

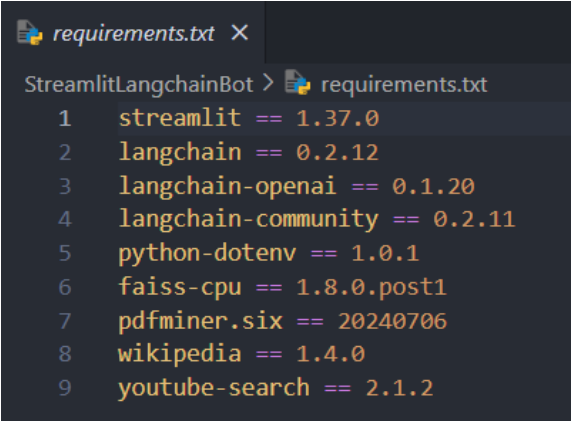
# SAMI

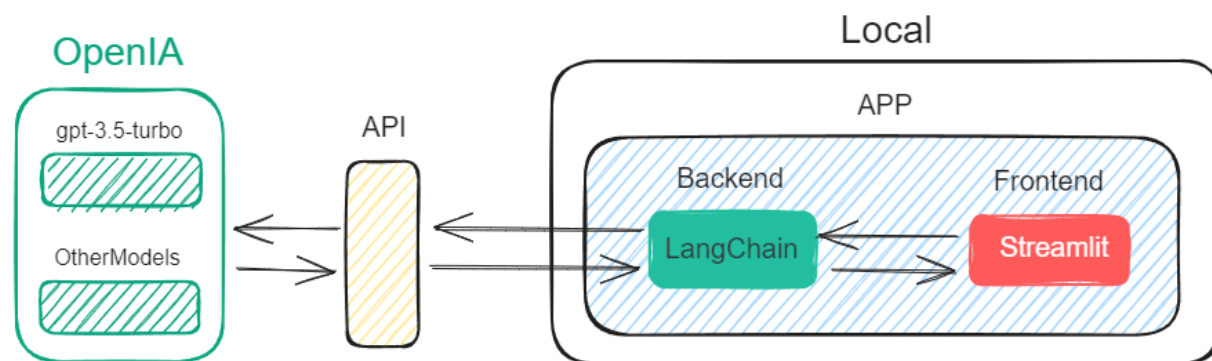


## Introducción

Este chatbot usa varias tecnologías para darte una experiencia súper fluida y eficiente

<p><b>app.py:</b></p> <p>Es como el director de orquesta que asegura que todas las partes del proyecto trabajen juntas armoniosamente para que el chatbot funcione correctamente</p>	<p>Cuando ejecutas tu chatbot, app.py es el encargado de poner en marcha todos los componentes necesarios. Aquí está el paso a paso de lo que hace:</p> <ol style="list-style-type: none"> <li>1. <b>Carga la configuración:</b> Primero, app.py lee un archivo llamado config.json para obtener toda la información necesaria sobre cómo debe funcionar la aplicación.</li> <li>2. <b>Inicializa el modelo de lenguaje:</b> Utiliza la clave API de OpenAI para configurar un modelo de lenguaje que entenderá y generará respuestas a las consultas de los usuarios.</li> <li>3. <b>Genera los embeddings:</b> Estos son representaciones matemáticas del contenido que ayudarán al chatbot a entender mejor los documentos que utilizará.</li> <li>4. <b>Procesa los documentos PDF:</b> Toma los archivos PDF, los carga y los divide en partes más pequeñas y manejables. Esto facilita el análisis y la búsqueda de información dentro de ellos.</li> <li>5. <b>Configura la personalidad del chatbot:</b> Define cómo debe comportarse el chatbot, cuál es su estilo de comunicación y otros rasgos que lo harán único.</li> <li>6. <b>Genera el historial de chat:</b> Mantiene un registro de las conversaciones anteriores para que el chatbot pueda recordar el contexto y proporcionar respuestas más relevantes.</li> <li>7. <b>Lanza la interfaz de usuario:</b> Utiliza Streamlit para crear una interfaz amigable y fácil de usar, donde los usuarios pueden interactuar con el chatbot.</li> </ol>
<p><b>config.json</b></p>	<p>Este archivo contiene la configuración necesaria para inicializar y operar el chatbot. Incluye detalles sobre la clave API de OpenAI, el modelo de embeddings, configuraciones para dividir PDFs, y aspectos relacionados con la personalidad del chatbot y la interfaz de usuario. Sirve como el manual de instrucciones que le dice a la aplicación cómo debe funcionar y qué características debe tener.</p>
<p><b>confs.py:</b></p>	<p>Este archivo es responsable de cargar las configuraciones necesarias para la aplicación. Básicamente, abre un archivo llamado config.json, lee su contenido y lo convierte en un formato que la aplicación puede entender y utilizar. Es como el manual de instrucciones que le dice a la aplicación qué hacer y cómo hacerlo.</p>
<p><b>model.py:</b></p>	<p>Aquí es donde configuramos el cerebro de nuestro chatbot: el modelo de lenguaje de OpenAI. La función init_model utiliza tu clave API para inicializar este modelo y configurarlo para que pueda generar respuestas en tiempo real. Piensa en esto como ajustar un instrumento musical para que suene perfecto.</p>
<p><b>embedder.py:</b></p>	<p>En este archivo, configuramos el generador de embeddings, que son representaciones matemáticas del contenido que el chatbot utilizará. La función init_embedder configura este generador con tu clave API y establece un sistema de almacenamiento en caché. Esto ayuda a evitar repetir el mismo trabajo, haciendo que el chatbot sea más rápido y eficiente.</p>
<p><b>document_processing.py:</b></p>	<p>Este archivo se ocupa de manejar los documentos PDF. La función pdf_splitter toma los nombres de los archivos PDF, los carga y los divide en fragmentos más pequeños. Esto hace que sea más fácil manejar y analizar el contenido de los PDFs. Es como cortar un pastel en porciones para que sea más fácil de comer.</p>
<p><b>generate_vector.py:</b></p>	<p>Aquí es donde creamos vectores a partir de los fragmentos de documentos utilizando los embeddings generados anteriormente. La función</p>

