ANALITICA NOVIEMBRE
BOOTCAMPS

BOOTCAMP GEN IA - AGOSTO 2024

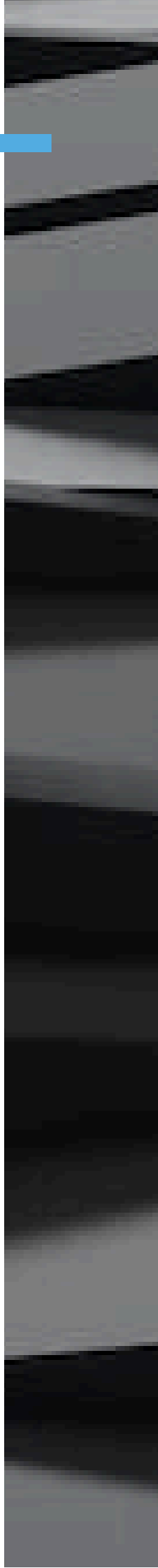
CHATBOT DE SOPORTE AL CLIENTE

AUTORA

MALENA CONSTANCIO

OBJETIVO

Este proyecto es un bot de soporte al cliente para Netflix desarrollado dentro del marco del **Bootcamp GEN IA** dictado por
Analítica Noviembre en Agosto de 2024.
El mismo utiliza **FastAPI** en el backend y **Streamlit** en el frontend para interactuar con los usuarios.
El bot está diseñado para proporcionar respuestas a problemas comunes que los usuarios pueden tener con Netflix.
Se utilizó **LangChain** para la integración con modelos de lenguaje y herramientas de soporte.



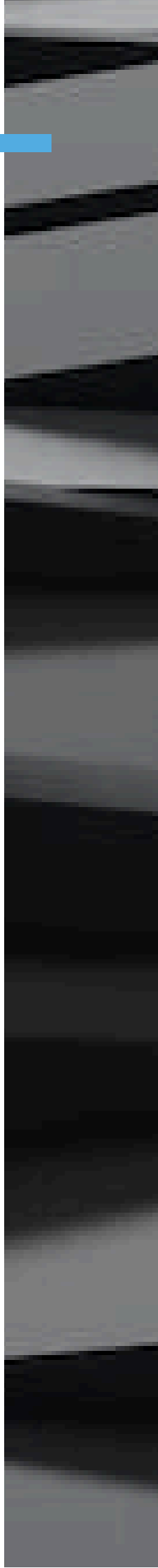
METODOLOGÍA

1. Estructura del proyecto.

2. Diagrama de arquitectura y flujos.

3. Visualización de Resultados.

4. Conclusiones.



1. ESTRUCTURA DEL PROYECTO.

REQUISITOS PREVIOS

- Python 3.12.3
- Git
- Acceso a SerpAPI , es necesario tener generada una api key propia https://serpapi.com/
- Acceso a OpenAI , es necesario tener generada una api key propia https://openai.com/api/

INSTRUCCIONES DE INSTALACIÓN

Pasos para configurar el entorno:

1. Clonar el Repositorio
   git clone https://github.com/analitica-noviembre/integrador_gen-ai_noviembre.git

2. Crear un Entorno Virtual
   python -m venv venv

3. Activar el Entorno Virtual

   En Linux/Mac:
   source venv/bin/activate

   En Windows:
   venv\Scripts\activate

4. Instalar las Dependencias

```
pip install -r requirements.txt
```

5. Configurar Variables de Entorno

Dentro del archivo **.env** en la carpeta principal del proyecto añadir las variables necesarias.

```
SERPAPI_API_KEY = tu_clave_api_de_serpapi
OPENAI_API_KEY = tu_clave_api_de_openai
```

6. Generar la Base de Datos de Embeddings

Para generar la base de datos utilizada por el bot ejecutar el script correspondiente para generar los embeddings. Asegurarse de tener activado el entorno virtual antes de proceder:

```
python backend/data/generate_embeddings.py
```

Este script tomará los documentos almacenados en la carpeta backend/data/documents/ y generará los embeddings necesarios, que serán almacenados en la carpeta backend/data/chroma/.

CHATBOT DE SOPORTE AL CLIENTE

EJECUCIÓN DEL PROYECTO

Backend

Para iniciar el servidor FastAPI, ejecuta el siguiente comando:

```
uvicorn backend.main:app --reload
```

Esto iniciará el servidor en `http://localhost:8000`.

Frontend

Para ejecutar el frontend con Streamlit, utiliza el siguiente comando:

```
streamlit run frontend/main_front.py
```

Esto iniciará la aplicación de Streamlit en tu navegador.

