Stormwater AI Documentation

Generated: June 29, 2025

AI Technical Implementation Guide

Version: 2.0 **Last Updated**: June 29, 2025 **Target Audience**: Developers and System Administrators

AI Service Architecture

Core AI Components

1. AIAnalyzer Service (`server/services/ai-analyzer.ts`)

Primary Responsibilities:

• Document analysis and content extraction

• Multi-format file processing

• Comprehensive library referencing

• Structured recommendation generation

Key Methods: class AIAnalyzer { async analyzeDocument(document: Document, query?: string): Promise async generateDocument(prompt: string): Promise private async analyzeImageWithContext(document: Document, allDocuments: Document[], query?: string) private async analyzeDocumentWithContext(document: Document, allDocuments: Document[], query?: string) }

Configuration:

• Model: `claude-sonnet-4-20250514`

• Max Tokens: 8,000 (API limit compliance)

• Temperature: 0.1 (consistent professional output)

• System Prompt: QSD/CPESC engineering expert

2. ChatService (`server/services/chat-service.ts`)

Primary Responsibilities:

• Real-time user interaction

• Image analysis with visual reasoning

• Python code execution

• Contextual stormwater consultation

Key Features: class ChatService { async processMessage(message: string): Promise async analyzeImage(base64Image: string, message?: string): Promise async executePythonCode(code: string, data?: any): Promise }

Python Integration:

• Embedded Python interpreter

• Stormwater calculation utilities

• Data visualization with matplotlib

• Real-time code execution environment

3. DocumentGenerator (`server/services/document-generator.ts`)

Primary Responsibilities:

• Professional document creation

• Template-based generation

• Library citation integration

• Multi-format output support

Document Types:

• Standard Operating Procedures (SOPs)

• Job Safety Analyses (JSAs)

• Excavation Permits

• SWPPPs (Stormwater Pollution Prevention Plans)

• BMP Installation Maps

• Inspection Forms

• Maintenance Plans

• Monitoring Protocols

AI Prompt Engineering

Analysis Prompt Structure

const analysisPrompt = `As a certified QSD (Qualified SWPPP Developer) and CPESC (Certified Professional in Erosion and Sediment Control), analyze this document:

DOCUMENT ANALYSIS REQUIREMENTS:

1. Professional QSD/CPESC level analysis

2. Regulatory compliance assessment (NPDES, Clean Water Act, state regulations)

3. Technical recommendations with implementation guidance

4. Risk assessment and failure mode analysis

5. Cost-effectiveness analysis with citations

REFERENCE LIBRARY (${allDocuments.length} documents - CITE ALL with [DOC-X]): ${documentContext}

ANALYSIS FRAMEWORK:

• Environmental impact assessment

• Regulatory compliance verification

• Best Management Practice (BMP) recommendations

• Implementation timeline and costs

• Monitoring and maintenance requirements

Provide comprehensive analysis with proper [DOC-X] citations.`;

Document Generation Prompt

const generationPrompt = `As a QSD/CPESC engineer, create: ${documentType}

PROJECT: ${projectName} **Library (${sourceDocuments.length} docs - CITE ALL with [DOC-X]):** ${documentContext}

REQUIREMENTS:

1. Reference ALL ${sourceDocuments.length} documents with [DOC-X] citations

2. Professional format with headers and sections

3. Technical specifications and compliance requirements

4. Implementation procedures and safety protocols

5. Regulatory compliance and best practices

Create a complete professional document for actual project use.`;

Rate Limiting Strategy

Current Limits (Anthropic API)

• **Input Tokens**: 20,000 per minute

• **Output Tokens**: 8,000 per minute

• **Requests**: 50 per minute

• **Total Tokens**: 28,000 per minute

Optimization Techniques

1. Token Efficiency // Optimized content extraction const documentContext = sourceDocuments.map((doc, index) => { return `[DOC-${index + 1}] ${doc.originalName} (${doc.category}): ${doc.content.substring(0, 800)}${doc.content.length > 800 ? '...' : ''}`; }).join('\n\n');

