Stormwater AI Documentation

Generated: June 29, 2025

Complete System Architecture Documentation

Version: 2.0 **Last Updated**: June 29, 2025 **Document Type**: Technical Architecture Specification

System Overview

The Stormwater AI platform is a full-stack TypeScript application providing professional-grade environmental consulting through AI-powered document analysis and generation. The system transforms problem descriptions into complete actionable solution packages with QSD/CPESC level expertise.

Architecture Diagram

■■■ ■■■■ ■ CLIENT LAYER ■ ■■■ ■■■■■■■■■■■■■■■■■■■■ ■ React + TypeScript Frontend ■ ■ • Vite Build System ■ ■ • Shadcn/UI Components ■ ■ • TanStack Query (State Management) ■ ■ • Wouter (Routing) ■ ■ • Tailwind CSS (Styling) ■ ■■■ ■■■■■■■■■■■■■■■■■■ ■ ▼ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■ API LAYER ■ ■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ Express.js + TypeScript Backend ■ ■ • RESTful API Endpoints ■ ■ • File Upload Processing (Multer) ■ ■ • Session Management ■ ■ • Error Handling & Validation ■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ ▼ ■■■■■■■■■■■■■■■■■■■■■■■ ■■ ■ SERVICE LAYER ■ ■■■■■■ ■■■ ■ AI Services ■ ■ • AIAnalyzer (Document Analysis) ■ ■ • ChatService (Real-time Consultation) ■ ■ • DocumentGenerator (Professional Documents) ■ ■ ■ ■ Core Services ■ ■ • DocumentExporter (Multi-format Export) ■ ■ • PythonInterpreter (Calculations) ■ ■ • Storage Service (Data Persistence) ■ ■■ ■■■■■ ■ ▼ ■■■ ■■■■■■■■■■■■ ■ DATA LAYER ■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■ PostgreSQL Database (Drizzle ORM) ■ ■ • Documents Table ■ ■ • Recommendations Table ■ ■ • AI Analyses Table ■ ■ • Admin Sessions Table ■ ■ ■ ■ File System Storage ■ ■ • Uploads Directory (Temporary Files) ■ ■ • Generated Documents ■ ■■■ ■■■■■■■■■■■■ ■ ▼ ■■ ■■■■■■■■■■■■■■■■■■■ ■ EXTERNAL SERVICES ■ ■■■■■■■■■■■■■■■■■■■■■■■

■■ ■ • Anthropic Claude 4 API (AI Analysis) ■ ■ • PostgreSQL Database (Neon/Local) ■ ■ • File Processing Libraries ■ ■■■■■■■■■ ■■

Frontend Architecture

Technology Stack

- **Framework**: React 18 with TypeScript
- **Build Tool**: Vite with optimized bundling
- **UI Library**: Shadcn/UI (Radix UI primitives)
- **Styling**: Tailwind CSS with CSS variables
- **State Management**: TanStack Query for server state
- **Routing**: Wouter for lightweight client-side routing
- **Form Handling**: React Hook Form with Zod validation

Component Structure

client/ ■■■ src/ ■ ■■■ components/ ■ ■ ■■■ ui/ # Shadcn components ■ ■ ■■■ analysis-preview.tsx # AI analysis display ■ ■ ■■■ chat-interface.tsx # Real-time chat ■ ■ ■■■ document-upload.tsx # File upload handling ■ ■ ■■■ document-generation-checklist.tsx ■ ■ ■■■ session-download.tsx # Export functionality ■ ■■■ pages/ ■ ■ ■■■ professional-main.tsx # Main interface ■ ■ ■■■ admin-dashboard.tsx # Admin panel ■ ■ ■■■ legacy-interface.tsx # Backup interface ■ ■■■ lib/ ■ ■ ■■■ queryClient.ts # API configuration ■ ■ ■■■ utils.ts # Utility functions ■ ■■■ App.tsx # Route configuration

State Management

- **Server State**: TanStack Query with automatic caching
- **Local State**: React hooks for component state
- **Form State**: React Hook Form with Zod schemas
- **Theme State**: CSS variables with dark mode support