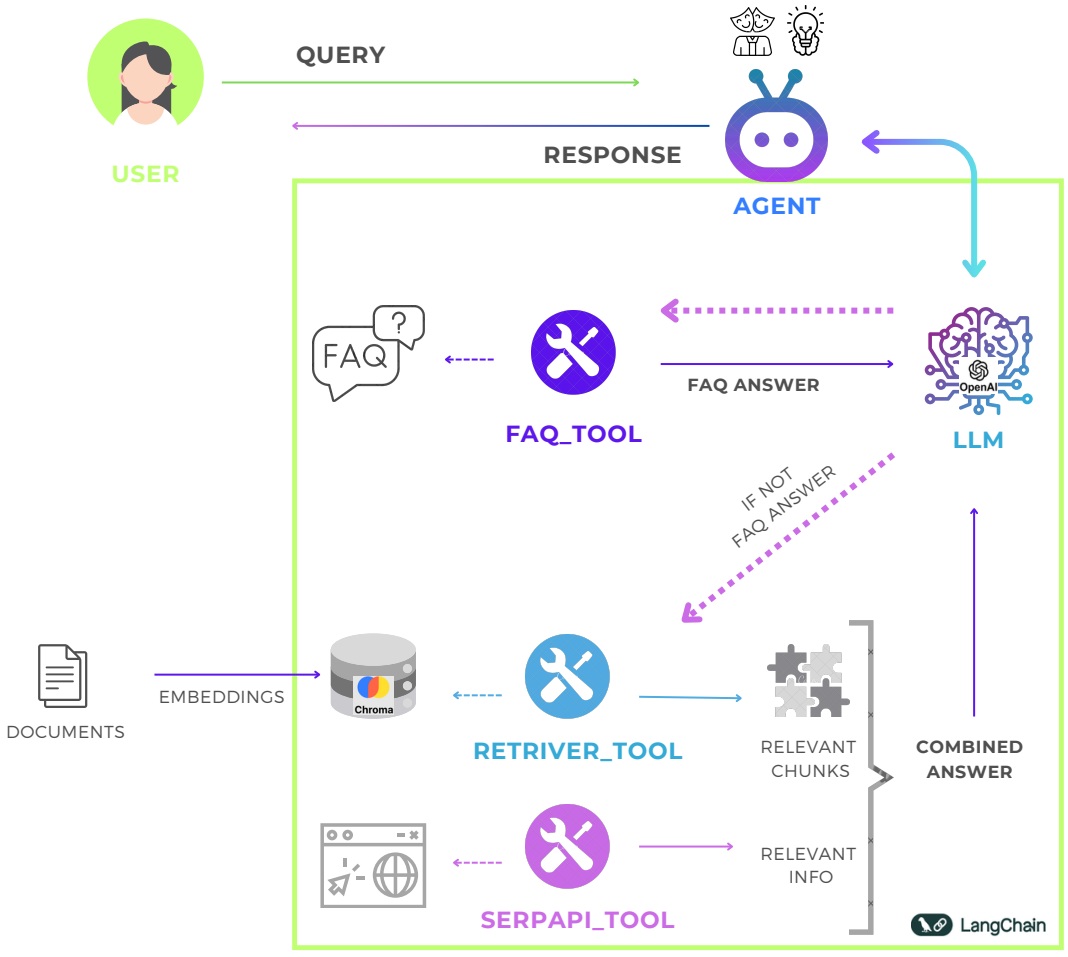
USO DEL BOT

1. Abrir el navegador y acceder al URL generado por Streamlit.
2. Ingresar una consulta en la interfaz del bot.
3. El bot procesará la consulta para devolver una respuesta.
4. Si necesitas Borrar el historial podes presionar el icono del bote de basura y comenzar una nueva conversación.

