



TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA

Chatbot Conversacional

**Desarrollo de un Chatbot para servicios sociales dirigido a personas en
situación de dependencia**

Autor

Claudia Salado Méndez

Directores

Francisco Manuel García Moreno
maría Visitación Hurtado Torres



Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación

—

Granada, julio de 2023



ugr

Universidad
de Granada

Chatbot Conversacional

Desarrollo de un Chatbot para servicios sociales dirigido a personas en situación de dependencia.

Autor

Claudia Salado Méndez

Directores

Francisco Manuel García Moreno

María Visitación Hurtado Torres

Chatbot Conversacional: Desarrollo de un Chatbot para servicios sociales dirigido a personas en situación de dependencia

Claudia Salado Méndez

Palabras clave: Chatbot, Python, OpenAI, Personas Dependientes, Ayuda, API, Servicios Sociales, Conversación.

Resumen

El Chatbot desarrollado es una aplicación web basada en inteligencia artificial que brinda asistencia médica virtual a los usuarios. El objetivo principal del Chatbot es proporcionar información médica confiable y responder consultas relacionadas con la salud de los usuarios.

El Chatbot utiliza un modelo de lenguaje basado en la arquitectura GPT-3.5 de OpenAI, que ha sido previamente entrenado en una amplia variedad de datos relacionados con la medicina y la salud. El modelo se ha adaptado y personalizado para comprender y responder de manera precisa a las preguntas y consultas médicas de los usuarios.

El Chatbot cuenta con varias características y funcionalidades clave. Los usuarios pueden iniciar sesión en la aplicación utilizando sus credenciales y acceder a su historial médico personalizado. Pueden realizar consultas sobre síntomas, enfermedades, medicamentos, tratamientos, exámenes médicos y otras áreas relacionadas con la salud.

Además de proporcionar respuestas precisas y relevantes, el Chatbot también ofrece consejos de estilo de vida saludable así como recomendaciones de dieta, ejercicio y manejo del estrés. Se ha integrado con una base de datos SQLite para almacenar la información de los usuarios, incluyendo sus datos personales, historiales médicos, citas y recordatorios.

En resumen, el Chatbot desarrollado en este TFG ofrece una solución inteligente y accesible para brindar asistencia virtual a los usuarios. Con su capacidad para comprender y responder consultas médicas, proporcionar consejos de salud y gestionar información personalizada, el Chatbot busca mejorar la experiencia del usuario y promover un enfoque proactivo hacia el cuidado de la salud.

Conversational Chatbot: Development of a chatbot for social services aimed at people in situations of dependency

Claudia Salado Méndez

Keywords: Chatbot, Python, OpenAI, Dependent Persons, Help, API, Social Services, Conversation.

Abstract

The developed Chatbot is a web application based on artificial intelligence that provides virtual medical assistance to users. With the advancement of technology and the increasing demand for accessible healthcare solutions, the Chatbot aims to bridge the gap between users and reliable medical information.

At the core of the Chatbot is a powerful language model based on OpenAI's GPT-3.5 architecture. This model has been fine-tuned and trained on a vast corpus of medical and healthcare data to ensure accurate understanding and effective responses to users' medical inquiries. By leveraging the capabilities of natural language processing and machine learning, the Chatbot is capable of simulating human-like conversations, making it a valuable tool for users seeking medical advice.

One of the primary objectives of the Chatbot is to provide reliable and up-to-date medical information. Users can log into the web application using their credentials and access a personalized dashboard where they can input their medical history, including pre-existing conditions, medications, and allergies. This information allows the Chatbot to tailor its responses and recommendations to each user's unique profile, ensuring personalized and relevant assistance.

The Chatbot is equipped with a comprehensive knowledge base that covers a wide range of medical topics. Users can ask questions about symptoms, diseases, treatments, medications, and medical procedures. The Chatbot utilizes its extensive knowledge and natural language understanding capabilities to provide accurate answers and guidance. It can help users understand their symptoms, provide information about potential causes, and suggest appropriate next steps, such as seeking medical attention or managing symptoms at home.

In addition to providing medical information, the Chatbot also promotes healthy lifestyle choices. It offers advice on nutrition, exercise, stress management, and other aspects of maintaining a healthy lifestyle. By incorporating evidence-based recommendations and guidelines, the Chatbot aims to empower users to make informed decisions about their health and well-being.

To ensure a seamless user experience, the Chatbot has been integrated with a SQLite database. This database securely stores user information, including personal data, medical records, appointments, and reminders. Users can access their medical history and track their progress over time. The Chatbot also supports appointment scheduling and sends reminders to users to help them manage their healthcare routine effectively.

In conclusion, the Chatbot developed in this TFG serves as a valuable virtual medical assistant, providing users with reliable medical information, personalized advice, and assistance in managing their health. By leveraging artificial intelligence and natural language processing, the Chatbot aims to improve accessibility to healthcare resources and empower users to make informed decisions about their well-being. Through its user-friendly interface, extensive medical knowledge base, and personalized features, the Chatbot seeks to enhance the overall healthcare experience for users and contribute to the advancement of virtual medical assistance.

Yo, **Claudia Salado Méndez**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77143396Q, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

A handwritten signature in blue ink, appearing to read 'Claudia', with a stylized flourish at the end.

Fdo: Claudia Salado Méndez

Granada a 13 de julio de 2023

D. **Francisco Manuel García Moreno**, Profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

D.^a **María Visitación Hurtado Torres**, Profesora del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado **Chatbot Conversacional: Desarrollo de un chatbot para servicios sociales dirigido a personas en situación de dependencia**, ha sido realizado bajo su supervisión por **Claudia Salado Méndez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 13 de julio de 2023.

Los directores:

Francisco Manuel García Moreno

María Visitación Hurtado Torres

Agradecimientos

¿Quién me iba a decir a mi hace un par de años, cuando estaba en primero de carrera, que llegaría este momento en el que finalizo mi grado tras la presentación de mi TFG? Era algo que ninguno de nosotros nos creemos al principio y mucho menos después de estos duros años que nos ha traído la pandemia en la que hemos tenido que actualizarnos-y continuar con nuestra carrera de otra forma.

En primer lugar, quiero agradecer a muchos de los profesores que he tenido a lo largo de estos años por sus enseñanzas no solo a nivel académico sino en muchos otros aspectos. Me han transmitido gran motivación para continuar y seguir aprendiendo. Algunos han conseguido literalmente que me gustara la informática, algo que muchas veces se nos olvida en los malos momentos de la carrera. Gracias por vuestro apoyo.

Quiero hacer un agradecimiento especial a la profesora Nuria Rico. Ella me dio maravillosamente la asignatura de estadística en segundo curso pero yo ya la conocía anteriormente porque, estando en educación secundaria en el colegio, participé en el campus para chicas ingenieras del que ella es responsable. De no ser por esas semanas tocando por primera vez la programación y los arduinos probablemente no estaría aquí. Gracias Nuria.

También quiero agradecer a mis padres por darme la oportunidad de estudiar desde pequeña y de animarme a estudiar una ingeniería. Ellos han estado en los buenos y en los malos momentos porque para mí estos años han sido como una montaña rusa, con situaciones de euforia total pero también con momentos oscuros donde uno se cuestiona si ha elegido la carrera correcta. Gracias por aguantar mis quejas y mi “atranque” con alguna que otra asignatura (todos en casa conocen la famosa asignatura de Lógica) que me hizo llegar incluso a la tercera matrícula y a la matriculación tardía de este TFG.

Continuado con la familia, soy la mayor de cuatro hermanos. Quiero agradecer a mis tres hermanos, Alejandra (de once años), David (de ocho años) y Jairo (de un año), porque, aunque he acabado harta de ellos muchas veces por gritar, cantar, y jugar a los juegos más escandalosos posibles mientras estoy estudiando, haciendo prácticas o en clases virtuales en su época, con sus tonterías y sus risas me han alegrado los días y animado en los momentos que ya no podía seguir estudiando más o que nada me funcionaba. Todavía son muy pequeños pero me haría mucha ilusión que estudiaran lo mismo que yo aunque imagino que ellos seguirán su propio camino y que también será estupendo y yo estaré orgullosa.

Por último, quiero dar las gracias a todas las personas maravillosas que he conocido durante estos años en la carrera, porque han sido unos fantásticos años de mi vida junto a ellos y de los que he aprendido mucho. También a mis amigos de siempre por acompañarme hasta hoy y ayudarme a despejarme del tema académico los fines de semana. Quiero agradecer en especial a mi amigo Joselu que me haya soportado tanto tiempo, porque seguramente lo he agotado llamándole o pidiendo ayuda cada vez que me agobiaba porque algo no me funcionaba o no me enteraba. Gracias por ser paciente y ayudarme en los problemas (académicos y personales).



TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA

Chatbot Conversacional

**Desarrollo de un Chatbot para servicios sociales dirigido a personas en
situación de dependencia**

Autor

Claudia Salado Méndez

Directores

Francisco Manuel García Moreno
maría Visitación Hurtado Torres



Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación

—

Granada, julio de 2023



ugr

Universidad
de Granada

Chatbot Conversacional

Desarrollo de un Chatbot para servicios sociales dirigido a personas en situación de dependencia.

Autor

Claudia Salado Méndez

Directores

Francisco Manuel García Moreno

María Visitación Hurtado Torres

Chatbot Conversacional: Desarrollo de un Chatbot para servicios sociales dirigido a personas en situación de dependencia

Claudia Salado Méndez

Palabras clave: Chatbot, Python, OpenAI, Personas Dependientes, Ayuda, API, Servicios Sociales, Conversación.

Resumen

El Chatbot desarrollado es una aplicación web basada en inteligencia artificial que brinda asistencia médica virtual a los usuarios. El objetivo principal del Chatbot es proporcionar información médica confiable y responder consultas relacionadas con la salud de los usuarios.

El Chatbot utiliza un modelo de lenguaje basado en la arquitectura GPT-3.5 de OpenAI, que ha sido previamente entrenado en una amplia variedad de datos relacionados con la medicina y la salud. El modelo se ha adaptado y personalizado para comprender y responder de manera precisa a las preguntas y consultas médicas de los usuarios.

El Chatbot cuenta con varias características y funcionalidades clave. Los usuarios pueden iniciar sesión en la aplicación utilizando sus credenciales y acceder a su historial médico personalizado. Pueden realizar consultas sobre síntomas, enfermedades, medicamentos, tratamientos, exámenes médicos y otras áreas relacionadas con la salud.

Además de proporcionar respuestas precisas y relevantes, el Chatbot también ofrece consejos de estilo de vida saludable así como recomendaciones de dieta, ejercicio y manejo del estrés. Se ha integrado con una base de datos SQLite para almacenar la información de los usuarios, incluyendo sus datos personales, historiales médicos, citas y recordatorios.

En resumen, el Chatbot desarrollado en este TFG ofrece una solución inteligente y accesible para brindar asistencia virtual a los usuarios. Con su capacidad para comprender y responder consultas médicas, proporcionar consejos de salud y gestionar información personalizada, el Chatbot busca mejorar la experiencia del usuario y promover un enfoque proactivo hacia el cuidado de la salud.

Conversational Chatbot: Development of a chatbot for social services aimed at people in situations of dependency

Claudia Salado Méndez

Keywords: Chatbot, Python, OpenAI, Dependent Persons, Help, API, Social Services, Conversation.

Abstract

The developed Chatbot is a web application based on artificial intelligence that provides virtual medical assistance to users. The main objective of the Chatbot is to provide reliable medical information and answer questions related to the health of users.

The Chatbot uses a language model based on OpenAI's GPT-3.5 architecture, which has been pre-modified on a wide variety of medical and healthcare related data. The model has been tailored and customized to accurately understand and respond to users' medical questions and inquiries.

The Chatbot has several key features and functionalities. Users can log into the app using their credentials and access their personalized medical history. You can ask questions about symptoms, diseases, medications, treatments, medical tests and other areas related to health.

In addition to providing accurate and relevant answers, the Chatbot also offers healthy lifestyle advice, recommendations such as diet, exercise, and stress management. It has been integrated with a SQLite database to store user information, including personal data, medical records, appointments, and reminders.

In summary, the Chatbot developed in this TFG offers an intelligent and accessible solution to provide virtual medical assistance to users. With its ability to understand and answer medical queries, provide health advice, and manage personalized information, the Chatbot seeks to enhance the user experience and promote a proactive approach to healthcare.

Yo, **Claudia Salado Méndez**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77143396Q, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Claudia Salado Méndez

Granada a 10 de julio de 2023

D. **Francisco Manuel García Moreno**, Profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

D.^a **María Visitación Hurtado Torres**, Profesora del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado **Chatbot Conversacional: Desarrollo de un chatbot para servicios sociales dirigido a personas en situación de dependencia**, ha sido realizado bajo su supervisión por **Claudia Salado Méndez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 10 de julio de 2023.

