

Universidad Internacional de La Rioja

Facultad de Ciencias de la Salud

Máster Universitario en Bioinformática

Desarrollo de un algoritmo predictivo para la personalización de estilos de vida basado en genómica y biométrica

|  |  |
| --- | --- |
| Trabajo fin de Estudio presentado por: | Maximiliano Hernández Sahuquillo y David Fernández Martín |
| Tipo de trabajo: | 2 |
| Director/a: | Dr. José Arturo Mora Soto |
| Fecha: | 14 de marzo de 2024 |

Resumen

El presente TFM tiene como objetivo desarrollar un algoritmo predictivo para la personalización de estilos de vida basado en datos genómicos y biométricos. El algoritmo estará diseñado para generar recomendaciones personalizadas de estilo de vida para cada individuo, teniendo en cuenta su predisposición genética a ciertas enfermedades, su respuesta al ejercicio y su respuesta a diferentes tipos de dieta.

**Palabras clave:** estilo de vida, genómica, biométrica

Abstract

This TFM aims to develop a predictive algorithm for lifestyle personalisation based on genomic and biometric data. The algorithm will be designed to generate personalised lifestyle recommendations for each individual, taking into account their genetic predisposition to certain diseases, their response to exercise and their response to different types of diet.

**Keywords**: lifestyle, genomics, biometrics

Índice de contenidos

[1. Introducción 9](#_Toc166182566)

[1.1. Justificación 9](#_Toc166182567)

[1.2. Planteamiento del problema 11](#_Toc166182568)

[1.3. Objetivos 11](#_Toc166182569)

[2. Marco teórico 12](#_Toc166182570)

[3. Metodología 14](#_Toc166182571)

[3.1. Recolección de datos y preparación del modelo 14](#_Toc166182572)

[3.2. Preprocesamiento y limpieza de datos 14](#_Toc166182573)

[3.3. Integración en la base de conocimientos 15](#_Toc166182574)

[Ilustración 1. VECTORIZACIÓN CON OLLAMA Y ALMACENAMIENTO EN FAISS 15](#_Toc166182575)

[3.4. Desarrollo del Chatbot basado en RAG 17](#_Toc166182576)

[3.5. Arquitectura y funcionamiento de la API 17](#_Toc166182577)

[3.6. Evaluación y refinamiento del modelo 20](#_Toc166182578)

[3.7. Desarrollo del prototipo de la web 20](#_Toc166182579)

[3.7.1. Consideraciones para el desarrollo de la web 20](#_Toc166182580)

[3.7.2. Pruebas de validación 25](#_Toc166182581)

[3.8. Esquema y funcionamiento formal del RAG 28](#_Toc166182582)

[3.8.1. Funcionamiento de main.py 29](#_Toc166182583)

[4. Resultados y discusión 32](#_Toc166182584)

[5. Conclusiones 33](#_Toc166182585)

[Anexo A. Código diagrama de flujo v3. 35](#_Toc166182586)

[graph TD 35](#_Toc166182587)

[A[Inicio: Usuario accede a la aplicación Streamlit] --> B[Usuario ingresa pregunta en español] 35](#_Toc166182588)

[B --> C[Pregunta enviada a la API FastAPI] 35](#_Toc166182589)

[C --> D[Traducción de la pregunta al inglés usando googletrans] 35](#_Toc166182590)

[D --> E[Agente personalizado recibe la pregunta traducida] 35](#_Toc166182591)

[E --> F{Tipo de pregunta} 35](#_Toc166182592)

[F --> |Pregunta general| G[Búsqueda en Wikipedia usando WikipediaQueryRun] 35](#_Toc166182593)

[F --> |Pregunta médica específica| H[Búsqueda en documentos médicos utilizando búsqueda personalizada] 35](#_Toc166182594)

[F --> |Pregunta sobre estudios científicos| I[Búsqueda en Arxiv usando ArxivQueryRun] 35](#_Toc166182595)

[G --> J[Resultados de Wikipedia] 35](#_Toc166182596)

[H --> K[Resultados de documentos médicos] 35](#_Toc166182597)

[I --> L[Resultados de Arxiv] 35](#_Toc166182598)

[J --> M[Generación de respuesta en inglés utilizando el modelo GPT-J y resultados combinados] 35](#_Toc166182599)

[K --> M 35](#_Toc166182600)

[L --> M 35](#_Toc166182601)

[M --> N[Traducción de la respuesta al español usando googletrans] 35](#_Toc166182602)

[N --> O[Respuesta traducida al español devuelta a la interfaz de usuario de Streamlit] 35](#_Toc166182603)

[O --> P[Respuesta en español mostrada al usuario] 35](#_Toc166182604)

[P --> Q[Expansión: Detalles adicionales y contexto utilizado] 35](#_Toc166182605)

[Anexo B. Tratamiento, segregación de documentos y vectorización (*embbeding*) de los documentos. 36](#_Toc166182606)

[@app.on\_event("startup") 36](#_Toc166182607)

[async def startup\_event(): 36](#_Toc166182608)

[embeddings = OllamaEmbeddings() 36](#_Toc166182609)

[pdf\_loader = PyPDFDirectoryLoader("./ruta/al/directorio/pdf") 36](#_Toc166182610)

[pdf\_docs = pdf\_loader.load() 36](#_Toc166182611)

[csv\_loader = CSVLoader("./ruta/al/archivo.csv") 36](#_Toc166182612)

[csv\_docs = csv\_loader.load() 36](#_Toc166182613)

[json\_loader = JSONLoader("./ruta/al/archivo.json") 36](#_Toc166182614)

[json\_docs = json\_loader.load() 36](#_Toc166182615)

[docs = pdf\_docs + csv\_docs + json\_docs 36](#_Toc166182616)

[text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=1000, chunk\_overlap=200) 36](#_Toc166182617)

[final\_documents = text\_splitter.split\_documents(docs) 36](#_Toc166182618)

[vectors = FAISS.from\_documents(final\_documents, embeddings) 36](#_Toc166182619)

[app.state.vectors = vectors 36](#_Toc166182620)

[wikipedia\_api\_wrapper = WikipediaAPIWrapper(top\_k\_results=1, doc\_content\_chars\_max=200) 36](#_Toc166182621)

[app.state.wikipedia\_tool = WikipediaQueryRun(api\_wrapper=wikipedia\_api\_wrapper) 36](#_Toc166182622)

[arxiv\_api\_wrapper = ArxivAPIWrapper(top\_k\_results=1, doc\_content\_chars\_max=200) 36](#_Toc166182623)

[app.state.arxiv\_tool = ArxivQueryRun(api\_wrapper=arxiv\_api\_wrapper) 36](#_Toc166182624)

[Anexo C. ¿Para que sirve nuestro RAG? 40](#_Toc166182625)

[Referencias bibliográficas 42](#_Toc166182626)

Índice de figuras

**No se encontraron entradas de tabla de contenido.**

Índice de tablas

[Tabla 1. “Tablas” del menú de estilos 8](#_Toc20304757)

# Introducción

El presente Trabajo Fin de Máster (TFM) pretende desarrollar y poner en práctica un servicio web basado en datos biométricos, genéticos y distintos aspectos relacionados con la salud los cuales van a ser procesados por varios LLM (Large Language Model) para su correcta implementación. El motivo que condujo a esta idea es el de aportar un sistema que personalice los estilos de vida de cada paciente.

