataPanel  
 materialType="tile"  
 metadata={tileData.metadata}  
 onMetadataChange={handleMetadataChange}  
/>

<MaterialMetadataPanel  
 materialType="tile"  
 metadata={tileData.metadata}  
 onMetadataChange={handleMetadataChange}  
/>

### Visual Reference Library

The Visual Reference Library is a comprehensive database of images that illustrate different property values for materials. For example, it provides visual examples of different tile finishes (matte, glossy, etc.), R-ratings, material types, and more.

#### Accessing the Visual Reference Library

##### In the Admin Panel

The Visual Reference Library is integrated directly into the Material Metadata Panel in the admin interface. When editing material properties, you'll see a gallery icon next to dropdown fields that have visual references available.

To view or add visual references:  
1. Open a material for editing in the admin panel  
2. In the Material Metadata Panel, look for dropdown fields with a gallery icon  
3. Click the icon to open the Visual Reference Gallery  
4. View existing references or add new ones

##### Via the API

You can also access the Visual Reference Library programmatically via the API:

// Get visual references for a specific property value  
const references = await fetch('/api/property-references?propertyName=finish&propertyValue=matte&materialType=tile');  
  
// Add a new visual reference  
const formData = new FormData();  
formData.append('file', imageFile);  
formData.append('propertyName', 'finish');  
formData.append('propertyValue', 'matte');  
formData.append('materialType', 'tile');  
formData.append('description', 'Example of matte finish on porcelain tile');  
  
await fetch('/api/property-references', {  
 method: 'POST',  
 body: formData  
});

// Get visual references for a specific property value  
const references = await fetch('/api/property-references?propertyName=finish&propertyValue=matte&materialType=tile');  
  
// Add a new visual reference  
const formData = new FormData();  
formData.append('file', imageFile);  
formData.append('propertyName', 'finish');  
formData.append('propertyValue', 'matte');  
formData.append('materialType', 'tile');  
formData.append('description', 'Example of matte finish on porcelain tile');  
  
await fetch('/api/property-references', {  
 method: 'POST',  
 body: formData  
});

#### Adding Visual References

When adding visual references, follow these guidelines:

1. Image Quality: Use high-quality images that clearly show the property value
2. Isolation: Try to isolate the property being illustrated (e.g., for finish, show a close-up of the surface)
3. Variety: Add multiple examples of each property value to show variation
4. Description: Include a detailed description of what the image shows
5. Primary Reference: Mark one image as the primary reference for each property value

#### Best Practices for Different Properties

##### Finish

* Show close-ups of the surface to clearly illustrate the finish
* Include images under different lighting conditions
* For reflective finishes (glossy, polished), show reflection characteristics

##### R-Rating (Slip Resistance)

* Show the surface texture that provides the slip resistance
* Include images of the surface when wet if possible
* Add context about typical applications for each rating

##### Material Type

* Show the characteristic appearance of each material
* Include close-ups of texture and grain
* Show how the material responds to light

##### Shape/Format

* Show the entire tile to illustrate the shape
* Include a scale reference if possible
* Show installation patterns typical for the format

#### Visual Reference AI Integration

The Visual Reference Library not only serves as a visual database for property values but also provides powerful capabilities for:

1. AI Model Training: Train computer vision models to recognize different property values from images
2. OCR Enhancement: Improve OCR extraction accuracy by providing visual context for extracted text

These integrations enable more accurate property identification and extraction, improving the overall system's ability to understand and work with materials.

##### AI Model Training

The AI model training integration allows you to:

1. Create training datasets from the Visual Reference Library
2. Train computer vision models to recognize property values
3. Use the trained models for automated property identification

###### API Endpoints for AI Model Training

The following API endpoints are available for AI model training:

Create a Training Dataset

POST /api/ai/visual-reference/datasets

POST /api/ai/visual-reference/datasets

Request Body:

{  
 "propertyName": "finish",  
 "materialType": "tile"  
}

{  
 "propertyName": "finish",  
 "materialType": "tile"  
}

Train a Model

POST /api/ai/visual-reference/models

POST /api/ai/visual-reference/models

Request Body:

{  
 "datasetId": "550e8400-e29b-41d4-a716-446655440000",  
 "modelType": "classification",  
 "options": {  
 "epochs": 10,  
 "batchSize": 16,  
 "learningRate": 0.001  
 }  
}

{  
 "datasetId": "550e8400-e29b-41d4-a716-446655440000",  
 "modelType": "classification",  
 "options": {  
 "epochs": 10,  
 "batchSize": 16,  
 "learningRate": 0.001  
 }  
}

Create Dataset and Train Model in One Step

POST /api/ai/visual-reference/train

POST /api/ai/visual-reference/train

Request Body:

{  
 "propertyName": "finish",  
 "materialType": "tile",  
 "modelType": "classification",  
 "options": {  
 "epochs": 10,  
 "batchSize": 16,  
 "learningRate": 0.001  
 }  
}

{  
 "propertyName": "finish",  
 "materialType": "tile",  
 "modelType": "classification",  
 "options": {  
 "epochs": 10,  
 "batchSize": 16,  
 "learningRate": 0.001  
 }  
}

###### Usage Examples for AI Training

Training a Model for Finish Recognition

// Create a dataset and train a model for finish recognition  
const response = await fetch('/api/ai/visual-reference/train', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 propertyName: 'finish',  
 materialType: 'tile',  
 modelType: 'classification',  
 options: {  
 epochs: 20,  
 batchSize: 32  
 }  
 })  
});  
  
const result = await response.json();  
console.log(`Model ID: ${result.modelId}`);

// Create a dataset and train a model for finish recognition  
const response = await fetch('/api/ai/visual-reference/train', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 propertyName: 'finish',  
 materialType: 'tile',  
 modelType: 'classification',  
 options: {  
 epochs: 20,  
 batchSize: 32  
 }  
 })  
});  
  
const result = await response.json();  
console.log(`Model ID: ${result.modelId}`);

Training Models for Multiple Properties

// Properties to train models for  
const properties = ['finish', 'rRating', 'material', 'lookType'];  
  
// Train models for each property  
for (const propertyName of properties) {  
 await fetch('/api/ai/visual-reference/train', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 propertyName,  
 materialType: 'tile',  
 modelType: 'classification'  
 })  
 });  
  
 console.log(`Trained model for ${propertyName}`);  
}

// Properties to train models for  
const properties = ['finish', 'rRating', 'material', 'lookType'];  
  
// Train models for each property  
for (const propertyName of properties) {  
 await fetch('/api/ai/visual-reference/train', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 propertyName,  
 materialType: 'tile',  
 modelType: 'classification'  
 })  
 });  
  
 console.log(`Trained model for ${propertyName}`);  
}

###### Best Practices for AI Model Training

1. Sufficient Examples: Ensure you have at least 10-20 examples for each property value
2. Diverse Examples: Include examples with different lighting, angles, and contexts
3. Balanced Dataset: Try to have a similar number of examples for each property value
4. Regular Retraining: Retrain models as you add more visual references
5. Validation: Test models with new images to ensure they generalize well

##### OCR Enhancement

The OCR enhancement integration allows you to:

1. Verify OCR-extracted property values using visual references
2. Get alternative suggestions when extracted values don't match visual references
3. Generate extraction patterns based on visual references

###### API Endpoints for OCR Enhancement

The following API endpoints are available for OCR enhancement:

Enhance OCR Extraction

POST /api/ocr/visual-reference/enhance

POST /api/ocr/visual-reference/enhance

Request Body:

{  
 "propertyName": "finish",  
 "extractedValue": "matte",  
 "imageUrl": "https://example.com/image.jpg",  
 "materialType": "tile"  
}

{  
 "propertyName": "finish",  
 "extractedValue": "matte",  
 "imageUrl": "https://example.com/image.jpg",  
 "materialType": "tile"  
}

Enhance Multiple OCR Extractions

POST /api/ocr/visual-reference/enhance-multiple

POST /api/ocr/visual-reference/enhance-multiple

Request Body:

{  
 "extractedProperties": {  
 "finish": "matte",  
 "rRating": "R10",  
 "material": "porcelain"  
 },  
 "imageUrl": "https://example.com/image.jpg",  
 "materialType": "tile"  
}

{  
 "extractedProperties": {  
 "finish": "matte",  
 "rRating": "R10",  
 "material": "porcelain"  
 },  
 "imageUrl": "https://example.com/image.jpg",  
 "materialType": "tile"  
}

Get Extraction Patterns

GET /api/ocr/visual-reference/patterns/:propertyName/:materialType

GET /api/ocr/visual-reference/patterns/:propertyName/:materialType

###### Usage Examples for OCR Enhancement

Enhancing OCR Extraction

// Enhance OCR extraction for a finish value  
const response = await fetch('/api/ocr/visual-reference/enhance', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 propertyName: 'finish',  
 extractedValue: 'matte',  
 imageUrl: 'https://example.com/image.jpg',  
 materialType: 'tile'  
 })  
});  
  
const result = await response.json();  
  
if (result.success) {  
 const { extractedValue, confidence, visuallyVerified } = result.result;  
  
 if (visuallyVerified) {  
 console.log(`Verified: ${extractedValue} (${confidence.toFixed(2)})`);  
 } else {  
 console.log(`Not verified: ${extractedValue} (${confidence.toFixed(2)})`);  
  
 // Check for alternative suggestions  
 if (result.result.alternativeSuggestions) {  
 console.log('Suggestions:');  
 result.result.alternativeSuggestions.forEach(suggestion => {  
 console.log(`- ${suggestion.value} (${suggestion.confidence.toFixed(2)})`);  
 });  
 }  
 }  
}

// Enhance OCR extraction for a finish value  
const response = await fetch('/api/ocr/visual-reference/enhance', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 propertyName: 'finish',  
 extractedValue: 'matte',  
 imageUrl: 'https://example.com/image.jpg',  
 materialType: 'tile'  
 })  
});  
  
const result = await response.json();  
  
if (result.success) {  
 const { extractedValue, confidence, visuallyVerified } = result.result;  
  
 if (visuallyVerified) {  
 console.log(`Verified: ${extractedValue} (${confidence.toFixed(2)})`);  
 } else {  
 console.log(`Not verified: ${extractedValue} (${confidence.toFixed(2)})`);  
  
 // Check for alternative suggestions  
 if (result.result.alternativeSuggestions) {  
 console.log('Suggestions:');  
 result.result.alternativeSuggestions.forEach(suggestion => {  
 console.log(`- ${suggestion.value} (${suggestion.confidence.toFixed(2)})`);  
 });  
 }  
 }  
}

Enhancing Multiple OCR Extractions

// Enhance multiple OCR extractions  
const response = await fetch('/api/ocr/visual-reference/enhance-multiple', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 extractedProperties: {  
 finish: 'matte',  
 rRating: 'R10',  
 material: 'porcelain'  
 },  
 imageUrl: 'https://example.com/image.jpg',  
 materialType: 'tile'  
 })  
});  
  
const result = await response.json();  
  
if (result.success) {  
 const { results } = result;  
  
 // Process each property result  
 Object.entries(results).forEach(([propertyName, propertyResult]) => {  
 const { extractedValue, confidence, visuallyVerified } = propertyResult;  
  
 console.log(`${propertyName}: ${extractedValue} (${confidence.toFixed(2)}, ${visuallyVerified ? 'verified' : 'not verified'})`);  
 });  
}

// Enhance multiple OCR extractions  
const response = await fetch('/api/ocr/visual-reference/enhance-multiple', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 extractedProperties: {  
 finish: 'matte',  
 rRating: 'R10',  
 material: 'porcelain'  
 },  
 imageUrl: 'https://example.com/image.jpg',  
 materialType: 'tile'  
 })  
});  
  
const result = await response.json();  
  
if (result.success) {  
 const { results } = result;  
  
 // Process each property result  
 Object.entries(results).forEach(([propertyName, propertyResult]) => {  
 const { extractedValue, confidence, visuallyVerified } = propertyResult;  
  
 console.log(`${propertyName}: ${extractedValue} (${confidence.toFixed(2)}, ${visuallyVerified ? 'verified' : 'not verified'})`);  
 });  
}

Using Extraction Patterns

// Get extraction patterns for finish  
const response = await fetch('/api/ocr/visual-reference/patterns/finish/tile', {  
 headers: {  
 'Authorization': `Bearer ${token}`  
 }  
});  
  
const result = await response.json();  
  
if (result.success) {  
 const { patterns } = result.patterns;  
  
 // Use patterns for regex extraction  
 const text = 'The tile has a matte finish and R10 slip rating.';  
  
 for (const pattern of patterns) {  
 const regex = new RegExp(pattern, 'i');  
 const match = regex.exec(text);  
  
 if (match) {  
 console.log(`Matched pattern: ${pattern}`);  
 console.log(`Extracted value: ${match[1]}`);  
 break;  
 }  
 }  
}

// Get extraction patterns for finish  
const response = await fetch('/api/ocr/visual-reference/patterns/finish/tile', {  
 headers: {  
 'Authorization': `Bearer ${token}`  
 }  
});  
  
const result = await response.json();  
  
if (result.success) {  
 const { patterns } = result.patterns;  
  
 // Use patterns for regex extraction  
 const text = 'The tile has a matte finish and R10 slip rating.';  
  
 for (const pattern of patterns) {  
 const regex = new RegExp(pattern, 'i');  
 const match = regex.exec(text);  
  
 if (match) {  
 console.log(`Matched pattern: ${pattern}`);  
 console.log(`Extracted value: ${match[1]}`);  
 break;  
 }  
 }  
}

###### Best Practices for OCR Enhancement

1. Comprehensive References: Ensure you have visual references for all common property values
2. High-Quality Images: Use clear, well-lit images for both references and extraction
3. Confidence Thresholds: Set appropriate confidence thresholds based on your needs
4. Fallback Strategies: Have fallback strategies when visual verification fails
5. Feedback Loop: Use verification results to improve OCR extraction over time

##### Integration with Existing Systems

###### Integration with OCR Pipeline

The Visual Reference OCR enhancement can be integrated into your existing OCR pipeline:

// Example OCR pipeline with visual reference enhancement  
async function processDocument(documentUrl) {  
 // 1. Extract text from document using OCR  
 const extractedText = await performOcr(documentUrl);  
  
 // 2. Extract properties from text using regex patterns  
 const extractedProperties = extractPropertiesFromText(extractedText);  
  
 // 3. Extract image from document  
 const imageUrl = await extractImageFromDocument(documentUrl);  
  
 // 4. Enhance OCR extraction with visual reference verification  
 const enhancedProperties = await enhanceWithVisualReferences(  
 extractedProperties,  
 imageUrl,  
 'tile'  
 );  
  
 // 5. Return the enhanced properties  
 return enhancedProperties;  
}  
  
// Helper function to enhance properties with visual references  
async function enhanceWithVisualReferences(properties, imageUrl, materialType) {  
 const response = await fetch('/api/ocr/visual-reference/enhance-multiple', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 extractedProperties: properties,  
 imageUrl,  
 materialType  
 })  
 });  
  
 const result = await response.json();  
  
 if (result.success) {  
 return result.results;  
 }  
  
 return properties; // Fallback to original properties if enhancement fails  
}

// Example OCR pipeline with visual reference enhancement  
async function processDocument(documentUrl) {  
 // 1. Extract text from document using OCR  
 const extractedText = await performOcr(documentUrl);  
  
 // 2. Extract properties from text using regex patterns  
 const extractedProperties = extractPropertiesFromText(extractedText);  
  
 // 3. Extract image from document  
 const imageUrl = await extractImageFromDocument(documentUrl);  
  
 // 4. Enhance OCR extraction with visual reference verification  
 const enhancedProperties = await enhanceWithVisualReferences(  
 extractedProperties,  
 imageUrl,  
 'tile'  
 );  
  
 // 5. Return the enhanced properties  
 return enhancedProperties;  
}  
  
// Helper function to enhance properties with visual references  
async function enhanceWithVisualReferences(properties, imageUrl, materialType) {  
 const response = await fetch('/api/ocr/visual-reference/enhance-multiple', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 extractedProperties: properties,  
 imageUrl,  
 materialType  
 })  
 });  
  
 const result = await response.json();  
  
 if (result.success) {  
 return result.results;  
 }  
  
 return properties; // Fallback to original properties if enhancement fails  
}

###### Integration with Computer Vision Pipeline

The Visual Reference AI training can be integrated into your computer vision pipeline:

// Example computer vision pipeline with visual reference training  
async function setupPropertyRecognition() {  
 // 1. Define properties to recognize  
 const properties = ['finish', 'material', 'lookType'];  
  
 // 2. Train models for each property  
 const modelIds = await trainModelsForProperties(properties);  
  
 // 3. Set up recognition pipeline  
 setupRecognitionPipeline(modelIds);  
}  
  
// Helper function to train models for properties  
async function trainModelsForProperties(properties) {  
 const modelIds = {};  
  
 for (const propertyName of properties) {  
 const response = await fetch('/api/ai/visual-reference/train', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 propertyName,  
 materialType: 'tile',  
 modelType: 'classification'  
 })  
 });  
  
 const result = await response.json();  
  
 if (result.success) {  
 modelIds[propertyName] = result.modelId;  
 }  
 }  
  
 return modelIds;  
}

// Example computer vision pipeline with visual reference training  
async function setupPropertyRecognition() {  
 // 1. Define properties to recognize  
 const properties = ['finish', 'material', 'lookType'];  
  
 // 2. Train models for each property  
 const modelIds = await trainModelsForProperties(properties);  
  
 // 3. Set up recognition pipeline  
 setupRecognitionPipeline(modelIds);  
}  
  
// Helper function to train models for properties  
async function trainModelsForProperties(properties) {  
 const modelIds = {};  
  
 for (const propertyName of properties) {  
 const response = await fetch('/api/ai/visual-reference/train', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 propertyName,  
 materialType: 'tile',  
 modelType: 'classification'  
 })  
 });  
  
 const result = await response.json();  
  
 if (result.success) {  
 modelIds[propertyName] = result.modelId;  
 }  
 }  
  
 return modelIds;  
}

#### Maintaining the Visual Reference Library

To keep the Visual Reference Library valuable and up-to-date:

1. Regular Review: Periodically review existing references for quality and relevance
2. Expand Coverage: Add references for new property values as they are introduced
3. User Feedback: Incorporate feedback from users about the usefulness of references
4. Quality Control: Remove or replace low-quality or misleading images
5. Consistency: Ensure consistent representation across different properties

### Field Descriptions for AI and API Integration

The system includes detailed descriptions for each metadata field that explain what the field means, its importance, and how it should be used. These descriptions serve multiple purposes:

1. User Guidance: Help users understand what each field means in the admin panel
2. API Documentation: Provide context for developers using the API
3. OCR AI Training: Assist in training AI models to extract these properties from documents
4. Agent Context: Give AI agents the necessary context to understand and work with material data

#### Using Descriptions for OCR AI

When training OCR AI models to extract material properties from documents, catalogs, or technical sheets, these descriptions provide valuable context:

1. Field Identification: The descriptions help the AI understand what to look for in the document. For example, when looking for "R-Rating", the AI knows to search for slip resistance ratings following the R9-R13 pattern.
2. Value Validation: The descriptions often include information about valid values or formats, which helps the AI validate extracted data. For example, knowing that V-Rating ranges from V1 to V4 helps the AI correctly interpret and normalize extracted values.
3. Contextual Clues: The descriptions provide contextual information that helps the AI identify fields even when they're not explicitly labeled. For example, if a document mentions "frost resistance" without using that exact term, the AI can use the description to recognize related terms like "freeze-thaw resistance" or "suitable for outdoor use in cold climates".

Field Identification: The descriptions help the AI understand what to look for in the document. For example, when looking for "R-Rating", the AI knows to search for slip resistance ratings following the R9-R13 pattern.

Value Validation: The descriptions often include information about valid values or formats, which helps the AI validate extracted data. For example, knowing that V-Rating ranges from V1 to V4 helps the AI correctly interpret and normalize extracted values.

Contextual Clues: The descriptions provide contextual information that helps the AI identify fields even when they're not explicitly labeled. For example, if a document mentions "frost resistance" without using that exact term, the AI can use the description to recognize related terms like "freeze-thaw resistance" or "suitable for outdoor use in cold climates".

#### Using Descriptions for API Integration

The field descriptions are valuable for API documentation and integration:

1. API Documentation: The descriptions can be used to generate comprehensive API documentation that explains each field in detail.
2. Request Validation: The descriptions provide context for validating API requests, ensuring that clients provide valid values for each field.
3. Response Enrichment: API responses can include field descriptions to help clients understand the data they're receiving.

API Documentation: The descriptions can be used to generate comprehensive API documentation that explains each field in detail.

Request Validation: The descriptions provide context for validating API requests, ensuring that clients provide valid values for each field.

Response Enrichment: API responses can include field descriptions to help clients understand the data they're receiving.

#### Using Descriptions for AI Agents

AI agents can use the field descriptions to better understand and work with material data:

1. Context Understanding: The descriptions provide the necessary context for agents to understand what each field means and how it relates to other fields.
2. User Assistance: Agents can use the descriptions to explain fields to users in natural language.
3. Data Extraction: When extracting information from unstructured text, agents can use the descriptions to identify relevant information.

Context Understanding: The descriptions provide the necessary context for agents to understand what each field means and how it relates to other fields.

User Assistance: Agents can use the descriptions to explain fields to users in natural language.

Data Extraction: When extracting information from unstructured text, agents can use the descriptions to identify relevant information.

### Conclusion

The material metadata system provides a comprehensive solution for handling different material types and their unique properties. By using a combination of database schema, TypeScript interfaces, and React components, it offers a consistent and type-safe way to work with material metadata throughout the application.

The automatic extraction capabilities make it particularly powerful for processing large volumes of material data from external sources, significantly improving the efficiency of the data import process and helping to build more accurate AI models.

The Visual Reference Library enhances this system by providing visual examples that improve understanding, training, and identification capabilities. The AI integration capabilities further extend the system's power by enabling:

1. Automated property recognition through trained AI models
2. Enhanced OCR extraction with visual verification
3. Improved accuracy through multi-modal analysis (combining text and image data)
4. Seamless integration with existing OCR and computer vision pipelines

Together, these components create a robust foundation for working with material metadata across the entire platform, enabling more accurate and efficient material recognition, classification, and data extraction.

# Shared Material Metadata Fields

Source: readme/shared-material-metadata-fields.md

---

## Material Metadata Fields

This document defines metadata fields for different material categories in the system. These fields will be used in the admin panel for displaying, filtering and sorting materials, as well as providing extraction hints for AI processing when importing materials from PDFs or websites.

### Global Fields

These fields apply to all material types:

Table content:

Field Name | Display Name | Type | Required | Description | Extraction Hints

manufacturer | Manufacturer | text | yes | Company that produces the material | Look for company logo, "manufactured by", or prominent branding

collection | Collection | text | no | Product collection or series name | Near "collection", "series", or as a prominent subtitle

productCode | Product Code | text | no | Manufacturer's product code/reference | Pattern "Ref:", "Code:", "Art. Nr.", alphanumeric code

year | Year | number | no | Year of production/release | Four-digit year, often near copyright or catalog information

countryOfOrigin | Country of Origin | text | no | Manufacturing country | "Made in", "Produced in", "Origin:"

warranty | Warranty | text | no | Warranty information | Near "warranty", "guarantee", often as "X years"

certifications | Certifications | text | no | Product certifications | Look for certification logos, "Certified by", certification codes

applicationArea | Application Area | dropdown | no | Where the material can be used | "Suitable for", "Application:", "Recommended use:"

price | Price Range | dropdown | no | Price category | Look for price indicators, "$", "€", "price category"

sustainability | Sustainability | dropdown | no | Environmental friendliness rating | "Eco-rating", "Environmental impact", "Green score"

### Tile-specific Fields

Table content:

Field Name | Display Name | Type | Required | Description | Extraction Hints | Validation/Options

vRating | V-Rating | dropdown | no | Version/variation of patterns | "V Rating:", "Variation:", "V2", "V3", "V4" | Options: V1, V2, V3, V4

rRating | R-Rating | dropdown | no | Slip resistance rating (Ramp Test) | "R Rating:", "Slip resistance:", "R9", "R10", "R11", "R12", "R13" | Options: R9, R10, R11, R12, R13

size | Size | text | yes | Dimensions in cm or mm | "Format:", "Size:", "Dimensions:", patterns like "60x60", "30x60" | Regex: ^\d+x\d+(.\d+)?$

thickness | Thickness | number | yes | Material thickness in mm | "Thickness:", "Height:", often followed by "mm" | Min: 3, Max: 30, Unit: mm

waterAbsorption | Water Absorption | dropdown | no | Water absorption class | "Water absorption:", "Absorption class:", "E ≤ 0.5%" | Options: BIa (≤0.5%), BIb (0.5-3%), BIIa (3-6%), BIIb (6-10%), BIII (>10%)

frostResistance | Frost Resistance | boolean | no | Whether the tile is frost resistant | "Frost resistant:", "Suitable for outdoors", "Frost proof" |

peiRating | PEI Rating | dropdown | no | Surface abrasion resistance | "PEI:", "Abrasion class:", "PEI II", "Class 3" | Options: PEI I, PEI II, PEI III, PEI IV, PEI V

moh | Mohs Hardness | dropdown | no | Surface hardness | "Mohs:", "Hardness:", "Mohs scale" | Options: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

material | Material | dropdown | yes | Type of tile material | "Material:", "Type:" | Options: Ceramic, Porcelain, Marble, Granite, Terracotta, Quartzite, Limestone, Slate, Glass, Cement

finish | Finish | dropdown | yes | Surface finish | "Finish:", "Surface:" | Options: Matte, Glossy, Polished, Honed, Textured, Lappato, Semi-polished, Natural, Structured, Satin

color | Color | dropdown | yes | Predominant color | Look for color descriptions, often prominently displayed | Options: White, Black, Gray, Beige, Brown, Red, Blue, Green, Yellow, Multicolor, etc.

usage | Usage Area | dropdown | yes | Recommended installation areas | "Application:", "Suitable for:", "Usage:" | Options: Floor, Wall, Floor & Wall, Outdoor, Indoor, Bathroom, Kitchen, Living Room, Commercial

rectified | Rectified | boolean | no | Whether the tile edges are precisely cut | "Rectified:", "Calibrated edges:" |

antibacterial | Antibacterial | boolean | no | Has antibacterial properties | "Antibacterial:", "Hygienic properties:" |

### Wood-specific Fields

Table content:

Field Name | Display Name | Type | Required | Description | Extraction Hints | Validation/Options

woodType | Wood Type | dropdown | yes | Type of wood | "Species:", "Wood type:", "Material:" | Options: Oak, Maple, Cherry, Walnut, Pine, Birch, Ash, Mahogany, Teak, Bamboo, etc.

grade | Grade | dropdown | no | Quality/appearance grade | "Grade:", "Quality:", "Class:" | Options: Prime, Select, Natural, Rustic, Character

construction | Construction | dropdown | yes | How the wood is constructed | "Construction:", "Structure:" | Options: Solid, Engineered, Laminate, Veneer

thickness | Thickness | number | yes | Material thickness in mm | "Thickness:", "Height:" | Min: 7, Max: 22, Unit: mm

width | Width | number | yes | Plank width in mm | "Width:", "Plank width:" | Min: 80, Max: 300, Unit: mm

length | Length | number | yes | Plank length in mm | "Length:", "Plank length:" | Min: 300, Max: 2500, Unit: mm

finish | Finish | dropdown | yes | Surface treatment | "Finish:", "Surface treatment:" | Options: Oiled, Lacquered, Waxed, Brushed, Untreated, Smoked, Distressed

color | Color | dropdown | yes | Predominant color/tone | Look for color descriptions | Options: Light, Medium, Dark, White, Golden, Brown, Gray, Black, etc.

hardness | Hardness (Janka) | number | no | Janka hardness rating | "Janka hardness:", "Hardness rating:" | Min: 300, Max: 4000

installationSystem | Installation System | dropdown | no | How planks connect | "Installation:", "Fitting system:" | Options: Tongue & Groove, Click System, Glue-Down, Floating, Nail-Down

moisture | Moisture Content | number | no | Wood moisture percentage | "Moisture content:", "Humidity:" | Min: 5, Max: 12, Unit: %

stability | Dimensional Stability | dropdown | no | How stable under humidity changes | "Stability:", "Dimensional changes:" | Options: Low, Medium, High

underfloorHeating | Suitable for Underfloor Heating | boolean | no | Compatible with heated floors | "Underfloor heating:", "UFH compatible:" |

### Lighting-specific Fields

Table content:

Field Name | Display Name | Type | Required | Description | Extraction Hints | Validation/Options

lightingType | Lighting Type | dropdown | yes | Type of lighting fixture | "Type:", "Product type:" | Options: Pendant, Chandelier, Wall Sconce, Table Lamp, Floor Lamp, Ceiling Light, Track Light, Recessed Light, LED Strip

bulbType | Bulb Type | dropdown | yes | Type of bulb required | "Bulb:", "Light source:", "Lamp type:" | Options: LED, Incandescent, Halogen, Fluorescent, CFL, Smart Bulb

bulbIncluded | Bulb Included | boolean | no | Whether bulbs come with the fixture | "Bulbs included:", "Includes bulb:" |

wattage | Wattage | number | yes | Power consumption in watts | "Wattage:", "Power:", "...W" | Min: 1, Max: 1000, Unit: W

lumens | Lumens | number | no | Brightness output | "Lumens:", "Light output:", "Brightness:" | Min: 10, Max: 50000, Unit: lm

colorTemperature | Color Temperature | dropdown | no | Light warmth/coolness | "Color temperature:", "Kelvin:", "Warm white" | Options: Warm White (2700K-3000K), Neutral White (3500K-4100K), Cool White (5000K-6500K)

cri | CRI | number | no | Color Rendering Index | "CRI:", "Color rendering index:" | Min: 70, Max: 100

dimmable | Dimmable | boolean | no | Whether light can be dimmed | "Dimmable:", "Dimming:" |

ipRating | IP Rating | dropdown | no | Dust/water resistance | "IP rating:", "Protection class:" | Options: IP20, IP44, IP54, IP65, IP67

material | Material | dropdown | yes | Primary material of fixture | "Material:", "Made of:" | Options: Metal, Glass, Plastic, Wood, Fabric, Ceramic, Concrete, Crystal

dimensions | Dimensions | text | yes | Size of the fixture | "Dimensions:", "Size:", "Measurements:" | Regex: ^(\d+(.\d+)? x \d+(.\d+)? x \d+(.\d+)?)$

weight | Weight | number | no | Weight of the fixture in kg | "Weight:", "...kg" | Min: 0.1, Max: 100, Unit: kg

voltage | Voltage | number | yes | Operating voltage | "Voltage:", "...V" | Min: 12, Max: 240, Unit: V

energyClass | Energy Efficiency Class | dropdown | no | Energy efficiency rating | "Energy class:", "Energy rating:" | Options: A+++, A++, A+, A, B, C, D, E, F, G

controlSystem | Control System | dropdown | no | How the light is controlled | "Control:", "Operation:" | Options: Switch, Remote, Smart App, Voice, Motion Sensor, Touch

### Furniture-specific Fields

Table content:

Field Name | Display Name | Type | Required | Description | Extraction Hints | Validation/Options

furnitureType | Furniture Type | dropdown | yes | Category of furniture | "Type:", "Category:" | Options: Chair, Table, Sofa, Bed, Shelf, Cabinet, Desk, Stool, Armchair, Dresser, Wardrobe, Bookcase, Ottoman

style | Style | dropdown | yes | Design style | "Style:", "Design:" | Options: Modern, Scandinavian, Industrial, Traditional, Mid-Century, Rustic, Minimalist, Contemporary, Bohemian, Art Deco

material | Primary Material | dropdown | yes | Main material | "Material:", "Made of:" | Options: Wood, Metal, Glass, Plastic, Fabric, Leather, Rattan, Stone, Marble, Composite

color | Color | dropdown | yes | Predominant color | Look for color descriptions | Options: White, Black, Gray, Beige, Brown, Blue, Green, Red, Yellow, Orange, Multicolor

dimensions | Dimensions | text | yes | Size (Width x Depth x Height) | "Dimensions:", "Measurements:", "Size:" | Regex: ^(\d+(.\d+)? x \d+(.\d+)? x \d+(.\d+)?)$

weight | Weight | number | no | Weight in kg | "Weight:", "...kg" | Min: 0.5, Max: 500, Unit: kg

weightCapacity | Weight Capacity | number | no | Maximum load capacity | "Weight capacity:", "Max load:", "Supports up to:" | Min: 1, Max: 1000, Unit: kg

assembly | Assembly Required | boolean | no | Whether product needs assembly | "Assembly required:", "Self-assembly:" |

cushionFilling | Cushion Filling | dropdown | no | Type of filling in cushions | "Filling:", "Cushion material:" | Options: Foam, Memory Foam, Down, Polyester, Feather, Spring

upholstery | Upholstery Material | dropdown | no | Fabric covering type | "Upholstery:", "Cover material:" | Options: Cotton, Linen, Polyester, Velvet, Leather, Faux Leather, Wool, Microfiber

frameConstruction | Frame Construction | dropdown | no | Frame material/construction | "Frame:", "Structure:" | Options: Solid Wood, Plywood, MDF, Metal, Particle Board

adjustable | Adjustable | boolean | no | Has adjustable features | "Adjustable:", "Configurable:" |

outdoor | Suitable for Outdoor | boolean | no | Can be used outdoors | "Outdoor:", "Weather resistant:", "Indoor/Outdoor:" |

sustainability | Sustainability | dropdown | no | Environmental certification | "Sustainability:", "Eco-friendly:", "Environmental:" | Options: FSC Certified, Recycled Materials, Low-VOC, GREENGUARD, None

features | Special Features | text | no | Additional notable features | "Features:", "Special:", "Also includes:" |

### Decoration-specific Fields

Table content:

Field Name | Display Name | Type | Required | Description | Extraction Hints | Validation/Options

decorationType | Decoration Type | dropdown | yes | Type of decorative item | "Type:", "Category:" | Options: Wall Art, Vase, Sculpture, Mirror, Candle Holder, Rug, Cushion, Throw, Clock, Bookend, Plant Pot, Figurine

style | Style | dropdown | yes | Design style | "Style:", "Design:" | Options: Modern, Scandinavian, Industrial, Traditional, Mid-Century, Rustic, Minimalist, Contemporary, Bohemian, Art Deco

material | Material | dropdown | yes | Primary material | "Material:", "Made of:" | Options: Ceramic, Glass, Metal, Wood, Textile, Paper, Plastic, Stone, Resin, Concrete

color | Color | dropdown | yes | Predominant color | Look for color descriptions | Options: White, Black, Gray, Beige, Brown, Blue, Green, Red, Yellow, Gold, Silver, Multicolor

dimensions | Dimensions | text | yes | Size in cm | "Dimensions:", "Size:", "Measurements:" |

theme | Theme | dropdown | no | Thematic design element | "Theme:", "Inspiration:" | Options: Geometric, Floral, Abstract, Nature, Animal, Architectural, Seasonal, Coastal, Ethnic, Typography

technique | Technique | dropdown | no | Production technique | "Technique:", "Process:", "Handmade:" | Options: Handmade, Machine-made, Hand-painted, Printed, Carved, Woven, Cast, Blown, Embroidered

occasion | Occasion | dropdown | no | If specific to an occasion | "Occasion:", "Perfect for:" | Options: Everyday, Holiday, Christmas, Halloween, Wedding, Birthday, Anniversary, Housewarming

setSize | Set Size | number | no | Number of pieces in a set | "Set of:", "Pieces:", "Quantity:" | Min: 1, Max: 100

careInstructions | Care Instructions | text | no | How to clean/maintain | "Care:", "Cleaning:", "Maintenance:" |

indoor | Indoor/Outdoor | dropdown | no | Where it can be used | "Indoor/Outdoor:", "Suitable for:" | Options: Indoor Only, Outdoor Only, Indoor/Outdoor

mountingType | Mounting Type | dropdown | no | How to install/display | "Mounting:", "Installation:", "Hanging:" | Options: Wall Mounted, Tabletop, Freestanding, Hanging, Floor Standing

fragile | Fragile | boolean | no | Whether item is delicate | "Fragile:", "Delicate:", "Handle with care:" |

sustainability | Eco-Friendly | dropdown | no | Environmental aspects | "Eco-friendly:", "Sustainable:", "Recycled:" | Options: Recycled Materials, Biodegradable, Sustainable Source, Fair Trade, Handcrafted, None

### Extraction Strategy

When extracting metadata from PDFs or websites, the system will use:

1. Field Names: Look for exact field names or display names in the text
2. Extraction Hints: Use provided hints to locate information in context
3. Pattern Matching: Use regular expressions to extract structured information like dimensions
4. Visual Context: For images, look for text near product images
5. Common Formats: Recognize standard industry formatting (e.g., "60x60" for tile sizes)

### Integration with Admin Panel

In the admin panel, these fields should be:

1. Searchable: Allow filtering and searching by field values
2. Sortable: Allow sorting by numeric fields (e.g., thickness, weight)
3. Groupable: Allow grouping by categorical fields (e.g., material type, color)
4. Editable: Allow admins to edit field values with appropriate validation
5. Exportable: Include in data exports

### Implementation Notes

When implementing these fields:

1. Create appropriate validation rules for each field type
2. Define extraction patterns for automatic import
3. Set up appropriate indexes for search performance
4. Ensure field descriptions are accessible in the UI for clarity
5. Consider localization needs for international deployments

# Subscription Management System

Source: readme/subscription-management-system.md

---

## Subscription Management System

### Executive Summary

The Subscription Management System provides a comprehensive framework for managing user subscriptions, payment processing, credit management, and access control. It integrates with Stripe for payment processing and offers a flexible, module-based access control system with tiered pricing and granular feature access.

Key capabilities:  
- Multiple subscription tiers with different feature sets and resource limits  
- Stripe integration for payment processing and subscription management  
- Credit system for purchasing and using credits for premium features  
- Module-based access control for granular permission management  
- API rate limiting with tier-specific quotas  
- Usage tracking and analytics  
- Subscription state machine for lifecycle management  
- Plan versioning for managing changes to subscription tiers

### System Architecture

The system follows a layered architecture that integrates with existing authentication:

1. Authentication Layer: Validates user identity via Supabase Auth and determines user type
2. User Type Layer: Categorizes users as regular users, factories, b2b, or admin
3. Subscription Layer: Verifies subscription status and tier permissions based on user type
4. Payment Processing Layer: Handles payments and subscription billing via Stripe
5. Credit Management Layer: Manages user credits for premium features
6. Access Control Layer: Enforces module-specific permissions
7. Rate Limiting Layer: Controls API usage based on tier limits
8. Analytics Layer: Tracks usage patterns and subscription metrics

### Core Components

#### 1. Subscription Tiers

Subscription tiers define the available plans users can subscribe to. Each tier specifies:

* Basic Information: Name, price, description, currency, and visibility (public/private)
* Module Access: Which platform modules are enabled for this tier
* API Limits: Requests per minute, day, and month
* Storage Limits: Maximum storage space, file size, and files per project
* Credit Limits: Included credits, maximum purchasable credits, and credit price multiplier
* Resource Limits: Maximum projects, team members, and moodboards

Tiers are stored in the subscription\_tiers table in Supabase and offer different capability levels:

subscription\_tiers

Table content:

Tier | Description | Key Features

Free | Basic access with limited features | Basic material recognition, limited storage, no API access

Basic | Standard access for individual users | Material recognition, knowledge base, basic agent access

Professional | Advanced access for professionals | All basic features + advanced agents, 3D designer access

Enterprise | Full access for organizations | Unlimited access to all features, maximum API limits

Custom | Tailored solutions for specific needs | Custom configuration of all modules and limits

#### 2. User Types

The system supports different types of users, each with access to specific subscription tiers:

* User: Regular end users of the application
* Factory: Factory/manufacturer users with specialized access
* B2B: Business-to-business users with specialized access
* Admin: Administrators with full access to the system

Special cases:  
- The email basiliskan@gmail.com is automatically assigned the admin user type when registering  
- Admins can convert regular users to factory or b2b users through the admin panel  
- Each user type has access to different subscription tiers tailored to their needs

basiliskan@gmail.com

admin

#### 3. User Subscriptions

User subscriptions link users to their selected subscription tier and track usage metrics:

* Subscription Information: Tier ID, status (active, trialing, past\_due, canceled, etc.), renewal date
* Stripe Integration: Customer ID, subscription ID, payment method ID
* Billing Details: Payment method, billing cycle, current period start/end
* Usage Tracking: API requests count, storage usage, module-specific usage
* User Type: The type of user (user, factory, b2b, admin) which determines available subscription tiers

The subscription state machine manages the lifecycle of subscriptions with the following states:  
- Active: Subscription is active and paid  
- Trialing: Subscription is in trial period  
- Past Due: Payment has failed but subscription is still active  
- Canceled: Subscription has been canceled  
- Incomplete: Subscription creation is incomplete  
- Paused: Subscription is temporarily paused

#### 4. Credit System

The credit system allows users to purchase and use credits for premium features:

* Credit Balance: Current credit balance for each user
* Credit Transactions: History of credit additions and usage
* Credit Pricing: Tier-specific credit pricing with potential discounts
* Credit Usage: Using credits for specific actions like generating 3D models or running AI agents

Credits can be:  
- Included with a subscription  
- Purchased separately  
- Used for various actions (e.g., generating 3D models, running agents)

#### 5. Stripe Integration

The system integrates with Stripe for payment processing:

* Customer Management: Creating and managing Stripe customers
* Subscription Management: Creating, updating, and canceling subscriptions
* Payment Method Management: Adding, updating, and removing payment methods
* Webhook Handling: Processing Stripe events for subscription lifecycle management
* Invoice Management: Generating and managing invoices

#### 6. Module-Based Access Control

Access to specific platform functionalities is controlled at the module level:

"moduleAccess": [  
 { "name": "materialRecognition", "enabled": true },  
 { "name": "knowledgeBase", "enabled": true },  
 { "name": "agents", "enabled": true },  
 { "name": "advancedAgents", "enabled": false },  
 { "name": "3dDesigner", "enabled": false },  
 { "name": "api", "enabled": false },  
 { "name": "pdf", "enabled": true },  
 { "name": "crawler", "enabled": false },  
 { "name": "mlTraining", "enabled": false },  
 { "name": "materialPromotion", "enabled": false }  
]

"moduleAccess": [  
 { "name": "materialRecognition", "enabled": true },  
 { "name": "knowledgeBase", "enabled": true },  
 { "name": "agents", "enabled": true },  
 { "name": "advancedAgents", "enabled": false },  
 { "name": "3dDesigner", "enabled": false },  
 { "name": "api", "enabled": false },  
 { "name": "pdf", "enabled": true },  
 { "name": "crawler", "enabled": false },  
 { "name": "mlTraining", "enabled": false },  
 { "name": "materialPromotion", "enabled": false }  
]

This granular approach allows for flexible tier configuration and precise access control.

#### 7. Plan Versioning

The plan versioning system allows for the creation and management of different versions of subscription plans:

* Version Number: Sequential identifier for the version
* Changes: The specific changes made in this version
* Effective Date: When the version becomes active
* Created By: Who created the version
* Created At: When the version was created

This enables tracking changes to subscription tiers and applying them at specific times.

