«Talento Tech»

Inteligencia Artificial

Clase 08

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

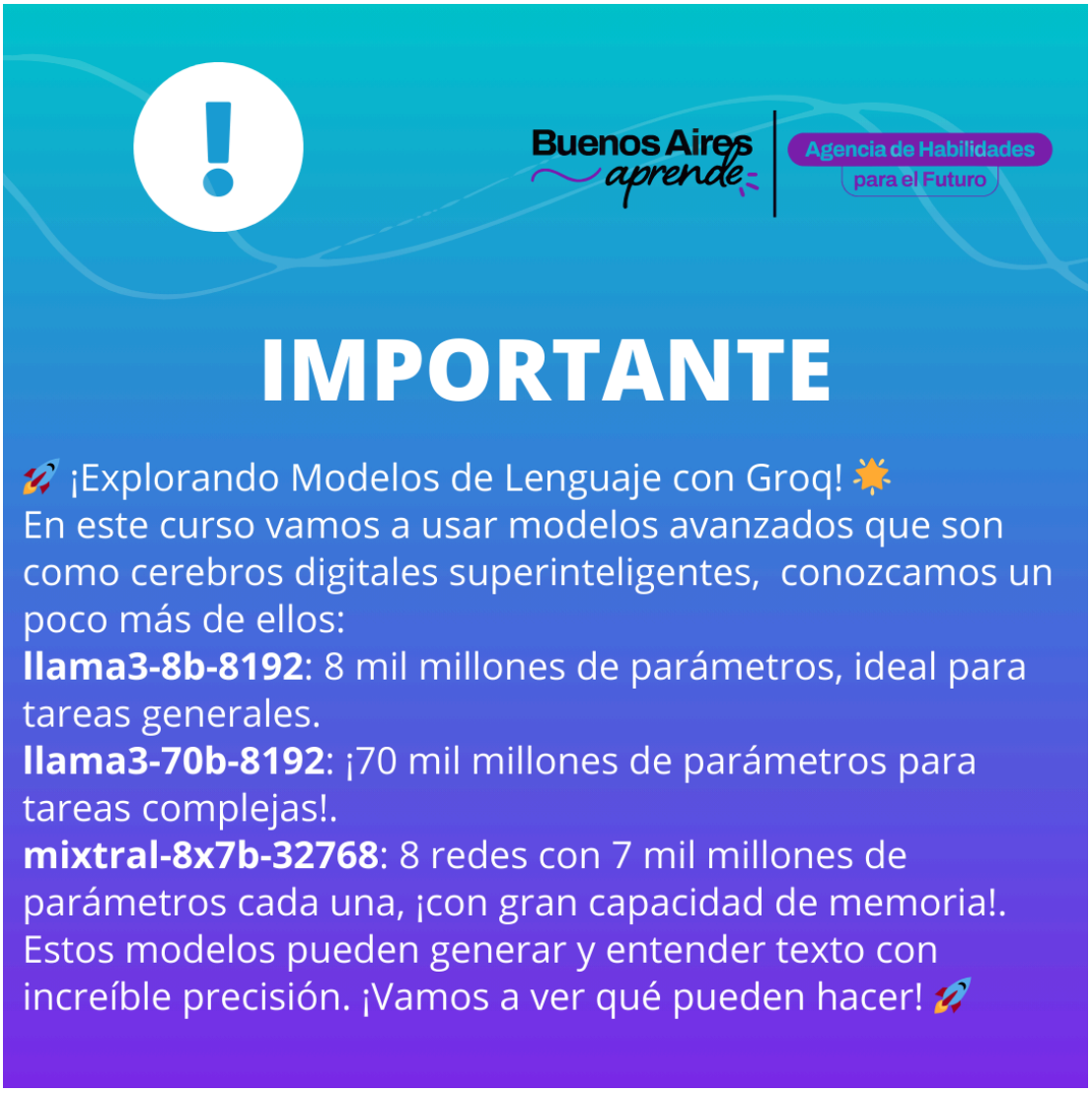
Clase N° 8 | Implementación del historial de chat

Temario:

- Manejo de historiales de chat con Streamlit.
- Modelos de lenguaje de nuestro ChatBot.¿cómo funciona?
- Modelos famosos basados en Transformers.
- Almacenamiento y actualización del historial de chat.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Repasemos nuestros modelos de lenguaje.



Vamos a retomar esta imagen para explicar un poco más desde una perspectiva más técnica nuestro chatBot en lo que respecta a estos modelos y así como es ese proceso de generar las respuestas que nos brinda.