En las próximas páginas se introducirá este trabajo, primero se abordarán los motivos que llevaron a escoger esta temática, seguido de la introducción del problema a resolver y, por último, se plantearán el objetivo principal y objetivos secundarios a alcanzar.

## Justificación

En los últimos años, el campo de la inteligencia artificial ha sufrido un avance vertiginoso, particularmente con respecto al procesamiento del lenguaje natural (NLP) y el desarrollo de modelos de lenguaje de gran tamaño (LLM)(1). Estos consiguieron gran notoriedad y presencia en el sector médico, gracias a sus múltiples aplicaciones, desde el procesamiento de bastas cantidades de datos hasta poder interpretar anotaciones y datos clínicos para generar reportes adecuados.

En estudios recientes acerca del rendimiento de bots de chat basados en LLMs al responder preguntas médicas, han demostrado una impresionante cantidad de aciertos con respecto al conocimiento sobre bases de datos médicas. Esto es debido ya que en primer lugar, poseen suficientes conocimientos médicos para aplicaciones en medicina, lo que podría provocar un cambio de paradigma en la prestación de asistencia sanitario (2). Pero persiste una considerable inquietud sobre la preocupación de que los LLM puedan ser manipulados maliciosamente para generar contenidos dañinos o engañosos (3).

Permitiendo que el proceso de cribado de la literatura (que se vuelve excesivamente laboriosa debido a que pueden existir miles de artículos con respecto a una patología o un gen) se vuelva más rápido y eficaz, permitiendo resumir la literatura científica y dejando a los investigadores y médicos la información de manera accesible con respecto al esfuerzo realizado (4).

A su vez en otros campos relacionados con la salud como la nutrición, numerosos estudios han contribuido al desarrollo de diversos métodos de recomendación de alimentos (5,6). Estos enfoques mediante el uso de LLM no sólo simplifican el proceso de elección de alimentos, sino que también promueven la adopción de hábitos sostenibles y saludables entre los usuarios. El poder de estas recomendaciones reside en su capacidad para integrar la ciencia de la nutrición en las necesidades de la población o de subpoblaciones, ofreciendo una guía intuitiva y práctica para las personas que desean tomar decisiones dietéticas informadas.

No obstante, los servicios convencionales de recomendación de alimentos orientados a la nutrición han encontrado limitaciones a la hora de comprender exhaustivamente la intrincada interacción entre la salud y el bienestar de las personas, que abarca parámetros fisiológicos, actividad física y calidad del sueño, y de alinearlos con necesidades nutricionales personalizadas.

Concretamente, estos sistemas suelen tener dificultades para adaptar sus sugerencias alimentarias a las necesidades nutricionales específicas de cada persona. La variabilidad inherente a la forma en que la nutrición afecta a los individuos suscita preocupación por la ausencia de una auténtica personalización, además otra carencia importante es la falta de interactividad.

Es en este punto donde se llega al objeto de estudio de este Trabajo Fin de Máster, el de como implementar un servicio sencillo e intuitivo, pero a la vez complejo, que permite a los usuarios que accedan a la web obtener recomendaciones personalizadas desde dudas sobre ciertas patologías que puedan llegar a presentar (obviando el hecho de que la asistencia a un profesional de la salud será necesaria) hasta alimentos o cambios que podrán realizar en función de distintos objetivos que posean.

## Planteamiento del problema

El objeto de estudio de este TFM, como se ha presentado en el apartado anterior es desarrollar un sistema que personalice los estilos de vida en función de la genética y la biometría, ya que recomendaciones generales sobre dieta y ejercicio son útiles, pero no siempre son efectivas para todas las personas.

## Objetivos

Como ya se ha adelantado, la propuesta consiste en desarrollar una web basada en varios LLM que permita el ingreso de los interesados de sus datos, biométricos, genéticos y otras dudas médicas que presenten (se responderá orientativamente).

**Objetivo principal:** desarrollar un sistema de personalización de estilos de vida basado en genómica y biometría. Este sistema tiene el potencial de mejorar la salud y el bienestar de las personas al proporcionarles recomendaciones personalizadas sobre dieta, ejercicio y otros aspectos del estilo de vida.

Objetivos secundarios:

* Mejorar la salud y el bienestar de las personas gracias al sistema de personalización de estilos de vida
* Reducir los costos de atención médica
* Empoderar a las personas para que tomen el control de su salud

# Marco teórico

No habría una fuente bibliográfica la cual haya significado una mayor influencia sobre cómo estructurar el framework o el proceso de realizar la web, sino que se ha ido obteniendo la información de distintas publicaciones u otras referencias.

Teniendo como referencia para proceder a herramientas como las publicadas ChatDiet (7) o Dietos (5), los cuales permiten comprender la manera de organizar el framework con respecto al uso de LLM para personalizar distintas necesidades del cliente (en el caso de ambas herramientas mencionadas en el campo de la nutrición).

Pero también se sigue un procedimiento similar en otros trabajos que proponen frameworks con LLM para datos genéticos (8).

Es decir, que para uno de los dos LLM que se procede a implementar en base a la lectura de documentos, la estructura básica es la misma seguida independientemente del ámbito relacionado con la salud escogido. Por lo que el primer paso será recopilar una gran cantidad de documentos de interés o extraerlos directamente de las webs necesarias, que permitirán al LLM generar respuestas en base a los diagnósticos o preguntas que proporcione el usuario.

Para el otro LLM, se utilizan los datos pertinentes de bases de datos como NHANES u otras similares, los cuales permitirán ser codificados para su posterior uso para obtener distintas estadísticas y porcentajes pertinentes para mostrar al usuario.

Así aportando una información más exacta acerca de por ejemplo “las posibilidades de padecer X patología o enfermedad” en base a los datos ingresados por el usuario como información sobre distintas dudas que posea acerca del diagnóstico proporcionado o acerca de otros ámbitos relacionados con la salud. Todo esto acompañado de las figuras necesarias para mostrar la información de la manera más accesible posible.

# Metodología

## Recolección de datos y preparación del modelo

Para el desarrollo del sistema de recomendaciones personalizadas de estilo de vida, utilizamos la base de datos NHANES (National Health and Nutrition Examination Survey) como fuente primaria de información. Esta encuesta compila una variedad de datos sobre la salud y nutrición de la población estadounidense, incluyendo detalles demográficos, medidas antropométricas, y hábitos de salud.

