Knowledge Management System - Updated Implementation Plan

Date: August 15, 2025

Status: Phase 2A Complete, Moving to Phase 2B

Stack: Python 3.10, FastAPI, Azure SQL Server, Azure Web Apps

Deployment: GitHub Actions → Azure Web Apps (No Docker)

🎯 Executive Summary

We have successfully completed the foundation of the Knowledge Management System with **km-mcp-sql-docs** now production-ready. The system features:

- ✅ Document storage with interactive web UI
- ✅ Full-text search capabilities
- ✅ Working statistics and health monitoring
- ✅ Automated CI/CD deployment pipeline
- ✅ 100% functional production system

Next Phase: Build km-mcp-phi4 to enable AI reasoning over stored documents.

🏗 Current System Architecture

```
┌─────────────────┐
│   Astro UI      │  (Frontend - Future)
└────────┬────────┘
         │ HTTP/REST
         ▼
┌─────────────────┐
│ km-orchestrator │  (FastAPI Router - Future)
│                 │  Routes requests to MCP servers
└────────┬────────┘
         │
    ┌────┴────┬─────────┬─────────┬─────────┐
    │         │         │         │         │
    ▼         ▼         ▼         ▼         ▼
┌─────────┐ ┌─────────┐ ┌─────────┐ ┌─────────┐ ┌─────────┐
│km-mcp-  │ │km-mcp-  │ │km-mcp-  │ │km-mcp-  │ │km-mcp-  │
│sql-docs │ │phi4     │ │search   │ │graphrag │ │vector   │
│ ✅ DONE  │ │ 🔄 NEXT  │ │ 🔄 PLAN  │ │ 🔄 PLAN  │ │ 🔄 PLAN  │
└─────────┘ └─────────┘ └─────────┘ └─────────┘ └─────────┘
    │         │         │         │         │
    ▼         ▼         ▼         ▼         ▼
[Azure SQL] [Phi-4]  [Search]  [GraphRAG] [Vector DB]
```

✅ Phase 2A: Completed Components

km-mcp-sql-docs (Production Ready)

URL: https://km-mcp-sql-docs.azurewebsites.net

Features Implemented:

- ✅ Document storage with metadata support
- ✅ Full-text search with filters
- ✅ Interactive web UI with forms
- ✅ Real-time statistics (fixed to use 'status' column)
- ✅ Health monitoring and database connectivity
- ✅ GitHub Actions automated deployment
- ✅ Beautiful responsive design with clickable endpoints

Database Schema Used:

- Uses existing table with columns: id, title, content, classification, entities, metadata, file_data, file_name, file_type, file_size, category_id, created_at, updated_at, indexed_at, **status**, user_id, source_url, embedding_vector

API Endpoints:

- `GET /` - Interactive web interface
- `GET /health` - Health check and database status
- `POST /tools/store-document` - Store new documents
- `POST /tools/search-documents` - Search with filters
- `GET /tools/database-stats` - System statistics
- `GET /docs` - API documentation

Key Achievement: Successfully identified and fixed the `is_active` vs `status` column issue that was preventing statistics from working.

🚀 Phase 2B: Document Processing Pipeline (NEXT)

Priority 1: km-mcp-phi4 (Immediate Next Step)

Purpose: AI reasoning and analysis over stored documents

Planned Features:

- 🔄 Document analysis and insights extraction
- 🔄 Q&A system using document context
- 🔄 Intelligent summarization
- 🔄 Content classification and tagging
- 🔄 Interactive chat interface for document exploration

Technical Implementation:

```python
# Planned API endpoints
POST /tools/analyze-document     # Analyze single document
POST /tools/summarize-content    # Create document summaries
POST /tools/answer-question      # Q&A with document context
POST /tools/extract-insights     # Extract key themes and insights
POST /tools/chat-with-docs       # Interactive document chat
```

Integration Points:

- Connects to km-mcp-sql-docs via HTTP API
- Retrieves documents for analysis
- Stores analysis results back to database
- Provides AI insights through web interface

Deployment:

- Same pattern as km-mcp-sql-docs
- Azure Web App with Python 3.10
- GitHub Actions automation
- Beautiful interactive UI

Priority 2: Document Processing Pipeline

Purpose: Intelligent chunking and tokenization for AI processing

Planned Features:

- 🔄 Smart document chunking (semantic boundaries)
- 🔄 Token counting and optimization
- 🔄 Metadata extraction and enrichment
- 🔄 Content preprocessing for downstream services

Workflow:

1. Document uploaded to km-mcp-sql-docs
2. Processing pipeline triggered automatically
3. Content chunked and analyzed
4. Chunks prepared for GraphRAG and Vector search
5. Results stored for fast retrieval

🔄 Phase 2C: Advanced Search & Knowledge Graph

Priority 3: km-mcp-search

Purpose: Semantic and vector-based search capabilities

Planned Features:

- 🔄 Vector embeddings generation
- 🔄 Semantic similarity search
- 🔄 Hybrid search (keyword + semantic)
- 🔄 Advanced result ranking and filtering

Integration:

- Processes document chunks from processing pipeline
- Maintains vector index for fast similarity search
- Provides search API for km-orchestrator

Priority 4: km-mcp-graphrag

Purpose: Knowledge graph construction and graph-based reasoning

Planned Features:

- 🔄 Entity extraction from documents
- 🔄 Relationship mapping and graph construction
- 🔄 Graph-based query answering
- 🔄 Complex reasoning over connected knowledge

Integration:

- Builds knowledge graph from processed document chunks
- Provides graph traversal and reasoning APIs
- Enables complex multi-hop questions

🎯 Phase 3: Orchestration & Frontend

Priority 5: km-orchestrator

Purpose: Intelligent request routing and workflow orchestration

Planned Features:

- 🔄 Smart routing to appropriate MCP services
- 🔄 Result combination from multiple services
- 🔄 Workflow management for complex operations
- 🔄 API gateway with authentication

API Routes:

```python
POST /api/upload      # → km-mcp-sql-docs + processing pipeline
POST /api/search      # → km-mcp-search + km-mcp-sql-docs
POST /api/analyze     # → km-mcp-phi4 + km-mcp-graphrag
POST /api/insights    # → Combined multi-service responses
POST /api/chat        # → Interactive AI chat across all services
```

Priority 6: km-ui (Astro Frontend)

Purpose: Unified user interface for the entire system

Planned Features:

- 🔄 Document upload and management interface
- 🔄 Advanced search interface with filters
- 🔄 AI chat interface for document exploration
- 🔄 Analytics dashboard with visualizations
- 🔄 Knowledge graph visualization
- 🔄 System administration tools

📁 Updated Project Structure

```
km-system/
├── .github/workflows/
│   ├── deploy-km-sql-docs.yml    ✅ PRODUCTION
│   ├── deploy-km-phi4.yml    🔄 BUILD NEXT
│   ├── deploy-km-search.yml     🔄 PLAN
│   ├── deploy-km-graphrag.yml    🔄 PLAN
│   └── deploy-km-orchestrator.yml 🔄 PLAN
├── km-mcp-sql-docs/           ✅ PRODUCTION READY
│   ├── app.py                 # FastAPI with interactive UI
│   ├── km_docs_operations.py   # Database operations (fixed)
│   ├── km_docs_config.py       # Configuration management
│   ├── km_docs_schemas.py      # Pydantic models
│   ├── requirements.txt        # Dependencies
│   ├── index.html              # Beautiful web interface
│   └── .deployment             # Azure deployment config
├── km-mcp-phi4/               🔄 BUILD NEXT
│   ├── app.py                 # FastAPI main application
│   ├── km_phi4_operations.py   # Phi-4 AI operations
│   ├── km_phi4_config.py       # Phi-4 configuration
│   ├── km_phi4_schemas.py      # Request/response models
│   ├── requirements.txt        # AI/ML dependencies
│   └── .deployment             # Azure deployment config
├── km-mcp-search/             🔄 PLAN
├── km-mcp-graphrag/            🔄 PLAN
├── km-orchestrator/           🔄 PLAN
├── km-ui/                  🔄 PLAN
└── shared/
    ├── km_common_utils.py      # Shared utilities
    └── km_base_config.py       # Base configuration
```

🛠 Technical Specifications

Deployment Pattern (Proven Successful)

- **Platform:** Azure Web Apps (Python 3.10)
- **Framework:** FastAPI with interactive HTML UI
- **Database:** Azure SQL Server (existing schema)
- **CI/CD:** GitHub Actions with automated deployment
- **Monitoring:** Built-in health checks and statistics

Common Dependencies (requirements.txt)

```python
# Core Framework
fastapi==0.104.1
uvicorn[standard]==0.24.0

# Database
pyodbc==5.0.1
sqlalchemy==2.0.23

# HTTP & API
httpx==0.25.1
pydantic==2.5.0
pydantic-settings==2.1.0

# AI/ML (for phi4 service)
torch>=2.0.0
transformers>=4.30.0
sentence-transformers>=2.2.0

# Utilities
python-dotenv==1.0.0
python-multipart==0.0.6

# Testing
pytest==7.4.3
pytest-asyncio==0.21.1
```

Azure Configuration

```ini
```

```
# Startup Command
python -m uvicorn app:app --host 0.0.0.0 --port 8000

# Environment Variables
WEBSITES_PORT=8000
SCM_DO_BUILD_DURING_DEPLOYMENT=true
ENABLE_ORYX_BUILD=true

# Database Connection (from existing setup)
KM_SQL_SERVER=knowledge-sql.database.windows.net
KM_SQL_DATABASE=knowledge-base
KM_SQL_USERNAME=mcpadmin
KM_SQL_PASSWORD=Theodore03$
```