#### 8. Analytics and Reporting

The analytics system provides insights into subscription metrics, revenue, user behavior, and resource utilization:

* Revenue Metrics: MRR, ARR, average revenue per user
* Subscription Metrics: Total subscribers, active subscribers, churn rate, conversion rate
* Usage Metrics: API usage, storage utilization, feature adoption
* Credit Usage: Credit consumption by feature, purchase patterns, usage trends

### Backend Implementation

#### Models

1. SubscriptionTier Model (packages/server/src/models/subscriptionTier.model.ts)
2. Core data structure for subscription tiers
3. CRUD operations for tier management
4. Includes price, features, module access permissions, and resource limits
5. UserSubscription Model (packages/server/src/models/userSubscription.model.ts)
6. Links users to subscription tiers
7. Tracks API usage and module access
8. Handles resets for usage limits
9. Provides utility functions for checking permissions and limits
10. Integrates with Stripe for payment processing
11. UserCredit Model (packages/server/src/models/userCredit.model.ts)
12. Manages user credit balances
13. Tracks credit transactions
14. Provides functions for adding and using credits
15. SubscriptionTierVersion Model (packages/server/src/models/subscriptionTierVersion.model.ts)
16. Manages versions of subscription tiers
17. Tracks changes between versions
18. Provides functions for applying versions
19. SubscriptionStateMachine Model (packages/server/src/models/subscriptionStateMachine.model.ts)
20. Manages subscription state transitions
21. Enforces valid state transitions
22. Records state transition history

packages/server/src/models/subscriptionTier.model.ts

Includes price, features, module access permissions, and resource limits

UserSubscription Model (packages/server/src/models/userSubscription.model.ts)

packages/server/src/models/userSubscription.model.ts

Integrates with Stripe for payment processing

UserCredit Model (packages/server/src/models/userCredit.model.ts)

packages/server/src/models/userCredit.model.ts

Provides functions for adding and using credits

SubscriptionTierVersion Model (packages/server/src/models/subscriptionTierVersion.model.ts)

packages/server/src/models/subscriptionTierVersion.model.ts

Provides functions for applying versions

SubscriptionStateMachine Model (packages/server/src/models/subscriptionStateMachine.model.ts)

packages/server/src/models/subscriptionStateMachine.model.ts

#### Services

1. Stripe Service (packages/server/src/services/payment/stripeService.ts)
2. Handles all interactions with the Stripe API
3. Manages customers, subscriptions, and payment methods
4. Processes webhooks for Stripe events
5. Subscription Analytics Service (packages/server/src/services/analytics/subscriptionAnalytics.service.ts)
6. Provides analytics for subscriptions
7. Calculates revenue metrics, churn rate, and conversion rate
8. Analyzes subscription distribution and credit usage

packages/server/src/services/payment/stripeService.ts

Processes webhooks for Stripe events

Subscription Analytics Service (packages/server/src/services/analytics/subscriptionAnalytics.service.ts)

packages/server/src/services/analytics/subscriptionAnalytics.service.ts

#### Controllers

1. Subscription Controller (packages/server/src/controllers/subscription.controller.ts)
2. Handles user-facing subscription management
3. Manages credit purchases and usage
4. Processes subscription changes
5. Webhook Controller (packages/server/src/controllers/webhook.controller.ts)
6. Processes Stripe webhook events
7. Updates subscription status based on payment events
8. Handles credit purchases and subscription changes
9. Admin Subscription Controller (packages/server/src/controllers/admin/subscription.admin.controller.ts)
10. Manages subscription tiers and versions
11. Handles user subscription management
12. Provides subscription analytics

packages/server/src/controllers/subscription.controller.ts

Processes subscription changes

Webhook Controller (packages/server/src/controllers/webhook.controller.ts)

packages/server/src/controllers/webhook.controller.ts

Handles credit purchases and subscription changes

Admin Subscription Controller (packages/server/src/controllers/admin/subscription.admin.controller.ts)

packages/server/src/controllers/admin/subscription.admin.controller.ts

#### Middleware

1. Module Access Middleware (packages/server/src/middleware/module-access.middleware.ts)
2. Verifies module access permissions based on subscription tier
3. Protects routes by checking subscription status
4. Integrates with existing auth middleware
5. Rate Limiting Middleware (packages/server/src/middleware/rate-limit.middleware.ts)
6. Enforces API request limits based on subscription tier
7. Provides per-user rate limiting with tier-specific quotas
8. Tracks and limits concurrent API usage

packages/server/src/middleware/module-access.middleware.ts

Integrates with existing auth middleware

Rate Limiting Middleware (packages/server/src/middleware/rate-limit.middleware.ts)

packages/server/src/middleware/rate-limit.middleware.ts

#### API Routes

1. Subscription Routes (packages/server/src/routes/subscription.routes.ts)
2. Public endpoints for retrieving available tiers
3. Protected endpoints for managing user subscriptions
4. Credit management endpoints
5. Payment method management endpoints
6. Webhook Routes (packages/server/src/routes/webhook.routes.ts)
7. Endpoints for processing Stripe webhook events
8. Admin Subscription Routes (packages/server/src/routes/admin/subscription.admin.routes.ts)
9. Admin endpoints for managing subscription tiers
10. Analytics endpoints for subscription metrics
11. User subscription management endpoints

packages/server/src/routes/subscription.routes.ts

Payment method management endpoints

Webhook Routes (packages/server/src/routes/webhook.routes.ts)

packages/server/src/routes/webhook.routes.ts

Endpoints for processing Stripe webhook events

Admin Subscription Routes (packages/server/src/routes/admin/subscription.admin.routes.ts)

packages/server/src/routes/admin/subscription.admin.routes.ts

### Database Schema

The subscription system uses the following database tables:

1. subscription\_tiers
2. id: UUID primary key
3. name: String
4. description: String
5. price: Decimal
6. currency: String
7. billing\_interval: String (monthly, yearly, one-time)
8. stripe\_price\_id: String
9. stripe\_product\_id: String
10. module\_access: JSONB array
11. api\_limits: JSONB object
12. storage\_limits: JSONB object
13. credit\_limits: JSONB object
14. max\_projects: Integer
15. max\_team\_members: Integer
16. max\_moodboards: Integer
17. support\_level: String
18. is\_public: Boolean
19. custom\_features: JSONB array
20. user\_types: JSONB array (user types that can access this tier)
21. created\_at: Timestamp
22. updated\_at: Timestamp
23. user\_subscriptions
24. id: UUID primary key
25. user\_id: UUID foreign key to users
26. tier\_id: UUID foreign key to subscription\_tiers
27. status: String (active, trialing, past\_due, canceled, etc.)
28. start\_date: Timestamp
29. end\_date: Timestamp
30. renewal\_date: Timestamp
31. canceled\_at: Timestamp
32. trial\_end\_date: Timestamp
33. payment\_method: String
34. payment\_id: String
35. stripe\_customer\_id: String
36. stripe\_subscription\_id: String
37. stripe\_price\_id: String
38. stripe\_payment\_method\_id: String
39. current\_period\_start: Timestamp
40. current\_period\_end: Timestamp
41. cancel\_at\_period\_end: Boolean
42. auto\_renew: Boolean
43. usage: JSONB object
44. metadata: JSONB object
45. created\_at: Timestamp
46. updated\_at: Timestamp
47. user\_credits
48. id: UUID primary key
49. user\_id: UUID foreign key to users
50. balance: Integer
51. last\_updated\_at: Timestamp
52. created\_at: Timestamp
53. credit\_transactions
54. id: UUID primary key
55. user\_id: UUID foreign key to users
56. amount: Integer
57. balance: Integer
58. description: String
59. type: String (purchase, usage, refund, expiration, adjustment, subscription)
60. metadata: JSONB object
61. created\_at: Timestamp
62. expires\_at: Timestamp
63. subscription\_tier\_versions
64. id: UUID primary key
65. tier\_id: UUID foreign key to subscription\_tiers
66. version\_number: Integer
67. changes: JSONB object
68. effective\_date: Timestamp
69. created\_at: Timestamp
70. created\_by: UUID foreign key to users
71. subscription\_state\_transitions
72. id: UUID primary key
73. subscription\_id: UUID foreign key to user\_subscriptions
74. from\_state: String
75. to\_state: String
76. reason: String
77. metadata: JSONB object
78. created\_at: Timestamp
79. subscription\_tier\_user\_types
80. id: UUID primary key
81. tier\_id: UUID foreign key to subscription\_tiers
82. user\_type: String enum ('user', 'factory', 'b2b', 'admin')
83. created\_at: Timestamp
84. updated\_at: Timestamp

updated\_at: Timestamp

user\_subscriptions

updated\_at: Timestamp

user\_credits

created\_at: Timestamp

credit\_transactions

expires\_at: Timestamp

subscription\_tier\_versions

created\_by: UUID foreign key to users

subscription\_state\_transitions

created\_at: Timestamp

subscription\_tier\_user\_types

### Frontend Implementation

#### User Interface Components

1. SubscriptionPlans (packages/client/src/components/subscription/SubscriptionPlans.tsx)
2. Displays available subscription plans
3. Shows features and pricing for each plan
4. Allows users to select a plan
5. PaymentForm (packages/client/src/components/subscription/PaymentForm.tsx)
6. Collects payment information using Stripe Elements
7. Handles subscription creation and updates
8. Processes payment method changes
9. CreditManagement (packages/client/src/components/subscription/CreditManagement.tsx)
10. Displays credit balance and transaction history
11. Allows users to purchase credits
12. Shows credit usage by feature
13. PaymentMethodSelector (packages/client/src/components/subscription/PaymentMethodSelector.tsx)
14. Displays saved payment methods
15. Allows users to add, edit, and remove payment methods
16. Sets default payment method

packages/client/src/components/subscription/SubscriptionPlans.tsx

Allows users to select a plan

PaymentForm (packages/client/src/components/subscription/PaymentForm.tsx)

packages/client/src/components/subscription/PaymentForm.tsx

Processes payment method changes

CreditManagement (packages/client/src/components/subscription/CreditManagement.tsx)

packages/client/src/components/subscription/CreditManagement.tsx

Shows credit usage by feature

PaymentMethodSelector (packages/client/src/components/subscription/PaymentMethodSelector.tsx)

packages/client/src/components/subscription/PaymentMethodSelector.tsx

#### Admin Interface Components

1. AdminSubscriptionManagement (packages/admin/src/components/subscription/AdminSubscriptionManagement.tsx)
2. Manages subscription tiers and versions
3. Displays user subscriptions and credits
4. Shows subscription analytics
5. SubscriptionTierForm (packages/admin/src/components/subscription/SubscriptionTierForm.tsx)
6. Creates and edits subscription tiers
7. Configures module access, API limits, and resource limits
8. Sets pricing and billing intervals
9. Specifies which user types can access the tier
10. UserTypeManagement (packages/admin/src/components/user/UserTypeManagement.tsx)
11. Displays users with their current types
12. Allows changing a user's type between user, factory, b2b, and admin
13. Shows available subscription tiers for each user type
14. SubscriptionAnalyticsChart (packages/admin/src/components/subscription/SubscriptionAnalyticsChart.tsx)
15. Displays subscription analytics in chart form
16. Shows revenue, subscriber counts, and churn rate
17. Visualizes subscription distribution by user type

packages/admin/src/components/subscription/AdminSubscriptionManagement.tsx

Shows subscription analytics

SubscriptionTierForm (packages/admin/src/components/subscription/SubscriptionTierForm.tsx)

packages/admin/src/components/subscription/SubscriptionTierForm.tsx

Specifies which user types can access the tier

UserTypeManagement (packages/admin/src/components/user/UserTypeManagement.tsx)

packages/admin/src/components/user/UserTypeManagement.tsx

Shows available subscription tiers for each user type

SubscriptionAnalyticsChart (packages/admin/src/components/subscription/SubscriptionAnalyticsChart.tsx)

packages/admin/src/components/subscription/SubscriptionAnalyticsChart.tsx

#### Pages

1. Subscription Page (packages/client/src/pages/subscription/index.tsx)
2. Main subscription management page for users
3. Shows current subscription details
4. Allows users to change plans, manage credits, and update payment methods
5. Admin Subscription Page (packages/admin/src/pages/subscriptions/index.tsx)
6. Main subscription management page for admins
7. Manages subscription tiers and user subscriptions
8. Displays subscription analytics
9. Configures which user types can access each tier
10. Admin User Type Page (packages/admin/src/pages/user-types/index.tsx)
11. Manages user types (user, factory, b2b, admin)
12. Allows changing a user's type
13. Shows subscription tiers available for each user type

packages/client/src/pages/subscription/index.tsx

Allows users to change plans, manage credits, and update payment methods

Admin Subscription Page (packages/admin/src/pages/subscriptions/index.tsx)

packages/admin/src/pages/subscriptions/index.tsx

Configures which user types can access each tier

Admin User Type Page (packages/admin/src/pages/user-types/index.tsx)

packages/admin/src/pages/user-types/index.tsx

### Stripe Integration

The system integrates with Stripe for payment processing:

#### Customer Management

* Creating customers in Stripe when users sign up
* Linking Stripe customers to users in the database
* Managing customer payment methods

#### Subscription Management

* Creating subscriptions in Stripe
* Updating subscriptions when users change plans
* Canceling subscriptions
* Handling subscription lifecycle events (renewal, payment failure, etc.)

#### Webhook Handling

The system handles the following Stripe webhook events:

* customer.subscription.created
* customer.subscription.updated
* customer.subscription.deleted
* invoice.payment\_succeeded
* invoice.payment\_failed
* payment\_intent.succeeded
* payment\_intent.payment\_failed

customer.subscription.created

customer.subscription.updated

customer.subscription.deleted

invoice.payment\_succeeded

invoice.payment\_failed

payment\_intent.succeeded

payment\_intent.payment\_failed

### Credit System

#### Credit Allocation

* Credits included with subscription
* Credits purchased separately
* Credits with expiration dates

#### Credit Usage

* Using credits for specific actions
* Tracking credit usage
* Preventing actions when credits are insufficient

### Analytics and Reporting

The analytics system provides insights into subscription metrics:

#### Revenue Metrics

* Monthly Recurring Revenue (MRR)
* Annual Recurring Revenue (ARR)
* Average Revenue Per User (ARPU)

#### Subscription Metrics

* Total subscribers
* Active subscribers
* Churn rate
* Conversion rate

#### Usage Metrics

* API usage
* Storage utilization
* Feature adoption

#### Credit Usage

* Credit consumption by feature
* Credit purchase patterns
* Credit usage trends

### Implementation Guides

#### Creating a New Subscription Tier

// Example: Creating a new tier  
const newTier = {  
 name: "Premium",  
 description: "Premium access with advanced features",  
 price: 29.99,  
 currency: "usd",  
 billingInterval: "monthly",  
 moduleAccess: [  
 { name: "materialRecognition", enabled: true },  
 { name: "knowledgeBase", enabled: true },  
 { name: "agents", enabled: true },  
 { name: "advancedAgents", enabled: true },  
 { name: "3dDesigner", enabled: true },  
 { name: "api", enabled: true },  
 { name: "pdf", enabled: true },  
 { name: "crawler", enabled: false },  
 { name: "mlTraining", enabled: false }  
 ],  
 apiLimits: {  
 requestsPerMinute: 60,  
 requestsPerDay: 500,  
 requestsPerMonth: 5000,  
 includedModules: ["materialRecognition", "knowledgeBase", "agents"]  
 },  
 storageLimits: {  
 maxStorageGB: 10,  
 maxFileSize: 50,  
 maxFilesPerProject: 500  
 },  
 creditLimits: {  
 includedCredits: 100,  
 maxPurchasableCredits: 10000,  
 creditPriceMultiplier: 0.9  
 },  
 maxProjects: 10,  
 maxTeamMembers: 5,  
 maxMoodboards: 20,  
 supportLevel: "priority",  
 isPublic: true,  
 userTypes: ["user", "factory"] // This tier is available for regular users and factories  
};  
  
// Create the tier  
const createdTier = await createSubscriptionTier(newTier);

// Example: Creating a new tier  
const newTier = {  
 name: "Premium",  
 description: "Premium access with advanced features",  
 price: 29.99,  
 currency: "usd",  
 billingInterval: "monthly",  
 moduleAccess: [  
 { name: "materialRecognition", enabled: true },  
 { name: "knowledgeBase", enabled: true },  
 { name: "agents", enabled: true },  
 { name: "advancedAgents", enabled: true },  
 { name: "3dDesigner", enabled: true },  
 { name: "api", enabled: true },  
 { name: "pdf", enabled: true },  
 { name: "crawler", enabled: false },  
 { name: "mlTraining", enabled: false }  
 ],  
 apiLimits: {  
 requestsPerMinute: 60,  
 requestsPerDay: 500,  
 requestsPerMonth: 5000,  
 includedModules: ["materialRecognition", "knowledgeBase", "agents"]  
 },  
 storageLimits: {  
 maxStorageGB: 10,  
 maxFileSize: 50,  
 maxFilesPerProject: 500  
 },  
 creditLimits: {  
 includedCredits: 100,  
 maxPurchasableCredits: 10000,  
 creditPriceMultiplier: 0.9  
 },  
 maxProjects: 10,  
 maxTeamMembers: 5,  
 maxMoodboards: 20,  
 supportLevel: "priority",  
 isPublic: true,  
 userTypes: ["user", "factory"] // This tier is available for regular users and factories  
};  
  
// Create the tier  
const createdTier = await createSubscriptionTier(newTier);

#### Subscribing to a Paid Plan

// Example: Subscribing to a paid plan  
const subscription = await subscribeToPaidPlan(  
 userId,  
 tierId,  
 paymentMethodId,  
 {  
 trialDays: 14,  
 metadata: { referral: "marketing\_campaign" }  
 }  
);

// Example: Subscribing to a paid plan  
const subscription = await subscribeToPaidPlan(  
 userId,  
 tierId,  
 paymentMethodId,  
 {  
 trialDays: 14,  
 metadata: { referral: "marketing\_campaign" }  
 }  
);

#### Managing Credits

// Example: Adding credits to a user  
const result = await addCredits(  
 userId,  
 100,  
 "Credit purchase",  
 "purchase",  
 { paymentId: "pi\_123456" }  
);  
  
// Example: Using credits  
const result = await useCredits(  
 userId,  
 10,  
 "Generated 3D model",  
 "usage",  
 { feature: "3dDesigner", modelId: "model\_123" }  
);

// Example: Adding credits to a user  
const result = await addCredits(  
 userId,  
 100,  
 "Credit purchase",  
 "purchase",  
 { paymentId: "pi\_123456" }  
);  
  
// Example: Using credits  
const result = await useCredits(  
 userId,  
 10,  
 "Generated 3D model",  
 "usage",  
 { feature: "3dDesigner", modelId: "model\_123" }  
);

#### Checking Module Access

// Example: Checking if a user has access to a module  
const hasAccess = await hasModuleAccess(userId, 'advancedAgents');  
  
if (hasAccess) {  
 // Allow access to advanced agents  
} else {  
 // Deny access with upgrade suggestion  
}

// Example: Checking if a user has access to a module  
const hasAccess = await hasModuleAccess(userId, 'advancedAgents');  
  
if (hasAccess) {  
 // Allow access to advanced agents  
} else {  
 // Deny access with upgrade suggestion  
}

#### Managing User Types

// Example: Changing a user's type  
const updatedUser = await updateUserType(  
 userId,  
 'factory',  
 {  
 reason: 'User requested factory access',  
 approvedBy: adminId  
 }  
);  
  
// Example: Getting subscription tiers for a user type  
const factoryTiers = await getSubscriptionTiersByUserType('factory');  
  
// Example: Associating a tier with a user type  
const result = await associateTierWithUserType(tierId, 'b2b');

// Example: Changing a user's type  
const updatedUser = await updateUserType(  
 userId,  
 'factory',  
 {  
 reason: 'User requested factory access',  
 approvedBy: adminId  
 }  
);  
  
// Example: Getting subscription tiers for a user type  
const factoryTiers = await getSubscriptionTiersByUserType('factory');  
  
// Example: Associating a tier with a user type  
const result = await associateTierWithUserType(tierId, 'b2b');

#### Tracking API Usage

// Example: Tracking API usage  
app.use('/api/external', async (req, res, next) => {  
 const userId = req.user.id;  
  
 // Check if user has reached the limit  
 const hasReachedLimit = await hasReachedApiLimit(userId);  
  
 if (hasReachedLimit) {  
 return res.status(429).json({  
 error: 'Rate limit exceeded',  
 message: 'Please upgrade your subscription for higher limits'  
 });  
 }  
  
 // Track the API usage  
 await trackApiUsage(userId);  
  
 // Continue with the request  
 next();  
});

// Example: Tracking API usage  
app.use('/api/external', async (req, res, next) => {  
 const userId = req.user.id;  
  
 // Check if user has reached the limit  
 const hasReachedLimit = await hasReachedApiLimit(userId);  
  
 if (hasReachedLimit) {  
 return res.status(429).json({  
 error: 'Rate limit exceeded',  
 message: 'Please upgrade your subscription for higher limits'  
 });  
 }  
  
 // Track the API usage  
 await trackApiUsage(userId);  
  
 // Continue with the request  
 next();  
});

### Deployment Considerations

1. Database Migrations
2. Execute migrations to create necessary tables
3. Set up foreign key relationships and indices
4. Environment Configuration
5. Configure Stripe API keys
6. Set up webhook endpoints
7. Configure email notifications
8. Default Tiers
9. Create default subscription tiers during initial deployment
10. Ensure a free tier exists for new users
11. Configure which user types can access each tier
12. User Migration
13. Associate existing users with appropriate subscription tiers
14. Initialize credit balances for existing users
15. Assign appropriate user types to existing users

Set up foreign key relationships and indices

Environment Configuration

Configure email notifications

Default Tiers

Configure which user types can access each tier

User Migration

### Security Considerations

1. Payment Information
2. Use Stripe Elements to securely collect payment information
3. Never store credit card details in your database
4. Use Stripe's secure webhooks for payment events
5. Access Control
6. Validate subscription status and permissions for all protected routes
7. Prevent unauthorized access to subscription management
8. Implement proper authentication for webhook endpoints
9. Rate Limiting
10. Implement robust rate limiting to prevent abuse
11. Provide clear feedback on rate limit status
12. Monitor for unusual usage patterns
13. Admin Protection
14. Restrict tier management to admin users only
15. Log all changes to subscription tiers and user subscriptions
16. Implement proper audit trails for sensitive operations
17. Secure user type changes with proper authorization checks
18. Log all user type changes for audit purposes

Use Stripe's secure webhooks for payment events

Access Control

Implement proper authentication for webhook endpoints

Rate Limiting

Monitor for unusual usage patterns

Admin Protection

### Best Practices

1. Subscription Management
2. Provide clear information about subscription features and limits
3. Implement smooth upgrade/downgrade flows
4. Send notifications for important subscription events
5. Show appropriate subscription tiers based on user type
6. Credit System
7. Clearly communicate credit costs for different actions
8. Provide usage history and balance information
9. Implement automatic credit purchase options
10. Payment Processing
11. Handle payment failures gracefully
12. Implement retry logic for failed payments
13. Provide clear error messages for payment issues
14. User Type Management
15. Provide clear information about the benefits of each user type
16. Implement a smooth process for users to request type changes
17. Ensure proper validation before changing user types
18. Notify users when their type changes
19. Analytics
20. Regularly review subscription metrics
21. Use analytics to inform pricing decisions
22. Monitor for unusual patterns or potential issues
23. Track subscription distribution by user type

Show appropriate subscription tiers based on user type

Credit System

Implement automatic credit purchase options

Payment Processing

Provide clear error messages for payment issues

User Type Management

Notify users when their type changes

Analytics

### Conclusion

The Subscription Management System provides a comprehensive solution for managing subscriptions, payments, and access control. It integrates with Stripe for payment processing and offers a flexible, module-based access control system with tiered pricing and granular feature access.

By implementing this system, you can:  
- Support different user types (users, factories, b2b, admin) with specialized access  
- Offer multiple subscription tiers with different features and limits for each user type  
- Process payments securely using Stripe  
- Manage user credits for premium features  
- Track and enforce usage limits  
- Provide a seamless user experience for subscription management  
- Gain valuable insights into subscription metrics and user behavior  
- Easily manage user type changes through the admin panel

# Subscription Tier System

Source: readme/subscription-tier-system.md

---

## Subscription Tier System

### Executive Summary

The Subscription Tier System implements a flexible, module-based access control framework for the Kai platform, enabling tiered pricing with granular feature access. It controls what features users can access based on their subscription level and enforces API usage limits for both internal and external requests.

Key capabilities:  
- Multiple pricing tiers with different feature sets  
- Module-based access control for granular permission management  
- API rate limiting with tier-specific quotas  
- Usage tracking and analytics  
- Framework for future Stripe integration

### System Architecture

The system follows a layered architecture that integrates with existing authentication:

1. Authentication Layer: Validates user identity via Supabase Auth
2. Subscription Layer: Verifies subscription status and tier permissions
3. Access Control Layer: Enforces module-specific permissions
4. Rate Limiting Layer: Controls API usage based on tier limits
5. Analytics Layer: Tracks usage patterns and subscription metrics

### Core Components

#### 1. Subscription Tiers

Subscription tiers define the available plans users can subscribe to. Each tier specifies:

* Basic Information: Name, price, description, and visibility (public/private)
* Module Access: Which platform modules are enabled for this tier
* API Limits: Daily, monthly, and concurrent API request limits

Tiers are stored in the subscription\_tiers table in Supabase and offer different capability levels:

subscription\_tiers

Table content:

Tier | Description | Key Features

Free | Basic access with limited features | Basic material recognition, limited knowledge base

Basic | Standard access for individual users | Material recognition, knowledge base, basic agent access

Professional | Advanced access for professionals | All basic features + advanced agents, 3D designer access

Enterprise | Full access for organizations | Unlimited access to all features, maximum API limits

Custom | Tailored solutions for specific needs | Custom configuration of all modules and limits

#### 2. User Subscriptions

User subscriptions link users to their selected subscription tier and track usage metrics:

* Subscription Information: Tier ID, status (active, trialing, canceled, etc.), renewal date
* Usage Tracking: API requests count, reset periods, and module-specific usage
* Billing Details: Payment method, billing cycle (future Stripe integration)

#### 3. Module-Based Access Control

Access to specific platform functionalities is controlled at the module level:

"moduleAccess": [  
 { "name": "materialRecognition", "enabled": true },  
 { "name": "knowledgeBase", "enabled": true },  
 { "name": "agents", "enabled": true },  
 { "name": "advancedAgents", "enabled": false },  
 { "name": "3dDesigner", "enabled": false },  
 { "name": "api", "enabled": false },  
 { "name": "pdf", "enabled": true },  
 { "name": "crawler", "enabled": false },  
 { "name": "mlTraining", "enabled": false }  
]

"moduleAccess": [  
 { "name": "materialRecognition", "enabled": true },  
 { "name": "knowledgeBase", "enabled": true },  
 { "name": "agents", "enabled": true },  
 { "name": "advancedAgents", "enabled": false },  
 { "name": "3dDesigner", "enabled": false },  
 { "name": "api", "enabled": false },  
 { "name": "pdf", "enabled": true },  
 { "name": "crawler", "enabled": false },  
 { "name": "mlTraining", "enabled": false }  
]

This granular approach allows for flexible tier configuration and precise access control.

#### 4. API Rate Limiting

The system enforces usage limits based on subscription tier:

* Daily Limits: Maximum requests per day
* Monthly Limits: Maximum requests per month
* Concurrent Limits: Maximum simultaneous requests

Usage is tracked per user and automatically reset on appropriate periods (daily or monthly).

### Backend Implementation

#### Models

1. SubscriptionTier Model (packages/server/src/models/subscriptionTier.model.ts)
2. Core data structure for subscription tiers
3. CRUD operations for tier management
4. Includes price, features, module access permissions, and API limits
5. UserSubscription Model (packages/server/src/models/userSubscription.model.ts)
6. Links users to subscription tiers
7. Tracks API usage and module access
8. Handles resets for usage limits
9. Provides utility functions for checking permissions and limits

packages/server/src/models/subscriptionTier.model.ts

Includes price, features, module access permissions, and API limits

UserSubscription Model (packages/server/src/models/userSubscription.model.ts)

packages/server/src/models/userSubscription.model.ts

#### Middleware

1. Module Access Middleware (packages/server/src/middleware/module-access.middleware.ts)
2. Verifies module access permissions based on subscription tier
3. Protects routes by checking subscription status
4. Integrates with existing auth middleware
5. Rate Limiting Middleware (packages/server/src/middleware/rate-limit.middleware.ts)
6. Enforces API request limits based on subscription tier
7. Provides per-user rate limiting with tier-specific quotas
8. Tracks and limits concurrent API usage

packages/server/src/middleware/module-access.middleware.ts

Integrates with existing auth middleware

Rate Limiting Middleware (packages/server/src/middleware/rate-limit.middleware.ts)

packages/server/src/middleware/rate-limit.middleware.ts

#### API Routes

1. Subscription Routes (packages/server/src/routes/subscription.routes.ts)
2. Public endpoints for retrieving available tiers
3. Protected endpoints for managing user subscriptions
4. Admin endpoints for CRUD operations on subscription tiers

packages/server/src/routes/subscription.routes.ts

#### Database Schema

Subscription data is stored in Supabase with these primary tables:

1. subscription\_tiers
2. id: UUID primary key
3. name: String
4. price: Decimal
5. description: Text
6. isPublic: Boolean
7. moduleAccess: JSONB array
8. apiLimits: JSONB object
9. createdAt: Timestamp
10. updatedAt: Timestamp
11. user\_subscriptions
12. id: UUID primary key
13. userId: UUID foreign key to users
14. tierId: UUID foreign key to subscription\_tiers
15. status: String (active, trialing, canceled, etc.)
16. usage: JSONB object for tracking API usage
17. startDate: Timestamp
18. endDate: Timestamp
19. renewalDate: Timestamp
20. createdAt: Timestamp
21. updatedAt: Timestamp

updatedAt: Timestamp

user\_subscriptions

### Frontend Implementation

#### User Interface

1. User Profile Page (packages/client/src/pages/profile.tsx)
2. Displays current subscription details
3. Shows API usage statistics with visual indicators
4. Lists available modules with access status
5. Allows changing subscription tier
6. Admin Subscription Management (packages/admin/src/pages/subscription-tiers/index.tsx)
7. CRUD operations for subscription tiers
8. Detailed configuration of module permissions
9. Setting API rate limits
10. Controlling tier visibility

packages/client/src/pages/profile.tsx

Allows changing subscription tier

Admin Subscription Management (packages/admin/src/pages/subscription-tiers/index.tsx)

packages/admin/src/pages/subscription-tiers/index.tsx

#### Navigation

The subscription management functionality is integrated into both user and admin navigation:

* User profile section includes subscription management
* Admin sidebar includes a dedicated "Subscription Tiers" section

### Module Access Control

The system controls access to the following modules:

Table content:

Module | Description

materialRecognition | Material recognition from images

knowledgeBase | Knowledge base search and access

agents | Basic agent access (Material Expert)

advancedAgents | Advanced specialized agents

3dDesigner | 3D visualization and design tools

api | External API access

pdf | PDF processing and extraction

crawler | Web crawler functionality

mlTraining | ML model training capabilities

### API Usage Tracking

API usage is tracked at multiple levels:

1. Request Counting
2. Increments usage counters for both internal and external API requests
3. Resets based on configured periods (daily/monthly)
4. Usage Visualization
5. Provides users with visual indicators of their usage
6. Shows remaining quota and reset dates
7. Rate Limiting
8. Enforces tier-specific request limits
9. Returns appropriate error responses when limits are exceeded

Resets based on configured periods (daily/monthly)

Usage Visualization

Shows remaining quota and reset dates

Rate Limiting

### Future Stripe Integration

The system is designed for future integration with Stripe for payment processing:

1. Billing Information
2. User subscription will store payment method details
3. Support for recurring billing cycles
4. Subscription Management
5. Handling tier upgrades/downgrades
6. Prorated billing for mid-cycle changes
7. Free trial periods with automatic conversion
8. Webhook Integration
9. Handling Stripe events (successful payments, failed payments, etc.)
10. Automatic subscription status updates

Support for recurring billing cycles

Subscription Management

Free trial periods with automatic conversion

Webhook Integration

### Authentication Integration

The subscription system integrates with the existing Supabase authentication:

1. User Registration
2. New users automatically get a free tier subscription
3. Option to upgrade during registration
4. Access Control
5. JWT tokens include subscription information
6. Auth middleware works with module access middleware

Option to upgrade during registration

Access Control

### Implementation Guides

#### Creating a New Subscription Tier

// Example: Creating a new tier  
const newTier = {  
 name: "Premium",  
 price: 29.99,  
 isPublic: true,  
 description: "Premium access with advanced features",  
 moduleAccess: [  
 { name: "materialRecognition", enabled: true },  
 { name: "knowledgeBase", enabled: true },  
 { name: "agents", enabled: true },  
 { name: "advancedAgents", enabled: true },  
 { name: "3dDesigner", enabled: true },  
 { name: "api", enabled: true },  
 { name: "pdf", enabled: true },  
 { name: "crawler", enabled: false },  
 { name: "mlTraining", enabled: false }  
 ],  
 apiLimits: {  
 requestsPerDay: 500,  
 requestsPerMonth: 5000,  
 concurrentRequests: 10  
 }  
};  
  
// Create the tier  
const createdTier = await createSubscriptionTier(newTier);

// Example: Creating a new tier  
const newTier = {  
 name: "Premium",  
 price: 29.99,  
 isPublic: true,  
 description: "Premium access with advanced features",  
 moduleAccess: [  
 { name: "materialRecognition", enabled: true },  
 { name: "knowledgeBase", enabled: true },  
 { name: "agents", enabled: true },  
 { name: "advancedAgents", enabled: true },  
 { name: "3dDesigner", enabled: true },  
 { name: "api", enabled: true },  
 { name: "pdf", enabled: true },  
 { name: "crawler", enabled: false },  
 { name: "mlTraining", enabled: false }  
 ],  
 apiLimits: {  
 requestsPerDay: 500,  
 requestsPerMonth: 5000,  
 concurrentRequests: 10  
 }  
};  
  
// Create the tier  
const createdTier = await createSubscriptionTier(newTier);

#### Checking Module Access

// Example: Checking if a user has access to a module  
const hasAccess = await hasModuleAccess(userId, 'advancedAgents');  
  
if (hasAccess) {  
 // Allow access to advanced agents  
} else {  
 // Deny access with upgrade suggestion  
}

// Example: Checking if a user has access to a module  
const hasAccess = await hasModuleAccess(userId, 'advancedAgents');  
  
if (hasAccess) {  
 // Allow access to advanced agents  
} else {  
 // Deny access with upgrade suggestion  
}

#### Tracking API Usage

// Example: Tracking API usage  
app.use('/api/external', async (req, res, next) => {  
 const userId = req.user.id;  
  
 // Check if user has reached the limit  
 const hasReachedLimit = await hasReachedApiLimit(userId);  
  
 if (hasReachedLimit) {  
 return res.status(429).json({   
 error: 'Rate limit exceeded',  
 message: 'Please upgrade your subscription for higher limits'  
 });  
 }  
  
 // Track the API usage  
 await trackApiUsage(userId);  
  
 // Continue with the request  
 next();  
});

// Example: Tracking API usage  
app.use('/api/external', async (req, res, next) => {  
 const userId = req.user.id;  
  
 // Check if user has reached the limit  
 const hasReachedLimit = await hasReachedApiLimit(userId);  
  
 if (hasReachedLimit) {  
 return res.status(429).json({   
 error: 'Rate limit exceeded',  
 message: 'Please upgrade your subscription for higher limits'  
 });  
 }  
  
 // Track the API usage  
 await trackApiUsage(userId);  
  
 // Continue with the request  
 next();  
});

### Deployment Considerations

1. Database Migrations
2. Execute migrations to create subscription\_tiers and user\_subscriptions tables
3. Set up foreign key relationships and indices
4. Default Tiers
5. Create default subscription tiers during initial deployment
6. Ensure a free tier exists for new users
7. User Migration
8. Associate existing users with appropriate subscription tiers
9. Provide a migration path for existing data

Set up foreign key relationships and indices

Default Tiers

Ensure a free tier exists for new users

User Migration

### Security Considerations

1. Access Control
2. Validate subscription status and permissions for all protected routes
3. Prevent unauthorized access to subscription management
4. Rate Limiting
5. Implement robust rate limiting to prevent abuse
6. Provide clear feedback on rate limit status
7. Admin Protection
8. Restrict tier management to admin users only
9. Log all changes to subscription tiers and user subscriptions

Prevent unauthorized access to subscription management

Rate Limiting

Provide clear feedback on rate limit status

Admin Protection

### Monitoring and Analytics

The system provides analytics and monitoring capabilities:

1. Usage Patterns
2. Track most used features per tier
3. Identify upgrade opportunities
4. Revenue Metrics
5. Monitor subscription revenue by tier
6. Track conversion rates and upgrades
7. System Health
8. Monitor rate limit effectiveness
9. Identify potential abuse patterns

Identify upgrade opportunities

Revenue Metrics

Track conversion rates and upgrades

System Health

# Supabase Setup Guide

Source: readme/supabase-setup-guide.md

---

## Supabase Setup Guide for Kai

This guide provides detailed instructions for setting up Supabase for the Kai application, covering authentication, database configuration, storage, and realtime features.

### Table of Contents

* Prerequisites
* Creating a Supabase Project
* Database Schema Setup
* Authentication Configuration
* Storage Buckets
* Realtime Configuration
* API Keys and Security
* Monitoring and Maintenance

### Prerequisites

Before starting, you'll need:

* A Supabase account
* Basic understanding of SQL
* Domain name(s) for your application

### Creating a Supabase Project

#### Step 1: Sign Up or Log in to Supabase

Visit https://app.supabase.io/ and log in with your account.

#### Step 2: Create a New Project

1. Click New Project
2. Enter project details:
3. Name: kai-production (or your preferred name)
4. Database Password: Generate a strong password (save this securely)
5. Region: Choose the region closest to your users and other services
6. Pricing Plan: Choose appropriate plan (Free tier for development, Pro for production)
7. Click Create New Project

kai-production

Project creation will take 1-2 minutes.

### Database Schema Setup

#### Setting Up the Queue System Tables

The Kai application uses Supabase for its queue system. Set up the required tables:

1. Navigate to the SQL Editor in the Supabase dashboard
2. Create a new query and enter the following SQL:

-- Create necessary tables for queue management  
CREATE TABLE public.queue\_jobs (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 job\_type VARCHAR NOT NULL,  
 status VARCHAR NOT NULL DEFAULT 'pending',  
 payload JSONB NOT NULL,  
 result JSONB,  
 error TEXT,  
 priority INT NOT NULL DEFAULT 0,  
 attempts INT NOT NULL DEFAULT 0,  
 max\_attempts INT NOT NULL DEFAULT 3,  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 started\_at TIMESTAMPTZ,  
 completed\_at TIMESTAMPTZ  
);  
  
-- Create index for job processing  
CREATE INDEX queue\_jobs\_status\_priority\_created\_idx ON public.queue\_jobs (status, priority DESC, created\_at);  
  
-- Create tables for job relationships  
CREATE TABLE public.job\_dependencies (  
 job\_id UUID REFERENCES public.queue\_jobs(id) ON DELETE CASCADE,  
 depends\_on UUID REFERENCES public.queue\_jobs(id) ON DELETE CASCADE,  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 PRIMARY KEY (job\_id, depends\_on)  
);  
  
-- Create job history table for completed/failed jobs  
CREATE TABLE public.job\_history (  
 id UUID PRIMARY KEY,  
 job\_type VARCHAR NOT NULL,  
 status VARCHAR NOT NULL,  
 payload JSONB NOT NULL,  
 result JSONB,  
 error TEXT,  
 attempts INT NOT NULL,  
 processing\_time\_ms INT,  
 created\_at TIMESTAMPTZ NOT NULL,  
 completed\_at TIMESTAMPTZ NOT NULL,  
 archived\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()  
);

-- Create necessary tables for queue management  
CREATE TABLE public.queue\_jobs (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 job\_type VARCHAR NOT NULL,  
 status VARCHAR NOT NULL DEFAULT 'pending',  
 payload JSONB NOT NULL,  
 result JSONB,  
 error TEXT,  
 priority INT NOT NULL DEFAULT 0,  
 attempts INT NOT NULL DEFAULT 0,  
 max\_attempts INT NOT NULL DEFAULT 3,  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 started\_at TIMESTAMPTZ,  
 completed\_at TIMESTAMPTZ  
);  
  
-- Create index for job processing  
CREATE INDEX queue\_jobs\_status\_priority\_created\_idx ON public.queue\_jobs (status, priority DESC, created\_at);  
  
-- Create tables for job relationships  
CREATE TABLE public.job\_dependencies (  
 job\_id UUID REFERENCES public.queue\_jobs(id) ON DELETE CASCADE,  
 depends\_on UUID REFERENCES public.queue\_jobs(id) ON DELETE CASCADE,  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 PRIMARY KEY (job\_id, depends\_on)  
);  
  
-- Create job history table for completed/failed jobs  
CREATE TABLE public.job\_history (  
 id UUID PRIMARY KEY,  
 job\_type VARCHAR NOT NULL,  
 status VARCHAR NOT NULL,  
 payload JSONB NOT NULL,  
 result JSONB,  
 error TEXT,  
 attempts INT NOT NULL,  
 processing\_time\_ms INT,  
 created\_at TIMESTAMPTZ NOT NULL,  
 completed\_at TIMESTAMPTZ NOT NULL,  
 archived\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()  
);

1. Execute the query

#### Setting Up User Profiles and Permissions

1. Create user profiles table:

CREATE TABLE public.profiles (  
 id UUID PRIMARY KEY REFERENCES auth.users(id) ON DELETE CASCADE,  
 email VARCHAR NOT NULL,  
 first\_name VARCHAR,  
 last\_name VARCHAR,  
 avatar\_url VARCHAR,  
 role VARCHAR NOT NULL DEFAULT 'user',  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()  
);  
  
-- Enable Row Level Security  
ALTER TABLE public.profiles ENABLE ROW LEVEL SECURITY;  
  
-- Create RLS policies  
CREATE POLICY "Users can view their own profile"  
 ON public.profiles FOR SELECT  
 USING (auth.uid() = id);  
  
CREATE POLICY "Users can update their own profile"  
 ON public.profiles FOR UPDATE  
 USING (auth.uid() = id);  
  
-- Only allow administrators to select all profiles  
CREATE POLICY "Admins can view all profiles"  
 ON public.profiles FOR SELECT  
 USING (  
 EXISTS (  
 SELECT 1 FROM public.profiles  
 WHERE id = auth.uid() AND role = 'admin'  
 )  
 );

CREATE TABLE public.profiles (  
 id UUID PRIMARY KEY REFERENCES auth.users(id) ON DELETE CASCADE,  
 email VARCHAR NOT NULL,  
 first\_name VARCHAR,  
 last\_name VARCHAR,  
 avatar\_url VARCHAR,  
 role VARCHAR NOT NULL DEFAULT 'user',  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT NOW()  
);  
  
-- Enable Row Level Security  
ALTER TABLE public.profiles ENABLE ROW LEVEL SECURITY;  
  
-- Create RLS policies  
CREATE POLICY "Users can view their own profile"  
 ON public.profiles FOR SELECT  
 USING (auth.uid() = id);  
  
CREATE POLICY "Users can update their own profile"  
 ON public.profiles FOR UPDATE  
 USING (auth.uid() = id);  
  
-- Only allow administrators to select all profiles  
CREATE POLICY "Admins can view all profiles"  
 ON public.profiles FOR SELECT  
 USING (  
 EXISTS (  
 SELECT 1 FROM public.profiles  
 WHERE id = auth.uid() AND role = 'admin'  
 )  
 );

#### Setting Up Materials Management Tables

Create tables for storing materials metadata (if not using MongoDB for this):

-- Materials table  
CREATE TABLE public.materials\_metadata (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 external\_id VARCHAR,  
 name VARCHAR NOT NULL,  
 description TEXT,  
 material\_type VARCHAR NOT NULL,  
 manufacturer VARCHAR,  
 tags JSONB DEFAULT '[]'::jsonb,  
 properties JSONB DEFAULT '{}'::jsonb,  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 created\_by UUID REFERENCES auth.users(id)  
);  
  
-- Enable Row Level Security  
ALTER TABLE public.materials\_metadata ENABLE ROW LEVEL SECURITY;  
  
-- Create RLS policies  
CREATE POLICY "Materials are viewable by everyone"  
 ON public.materials\_metadata FOR SELECT  
 USING (true);  
  
CREATE POLICY "Materials are insertable by authenticated users"  
 ON public.materials\_metadata FOR INSERT  
 WITH CHECK (auth.role() = 'authenticated');  
  
CREATE POLICY "Materials are updatable by creators or admins"  
 ON public.materials\_metadata FOR UPDATE  
 USING (  
 created\_by = auth.uid() OR  
 EXISTS (  
 SELECT 1 FROM public.profiles  
 WHERE id = auth.uid() AND role = 'admin'  
 )  
 );

-- Materials table  
CREATE TABLE public.materials\_metadata (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 external\_id VARCHAR,  
 name VARCHAR NOT NULL,  
 description TEXT,  
 material\_type VARCHAR NOT NULL,  
 manufacturer VARCHAR,  
 tags JSONB DEFAULT '[]'::jsonb,  
 properties JSONB DEFAULT '{}'::jsonb,  
 created\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 updated\_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
 created\_by UUID REFERENCES auth.users(id)  
);  
  
-- Enable Row Level Security  
ALTER TABLE public.materials\_metadata ENABLE ROW LEVEL SECURITY;  
  
-- Create RLS policies  
CREATE POLICY "Materials are viewable by everyone"  
 ON public.materials\_metadata FOR SELECT  
 USING (true);  
  
CREATE POLICY "Materials are insertable by authenticated users"  
 ON public.materials\_metadata FOR INSERT  
 WITH CHECK (auth.role() = 'authenticated');  
  
CREATE POLICY "Materials are updatable by creators or admins"  
 ON public.materials\_metadata FOR UPDATE  
 USING (  
 created\_by = auth.uid() OR  
 EXISTS (  
 SELECT 1 FROM public.profiles  
 WHERE id = auth.uid() AND role = 'admin'  
 )  
 );

#### Setting Up MoodBoard Tables

Create tables for the MoodBoard feature:

-- MoodBoards table  
CREATE TABLE public.moodboards (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 name TEXT NOT NULL,  
 description TEXT,  
 user\_id UUID NOT NULL REFERENCES auth.users(id) ON DELETE CASCADE,  
 is\_public BOOLEAN NOT NULL DEFAULT false,  
 view\_preference TEXT NOT NULL DEFAULT 'grid',  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT now(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT now()  
);  
  
-- MoodBoardItems table  
CREATE TABLE public.moodboard\_items (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 board\_id UUID NOT NULL REFERENCES public.moodboards(id) ON DELETE CASCADE,  
 material\_id TEXT NOT NULL,  
 notes TEXT,  
 position INTEGER DEFAULT 0,  
 added\_at TIMESTAMP WITH TIME ZONE DEFAULT now()  
);  
  
-- Create indexes  
CREATE INDEX idx\_moodboards\_user\_id ON public.moodboards(user\_id);  
CREATE INDEX idx\_moodboard\_items\_board\_id ON public.moodboard\_items(board\_id);  
CREATE INDEX idx\_moodboard\_items\_material\_id ON public.moodboard\_items(material\_id);  
  
-- Enable Row Level Security  
ALTER TABLE public.moodboards ENABLE ROW LEVEL SECURITY;  
ALTER TABLE public.moodboard\_items ENABLE ROW LEVEL SECURITY;  
  
-- Policies for MoodBoards  
-- Users can view their own boards  
CREATE POLICY "Users can view their own boards"  
 ON public.moodboards  
 FOR SELECT  
 USING (user\_id = auth.uid());  
  
-- Users can view public boards  
CREATE POLICY "Users can view public boards"  
 ON public.moodboards  
 FOR SELECT  
 USING (is\_public = true);  
  
-- Users can create their own boards  
CREATE POLICY "Users can create their own boards"  
 ON public.moodboards  
 FOR INSERT  
 WITH CHECK (user\_id = auth.uid());  
  
-- Users can update their own boards  
CREATE POLICY "Users can update their own boards"  
 ON public.moodboards  
 FOR UPDATE  
 USING (user\_id = auth.uid());  
  
-- Users can delete their own boards  
CREATE POLICY "Users can delete their own boards"  
 ON public.moodboards  
 FOR DELETE  
 USING (user\_id = auth.uid());  
  
-- Policies for MoodBoardItems  
-- Users can view items in their own boards  
CREATE POLICY "Users can view items in their own boards"  
 ON public.moodboard\_items  
 FOR SELECT  
 USING (board\_id IN (SELECT id FROM public.moodboards WHERE user\_id = auth.uid()));  
  
-- Users can view items in public boards  
CREATE POLICY "Users can view items in public boards"  
 ON public.moodboard\_items  
 FOR SELECT  
 USING (board\_id IN (SELECT id FROM public.moodboards WHERE is\_public = true));  
  
-- Users can add items to their own boards  
CREATE POLICY "Users can add items to their own boards"  
 ON public.moodboard\_items  
 FOR INSERT  
 WITH CHECK (board\_id IN (SELECT id FROM public.moodboards WHERE user\_id = auth.uid()));  
  
-- Users can update items in their own boards  
CREATE POLICY "Users can update items in their own boards"  
 ON public.moodboard\_items  
 FOR UPDATE  
 USING (board\_id IN (SELECT id FROM public.moodboards WHERE user\_id = auth.uid()));  
  
-- Users can delete items from their own boards  
CREATE POLICY "Users can delete items from their own boards"  
 ON public.moodboard\_items  
 FOR DELETE  
 USING (board\_id IN (SELECT id FROM public.moodboards WHERE user\_id = auth.uid()));

-- MoodBoards table  
CREATE TABLE public.moodboards (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 name TEXT NOT NULL,  
 description TEXT,  
 user\_id UUID NOT NULL REFERENCES auth.users(id) ON DELETE CASCADE,  
 is\_public BOOLEAN NOT NULL DEFAULT false,  
 view\_preference TEXT NOT NULL DEFAULT 'grid',  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT now(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT now()  
);  
  
-- MoodBoardItems table  
CREATE TABLE public.moodboard\_items (  
 id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),  
 board\_id UUID NOT NULL REFERENCES public.moodboards(id) ON DELETE CASCADE,  
 material\_id TEXT NOT NULL,  
 notes TEXT,  
 position INTEGER DEFAULT 0,  
 added\_at TIMESTAMP WITH TIME ZONE DEFAULT now()  
);  
  
-- Create indexes  
CREATE INDEX idx\_moodboards\_user\_id ON public.moodboards(user\_id);  
CREATE INDEX idx\_moodboard\_items\_board\_id ON public.moodboard\_items(board\_id);  
CREATE INDEX idx\_moodboard\_items\_material\_id ON public.moodboard\_items(material\_id);  
  
-- Enable Row Level Security  
ALTER TABLE public.moodboards ENABLE ROW LEVEL SECURITY;  
ALTER TABLE public.moodboard\_items ENABLE ROW LEVEL SECURITY;  
  
-- Policies for MoodBoards  
-- Users can view their own boards  
CREATE POLICY "Users can view their own boards"  
 ON public.moodboards  
 FOR SELECT  
 USING (user\_id = auth.uid());  
  
-- Users can view public boards  
CREATE POLICY "Users can view public boards"  
 ON public.moodboards  
 FOR SELECT  
 USING (is\_public = true);  
  
-- Users can create their own boards  
CREATE POLICY "Users can create their own boards"  
 ON public.moodboards  
 FOR INSERT  
 WITH CHECK (user\_id = auth.uid());  
  
-- Users can update their own boards  
CREATE POLICY "Users can update their own boards"  
 ON public.moodboards  
 FOR UPDATE  
 USING (user\_id = auth.uid());  
  
-- Users can delete their own boards  
CREATE POLICY "Users can delete their own boards"  
 ON public.moodboards  
 FOR DELETE  
 USING (user\_id = auth.uid());  
  
-- Policies for MoodBoardItems  
-- Users can view items in their own boards  
CREATE POLICY "Users can view items in their own boards"  
 ON public.moodboard\_items  
 FOR SELECT  
 USING (board\_id IN (SELECT id FROM public.moodboards WHERE user\_id = auth.uid()));  
  
-- Users can view items in public boards  
CREATE POLICY "Users can view items in public boards"  
 ON public.moodboard\_items  
 FOR SELECT  
 USING (board\_id IN (SELECT id FROM public.moodboards WHERE is\_public = true));  
  
-- Users can add items to their own boards  
CREATE POLICY "Users can add items to their own boards"  
 ON public.moodboard\_items  
 FOR INSERT  
 WITH CHECK (board\_id IN (SELECT id FROM public.moodboards WHERE user\_id = auth.uid()));  
  
-- Users can update items in their own boards  
CREATE POLICY "Users can update items in their own boards"  
 ON public.moodboard\_items  
 FOR UPDATE  
 USING (board\_id IN (SELECT id FROM public.moodboards WHERE user\_id = auth.uid()));  
  
-- Users can delete items from their own boards  
CREATE POLICY "Users can delete items from their own boards"  
 ON public.moodboard\_items  
 FOR DELETE  
 USING (board\_id IN (SELECT id FROM public.moodboards WHERE user\_id = auth.uid()));

### Authentication Configuration

#### Step 1: Configure Auth Settings

1. Navigate to Authentication → Settings in the Supabase dashboard
2. Set the Site URL to your production frontend URL (e.g., https://kai.yourdomain.com)
3. Configure Redirect URLs:
4. https://kai.yourdomain.com/auth/callback
5. https://admin.kai.yourdomain.com/auth/callback
6. Enable the Confirm email option for added security

https://kai.yourdomain.com

https://kai.yourdomain.com/auth/callback

https://admin.kai.yourdomain.com/auth/callback

#### Step 2: Configure Auth Providers

1. Navigate to Authentication → Providers
2. Configure Email Auth:
3. Enable Email provider
4. Choose Secure email change and recovery flow
5. (Optional) Configure additional providers:
6. Google
7. GitHub
8. Microsoft
9. etc.

Choose Secure email change and recovery flow

(Optional) Configure additional providers:

For each provider, you'll need to:  
- Create OAuth applications in the respective developer portals  
- Configure redirect URIs  
- Add client IDs and secrets to Supabase

#### Step 3: Create Initial Admin User

1. Navigate to Authentication → Users
2. Click Add User
3. Enter admin user details:
4. Email: admin@yourdomain.com (use a real email)
5. Password: Generate a strong password
6. Create the user
7. Update their role in the profiles table:

admin@yourdomain.com

INSERT INTO public.profiles (id, email, role)  
VALUES (  
 'admin-user-uuid-from-auth-users-table',  
 'admin@yourdomain.com',  
 'admin'  
);

INSERT INTO public.profiles (id, email, role)  
VALUES (  
 'admin-user-uuid-from-auth-users-table',  
 'admin@yourdomain.com',  
 'admin'  
);

### Storage Buckets

Kai uses Supabase Storage for temporary file uploads and user avatars.