Modelos de lenguaje que utiliza el ChatBot:

El chatbot utiliza tres modelos avanzados, cada uno con diferentes capacidades:

1. **Llama3-8b-8192**: Con 8 mil millones de parámetros, este modelo es rápido y efectivo para responder preguntas generales. Los parámetros son como "configuraciones internas" que determinan cómo el modelo procesa y genera texto.
2. **Llama3-70b-8192**: Este modelo, con 70 mil millones de parámetros, es más complejo y está diseñado para tareas más difíciles. La mayor cantidad de parámetros permite al modelo captar matices y relaciones más complejas en el texto.
3. **Mixtral-8x7b-32768**: Compuesto por 8 redes, cada una con 7 mil millones de parámetros. Este modelo tiene una gran capacidad para manejar conversaciones largas, recordando detalles y asegurando respuestas coherentes.

¿Cómo funciona nuestro ChatBot para generar respuestas?

Cuando haces una pregunta, el chatbot sigue varios pasos para generar la respuesta:

1. **División de Tokens**:
   - El texto de la pregunta se divide en unidades más pequeñas llamadas **tokens**. Un token puede ser una palabra completa, una parte de una palabra, o incluso un carácter, dependiendo del modelo.
   - Esta división es esencial porque los modelos de lenguaje no procesan palabras enteras, sino tokens, y cuantos más tokens, más trabajo tiene el modelo para entender la pregunta.
2. **Selección del Modelo**:
   - Dependiendo de la pregunta y la complejidad esperada, deberíamos elegir el modelo más adecuado. Por ejemplo, para una pregunta sencilla, puede usar Llama3-8b-8192, que es más rápido. Para preguntas complejas, podría usar Llama3-70b-8192 o Mixtral, que tienen más parámetros y pueden manejar más tokens.
3. **Procesamiento del Historial**:

- El chatbot también tiene en cuenta el historial de la conversación. Esto significa que los tokens de mensajes anteriores se combinan con los de la pregunta actual, lo que ayuda al modelo a entender mejor el contexto.

4. **Generación de Respuestas:**
   - Dentro del modelo, los **parámetros** juegan un papel vital porque deciden cómo se combinan tokens para generar la respuesta. Este proceso comienza prediciendo cuál sería el siguiente token en la secuencia. Luego, el modelo sigue prediciendo el siguiente token basado en los tokens anteriores y el contexto proporcionado por los parámetros.

     Este proceso se repite hasta que se ha construido toda la respuesta. La mayor cantidad de parámetros permite que el modelo tome decisiones más complejas y precisas para la generación de cada token.

Ejemplo: Proceso de generación de una respuesta

Ya conociendo un poco más esto, ahora vamos a ver un ejemplo de cómo funciona nuestro ChatBot a partir de una pregunta para entenderlo un poco más:

Pregunta del Usuario: *"¿Cuál es la capital de Argentina?"*

1. **División de Tokens:**
   - La pregunta se divide en tokens. Por ejemplo:
     - "¿Cuál" → `token1`
     - "es" → `token2`
     - "la" → `token3`
     - "capital" → `token4`
     - "de" → `token5`
     - "Argentina?" → `token6`
2. **Selección del Modelo:**
   - El chatbot selecciona, por ejemplo, el modelo **Llama3-8b-8192** porque es una pregunta directa, y este modelo es rápido y eficiente para responder este tipo de consultas.
3. **Procesamiento del Historial:**

- En este caso, si no hay un historial de conversación relevante, el modelo se centrará solo en la pregunta actual. Sin embargo, si hubiera preguntas previas relacionadas con países, el modelo podría usar ese contexto para mejorar la precisión.

4. **Generación de Respuesta:**
   - **Paso 1:** El modelo utiliza sus parámetros para predecir el siguiente token en la secuencia. Aquí, el primer token de la respuesta sería "Buenos".
   - **Paso 2:** Después de "Buenos", el modelo predice el siguiente token, que es "Aires".
   - **Paso 3:** Finalmente, el modelo predice un token de cierre como un punto ".".
   - **Resultado Final:** La respuesta completa generada es: *"Buenos Aires."*
5. **Visualmente:**
   - Pregunta: "¿Cuál es la capital de Argentina?"
   - Tokens: `token1` "¿Cuál" → `token2` "es" → `token3` "la" → `token4` "capital" → `token5` "de" → `token6` "Argentina?"
   - Respuesta Generada: "Buenos Aires."
6. **Simulación de Escritura en Tiempo Real:**
   - En la pantalla, la respuesta aparecería gradualmente:
     - Primero, se muestra "B"
     - Luego, "u", seguido de "e", "n", "o", "s"
     - Después, "A", "i", "r", "e", "s"
     - Finalmente, el punto "." para completar la respuesta.