📊 Success Metrics & Milestones

Phase 2A: ✅ ACHIEVED

- [x] Document storage working (17+ documents stored)
- [x] Search functionality operational
- [x] Statistics displaying correct counts
- [x] Interactive web UI deployed
- [x] Automated deployment pipeline

Phase 2B: 🎯 TARGET METRICS

- [ ] km-mcp-phi4 can analyze documents from km-mcp-sql-docs
- [ ] Q&A system provides relevant answers using document context
- [ ] Document summarization generates coherent summaries
- [ ] Processing pipeline chunks documents intelligently
- [ ] All services have beautiful interactive UIs

Phase 2C: 🎯 TARGET METRICS

- [ ] Vector search returns semantically relevant results
- [ ] Knowledge graph extracts meaningful entities and relationships
- [ ] Hybrid search combines keyword and semantic results effectively
- [ ] Graph-based reasoning answers complex multi-hop questions

Phase 3: 🎯 TARGET METRICS

- [ ] km-orchestrator routes requests intelligently

- [ ] Frontend provides unified access to all capabilities
- [ ] End-to-end workflows: upload → process → search → analyze
- [ ] System handles complex document analysis workflows

🔄 Immediate Next Steps

Step 1: Build km-mcp-phi4 Service

1. **Create service structure** following km-mcp-sql-docs pattern
2. **Implement Phi-4 integration** for document analysis
3. **Build interactive web UI** with analysis forms
4. **Set up GitHub Actions deployment**
5. **Test integration** with existing km-mcp-sql-docs

Step 2: Document Processing Pipeline

1. **Design chunking strategy** for different document types
2. **Implement tokenization** with proper token counting
3. **Create metadata extraction** workflows
4. **Build processing triggers** from document uploads

Step 3: Advanced Search Implementation

1. **Set up vector database** (Azure Cognitive Search or similar)
2. **Implement embedding generation** for documents
3. **Build search API** with ranking and filtering
4. **Create search interface** in web UI

🔧 Development Workflow

Proven Pattern (from km-mcp-sql-docs success)

1. **Local Development**

```bash
```

```
cd km-mcp-{service}
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate
pip install -r requirements.txt
uvicorn app:app --reload --port 8000
```

2. **GitHub Integration**

```bash
git add km-mcp-{service}/
git commit -m "ADD: {service} implementation"
git push origin master
```

3. **Automated Deployment**
    - GitHub Actions triggers on file changes
    - Deploys to Azure Web App automatically
    - Health checks verify deployment success
4. **Testing & Verification**
    - Interactive web UI for manual testing
    - Health endpoints for monitoring
    - Integration testing between services

📋 Risk Mitigation & Lessons Learned

Successful Patterns ✅

- **Interactive web UIs** provide excellent debugging and user experience
- **Direct database inspection** helps identify real vs assumed schema
- **GitHub Actions automation** enables rapid iteration
- **Health checks and statistics** provide operational visibility

Avoided Issues ✅

- **Docker complexity** - Direct Azure Web Apps much simpler
- **Configuration mismatch** - Environment variables work reliably
- **Database schema assumptions** - Always verify actual column names
- **Deployment delays** - GitHub Actions faster than manual deployment

Future Considerations 🔄

- **Service communication** - HTTP APIs between services initially
- **Error handling** - Comprehensive error responses and logging
- **Performance** - Monitor response times as system grows
- **Security** - Add authentication as system matures

🎉 Achievement Summary

What We've Built

- **Production-ready document storage system** with beautiful UI
- **Automated CI/CD pipeline** that deploys on every commit
- **Robust database integration** with proper schema handling
- **Interactive web interfaces** for all functionality
- **Comprehensive monitoring** with health checks and statistics

What We've Learned

- **Azure Web Apps** deployment pattern works excellently
- **FastAPI + Interactive HTML** provides great developer experience
- **GitHub Actions** enables rapid iteration and deployment
- **Database schema verification** is critical for avoiding issues

Ready for Next Phase

With km-mcp-sql-docs proven and stable, we have:

- **Solid foundation** for building additional services
- **Proven deployment pattern** to replicate
- **Working database integration** to extend
- **Beautiful UI pattern** to follow for new services

📞 Contact & Repository Information

- **GitHub Repository:** https://github.com/software-tim/km-system
- **Production Service:** https://km-mcp-sql-docs.azurewebsites.net
- **GitHub Actions:** https://github.com/software-tim/km-system/actions

---

Document Version: 2.0

Last Updated: August 15, 2025

Status: Phase 2A Complete, Ready for Phase 2B

Next Milestone: km-mcp-phi4 Service Implementation