#### Step 1: Create Storage Buckets

1. Navigate to Storage in the Supabase dashboard
2. Create the following buckets:
3. avatars - For user profile pictures
4. temp-uploads - For temporary file uploads before processing
5. material-thumbnails - For material thumbnail images

avatars

temp-uploads

material-thumbnails

#### Step 2: Configure Storage Permissions

For each bucket, configure security rules:

Avatars Bucket:  
1. Go to Storage → avatars → Policies  
2. Create the following policies:

avatars

-- Allow anyone to view avatars  
CREATE POLICY "Avatars are publicly accessible"  
ON storage.objects FOR SELECT  
USING (bucket\_id = 'avatars');  
  
-- Allow authenticated users to upload their own avatar  
CREATE POLICY "Users can upload their own avatar"  
ON storage.objects FOR INSERT  
WITH CHECK (  
 bucket\_id = 'avatars' AND  
 auth.uid() = SUBSTRING(name, 1, POSITION('/' IN name) - 1)::uuid  
);  
  
-- Allow users to update their own avatar  
CREATE POLICY "Users can update their own avatar"  
ON storage.objects FOR UPDATE  
USING (  
 bucket\_id = 'avatars' AND  
 auth.uid() = SUBSTRING(name, 1, POSITION('/' IN name) - 1)::uuid  
);  
  
-- Allow users to delete their own avatar  
CREATE POLICY "Users can delete their own avatar"  
ON storage.objects FOR DELETE  
USING (  
 bucket\_id = 'avatars' AND  
 auth.uid() = SUBSTRING(name, 1, POSITION('/' IN name) - 1)::uuid  
);

-- Allow anyone to view avatars  
CREATE POLICY "Avatars are publicly accessible"  
ON storage.objects FOR SELECT  
USING (bucket\_id = 'avatars');  
  
-- Allow authenticated users to upload their own avatar  
CREATE POLICY "Users can upload their own avatar"  
ON storage.objects FOR INSERT  
WITH CHECK (  
 bucket\_id = 'avatars' AND  
 auth.uid() = SUBSTRING(name, 1, POSITION('/' IN name) - 1)::uuid  
);  
  
-- Allow users to update their own avatar  
CREATE POLICY "Users can update their own avatar"  
ON storage.objects FOR UPDATE  
USING (  
 bucket\_id = 'avatars' AND  
 auth.uid() = SUBSTRING(name, 1, POSITION('/' IN name) - 1)::uuid  
);  
  
-- Allow users to delete their own avatar  
CREATE POLICY "Users can delete their own avatar"  
ON storage.objects FOR DELETE  
USING (  
 bucket\_id = 'avatars' AND  
 auth.uid() = SUBSTRING(name, 1, POSITION('/' IN name) - 1)::uuid  
);

Temp Uploads Bucket:

-- Allow authenticated users to upload files  
CREATE POLICY "Authenticated users can upload temp files"  
ON storage.objects FOR INSERT  
WITH CHECK (  
 bucket\_id = 'temp-uploads' AND  
 auth.role() = 'authenticated'  
);  
  
-- Allow users to access their own uploads  
CREATE POLICY "Users can access their own temp uploads"  
ON storage.objects FOR SELECT  
USING (  
 bucket\_id = 'temp-uploads' AND  
 auth.uid()::text = SUBSTRING(name, 1, POSITION('/' IN name) - 1)  
);  
  
-- Files automatically expire after 24 hours (set up a cron job)

-- Allow authenticated users to upload files  
CREATE POLICY "Authenticated users can upload temp files"  
ON storage.objects FOR INSERT  
WITH CHECK (  
 bucket\_id = 'temp-uploads' AND  
 auth.role() = 'authenticated'  
);  
  
-- Allow users to access their own uploads  
CREATE POLICY "Users can access their own temp uploads"  
ON storage.objects FOR SELECT  
USING (  
 bucket\_id = 'temp-uploads' AND  
 auth.uid()::text = SUBSTRING(name, 1, POSITION('/' IN name) - 1)  
);  
  
-- Files automatically expire after 24 hours (set up a cron job)

Material Thumbnails Bucket:

-- Allow anyone to view material thumbnails  
CREATE POLICY "Material thumbnails are publicly accessible"  
ON storage.objects FOR SELECT  
USING (bucket\_id = 'material-thumbnails');  
  
-- Allow authenticated users to upload material thumbnails  
CREATE POLICY "Authenticated users can upload material thumbnails"  
ON storage.objects FOR INSERT  
WITH CHECK (  
 bucket\_id = 'material-thumbnails' AND  
 auth.role() = 'authenticated'  
);  
  
-- Only admins can delete material thumbnails  
CREATE POLICY "Only admins can delete material thumbnails"  
ON storage.objects FOR DELETE  
USING (  
 bucket\_id = 'material-thumbnails' AND  
 EXISTS (  
 SELECT 1 FROM public.profiles  
 WHERE id = auth.uid() AND role = 'admin'  
 )  
);

-- Allow anyone to view material thumbnails  
CREATE POLICY "Material thumbnails are publicly accessible"  
ON storage.objects FOR SELECT  
USING (bucket\_id = 'material-thumbnails');  
  
-- Allow authenticated users to upload material thumbnails  
CREATE POLICY "Authenticated users can upload material thumbnails"  
ON storage.objects FOR INSERT  
WITH CHECK (  
 bucket\_id = 'material-thumbnails' AND  
 auth.role() = 'authenticated'  
);  
  
-- Only admins can delete material thumbnails  
CREATE POLICY "Only admins can delete material thumbnails"  
ON storage.objects FOR DELETE  
USING (  
 bucket\_id = 'material-thumbnails' AND  
 EXISTS (  
 SELECT 1 FROM public.profiles  
 WHERE id = auth.uid() AND role = 'admin'  
 )  
);

### Realtime Configuration

Kai uses Supabase Realtime for the queue system and real-time updates.

#### Step 1: Enable Realtime

1. Navigate to Database → Replication in the Supabase dashboard
2. Create a publication for the queue tables:

BEGIN;  
 -- Drop existing publication if it exists  
 DROP PUBLICATION IF EXISTS supabase\_realtime;  
  
 -- Create publication for realtime tables  
 CREATE PUBLICATION supabase\_realtime FOR TABLE  
 public.queue\_jobs,  
 public.job\_dependencies,  
 public.materials\_metadata;  
COMMIT;

BEGIN;  
 -- Drop existing publication if it exists  
 DROP PUBLICATION IF EXISTS supabase\_realtime;  
  
 -- Create publication for realtime tables  
 CREATE PUBLICATION supabase\_realtime FOR TABLE  
 public.queue\_jobs,  
 public.job\_dependencies,  
 public.materials\_metadata;  
COMMIT;

#### Step 2: Configure Client for Realtime

In your frontend application, configure the Supabase client to use Realtime:

// In packages/client/src/services/supabaseClient.ts  
import { createClient } from '@supabase/supabase-js'  
  
const supabaseUrl = process.env.NEXT\_PUBLIC\_SUPABASE\_URL!  
const supabaseAnonKey = process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY!  
  
export const supabase = createClient(supabaseUrl, supabaseAnonKey, {  
 realtime: {  
 params: {  
 eventsPerSecond: 10  
 }  
 }  
})  
  
// Subscribe to queue updates  
export const subscribeToQueue = (callback: (payload: any) => void) => {  
 const subscription = supabase  
 .channel('queue\_updates')  
 .on(  
 'postgres\_changes',  
 {  
 event: '\*',  
 schema: 'public',  
 table: 'queue\_jobs'  
 },  
 callback  
 )  
 .subscribe()  
  
 return subscription  
}

// In packages/client/src/services/supabaseClient.ts  
import { createClient } from '@supabase/supabase-js'  
  
const supabaseUrl = process.env.NEXT\_PUBLIC\_SUPABASE\_URL!  
const supabaseAnonKey = process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY!  
  
export const supabase = createClient(supabaseUrl, supabaseAnonKey, {  
 realtime: {  
 params: {  
 eventsPerSecond: 10  
 }  
 }  
})  
  
// Subscribe to queue updates  
export const subscribeToQueue = (callback: (payload: any) => void) => {  
 const subscription = supabase  
 .channel('queue\_updates')  
 .on(  
 'postgres\_changes',  
 {  
 event: '\*',  
 schema: 'public',  
 table: 'queue\_jobs'  
 },  
 callback  
 )  
 .subscribe()  
  
 return subscription  
}

### API Keys and Security

#### Step 1: Retrieve API Keys

1. Navigate to Settings → API in the Supabase dashboard
2. Copy the following values:
3. Project URL - The URL of your Supabase project
4. anon public - Public API key for client-side authentication
5. service\_role - Admin API key for server-side operations

#### Step 2: Store API Keys Securely

Add these keys to:

1. GitHub Secrets for CI/CD:
2. SUPABASE\_URL
3. SUPABASE\_KEY (service\_role key)
4. SUPABASE\_ANON\_KEY (anon public key)
5. Vercel Environment Variables:
6. For Admin Panel (Next.js):  
     
   NEXT\_PUBLIC\_SUPABASE\_URL  
   NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY
7. For Client App (Gatsby):  
     
   GATSBY\_SUPABASE\_URL  
   GATSBY\_SUPABASE\_ANON\_KEY
8. Kubernetes Secrets:  
    bash  
    kubectl create secret generic kai-secrets -n kai \  
    --from-literal=supabase-url="https://your-project.supabase.co" \  
    --from-literal=supabase-key="your-service-role-key"

SUPABASE\_URL

SUPABASE\_KEY

SUPABASE\_ANON\_KEY (anon public key)

SUPABASE\_ANON\_KEY

Vercel Environment Variables:

For Admin Panel (Next.js):

* NEXT\_PUBLIC\_SUPABASE\_URL
* NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY

NEXT\_PUBLIC\_SUPABASE\_URL

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY

For Client App (Gatsby):

* GATSBY\_SUPABASE\_URL
* GATSBY\_SUPABASE\_ANON\_KEY

GATSBY\_SUPABASE\_URL

GATSBY\_SUPABASE\_ANON\_KEY

Kubernetes Secrets:  
 bash  
 kubectl create secret generic kai-secrets -n kai \  
 --from-literal=supabase-url="https://your-project.supabase.co" \  
 --from-literal=supabase-key="your-service-role-key"

bash  
 kubectl create secret generic kai-secrets -n kai \  
 --from-literal=supabase-url="https://your-project.supabase.co" \  
 --from-literal=supabase-key="your-service-role-key"

#### Step 3: Configure API Security Settings

1. Navigate to Settings → API → API Settings
2. Configure JWT expiry time (default is 3600 seconds)
3. Enable JWT autorefresh

### Monitoring and Maintenance

#### Database Maintenance

1. Regular Backups:
2. Supabase automatically creates daily backups
3. For additional safety, periodically export your data:  
     
   Navigate to Database → Backups  
   Click Create a backup
4. Performance Monitoring:
5. Navigate to Database → Performance
6. Monitor query performance and optimize slow queries

For additional safety, periodically export your data:

* Navigate to Database → Backups
* Click Create a backup

Performance Monitoring:

#### Storage Maintenance

1. Set up a cleanup function to remove expired temporary files:

-- Create a function to clean up expired temp files  
CREATE OR REPLACE FUNCTION cleanup\_expired\_temp\_files()  
RETURNS void AS $$  
DECLARE  
 file\_record RECORD;  
 now\_timestamp TIMESTAMP := NOW();  
BEGIN  
 FOR file\_record IN  
 SELECT \*  
 FROM storage.objects  
 WHERE  
 bucket\_id = 'temp-uploads' AND  
 created\_at < (now\_timestamp - INTERVAL '1 day')  
 LOOP  
 PERFORM storage.delete\_object(file\_record.bucket\_id, file\_record.name);  
 END LOOP;  
END;  
$$ LANGUAGE plpgsql SECURITY DEFINER;  
  
-- Create a cron job to run this function daily  
SELECT cron.schedule(  
 'cleanup-temp-files',  
 '0 0 \* \* \*', -- Run at midnight every day  
 'SELECT cleanup\_expired\_temp\_files();'  
);

-- Create a function to clean up expired temp files  
CREATE OR REPLACE FUNCTION cleanup\_expired\_temp\_files()  
RETURNS void AS $$  
DECLARE  
 file\_record RECORD;  
 now\_timestamp TIMESTAMP := NOW();  
BEGIN  
 FOR file\_record IN  
 SELECT \*  
 FROM storage.objects  
 WHERE  
 bucket\_id = 'temp-uploads' AND  
 created\_at < (now\_timestamp - INTERVAL '1 day')  
 LOOP  
 PERFORM storage.delete\_object(file\_record.bucket\_id, file\_record.name);  
 END LOOP;  
END;  
$$ LANGUAGE plpgsql SECURITY DEFINER;  
  
-- Create a cron job to run this function daily  
SELECT cron.schedule(  
 'cleanup-temp-files',  
 '0 0 \* \* \*', -- Run at midnight every day  
 'SELECT cleanup\_expired\_temp\_files();'  
);

#### System Monitoring

1. Set up Alerts:
2. Navigate to Database → Database Settings → Pooler Settings
3. Configure connection pool size based on your application needs
4. Set up email notifications for quota usage
5. Usage Monitoring:
6. Navigate to Reports
7. Monitor API usage, storage consumption, and database performance

Set up email notifications for quota usage

Usage Monitoring:

### Conclusion

This guide has walked you through setting up Supabase for the Kai application. Your Supabase project is now configured with:

* Authentication and user management
* Database tables for the queue system and materials metadata
* Storage buckets with appropriate security policies
* Realtime features for live updates
* API keys and security settings

Remember to regularly monitor your Supabase project for performance issues, storage usage, and security concerns. As your application grows, you may need to adjust database indexes, connection pool settings, and security policies.

# Supabase

Source: readme/supabase.md

---

## Supabase Integration

The Kai platform leverages Supabase as its primary database and backend infrastructure, providing a robust foundation for data storage, real-time functionality, authentication, and vector search capabilities. This document details how Supabase is integrated throughout the system, its benefits, and implementation details.

### Overview and Benefits

#### Why Supabase?

Supabase provides several key advantages for the Kai platform:

1. PostgreSQL Foundation
2. Enterprise-grade relational database
3. Rich ecosystem of extensions
4. Powerful query capabilities
5. Transaction support
6. Role-based security
7. Vector Search Capabilities
8. pgvector extension for similarity search
9. Efficient vector indexing
10. Multiple distance metrics
11. Hybrid search support
12. Performance-optimized queries
13. Real-time Functionality
14. WebSocket-based data synchronization
15. Event-driven architecture
16. Pub/sub messaging patterns
17. Status monitoring
18. Client-side subscriptions
19. Authentication and Authorization
20. JWT-based authentication
21. Role-based access control
22. Row-level security policies
23. OAuth provider integration
24. Secure password handling
25. Storage Solutions
26. Managed file storage
27. Access control policies
28. Image transformations
29. Content delivery optimization
30. Secure direct uploads

Role-based security

Vector Search Capabilities

Performance-optimized queries

Real-time Functionality

Client-side subscriptions

Authentication and Authorization

Secure password handling

Storage Solutions

#### Integration Architecture

Supabase serves as the core data platform with these integration points:

1. Data Layer
2. Material data storage
3. Dataset management
4. User and permission storage
5. File metadata tracking
6. Configuration storage
7. Search Infrastructure
8. Vector embeddings storage
9. Semantic search functionality
10. Full-text search capabilities
11. Hybrid search algorithms
12. Search relevance optimization
13. Real-time Communication
14. Queue status updates
15. Training progress tracking
16. Admin dashboard updates
17. System event propagation
18. Client-side state synchronization
19. Security Layer
20. User authentication
21. API access control
22. Data access permissions
23. Credential management
24. Row-level security
25. Storage Integration
26. Dataset file storage
27. Material image storage
28. Extracted content storage
29. Temporary file handling
30. Backup management

Configuration storage

Search Infrastructure

Search relevance optimization

Real-time Communication

Client-side state synchronization

Security Layer

Row-level security

Storage Integration

### Technical Implementation

#### Client Management

The system implements a singleton Supabase client manager:

/\*\*  
 \* Supabase Client Manager  
 \* Handles the configuration and initialization of the Supabase client  
 \*/  
class SupabaseClientManager {  
 private config: SupabaseConfig;  
 private client: SupabaseClient | null = null;  
 private initialized = false;  
  
 /\*\*  
 \* Initialize the Supabase client with configuration  
 \* @param config Supabase configuration  
 \*/  
 init(config: SupabaseConfig): void {  
 if (this.initialized) {  
 logger.warn('Supabase client already initialized');  
 return;  
 }  
  
 this.config = config;  
  
 // Skip client creation until actually needed  
 this.initialized = true;  
  
 logger.info('Supabase client configuration initialized');  
 }  
  
 /\*\*  
 \* Get the Supabase client instance  
 \* Initializes the client if it doesn't exist  
 \* @returns Supabase client  
 \*/  
 getClient(): SupabaseClient {  
 if (!this.client) {  
 if (!this.config.url || !this.config.key) {  
 throw new Error('Supabase URL and key are required. Call init() first or set SUPABASE\_URL and SUPABASE\_KEY environment variables.');  
 }  
  
 // Create the client using the Supabase SDK  
 this.client = createClient(this.config.url, this.config.key);  
  
 logger.info('Supabase client initialized');  
 }  
  
 return this.client;  
 }  
  
 /\*\*  
 \* Check if the Supabase client is initialized  
 \* @returns True if initialized  
 \*/  
 isInitialized(): boolean {  
 return this.initialized;  
 }  
  
 /\*\*  
 \* Reset the Supabase client  
 \* Useful for testing or reconfiguration  
 \*/  
 reset(): void {  
 this.client = null;  
 this.initialized = false;  
  
 logger.info('Supabase client reset');  
 }  
}  
  
// Export a singleton instance  
export const supabaseClient = new SupabaseClientManager();

/\*\*  
 \* Supabase Client Manager  
 \* Handles the configuration and initialization of the Supabase client  
 \*/  
class SupabaseClientManager {  
 private config: SupabaseConfig;  
 private client: SupabaseClient | null = null;  
 private initialized = false;  
  
 /\*\*  
 \* Initialize the Supabase client with configuration  
 \* @param config Supabase configuration  
 \*/  
 init(config: SupabaseConfig): void {  
 if (this.initialized) {  
 logger.warn('Supabase client already initialized');  
 return;  
 }  
  
 this.config = config;  
  
 // Skip client creation until actually needed  
 this.initialized = true;  
  
 logger.info('Supabase client configuration initialized');  
 }  
  
 /\*\*  
 \* Get the Supabase client instance  
 \* Initializes the client if it doesn't exist  
 \* @returns Supabase client  
 \*/  
 getClient(): SupabaseClient {  
 if (!this.client) {  
 if (!this.config.url || !this.config.key) {  
 throw new Error('Supabase URL and key are required. Call init() first or set SUPABASE\_URL and SUPABASE\_KEY environment variables.');  
 }  
  
 // Create the client using the Supabase SDK  
 this.client = createClient(this.config.url, this.config.key);  
  
 logger.info('Supabase client initialized');  
 }  
  
 return this.client;  
 }  
  
 /\*\*  
 \* Check if the Supabase client is initialized  
 \* @returns True if initialized  
 \*/  
 isInitialized(): boolean {  
 return this.initialized;  
 }  
  
 /\*\*  
 \* Reset the Supabase client  
 \* Useful for testing or reconfiguration  
 \*/  
 reset(): void {  
 this.client = null;  
 this.initialized = false;  
  
 logger.info('Supabase client reset');  
 }  
}  
  
// Export a singleton instance  
export const supabaseClient = new SupabaseClientManager();

#### Database Schema

The system uses SQL migrations to define the Supabase PostgreSQL schema:

-- Enable the pgvector extension for vector similarity search  
CREATE EXTENSION IF NOT EXISTS pgvector;  
  
-- Materials table with full-text search and vector support  
CREATE TABLE IF NOT EXISTS materials (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 name VARCHAR(255) NOT NULL,  
 description TEXT,  
 material\_type VARCHAR(100),  
 manufacturer VARCHAR(255),  
 product\_code VARCHAR(100),  
 dimensions JSONB,  
 color JSONB,  
 finish VARCHAR(100),  
 tags TEXT[],  
 metadata JSONB,  
 created\_by UUID REFERENCES users(id),  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
  
 -- Generated column for full-text search  
 search\_text TSVECTOR GENERATED ALWAYS AS (  
 to\_tsvector('english', coalesce(name, '') || ' ' || coalesce(description, '') || ' ' ||  
 coalesce(type, '') || ' ' || coalesce(array\_to\_string(tags, ' '), ''))  
 ) STORED  
);  
  
-- Vector embeddings table for similarity search  
CREATE TABLE IF NOT EXISTS vector\_embeddings (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 material\_id UUID REFERENCES materials(id) ON DELETE CASCADE,  
 embedding vector(384), -- Adjust dimension as needed  
 metadata JSONB,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Create an index for cosine distance similarity search  
CREATE INDEX IF NOT EXISTS vector\_embeddings\_embedding\_idx   
ON vector\_embeddings USING ivfflat (embedding vector\_cosine\_ops)   
WITH (lists = 100);  
  
-- Create an index for material ID lookups  
CREATE INDEX IF NOT EXISTS vector\_embeddings\_material\_id\_idx   
ON vector\_embeddings (material\_id);  
  
-- Function to find similar materials by vector embedding  
CREATE OR REPLACE FUNCTION find\_similar\_materials(  
 query\_embedding vector,  
 match\_threshold float,  
 match\_count int  
)  
RETURNS TABLE (  
 id UUID,  
 similarity float  
)  
LANGUAGE SQL  
AS $$  
SELECT  
 m.id,  
 1 - (ve.embedding <=> query\_embedding) AS similarity  
FROM  
 vector\_embeddings ve  
JOIN  
 materials m ON ve.material\_id = m.id  
WHERE  
 1 - (ve.embedding <=> query\_embedding) > match\_threshold  
ORDER BY  
 similarity DESC  
LIMIT  
 match\_count;  
$$;  
  
-- Create row-level security policies  
CREATE POLICY "Allow authenticated users to read materials"   
ON materials FOR SELECT   
USING (auth.role() = 'authenticated');  
  
CREATE POLICY "Allow admins full access to materials"   
ON materials   
USING (auth.role() = 'authenticated' AND auth.jwt() ->> 'role' = 'admin');

-- Enable the pgvector extension for vector similarity search  
CREATE EXTENSION IF NOT EXISTS pgvector;  
  
-- Materials table with full-text search and vector support  
CREATE TABLE IF NOT EXISTS materials (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 name VARCHAR(255) NOT NULL,  
 description TEXT,  
 material\_type VARCHAR(100),  
 manufacturer VARCHAR(255),  
 product\_code VARCHAR(100),  
 dimensions JSONB,  
 color JSONB,  
 finish VARCHAR(100),  
 tags TEXT[],  
 metadata JSONB,  
 created\_by UUID REFERENCES users(id),  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
  
 -- Generated column for full-text search  
 search\_text TSVECTOR GENERATED ALWAYS AS (  
 to\_tsvector('english', coalesce(name, '') || ' ' || coalesce(description, '') || ' ' ||  
 coalesce(type, '') || ' ' || coalesce(array\_to\_string(tags, ' '), ''))  
 ) STORED  
);  
  
-- Vector embeddings table for similarity search  
CREATE TABLE IF NOT EXISTS vector\_embeddings (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 material\_id UUID REFERENCES materials(id) ON DELETE CASCADE,  
 embedding vector(384), -- Adjust dimension as needed  
 metadata JSONB,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Create an index for cosine distance similarity search  
CREATE INDEX IF NOT EXISTS vector\_embeddings\_embedding\_idx   
ON vector\_embeddings USING ivfflat (embedding vector\_cosine\_ops)   
WITH (lists = 100);  
  
-- Create an index for material ID lookups  
CREATE INDEX IF NOT EXISTS vector\_embeddings\_material\_id\_idx   
ON vector\_embeddings (material\_id);  
  
-- Function to find similar materials by vector embedding  
CREATE OR REPLACE FUNCTION find\_similar\_materials(  
 query\_embedding vector,  
 match\_threshold float,  
 match\_count int  
)  
RETURNS TABLE (  
 id UUID,  
 similarity float  
)  
LANGUAGE SQL  
AS $$  
SELECT  
 m.id,  
 1 - (ve.embedding <=> query\_embedding) AS similarity  
FROM  
 vector\_embeddings ve  
JOIN  
 materials m ON ve.material\_id = m.id  
WHERE  
 1 - (ve.embedding <=> query\_embedding) > match\_threshold  
ORDER BY  
 similarity DESC  
LIMIT  
 match\_count;  
$$;  
  
-- Create row-level security policies  
CREATE POLICY "Allow authenticated users to read materials"   
ON materials FOR SELECT   
USING (auth.role() = 'authenticated');  
  
CREATE POLICY "Allow admins full access to materials"   
ON materials   
USING (auth.role() = 'authenticated' AND auth.jwt() ->> 'role' = 'admin');

#### Vector Search Integration

The system implements vector similarity search using pgvector:

/\*\*  
 \* Vector Search Service for Supabase  
 \* Provides methods for storing and querying vector embeddings  
 \*/  
export class SupabaseVectorSearch {  
 /\*\*  
 \* Find similar vectors to the provided embedding  
 \*  
 \* @param embedding The query vector to find similar vectors for  
 \* @param tableName The table containing vector embeddings  
 \* @param vectorColumn The column name for the vector data (default: 'embedding')  
 \* @param config Search configuration options  
 \* @returns Array of matching records with similarity scores  
 \*/  
 async findSimilar(  
 embedding: number[],  
 tableName: string,  
 vectorColumn: string = 'embedding',  
 config: VectorSearchConfig = {}  
 ): Promise<Array<Record<string, any>>> {  
 try {  
 // Initialize query with proper client  
 const client = supabaseClient.getClient();  
  
 const limit = config.limit || 10;  
 const threshold = config.threshold || 0.75;  
  
 // Use vector operators in PostgreSQL query  
 const { data, error } = await client  
 .from(tableName)  
 .select(`  
 \*,  
 similarity:1 - (${vectorColumn} <=> ${'embedding'})  
 `)  
 .gte('similarity', threshold)  
 .order('similarity', { ascending: false })  
 .limit(limit);  
  
 if (error) throw error;  
  
 return data || [];  
  
 } catch (error) {  
 logger.error(`Vector search error: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Store a vector embedding in Supabase  
 \*  
 \* @param embedding The vector to store  
 \* @param metadata Additional metadata to store with the vector  
 \* @param tableName The table to store the vector in  
 \* @param vectorColumn The column name for the vector data (default: 'embedding')  
 \* @returns The created record ID  
 \*/  
 async storeVector(  
 embedding: number[],  
 metadata: Record<string, any>,  
 tableName: string,  
 vectorColumn: string = 'embedding'  
 ): Promise<string> {  
 try {  
 const record = {  
 ...metadata,  
 [vectorColumn]: embedding,  
 created\_at: new Date().toISOString()  
 };  
  
 // Insert the data using proper client  
 const client = supabaseClient.getClient();  
  
 const { data, error } = await client  
 .from(tableName)  
 .insert(record)  
 .select('id')  
 .single();  
  
 if (error) throw error;  
  
 return data.id;  
  
 } catch (error) {  
 logger.error(`Vector storage error: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Create a vector index for faster similarity searches  
 \*  
 \* @param tableName The table to create the index for  
 \* @param vectorColumn The column name for the vector data  
 \* @param indexMethod The index method to use ('ivfflat', 'hnsw')  
 \* @param dimensions Optional vector dimensions (default: 1536 for typical embeddings)  
 \* @returns True if the index was created successfully  
 \*/  
 async createIndex(  
 tableName: string,  
 vectorColumn: string = 'embedding',  
 indexMethod: 'ivfflat' | 'hnsw' = 'hnsw',  
 dimensions: number = 1536  
 ): Promise<boolean> {  
 try {  
 // For some operations, we need to use raw SQL to create indices  
 const indexName = `idx\_${tableName}\_${vectorColumn}\_${indexMethod}`;  
  
 // Note: The actual SQL execution will depend on whether Supabase allows raw SQL  
 // This might need to be done through database migrations or admin console  
 const sql = indexMethod === 'hnsw'  
 ? `CREATE INDEX IF NOT EXISTS ${indexName} ON ${tableName} USING hnsw (${vectorColumn} vector\_l2\_ops) WITH (dims=${dimensions});`  
 : `CREATE INDEX IF NOT EXISTS ${indexName} ON ${tableName} USING ivfflat (${vectorColumn} vector\_l2\_ops) WITH (lists=100);`;  
  
 // Execute SQL to create index  
 // Implementation details may vary based on Supabase access level  
  
 return true;  
 } catch (error) {  
 logger.error(`Error creating vector index: ${error}`);  
 throw error;  
 }  
 }  
}  
  
// Export singleton instance  
export const vectorSearch = new SupabaseVectorSearch();

/\*\*  
 \* Vector Search Service for Supabase  
 \* Provides methods for storing and querying vector embeddings  
 \*/  
export class SupabaseVectorSearch {  
 /\*\*  
 \* Find similar vectors to the provided embedding  
 \*  
 \* @param embedding The query vector to find similar vectors for  
 \* @param tableName The table containing vector embeddings  
 \* @param vectorColumn The column name for the vector data (default: 'embedding')  
 \* @param config Search configuration options  
 \* @returns Array of matching records with similarity scores  
 \*/  
 async findSimilar(  
 embedding: number[],  
 tableName: string,  
 vectorColumn: string = 'embedding',  
 config: VectorSearchConfig = {}  
 ): Promise<Array<Record<string, any>>> {  
 try {  
 // Initialize query with proper client  
 const client = supabaseClient.getClient();  
  
 const limit = config.limit || 10;  
 const threshold = config.threshold || 0.75;  
  
 // Use vector operators in PostgreSQL query  
 const { data, error } = await client  
 .from(tableName)  
 .select(`  
 \*,  
 similarity:1 - (${vectorColumn} <=> ${'embedding'})  
 `)  
 .gte('similarity', threshold)  
 .order('similarity', { ascending: false })  
 .limit(limit);  
  
 if (error) throw error;  
  
 return data || [];  
  
 } catch (error) {  
 logger.error(`Vector search error: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Store a vector embedding in Supabase  
 \*  
 \* @param embedding The vector to store  
 \* @param metadata Additional metadata to store with the vector  
 \* @param tableName The table to store the vector in  
 \* @param vectorColumn The column name for the vector data (default: 'embedding')  
 \* @returns The created record ID  
 \*/  
 async storeVector(  
 embedding: number[],  
 metadata: Record<string, any>,  
 tableName: string,  
 vectorColumn: string = 'embedding'  
 ): Promise<string> {  
 try {  
 const record = {  
 ...metadata,  
 [vectorColumn]: embedding,  
 created\_at: new Date().toISOString()  
 };  
  
 // Insert the data using proper client  
 const client = supabaseClient.getClient();  
  
 const { data, error } = await client  
 .from(tableName)  
 .insert(record)  
 .select('id')  
 .single();  
  
 if (error) throw error;  
  
 return data.id;  
  
 } catch (error) {  
 logger.error(`Vector storage error: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Create a vector index for faster similarity searches  
 \*  
 \* @param tableName The table to create the index for  
 \* @param vectorColumn The column name for the vector data  
 \* @param indexMethod The index method to use ('ivfflat', 'hnsw')  
 \* @param dimensions Optional vector dimensions (default: 1536 for typical embeddings)  
 \* @returns True if the index was created successfully  
 \*/  
 async createIndex(  
 tableName: string,  
 vectorColumn: string = 'embedding',  
 indexMethod: 'ivfflat' | 'hnsw' = 'hnsw',  
 dimensions: number = 1536  
 ): Promise<boolean> {  
 try {  
 // For some operations, we need to use raw SQL to create indices  
 const indexName = `idx\_${tableName}\_${vectorColumn}\_${indexMethod}`;  
  
 // Note: The actual SQL execution will depend on whether Supabase allows raw SQL  
 // This might need to be done through database migrations or admin console  
 const sql = indexMethod === 'hnsw'  
 ? `CREATE INDEX IF NOT EXISTS ${indexName} ON ${tableName} USING hnsw (${vectorColumn} vector\_l2\_ops) WITH (dims=${dimensions});`  
 : `CREATE INDEX IF NOT EXISTS ${indexName} ON ${tableName} USING ivfflat (${vectorColumn} vector\_l2\_ops) WITH (lists=100);`;  
  
 // Execute SQL to create index  
 // Implementation details may vary based on Supabase access level  
  
 return true;  
 } catch (error) {  
 logger.error(`Error creating vector index: ${error}`);  
 throw error;  
 }  
 }  
}  
  
// Export singleton instance  
export const vectorSearch = new SupabaseVectorSearch();

#### Hybrid Search Implementation

The system combines vector similarity and full-text search:

/\*\*  
 \* Hybrid Search Service for Supabase  
 \* Combines full-text search and vector similarity search for better results  
 \*/  
export class SupabaseHybridSearch {  
 private vectorSearch: SupabaseVectorSearch;  
  
 constructor() {  
 this.vectorSearch = new SupabaseVectorSearch();  
 }  
  
 /\*\*  
 \* Perform hybrid search on materials  
 \*  
 \* @param textQuery Text search query  
 \* @param embedding Vector embedding for similarity comparison  
 \* @param options Hybrid search options  
 \* @returns Array of materials with text, vector, and combined scores  
 \*/  
 async searchMaterials(  
 textQuery: string,  
 embedding: number[],  
 options: HybridSearchOptions = {}  
 ): Promise<Array<HybridSearchResult>> {  
 try {  
 // Get Supabase client  
 const client = supabaseClient.getClient();  
  
 // Configure search parameters  
 const searchParams = {  
 textWeight: options.textWeight ?? 0.5,  
 vectorWeight: options.vectorWeight ?? 0.5,  
 limit: options.limit ?? 10,  
 threshold: options.threshold ?? 0.3  
 };  
  
 // Call custom PostgreSQL function for hybrid search  
 const { data, error } = await client.rpc('hybrid\_search\_materials', {  
 query\_text: textQuery,  
 query\_embedding: embedding,  
 text\_weight: searchParams.textWeight,  
 vector\_weight: searchParams.vectorWeight,  
 match\_count: searchParams.limit,  
 score\_threshold: searchParams.threshold  
 });  
  
 if (error) throw error;  
  
 // Map results to application format  
 return data.map((result: any) => ({  
 id: result.id,  
 name: result.name,  
 description: result.description,  
 materialType: result.material\_type,  
 scores: {  
 textScore: result.text\_score,  
 vectorScore: result.vector\_score,  
 combinedScore: result.combined\_score  
 },  
 // Other properties...  
 }));  
 } catch (error) {  
 logger.error(`Hybrid search failed: ${error}`);  
 throw error;  
 }  
 }  
}  
  
// Export singleton instance  
export const hybridSearch = new SupabaseHybridSearch();

/\*\*  
 \* Hybrid Search Service for Supabase  
 \* Combines full-text search and vector similarity search for better results  
 \*/  
export class SupabaseHybridSearch {  
 private vectorSearch: SupabaseVectorSearch;  
  
 constructor() {  
 this.vectorSearch = new SupabaseVectorSearch();  
 }  
  
 /\*\*  
 \* Perform hybrid search on materials  
 \*  
 \* @param textQuery Text search query  
 \* @param embedding Vector embedding for similarity comparison  
 \* @param options Hybrid search options  
 \* @returns Array of materials with text, vector, and combined scores  
 \*/  
 async searchMaterials(  
 textQuery: string,  
 embedding: number[],  
 options: HybridSearchOptions = {}  
 ): Promise<Array<HybridSearchResult>> {  
 try {  
 // Get Supabase client  
 const client = supabaseClient.getClient();  
  
 // Configure search parameters  
 const searchParams = {  
 textWeight: options.textWeight ?? 0.5,  
 vectorWeight: options.vectorWeight ?? 0.5,  
 limit: options.limit ?? 10,  
 threshold: options.threshold ?? 0.3  
 };  
  
 // Call custom PostgreSQL function for hybrid search  
 const { data, error } = await client.rpc('hybrid\_search\_materials', {  
 query\_text: textQuery,  
 query\_embedding: embedding,  
 text\_weight: searchParams.textWeight,  
 vector\_weight: searchParams.vectorWeight,  
 match\_count: searchParams.limit,  
 score\_threshold: searchParams.threshold  
 });  
  
 if (error) throw error;  
  
 // Map results to application format  
 return data.map((result: any) => ({  
 id: result.id,  
 name: result.name,  
 description: result.description,  
 materialType: result.material\_type,  
 scores: {  
 textScore: result.text\_score,  
 vectorScore: result.vector\_score,  
 combinedScore: result.combined\_score  
 },  
 // Other properties...  
 }));  
 } catch (error) {  
 logger.error(`Hybrid search failed: ${error}`);  
 throw error;  
 }  
 }  
}  
  
// Export singleton instance  
export const hybridSearch = new SupabaseHybridSearch();

#### Real-time Messaging Integration

The system leverages Supabase Realtime for messaging:

/\*\*  
 \* Supabase Message Broker  
 \*  
 \* Provides a pub/sub system for messaging between services using Supabase Realtime  
 \*/  
export class SupabaseMessageBroker implements MessageBroker {  
 private channels: Map<string, RealtimeChannel> = new Map();  
 private isInitialized: boolean = false;  
  
 constructor() {  
 logger.info('Supabase message broker created');  
 }  
  
 /\*\*  
 \* Initialize the message broker  
 \*/  
 async init(): Promise<void> {  
 try {  
 // We don't need any specific initialization for Supabase Realtime  
 // as channels are created on-demand  
 this.isInitialized = true;  
  
 logger.info('Supabase message broker initialized');  
 } catch (err) {  
 logger.error(`Failed to initialize Supabase message broker: ${err}`);  
 throw err;  
 }  
 }  
  
 /\*\*  
 \* Create or get a Supabase Realtime channel  
 \* @param channelKey Channel key  
 \* @returns Channel data object  
 \*/  
 private async getOrCreateChannel(channelKey: string): Promise<ChannelData> {  
 // Check if channel already exists  
 let channelData = this.channels.get(channelKey);  
  
 if (!channelData) {  
 const client = supabaseClient.getClient();  
  
 // Create a new Supabase Realtime channel  
 const channel = client.channel(`queue:${channelKey}`, {  
 config: {  
 broadcast: {  
 self: true  
 }  
 }  
 });  
  
 // Set up channel status handler  
 channel.on('system', { event: 'status' }, (status) => {  
 if (status === 'SUBSCRIBED') {  
 logger.info(`Subscribed to Supabase Realtime channel: ${channelKey}`);  
 } else if (status === 'CLOSED' || status === 'CHANNEL\_ERROR') {  
 logger.warn(`Supabase Realtime channel ${channelKey} status: ${status}`);  
 }  
 });  
  
 // Subscribe to the channel  
 channel.subscribe();  
  
 // Store the channel  
 channelData = { channel, subscribers: [] };  
 this.channels.set(channelKey, channelData);  
 }  
  
 return channelData;  
 }  
  
 /\*\*  
 \* Publish a message to a channel  
 \* @param channelKey Channel key  
 \* @param messageType Message type  
 \* @param payload Message payload  
 \*/  
 async publish(channelKey: string, messageType: string, payload: any): Promise<void> {  
 // Get or create the channel  
 const { channel } = await this.getOrCreateChannel(channelKey);  
  
 // Publish message to Supabase Realtime channel  
 await channel.send({  
 type: messageType,  
 data: payload  
 });  
 }  
  
 /\*\*  
 \* Subscribe to messages on a channel  
 \* @param channelKey Channel key  
 \* @param callback Function to call when a message is received  
 \* @returns Unsubscribe function  
 \*/  
 async subscribe(channelKey: string, callback: MessageCallback): Promise<UnsubscribeFunction> {  
 // Get or create the channel  
 const channelData = await this.getOrCreateChannel(channelKey);  
  
 // Set up message handler  
 const handler = (payload: any) => {  
 callback({  
 type: payload.type,  
 data: payload.data,  
 timestamp: new Date()  
 });  
 };  
  
 // Subscribe to messages  
 channelData.channel.on('broadcast', { event: '\*' }, handler);  
  
 // Store the subscriber  
 const subscriber = { callback, handler };  
 channelData.subscribers.push(subscriber);  
  
 // Return unsubscribe function  
 return async () => {  
 // Remove the subscriber  
 const index = channelData.subscribers.findIndex(s => s === subscriber);  
 if (index >= 0) {  
 channelData.subscribers.splice(index, 1);  
 }  
  
 // If no more subscribers, remove the channel  
 if (channelData.subscribers.length === 0) {  
 // Unsubscribe from the channel  
 channelData.channel.unsubscribe();  
  
 // Remove from channels map  
 this.channels.delete(channelKey);  
  
 logger.info(`Removed Supabase Realtime channel ${channelKey}`);  
 }  
 };  
 }  
  
 /\*\*  
 \* Shut down the message broker  
 \*/  
 async shutdown(): Promise<void> {  
 // Unsubscribe from all channels  
 for (const [key, { channel }] of this.channels.entries()) {  
 channel.unsubscribe()  
 .then(() => {  
 logger.info(`Closed Supabase Realtime channel: ${key}`);  
 })  
 .catch((err: Error) => {  
 logger.error(`Error closing Supabase Realtime channel ${key}: ${err}`);  
 });  
 }  
  
 // Clear channels  
 this.channels.clear();  
  
 logger.info('Supabase message broker shutdown complete');  
 }  
}

/\*\*  
 \* Supabase Message Broker  
 \*  
 \* Provides a pub/sub system for messaging between services using Supabase Realtime  
 \*/  
export class SupabaseMessageBroker implements MessageBroker {  
 private channels: Map<string, RealtimeChannel> = new Map();  
 private isInitialized: boolean = false;  
  
 constructor() {  
 logger.info('Supabase message broker created');  
 }  
  
 /\*\*  
 \* Initialize the message broker  
 \*/  
 async init(): Promise<void> {  
 try {  
 // We don't need any specific initialization for Supabase Realtime  
 // as channels are created on-demand  
 this.isInitialized = true;  
  
 logger.info('Supabase message broker initialized');  
 } catch (err) {  
 logger.error(`Failed to initialize Supabase message broker: ${err}`);  
 throw err;  
 }  
 }  
  
 /\*\*  
 \* Create or get a Supabase Realtime channel  
 \* @param channelKey Channel key  
 \* @returns Channel data object  
 \*/  
 private async getOrCreateChannel(channelKey: string): Promise<ChannelData> {  
 // Check if channel already exists  
 let channelData = this.channels.get(channelKey);  
  
 if (!channelData) {  
 const client = supabaseClient.getClient();  
  
 // Create a new Supabase Realtime channel  
 const channel = client.channel(`queue:${channelKey}`, {  
 config: {  
 broadcast: {  
 self: true  
 }  
 }  
 });  
  
 // Set up channel status handler  
 channel.on('system', { event: 'status' }, (status) => {  
 if (status === 'SUBSCRIBED') {  
 logger.info(`Subscribed to Supabase Realtime channel: ${channelKey}`);  
 } else if (status === 'CLOSED' || status === 'CHANNEL\_ERROR') {  
 logger.warn(`Supabase Realtime channel ${channelKey} status: ${status}`);  
 }  
 });  
  
 // Subscribe to the channel  
 channel.subscribe();  
  
 // Store the channel  
 channelData = { channel, subscribers: [] };  
 this.channels.set(channelKey, channelData);  
 }  
  
 return channelData;  
 }  
  
 /\*\*  
 \* Publish a message to a channel  
 \* @param channelKey Channel key  
 \* @param messageType Message type  
 \* @param payload Message payload  
 \*/  
 async publish(channelKey: string, messageType: string, payload: any): Promise<void> {  
 // Get or create the channel  
 const { channel } = await this.getOrCreateChannel(channelKey);  
  
 // Publish message to Supabase Realtime channel  
 await channel.send({  
 type: messageType,  
 data: payload  
 });  
 }  
  
 /\*\*  
 \* Subscribe to messages on a channel  
 \* @param channelKey Channel key  
 \* @param callback Function to call when a message is received  
 \* @returns Unsubscribe function  
 \*/  
 async subscribe(channelKey: string, callback: MessageCallback): Promise<UnsubscribeFunction> {  
 // Get or create the channel  
 const channelData = await this.getOrCreateChannel(channelKey);  
  
 // Set up message handler  
 const handler = (payload: any) => {  
 callback({  
 type: payload.type,  
 data: payload.data,  
 timestamp: new Date()  
 });  
 };  
  
 // Subscribe to messages  
 channelData.channel.on('broadcast', { event: '\*' }, handler);  
  
 // Store the subscriber  
 const subscriber = { callback, handler };  
 channelData.subscribers.push(subscriber);  
  
 // Return unsubscribe function  
 return async () => {  
 // Remove the subscriber  
 const index = channelData.subscribers.findIndex(s => s === subscriber);  
 if (index >= 0) {  
 channelData.subscribers.splice(index, 1);  
 }  
  
 // If no more subscribers, remove the channel  
 if (channelData.subscribers.length === 0) {  
 // Unsubscribe from the channel  
 channelData.channel.unsubscribe();  
  
 // Remove from channels map  
 this.channels.delete(channelKey);  
  
 logger.info(`Removed Supabase Realtime channel ${channelKey}`);  
 }  
 };  
 }  
  
 /\*\*  
 \* Shut down the message broker  
 \*/  
 async shutdown(): Promise<void> {  
 // Unsubscribe from all channels  
 for (const [key, { channel }] of this.channels.entries()) {  
 channel.unsubscribe()  
 .then(() => {  
 logger.info(`Closed Supabase Realtime channel: ${key}`);  
 })  
 .catch((err: Error) => {  
 logger.error(`Error closing Supabase Realtime channel ${key}: ${err}`);  
 });  
 }  
  
 // Clear channels  
 this.channels.clear();  
  
 logger.info('Supabase message broker shutdown complete');  
 }  
}

#### Storage Service Integration

The system utilizes Supabase Storage for file management:

/\*\*  
 \* Supabase Storage Service  
 \*   
 \* Provides an interface for storing and retrieving files using Supabase Storage  
 \*/  
export class SupabaseStorageService {  
 private static instance: SupabaseStorageService;  
 private bucket: string;  
  
 private constructor() {  
 this.bucket = this.getStorageBucketName();  
 logger.info(`Supabase Storage Service initialized with bucket: ${this.bucket}`);  
 }  
  
 /\*\*  
 \* Get the singleton instance  
 \* @returns SupabaseStorageService instance  
 \*/  
 public static getInstance(): SupabaseStorageService {  
 if (!SupabaseStorageService.instance) {  
 SupabaseStorageService.instance = new SupabaseStorageService();  
 }  
 return SupabaseStorageService.instance;  
 }  
  
 /\*\*  
 \* Get the configured storage bucket name from admin settings  
 \* Falls back to environment variable or default if not configured  
 \*/  
 private getStorageBucketName(): string {  
 const SUPABASE\_BUCKET = process.env.SUPABASE\_STORAGE\_BUCKET || 'kai-materials';  
  
 try {  
 // Try to get from admin settings  
 const { data, error } = supabaseClient.getClient()  
 .from('admin\_settings')  
 .select('value')  
 .eq('key', 'storage\_bucket')  
 .single();  
  
 if (error || !data) {  
 logger.info(`No storage bucket configured in admin settings, using default: ${SUPABASE\_BUCKET}`);  
 return SUPABASE\_BUCKET;  
 }  
  
 logger.info(`Using admin-configured storage bucket: ${data.value}`);  
 return data.value;  
 } catch (error) {  
 logger.warn(`Error getting storage bucket from settings, using default: ${SUPABASE\_BUCKET}`);  
 return SUPABASE\_BUCKET;  
 }  
 }  
  
 /\*\*  
 \* Upload a file to Supabase Storage  
 \* @param filePath Local path to the file  
 \* @param storageKey Key to use in storage (folder/filename)  
 \* @returns Promise with upload result  
 \*/  
 async uploadFile(filePath: string, storageKey: string): Promise<{ url: string }> {  
 try {  
 // Read the file  
 const fileContent = await fs.promises.readFile(filePath);  
  
 // Upload to Supabase Storage  
 const { data, error } = await supabaseClient.getClient()  
 .storage  
 .from(this.bucket)  
 .upload(storageKey, fileContent, {  
 cacheControl: '3600',  
 upsert: true  
 });  
  
 if (error) throw error;  
  
 // Get the public URL  
 const { data: urlData } = supabaseClient.getClient()  
 .storage  
 .from(this.bucket)  
 .getPublicUrl(storageKey);  
  
 return { url: urlData.publicUrl };  
 } catch (error) {  
 logger.error(`Error uploading file to Supabase Storage: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Download a file from Supabase Storage  
 \* @param storageKey Key of the file in storage  
 \* @param outputPath Local path to save the file  
 \* @returns Promise with download result  
 \*/  
 async downloadFile(storageKey: string, outputPath: string): Promise<void> {  
 try {  
 // Download from Supabase Storage  
 const { data, error } = await supabaseClient.getClient()  
 .storage  
 .from(this.bucket)  
 .download(storageKey);  
  
 if (error) throw error;  
  
 // Ensure the directory exists  
 await fs.promises.mkdir(path.dirname(outputPath), { recursive: true });  
  
 // Write the file  
 await fs.promises.writeFile(outputPath, data);  
 } catch (error) {  
 logger.error(`Error downloading file from Supabase Storage: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Delete a file from Supabase Storage  
 \* @param storageKey Key of the file in storage  
 \* @returns Promise with deletion result  
 \*/  
 async deleteFile(storageKey: string): Promise<void> {  
 try {  
 // Delete from Supabase Storage  
 const { error } = await supabaseClient.getClient()  
 .storage  
 .from(this.bucket)  
 .remove([storageKey]);  
  
 if (error) throw error;  
 } catch (error) {  
 logger.error(`Error deleting file from Supabase Storage: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Get a signed URL for a file in Supabase Storage  
 \* @param storageKey Key of the file in storage  
 \* @param expiresIn Expiration time in seconds (default: 3600)  
 \* @returns Promise with signed URL  
 \*/  
 async getSignedUrl(storageKey: string, expiresIn: number = 3600): Promise<string> {  
 try {  
 // Get signed URL from Supabase Storage  
 const { data, error } = await supabaseClient.getClient()  
 .storage  
 .from(this.bucket)  
 .createSignedUrl(storageKey, expiresIn);  
  
 if (error) throw error;  
  
 return data.signedUrl;  
 } catch (error) {  
 logger.error(`Error getting signed URL from Supabase Storage: ${error}`);  
 throw error;  
 }  
 }  
}  
  
// Export singleton instance  
export const storageService = SupabaseStorageService.getInstance();

