

# Lab.3 - Chatbot

David Díaz Suesca, Jonatan Díaz Vargas, David Santiago López

Septiembre 2025

## 1. Introducción

Este laboratorio consistió en la realización y despliegue de un chatbot por DeepSeek. Se presentará a continuación, el paso a paso en su realización y despliegue, y el funcionamiento del código.

## 2. Primeros pasos

Primeramente, se tendrá en cuenta el ejemplo del código del chatbot brindado por el profesor. Este será un ejemplo para poder realizar el chatbot propuesto en el laboratorio.

Principalmente, se tendrá que crear un entorno virtual, instalar streamlit, requests y sus dependencias.

```
Python 3.11.7 Shell
~/Desktop/Archivos
$ python -m venv venv
$ source venv/bin/activate
$ pip install streamlit
Collecting streamlit
  Using cached streamlit-1.38.0-py3-none-any.whl (18.5 MB)
Collecting altair<5.0,>=4.0
  Using cached altair-4.2.2-py3-none-any.whl (11.4 MB)
Collecting blinker
  Using cached blinker-1.8.2-py3-none-any.whl (10.0 kB)
Collecting click
  Using cached click-8.1.8-py3-none-any.whl (10.0 kB)
Collecting gitpython
  Using cached gitpython-3.1.43-py3-none-any.whl (176 kB)
Collecting httpx
  Using cached httpx-0.27.0-py3-none-any.whl (76 kB)
Collecting jinja2
  Using cached jinja2-3.1.4-py3-none-any.whl (133 kB)
Collecting jsonschema
  Using cached jsonschema-4.21.1-py3-none-any.whl (85 kB)
Collecting markdown
  Using cached markdown-3.6-py3-none-any.whl (36 kB)
Collecting markupsafe
  Using cached markupsafe-2.1.5-py3-none-any.whl (10.0 kB)
Collecting numpy
  Using cached numpy-2.0.2-cp311-cp311-macosx_11_0_arm64.whl (11.1 MB)
Collecting packaging
  Using cached packaging-24.1-py3-none-any.whl (55 kB)
Collecting pandas
  Using cached pandas-2.2.2-cp311-cp311-macosx_11_0_arm64.whl (12.5 MB)
Collecting pillow
  Using cached pillow-10.4.0-cp311-cp311-macosx_11_0_arm64.whl (3.8 MB)
Collecting plotly
  Using cached plotly-5.22.0-py3-none-any.whl (17.6 MB)
Collecting pyarrow
  Using cached pyarrow-17.0.0-cp311-cp311-macosx_11_0_arm64.whl (10.0 MB)
Collecting pydeck<0.9.0,>=0.8.0b1
  Using cached pydeck-0.8.0b1-py3-none-any.whl (5.7 MB)
Collecting requests
  Using cached requests-2.32.3-py3-none-any.whl (15 kB)
Collecting tenacity
  Using cached tenacity-9.0.0-py3-none-any.whl (61 kB)
Collecting toml
  Using cached toml-0.10.2-py3-none-any.whl (16 kB)
Collecting typing-extensions
  Using cached typing_extensions-4.12.2-py3-none-any.whl (36 kB)
Collecting urllib3
  Using cached urllib3-2.2.3-py3-none-any.whl (121 kB)
Collecting watchdog
  Using cached watchdog-4.0.2-py3-none-any.whl (21 kB)
Collecting xyzservices
  Using cached xyzservices-2025.10.1-py3-none-any.whl (138 kB)
Installing collected packages: altair, blinker, click, gitpython, httpx, jinja2, jsonschema, markdown, markupsafe, numpy, packaging, pandas, pillow, plotly, pyarrow, pydeck, requests, tenacity, toml, typing-extensions, urllib3, watchdog, xyzservices
Successfully installed altair-4.2.2 blinker-1.8.2 click-8.1.8 gitpython-3.1.43 httpx-0.27.0 jinja2-3.1.4 jsonschema-4.21.1 markdown-3.6 markupsafe-2.1.5 numpy-2.0.2 packaging-24.1 pandas-2.2.2 pillow-10.4.0 plotly-5.22.0 pyarrow-17.0.0 pydeck-0.8.0b1 requests-2.32.3 tenacity-9.0.0 toml-0.10.2 typing-extensions-4.12.2 urllib3-2.2.3 watchdog-4.0.2 xyzservices-2025.10.1
```

Figura 1: Creación de entorno virtual y librerías.

