**Project Proposal: Document-Based QA System using RAG (Retrieval-Augmented Generation)**

**📌 Project Title**

**"RAG-QA: A Document-Aware Question Answering System using Hybrid Retrieval"**

**🎯 Objective**

To build a **smart question-answering system** that can read and understand content from **uploaded documents (PDF, DOCX, TXT)** and accurately **answer user queries** based on the document's content using a hybrid of **TF-IDF + Semantic Search (FAISS + Embeddings)** and an **NLP-powered QA pipeline**.

**💡 Key Features**

* 📤 Upload documents in PDF, DOCX, or TXT formats.
* 🧠 Split and preprocess text into manageable chunks.
* 🔍 Hybrid search combining **TF-IDF relevance scoring** and **FAISS-based vector similarity**.
* 🤖 Use of **transformer-based models** (e.g., roberta-base-squad2) for question answering.
* 💬 Intuitive UI using **Streamlit**.
* 🛠️ Debug mode to inspect TF-IDF scores and chunks.
* 🔄 Cache management to speed up repeated queries and control resource usage.

**🛠️ Tech Stack**

| **Layer** | **Technology Used** |
| --- | --- |
| Frontend | Streamlit |
| Backend | Python |
| Text Extraction | PyMuPDF (fitz), python-docx |
| Chunking | LangChain’s RecursiveCharacterTextSplitter |
| Embeddings | SentenceTransformers (all-MiniLM-L6-v2) |
| Vector Store | FAISS |
| QA Model | Hugging Face Transformers (roberta-base-squad2) |
| Vector Scoring | TF-IDF (Scikit-learn) + FAISS |

**📂 Functional Flow**

1. **Document Upload**  
   User uploads a .pdf, .docx, or .txt file.
2. **Text Extraction**  
   Content is extracted using appropriate libraries depending on file type.
3. **Chunking**  
   Text is split into overlapping chunks using a recursive strategy to maintain context.
4. **Embedding**  
   Each chunk is encoded into a dense vector using Sentence Transformers.
5. **Indexing**  
   Vectors are added to a **FAISS index**. A parallel **TF-IDF matrix** is also created for keyword-based scoring.
6. **User Query**  
   User inputs a question. The system:
   * Scores chunks using **TF-IDF** for relevance.
   * Filters top-k chunks and applies **FAISS similarity** on them.
   * Combines results to form a context window.
7. **Answer Generation**  
   The context and question are passed to the **QA model**, which generates a precise answer.
8. **Output Display**  
   The final answer and supporting chunks are shown, with optional debug info.

**📊 System Architecture**

css

CopyEdit

[Upload Document] --> [Text Extraction]

↓

[Text Chunking]

↓

┌───────────────┴───────────────┐

│ │

[TF-IDF Vectorizer] [Embedding → FAISS Index]

│ │

└──────────────┬─────────────┘

↓

[Top-K Chunk Selection]

↓

[Question Answering Pipeline]

↓

[Answer + Relevant Chunks]

**✅ Strengths**

* Combines traditional (TF-IDF) and modern (semantic search) retrieval.
* Lightweight and fast due to model and index selection.
* Deployable as a **local app or cloud app** (via Streamlit Sharing or HuggingFace Spaces).
* Easy extensibility to other domains like medical/legal/educational document QA.

**🧪 Test Cases**

| **Scenario** | **Expected Outcome** |
| --- | --- |
| Upload valid PDF and ask a question | Answer extracted correctly from the document |
| Upload unsupported file type | Show error message |
| Upload blank or corrupted file | Show warning / error |
| No chunks extracted | Inform user and halt further steps |
| Ask question with no relevant info | Returns “No clear answer found in document.” |

**📦 Future Enhancements**

* ✅ Support for OCR on scanned PDFs.
* 🔐 Add user login & access control.
* 📁 Support multiple documents for multi-file querying.
* 🌐 Deploy on cloud platforms (e.g., Hugging Face Spaces, Streamlit Cloud).
* 🧠 Add memory and conversational context across multiple queries.