/\*\*  
 \* Supabase Storage Service  
 \*   
 \* Provides an interface for storing and retrieving files using Supabase Storage  
 \*/  
export class SupabaseStorageService {  
 private static instance: SupabaseStorageService;  
 private bucket: string;  
  
 private constructor() {  
 this.bucket = this.getStorageBucketName();  
 logger.info(`Supabase Storage Service initialized with bucket: ${this.bucket}`);  
 }  
  
 /\*\*  
 \* Get the singleton instance  
 \* @returns SupabaseStorageService instance  
 \*/  
 public static getInstance(): SupabaseStorageService {  
 if (!SupabaseStorageService.instance) {  
 SupabaseStorageService.instance = new SupabaseStorageService();  
 }  
 return SupabaseStorageService.instance;  
 }  
  
 /\*\*  
 \* Get the configured storage bucket name from admin settings  
 \* Falls back to environment variable or default if not configured  
 \*/  
 private getStorageBucketName(): string {  
 const SUPABASE\_BUCKET = process.env.SUPABASE\_STORAGE\_BUCKET || 'kai-materials';  
  
 try {  
 // Try to get from admin settings  
 const { data, error } = supabaseClient.getClient()  
 .from('admin\_settings')  
 .select('value')  
 .eq('key', 'storage\_bucket')  
 .single();  
  
 if (error || !data) {  
 logger.info(`No storage bucket configured in admin settings, using default: ${SUPABASE\_BUCKET}`);  
 return SUPABASE\_BUCKET;  
 }  
  
 logger.info(`Using admin-configured storage bucket: ${data.value}`);  
 return data.value;  
 } catch (error) {  
 logger.warn(`Error getting storage bucket from settings, using default: ${SUPABASE\_BUCKET}`);  
 return SUPABASE\_BUCKET;  
 }  
 }  
  
 /\*\*  
 \* Upload a file to Supabase Storage  
 \* @param filePath Local path to the file  
 \* @param storageKey Key to use in storage (folder/filename)  
 \* @returns Promise with upload result  
 \*/  
 async uploadFile(filePath: string, storageKey: string): Promise<{ url: string }> {  
 try {  
 // Read the file  
 const fileContent = await fs.promises.readFile(filePath);  
  
 // Upload to Supabase Storage  
 const { data, error } = await supabaseClient.getClient()  
 .storage  
 .from(this.bucket)  
 .upload(storageKey, fileContent, {  
 cacheControl: '3600',  
 upsert: true  
 });  
  
 if (error) throw error;  
  
 // Get the public URL  
 const { data: urlData } = supabaseClient.getClient()  
 .storage  
 .from(this.bucket)  
 .getPublicUrl(storageKey);  
  
 return { url: urlData.publicUrl };  
 } catch (error) {  
 logger.error(`Error uploading file to Supabase Storage: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Download a file from Supabase Storage  
 \* @param storageKey Key of the file in storage  
 \* @param outputPath Local path to save the file  
 \* @returns Promise with download result  
 \*/  
 async downloadFile(storageKey: string, outputPath: string): Promise<void> {  
 try {  
 // Download from Supabase Storage  
 const { data, error } = await supabaseClient.getClient()  
 .storage  
 .from(this.bucket)  
 .download(storageKey);  
  
 if (error) throw error;  
  
 // Ensure the directory exists  
 await fs.promises.mkdir(path.dirname(outputPath), { recursive: true });  
  
 // Write the file  
 await fs.promises.writeFile(outputPath, data);  
 } catch (error) {  
 logger.error(`Error downloading file from Supabase Storage: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Delete a file from Supabase Storage  
 \* @param storageKey Key of the file in storage  
 \* @returns Promise with deletion result  
 \*/  
 async deleteFile(storageKey: string): Promise<void> {  
 try {  
 // Delete from Supabase Storage  
 const { error } = await supabaseClient.getClient()  
 .storage  
 .from(this.bucket)  
 .remove([storageKey]);  
  
 if (error) throw error;  
 } catch (error) {  
 logger.error(`Error deleting file from Supabase Storage: ${error}`);  
 throw error;  
 }  
 }  
  
 /\*\*  
 \* Get a signed URL for a file in Supabase Storage  
 \* @param storageKey Key of the file in storage  
 \* @param expiresIn Expiration time in seconds (default: 3600)  
 \* @returns Promise with signed URL  
 \*/  
 async getSignedUrl(storageKey: string, expiresIn: number = 3600): Promise<string> {  
 try {  
 // Get signed URL from Supabase Storage  
 const { data, error } = await supabaseClient.getClient()  
 .storage  
 .from(this.bucket)  
 .createSignedUrl(storageKey, expiresIn);  
  
 if (error) throw error;  
  
 return data.signedUrl;  
 } catch (error) {  
 logger.error(`Error getting signed URL from Supabase Storage: ${error}`);  
 throw error;  
 }  
 }  
}  
  
// Export singleton instance  
export const storageService = SupabaseStorageService.getInstance();

### Integration with Other Systems

#### Material Recognition System

Supabase integrates with the material recognition system:

1. Vector Storage and Retrieval
2. Storage of feature vectors for materials
3. Efficient similarity search for recognition
4. Storage of recognition results
5. Material metadata management
6. Versioning and history tracking
7. Recognition Workflow
8. Image upload to Supabase Storage
9. Vector embedding storage in pgvector
10. Similarity search for matching
11. Result storage and retrieval
12. Performance monitoring
13. Feedback Loop
14. Storage of user feedback
15. Correction tracking
16. Iterative improvement data
17. Performance metrics storage
18. Training data generation

Versioning and history tracking

Recognition Workflow

Performance monitoring

Feedback Loop

#### Dataset Management

Supabase powers the dataset management system:

1. Dataset Storage
2. Dataset metadata in PostgreSQL
3. Image files in Supabase Storage
4. Class hierarchies and organization
5. Version control and history
6. Preprocessing results
7. Dataset Operations
8. Dataset creation and modification
9. Image upload and organization
10. Class management
11. Quality assessment storage
12. Versioning and rollback
13. Training Integration
14. Dataset selection for training
15. Progress tracking with real-time updates
16. Performance metrics storage
17. Model-dataset relationships
18. Validation results

Preprocessing results

Dataset Operations

Versioning and rollback

Training Integration

#### Knowledge Base

Supabase serves as the foundation for the knowledge base:

1. Entity Storage
2. Material information storage
3. Relationship management
4. Taxonomy and categorization
5. Attribute and property handling
6. Versioning and history
7. Search Capabilities
8. Full-text search for content
9. Vector search for similarity
10. Hybrid search combining approaches
11. Faceted search support
12. Relevance tuning
13. Content Organization
14. Collection and category management
15. Hierarchical structures
16. Cross-referencing and relationships
17. User-specific content views
18. Access control and permissions

Versioning and history

Search Capabilities

Relevance tuning

Content Organization

#### Queue System

Supabase supports the queue management system:

1. Job Management
2. Job storage and tracking
3. Status updates via real-time channels
4. Priority and scheduling management
5. Resource allocation tracking
6. Error handling and retry logic
7. Event Communication
8. Real-time status notifications
9. Cross-service coordination
10. Client updates via WebSockets
11. Processing milestone tracking
12. Completion notifications
13. Performance Monitoring
14. Queue metrics storage
15. Processing time tracking
16. Resource utilization monitoring
17. Bottleneck identification
18. Historical performance analysis

Error handling and retry logic

Event Communication

Completion notifications

Performance Monitoring

### Usage Examples

#### Initializing Supabase

import { supabaseClient } from './services/supabase/supabaseClient';  
  
// In application startup  
async function initializeServices() {  
 // Initialize Supabase with configuration  
 supabaseClient.init({  
 url: process.env.SUPABASE\_URL,  
 key: process.env.SUPABASE\_KEY  
 });  
  
 // Verify connection  
 try {  
 const client = supabaseClient.getClient();  
 const { data, error } = await client.from('health\_check').select('\*').limit(1);  
  
 if (error) {  
 console.error('Supabase connection error:', error);  
 throw error;  
 }  
  
 console.log('Supabase connection successful');  
 } catch (error) {  
 console.error('Failed to initialize Supabase:', error);  
 process.exit(1);  
 }  
}

import { supabaseClient } from './services/supabase/supabaseClient';  
  
// In application startup  
async function initializeServices() {  
 // Initialize Supabase with configuration  
 supabaseClient.init({  
 url: process.env.SUPABASE\_URL,  
 key: process.env.SUPABASE\_KEY  
 });  
  
 // Verify connection  
 try {  
 const client = supabaseClient.getClient();  
 const { data, error } = await client.from('health\_check').select('\*').limit(1);  
  
 if (error) {  
 console.error('Supabase connection error:', error);  
 throw error;  
 }  
  
 console.log('Supabase connection successful');  
 } catch (error) {  
 console.error('Failed to initialize Supabase:', error);  
 process.exit(1);  
 }  
}

#### Material Management with Supabase

import { supabaseMaterialService } from './services/supabase/supabase-material-service';  
  
async function manageMaterials() {  
 try {  
 // Create a new material  
 const newMaterial = await supabaseMaterialService.createMaterial({  
 name: 'Ceramic Tile XYZ',  
 description: 'Premium ceramic tile with matte finish',  
 materialType: 'ceramic',  
 manufacturer: 'TileCo',  
 dimensions: {  
 width: 30,  
 height: 30,  
 thickness: 1.2  
 },  
 color: {  
 name: 'Slate Gray',  
 hex: '#708090'  
 },  
 finish: 'matte',  
 tags: ['ceramic', 'tile', 'floor', 'matte']  
 });  
  
 console.log(`Created material with ID: ${newMaterial.id}`);  
  
 // Search for materials  
 const searchResults = await supabaseMaterialService.searchMaterials({  
 query: 'ceramic matte',  
 materialType: 'ceramic',  
 limit: 10  
 });  
  
 console.log(`Found ${searchResults.length} matching materials`);  
  
 // Find similar materials using vector search  
 if (newMaterial.vectorRepresentation) {  
 const similarMaterials = await supabaseMaterialService.findSimilarMaterials(  
 newMaterial.vectorRepresentation,  
 {  
 threshold: 0.7,  
 limit: 5,  
 excludeIds: [newMaterial.id]  
 }  
 );  
  
 console.log(`Found ${similarMaterials.length} similar materials`);  
 }  
  
 // Update a material  
 const updatedMaterial = await supabaseMaterialService.updateMaterial(newMaterial.id, {  
 description: 'Premium ceramic tile with matte finish, updated description',  
 tags: [...newMaterial.tags, 'premium']  
 });  
  
 console.log(`Updated material: ${updatedMaterial.name}`);  
  
 // Get material statistics  
 const stats = await supabaseMaterialService.getMaterialStats();  
  
 console.log(`Total materials: ${stats.totalCount}`);  
 console.log(`Materials by type: ${JSON.stringify(stats.byType)}`);  
  
 } catch (error) {  
 console.error('Material management failed:', error);  
 }  
}

import { supabaseMaterialService } from './services/supabase/supabase-material-service';  
  
async function manageMaterials() {  
 try {  
 // Create a new material  
 const newMaterial = await supabaseMaterialService.createMaterial({  
 name: 'Ceramic Tile XYZ',  
 description: 'Premium ceramic tile with matte finish',  
 materialType: 'ceramic',  
 manufacturer: 'TileCo',  
 dimensions: {  
 width: 30,  
 height: 30,  
 thickness: 1.2  
 },  
 color: {  
 name: 'Slate Gray',  
 hex: '#708090'  
 },  
 finish: 'matte',  
 tags: ['ceramic', 'tile', 'floor', 'matte']  
 });  
  
 console.log(`Created material with ID: ${newMaterial.id}`);  
  
 // Search for materials  
 const searchResults = await supabaseMaterialService.searchMaterials({  
 query: 'ceramic matte',  
 materialType: 'ceramic',  
 limit: 10  
 });  
  
 console.log(`Found ${searchResults.length} matching materials`);  
  
 // Find similar materials using vector search  
 if (newMaterial.vectorRepresentation) {  
 const similarMaterials = await supabaseMaterialService.findSimilarMaterials(  
 newMaterial.vectorRepresentation,  
 {  
 threshold: 0.7,  
 limit: 5,  
 excludeIds: [newMaterial.id]  
 }  
 );  
  
 console.log(`Found ${similarMaterials.length} similar materials`);  
 }  
  
 // Update a material  
 const updatedMaterial = await supabaseMaterialService.updateMaterial(newMaterial.id, {  
 description: 'Premium ceramic tile with matte finish, updated description',  
 tags: [...newMaterial.tags, 'premium']  
 });  
  
 console.log(`Updated material: ${updatedMaterial.name}`);  
  
 // Get material statistics  
 const stats = await supabaseMaterialService.getMaterialStats();  
  
 console.log(`Total materials: ${stats.totalCount}`);  
 console.log(`Materials by type: ${JSON.stringify(stats.byType)}`);  
  
 } catch (error) {  
 console.error('Material management failed:', error);  
 }  
}

#### Working with Supabase Storage

import { storageService } from './services/storage/supabaseStorageService';  
import \* as fs from 'fs';  
import \* as path from 'path';  
  
async function manageFiles() {  
 try {  
 // Upload a file  
 const filePath = path.join(process.cwd(), 'uploads', 'sample\_image.jpg');  
 const storageKey = `materials/ceramic/sample\_${Date.now()}.jpg`;  
  
 const { url } = await storageService.uploadFile(filePath, storageKey);  
  
 console.log(`File uploaded successfully. Public URL: ${url}`);  
  
 // Get a signed URL for temporary access  
 const signedUrl = await storageService.getSignedUrl(storageKey, 1800); // 30 minutes  
  
 console.log(`Signed URL valid for 30 minutes: ${signedUrl}`);  
  
 // Download the file to a different location  
 const downloadPath = path.join(process.cwd(), 'downloads', 'sample\_downloaded.jpg');  
  
 await storageService.downloadFile(storageKey, downloadPath);  
  
 console.log(`File downloaded to: ${downloadPath}`);  
  
 // Get file metadata  
 const metadata = await fs.promises.stat(downloadPath);  
  
 console.log(`Downloaded file size: ${metadata.size} bytes`);  
  
 // Delete the file after processing  
 await storageService.deleteFile(storageKey);  
  
 console.log(`File deleted from storage: ${storageKey}`);  
  
 } catch (error) {  
 console.error('File management failed:', error);  
 }  
}

import { storageService } from './services/storage/supabaseStorageService';  
import \* as fs from 'fs';  
import \* as path from 'path';  
  
async function manageFiles() {  
 try {  
 // Upload a file  
 const filePath = path.join(process.cwd(), 'uploads', 'sample\_image.jpg');  
 const storageKey = `materials/ceramic/sample\_${Date.now()}.jpg`;  
  
 const { url } = await storageService.uploadFile(filePath, storageKey);  
  
 console.log(`File uploaded successfully. Public URL: ${url}`);  
  
 // Get a signed URL for temporary access  
 const signedUrl = await storageService.getSignedUrl(storageKey, 1800); // 30 minutes  
  
 console.log(`Signed URL valid for 30 minutes: ${signedUrl}`);  
  
 // Download the file to a different location  
 const downloadPath = path.join(process.cwd(), 'downloads', 'sample\_downloaded.jpg');  
  
 await storageService.downloadFile(storageKey, downloadPath);  
  
 console.log(`File downloaded to: ${downloadPath}`);  
  
 // Get file metadata  
 const metadata = await fs.promises.stat(downloadPath);  
  
 console.log(`Downloaded file size: ${metadata.size} bytes`);  
  
 // Delete the file after processing  
 await storageService.deleteFile(storageKey);  
  
 console.log(`File deleted from storage: ${storageKey}`);  
  
 } catch (error) {  
 console.error('File management failed:', error);  
 }  
}

#### Real-time Communication with Supabase

import { messageBroker } from './services/messaging/messageBroker';  
  
async function setupRealTimeUpdates() {  
 try {  
 // Initialize the message broker  
 await messageBroker.init();  
  
 // Subscribe to PDF processing queue events  
 const pdfUnsubscribe = await messageBroker.subscribe('pdf', async (message) => {  
 const { type, data } = message;  
  
 console.log(`Received PDF queue event: ${type}`);  
  
 switch (type) {  
 case 'job\_added':  
 console.log(`New PDF job added: ${data.jobId}`);  
 break;  
 case 'job\_started':  
 console.log(`PDF job started: ${data.jobId}`);  
 break;  
 case 'job\_completed':  
 console.log(`PDF job completed: ${data.jobId}, extracted ${data.pageCount} pages`);  
 break;  
 case 'job\_failed':  
 console.error(`PDF job failed: ${data.jobId}, error: ${data.error}`);  
 break;  
 }  
 });  
  
 // Subscribe to crawler queue events  
 const crawlerUnsubscribe = await messageBroker.subscribe('crawler', async (message) => {  
 const { type, data } = message;  
  
 console.log(`Received crawler queue event: ${type}`);  
  
 // Handle crawler events...  
 });  
  
 // Publish an event to the PDF queue  
 await messageBroker.publish('pdf', 'system\_status', {  
 status: 'ready',  
 timestamp: new Date().toISOString(),  
 queueSize: 0  
 });  
  
 // Keep subscriptions active for some time...  
 setTimeout(() => {  
 // Unsubscribe when no longer needed  
 pdfUnsubscribe();  
 crawlerUnsubscribe();  
  
 console.log('Unsubscribed from queue events');  
  
 // Shut down the message broker  
 messageBroker.shutdown();  
 }, 3600000); // 1 hour  
  
 } catch (error) {  
 console.error('Real-time setup failed:', error);  
 }  
}

import { messageBroker } from './services/messaging/messageBroker';  
  
async function setupRealTimeUpdates() {  
 try {  
 // Initialize the message broker  
 await messageBroker.init();  
  
 // Subscribe to PDF processing queue events  
 const pdfUnsubscribe = await messageBroker.subscribe('pdf', async (message) => {  
 const { type, data } = message;  
  
 console.log(`Received PDF queue event: ${type}`);  
  
 switch (type) {  
 case 'job\_added':  
 console.log(`New PDF job added: ${data.jobId}`);  
 break;  
 case 'job\_started':  
 console.log(`PDF job started: ${data.jobId}`);  
 break;  
 case 'job\_completed':  
 console.log(`PDF job completed: ${data.jobId}, extracted ${data.pageCount} pages`);  
 break;  
 case 'job\_failed':  
 console.error(`PDF job failed: ${data.jobId}, error: ${data.error}`);  
 break;  
 }  
 });  
  
 // Subscribe to crawler queue events  
 const crawlerUnsubscribe = await messageBroker.subscribe('crawler', async (message) => {  
 const { type, data } = message;  
  
 console.log(`Received crawler queue event: ${type}`);  
  
 // Handle crawler events...  
 });  
  
 // Publish an event to the PDF queue  
 await messageBroker.publish('pdf', 'system\_status', {  
 status: 'ready',  
 timestamp: new Date().toISOString(),  
 queueSize: 0  
 });  
  
 // Keep subscriptions active for some time...  
 setTimeout(() => {  
 // Unsubscribe when no longer needed  
 pdfUnsubscribe();  
 crawlerUnsubscribe();  
  
 console.log('Unsubscribed from queue events');  
  
 // Shut down the message broker  
 messageBroker.shutdown();  
 }, 3600000); // 1 hour  
  
 } catch (error) {  
 console.error('Real-time setup failed:', error);  
 }  
}

### Configuration and Deployment

#### Environment Variables

Supabase integration is configured using environment variables:

# Supabase Connection  
SUPABASE\_URL=https://your-project.supabase.co  
SUPABASE\_KEY=your-service-role-key  
SUPABASE\_ANON\_KEY=your-anon-key  
  
# Storage Configuration  
SUPABASE\_STORAGE\_BUCKET=kai-materials  
  
# Security Settings  
SUPABASE\_JWT\_SECRET=your-jwt-secret

# Supabase Connection  
SUPABASE\_URL=https://your-project.supabase.co  
SUPABASE\_KEY=your-service-role-key  
SUPABASE\_ANON\_KEY=your-anon-key  
  
# Storage Configuration  
SUPABASE\_STORAGE\_BUCKET=kai-materials  
  
# Security Settings  
SUPABASE\_JWT\_SECRET=your-jwt-secret

#### Deployment Considerations

1. Database Migration
2. Apply migrations before deployment
3. Test migrations in staging environment
4. Have rollback plans for failed migrations
5. Implement zero-downtime migration strategy
6. Monitor database performance after migrations
7. Access Management
8. Use service role key for server operations
9. Use anon key for client-side operations
10. Implement row-level security policies
11. Review and test permission settings
12. Regularly rotate keys and credentials
13. Performance Configuration
14. Optimize vector indices for dataset size
15. Configure connection pooling for server instances
16. Set up caching for frequent queries
17. Monitor and adjust statement timeouts
18. Implement proper connection handling
19. Data Backups
20. Configure regular Supabase backups
21. Implement point-in-time recovery options
22. Validate backup restoration process
23. Maintain backup retention policy
24. Monitor backup successes and failures
25. Scaling Considerations
26. Monitor database connection utilization
27. Scale compute resources as needed
28. Implement query optimization for large tables
29. Consider read replicas for heavy workloads
30. Plan for data volume growth

Monitor database performance after migrations

Access Management

Regularly rotate keys and credentials

Performance Configuration

Implement proper connection handling

Data Backups

Monitor backup successes and failures

Scaling Considerations

#### Health Monitoring

To ensure Supabase integration remains healthy:

/\*\*  
 \* Check Supabase connection health  
 \* @returns Health check results  
 \*/  
async function checkSupabaseHealth(): Promise<HealthCheckResult> {  
 try {  
 const startTime = Date.now();  
  
 // Get the Supabase client  
 const client = supabaseClient.getClient();  
  
 // Check basic connectivity with a simple query  
 const { data, error } = await client  
 .from('health\_check')  
 .select('\*')  
 .limit(1);  
  
 if (error) {  
 return {  
 status: 'error',  
 message: `Database connection error: ${error.message}`,  
 latency: Date.now() - startTime  
 };  
 }  
  
 // Check pgvector extension  
 const { data: pgvectorData, error: pgvectorError } = await client.rpc(  
 'check\_extension',  
 { extension\_name: 'vector' }  
 );  
  
 // Check storage access  
 const { data: storageData, error: storageError } = await client.storage  
 .from(process.env.SUPABASE\_STORAGE\_BUCKET || 'kai-materials')  
 .list('', { limit: 1 });  
  
 // Return health status  
 return {  
 status: 'healthy',  
 message: 'Supabase connection is healthy',  
 latency: Date.now() - startTime,  
 details: {  
 database: error ? 'error' : 'connected',  
 pgvector: pgvectorError ? 'not available' : 'available',  
 storage: storageError ? 'error' : 'connected'  
 }  
 };  
 } catch (error) {  
 return {  
 status: 'error',  
 message: `Supabase health check failed: ${error instanceof Error ? error.message : String(error)}`,  
 latency: -1  
 };  
 }  
}

/\*\*  
 \* Check Supabase connection health  
 \* @returns Health check results  
 \*/  
async function checkSupabaseHealth(): Promise<HealthCheckResult> {  
 try {  
 const startTime = Date.now();  
  
 // Get the Supabase client  
 const client = supabaseClient.getClient();  
  
 // Check basic connectivity with a simple query  
 const { data, error } = await client  
 .from('health\_check')  
 .select('\*')  
 .limit(1);  
  
 if (error) {  
 return {  
 status: 'error',  
 message: `Database connection error: ${error.message}`,  
 latency: Date.now() - startTime  
 };  
 }  
  
 // Check pgvector extension  
 const { data: pgvectorData, error: pgvectorError } = await client.rpc(  
 'check\_extension',  
 { extension\_name: 'vector' }  
 );  
  
 // Check storage access  
 const { data: storageData, error: storageError } = await client.storage  
 .from(process.env.SUPABASE\_STORAGE\_BUCKET || 'kai-materials')  
 .list('', { limit: 1 });  
  
 // Return health status  
 return {  
 status: 'healthy',  
 message: 'Supabase connection is healthy',  
 latency: Date.now() - startTime,  
 details: {  
 database: error ? 'error' : 'connected',  
 pgvector: pgvectorError ? 'not available' : 'available',  
 storage: storageError ? 'error' : 'connected'  
 }  
 };  
 } catch (error) {  
 return {  
 status: 'error',  
 message: `Supabase health check failed: ${error instanceof Error ? error.message : String(error)}`,  
 latency: -1  
 };  
 }  
}

### Performance Considerations

1. Connection Management
2. Use a singleton client to prevent connection pool exhaustion
3. Implement proper connection error handling
4. Close connections when no longer needed
5. Monitor connection pool utilization
6. Configure timeouts appropriately
7. Query Optimization
8. Use indexes for frequently queried columns
9. Optimize vector queries with appropriate indexing
10. Implement pagination for large result sets
11. Use optimized PostgreSQL functions
12. Monitor and optimize slow queries
13. Vector Search Performance
14. Choose appropriate indexing methods (HNSW vs. IVF-Flat)
15. Optimize vector dimensions for balance of accuracy/performance
16. Implement approximate nearest neighbor search
17. Tune similarity thresholds for result quality
18. Cache common search results
19. Real-time Performance
20. Monitor WebSocket connection count
21. Implement message throttling for high-volume events
22. Use selective subscriptions rather than generic ones
23. Implement backoff strategies for reconnections
24. Close unneeded subscriptions promptly
25. Storage Efficiency
26. Use appropriate file formats and compression
27. Implement file lifecycle policies
28. Use presigned URLs for client-side uploads
29. Configure appropriate cache headers
30. Monitor storage utilization and growth

Configure timeouts appropriately

Query Optimization

Monitor and optimize slow queries

Vector Search Performance

Cache common search results

Real-time Performance

Close unneeded subscriptions promptly

Storage Efficiency

# System Dependencies And Integrations

Source: readme/system-dependencies-and-integrations.md

---

## System Dependencies and Integrations

This document provides a comprehensive overview of all dependencies, integrations, and packages used across different systems within the KAI platform. It serves as a reference for developers and should be updated whenever new dependencies are added.

### Table of Contents

* 3D Visualization & Model Generation
* Material Recognition & Property Extraction
* Scene Understanding & Reconstruction
* Interior Design & Automation
* Texture & Content Generation
* Mobile & Performance Optimization
* Infrastructure & Processing
* Development Tools

### 3D Visualization & Model Generation

Table content:

Dependency | Version | Purpose | Integration Points

hdrnet-pytorch | ^0.2.0 | Automatic lighting inference and environment map generation | Used by LightingEstimationService to generate HDR environment maps for realistic lighting in 3D visualizations

envmapnet | ^0.1.0 | Environment map processing and enhancement | Complements HDRNet for improved environment map quality and tone mapping

Three.js | N/A | 3D rendering library for web-based visualization | Core component for all 3D visualization features in client applications

LightingEstimationService

#### Key Components:

* LightingEstimationService: Leverages HDRNet for automatic lighting inference from images
* MaterialVisualizationProvider: Uses environment maps for physically-based lighting in 3D scenes
* ThreeJsViewer: Renders 3D models with realistic lighting and materials

#### Integration Flow:

1. Material images are processed through HDRNet to extract lighting information
2. Generated environment maps are used for physically-based rendering
3. Three.js creates the final visualization with accurate lighting and shadows

### Material Recognition & Property Extraction

Table content:

Dependency | Version | Purpose | Integration Points

materialnet | ^1.0.0 | Automatic PBR material property extraction | Used by MaterialNetProvider to extract physically-based rendering properties from material images

tensorflow-js | N/A | ML inference for front-end material recognition | Powers client-side material recognition features

pytorch | N/A | Deep learning framework for advanced material analysis | Backend for material recognition model training and inference

MaterialNetProvider

#### Key Components:

* MaterialNetProvider: Extracts PBR properties from material images
* MaterialRecognitionProvider: Identifies materials from images
* MaterialVisualizationProvider: Uses extracted properties for accurate rendering

#### Integration Flow:

1. Material images are processed through MaterialNet to extract PBR properties
2. Extracted properties are used to enhance 3D visualizations
3. Recognition results inform material selection and recommendations

### Scene Understanding & Reconstruction

Table content:

Dependency | Version | Purpose | Integration Points

pycolmap | ^0.3.0 | Camera pose estimation and multi-view consistency | Used by CameraEstimationProvider to extract camera poses from multiple images for consistent 3D reconstruction

diffusionnerf | ^0.1.0 | Enhanced scene optimization with diffusion models | Used by DiffusionNeRFProvider to improve scene reconstruction with adaptive quality selection based on input images

point-e | ^0.1.0 | Point cloud processing, noise reduction, and geometry optimization | Used by PointCloudProvider to enhance point cloud processing prior to mesh construction, improve geometry quality, and add denoising capabilities

scene-graph-3d | ^0.1.0 | 3D scene graph generation for relational understanding | Powers the SceneGraphProvider for creating relationship-aware scene representations, semantic queries, and intelligent scene editing tools

CameraEstimationProvider

DiffusionNeRFProvider

PointCloudProvider

SceneGraphProvider

#### Key Components:

* CameraEstimationProvider: Uses COLMAP for Structure-from-Motion processing and camera pose extraction
* colmap\_sfm\_service.py: Python service that performs the actual COLMAP processing
* camera-pose.routes.ts: API endpoints for camera pose estimation and NeRF enhancement
* DiffusionNeRFProvider: Extends BaseThreeDProvider to optimize scene reconstruction using diffusion models
* diffusion\_nerf\_service.py: Python service that implements quality assessment and adaptive reconstruction methods
* scene-optimization.routes.ts: API endpoints for scene optimization and quality-based reconstruction
* NeRFProvider: Handles neural radiance field processing for 3D reconstruction
* PointCloudProvider: Processes and optimizes point cloud data with noise reduction and geometry enhancement
* point\_cloud\_service.py: Python service that implements point cloud processing, denoising, and optimization algorithms
* point-cloud.routes.ts: API endpoints for point cloud processing, generation, completion, and mesh improvement
* SceneGraphProvider: Generates scene graphs for semantic understanding

#### Integration Flow:

1. Camera Pose Estimation:
2. Multiple images are uploaded through the /api/camera-pose/estimate endpoint
3. Images are processed by the COLMAP service to extract camera poses
4. Results include camera intrinsics, extrinsics, and 3D points
5. Optional visualization and NeRF format conversion
6. NeRF Enhancement:
7. Existing NeRF data is enhanced with camera poses via the /api/camera-pose/enhance-nerf endpoint
8. Improved camera positioning leads to more accurate 3D reconstructions
9. Results are compatible with standard NeRF training pipelines
10. Scene Optimization:
11. Images are analyzed for quality assessment via the /api/scene-optimization/assess endpoint
12. Based on quality metrics, the appropriate reconstruction method is selected
13. For sparse or incomplete views, DiffusionNeRF is used to fill in missing information
14. Results include enhanced 3D models with improved geometry and textures
15. Optional caching for faster processing of similar scenes
16. 3D Visualization:
17. Camera pose data is used by the ThreeJsViewer for accurate perspective rendering
18. MaterialVisualizationProvider uses camera positions for consistent material appearance
19. DiffusionNeRFProvider generates optimized scenes for visualization
20. PointCloudProvider optimizes point clouds and improves mesh quality
21. SceneController leverages camera transformation matrices for proper navigation
22. Material Recognition Integration:
23. Camera poses provide geometric context for material recognition
24. Multiple views of the same material improve property extraction accuracy
25. DiffusionNeRF models enhance material surfaces for better property extraction
26. MaterialNetProvider benefits from spatial relationships between detected materials
27. Adaptive Processing Pipeline:
28. Quality assessment determines the best reconstruction approach:  
    High-quality inputs: Standard NeRF reconstruction  
    Medium-quality inputs: Hybrid approach with DiffusionNeRF enhancement  
    Low-quality inputs: Full DiffusionNeRF reconstruction with regularization
29. Processing results are cached for similar future requests
30. Point-E Integration:  
    Raw point clouds are processed with noise reduction via /api/point-cloud/process  
    Text-to-point-cloud generation via /api/point-cloud/generate  
    Partial point cloud completion via /api/point-cloud/complete  
    Mesh geometry improvement via /api/point-cloud/improve-mesh  
    Image-to-point-cloud extraction via /api/point-cloud/process-image
31. Scene Graph Generation:  
    SceneGraphProvider: Creates relationship-aware scene representations via /api/scene-graph/\* endpoints  
    scene\_graph\_service.py: Python ML service that implements the 3DSSG algorithms  
    Supports multiple input types:  
    3D model-based scene graph generation via /api/scene-graph/generate-from-model  
    Point cloud-based graph generation via /api/scene-graph/generate-from-point-cloud  
    Image-based graph generation via /api/scene-graph/generate-from-images  
    Text-based scene description via /api/scene-graph/generate-from-text  
    Advanced features:  
    Semantic queries on scene graphs via /api/scene-graph/query  
    Layout suggestions via /api/scene-graph/generate-suggestions  
    Relationship-aware editing with contextual understanding  
    Support for multiple confidence levels and relationship limits  
    Integration with other components:  
    Works with PointCloudProvider for geometry understanding  
    Enhances MaterialNetProvider with semantic material relationships  
    Improves RoomLayoutProvider with object relationship context

/api/camera-pose/estimate

Optional visualization and NeRF format conversion

NeRF Enhancement:

/api/camera-pose/enhance-nerf

Results are compatible with standard NeRF training pipelines

Scene Optimization:

/api/scene-optimization/assess

Optional caching for faster processing of similar scenes

3D Visualization:

SceneController leverages camera transformation matrices for proper navigation

Material Recognition Integration:

MaterialNetProvider benefits from spatial relationships between detected materials

Adaptive Processing Pipeline:

* High-quality inputs: Standard NeRF reconstruction
* Medium-quality inputs: Hybrid approach with DiffusionNeRF enhancement
* Low-quality inputs: Full DiffusionNeRF reconstruction with regularization
* Raw point clouds are processed with noise reduction via /api/point-cloud/process
* Text-to-point-cloud generation via /api/point-cloud/generate
* Partial point cloud completion via /api/point-cloud/complete
* Mesh geometry improvement via /api/point-cloud/improve-mesh
* Image-to-point-cloud extraction via /api/point-cloud/process-image

/api/point-cloud/process

/api/point-cloud/generate

/api/point-cloud/complete

/api/point-cloud/improve-mesh

/api/point-cloud/process-image

* SceneGraphProvider: Creates relationship-aware scene representations via /api/scene-graph/\* endpoints
* scene\_graph\_service.py: Python ML service that implements the 3DSSG algorithms
* Supports multiple input types:
* 3D model-based scene graph generation via /api/scene-graph/generate-from-model
* Point cloud-based graph generation via /api/scene-graph/generate-from-point-cloud
* Image-based graph generation via /api/scene-graph/generate-from-images
* Text-based scene description via /api/scene-graph/generate-from-text
* Advanced features:
* Semantic queries on scene graphs via /api/scene-graph/query
* Layout suggestions via /api/scene-graph/generate-suggestions
* Relationship-aware editing with contextual understanding
* Support for multiple confidence levels and relationship limits
* Integration with other components:
* Works with PointCloudProvider for geometry understanding
* Enhances MaterialNetProvider with semantic material relationships
* Improves RoomLayoutProvider with object relationship context

/api/scene-graph/\*

/api/scene-graph/generate-from-model

/api/scene-graph/generate-from-point-cloud

/api/scene-graph/generate-from-images

/api/scene-graph/generate-from-text

/api/scene-graph/query

/api/scene-graph/generate-suggestions

### Interior Design & Automation

Table content:

Dependency | Version | Purpose | Integration Points

spaceformer | ^0.2.0 | Layout generation and furniture placement | Powers automated interior design features through RoomLayoutProvider and ArchitecturalProvider

#### Key Components:

* RoomLayoutProvider: Generates optimized room layouts and handles furniture placement optimization using design principles
* FurniturePlacementService: Automatically places furniture in scenes based on accessibility and flow
* ArchitecturalProvider: Handles architectural elements, layout constraints, and room structure
* space\_former\_service.py: Python ML service that implements the core SpaceFormer functionality

#### Features:

* Automated room layout generation based on room dimensions and type
* Intelligent furniture placement using design principles
* Multi-criteria optimization (flow, accessibility, balance, etc.)
* Layout analysis with improvement suggestions
* Room image processing for layout detection
* Design style enforcement and consistency
* Accessibility scoring and optimization

#### API Endpoints:

* /api/room-layout/generate: Generate optimized room layouts
* /api/room-layout/optimize-furniture: Optimize furniture placement
* /api/room-layout/analyze: Analyze existing layouts and provide suggestions
* /api/room-layout/process-image: Process room images for layout recognition
* /api/room-layout/optimize-existing: Optimize existing layouts

#### Integration Flow:

1. Room dimensions and constraints are processed through SpaceFormer
2. Layout suggestions are generated based on design principles
3. Furniture is automatically placed according to the layout
4. Optimization goals can be specified (e.g., prioritize flow, accessibility, or style)
5. Results include metrics for flow, occupancy, accessibility, and design principle compliance

### Texture & Content Generation

Table content:

Dependency | Version | Purpose | Integration Points

text2texture | ^0.1.0 | Texture generation and enhancement | Improves texture quality and generates new textures from descriptions

#### Key Components:

* TextureEnhancementProvider: Enhances and generates textures
* TextToTextureService: Converts text descriptions to textures

#### Integration Flow:

1. Low-resolution textures are processed for enhancement
2. Text descriptions are converted to high-quality textures
3. Generated textures are applied to 3D models

### Mobile & Performance Optimization

Table content:

Dependency | Version | Purpose | Integration Points

draco3d | ^1.5.0 | 3D mesh compression for mobile delivery | Optimizes 3D models for mobile and web delivery

assemblyscript | ^0.27.1 | WebAssembly compilation for browser performance | Used to compile performance-critical components to WebAssembly

#### Key Components:

* LODGenerator: Creates multiple detail levels for progressive loading
* MeshOptimizer: Compresses and optimizes meshes for delivery
* WasmProcessor: Handles WebAssembly compilation and execution

#### Integration Flow:

1. 3D models are optimized and compressed using Draco
2. Multiple LODs are generated for progressive loading
3. Performance-critical code is compiled to WebAssembly

### Infrastructure & Processing

Table content:

Dependency | Version | Purpose | Integration Points

serverless | ^3.30.1 | Serverless function deployment | Used for cloud function management

aws-lambda | ^1.0.7 | AWS Lambda functionality | Powers serverless processing in AWS environment

apache-airflow-client | ^2.5.1 | Workflow orchestration | Manages complex processing pipelines

bull | ^4.10.4 | Redis-based queue for Node.js | Powers distributed task processing

#### Key Components:

* QueueManager: Handles distributed task processing
* WorkflowManager: Orchestrates complex processing pipelines
* CloudFunctionService: Manages serverless function deployment and execution

#### Integration Flow:

1. Tasks are submitted to processing queues
2. Workers process tasks in distributed fashion
3. Complex workflows are orchestrated through Airflow
4. Results are stored and made available to clients

### Development Tools

Table content:

Dependency | Version | Purpose | Integration Points

wasm-pack | ^0.10.3 | WebAssembly packaging | Packages Rust code for WebAssembly

wasm-bindgen-cli | ^0.2.87 | WebAssembly bindings generator | Generates JavaScript bindings for WebAssembly modules

#### Key Components:

* BuildTools: Handles compilation and packaging
* DevEnvironment: Manages development environment setup

### Updating This Documentation

When adding new dependencies or integrations:

1. Add the dependency to the appropriate section
2. Document its purpose and integration points
3. Update any affected component descriptions
4. If creating a new category, add it to the Table of Contents

Last updated: April 2025

# System Updates Summary

Source: readme/system-updates-summary.md

---

## System Updates Summary

### Recent Platform Enhancements

The KAI platform has been enhanced with several major new features that require deployment updates:

#### 1. Notification & Webhook System

Multi-channel messaging framework supporting in-app, email, SMS, and webhook notifications.

#### 2. Parameter Registry System

Hyperparameter management system for material analysis that stores, retrieves, and suggests optimal parameters based on material type.

#### 3. MultiModal Pattern Recognition

Advanced ML system bridging visual patterns and textual specifications using transformer architecture with cross-modal attention.

#### 4. Domain-Specific Neural Networks

Specialized neural architectures optimized for material texture analysis with custom convolutional filters and attention mechanisms.

### Infrastructure Impact

These enhancements require significant infrastructure updates:

* GPU Resources: Additional NVIDIA L40S/H100 GPUs with 48-80GB VRAM for ML workloads
* Kubernetes Components: New deployments, services, and workflow templates
* Storage: Expanded persistent storage for parameter history and ML artifacts
* External Services: Integration with email/SMS providers for notifications

### Deployment Checklist

1. Prerequisites
2. [ ] NVIDIA GPU operators installed
3. [ ] Nodes with appropriate GPUs available
4. [ ] Persistent storage configured
5. [ ] External service credentials stored as secrets
6. Database Setup
7. [ ] Apply notification system migrations
8. [ ] Apply parameter registry migrations
9. [ ] Configure backup strategy
10. Core Services
11. [ ] Deploy notification system
12. [ ] Deploy webhook service
13. [ ] Deploy parameter registry
14. [ ] Update GPU requirements
15. ML Components
16. [ ] Deploy multimodal pattern recognition service
17. [ ] Deploy domain-specific networks service
18. [ ] Configure workflow templates
19. Integration
20. [ ] Update existing workflows to use new components
21. [ ] Configure monitoring for new services
22. [ ] Validate end-to-end functionality

[ ] External service credentials stored as secrets

Database Setup

[ ] Configure backup strategy

Core Services

[ ] Update GPU requirements

ML Components

[ ] Configure workflow templates

Integration

### Verification Steps

After deployment, verify:

1. Notification System
2. Send test notifications
3. Verify delivery across channels
4. Test webhook configurations
5. Parameter Registry
6. Query parameters for test materials
7. Test integration with ML pipelines
8. ML Features
9. Submit pattern recognition jobs
10. Verify domain-specific analysis results
11. Check integration with existing visualization

Test webhook configurations

Parameter Registry

Test integration with ML pipelines

ML Features

### Resource Requirements

Table content:

Component | CPU | Memory | GPU | Storage

Notification System | 2-4 cores | 4-8 GB | N/A | 10 GB

Webhook Service | 2-4 cores | 4-8 GB | N/A | 20 GB

Parameter Registry | 4-8 cores | 8-16 GB | N/A | 50 GB

MultiModal Pattern Recognition | 8-16 cores | 16-32 GB | 1-2 NVIDIA L40S/H100 | 100 GB

Domain-Specific Networks | 8-16 cores | 16-32 GB | 1-2 NVIDIA L40S/H100 | 100 GB

### Integration Architecture

These new components integrate with the existing KAI platform through:

1. API Endpoints: REST and gRPC interfaces for service-to-service communication
2. Workflow Templates: Argo Workflows for orchestrating complex ML pipelines
3. Event Streams: Kafka topics for asynchronous communication
4. Shared Storage: Access to common persistent volumes for ML artifacts

### Monitoring Recommendations

Key metrics to monitor:

* Notification delivery rates and latencies
* Parameter registry query performance
* GPU utilization across ML workloads
* Model inference latencies
* Workflow completion times

For detailed deployment instructions, configuration options, and rollback procedures, refer to the Deployment Updates document.

# Telemetry Service

Source: readme/telemetry-service.md

---

## Telemetry Service

This document describes the telemetry service implementation in the KAI platform. The telemetry service provides a unified interface for collecting metrics, traces, and logs across the application.

### Overview

The telemetry service is designed to collect and analyze application telemetry data, including metrics, traces, and logs. It provides a consistent API for telemetry operations, regardless of the underlying telemetry provider. The service supports both console-based telemetry for development and more advanced telemetry providers for production.

### Architecture

The telemetry service follows a provider pattern, allowing different telemetry implementations to be used interchangeably. The service consists of the following components:

1. Telemetry Service: The main service that provides a unified interface for telemetry operations.
2. Telemetry Provider Interface: An interface that defines the contract for telemetry providers.
3. Console Telemetry Provider: A provider that logs telemetry events to the console.
4. Telemetry Initializer: A utility for initializing the telemetry service with different providers.

### Usage

#### Basic Usage

import { telemetry } from '@kai/shared';  
  
// Track an event  
await telemetry.trackEvent({  
 type: 'user',  
 name: 'user\_login',  
 timestamp: Date.now(),  
 status: 'success',  
 properties: {  
 userId: '123',  
 method: 'password'  
 }  
});  
  
// Track an error  
await telemetry.trackEvent({  
 type: 'error',  
 name: 'api\_error',  
 timestamp: Date.now(),  
 status: 'error',  
 properties: {  
 endpoint: '/api/users',  
 method: 'GET'  
 },  
 error: {  
 message: 'Failed to fetch users',  
 stack: error.stack,  
 name: error.name  
 }  
});

import { telemetry } from '@kai/shared';  
  
// Track an event  
await telemetry.trackEvent({  
 type: 'user',  
 name: 'user\_login',  
 timestamp: Date.now(),  
 status: 'success',  
 properties: {  
 userId: '123',  
 method: 'password'  
 }  
});  
  
// Track an error  
await telemetry.trackEvent({  
 type: 'error',  
 name: 'api\_error',  
 timestamp: Date.now(),  
 status: 'error',  
 properties: {  
 endpoint: '/api/users',  
 method: 'GET'  
 },  
 error: {  
 message: 'Failed to fetch users',  
 stack: error.stack,  
 name: error.name  
 }  
});

#### Performance Measurements

import { telemetry } from '@kai/shared';  
  
// Manual measurement  
const id = telemetry.startMeasurement('database\_query', {  
 query: 'SELECT \* FROM users',  
 params: { limit: 10 }  
});  
  
try {  
 const result = await db.query('SELECT \* FROM users', { limit: 10 });  
 await telemetry.stopMeasurement(id, { success: true, count: result.length });  
 return result;  
} catch (error) {  
 await telemetry.stopMeasurement(id, { success: false, error: error.message });  
 throw error;  
}  
  
// Automatic measurement  
const result = await telemetry.measure(  
 'database\_query',  
 async () => {  
 return await db.query('SELECT \* FROM users', { limit: 10 });  
 },  
 {  
 query: 'SELECT \* FROM users',  
 params: { limit: 10 }  
 }  
);

import { telemetry } from '@kai/shared';  
  
// Manual measurement  
const id = telemetry.startMeasurement('database\_query', {  
 query: 'SELECT \* FROM users',  
 params: { limit: 10 }  
});  
  
try {  
 const result = await db.query('SELECT \* FROM users', { limit: 10 });  
 await telemetry.stopMeasurement(id, { success: true, count: result.length });  
 return result;  
} catch (error) {  
 await telemetry.stopMeasurement(id, { success: false, error: error.message });  
 throw error;  
}  
  
// Automatic measurement  
const result = await telemetry.measure(  
 'database\_query',  
 async () => {  
 return await db.query('SELECT \* FROM users', { limit: 10 });  
 },  
 {  
 query: 'SELECT \* FROM users',  
 params: { limit: 10 }  
 }  
);

### Configuration

The telemetry service can be configured through environment variables or the unified configuration system. The following configuration options are available:

// In .env file  
TELEMETRY\_ENABLED=true  
TELEMETRY\_TYPE=console

// In .env file  
TELEMETRY\_ENABLED=true  
TELEMETRY\_TYPE=console

### Telemetry Providers

#### Console Telemetry Provider

The console telemetry provider logs telemetry events to the console. It's suitable for development and debugging. Features include:

* Console-based logging
* Performance measurements
* Event buffering
* Error handling

### Implementation Details

#### Telemetry Service

The telemetry service provides a unified interface for telemetry operations. It delegates all operations to the configured provider and adds additional functionality like event buffering and automatic measurements.

class TelemetryService {  
 private provider: TelemetryProvider | null = null;  
 private enabled: boolean = false;  
  
 // Set the telemetry provider  
 setProvider(provider: TelemetryProvider): void;  
  
 // Enable telemetry  
 async enable(): Promise<void>;  
  
 // Disable telemetry  
 async disable(): Promise<void>;  
  
 // Track an event  
 async trackEvent(event: TelemetryEvent): Promise<void>;  
  
 // Start a performance measurement  
 startMeasurement(name: string, properties?: Record<string, any>): string;  
  
 // Stop a performance measurement  
 async stopMeasurement(id: string, additionalProperties?: Record<string, any>): Promise<void>;  
  
 // Measure the execution time of a function  
 async measure<T>(  
 name: string,  
 fn: () => Promise<T> | T,  
 properties?: Record<string, any>  
 ): Promise<T>;  
  
 // Flush all pending telemetry  
 async flush(): Promise<void>;  
}

class TelemetryService {  
 private provider: TelemetryProvider | null = null;  
 private enabled: boolean = false;  
  
 // Set the telemetry provider  
 setProvider(provider: TelemetryProvider): void;  
  
 // Enable telemetry  
 async enable(): Promise<void>;  
  
 // Disable telemetry  
 async disable(): Promise<void>;  
  
 // Track an event  
 async trackEvent(event: TelemetryEvent): Promise<void>;  
  
 // Start a performance measurement  
 startMeasurement(name: string, properties?: Record<string, any>): string;  
  
 // Stop a performance measurement  
 async stopMeasurement(id: string, additionalProperties?: Record<string, any>): Promise<void>;  
  
 // Measure the execution time of a function  
 async measure<T>(  
 name: string,  
 fn: () => Promise<T> | T,  
 properties?: Record<string, any>  
 ): Promise<T>;  
  
 // Flush all pending telemetry  
 async flush(): Promise<void>;  
}

#### Telemetry Provider Interface

The telemetry provider interface defines the contract for telemetry providers. All providers must implement this interface.

interface TelemetryProvider {  
 // Initialize the provider  
 initialize(): Promise<void>;  
  
 // Track an event  
 trackEvent(event: TelemetryEvent): Promise<void>;  
  
 // Start a performance measurement  
 startMeasurement(name: string, properties?: Record<string, any>): string;  
  
 // Stop a performance measurement  
 stopMeasurement(id: string, additionalProperties?: Record<string, any>): Promise<void>;  
  
 // Flush all pending telemetry  
 flush(): Promise<void>;  
}

interface TelemetryProvider {  
 // Initialize the provider  
 initialize(): Promise<void>;  
  
 // Track an event  
 trackEvent(event: TelemetryEvent): Promise<void>;  
  
 // Start a performance measurement  
 startMeasurement(name: string, properties?: Record<string, any>): string;  
  
 // Stop a performance measurement  
 stopMeasurement(id: string, additionalProperties?: Record<string, any>): Promise<void>;  
  
 // Flush all pending telemetry  
 flush(): Promise<void>;  
}

#### Telemetry Event

The telemetry event interface defines the structure of telemetry events.

interface TelemetryEvent {  
 // Event type  
 type: TelemetryEventType;  
  
 // Event name  
 name: string;  
  
 // Event timestamp  
 timestamp: number;  
  
 // Event duration in milliseconds (if applicable)  
 duration?: number;  
  
 // Event status (success, error, etc.)  
 status?: 'success' | 'error' | 'warning' | 'info';  
  
 // Event properties  
 properties?: Record<string, any>;  
  
 // Event measurements  
 measurements?: Record<string, number>;  
  
 // Error information (if applicable)  
 error?: {  
 message: string;  
 stack?: string;  
 code?: string;  
 name?: string;  
 };  
}

interface TelemetryEvent {  
 // Event type  
 type: TelemetryEventType;  
  
 // Event name  
 name: string;  
  
 // Event timestamp  
 timestamp: number;  
  
 // Event duration in milliseconds (if applicable)  
 duration?: number;  
  
 // Event status (success, error, etc.)  
 status?: 'success' | 'error' | 'warning' | 'info';  
  
 // Event properties  
 properties?: Record<string, any>;  
  
 // Event measurements  
 measurements?: Record<string, number>;  
  
 // Error information (if applicable)  
 error?: {  
 message: string;  
 stack?: string;  
 code?: string;  
 name?: string;  
 };  
}

### Benefits

The telemetry service provides several benefits:

1. Improved Monitoring: Comprehensive telemetry data helps identify and diagnose issues.
2. Consistent API: The unified interface provides a consistent API for telemetry operations, regardless of the underlying provider.
3. Provider Flexibility: The provider pattern allows different telemetry implementations to be used interchangeably.
4. Performance Measurements: Built-in support for performance measurements helps identify bottlenecks.
5. Error Tracking: Structured error tracking helps diagnose and fix issues.
6. Event Buffering: Event buffering ensures that telemetry data is not lost if the provider is temporarily unavailable.