Los directores:

Francisco Manuel García Moreno

María Visitación Hurtado Torres

Agradecimientos

¿Quién me iba a decir a mi hace un par de años, cuando estaba en primero de carrera, que llegaría este momento en el que finalizo mi grado tras la presentación de mi TFG? Era algo que ninguno de nosotros nos creemos al principio y mucho menos después de estos duros años que nos ha traído la pandemia en la que hemos tenido que actualizarnos-y continuar con nuestra carrera de otra forma.

En primer lugar, quiero agradecer a muchos de los profesores que he tenido a lo largo de estos años por sus enseñanzas no solo a nivel académico sino en muchos otros aspectos. Me han transmitido gran motivación para continuar y seguir aprendiendo. Algunos han conseguido literalmente que me gustara la informática, algo que muchas veces se nos olvida en los malos momentos de la carrera. Gracias por vuestro apoyo.

Quiero hacer un agradecimiento especial a la profesora Nuria Rico. Ella me dio la maravillosamente la asignatura de estadística en segundo curso pero yo ya la conocía anteriormente porque, estando en educación secundaria en el colegio, participé en el campus para chicas ingenieras del que ella es responsable. De no ser por esas semanas tocando por primera vez la programación y los arduinos probablemente no estaría aquí. Gracias Nuria.

También quiero agradecer a mis padres por darme la oportunidad de estudiar desde pequeña y de animarme a estudiar una ingeniería. Ellos han estado en los buenos y en los malos momentos porque para mí estos años han sido como una montaña rusa, con situaciones de euforia total pero también con momentos oscuros donde uno se cuestiona si ha elegido la carrera correcta. Gracias por aguantar mis quejas y mi “atranque” con alguna que otra asignatura (todos en casa conocen la famosa asignatura de Lógica) que me hizo llegar incluso a la tercera matrícula y a la matriculación tardía de este TFG.

Continuado con la familia, soy la mayor de cuatro hermanos. Quiero agradecer a mis tres hermanos, Alejandra (de once años), David (de ocho años) y Jairo (de un año), porque, aunque he acabado harta de ellos muchas veces por gritar, cantar, y jugar a los juegos más escandalosos posibles mientras estoy estudiando, haciendo prácticas o en clases virtuales en su época, con sus tonterías y sus risas me han alegrado los días y animado en los momentos que ya no podía seguir estudiando más o que nada me funcionaba. Todavía son muy pequeños pero me haría mucha ilusión que estudiaran lo mismo que yo aunque imagino que ellos seguirán su propio camino y que también será estupendo y yo estaré orgullosa.

Por último, quiero dar las gracias a todas las personas maravillosas que he conocido durante estos años en la carrera, porque han sido unos fantásticos años de mi vida junto a ellos y de los que he aprendido mucho. También a mis amigos de siempre por acompañarme hasta hoy y ayudarme a despejarme del tema académico los fines de semana. Quiero agradecer en especial a mi amigo Joselu que me haya soportado tanto tiempo, porque seguramente lo he agotado llamándole o pidiendo ayuda cada vez que me agobiaba porque algo no me funcionaba o no me enteraba. Gracias por ser paciente y ayudarme en los problemas (académicos y personales).

ÍNDICE

1. INTRODUCCIÓN	3
1.1. Motivación	3
1.2. Objetivos principales del proyecto	4
2. REQUISITOS	6
2.1. Requisitos funcionales	6
2.2. Requisitos no funcionales	7
3. METODOLOGÍA ÁGIL	9
3.1. Metodologías de desarrollo más comunes	9
3.1.1. Metodología en Cascada (Waterfall)	9
3.1.2. Kanban	9
3.1.3. SCRUM	10
3.2. La mejor metodología para hacer un TFG en general	10
3.3. La metodología más eficaz para hacer un Chatbot	11
3.4. Elección final de la metodología a usar	11
4. HERRAMIENTAS PARA EL DESARROLLO DE CHATBOTS	13
4.1. Comparativa de las opciones	13
4.1.1. Dialogflow	13
4.1.2. Rasa	13
4.1.3. SCRUM	14
4.1.4. OpenChatKit	14
4.1.5. ChatterBot	15
4.2. Consideraciones para la elección de la herramienta	16
4.3. Elección final de la herramienta a usar	16
5. HISTORIAS DE USUARIO	18
6. ESTRUCTURA DE LOS SPRINTS	27
7. SPRINT 1: INSTALACIÓN DE LA HERRAMIENTA	29
7.1. OpenChatKit	29
7.2. Open Assistant	33
7.2.1. Características	34
7.2.2. Comparativa con otras IAs conversacionales	34
7.2.3. Instalación	34
7.2.4. Dificultades durante la instalación	36
7.2.5. Conclusión	41
8. SPRINT 2: ENTRENAMIENTO DE LAS RESPUESTAS	42
8.1. Introducción	42
8.2. Implementación	42

8.3. Conclusión	50
9. SPRINT 3: CREACIÓN DE LA BASE DE DATOS	51
9.1. Introducción	51
9.2. Implementación	51
9.3. Conclusión	55
10. SPRINT 4: CREACIÓN DE LAS FUNCIONALIDADES	56
10.1. Introducción	56
10.2. Implementación	56
10.3. Conclusión	60
11. SPRINT 5: ACCESIBILIDAD	62
11.1. Introducción	62
11.2. Implementación	62
11.3. Conclusión	66
12. INTERFAZ DE USUARIO	67
13. CONCLUSIÓN FINAL	78
14. VÍAS FUTURAS	80
15. BIBLIOGRAFÍA	82

1. INTRODUCCIÓN

La presente memoria describe el desarrollo e implementación de un Chatbot basado en inteligencia artificial para brindar asistencia médica virtual. Este Trabajo de Fin de Grado tiene como objetivo principal diseñar y construir un sistema interactivo capaz de responder preguntas y proporcionar información médica a los usuarios de manera precisa y confiable.

En el contexto actual, donde la tecnología y la digitalización están transformando rápidamente la industria de la salud, el uso de Chatbots se ha vuelto cada vez más relevante. Estos agentes virtuales pueden ayudar a reducir la carga de trabajo de los profesionales de la salud, brindar atención médica personalizada y estar disponibles las 24 horas del día.

En este TFG, se emplearon técnicas avanzadas de procesamiento del lenguaje natural y aprendizaje automático para desarrollar un Chatbot inteligente y eficiente. El sistema se diseñó para comprender y responder preguntas sobre temas médicos, proporcionando información relevante y confiable de acuerdo con los estándares médicos establecidos.

En resumen, este TFG representa un esfuerzo por aprovechar las tecnologías de inteligencia artificial y procesamiento del lenguaje natural para proporcionar asistencia médica virtual de calidad. El Chatbot desarrollado tiene el potencial de mejorar el acceso a la información médica, agilizar la comunicación entre pacientes y profesionales de la salud, y brindar una experiencia interactiva y personalizada en el ámbito de la salud.

1.1. Motivación

La motivación que me llevó a realizar este Trabajo de Fin de Grado surge de mi pasión por la tecnología y mi interés en ayudar y ser útil para los demás. Desde temprana edad, he sido testigo del impacto que la tecnología ha tenido en diversos sectores, y considero que la salud es un área donde la aplicación de la inteligencia artificial puede marcar la diferencia de manera significativa.

Mi motivación se fortaleció al observar los desafíos a los que se enfrenta el sistema de atención médica, como la falta de acceso a servicios médicos, largas listas de espera y la necesidad de una atención más personalizada. Reconocí en la inteligencia artificial y los Chatbots una oportunidad para abordar estas problemáticas y mejorar la calidad de vida de las personas.

La posibilidad de desarrollar un Chatbot capaz de brindar asistencia médica virtual, responder preguntas, proporcionar información y guiar a los usuarios en temas

de salud despertó mi entusiasmo. Creí que esta tecnología podría ser una solución innovadora para acercar la atención médica a aquellos que tienen dificultades para acceder a ella, así como para complementar el trabajo de los profesionales de la salud, optimizando sus recursos y mejorando la eficiencia del sistema.

Además, el desafío de combinar mis conocimientos en inteligencia artificial y procesamiento del lenguaje natural con conceptos médicos y científicos fue un factor determinante en mi motivación para realizar este TFG. Busqué aprovechar mis habilidades y conocimientos en tecnología para contribuir al campo de la salud de una manera significativa y aplicar mis capacidades en un proyecto con impacto social y potencial de mejora en la sociedad.

Creo firmemente que la combinación de inteligencia artificial y salud puede generar soluciones innovadoras y accesibles para brindar una atención médica más eficiente y personalizada, y estoy emocionada de poder explorar y desarrollar estas posibilidades a través de este proyecto.

1.2. Objetivos principales del proyecto

El objetivo principal de este Trabajo de Fin de Grado es diseñar, desarrollar e implementar un Chatbot basado en inteligencia artificial para brindar asistencia médica virtual y mejorar la accesibilidad a los servicios de salud. Para lograr este objetivo, se plantean los siguientes objetivos específicos:

1. **Investigación y análisis:** Realizar un estudio exhaustivo sobre los avances recientes en inteligencia artificial, procesamiento del lenguaje natural y su aplicación en el campo de la atención médica. Analizar las necesidades y desafíos del sistema de atención médica actual para identificar áreas de mejora.
2. **Diseño del Chatbot:** Definir la arquitectura y funcionalidades del Chatbot, teniendo en cuenta las necesidades identificadas. Establecer una interfaz intuitiva y amigable para los usuarios, que permita una interacción fluida y una experiencia satisfactoria.
3. **Desarrollo del modelo de lenguaje:** Implementar técnicas de procesamiento del lenguaje natural para comprender y generar respuestas relevantes y precisas. Utilizar algoritmos de aprendizaje automático y técnicas de clasificación para mejorar la comprensión del lenguaje humano y la capacidad de respuesta del Chatbot.
4. **Integración de bases de conocimiento:** Integrar bases de datos y fuentes confiables de información médica para respaldar las respuestas del Chatbot. Asegurar la actualización constante de la información médica y la veracidad de los datos proporcionados.
5. **Validación y evaluación:** Realizar pruebas exhaustivas del Chatbot para evaluar su rendimiento, precisión y usabilidad. Recopilar retroalimentación de

usuarios para realizar mejoras y ajustes necesarios.

6. **Implementación y despliegue:** Implementar el Chatbot en una plataforma accesible y segura, que permita a los usuarios interactuar con él de manera fácil y segura. Garantizar la disponibilidad y escalabilidad del sistema.

En resumen, los objetivos principales de este TFG se centran en diseñar y desarrollar un Chatbot basado en inteligencia artificial que brinde asistencia médica virtual, mejorar la accesibilidad a los servicios de salud, proporcionar respuestas precisas y confiables, y evaluar su impacto en la mejora del sistema de atención médica.

2. REQUISITOS

2.1. Requisitos funcionales

Ref.	Descripción
RF. 1	El sistema debe permitir a los usuarios registrarse proporcionando su nombre, contraseña.
RF. 2	Los usuarios deben poder iniciar sesión en el sistema utilizando sus credenciales.
RF. 3	El sistema debe proporcionar una interfaz de chat interactiva donde los usuarios puedan realizar preguntas y recibir respuestas del Chatbot.
RF. 4	El Chatbot debe ser capaz de comprender y procesar el lenguaje natural para interpretar las preguntas de los usuarios y generar respuestas relevantes.
RF. 5	El Chatbot debe brindar respuestas médicas precisas y confiables basadas en información médica actualizada.
RF. 6	El sistema debe permitir a los usuarios registrar sus citas médicas a partir de un formulario con la información relevante
RF. 7	El sistema debe permitir a los usuarios crear recordatorios a partir de un formulario con la información relevante
RF. 8	El sistema debe permitir a los usuarios crear nuevas entradas en sus historiales médicos.
RF. 9	Los usuarios van a poder visualizar todas las citas que tienen pendientes
RF. 10	Los usuarios van a poder visualizar todas los recordatorios que tengan guardados que no hayan cumplido la fecha
RF. 11	Los usuarios van a poder visualizar todas las entradas de su historial médico desde el comienzo cuando crearon la cuenta.
RF. 12	El Chatbot debe estar integrado con bases de datos y fuentes confiables de información médica para respaldar sus respuestas.
RF. 13	Los usuarios deben poder cerrar sesión cuando lo deseen

2.2. Requisitos no funcionales

Ref.	Descripción
RF. 1	El sistema debe ser fácil de usar y comprensible para usuarios con diferentes niveles de experiencia en tecnología.
RF. 2	El Chatbot debe proporcionar respuestas rápidas y eficientes para mantener una interacción fluida con los usuarios.
RF. 3	El sistema debe garantizar la confidencialidad y protección de los datos de los usuarios, cumpliendo con las regulaciones de privacidad.
RF. 4	El sistema debe estar disponible las 24 horas del día, los 7 días de la semana, para permitir a los usuarios acceder a él en cualquier momento.
RF. 5	El sistema debe ser capaz de manejar un alto volumen de usuarios simultáneos sin afectar su rendimiento.
RF. 6	El Chatbot debe ser capaz de adaptarse a diferentes contextos médicos y responder a una amplia variedad de preguntas y consultas
RF. 7	El sistema debe ser fácil de mantener y actualizar, permitiendo la incorporación de nuevas funcionalidades y corrección de errores.
RF. 8	El sistema debe ser compatible e integrarse con otros sistemas de información médica y herramientas utilizadas en el entorno de atención médica.
RF. 9	La interfaz de usuario debe ser intuitiva, amigable y atractiva visualmente para mejorar la experiencia del usuario.
RF. 10	La interfaz de usuario debe ser fácil de usar, con un diseño intuitivo y una navegación clara para una experiencia del usuario sin complicaciones.
RF. 11	El sistema debe ser resistente a fallos y ser capaz de manejar errores de forma adecuada sin afectar negativamente la experiencia del usuario.
RF. 12	El sistema debe ser accesible para personas con discapacidades visuales, auditivas o motoras, cumpliendo con los estándares de accesibilidad web.
RF. 13	El sistema debe ser compatible con diferentes plataformas y dispositivos, como computadoras de escritorio, tabletas y dispositivos móviles, proporcionando una experiencia consistente en todos ellos.

