Infosys Virtual Internship 6.0

Project Title - **SwiftVisaAI- Based Visa Eligibility Screening Agent**

Student Name – Aayush Shrivastava

Mentor Name – Siddarth sir

Sr. No.	Content
1	Introduction
2	System Architecture Overview
3	Dataset Preparation
3.1	**Policy Document Collection**
3.2	Text Extraction
3.3	Text Cleaning and Preprocessing
4	**Embedding Generation**
4.1	Embedding Model Used
4.2	Query Embedding
5	**Vector Database Implementation (FAISS)**
5.1	Data Stored in FAISS
5.2	Index Construction Process
6	**Query Processing and Retrieval**
6.1	User Query Handling
6.2	Top-K Document Retrieval
7	**Retrieval-Augmented Generation (RAG) Pipeline**
7.1	RAG Workflow
7.2	Prompt Design and Context Injection
7.3	LLM-Based Eligibility Reasoning
8	**Confidence Scoring and Explainability**
9	**Streamlit Application**
10	**Results and Observations**
11	**Tools and Technologies Used**

1. Introduction -

Visa application processes involve strict eligibility rules, complex documentation, and legal language that is often difficult for applicants to understand. Many visa rejections occur due to incomplete knowledge of requirements rather than actual ineligibility.

SwiftVisaAI is an AI-based Visa Eligibility Screening Agent designed to provide preliminary visa eligibility assessment before applicants proceed with official submissions. The system analyzes user-provided details and visa-related questions by referencing official visa policy documents.

The project uses **Retrieval-Augmented Generation (RAG)**, where relevant policy documents are retrieved using semantic search and then passed to a Large Language Model (LLM) to generate grounded and explainable responses. This ensures that answers are based on real documents rather than assumptions.

2. System Architecture Overview

SwiftVisaAI follows a modular architecture consisting of document preprocessing, vector indexing, semantic retrieval, LLM-based reasoning, and a user-facing interface.

The system architecture includes:

- Document processing and embedding generation
- FAISS-based vector database for semantic search
- RAG pipeline for controlled LLM responses
- Streamlit-based frontend for user interaction

This architecture ensures scalability, transparency, and reliable AI-driven decision support.

3. Dataset Preparation

Dataset preparation is a critical step in ensuring accurate retrieval and eligibility assessment. SwiftVisaAI uses official visa policy documents as its primary data source.

The dataset is processed and structured to enable efficient semantic retrieval and context-aware reasoning.

3.1 Policy Document Collection

Official visa policy documents were collected from authoritative sources such as government immigration portals. These documents include:

- Visa eligibility criteria
- Financial requirements
- Age and travel purpose rules
- Supporting document guidelines

The collected documents serve as the factual foundation of the system. In my Case I have collected the documents from countries like USA , UK, Schengen , and also Ireland etc .

3.2 Text Extraction

The collected PDF documents were processed to extract raw textual content. Text extraction ensures that all relevant policy information is available in machine-readable format for further processing.

```python
# -- READ PDF --
def read_pdf(path):
    text = ""
    reader = PyPDF2.PdfReader(path)
    for page in reader.pages:
        content = page.extract_text()
        if content:
            text += " " + content
    return text.strip()
```

Figure 1 : extracting the data

3.3 Text Cleaning and Preprocessing

Extracted text was cleaned to improve quality and retrieval accuracy. This includes removal of headers and footers, normalization of whitespace, and elimination of irrelevant or empty lines.

Text cleaning helps reduce noise in embeddings and improves the relevance of retrieved document chunks during semantic search.

```python
def clean_text(text):
    text = re.sub(r'\s+', ' ', text)
    return text.strip()
```

4. Embedding Generation

To enable semantic search, textual data must be converted into numerical vector representations. SwiftVisaAI uses embeddings to represent both policy documents and user queries within the same semantic space.

This transformation allows similarity-based comparison between user queries and policy documents, which is essential for effective retrieval.