### Next Steps

The following steps are recommended to further improve the telemetry service:

1. Add More Providers: Add support for more telemetry providers (Application Insights, Datadog, etc.).
2. Add Distributed Tracing: Add support for distributed tracing to track requests across services.
3. Add Sampling: Add support for sampling to reduce the volume of telemetry data.
4. Add Custom Dimensions: Add support for custom dimensions to enrich telemetry data.
5. Add Correlation: Add support for correlation to link related telemetry events.
6. Add Alerting: Add support for alerting based on telemetry data.

# Testing Approach

Source: readme/testing-approach.md

---

## Testing Approach

This document outlines the comprehensive testing approach used in the Kai platform, including the different types of tests, how to run them, and best practices for implementing tests across the application. Use this guide when adding new tests to ensure consistency and proper test coverage.

### Testing Philosophy

Our testing approach is based on the following principles:

1. Test Pyramid: We follow the test pyramid approach, with more unit tests than integration tests, and more integration tests than end-to-end tests.
2. Test Coverage: We aim for high test coverage, but prioritize testing critical paths and business logic.
3. Test Independence: Tests should be independent of each other and should not rely on the state of other tests.
4. Test Readability: Tests should be easy to read and understand, with clear assertions and minimal setup.
5. Test Maintainability: Tests should be easy to maintain and should not break when implementation details change.

### Types of Tests

The Kai platform uses a multi-layered testing approach to ensure comprehensive test coverage. Each type of test serves a specific purpose and should be used in appropriate contexts.

#### Unit Tests

Unit tests focus on testing individual functions, methods, or classes in isolation. They are fast, reliable, and provide quick feedback during development.

When to use:  
- Testing business logic in services  
- Validating utility functions  
- Verifying controller methods  
- Testing data transformations  
- Validating model methods

Example:

// Testing a utility function  
describe('formatCurrency', () => {  
 it('should format currency correctly', () => {  
 expect(formatCurrency(1000)).toBe('$1,000.00');  
 });  
});

// Testing a utility function  
describe('formatCurrency', () => {  
 it('should format currency correctly', () => {  
 expect(formatCurrency(1000)).toBe('$1,000.00');  
 });  
});

#### Integration Tests

Integration tests verify that different parts of the application work together correctly. They test the integration between components, services, and external dependencies.

When to use:  
- Testing API endpoints with actual database interactions  
- Verifying service-to-service communication  
- Testing database queries and transactions  
- Validating middleware chains  
- Testing authentication flows  
- Verifying event handling across components

Example:

// Testing an API endpoint  
describe('User API', () => {  
 it('should create a new user', async () => {  
 const response = await request(app)  
 .post('/api/users')  
 .send({ name: 'John Doe', email: 'john@example.com' });  
  
 expect(response.status).toBe(201);  
 expect(response.body.name).toBe('John Doe');  
 });  
});

// Testing an API endpoint  
describe('User API', () => {  
 it('should create a new user', async () => {  
 const response = await request(app)  
 .post('/api/users')  
 .send({ name: 'John Doe', email: 'john@example.com' });  
  
 expect(response.status).toBe(201);  
 expect(response.body.name).toBe('John Doe');  
 });  
});

#### Contract Tests

Contract tests verify that the integration between our application and external services follows the agreed-upon contract. They ensure that our application correctly formats requests and handles responses from external services.

When to use:  
- Testing integrations with the MCP server  
- Verifying Supabase API interactions  
- Testing third-party API integrations  
- Validating webhook implementations  
- Testing payment processor integrations  
- Verifying authentication provider integrations

Example:

// Testing MCP integration  
describe('MCP Integration', () => {  
 it('should format requests according to MCP contract', async () => {  
 await mcpClientService.generateTimeSeriesForecast(  
 'user-123',  
 {  
 eventType: 'search',  
 startDate: '2023-01-01T00:00:00Z',  
 endDate: '2023-01-02T00:00:00Z',  
 forecastPeriods: 7,  
 interval: 'day'  
 }  
 );  
  
 expect(axios.post).toHaveBeenCalledWith(  
 'http://mcp-server.example.com/api/v1/analytics/forecast',  
 {  
 event\_type: 'search',  
 start\_date: '2023-01-01T00:00:00Z',  
 end\_date: '2023-01-02T00:00:00Z',  
 forecast\_periods: 7,  
 interval: 'day'  
 },  
 expect.any(Object)  
 );  
 });  
});

// Testing MCP integration  
describe('MCP Integration', () => {  
 it('should format requests according to MCP contract', async () => {  
 await mcpClientService.generateTimeSeriesForecast(  
 'user-123',  
 {  
 eventType: 'search',  
 startDate: '2023-01-01T00:00:00Z',  
 endDate: '2023-01-02T00:00:00Z',  
 forecastPeriods: 7,  
 interval: 'day'  
 }  
 );  
  
 expect(axios.post).toHaveBeenCalledWith(  
 'http://mcp-server.example.com/api/v1/analytics/forecast',  
 {  
 event\_type: 'search',  
 start\_date: '2023-01-01T00:00:00Z',  
 end\_date: '2023-01-02T00:00:00Z',  
 forecast\_periods: 7,  
 interval: 'day'  
 },  
 expect.any(Object)  
 );  
 });  
});

#### Component Tests

Component tests verify that React components render correctly and respond to user interactions as expected. They use React Testing Library to simulate user interactions and verify the rendered output.

When to use:  
- Testing UI components in isolation  
- Verifying component rendering logic  
- Testing user interactions (clicks, form inputs)  
- Validating component state changes  
- Testing component lifecycle behavior  
- Verifying accessibility compliance

Example:

// Testing a React component  
describe('Button', () => {  
 it('should call onClick when clicked', () => {  
 const onClick = jest.fn();  
 render(<Button onClick={onClick}>Click me</Button>);  
  
 fireEvent.click(screen.getByText('Click me'));  
  
 expect(onClick).toHaveBeenCalled();  
 });  
});

// Testing a React component  
describe('Button', () => {  
 it('should call onClick when clicked', () => {  
 const onClick = jest.fn();  
 render(<Button onClick={onClick}>Click me</Button>);  
  
 fireEvent.click(screen.getByText('Click me'));  
  
 expect(onClick).toHaveBeenCalled();  
 });  
});

#### End-to-End Tests

End-to-end (E2E) tests verify that the entire application works correctly from the user's perspective. They simulate real user scenarios by interacting with the application through the UI.

When to use:  
- Testing critical user flows (registration, login, checkout)  
- Verifying multi-step processes  
- Testing cross-component interactions  
- Validating application behavior in production-like environments  
- Testing browser compatibility

Example:

// Testing a user registration flow with Cypress  
describe('User Registration', () => {  
 it('should allow a new user to register', () => {  
 cy.visit('/register');  
 cy.get('input[name="email"]').type('test@example.com');  
 cy.get('input[name="password"]').type('securePassword123');  
 cy.get('input[name="confirmPassword"]').type('securePassword123');  
 cy.get('button[type="submit"]').click();  
  
 // Verify successful registration  
 cy.url().should('include', '/dashboard');  
 cy.get('[data-testid="welcome-message"]').should('contain', 'Welcome');  
 });  
});

// Testing a user registration flow with Cypress  
describe('User Registration', () => {  
 it('should allow a new user to register', () => {  
 cy.visit('/register');  
 cy.get('input[name="email"]').type('test@example.com');  
 cy.get('input[name="password"]').type('securePassword123');  
 cy.get('input[name="confirmPassword"]').type('securePassword123');  
 cy.get('button[type="submit"]').click();  
  
 // Verify successful registration  
 cy.url().should('include', '/dashboard');  
 cy.get('[data-testid="welcome-message"]').should('contain', 'Welcome');  
 });  
});

#### Visual Regression Tests

Visual regression tests capture screenshots of components and pages and compare them to baseline images to detect visual changes.

When to use:  
- Testing UI components for visual consistency  
- Verifying layout across different screen sizes  
- Detecting unintended visual changes  
- Testing theme implementations

Example:

// Using Storybook and Chromatic for visual testing  
describe('Button Component', () => {  
 it('should match visual snapshot', async () => {  
 // Capture screenshot and compare to baseline  
 const image = await page.screenshot();  
 expect(image).toMatchImageSnapshot();  
 });  
});

// Using Storybook and Chromatic for visual testing  
describe('Button Component', () => {  
 it('should match visual snapshot', async () => {  
 // Capture screenshot and compare to baseline  
 const image = await page.screenshot();  
 expect(image).toMatchImageSnapshot();  
 });  
});

### Running Tests

#### Server Tests

# Run all tests  
yarn test  
  
# Run unit tests only  
yarn test:unit  
  
# Run integration tests only  
yarn test:integration  
  
# Run contract tests only  
yarn test:contract

# Run all tests  
yarn test  
  
# Run unit tests only  
yarn test:unit  
  
# Run integration tests only  
yarn test:integration  
  
# Run contract tests only  
yarn test:contract

#### Admin Tests

# Run all tests  
yarn test  
  
# Run component tests only  
yarn test:components  
  
# Run analytics component tests only  
yarn test:analytics

# Run all tests  
yarn test  
  
# Run component tests only  
yarn test:components  
  
# Run analytics component tests only  
yarn test:analytics

#### Client Tests

# Run all tests  
yarn workspace @kai/client test  
  
# Run component tests only  
yarn workspace @kai/client test:components  
  
# Run with coverage  
yarn workspace @kai/client test --coverage

# Run all tests  
yarn workspace @kai/client test  
  
# Run component tests only  
yarn workspace @kai/client test:components  
  
# Run with coverage  
yarn workspace @kai/client test --coverage

#### End-to-End Tests

# Start the E2E testing environment  
yarn e2e:setup  
  
# Run E2E tests  
yarn e2e  
  
# Run specific E2E test  
yarn e2e --spec "cypress/integration/login.spec.js"

# Start the E2E testing environment  
yarn e2e:setup  
  
# Run E2E tests  
yarn e2e  
  
# Run specific E2E test  
yarn e2e --spec "cypress/integration/login.spec.js"

### Test Structure

#### Test File Organization

##### Server Package

* Unit tests: src/tests/\*.test.ts
* Integration tests: src/tests/integration/\*.test.ts
* Contract tests: src/tests/contract/\*.test.ts
* API tests: src/tests/api/\*.test.ts

src/tests/\*.test.ts

src/tests/integration/\*.test.ts

src/tests/contract/\*.test.ts

src/tests/api/\*.test.ts

##### Admin Package

* Component tests: src/components/\*\*/\_\_tests\_\_/\*.test.tsx
* Page tests: src/pages/\*\*/\_\_tests\_\_/\*.test.tsx
* Hook tests: src/hooks/\*\*/\_\_tests\_\_/\*.test.tsx
* Utility tests: src/utils/\*\*/\_\_tests\_\_/\*.test.ts

src/components/\*\*/\_\_tests\_\_/\*.test.tsx

src/pages/\*\*/\_\_tests\_\_/\*.test.tsx

src/hooks/\*\*/\_\_tests\_\_/\*.test.tsx

src/utils/\*\*/\_\_tests\_\_/\*.test.ts

##### Client Package

* Component tests: src/components/\*\*/\_\_tests\_\_/\*.test.tsx
* Page tests: src/pages/\*\*/\_\_tests\_\_/\*.test.tsx
* Hook tests: src/hooks/\*\*/\_\_tests\_\_/\*.test.tsx
* Utility tests: src/utils/\*\*/\_\_tests\_\_/\*.test.ts

src/components/\*\*/\_\_tests\_\_/\*.test.tsx

src/pages/\*\*/\_\_tests\_\_/\*.test.tsx

src/hooks/\*\*/\_\_tests\_\_/\*.test.tsx

src/utils/\*\*/\_\_tests\_\_/\*.test.ts

##### Shared Package

* Utility tests: src/utils/\*\*/\_\_tests\_\_/\*.test.ts
* Type tests: src/types/\*\*/\_\_tests\_\_/\*.test.ts

src/utils/\*\*/\_\_tests\_\_/\*.test.ts

src/types/\*\*/\_\_tests\_\_/\*.test.ts

##### End-to-End Tests

* E2E tests: cypress/integration/\*\*/\*.spec.js

cypress/integration/\*\*/\*.spec.js

#### Test File Naming

* Unit tests: \*.test.ts
* Integration tests: \*.integration.test.ts
* Contract tests: \*.contract.test.ts
* Component tests: \*.test.tsx
* E2E tests: \*.spec.js

\*.test.ts

\*.integration.test.ts

\*.contract.test.ts

\*.test.tsx

\*.spec.js

#### Test Directory Structure

Tests should be organized to mirror the structure of the code they're testing:

src/  
 components/  
 Button/  
 Button.tsx  
 \_\_tests\_\_/  
 Button.test.tsx  
 services/  
 userService.ts  
 \_\_tests\_\_/  
 userService.test.ts  
 utils/  
 formatters.ts  
 \_\_tests\_\_/  
 formatters.test.ts

src/  
 components/  
 Button/  
 Button.tsx  
 \_\_tests\_\_/  
 Button.test.tsx  
 services/  
 userService.ts  
 \_\_tests\_\_/  
 userService.test.ts  
 utils/  
 formatters.ts  
 \_\_tests\_\_/  
 formatters.test.ts

### Best Practices

#### General Best Practices

1. Use descriptive test names: Test names should describe what the test is testing and what the expected outcome is.  
    ```typescript  
    // Good  
    it('should return 404 when user is not found', () => {...});

// Bad  
 it('test user not found', () => {...});  
 ```

1. Keep tests small and focused: Each test should test one thing and have a clear purpose.  
    ```typescript  
    // Good - separate tests for different behaviors  
    it('should validate email format', () => {...});  
    it('should validate password length', () => {...});

// Bad - testing multiple behaviors in one test  
 it('should validate form inputs', () => {...});  
 ```

1. Use setup and teardown: Use beforeEach and afterEach to set up and tear down test state.  
    ```typescript  
    describe('UserService', () => {  
    let userService: UserService;  
   beforeEach(() => {  
    userService = new UserService();  
    });  
   afterEach(() => {  
    // Clean up resources  
    });  
   it('should create a user', () => {...});  
    });  
    ```
2. Mock external dependencies: Use Jest's mocking capabilities to mock external dependencies.  
    typescript  
    jest.mock('../../services/databaseService', () => ({  
    query: jest.fn().mockResolvedValue([{ id: 1, name: 'Test User' }])  
    }));
3. Use test data factories: Create factory functions to generate test data.  
    typescript  
    const createTestUser = (overrides = {}) => ({  
    id: 1,  
    name: 'Test User',  
    email: 'test@example.com',  
    ...overrides  
    });
4. Test edge cases: Test edge cases and error conditions, not just the happy path.  
    typescript  
    it('should handle empty input', () => {...});  
    it('should handle maximum input length', () => {...});  
    it('should handle special characters', () => {...});
5. Use snapshots sparingly: Use snapshots only for stable UI components, not for testing business logic.  
    typescript  
    it('should render correctly', () => {  
    const { container } = render(<Button>Click me</Button>);  
    expect(container).toMatchSnapshot();  
    });
6. Keep tests independent: Tests should not depend on the state of other tests.  
    ```typescript  
    // Good - each test sets up its own state  
    it('test A', () => {  
    const data = setupTestData();  
    // Test using data  
    });

Use setup and teardown: Use beforeEach and afterEach to set up and tear down test state.  
 ```typescript  
 describe('UserService', () => {  
 let userService: UserService;

beforeEach

afterEach

beforeEach(() => {  
 userService = new UserService();  
 });

afterEach(() => {  
 // Clean up resources  
 });

it('should create a user', () => {...});  
 });  
 ```

Mock external dependencies: Use Jest's mocking capabilities to mock external dependencies.  
 typescript  
 jest.mock('../../services/databaseService', () => ({  
 query: jest.fn().mockResolvedValue([{ id: 1, name: 'Test User' }])  
 }));

typescript  
 jest.mock('../../services/databaseService', () => ({  
 query: jest.fn().mockResolvedValue([{ id: 1, name: 'Test User' }])  
 }));

Use test data factories: Create factory functions to generate test data.  
 typescript  
 const createTestUser = (overrides = {}) => ({  
 id: 1,  
 name: 'Test User',  
 email: 'test@example.com',  
 ...overrides  
 });

typescript  
 const createTestUser = (overrides = {}) => ({  
 id: 1,  
 name: 'Test User',  
 email: 'test@example.com',  
 ...overrides  
 });

Test edge cases: Test edge cases and error conditions, not just the happy path.  
 typescript  
 it('should handle empty input', () => {...});  
 it('should handle maximum input length', () => {...});  
 it('should handle special characters', () => {...});

typescript  
 it('should handle empty input', () => {...});  
 it('should handle maximum input length', () => {...});  
 it('should handle special characters', () => {...});

Use snapshots sparingly: Use snapshots only for stable UI components, not for testing business logic.  
 typescript  
 it('should render correctly', () => {  
 const { container } = render(<Button>Click me</Button>);  
 expect(container).toMatchSnapshot();  
 });

typescript  
 it('should render correctly', () => {  
 const { container } = render(<Button>Click me</Button>);  
 expect(container).toMatchSnapshot();  
 });

Keep tests independent: Tests should not depend on the state of other tests.  
 ```typescript  
 // Good - each test sets up its own state  
 it('test A', () => {  
 const data = setupTestData();  
 // Test using data  
 });

it('test B', () => {  
 const data = setupTestData();  
 // Test using data  
 });  
 ```

1. Test behavior, not implementation: Test what the code does, not how it does it.  
    typescript  
    // Good - testing behavior  
    it('should show error message when login fails', async () => {  
    render(<LoginForm />);  
    fireEvent.click(screen.getByText('Login'));  
    expect(await screen.findByText('Invalid credentials')).toBeInTheDocument();  
    });
2. Write tests first: Consider writing tests before implementing features (TDD).

Test behavior, not implementation: Test what the code does, not how it does it.  
 typescript  
 // Good - testing behavior  
 it('should show error message when login fails', async () => {  
 render(<LoginForm />);  
 fireEvent.click(screen.getByText('Login'));  
 expect(await screen.findByText('Invalid credentials')).toBeInTheDocument();  
 });

typescript  
 // Good - testing behavior  
 it('should show error message when login fails', async () => {  
 render(<LoginForm />);  
 fireEvent.click(screen.getByText('Login'));  
 expect(await screen.findByText('Invalid credentials')).toBeInTheDocument();  
 });

Write tests first: Consider writing tests before implementing features (TDD).

#### Package-Specific Best Practices

##### Server Package

1. Test database interactions with real queries: Use a test database for integration tests.  
    ```typescript  
    it('should save user to database', async () => {  
    const user = createTestUser();  
    await userService.createUser(user);  
   const savedUser = await db.query('SELECT \* FROM users WHERE id = $1', [user.id]);  
    expect(savedUser).toEqual(user);  
    });  
    ```
2. Test API endpoints with supertest: Use supertest to test API endpoints.  
    ``typescript  
    it('should return 200 for valid request', async () => {  
    const response = await request(app)  
    .get('/api/users/1')  
    .set('Authorization',Bearer ${testToken}`);  
   expect(response.status).toBe(200);  
    expect(response.body).toEqual(expect.objectContaining({  
    id: 1,  
    name: expect.any(String)  
    }));  
    });  
    ```
3. Test middleware in isolation: Test middleware functions separately from routes.  
    ```typescript  
    it('should call next() for authenticated user', () => {  
    const req = { user: { id: 1 } };  
    const res = {};  
    const next = jest.fn();  
   authMiddleware(req, res, next);  
   expect(next).toHaveBeenCalled();  
    });  
    ```

Test database interactions with real queries: Use a test database for integration tests.  
 ```typescript  
 it('should save user to database', async () => {  
 const user = createTestUser();  
 await userService.createUser(user);

const savedUser = await db.query('SELECT \* FROM users WHERE id = $1', [user.id]);  
 expect(savedUser).toEqual(user);  
 });  
 ```

Test API endpoints with supertest: Use supertest to test API endpoints.  
 ``typescript  
 it('should return 200 for valid request', async () => {  
 const response = await request(app)  
 .get('/api/users/1')  
 .set('Authorization',Bearer ${testToken}`);

``typescript  
 it('should return 200 for valid request', async () => {  
 const response = await request(app)  
 .get('/api/users/1')  
 .set('Authorization',

expect(response.status).toBe(200);  
 expect(response.body).toEqual(expect.objectContaining({  
 id: 1,  
 name: expect.any(String)  
 }));  
 });  
 ```

Test middleware in isolation: Test middleware functions separately from routes.  
 ```typescript  
 it('should call next() for authenticated user', () => {  
 const req = { user: { id: 1 } };  
 const res = {};  
 const next = jest.fn();

authMiddleware(req, res, next);

expect(next).toHaveBeenCalled();  
 });  
 ```

##### Admin/Client Packages

1. Test component rendering: Verify that components render correctly.  
    typescript  
    it('should render the component', () => {  
    render(<Button>Click me</Button>);  
    expect(screen.getByText('Click me')).toBeInTheDocument();  
    });
2. Test user interactions: Verify that components respond to user interactions.  
    ```typescript  
    it('should call onClick when clicked', () => {  
    const onClick = jest.fn();  
    render(Click me);  
   fireEvent.click(screen.getByText('Click me'));  
   expect(onClick).toHaveBeenCalled();  
    });  
    ```
3. Test form submissions: Verify that forms submit correctly.  
    ```typescript  
    it('should submit the form with correct values', async () => {  
    const onSubmit = jest.fn();  
    render();  
   fireEvent.change(screen.getByLabelText('Email'), {  
    target: { value: 'test@example.com' }  
    });  
   fireEvent.change(screen.getByLabelText('Password'), {  
    target: { value: 'password123' }  
    });  
   fireEvent.click(screen.getByText('Login'));  
   expect(onSubmit).toHaveBeenCalledWith({  
    email: 'test@example.com',  
    password: 'password123'  
    });  
    });  
    ```
4. Test hooks with renderHook: Use renderHook to test custom hooks.  
    ```typescript  
    it('should update count when increment is called', () => {  
    const { result } = renderHook(() => useCounter());  
   act(() => {  
    result.current.increment();  
    });  
   expect(result.current.count).toBe(1);  
    });  
    ```

Test component rendering: Verify that components render correctly.  
 typescript  
 it('should render the component', () => {  
 render(<Button>Click me</Button>);  
 expect(screen.getByText('Click me')).toBeInTheDocument();  
 });

typescript  
 it('should render the component', () => {  
 render(<Button>Click me</Button>);  
 expect(screen.getByText('Click me')).toBeInTheDocument();  
 });

Test user interactions: Verify that components respond to user interactions.  
 ```typescript  
 it('should call onClick when clicked', () => {  
 const onClick = jest.fn();  
 render(Click me);

fireEvent.click(screen.getByText('Click me'));

expect(onClick).toHaveBeenCalled();  
 });  
 ```

Test form submissions: Verify that forms submit correctly.  
 ```typescript  
 it('should submit the form with correct values', async () => {  
 const onSubmit = jest.fn();  
 render();

fireEvent.change(screen.getByLabelText('Email'), {  
 target: { value: 'test@example.com' }  
 });

fireEvent.change(screen.getByLabelText('Password'), {  
 target: { value: 'password123' }  
 });

fireEvent.click(screen.getByText('Login'));

expect(onSubmit).toHaveBeenCalledWith({  
 email: 'test@example.com',  
 password: 'password123'  
 });  
 });  
 ```

Test hooks with renderHook: Use renderHook to test custom hooks.  
 ```typescript  
 it('should update count when increment is called', () => {  
 const { result } = renderHook(() => useCounter());

act(() => {  
 result.current.increment();  
 });

