Consultorio Médico Virtual con IA

Juan David Cotacio Sánchez / Valery Dayana Triana Garcia / Juliana Carolina Avila Sánchez

I. Introducción

Este proyecto se centra en el diseño y desarrollo de un prototipo de clínica médica virtual que permita interactuar en tiempo real con los médicos a través de streaming de vídeo, con el apoyo de la inteligencia artificial. La clínica utilizará tecnologías como Ollama, FastAPI, HTML, CSS y JavaScript para ofrecer una experiencia de usuario óptima. El frontend se diseñará con una interfaz fácil de usar y con capacidad de respuesta mediante HTML, CSS y JavaScript, mientras que el backend gestionará la lógica del servidor y la funcionalidad de transmisión de vídeo mediante FastAPI. La integración de la IA se logra mediante la conexión a modelos avanzados (como Llama, GPT, PHI u otros) utilizando Ollama, lo que permite a la IA realizar consultas médicas. Las principales tareas incluyen el diseño y la codificación de la interfaz, la integración del streaming de vídeo en tiempo real y la implementación de la IA para responder a consultas médicas.

A. Como empezar el prototipo

Para crear nuestro prototipo, empezamos a diseñar nuestro backend con el lenguaje Python, usando diferentes librerías y Framework, después iniciamos la implementación de FastApi y nuestro modelo IA de ollama, para nuestro modelo Ollama escogimos un modelo que ya se encuentra entrenado para responder preguntas de tipo médica ”medllama2” entonces realizamos la implementación de ese modelo en nuestro código a través de la API, creamos nuestro enpoint consultar para poder realizar nuestra lógica y para más adelante lograr la conexión con nuestro fronted y servidor.

```python
from fastapi import FastAPI, Request
from fastapi.responses import JSONResponse
from fastapi.middleware.cors import CORSMiddleware
import aiohttp
import json

app = FastAPI()
```

Figure 1. Implementación de FastAPI

B. Como se Implemento Ollama y FastApi

Importación e inicialización: se importan las bibliotecas necesarias, incluidas FastAPI, transformers,aiohttp y json, estas dos ultimas bibliotecas se utilizan para lo siguiente:

eaiohttp: Utilizado para realizar solicitudes HTTP de manera asíncrona, lo que permite manejar múltiples solicitudes de manera eficiente sin bloquear la ejecución del programa.

json: Utilizado para trabajar con datos en formato JSON, facilitando la conversión entre objetos de Python y cadenas JSON, así como la lectura y escritura de archivos JSON.

Configuración de CORS: CORS está configurado para permitir peticiones de cualquier origen.

```python
OLLAMA_API_URL = "http://localhost:11434/api/generate"
MODEL_NAME = "medllama2"

@app.post("/consultar")
async def consultar(request: Request):
    data = await request.json()
    user_input = data.get("message")

    payload = {
        "model": MODEL_NAME,
        "prompt": user_input
    }

    async with aiohttp.ClientSession() as session:
        async with session.post(OLLAMA_API_URL, json=payload) as response:
            response_text = await response.text()
            response_lines = response_text.strip().split("\n")
            response_data = [json.loads(line) for line in response_lines]

    return JSONResponse(content=response_data)
```

Figure 2. Implemtacion del EndPoint y del Modelo IA ollama

C. Implementación Java Script

Este código implemntamos JavaScript que permite acceder a la cámara en tiempo real y enviar mensajes a un servidor cuando se hace clic en un botón. La primera parte del código se encarga de obtener acceso a la cámara del dispositivo y mostrar el video en un elemento HTML ¡video¿. Utiliza la API navigator.mediaDevices.getUserMedia para solicitar acceso a la cámara y, si se concede, asigna el flujo de video (stream) al atributo srcObject del elemento ¡video¿. En caso de error, se muestra un mensaje en la consola.

La segunda parte del código agrega un evento al botón con la clase .send-button. Cuando se hace clic, el código captura el valor del campo de entrada (user-input) y verifica que no esté vacío. Si hay un mensaje, se envía una solicitud POST al servidor en la URL ”http://localhost:3001/consultar”, incluyendo el mensaje en el cuerpo de la solicitud como un objeto JSON. Si la respuesta del servidor es exitosa, se procesa el JSON de respuesta y se muestra en un contenedor HTML (response-container). En caso de error durante el envío de la solicitud o en la respuesta del servidor, se muestra un mensaje de error en la consola.