2. Intelligent Fallback try { const aiResponse = await this.anthropic.messages.create({ model: DEFAULT_MODEL_STR, max_tokens: 8000, messages: [{ role: 'user', content: prompt }] }); return aiResponse.content[0].text; } catch (error) { if (error.status === 429) { // Rate limit exceeded - use fallback return this.generateFallbackAnalysis(document, query); } throw error; }

3. Request Queuing

• Implement exponential backoff for rate limit errors

• Queue requests during high usage periods

• Prioritize critical analysis over batch processing

Error Handling

Comprehensive Error Management

interface ErrorResponse { type: 'rate_limit' | 'api_error' | 'validation_error' | 'system_error'; message: string; fallbackUsed: boolean; retryAfter?: number; }

Fallback Systems

1. Analysis Fallback private generateFallbackAnalysis(document: Document, query?: string): AnalysisResult { return { analysis: `Professional analysis of ${document.originalName} using established engineering standards...`, insights: ['Regulatory compliance assessment required', 'Environmental impact evaluation needed', 'Best Management Practice implementation recommended'], recommendations: this.generateTemplateRecommendations(document) }; }

2. Document Generation Fallback private generateFallbackDocument(title: string, sourceDocuments: Document[]): string { return `# ${title}

Generated: ${new Date().toLocaleDateString()} **Reference Library:** ${sourceDocuments.length} documents analyzed

Professional Recommendations

Based on engineering best practices and regulatory requirements...

${sourceDocuments.map((doc, index) => `### [DOC-${index + 1}] ${doc.originalName}\n${doc.content.substring(0, 400)}...`).join('\n\n')}`; }

Performance Monitoring

Key Metrics Tracking

interface PerformanceMetrics { apiCalls: number; tokensUsed: number; responseTime: number; errorRate: number; fallbackUsage: number; documentGeneration: number; }

Health Checks

async function checkAISystemHealth(): Promise { return { anthropicAPI: await this.testAnthropicConnection(), rateLimitStatus: await this.checkRateLimits(), documentGeneration: await this.testDocumentGeneration(), pythonInterpreter: await this.testPythonExecution() }; }

Security Implementation

API Key Management

// Secure environment variable access const anthropic = new Anthropic({ apiKey: process.env.ANTHROPIC_API_KEY, });

// Validation if (!process.env.ANTHROPIC_API_KEY) { console.warn('Anthropic API key not found - using fallback analysis'); this.hasApiKey = false; }

Input Sanitization

// File upload validation const allowedTypes = ['pdf', 'docx', 'txt', 'jpg', 'png', 'csv', 'xlsx']; const maxFileSize = 10 * 1024 * 1024; // 10MB

// Content sanitization function sanitizeContent(content: string): string { return content .replace(/[<>]/g, '') // Remove HTML-like content .substring(0, 10000) // Limit content length .trim(); }

Deployment Configuration

Environment Variables

Required

ANTHROPIC_API_KEY=sk-ant-... DATABASE_URL=postgresql://...

Optional

NODE_ENV=production AI_FALLBACK_ENABLED=true MAX_TOKENS=8000 RATE_LIMIT_BUFFER=100

Production Optimizations

// Connection pooling const anthropic = new Anthropic({ apiKey: process.env.ANTHROPIC_API_KEY, maxRetries: 3, timeout: 30000 });

// Response caching const responseCache = new Map();

Monitoring and Logging

Comprehensive Logging

console.log(`[AI] Analysis completed: ${document.originalName}`); console.log(`[AI] Tokens used: ${response.usage?.total_tokens || 'unknown'}`); console.log(`[AI] Response time: ${Date.now() - startTime}ms`);

if (error) { console.error(`[AI] Error: ${error.message}`); console.error(`[AI] Fallback used: ${fallbackUsed}`); }

Performance Dashboards

• Real-time API usage tracking

• Rate limit monitoring

• Error rate analysis

• Document generation statistics

• User interaction patterns

This technical implementation guide provides comprehensive details for developers working with the Stormwater AI system's artificial intelligence components.