Representación Visual

Podemos imaginar este proceso en un diagrama simplificado:

1. **Entrada del Usuario:** `¿Cuál es la capital de Argentina?`
   - | Token 1: "¿Cuál" | Token 2: "es" | Token 3: "la" | Token 4: "capital" | Token 5: "de" | Token 6: "Argentina?" |
2. **Procesamiento en el Modelo:**
   - | Parámetros Activos (8 mil millones) | → Predicción de Token 1: "Buenos" → Predicción de Token 2: "Aires" → Predicción de Token 3: "." |
3. **Salida del ChatBot:**
   - "Buenos Aires."

Ilustración

Visualizando este proceso, podemos imaginar:

- La pregunta está siendo dividida en tokens.
- Estos tokens entran en el modelo.
- Dentro del modelo, millones de parámetros procesan los tokens y predicen la secuencia correcta de palabras.
- Finalmente, la respuesta "Buenos Aires." aparece en la pantalla como si estuviera siendo escrita en tiempo real.

Los modelos que podemos elegir en nuestro chatbot usan una arquitectura llamada "Transformer", que hace que nuestro chat sea como un súper cerebro que puede:

a) **Entender el contexto:** Capta el significado general de lo que dices, no solo palabras sueltas.
b) **Trabajar rápido:** Procesa muchas partes de tu mensaje al mismo tiempo.
c) **Aprender de ejemplos:** Se entrena con millones de conversaciones para mejorar sus respuestas.

Estos modelos son comúnmente utilizados para implementar chatbots debido a su capacidad y velocidad para comprender y generar lenguaje humano, incluyendo la generación de texto coherente y la comprensión del contexto en conversaciones. Los Transformers son un tipo de modelo de aprendizaje profundo introducido en el artículo "Attention is All You Need" en 2017.

Modelos famosos basados en la Arquitectura transformers

Los modelos basados en arquitecturas de Transformer, como:

- **BERT**: Representaciones de codificadores bidireccionales de transformadores (Bidirectional Encoder Representations from Transformers). Este modelo se entrena para comprender el contexto en ambas direcciones (izquierda y derecha) en una

oración. Es excelente para tareas de comprensión del lenguaje, como la clasificación de textos y la respuesta a preguntas.

- **GPT**: Transformador generativo pre entrenado (Generative Pre-trained Transformer) A diferencia de BERT, que es un modelo de codificación, GPT es un modelo generativo que predice la siguiente palabra en una secuencia. Es altamente efectivo para generación de texto, por lo que es ampliamente utilizado en chatbots. La última versión, GPT-4, trae mejoras en la generación de texto, coherencia y comprensión contextuales

- **T5**: Transformador de transferencia de texto a texto (Text-to-Text Transfer Transformer) Este modelo establece un marco en el que todas las tareas de NLP(Comprensión del lenguaje natural) se ven como problemas de convertir un texto de entrada en un texto de salida, lo cual es beneficioso para tareas de generación de texto y traducción.

Esta arquitectura ha revolucionado el procesamiento del lenguaje natural (NLP) y se caracteriza por las siguientes características:

Atención: En lugar de procesar la información secuencialmente (como lo hacen las redes neuronales recurrentes), los Transformers usan un mecanismo de atención que permite al modelo enfocarse en diferentes partes de la entrada simultáneamente. Esto ayuda a capturar relaciones a largo plazo entre las palabras en un texto.

Paralelización: Gracias a la atención, pueden procesar múltiples elementos de una secuencia en paralelo, lo que aumenta la eficiencia y reduce el tiempo de entrenamiento.

Codificadores y Decodificadores: La arquitectura consta de dos partes principales; componentes de codificación que comprenden la entrada y componentes de decodificación que generan la salida. En el caso de los chatbots, estos componentes trabajan juntos para proporcionar respuestas coherentes basadas en la entrada del usuario.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Manejo de historiales de chat con Streamlit

Para manejar el historial del chat en streamlit, vamos a crear 3 funciones. En esta clase nos enfocaremos en el manejo del historial de chat, porque es una parte fundamental para mantener el contexto de la conversación.

1. Una función para actualizar el historial constantemente, lo que nos va a permitir registrar nuestra conversación en una variable
2. Una función para mostrar el historial registrado.
3. Una función que me genere un espacio donde mostrar mi chat.

Conceptos básicos que usaremos en esta clase:

Listas
Para entender nuestra primera función, será necesario recordar las propiedades de las listas. Supongamos que tengo una variable llamada mensaje en mi código:

```
mensaje = []
```

La variable “mensaje”, en este casó, es una lista vacía. Los tipos de datos **list**, permiten almacenar cualquier tipo de información. Podemos almacenar números enteros (int), números flotantes (float), etc. Entre todos los tipos de datos que podemos almacenar, también podemos almacenar valores de tipo diccionario (dict).