expect(result.current.count).toBe(1);  
 });  
 ```

### Mocking

#### Mocking External Dependencies

// Mock axios  
jest.mock('axios');  
  
// Mock a service  
jest.mock('../../services/userService', () => ({  
 getUser: jest.fn().mockResolvedValue({ id: 1, name: 'John Doe' }),  
}));

// Mock axios  
jest.mock('axios');  
  
// Mock a service  
jest.mock('../../services/userService', () => ({  
 getUser: jest.fn().mockResolvedValue({ id: 1, name: 'John Doe' }),  
}));

#### Mocking React Components

// Mock a React component  
jest.mock('../../components/Button', () => ({  
 \_\_esModule: true,  
 default: (props) => <button {...props} data-testid="mocked-button" />,  
}));

// Mock a React component  
jest.mock('../../components/Button', () => ({  
 \_\_esModule: true,  
 default: (props) => <button {...props} data-testid="mocked-button" />,  
}));

#### Mocking Supabase

// Mock Supabase client  
jest.mock('../../services/supabase/supabaseClient', () => ({  
 supabase: {  
 from: jest.fn().mockReturnThis(),  
 select: jest.fn().mockReturnThis(),  
 eq: jest.fn().mockReturnThis(),  
 single: jest.fn().mockResolvedValue({  
 data: { id: 1, name: 'Test User' },  
 error: null  
 }),  
 insert: jest.fn().mockResolvedValue({  
 data: { id: 1 },  
 error: null  
 }),  
 update: jest.fn().mockResolvedValue({  
 data: { id: 1 },  
 error: null  
 }),  
 delete: jest.fn().mockResolvedValue({  
 data: {},  
 error: null  
 })  
 }  
}));

// Mock Supabase client  
jest.mock('../../services/supabase/supabaseClient', () => ({  
 supabase: {  
 from: jest.fn().mockReturnThis(),  
 select: jest.fn().mockReturnThis(),  
 eq: jest.fn().mockReturnThis(),  
 single: jest.fn().mockResolvedValue({  
 data: { id: 1, name: 'Test User' },  
 error: null  
 }),  
 insert: jest.fn().mockResolvedValue({  
 data: { id: 1 },  
 error: null  
 }),  
 update: jest.fn().mockResolvedValue({  
 data: { id: 1 },  
 error: null  
 }),  
 delete: jest.fn().mockResolvedValue({  
 data: {},  
 error: null  
 })  
 }  
}));

#### Mocking MCP Client

// Mock MCP client  
jest.mock('../../services/mcp/mcpClientService', () => ({  
 isMCPAvailable: jest.fn().mockResolvedValue(true),  
 generateTimeSeriesForecast: jest.fn().mockResolvedValue({  
 historical: [],  
 forecast: [],  
 modelInfo: { name: 'TestModel', version: '1.0' }  
 }),  
 detectAnalyticsAnomalies: jest.fn().mockResolvedValue({  
 timeSeries: [],  
 anomalies: [],  
 statistics: { mean: 10, stdDev: 2 }  
 }),  
 predictUserBehavior: jest.fn().mockResolvedValue({  
 userId: 'test-user',  
 predictions: [],  
 userInsights: { activityLevel: 'medium' }  
 })  
}));

// Mock MCP client  
jest.mock('../../services/mcp/mcpClientService', () => ({  
 isMCPAvailable: jest.fn().mockResolvedValue(true),  
 generateTimeSeriesForecast: jest.fn().mockResolvedValue({  
 historical: [],  
 forecast: [],  
 modelInfo: { name: 'TestModel', version: '1.0' }  
 }),  
 detectAnalyticsAnomalies: jest.fn().mockResolvedValue({  
 timeSeries: [],  
 anomalies: [],  
 statistics: { mean: 10, stdDev: 2 }  
 }),  
 predictUserBehavior: jest.fn().mockResolvedValue({  
 userId: 'test-user',  
 predictions: [],  
 userInsights: { activityLevel: 'medium' }  
 })  
}));

#### Mocking Environment Variables

// Mock environment variables  
const originalEnv = process.env;  
  
beforeEach(() => {  
 jest.resetModules();  
 process.env = { ...originalEnv };  
 process.env.API\_URL = 'http://test-api.example.com';  
 process.env.JWT\_SECRET = 'test-jwt-secret';  
});  
  
afterEach(() => {  
 process.env = originalEnv;  
});

// Mock environment variables  
const originalEnv = process.env;  
  
beforeEach(() => {  
 jest.resetModules();  
 process.env = { ...originalEnv };  
 process.env.API\_URL = 'http://test-api.example.com';  
 process.env.JWT\_SECRET = 'test-jwt-secret';  
});  
  
afterEach(() => {  
 process.env = originalEnv;  
});

### Testing Tools

#### Jest

Jest is our primary testing framework for both frontend and backend tests. It provides a comprehensive testing solution with built-in assertion library, mocking capabilities, and code coverage reporting.

Configuration:  
- Server: packages/server/jest.config.js  
- Admin: packages/admin/jest.config.js  
- Client: packages/client/jest.config.js  
- Shared: packages/shared/jest.config.js

packages/server/jest.config.js

packages/admin/jest.config.js

packages/client/jest.config.js

packages/shared/jest.config.js

#### React Testing Library

React Testing Library is used for testing React components. It encourages testing components from the user's perspective, focusing on behavior rather than implementation details.

Example:

import { render, screen, fireEvent } from '@testing-library/react';  
import Button from '../Button';  
  
describe('Button', () => {  
 it('should render correctly', () => {  
 render(<Button>Click me</Button>);  
 expect(screen.getByText('Click me')).toBeInTheDocument();  
 });  
});

import { render, screen, fireEvent } from '@testing-library/react';  
import Button from '../Button';  
  
describe('Button', () => {  
 it('should render correctly', () => {  
 render(<Button>Click me</Button>);  
 expect(screen.getByText('Click me')).toBeInTheDocument();  
 });  
});

#### Supertest

Supertest is used for testing HTTP endpoints in the server package. It provides a high-level abstraction for testing HTTP requests and responses.

Example:

import request from 'supertest';  
import app from '../../app';  
  
describe('User API', () => {  
 it('should return user data', async () => {  
 const response = await request(app)  
 .get('/api/users/1')  
 .set('Authorization', `Bearer ${testToken}`);  
  
 expect(response.status).toBe(200);  
 expect(response.body).toHaveProperty('name');  
 });  
});

import request from 'supertest';  
import app from '../../app';  
  
describe('User API', () => {  
 it('should return user data', async () => {  
 const response = await request(app)  
 .get('/api/users/1')  
 .set('Authorization', `Bearer ${testToken}`);  
  
 expect(response.status).toBe(200);  
 expect(response.body).toHaveProperty('name');  
 });  
});

#### Cypress

Cypress is used for end-to-end testing. It allows testing the application from the user's perspective by automating browser interactions.

Example:

describe('Login Flow', () => {  
 it('should log in successfully', () => {  
 cy.visit('/login');  
 cy.get('input[name="email"]').type('test@example.com');  
 cy.get('input[name="password"]').type('password123');  
 cy.get('button[type="submit"]').click();  
 cy.url().should('include', '/dashboard');  
 });  
});

describe('Login Flow', () => {  
 it('should log in successfully', () => {  
 cy.visit('/login');  
 cy.get('input[name="email"]').type('test@example.com');  
 cy.get('input[name="password"]').type('password123');  
 cy.get('button[type="submit"]').click();  
 cy.url().should('include', '/dashboard');  
 });  
});

### Continuous Integration

Tests are run automatically on every pull request and push to the main branch using GitHub Actions. The CI pipeline will fail if any tests fail, ensuring that only code with passing tests is merged.

CI Workflow:  
- Unit and integration tests run on every PR  
- Contract tests run on every PR  
- Component tests run on every PR  
- E2E tests run on selected PRs (tagged with e2e-test)

e2e-test

### Test Coverage

We use Jest's built-in coverage reporting to track test coverage. Coverage reports are generated after running tests and can be viewed in the coverage directory.

coverage

# Generate coverage report  
yarn test -- --coverage

# Generate coverage report  
yarn test -- --coverage

Coverage Targets:  
- Statements: 80%  
- Branches: 75%  
- Functions: 80%  
- Lines: 80%

### Testing Decision Guide

Use this guide to determine which type of test to write for different parts of the application:

Table content:

Component Type | Primary Test Type | Secondary Test Type | Example

UI Components | Component Tests | Visual Regression | Button, Card, Modal

Pages | Component Tests | E2E Tests | Dashboard, Login, Profile

Hooks | Unit Tests | Integration Tests | useAuth, useForm, useData

Utilities | Unit Tests | - | formatDate, validateEmail

API Controllers | Unit Tests | Integration Tests | UserController, AuthController

Services | Unit Tests | Integration Tests | UserService, AuthService

Database Models | Integration Tests | - | User, Material, Collection

Middleware | Unit Tests | Integration Tests | Auth, Logging, Error Handling

External Integrations | Contract Tests | Integration Tests | MCP, Payment Processors

Critical Flows | E2E Tests | - | Registration, Checkout, Material Upload

### Conclusion

Following this testing approach ensures that our application is well-tested, reliable, and maintainable. By using a combination of unit tests, integration tests, contract tests, component tests, and end-to-end tests, we can have confidence that our application works as expected and that changes don't introduce regressions.

When adding new features or modifying existing ones, refer to this guide to determine the appropriate testing strategy. Remember that tests are an investment in the long-term health of the codebase, and the time spent writing good tests will pay off in reduced bugs and easier maintenance.

# Text To 3D Generation

Source: readme/text-to-3d-generation.md

---

## Text-to-3D Generation System

A comprehensive system for generating 3D house models from text descriptions, combining advanced AI models and techniques for realistic and physically accurate results.

### Core Components

#### 1. House Outline Generation

* ControlNet + Stable Diffusion for initial architectural sketching
* Edge detection and guidance for accurate outlines
* Architectural feasibility validation
* Sketch refinement with professional blueprint styling

#### 2. Improved Text-to-3D Models

We've replaced our previous stack (Stable Diffusion + Shap-E + GET3D) with newer, more advanced models:

##### Triposr

* Single-view reconstruction with high fidelity and speed
* Significantly improved geometric accuracy over Shap-E
* Better texture detail preservation
* Faster reconstruction times (minutes vs. hours)
* Hardware requirements: 8GB+ VRAM, CUDA-compatible GPU

##### Wonder3D

* High-quality 3D assets from single images
* Exceptional texture detail and consistency
* Multi-view generation from single-view input
* Specialized for detailed object reconstruction
* Hardware requirements: 10GB+ VRAM, CUDA-compatible GPU

##### Instant3D

* Generates detailed 3D models directly from text
* Higher geometric accuracy than previous text-to-3D pipelines
* Improved surface details and material properties
* Faster generation pipeline
* Hardware requirements: 12GB+ VRAM, CUDA-compatible GPU

#### 3. Object & Furniture Integration

* Integration with multiple generation models
* 3D-FRONT dataset for reference and training
* CLIP-based validation for style matching
* Furniture optimization and placement

#### 4. Scene Layout & Physics

* DiffuScene/SceneDiffuser for layout optimization
* PyBullet physics-based validation
* Graph-based planning for multi-level homes
* Manual adjustment capabilities

### Technical Implementation

#### House Outline Generation

Text Description → ControlNet Sketch → Stable Diffusion Refinement → Architectural Blueprint

Text Description → ControlNet Sketch → Stable Diffusion Refinement → Architectural Blueprint

Key features:  
- Canny edge detection for architectural guidance  
- Professional blueprint style enforcement  
- Architectural feasibility validation

#### House Shell Generation

Text → Shap-E Model → Base Structure → Refinement → Final Shell

Text → Shap-E Model → Base Structure → Refinement → Final Shell

Features:  
- Feature-preserving mesh processing  
- Normal computation  
- UV mapping for texturing  
- Interactive refinement

#### Furniture Generation & Placement

Text → GET3D → 3D-FRONT Reference → CLIP Validation → Optimized Furniture

Text → GET3D → 3D-FRONT Reference → CLIP Validation → Optimized Furniture

Capabilities:  
- Style-matched furniture generation  
- Physics-based placement validation  
- Multi-level planning support

#### Scene Optimization

Layout → DiffuScene → Physics Validation → Final Scene

Layout → DiffuScene → Physics Validation → Final Scene

Features:  
- Graph-based room connectivity  
- Physics-based stability checking  
- Manual adjustment support

### Integration with External Models

#### ControlNet Integration

* Uses sd-controlnet-canny for edge detection
* Custom architectural guidance parameters
* Blueprint style enforcement

sd-controlnet-canny

#### Shap-E Integration

* Base model: openai/shap-e-base
* Custom refinement pipeline
* Feature preservation system

openai/shap-e-base

#### GET3D Integration

* Base model: nvidia/get3d-base
* 3D-FRONT dataset integration
* CLIP-based validation

nvidia/get3d-base

#### DiffuScene Integration

* Scene optimization with physics
* Multi-level planning support
* PyBullet physics validation

### System Requirements

#### Hardware Requirements

* GPU memory requirements:
* ControlNet + Stable Diffusion: ~8GB
* Shap-E: ~6GB
* GET3D: ~8GB
* DiffuScene: ~4GB
* CPU requirements:
* Multi-core processor recommended
* 16GB+ RAM for large scenes
* Fast storage for model weights
* Network requirements:
* Initial model downloads: ~20GB
* Runtime API calls for style matching

DiffuScene: ~4GB

CPU requirements:

Fast storage for model weights

Network requirements:

### Model Weights and Dependencies

#### Required Models

* ControlNet: lllyasviel/sd-controlnet-canny
* Stable Diffusion: runwayml/stable-diffusion-v1-5
* Shap-E: openai/shap-e-base
* GET3D: nvidia/get3d-base
* CLIP: openai/clip-vit-base-patch32
* DiffuScene: scene-diffuser/diffuscene-base

lllyasviel/sd-controlnet-canny

runwayml/stable-diffusion-v1-5

openai/shap-e-base

nvidia/get3d-base

openai/clip-vit-base-patch32

scene-diffuser/diffuscene-base

#### Dataset Requirements

* 3D-FRONT dataset for furniture reference
* House templates for architectural guidance
* Style reference database

### Physics Validation

#### PyBullet Configuration

* Gravity: -9.81 m/s²
* Solver iterations: 50
* Contact breaking threshold: 0.001
* Cone friction enabled

#### Stability Checks

* Vertical movement threshold: 0.05m
* Tilt threshold: ~5.7 degrees
* Simulation duration: 4 seconds at 60Hz

### Multi-Level Planning

#### Graph-Based Approach

* Room connectivity analysis
* Level transition optimization
* Traffic flow consideration
* Clearance validation

#### Connection Types

* Stairs
* Elevators
* Open spaces
* Doorways

### Style Application

#### Geometric Patterns

* Wave patterns
* Noise patterns
* Custom deformations

#### Style Parameters

* Pattern scale
* Pattern strength
* Deformation types

### Optimization Features

#### Mesh Optimization

* Vertex count limitation (10,000 max)
* Feature preservation
* Duplicate vertex removal
* Vertex cache optimization

#### Layout Optimization

* Room connectivity
* Furniture placement
* Physics constraints
* Multi-level alignment

### Error Handling

#### Physics Validation

* Unstable placement detection
* Automatic position adjustment
* Collision resolution
* Floor contact enforcement

#### Model Fallbacks

* Alternative position sampling
* Style matching thresholds
* Geometry simplification
* Layout adjustment strategies

### Future Improvements

#### Planned Enhancements

* Real-time visualization
* Interactive refinement UI
* Additional style references
* Enhanced physics simulation

#### Research Areas

* Advanced material generation
* Dynamic furniture placement
* Improved style transfer
* Real-time optimization

### Contributing

#### Development Setup

1. Clone repository
2. Install dependencies
3. Download model weights
4. Configure environment

#### Testing

* Unit tests for components
* Integration tests for pipeline
* Physics validation tests
* Style application tests

### Rendering & Visualization Layer

#### Web-Based Visualization

* Three.js/Babylon.js integration for real-time rendering
* WebGL-based rendering pipeline
* Custom shaders for material visualization
* Progressive loading for large scenes
* Gaussian Splatting rendering support for high-performance visualization

#### User Interaction

* Orbit controls for camera navigation
* Object transformation controls (translate, rotate, scale)
* Scene hierarchy manipulation
* Real-time property editing

#### Real-Time Preview System

* Progressive refinement rendering
* Enhanced MaterialX-based material preview
* Procedural material generation capabilities
* Lighting preview with real-time shadows
* Interactive furniture placement

#### Export Capabilities

* GLB/GLTF export with full material support
* FBX export with scene hierarchy preservation
* OBJ export with material and UV preservation
* Custom metadata preservation across formats

#### Performance Optimization

* BVH (Bounding Volume Hierarchy) implementation
* Spatial partitioning for ray tracing
* Dynamic updates for scene modifications
* Optimized intersection testing
* Level-of-Detail (LOD) management

#### WebXR Integration

* VR mode with motion controls
* AR mode for real-world visualization
* Hand tracking for natural interaction
* Spatial anchoring for AR placement

#### Technical Details

##### Rendering Pipeline

Scene Graph → BVH Update → Frustum Culling → Draw Call Optimization → WebGL Render

Scene Graph → BVH Update → Frustum Culling → Draw Call Optimization → WebGL Render

##### BVH Implementation

* Dynamic BVH construction
* Surface Area Heuristic (SAH)
* Parallel BVH traversal
* Automatic rebalancing

##### WebXR Features

* 6-DOF tracking
* Controller ray-casting
* Occlusion handling
* Spatial mapping

##### Export Process

Scene → Metadata Collection → Format-Specific Optimization → Buffer Generation → File Export

Scene → Metadata Collection → Format-Specific Optimization → Buffer Generation → File Export

#### Performance Considerations

##### BVH Optimization

* Maximum tree depth: 16
* Minimum node size: 32 triangles
* Rebalance threshold: 25% imbalance
* Update frequency: Per frame for dynamic objects

##### Rendering Optimization

* Draw call batching
* Texture atlas generation
* Shader permutation management
* Occlusion culling

##### Memory Management

* Geometry instancing
* Texture streaming
* Mesh decimation
* Resource pooling

#### Development Guidelines

##### Adding New Features

1. Implement core functionality
2. Add WebXR support
3. Optimize performance
4. Add export support

##### Testing Requirements

* Performance benchmarks
* VR/AR compatibility
* Export format validation
* Cross-browser testing

### License

MIT License - See LICENSE file for details

# Threed Designer Agent

Source: readme/threeD-designer-agent.md

---

## 3D Designer Agent

This document provides detailed information about the 3D Designer Agent, a specialized crewAI agent focused on 3D visualization, design, and intelligent furniture placement within the KAI platform.

### Overview

The 3D Designer Agent enables the transformation of both images and text descriptions into detailed 3D environments. It can process 2D architectural drawings, generate room layouts from text descriptions, place furniture with physical accuracy, and refine designs based on user feedback. The agent integrates material knowledge with spatial understanding to create coherent, realistic 3D visualizations.

### Key Capabilities

The 3D Designer Agent offers multiple specialized functions:

1. 2D to 3D Conversion
2. Process architectural drawings and convert to 3D models
3. Identify room layouts, walls, windows, and doors
4. Maintain architectural proportions and standards
5. Suggest appropriate materials for detected elements
6. Text-to-3D Generation
7. Generate 3D room layouts from text descriptions
8. Create multiple interconnected rooms with correct relationships
9. Implement specified architectural styles and proportions
10. Produce physically accurate and aesthetically pleasing environments
11. Intelligent Furniture Placement
12. Position furniture with physics-based constraints
13. Ensure proper clearances and functional arrangements
14. Optimize layouts for traffic flow and usability
15. Maintain style consistency across furniture selections
16. Material Integration
17. Search and recommend appropriate materials for surfaces
18. Apply materials with correct texture mappings and scale
19. Ensure material compatibility and style coherence
20. Integrate with KAI's material database for rich selections
21. Design Refinement
22. Iteratively improve designs based on feedback
23. Modify elements while maintaining structural integrity
24. Adjust material selections and furniture placement
25. Provide explanations for design decisions and changes

Suggest appropriate materials for detected elements

Text-to-3D Generation

Produce physically accurate and aesthetically pleasing environments

Intelligent Furniture Placement

Maintain style consistency across furniture selections

Material Integration

Integrate with KAI's material database for rich selections

Design Refinement

### Architecture

The 3D Designer Agent integrates with the broader KAI platform through several key components:

#### Component Structure

packages/  
├── agents/  
│ ├── src/  
│ │ ├── frontend/  
│ │ │ └── threeDDesignerAgent.ts # Agent implementation  
│ │ ├── services/  
│ │ │ ├── 3d-designer/  
│ │ │ │ ├── threeDService.ts # 3D visualization service  
│ │ │ │ └── furniturePlacementService.ts # Furniture layout  
│ │ │ ├── materialService.ts # Material data access  
│ │ │ ├── vectorService.ts # Vector search capabilities  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── client/  
 └── src/  
 └── components/  
 └── agents/  
 ├── ThreeDDesignerPanel.tsx # Client-side interface  
 └── AgentDashboard.tsx # Agent integration in UI

packages/  
├── agents/  
│ ├── src/  
│ │ ├── frontend/  
│ │ │ └── threeDDesignerAgent.ts # Agent implementation  
│ │ ├── services/  
│ │ │ ├── 3d-designer/  
│ │ │ │ ├── threeDService.ts # 3D visualization service  
│ │ │ │ └── furniturePlacementService.ts # Furniture layout  
│ │ │ ├── materialService.ts # Material data access  
│ │ │ ├── vectorService.ts # Vector search capabilities  
│ │ │ └── serviceFactory.ts # Service creation system  
│ │ └── core/  
│ │ └── types.ts # Agent type definitions  
└── client/  
 └── src/  
 └── components/  
 └── agents/  
 ├── ThreeDDesignerPanel.tsx # Client-side interface  
 └── AgentDashboard.tsx # Agent integration in UI

#### Architectural Layers

1. Agent Layer (threeDDesignerAgent.ts)
2. Implements the agent's core capabilities
3. Processes user inputs (drawings and text descriptions)
4. Coordinates multiple services for comprehensive results
5. Generates detailed explanations of designs and choices
6. Service Layer
7. threeDService.ts: Handles 3D visualization and rendering
8. furniturePlacementService.ts: Manages physics-based furniture placement
9. materialService.ts: Accesses material database
10. vectorService.ts: Provides vector-based similarity searches
11. Visualization Layer
12. Integrates with multiple 3D generation models
13. Processes inputs through specialized neural networks
14. Generates renderable 3D models with proper materials
15. Validates physical constraints and optimization
16. UI Layer (ThreeDDesignerPanel.tsx)
17. Provides input interfaces for drawings and text descriptions
18. Displays 3D visualization results
19. Enables interactive feedback and refinement
20. Offers material browsing and selection

threeDDesignerAgent.ts

Generates detailed explanations of designs and choices

Service Layer

threeDService.ts

furniturePlacementService.ts

materialService.ts

vectorService.ts: Provides vector-based similarity searches

vectorService.ts

Visualization Layer

Validates physical constraints and optimization

UI Layer (ThreeDDesignerPanel.tsx)

ThreeDDesignerPanel.tsx

### Implementation Details

#### Agent Implementation

The 3D Designer Agent implements several specialized methods for 3D design tasks:

export class ThreeDDesignerAgent extends Agent {  
 // Service integrations  
 private threeDService: ThreeDService;  
 private materialService: MaterialService;  
 private vectorService: VectorService;  
 private furniturePlacementService: FurniturePlacementService;  
  
 // Core functionality methods  
 public async process2DDrawing(task: Task): Promise<any>;  
 public async generateRoomFromText(task: Task): Promise<any>;  
 public async refineResult(task: Task): Promise<any>;  
  
 // Helper methods  
 private async searchRelevantMaterials(result: any): Promise<MaterialDetails[]>;  
 private extractMaterialRequirements(result: any): { query: string; filters: Record<string, any> };  
 private async generateArchitecturalResponse(layout: RoomLayout, materials: MaterialDetails[]): Promise<string>;  
 private async generateRoomLayoutResponse(rooms: RoomLayout[], materials: MaterialDetails[]): Promise<string>;  
 private async generateRefinementResponse(result: any, feedback: string): Promise<string>;  
}

export class ThreeDDesignerAgent extends Agent {  
 // Service integrations  
 private threeDService: ThreeDService;  
 private materialService: MaterialService;  
 private vectorService: VectorService;  
 private furniturePlacementService: FurniturePlacementService;  
  
 // Core functionality methods  
 public async process2DDrawing(task: Task): Promise<any>;  
 public async generateRoomFromText(task: Task): Promise<any>;  
 public async refineResult(task: Task): Promise<any>;  
  
 // Helper methods  
 private async searchRelevantMaterials(result: any): Promise<MaterialDetails[]>;  
 private extractMaterialRequirements(result: any): { query: string; filters: Record<string, any> };  
 private async generateArchitecturalResponse(layout: RoomLayout, materials: MaterialDetails[]): Promise<string>;  
 private async generateRoomLayoutResponse(rooms: RoomLayout[], materials: MaterialDetails[]): Promise<string>;  
 private async generateRefinementResponse(result: any, feedback: string): Promise<string>;  
}

#### Configuration Options

The 3D Designer Agent is configured with specific endpoints and resources:

interface ThreeDDesignerConfig {  
 knowledgeBaseUrl: string;  
 modelEndpoints: {  
 nerfStudio: string;  
 instantNgp: string;  
 shapE: string;  
 get3d: string;  
 hunyuan3d: string;  
 blenderProc: string;  
 architecturalRecognition: string;  
 roomLayoutGenerator: string;  
 };  
 threeDFrontPath: string; // Path to 3D-FRONT dataset for furniture models  
}

interface ThreeDDesignerConfig {  
 knowledgeBaseUrl: string;  
 modelEndpoints: {  
 nerfStudio: string;  
 instantNgp: string;  
 shapE: string;  
 get3d: string;  
 hunyuan3d: string;  
 blenderProc: string;  
 architecturalRecognition: string;  
 roomLayoutGenerator: string;  
 };  
 threeDFrontPath: string; // Path to 3D-FRONT dataset for furniture models  
}

#### Response Generation

The agent uses Claude's multimodal capabilities to generate detailed explanations:

private async invokeLLM(prompt: string, options?: {  
 system?: string;  
 messages?: Array<{  
 role: string;  
 content: Array<{  
 type: string;  
 text?: string;  
 source?: {  
 type: string;  
 media\_type: string;  
 data: string;  
 };  
 }>;  
 }>;  
}): Promise<string>

private async invokeLLM(prompt: string, options?: {  
 system?: string;  
 messages?: Array<{  
 role: string;  
 content: Array<{  
 type: string;  
 text?: string;  
 source?: {  
 type: string;  
 media\_type: string;  
 data: string;  
 };  
 }>;  
 }>;  
}): Promise<string>

#### Client-Side Integration

The 3D Designer Agent is integrated into the client interface through a specialized panel with:

1. Input Options - Drawing upload and text description interface
2. 3D Visualization - Interactive 3D viewer for results
3. Material Selection - Interface for browsing and applying materials
4. Feedback Mechanisms - Tools for refining and adjusting designs

### Setup Instructions

#### Prerequisites

* Functioning KAI platform with 3D visualization capabilities
* CrewAI integration set up according to CrewAI installation guide
* Materials database with vector search capabilities
* Access to 3D furniture models (e.g., 3D-FRONT dataset)
* Anthropic API key for Claude model access

#### Installation

The 3D Designer Agent is included in the standard crewAI integration package:

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

# Navigate to the agents directory  
cd packages/agents  
  
# Install dependencies if not already done  
yarn install

#### Configuration

Configure the agent in your application initialization:

import { ThreeDDesignerAgent } from '@kai/agents/frontend/threeDDesignerAgent';  
  
// Create a 3D Designer Agent instance  
const threeDDesignerAgent = new ThreeDDesignerAgent({  
 knowledgeBaseUrl: process.env.KNOWLEDGE\_BASE\_URL,  
 modelEndpoints: {  
 nerfStudio: process.env.NERF\_STUDIO\_ENDPOINT,  
 instantNgp: process.env.INSTANT\_NGP\_ENDPOINT,  
 shapE: process.env.SHAPE\_ENDPOINT,  
 get3d: process.env.GET3D\_ENDPOINT,  
 hunyuan3d: process.env.HUNYUAN3D\_ENDPOINT,  
 blenderProc: process.env.BLENDER\_PROC\_ENDPOINT,  
 architecturalRecognition: process.env.ARCHITECTURAL\_RECOGNITION\_ENDPOINT,  
 roomLayoutGenerator: process.env.ROOM\_LAYOUT\_GENERATOR\_ENDPOINT  
 },  
 threeDFrontPath: process.env.THREE\_D\_FRONT\_PATH  
});

import { ThreeDDesignerAgent } from '@kai/agents/frontend/threeDDesignerAgent';  
  
// Create a 3D Designer Agent instance  
const threeDDesignerAgent = new ThreeDDesignerAgent({  
 knowledgeBaseUrl: process.env.KNOWLEDGE\_BASE\_URL,  
 modelEndpoints: {  
 nerfStudio: process.env.NERF\_STUDIO\_ENDPOINT,  
 instantNgp: process.env.INSTANT\_NGP\_ENDPOINT,  
 shapE: process.env.SHAPE\_ENDPOINT,  
 get3d: process.env.GET3D\_ENDPOINT,  
 hunyuan3d: process.env.HUNYUAN3D\_ENDPOINT,  
 blenderProc: process.env.BLENDER\_PROC\_ENDPOINT,  
 architecturalRecognition: process.env.ARCHITECTURAL\_RECOGNITION\_ENDPOINT,  
 roomLayoutGenerator: process.env.ROOM\_LAYOUT\_GENERATOR\_ENDPOINT  
 },  
 threeDFrontPath: process.env.THREE\_D\_FRONT\_PATH  
});

### Usage Examples

#### Client-Side Integration

import React from 'react';  
import { ThreeDDesignerPanel } from '../components/agents/ThreeDDesignerPanel';  
  
const ThreeDDesignerPage: React.FC = () => {  
 return (  
 <div className="threeD-designer-page">  
 <h1>3D Designer</h1>  
 <ThreeDDesignerPanel />  
 </div>  
 );  
};  
  
export default ThreeDDesignerPage;

import React from 'react';  
import { ThreeDDesignerPanel } from '../components/agents/ThreeDDesignerPanel';  
  
const ThreeDDesignerPage: React.FC = () => {  
 return (  
 <div className="threeD-designer-page">  
 <h1>3D Designer</h1>  
 <ThreeDDesignerPanel />  
 </div>  
 );  
};  
  
export default ThreeDDesignerPage;

#### Processing a 2D Architectural Drawing

import { ThreeDDesignerAgent } from '@kai/agents/frontend/threeDDesignerAgent';  
import { Task } from 'crewai';  
  
// Create the 3D Designer Agent  
const threeDDesignerAgent = new ThreeDDesignerAgent({  
 // Configuration options  
 knowledgeBaseUrl: 'https://api.example.com/knowledge-base',  
 modelEndpoints: {  
 // Model endpoints  
 },  
 threeDFrontPath: '/path/to/3d-front'  
});  
  
// Process a 2D architectural drawing  
const task = new Task({  
 description: JSON.stringify({  
 drawing: 'base64-encoded-image-of-architectural-drawing'  
 })  
});  
  
const result = await threeDDesignerAgent.process2DDrawing(task);  
console.log(result.layout);  
console.log(result.materials);  
console.log(result.explanation);

import { ThreeDDesignerAgent } from '@kai/agents/frontend/threeDDesignerAgent';  
import { Task } from 'crewai';  
  
// Create the 3D Designer Agent  
const threeDDesignerAgent = new ThreeDDesignerAgent({  
 // Configuration options  
 knowledgeBaseUrl: 'https://api.example.com/knowledge-base',  
 modelEndpoints: {  
 // Model endpoints  
 },  
 threeDFrontPath: '/path/to/3d-front'  
});  
  
// Process a 2D architectural drawing  
const task = new Task({  
 description: JSON.stringify({  
 drawing: 'base64-encoded-image-of-architectural-drawing'  
 })  
});  
  
const result = await threeDDesignerAgent.process2DDrawing(task);  
console.log(result.layout);  
console.log(result.materials);  
console.log(result.explanation);

#### Generating a Room from Text Description

import { ThreeDDesignerAgent } from '@kai/agents/frontend/threeDDesignerAgent';  
import { Task } from 'crewai';  
  
// Create the 3D Designer Agent  
const threeDDesignerAgent = new ThreeDDesignerAgent({  
 // Configuration options  
});  
  
// Generate rooms from text description  
const task = new Task({  
 description: JSON.stringify({  
 rooms: [  
 {  
 type: 'living',  
 dimensions: { width: 5, length: 7, height: 2.8 },  
 features: ['large windows', 'fireplace', 'open plan']  
 },  
 {  
 type: 'kitchen',  
 dimensions: { width: 4, length: 5, height: 2.8 },  
 features: ['island counter', 'modern appliances']  
 }  
 ],  
 style: 'modern minimalist'  
 })  
});  
  
const result = await threeDDesignerAgent.generateRoomFromText(task);  
console.log(result.rooms);  
console.log(result.materials);  
console.log(result.explanation);

import { ThreeDDesignerAgent } from '@kai/agents/frontend/threeDDesignerAgent';  
import { Task } from 'crewai';  
  
// Create the 3D Designer Agent  
const threeDDesignerAgent = new ThreeDDesignerAgent({  
 // Configuration options  
});  
  
// Generate rooms from text description  
const task = new Task({  
 description: JSON.stringify({  
 rooms: [  
 {  
 type: 'living',  
 dimensions: { width: 5, length: 7, height: 2.8 },  
 features: ['large windows', 'fireplace', 'open plan']  
 },  
 {  
 type: 'kitchen',  
 dimensions: { width: 4, length: 5, height: 2.8 },  
 features: ['island counter', 'modern appliances']  
 }  
 ],  
 style: 'modern minimalist'  
 })  
});  
  
const result = await threeDDesignerAgent.generateRoomFromText(task);  
console.log(result.rooms);  
console.log(result.materials);  
console.log(result.explanation);

#### Refining a Design Based on Feedback

import { ThreeDDesignerAgent } from '@kai/agents/frontend/threeDDesignerAgent';  
import { Task } from 'crewai';  
  
// Create the 3D Designer Agent  
const threeDDesignerAgent = new ThreeDDesignerAgent({  
 // Configuration options  
});  
  
// Refine a design based on feedback  
const task = new Task({  
 description: JSON.stringify({  
 result: previousDesignResult,  
 feedback: "The living room feels cramped. Please make it more spacious and move the sofa away from the window. Also, I'd prefer warmer wood tones for the flooring.",  
 options: {  
 preserveLayout: true,  
 adjustFurniture: true,  
 changeMaterials: true  
 }  
 })  
});  
  
const refinedResult = await threeDDesignerAgent.refineResult(task);  
console.log(refinedResult.result);  
console.log(refinedResult.explanation);

import { ThreeDDesignerAgent } from '@kai/agents/frontend/threeDDesignerAgent';  
import { Task } from 'crewai';  
  
// Create the 3D Designer Agent  
const threeDDesignerAgent = new ThreeDDesignerAgent({  
 // Configuration options  
});  
  
// Refine a design based on feedback  
const task = new Task({  
 description: JSON.stringify({  
 result: previousDesignResult,  
 feedback: "The living room feels cramped. Please make it more spacious and move the sofa away from the window. Also, I'd prefer warmer wood tones for the flooring.",  
 options: {  
 preserveLayout: true,  
 adjustFurniture: true,  
 changeMaterials: true  
 }  
 })  
});  
  
const refinedResult = await threeDDesignerAgent.refineResult(task);  
console.log(refinedResult.result);  
console.log(refinedResult.explanation);

### Advanced Configuration

#### Custom Material Integration

Enhance the 3D Designer Agent with specialized material integration:

import { ThreeDDesignerAgent } from '@kai/agents/frontend/threeDDesignerAgent';  
import { CustomMaterialService } from './customMaterialService';  
  
// Create a custom material service  
const customMaterialService = new CustomMaterialService({  
 baseURL: 'https://api.example.com/custom-materials',  
 apiKey: process.env.CUSTOM\_MATERIALS\_API\_KEY  
});  
  
// Override the material service in the agent  
const threeDDesignerAgent = new ThreeDDesignerAgent({  
 // Standard configuration  
});  
  
// Replace the material service with custom implementation  
threeDDesignerAgent.setMaterialService(customMaterialService);

import { ThreeDDesignerAgent } from '@kai/agents/frontend/threeDDesignerAgent';  
import { CustomMaterialService } from './customMaterialService';  
  
// Create a custom material service  
const customMaterialService = new CustomMaterialService({  
 baseURL: 'https://api.example.com/custom-materials',  
 apiKey: process.env.CUSTOM\_MATERIALS\_API\_KEY  
});  
  
// Override the material service in the agent  
const threeDDesignerAgent = new ThreeDDesignerAgent({  
 // Standard configuration  
});  
  
// Replace the material service with custom implementation  
threeDDesignerAgent.setMaterialService(customMaterialService);

#### Physics-Based Optimization Settings

Customize the physics constraints for furniture placement:

import { ThreeDDesignerAgent } from '@kai/agents/frontend/threeDDesignerAgent';  
  
// Create the 3D Designer Agent  
const threeDDesignerAgent = new ThreeDDesignerAgent({  
 // Standard configuration  
});  
  
// Generate a room with custom physics constraints  
const task = new Task({  
 description: JSON.stringify({  
 rooms: [{  
 type: 'living',  
 dimensions: { width: 5, length: 7, height: 2.8 }  
 }],  
 style: 'modern',  
 physicsConstraints: {  
 gravitationalForce: 9.8,  
 frictionCoefficient: 0.5,  
 minClearance: {  
 walkways: 0.8, // 80cm for walkways  
 seating: 0.45, // 45cm between seating  
 walls: 0.05 // 5cm from walls  
 },  
 optimizationGoals: {  
 conversation: 0.8, // Weight for conversation-friendly layouts  
 viewingAngles: 0.6, // Weight for optimal TV/window viewing  
 traffic: 0.7 // Weight for smooth traffic flow  
 }  
 }  
 })  
});  
  
const result = await threeDDesignerAgent.generateRoomFromText(task);

import { ThreeDDesignerAgent } from '@kai/agents/frontend/threeDDesignerAgent';  
  
// Create the 3D Designer Agent  
const threeDDesignerAgent = new ThreeDDesignerAgent({  
 // Standard configuration  
});  
  
// Generate a room with custom physics constraints  
const task = new Task({  
 description: JSON.stringify({  
 rooms: [{  
 type: 'living',  
 dimensions: { width: 5, length: 7, height: 2.8 }  
 }],  
 style: 'modern',  
 physicsConstraints: {  
 gravitationalForce: 9.8,  
 frictionCoefficient: 0.5,  
 minClearance: {  
 walkways: 0.8, // 80cm for walkways  
 seating: 0.45, // 45cm between seating  
 walls: 0.05 // 5cm from walls  
 },  
 optimizationGoals: {  
 conversation: 0.8, // Weight for conversation-friendly layouts  
 viewingAngles: 0.6, // Weight for optimal TV/window viewing  
 traffic: 0.7 // Weight for smooth traffic flow  
 }  
 }  
 })  
});  
  
const result = await threeDDesignerAgent.generateRoomFromText(task);

### Performance Considerations

#### 3D Model Optimization

1. Level-of-Detail Management
2. Implement progressive LOD based on camera distance
3. Optimize polygon counts for interactive viewing
4. Use instancing for repeated elements (furniture, fixtures)
5. Texture Optimization
6. Apply texture compression appropriate for web viewing
7. Implement material atlasing for reduced draw calls
8. Use mipmap generation for distant textures
9. Scene Graph Optimization
10. Implement frustum culling for large scenes
11. Use spatial partitioning for complex environments
12. Batch similar materials to reduce state changes

Use instancing for repeated elements (furniture, fixtures)

Texture Optimization

Use mipmap generation for distant textures

Scene Graph Optimization

### Security Considerations

1. Input Validation
2. Sanitize all image and text inputs
3. Validate architectural specifications against safety standards
4. Implement size and complexity limits for 3D generation
5. Resource Management
6. Implement timeouts for long-running 3D operations
7. Set memory limits for complex scene generation
8. Use job queuing for resource-intensive operations
9. Access Control
10. Restrict access to 3D generation capabilities as needed
11. Implement usage quotas for resource-intensive operations
12. Track and audit usage patterns for security monitoring

Implement size and complexity limits for 3D generation

Resource Management

Use job queuing for resource-intensive operations

Access Control

### Related Documentation

* 3D Reconstruction Pipeline - Details on the underlying 3D technologies
* 3D Visualization - Visualization system architecture
* Material Recognition - Integration with material recognition
* CrewAI Integration - Overall agent system architecture
* Text-to-3D Generation - Text-to-3D model details

# Tracing Service

Source: readme/tracing-service.md

---

## Distributed Tracing Service

This document describes the distributed tracing service implementation in the KAI platform. The tracing service provides a unified interface for tracking requests across services, enabling end-to-end visibility into request flows.

### Overview

The distributed tracing service is designed to track requests as they flow through different services and components of the application. It provides a consistent API for tracing operations, regardless of the underlying tracing provider. The service supports OpenTelemetry for distributed tracing and integrates with the telemetry service.

### Architecture

The tracing service follows a provider pattern, allowing different tracing implementations to be used interchangeably. The service consists of the following components:

1. Tracing Service: The main service that provides a unified interface for tracing operations.
2. Tracing Provider Interface: An interface that defines the contract for tracing providers.
3. OpenTelemetry Provider: A provider that implements distributed tracing using OpenTelemetry.
4. Tracing Initializer: A utility for initializing the tracing service with different providers.

### Usage

#### Basic Usage

import { tracing, SpanKind, SpanStatusCode } from '@kai/shared';  
  
// Start a span  
const spanResult = tracing.startSpan('operation\_name', {  
 kind: SpanKind.SERVER,  
 attributes: {  
 'service.name': 'my-service',  
 'operation.name': 'process-request'  
 }  
});  
  
// Get the current span  
const span = tracing.getCurrentSpan();  
  
// Add attributes to the span  
tracing.addSpanAttributes(span, {  
 'user.id': '123',  
 'request.id': 'abc-123'  
});  
  
// Add events to the span  
tracing.addSpanEvent(span, 'processing\_started', {  
 'timestamp': Date.now()  
});  
  
// Set span status  
tracing.setSpanStatus(span, SpanStatusCode.OK);  
  
// End the span  
tracing.endSpan(span);

import { tracing, SpanKind, SpanStatusCode } from '@kai/shared';  
  
// Start a span  
const spanResult = tracing.startSpan('operation\_name', {  
 kind: SpanKind.SERVER,  
 attributes: {  
 'service.name': 'my-service',  
 'operation.name': 'process-request'  
 }  
});  
  
// Get the current span  
const span = tracing.getCurrentSpan();  
  
// Add attributes to the span  
tracing.addSpanAttributes(span, {  
 'user.id': '123',  
 'request.id': 'abc-123'  
});  
  
// Add events to the span  
tracing.addSpanEvent(span, 'processing\_started', {  
 'timestamp': Date.now()  
});  
  
// Set span status  
tracing.setSpanStatus(span, SpanStatusCode.OK);  
  
// End the span  
tracing.endSpan(span);

#### Automatic Span Management

import { tracing, SpanKind } from '@kai/shared';  
  
// Automatically manage span lifecycle  
const result = await tracing.withSpan(  
 'database\_query',  
 async () => {  
 // This code is executed within the span  
 return await db.query('SELECT \* FROM users');  
 },  
 {  
 kind: SpanKind.CLIENT,  
 attributes: {  
 'db.system': 'postgresql',  
 'db.statement': 'SELECT \* FROM users'  
 }  
 }  
);

import { tracing, SpanKind } from '@kai/shared';  
  
// Automatically manage span lifecycle  
const result = await tracing.withSpan(  
 'database\_query',  
 async () => {  
 // This code is executed within the span  
 return await db.query('SELECT \* FROM users');  
 },  
 {  
 kind: SpanKind.CLIENT,  
 attributes: {  
 'db.system': 'postgresql',  
 'db.statement': 'SELECT \* FROM users'  
 }  
 }  
);

#### Trace Context Propagation

import { tracing } from '@kai/shared';  
  
// Extract trace context from incoming request  
const headers = request.headers;  
const traceContext = tracing.extractContext(headers);  
  
// Create a child span  
const spanResult = tracing.startSpan('child\_operation', {  
 parent: traceContext  
});  
  
// Inject trace context into outgoing request  
const outgoingHeaders = {};  
tracing.injectContext(spanResult.traceContext, outgoingHeaders);  
  
// Make request with trace context  
const response = await fetch('https://api.example.com', {  
 headers: outgoingHeaders  
});

import { tracing } from '@kai/shared';  
  
// Extract trace context from incoming request  
const headers = request.headers;  
const traceContext = tracing.extractContext(headers);  
  
// Create a child span  
const spanResult = tracing.startSpan('child\_operation', {  
 parent: traceContext  
});  
  
// Inject trace context into outgoing request  
const outgoingHeaders = {};  
tracing.injectContext(spanResult.traceContext, outgoingHeaders);  
  
// Make request with trace context  
const response = await fetch('https://api.example.com', {  
 headers: outgoingHeaders  
});

### Configuration

The tracing service can be configured through environment variables or the unified configuration system. The following configuration options are available:

// In .env file  
TRACING\_ENABLED=true  
TRACING\_TYPE=opentelemetry

// In .env file  
TRACING\_ENABLED=true  
TRACING\_TYPE=opentelemetry

### Tracing Providers

#### OpenTelemetry Provider

The OpenTelemetry provider implements distributed tracing using OpenTelemetry. It's suitable for production and distributed applications. Features include:

* OpenTelemetry-based tracing
* W3C Trace Context propagation
* Span attributes and events
* Span status and error handling
* Trace context extraction and injection

### Integration with API Client

The API client has been integrated with the tracing service to automatically trace HTTP requests. Features include:

* Automatic span creation for HTTP requests
* Trace context propagation in request headers
* Span attributes for request and response details
* Span events for retries and errors
* Span status based on response status

import { apiClient } from '@kai/shared';  
  
// Tracing is enabled by default  
const result = await apiClient.get('/api/users');  
  
// Disable tracing for specific client  
const customClient = createApiClient({  
 useTracing: false  
});  
  
// Add custom tracing attributes  
const customClient = createApiClient({  
 tracingAttributes: {  
 'service.name': 'custom-service',  
 'client.version': '1.0.0'  
 }  
});

import { apiClient } from '@kai/shared';  
  
// Tracing is enabled by default  
const result = await apiClient.get('/api/users');  
  
// Disable tracing for specific client  
const customClient = createApiClient({  
 useTracing: false  
});  
  
// Add custom tracing attributes  
const customClient = createApiClient({  
 tracingAttributes: {  
 'service.name': 'custom-service',  
 'client.version': '1.0.0'  
 }  
});

### Implementation Details

#### Tracing Service

The tracing service provides a unified interface for tracing operations. It delegates all operations to the configured provider and adds additional functionality like automatic span management and integration with the telemetry service.

class TracingService {  
 private provider: TracingProvider | null = null;  
 private enabled: boolean = false;  
  
 // Set the tracing provider  
 setProvider(provider: TracingProvider): void;  
  
 // Enable tracing  
 async enable(): Promise<void>;  
  
 // Disable tracing  
 async disable(): Promise<void>;  
  
 // Start a span  
 startSpan(name: string, options?: SpanOptions): SpanResult | undefined;  
  
 // End a span  
 endSpan(span: Span, endTime?: number): void;  
  
 // Set span status  
 setSpanStatus(span: Span, code: SpanStatusCode, message?: string): void;  
  
 // Add span attributes  
 addSpanAttributes(span: Span, attributes: Record<string, string | number | boolean | string[] | number[] | boolean[]>): void;  
  
 // Add span events  
 addSpanEvent(span: Span, name: string, attributes?: Record<string, string | number | boolean>, timestamp?: number): void;  
  
 // Get current span  
 getCurrentSpan(): Span | undefined;  
  
 // Run with span  
 async withSpan<T>(name: string, fn: () => Promise<T> | T, options?: SpanOptions): Promise<T>;  
  
 // Extract trace context from carrier  
 extractContext(carrier: Record<string, string>): TraceContext | undefined;  
  
 // Inject trace context into carrier  
 injectContext(traceContext: TraceContext, carrier: Record<string, string>): void;  
}

class TracingService {  
 private provider: TracingProvider | null = null;  
 private enabled: boolean = false;  
  
 // Set the tracing provider  
 setProvider(provider: TracingProvider): void;  
  
 // Enable tracing  
 async enable(): Promise<void>;  
  
 // Disable tracing  
 async disable(): Promise<void>;  
  
 // Start a span  
 startSpan(name: string, options?: SpanOptions): SpanResult | undefined;  
  
 // End a span  
 endSpan(span: Span, endTime?: number): void;  
  
 // Set span status  
 setSpanStatus(span: Span, code: SpanStatusCode, message?: string): void;  
  
 // Add span attributes  
 addSpanAttributes(span: Span, attributes: Record<string, string | number | boolean | string[] | number[] | boolean[]>): void;  
  
 // Add span events  
 addSpanEvent(span: Span, name: string, attributes?: Record<string, string | number | boolean>, timestamp?: number): void;  
  
 // Get current span  
 getCurrentSpan(): Span | undefined;  
  
 // Run with span  
 async withSpan<T>(name: string, fn: () => Promise<T> | T, options?: SpanOptions): Promise<T>;  
  
 // Extract trace context from carrier  
 extractContext(carrier: Record<string, string>): TraceContext | undefined;  
  
 // Inject trace context into carrier  
 injectContext(traceContext: TraceContext, carrier: Record<string, string>): void;  
}

#### Tracing Provider Interface

The tracing provider interface defines the contract for tracing providers. All providers must implement this interface.

interface TracingProvider {  
 // Initialize the provider  
 initialize(): Promise<void>;  
  
 // Start a span  
 startSpan(name: string, options?: SpanOptions): Span;  
  
 // End a span  
 endSpan(span: Span, endTime?: number): void;  
  
 // Set span status  
 setSpanStatus(span: Span, code: SpanStatusCode, message?: string): void;  
  
 // Add span attributes  
 addSpanAttributes(span: Span, attributes: Record<string, string | number | boolean | string[] | number[] | boolean[]>): void;  
  
 // Add span events  
 addSpanEvent(span: Span, name: string, attributes?: Record<string, string | number | boolean>, timestamp?: number): void;  
  
 // Get current span  
 getCurrentSpan(): Span | undefined;  
  
 // Run with span  
 withSpan<T>(span: Span, fn: () => Promise<T> | T): Promise<T>;  
  
 // Extract trace context from carrier  
 extractContext(carrier: Record<string, string>): TraceContext | undefined;  
  
 // Inject trace context into carrier  
 injectContext(traceContext: TraceContext, carrier: Record<string, string>): void;  
}

interface TracingProvider {  
 // Initialize the provider  
 initialize(): Promise<void>;  
  
 // Start a span  
 startSpan(name: string, options?: SpanOptions): Span;  
  
 // End a span  
 endSpan(span: Span, endTime?: number): void;  
  
 // Set span status  
 setSpanStatus(span: Span, code: SpanStatusCode, message?: string): void;  
  
 // Add span attributes  
 addSpanAttributes(span: Span, attributes: Record<string, string | number | boolean | string[] | number[] | boolean[]>): void;  
  
 // Add span events  
 addSpanEvent(span: Span, name: string, attributes?: Record<string, string | number | boolean>, timestamp?: number): void;  
  
 // Get current span  
 getCurrentSpan(): Span | undefined;  
  
 // Run with span  
 withSpan<T>(span: Span, fn: () => Promise<T> | T): Promise<T>;  
  
 // Extract trace context from carrier  
 extractContext(carrier: Record<string, string>): TraceContext | undefined;  
  
 // Inject trace context into carrier  
 injectContext(traceContext: TraceContext, carrier: Record<string, string>): void;  
}

### Benefits

The distributed tracing service provides several benefits:

1. End-to-End Visibility: Track requests as they flow through different services and components.
2. Performance Monitoring: Identify bottlenecks and performance issues in the application.
3. Error Diagnosis: Quickly identify the root cause of errors and failures.
4. Dependency Analysis: Understand the dependencies between different services and components.
5. Service Level Objectives: Monitor and enforce service level objectives based on trace data.
6. Consistent API: The unified interface provides a consistent API for tracing operations, regardless of the underlying provider.
7. Provider Flexibility: The provider pattern allows different tracing implementations to be used interchangeably.

### Next Steps

The following steps are recommended to further improve the distributed tracing service:

1. Add More Providers: Add support for more tracing providers (Jaeger, Zipkin, etc.).
2. Add Sampling Strategies: Add support for different sampling strategies to reduce the volume of trace data.
3. Add Trace Analytics: Add support for analyzing trace data to identify patterns and trends.
4. Add Trace Visualization: Add support for visualizing trace data to understand request flows.
5. Add Trace Alerting: Add support for alerting based on trace data to proactively identify issues.
6. Add Trace Retention: Add support for configuring trace data retention to manage storage costs.

# Training Monitoring System

Source: readme/training-monitoring-system.md

---

## Training Monitoring System

The Kai platform includes a comprehensive Training Monitoring System that provides real-time insights into machine learning model training processes. This system is accessible through the admin panel and offers tools for visualizing metrics, managing checkpoints, and tuning model parameters.

### Overview

The Training Monitoring System is designed to help administrators and ML engineers:

* Track training progress in real-time
* Visualize performance metrics through customizable charts
* Create and manage model checkpoints during training
* Compare checkpoint performance and model versions
* Fine-tune hyperparameters to optimize model performance
* Rollback to previous model versions when needed

### Architecture

The system consists of a parent TrainingMonitor component that integrates three specialized components:

1. MetricsVisualizer: Displays real-time training metrics with customizable charts
2. CheckpointManager: Manages model checkpoint operations (creation, comparison, rollback)
3. ParameterTuner: Allows adjustment of hyperparameters during training

### Components

#### TrainingMonitor

The TrainingMonitor serves as the container component that integrates all training monitoring functionality into a cohesive interface. It provides:

* Tab-based navigation between specialized components
* Unified job ID management
* Shared state for training parameters
* System-wide notifications and alerts

##### Implementation

The TrainingMonitor is implemented as a React component that dynamically loads its child components and manages state across them:

// Simplified implementation  
const TrainingMonitor: React.FC<TrainingMonitorProps> = ({   
 jobId,   
 modelType,  
 onComplete   
}) => {  
 const [activeTab, setActiveTab] = useState('metrics');  
  
 return (  
 <Box>  
 <Tabs value={activeTab} onChange={(\_, value) => setActiveTab(value)}>  
 <Tab value="metrics" label="Training Metrics" />  
 <Tab value="checkpoints" label="Checkpoints" />  
 <Tab value="parameters" label="Parameters" />  
 </Tabs>  
  
 {activeTab === 'metrics' && <MetricsVisualizer jobId={jobId} modelType={modelType} />}  
 {activeTab === 'checkpoints' && <CheckpointManager jobId={jobId} modelType={modelType} />}  
 {activeTab === 'parameters' && <ParameterTuner jobId={jobId} modelType={modelType} />}  
 </Box>  
 );  
};

// Simplified implementation  
const TrainingMonitor: React.FC<TrainingMonitorProps> = ({   
 jobId,   
 modelType,  
 onComplete   
}) => {  
 const [activeTab, setActiveTab] = useState('metrics');  
  
 return (  
 <Box>  
 <Tabs value={activeTab} onChange={(\_, value) => setActiveTab(value)}>  
 <Tab value="metrics" label="Training Metrics" />  
 <Tab value="checkpoints" label="Checkpoints" />  
 <Tab value="parameters" label="Parameters" />  
 </Tabs>  
  
 {activeTab === 'metrics' && <MetricsVisualizer jobId={jobId} modelType={modelType} />}  
 {activeTab === 'checkpoints' && <CheckpointManager jobId={jobId} modelType={modelType} />}  
 {activeTab === 'parameters' && <ParameterTuner jobId={jobId} modelType={modelType} />}  
 </Box>  
 );  
};

#### MetricsVisualizer

The MetricsVisualizer component provides real-time visualization of training metrics. It offers:

* Interactive line charts for tracking metrics over time
* Customizable chart views and metric selection
* Support for comparing multiple training runs
* Data export and sharing capabilities
* Chart customization options (timeframes, scaling, etc.)

##### Key Features

* Real-time Updates: Displays training metrics as they're generated
* Multi-metric Visualization: Can show multiple metrics simultaneously (loss, accuracy, etc.)
* Custom Visualization Controls: Timeframe selection, smoothing, and scaling options
* Adaptive Charts: Automatically adjusts to available metrics for different model types
* Performance Comparison: Overlay metrics from previous training runs

##### Implementation

// Simplified implementation  
const MetricsVisualizer: React.FC<MetricsVisualizerProps> = ({   
 jobId,   
 modelType   
}) => {  
 const [metrics, setMetrics] = useState<TrainingMetrics[]>([]);  
 const [selectedMetrics, setSelectedMetrics] = useState<string[]>(['loss', 'accuracy']);  
 const [timeframe, setTimeframe] = useState<Timeframe>('full');  
  
 // Fetch metrics on interval  
 useEffect(() => {  
 // Implementation details  
 }, [jobId]);  
  
 return (  
 <Box>  
 <MetricControls   
 availableMetrics={getAvailableMetrics(modelType)}  
 selectedMetrics={selectedMetrics}  
 onMetricsChange={setSelectedMetrics}  
 timeframe={timeframe}  
 onTimeframeChange={setTimeframe}  
 />  
  
 <MetricsChart   
 data={metrics}  
 selectedMetrics={selectedMetrics}  
 timeframe={timeframe}  
 />  
  
 <MetricsTable   
 data={metrics}  
 selectedMetrics={selectedMetrics}  
 />  
 </Box>  
 );  
};

// Simplified implementation  
const MetricsVisualizer: React.FC<MetricsVisualizerProps> = ({   
 jobId,   
 modelType   
}) => {  
 const [metrics, setMetrics] = useState<TrainingMetrics[]>([]);  
 const [selectedMetrics, setSelectedMetrics] = useState<string[]>(['loss', 'accuracy']);  
 const [timeframe, setTimeframe] = useState<Timeframe>('full');  
  
 // Fetch metrics on interval  
 useEffect(() => {  
 // Implementation details  
 }, [jobId]);  
  
 return (  
 <Box>  
 <MetricControls   
 availableMetrics={getAvailableMetrics(modelType)}  
 selectedMetrics={selectedMetrics}  
 onMetricsChange={setSelectedMetrics}  
 timeframe={timeframe}  
 onTimeframeChange={setTimeframe}  
 />  
  
 <MetricsChart   
 data={metrics}  
 selectedMetrics={selectedMetrics}  
 timeframe={timeframe}  
 />  
  
 <MetricsTable   
 data={metrics}  
