UniPolicyQA

A RAG-powered Legal & Policy Question Answering Assistant for University Documents

Group Members

21i-0447 **Usman Afzal**
21i-0847 **Babar Shaheen**
21i-0848 **Muhammad Ahmed**
21i-0520 **Munib Akhtar**

Final Project Deliverables — Proposal

Contents

1. Final project title

UniPolicyQA — A RAG-powered Legal & Policy Question Answering Assistant for University Documents

2. Problem statement

Universities publish a wide range of formal documents (academic regulations, attendance rules, plagiarism policies, grading rubrics, accommodation policies) that are long, legalistic, and hard for students and staff to navigate. Misinterpretation causes confusion, incorrect actions, and extra burden on administrative staff.

UniPolicyQA provides an interactive, citation-backed question-answering assistant: users upload university documents (PDFs, DOCX), ask natural-language questions, and receive precise, grounded answers with exact citations (document + page/paragraph). The system uses RAG to locate supporting text and a policy-checker agent to warn when answers are uncertain or conflict with multiple sources.

Goals

- Reduce time-to-answer for students and staff.
- Provide legally-grounded, auditable answers with citations.
- Offer a simple web UI and REST API for integration with university portals.

2.1. Overview

The system has two main flows: document ingestion (offline) and query-time retrieval+generation (online). Ingestion converts documents into chunked vectors with metadata; query-time uses RAG to retrieve relevant chunks and an LLM to produce citation-backed answers. A policy-checker post-processes answers to detect ambiguity or contradictions.

2.2. Written pipeline (step-by-step)

1. Upload

- User uploads PDFs or DOCX through Web UI or API.
- Files are stored in a document storage (S3 or local ‘uploads/‘).

2. Ingestion Agent

- Extract text with PyMuPDF / PDFPlumber / Apache Tika; for scanned docs use Tesseract OCR.
- Extract metadata (filename, upload date, document type, page offsets).
- Normalize text (dehyphenation, normalize whitespace and encodings).

3. Chunking & Metadata

- Split into chunks (e.g., 400–800 tokens) with overlap (100–200 tokens).
- Attach chunk-level metadata: source filename, page number, section header (if detected), paragraph index.

4. Embedding

- Generate embeddings per chunk (OpenAI embeddings or local sentence-transformers).
- Store embeddings + metadata in a vector DB (Chroma / FAISS / Weaviate).

5. Retriever + RAG

- At query time, embed the query and retrieve top-K chunks (K = 5–10).
- Compose retrieval context using a prompt template that instructs the LLM to answer and cite exact chunk metadata.

6. Answer Generation

- Use LLM (OpenAI, Llama, or Mistral) via LangChain or LangGraph agent to generate an answer limited to the provided context.
- Include explicit citation lines: (DocumentName — page X, para Y).

7. Policy Checker Agent

- Post-process answer to detect contradictions across documents and ambiguity (e.g., "may" vs "must").
- Flag legal-advice-like answers and recommend contacting authorities.
- Assign a confidence score.

8. User Response & UI

- Return answer, citations, confidence, and supporting snippet(s).
- Provide "Request Official Review" button to forward to admin if needed.
- Allow user feedback (correct / incorrect).

9. Feedback Loop

- Store feedback to re-rank or fine-tune retrieval and maintain an audit log of queries and responses.

3. Agent design

Implement a small set of agents using LangChain or LangGraph. Each agent is concise and single-responsibility.

3.1. Uploader Agent

- Save uploads, validate file types, trigger ingestion pipeline (enqueue job).
- Tools: file storage, ingestion queue.

3.2. Ingestion Agent

- OCR, parsing, heading detection, text normalization.
- Components: PyMuPDF / PDFPlumber, Tesseract (pytesseract), heading heuristics (font/position or regex).

3.3. Chunker Agent

- Chunk text, attach metadata, compute token counts.
- Parameters: `chunk_size_tokens` (e.g., 512), `overlap` (e.g., 128).

3.4. Embedding Agent

- Call embedding model, batch requests, handle rate limits.
- Models: OpenAI embeddings or `sentence-transformers/all-MiniLM-L6-v2` for local.

3.5. Vector DB Agent

- Insert, update, and query vectors (Chroma/FAISS).
- Provide filtering by metadata (document type, date).

3.6. Retriever Agent

- Retrieve top-k relevant chunks; optional reranker.
- Approach: dense retrieval (vector) ± sparse BM25 for recall.

3.7. Answer Generator Agent (RAG)

- Combine retrieved chunks into a prompt and generate answer with citations.
- Behavior: strict citation policy; if unsupported, respond "I don't know — consult official office".

3.8. Policy Checker Agent

- Analyze policy language for ambiguity and contradictions.
- Detect keywords: "must", "shall", "may", "recommended"; flag risk-levels.

3.9. Audit & Feedback Agent

- Log queries, responses, and feedback; update vector relevance weights.
- Provide admin export and analytics.

4. Prompt templates (examples)

Retriever + Answer Generator Prompt

```
You are a helpful assistant that answers questions about university policies.
Use ONLY the provided snippets below. Each snippet includes metadata in this format:
[DOC: {filename} | page: {page} | paragraph: {p}]
If the answer is not contained in the snippets, say:
"I don't know - please consult the official office" and provide the best next step.
When you answer, include explicit citations inline like: (filename - page X, para Y).

Snippets:
--- SNIPPETS START ---
{retrieved_chunks}
--- SNIPPETS END ---
```

```
Question: {user_question}
Answer:
```

Policy Checker Prompt (example)

```
You are a policy-checker assistant. Given an answer and its supporting snippets,
detect:
 - contradictions across snippets
 - ambiguous modal verbs ("must", "may", "should")
 - whether the answer looks like legal advice
Return a confidence score between 0 and 1 and a short explanation.
```

5. Datasets & data sources

1. **Primary** : University policy documents (student handbook, academic regulations, plagiarism policy, exam regulations, disability services).

6. Evaluation criteria & metrics

- **Accuracy / F1** vs ground-truth answers on a QA eval set.
- **Citation precision**: percent of answers where the cited snippet supports the claim.
- **Hallucination rate**: percent of answers with unsupported facts.
- **User satisfaction**: survey (Likert scale).
- **Latency**: average response time; target < 3s for retrieval + generation (dependent on LLM).

7. Tech stack

- **Orchestration / LLM framework:** LangChain or LangGraph.
- **LLM / embeddings:** OpenAI (gpt-4o / embeddings) or open-source LLMs (Llama 3) + sentence-transformers for embeddings.
- **Vector DB:** Chroma / FAISS / Weaviate.
- **Storage:** S3-compatible bucket or local ‘uploads/‘.
- **Backend:** FastAPI (REST endpoints: upload, query, admin).
- **Frontend:** React + Tailwind.
- **OCR / parsing:** PyMuPDF / PDFPlumber + Tesseract (pytesseract).
- **Containerization:** Docker + docker-compose.
- **Auth (optional):** University SSO (OAuth2) or simple JWT for demo.
- **Testing & CI:** pytest, GitHub Actions (optional).
- **Deployment:** Docker on VPS / Render / AWS ECS.