4.1 Embedding Model Used

The project uses the **SentenceTransformer model all-MiniLM-L6-v2**, which is optimized for semantic similarity tasks. The model is lightweight and provides fast inference, making it suitable for real-time applications.

It generates dense vector representations that capture the contextual meaning of policy text and user queries.

4.2 Query Embedding

User queries, enriched with applicant details such as age and travel purpose, are converted into embeddings using the same model. This ensures consistency between document vectors and query vectors.

Using the same embedding space allows accurate similarity matching during FAISS-based retrieval.

```python
# -- CHUNKING LOGIC --
def chunk_text(text, size=CHUNK_SIZE, overlap=OVERLAP):
    words = text.split()
    chunks = []
    start = 0

    while start < len(words):
        chunk = " ".join(words[start:start + size])
        chunks.append(chunk)
        start += size - overlap

    return chunks
```

5. Vector Database Implementation (FAISS)

FAISS (Facebook AI Similarity Search) is used as the vector database to store and retrieve embeddings efficiently. It is optimized for high-speed similarity search on large vector datasets.

FAISS enables SwiftVisaAI to retrieve relevant policy documents quickly, even as the dataset grows.

5.1 Data Stored in FAISS

FAISS stores vector embeddings of policy document chunks. Each vector represents a small portion of a policy document.

Associated metadata such as text content, PDF source name, and chunk identifiers is stored separately in JSON format for transparency and traceability.

5.2 Index Construction Process

The FAISS index is built using the generated embeddings and stored on disk. This index is loaded during application runtime.

Persisting the index avoids repeated computation and significantly improves system performance during user queries.

```python
# Load your final embeddings JSON
EMB_FILE = "chunk_embeddings.json"

with open(EMB_FILE, "r", encoding="utf-8") as f:
    data = json.load(f)

# Extract vectors + metadata
vectors = []
metadata = []

for item in data:
    vectors.append(item["embedding"])
    metadata.append({
        "pdf_name": item["pdf_name"],
        "chunk_id": item["chunk_id"],
        "text": item["text"]
    })

# Convert to numpy float32
vectors_np = np.array(vectors).astype("float32")

# Vector dimension
d = vectors_np.shape[1]

# Create FAISS index (L2 similarity)
index = faiss.IndexFlatL2(d)
```

6. Query Processing and Retrieval

User queries are processed and matched against stored policy document embeddings to retrieve the most relevant information. This step bridges user input and policy knowledge.

Accurate query processing is essential for ensuring meaningful and contextually relevant eligibility assessments.

6.1 User Query Handling

User input includes personal details such as age and nationality along with a visa-related question. These details are combined to form a structured query. This enrichment helps the system better understand the applicant's context during retrieval and reasoning.

```
question = input("Enter your visa question: ")

chunks = retrieve_chunks(question)
model_answer = ask_groq(question, chunks)
```

6.2 Top-K Document Retrieval

FAISS retrieves the top-k most relevant document chunks based on semantic similarity scores. These chunks represent the most applicable policy sections.

The retrieved documents act as supporting evidence for the eligibility assessment generated by the LLM.

```
def retrieve_chunks(query, k=5):
    vec = embed_text(query)
    _, ids = index.search(np.array([vec]), k)
    return [metadata[cid] for cid in ids[0] if cid != -1]
```

7. Retrieval-Augmented Generation (RAG) Pipeline

The RAG pipeline ensures that LLM responses are grounded in retrieved policy documents. This reduces hallucinations and improves reliability.

By combining retrieval and generation, the system balances accuracy with natural language explanation.

7.1 RAG Workflow

The RAG workflow involves query embedding, semantic retrieval from FAISS, context injection into the LLM prompt, and structured response generation. Each step is carefully controlled to ensure consistency and explainability.

