Katherine Michelle Ortiz Burgos
Código: 506221041
Juan Esteban Acosta Rodriguez
Código: 506221068
Profesor: Oscar Alexander Mendez Aguirre
Fundación Universitaria Konrad Lorenz

Prototipo Consultorio Médico Virtual Con IA

I. Resumen

Este prototipo tiene como objetivo desarrollar un prototipo funcional de un consultorio médico virtual que permita la interacción en tiempo real mediante video streaming con un médico impulsado por inteligencia artificial. Utilizando tecnologías como Ollama, FastAPI, HTML, CSS y JavaScript, se creará una interfaz de usuario amigable y responsiva, gestionada por FastAPI para la lógica del servidor y video streaming, y con una IA conectada a través de Ollama para responder consultas médicas. El prototipo incluirá el código fuente completo en GitLab y un reporte detallado, abarcando desde el diseño de la interfaz hasta la integración de IA, asegurando una experiencia innovadora y accesible en telemedicina.

II. Introducción

El prototipo del consultorio médico virtual que hemos desarrollado integra de manera eficiente el frontend y el backend para ofrecer una experiencia completa a través de una interfaz web intuitiva. Esta integración es crucial para proporcionar una aplicación funcional en el ámbito de la telemedicina.

El frontend de la aplicación se encarga de la parte visible para el usuario en su navegador web. Esto incluye una interfaz gráfica donde los usuarios pueden iniciar sesiones de consulta médica virtual mediante video streaming y realizar preguntas médicas en tiempo real. Además, ofrece controles interactivos, como botones para iniciar la consulta y formulario para ingresar preguntas médicas.

Por otro lado, el backend de la aplicación maneja la lógica detrás de la funcionalidad de video streaming y la interacción con la inteligencia artificial. Utiliza FastAPI para gestionar las sesiones de video en tiempo real y conecta con modelos avanzados de IA a través de Ollama para proporcionar respuestas precisas a las consultas médicas de los usuarios. El backend también se encarga de procesar los datos recibidos y enviar los resultados al frontend para su visualización, asegurando una experiencia de usuario fluida.

Esta colaboración entre el frontend y el backend permite que la aplicación funcione de manera fluida y eficiente. Mientras que el frontend proporciona una interfaz amigable para que los usuarios interactúen con el sistema, el backend realiza el procesamiento intensivo de datos y el análisis de sentimientos, garantizando una experiencia de usuario óptima y resultados precisos.

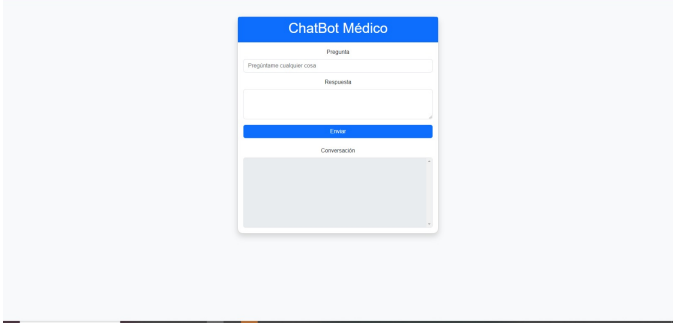

Figura 1. Chatbot.

III. Diseño y desarrollo de la API backend

En el desarrollo del prototipo del consultorio médico virtual, la API backend desempeña un papel crucial al gestionar la lógica del servidor, la funcionalidad de video streaming y la integración con la inteligencia artificial para responder a las consultas médicas. La API backend está construida utilizando FastAPI, un framework moderno y de alto rendimiento que facilita la creación de APIs robustas y eficientes. Esta API no solo maneja las solicitudes de video streaming en tiempo real, sino que también procesa las interacciones entre el usuario y el modelo de IA.

Estructura main.py

1. Importación de módulos: La aplicación FastAPI se inicializa mediante app = FastAPI(), creando una instancia que gestionará las rutas y las solicitudes HTTP. Simultáneamente, templates = Jinja2Templates(directory=”templates”) configura el directorio de plantillas HTML, asegurando que los archivos HTML necesarios para el frontend sean accesibles y puedan ser servidos correctamente por el servidor.
2. Inicialización de la Aplicación: La aplicación FastAPI se inicializa mediante app = FastAPI(), creando una instancia que gestionará las rutas y las solicitudes HTTP. Simultáneamente, templates = Jinja2Templates(directory=”templates”) configura el directorio de plantillas HTML, asegurando que los archivos HTML necesarios para el frontend sean accesibles y puedan ser servidos correctamente por el servidor.
3. Ruta /video_feed: La ruta /video_feed se define con el decorador `@app.get("/video_feed")`,

especificando el punto final para el video streaming. Dentro de esta ruta, `cap = cv2.VideoCapture(0)` inicia la captura de video desde la cámara web. La función generadora `generate_frames()` captura continuamente frames de video, los codifica en formato JPEG y los transmite como un flujo continuo. El bucle `while True` mantiene la captura de frames, y cada frame se lee, codifica y transmite hasta que no se puede leer más. Finalmente, `return StreamingResponse(generate_frames(), media_type="multipart/x-mixed-replace; boundary=frame")` devuelve este flujo continuo de frames como una respuesta de streaming, permitiendo la transmisión en tiempo real al cliente.

4. Ruta / :La ruta raíz / se define con @app.get("/"), sirviendo la página HTML principal de la aplicación. Al recibir una solicitud en esta ruta, return templates.TemplateResponse("index.html", "request": request) renderiza y devuelve la plantilla index.html al cliente. Esta configuración permite que el frontend se integre sin problemas con el backend, presentando al usuario la interfaz web necesaria para interactuar con el sistema de video streaming y el chatbot médico.

