AUSLegalSearch v3 — Consultancy Handover

Table of Contents

<!– AUSLegalSearch v3 — Comprehensive Code & Architecture Walkthrough

Prepared for: [Insert Client/Department name] Prepared by: [Insert Consultant/Author] Date: {{DATE}} Revision: 1.0

# Title Page

**AUSLegalSearch v3 — Full Technical & Functional Handover**

*Professional walkthrough and architecture explainer of the AUSLegalSearch platform: Streamlit, Gradio, FastAPI, Embedding and Ingestion, Database Schema, and System Extension.*

# Table of Contents

1. Executive Summary
2. Application Architecture & Component Diagram
3. Deployment & Environment Setup
4. Database Schema, Extensions, and DDL
5. Document Ingestion & Chunking Pipeline
6. Embedding Model and Vector Storage
7. Backend REST API: fastapi\_app.py Walkthrough
8. Streamlit Application: app.py Walkthrough
9. Gradio Frontend: gradio\_app.py Walkthrough
10. RAG Pipeline Details (rag/)
11. Embedding Pipeline (embedding/)
12. Ingest & Parsing Pipeline (ingest/)
13. UI Workflows: Login, Search, RAG, Agentic Chat
14. Security & User Management
15. Extending the System & Troubleshooting Guide
16. Appendix: Code Snippets & Example Sessions

## 1. Executive Summary

AUSLegalSearchv3 integrates advanced vector-based legal document retrieval, easy ingestion & chunking, with GenAI/AIRAG and hybrid search, delivered as powerful Streamlit and Gradio UIs on a robust Python/FastAPI backend.

### Key Features

* Hybrid legal search—semantics + keyword/bm25
* Full text and metadata FTS with phrase, stemming, fuzzy
* Robust ingest/embedding workflows, parallelized for scale
* Support for local models (Ollama, BGE, MiniLM, Nomic) and OCI GenAI
* Modular, extensible, schema-auto-initializing design

## 2. Architecture Overview

[!NOTE: Add a diagram using a tool like draw.io; for now, describe:]

* **Frontend:** Streamlit (interactive legal RAG, ingestion), Gradio (multitabs: Hybrid, Vector, RAG, Agentic)
* **API Layer:** FastAPI server exposes endpoints for ingestion, search, chat, etc.
* **Database:** PostgreSQL (with pgvector, FTS, Trigram extensions, all auto-managed)
* **Embeddings:** Nomic-AI v1.5 (default), 768D; extensible to other HuggingFace, OpenAI, etc.
* **Pipelines:**
  + Data flows from ingestion through chunking to embedding, storage, index, and downstream search.

### Component Interactions

* Users upload data/search via UI → Backend API → Ingests, Embeds, Stores, Responds

## 3. Deployment & Environment Setup

**Dependencies:**  
- Python 3.9+  
- PostgreSQL with required extensions (auto-handled) - [See requirements.txt for package list, e.g. sentence-transformers, streamlit, gradio, fastapi, etc.]

**Initial Setup:**

python3 -m venv venv  
source venv/bin/activate  
pip install -r requirements.txt  
python -m db.store # auto-initializes all tables, triggers, indexes, extensions

**Run API:**

uvicorn fastapi\_app:app --port 8000

**Run Streamlit UI:**

streamlit run app.py

**Run Gradio UI:**

python gradio\_app.py

## 4. Database Schema, Extensions, and DDL

(Include schema diagram if possible; see db/store.py)

### Tables:

* users
* documents
* embeddings
* embedding\_sessions, embedding\_session\_files
* chat\_sessions
* conversion\_files

Include sample code for models (see db/store.py).

### PostgreSQL Extensions/Indexes (auto-applied):

* CREATE EXTENSION IF NOT EXISTS vector
* CREATE EXTENSION IF NOT EXISTS pg\_trgm
* CREATE EXTENSION IF NOT EXISTS “uuid-ossp”
* CREATE EXTENSION IF NOT EXISTS fuzzystrmatch

def create\_all\_tables():  
 # Enables extensions, creates DDLs, triggers for FTS/jobs/indexes  
 ...