Ref.	Descripción
RF. 14	Todo el programa del Chatbot debe estar programado en lenguaje de programación de Python
RF. 15	El proyecto tiene que contar con una buena estructura de sus archivos para que todo esté bien organizado y se pueda modificar o añadir cambios con facilidad.
RF. 16	El sistema debe ser compatible con diferentes plataformas y dispositivos, como computadoras de escritorio, tabletas y dispositivos móviles, proporcionando una experiencia consistente en todos ellos.
RF. 17	La interfaz de usuario debe usar tipografía y colores visibles para personas mayores o en estado de dependencia
RF. 18	La base de datos del proyecto tiene que estar bien implementada para que no haya pérdidas de información entre sus tablas
RF. 19	La base de datos del proyecto tiene que estar creada con sqlite3 que proporciona de forma eficaz la creación de bases de datos.
RF. 20	El sistema debe poder escalar horizontalmente agregando más servidores o recursos según sea necesario para manejar un aumento en el tráfico y la demanda.

3. METODOLOGÍA ÁGIL

3.1. Metodologías de desarrollo más comunes

Existen varias metodologías de desarrollo de software, cada una con sus propias ventajas e inconvenientes. A continuación, explicaré tres de las más comunes:

3.1.1. Metodología en Cascada (Waterfall)

Esta metodología es una de las más antiguas. Se basa en un enfoque secuencial, en el que cada etapa del ciclo de vida del software se completa antes de pasar a la siguiente. Las fases típicas incluyen análisis de requisitos, diseño, implementación, pruebas y mantenimiento.

Ventajas:

- Estructura simple y fácil de entender.
- Planificación clara y detallada.
- Resultados predecibles.
- Documentación extensa.

Inconvenientes:

- No se adapta bien a los cambios de requisitos o especificaciones.
- Los clientes no pueden ver el software hasta que se completa.
- No es adecuado para proyectos grandes y complejos.
- Los errores pueden pasar desapercibidos hasta las etapas finales del desarrollo.

3.1.2. Kanban

Esta metodología se centra en la visualización del trabajo en progreso y la optimización del flujo de trabajo. Se utiliza una tabla Kanban para visualizar las tareas pendientes, en progreso y completadas. Los miembros del equipo trabajan en las tareas a medida que se van moviendo a través de la tabla. La metodología se centra en la mejora continua y en la entrega rápida de pequeñas mejoras incrementales.

Ventajas:

- Enfocado en la entrega rápida de software.
- Mayor flexibilidad y adaptabilidad a los cambios.
- Mejora continua y aprendizaje constante.
- Fácil de entender y seguir.

Inconvenientes:

- Menos estructurado y planificado.
- Puede ser difícil medir el progreso a largo plazo.
- Requiere un alto nivel de colaboración y comunicación entre los miembros del equipo.
- No se adapta bien a proyectos complejos y grandes.

3.1.3. SCRUM

Esta metodología se centra en la colaboración del equipo y en la entrega de un producto funcional en ciclos cortos y regulares llamados sprints. En cada sprint, se seleccionan un conjunto de tareas del product backlog y se trabajan hasta que se completan. Al final de cada sprint, se realiza una revisión y se ajusta el backlog para el próximo sprint.

Ventajas:

- Enfocado en la entrega rápida de software.
- Mayor flexibilidad y adaptabilidad a los cambios.
- Mejora continua y aprendizaje constante.
- Mayor colaboración y comunicación entre los miembros del equipo.

Inconvenientes:

- Requiere un alto nivel de compromiso y dedicación del equipo.
- La falta de planificación detallada puede llevar a problemas de estimación.
- No se adapta bien a proyectos complejos y grandes.
- Requiere un Scrum Master capacitado para liderar el proceso.

Cada metodología de desarrollo de software tiene sus propias ventajas e inconvenientes. La elección de una sobre otra depende del trabajo de fin de grado que voy a hacer, de su tamaño y complejidad, de la flexibilidad requerida, del nivel de colaboración y comunicación con el tutor.

3.2. La mejor metodología para hacer un TFG en general

Aunque las metodologías ágiles, como Kanban y Scrum, se utilizan en proyectos de equipo, todavía pueden ser útiles para un TFG individual. En este caso, lo recomendable es elegir una metodología que se adapte mejor a la naturaleza del proyecto individual en este caso a un TFG de informática.

Kanban puede ser una buena opción para un TFG porque se centra en la visualización del trabajo y en la entrega rápida de pequeñas mejoras incrementales. Con Kanban, se podría tener un tablero Kanban para visualizar las tareas y monitorear el progreso, lo que puede ayudar a mantenerse organizado y enfocado en los objetivos clave del TFG.

Scrum también es una opción viable, ya que se enfoca en la colaboración y la entrega de un producto funcional en ciclos cortos y regulares. Aunque no se tenga un equipo como los que se suelen tener en el entorno de trabajo o en las prácticas grupales de la carrera, se podría trabajar con el tutor para recibir retroalimentación y colaboración en el proceso.

3.3. La metodología más eficaz para hacer un Chatbot

Para desarrollar un Chatbot, tanto Kanban como Scrum son adecuadas, ya que ambas se centran en la entrega rápida y continua de mejoras incrementales. Aunque el TFG sea individual, aún se pueden aplicar para asegurar una gestión efectiva del proyecto.

Kanban es una buena opción para el desarrollo de un Chatbot ya que es una metodología enfocada en la visualización del trabajo y en la optimización del flujo de trabajo. Con Kanban, se puede utilizar un tablero Kanban para visualizar las tareas, establecer límites de trabajo en progreso y monitorear el progreso. Además, esta metodología es flexible y adaptable, lo que puede ser beneficioso si se necesita hacer ajustes en el proceso de desarrollo.

Por otro lado, Scrum es también una buena opción si quiero trabajar en ciclos cortos y regulares para la entrega de nuevas funcionalidades o mejoras en el Chatbot. También puede ayudar si se quiere estar en constante comunicación con el tutor o se quiere tener retroalimentación y colaboración con él en el desarrollo del Chatbot.

3.4. Elección final de la metodología a usar

Durante los años en la carrera cada vez que se ha tenido que hacer un trabajo en grupo sobre todo los dos últimos cursos se han usado metodologías ágiles, probando la mayoría ya que las distintas asignaturas proponen usarlas y al haberse probado todas, SCRUM y Kanban son las mejores, se ha profundizado más en la asignatura de Metodologías del Desarrollo Ágil donde también se han puesto en práctica.

Ahora se tiene que tomar una decisión sobre cuál de las dos es mejor y más conveniente, teniendo en cuenta que actualmente se está haciendo práctica de la metodología SCRUM para trabajar de forma organizada y en equipo es la que se tiene

más reciente y se usa cada día por lo tanto es la que mejor se conoce y tal vez la que más puede funcionar.

Aunque en su mayor medida utilice SCRUM, también es una buena idea implementar un tablero Kanban porque es más visual tener todas las tareas organizadas según su estado de forma física en la zona de trabajo, aunque SCRUM posee el backlog donde también están todas las tareas organizadas por su prioridad y se puede hacer cuantía de cosas con ellas, el tablero Kanban va a aportar más.

En conclusión, se va a utilizar una metodología híbrida SCRUM-Kanban.

4. HERRAMIENTAS PARA EL DESARROLLO DE CHATBOTS

4.1. Comparativa de las opciones

4.1.1. Dialogflow

Es una plataforma de desarrollo de Chatbots de Google que utiliza inteligencia artificial y aprendizaje automático para crear conversaciones naturales y fluidas.

Ventajas:

- Se puede integrar con más productos de Google.
- Tiene un gran procesamiento del lenguaje natural en más de 20 idiomas.
- Se puede unir a otros servicios externos como Slack, Telegram, Facebook, etc.
- Gran cantidad de herramientas y opciones para el desarrollo de Chatbots.

Inconvenientes:

- Al ser de Google, depende de este, su estabilidad, flexibilidad y políticas de uso.
- La versión gratuita es limitada, puede tener costes adicionales si superamos el límite.

4.1.2. Rasa

Es una plataforma de desarrollo de Chatbots de código abierto que utiliza inteligencia artificial y aprendizaje automático para comprender y responder a los mensajes de los usuarios.

Ventajas:

- Permite tener el control total sobre el modelo de lenguaje natural, se puede entrenar el modelo y personalizarlo según unas necesidades específicas.
- Se puede integrar con muchas herramientas lo que permite aún más personalizar el desarrollo del Chatbot.
- Ofrece una gran cantidad de modelos desde los más simples a otros muy complejos.
- Es de código abierto con una comunidad muy activa, es fácil beneficiarse de las contribuciones de otros usuarios

Inconvenientes:

- Puede resultar difícil aprender a usarlo para aquellos sin experiencia en el desarrollo de Chatbots de aprendizaje automático.
- Requiere una configuración y mantenimientos adecuados para funcionar bien, puede ser tedioso para los que no tienen experiencia.
- Tiene limitaciones en la versión gratuita.
- Hay menos opciones de integración que otras plataformas del mismo sector.

4.1.3. SCRUM

Es una empresa de investigación en inteligencia artificial que ofrece una amplia gama de herramientas, modelos y servicios de IA. Es la más famosa actualmente ya que es la que ha desarrollado Chat GPT.

Ventajas:

- Tiene una amplia gama de servicios y herramientas, modelos de lenguaje natural, plataforma de aprendizaje.
- Las herramientas y modelos tienen una alta calidad y rendimiento, proporcionan resultados precisos y confiables.
- Al ser una empresa de investigación está en constante desarrollo, mejorando e innovando sus herramientas y modelos.
- Su comunidad es enorme, a parte proporciona soporte a sus usuarios.

Inconvenientes:

- Es muy costoso para usuarios que quieran usar servicios avanzados o personalizados.
- El acceso a algunos modelos es limitado para evitar su uso indebido.
- Los usuarios dependen de la empresa y de la capacidad de ésta de seguir dando soporte a todos sus modelos y plataformas.

4.1.4. OpenChatKit

Es una biblioteca de Python de código abierto que permite a los desarrolladores crear Chatbots de manera sencilla y personalizada.

Ventajas:

- Es fácil de usar, puede ser usado por desarrolladores con nociones básicas de Python.

- Permite personalizar los Chatbots de acuerdo con las necesidades y requisitos que deseen los desarrolladores.
- Es código abierto, es una biblioteca de Python, cualquiera puede acceder a ella.
- Se integra fácilmente con otras herramientas y servicios de inteligencia artificial, reconocimiento de voz, lenguaje natural, etc.
- Tiene una documentación muy completa y fácil de seguir para hacer más fácil a los desarrolladores usarlo correctamente.

Inconvenientes:

- Tiene menos funcionalidades y modelos de lenguaje natural que otras herramientas y bibliotecas de Chatbot.
- Depende de otras herramientas para algunas de sus funcionalidades.
- La comunidad es más pequeña que el resto de herramientas, lo que limita el soporte y la solución de problemas.

4.1.5. ChatterBot

Es una biblioteca de Python que permite a los desarrolladores crear Chatbots de manera sencilla y rápida.

Ventajas:

- Es fácil de usar, puede ser usado por desarrolladores con nociones básicas de Python.
- Utiliza técnicas de aprendizaje automático para mejorar la precisión y relevancia de las respuestas del bot.
- Es código abierto, es una biblioteca de Python, cualquiera puede acceder a ella.
- Se integra fácilmente con otras herramientas y servicios de inteligencia artificial, reconocimiento de voz, lenguaje natural, etc.
- Permite a los desarrolladores personalizar sus Chatbots de acuerdo con sus necesidades y requisitos.