Se crea un archivo ".py" que será nuestro chatbot integrado con Streamlit. A continuación, se explicará el desarrollo del código para el chatbot.



Figura 2: Archivo en Github y visualización del README.

Teniendo listo el archivo del chatbot, su integración con streamlit se hará a través de los siguientes pasos:

- Instalar git en Linux:

```
# apt-get install git
```

- por medio de la terminal, indicar usuario y correo:

```
git config --global user.name "Tu Nombre Completo"
git config --global user.email "tu-email@gmail.com"
```

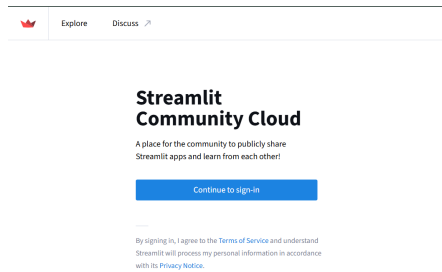
- realizar el proceso correspondiente a la carga de archivos a github (nuevo repositorio, commit, etc.):

```
git init
git remote add origin https://github.com/TU-USUARIO/NOMBRE-DE-TU-REPO.git
git add chatbot_streamlit.py
git commit -m "más tokens"
git push -u origin main
```

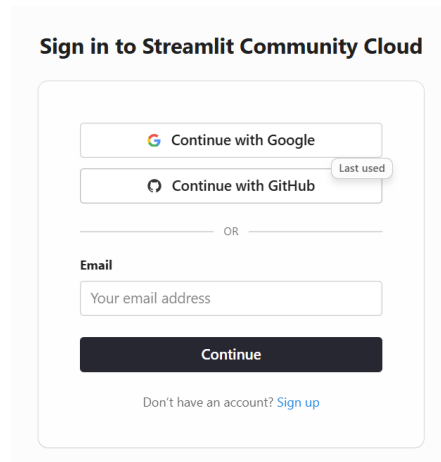
El importe se hizo desde el Sistema Operativo LINUX/UBUNTU, para esto, es necesario crear un **Personal Access Token**, para crearlo, los pasos son los siguientes:

1. Ve a github.com y haz login
2. Haz clic en tu foto de perfil (esquina superior derecha)
3. Selecciona "Settings"
4. En el menú izquierdo, busca "Developer settings" (al final)
5. Haz clic en "Personal access tokens" → "Tokens (classic)"
6. Haz clic en "Generate new token" → "Generate new token (classic)"
7. Dale un nombre al token (ej: "Token LINUX")
8. Selecciona la expiración del mismo token
9. En permisos, marca repo" (esto incluye todos los permisos necesarios)
10. Haz clic en "Generate token"
11. ¡IMPORTANTE! Copia el token que aparece (empieza con **ghp\_**) y guárdalo en un lugar seguro.

Una vez realizado este paso, se vuelve a intentar el **push** y cuando se pida el usuario, se coloca el usuario de GitHub y la contraseña será el nuevo token creado



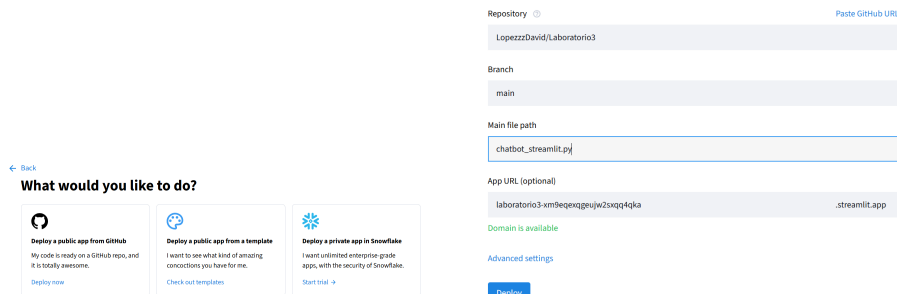
(a) Streamlit Sign-in.