 selectedMetrics={selectedMetrics}  
 />  
 </Box>  
 );  
};

#### CheckpointManager

The CheckpointManager component provides a comprehensive interface for managing model checkpoints during and after training. It enables:

* Viewing all available checkpoints with metadata
* Creating new checkpoints during training
* Comparing metrics between checkpoints
* Rolling back to previous checkpoints
* Managing checkpoint lifecycle

##### Key Features

* Checkpoint Creation: Create manual checkpoints during training with custom descriptions and tags
* Checkpoint Comparison: Side-by-side comparison of metrics and parameters between any two checkpoints
* Visual Differencing: Highlight parameter differences between checkpoints
* Rollback Capability: Roll back to any previous checkpoint
* Tagging System: Organize checkpoints with customizable tags

##### Implementation

The CheckpointManager integrates with the backend API to manage checkpoint operations:

// Simplified implementation  
const CheckpointManager: React.FC<CheckpointManagerProps> = ({  
 jobId,  
 modelType  
}) => {  
 const [checkpoints, setCheckpoints] = useState<Checkpoint[]>([]);  
 const [selectedCheckpoints, setSelectedCheckpoints] = useState<string[]>([]);  
 const [loading, setLoading] = useState<boolean>(true);  
  
 // Load checkpoints using API  
 const loadCheckpoints = async () => {  
 try {  
 setLoading(true);  
 const result = await checkpointApi.fetchCheckpoints(jobId);  
 setCheckpoints(result);  
 } catch (err) {  
 // Error handling  
 } finally {  
 setLoading(false);  
 }  
 };  
  
 // Fetch checkpoints when component mounts or jobId changes  
 useEffect(() => {  
 loadCheckpoints();  
 }, [jobId]);  
  
 return (  
 <Box>  
 {/\* Create checkpoint button \*/}  
 {/\* Checkpoints table \*/}  
 {/\* Checkpoint comparison section \*/}  
 {/\* Dialogs for checkpoint operations \*/}  
 </Box>  
 );  
};

// Simplified implementation  
const CheckpointManager: React.FC<CheckpointManagerProps> = ({  
 jobId,  
 modelType  
}) => {  
 const [checkpoints, setCheckpoints] = useState<Checkpoint[]>([]);  
 const [selectedCheckpoints, setSelectedCheckpoints] = useState<string[]>([]);  
 const [loading, setLoading] = useState<boolean>(true);  
  
 // Load checkpoints using API  
 const loadCheckpoints = async () => {  
 try {  
 setLoading(true);  
 const result = await checkpointApi.fetchCheckpoints(jobId);  
 setCheckpoints(result);  
 } catch (err) {  
 // Error handling  
 } finally {  
 setLoading(false);  
 }  
 };  
  
 // Fetch checkpoints when component mounts or jobId changes  
 useEffect(() => {  
 loadCheckpoints();  
 }, [jobId]);  
  
 return (  
 <Box>  
 {/\* Create checkpoint button \*/}  
 {/\* Checkpoints table \*/}  
 {/\* Checkpoint comparison section \*/}  
 {/\* Dialogs for checkpoint operations \*/}  
 </Box>  
 );  
};

#### ParameterTuner

The ParameterTuner component allows administrators to adjust hyperparameters during training. It provides:

* Real-time adjustment of training parameters
* Visualization of parameter impact on training
* Preset parameter configurations for common scenarios
* Advanced parameter scheduling

##### Key Features

* Dynamic Parameter Updates: Adjust parameters while training is in progress
* Parameter Presets: Apply predefined parameter sets for common scenarios
* Parameter Validation: Ensure parameters stay within valid ranges
* Parameter Scheduling: Set up automatic parameter changes during training
* Impact Analysis: Visualize the impact of parameter changes on training metrics

##### Implementation

// Simplified implementation  
const ParameterTuner: React.FC<ParameterTunerProps> = ({  
 jobId,  
 modelType  
}) => {  
 const [parameters, setParameters] = useState<Record<string, number>>({});  
 const [presets, setPresets] = useState<ParameterPreset[]>([]);  
 const [loading, setLoading] = useState<boolean>(true);  
  
 // Load current parameters  
 useEffect(() => {  
 // Implementation details  
 }, [jobId]);  
  
 const handleParameterChange = async (key: string, value: number) => {  
 try {  
 await parameterApi.updateParameter(jobId, key, value);  
 setParameters(prev => ({ ...prev, [key]: value }));  
 } catch (err) {  
 // Error handling  
 }  
 };  
  
 return (  
 <Box>  
 <ParameterControls   
 parameters={parameters}  
 onChange={handleParameterChange}  
 presets={presets}  
 onApplyPreset={handleApplyPreset}  
 />  
  
 <ParameterImpactChart   
 jobId={jobId}  
 parameterChanges={parameterChanges}  
 />  
 </Box>  
 );  
};

// Simplified implementation  
const ParameterTuner: React.FC<ParameterTunerProps> = ({  
 jobId,  
 modelType  
}) => {  
 const [parameters, setParameters] = useState<Record<string, number>>({});  
 const [presets, setPresets] = useState<ParameterPreset[]>([]);  
 const [loading, setLoading] = useState<boolean>(true);  
  
 // Load current parameters  
 useEffect(() => {  
 // Implementation details  
 }, [jobId]);  
  
 const handleParameterChange = async (key: string, value: number) => {  
 try {  
 await parameterApi.updateParameter(jobId, key, value);  
 setParameters(prev => ({ ...prev, [key]: value }));  
 } catch (err) {  
 // Error handling  
 }  
 };  
  
 return (  
 <Box>  
 <ParameterControls   
 parameters={parameters}  
 onChange={handleParameterChange}  
 presets={presets}  
 onApplyPreset={handleApplyPreset}  
 />  
  
 <ParameterImpactChart   
 jobId={jobId}  
 parameterChanges={parameterChanges}  
 />  
 </Box>  
 );  
};

### Integration with Admin Panel

The Training Monitoring System is integrated into the admin panel through a dedicated "Training" page. This page provides access to all training monitoring capabilities and is accessible to administrators with appropriate permissions.

#### URL Structure

* /admin/training - Main training dashboard
* /admin/training/:jobId - Specific training job monitoring

/admin/training

/admin/training/:jobId

#### Access Control

The training monitoring features require specific permissions:

* training:view - View training metrics and checkpoints
* training:manage - Create checkpoints and adjust parameters
* training:admin - Roll back to previous checkpoints and manage training jobs

training:view

training:manage

training:admin

### API Integration

The Training Monitoring System integrates with several backend APIs:

#### Metrics API

* GET /api/admin/training/:jobId/metrics - Fetch training metrics
* GET /api/admin/training/:jobId/metrics/latest - Get latest metrics

GET /api/admin/training/:jobId/metrics

GET /api/admin/training/:jobId/metrics/latest

#### Checkpoint API

* GET /api/admin/training/:jobId/checkpoints - List all checkpoints
* POST /api/admin/training/:jobId/checkpoints - Create a new checkpoint
* PUT /api/admin/training/:jobId/checkpoints/:checkpointId/rollback - Roll back to a checkpoint
* DELETE /api/admin/training/:jobId/checkpoints/:checkpointId - Delete a checkpoint

GET /api/admin/training/:jobId/checkpoints

POST /api/admin/training/:jobId/checkpoints

PUT /api/admin/training/:jobId/checkpoints/:checkpointId/rollback

DELETE /api/admin/training/:jobId/checkpoints/:checkpointId

#### Parameter API

* GET /api/admin/training/:jobId/parameters - Get current parameters
* PUT /api/admin/training/:jobId/parameters/:key - Update a parameter
* POST /api/admin/training/:jobId/parameters/preset/:presetId - Apply a parameter preset

GET /api/admin/training/:jobId/parameters

PUT /api/admin/training/:jobId/parameters/:key

POST /api/admin/training/:jobId/parameters/preset/:presetId

### Usage Examples

#### Monitoring Training Progress

1. Navigate to Admin Panel > Training
2. Select an active training job
3. View the MetricsVisualizer tab to monitor training progress
4. Customize the chart view to focus on relevant metrics
5. Export metrics data if needed for further analysis

#### Managing Checkpoints

1. Navigate to Admin Panel > Training > [Job ID]
2. Click on the Checkpoints tab
3. View existing checkpoints and their metrics
4. Create a new checkpoint with a descriptive name and relevant tags
5. Compare checkpoints by selecting two checkpoints for side-by-side comparison
6. Roll back to a previous checkpoint if needed

#### Tuning Parameters

1. Navigate to Admin Panel > Training > [Job ID]
2. Click on the Parameters tab
3. Adjust parameters as needed based on training performance
4. Apply a parameter preset for common scenarios
5. Observe the impact of parameter changes on the training metrics

### Best Practices

1. Regular Checkpointing: Create checkpoints at key moments during training to enable easy rollback if needed
2. Descriptive Naming: Use clear, descriptive names and tags for checkpoints to facilitate management
3. Parameter Tuning: Make small, incremental changes to parameters to understand their impact
4. Metric Monitoring: Focus on multiple metrics to get a comprehensive view of training performance
5. Comparison Analysis: Regularly compare checkpoints to understand the impact of changes
6. Documentation: Document parameter changes and their impacts for future reference

### Troubleshooting

Common issues and their solutions:

Table content:

Issue | Solution

Metrics not updating | Check that the training job is active and properly connected to the metrics system

Checkpoint creation fails | Ensure sufficient storage space and proper permissions

Parameter changes have no effect | Verify that the training system supports live parameter updates

Chart display issues | Try adjusting the timeframe or refreshing the page

Rollback operation fails | Check training job status and ensure the checkpoint is compatible

### Future Enhancements

Planned improvements for the Training Monitoring System:

1. Automated Checkpoint Recommendations: AI-driven suggestions for when to create checkpoints
2. Advanced Visualization Tools: 3D parameter space visualization and correlation analysis
3. Collaborative Annotations: Allow team members to annotate checkpoints and training runs
4. Predictive Analytics: Predict training outcomes based on current metrics and parameters
5. Integration with Experiment Tracking: Connect with experiment tracking systems like MLflow or Weights & Biases

### Related Documentation

* ML Training API Improvements
* Monitoring System
* Admin Panel
* Model Extension Guide

# Training With Existing Materials

Source: readme/training-with-existing-materials.md

---

## Training with Existing Materials

This guide explains how to train the system with existing materials, focusing on dataset import capabilities and model training.

### Dataset Import Options

Our system supports multiple methods for importing existing material data:

#### 1. Hugging Face Datasets

The system can import datasets directly from Hugging Face:

// Example dataset: Material in Context (MINC-2500)  
// https://huggingface.co/datasets/mcimpoi/minc-2500\_split\_1

// Example dataset: Material in Context (MINC-2500)  
// https://huggingface.co/datasets/mcimpoi/minc-2500\_split\_1

Import Process:

1. Navigate to the Admin Panel → Datasets section
2. Select "Import External Dataset"
3. Enter the Hugging Face dataset ID: mcimpoi/minc-2500\_split\_1
4. Configure import options:
5. Map dataset categories to system material types
6. Set maximum images per class (recommended: 250-500)
7. Enable/disable metadata import
8. Select specific classes to import (optional)
9. Start the import process

mcimpoi/minc-2500\_split\_1

The system will:  
- Download dataset samples from Hugging Face  
- Extract category and image information  
- Map external dataset fields to internal metadata fields  
- Create properly categorized material samples

#### 2. CSV/Structured Dataset Import

For CSV or structured datasets with mapped fields:

1. Prepare a CSV with columns mapping to system fields
2. Include material type, properties, and image file paths
3. Upload through the Admin Panel → Datasets → Import CSV

#### 3. Local Directory Import

For datasets stored in local directories:

1. Organize materials by category in subdirectories
2. Specify the root directory path during import
3. The system will analyze the structure and suggest mappings

### Field Mapping System

The system includes a flexible field mapping capability:

* Maps external dataset fields to internal metadata fields
* Provides predefined mappings for common material datasets
* Allows custom mapping configuration
* Handles automatic property extraction

### Training Models with Imported Materials

After importing materials, you can train recognition models:

1. Navigate to Admin Panel → Models → Training
2. Select the imported dataset from available datasets
3. Configure training parameters:
4. Base model (ResNet, MobileNet, etc.)
5. Batch size and learning rate
6. Number of epochs
7. Transfer learning settings
8. Start training
9. Monitor progress in real-time
10. Evaluate model performance with validation metrics

#### Detailed Training Process

The training process involves several sophisticated steps:

1. Model Initialization
2. Base models are loaded dynamically from ML framework libraries (not stored in our repo)
3. TensorFlow or PyTorch pre-trained architectures (MobileNetV2, ResNet, EfficientNet)
4. Classification layers are replaced with custom layers for material recognition
5. Transfer Learning Optimization
6. Initial layers of base models are frozen to preserve general features
7. Only the top classification layers are trained initially
8. Sparse categorical cross-entropy loss is used for classification tasks
9. Adam optimizer with carefully tuned learning rate (typically 0.0001)
10. Later training phases gradually unfreeze more layers for fine-tuning
11. Training Enhancement Techniques
12. Early stopping with validation loss monitoring
13. Learning rate reduction on plateau
14. Data augmentation specific to material properties
15. Regularization to prevent overfitting (dropout, L2)
16. Model Storage and Versioning
17. Trained models are saved with metadata in the output directory structure:  
     /models/  
     ├── {model\_id}/  
     │ ├── model.h5 (or .pt for PyTorch)  
     │ ├── metadata.json  
     │ ├── training\_history.json  
     │ └── hyperparameters.json
18. Complete training history is preserved for analysis
19. Models are versioned for tracking improvements
20. Vector Database Integration
21. The trained models generate embeddings for all materials
22. These embeddings (not the models themselves) are stored in the vector database
23. FAISS indexing enables efficient similarity search
24. Each material's embedding links to knowledge base entries through material IDs

Classification layers are replaced with custom layers for material recognition

Transfer Learning Optimization

Later training phases gradually unfreeze more layers for fine-tuning

Training Enhancement Techniques

Regularization to prevent overfitting (dropout, L2)

Model Storage and Versioning

/models/  
 ├── {model\_id}/  
 │ ├── model.h5 (or .pt for PyTorch)  
 │ ├── metadata.json  
 │ ├── training\_history.json  
 │ └── hyperparameters.json

Models are versioned for tracking improvements

Vector Database Integration

### Example: Training with MINC-2500

For the MINC-2500 dataset, which contains material images across 10 categories:

1. Import the dataset using the Hugging Face importer:
2. Use ID: mcimpoi/minc-2500\_split\_1
3. The system will automatically map categories like 'wood', 'metal', 'fabric', etc.
4. System will assign appropriate material types based on content
5. Train a material recognition model:
6. Use transfer learning on a pre-trained image model
7. Configure 10-20 epochs for good results
8. Enable data augmentation for improved generalization
9. Set learning rate to ~0.0001 for stable training
10. Apply sparse categorical cross-entropy loss for classification
11. Evaluate results:
12. The system will display accuracy per material category
13. Review performance metrics to identify areas for improvement
14. Test with sample images to verify recognition quality
15. Analyze confusion matrix to understand misclassifications
16. Review embedding quality metrics for similarity search applications
17. Model Deployment:
18. The trained model is automatically versioned and stored
19. Embeddings are generated for all materials in the dataset
20. Vector database is updated with new embeddings
21. Recognition system starts using the new model immediately

mcimpoi/minc-2500\_split\_1

System will assign appropriate material types based on content

Train a material recognition model:

Apply sparse categorical cross-entropy loss for classification

Evaluate results:

Review embedding quality metrics for similarity search applications

Model Deployment:

### Implementation Notes

* The dataset importer supports automatic detection of dataset structure
* Field mapping can be customized for specific material types
* Metadata extracted from datasets is stored for future reference
* Training parameters are automatically optimized based on dataset characteristics

For more advanced training techniques, see the ML Training Documentation.

# Unified Services Implementation

Source: readme/unified-services-implementation.md

---

## Unified Services Implementation

This document describes the implementation of the unified services architecture in the KAI platform. The unified services architecture consolidates duplicate implementations of common functionality into a single, shared implementation to improve maintainability and ensure consistent behavior.

### Implementation Overview

The unified services architecture was implemented in the following steps:

1. Created unified service implementations in the shared package:
2. Storage service with provider pattern (Supabase, S3)
3. Authentication service with provider pattern
4. API client with retry logic and error handling
5. Configuration management
6. Logging
7. Updated the server package to use the unified services:
8. Updated the storage initializer
9. Created adapter files for backward compatibility
10. Updated the Supabase client
11. Updated the MCP client to use the unified API client
12. Documented the unified services architecture in the readme/unified-services.md file

Logging

Updated the server package to use the unified services:

Updated the Supabase client

Updated the MCP client to use the unified API client

Documented the unified services architecture in the readme/unified-services.md file

### Unified Services

#### Storage Service

The unified storage service provides a consistent interface for file storage operations across different storage backends (Supabase Storage, S3, etc.). It uses a provider pattern to allow switching between storage backends without changing the client code.

Key files:  
- packages/shared/src/services/storage/unifiedStorageService.ts  
- packages/shared/src/services/storage/supabaseStorageProvider.ts  
- packages/shared/src/services/storage/s3StorageProvider.ts  
- packages/shared/src/services/storage/storageInitializer.ts

packages/shared/src/services/storage/unifiedStorageService.ts

packages/shared/src/services/storage/supabaseStorageProvider.ts

packages/shared/src/services/storage/s3StorageProvider.ts

packages/shared/src/services/storage/storageInitializer.ts

#### Authentication Service

The unified authentication service provides a consistent interface for authentication operations across different authentication providers (Supabase Auth, JWT, API keys, etc.). It uses a provider pattern to allow switching between authentication providers without changing the client code.

Key files:  
- packages/shared/src/services/auth/authService.ts  
- packages/shared/src/services/auth/supabaseAuthProvider.ts  
- packages/shared/src/services/auth/authInitializer.ts

packages/shared/src/services/auth/authService.ts

packages/shared/src/services/auth/supabaseAuthProvider.ts

packages/shared/src/services/auth/authInitializer.ts

#### API Client

The unified API client provides a consistent interface for making HTTP requests with built-in error handling, authentication, and retry logic. It serves as a base class for specialized API clients like the MCP client.

Key files:  
- packages/shared/src/services/api/apiClient.ts  
- packages/shared/src/services/api/mcpClient.ts  
- packages/mcp-client/src/index.ts

packages/shared/src/services/api/apiClient.ts

packages/shared/src/services/api/mcpClient.ts

packages/mcp-client/src/index.ts

#### Configuration Management

The unified configuration system provides a consistent interface for accessing configuration values across the application. It supports environment-specific configuration, hierarchical configuration with overrides, and default values for missing configuration.

Key files:  
- packages/shared/src/utils/unified-config.ts

packages/shared/src/utils/unified-config.ts

#### Logging

The unified logging system provides a consistent interface for logging across the application. It supports different log levels, contextual logging, and can be extended to support different output formats and destinations.

Key files:  
- packages/shared/src/utils/unified-logger.ts

packages/shared/src/utils/unified-logger.ts

### Server Package Adapters

To maintain backward compatibility with existing code, adapter files were created in the server package that use the unified services but expose the same interface as the old implementations.

Key files:  
- packages/server/src/services/storage/unifiedStorageAdapter.ts  
- packages/server/src/services/storage/storageInitializer.ts  
- packages/server/src/services/storage/s3Service.ts

packages/server/src/services/storage/unifiedStorageAdapter.ts

packages/server/src/services/storage/storageInitializer.ts

packages/server/src/services/storage/s3Service.ts

### MCP Client

The MCP client was updated to use the unified API client as its base class, which provides consistent error handling, retry logic, and other features.

Key files:  
- packages/mcp-client/src/index.ts

packages/mcp-client/src/index.ts

### Benefits

The unified services architecture provides several benefits:

1. Reduced code duplication: Common functionality is implemented once in the shared package.
2. Improved maintainability: Changes to common functionality only need to be made in one place.
3. Consistent behavior: All parts of the application use the same implementation of common functionality.
4. Type safety: The unified services provide type-safe interfaces for common operations.
5. Extensibility: The provider pattern allows adding new implementations without changing client code.

### Next Steps

The following steps are recommended to further improve the unified services architecture:

1. Update client package to use the unified services
2. Update agents package to use the unified services
3. Update ml package to use the unified services
4. Add more storage providers (Google Cloud Storage, Azure Blob Storage, etc.)
5. Add more authentication providers (OAuth, SAML, etc.)
6. Improve error handling and retry logic
7. Add more logging destinations (Elasticsearch, Datadog, etc.)
8. Add more configuration sources (environment variables, JSON files, etc.)
9. Expand API client capabilities with more specialized clients
10. Add more database providers beyond Supabase
11. Implement caching mechanisms for improved performance
12. Add more comprehensive monitoring and telemetry

# Unified Services

Source: readme/unified-services.md

---

## Unified Services

This document describes the unified services architecture implemented to reduce code duplication and standardize service implementations across the KAI platform. The unified services architecture consolidates duplicate implementations of common functionality into a single, shared implementation to improve maintainability and ensure consistent behavior.

### Overview

The unified services architecture consolidates duplicate implementations of common functionality into a single, shared implementation. This reduces code duplication, improves maintainability, and ensures consistent behavior across the application.

The unified services are implemented in the @kai/shared package and can be used by all other packages in the monorepo.

@kai/shared

### Unified Services

#### Storage Service

The unified storage service provides a consistent interface for file storage operations across different storage backends (Supabase Storage, S3, etc.).

import { storage, initializeStorage } from '@kai/shared';  
  
// Initialize storage (done automatically by initializeServices)  
initializeStorage();  
  
// Upload a file  
const result = await storage.uploadFile('/path/to/file.jpg', 'uploads/file.jpg', {  
 isPublic: true,  
 contentType: 'image/jpeg',  
 metadata: {  
 userId: '123',  
 source: 'user-upload'  
 }  
});  
  
// Get the URL  
console.log(result.url);

import { storage, initializeStorage } from '@kai/shared';  
  
// Initialize storage (done automatically by initializeServices)  
initializeStorage();  
  
// Upload a file  
const result = await storage.uploadFile('/path/to/file.jpg', 'uploads/file.jpg', {  
 isPublic: true,  
 contentType: 'image/jpeg',  
 metadata: {  
 userId: '123',  
 source: 'user-upload'  
 }  
});  
  
// Get the URL  
console.log(result.url);

#### Authentication Service

The unified authentication service provides a consistent interface for authentication operations across different authentication providers (Supabase Auth, JWT, API keys, etc.).

import { auth, initializeAuth } from '@kai/shared';  
  
// Initialize auth (done automatically by initializeServices)  
initializeAuth();  
  
// Login  
const result = await auth.login({  
 email: 'user@example.com',  
 password: 'password'  
});  
  
// Get the current user  
const user = await auth.getUser();  
  
// Check if the user has a role  
const isAdmin = await auth.hasRole('admin');

import { auth, initializeAuth } from '@kai/shared';  
  
// Initialize auth (done automatically by initializeServices)  
initializeAuth();  
  
// Login  
const result = await auth.login({  
 email: 'user@example.com',  
 password: 'password'  
});  
  
// Get the current user  
const user = await auth.getUser();  
  
// Check if the user has a role  
const isAdmin = await auth.hasRole('admin');

#### API Client

The unified API client provides a consistent interface for making HTTP requests with built-in error handling, authentication, retry logic, and caching.

import { apiClient, createApiClient } from '@kai/shared';  
  
// Use the default client  
const result = await apiClient.get('/api/materials');  
  
// Create a custom client  
const customClient = createApiClient({  
 baseURL: 'https://api.example.com',  
 timeout: 60000,  
 useAuth: false,  
 useCache: true,  
 cacheTtl: 300 // 5 minutes  
});  
  
// Make a request with the custom client  
const result = await customClient.post('/api/materials', {  
 name: 'New Material',  
 type: 'ceramic'  
});  
  
// Clear the cache for a specific endpoint  
await apiClient.clearCache('/api/materials');

import { apiClient, createApiClient } from '@kai/shared';  
  
// Use the default client  
const result = await apiClient.get('/api/materials');  
  
// Create a custom client  
const customClient = createApiClient({  
 baseURL: 'https://api.example.com',  
 timeout: 60000,  
 useAuth: false,  
 useCache: true,  
 cacheTtl: 300 // 5 minutes  
});  
  
// Make a request with the custom client  
const result = await customClient.post('/api/materials', {  
 name: 'New Material',  
 type: 'ceramic'  
});  
  
// Clear the cache for a specific endpoint  
await apiClient.clearCache('/api/materials');

#### MCP Client

The unified MCP (Model Context Protocol) client provides a consistent interface for interacting with the MCP server.

import { mcpClient, createMCPClient } from '@kai/shared';  
  
// Use the default client  
const result = await mcpClient.recognizeMaterial('/path/to/image.jpg', {  
 modelType: 'hybrid',  
 confidenceThreshold: 0.7  
});  
  
// Create a custom client  
const customClient = createMCPClient({  
 baseURL: 'http://localhost:8000',  
 timeout: 60000  
});  
  
// Make a request with the custom client  
const models = await customClient.listModels();

import { mcpClient, createMCPClient } from '@kai/shared';  
  
// Use the default client  
const result = await mcpClient.recognizeMaterial('/path/to/image.jpg', {  
 modelType: 'hybrid',  
 confidenceThreshold: 0.7  
});  
  
// Create a custom client  
const customClient = createMCPClient({  
 baseURL: 'http://localhost:8000',  
 timeout: 60000  
});  
  
// Make a request with the custom client  
const models = await customClient.listModels();

#### Cache Service

The unified cache service provides a consistent interface for caching operations across different cache backends (Memory, Redis, etc.).

import { cache, initializeCache } from '@kai/shared';  
  
// Initialize cache (done automatically by initializeServices)  
initializeCache();  
  
// Set a value in the cache  
await cache.set('user:123', { name: 'John', email: 'john@example.com' }, {  
 ttl: 3600, // 1 hour  
 namespace: 'users'  
});  
  
// Get a value from the cache  
const user = await cache.get<User>('user:123', { namespace: 'users' });  
  
// Delete a value from the cache  
await cache.delete('user:123', { namespace: 'users' });  
  
// Check if a key exists in the cache  
const exists = await cache.has('user:123', { namespace: 'users' });  
  
// Clear all values from a namespace  
await cache.clear({ namespace: 'users' });  
  
// Cache a function result  
const getCachedUserById = cache.cached(  
 getUserById,  
 (id) => `user:${id}`,  
 { ttl: 3600, namespace: 'users' }  
);  
  
// Use the cached function  
const user = await getCachedUserById('123');

import { cache, initializeCache } from '@kai/shared';  
  
// Initialize cache (done automatically by initializeServices)  
initializeCache();  
  
// Set a value in the cache  
await cache.set('user:123', { name: 'John', email: 'john@example.com' }, {  
 ttl: 3600, // 1 hour  
 namespace: 'users'  
});  
  
// Get a value from the cache  
const user = await cache.get<User>('user:123', { namespace: 'users' });  
  
// Delete a value from the cache  
await cache.delete('user:123', { namespace: 'users' });  
  
// Check if a key exists in the cache  
const exists = await cache.has('user:123', { namespace: 'users' });  
  
// Clear all values from a namespace  
await cache.clear({ namespace: 'users' });  
  
// Cache a function result  
const getCachedUserById = cache.cached(  
 getUserById,  
 (id) => `user:${id}`,  
 { ttl: 3600, namespace: 'users' }  
);  
  
// Use the cached function  
const user = await getCachedUserById('123');

#### Telemetry Service

The unified telemetry service provides a consistent interface for collecting metrics, traces, and logs across the application.

import { telemetry, initializeTelemetry } from '@kai/shared';  
  
// Initialize telemetry (done automatically by initializeServices)  
await initializeTelemetry();  
  
// Track an event  
await telemetry.trackEvent({  
 type: 'user',  
 name: 'user\_login',  
 timestamp: Date.now(),  
 status: 'success',  
 properties: {  
 userId: '123',  
 method: 'password'  
 }  
});  
  
// Track an error  
await telemetry.trackEvent({  
 type: 'error',  
 name: 'api\_error',  
 timestamp: Date.now(),  
 status: 'error',  
 properties: {  
 endpoint: '/api/users',  
 method: 'GET'  
 },  
 error: {  
 message: 'Failed to fetch users',  
 stack: error.stack,  
 name: error.name  
 }  
});  
  
// Measure performance  
const id = telemetry.startMeasurement('database\_query', {  
 query: 'SELECT \* FROM users'  
});  
  
try {  
 const result = await db.query('SELECT \* FROM users');  
 await telemetry.stopMeasurement(id, { success: true, count: result.length });  
 return result;  
} catch (error) {  
 await telemetry.stopMeasurement(id, { success: false, error: error.message });  
 throw error;  
}  
  
// Automatic measurement  
const result = await telemetry.measure(  
 'database\_query',  
 async () => {  
 return await db.query('SELECT \* FROM users');  
 },  
 { query: 'SELECT \* FROM users' }  
);

import { telemetry, initializeTelemetry } from '@kai/shared';  
  
// Initialize telemetry (done automatically by initializeServices)  
await initializeTelemetry();  
  
// Track an event  
await telemetry.trackEvent({  
 type: 'user',  
 name: 'user\_login',  
 timestamp: Date.now(),  
 status: 'success',  
 properties: {  
 userId: '123',  
 method: 'password'  
 }  
});  
  
// Track an error  
await telemetry.trackEvent({  
 type: 'error',  
 name: 'api\_error',  
 timestamp: Date.now(),  
 status: 'error',  
 properties: {  
 endpoint: '/api/users',  
 method: 'GET'  
 },  
 error: {  
 message: 'Failed to fetch users',  
 stack: error.stack,  
 name: error.name  
 }  
});  
  
// Measure performance  
const id = telemetry.startMeasurement('database\_query', {  
 query: 'SELECT \* FROM users'  
});  
  
try {  
 const result = await db.query('SELECT \* FROM users');  
 await telemetry.stopMeasurement(id, { success: true, count: result.length });  
 return result;  
} catch (error) {  
 await telemetry.stopMeasurement(id, { success: false, error: error.message });  
 throw error;  
}  
  
// Automatic measurement  
const result = await telemetry.measure(  
 'database\_query',  
 async () => {  
 return await db.query('SELECT \* FROM users');  
 },  
 { query: 'SELECT \* FROM users' }  
);

#### Configuration

The unified configuration system provides a consistent interface for accessing configuration values across the application.

import { config } from '@kai/shared';  
  
// Initialize configuration (done automatically by initializeServices)  
config.init({  
 environment: 'development',  
 envPath: '/path/to/env/files'  
});  
  
// Get a configuration value  
const apiUrl = config.get('api').url;  
  
// Set a configuration value  
config.set('api', {  
 url: 'https://api.example.com',  
 timeout: 60000,  
 version: '1.0.0'  
});

import { config } from '@kai/shared';  
  
// Initialize configuration (done automatically by initializeServices)  
config.init({  
 environment: 'development',  
 envPath: '/path/to/env/files'  
});  
  
// Get a configuration value  
const apiUrl = config.get('api').url;  
  
// Set a configuration value  
config.set('api', {  
 url: 'https://api.example.com',  
 timeout: 60000,  
 version: '1.0.0'  
});

#### Logging

The unified logging system provides a consistent interface for logging across the application.

import { createLogger } from '@kai/shared';  
  
// Create a logger for a specific module  
const logger = createLogger('MyModule');  
  
// Log messages  
logger.debug('Debug message');  
logger.info('Info message');  
logger.warn('Warning message');  
logger.error('Error message', { error: new Error('Something went wrong') });

import { createLogger } from '@kai/shared';  
  
// Create a logger for a specific module  
const logger = createLogger('MyModule');  
  
// Log messages  
logger.debug('Debug message');  
logger.info('Info message');  
logger.warn('Warning message');  
logger.error('Error message', { error: new Error('Something went wrong') });

### Initialization

All unified services can be initialized at once using the initializeServices function:

initializeServices

import { initializeServices } from '@kai/shared';  
  
// Initialize all services  
await initializeServices({  
 environment: 'development',  
 envPath: '/path/to/env/files',  
 configOverrides: {  
 api: {  
 url: 'https://api.example.com'  
 },  
 cache: {  
 type: 'memory',  
 namespace: 'default',  
 ttl: 3600  
 },  
 telemetry: {  
 enabled: true,  
 type: 'console'  
 }  
 }  
});

import { initializeServices } from '@kai/shared';  
  
// Initialize all services  
await initializeServices({  
 environment: 'development',  
 envPath: '/path/to/env/files',  
 configOverrides: {  
 api: {  
 url: 'https://api.example.com'  
 },  
 cache: {  
 type: 'memory',  
 namespace: 'default',  
 ttl: 3600  
 },  
 telemetry: {  
 enabled: true,  
 type: 'console'  
 }  
 }  
});

The services are initialized in the following order:

1. Configuration (config)
2. Storage (storage)
3. Authentication (auth)
4. Cache (cache)
5. Telemetry (telemetry)

Each service can also be initialized individually if needed:

import {  
 config,  
 initializeStorage,  
 initializeAuth,  
 initializeCache,  
 initializeTelemetry  
} from '@kai/shared';  
  
// Initialize configuration  
config.init({  
 environment: 'development',  
 envPath: '/path/to/env/files'  
});  
  
// Initialize storage  
initializeStorage();  
  
// Initialize authentication  
initializeAuth();  
  
// Initialize cache  
initializeCache();  
  
// Initialize telemetry  
await initializeTelemetry();

import {  
 config,  
 initializeStorage,  
 initializeAuth,  
 initializeCache,  
 initializeTelemetry  
} from '@kai/shared';  
  
// Initialize configuration  
config.init({  
 environment: 'development',  
 envPath: '/path/to/env/files'  
});  
  
// Initialize storage  
initializeStorage();  
  
// Initialize authentication  
initializeAuth();  
  
// Initialize cache  
initializeCache();  
  
// Initialize telemetry  
await initializeTelemetry();

### Migration Guide

To migrate from the old, duplicate implementations to the unified services:

1. Storage Services:
2. Replace uploadToStorage, uploadBufferToStorage, etc. with storage.uploadFile, storage.uploadBuffer, etc.
3. Replace uploadToS3, uploadBufferToS3, etc. with the same unified methods.
4. Authentication Services:
5. Replace package-specific auth services with auth.login, auth.register, etc.
6. Replace token management with auth.getToken, auth.setToken, etc.
7. API Clients:
8. Replace package-specific API clients with apiClient.get, apiClient.post, etc.
9. Replace MCP client implementations with mcpClient.recognizeMaterial, mcpClient.listModels, etc.
10. Configuration:
11. Replace package-specific configuration with config.get, config.set, etc.
12. Logging:
13. Replace package-specific loggers with createLogger('ModuleName').
14. Caching:
15. Replace custom caching implementations with cache.get, cache.set, etc.
16. Use the cache.cached method to cache function results.
17. Telemetry:
18. Replace custom telemetry implementations with telemetry.trackEvent, etc.
19. Use the telemetry.measure method to measure function performance.

uploadToStorage

uploadBufferToStorage

storage.uploadFile

storage.uploadBuffer

Replace uploadToS3, uploadBufferToS3, etc. with the same unified methods.

uploadToS3

uploadBufferToS3

Authentication Services:

auth.login

auth.register

Replace token management with auth.getToken, auth.setToken, etc.

auth.getToken

auth.setToken

API Clients:

apiClient.get

apiClient.post

Replace MCP client implementations with mcpClient.recognizeMaterial, mcpClient.listModels, etc.

mcpClient.recognizeMaterial

mcpClient.listModels

Configuration:

Replace package-specific configuration with config.get, config.set, etc.

config.get

config.set

Logging:

Replace package-specific loggers with createLogger('ModuleName').

createLogger('ModuleName')

Caching:

cache.get

cache.set

Use the cache.cached method to cache function results.

cache.cached

Telemetry:

telemetry.trackEvent

telemetry.measure

### Best Practices

1. Always use the unified services instead of creating new implementations.
2. Initialize services early in the application lifecycle.
3. Use type-safe methods provided by the unified services.
4. Extend the unified services if you need custom functionality, rather than creating new implementations.
5. Contribute improvements to the unified services rather than working around limitations.

### Consolidated Services

The following services have been consolidated into unified implementations:

#### Storage Services

* supabaseStorageService.ts and s3Service.ts in the server package
* storageService.ts and s3StorageAdapter.ts in the shared package
* Various utility functions for file uploads in different modules

supabaseStorageService.ts

s3Service.ts

storageService.ts

s3StorageAdapter.ts

#### Authentication Services

* auth.middleware.ts in the server package
* authService.ts in the agents package
* supabaseAuth.service.ts in the client package
* Duplicate token handling across different services

auth.middleware.ts

authService.ts

supabaseAuth.service.ts

#### Logging Implementations

* logger.ts in the server package
* logger.ts in the ml package
* logger.ts in the agents package
* logger.ts in the shared package

logger.ts

logger.ts

logger.ts

logger.ts

#### Environment Configuration

* environment.ts in the agents package
* environment.ts in the shared package
* config.ts in the client package
* config.ts in the shared package
* Multiple .env files with overlapping variables

environment.ts

environment.ts

config.ts

config.ts

#### API Clients

* mcpClientService.ts in the server package
* mcp-integration.ts in the ml package
* apiClient.ts in the client package
* huggingFaceClient.ts in the server package

mcpClientService.ts

mcp-integration.ts

apiClient.ts

huggingFaceClient.ts

#### Supabase Integration

* supabaseClient.ts in the server package
* supabaseClient.ts in the client package
* supabaseClient.ts in the shared package
* supabaseHelper.ts in the server package
* supabaseHelpers.ts in the shared package

supabaseClient.ts

supabaseClient.ts

supabaseClient.ts

supabaseHelper.ts

supabaseHelpers.ts

### Future Improvements

1. Add more storage providers (Google Cloud Storage, Azure Blob Storage, etc.)
2. Enhance authentication:
3. Add direct SAML support (if not already provided by Supabase)
4. Implement custom multi-factor authentication flows
5. Support passwordless authentication methods
6. Add biometric authentication options
7. Implement social login analytics and conversion tracking
8. Improve error handling and retry logic
9. Enhance logging:
10. Add more logging destinations (Elasticsearch, Datadog, etc.)
11. Implement structured logging
12. Add log rotation and archiving
13. Enhance configuration:
14. Add more configuration sources (environment variables, JSON files, etc.)
15. Implement configuration validation
16. Add support for dynamic configuration updates
17. Expand API client capabilities:
18. Add more specialized clients (GraphQL, WebSocket, etc.)
19. Implement request batching and deduplication
20. Add support for API versioning
21. Add more database providers beyond Supabase
22. Enhance caching:
23. Add more cache providers (Memcached, DynamoDB, etc.)
24. Implement cache invalidation based on events
25. Add cache warming and prefetching
26. Implement distributed caching
27. Enhance telemetry:
28. Add more telemetry providers (Application Insights, Datadog, etc.)
29. Implement distributed tracing
30. Add support for custom dimensions and metrics
31. Implement sampling and filtering
32. Add alerting based on telemetry data

# Vercel Deployment Guide

Source: readme/vercel-deployment-guide.md

---

## Vercel Deployment Guide for Kai

This guide provides detailed instructions for deploying the Kai application's frontend components to Vercel, including the Next.js admin panel and Gatsby client application.

### Table of Contents

* Prerequisites
* Vercel Account Setup
* Admin Panel Deployment (Next.js)
* Client App Deployment (Gatsby)
* Environment Variables
* Custom Domain Configuration
* Deployment Settings
* Preview Deployments
* Monitoring and Analytics
* Troubleshooting

### Prerequisites

Before proceeding with Vercel deployment, ensure you have:

* A GitHub account with the Kai repository
* A Vercel account
* Supabase project already set up (see the Supabase Setup Guide)
* Backend services deployed to Digital Ocean Kubernetes (see the Digital Ocean Kubernetes Setup Guide)
* Domain name(s) for your application

### Vercel Account Setup

#### Step 1: Create a Vercel Account

If you don't have a Vercel account, create one at https://vercel.com/signup. It's recommended to sign up using your GitHub account for seamless integration.

#### Step 2: Connect GitHub Repository

1. After signing in to Vercel, click Add New... → Project
2. Select your GitHub account
3. Find and select the Kai repository
4. If you don't see your repository, click Adjust GitHub App Permissions and grant Vercel access to the repository

### Admin Panel Deployment (Next.js)

The admin panel is a Next.js application located in packages/admin.

packages/admin

#### Step 1: Import Project

1. Click Import on the Kai repository
2. Configure the project settings:
3. Framework Preset: Next.js
4. Root Directory: packages/admin
5. Build Command: yarn build (default)
6. Output Directory: .next (default for Next.js)
7. Install Command: cd ../.. && yarn install

packages/admin

yarn build

.next

cd ../.. && yarn install

#### Step 2: Configure Environment Variables

Add the following environment variables:

NEXT\_PUBLIC\_API\_URL=https://api.kai.yourdomain.com  
NEXT\_PUBLIC\_SUPABASE\_URL=https://your-project.supabase.co  
NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=your-supabase-anon-key

NEXT\_PUBLIC\_API\_URL=https://api.kai.yourdomain.com  
NEXT\_PUBLIC\_SUPABASE\_URL=https://your-project.supabase.co  
NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=your-supabase-anon-key

For more detailed environment variables, see the Environment Variables section below.

#### Step 3: Deploy the Admin Panel

1. Click Deploy
2. Wait for the build and deployment to complete
3. Once deployed, you'll receive a URL like kai-admin.vercel.app

kai-admin.vercel.app

### Client App Deployment (Gatsby)

The client application is a Gatsby app located in packages/client.

packages/client

#### Step 1: Import Project

1. Click Add New... → Project
2. Select your GitHub account and the Kai repository again
3. Configure the project settings:
4. Framework Preset: Gatsby
5. Root Directory: packages/client
6. Build Command: yarn build (default)
7. Output Directory: public (default for Gatsby)
8. Install Command: cd ../.. && yarn install

packages/client

yarn build

public

cd ../.. && yarn install

#### Step 2: Configure Environment Variables

Add the following environment variables:

GATSBY\_API\_URL=https://api.kai.yourdomain.com  
GATSBY\_SUPABASE\_URL=https://your-project.supabase.co  
GATSBY\_SUPABASE\_ANON\_KEY=your-supabase-anon-key  
GATSBY\_STORAGE\_URL=https://your-s3-bucket.s3.amazonaws.com

GATSBY\_API\_URL=https://api.kai.yourdomain.com  
GATSBY\_SUPABASE\_URL=https://your-project.supabase.co  
GATSBY\_SUPABASE\_ANON\_KEY=your-supabase-anon-key  
GATSBY\_STORAGE\_URL=https://your-s3-bucket.s3.amazonaws.com

For more detailed environment variables, see the Environment Variables section below.

#### Step 3: Deploy the Client App

1. Click Deploy
2. Wait for the build and deployment to complete
3. Once deployed, you'll receive a URL like kai-client.vercel.app

kai-client.vercel.app

### Environment Variables

#### Admin Panel Environment Variables

Table content:

Variable | Description | Example

NEXT\_PUBLIC\_API\_URL | URL of your backend API | https://api.kai.yourdomain.com

NEXT\_PUBLIC\_SUPABASE\_URL | URL of your Supabase project | https://your-project.supabase.co

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY | Supabase anonymous key | eyJhbGciOiJIUzI1NiIsImtpZCI6...

NEXT\_PUBLIC\_APP\_ENV | Environment name | production

NEXT\_PUBLIC\_STORAGE\_URL | URL for S3 storage | https://your-s3-bucket.s3.amazonaws.com

NEXT\_PUBLIC\_ENABLE\_ANALYTICS | Enable analytics | true

NEXT\_PUBLIC\_API\_URL

https://api.kai.yourdomain.com

NEXT\_PUBLIC\_SUPABASE\_URL

https://your-project.supabase.co

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY

eyJhbGciOiJIUzI1NiIsImtpZCI6...

NEXT\_PUBLIC\_APP\_ENV

production

NEXT\_PUBLIC\_STORAGE\_URL

https://your-s3-bucket.s3.amazonaws.com

NEXT\_PUBLIC\_ENABLE\_ANALYTICS

true

#### Client App Environment Variables

Table content:

Variable | Description | Example

GATSBY\_API\_URL | URL of your backend API | https://api.kai.yourdomain.com

GATSBY\_SUPABASE\_URL | URL of your Supabase project | https://your-project.supabase.co

GATSBY\_SUPABASE\_ANON\_KEY | Supabase anonymous key | eyJhbGciOiJIUzI1NiIsImtpZCI6...

GATSBY\_STORAGE\_URL | URL for S3 storage | https://your-s3-bucket.s3.amazonaws.com

GATSBY\_DEFAULT\_LOCALE | Default locale | en

GATSBY\_ENABLE\_OFFLINE | Enable offline support | true

GATSBY\_GOOGLE\_ANALYTICS\_ID | Google Analytics ID | G-XXXXXXXXXX

GATSBY\_API\_URL

https://api.kai.yourdomain.com

GATSBY\_SUPABASE\_URL

https://your-project.supabase.co

GATSBY\_SUPABASE\_ANON\_KEY

eyJhbGciOiJIUzI1NiIsImtpZCI6...

GATSBY\_STORAGE\_URL

https://your-s3-bucket.s3.amazonaws.com

GATSBY\_DEFAULT\_LOCALE

en

GATSBY\_ENABLE\_OFFLINE

true

GATSBY\_GOOGLE\_ANALYTICS\_ID

G-XXXXXXXXXX

### Custom Domain Configuration

#### Step 1: Add Custom Domains

For both the admin panel and client app projects:

1. Go to the project in the Vercel dashboard
2. Navigate to Settings → Domains
3. Click Add
4. Enter the domain:
5. Admin Panel: admin.kai.yourdomain.com
6. Client App: kai.yourdomain.com
7. Click Add

admin.kai.yourdomain.com

kai.yourdomain.com

#### Step 2: Configure DNS

Vercel will provide instructions for configuring your DNS settings. You have two options:

Option 1: Using Vercel as the DNS provider  
1. Click Manage DNS on Vercel in the domain settings  
2. Add any additional DNS records as needed

Option 2: Using your own DNS provider  
1. Add a CNAME record pointing to cname.vercel-dns.com:  
 - Admin Panel: admin.kai → cname.vercel-dns.com  
 - Client App: kai → cname.vercel-dns.com  
2. Verify the domain in Vercel

cname.vercel-dns.com

admin.kai

cname.vercel-dns.com

kai

cname.vercel-dns.com

#### Step 3: SSL Configuration

Vercel automatically provisions and renews SSL certificates for your domains. No additional configuration is required.

### Deployment Settings

#### Build & Development Settings

For optimal performance in a monorepo setup:

1. Go to Settings → General → Build & Development Settings
2. Configure the following:
3. Framework Preset: Next.js (admin) or Gatsby (client)
4. Node.js Version: 16.x
5. Include source files outside of the Root Directory in the Build Step: Yes
6. Install Command: cd ../.. && yarn install

cd ../.. && yarn install

#### Build Cache

Enable build cache to speed up deployments:

1. Go to Settings → General → Build & Development Settings
2. Ensure Cache is turned on

#### Production Branch

Configure which branch triggers production deployments:

1. Go to Settings → Git
2. Set Production Branch to main or master

main

master

### Preview Deployments

Vercel automatically creates preview deployments for each pull request:

#### Branch Previews

1. Go to Settings → Git
2. Ensure Preview Deployment for Pull Requests is enabled

#### Preview URLs

Preview deployments will have URLs like:  
- Admin Panel: kai-admin-git-feature-branch-username.vercel.app  
- Client App: kai-client-git-feature-branch-username.vercel.app

kai-admin-git-feature-branch-username.vercel.app

kai-client-git-feature-branch-username.vercel.app

#### Environment Variables for Previews

You can configure different environment variables for preview deployments:

1. Go to Settings → Environment Variables
2. Add the variables you want to override in preview environments
3. Select Preview from the environment dropdown

### Monitoring and Analytics

#### Performance Monitoring

Vercel provides built-in analytics and performance monitoring:

1. Go to the project dashboard
2. Navigate to Analytics
3. View metrics such as:
4. Web Vitals (LCP, CLS, FID)
5. Page views
6. API response times
7. Error rates

#### Error Tracking

To set up error tracking:

1. Go to Settings → Integrations
2. Add an error tracking integration:
3. Sentry
4. Datadog
5. New Relic
6. LogRocket

#### Real User Monitoring

Enable Real User Monitoring (RUM) for detailed performance insights:

1. Go to Settings → Analytics
2. Enable Real User Monitoring

### Troubleshooting

#### Build Failures

If your build fails, check the following:

1. Environment Variables: Ensure all required variables are set
2. Dependencies: Check for missing or incompatible dependencies
3. Examine the build logs in the Vercel dashboard
4. Verify that all dependencies are properly declared in package.json
5. Build Command: Verify the build command is correct
6. For Next.js admin: yarn build
7. For Gatsby client: yarn build
8. Monorepo Issues: If you're having issues with dependencies in the monorepo:
9. Ensure you're using cd ../.. && yarn install as the install command
10. Check that your package.json workspaces are properly configured

Verify that all dependencies are properly declared in package.json

Build Command: Verify the build command is correct

yarn build

For Gatsby client: yarn build

yarn build

Monorepo Issues: If you're having issues with dependencies in the monorepo:

cd ../.. && yarn install

#### Deployment Issues

If your deployment succeeds but the application doesn't work correctly:

1. API Connection: Check that the application can connect to your API
2. Verify the API URL in environment variables
3. Ensure CORS is properly configured on your API
4. Supabase Connection: Verify Supabase connectivity
5. Check Supabase URL and anon key in environment variables
6. Ensure the Supabase project is running and accessible
7. Browser Console Errors: Inspect the browser console for JavaScript errors
8. Use browser developer tools to diagnose client-side issues

Ensure CORS is properly configured on your API

Supabase Connection: Verify Supabase connectivity

Ensure the Supabase project is running and accessible

Browser Console Errors: Inspect the browser console for JavaScript errors

#### Domain Issues

If your custom domain isn't working:

1. DNS Propagation: DNS changes can take up to 48 hours to propagate
2. Use dig or nslookup to check if DNS records have propagated
3. SSL Certificate: Ensure SSL certificate is properly provisioned
4. Vercel should automatically provision certificates
5. Check for SSL errors in the Vercel dashboard
6. Domain Configuration: Verify domain settings in Vercel
7. Ensure your DNS records match Vercel's recommendations

Use dig or nslookup to check if DNS records have propagated

dig

nslookup

SSL Certificate: Ensure SSL certificate is properly provisioned

Check for SSL errors in the Vercel dashboard

Domain Configuration: Verify domain settings in Vercel

### Automatic Deployments

Vercel automatically deploys your application when you push to the production branch (main/master). To set up additional deployment controls:

#### Github Actions Integration

For more advanced deployment workflows, integrate with GitHub Actions:

1. Add a GitHub Actions workflow file at .github/workflows/vercel-deploy.yml:

.github/workflows/vercel-deploy.yml

name: Deploy Frontend to Vercel  
  
on:  
 push:  
 branches: [ main, master ]  
 paths:  
 - 'packages/admin/\*\*'  
 - 'packages/client/\*\*'  
 - 'packages/shared/\*\*'  
 workflow\_dispatch:  
  
jobs:  
 deploy-admin:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
  
 - name: Deploy Admin to Vercel  
 uses: amondnet/vercel-action@v25  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 github-token: ${{ secrets.GITHUB\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_ADMIN\_PROJECT\_ID }}  
 working-directory: ./packages/admin  
 vercel-args: '--prod'  
  
 deploy-client:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
  
 - name: Deploy Client to Vercel  
 uses: amondnet/vercel-action@v25  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 github-token: ${{ secrets.GITHUB\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_CLIENT\_PROJECT\_ID }}  
 working-directory: ./packages/client  
 vercel-args: '--prod'

name: Deploy Frontend to Vercel  
  
on:  
 push:  
 branches: [ main, master ]  
 paths:  
 - 'packages/admin/\*\*'  
 - 'packages/client/\*\*'  
 - 'packages/shared/\*\*'  
 workflow\_dispatch:  
  
jobs:  
 deploy-admin:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
  
 - name: Deploy Admin to Vercel  
 uses: amondnet/vercel-action@v25  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 github-token: ${{ secrets.GITHUB\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_ADMIN\_PROJECT\_ID }}  
 working-directory: ./packages/admin  
 vercel-args: '--prod'  
  
 deploy-client:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
  
 - name: Deploy Client to Vercel  
 uses: amondnet/vercel-action@v25  
 with:  
 vercel-token: ${{ secrets.VERCEL\_TOKEN }}  
 github-token: ${{ secrets.GITHUB\_TOKEN }}  
 vercel-org-id: ${{ secrets.VERCEL\_ORG\_ID }}  
 vercel-project-id: ${{ secrets.VERCEL\_CLIENT\_PROJECT\_ID }}  
 working-directory: ./packages/client  
 vercel-args: '--prod'

1. Configure required secrets in GitHub:
2. VERCEL\_TOKEN: API token from Vercel
3. VERCEL\_ORG\_ID: Organization ID from Vercel
4. VERCEL\_ADMIN\_PROJECT\_ID: Project ID for admin panel
5. VERCEL\_CLIENT\_PROJECT\_ID: Project ID for client app

VERCEL\_TOKEN

VERCEL\_ORG\_ID

VERCEL\_ADMIN\_PROJECT\_ID

VERCEL\_CLIENT\_PROJECT\_ID

### Conclusion

Following this guide, you've successfully deployed the Kai application's frontend components to Vercel. The Next.js admin panel and Gatsby client application are now accessible via their respective domains and integrated with your backend services on Digital Ocean Kubernetes and Supabase.

Remember to regularly monitor your deployments for performance issues, error rates, and resource usage. As your application evolves, you may need to adjust build settings, environment variables, and integration points.

# Visual Reference Library

Source: readme/visual-reference-library.md

---

## Visual Reference Library

This document describes the Visual Reference Library feature, which provides a comprehensive system for managing visual references for material properties and characteristics.

### Overview

The Visual Reference Library is a centralized repository of visual references for material properties, finishes, textures, and other characteristics. It enables:

1. Visual Property Documentation: Documenting material properties with visual examples
2. AI Training Support: Providing labeled visual data for AI model training
3. User Education: Helping users understand material properties through visual examples
4. Consistent Terminology: Ensuring consistent visual representation of properties across the platform
5. Material-Specific Training: Training models for specific material types and properties
6. Property Recognition: Automatically recognizing properties from images

### Architecture

The Visual Reference Library feature consists of the following components:

#### Database Schema

* visual\_references: Stores metadata about visual references, including property associations
* visual\_reference\_images: Stores images associated with visual references
* visual\_reference\_tags: Stores tags for visual references to improve searchability
* visual\_reference\_annotations: Stores annotations for visual references to highlight specific features

#### API Endpoints

The following API endpoints are available for managing the Visual Reference Library:

##### Visual References

* GET /api/visual-references: Get visual references with filtering options
* GET /api/visual-references/:id: Get a visual reference by ID
* POST /api/visual-references: Create a new visual reference
* PUT /api/visual-references/:id: Update a visual reference
* DELETE /api/visual-references/:id: Delete a visual reference

GET /api/visual-references

GET /api/visual-references/:id

POST /api/visual-references

PUT /api/visual-references/:id

DELETE /api/visual-references/:id

##### Visual Reference Images

* GET /api/visual-references/:id/images: Get images for a visual reference
* POST /api/visual-references/:id/images: Add an image to a visual reference
* DELETE /api/visual-references/images/:id: Delete an image from a visual reference

GET /api/visual-references/:id/images

POST /api/visual-references/:id/images