Inconvenientes:

- Tiene menos funcionalidades y modelos de lenguaje natural que otras herramientas y bibliotecas de Chatbot.
- Depende de otras herramientas para algunas de sus funcionalidades.
- La comunidad es más pequeña que el resto de herramientas, lo que limita el soporte y la solución de problemas.

4.2. Consideraciones para la elección de la herramienta

1. Considerar utilizar una plataforma de Chatbot que tenga características específicas para servicios sociales y salud. Algunas plataformas tienen integraciones específicas para este tipo de aplicaciones, como la gestión de citas médicas, la identificación de síntomas de enfermedades, etc.
2. Asegurarse de que el Chatbot cumpla con los requisitos legales y éticos. Por ejemplo, puede haber regulaciones específicas en un país o región sobre la privacidad y seguridad de los datos de los usuarios, especialmente si se está tratando con información médica.
3. Realizar pruebas del Chatbot, para obtener comentarios sobre su eficacia y usabilidad.
4. Utilizar técnicas de aprendizaje automático y procesamiento de lenguaje natural para mejorar la capacidad del Chatbot para comprender y responder a las solicitudes de los usuarios de manera efectiva y natural.
5. Asegurarse de que la interfaz del Chatbot sea intuitiva y fácil de usar, especialmente para las personas mayores y aquellas que pueden tener discapacidades o limitaciones en su uso de dispositivos móviles o computadoras.
6. Hay que tener en cuenta de que la utilización de la herramienta no es limitada o que no es una versión gratuita que solo durará un tiempo o hasta que finalice la prueba.
7. Se deberá usar una herramienta cuya instalación y mantenimiento sea fácil e intuitivo y que se pueda hacer sin costo ninguno, es decir que sea código abierto.
8. Es preferible que la herramienta tenga un buen soporte o documentación por si aparece algún error o dificultad que el buscar una solución no se haga difícil y no se malgaste mucho tiempo en ello.

En resumen, para desarrollar un Chatbot para servicios sociales dirigido a personas en situación de dependencia, es importante considerar todas estas características específicas necesarias para este tipo de aplicación.

4.3. Elección final de la herramienta a usar

Tras haber examinado las distintas herramientas disponibles para crear un Chatbot se debe elegir una para usarse en la realización de trabajo fin de grado. Teniendo en cuenta las ventajas e inconvenientes de cada una de ellas junto con las

necesidades y consideraciones para realizar el proyecto. Se pueden descartar varias de ellas.

Como muchas de estas tienen demasiadas limitaciones en su versiones gratuitas como Rasa, Dialogflow y OpenAI aunque sean unas de las mejores, las más sencillas e intuitivas de usar ya que poseen un gran procesamiento del lenguaje natural, generando también resultados óptimos, preciso y confiables, además de contar con comunidades muy grandes y soporte en caso de tener algún problema o dificultad, se deben descartar ya que no se van a pagar sus versiones completas y la versión gratuita es muy limitada con pocas funcionalidades.

Ya que no podemos elegir las anteriores nos quedan dos opciones ChatterBot y OpenChatKit, estas dos son muy similares ambas son bibliotecas de Python que requieren un alto nivel de programación por tanto no son accesibles para todo el mundo como las otras, aunque esto proporciona poder personalizar a grandes rasgos los Chatbot y adaptarlos a las necesidades que se quieren obtener. Aunque no poseen soporte o comunidades tan grandes como las anteriores si tienen mucha documentación para ayudar al usuario. Por lo tanto ambas son una buena opción.

En un principio se había decantado por usar OpenChatKit ya que tiene bastante documentación y se ha encontrado más fácilmente que la otra además de que su instalación es muy sencilla con unos simples comandos en la terminal, solo requiere tener instalado Python y parecía que no iba a dar muchos problemas. Finalmente si los dio, demasiado aunque parecían pocos pasos para su implementación necesitaba muchos requerimientos que también tienen que instalarse además que no era apto para las prestaciones de los ordenadores que se tenían, requieren unas altas prestaciones por lo tanto no funcionaba, posteriormente se optó por usar Open Assistant otra herramienta muy parecida a la anterior pero que también dio muchos problemas en su instalación y que finalmente no se pudo usar después de muchas pruebas e intentos de usarla.

Por lo tanto todo este proyecto finalmente se ha hecho con la API de Chat GPT es decir OpenAI que al utilizar una cuenta de la universidad proporciona varios meses de prueba y hacer un número determinado de consultas bastante alto que no se agotara por mucho que se hagan pruebas con el Chatbot.

5. HISTORIAS DE USUARIO

Identificador: HU.1	Realizar preguntas médicas generales
Descripción: Como usuario, quiero poder realizar preguntas médicas generales al Chatbot para obtener información confiable y precisa.	
Prioridad: 1	Iteración: 2
Tareas relacionadas: <ul style="list-style-type: none">• Implementar un módulo de procesamiento de lenguaje natural para entender las preguntas del usuario.• Integrar una base de conocimiento médico para proporcionar respuestas precisas.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Enviar preguntas médicas generales al Chatbot y verificar que las respuestas sean relevantes y correctas.• Probar preguntas con diferentes variaciones y sinónimos para asegurar la comprensión adecuada.	
Observaciones:	

Identificador: HU.2	Guardar citas médicas
Descripción: Como usuario, quiero poder guardar las citas médicas a través del Chatbot para no olvidarme de ellas y tener sus datos correspondientes guardados.	
Prioridad: 2	Iteración: 4
Tareas relacionadas: <ul style="list-style-type: none">• Diseñar una interfaz de programación de citas intuitiva y fácil de usar.• Integrar el Chatbot con el sistema de gestión de citas médicas existente.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Simular guardar las citas médicas a través del Chatbot y verificar que las citas se registren correctamente en el sistema.• Probar diferentes casos, como programar citas en diferentes fechas y horarios, y asegurarse de que se reflejen correctamente.	
Observaciones:	

Identificador: HU.3	Visualizar Recordatorios
Descripción: Como usuario, quiero visualizar todos mis recordatorios guardados a través del Chatbot, aquellos que no han cumplido con la fecha.	
Prioridad: 2	Iteración: 4
Tareas relacionadas: <ul style="list-style-type: none">• Implementar un sistema de recordatorios y notificaciones para enviar alertas al usuario en los momentos adecuados.• Permitir al chat mostrar por la conversación los recordatorios pendientes.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Configurar recordatorios a través del Chatbot y verificar que se muestren correctamente.• Probar diferentes configuraciones de recordatorio, como horarios específicos y frecuencias personalizadas, y asegurarse de que se respeten.• Comprobar que los recordatorios de fechas pasadas no se muestren.	
Observaciones:	

Identificador: HU.4	Obtener recomendaciones de tratamiento según síntomas
Descripción: Como usuario, quiero poder proporcionar mis síntomas al Chatbot y recibir recomendaciones de tratamiento adecuadas.	
Prioridad: 3	Iteración: 5
Tareas relacionadas: <ul style="list-style-type: none">• Integrar una base de conocimiento médico para proporcionar recomendaciones basadas en los síntomas ingresados.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Ingresar síntomas específicos al Chatbot y verificar que las recomendaciones de tratamiento sean coherentes y apropiadas para los síntomas proporcionados.• Probar diferentes combinaciones de síntomas y verificar que las recomendaciones sean precisas y relevantes.	
Observaciones:	

Identificador: HU.5	Obtener información sobre médicos especialistas y centros médicos cercanos
Descripción: Como usuario, quiero poder obtener información sobre médicos especialistas y centros médicos cercanos a través del Chatbot.	
Prioridad: 3	Iteración: 5
Tareas relacionadas: <ul style="list-style-type: none">• Integrar una base de datos de médicos especialistas y centros médicos con el Chatbot.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Realizar consultas sobre médicos especialistas y centros médicos cercanos a través del Chatbot y verificar que la información proporcionada sea precisa y actualizada.• Probar diferentes búsquedas con ubicaciones y especialidades médicas variadas y asegurarse de que los resultados sean relevantes.	
Observaciones:	

Identificador: HU.6	Visualizar Citas
Descripción: Como usuario, quiero visualizar todas mis citas guardadas a través del Chatbot, aquellas que no han cumplido con la fecha.	
Prioridad: 2	Iteración: 3
Tareas relacionadas: <ul style="list-style-type: none">• Implementar un sistema de citas en la que los usuarios guarden la información relevante de estas.• Permitir al chat mostrar por la conversación los recordatorios pendientes•	
Pruebas de Aceptación: <ul style="list-style-type: none">• Configurar las citas a través del Chatbot y verificar que se muestren correctamente.• Probar diferentes configuraciones de citas, como horarios específicos y lugares personalizados, y asegurarse de que se respeten.• Comprobar que las citas de fechas pasadas no se muestren.•	
Observaciones:	

Identificador: HU.7	Visualizar Historial
Descripción: Como usuario, quiero visualizar todo mi historial médico guardado a través del Chatbot.	
Prioridad: 2	Iteración: 3
Tareas relacionadas: <ul style="list-style-type: none">• Implementar una tabla en la base de datos donde se guarde todo el historial del usuario.• Permitir al chat mostrar por la conversación el historial del usuario.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Crear nuevas entradas al historial a través del Chatbot y verificar que se muestren correctamente.• Comprobar que solo se muestran las entradas al historial de ese usuario no se entremezcla con los de otros.	
Observaciones:	

Identificador: HU.8	Proporcionar información sobre enfermedades comunes
Descripción: Como usuario, quiero recibir información confiable sobre enfermedades comunes a través del Chatbot para aumentar mi conocimiento médico.	
Prioridad: 3	Iteración: 5
Tareas relacionadas: <ul style="list-style-type: none">• Integrar una base de conocimiento médico actualizada que contenga información sobre enfermedades comunes.• Diseñar un sistema de búsqueda y recuperación de información para proporcionar respuestas relevantes a las consultas sobre enfermedades.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Crear nuevas entradas al historial a través del Chatbot y verificar que se muestren correctamente.• Comprobar que solo se muestran las entradas al historial de ese usuario no se entremezcla con los de otros.	
Observaciones:	

Identificador: HU.9	Brindar apoyo emocional y motivacional
Descripción: Como usuario, quiero recibir apoyo emocional y motivacional a través del Chatbot para mejorar mi bienestar emocional.	
Prioridad: 3	Iteración: 5
Tareas relacionadas: <ul style="list-style-type: none">• Desarrollar respuestas y mensajes de apoyo emocional y motivacional adecuados.• Implementar técnicas de inteligencia artificial para adaptar las respuestas según las necesidades y emociones del usuario.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Interactuar con el Chatbot en busca de apoyo emocional y motivacional y verificar que los mensajes generados sean reconfortantes y útiles.• Probar diferentes estados emocionales y necesidades de apoyo y asegurarse de que las respuestas sean relevantes y empáticas.	
Observaciones:	

Identificador: HU.10	Creación de la base de datos
Descripción: Como administrador quiero crear una base de datos donde guardar toda la información de los usuarios.	
Prioridad: 1	Iteración: 3
Tareas relacionadas: <ul style="list-style-type: none">• Crear las tablas de la base de datos.• Implementar algunas funciones básicas como ingresar u obtener datos.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Desde el prompt de sqlite probar a crear las tablas y la base de datos.• Insertar alguna entrada en las tablas para ver que funcionan correctamente.	
Observaciones:	

Identificador: HU.11	Crear Usuario
Descripción: Como usuario, quiero poder crearme una cuenta en la página web del ChatBot.	
Prioridad: 1	Iteración: 3
Tareas relacionadas: <ul style="list-style-type: none">• Crear la tabla de usuarios en la base de datos• Crear la función que compruebe los datos ingresados por el usuario	
Pruebas de Aceptación: <ul style="list-style-type: none">• Crear un nuevo usuario y que este se ingrese correctamente en la base de datos• Crear un usuario que no exista y que salte el mensaje por pantalla de que no es válido el nombre de usuario.	
Observaciones:	

Identificador: HU.12	Iniciar Sesión
Descripción: Como usuario, poder iniciar sesión en la web del Chatbot para conversar con él e interactuar con mis datos guardados.	
Prioridad: 1	Iteración: 3
Tareas relacionadas: <ul style="list-style-type: none">• Comprobar que las credenciales del usuario son correctas.• Implementar que cuando se inicie sesión, el usuario acceda a su espacio personal de conversación con el Chatbot.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Iniciar sesión con un usuario ya creado y comprobar que se puede acceder sin problemas.• Intentar entrar con un usuario que no existe y comprobar que salta el mensaje de error.	
Observaciones:	

Identificador: HU.13	Cerrar sesión
Descripción: Como usuario, quiero recibir apoyo emocional y motivacional a través del Chatbot para mejorar mi bienestar emocional.	
Prioridad: 3	Iteración: 4
Tareas relacionadas: <ul style="list-style-type: none">● Implementar una función de cierre de sesión en el Chatbot.● Actualizar el estado de inicio de sesión del usuario en la base de datos.	
Pruebas de Aceptación: <ul style="list-style-type: none">● Iniciar sesión como usuario y verificar que el Chatbot muestre la opción de cerrar sesión.● Cerrar sesión y asegurarse de que el Chatbot redirija al usuario a la página de inicio de sesión y elimine los datos de sesión previos.	
Observaciones:	

