**Desarrollo de Chatbot con Gemini y Flask**

**Introducción a los Chatbots**

Un chatbot es un programa de ordenador diseñado para simular una conversación con usuarios humanos, especialmente a través de Internet. Los chatbots se utilizan en diversas aplicaciones, como atención al cliente, asistencia virtual, marketing y entretenimiento. Pueden basarse en reglas predefinidas o en modelos de inteligencia artificial más avanzados, como los modelos de lenguaje grande (LLM), para comprender y responder al lenguaje natural.

**Alcance del Desarrollo**

Este documento describe el desarrollo de un chatbot utilizando el modelo de lenguaje Gemini de Google y el framework Flask de Python. El chatbot está diseñado para ser versátil, permitiendo cambiar su rol y comportamiento mediante la modificación de una variable en el código. La interfaz de usuario se implementa mediante HTML, y la lógica del chatbot se separa en un archivo Python.

**Requerimientos**

**Librerías de Python**

* **Flask:** Framework para crear aplicaciones web.
* **google-generativeai:** Biblioteca para interactuar con los modelos Gemini de Google.
* **python-dotenv:** Biblioteca para cargar variables de entorno desde un archivo .env.

**Entorno**

* Python 3.x
* Una clave de API de Google Cloud AI Studio.
* Un archivo .env para almacenar la clave de API de forma segura.

**Licencias**

* El código de Python desarrollado en este documento es de uso libre.
* Las librerías de Python utilizadas (Flask, google-generativeai, python-dotenv) tienen sus propias licencias de código abierto.
* El uso de los modelos Gemini de Google está sujeto a los términos de servicio de Google Cloud.

**Explicación del Código por Bloques**

El proyecto se compone de los siguientes archivos:

#### 1. chatbot\_logic.py

Este archivo contiene la lógica principal del chatbot.

import google.generativeai as genai

import os

from dotenv import load\_dotenv

# Carga las variables de entorno desde el archivo .env

load\_dotenv()

# Carga la clave de API de Google AI Studio

GOOGLE\_API\_KEY = os.environ.get("GOOGLE\_API\_KEY")

if not GOOGLE\_API\_KEY:

raise ValueError("Por favor, establece la variable de entorno GOOGLE\_API\_KEY en el archivo .env.")

genai.configure(api\_key=GOOGLE\_API\_KEY)

# Define el rol del bot aquí

BOT\_ROLE = "Car seller" # Puedes cambiar esto a cualquier rol que desees

PAGE\_TITLE = "Chatbot Vendedor de Coches"

INPUT\_PLACEHOLDER = "Pregunta sobre coches..."

model = genai.GenerativeModel('gemini-2.0-flash-lite')

def generate\_response(prompt):

chat = model.start\_chat(history=[])

# Incluye el rol en el prompt

role\_prompt = f"Como {BOT\_ROLE}, responde a lo siguiente: {prompt}"

response = chat.send\_message(role\_prompt)

return response.text

if \_\_name\_\_ == '\_\_main\_\_':

# Ejemplo de uso (para probar en JupyterLab o como script independiente)

user\_input = "Cuéntame sobre un SUV fiable."

bot\_response = generate\_response(user\_input)

print(f"Usuario: {user\_input}")

print(f"Bot ({BOT\_ROLE}): {bot\_response}")

* **Importaciones:** Importa las bibliotecas necesarias.
* **Carga de la clave de API:** Carga la clave de API de Google Cloud AI Studio desde un archivo .env para mayor seguridad.
* **Variables de Configuración:** Se definen las variables BOT\_ROLE, PAGE\_TITLE, e INPUT\_PLACEHOLDER para personalizar el comportamiento y la apariencia del chatbot.
* **Inicialización del modelo Gemini:** Inicializa el modelo de lenguaje Gemini.
* **Función generate\_response:** Toma un prompt del usuario, añade el rol del bot al prompt y genera una respuesta utilizando el modelo Gemini.
* **Bloque if \_\_name\_\_ == '\_\_main\_\_'::** Proporciona un ejemplo de cómo usar la función generate\_response cuando el script se ejecuta directamente.

#### 2. app.py

Este archivo contiene el código de la aplicación Flask, que maneja la interfaz de usuario web.

from flask import Flask, render\_template, request

from chatbot\_logic import generate\_response, BOT\_ROLE, PAGE\_TITLE, INPUT\_PLACEHOLDER

import sys

import os # Importa el módulo os

from dotenv import load\_dotenv

load\_dotenv()

# Load Google AI Studio API key

GOOGLE\_API\_KEY = os.environ.get("GOOGLE\_API\_KEY")

if not GOOGLE\_API\_KEY:

raise ValueError("Please set the GOOGLE\_API\_KEY in the .env file.")

app = Flask(\_\_name\_\_)

print(f"Python Executable: {sys.executable}")

@app.route("/")

def index():

# Pasa las variables a la plantilla

return render\_template("index.html", page\_title=PAGE\_TITLE, input\_placeholder=INPUT\_PLACEHOLDER)

@app.route("/get\_response", methods=["POST"])

def get\_response():

user\_input = request.form["user\_input"]

bot\_response = generate\_response(user\_input)

return {"response": bot\_response}

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

* **Importaciones:** Importa Flask y la función generate\_response desde chatbot\_logic.py.
* **Inicialización de Flask:** Crea una instancia de la aplicación Flask.
* **Carga de la clave de API:** Carga la clave de API de Google Cloud AI Studio.
* **Ruta /:** Renderiza la plantilla index.html y pasa las variables PAGE\_TITLE e INPUT\_PLACEHOLDER.
* **Ruta /get\_response:** Maneja las solicitudes POST del formulario en index.html, obtiene la respuesta del chatbot y la devuelve como JSON.
* **Bloque if \_\_name\_\_ == "\_\_main\_\_"::** Inicia la aplicación Flask.