7.2 Prompt Design and Context Injection

A strict prompt format is used to instruct the LLM to answer **only using the provided context**. This prevents unsupported assumptions. Prompt control plays a key role in maintaining factual correctness in sensitive domains like visa eligibility.

```
    prompt = f"""
You are a visa eligibility officer.
Answer ONLY using the PDF context provided.
Do NOT add any information that is not present in the context.

Question:
{question}

Context:
{ctx}

Return EXACTLY the following format:

Eligibility: Yes / No / Partial
Final Answer: (2-3 lines summary ONLY)
Explanation:
- Provide EXACTLY 3 to 5 bullet points.
- Each bullet MUST be unique.
- No repeating, rephrasing, or expanding the same idea.
- DO NOT show chunk IDs or chunk numbers.
Confidence: (0 to 1)
"""
```

7.3 LLM-Based Eligibility Reasoning

I choose Groq LLM API key for my model . The LLM analyzes retrieved policy content and produces a structured output including eligibility status, explanation, and confidence score. This reasoning mimics real-world decision support while remaining grounded in official documentation

8. Confidence Scoring and Explainability

Each eligibility assessment includes a confidence score indicating the strength of the decision. The score reflects the relevance of retrieved documents and contextual alignment.

Displaying retrieved policy documents improves explainability and user trust in the system's decisions.

```
# Extract confidence
confidence = extract_confidence(model_answer)
answer_text = model_answer.lower()

if "eligibility: yes" in answer_text:
    confidence = min(0.9, confidence)
elif "eligibility: no" in answer_text:
    confidence = min(0.6, confidence)
else:
    confidence = 0.3

# FIX: overwrite confidence inside the model's text
import re
model_answer = re.sub(r"Confidence:\s*\d+(\.\d+)?", f"Confidence: {confidence}", model_answer)

print("\nResponse:\n")
print(model_answer)
```

9. Streamlit Application

The final system is deployed as a **Streamlit web application** to provide an interactive user experience. The interface is designed to be simple and intuitive.

It supports user-friendly form input, real-time eligibility assessment, confidence score visualization, and application history tracking.

10. Results and Observations

The SwiftVisaAI system was evaluated using multiple realistic visa application scenarios by varying applicant details such as age, nationality, and travel purpose. The results demonstrate that the system provides accurate, structured, and policy-grounded eligibility guidance.

Key observations include:

- Semantic retrieval using **FAISS** returned more relevant policy documents than keyword-based search.
- The **RAG pipeline** significantly reduced hallucinated responses by grounding answers in retrieved policy text.
- Output responses were **well-structured**, including eligibility status, explanation, and confidence score.
- Retrieved policy excerpts improved **transparency and user trust**.

- The system performed effectively as a **preliminary visa screening and decision-support tool**.

11. Tools and Technologies Used

The SwiftVisaAI project was developed using a combination of programming tools, libraries, and AI services to support document processing, semantic search, and interactive deployment. Each tool was selected based on performance, reliability, and ease of integration.

Tools and technologies used include:

- **Python** – Core programming language used for implementing the entire system.
- **Visual Studio Code (VS Code)** – Used as the primary code editor for development and debugging.
- **SentenceTransformers** – Used for generating semantic embeddings of visa policy documents and user queries.
- **FAISS (Facebook AI Similarity Search)** – Used as a vector database for fast similarity-based document retrieval.
- **Groq API** – Used to access a Large Language Model (LLM) for eligibility reasoning and structured response generation.
- **Streamlit** – Used to build an interactive and user-friendly web application interface.
- **PyPDF2** – Used for extracting text content from visa policy PDF documents.
- **JSON** – Used for storing metadata and maintaining structured information.
- **dotenv (.env)** – Used for secure management of API keys and environment variables.

CONCLUSION

SwiftVisaAI demonstrates the effective use of semantic search and Retrieval-Augmented Generation (RAG) for visa eligibility screening. By grounding Large Language Model responses in official policy documents, the system provides accurate and explainable eligibility assessments. The use of FAISS enables efficient document retrieval, while the Streamlit interface ensures user-friendly interaction. Overall, SwiftVisaAI functions as a reliable preliminary decision-support tool that helps applicants better understand visa requirements before applying.

THANK YOU