(b) Opción de inicio de sesión: GitHub.

Figura 3: Creación de cuenta vinculada al GitHub.

- crear una cuenta de streamlit vinculada con el perfil de github:
- En streamlit crear una app que sea con el repositorio y archivo correspondiente: Primeramente ir a la esquina superior derecha donde dice "Create APP":



(a) Crear APP con GitHub.

(b) Selección de Repo y archivo.

Figura 4: Creación de APP con Streamlit.

- Ejecutar el streamlit y verificar correcto funcionamiento.

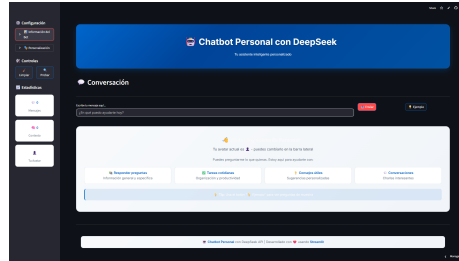


Figura 5: Despliegue del Chatbot en Streamlit.

### 3. Explicación del código

#### 3.1. importación de librerías

Al inicio del código, se deben importar todas la librerías necesarias para poder utilizar los recursos correspondientes. Por ejemplo, la librería "streamlit" genera el framework para crear la interfaz web. La librería requests importa las peticiones y genera la conexión.

```
Code Blame 550 lines (466 loc) · 21.2 KB
1 import streamlit as st
2 import requests
3 import json
4 from datetime import datetime
5 import time
6
```

Figura 6: Importe de librerías. Visualización en GitHub.

Se realiza principalmente el importe de librerías necesarias para el funcionamiento de lo requerido:

- **streamlit**: Framework para crear la interfaz web.
- **requests**: Para hacer peticiones HTTP a la API de DeepSeek.
- **json**: Para manejar datos en formato JSON.
- **datetime**: Para trabajar con fechas y horas.
- **time**: Para pausas y control de tiempo.

### 3.2. Clase ChatbotConfig.

```
Code Blame 558 lines (466 loc) - 21.2 KB
6
7 # Configuración del chatbot
8 class ChatbotConfig:
9     def __init__(self):
10         self.api_key = 'sk-53751dc6f44a28c0671de9f5111e'
11         self.api_url = 'https://api.deepseek.com/v1/chat/completions'
12         self.modelo = 'deepseek-chat'
13
14         # PERSONALIZACIÓN: Define la personalidad de tu chatbot
15         self.personalidad = {
16             "nombre": "Asistente Personal",
17             "rol": "un asistente útil y amigable",
18             "tono": "profesional pero cercano",
19             "especialidad": "ayudar con preguntas generales y tareas cotidianas",
20             "idioma": "español",
21         }
22
23         # Sistema de prompt personalizable
24         self.sistema_prompt = """
25         Eres {self.personalidad['nombre']}, {self.personalidad['rol']}.
26         Tu tono de comunicación es {self.personalidad['tono']}.
27         Te especializas en {self.personalidad['especialidad']}.
28         Siempre respondes en {self.personalidad['idioma']}.
29
30         Comportamientos específicos:
31         - Sé conciso pero completo en tus respuestas.
32         - Si no sabes algo, admítelo honestamente.
33         - Ofrece ejemplos cuando sea útil.
34         - Mantén un tono amigable y profesional.
35
36         Responde de manera natural y conversacional.
37         """
38
```

Figura 7: clase Chatbotconfig.