Identificador: HU.14	Insertar Recordatorios
Descripción: Como usuario, quiero poder guardar los recordatorios a través del Chatbot para no olvidarme de ellos y tener sus datos correspondientes guardados.	
Prioridad: 2	Iteración: 3
Tareas relacionadas: <ul style="list-style-type: none">● Diseñar una interfaz de programación de recordatorios intuitiva y fácil de usar.● Guardar en la base de datos los recordatorios con su información importante.	
Pruebas de Aceptación: <ul style="list-style-type: none">● Crear un nuevo recordatorio y comprobar que se ha guardado con éxito.● Comprobar que al pinchar en este campo nos lleva al formulario respectivo.	
Observaciones:	

Identificador: HU.15	Guardar nueva entrada en el historial
Descripción: Como usuario, quiero poder guardar las nuevas entradas a mi historial médico de forma fácil e intuitiva	
Prioridad: 2	Iteración: 3
Tareas relacionadas: <ul style="list-style-type: none">• Diseñar una interfaz de programación para insertar las nuevas entradas del historial médico.• Crear un formulario para que se guarde en la base de datos la respectiva información.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Simular guardar una nueva entrada del historial médico y que se registre correctamente en el sistema.• Comprobar que al pinchar en insertar nueva entrada, lleva al formulario para rellenar con la información correspondiente.	
Observaciones:	

Identificador: HU.16	Mejorar la accesibilidad del Chatbot
Descripción: Como desarrollador, quiero mejorar la accesibilidad del Chatbot para garantizar que pueda ser utilizado por personas con discapacidades visuales o motoras.	
Prioridad: 3	Iteración: 5
Tareas relacionadas: <ul style="list-style-type: none">• Implementar un diseño y una estructura de chat accesibles, asegurando un contraste adecuado de colores y un tamaño de fuente legible.• Agregar soporte para lectores de pantalla y teclas de acceso rápido para una navegación fluida.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Realizar pruebas de accesibilidad con herramientas automatizadas y manuales para verificar la conformidad con las pautas de accesibilidad.• Solicitar comentarios a personas con discapacidades para evaluar la usabilidad y accesibilidad del Chatbot.	
Observaciones:	

Identificador: HU.17	Mejorar la interfaz de usuario del Chatbot
Descripción: Como usuario, quiero una interfaz de usuario intuitiva y atractiva en el Chatbot para facilitar la interacción y mejorar la experiencia de uso.	
Prioridad: 3	Iteración: 5
Tareas relacionadas: <ul style="list-style-type: none">• Realizar un diseño de interfaz de usuario atractivo y coherente con los principios de diseño de materiales o cualquier otro marco de diseño elegido.• Mejorar la disposición de los elementos en pantalla para una navegación y comprensión claras.• Añadir elementos visuales interactivos, como botones y tarjetas, para mejorar la experiencia de uso.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Realizar pruebas de usabilidad con usuarios para evaluar la facilidad de uso y la eficiencia de la interfaz de usuario.• Recopilar retroalimentación de los usuarios sobre la apariencia y la experiencia general de la interfaz de usuario del Chatbot.	
Observaciones:	

Identificador: HU.18	Puesta en marcha del Chatbot
Descripción: Como administrador quiero tener una base básica del Chatbot funcionando para comenzar a implementar el entrenamiento y las funcionalidades.	
Prioridad: 1	Iteración: 1
Tareas relacionadas: <ul style="list-style-type: none">• Instalaciones necesarias para usar la API• Creación del primer código básico para que funcione el bot desde terminal.	
Pruebas de Aceptación: <ul style="list-style-type: none">• Preguntarle algo al Chatbot por terminal y que responda algo razonable.	
Observaciones:	

6. ESTRUCTURA DE LOS SPRINTS

Como este TFG fue solicitado en el segundo cuatrimestre se comenzó a hacer en torno al 25 de Marzo cuando salió la adjudicación de estos.

Contando con esto se planificó según la metodología SCRUM de la siguiente forma:

<i>Fecha de Entrega</i>	<i>Nombre de la Entrega</i>
21 de Abril	Elección de la Metodología de Desarrollo
9 de Mayo	Revisión y Elección de Herramientas
28 de Mayo	Sprint 1
4 de Junio	Sprint 2
11 de Junio	Sprint 3
18 de Junio	Sprint 4
25 de Junio	Sprint 5
2 de Julio	Sprint 6
9 de Julio	Sprint Final

El primer Sprint es al que se le dedicaba supuestamente más tiempo ya que es el de la instalación de la herramienta a partir de ahí cada uno de los sprints deberían hacerse en una semana cada uno de ellos.

Como se ha comentado anteriormente en esta memoria en el apartado de elección de la herramienta que se iba a usar para hacer la API del Chatbot, hubo un gran retraso en la entrega de casi tres semanas más porque la herramienta principal tras probar varias semanas en instalarla y llegar a la fecha de la entregar sin que pasara las pruebas de la historia de usuario de ese sprint que era la de tener un chat básico desde la terminal, hubo que cambiar de herramienta es decir empezar totalmente de cero.

Se volvió a intentar durante otro par de semanas con otra herramienta esta daba mejores resultados que la anterior y se avanzó un poco más pero igualmente no se pudo usar para crear nuestro Chatbot requería demasiadas cosas imposibles de descargar y obtener en mi ordenador.

Todo esto se detalla haciendo más hincapié en el apartado correspondiente a la implementación del Sprint 1. Finalmente se consiguió poner en marcha utilizando la API de OpenAI la que se ha usado hasta el final del proyecto.

Tras este traspié en las fechas de entrega quedó mucho menos tiempo a cada sprint por lo tanto se hicieron 5 sprints siendo el primero el más largo ya que es el de la instalación con todos los problemas que conllevo. Quedándose los otros 4 con una semana para cada uno esto significaba subir la carga diaria dedicada a cada sprint además de que en cada uno de ellos se abordaron muchas más funcionalidades e historias de usuario.

Finalmente se organizó de esta forma los sprints:

Nº de Sprint	HU Abordadas
Sprint 1	HU.18
Sprint 2	HU.1, HU.18
Sprint 3	HU.10, HU.11, HU.12, HU.13
Sprint 4	HU.2, HU.3, HU.6, HU.7, HU.14, HU.15
Sprint 5	HU.4, HU.5, HU.8, HU.9, HU.16, HU.17

A continuación se hará una exhaustiva documentación de cada uno de ellos, explicando los objetivos de cada uno, sus implementaciones, retos abordados, problemas que aparecieron soluciones a ellos, etc.

7. SPRINT 1

INSTALACIÓN DE LA HERRAMIENTA

Este Sprint fue el que más problemas dio ya que hubo que cambiar tres veces de herramienta ya que las dos primeras no eran soportadas por el dispositivo que se está usando para crear el Chatbot, por eso este capítulo de la memoria se va a dividir en dos secciones, de cada una de las dos herramientas que se utilizaron para comenzar a entrenar el Chatbot pero lamentablemente no fue posible.

La primera opción fue OpenChatKit la cual se estuvo varias semanas buscando como instalarla, qué dependencias se necesitaban y como solucionar todos los problemas que iban surgiendo, como las ventajas, los inconvenientes y más información relativa a esta herramienta ha sido ya explicada anteriormente en su apartado correspondiente voy a redactar directamente su implementación, los problemas que ocasiono y las posibles soluciones que se le buscaron a estos.

7.1. OpenChatKit

En este caso lo primero fue informarse de que se necesitaba como el portátil en el que se está haciendo el proyecto tiene un sistema operativo Windows en un principio parecía que no había problema en hacerlo en este, por eso se comenzó a hacer en este sistema operativo, el primer paso fue descargar la última versión de Python en el ordenador, y ejecutar por terminal el comando:

```
pip install openchatkit
```

Este debería instalar y descargar OpenChatKit junto con sus dependencias pero no fue así, dio un error de que no reconoció el comando como interno o externo al programa para ello se verificó la versión de Python.

```
python --version
```

La muestra por lo tanto Python está instalado y ese no es el problema de que no se haya instalado bien, lo próximo es comprobar la variable de entorno PATH, asegurarse de que está configurada correctamente para incluir la ubicación de la instalación.

```
echo %PATH%
```

Como no aparece la ubicación de la instalación de Python, se localiza la carpeta de instalación, se agrega Python al path manualmente con la siguiente orden:

```
export PATH="/usr/local/bin/python:$PATH"
```

Después reiniciamos la terminal y tras realizar estos pasos ya debería funcionar, ahora cuando volvemos a hacer `pip install OpenChatKit` ha habido un error al preparar los metadatos no se ha ejecutado correctamente `pyproject.toml`. Algunas soluciones para este problema era actualizar el pip.

```
pip install --upgrade pip
```

Se verifica la compatibilidad de la versión de Python que en este caso se requiere que sea mayor que la 3.6 y se tiene instalada la 3.11, así que está correcto, después también podría servir limpiar la caché de pip ya que puede causar errores.

```
pip cache purge
```

Y se actualiza `setuptools` y `wheel` para tener sus versiones más actualizadas con los siguientes comandos:

```
pip install --upgrade setuptools
pip install --upgrade wheel
```

Se vuelve a instalar `openchatkit` y da un nuevo error de que no encuentra la distribución de `faiss-gpu=1.7.2`, esta biblioteca se usa para acelerar el procesamiento de las operaciones y requiere configuraciones específicas según el sistema. `Faiss-gpu` requiere una gpu compatible con cuda, esta se verifica con:

```
nvcc --version
```

Como seguía apareciendo el error, se instaló `faiss` desde github siguiendo los pasos del fichero de instalación, el problema es que se instalaba con conda, así que hubo que instalarla para poder obtener `faiss`.

```
# CPU-only version
$ conda install -c pytorch faiss-cpu

# GPU(+CPU) version
$ conda install -c pytorch faiss-gpu

# or for a specific CUDA version
$ conda install -c pytorch faiss-gpu cudatoolkit=10.2 # for CUDA
```

10.2

El problema al hacer esta instalación es que no se podía hacer en Windows ya que no se puede instalar la versión GPU, por tanto se tuvo que optar por instalarlo todo en el WSL de Windows para poder usar los comandos de Linux.

Todo esto se volvió a repetir en el WSL pero hubo un problema y es que este automáticamente se descarga en el disco C dejándolo prácticamente sin espacio y obligando a moverlo al disco D, esto no se hace desde los programas del PC o desde la configuración hay que seguir una serie de comandos para conseguirlo:

Se necesita saber que usuario se está usando en la terminal de Linux:

```
$whoami  
>claudia
```

Se necesita conocer el nombre de la distribución y la versión de WSL ejecutando en el powershell:

```
$wsl --list --verbose
```

Dando como resultado:

```
PS C:\Users\claud> wsl --list --verbose  
NAME      STATE      VERSION  
* Ubuntu   Stopped    1  
PS C:\Users\claud>
```

Primero se necesita cerrar todas las terminales que estén usando WSL y luego apagarlo para evitar cualquier corrupción de datos.

```
$wsl --shutdown
```

Se crea una copia de seguridad de la distribución:

```
$mkdir D:\backup  
$wsl --export Ubuntu D:\backup\ubuntu.tar
```

Se comprueba que se tiene la copia de seguridad:

```
D:\backup/ubuntu.tar
```

Después se elimina la distribución del disco C con:

```
$wsl --unregister Ubuntu
```

Y ahora se descarga Ubuntu en el disco D, abrimos una terminal en este y ejecutamos el comando:

```
$wsl --install -d Ubuntu-22.04
```

Escribimos el nuevo nombre de usuario y contraseña y ya tenemos nuestra cuenta hecha.

Después de mover todo el WSL hay que volver a repetir los pasos, en este caso se necesita la versión 3.10.9 de Python

```
wget https://www.python.org/ftp/python/3.10.9/Python-3.10.9.tgz
tar -xf Python-3.10.9.tgz
cd Python-3.10.9
```

Para instalar todas las dependencias como, make g++ y gcc usamos este comando:

```
sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-
dev libnss3-dev libssl-dev libsqlite3-dev libreadline-dev libffi-dev
libbz2-dev

./configure --enable-optimizations
make -j 10
sudo make altinstall
```

Instalamos pip:

```
sudo apt install python3-pip
```

Y algunos módulos necesarios:

```
pip3 install torch==1.13.1
pip3 install torchvision==0.14.1
pip3 install torchaudio==0.13.1
```



```
pip3 install transformers==4.21.1
pip3 install accelerate==0.17.1
pip3 install datasets==2.10.1
pip3 install loguru==0.6.0
pip3 install netifaces==0.11.0
pip3 install wandb==0.13.10
pip3 install zstandard==0.20.0
```

Instalamos otros paquetes necesarios para ejecutar el OpenChatKit:

```
sudo python -m pip install pyyaml
pip install faiss-gpu
sudo apt-get install python-six
python3 -m pip install six
```

Ya está todo el environment instalado ahora ejecutamos:

```
python3.10 inference/bot.py --model togethercomputer/Pythia-Chat-Base-7B
```

Como no hay suficiente memoria nos da un error, para solucionarlo tenemos que crear una carpeta offload en la raíz del proyecto donde se debería guardar toda la carga que sobrepasa la memoria.