[Explain DDLs for FTS/triggers/indexes]

## 5. Document Ingestion & Chunking Pipeline

Code: ingest/loader.py

* Supported input: .txt, .html
* Metadata block parsed (YAML-like), body extracted
* Chunk by paragraphs, max 1500 chars, fall back to sentence
* Special chunking for legislation/case/journal
* Each chunk holds per-chunk metadata

**Sample code:**

def chunk\_document(doc: dict, chunk\_size: int = 1500, overlap: int = 200) -> list[dict]:  
 ...

## 6. Embedding Model and Vector Storage

Code: embedding/embedder.py

* Default: nomic-ai/nomic-embed-text-v1.5 (768d)
* Model selection by env/config
* Each chunk is embedded, vector stored in embeddings table as pgvector[768]
* Chunk metadata always preserved/attached

Example:

from embedding.embedder import Embedder  
vec = Embedder().embed([text])[0]

## 7. Backend API (fastapi\_app.py) — Module Walkthrough

* **Security:** HTTPBasic with environment credentials
* **User management:** /users, /login, /session endpoints
* **Ingestion:** /ingest/start, /ingest/stop (tracks sessions)
* **Document endpoints:** /documents, /documents/{id}
* **Search:** /search/hybrid, /search/vector, /search/bm25, /search/fts, /search/rag, /search/oci\_rag, /chat (agentic/RAG/conversation)
* **Models:** /models/reranker, /models/ollama, /models/oci\_genai

Include code snippets and explain endpoint flows.

## 8. Streamlit Frontend (app.py) — Module Walkthrough

* **Session management:** Tracks user/logins/session for ingestion/workflow
* **Corpus ingestion & parallel embedding:** Multi-GPU support, session partitioning, embedding\_worker launch logic
* **Hybrid search UI:** collects question, runs search via API, displays rich metadata chunks
* **RAG LLM display:** Calls backend for answer, shows sources/metadata, user feedback built-in

Include how the UI is constructed and how each button/action flows into the backend.

## 9. Gradio Frontend (gradio\_app.py) — Module Walkthrough

* **Tabbed UI:**
  + Hybrid search
  + RAG chat (LLM-based)
  + Full Text Search
  + Conversational RAG
  + Agentic RAG (chain-of-thought)
* **Login:** Simple login\_fn() checks against backend API
* **APIs called for each tab, handling LLM source and model selection (Ollama, OCI GenAI)**
* **Beautiful context cards and citation rendering for each answer**

Show sample UI configurations, event handlers, and user journey for each workflow.

## 10. RAG Pipeline (rag/)

* rag\_pipeline.py: RAG orchestration, provides query() for LLM-based answer from search context
* oci\_rag\_pipeline.py: Oracle GenAI LLM integration, handles OCI authentication and querying

## 11. Embedding Pipeline (embedding/)

* embedder.py: Class-based, model configurable
* EmbeddingWorker.py and session tracking: parallelized embedding

## 12. Ingestion & Parsing (ingest/)

* loader.py: Walking files, parsing metadata, chunking text

## 13. UI Workflows

* Login (Gradio, Streamlit)
* Search flow (tab to API to backend)
* Corpus ingestion: user triggers, status tracking, embedding/reporting
* RAG: input, chunk collection, answer display
* Agentic: step-by-step LLM response with citations

## 14. Security & User Management

* HTTP basic for API
* Users table and login for UI
* Environment credentials

## 15. Extending, Maintenance & Troubleshooting

* How to add new chunkers/parsers
* Adding new embedding models
* Scaling horizontally (background workers, multi-gpu)
* Common issues (env setup, GPU errors, DB problems)
* Health endpoints, system checks

## 16. Appendix: Key Code Snippets and Example Sessions

* Example: Adding a new document, chunking, embedding, storing, searching
* Sample hybrid search workflow and agentic chat example

*This document is intended to be thorough. Insert diagrams, highlight critical code, and give usage examples throughout. Expect 20+ pages upon DOCX conversion with all explanations, code, and illustrations included.*