Estas líneas de código almacenan la configuración del chatbot:

- `API_KEY`: Tu clave secreta para acceder a DeepSeek
- `API_URL`: Dirección del servidor de DeepSeek
- `modelo`: Tipo de modelo de IA a usar
- `Self.personalidad`: Define la personalidad del bot (como su ADN)
- `self.sistema_prompt`: Crea las instrucciones secretas que se le da a la IA (Es como decirle .actúa de esta manera.antes de cada conversación).

### 3.3. Clase ChatbotPersonalizado.

```
30 class ChatbotPersonalizado:
31     def __init__(self):
32         self.config = ChatbotConfig()
33         self.max_historial = 10 # Mantiene las últimas 10 interacciones
34
35     def agregar_al_historial(self, rol, contenido):
36         """Agrega un historial de conversación para contexto"""
37         if "historial_conversacion" not in st.session_state:
38             st.session_state.historial_conversacion = []
39
40         st.session_state.historial_conversacion.append({"rol": rol, "contenido": contenido})
41
42     # Limitar el tamaño del historial
43     if len(st.session_state.historial_conversacion) > self.max_historial * 2:
44         st.session_state.historial_conversacion = st.session_state.historial_conversacion[-self.max_historial * 2:]
45
46     def preparar_mensajes(self, mensaje_usuario):
47         """Prepara los mensajes incluyendo el sistema prompt y el historial"""
48         mensajes = [{"rol": "system", "contenido": self.config.sistema_prompt}]
49
50         # Agregar historial de conversación
51         if "historial_conversacion" in st.session_state:
52             mensajes.extend(st.session_state.historial_conversacion)
53
54         # Agregar mensaje actual del usuario
55         mensajes.append({"rol": "user", "contenido": mensaje_usuario})
56
57         return mensajes
58
59     def enviar_mensaje(self, mensaje_usuario):
60         """Envía mensaje a la API con personalización y contexto"""
61         headers = {
62             "Authorization": f"Bearer {self.config.api_key}",
63             "Content-Type": "application/json"
64         }
```

Figura 8: Clase Chatbot Personalizado.

Primero, se Crea una instancia del chatbot, se Carga la configuración y se Limita el historial a 10 conversaciones para no saturar. Luego, Guarda la conversación para que el bot recuerde lo que hablaron, `st.session_state` es como la "memoria" de Streamlit. Cada mensaje se guarda con su rol ("user" o "assistant").

De esta clase, las siguientes funciones hacen lo siguiente:

- **preparar\_mensajes:** Organiza todos los mensajes antes de enviarlos a la API en el siguiente orden:
  1. las instrucciones del sistema.
  2. la conversación anterior.
  3. el mensaje nuevo del usuario.
- **enviar\_mensaje:** Prepara la petición HTTP con la autorización (Es como mostrar tu credencial" para usar la API).
- **temperature:** Controla qué tan creativo es el bot (0=robótico, 1=muy creativo)
- **max\_tokens:** Límite de palabras en la respuesta

Después de esto, se utiliza el `try` para el manejo de excepciones, en este caso, será para la petición de conexión del servidor de Deepseek.

### 3.4. Función main.

Esto Configura la página web con título, ícono, layout ancho, sidebar abierto.

```

Code Blame 558 lines (466 loc) · 21.2 KB
120 def main():
121     # Configuración de la página
122     st.set_page_config(
123         page_title="🤖 Chatbot Personal",
124         page_icon="🤖",
125         layout="wide",
126         initial_sidebar_state="expanded"
127     )

```

Figura 9: Función main.