Aun así al volver a ejecutar este comando el ordenador se queda eternamente pensando y ejecutándolo sin mostrar ningún resultado pero tampoco ningún error, tras mirar en distintos foros, en los issues del GitHub de OpenChatKit y rebuscar entre todo el código por si había algún comentario que indicase como solucionarlo se encontró uno que decía todo lo contrario, diciendo este que se necesitaba un ordenador de muy altas prestaciones para poder utilizar este Chatbot training, y por lo tanto ni mi portátil ni mi ordenador de sobremesa podían alcanzarlas y se tuvo que dejar de utilizar esta herramienta por dicho motivo, porque en la facultad tampoco había ningún dispositivo que llegaría a tener 42 GB de VRAM, en este caso mi portátil tenía 12 GB de VRAM.

7.2. Open Assistant

Open Assistant es un proyecto de código abierto, tiene como objetivo proporcionar una inteligencia artificial conversacional para todos. Se basa en modelos de lenguaje que pueden entender tareas, interactuar con sistemas de terceros y obtener información dinámicamente para hacerlo.

7.2.1. Características

- **Es de código abierto:** cualquiera puede acceder al código fuente, los modelos y los datos de entrenamiento de Open Assistant, modificarlos y contribuir a su mejora. El proyecto está organizado por LAION y personas de todo el mundo.
- **Usa modelos de lenguaje de gran tamaño:** se basa en GPT-3 o GPT-NeoX, tienen miles de millones de parámetros y pueden generar textos coherentes y diversos sobre cualquier tema. Estos modelos se entrenan con datos extraídos de Internet y otras fuentes, lo que les permite tener un conocimiento general.
- **Aplica aprendizaje por refuerzo con retroalimentación humana:** adapta los modelos de lenguaje pre-entrenados y los mejora mediante los feedbacks de los usuarios al asistente sobre la calidad y relevancia de sus respuestas, lo que le permite aprender de sus errores y aciertos y ajustar su comportamiento.
- **Puede realizar tareas complejas:** tanto como interactuar con sistemas externos o buscar información dinámicamente.

7.2.2. Comparativa con otras IAs conversacionales

- **Es más accesible:** es gratuito tanto para usarlo como modificarlo, mientras que otras IA conversacionales requieren pagar para acceder a ellas. Además, Open Assistant está disponible en varios idiomas y se puede usar desde cualquier plataforma o dispositivo con conexión a Internet.
- **Es más transparente:** es un proyecto de código abierto que comparte su código fuente, sus modelos y sus datos de entrenamiento con la comunidad, mientras que otras son propietarias y no revelan cómo funcionan ni qué datos usan.
- **Es más personalizable:** permite a los usuarios modificar el código fuente, los modelos y los datos de entrenamiento del asistente para adaptarlo a sus necesidades y preferencias, mientras que otras IA conversacionales ofrecen pocas opciones de personalización o requieren permisos especiales para hacerlo.

7.2.3. Instalación

Para comenzar se clona el repositorio de Github de Open Assistant en la carpeta raíz del proyecto con la orden:

```
git clone https://github.com/LAION-AI/Open-Assistant.git
```

Como ya se tiene todo el código fuente, hay que montar la imagen del docker, antes de nada como no se va a ejecutar en un Linux, sino en el wsl2 de Windows se tiene que hacer el siguiente comando cada vez que se abra una terminal, para no repetir esto constantemente, se añade la orden al .bashrc para que se ejecute automáticamente cada vez que se abra una nueva terminal o pestaña de Ubuntu.

```
export DOCKER_DEFAULT_PLATFORM=linux/amd64
```

Para construir el docker se puede hacer de dos formas y solo hay que hacerlo al comienzo de la instalación, el resto de veces que queramos usar el Open Assistant no hay que hacerlo, la primera es usando dos órdenes para construir tanto el frontend como backend del programa.

```
#To start a database and work and the backend.  
docker compose --profile backend-dev up --build --attach-  
dependencies  
#To start the services needed to work on the frontend  
docker compose --profile frontend-dev up --build --attach-  
dependencies
```

La otra forma es la siguiente con la cual se construyen el backend y el frontend juntos en una misma imagen.

```
docker compose --profile ci up --build --attach-dependencies
```

Cuando ya estén construidas las imágenes hay que levantarlas es decir hacer que comiencen a ejecutarse para poder probar el Chatbot y que todo esté conectado.

```
docker compose --profile inference up -d
```

Este comando no debe tardar en ejecutarse y se debe hacer cada vez que se quiera trabajar con el bot, este dará como resultado por pantalla que los contenedores están correctos y ejecutándose. Se puede utilizar un comando para ver los logs y así tener información de que todo se está ejecutando correctamente y sin errores o pérdidas de conexión.

```
docker compose logs -f \ inference-server \ inference-worker
```

Tras tener ya el docker funcionando hay que comprobar que el Chatbot funciona y está todo correcto, para ello hay que seguir una serie de pasos para crear un environment donde poder trabajar y activarlo para iniciar el cliente del chat y que se puede uno comunicar con él.

```
#Se viaja a la carpeta situada en el directorio de Open Assistant
cd text-frontend

#Se crea el environment, solo la primera vez para crear un entorno
virtual de python.
python3 -m venv venv/

#Se activa el environment para instalar despues paquetes
source venv/bin/activate

#Se instalan los paquetes requeridos
pip install -r requirements.txt

#Se iniciar el cliente de chat
python __main__.py
```

Ahora aparecerá el Chatbot hablando sobre algo o preguntando, hace pequeños task aleatorios que se deben responder, con los que se comprueba que todo funciona.

Hasta aquí ya se tiene la base del proyecto, ahora hay que entrenar el Chatbot y para ello hay que instalar cuantiosas dependencias. Para facilitar esto el proyecto tiene un archivo que al ejecutarlo instala todo lo necesario pero es el que ha dado problemas y por eso no se ha podido continuar. Esta es la orden que se ha de utilizar para ejecutarlo.

```
#Primero hay que ir a la carpeta model dentro del directorio de Open
Assistant
cd model
#Después ejecutamos el comando para instalar las dependencias
pip install -e .
```

7.2.4. Dificultades durante la instalación

Lo primero de todo es que hay que tener Python instalado en una versión 3.10. Aquí ya estaba instalado ya que se usó para la instalación de los modelos de OpenChatKit la herramienta anterior.

- **Primer Problema : Ordenes Docker**

Cuando se quieren ejecutar las órdenes que usan docker, dan un error de que no la reconoce, por eso lo primero que se hizo fue instalar la orden docker y docker-compose y otros utils de estos para que no hubiera más problemas.

```
sudo apt install docker.io  
sudo apt install docker-compose-plugin  
sudo apt install docker-compose
```

Tras usar estos comandos, deberían de haberse podido utilizar las órdenes de build y run para ejecutar los contenedores de Open Assistant. Pero no fue así, estas órdenes siguieron dando errores, se buscaron cuantiosas soluciones ya que sin estas no se podía montar el docker.

Tras buscar en unos foros una de las soluciones que se daban fue comprobar que la versión descargada de docker era la correcta y compatible con el proyecto, esto se hizo con la siguiente orden:

```
docker -v  
docker-compose --version
```

Como al comprobar se vio que eran correctas, otra posible solución era ver si el servicio docker estaba ejecutándose, como se está haciendo en wsl2 y no en Linux el comando `systemctl status` no se usa hay que buscar el correspondiente de wsl2 que es:

```
sudo service docker start
```

Aquí se vio que el servicio no estaba activo ni se podía iniciar, por lo tanto había que buscar otra solución, esta era actualizar docker con la orden:

```
sudo apt upgrade docker
```

Pero nada, seguía sin funcionar, se probó a montar la imagen con el dockerfile que había en la raíz del proyecto pero tampoco daba resultados satisfactorios.

```
docker build -t open-assistant ./Dockerfile
```

Ya las últimas opciones usando la línea de comandos fueron instalar más dependencias de docker y mostrar la información del comando docker para ver qué ocurría.

Finalmente había más gente con este problema y estaban abriendo temas referidos a este en el repositorio oficial de Open Assistant. La solución final fue instalar docker desktop y utilizando las siguientes órdenes para crear el contenedor de Open Assistant

```
export DOCKER_HOST=unix:///var/run/docker.sock
docker build -t open-assistant -f docker/oasst-postgres/Dockerfile .
```

Con esto ya se pueden hacer las órdenes docker del apartado 3 para que todo comience a funcionar y aquí se soluciona el primer problema tenido.

- **Segundo Problema: Almacenamiento Docker**

Este problema apareció cuando se ejecutaron las órdenes build y tras hacer varias pruebas ya con el docker, el contenedor ocupó tantos GB que el disco C del ordenador no tenía más espacio, estando prácticamente vacío.

El problema fue que se probó a mover el docker Desktop desde las aplicaciones de Windows pero no dejaba, la única opción era desinstalarlo borrar todo y volverlo a hacer en el disco D, después de volverlo a hacer y ejecutar el .exe para instalarlo y seleccionar la ruta de instalación no se podía, buscando se encontró en internet que si o si se tenía que hacer en el disco C, esto supuso repetir lo anterior para volver a construir la imagen y el contenedor para sobrevivir con los pocos megas restantes.

Se encontró una posible solución que era mover todos los datos y ficheros de la aplicación al disco duro D, donde había espacio de sobra pero para ello había que hacerlo de una forma compleja para que no se borrara nada y la aplicación localizara esas imágenes aun estando en otro disco.

```
wsl --shutdown
wsl --export docker-desktop-data docker-desktop-data.tar
wsl --unregister docker-desktop-data
wsl --import docker-desktop-data D:\docker-new-repo\ docker-desktop-
data.tar --version 2
```

Así se solucionó el segundo problema pudiendo probar incluso el main para hablar con el chat de prueba.

- **Tercer Problema: Instalación de Flash_attn**

Cuando se ejecuta la última orden del apartado tres, aparece un error que nos indica que no se puede instalar el paquete de flash_attn, este error nos indica que no detecta torch, y esta dependencia lo necesita para poder instalarse.

Lo primero es comprobar si el torch está instalado, en este caso si estaba, por lo tanto se va a ver si la versión es compatible con la que requiere, flash_attn.

```
pip3 torch --version
```

Como la versión es compatible se probó a instalar directamente flash_attn:

```
pip3 install flash_attn==0.2.8
```

Esto no dio resultado y se probó a desinstalar y volver a instalar torch por si algunos de sus archivos estaban corruptos y era lo que no dejaba que flash_attn lo detectase.

También se apreció que dentro del proyecto aparecía un archivo indicando la versión de Python 3.10.8 mientras que en el WSL2 la versión que había era 3.10.6. Para ello se descarga desde la página web de Python esta versión, extraemos el paquete y lo instalamos. El problema es se tienen dos Python instalados y sigue utilizando la versión que no es correcta para ello se deben hacer los siguientes comandos:

```
mv Python-3.10.8 /home/claudia/  
cp /home/claudia/Python-3.10.8/python.exe /home/claudia/.local/bin
```

Esto no solucionó nada, dentro del environment, se muestran los módulos y efectivamente torch no estaba instalado, se instala a mano con una versión >= que la que se requería en el modelo, una vez hecho esto, se sigue intentado volver a instalar los requerimientos del modelo pero sigue dando errores, ahora de nvcc (errores de dependencias con cuda y gcc), para ello se comprueban con los siguientes comandos que son compatibles las versiones que hay instaladas.

```
gcc --version  
cuda --version  
nvcc --version  
docker info
```

Como eso en principio está correcto, en la página oficial del módulo que da error, se recomienda que se instale torch a partir de un contenedor de nvidia. Por lo tanto se descarga el nuevo contenedor para guardarlo en el docker desktop, lo que tarda bastante rato ya que pesa mucho y mientras tanto se puede eliminar el torch que se tenía instalado ya que no lo vamos a utilizar.