D. Pruebas del Prototipo

Al probar un prototipo desarrollado con FastAPI, JavaScript, Python, CSS, HTML y Ollama, es importante comprobar características clave como la funcionalidad del modelo de ollama,

```
document.querySelector(".send-button").addEventListener("click", async () => {
    const userInput = document.getElementById("user-input").value;

    // Verifica que hay un mensaje para enviar
    if (!userInput) {
        console.log("El campo de mensaje está vacío.");
        return;
    }

    try {
        const response = await fetch("http://localhost:3001/consultar", {
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify({ message: userInput })
        });

        // Verifica si la respuesta no es OK
        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }

        // Intenta convertir la respuesta a JSON
        const data = await response.json();
        console.log("Respuesta del servidor:", data);

        // Manejar la respuesta y mostrarla en el frontend
        const responseContainer = document.getElementById("response-container");
        responseContainer.innerHTML = ""; // Limpiar el contenedor antes de agregar nuevas respuestas

        data.forEach(item => {
            const responseParagraph = document.createElement("p");
            responseParagraph.innerText = item.response || item.error || "No se recibió una respuesta válida.";
            responseContainer.appendChild(responseParagraph);
        });

    } catch (error) {
        console.error("Error al enviar el mensaje:", error);
    }
});
```

Figure 3. Conexion endpoint consultar con JS

```
const video = document.getElementById('video');

if (navigator.mediaDevices.getUserMedia) {
    navigator.mediaDevices.getUserMedia({ video: true })
    .then(function (stream) {
        video.srcObject = stream;
    })
    .catch(function (error) {
        console.log("Algo salió mal al intentar acceder a la cámara: " + error);
    });
}
```

Figure 4. Implementación del Straming con JS

la respuesta al prompt del usuario y del Streaming. También deben evaluarse aspectos de rendimiento como la estabilidad del servidor y el tiempo de respuesta bajo carga. Las pruebas de usabilidad son cruciales para garantizar una interfaz de usuario intuitiva y accesible en distintos dispositivos. Mediante pruebas exhaustivas en estos ámbitos, se puede detectar y resolver cualquier problema para garantizar que el prototipo funciona de forma eficaz y satisfactoria para los usuarios.

Respuesta a un prompt realizado sobre el dolor muscular: ”The best way to relieve pain from a muscle strain is through rest, ice application and over-the-counter painkillers like ibuprofen or acetaminophen. If the discomfort persists, consult with your doctor for further evaluation and treatment plan.”

E. Vista General del Prototipo

Al centrarnos en una interfaz de usuario intuitiva, los prototipos son accesibles a una amplia gama de usuarios y pueden interactuar cómodamente con ellos usuarios de todos los niveles de destreza. Esta facilidad de uso no sólo aumenta la versatilidad de aplicación en distintos ámbitos, sino que también promueve una experiencia de usuario positiva que facilita la adopción y la utilidad en diversos contextos. Además, el proceso de desarrollo del proyecto no sólo permitió crear aplicaciones funcionales, sino que también brindó la oportunidad de aprender y aplicar nuevos conocimientos, reforzando así sus competencias y preparándoles para futuros retos y oportunidades en el ámbito técnico.

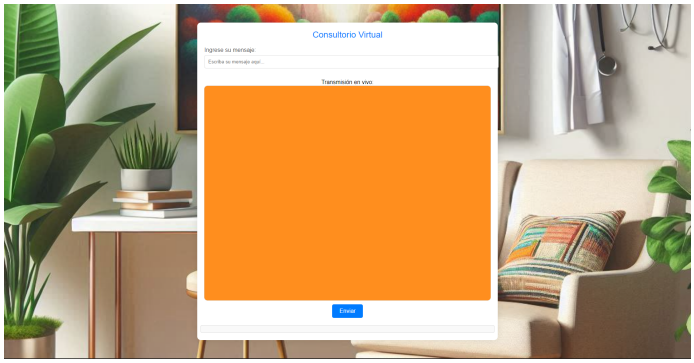

Figure 5. Vista Final Del Prototipo

F. Dificultades

Durante el desarrollo de este proyecto, se encontraron algunas dificultades a la hora de realizar la implementación del streaming con JavaScript fue tema complejo para realizar la conexión de la cámara con el HTML para poder darle ese toque de un consultorio virtual además otro de nuestros retos fue la implementación de nuestro modelo Ollama, pues en nuestra primera pruebas no se mostraba el prompt que mandaba el usuario, entonces su idea principal no estaba funcionado, pero después de solucionar este defecto se logró la implementación.