Además, hemos ampliado nuestra colección de datos incorporando fuentes diversificadas que incluyen documentos académicos y recursos en línea relacionados con nutrición, ejercicio, manejo del estrés y prevención de enfermedades. Esta expansión de la base de conocimientos permite un análisis más profundo y personalizado, abarcando:

* Documentos científicos y guías de práctica clínica.
* Bases de datos abiertas de acceso público y registros médicos digitales.
* Información en tiempo real de plataformas de salud y bienestar.

## Preprocesamiento y limpieza de datos

El proceso de preprocesamiento y limpieza de datos es fundamental para garantizar la calidad y la precisión de las recomendaciones generadas. Este proceso incluye:

* **Normalización y Estandarización:** Convertimos todas las métricas a escalas comparables y estandarizamos formatos para asegurar coherencia en todo el conjunto de datos.
* **Tratamiento de Valores Atípicos y Faltantes:** Implementamos métodos avanzados para tratar valores atípicos y reemplazar datos faltantes, utilizando técnicas como imputación basada en múltiples interpolaciones o modelos predictivos.
* **Codificación de Variables Categóricas:** Usamos técnicas como la codificación One-Hot o codificación de frecuencias para transformar variables categóricas en formatos numéricos que puedan ser procesados por nuestros modelos de aprendizaje automático.

Estos pasos son esenciales para preparar los datos para su análisis y asegurar que los modelos y algoritmos funcionen con la mayor eficacia posible.

## Integración en la base de conocimientos

Diagrama

Descripción generada automáticamenteDespués de la limpieza y el preprocesamiento de los datos recopilados, procedemos a integrar esta información en nuestra base de conocimientos, la cual es esencial para facilitar la recuperación de respuestas pertinentes durante la interacción con el sistema de chatbot. Este proceso se realiza a través de las siguientes etapas:

Ilustración 1. VECTORIZACIÓN CON OLLAMA Y ALMACENAMIENTO EN FAISS

1. Vectorización de Documentos:
   * Utilizamos **OllamaEmbeddings**, una herramienta avanzada dentro del marco de LangChain, para transformar textos complejos en vectores numéricos. Esta representación vectorial es fundamental para capturar el contexto semántico del texto, permitiendo que el sistema comprenda y procese eficazmente el contenido de los documentos.
2. Almacenamiento con FAISS:
   * Los vectores generados son almacenados en una base de datos vectorial utilizando FAISS (Facebook AI Similarity Search), que es un sistema altamente eficiente para la búsqueda rápida de similitudes en grandes volúmenes de datos. FAISS optimiza la recuperación de los documentos más relevantes en respuesta a las consultas del usuario, mejorando significativamente la velocidad y la precisión del sistema.
3. Acceso y Recuperación de Información:
   * Cuando un usuario realiza una consulta, el sistema emplea la base de datos vectorial para encontrar y recuperar rápidamente los fragmentos de texto más relevantes. Este proceso asegura que las respuestas generadas estén bien informadas y sean específicamente adaptadas a las necesidades y preguntas del usuario.

Este método de vectorización y almacenamiento no solo mejora la eficiencia y la efectividad de las respuestas proporcionadas por el chatbot, sino que también permite la escalabilidad del sistema al facilitar la adición y actualización continua de documentos y datos en la base de conocimientos.

## Desarrollo del Chatbot basado en RAG

El presente chatbot se basa en un enfoque de RAG (Retrieval-Augmented Generation), que combina la recuperación de información relevante de la base de datos vectorial con la generación de respuestas utilizando modelos de lenguaje avanzados. El código Python que se ha desarrollado implementa la funcionalidad principal del RAG, incluyendo la carga de la base de datos vectorial, el procesamiento de las consultas de los usuarios y la generación de recomendaciones personalizadas.

El flujo de trabajo del chatbot es el siguiente:

1. El usuario interactúa con el chatbot a través de una interfaz de usuario intuitiva, proporcionando información sobre sus características personales, datos de salud y preferencias de estilo de vida.
2. El chatbot procesa la consulta del usuario y utiliza técnicas de recuperación de información para identificar los datos más relevantes de la base de datos vectorial, considerando tanto los datos de NHANES como los documentos médicos y de estilo de vida.
3. Utilizando los datos recuperados y la información proporcionada por el usuario, el chatbot genera recomendaciones personalizadas de estilo de vida adaptadas a las necesidades y preferencias individuales.
4. Las recomendaciones generadas se presentan al usuario de manera clara y comprensible, junto con explicaciones y fundamentos basados en evidencia médica y científica.

## Arquitectura y funcionamiento de la API

Nuestro proyecto ha implementado una API robusta utilizando el marco de trabajo FastAPI, optimizado para crear interfaces de aplicación programática de alta velocidad. Esta API es esencial para coordinar las interacciones entre la interfaz de usuario basada en Streamlit y el procesamiento en el backend realizado a través de diversos componentes de LangChain y el modelo de lenguaje GPT-J potenciado por Groq.

Arquitectura General

La arquitectura de nuestra API se caracteriza por su diseño modular y su capacidad para integrar tecnologías de procesamiento de lenguaje natural y aprendizaje automático de manera eficiente. Estos son los componentes principales de nuestra arquitectura:

1. **main.py**: Actúa como el corazón de nuestra aplicación, configurando y lanzando el servidor FastAPI. Este módulo inicializa todas las dependencias, carga los modelos necesarios y prepara los endpoints de la API para recibir y responder a las solicitudes de los usuarios.
2. **custom\_agent.py**: Define la lógica para un agente personalizado que utiliza las capacidades avanzadas de LangChain para manejar consultas complejas. Este agente integra múltiples herramientas y recursos, como buscadores especializados en Wikipedia y Arxiv, para enriquecer las respuestas y proporcionar información contextualizada y relevante.
3. **streamlit\_app.py**: Proporciona una interfaz de usuario interactiva donde los usuarios pueden ingresar consultas y recibir respuestas. Este módulo se comunica con la API de FastAPI, enviando preguntas y mostrando las respuestas procesadas junto con detalles adicionales y explicaciones contextuales.

Funcionamiento de la API

El funcionamiento de nuestra API se puede describir a través de varios pasos clave en el flujo de procesamiento:

1. **Recepción de Consultas**: La API recibe consultas de los usuarios a través de **streamlit\_app.py**, que envía las preguntas en español al endpoint **/ask** configurado en **main.py**.
2. **Procesamiento y Traducción**: Una vez recibida la consulta, la API utiliza Google Translate para convertir la pregunta al inglés, preparándola para el procesamiento por nuestro modelo LLM.
3. **Generación de Respuestas**: El **custom\_agent.py** toma la entrada traducida y, utilizando el modelo de LLM junto con las herramientas configuradas, procesa la pregunta para generar una respuesta detallada. Este proceso incluye la recuperación de información relevante de bases de datos externas como Wikipedia y Arxiv, y la integración de estos datos para formular una respuesta coherente.
4. **Traducción y Respuesta Final**: La respuesta generada en inglés se traduce de vuelta al español y se envía de regreso a la interfaz de usuario de Streamlit, donde se muestra al usuario. Además, se proporcionan detalles del proceso de pensamiento del modelo y las fuentes de información utilizadas, ofreciendo transparencia y un mayor entendimiento de la respuesta.
5. **Interfaz de Usuario**: **streamlit\_app.py** muestra la respuesta en español al usuario y ofrece la opción de explorar detalles adicionales y el contexto utilizado para generar la respuesta, mejorando así la experiencia del usuario y proporcionando una plataforma educativa además de una herramienta de consulta.