Estructura index.js

1. Inicialización del Documento: Inicialización del Documento: El archivo `index.js` comienza su ejecución asegurando que el código JavaScript se ejecute en el momento adecuado, una vez que el documento HTML esté completamente cargado. Esto se logra mediante `$(document).ready(function() { ... });`, lo que garantiza una interacción sin problemas con el usuario.
2. Variables y Almacenamiento: En el frontend, se inicializa un arreglo llamado conversacion para almacenar el historial de interacciones entre el usuario y el chatbot. Esta estructura de datos juega un papel crucial en el seguimiento y la visualización de la conversación en la interfaz de usuario.
3. Manejo del Botón de Envío: El código establece un evento de clic para el botón con id buscar, encargado de manejar el envío de preguntas por parte del usuario. Se captura la pregunta ingresada por el usuario, se limpia y se verifica si está vacía. En caso afirmativo, se muestra una alerta. Luego, la pregunta se agrega al historial de conversación.
4. Configuración de la Solicitud AJAX:La configuración de la solicitud AJAX es esencial para la comunicación entre el frontend y el backend. Esta configuración permite enviar la pregunta al servidor y recibir la respuesta del chatbot médico en tiempo real. Se construye y envía la solicitud al endpoint correspondiente en el backend, mientras se actualiza dinámicamente la interfaz de usuario con la respuesta recibida.
5. Manejo del Éxito de la Solicitud: Cuando la solicitud al backend tiene éxito, index.js actualiza el historial de conversación con la respuesta del chatbot y refleja estos cambios en la interfaz de usuario. Además, limpia los campos de entrada para preparar la interfaz para la próxima interacción del usuario.
6. Manejo de Errores: En caso de que ocurra un error durante la comunicación con el servidor, index.js muestra una alerta al usuario para informar sobre el problema. Esto asegura que cualquier inconveniente se maneje de manera transparente y que el usuario esté al tanto de posibles errores.

IV. Creación de la interfaz gráfica y conexión con el backend

El frontend del proyecto del consultorio médico virtual está diseñado para ofrecer una interfaz de usuario intuitiva y responsiva, permitiendo la interacción fluida con un chatbot médico impulsado por inteligencia artificial. Este componente de la aplicación está construido utilizando HTML, CSS y JavaScript, junto con la biblioteca Bootstrap para asegurar un diseño moderno y atractivo. La página principal presenta un campo de entrada para que los usuarios ingresen sus preguntas médicas, un área de texto para mostrar las respuestas del chatbot, y un botón para enviar las consultas. Además, un contenedor dedicado muestra el historial de conversación, facilitando el seguimiento de las interacciones anteriores. La combinación de estos elementos asegura que los usuarios puedan realizar consultas médicas de manera eficiente y ver las respuestas en tiempo real, proporcionando una experiencia de usuario completa y satisfactoria.

Estructura index.html

El archivo `index.html` constituye la estructura básica y la interfaz de usuario de la aplicación. Utilizando HTML5 y Bootstrap, proporciona un diseño limpio y responsivo. El archivo incluye elementos como un campo de entrada de texto para preguntas, un área de texto para mostrar respuestas y un botón de envío. Además, contiene un contenedor para mostrar el historial de la conversación. Los scripts necesarios (como `index.js`) y hojas de estilo (`index.css`) también se enlazan aquí, asegurando que la funcionalidad y el estilo se apliquen correctamente. Esto proporciona al usuario una interfaz intuitiva y fácil de usar para interactuar con el chatbot médico.

Estilos CSS

El archivo `index.css` aplica estilos personalizados a la interfaz de usuario. Define el aspecto visual, como los colores de fondo, las fuentes y el diseño de los elementos de la página, garantizando una experiencia de usuario atractiva y coherente. Se establece un fondo claro, se define la fuente para el texto y se ajustan los estilos para los contenedores y tarjetas, proporcionando un diseño limpio y profesional.

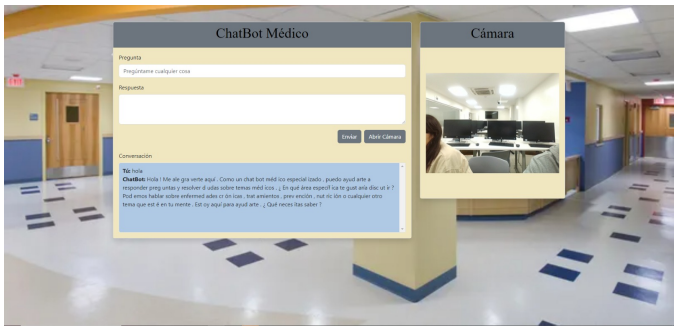

Figura 3. Prototipo Chatbot Médico.

V. Conclusiones

1. La configuración detallada de rutas en la aplicación permite una integración fluida entre el frontend y el backend, asegurando que los usuarios puedan interactuar de manera efectiva con el sistema de video streaming y el chatbot médico.
2. Gracias a la combinación de HTML5, Bootstrap, y CSS personalizado, la aplicación ofrece una interfaz de usuario que es tanto intuitiva como visualmente atractiva. Los elementos de diseño responsivo y los estilos coherentes facilitan la navegación y mejoran la experiencia general del usuario.
3. La inicialización del documento y la gestión de eventos mediante JavaScript aseguran que los componentes interactivos de la aplicación se carguen y ejecuten de manera eficiente. Esto no solo mejora la capacidad de respuesta de la aplicación, sino que también garantiza que las funciones críticas, como la captura de video y la interacción con el chatbot, se ejecuten sin problemas.

VI. Referencias

1. Google Colab. (s/f). Google.com. Recuperado el 26 de mayo de 2024, de https://colab.research.google.com/drive/1MIoprqtyD8XjFyaa_rjHZZ8xW6RRN4yp?hl=es