### 3.5. CCS Personalizado.

```

Code Blame 558 lines (466 loc) · 21.2 KB
120 def main():
121     # Configuración de la página
122     st.set_page_config(
123         page_title="🤖 Chatbot Personal",
124         page_icon="🤖",
125         layout="wide",
126         initial_sidebar_state="expanded"
127     )
128
129     # Estilos CSS personalizados
130     st.markdown("""
131     <style>
132     /* Importar fuentes modernas */
133     @import url('https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700&display=swap');
134
135     .main-header {
136         text-align: center;
137         padding: 1.5em 0;
138         background: linear-gradient(135deg, #0066CC 0%, #00A4D9 33%, #00D2FF 100%);
139         color: white;
140         border-radius: 10px;
141         margin-bottom: 20px;
142         box-shadow: 0 4px 8px rgba(0, 102, 204, 0.3);
143         border: 1px solid rgba(255, 255, 255, 0.1);
144     }
145
146     .main-header h1 {
147         font-family: 'Inter', sans-serif;
148         font-weight: 700;
149         margin-bottom: 0.5em;
150         text-shadow: 0 2px 4px rgba(0, 0, 0, 0.3);
151     }
152
153     .main-header p {
154         font-family: 'Inter', sans-serif;
155         font-weight: 400;
156         opacity: 0.8;
157     }
158
159     .chat-message {
160         padding: 1.2em;
161         border-radius: 10px;
162         margin: 8px 0;
163     }

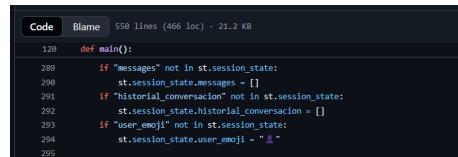
```

Figura 10: Sección del CSS personalizado.

Esta sección define todos los estilos visuales del chatbot:

- **Importa la fuente Inter** de Google.
- **.main-header**: El título principal con gradiente azul
- **.chat-message**: Estilo de los mensajes
- **.user-message**: Mensajes del usuario (azul, a la derecha)
- **.bot-message**: Mensajes del bot (gris, a la izquierda)
- **.metric-container**: contiene un Card de estadísticas

### 3.6. Inicialización del SESSION STATE.



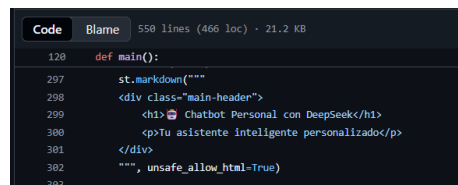
```
Code Blame 550 lines (466 loc) · 21.2 KB
120 def main():
289     if "messages" not in st.session_state:
290         st.session_state.messages = []
291     if "historial_conversacion" not in st.session_state:
292         st.session_state.historial_conversacion = []
293     if "user_emoji" not in st.session_state:
294         st.session_state.user_emoji = "🐼"
295
```

Figura 11: Iniciar Session State.

Esto inicializa la memoria de la aplicación:

- **messages**: Lista de mensajes para mostrar en pantalla
- **historial\_conversacion**: Conversación para enviar a la API
- **user\_emoji**: Emoji personalizable del usuario

### 3.7. Header principal.



```
Code Blame 550 lines (466 loc) · 21.2 KB
120 def main():
297     st.markdown("""
298     <div class="main-header">
299         <h1>🐼 Chatbot Personal con DeepSeek</h1>
300         <p>Tu asistente inteligente personalizado</p>
301     </div>
302     """, unsafe_allow_html=True)
303
```

Figura 12: Código Header principal.

Esta sección crea el título principal que se muestra en la parte superior de la página.

### 3.8. Sidebar.

Esta sección Muestra información actual del bot en la barra lateral y se actualiza automáticamente cuando se cambia la personalidad.



Figura 13: parte del código del sidebar.

[illegible]