Ejemplo:

Las salidas por la terminal están comentadas al lado del print

```
mensaje = []
print(mensaje) # []
```

Diccionarios
Los diccionarios son típicos a la hora de almacenar pares de información. En nuestro caso, lo que queremos almacenar es el par **emisor-mensaje.** En nuestro chatbot, vamos a ser

dos: un caso será el emisor, que puedo ser yo, como “usuario o user”, y en otro caso va a ser el motor de inteligencia artificial de groq.

```
usuario= {"emisor":"usuario", "mensaje":"Holis!"}
print(usuario) # {'emisor': 'usuario', 'mensaje': 'Holis!'}
```

Método append
El método append() se utiliza para agregar elementos al final de una lista.

```
mensaje.append({"emisor":"usuario", "mensaje":"Holis!"})
print(mensaje) # [{'emisor': 'usuario', 'mensaje': 'Holis!'}]
```

Teniendo la función append y el objetivo de mi función en mente, es que vamos a crear nuestra función para almacenar el historial en mi estado de sesion.
Recordemos que funciones anteriores, guardamos en nuestra sesión (**session_state)** una variable llamada mensaje.

Funciones para el manejo del chat

Actualizar historial

Mantener el historial del chat actualizado nos va a servir para mantener una coherencia de la conversación al guardar cada interacción y nos va a permitir acceder al contexto completo, haciendo que las respuestas sean más precisas. Además le vamos a agregar un ávatar que nos permita reconocer visualmente quién está hablando.

Para crear esta primer función vamos a escribir lo siguiente:

```
def actualizar_historial(rol, contenido, avatar):
    st.session_state.mensajes.append({"role": rol, "content": contenido,
"avatar":avatar})
```

1. La función toma tres parámetros: rol (quién envía el mensaje), contenido (el mensaje en sí) y avatar que será, mas adelante, un emoji que represente al rol.

2. Utilizamos **st.session_state.mensajes** para almacenar el historial de manera persistente en Streamlit, incluso entre recargas de las páginas, en la variable mensaje.
3. Con el método append, agregamos un nuevo diccionario al final de cada lista de mensajes.

En este caso estamos agregando a nuestra variable mensaje, que es de tipo list, un diccionario, que tiene las tres claves: rol, content y avatar; y cada una tiene su valor correspondiente.
Similar al ejemplo que mencionamos anteriormente.

La variable mensajes que creamos anteriormente en la variable "**session_state"**, es una lista que la inicializamos vacía, y que con el tiempo vamos a ir agregando un diccionario que tenga el rol y el contenido. **Quien "habla" y "que dice".**

Mostrar historial
Lo que buscamos con esta función es mostrar visualmente el historial completo del chat en la interfaz de streamlit, visualizando fácilmente el flujo de la conversación.

Para hacerlo será necesario seguir los siguientes pasos:
1. Recorremos el historial y abrimos cada diccionario.
2. A cada diccionario lo divido en dos: por un lado quien el rol de quien habla y su emoi representativo; y por otro lado, que dice o el contenido.
3. Usaremos una función de streamlit llamada **st.chat_message** para crear burbujas de chat para cada mensaje.

Debemos definir la función la cual llamaremos "**mostrar_historial"** y nos quedaría de esta forma:

```
def mostrar_historial():
    for mensaje in st.session_state.mensajes:
        with st.chat_message(mensaje["role"], avatar=mensaje["avatar"]):
            st.markdown(mensaje["content"])
```

1. **"for mensaje in st.session_state.mensajes":** iteramos sobre cada mensaje almacenado en la lista mensajes, dentro de la sesión de estado.
2. Para cada mensaje, la función **st.chat_message**, crea una "burbuja" de chat visual en streamlit.

3. Dentro de cada burbuja, y con lenguaje markdown (puede ser con un write simplemente) mostramos el contenido del mensaje.

Área del chat
Una vez que tengamos las funciones para manejar nuestro historial, su contenido y su visualización, tenemos que definir en qué parte de nuestra aplicación vamos a mostrar el chat.

