# Knowledge Assistant Architecture

This document describes the architecture of the Knowledge Assistant prototype for the AI Coding Challenge. It outlines how support documentation is processed, embedded, and retrieved to answer user queries via a minimal RAG pipeline.

The system is designed to balance simplicity, correctness, and production-readiness.

## 1. Overview

The system:

* Accepts support tickets via a single API endpoint (POST /resolve-ticket)
* Retrieves relevant knowledge from a vector store (FAISS)
* Generates structured, MCP-compliant responses using a local LLM (cpp-llama)
* Supports fallback mechanisms for prompts and embeddings in production

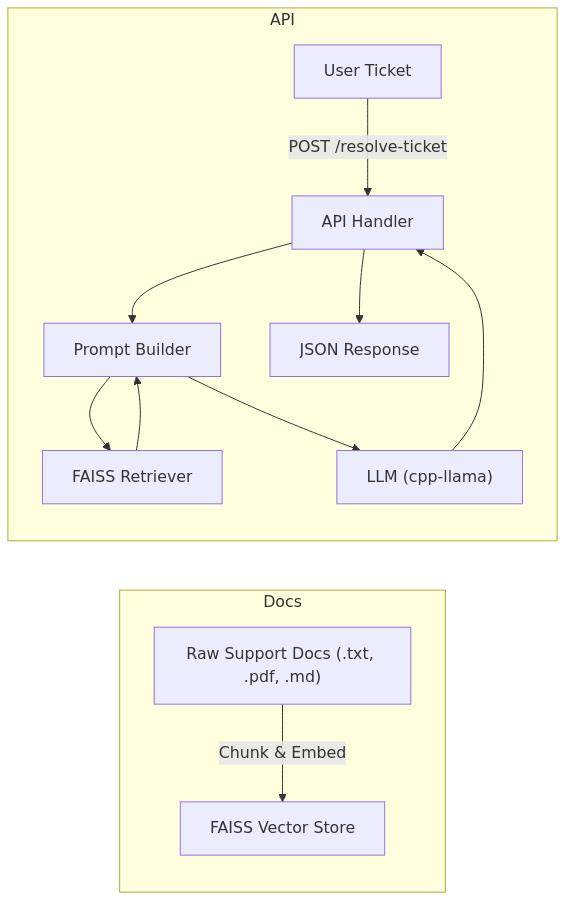
## 2. High-Level Workflow

RAG API FLOW

RAG API FLOW

* User: Provides the support ticket text.
* API Endpoint: Receives the request and triggers the retrieval pipeline.
* Prompt Builder: Combines the ticket text with retrieved context and ensures MCP-compliant output format.
* Retrieval Module: Fetches top\_k relevant document chunks from FAISS embeddings.
* LLM: Generates the answer based on context and ticket, adhering to MCP principles.
* Response: Structured JSON returned to the user.

## 3. Data Flow Diagram



FAISS Doc Pipeline

* Raw Docs: Stored in data/raw/. Pre-processed, chunked, and embedded into FAISS (data/faiss\_index/).
* FAISS Vector Store: Stores embeddings with provenance metadata.
* Prompt Builder: Combines user query and retrieved chunks into an MCP-compliant prompt.
* LLM (cpp-llama): Generates structured output.
* Response: Returns JSON including answer, references, and action\_required.

## 4. Module Breakdown

| Module | Responsibility |
| --- | --- |
| data\_handling | Parse, chunk, embed documents; build FAISS index; manage fallback embeddings. |
| core | Retrieval module and LLM integration; constructs MCP prompt and generates responses. |
| api | FastAPI endpoint; receives tickets and returns structured JSON. |
| utils | Configuration management, logging, and helper functions. |

## 5. Fallback & Production Considerations

**Prompts** - Stored in prompts/latest.json. - If the latest file is missing, **default prompts** baked into the container are used to ensure robustness.

**Embeddings** - Pre-embedded FAISS vectors are loaded from data/faiss\_index/. - If unavailable, the **CI/CD pipeline** guarantees that a default index is built and available in the deployment volume.

**Deployment** - Docker **multi-stage build** is used for deployment only. - API and FAISS index are deployed; developers do **not** need to run models inside Docker. - Future work could integrate **Kubernetes** for scalable deployment.

## 6. Future Enhancements

* Hot-reload prompts and embeddings.
* Human-in-the-loop evaluation and MLflow tracking for prompt optimization.
* Swap FAISS for cloud vector DBs for scaling.
* Kubernetes deployment for high availability and scaling.
* Observability with logging and metrics.

## 7. Notes

All document chunks include provenance metadata (filename, section, subsection) for traceable references in responses.

MCP compliance ensures responses are structured, reproducible, and easy to evaluate.