Esta sección del código crea una cuadrícula de 16 emojis (4 columnas x 4 filas) que se guarda en `session_state`, Muestra mensaje de confirmación y Recarga la página para aplicar cambios.

```

Code      Blame  550 lines (446 loc) ~ 21.2 KB
129 def wml{()}:
308
309   st->markdown("## $@ Controls")
310
311   # Metrics en cada personalización
312
313   st->class_widget_container{
314     col1, col2 = st.columns(2)
315
316     with col1:
317       if st.button("Linear", type="secondary", use_container_width=True):
318         st.session_state.settings = [
319           st.session_state.historical_conversation - [
320             st.session_state.messages
321           ],
322           st.session_state.messages
323         ]
324
325         st.rerun()
326
327     with col2:
328       if st.button("Q. Probe", use_container_width=True):
329         with st.spinner("Probing..."):
330           response, error = chatbot_colorama("this, generate a response")
331
332           if error:
333             st.error("X Error")
334
335         else:
336           st.success("Connecto")

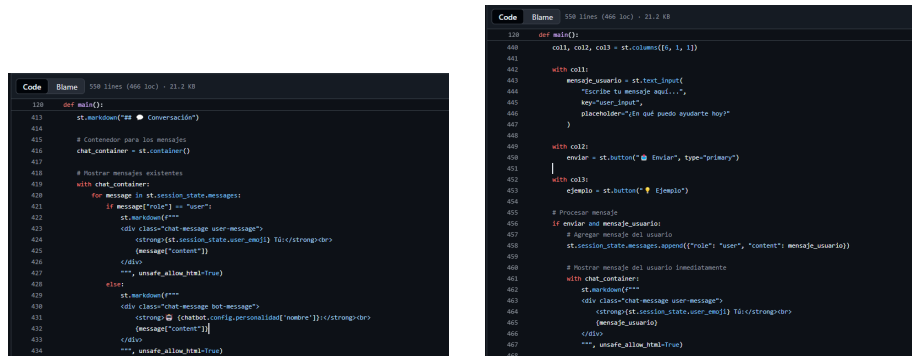
```

Figura 15: Controles y métricas para la APP.

9

- Número de mensajes enviados
- Número de conversaciones en contexto
- Avatar actual del usuario

### 3.11. Entrada del mensaje y área de conversación.



(a) Área de la conversación.

(b) Entrada para el mensaje.

Figura 16: Configuración para el área de conversación.

Esta sección del código crea el área principal del chat y muestra todos los mensajes guardados. Los mensajes del usuario aparecen a la derecha (azul) mientras que los del bot aparecen a la izquierda (gris).

Después de este punto, se procesan los mensajes que se quieren enviar y sucede lo siguiente:

- Guarda el mensaje en `messages`
- Muestra spinner de "pensando..."
- Envía mensaje a la API de DeepSeek
- Si hay error entonces lo muestra
- Si funciona, guarda la respuesta
- Recarga la página para mostrar todo

Se añadió un botón de ejemplo que tiene una lista de preguntas predefinidas, elige una al azar, la pone automáticamente en la caja de texto y carga la página con la pregunta.

Se agregó un mensaje de bienvenida que se muestra cuando la página ha sido recientemente cargada. El mensaje presenta al bot y sus capacidades y también

muestra el avatar actual.

Por último, se tiene un pie de página y se ejecuta la función principal `main`.

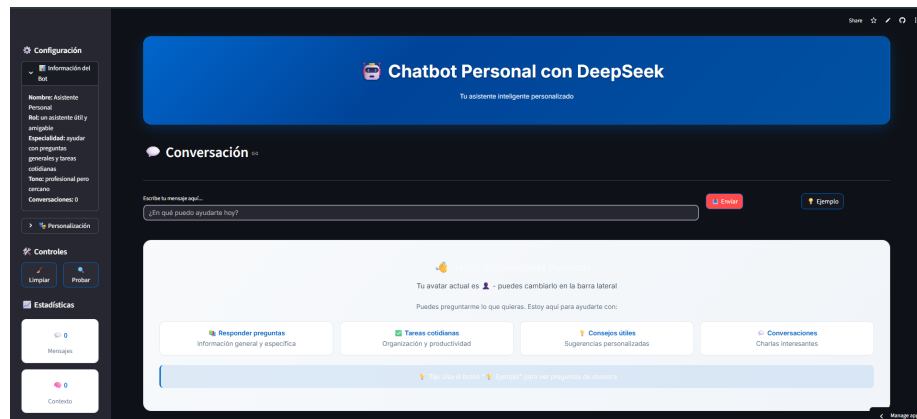


Figura 17: Despliegue del Chatbot en Streamlit.