Backend Architecture

Express.js Server Structure

server/ ■■■ index.ts # Application entry point ■■■ routes.ts # API route definitions ■■■ db.ts # Database connection ■■■ storage.ts # Data access layer ■■■ vite.ts # Vite integration ■■■ services/

■■■ ai-analyzer.ts # AI document analysis ■■■ chat-service.ts # Interactive chat ■■■ document-generator.ts # Professional documents ■■■ document-exporter.ts # Export functionality ■■■ python-interpreter.ts # Calculation engine

API Endpoints

// Document Management GET /api/documents # List all documents POST /api/documents # Upload document GET /api/documents/:id # Get specific document DELETE /api/documents/:id # Delete document

// AI Analysis POST /api/analyze # Analyze document GET /api/analyses # List analyses GET /api/analyses/:id # Get specific analysis

// Document Generation POST /api/generate-documents # Generate professional docs GET /api/generated-documents # List generated docs GET /api/download/:filename # Download document

// Interactive Features POST /api/chat # Chat with AI POST /api/chat/image # Image analysis POST /api/python/execute # Python calculations

// System Management GET /api/stats # System statistics GET /api/health # Health check POST /api/admin/login # Admin authentication

Database Schema

Documents Table

CREATE TABLE documents (id SERIAL PRIMARY KEY, original_name VARCHAR(255) NOT NULL, file_path VARCHAR(500) NOT NULL, content TEXT, file_size INTEGER, mime_type VARCHAR(100), category VARCHAR(50) DEFAULT 'stormwater', subcategory VARCHAR(50), uploaded_at TIMESTAMP DEFAULT NOW(), is_library_document BOOLEAN DEFAULT FALSE);

Recommendations Table

CREATE TABLE recommendations (id SERIAL PRIMARY KEY, title VARCHAR(255) NOT NULL, content TEXT NOT NULL, category VARCHAR(50) DEFAULT 'stormwater', subcategory VARCHAR(50), citation VARCHAR(100), is_bookmarked BOOLEAN DEFAULT FALSE, created_at TIMESTAMP DEFAULT NOW());

AI Analyses Table

CREATE TABLE ai_analyses (id SERIAL PRIMARY KEY, document_id INTEGER REFERENCES documents(id), query TEXT, analysis TEXT NOT NULL, insights TEXT[], created_at TIMESTAMP DEFAULT NOW());

Admin Sessions Table

CREATE TABLE admin_sessions (id SERIAL PRIMARY KEY, email VARCHAR(255) NOT NULL, token VARCHAR(255) UNIQUE NOT NULL, created_at TIMESTAMP DEFAULT NOW(), expires_at TIMESTAMP NOT NULL);

AI Service Architecture

AIAnalyzer Service

Purpose: Core document analysis and 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[]) private async analyzeDocumentWithContext(document: Document, allDocuments: Document[]) private buildReferenceContext(allDocuments: Document[]): string private parseAnalysisResponse(analysisText: string): AnalysisResult }

Features:

- Multi-format document processing
- Comprehensive library referencing with [DOC-X] citations
- Professional QSD/CPESC level analysis
- Intelligent fallback system during rate limits

ChatService

Purpose: Real-time interactive consultation

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

Features:

- Real-time AI consultation
- Image analysis with visual reasoning
- Embedded Python interpreter
- Contextual stormwater engineering responses

DocumentGenerator

Purpose: Professional document creation

Key Methods: class DocumentGenerator { async generateDocument(request: DocumentGenerationRequest): Promise async generateSolutionDocuments(params: SolutionParams): Promise private async generateComprehensiveReport(title: string, query?: string): Promise private determineSolutionDocumentTypes(problem: string): string[] }

Document Types:

- Standard Operating Procedures (SOPs)
- Job Safety Analyses (JSAs)
- Excavation Permits
- Stormwater Pollution Prevention Plans (SWPPPs)
- Best Management Practice (BMP) Maps
- Inspection Forms
- Maintenance Plans
- Monitoring Protocols

Security Architecture

Authentication System

// Admin authentication for library management interface AdminSession { email: string; token: string; expiresAt: Date; }