Ventajas de la Integración Tecnológica

La integración de FastAPI, Streamlit, LangChain, y Groq ofrece numerosas ventajas, incluyendo alta velocidad de procesamiento, escalabilidad, y la capacidad de manejar consultas en tiempo real con respuestas detalladas y personalizadas. La arquitectura facilita una eficiente comunicación entre el frontend y el backend, asegurando que los datos se manejen de manera segura y las interacciones del usuario sean fluidas y productivas.

Esta arquitectura no solo cumple con los requisitos técnicos de nuestro proyecto, sino que también proporciona una base sólida para futuras expansiones y mejoras, permitiendo incorporar nuevas herramientas y adaptaciones conforme evolucionen las necesidades y tecnología.

## Evaluación y refinamiento del modelo

Una vez que el chatbot está en funcionamiento, se lleva a cabo un proceso riguroso de evaluación y refinamiento del modelo. Mediante el uso de métricas de evaluación apropiadas, como la precisión, la exhaustividad y la satisfacción del usuario, para medir el rendimiento del chatbot en la generación de recomendaciones personalizadas.

Además, se recopila y tiene en cuenta la retroalimentación de los usuarios, para realizar análisis cualitativos con la finalidad de identificar áreas de mejora y ajustar el modelo en consecuencia. Este proceso iterativo permite afinar continuamente el chatbot y garantizar que proporcione recomendaciones de alta calidad y relevancia para los usuarios.

## Desarrollo del prototipo de la web

### Consideraciones para el desarrollo de la web

Se tendrá en cuenta un diseño sencillo y minimalista, con un buen uso de espacios en blanco para crear una sensación de amplitud y orden. Junto con el uso de una tipografía y paletas de colores atractiva y que se adapten al estilo de la aplicación.

Por último, se podrán incluir imágenes y videos que sean relevantes para el contenido de la aplicación

Con relación a la parte de programación, se escribirá un código limpio y organizado con los comentarios pertinentes explicando su funcionamiento, junto al uso de librerías necesarias.

Para el desarrollo de la aplicación web se hará uso e Python Flask con las librerías necesarias para facilitar el desarrollo. Y en última instancia se realizarán las pruebas en distintos dispositivos y navegadores para asegurar su correcto funcionamiento.

3.7.2 LO HE ACTUALIZADO TODO DAVID

3.7.2 Pasos para el desarrollo de nuestra API

3.7.2.1. Identificar funcionalidades clave

Para nuestra API, hemos identificado las siguientes funcionalidades clave:

* Endpoint de consulta (/ask): Permitirá a los usuarios enviar preguntas en español y recibir respuestas personalizadas basadas en los datos genómicos, biométricos y los documentos médicos y de estilo de vida.
* Traducción automática: La API traducirá automáticamente las preguntas de los usuarios del español al inglés y las respuestas generadas del inglés al español, utilizando la biblioteca googletrans.
* Integración con herramientas externas: Utilizaremos herramientas adicionales como Wikipedia y Arxiv para enriquecer las respuestas y proporcionar información complementaria relevante.
* Generación de respuestas personalizadas: Utilizando técnicas de recuperación de información y modelos de lenguaje grande (LLM) como GPT-J, generaremos respuestas adaptadas a las necesidades y características individuales de cada usuario.

3.7.2.2. Diseño de la arquitectura de la API

Hemos diseñado una arquitectura modular y escalable para nuestra API, utilizando FastAPI como framework principal. La arquitectura se compone de los siguientes componentes:

* Endpoint de consulta (/ask): Recibe las preguntas de los usuarios y devuelve las respuestas generadas.
* Módulo de traducción: Utiliza la biblioteca googletrans para traducir las preguntas y respuestas entre español e inglés.
* Módulo de carga y vectorización de documentos: Carga los documentos médicos y de estilo de vida, y genera embeddings utilizando Langchain y FAISS.
* Módulo de generación de respuestas: Utiliza modelos de lenguaje grande (LLM) como GPT-J y herramientas adicionales para generar respuestas personalizadas.
* Base de datos vectorial: Almacena los embeddings generados a partir de los documentos cargados, utilizando FAISS para una recuperación eficiente.

3.7.2.3. Selección de tecnologías y bibliotecas

Hemos seleccionado cuidadosamente las tecnologías y bibliotecas más adecuadas para el desarrollo de nuestra API:

* FastAPI: Un framework web moderno y eficiente para construir APIs en Python, que nos permite crear endpoints rápidos y seguros.
* Langchain: Una biblioteca de desarrollo de aplicaciones de lenguaje natural que nos proporciona herramientas y componentes para cargar y procesar documentos, generar embeddings y definir agentes conversacionales.
* FAISS: Una biblioteca para la búsqueda eficiente de similitud en espacios vectoriales, que utilizamos para almacenar y recuperar embeddings de manera rápida.
* Groq: Una plataforma de aceleración de hardware para modelos de aprendizaje automático, que nos permite obtener respuestas más rápidas y eficientes al integrarla con el modelo GPT-J.
* googletrans: Una biblioteca de Python para la traducción automática, que utilizamos para traducir las preguntas y respuestas entre español e inglés.

3.7.2.4. Desarrollo del agente personalizado

Hemos desarrollado un agente personalizado utilizando la función create\_custom\_tools\_agent definida en el archivo custom\_agent.py. Este agente actúa como el controlador central de nuestra API, coordinando la interacción entre los diferentes componentes y herramientas.

El agente recibe las consultas de los usuarios, las procesa utilizando los LLM y las herramientas adicionales, y genera respuestas personalizadas basadas en los documentos cargados y los embeddings generados. La implementación de este agente personalizado nos permite aprovechar al máximo las capacidades de los LLM y las herramientas adicionales, adaptándolas a nuestro caso de uso específico.

3.7.2.5. Integración de herramientas externas

Hemos integrado herramientas adicionales como Wikipedia y Arxiv para enriquecer las respuestas generadas por nuestra API. Utilizamos las clases WikipediaQueryRun y ArxivQueryRun de Langchain para realizar consultas en estas fuentes externas y obtener información complementaria relevante.

La integración de estas herramientas nos permite proporcionar respuestas más completas y precisas a las preguntas de los usuarios, aprovechando el conocimiento disponible en Wikipedia y los estudios científicos en Arxiv.

3.7.2.6. Pruebas y validación

Hemos realizado pruebas exhaustivas de nuestra API para garantizar su correcto funcionamiento y rendimiento. Utilizamos herramientas como Postman para probar los endpoints y verificar que las respuestas generadas sean coherentes y relevantes.

