



Get Talent - Challenge Semana 4

Construcción de un sistema RAG con APIs

Descripción

Se requiere implementar una API utilizando FastAPI que permita construir un sistema de **búsqueda semántica y generación de respuestas contextuales** basado en una arquitectura **RAG (Retrieval Augmented Generation)**. Este sistema trabajará con documentos de texto cargados por los usuarios, procesados para extraer embeddings y realizar búsquedas inteligentes.

La API manejará datos en memoria o almacenados localmente (sin necesidad de base de datos en esta versión) y utilizará servicios externos (APIs) para generar embeddings y respuestas.

Objetivos

Implementar los siguientes endpoints que permitan interactuar con los documentos y preguntas:



1. POST /upload

- **Descripción:** Carga un nuevo documento en el sistema.

```
● ● ●

// Body request
{
    "title": "Historia de Sol y Luna",
    "content": "Sol y Luna eran dos pequeños gatitos que vivían en una casa..."
}

// Body response
{
    "message": "Document uploaded successfully",
    "document_id": "12345"
}
```

2. POST /generate-embeddings

- **Descripción:** Genera embeddings para un documento específico o para todos los documentos almacenados.

```
● ● ●

// Body request
{
    "document_id": "12345"
}

// Body response
{
    "message": "Embeddings generated successfully for document 12345"
}
```

3. POST /search

- **Descripción:** Busca los documentos más relevantes basados en una consulta.



```
// Body request
{
    "query": "¿Qué aprendió Sol durante la tormenta?"
}

// Body response
{
    "results": [
        {
            "document_id": "12345",
            "title": "Historia de Sol y Luna",
            "content_snippet": "Sol aprendió a valorar la calma..."
        }
    ]
}
```

4. POST /ask

- **Descripción:** Genera una respuesta a una pregunta utilizando los documentos relevantes.

```
// Body request
{
    "question": "¿Qué aprendió Sol durante la tormenta?"
}

// Body response
{
    "answer": "Sol aprendió a valorar la calma y a no temer las tormentas."
}
```



Descripción de los campos

Datos de Entrada (Input)

1. POST /upload

- a. **title:** Título del documento que se va a cargar (cadena de texto, obligatorio).
- b. **content:** Contenido del documento (cadena de texto, obligatorio).

2. POST /generate-embeddings

- a. **document_id:** Identificador único del documento para el cual se generarán los embeddings (cadena alfanumérica, obligatorio).

3. POST /search

- a. **query:** Consulta del usuario en lenguaje natural (cadena de texto, obligatorio).

4. POST /ask

- a. **question:** Pregunta formulada por el usuario en lenguaje natural (cadena de texto, obligatorio).

Datos de Salida (Output)

1. POST /upload

- a. **message:** Mensaje indicando que el documento se cargó correctamente (cadena de texto).
- b. **document_id:** Identificador único del documento cargado (cadena alfanumérica).

2. POST /generate-embeddings

- a. **message:** Mensaje indicando que los embeddings se generaron correctamente (cadena de texto).



- b. **document_id:** Identificador del documento para el cual se generaron los embeddings (cadena alfanumérica).

3. POST /search

- a. **results:** Lista de documentos relevantes para la consulta.
 - i. **document_id:** Identificador único del documento relevante (cadena alfanumérica).
 - ii. **title:** Título del documento relevante (cadena de texto).
 - iii. **content_snippet:** Fragmento del contenido relevante del documento (cadena de texto).
 - iv. **similarity_score:** Puntuación de relevancia con respecto a la consulta (número flotante entre 0 y 1).

4. POST /ask

- a. **question:** Pregunta formulada por el usuario (cadena de texto).
- b. **answer:** Respuesta generada basada en los documentos relevantes (cadena de texto).

Requisitos Técnicos

- **FastAPI:** Implementar la API para los endpoints especificados.
- **Embeddings API:** Utilizar un servicio externo (como Cohere o Hugging Face) para generar embeddings.
- **Modelo de Lenguaje:** Usar una API de un LLM (como Cohere para generar respuestas contextuales).
- **Persistencia:** Manejar los documentos y embeddings en memoria o en almacenamiento local (e.g. utilizando un vector store como ChromaDB o FAISS).
- **Validaciones y Errores:**



- Validar datos de entrada (por ejemplo, que el título y el contenido no estén vacíos).
- Manejar errores como documentos no encontrados (404) o problemas con servicios externos (500).

Todas las implementaciones deben incorporar prácticas básicas de **IA Responsable** en los endpoints /search y /ask. Esto incluye:

1. Grounding obligatorio (NO inventar información)

- El endpoint /ask **solo puede responder usando contenido presente en los documentos relevantes recuperados desde /search.**
- Si no hay contexto suficiente, la API debe devolver: "**answer": "Nouento con información suficiente para responder a esta consulta."**"

2. Protección de datos sensibles

- No incluir ni devolver información sensible que no esté explícitamente en los documentos cargados.
- No imprimir documentos completos en logs.
- En caso de error, los mensajes deben ser genéricos (no exponer claves API ni paths internos).

3. Transparencia

El endpoint /ask debe devolver SIEMPRE:

- el fragmento de contexto utilizado
- la similitud obtenida
- un campo adicional "grounded": true/false indicando si la respuesta se basó en contexto real



Ejemplo:

```
{  
    "answer": "Sol aprendió a  
    valorar la calma. ",  
    "context_used": "Sol aprendió a ca  
    .00. similarity_score: 0.89  
    "grounded": true  
}
```

4. Manejo responsable de errores

- Si el servicio de embeddings o el LLM falla, la API debe responder con un mensaje de error controlado:
"error": "El servicio externo no pudo procesar la solicitud en este momento."

5. Equidad y no discriminación

- La API no debe generar respuestas que incluyan estereotipos, insultos o juicios subjetivos.
- En caso de detectar lenguaje inapropiado en la pregunta, responder con seguridad:
"answer": "No puedo responder a este tipo de consultas."

6. Control humano

- El sistema debe loggear (sin datos sensibles) cada paso del flujo: carga → embeddings → búsqueda → respuesta.
- Toda decisión automatizada debe poder rastrearse.