Para hacerlo similar a cualquier chatbot, lo haremos en el centro de nuestra aplicación encuadrado en un marco, y para asegurarnos un área correcta y que todo el chat se muestre dentro de ese área, haremos uso de la función **container** de streamlit:
sintaxis:

st.container(height, border)

Llamaré a esta función "**area_chat()"** y hará uso de la función "**mostrar_historial**" que creamos anteriormente. Nuestra función quedará de la siguiente manera:

```
def area_chat():
    contenedorDelChat = st.container(height=400,border=True)
    # Abrimos el contenedor del chat y mostramos el historial.
    with contenedorDelChat:
        mostrar_historial()
```

Creo una variable que será el propio contenedor del chat, y le asignó con el parámetro **height** qué va a tener un alto de 400 pixeles; si no uso este parámetro, el alto del chat va a depender del contenido que le pasemos. Con "border=True", le digo que marque un contorno al contenedor(un borde). Luego abro el contenedor, con la función mostrar_historial muestro el mensaje.

Organización de las funciones

De las funciones que creamos anteriormente, la función que nos muestra el área del chat, la vamos a escribir debajo de la función inicializar estado, o simplemente por fuera del if.

```
modelo = configurar_pagina()
clienteUsuario = crear_usuario_groq()
inicializar_estado()
area_chat() # Función de esta clase
mensaje = st.chat_input("Escribí tu mensaje:")
# print(mensaje)
```

IMPORTANTE: En la clase 7, nosotros creamos lo siguiente:

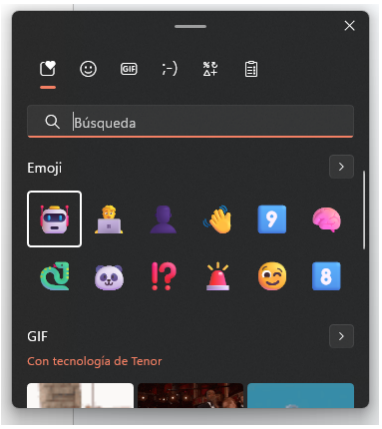
```
modelo = configurar_pagina()
clienteUsuario = crear_usuario_groq()
```

Esto debemos ir y comentarlo con #, esto principalmente porque lo vamos a usar de vuelta en la clase 8, de no comentarlo / eliminarlo nos va a crear problemas de compatibilidad.

Dentro de nuestro condicional, vamos a escribir las funciones creadas.
Buscaremos tomar el mensaje que envía el usuario, que anteriormente guardamos en una variable e imprimimos por la terminal, para pasarlo como parámetro a la función "**actualizar_historial".** Esto nos va a permitir guardar el mensaje que envía el usuario en la variable con el mismo nombre. Como tercer parámetro, le pasamos un emoji que nos represente como usuarios de un chatbot, en mi caso voy a elegir una persona con una computadora.

```
if mensaje:
        actualizar_historial("user", mensaje,"👩‍💻") # Función de esta clase
```

Para seleccionar una emoji en windows, es necesario apretar en conjunto la tecla windows, y la tecla punto: (windows + .) Al hacerlo te saldrá la siguiente pestaña donde podrás buscar todos los emojis que necesites:



Debajo crearé una variable que tenga la respuesta que nos devuelve groq, que venía almacenada en la función "**configurar_modelo"** (que vimos en clase 7).

En mi caso la variable se llamará "**chat_completo**" porque contendrá la respuesta del modelo al mensaje del usuario.

```
if mensaje:
        actualizar_historial("user", mensaje)

        chat_completo = configurar_modelo(clienteUsuario, modelo, mensaje)
```

Lo único que nos queda es actualizar el historial con la respuesta del modelo, volviendo a usar la función **actualizar_historial**, en este caso, le vamos a cambiar el rol, para que sea ***assistant* o *asistente***, el parámetro contenido, que será la respuesta de nuestro chatbot: **chat_completo**; y también le cambiaremos el avatar, para que sea representativo. Elegí un robot.

Nuestro código nos queda de esta forma:

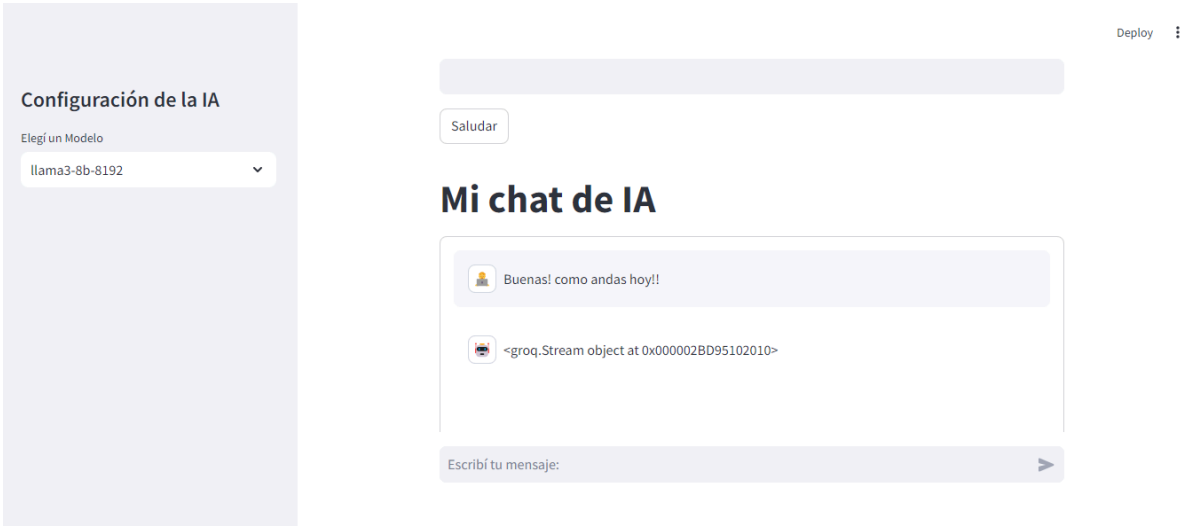
```
if mensaje:
        actualizar_historial("user", mensaje)
```

```
        chat_completo = configurar_modelo(clienteUsuario, modelo,
mensaje)
        actualizar_historial("assistant", chat_completo,"🤖")

        st.rerun()
```

🚨**Atención! Luego de hacer uso de nuestras funciones, agregamos un "st.rerun" en nuestras líneas de código. Esto permite que la página se actualice automáticamente, sin necesidad de apretar f5, o rerun, manteniéndose, por nuestra función, el historial de chat en la memoria.**

Para probar nuestro chatBot, será necesario correr nuestro código desde la terminal, con el comando ***streamlit run nombreDelArchivo.*** Una vez hecho esto, veremos que la salida por nuestro navegador será parecido a esto:



Si llegamos hasta acá quiere decir que vamos bien! ¡Felicitaciones! Continuemos viendo como darle mas estilo a nuestro chatbot y pulir detalles.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

¿Dónde están los tokens en mi chat?

Como vimos al principio de la clase, en ***¿cómo funciona nuestro ChatBot para generar respuesta?*** vimos que contabilizamos las palabras con tokens. Para aclarar la idea de los tokens en nuestro chatbot, podemos hacer la prueba. Al momento de configurar nuestro modelo, vamos a cambiar el valor de la variable stream de True a False, quedandonos el código de la siguiente manera:

```
def configurar_modelo(cliente, modelo, mensajeDeEntrada):
    return cliente.chat.completions.create(
        model=modelo,
        messages=[{"role": "user", "content": mensajeDeEntrada}],
        stream=False
    )
```

Luego corremos el código como lo venimos haciendo, y vemos que en nuestra pantalla vamos a ver lo siguiente:

Buenos Aires aprende
Agencia de Habilidades para el Futuro

En esta imágen vemos que al enviarle un mensaje simple, el chat nos devuelve (en verder) el contenido de la respuesta que tenemos que obtener para nuestro chat, (amarillo) el rol que le dimos y que representará la respuesta del chatbot, (celeste) y todos los tokens que usamos. Vemos que tenemos tokens por el prompt que le armamos, como también tokens por el mensaje que nos devuelve. En este caso la partición en tokens no es ni siquiera letra por letra, y es una particularidad del algoritmo que nos dió, como se ve en la imágen (buscando un poquito) el modelo llama3-8b-8192. Ambos tokens se suman y hacen parte de la respuesta final.

🎁 **Te invitamos a que veas la suma de tokens que te da la misma pregunta con diferentes modelos. Además de la resolución de alguna pregunta técnica que se te ocurra y su comparación.**

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Personalización de mi chatbot

Tengo varias formas de personalizar mi chatbot, de forma tal de poder darle una distinción a nuestro chatbot. En esta clase vamos a ver cómo podemos personalizar nuestro chatBot con colores.

Antes de cambiar nuestros colores, debemos saber que en streamlit podemos acceder a 3 colores de nuestra aplicación. Miremos como:

VideoExplicativo - Editado

Dentro de esas tres opciones, podemos agregar una más, agregando un archivo a la configuración de nuestra aplicación.