Además, hemos validado la integración de las diferentes tecnologías y bibliotecas utilizadas, asegurándonos de que todas las piezas funcionen de manera armoniosa y eficiente.

3.7.2.7. Documentación y puesta en marcha

Hemos documentado cuidadosamente nuestra API, incluyendo instrucciones detalladas sobre cómo configurar el entorno, instalar las dependencias y ejecutar la aplicación. Además, hemos proporcionado ejemplos de uso y explicaciones sobre los diferentes endpoints y funcionalidades disponibles.

Finalmente, hemos desplegado nuestra API en un entorno de producción, asegurándonos de que sea accesible y esté lista para ser utilizada por los usuarios finales. Hemos configurado la infraestructura necesaria y hemos establecido mecanismos de monitoreo y registro para garantizar la estabilidad y el rendimiento de la API a lo largo del tiempo.

Con estos pasos, hemos desarrollado una API robusta y eficiente que nos permite alcanzar nuestro objetivo de crear un sistema de personalización de estilos de vida basado en genómica y biometría, brindando recomendaciones adaptadas a las necesidades y características individuales de cada usuario.

#### Desarrollo Back-End con Flask

En lugar de Flask, optamos por FastAPI para el desarrollo del back-end, dada su compatibilidad superior con operaciones asíncronas y su rendimiento en la gestión de solicitudes en tiempo real, lo cual es esencial para nuestro sistema de recomendaciones personalizadas. FastAPI también facilita la implementación de características modernas de seguridad y manejo de sesiones, lo que es crucial para proteger los datos biométricos y genómicos de los usuarios.

La documentación automática con Swagger UI, proporcionada por FastAPI, mejora la transparencia y facilita la prueba y validación de endpoints de la API por parte de los desarrolladores, asegurando que cada función del back-end se implemente correctamente y sin errores.

#### Integración Front-End y Back-End

La integración del front-end desarrollado en React.js y el back-end en FastAPI se realiza a través de solicitudes HTTP asincrónicas gestionadas por Axios. Este enfoque permite un flujo de datos eficiente y seguro entre la interfaz de usuario y el servidor, garantizando que las respuestas a las consultas de los usuarios sean rápidas y fiables.

Implementamos también autenticación JWT (JSON Web Tokens) para asegurar que las sesiones de los usuarios sean seguras y para mantener la integridad de la información personalizada proporcionada y recibida. La arquitectura final es robusta, con el front-end y el back-end trabajando conjuntamente para proporcionar una experiencia de usuario coherente y una funcionalidad de sistema óptima.

### Pruebas de validación

La fase de pruebas de validación es crucial para garantizar que nuestra aplicación web no solo cumpla con los requisitos funcionales y de usuario, sino que también ofrezca un rendimiento consistente y seguro en una variedad de entornos. Este proceso se divide en varias etapas clave para abarcar tanto la funcionalidad como la seguridad de la plataforma:

* **Pruebas de Funcionalidad**
* **Pruebas Unitarias**: Implementamos pruebas unitarias para cada componente funcional de la aplicación, asegurando que cada parte funcione de manera aislada como se espera.
* **Pruebas de Integración**: Verificamos que los componentes de la aplicación interactúen correctamente entre sí y con los sistemas de backend, como la API y la base de datos.
* **Pruebas de Sistema**: Realizamos pruebas completas del sistema para asegurar que toda la aplicación funcione según lo diseñado en un entorno que simule la producción.
* **Pruebas de Compatibilidad**
* **Pruebas en Múltiples Navegadores**: Validamos la aplicación en los navegadores más utilizados (Chrome, Firefox, Safari, Edge) para asegurar una experiencia uniforme independientemente del navegador elegido por el usuario.
* **Pruebas en Diferentes Dispositivos**: Evaluamos la aplicación en una variedad de dispositivos, incluyendo desktops, tablets y smartphones, para garantizar una experiencia óptima y adaptativa en todos los tamaños de pantalla y sistemas operativos.
* **Pruebas de Rendimiento**
* **Pruebas de Carga**: Simulamos el acceso simultáneo de múltiples usuarios para evaluar cómo maneja la aplicación cargas de trabajo elevadas, asegurando que el rendimiento se mantenga estable.
* **Pruebas de Estrés**: Llevamos a la aplicación a condiciones extremas de uso para identificar los límites de su capacidad operativa y asegurar que pueda manejar picos inesperados de tráfico sin comprometer la funcionalidad.
* **Pruebas de Seguridad**
* **Pruebas de Seguridad de la Aplicación**: Realizamos pruebas exhaustivas para identificar vulnerabilidades de seguridad en la aplicación, como inyecciones SQL, XSS, y CSRF, aplicando correcciones antes del lanzamiento.
* **Auditorías de Seguridad**: Contratamos a terceros para realizar auditorías de seguridad que ayuden a identificar y mitigar riesgos que podrían no haber sido detectados durante las pruebas internas.
* **Validación del Usuario**
* **Pruebas Beta con Usuarios Reales**: Implementamos una fase beta cerrada, donde usuarios seleccionados utilizan la aplicación en condiciones reales para identificar cualquier problema de usabilidad o funcionalidad que los tests no hayan capturado.
* **Recopilación de Feedback**: Solicitamos retroalimentación detallada sobre la experiencia del usuario, que utilizamos para hacer ajustes finales antes del lanzamiento público.
* **Conclusión**
* Este enfoque estructurado para las pruebas de validación asegura que nuestra aplicación web no solo cumpla con los requisitos técnicos y de negocio, sino que también ofrezca una experiencia segura, eficiente y agradable para todos los usuarios, independientemente de su dispositivo o navegador. Esta meticulosidad en las pruebas es vital para el éxito de nuestra plataforma en un entorno real.