// Secure email verification const ADMIN_EMAIL = 'guzman.danield@outlook.com';

File Security

- **Upload Validation**: File type and size restrictions
- **Temporary Storage**: Automatic cleanup after processing
- **Content Sanitization**: Input validation and XSS prevention
- **Path Security**: Prevent directory traversal attacks

API Security

- **Environment Variables**: Secure API key storage
- **Rate Limiting**: Request throttling and abuse prevention
- **CORS Configuration**: Cross-origin request management
- **Error Sanitization**: Prevent information leakage

Performance Architecture

Response Time Optimization

- **Database Indexing**: Optimized queries for common operations
- **Response Caching**: Intelligent caching for frequent requests
- **Parallel Processing**: Simultaneous document analysis
- **Token Optimization**: Efficient AI prompt engineering

Scalability Design

- **Stateless Services**: Horizontal scaling capability
- **Database Connection Pooling**: Efficient resource utilization
- **File Storage Strategy**: Scalable temporary file management
- **Load Balancing Ready**: Multiple instance support

Rate Limiting Management

```
// Current Anthropic API limits const RATE_LIMITS = { inputTokens: 20000, // per minute outputTokens:
8000, // per minute requests: 50, // per minute totalTokens: 28000 // per minute };

// Intelligent fallback system async function handleRateLimit(error: RateLimitError): Promise { if
(error.status === 429) { return generateFallbackResponse(); } throw error; }
```

Deployment Architecture

Production Environment

```
// Environment configuration interface Environment { NODE_ENV: 'production' | 'development';
DATABASE_URL: string; // PostgreSQL connection ANTHROPIC_API_KEY: string; // Claude API
access PORT: number; // Server port (default: 5000) }
```

Build Process

1. **Frontend Build**: Vite bundles React app to `dist/public`
2. **Backend Build**: ESBuild compiles TypeScript to `dist/index.js`
3. **Database Migration**: Drizzle applies schema changes
4. **Asset Optimization**: Static file compression and caching

Monitoring and Logging

```
// Performance monitoring interface SystemMetrics { documentCount: number; analysisCount: number;
apiCallsPerMinute: number; averageResponseTime: number; errorRate: number; uptime: number; }
```

```
// Health checks async function healthCheck(): Promise { return { database: await
checkDatabaseConnection(), anthropicAPI: await checkAnthropicStatus(), fileSystem: await
checkFileSystemAccess(), overallStatus: 'healthy' | 'degraded' | 'down' }; }
```

Integration Points

External Services

- **Anthropic Claude 4**: AI analysis and document generation
- **PostgreSQL**: Primary data storage (Neon for cloud deployment)
- **File Processing**: PDF, DOCX, image, and spreadsheet libraries
- **Python Runtime**: Embedded interpreter for calculations

API Integrations

```
// Anthropic Claude 4 integration const anthropic = new Anthropic({ apiKey:
process.env.ANTHROPIC_API_KEY, model: 'claude-sonnet-4-20250514', maxTokens: 8000 });
```

```
// Database integration with Drizzle ORM const db = drizzle({ client: pool, schema });
```

Data Flow Architecture

Document Analysis Flow

1. **User Upload** → File validation and temporary storage
2. **Content Extraction** → Text/image content parsing
3. **Library Analysis** → AI analysis against all reference documents
4. **Recommendation Generation** → Structured professional recommendations
5. **Document Creation** → Auto-generation of solution documents
6. **Response Delivery** → Formatted results with citations

Interactive Chat Flow

1. **User Message** → Input validation and processing
2. **Context Building** → Integration with document library

3. **AI Processing** → Claude 4 analysis with extended reasoning
4. **Response Generation** → Professional consultation response
5. **History Management** → Session-based conversation tracking

Document Generation Flow

1. **Problem Analysis** → Automatic document type determination
2. **Template Selection** → Professional document templates
3. **Content Generation** → AI-powered content creation with citations
4. **Quality Assurance** → Validation and formatting checks
5. **Delivery** → Multiple format options (PDF, DOCX, TXT)

This complete system architecture documentation provides comprehensive technical details for understanding, maintaining, and extending the Stormwater AI platform.