ESTRUCTURA DEL PROYECTO

```
netflix_support_bot/
├── backend/                    # Carpeta principal del back-end
│   ├── agent/                  # Componentes relacionados con el comportamiento del agente
│   │   ├── agent_config.py     # Archivo de configuración del agente
│   │   ├── agent_initializer.py  # Inicializador del agente
│   │   ├── llm_config.py       # Configuración del LLM
│   │   ├── memory.py           # Clase para manejar la memoria del bot (historial de conversaciones)
│   │   └── personality.py      # Clase para definir la personalidad del bot
│   ├── api/                    # Carpeta de la aplicación FastAPI
│   │   └── main.py             # Punto de entrada principal para el servidor FastAPI
│   ├── config/                 # Configuración general del proyecto
│   │   └── settings.py         # Configuraciones del proyecto (variables de entorno, claves API)
│   ├── data/                   # Datos y recursos utilizados por el LLM
│   │   ├── chroma/             # Base de datos de Chroma para embeddings
│   │   │   ├── chroma.sqlite3  # Base de datos de Chroma
│   │   │   └── (archivos internos de Chroma)
│   │   ├── documents/          # Documentos para el Retriever Tool
│   │   │   └── manual_netflix.pdf  # Manual de soporte de Netflix
│   │   └── generate_embeddings.py  # Script para generar los embeddings
│   ├── models/                 # Modelos de datos utilizados en el proyecto
│   │   └── query_request.py    # Modelo de solicitud de consulta del usuario
│   └── tools/                  # Herramientas para el LLM
│       ├── faq_tool.py         # Herramienta para preguntas frecuentes
│       └── retriever_tool.py   # Herramienta de búsqueda interna
├── frontend/
│   ├── main_front.py           # Archivo principal de Streamlit
│   └── assets/                 # Recursos del frontend (imágenes, estilos CSS)
│       └── images/             # Imágenes para el front-end (logos, etc.)
│           ├── landing_netflix.JPG
│           └── logo_netflix.png
├── .env                        # Variables de entorno (API keys)
├── README.md                   # Descripción general del proyecto
└── requirements.txt            # Dependencias del proyecto
```

2. DIAGRAMA DE ARQUITECTURA Y FLUJOS.



User (Usuario):

El usuario es quien inicia el proceso al enviar una consulta (query) al chatbot a través de la interfaz de usuario.

Agent (Agente):

El agente es el punto central que coordina las herramientas (tools) y el modelo de lenguaje para proporcionar una respuesta al usuario. Recibe la consulta, decide qué herramienta utilizar y coordina la respuesta final.

En este caso le asigne el uso de memoria para que recuerde las interacciones con el usuario puediendose setear la cantidad de interacciones a recordar mediante un parámetro y por otro lado al definir la personalidad del mismo existe la posibilidad de adaptarla al contexto e idiosincracia del país el cual tambien recibe por parámetro, a si mismo como parte del prompt base recibe ejemplos del léxico y tiempo verbal a utilizar mediante la estratégia de few shots, los mismos tambien son configurables dentro del proyecto.

FAQ Tool:

El objetivo del FAQ Tool es proporcionar respuestas rápidas y directas a preguntas frecuentes sin tener que procesarlas en profundidad.
Si la consulta coincide con una de las preguntas frecuentes, el FAQ Tool genera una respuesta automáticamente y se la envía al agente.

LLM (Large Language Model):

El modelo de lenguaje, en este caso GPT 3.5 , se encarga de procesar las consultas más complejas que requieren una interpretación avanzada.

Camino Complejo: Si el FAQ Tool no puede responder la consulta (es decir, no encuentra una coincidencia), el agente pasa la consulta al LLM.
El LLM se utiliza para integrar respuestas de diferentes fuentes y proporcionar una respuesta coherente y completa al usuario.

Retriever Tool:

El Retriever Tool se utiliza para buscar información relevante en una base de datos o en un sistema de almacenamiento de conocimientos, en este caso Chroma DB.
A través de embeddings, el Retriever Tool encuentra los fragmentos de información más relevantes para la consulta del usuario.

Camino de Recuperación: Si la consulta no puede ser respondida por el FAQ Tool, el agente utiliza el Retriever Tool para buscar fragmentos de documentos relevantes. Estos fragmentos se pasan luego al LLM para generar una respuesta completa.

SerpAPI Tool:

Este componente permite al sistema realizar búsquedas en internet para encontrar información actualizada y relevante.

Camino de Búsqueda Externa: Independientemente de si la información necesaria está disponible en los documentos internos, el agente utiliza el SerpAPI Tool para buscar en la web. La información obtenida se combina con otras fuentes para generar una respuesta más completa.

FLUJO DE OPERACIÓN Y CAMINOS POSIBLES

Inicio del Proceso:

El usuario envía una consulta (query) al agente a través de la interfaz.

Evaluación Inicial por el Agent:

El agente evalúa si la consulta puede ser respondida rápidamente a través del FAQ Tool o si requiere un procesamiento más avanzado.

Camino FAQ:

Si el FAQ Tool encuentra una respuesta para la consulta, esta respuesta se envía directamente al usuario, y el proceso termina.

Camino Complejo con el LLM:

Si el FAQ Tool no puede responder la consulta, el agente decide pasar la consulta al LLM.
El LLM puede generar una respuesta completa basado en fragmentos relevantes de la base de conocimientos o combinando la información de diferentes herramientas.

Camino de Recuperación (Retriever Tool):

Cuando el LLM necesita información adicional para responder la consulta, el agente invoca al Retriever Tool.
El Retriever Tool realiza una búsqueda en la base de datos (Chroma DB) utilizando embeddings para encontrar los fragmentos más relevantes.
Los fragmentos recuperados son procesados por el LLM, que los utiliza para crear una respuesta informada.