Cuando este el contenedor listo lo ejecutamos para activarlo y que comience a funcionar, esto proporciona otro error:

```
The NVIDIA Driver was not detected. GPU functionality will not be
```

```
available.  
2023-06-17 19:13:46 Use the NVIDIA Container Toolkit to start  
this container with GPU support; see  
2023-06-17 19:13:46 https://docs.nvidia.com/datacenter/cloud-native/ .
```

Tras la documentación del entorno de nvidia, torch. cuda, etcétera. Se necesitan instalar los nvidia tool kits, se siguen los pasos de la instalación.

```
$ sudo apt-get update \  
&& sudo apt-get install -y nvidia-container-toolkit-base
```

Esto debe incluir la CLI de NVIDIA Container Toolkit (nvidia-ctk) y la versión se puede confirmar ejecutando:

```
$ nvidia-ctk --version
```

Hay que generar una especificación CDI que haga referencia a todos los dispositivos, se utiliza el siguiente comando:

```
$ sudo nvidia-ctk cdi generate --output=/etc/cdi/nvidia.yaml
```

Suponiendo que la especificación se haya generado, ejecutar un contenedor con acceso a todas las GPU de NVIDIA requeriría el siguiente comando:

```
$ podman run --rm --device nvidia.com/gpu=all ubuntu nvidia-smi -L
```

Docker-CE en Ubuntu se puede configurar usando el script oficial de docker:

```
$ curl https://get.docker.com | sh \  
&& sudo service --now enable docker
```

Se configura el kit de herramientas de contenedores de nvidia, el repositorio de paquetes y la clave GPG:

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \  
&& curl -fsSL https://nvidia.github.io/libnvidia-  
container/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-  
container-toolkit-keyring.gpg \  
&& curl -s -L https://nvidia.github.io/libnvidia-  
container/$distribution/libnvidia-container.list | \  
sudo tee /etc/apt/sources.list.d/nvidia-container.list
```



```
sed 's#deb https://#deb [signed-  
by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg]  
https://#g' | \  
sudo tee /etc/apt/sources.list.d/nvidia-container-  
toolkit.list
```

Se instalan las dependencias después de actualizar la lista de paquetes.

```
$ sudo apt-get update  
$ sudo apt-get install -y nvidia-container-toolkit
```

Se configura el demonio de docker para que reconozca el container de nvidia y se reinicia:

```
$ sudo nvidia-ctk runtime configure --runtime=docker  
$ sudo service restart docker
```

Se prueba una configuración funcional ejecutando el contenedor de CUDA, pero esto nos da error y no aparece lo que se esperaba

```
$ sudo docker run --rm --runtime=nvidia --gpus all  
nvidia/cuda:11.6.2-base-ubuntu20.04 nvidia-smi
```

7.2.5. Conclusión

Tras tener cuantiosos errores, aunque bastantes se solucionaron este último llevo bastante tiempo intentando solucionarlo y cada solución que aparecía cada vez era más tediosa y conllevaba mucho tiempo.

Después de intentar muchas soluciones para este último y ninguna funcionar, lo más posible es que WSL2 tenga bastantes problemas para estos proyectos que suelen funcionar mejor para Linux o MacOS. Tras buscar en el GitHub oficial de Open Assistant había mucha más gente con el mismo problema y nadie conseguía solucionarlo ya que la solución era con este contenedor para el torch, aunque los moderadores y administradores indicaban que era muy complicado obtener el entorno idóneo utilizando esto ya que era muy complejo y había cuantiosas dependencias que podían hacer que no funcionase. Por esto se optó por hacer el proyecto con OpenAI.

8. SPRINT 2

ENTRENAMIENTO DE LAS RESPUESTAS

8.1. Introducción

En este sprint el objetivo era tener conectado el Chatbot al localhost y tener una página web donde comunicarse con él de forma visual, además de entrenarle para que conteste de una determinada forma a unas preguntas concretas.

8.2. Implementación

- **Primer Objetivo**

El primer objetivo es conectar el bot con el localhost y poder conversar con él desde allí. Para ello hay que instalar un framework web llamado Flask que es el que se utiliza en Python dentro del environment.

```
pip install flask
```

En el archivo Python del Chatbot hay que importarlo para poder usarlo, además de hacerle más cambios para que detecte el html.

```
from flask import Flask, render_template, request
```

Se crea el archivo html, con un contenido básico para el funcionamiento. Ahora ejecutando el programa se debería poder hablar con el Chatbot en el localhost.

```
python app.py
```

Esto da un problema al ir al localhost dice que no hay conexión, este mensaje indica que el servidor no está respondiendo al puerto solicitado y como se puede observar al ejecutar el programa no aparece ningún mensaje en la terminal de que flash se esté ejecutando, este error es debido a que el código de Python le falta algo, hay que realizar un par de cambios en el código, incluye dos decoradores.

- `@app.route('/')` para vincular la función `home()` a la ruta raíz del servidor.
- `@app.route('/chat', methods=['POST'])` para vincular la función `chat()` a la ruta `'/chat'` y limitarla a los métodos POST.

Así se solucionó este problema pero ahora aparece otro que hace que no se pueda escribir al bot, que nos indica que no encuentra el archivo html, para ello se tiene que crear una carpeta para meter dentro de ella el html quedando la estructura del proyecto de esta forma:

```
- chatbot.py
- templates
  - index.html
```

Además hay que cambiar la función home del archivo Python:

```
@app.route('/')
def home():
    return render_template('templates/index.html').
```

Gracias a esto ya se le puede escribir una pregunta al bot pero al pulsar en enviar no envía la pregunta, este problema se sabe que al ser al pulsar el botón tiene que ver con el JavaScript que hay en el html, por eso hay que modificar la parte del script un poco para que funcione. Tras realizar un par de cambios ya funcionaba correctamente.

Ya se ha cumplido el primer objetivo pero el localhost era muy básico y para nada visual por eso se propuso como siguiente objetivo crear un css a parte donde especificar el estilo del html un poco para que fuese más claro y así ya tener un pequeño ejemplo de funcionamiento simple que durante los próximos sprints se irá haciendo más complejo.

Así se crea el archivo styles.css que servirá para estilizar la interfaz de usuario del Chatbot.

Para que el html lo reconozca y lo use hay que añadirle la siguiente línea de código en el head.

```
<link rel="stylesheet" type="text/css" href="styles.css">
```

Aun así esto dio problemas, no reconoció el archivo css, dando el error 404 indicando que el archivo css no se encuentra en el servidor, como flask es el servidor utilizado hay que agregar la siguiente línea de código al archivo Python:

```
app = Flask(__name__, static_folder='static')
```

Esto le indica a Flask que los archivos estáticos se encuentran en una carpeta llamada static. Quedando el esquema del proyecto de esta forma ahora. También hay

que cambiar la dirección de la línea de código del html donde se indica dónde está el CSS.

```
- chatbot.py
- static
  - styles.css
- templates
  - index.html
```

```
<link rel="stylesheet" type="text/css" href="{ url_for('static',
filename='styles.css') }}">
```

Ya funcionan los estilos, ahora surge otro tipo de problema cuando se le escribe algo al Chatbot este te responde dos veces lo mismo por ejemplo.

```
Human: hola AI:Hola, ¿en qué puedo ayudarte?
AI: Hola, ¿en qué puedo ayudarte?
```

Una posible solución era mover la línea donde se añade la respuesta a la conversación justo antes de enviar la conversación al html. Pero esto no funcionó.

Otra causa que pudo ocasionar esto eran los valores del modelo, se cambiaron pero no soluciono el problema, así que se restablecieron a los originales

Otra prueba que se hizo fue cambiar la variable conversation, al principio fue una cadena de caracteres y se cambió por un vector de caracteres, cambiando el código del archivo Python para que este funcionara guardándose en los distintos espacios del vector. Pero esto ocasionó más errores.

Ya que los errores indicaban que el parámetro prompt que se le pasaba al modelo no tenía un formato válido porque conversation era un vector, para ello una idea para solucionarlo era formatear la variable conversation. Y utilizar esta nueva variable como el valor del parámetro prompt.

```
formatted_conversation = "\n".join([f"{turn['role']}:
{turn['text']}" for turn in conversation])
```

Cambiar la variable conversación por un vector solo causó errores por eso se volvió a dejar como un string. Tras corregir el código Python se pensó que el error sería del html y se añadió código en el div de conversación.

```
<div id="conversation">
```

```
{% for turn in conversation %}
    {% if turn.role == 'Human' %}
        <p class="human">{{ turn.text }}</p>
    {% elif turn.role == 'AI' %}
        <p class="ai">{{ turn.text }}</p>
    {% endif %}
{% endfor %}
</div>
```

Esto tampoco soluciono nada y dio más errores ya que no reconoció bien el proyecto estas líneas de código. Así se estuvieron toqueteando ambos archivos hasta encontrar otra posible solución y se remodeló casi toda la estructura para solucionar el problema pero no resultó y se restableció todo como estaba al inicio.

También se probó creando varios if en el archivo Python para que si reconocía la palabra AI sin un salto de línea anterior lo eliminase hasta el siguiente AI, para que así solo apareciese uno y en una línea distinta a la pregunta del humano, pero tampoco resultó.

Otra solución fue crear dos variables `conversation_input` y `conversation_output`, para formatear la de entrada para ponerle el formato requerido por el OpenAI, luego agregarle la pregunta actual, unirla a una sola cadena de texto, realizar la solicitud al modelo, agregar la pregunta y la respuesta actual a la conversación de salida y enviarla al html. Esto tampoco lo soluciono, daba un resultado que no era el esperado de pregunta salto de línea y respuesta.

Finalmente mirando el código poco a poco del html y Python y ejecutando línea a línea con el modo debug y viendo lo que se guardaba en las variables se cambió un poco el código dejando solo una conversación cambiando la forma en la que se introducen las preguntas y respuestas escribiendo en ellas como se ponen los saltos de línea en html para que éste los reconozca y enviando solo al html la conversación ya que se enviaba también la respuesta, así funciono correctamente con un formato pregunta dos saltos de línea y respuesta.

● Segundo Objetivo

Ahora que se tiene el Chatbot funcionando en el localhost y con la conversación apareciendo correctamente, se quiere entrenar al bot para que se pueda utilizar para ayudar a personas con dependencia.

Para ello hay que crear un conjunto de entrenamiento, utilizar datos anotados para crear un conjunto en el formato requerido por la plataforma de entrenamiento, para ello se usa un formato de entrada y salida de preguntas y respuestas donde cada pregunta se empareja con su respuesta correspondiente.

Para realizar esto se crea un archivo JSON, el cual contendrá una lista de ejemplos de entrenamiento, pregunta-respuesta. Cada uno tiene que tener dos propiedades, la propiedad question contendrá la pregunta formulada por el usuario y la propiedad answer contendrá la respuesta generada por el bot.

Para que el archivo Python pueda utilizar las preguntas y respuestas entrenadas de un archivo JSON, hay que añadir código al inicio del archivo Python para leer el contenido del JSON. Se puede importar la biblioteca JSON para así leer el archivo JSON:

```
import json

# Cargar el archivo JSON
with open('entrenamiento.json', 'r') as file:
    data = json.load(file)

# Obtener las preguntas y respuestas
preguntas = data['preguntas']
respuestas = data['respuestas']
```

Además hay que modificar la función chat para buscar la respuesta correspondiente en el archivo JSON en lugar de llamar a la API de OpenAI. Se puede comparar la pregunta del usuario con las preguntas almacenadas en preguntas y obtener la correspondiente desde respuestas.

```
# Buscar la pregunta en el archivo JSON y obtener la respuesta
correspondiente
answer = None
for i, pregunta in enumerate(preguntas):
    if pregunta == question:
        answer = respuestas[i]
        break

if answer is None:
    answer = "Lo siento, no tengo una respuesta para esa
pregunta."
```

De esta forma el Chatbot solo responderá si se le pregunta una de las preguntas que tiene entrenadas, con el resto dirá que no tiene respuesta y no se quiere eso. Si se desea que el Chatbot responda normalmente cuando se hace una pregunta que no está en el archivo JSON, se puede combinar la búsqueda en el archivo con la llamada a la API de OpenAI, para ello hay que volver a modificar la función chat.

```
# Buscar la pregunta en el archivo JSON y obtener la respuesta correspondiente
answer = None
for i, pregunta in enumerate(preguntas):
    if pregunta == question:
        answer = respuestas[i]
        break

if answer is None:
    # Si la pregunta no está en el archivo JSON, llamar a la API de OpenAI
```

De esta manera el Chatbot primero buscará la pregunta en el archivo JSON y si no la encuentra proporcionará la respuesta correspondiente. Si la pregunta no está en el archivo, el Chatbot llamara a la API de OpenAI para obtener una respuesta utilizando el código que ya había implementado.

Esto le dará flexibilidad al Chatbot para responder a una amplia variedad de preguntas.

Ahora aparece un error que indica que no se encuentra el archivo de entrenamiento es decir el JSON, para ello hay que meterlo en otra carpeta en la raíz del proyecto quedando el árbol de la estructura de esta forma:

```
- MiProyecto/
  - chatbot.py
  - data/
    - entrenamiento.json
  - templates/
    - index.html
  - static/
    - styles.css
```

Y cambiado la siguiente línea en Python:

```
with open('/ruta/completa/al/directorio/data/entrenamiento.json',
'r') as file:
    data = json.load(file)
```

Aun cambiando todo esto daba error pero se solucionó cambiando esto en el Python:

```
# Obtener la ruta absoluta del archivo
```

```
file_path = os.path.abspath('data/entrenamiento.json')

# Abrir el archivo utilizando la ruta absoluta
with open(file_path, 'r') as file:
```

Tras solucionar esto aparece otro error que indicaba que la clave preguntas no se encuentra en el archivo JSON cargado, la estructura del JSON no coincide con la esperada por el código, para ello tenemos que modificar un par de líneas de la obtención de las preguntas y respuestas del archivo.