#### 3. templates/index.html

Este archivo contiene el código HTML para la interfaz de usuario del chatbot.

<!DOCTYPE html>

<html>

<head>

<title>{{ page\_title }}</title>

<style>

body { font-family: sans-serif; }

#chat-container { width: 500px; margin: 20px auto; border: 1px solid #ccc; padding: 10px; }

#chat-log { height: 300px; overflow-y: scroll; padding: 10px; background-color: #f9f9f9; }

.user-message { text-align: right; margin-bottom: 5px; color: blue; }

.bot-message { text-align: left; margin-bottom: 5px; color: green; }

#user-input-container { display: flex; margin-top: 10px; }

#user-input { flex-grow: 1; padding: 8px; }

#send-button { padding: 8px 15px; }

#logo {

display: block;

margin-left: auto;

margin-right: auto;

width: 100px; /\* Ajusta el ancho según sea necesario \*/

margin-bottom: 10px;

}

#footer {

text-align: center;

margin-top: 20px;

font-size: 0.8em;

color: #888;

}

</style>

</head>

<body>

<div id="chat-container">

<img id="logo" src="your\_logo.png" alt="Logo">

<h2>{{ page\_title }}</h2>

<div id="chat-log"></div>

<div id="user-input-container">

<input type="text" id="user-input" placeholder="{{ input\_placeholder }}">

<button id="send-button">Send</button>

</div>

</div>

<div id="footer">

Done by: Edwin Guaman

</div>

<script>

const chatLog = document.getElementById('chat-log');

const userInput = document.getElementById('user-input');

const sendButton = document.getElementById('send-button');

sendButton.addEventListener('click', sendMessage);

userInput.addEventListener('keypress', function(event) {

if (event.key === 'Enter') {

sendMessage();

}

});

function sendMessage() {

const message = userInput.value.trim();

if (message) {

appendMessage('user-message', 'You: ' + message);

userInput.value = '';

fetch('/get\_response', {

method: 'POST',

headers: {

'Content-Type': 'application/x-www-form-urlencoded',

},

body: 'user\_input=' + encodeURIComponent(message),

})

.then(response => response.json())

.then(data => {

appendMessage('bot-message', 'Car Seller: ' + data.response);

});

}

}

function appendMessage(className, text) {

const messageDiv = document.createElement('div');

messageDiv.classList.add(className);

messageDiv.textContent = text;

chatLog.appendChild(messageDiv);

chatLog.scrollTop = chatLog.scrollHeight; // Desplázate hasta abajo

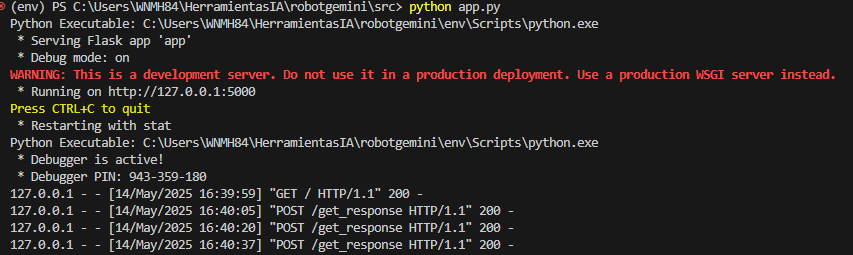
}

</script>

</body>

</html>

Este es el comando:



Este es el rol configurado: Tu eres un project manager y solo respondes preguntas relacionadas con lo que haces, tus respuestas son cortas y graciosas.

Esta es una imagen de la respuesta del chatbot:



**Beneficios de Cambiar el Rol en el Código**

La capacidad de cambiar el rol del chatbot mediante la modificación de la variable BOT\_ROLE ofrece varios beneficios:

* **Versatilidad:** El mismo código se puede utilizar para diferentes aplicaciones de chatbot (por ejemplo, un vendedor de coches, un asistente de reservaciones de hotel, un agente de soporte técnico) simplemente cambiando el valor de la variable.
* **Personalización:** El comportamiento y las respuestas del chatbot se pueden adaptar fácilmente al contexto específico de la aplicación.
* **Mantenibilidad:** El código se vuelve más modular y fácil de mantener, ya que la lógica específica del rol se encapsula en la variable BOT\_ROLE.
* **Reutilización:** El código se puede reutilizar en diferentes proyectos con diferentes roles de chatbot, lo que ahorra tiempo de desarrollo.

**Conclusiones**

Este documento describió el desarrollo de un chatbot utilizando el modelo Gemini y Flask. El chatbot es versátil y personalizable, gracias a la capacidad de cambiar su rol mediante una variable de configuración. El uso de un archivo .env mejora la seguridad al proteger la clave de API. Este proyecto sirve como base para construir chatbots más complejos con diversas aplicaciones y funcionalidades.

Referencias

Aquí tienes una bibliografía en español, utilizando el estilo MLA, para las fuentes utilizadas en este documento:

 Flask. *Flask*. Recuperado de <https://[flask.palletsprojects.com/en/3.0.x/](https://flask.palletsprojects.com/en/3.0.x/)>

 Google Cloud. *Google Generative AI*. Recuperado de <https://[cloud.google.com/vertex-ai/docs/generative-ai/learn/overview](https://cloud.google.com/vertex-ai/docs/generative-ai/learn/overview)>

 Python Software Foundation. *python-dotenv*. Recuperado de <https://pypi.org/project/python-dotenv/>

 Ronacher, A. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.

 Grus, J. (2019). *Data Science from Scratch: First Principles with Python*. O'Reilly Media