# RAG System Implementation and Troubleshooting: Summary

## Initial Setup and Challenges

- You were implementing a RAG (Retrieval-Augmented Generation) system using TinyLlama (1.1B model) with llama.cpp for your project management knowledge base (pmp_combined.txt).
- The system successfully processed the 9.9MB knowledge base file and created vector embeddings stored in FAISS format.
- The main challenge was the web interface getting stuck at "System Status: Checking" despite the backend successfully initializing.

## Troubleshooting Process

1. **Initial diagnosis**: The backend was showing successful initialization (6-7 seconds) but the frontend was not receiving status updates.
2. **Status API debugging**: Added debug logs and fixed the status endpoint but issues persisted.
3. **JavaScript debugging**: Fixed browser cache issues by adding cache-busting parameters and improving error handling.
4. **CORS implementation**: Added Flask-CORS to enable cross-origin requests for a standalone interface.

## Final Solution

1. **Decoupled architecture**:
   - Backend: Python Flask server providing API endpoints (/api/query, /api/status, /api/audit)
   - Frontend: Standalone HTML file (rag-interface.html) that connects to the backend
2. **Knowledge Auditing System**:
   - Implemented comprehensive knowledge tracking via the KnowledgeAuditor class
   - Records which parts of the knowledge base are used for each query
   - Provides analytics on knowledge utilization patterns
3. **Optimized RAG Pipeline**:
   - Improved prompt template with formatting instructions (bullet points, headers)
   - Added better error handling and debugging throughout the pipeline
   - Ensured stable initialization with proper thread management

## Technical Components

1. **Vector Store**: FAISS index for fast semantic similarity searches
2. **Embedding Model**: HuggingFaceEmbeddings with "sentence-transformers/all-MiniLM-L6-v2"
3. **LLM**: TinyLlama 1.1B using llama.cpp with optimized settings for M1 Mac
4. **Framework**: LangChain for the RAG pipeline components

5. **Web Server**: Flask with CORS support for API endpoints

## Usage Pattern

1. Start the server: ./start.sh or python3 app.py
2. Open rag-interface.html in a browser
3. Submit project management queries
4. View structured responses with source citations
5. Optionally review knowledge utilization audits

## Future Refinements (Planned for Next Week)

- Revisit the integrated UI approach
- Optimize response formatting
- Address browser caching issues
- Further improve the knowledge auditing capabilities

The system now provides a functional RAG interface for querying project management knowledge with comprehensive auditing capabilities to track how knowledge is being utilized.