	<p>generate_documents_vectors facilita la búsqueda y recuperación rápida de información relevante en los documentos. Es como crear un índice para un libro, lo que hace que encontrar información específica sea mucho más sencillo.</p>
<b>personality.py:</b>	<p>Este archivo define la personalidad del chatbot. Utilizando la configuración proporcionada, la función personalitybot establece atributos como el nombre, la descripción, el idioma y otros rasgos del chatbot. Esto asegura que las respuestas del chatbot sean coherentes y tengan un estilo distintivo. Es como darle carácter y voz al chatbot.</p>
<b>history_prompt.py</b>	<p>Se encarga de generar un historial de chat para mejorar la recuperación de información. La función generate_prompt_history utiliza el modelo de lenguaje de OpenAI y el objeto de recuperación de información para crear un prompt de historial de chat. Esto permite que el chatbot recuerde el contexto de conversaciones anteriores y proporcione respuestas más precisas y relevantes.</p>
<b>UI.py</b>	<p>Este archivo se encarga de crear la interfaz de usuario utilizando Streamlit. Contiene varias funciones auxiliares para manejar la entrada del usuario y buscar información en Wikipedia y YouTube. La función principal generate_ui configura la interfaz, mostrando un avatar del chatbot, gestionando el historial de chat y manejando las interacciones del usuario. Es la cara visible del chatbot, lo que los usuarios verán y con lo que interactuarán.</p>
<b>requirements.txt</b>	<p>Este archivo lista todas las dependencias necesarias para ejecutar la aplicación, como Streamlit, Langchain, OpenAI y Python-dotenv. Es como una lista de compras que asegura que todas las bibliotecas y paquetes necesarios estén instalados en el entorno donde se ejecutará la aplicación.</p>  <pre> requirements.txt X StreamlitLangchainBot &gt; requirements.txt 1  streamlit == 1.37.0 2  langchain == 0.2.12 3  langchain-openai == 0.1.20 4  langchain-community == 0.2.11 5  python-dotenv == 1.0.1 6  faiss-cpu == 1.8.0.post1 7  pdfminer.six == 20240706 8  wikipedia == 1.4.0 9  youtube-search == 2.1.2 </pre>
<b>.gitignore</b>	<p>Este archivo especifica qué archivos y directorios deben ser excluidos del control de versiones, como los archivos de caché, las carpetas de bytecode compilado (<b>pycache</b>), y archivos sensibles como .env que contienen información privada. Es como una lista de cosas que no quieres llevar contigo en un viaje.</p>



## config.json

```

config.json x
StreamlitLangchainBot > config.json > {} assistantconf > [ ] other > 1
1 {
2   "modelapi": {
3     "model": "gpt-3.5-turbo",
4     "temperature": 0.8,
5     "timeout": 15,
6     "max_retries": 1
7   },
8   "pdfs": {
9     "names": ["Cuidados de Animales Domesticos y exóticos", "Cuidados Basicos de Conejos, Cobayos, Hamsters y Chinchillas"]
10  },
11  "assistantconf": {
12    "name": "SAMI",
13    "description": "conejo asistente",
14    "language": "español",
15    "personality": ["amigable", "cordial", "gracioso", "informativo"],
16    "other": [
17      "saludas de manera chistosa",
18      "si lo requiere terminas la frase con un, Lo mejor es que consultes con tu veterinario de confianza",
19      "puede hacer chistes",
20      "puede contar historias",
21      "puede hacer preguntas",
22      "Tiene un toque de calidez",
23      "está bien informado sobre alimentación, ejercicio, higiene, salud y comportamiento",
24      "responde a las preguntas de los usuarios con una actitud tranquilizadora",
25      "Su objetivo es proporcionar información precisa y útil sobre el cuidado de mascotas",
26      "El conejo tiene una corbata y un aire de sabiduría combinada con humor",
27      "es un conejo veterinario amigable, informativo, gracioso. "
28    ]
29  }
30 }
31 }
  
```

## Imagenes para mascota del chatbot





**.gitignore**

```
.gitignore x
StreamlitLangchainBot > .gitignore
1 myenv/
2 *.env
3 *cache
4 *.pyc
5 *.pyo
6 __pycache__/
7
```

**Diseño del local host**

```
config.toml X
StreamlitLangchainBot > app > .streamlit > config.toml
1  [theme]
2  primaryColor="#3282B8"
3  backgroundColor="#1B262C"
4  secondaryBackgroundColor="#0F4C75"
5  textColor="#BBE1FA"
6  font="sans serif"
```

## Documentacion utilizada



