Technical Documentation: Daniel Rada AI Assistant Table of Contents Project Overview

1. Project Overview Personal AI assistant system using RAG (Retrieval-Augmented Generation) to answer questions about Daniel Rada's professional experience. Combines document processing, language models, and vector storage.

Key Tools:

Backend: FastAPI (Python)

Frontend: Flask (Python)

LLM Model: Mistral-7B (quantized GGUF)

Vector Store: ChromaDB

Embeddings: Multilingual Sentence Transformers

Memory: Redis

PDF Processing: PyPDF2 + Tesseract OCR

2. System Architecture Diagram Code

3. Core Components 3.1 Document Processing (pdf_to_text.py) Function: Convert CVs/PDFs to structured text

Features:

Automatic detection of scanned PDFs (OCR with Tesseract)

Parallel processing with ThreadPool

Text cleaning with regex

Usage:

bash python pdf_to_text.py --input pdfs --output data 3.2 Embedding Creation (ingest.py) Function: Generate embeddings and store in ChromaDB

Workflow:

Load TXT documents

Split text into chunks (800 tokens with 150 overlap)

Generate embeddings with multilingual model

Store in persistent ChromaDB

Model: sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2

3.3 Backend API (api.py) Endpoints:

POST /ask: Process questions using RAG

POST /feedback: Receive response corrections

GET /health: Check system status

Features:

Conversation memory with Redis

Session management

Detailed logging

Dependencies:

python vector_store = Chroma(persist_directory="embeddings") 3.4 RAG Model (model.py) Components:

Prompt Template: Strict bilingual structure (ES/EN)

LLM Model: Mistral-7B (4-bit quantized)

Configuration:

python LlamaCpp(model_path=".../mistral-7b.Q4_K_M.gguf", n_gpu_layers=33, temperature=0.3) Retrieval: MMR (Maximal Marginal Relevance) search with 4 relevant documents

3.5 Web Interface (web_app.py) Technology: Flask + Jinja2

Features:

Unique session ID generation

Backend API communication

Integrated feedback system

Structure:

text /web ├── templates/ │ └── index.html ├── static/ └── web_app.py

4. Support Scripts Script Function rename.sh Remove spaces from filenames remove_dashes.sh Remove hyphens from filenames update_ai.sh Update embeddings and restart services compatibility_check.py Verify dependency versions
5. Workflow Installation: Install dependencies:

bash pip install -r requirements.txt Download Mistral-7B model:

bash ./scripts/download_model.sh Start Redis:

bash docker run -p 6379:6379 redis Initial Processing: Diagram Code Execution: bash

Backend (API)

python api.py

Frontend (Web)

python web_app.py

6. Key Mechanisms Memory Management Stores conversation history per session in Redis

Key format: danielai:<session_id>

Retention: 30 days (configurable)

Feedback System Stores corrections in CSV:

text question,response,correction Path: feedback_data.csv

Language Handling Strict rules in prompt:

text IF spanish_question → spanish_response IF english_question → english_response

7. Technical Considerations Hardware Requirements:

Recommended GPU (≥8GB VRAM)

Minimum 16GB RAM

Storage: 10GB+ (models + embeddings)

Optimizations:

4-bit model quantization

Batch document processing

Redis shared memory usage

Security:

Sessions don't store personal data

Redis not publicly exposed

Input validation with Pydantic

8. Future Improvements Implement fine-tuning with feedback data

Add JWT authentication

Priority system for key documents

Conversation monitoring dashboard

9. Troubleshooting Issue: Model fails to load Solution: Verify model path in model.py

Issue: ChromaDB not persisting Solution: Check permissions for embeddings/ directory

Issue: Responses in wrong language Solution: Validate prompt template in model.py

bash

Verify compatibility:

python compatibility_check.py Note: This documentation covers system version 1.0. For updates, refer to the official repository.