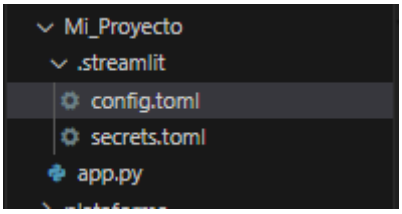
Para personalizar los colores, tengo que tener en cuenta que streamlit tiene su configuración especial. Al igual que guardar nuestros secretos y demás archivos en una carpeta **.streamlit** y un archivo **secrets.toml**, al momento de generar la configuración de los colores de nuestra aplicación, debemos crear, en esa misma carpeta, un archivo llamado **config.toml**

😉**Nota ¿que es un formato .toml? ¿por qué lo usamos tanto?**
TOML es un formato de archivo de configuración que es fácil de leer debido a la semántica obvia que pretende ser "mínima". TOML está diseñado para mapear inequívocamente un diccionario. El nombre es un acrónimo de "Tom's Obvious, Minimal Language"(El lenguaje mínimo y obvio de Tom) que hace referencia a su creador: Tom Preston-Werner. En software, se usa este formato para manejar configuraciones de un sistema, sin acceder a configuraciones que pongan en riesgo al producto.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Paso a paso

Dentro de mi carpeta .streamlit voy a crear un archivo llamado “**config.toml”** y para que se ve debe quedar de la siguiente manera:



🚨**importante! Para que se vean los cambios que hago en este archivo, tengo que reiniciar la aplicación. ¡No es lo mismo que recargar la página! Para reiniciar mi aplicación, desde la terminal, hago Ctrl + c (control + c) y vuelvo a correr mi aplicación con streamlit run nombreDeMiArchivo.**

La existencia de este archivo, crea una 4 opción de temas en la lista de settings que vimos anteriormente. Esta opción es “**Custom Theme**”.
En este archivo voy a poder escribir, bajo la clave de [theme], todos los colores que le quiera agregar a mi archivo. El formato de escritura de las claves será en inglés y los colores se recomienda agregarlos en hexadecimal. Nuestro archivo quedaría así.

```
[theme]
primaryColor="fd4b4b"
backgroundColor="#ffc689"
secondaryBackgroundColor="fdb466"
```

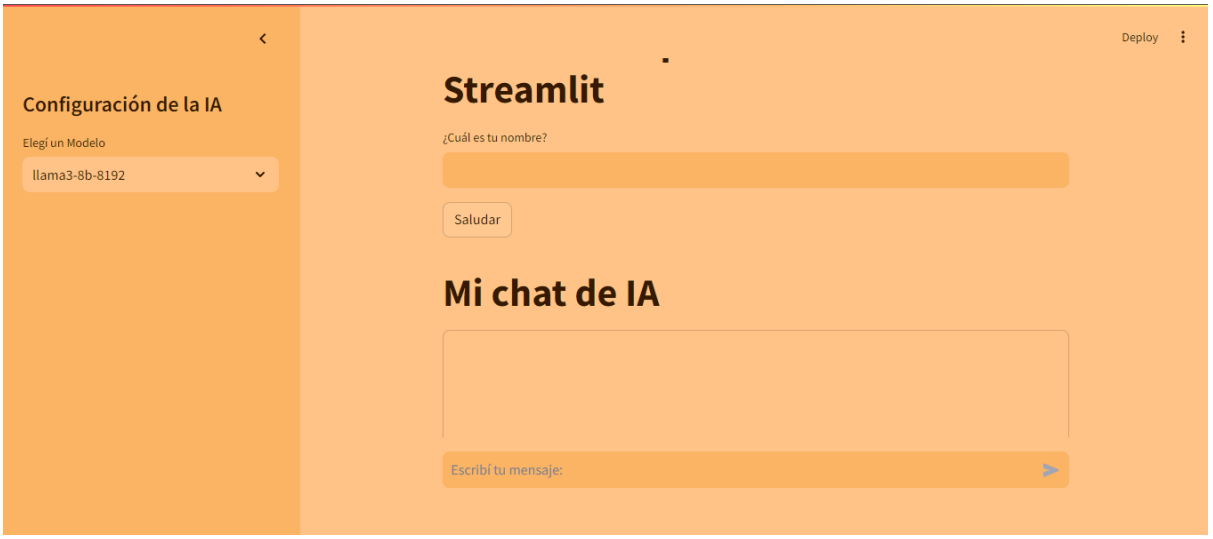
```
textColor="371d01"
font="sans serif"
```

Si tenés dudas de qué colores usar te recomiendo esta página: coloresCodes.

Con esta configuración damos a la aplicación las siguientes características:

1. **primaryColor** : Define el color primario de la aplicación, y se usa típicamente para elementos interactivos como botones, enlaces, etc.
2. **backgroundColor:** Define el color de fondo principal de la aplicación y se aplicará al fondo general de la aplicación.
3. **secondaryBackgroundColor:** Define un color de fondo secundario y se usa generalmente para nuestra barra lateral, el fondo de los widgets y otros elementos de interfaz.
4. **textColor:** Define el color del texto principal de la aplicación.
5. **font:** Especifica la familia de fuentes. Streamlit tiene tres opciones: "sans serif", "serif" y "monospace", siendo "sans serif" la opción predeterminada.