DELETE /api/visual-references/images/:id

##### Visual Reference Annotations

* GET /api/visual-references/:id/annotations: Get annotations for a visual reference
* POST /api/visual-references/:id/annotations: Add an annotation to a visual reference
* PUT /api/visual-references/annotations/:id: Update an annotation
* DELETE /api/visual-references/annotations/:id: Delete an annotation

GET /api/visual-references/:id/annotations

POST /api/visual-references/:id/annotations

PUT /api/visual-references/annotations/:id

DELETE /api/visual-references/annotations/:id

##### Visual Reference Search

* POST /api/visual-references/search: Search for visual references
* GET /api/visual-references/property/:propertyName: Get visual references for a property
* GET /api/visual-references/property/:propertyName/value/:propertyValue: Get visual references for a property value

POST /api/visual-references/search

GET /api/visual-references/property/:propertyName

GET /api/visual-references/property/:propertyName/value/:propertyValue

#### Client Components

The following client components are available for working with the Visual Reference Library:

* VisualReferenceGallery: Displays a gallery of visual references
* VisualReferenceViewer: Displays a visual reference with annotations
* VisualReferenceSelector: Allows users to select visual references for a property
* VisualReferenceUploader: Allows users to upload new visual references

#### Admin Components

The following admin components are available for managing the Visual Reference Library:

* VisualReferenceManager: Allows administrators to manage visual references
* VisualReferenceAnnotator: Allows administrators to annotate visual references

### Usage

#### Adding a Visual Reference

Visual references can be added through the admin interface or by using the API.

// Example: Adding a new visual reference  
const newReference = {  
 title: 'Matte Finish on Porcelain Tile',  
 description: 'Example of a matte finish on porcelain tile, showing the non-reflective surface',  
 propertyName: 'finish',  
 propertyValue: 'matte',  
 materialType: 'tile',  
 tags: ['matte', 'porcelain', 'finish', 'non-reflective'],  
 source: 'internal',  
 isPublic: true  
};  
  
const response = await fetch('/api/visual-references', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(newReference)  
});  
  
const data = await response.json();  
const referenceId = data.reference.id;  
  
// Upload an image for the visual reference  
const formData = new FormData();  
formData.append('image', imageFile);  
formData.append('caption', 'Close-up of matte finish on porcelain tile');  
formData.append('isPrimary', 'true');  
  
await fetch(`/api/visual-references/${referenceId}/images`, {  
 method: 'POST',  
 headers: {  
 'Authorization': `Bearer ${token}`  
 },  
 body: formData  
});

// Example: Adding a new visual reference  
const newReference = {  
 title: 'Matte Finish on Porcelain Tile',  
 description: 'Example of a matte finish on porcelain tile, showing the non-reflective surface',  
 propertyName: 'finish',  
 propertyValue: 'matte',  
 materialType: 'tile',  
 tags: ['matte', 'porcelain', 'finish', 'non-reflective'],  
 source: 'internal',  
 isPublic: true  
};  
  
const response = await fetch('/api/visual-references', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(newReference)  
});  
  
const data = await response.json();  
const referenceId = data.reference.id;  
  
// Upload an image for the visual reference  
const formData = new FormData();  
formData.append('image', imageFile);  
formData.append('caption', 'Close-up of matte finish on porcelain tile');  
formData.append('isPrimary', 'true');  
  
await fetch(`/api/visual-references/${referenceId}/images`, {  
 method: 'POST',  
 headers: {  
 'Authorization': `Bearer ${token}`  
 },  
 body: formData  
});

#### Adding Annotations

Annotations can be added to highlight specific features in a visual reference.

// Example: Adding an annotation to a visual reference image  
const newAnnotation = {  
 imageId: 'image-id',  
 x: 150,  
 y: 200,  
 width: 100,  
 height: 50,  
 text: 'Note the non-reflective surface characteristic of matte finishes',  
 type: 'rectangle'  
};  
  
await fetch(`/api/visual-references/${referenceId}/annotations`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(newAnnotation)  
});

// Example: Adding an annotation to a visual reference image  
const newAnnotation = {  
 imageId: 'image-id',  
 x: 150,  
 y: 200,  
 width: 100,  
 height: 50,  
 text: 'Note the non-reflective surface characteristic of matte finishes',  
 type: 'rectangle'  
};  
  
await fetch(`/api/visual-references/${referenceId}/annotations`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(newAnnotation)  
});

#### Displaying Visual References

The VisualReferenceGallery component can be used to display visual references for a property.

VisualReferenceGallery

import VisualReferenceGallery from '../components/visual-reference/VisualReferenceGallery';  
  
// Example: Displaying visual references for a property  
<VisualReferenceGallery  
 propertyName="finish"  
 propertyValue="matte"  
 materialType="tile"  
 maxItems={6}  
 showAnnotations={true}  
/>

import VisualReferenceGallery from '../components/visual-reference/VisualReferenceGallery';  
  
// Example: Displaying visual references for a property  
<VisualReferenceGallery  
 propertyName="finish"  
 propertyValue="matte"  
 materialType="tile"  
 maxItems={6}  
 showAnnotations={true}  
/>

#### Viewing a Visual Reference

The VisualReferenceViewer component can be used to display a specific visual reference with annotations.

VisualReferenceViewer

import VisualReferenceViewer from '../components/visual-reference/VisualReferenceViewer';  
  
// Example: Displaying a specific visual reference  
<VisualReferenceViewer  
 referenceId="reference-id"  
 showAnnotations={true}  
 enableZoom={true}  
/>

import VisualReferenceViewer from '../components/visual-reference/VisualReferenceViewer';  
  
// Example: Displaying a specific visual reference  
<VisualReferenceViewer  
 referenceId="reference-id"  
 showAnnotations={true}  
 enableZoom={true}  
/>

### Integration with Other Features

#### Material Metadata Panel Integration

The Visual Reference Library is integrated with the Material Metadata Panel to display visual references for properties.

// In MaterialMetadataPanel.tsx  
  
// Display visual references for a property  
const renderPropertyField = (property: string, value: string) => {  
 return (  
 <Box>  
 <TextField  
 label={property}  
 value={value}  
 onChange={(e) => handlePropertyChange(property, e.target.value)}  
 />  
 <IconButton  
 onClick={() => setShowVisualReferences({ property, value })}  
 title="View visual references"  
 >  
 <ImageIcon />  
 </IconButton>  
  
 {showVisualReferences?.property === property && (  
 <Dialog open={true} onClose={() => setShowVisualReferences(null)}>  
 <DialogTitle>Visual References for {property}: {value}</DialogTitle>  
 <DialogContent>  
 <VisualReferenceGallery  
 propertyName={property}  
 propertyValue={value}  
 materialType={materialType}  
 showAnnotations={true}  
 />  
 </DialogContent>  
 </Dialog>  
 )}  
 </Box>  
 );  
};

// In MaterialMetadataPanel.tsx  
  
// Display visual references for a property  
const renderPropertyField = (property: string, value: string) => {  
 return (  
 <Box>  
 <TextField  
 label={property}  
 value={value}  
 onChange={(e) => handlePropertyChange(property, e.target.value)}  
 />  
 <IconButton  
 onClick={() => setShowVisualReferences({ property, value })}  
 title="View visual references"  
 >  
 <ImageIcon />  
 </IconButton>  
  
 {showVisualReferences?.property === property && (  
 <Dialog open={true} onClose={() => setShowVisualReferences(null)}>  
 <DialogTitle>Visual References for {property}: {value}</DialogTitle>  
 <DialogContent>  
 <VisualReferenceGallery  
 propertyName={property}  
 propertyValue={value}  
 materialType={materialType}  
 showAnnotations={true}  
 />  
 </DialogContent>  
 </Dialog>  
 )}  
 </Box>  
 );  
};

#### Property Relationship Graph Integration

The Visual Reference Library can be integrated with the Property Relationship Graph to display visual examples of property relationships.

#### AI Training Integration

The Visual Reference Library provides labeled visual data for AI model training, particularly for visual property identification.

### Best Practices

#### Adding Visual References

When adding visual references, follow these best practices:

1. Clear Images: Use high-quality, clear images that clearly demonstrate the property
2. Multiple Angles: Include multiple angles or examples when relevant
3. Proper Lighting: Ensure proper lighting to accurately represent the property
4. Consistent Scale: Include scale references when size or dimension is important
5. Detailed Annotations: Add annotations to highlight specific features
6. Comprehensive Metadata: Include detailed metadata to improve searchability

#### Organizing Visual References

To maintain an organized Visual Reference Library:

1. Consistent Naming: Use consistent naming conventions for properties and values
2. Comprehensive Tagging: Add relevant tags to improve searchability
3. Proper Categorization: Categorize references by material type and property
4. Regular Updates: Regularly update references to ensure they remain accurate and relevant

### Benefits

The Visual Reference Library provides several benefits:

1. Improved Understanding: Users can better understand material properties through visual examples
2. Enhanced AI Training: AI models can be trained with labeled visual data
3. Consistent Terminology: Visual references ensure consistent understanding of property terms
4. Better Documentation: Properties are documented with visual examples for clarity

### Property-Specific Training Implementation

The Visual Reference Library now includes a comprehensive property-specific training implementation that enables training models to recognize specific material properties from visual references.

#### Property-Specific Training Module

The property-specific training module provides functions for training models to recognize specific material properties from visual references.

##### Features

* Material-specific model training
* Support for different model types (classification, regression, detection)
* Transfer learning with pre-trained models
* Data augmentation for improved model performance
* Comprehensive training metrics and evaluation

##### Training Process

1. Dataset Preparation: Organize visual references by property value
2. Model Selection: Choose appropriate model architecture based on property type
3. Training: Train model with transfer learning and data augmentation
4. Evaluation: Evaluate model performance with validation data
5. Deployment: Save model for use in prediction

#### Dataset Structure

The Visual Reference Library organizes visual references in a structured way to facilitate training and prediction.

##### Organization

* Material Type: References are organized by material type (tile, wood, lighting, etc.)
* Property Name: Within each material type, references are organized by property name (color, texture, finish, etc.)
* Property Value: Within each property, references are organized by property value (red, glossy, matte, etc.)

#### New API Endpoints

The Visual Reference Library now provides the following additional API endpoints:

##### Get Properties with Visual References

GET /api/ai/visual-reference/properties

GET /api/ai/visual-reference/properties

Returns a list of properties with visual references, optionally filtered by material type.

##### Train Model for Property

POST /api/ai/visual-reference/train

POST /api/ai/visual-reference/train

Trains a model for a specific property and material type.

##### Predict Property Value

POST /api/ai/visual-reference/predict

POST /api/ai/visual-reference/predict

Predicts a property value from an image using a trained model.

#### Database Schema

The Visual Reference Library now uses the following additional database schema:

##### VisualPropertyReference

* id: UUID primary key
* propertyName: Name of the property
* materialType: Type of material
* displayName: Display name for the property
* description: Description of the property
* previewImage: Path to a preview image
* modelPath: Path to the trained model
* modelAccuracy: Accuracy of the trained model
* lastTrainedAt: When the model was last trained
* createdAt: When the reference was created
* updatedAt: When the reference was last updated

id

propertyName

materialType

displayName

description

previewImage

modelPath

modelAccuracy

lastTrainedAt

createdAt

updatedAt

##### VisualPropertyReferenceItem

* id: UUID primary key
* referenceId: Reference to VisualPropertyReference
* imagePath: Path to the image file
* propertyValue: Value of the property
* metadata: Additional metadata (JSON)
* createdAt: When the item was created
* updatedAt: When the item was last updated

id

referenceId

imagePath

propertyValue

metadata

createdAt

updatedAt

### Crawler Integration

The Visual Reference Library now integrates with the platform's existing crawler services to automatically collect visual references from websites.

#### Integration Features

* Import from Crawler Jobs: Import images from completed crawler jobs as visual references
* Auto-Classification: Automatically classify imported images based on their visual characteristics
* Selective Import: Control which images to import and how many
* Metadata Preservation: Preserve source information as metadata for each reference

#### Import Process

1. Select Crawler Job: Choose a completed crawler job that contains images
2. Configure Import Options: Specify property name, material type, and classification options
3. Import Images: The system downloads images from the crawler results and imports them as visual references
4. Auto-Classification: If enabled, the system analyzes each image to determine its properties
5. Reference Creation: The system creates visual reference items with appropriate metadata

#### Benefits

* Efficient Data Collection: Quickly build comprehensive visual reference libraries
* Consistent Quality: Ensure consistent quality and format of visual references
* Automated Classification: Reduce manual tagging effort with auto-classification
* Traceability: Maintain links to source data for reference

### Advanced Features

The Visual Reference Library now includes several advanced features:

#### Model Comparison

The Model Comparison feature allows administrators to:

* Compare different models for the same property
* Visualize performance metrics side by side
* Identify disagreement examples between models
* Get recommendations for the best model to use
* Create and manage multiple model versions

#### Active Learning

The Active Learning system improves models over time by:

* Identifying uncertain samples where the model lacks confidence
* Collecting human feedback on these samples
* Incorporating this feedback into the training data
* Retraining models with the enhanced dataset
* Measuring improvement metrics

#### Cross-Property Models

Cross-Property Models can recognize multiple properties at once:

* Train a single model to detect multiple properties simultaneously
* More efficient than using separate models for each property
* Share learned features across related properties
* Reduce overall computational requirements
* Provide consistent predictions across properties

#### Specialized Crawlers

Specialized Crawlers are designed for specific property types:

* Configure crawlers with property-specific extraction rules
* Target websites that specialize in certain material properties
* Extract structured data along with images
* Automatically classify and categorize crawled content
* Seamlessly import results into the Visual Reference Library

### Future Enhancements

Potential future enhancements to the Visual Reference Library:

1. 3D References: Add support for 3D models as visual references
2. Video References: Add support for video demonstrations
3. AR Integration: Integrate with AR to visualize properties in real-world contexts
4. User Contributions: Allow users to contribute visual references with moderation
5. AI-Generated Annotations: Use AI to automatically generate annotations
6. Cross-Material Transfer Learning: Use knowledge from one material type to improve models for other types
7. User Feedback Loop: Incorporate user feedback to improve model accuracy
8. Federated Learning: Enable distributed training across multiple instances

### Conclusion

The Visual Reference Library provides a comprehensive system for managing visual references for material properties and characteristics. By providing visual examples, it improves understanding, enhances AI training, ensures consistent terminology, and provides better documentation for material properties. The new property-specific training implementation enables automatic recognition of properties from images, further enhancing the platform's capabilities.

# Web Crawling

Source: readme/web-crawling.md

---

## Web Crawling System

The Web Crawling System is a specialized component of Kai that enables automated collection of material data from websites and online catalogs. This document details the system's capabilities, architecture, and implementation.

### Features

#### Multi-Provider Crawler Support

The system integrates with multiple specialized crawling services:

1. Provider Ecosystem
2. JinaAI: Advanced ML-powered crawler with intelligent content extraction
3. Firecrawl: High-performance crawler optimized for scale and speed
4. Internal: Built-in crawling capabilities for specialized scenarios
5. Custom: Extensible framework for additional providers
6. Provider Management
7. Dynamic provider selection based on content type and requirements
8. Credential management for secure API key storage
9. Provider-specific configuration options
10. Automatic fallback mechanisms
11. Performance monitoring across providers
12. Unified Interface
13. Common configuration schema across providers
14. Standardized job lifecycle management
15. Consistent result formats for downstream processing
16. Provider-agnostic queuing system
17. Centralized monitoring and reporting

Custom: Extensible framework for additional providers

Provider Management

Performance monitoring across providers

Unified Interface

#### Intelligent Crawling Capabilities

The system offers advanced crawling functionality:

1. Crawl Configuration
2. URL targeting with domain boundary control
3. Crawl depth and breadth customization
4. Rate limiting and politeness settings
5. Content type filtering
6. Regular expression-based URL matching
7. Content Extraction
8. Intelligent material identification
9. Structured data extraction from HTML
10. Image and specification association
11. Metadata capture from context
12. Taxonomy mapping and categorization
13. Selective Harvesting
14. Material-focused extraction rules
15. Product page identification
16. Catalog structure understanding
17. Relevant content prioritization
18. Noise and decoration filtering

Regular expression-based URL matching

Content Extraction

Taxonomy mapping and categorization

Selective Harvesting

#### Queue and Job Management

The system includes comprehensive job management:

1. Job Lifecycle
2. Configurable job creation with priorities
3. Status tracking throughout execution
4. Real-time progress updates
5. Error handling and recovery
6. Completion notification
7. Queue Optimization
8. Priority-based processing
9. Concurrent job management
10. Resource allocation based on job type
11. Scheduling for off-peak execution
12. Rate limiting to respect target sites
13. Administrative Controls
14. Job inspection and monitoring
15. Manual intervention capabilities
16. Retry mechanisms for failed jobs
17. Queue clearing and management
18. Result inspection and validation

Completion notification

Queue Optimization

Rate limiting to respect target sites

Administrative Controls

#### Training Data Integration

The system prepares crawled data for ML training:

1. Data Transformation
2. Structured formatting for training pipeline
3. Automatic labeling based on source context
4. Image preprocessing for ML readiness
5. Metadata enrichment for context
6. Quality filtering for training suitability
7. Dataset Management
8. Versioned dataset creation
9. Class organization and balancing
10. Metadata preservation for provenance
11. Validation against training requirements
12. Integration with existing datasets
13. Training Pipeline Integration
14. Optional automatic training initiation
15. Training parameter inference from content
16. Crawl-to-model automated workflow
17. Performance metrics collection
18. Continuous improvement through crawl-train cycles

Quality filtering for training suitability

Dataset Management

Integration with existing datasets

Training Pipeline Integration

### Technical Implementation

#### Crawler Service Factory

The system uses a factory pattern for crawler service instantiation:

/\*\*  
 \* Factory for creating crawler services based on configuration  
 \*/  
class CrawlerServiceFactory {  
 /\*\*  
 \* Get the list of available crawler providers  
 \* @returns Array of supported providers  
 \*/  
 public getAvailableProviders(): CrawlerProvider[] {  
 return ['firecrawl', 'jina', 'internal', 'custom'];  
 }  
  
 /\*\*  
 \* Check if a provider is supported  
 \* @param provider Provider to check  
 \* @returns Boolean indicating support status  
 \*/  
 public isProviderSupported(provider: string): boolean {  
 return this.getAvailableProviders().includes(provider as CrawlerProvider);  
 }  
  
 /\*\*  
 \* Get a crawler service instance for a specific provider  
 \* @param provider The crawler provider to use  
 \* @returns Promise resolving to a crawler service  
 \*/  
 public async getService(provider: CrawlerProvider): Promise<CrawlerService> {  
 // Get credentials for the provider  
 const credentials = credentialsManager.getCredentials(provider);  
  
 switch (provider) {  
 case 'firecrawl':  
 return new FirecrawlService(credentials);  
 case 'jina':  
 return new JinaService(credentials);  
 case 'internal':  
 return new InternalCrawlerService();  
 case 'custom':  
 return new CustomCrawlerService(credentials);  
 default:  
 throw new Error(`Unsupported crawler provider: ${provider}`);  
 }  
 }  
  
 /\*\*  
 \* Get a crawler service based on a configuration object  
 \* @param config Crawler configuration  
 \* @returns Promise resolving to an appropriate crawler service  
 \*/  
 public async getServiceForConfig(config: CrawlerConfig): Promise<CrawlerService> {  
 // Validate config first  
 validateCrawlerConfig(config);  
  
 // Get appropriate service based on provider  
 return this.getService(config.provider as CrawlerProvider);  
 }  
}  
  
// Export singleton instance  
export const crawlerServiceFactory = new CrawlerServiceFactory();

/\*\*  
 \* Factory for creating crawler services based on configuration  
 \*/  
class CrawlerServiceFactory {  
 /\*\*  
 \* Get the list of available crawler providers  
 \* @returns Array of supported providers  
 \*/  
 public getAvailableProviders(): CrawlerProvider[] {  
 return ['firecrawl', 'jina', 'internal', 'custom'];  
 }  
  
 /\*\*  
 \* Check if a provider is supported  
 \* @param provider Provider to check  
 \* @returns Boolean indicating support status  
 \*/  
 public isProviderSupported(provider: string): boolean {  
 return this.getAvailableProviders().includes(provider as CrawlerProvider);  
 }  
  
 /\*\*  
 \* Get a crawler service instance for a specific provider  
 \* @param provider The crawler provider to use  
 \* @returns Promise resolving to a crawler service  
 \*/  
 public async getService(provider: CrawlerProvider): Promise<CrawlerService> {  
 // Get credentials for the provider  
 const credentials = credentialsManager.getCredentials(provider);  
  
 switch (provider) {  
 case 'firecrawl':  
 return new FirecrawlService(credentials);  
 case 'jina':  
 return new JinaService(credentials);  
 case 'internal':  
 return new InternalCrawlerService();  
 case 'custom':  
 return new CustomCrawlerService(credentials);  
 default:  
 throw new Error(`Unsupported crawler provider: ${provider}`);  
 }  
 }  
  
 /\*\*  
 \* Get a crawler service based on a configuration object  
 \* @param config Crawler configuration  
 \* @returns Promise resolving to an appropriate crawler service  
 \*/  
 public async getServiceForConfig(config: CrawlerConfig): Promise<CrawlerService> {  
 // Validate config first  
 validateCrawlerConfig(config);  
  
 // Get appropriate service based on provider  
 return this.getService(config.provider as CrawlerProvider);  
 }  
}  
  
// Export singleton instance  
export const crawlerServiceFactory = new CrawlerServiceFactory();

#### Crawler Service Interface

The system defines a common interface for all crawler services:

/\*\*  
 \* Interface for crawler services  
 \*/  
interface CrawlerService {  
 /\*\*  
 \* Start a crawl job  
 \* @param config Crawler configuration  
 \* @param options Job options  
 \* @returns Promise with job details  
 \*/  
 startJob(config: CrawlerConfig, options?: JobOptions): Promise<{ id: string }>;  
  
 /\*\*  
 \* Get status of a crawl job  
 \* @param jobId External job ID  
 \* @returns Promise with job status  
 \*/  
 getJobStatus(jobId: string): Promise<{ status: CrawlerJobStatus }>;  
  
 /\*\*  
 \* Cancel a crawl job  
 \* @param jobId External job ID  
 \* @returns Promise with cancellation status  
 \*/  
 cancelJob(jobId: string): Promise<{ success: boolean }>;  
  
 /\*\*  
 \* Get results from a completed crawl job  
 \* @param jobId External job ID  
 \* @returns Promise with crawl results  
 \*/  
 getResults(jobId: string): Promise<CrawlResult>;  
  
 /\*\*  
 \* Transform crawl results for training  
 \* @param results Crawl results  
 \* @param options Transformation options  
 \* @returns Promise with transformation results  
 \*/  
 transformDataForTraining(  
 results: CrawlResult,   
 options: TransformOptions  
 ): Promise<{ path: string }>;  
}

/\*\*  
 \* Interface for crawler services  
 \*/  
interface CrawlerService {  
 /\*\*  
 \* Start a crawl job  
 \* @param config Crawler configuration  
 \* @param options Job options  
 \* @returns Promise with job details  
 \*/  
 startJob(config: CrawlerConfig, options?: JobOptions): Promise<{ id: string }>;  
  
 /\*\*  
 \* Get status of a crawl job  
 \* @param jobId External job ID  
 \* @returns Promise with job status  
 \*/  
 getJobStatus(jobId: string): Promise<{ status: CrawlerJobStatus }>;  
  
 /\*\*  
 \* Cancel a crawl job  
 \* @param jobId External job ID  
 \* @returns Promise with cancellation status  
 \*/  
 cancelJob(jobId: string): Promise<{ success: boolean }>;  
  
 /\*\*  
 \* Get results from a completed crawl job  
 \* @param jobId External job ID  
 \* @returns Promise with crawl results  
 \*/  
 getResults(jobId: string): Promise<CrawlResult>;  
  
 /\*\*  
 \* Transform crawl results for training  
 \* @param results Crawl results  
 \* @param options Transformation options  
 \* @returns Promise with transformation results  
 \*/  
 transformDataForTraining(  
 results: CrawlResult,   
 options: TransformOptions  
 ): Promise<{ path: string }>;  
}

#### Crawler Queue Implementation

The system provides a queue for managing crawler jobs:

/\*\*  
 \* Manages queue of crawler jobs  
 \*/  
class CrawlerQueue {  
 private jobs: Map<string, CrawlerJob> = new Map();  
 private processing: Set<string> = new Set();  
 private config: QueueConfig;  
 private adapter: QueueAdapter;  
  
 /\*\*  
 \* Initialize crawler queue  
 \* @param config Queue configuration  
 \*/  
 constructor(config: QueueConfig) {  
 this.config = {  
 maxConcurrentJobs: 5,  
 retryLimit: 3,  
 ...config  
 };  
  
 this.adapter = createCrawlerQueueAdapter();  
 this.initMessageHandlers();  
 }  
  
 /\*\*  
 \* Add a job to the queue  
 \* @param config Crawler configuration  
 \* @param options Job options  
 \* @returns Promise with job ID  
 \*/  
 public async addJob(  
 config: CrawlerConfig,   
 options: { priority?: 'high' | 'normal' | 'low' } = {}  
 ): Promise<string> {  
 // Create a new job with unique ID  
 const jobId = uuid();  
 const priority = this.getPriorityValue(options.priority || 'normal');  
  
 const job: CrawlerJob = {  
 id: jobId,  
 config,  
 provider: config.provider,  
 status: 'pending',  
 priority,  
 createdAt: Date.now(),  
 progress: 0  
 };  
  
 // Store job in queue  
 this.jobs.set(jobId, job);  
  
 // Notify about new job  
 await this.adapter.publishEvent('job\_added', {   
 jobId,   
 provider: config.provider,  
 priority  
 });  
  
 // Start processing loop if not already running  
 this.startProcessing();  
  
 return jobId;  
 }  
  
 /\*\*  
 \* Get a job by ID  
 \* @param jobId Job ID  
 \* @returns Crawler job or undefined if not found  
 \*/  
 public getJob(jobId: string): CrawlerJob | undefined {  
 return this.jobs.get(jobId);  
 }  
  
 /\*\*  
 \* Get all jobs  
 \* @returns Array of all crawler jobs  
 \*/  
 public getAllJobs(): Map<string, CrawlerJob> {  
 return this.jobs;  
 }  
  
 /\*\*  
 \* Remove a job from the queue  
 \* @param jobId Job ID  
 \* @returns Boolean indicating success  
 \*/  
 public async removeJob(jobId: string): Promise<boolean> {  
 // Can't remove jobs that are currently processing  
 if (this.processing.has(jobId)) {  
 return false;  
 }  
  
 const removed = this.jobs.delete(jobId);  
  
 if (removed) {  
 await this.adapter.publishEvent('job\_removed', { jobId });  
 }  
  
 return removed;  
 }  
  
 // Additional implementation details...  
}  
  
// Export singleton instance  
export const crawlerQueue = new CrawlerQueue({  
 maxConcurrentJobs: 5,  
 retryLimit: 3  
});

/\*\*  
 \* Manages queue of crawler jobs  
 \*/  
class CrawlerQueue {  
 private jobs: Map<string, CrawlerJob> = new Map();  
 private processing: Set<string> = new Set();  
 private config: QueueConfig;  
 private adapter: QueueAdapter;  
  
 /\*\*  
 \* Initialize crawler queue  
 \* @param config Queue configuration  
 \*/  
 constructor(config: QueueConfig) {  
 this.config = {  
 maxConcurrentJobs: 5,  
 retryLimit: 3,  
 ...config  
 };  
  
 this.adapter = createCrawlerQueueAdapter();  
 this.initMessageHandlers();  
 }  
  
 /\*\*  
 \* Add a job to the queue  
 \* @param config Crawler configuration  
 \* @param options Job options  
 \* @returns Promise with job ID  
 \*/  
 public async addJob(  
 config: CrawlerConfig,   
 options: { priority?: 'high' | 'normal' | 'low' } = {}  
 ): Promise<string> {  
 // Create a new job with unique ID  
 const jobId = uuid();  
 const priority = this.getPriorityValue(options.priority || 'normal');  
  
 const job: CrawlerJob = {  
 id: jobId,  
 config,  
 provider: config.provider,  
 status: 'pending',  
 priority,  
 createdAt: Date.now(),  
 progress: 0  
 };  
  
 // Store job in queue  
 this.jobs.set(jobId, job);  
  
 // Notify about new job  
 await this.adapter.publishEvent('job\_added', {   
 jobId,   
 provider: config.provider,  
 priority  
 });  
  
 // Start processing loop if not already running  
 this.startProcessing();  
  
 return jobId;  
 }  
  
 /\*\*  
 \* Get a job by ID  
 \* @param jobId Job ID  
 \* @returns Crawler job or undefined if not found  
 \*/  
 public getJob(jobId: string): CrawlerJob | undefined {  
 return this.jobs.get(jobId);  
 }  
  
 /\*\*  
 \* Get all jobs  
 \* @returns Array of all crawler jobs  
 \*/  
 public getAllJobs(): Map<string, CrawlerJob> {  
 return this.jobs;  
 }  
  
 /\*\*  
 \* Remove a job from the queue  
 \* @param jobId Job ID  
 \* @returns Boolean indicating success  
 \*/  
 public async removeJob(jobId: string): Promise<boolean> {  
 // Can't remove jobs that are currently processing  
 if (this.processing.has(jobId)) {  
 return false;  
 }  
  
 const removed = this.jobs.delete(jobId);  
  
 if (removed) {  
 await this.adapter.publishEvent('job\_removed', { jobId });  
 }  
  
 return removed;  
 }  
  
 // Additional implementation details...  
}  
  
// Export singleton instance  
export const crawlerQueue = new CrawlerQueue({  
 maxConcurrentJobs: 5,  
 retryLimit: 3  
});

#### Crawler Adapter for Training

The system includes a Python adapter for preparing crawler data for training:

class CrawlerDataAdapter:  
 """Class for adapting crawler data for material recognition training"""  
  
 def \_\_init\_\_(self, data\_dir, output\_dir):  
 """  
 Initialize the crawler data adapter  
  
 Args:  
 data\_dir: Directory containing crawler data  
 output\_dir: Directory to save adapted data for training  
 """  
 self.data\_dir = data\_dir  
 self.output\_dir = output\_dir  
 os.makedirs(output\_dir, exist\_ok=True)  
  
 def prepare\_data(self, manifest\_path, min\_images\_per\_class=10):  
 """  
 Prepare crawler data for training  
  
 Args:  
 manifest\_path: Path to the crawler dataset manifest file  
 min\_images\_per\_class: Minimum number of images required per class  
  
 Returns:  
 Dictionary with preparation results  
 """  
 if not os.path.exists(manifest\_path):  
 return {  
 "status": "error",  
 "message": f"Manifest file not found: {manifest\_path}",  
 "dataset\_path": None  
 }  
  
 # Create dataset directory  
 timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")  
 dataset\_name = f"crawler\_dataset\_{timestamp}"  
 dataset\_dir = os.path.join(self.output\_dir, dataset\_name)  
 os.makedirs(dataset\_dir, exist\_ok=True)  
  
 # Load manifest  
 with open(manifest\_path, 'r') as f:  
 manifest = json.load(f)  
  
 # Process and organize images  
 # Implementation details...  
  
 # Create metadata file  
 metadata = {  
 "dataset\_name": dataset\_name,  
 "created\_at": datetime.now().isoformat(),  
 "source": manifest.get("source", "crawler"),  
 "job\_id": manifest.get("job\_id"),  
 "image\_count": total\_images,  
 "class\_count": len(class\_counts),  
 "class\_distribution": class\_counts,  
 "download\_errors": download\_errors,  
 "data\_provenance": "crawler",  
 "min\_images\_per\_class": min\_images\_per\_class,  
 "manifest\_path": manifest\_path  
 }  
  
 # Save metadata  
 metadata\_path = os.path.join(dataset\_dir, "metadata.json")  
 with open(metadata\_path, 'w') as f:  
 json.dump(metadata, f, indent=2)  
  
 return {  
 "status": "success",  
 "message": f"Created dataset with {total\_images} images across {len(class\_counts)} classes",  
 "dataset\_path": dataset\_dir,  
 "metadata": metadata  
 }  
  
 # Additional methods for validating and processing crawler data...

class CrawlerDataAdapter:  
 """Class for adapting crawler data for material recognition training"""  
  
 def \_\_init\_\_(self, data\_dir, output\_dir):  
 """  
 Initialize the crawler data adapter  
  
 Args:  
 data\_dir: Directory containing crawler data  
 output\_dir: Directory to save adapted data for training  
 """  
 self.data\_dir = data\_dir  
 self.output\_dir = output\_dir  
 os.makedirs(output\_dir, exist\_ok=True)  
  
 def prepare\_data(self, manifest\_path, min\_images\_per\_class=10):  
 """  
 Prepare crawler data for training  
  
 Args:  
 manifest\_path: Path to the crawler dataset manifest file  
 min\_images\_per\_class: Minimum number of images required per class  
  
 Returns:  
 Dictionary with preparation results  
 """  
 if not os.path.exists(manifest\_path):  
 return {  
 "status": "error",  
 "message": f"Manifest file not found: {manifest\_path}",  
 "dataset\_path": None  
 }  
  
 # Create dataset directory  
 timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")  
 dataset\_name = f"crawler\_dataset\_{timestamp}"  
 dataset\_dir = os.path.join(self.output\_dir, dataset\_name)  
 os.makedirs(dataset\_dir, exist\_ok=True)  
  
 # Load manifest  
 with open(manifest\_path, 'r') as f:  
 manifest = json.load(f)  
  
 # Process and organize images  
 # Implementation details...  
  
 # Create metadata file  
 metadata = {  
 "dataset\_name": dataset\_name,  
 "created\_at": datetime.now().isoformat(),  
 "source": manifest.get("source", "crawler"),  
 "job\_id": manifest.get("job\_id"),  
 "image\_count": total\_images,  
 "class\_count": len(class\_counts),  
 "class\_distribution": class\_counts,  
 "download\_errors": download\_errors,  
 "data\_provenance": "crawler",  
 "min\_images\_per\_class": min\_images\_per\_class,  
 "manifest\_path": manifest\_path  
 }  
  
 # Save metadata  
 metadata\_path = os.path.join(dataset\_dir, "metadata.json")  
 with open(metadata\_path, 'w') as f:  
 json.dump(metadata, f, indent=2)  
  
 return {  
 "status": "success",  
 "message": f"Created dataset with {total\_images} images across {len(class\_counts)} classes",  
 "dataset\_path": dataset\_dir,  
 "metadata": metadata  
 }  
  
 # Additional methods for validating and processing crawler data...

#### Credentials Management

The system securely manages crawler provider credentials:

/\*\*  
 \* Manages credentials for crawler services  
 \*/  
class CredentialsManager {  
 private credentials: Map<string, string> = new Map();  
 private encryptionKey: string;  
  
 constructor() {  
 // Initialize encryption key (in production, use secure methods)  
 this.encryptionKey = process.env.CREDENTIALS\_ENCRYPTION\_KEY || 'default-key';  
 this.loadCredentials();  
 }  
  
 /\*\*  
 \* Set credentials for a provider  
 \* @param provider Provider name  
 \* @param apiKey API key for the provider  
 \* @returns Success status  
 \*/  
 public setCredentials(provider: string, apiKey: string): boolean {  
 try {  
 // Encrypt the API key before storing  
 const encryptedKey = this.encrypt(apiKey);  
  
 // Store in memory  
 this.credentials.set(provider, encryptedKey);  
  
 // Persist to storage  
 this.saveCredentials();  
  
 return true;  
 } catch (error) {  
 logger.error(`Failed to set credentials for ${provider}: ${error}`);  
 return false;  
 }  
 }  
  
 /\*\*  
 \* Get credentials for a provider  
 \* @param provider Provider name  
 \* @returns Decrypted API key or null if not found  
 \*/  
 public getCredentials(provider: string): string | null {  
 const encryptedKey = this.credentials.get(provider);  
  
 if (!encryptedKey) {  
 return null;  
 }  
  
 try {  
 // Decrypt the stored key  
 return this.decrypt(encryptedKey);  
 } catch (error) {  
 logger.error(`Failed to retrieve credentials for ${provider}: ${error}`);  
 return null;  
 }  
 }  
  
 /\*\*  
 \* Check if credentials exist for a provider  
 \* @param provider Provider name  
 \* @returns Boolean indicating if credentials exist  
 \*/  
 public hasCredentials(provider: string): boolean {  
 return this.credentials.has(provider);  
 }  
  
 /\*\*  
 \* Remove credentials for a provider  
 \* @param provider Provider name  
 \* @returns Success status  
 \*/  
 public removeCredentials(provider: string): boolean {  
 if (!this.credentials.has(provider)) {  
 return false;  
 }  
  
 this.credentials.delete(provider);  
 this.saveCredentials();  
  
 return true;  
 }  
  
 // Private helper methods for loading, saving, encrypting, and decrypting  
 // Implementation details...  
}  
  
// Export singleton instance  
export const credentialsManager = new CredentialsManager();

/\*\*  
 \* Manages credentials for crawler services  
 \*/  
class CredentialsManager {  
 private credentials: Map<string, string> = new Map();  
 private encryptionKey: string;  
  
 constructor() {  
 // Initialize encryption key (in production, use secure methods)  
 this.encryptionKey = process.env.CREDENTIALS\_ENCRYPTION\_KEY || 'default-key';  
 this.loadCredentials();  
 }  
  
 /\*\*  
 \* Set credentials for a provider  
 \* @param provider Provider name  
 \* @param apiKey API key for the provider  
 \* @returns Success status  
 \*/  
 public setCredentials(provider: string, apiKey: string): boolean {  
 try {  
 // Encrypt the API key before storing  
 const encryptedKey = this.encrypt(apiKey);  
  
 // Store in memory  
 this.credentials.set(provider, encryptedKey);  
  
 // Persist to storage  
 this.saveCredentials();  
  
 return true;  
 } catch (error) {  
 logger.error(`Failed to set credentials for ${provider}: ${error}`);  
 return false;  
 }  
 }  
  
 /\*\*  
 \* Get credentials for a provider  
 \* @param provider Provider name  
 \* @returns Decrypted API key or null if not found  
 \*/  
 public getCredentials(provider: string): string | null {  
 const encryptedKey = this.credentials.get(provider);  
  
 if (!encryptedKey) {  
 return null;  
 }  
  
 try {  
 // Decrypt the stored key  
 return this.decrypt(encryptedKey);  
 } catch (error) {  
 logger.error(`Failed to retrieve credentials for ${provider}: ${error}`);  
 return null;  
 }  
 }  
  
 /\*\*  
 \* Check if credentials exist for a provider  
 \* @param provider Provider name  
 \* @returns Boolean indicating if credentials exist  
 \*/  
 public hasCredentials(provider: string): boolean {  
 return this.credentials.has(provider);  
 }  
  
 /\*\*  
 \* Remove credentials for a provider  
 \* @param provider Provider name  
 \* @returns Success status  
 \*/  
 public removeCredentials(provider: string): boolean {  
 if (!this.credentials.has(provider)) {  
 return false;  
 }  
  
 this.credentials.delete(provider);  
 this.saveCredentials();  
  
 return true;  
 }  
  
 // Private helper methods for loading, saving, encrypting, and decrypting  
 // Implementation details...  
}  
  
// Export singleton instance  
export const credentialsManager = new CredentialsManager();

### Integration with Other Systems

#### PDF Processing Pipeline

The web crawling system integrates with the PDF processing system:

1. Automatic PDF Handling
2. Detection of PDF catalogs during crawling
3. Automatic submission to PDF processing queue
4. Association of crawled context with PDFs
5. Correlation of PDF extraction results with web data
6. Combined knowledge extraction from both sources
7. Cross-Source Validation
8. Web data used to validate PDF extraction
9. PDF content used to enhance web data extraction
10. Confidence scoring based on multi-source correlation
11. Conflict resolution for disparate information
12. Complete material profiles from combined sources
13. Workflow Orchestration
14. Sequential processing from web to PDF
15. Parallel extraction for efficiency
16. Coordinated job prioritization
17. Status monitoring across both systems
18. Unified result presentation

Combined knowledge extraction from both sources

Cross-Source Validation

Complete material profiles from combined sources

Workflow Orchestration

#### Training Pipeline

The web crawling system feeds the training pipeline:

1. Dataset Creation
2. Automatic dataset generation from crawler results
3. Class organization based on material categories
4. Image processing for training readiness
5. Metadata annotation for context
6. Quality filtering for training suitability
7. Training Integration
8. Optional automatic training initiation
9. Configurable training parameters
10. Progress monitoring and reporting
11. Model versioning tied to crawler sources
12. Performance evaluation against source quality
13. Continuous Improvement
14. Feedback loop from model performance to crawling
15. Targeted recrawling for underperforming categories
16. Quality metrics influencing crawler configurations
17. Incremental dataset updates
18. Validation against existing material data

Quality filtering for training suitability

Training Integration

Performance evaluation against source quality

Continuous Improvement

#### Knowledge Base

The web crawling system enriches the knowledge base:

1. Knowledge Extraction
2. Structured data extraction from web sources
3. Entity recognition and classification
4. Relationship identification between materials
5. Attribute extraction for material properties
6. Hierarchical categorization
7. Knowledge Integration
8. Merging with existing knowledge entities
9. Conflict resolution for contradictory information
10. Confidence scoring for source reliability
11. Provenance tracking for attribution
12. Version control for knowledge evolution
13. Search Enhancement
14. Web-sourced data improving search relevance
15. Extracted terminology enhancing query understanding
16. Additional context for disambiguation
17. Rich metadata for filtering
18. Comprehensive material profiles for accurate matching

Hierarchical categorization

Knowledge Integration

Version control for knowledge evolution

Search Enhancement

### API Usage Examples

#### Basic Crawler Configuration

import { validateCrawlerConfig } from '@kai/shared/utils/validation';  
import { crawlerQueue } from '@kai/server/services/crawler/crawlerQueue';  
  
async function configureCrawler() {  
 try {  
 // Create a crawler configuration for JinaAI  
 const jinaConfig = {  
 id: generateUUID(),  
 name: 'Tile Manufacturer Catalog Crawler',  
 url: 'https://example-tile-company.com/products',  
 provider: 'jina',  
 depth: 3,  
 extractionRules: [  
 {  
 selector: '.product-item',  
 fields: [  
 { name: 'name', selector: '.product-title', attribute: 'text' },  
 { name: 'code', selector: '.product-code', attribute: 'text' },  
 { name: 'dimensions', selector: '.dimensions', attribute: 'text' },  
 { name: 'material', selector: '.material-type', attribute: 'text' },  
 { name: 'image', selector: '.product-image img', attribute: 'src' }  
 ]  
 }  
 ],  
 schedule: {  
 frequency: 'weekly',  
 day: 'monday',  
 time: '01:00'  
 },  
 status: 'scheduled'  
 };  
  
 // Validate the configuration  
 validateCrawlerConfig(jinaConfig);  
  
 // Add the job to the queue with high priority  
 const jobId = await crawlerQueue.addJob(jinaConfig, { priority: 'high' });  
  
 console.log(`Crawler job created with ID: ${jobId}`);  
  
 return jobId;  
 } catch (error) {  
 console.error('Crawler configuration failed:', error);  
 throw error;  
 }  
}

import { validateCrawlerConfig } from '@kai/shared/utils/validation';  
import { crawlerQueue } from '@kai/server/services/crawler/crawlerQueue';  
  
async function configureCrawler() {  
 try {  
 // Create a crawler configuration for JinaAI  
 const jinaConfig = {  
 id: generateUUID(),  
 name: 'Tile Manufacturer Catalog Crawler',  
 url: 'https://example-tile-company.com/products',  
 provider: 'jina',  
 depth: 3,  
 extractionRules: [  
 {  
 selector: '.product-item',  
 fields: [  
 { name: 'name', selector: '.product-title', attribute: 'text' },  
 { name: 'code', selector: '.product-code', attribute: 'text' },  
 { name: 'dimensions', selector: '.dimensions', attribute: 'text' },  
 { name: 'material', selector: '.material-type', attribute: 'text' },  
 { name: 'image', selector: '.product-image img', attribute: 'src' }  
 ]  
 }  
 ],  
 schedule: {  
 frequency: 'weekly',  
 day: 'monday',  
 time: '01:00'  
 },  
 status: 'scheduled'  
 };  
  
 // Validate the configuration  
 validateCrawlerConfig(jinaConfig);  
  
 // Add the job to the queue with high priority  
 const jobId = await crawlerQueue.addJob(jinaConfig, { priority: 'high' });  
  
 console.log(`Crawler job created with ID: ${jobId}`);  
  
 return jobId;  
 } catch (error) {  
 console.error('Crawler configuration failed:', error);  
 throw error;  
 }  
}

#### Monitoring Crawler Jobs

import { crawlerQueue } from '@kai/server/services/crawler/crawlerQueue';  
import { messageBroker } from '@kai/server/services/messaging/messageBroker';  
  
async function monitorCrawlerJobs() {  
 try {  
 // Get all active crawler jobs  
 const jobs = Array.from(crawlerQueue.getAllJobs().values());  
 const activeJobs = jobs.filter(job =>   
 job.status === 'pending' || job.status === 'processing'  
 );  
  
 console.log(`Monitoring ${activeJobs.length} active crawler jobs`);  
  
 // Subscribe to crawler job events  
 const unsubscribe = await messageBroker.subscribe('crawler', (event) => {  
 const { type, data } = event;  
  
 switch (type) {  
 case 'job\_started':  
 console.log(`Job ${data.jobId} started at ${new Date().toISOString()}`);  
 break;  
 case 'job\_progress':  
 console.log(`Job ${data.jobId} progress: ${data.progress}%`);  
 break;  
 case 'job\_completed':  
 console.log(`Job ${data.jobId} completed at ${new Date().toISOString()}`);  
 console.log(`Results: ${data.resultCount} items extracted`);  
 break;  
 case 'job\_failed':  
 console.error(`Job ${data.jobId} failed: ${data.error}`);  
 break;  
 }  
 });  
  
 // In a real application, you'd keep the subscription active  
 // For this example, we'll unsubscribe after a short delay  
 setTimeout(() => {  
 unsubscribe();  
 console.log('Stopped monitoring crawler jobs');  
 }, 3600000); // 1 hour  
  
 return activeJobs;  
 } catch (error) {  
 console.error('Job monitoring failed:', error);  
 throw error;  
 }  
}

import { crawlerQueue } from '@kai/server/services/crawler/crawlerQueue';  
import { messageBroker } from '@kai/server/services/messaging/messageBroker';  
  
async function monitorCrawlerJobs() {  
 try {  
 // Get all active crawler jobs  
 const jobs = Array.from(crawlerQueue.getAllJobs().values());  
 const activeJobs = jobs.filter(job =>   
 job.status === 'pending' || job.status === 'processing'  
 );  
  
 console.log(`Monitoring ${activeJobs.length} active crawler jobs`);  
  
 // Subscribe to crawler job events  
 const unsubscribe = await messageBroker.subscribe('crawler', (event) => {  
 const { type, data } = event;  
  
 switch (type) {  
 case 'job\_started':  
 console.log(`Job ${data.jobId} started at ${new Date().toISOString()}`);  
 break;  
 case 'job\_progress':  
 console.log(`Job ${data.jobId} progress: ${data.progress}%`);  
 break;  
 case 'job\_completed':  
 console.log(`Job ${data.jobId} completed at ${new Date().toISOString()}`);  
 console.log(`Results: ${data.resultCount} items extracted`);  
 break;  
 case 'job\_failed':  
 console.error(`Job ${data.jobId} failed: ${data.error}`);  
 break;  
 }  
 });  
  
 // In a real application, you'd keep the subscription active  
 // For this example, we'll unsubscribe after a short delay  
 setTimeout(() => {  
 unsubscribe();  
 console.log('Stopped monitoring crawler jobs');  
 }, 3600000); // 1 hour  
  
 return activeJobs;  
 } catch (error) {  
 console.error('Job monitoring failed:', error);  
 throw error;  
 }  
}

#### Processing Crawler Results for Training

import { prepareCrawlerDataForTraining } from '@kai/ml';  
  
async function prepareTrainingDataFromCrawler(jobId: string) {  
 try {  
 // Configure options for data preparation  
 const options = {  
 jobId: jobId,  
 inputDir: `./data/crawler-results/${jobId}`,  
 outputDir: './data/training-datasets',  
 minImagesPerClass: 20,  
 imageSize: {  
 width: 512,  
 height: 512  
 },  
 augmentData: true,  
 validationSplit: 0.2  
 };  
  
 // Prepare the crawler data for training  
 const result = await prepareCrawlerDataForTraining(options);  
  
 console.log(`Crawler data preparation completed`);  
 console.log(`Dataset path: ${result.datasetPath}`);  
 console.log(`Class count: ${result.classCount}`);  
 console.log(`Total images: ${result.imageCount}`);  
  
 // Automatically start training if preparation was successful  
 if (result.status === 'success' && result.datasetPath) {  
 console.log('Starting training process with prepared data...');  
  
 // Code to initiate training would go here  
 // For example:  
 // const trainingJobId = await startTraining({  
 // datasetPath: result.datasetPath,  
 // modelType: 'materialRecognition',  
 // epochs: 50  
 // });  
 }  
  
 return result;  
 } catch (error) {  
 console.error('Crawler data preparation failed:', error);  
 throw error;  
 }  
}

import { prepareCrawlerDataForTraining } from '@kai/ml';  
  
async function prepareTrainingDataFromCrawler(jobId: string) {  
 try {  
 // Configure options for data preparation  
 const options = {  
 jobId: jobId,  
 inputDir: `./data/crawler-results/${jobId}`,  
 outputDir: './data/training-datasets',  
 minImagesPerClass: 20,  
 imageSize: {  
 width: 512,  
 height: 512  
 },  
 augmentData: true,  
 validationSplit: 0.2  
 };  
  
 // Prepare the crawler data for training  
 const result = await prepareCrawlerDataForTraining(options);  
  
 console.log(`Crawler data preparation completed`);  
 console.log(`Dataset path: ${result.datasetPath}`);  
 console.log(`Class count: ${result.classCount}`);  
 console.log(`Total images: ${result.imageCount}`);  
  
 // Automatically start training if preparation was successful  
 if (result.status === 'success' && result.datasetPath) {  
 console.log('Starting training process with prepared data...');  
  
 // Code to initiate training would go here  
 // For example:  
 // const trainingJobId = await startTraining({  
 // datasetPath: result.datasetPath,  
 // modelType: 'materialRecognition',  
 // epochs: 50  
 // });  
 }  
  
 return result;  
 } catch (error) {  
 console.error('Crawler data preparation failed:', error);  
 throw error;  
 }  
}

#### Managing Crawler Credentials

import { credentialsManager } from '@kai/server/services/crawler/credentialsManager';  
  
async function manageCrawlerCredentials() {  
 try {  
 // Check for existing credentials  
 const hasJinaCredentials = credentialsManager.hasCredentials('jina');  
 const hasFirecrawlCredentials = credentialsManager.hasCredentials('firecrawl');  
  
 console.log(`JinaAI credentials: ${hasJinaCredentials ? 'Configured' : 'Not configured'}`);  
 console.log(`Firecrawl credentials: ${hasFirecrawlCredentials ? 'Configured' : 'Not configured'}`);  
  
 // Set new credentials (typically from a secure environment or user input)  
 const jinaApiKey = process.env.JINA\_API\_KEY || 'secure-api-key-from-user-input';  
  
 if (jinaApiKey && jinaApiKey !== 'secure-api-key-from-user-input') {  
 const success = credentialsManager.setCredentials('jina', jinaApiKey);  
  
 if (success) {  
 console.log('JinaAI credentials updated successfully');  
 } else {  
 console.error('Failed to update JinaAI credentials');  
 }  
 }  
  
 // Validate credentials by testing the connection  
 if (credentialsManager.hasCredentials('jina')) {  
 // Code to test connection would go here  
 console.log('JinaAI connection successful');  
 }  
  
 return {  
 jina: credentialsManager.hasCredentials('jina'),  
 firecrawl: credentialsManager.hasCredentials('firecrawl')  
 };  
 } catch (error) {  
 console.error('Credential management failed:', error);  
 throw error;  
 }  
}

import { credentialsManager } from '@kai/server/services/crawler/credentialsManager';  
  
async function manageCrawlerCredentials() {  
 try {  
 // Check for existing credentials  
 const hasJinaCredentials = credentialsManager.hasCredentials('jina');  
 const hasFirecrawlCredentials = credentialsManager.hasCredentials('firecrawl');  
  
 console.log(`JinaAI credentials: ${hasJinaCredentials ? 'Configured' : 'Not configured'}`);  
 console.log(`Firecrawl credentials: ${hasFirecrawlCredentials ? 'Configured' : 'Not configured'}`);  
  
 // Set new credentials (typically from a secure environment or user input)  
 const jinaApiKey = process.env.JINA\_API\_KEY || 'secure-api-key-from-user-input';  
  
 if (jinaApiKey && jinaApiKey !== 'secure-api-key-from-user-input') {  
 const success = credentialsManager.setCredentials('jina', jinaApiKey);  
  
 if (success) {  
 console.log('JinaAI credentials updated successfully');  
 } else {  
 console.error('Failed to update JinaAI credentials');  
 }  
 }  
  
 // Validate credentials by testing the connection  
 if (credentialsManager.hasCredentials('jina')) {  
 // Code to test connection would go here  
 console.log('JinaAI connection successful');  
 }  
  
 return {  
 jina: credentialsManager.hasCredentials('jina'),  
 firecrawl: credentialsManager.hasCredentials('firecrawl')  
 };  
 } catch (error) {  
 console.error('Credential management failed:', error);  
 throw error;  
 }  
}

### Performance Considerations

1. Crawling Efficiency
2. Configurable concurrency levels for parallel crawling
3. Intelligent rate limiting to respect target servers
4. Caching mechanisms to avoid duplicate requests
5. Incremental crawling for regular updates
6. Resource-aware scheduling for optimal performance
7. Resource Requirements
8. CPU: Varies by provider; JinaAI more CPU-intensive for ML extraction
9. Memory: 4GB+ recommended, scales with concurrent jobs
10. Storage: Temporary space for crawl results (varies by scope)
11. Network: Reliable internet connection with adequate bandwidth
12. API Quota: Provider-specific limits on request volume
13. Scaling Considerations
14. Horizontal scaling for multiple concurrent crawls
15. Queue-based architecture for job distribution
16. Provider load balancing for optimal throughput
17. Asynchronous processing to maximize throughput
18. Rate limiting to respect external service constraints
19. Storage Optimization
20. Temporary storage for raw crawl results
21. Processed data persisted to permanent storage
22. Automatic cleanup of temporary files
23. Configurable retention policies
24. Storage efficiency through selective data extraction
25. Integration Efficiency
26. Pipeline architecture minimizing handoff overhead
27. Batch processing for downstream systems
28. Asynchronous event-based coordination
29. Prioritization based on business value
30. Caching of common data across subsystems

Resource-aware scheduling for optimal performance

Resource Requirements

API Quota: Provider-specific limits on request volume

Scaling Considerations

Rate limiting to respect external service constraints

Storage Optimization

Storage efficiency through selective data extraction

Integration Efficiency

# Workflow Variables

Source: readme/workflow-variables.md

---

## GitHub Actions Workflow Variables

This document describes the environment variables and secrets used in the GitHub Actions workflows.

### Environment Variables

These variables are set during workflow execution:

Table content:

Variable | Description | Example Value

DEPLOY\_ENV | Deployment environment | staging or production

TAG\_SUFFIX | Docker image tag suffix | latest or staging

VERCEL\_ARGS | Arguments for Vercel deployment | --prod

API\_URL | URL for API testing | https://api.kai.example.com

TEST\_SCRIPT | Test script to run | test:e2e

KUBE\_CONTEXT | Kubernetes context name | kubernetes-cluster1

DEPLOY\_ENV

staging

production

TAG\_SUFFIX

latest

staging

VERCEL\_ARGS

--prod

API\_URL

https://api.kai.example.com

TEST\_SCRIPT

test:e2e

KUBE\_CONTEXT

kubernetes-cluster1

### Repository Secrets

These secrets need to be configured in the repository settings:

#### GitHub Container Registry

* GITHUB\_TOKEN: GitHub token with write:packages permission (automatically provided by GitHub Actions)

GITHUB\_TOKEN

write:packages

#### Kubernetes Configuration

* KUBE\_CONFIG\_DATA: Base64-encoded Kubernetes config

KUBE\_CONFIG\_DATA

#### Supabase Credentials

* SUPABASE\_URL\_PRODUCTION: Supabase production URL
* SUPABASE\_KEY\_PRODUCTION: Supabase production API key
* SUPABASE\_URL\_STAGING: Supabase staging URL
* SUPABASE\_KEY\_STAGING: Supabase staging API key

SUPABASE\_URL\_PRODUCTION

SUPABASE\_KEY\_PRODUCTION

SUPABASE\_URL\_STAGING

SUPABASE\_KEY\_STAGING

#### Vercel Deployment

* VERCEL\_TOKEN: Vercel API token
* VERCEL\_ORG\_ID: Vercel organization ID
* VERCEL\_PROJECT\_ID\_CLIENT: Vercel project ID for client
* VERCEL\_PROJECT\_ID\_ADMIN: Vercel project ID for admin

VERCEL\_TOKEN

VERCEL\_ORG\_ID

VERCEL\_PROJECT\_ID\_CLIENT

VERCEL\_PROJECT\_ID\_ADMIN

#### Notifications

* SLACK\_WEBHOOK: Slack notification webhook URL

SLACK\_WEBHOOK

# Processing Summary

Total files processed: 109

Successfully processed: 109

Failed to process: 0