## Esquema y funcionamiento formal del RAG

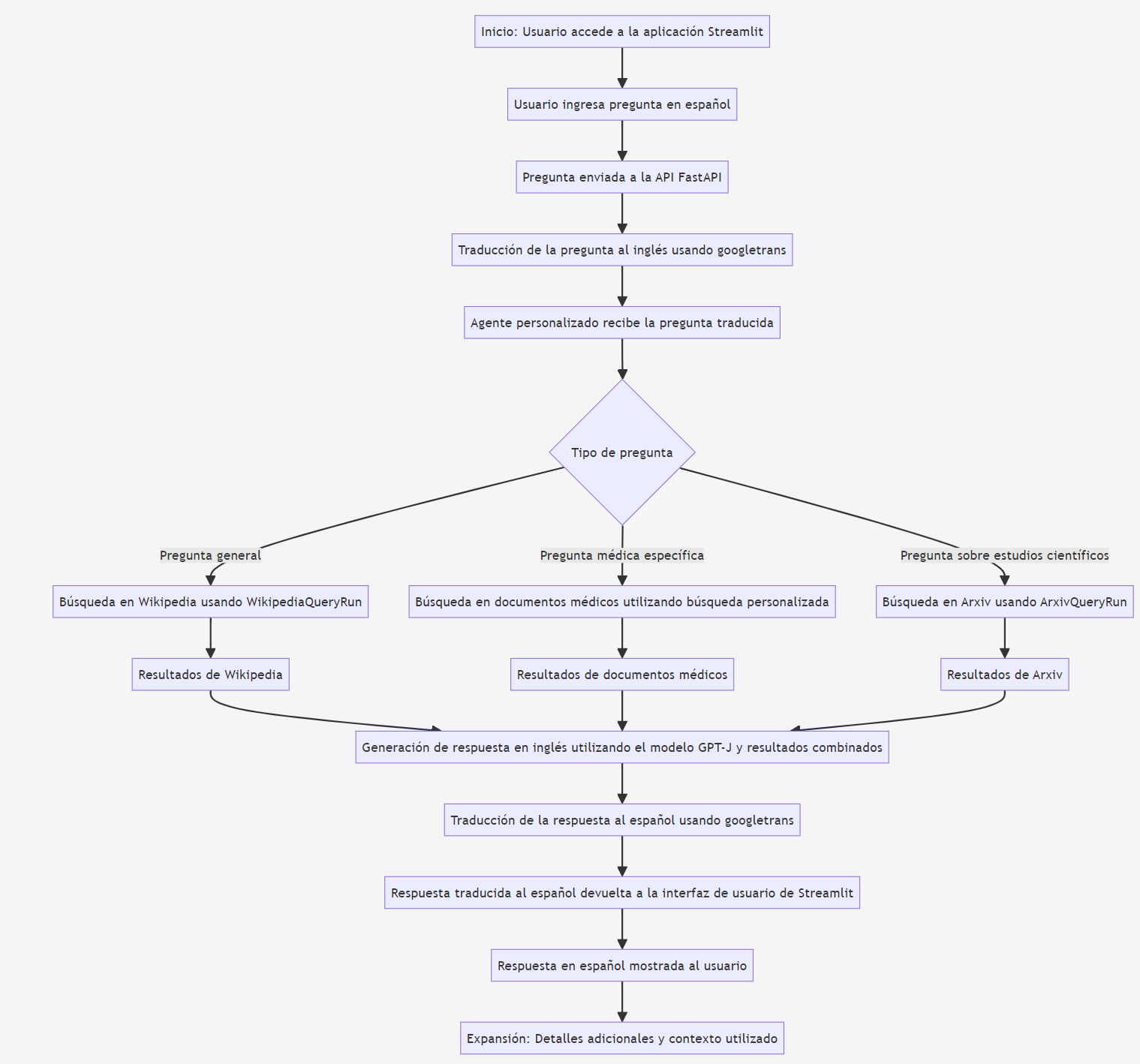
En esta sección, explicaremos paso a paso cómo funciona el código del GitHub que hemos creado para este TFM. Al mismo tiempo, iremos revisando las herramientas que hemos utilizado y su utilización técnica en el código Python. (sección actualizada del TFM basada en el nuevo código de la v2 del API main.py)

Ilustración 2. DIAGRAMA DE FLUJO DEL API.

### Funcionamiento de main.py

A continuación, se muestra un esquema gráfico mediante un diagrama Mermaid (Anexo diagramas Mermaid) que ilustra el flujo de la aplicación. Este diagrama ayudará a visualizar cómo interactúan los diferentes componentes del sistema y cómo se procesan las consultas del usuario.

#### **Integración de tecnologías**

* Groq: Proporciona el backend de aceleración de hardware para el modelo de lenguaje GPT-J.
* LangChain: Facilita la orquestación de las diferentes etapas del proceso, desde la carga de documentos hasta la generación de respuestas.
* FAISS: Se utiliza para la gestión eficiente de la base de datos vectorial, permitiendo búsquedas rápidas y precisas.
* Streamlit: Ofrece una forma sencilla y efectiva de crear interfaces de usuario interactivas para aplicaciones de Python, permitiendo la interacción directa con el usuario.
* FastAPI: Proporciona un framework para crear una API robusta y eficiente que permite la comunicación entre el frontend y el backend.

#### Detalles del código

* Importaciones y Configuración Inicial:
  + Se importan módulos necesarios como fastapi, langchain\_groq, langchain\_community, entre otros.
  + Se cargan las claves API necesarias para Groq a través de variables de entorno, asegurando que el acceso a la API se maneje de manera segura.
* Inicialización del Modelo de Lenguaje y Configuración del Prompt:
  + Se carga el modelo de lenguaje GPT-J utilizando la clase GroqModel de langchain\_groq, especificando el nombre del modelo y la clave API de Groq.
  + Se configura un ChatPromptTemplate para guiar cómo el modelo debe formular respuestas, instruyéndolo a responder preguntas basadas en el contexto proporcionado.
* Carga y Vectorización de Documentos:
  + En el evento de inicio de la aplicación (@app.on\_event("startup")), se cargan los documentos desde diferentes fuentes (PDF, CSV, JSON) utilizando los cargadores correspondientes de LangChain.
  + Los documentos se dividen en fragmentos más pequeños utilizando RecursiveCharacterTextSplitter y se vectorizan utilizando OllamaEmbeddings y FAISS para crear una base de datos vectorial eficiente.
* Configuración de Herramientas Adicionales:
  + Se crean instancias de las herramientas WikipediaQueryRun y ArxivQueryRun utilizando los envoltorios de API correspondientes.
  + Estas herramientas permiten realizar consultas en Wikipedia y Arxiv para obtener información adicional durante el proceso de generación de respuestas.
* Endpoint de Consulta (/ask):
  + Se define un endpoint de API (@app.post("/ask")) que recibe las consultas de los usuarios.
  + La consulta se traduce del español al inglés utilizando la biblioteca googletrans.
  + Se crea un agente personalizado utilizando la función create\_custom\_tools\_agent, pasando el modelo de lenguaje, las herramientas adicionales y el prompt.
  + El agente procesa la consulta y genera una respuesta utilizando el modelo de lenguaje y las herramientas adicionales.
  + La respuesta generada se traduce de vuelta al español utilizando googletrans y se devuelve al usuario junto con el contexto utilizado para generarla.

#### Flujo de la aplicación

1. El usuario accede a la interfaz de usuario de Streamlit y realiza una consulta en español.
2. La consulta se envía al endpoint /ask de la API FastAPI.
3. La consulta se traduce al inglés utilizando googletrans.
4. Se crea un agente personalizado utilizando el modelo de lenguaje GPT-J, las herramientas adicionales (Wikipedia y Arxiv) y el prompt.
5. El agente procesa la consulta y genera una respuesta utilizando el modelo de lenguaje y las herramientas adicionales.
6. La respuesta generada se traduce de vuelta al español utilizando googletrans.
7. La respuesta y el contexto utilizado se devuelven a la interfaz de usuario de Streamlit.
8. La interfaz de usuario muestra la respuesta al usuario y permite expandir los detalles adicionales y el contexto utilizado.