Nuestra página con esa configuración se verá de esta forma:



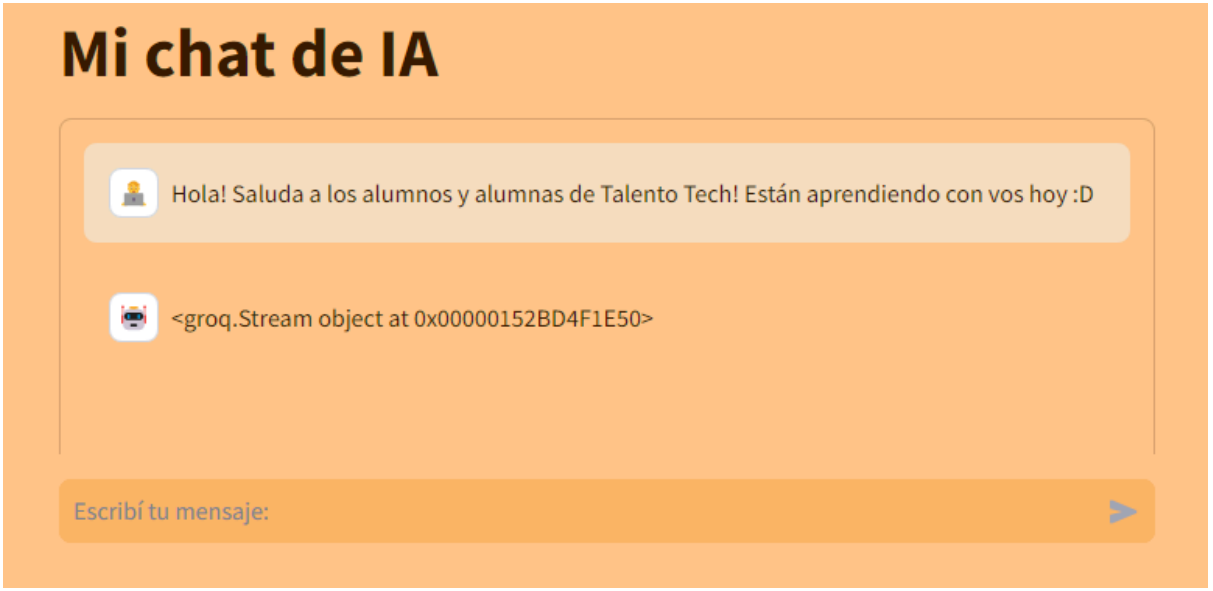
Estilo decidido
Si quiero usar de base un color ya definido como el dark o light y cambiar solamente algunas cosas, puedo usar de base un tema, y cambiar la característica que quiero. Para eso escribo el código de la siguiente manera:

```
[theme]
base="dark"
primaryColor="#ffc689"
```

De esta forma, cuando pase el tema de mi aplicación a dark, será el mismo dark que viene predeterminado todas las aplicaciones de streamlit, pero con la particularidad de que el color primario será #ffc689.

¿Qué es lo que viene?

Vimos que hasta acá obtenemos la respuesta de nuestro chat bot pero no es algo legible para nosotros. Lo que vemos en la salida del chatbot es lo siguiente:



¿Qué es esto que estamos viendo? ¿Por qué no veo una respuesta coherente?

Esto se debe a que Streamlit, no nos muestra directamente el resultado de la consulta, sino que nos devuelve un objeto que representa el "*flujo de datos que se obtendrá de la consulta"*
Esto le permite a Streamlit renderizar de manera eficiente el contenido a medida que se va obteniendo (stream = True de la función "**configurar_modelo"**).

Podemos pensarlo como si groq nos devuelva una caja con el contenido de la respuesta, y streamlit necesita de esa caja ir sacando palabra por palabra para armarla en la aplicación.
Esa función la terminamos de armar la clase que viene.
También seguiremos agregando más estilo a nuestro chat bot.

Desafío N° 8:

Buenos Aires aprende
Agencia de Habilidades para el Futuro

En este desafío, vamos a entregar nuestra aplicación, con un color característico. Te recomiendo que sigas los siguientes pasos para entregar la aplicación y agregarle el estilo que mas te guste.

1. Elegí una paleta de colores. Lo podemos elegir en la página que mencionamos o preguntarle a otro chat bot (por el momento)
2. Creá un archivo **config.toml** en la ruta correspondiente y asignar las características dentro de la variable [theme]
3. Corré tu aplicación de Streamlit con el comando ***streamlit run nombreDelArchivo*** y sacale captura para luego subirlo al campus virtual.


Buenos Aires
aprende
Buenos
Aires
Ciudad