

# Objetivo

- Implementar un sistema multiagente usando LLMs
- Construir un RAG simple con base de conocimiento local
- Usar Gemini (GCP) como modelo de lenguaje
- Exponer la solución mediante una API REST
- Aplicar buenas prácticas de backend, Git y Docker

# Contexto

El asistente debe responder únicamente las preguntas que se le realicen, basándose en la información proporcionada por el usuario.

Su función principal es ayudar con los Ejes del RICE, entregando orientación clara y pertinente según lo que se consulte.

En caso de que durante la conversación se detecte una situación relevante, el asistente puede indagar más profundamente en el caso cuando sea necesario, con el objetivo de comprender mejor el contexto y determinar internamente qué acción o procedimiento corresponde aplicar, sin mencionarlo explícitamente al usuario.

## Sistema Multiagente

Implementar al menos 3 agentes, orquestados con LangGraph:

Agente	Responsabilidad
Router Agent	Decide si la pregunta requiere RAG o respuesta directa
RAG Agent	Recupera contexto desde la knowledge base
Answer Agent	Genera y formatea la respuesta final

Requisitos:

- Uso real de LangGraph
- Estado compartido entre agentes
- El flujo no debe ser secuencial hardcodeado

## RAG Local con Knowledge Base

El sistema debe utilizar una carpeta como knowledge base (kb):

```
kb/  
├── doc1.pdf  
├── doc2.txt  
└── doc3.md
```

Requisitos del RAG:

- Carga automática de documentos desde la carpeta **kb**
- Chunking básico
- Generación de embeddings
- Búsqueda semántica por similitud
- Inyección del contexto recuperado en el prompt

Tecnologías permitidas:

- LangChain o implementación propia
- FAISS, Chroma o similar
- Embeddings con Gemini o SentenceTransformers

## LLM – Gemini

- Uso de Gemini (API de GCP) para:
  - Decisiones del Router Agent
  - Generación de la respuesta final
- Uso responsable de tokens

## Backend / API REST

Exponer el sistema mediante una API REST.

Endpoint

POST /query

Request

```
{  
  "question": "¿Qué es el protocolo de actuación?"  
}
```

Response

```
{  
  "answer": "Texto de la respuesta",  
  "sources": ["doc1.pdf"],  
  "agent_used": "rag_agent"  
}
```

Requisitos:

- FastAPI
- Validación del request
- Manejo básico de errores
- Uso de variables de entorno

# Buenas Prácticas de Git (Obligatorio)

El repositorio debe cumplir con las siguientes prácticas:

- Uso de Git desde el inicio del proyecto
- Commits claros y descriptivos (ej: `feat: add rag agent`)
- No subir secretos ni credenciales
- Uso obligatorio de `.gitignore`

`.gitignore` mínimo esperado

```
# Python
__pycache__/
*.pyc
*.pyo
*.pyd
.env
.venv/
venv/
```

```
# IDE
.vscode/
.idea/
```

```
# OS
.DS_Store
```

```
# Tests / coverage
.coverage
htmlcov/
```

```
# Docker
*.log
```

# Docker y Replicabilidad

El proyecto debe ser replicable y ejecutable con Docker.

Requisitos:

- Incluir un [Dockerfile](#)

El sistema debe levantarse con:

```
docker build -t ai-rag-app .
```

```
docker run -p 8000:8000 --env-file .env ai-rag-app
```

- La API debe quedar disponible en:  
`http://localhost:8000`

## Buenas Prácticas Generales

- Archivo [.env.example](#)
- [README.md](#) que incluya:
  - Descripción del sistema
  - Arquitectura (diagrama simple)
  - Pasos para ejecutar el proyecto (local y Docker)
- Código modular y legible
- Separación por capas