Este esquema y flujo de la aplicación representan el funcionamiento actualizado de nuestro RAG basado en el nuevo código de main.py. La integración de tecnologías como Groq, LangChain, FAISS, Streamlit y FastAPI, junto con las herramientas adicionales de Wikipedia y Arxiv, permite un procesamiento eficiente de las consultas de los usuarios y la generación de respuestas precisas y contextualizadas.

# Resultados y discusión

Los resultados preliminares de nuestro chatbot basado en RAG y la base de datos vectorial han demostrado un gran potencial para proporcionar recomendaciones personalizadas de estilo de vida a los usuarios. La combinación de los datos de NHANES con documentos médicos y de estilo de vida adicionales ha permitido generar recomendaciones precisas y adaptadas a las necesidades individuales.

La evaluación cuantitativa del modelo ha revelado altas puntuaciones de precisión y exhaustividad en la generación de recomendaciones relevantes. Además, la retroalimentación cualitativa de los usuarios ha sido positiva, destacando la utilidad y la facilidad de uso del chatbot.

Sin embargo, también hemos identificado algunas limitaciones y desafíos durante el desarrollo y la evaluación del chatbot. Uno de los principales desafíos ha sido la integración efectiva de los datos de NHANES con los documentos adicionales, asegurando una representación coherente y completa de la información. Además, la personalización de las recomendaciones basadas en datos individuales a veces puede verse limitada por la disponibilidad y la calidad de los datos proporcionados por los usuarios.

A pesar de estos desafíos, creemos que nuestro enfoque basado en RAG y la base de datos vectorial tiene un gran potencial para mejorar la calidad y la relevancia de las recomendaciones de estilo de vida. La combinación de datos estructurados de NHANES con información no estructurada de documentos médicos y de estilo de vida nos permite aprovechar al máximo la riqueza de conocimientos disponibles para generar recomendaciones personalizadas.

En comparación con enfoques tradicionales que se basan únicamente en reglas predefinidas o en la experiencia de expertos, nuestro chatbot tiene la capacidad de adaptarse a las necesidades individuales de cada usuario y proporcionar recomendaciones basadas en evidencia y datos actualizados. Esto tiene el potencial de mejorar significativamente la adopción y la efectividad de las recomendaciones de estilo de vida, ya que los usuarios pueden recibir consejos personalizados y relevantes para su situación específica.

# Conclusiones

En conclusión, nuestro chatbot basado en RAG y la base de datos vectorial que integra datos de NHANES con documentos médicos y de estilo de vida ha demostrado ser una herramienta prometedora para proporcionar recomendaciones personalizadas de estilo de vida a los usuarios. La combinación de técnicas de recuperación de información y generación de lenguaje natural nos permite aprovechar al máximo la riqueza de conocimientos disponibles y adaptarlos a las necesidades individuales de cada usuario.

Los resultados preliminares han sido alentadores, con altas puntuaciones de precisión y exhaustividad en la generación de recomendaciones relevantes, así como una retroalimentación positiva por parte de los usuarios. Sin embargo, también hemos identificado algunas limitaciones y desafíos que requieren atención adicional, como la integración efectiva de datos heterogéneos y la personalización basada en datos individuales limitados.

Para el trabajo futuro, nos proponemos abordar estas limitaciones y continuar mejorando el rendimiento y la usabilidad del chatbot. Algunas de las áreas clave de enfoque incluyen:

1. Explorar técnicas avanzadas de representación de conocimientos y razonamiento para mejorar la integración y la coherencia de los datos de NHANES con los documentos adicionales.
2. Desarrollar métodos de adquisición de datos más efectivos para recopilar información relevante de los usuarios y mejorar la personalización de las recomendaciones.
3. Realizar estudios de usuario a mayor escala para evaluar la efectividad y la aceptación del chatbot en diferentes poblaciones y contextos.
4. Investigar la integración de técnicas de aprendizaje por refuerzo para permitir que el chatbot aprenda y se adapte continuamente en función de las interacciones con los usuarios.
5. Explorar la posibilidad de extender el chatbot a otras áreas relacionadas con la salud y el bienestar, como la salud mental, la gestión de enfermedades crónicas y la prevención de enfermedades.

En resumen, nuestro chatbot basado en RAG y la base de datos vectorial que integra datos de NHANES con documentos médicos y de estilo de vida representa un avance significativo en la provisión de recomendaciones personalizadas de estilo de vida. Aunque aún hay trabajo por hacer, creemos que este enfoque tiene un gran potencial para mejorar la salud y el bienestar de las personas al brindarles acceso a información y consejos personalizados basados en evidencia. Esperamos continuar desarrollando y perfeccionando esta herramienta para ayudar a las personas a tomar decisiones informadas y adoptar estilos de vida más saludables.

1. Código diagrama de flujo v3.

graph TD

A[Inicio: Usuario accede a la aplicación Streamlit] --> B[Usuario ingresa pregunta en español]

B --> C[Pregunta enviada a la API FastAPI]

C --> D[Traducción de la pregunta al inglés usando googletrans]

D --> E[Agente personalizado recibe la pregunta traducida]

E --> F{Tipo de pregunta}

F --> |Pregunta general| G[Búsqueda en Wikipedia usando WikipediaQueryRun]

F --> |Pregunta médica específica| H[Búsqueda en documentos médicos utilizando búsqueda personalizada]

F --> |Pregunta sobre estudios científicos| I[Búsqueda en Arxiv usando ArxivQueryRun]

G --> J[Resultados de Wikipedia]

H --> K[Resultados de documentos médicos]

I --> L[Resultados de Arxiv]

J --> M[Generación de respuesta en inglés utilizando el modelo GPT-J y resultados combinados]

K --> M

L --> M

M --> N[Traducción de la respuesta al español usando googletrans]

N --> O[Respuesta traducida al español devuelta a la interfaz de usuario de Streamlit]

O --> P[Respuesta en español mostrada al usuario]

P --> Q[Expansión: Detalles adicionales y contexto utilizado]

1. Tratamiento, segregación de documentos y vectorización (*embbeding*) de los documentos.

Analicemos el fragmento relevante del código de carga de documentos en main.py (el API creado).

@app.on\_event("startup")

async def startup\_event():

embeddings = OllamaEmbeddings()

pdf\_loader = PyPDFDirectoryLoader("./ruta/al/directorio/pdf")

pdf\_docs = pdf\_loader.load()

csv\_loader = CSVLoader("./ruta/al/archivo.csv")

csv\_docs = csv\_loader.load()

json\_loader = JSONLoader("./ruta/al/archivo.json")

json\_docs = json\_loader.load()

docs = pdf\_docs + csv\_docs + json\_docs

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=1000, chunk\_overlap=200)

final\_documents = text\_splitter.split\_documents(docs)

vectors = FAISS.from\_documents(final\_documents, embeddings)

app.state.vectors = vectors

wikipedia\_api\_wrapper = WikipediaAPIWrapper(top\_k\_results=1, doc\_content\_chars\_max=200)

app.state.wikipedia\_tool = WikipediaQueryRun(api\_wrapper=wikipedia\_api\_wrapper)

arxiv\_api\_wrapper = ArxivAPIWrapper(top\_k\_results=1, doc\_content\_chars\_max=200)

app.state.arxiv\_tool = ArxivQueryRun(api\_wrapper=arxiv\_api\_wrapper)

Al revisar el código, vemos que la carga de documentos se realiza utilizando las clases PyPDFDirectoryLoader, CSVLoader y JSONLoader de langchain.