Camino de Búsqueda Externa (SerpAPI Tool):

Si la consulta del usuario requiere información actualizada o que no se encuentra en la base de conocimientos, el agente recurre al SerpAPI Tool para realizar una búsqueda en internet.
Los resultados de esta búsqueda se integran en la respuesta generada por el LLM.

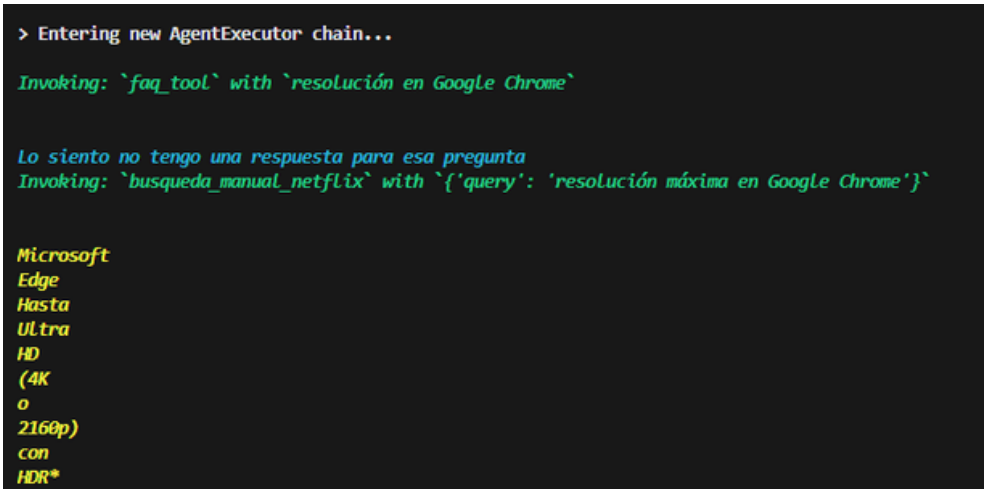
Respuesta Final:

Una vez que el agente ha recopilado toda la información necesaria, el LLM genera una respuesta combinada.
Esta respuesta es enviada al usuario a través de la interfaz de chat.

Botón Limpiar Historial:

El Agente posee asignada Memoria en este caso ConversationBufferMemory esto permite que se guarden las interacciones previas entre el User y el Agent ; en el caso de querer reiniciar la conversación desde 0 el User puede presionar el botón y esto no solo limpiará la pantalla con el historial sino que reiniciara la memoria del agente.

3. VISUALIZACIÓN DE RESULTADOS.



4. CONCLUSIONES.

Uno de los principales desafíos fue lograr que el bot se mantuviera dentro del ámbito temático de Netflix, especialmente cuando los usuarios hacían preguntas ambiguas o fuera de contexto. Para abordar este reto, incluí instrucciones claras en el prompt y diseñé un flujo de interacción que redirige amablemente las consultas fuera de alcance, recordando al usuario los temas específicos en los que el bot puede ayudar.

También enfrenté retos técnicos en el manejo de la interfaz, como el diseño responsivo del chat para evitar desbordamientos visuales. A través de varias iteraciones en CSS y ajustes en la arquitectura de Streamlit, logré una interfaz limpia y funcional que ofrece una experiencia visual agradable y fácil de usar.

Lecciones Aprendidas y Mejora Continua
A lo largo del proyecto, comprendí la importancia de un diseño de prompts sólido para guiar al modelo de lenguaje. La claridad en los objetivos, la coherencia temática y el ajuste cultural son factores cruciales para obtener respuestas efectivas y alineadas con la personalidad deseada del bot. Además, observé que la personalización de herramientas de recuperación y el ajuste de la estructura de respuesta ofrecen una mayor flexibilidad para abordar preguntas complejas sin perder precisión, permitiendo al bot adaptarse mejor a cada consulta específica.

Próximos Pasos y Posibles Mejoras
Como próximos pasos, considero la integración de nuevos módulos de aprendizaje automático que ayuden al bot a mejorar continuamente a partir de las interacciones previas, lo cual permitirá elevar la calidad de las respuestas con el tiempo.
También veo una oportunidad en implementar métricas de satisfacción del usuario en tiempo real para retroalimentar y ajustar la personalidad y el tono del bot según las necesidades de los usuarios.
Por último, podría explorar una mayor personalización del contenido según la cuenta del usuario, respetando siempre las restricciones de privacidad, para ofrecer un soporte más específico y adaptado. Esto ayudaría a crear una experiencia de soporte aún más relevante y personalizada.