Documentación Challenge final - Pi Consulting

Proyecto: Términos y Condiciones RAG

Escrito: Tomás Bertotto

Descripción del problema y solución del proyecto

Desarrollar un producto de software requiere de numerosas decisiones, una de ellas es el uso de otros productos de software (librerías, API's, etc) para resolver determinados problemas. Estos productos tienen ciertas normas de uso, pudiendo ser: uso gratuito con fines no comerciales, pagos en su totalidad, gratuitos bajo determinadas condiciones (ej: no utilizar LLM's), entre otras. Dichas normas se especifican normalmente en los términos y condiciones del producto y dada la enorme extensión de este tipo de documentos se complejiza la búsqueda de información clave y de interés para quien desee usar ese producto o similares. Frente a esto, este proyecto busca no sólo mejorar la búsqueda de secciones que una persona considere de interés sino también posibilitarle relacionar e investigar términos y condiciones de distintos productos.

Decisiones de diseño

Se decidió modularizar acorde a cada componente central de la arquitectura RAG:

- main: Gestor principal de responsabilidades de otros módulos y endpoint para la comunicación
- EmbeddingService: Maneja la base de datos vectorial, gestiona como se produce los vectores de cada documento así como también los documentos en sí
- LLMService: Administra todo lo relacionado a los modelos utilizados
- TermsDocument: Clase básica que determina qué atributos tiene un documento nuevo
- TermsHandler: Administra la instanciación y administración de los documentos
- index: Página web básica para comunicarse con el servidor

Interacción del usuario y desarrollo de cada modulo

Para visualizar naturalmente la responsabilidad y funcionalidad de cada módulo es útil ver la secuencialidad de eventos que ocurren frente al accionar del usuario. Este último tiene dos formas de interactuar: *haciendo una pregunta* o *agregando nuevos términos y condiciones*.

Agregar términos y condiciones

El usuario cuenta con dos campos en la página web que debe completar: un nombre de producto y los términos y condiciones de éste. Una vez que se estos campos son completados se conecta via HTTP al servidor local (socket: localhost:8000/) y da lugar a la siguiente secuencialidad de eventos:

1. El endpoint en main.py está articulado usando **FastAPI**, en este caso la request del usuario se conecta con: "/upload". De la request del usuario se extraen los campos mencionados anteriormente y se **validan**.
2. Desde main.py se delega la responsabilidad de detectar el **dominio** del documento(i.e. si se trata de un producto de software, financiero, cultural, medicinal, educacional, etc) al módulo de LLM. Aquí se utiliza el modelo "*command-r-plus-08-2024*" de **Cohere** para extraer la información. Indicar un dominio **mejorará luego búsquedas futuras del usuario**
3. Una vez recopilada la información anterior se procede a crear el documento (vía TermsHandler). Importante: en esta instancia no se guarda el documento en la base de datos. El módulo mencionado garantiza crear un objeto en **memoria principal** que contiene la información relevante de los términos y condiciones, y además, a modo de backup, se crea y guarda el documento de manera local con un **identificador único**
4. A partir del nuevo documento utiliza el módulo EmbeddingService.py mediante el cual se extraen todos los campos del objeto. Es importante remarcar que se optó por dividir el contenido del documento en segmentos/chunks con la herramienta *RecursiveCharacterTextSplitter* de **Langchain**. La idea es guardar cada segmento en la base de datos con: **su contenido, el embedding generado para ese segmento, y su metadata** (a qué documento pertenece, dominio, producto, id). Se consideró que este proceso puede conseguir **mejores resultados** luego frente a una consulta del usuario y posterior búsqueda de información relevante a diferencia de haber generado un sólo embedding de gran dimensión para cada documento nuevo.
5. Se le comunica al usuario el éxito o no del proceso

Realizar una pregunta

1. Una vez que el usuario completa el campo vinculado a realizar preguntas el endpoint en main.py se vincula con: “/ask”. Nuevamente se valida el campo de texto ingresado
2. Se procede a detectar a qué dominio parece estar vinculado la pregunta para luego mejorar la búsqueda de información relevante
3. Se llama nuevamente al módulo LLMService para **determinar si la pregunta espera resultados que involucren una entidad, muchas entidades o no se especifica**. El motivo detrás de esta decisión es poder ampliar aún más los documentos similares que se recuperarán ya que puede ocurrir que más de un segmento importante corresponda a un mismo documento lo cual **acotaría esa búsqueda de múltiples resultados que el usuario espera**. Si se trata de múltiples entidades o directamente no especifica entonces se buscan hasta 30 segmentos, en cambio si se trata de una sola entidad se buscan 5 como máximo
4. Como fue mencionado anteriormente tanto la pregunta como los documentos son asignados a un dominio lo que permite mejorar la búsqueda en la base de datos vectorial. No obstante, si no llegaran a ser compatibles los dominios y no se consiguen resultados se realizará una consulta sin tener en cuenta este campo. El motivo de esto es **no tratar al dominio como un filtro sino como una mejora** que en determinados casos puede ser muy útil pero jamás debe impedir una búsqueda.
5. Luego de la consulta a la base de datos se determinar si hay suficiente información para proporcionarle al modelo para luego elaborar una respuesta.
6. Si los segmentos recuperados llegaran a ser suficientes para proporcionarle un buen contexto al model entonces se procede a no sólo a concatenarlos en una única cadena de texto sino también **rotular/identificar** a qué producto pertenece cada uno. Esto es posible ya que esta información fue guardada previamente en la base de datos (**metadata**) y le permitirá al modelo **tener un mejor contexto general**. Importante: en el caso de haber identificado una respuesta que requiera múltiples entidades no se concatenan hasta treinta segmentos sino que se seleccionan tres resultados de productos diferentes
7. Se llama al método “answer_question” del módulo LLMService.py el cual recibe la pregunta del usuario y el contexto. Internamente se generan los prompts necesarios para el modelo de Cohere y se devuelve en forma de texto la respuesta del agente

---

Módulos de interés usados

- FastAPI: backend/endpoints
- uvicorn: servidor
- Cohere: modelos utilizados
- ChromaDB: base de datos vectorial persistente
- uuid: Generar identificadores únicos de cada nuevo documento guardado

---

Documentos de testeo

Para facilitar y puntualizar el testeo no se usaron términos y condiciones reales sino textos generados con ***Gemini 3.0*** para puntualizar aspectos específicos (términos y condiciones de productos gratuitos, pagos, gratuito bajo uso no-comercial, entre otros)