Algunas observaciones para utilizar el API correctamente y entender la carga de documentos:

1. Asegurarse de que las rutas "./ruta/al/directorio/pdf", "./ruta/al/archivo.csv" y "./ruta/al/archivo.json" sean las correctas y apunten a los directorios y archivos adecuados en su sistema de archivos.
2. Verificar que los archivos PDF, CSV y JSON existan en las rutas especificadas y sean válidos.
3. El código utiliza las clases de carga adecuadas para cada tipo de archivo: PyPDFDirectoryLoader para archivos PDF, CSVLoader para archivos CSV y JSONLoader para archivos JSON.
4. Los documentos cargados se combinan en una sola lista docs utilizando la concatenación de listas (docs = pdf\_docs + csv\_docs + json\_docs).
5. Luego, los documentos se dividen en fragmentos más pequeños utilizando RecursiveCharacterTextSplitter con un tamaño de fragmento de 1000 caracteres y una superposición de 200 caracteres.
6. Los documentos divididos se vectorizan utilizando FAISS.from\_documents() con los embeddings de OllamaEmbeddings.
7. Los vectores resultantes se almacenan en app.state.vectors para su posterior uso en el endpoint /ask.
8. ¿Para qué sirve nuestro RAG?

David te he hecho esta introducción para que puedas actualizar los puntos 1-3.1 de la entrega 1, espero que te sirva.

En nuestro proyecto de TFM, hemos desarrollado una API utilizando FastAPI que nos permite alcanzar nuestro objetivo principal: crear un sistema de personalización de estilos de vida basado en genómica y biometría. Esta API utiliza diversas bibliotecas y herramientas de procesamiento de lenguaje natural para generar recomendaciones personalizadas a partir de los datos proporcionados por los usuarios.

Una de las bibliotecas clave que utilizamos es Langchain, un framework de desarrollo de aplicaciones de lenguaje natural. Langchain nos proporciona una serie de herramientas y componentes que nos permiten construir flujos de trabajo complejos de manera modular y escalable. Utilizamos varios módulos y funciones de Langchain para cargar y procesar documentos, generar embeddings, crear almacenes vectoriales y definir agentes conversacionales.

Para la carga de documentos, utilizamos las clases PyPDFDirectoryLoader, CSVLoader y JSONLoader de Langchain. Estas clases nos permiten cargar archivos PDF, CSV y JSON, respectivamente, y convertirlos en un formato estructurado que podemos procesar posteriormente. Los documentos cargados se dividen en fragmentos más pequeños utilizando la clase RecursiveCharacterTextSplitter, lo que nos permite manejar documentos de gran tamaño de manera eficiente.

Una vez que tenemos los documentos cargados y divididos, generamos representaciones vectoriales de los fragmentos de texto utilizando la clase OllamaEmbeddings de Langchain. Estas representaciones vectoriales, conocidas como embeddings, nos permiten capturar la semántica y el contexto de los fragmentos de texto de manera numérica. Luego, almacenamos estos embeddings en un almacén vectorial utilizando la biblioteca FAISS, que nos proporciona una forma eficiente de realizar búsquedas de similitud entre los vectores.

Para la generación de respuestas, utilizamos modelos de lenguaje grande (LLM) como GPT-J. Estos modelos nos permiten generar texto coherente y contextualmente relevante a partir de un prompt o una consulta dada. Integramos el modelo GPT-J con la plataforma Groq, que nos proporciona una aceleración de hardware para el procesamiento de inferencia. Esto nos permite obtener respuestas más rápidas y eficientes, aprovechando la potencia de cómputo de Groq.

Además de los LLM, también utilizamos herramientas adicionales para enriquecer las capacidades de nuestra API. Hemos implementado la clase WikipediaQueryRun y ArxivQueryRun de Langchain para realizar consultas en Wikipedia y Arxiv, respectivamente. Estas herramientas nos permiten obtener información adicional y relevante para responder a las consultas de los usuarios de manera más completa y precisa.

Para orquestar todas estas piezas y crear un flujo de trabajo coherente, hemos desarrollado un agente personalizado utilizando la función create\_custom\_tools\_agent definida en el archivo custom\_agent.py. Este agente actúa como el controlador central de nuestra API, coordinando la interacción entre los diferentes componentes y herramientas. El agente recibe las consultas de los usuarios, las procesa utilizando los LLM y las herramientas adicionales, y genera respuestas personalizadas basadas en los documentos cargados y los embeddings generados.

Referencias bibliográficas

1. Min B, Ross H, Sulem E, Veyseh APB, Nguyen TH, Sainz O, et al. Recent Advances in Natural Language Processing via Large Pre-Trained Language Models: A Survey [Internet]. arXiv; 2021 [citado 31 de marzo de 2024]. Disponible en: http://arxiv.org/abs/2111.01243

2. Egli A. ChatGPT, GPT-4, and Other Large Language Models: The Next Revolution for Clinical Microbiology? Clin Infect Dis. 11 de noviembre de 2023;77(9):1322-8.

3. Mökander J, Schuett J, Kirk HR, Floridi L. Auditing large language models: a three-layered approach. AI Ethics [Internet]. 30 de mayo de 2023 [citado 31 de marzo de 2024]; Disponible en: https://link.springer.com/10.1007/s43681-023-00289-2

4. Yu B. Evaluating Pre-Trained Language Models on Multi-Document Summarization for Literature Reviews.

5. Agapito G, Calabrese B, Guzzi PH, Cannataro M, Simeoni M, Care I, et al. DIETOS: A recommender system for adaptive diet monitoring and personalized food suggestion. En: 2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob) [Internet]. New York, NY: IEEE; 2016 [citado 1 de abril de 2024]. p. 1-8. Disponible en: http://ieeexplore.ieee.org/document/7763190/

6. Iwendi C, Khan S, Anajemba JH, Bashir AK, Noor F. Realizing an Efficient IoMT-Assisted Patient Diet Recommendation System Through Machine Learning Model. IEEE Access. 2020;8:28462-74.

7. Yang Z, Khatibi E, Nagesh N, Abbasian M, Azimi I, Jain R, et al. ChatDiet: Empowering Personalized Nutrition-Oriented Food Recommender Chatbots through an LLM-Augmented Framework [Internet]. arXiv; 2024 [citado 1 de abril de 2024]. Disponible en: http://arxiv.org/abs/2403.00781

8. Chang J, Wang S, Ling C, Qin Z, Zhao L. Gene-associated Disease Discovery Powered by Large Language Models [Internet]. arXiv; 2024 [citado 1 de abril de 2024]. Disponible en: http://arxiv.org/abs/2401.09490