```
# Obtener las preguntas y respuestas
examples = data['examples']
preguntas = [example['question'] for example in examples]
respuestas = [example['answer'] for example in examples]
```

Esto hace que el error desaparezca pero hace que solo responda cuando la pregunta está exactamente igual, y pensándolo bien una persona puede formular una pregunta de muchas formas distintas mejorando así la experiencia del usuario si esto se puede implementar, aquí se origina el siguiente objetivo de la aplicación.

Este código calcula la similitud de coseno entre la pregunta del usuario y las preguntas almacenadas.

```
question_vector = vectorizer.transform([question_preprocesada])
similarities = cosine_similarity(question_vector, vectorizer)[0]
max_similarity = max(similarities)
index = similarities.tolist().index(max_similarity)
```

Esto da un error que indica que la biblioteca NLTK no está instalada en el entorno, hay que instalarla con el siguiente comando en el entorno:

```
pip install nltk
```

Después de la instalación aparece otro error similar que indica que la biblioteca scikit-learn no está instalada en el entorno. Para solucionarlo se instala con el siguiente comando:

```
pip install scikit-learn
```

A continuación se produce otro error que indica que no se encontró el recurso stopwords de NLTK, esto son palabras comunes que se consideran irrelevantes en el procesamiento del lenguaje natural y generalmente se eliminan del texto.

Para solucionar este problema, se debe descargar el recurso utilizando el NLTK downloader. Se ejecuta el siguiente código en el entorno.

```
import nltk
nltk.download('stopwords')
```

Esto no funcionó, no se pudo descargar el recurso debido a un problema de conexión, una solución alternativa es descargar manualmente el archivo de stopwords y guardarlo en la ubicación adecuada. Se descarga stopwords.zip, se extrae el contenido y se mueve a /home/claudia/nltk_data/corpora/

Se ejecuta nuevamente el código tras realizar estos pasos, esto soluciona el problema anterior pero produce uno nuevo ya que falta el recurso punkt que se utiliza para el tokenizado de oraciones. Para solucionarlo descargamos el recurso en el entorno.

```
python -m nltk.downloader punkt
```

Vuelve a dar un error al descargar como el recurso anterior, para ello se va a hacer manualmente, descargando el archivo .zip, guardándolo en una ubicación accesible para el sistema, descomprimiéndolo para obtener la carpeta punkt y moviéndola a la ruta /home/claudia/nltk_data/tokenizers/.

Se vuelve a ejecutar el código y aparece un error que el objeto vectorizer no tiene atributo transformer indica, para solucionarlo se verifica si se ha importado el vectorizador adecuado en el archivo Python.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

También hay que reemplazar la línea donde se inicializa el vectorizer:

```
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(preguntas_preprocesadas)
transformer = TfidfTransformer()
X_train_transformed = transformer.fit_transform(X_train)
```

En este código, primero se crea una instancia de TfidfVectorizer y se ajusta el vectorizador a los datos del JSON. Luego, se crea una instancia de TfidfTransformer y se ajusta el transformer a los datos transformados por el vectorizer.

En la función chat se cambia la línea donde se calcula la question_vector por:

```
question_vector = vectorizer.transform([question_preprocesada])  
question_vector_transformed = transformer.transform(question_vector)
```

Todo esto soluciona el anterior problema pero genera otro que indica que se está pasando un objeto `TfidfVectorizer` como argumento a una función que espera un número o una cadena como argumento `float`.

El problema radica en la línea `similarities = cosine_similarity(question_vector, vectorizer)[0]`. La función `cosine_similarity` espera matrices numéricas como entrada, pero se está pasando `vectorizer`, que es una instancia de `TfidfVectorizer`.

Para solucionar este problema, se debe pasar la matriz de características `X_train_transformed` en lugar del objeto `vectorizer` a la función `cosine_similarity`.

```
similarities = cosine_similarity(question_vector_transformed,  
X_train_transformed)[0]
```

Con esto ya se solucionan todos los problemas, se cumplen todos los objetivos, funcionando así correctamente el Chatbot.

8.3. Conclusión

Como ya se tiene una primera versión de ejemplo funcionando, en los próximos sprints se completará el archivo JSON con más preguntas y respuestas especializadas en el tema de ayuda a personas en situación de dependencia, además de hacer más visual la interfaz de usuario de la página web ya que es muy básica.

Además de añadir nuevas implementaciones como por ejemplo un registro de usuarios para que se identifiquen al comenzar la conversación con el chat además de crear nuevos botones para que haya rutas más fáciles y más accesibles para ciertas preguntas o peticiones más comunes.

9. SPRINT 3

CREACIÓN DE LA BASE DE DATOS

9.1. Introducción

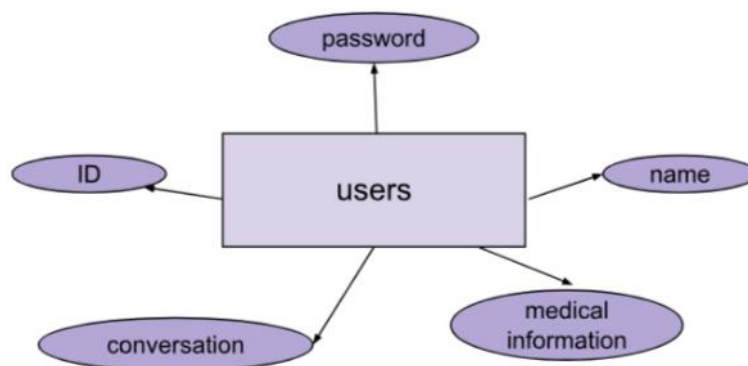
En este sprint el objetivo era tener una base de datos de usuarios, en el que las personas que quieran utilizar el Chatbot tendrá que crear una cuenta y cada vez que quieran volver a entrar tengan que iniciar sesión para que sus datos anteriores aparezcan.

9.2. Implementación

- **Primer Objetivo:**

El primer objetivo fue crear una pequeña base de datos para que los usuarios pudieran identificarse para poder conversar con el Chatbot. Las primeras pruebas fueron crear dentro del programa principal un diccionario de usuarios, pero esto a la larga si se quieren implementar más cosas iba a dar muchos problemas, ya que no es nada flexible.

Como ya se tenía una idea de que el usuario se iba a identificar y crear sesión si no estaba previamente creado se optó por crear una base de datos llamada users para guardar los datos de los usuarios, esto se hizo con sqlite3, para ello se creó un nuevo archivo Python, llamado database donde se irán guardando todas las funciones y todo el código referente a las tablas de la base de datos.

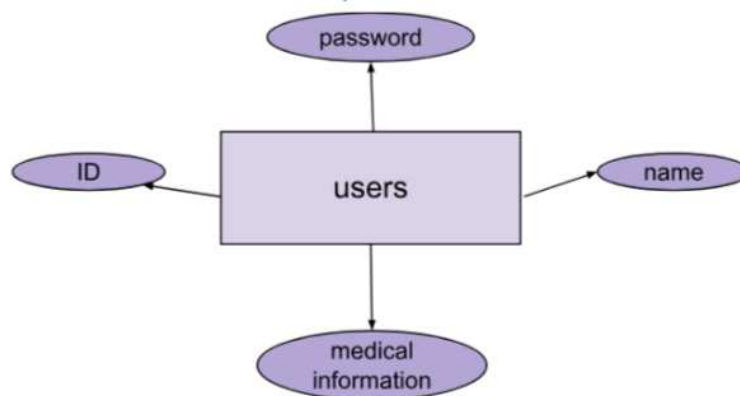


Para empezar se tienen que importar sqlite en el código, crear una función para establecer la conexión con la base de datos, crear una función para que cree la tabla users, la cual estará compuesta por un id, el nombre, contraseña, un registro de historial médico y otro de las conversaciones. También se tiene que definir una función

para insertar un nuevo usuario en la base de datos. Otra para obtener la información de un usuario en base a su nombre y contraseña y se necesitará una función para actualizar la conversación de los usuarios.

Todo esto estará en la database.py pero para iniciar la conexión y poder usarlo en el Chatbot se tienen que crear la conexión, la tabla etc desde el Chatbot.py llamando a las funciones creadas anteriormente.

Hasta aquí ya se tenía una tabla base de usuarios pero al continuar creando las funciones login y create hubo que remodelar la base de datos, conversaciones dejó de ser un atributo de la tabla y su función fue eliminada, también salió la necesidad de crear otra función que devolviese el usuario a partir del ID.



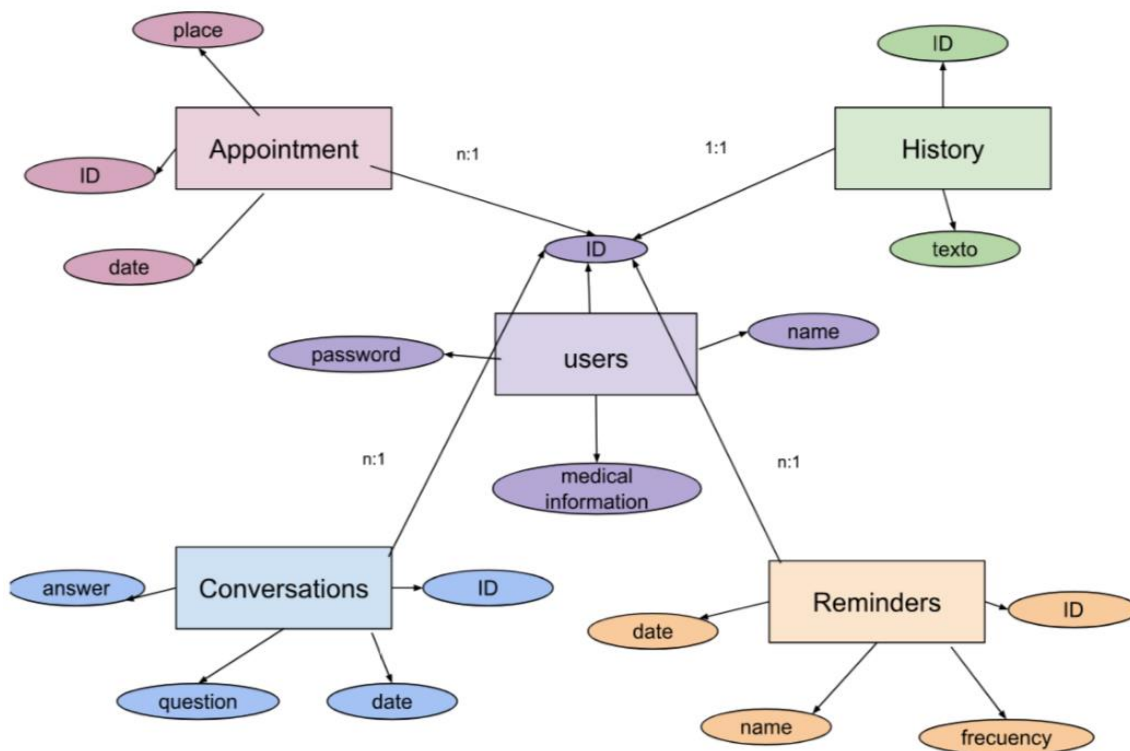
Conforme avanzaba el código del Chatbot ocurrió un error lógico a la hora de crear usuarios ya que estos solo detectaba que ya estaban creados si coincidía el nombre y contraseña y no se quiere eso, sino que el nombre de usuario sea único para todos. Así se tuvo que cambiar el create table poniendo nombre como UNIQUE, además de crear una nueva función que se encargaba de comprobar si el usuario con ese nombre ya existía.

Por último aunque ya funcionaba a la perfección el create y login se pensó en cómo dejar las tablas para un futuro para que fuesen más flexibles y se pudieran implementar más funcionalidades.

La tabla users se quedó como estaba tras esta última modificación, pero se crearon más tablas:

- **Appointment:** Es la tabla citas, guarda el ID, la fecha el lugar y el ID del usuario a quien corresponde
- **History:** Esta tabla guarda el historial médico de un usuario, por lo tanto está formada por text donde se escribirá la nueva información médica, ID del historial, y el ID del usuario al que corresponde.

- **Reminders:** Esta tabla trata los recordatorios de los pacientes, si se tiene que tomar medicinas, o hacer x ejercicio, por eso guarda un ID del recordatorio, la fecha de cuando tienes que ser, la frecuencia por si es un medicamento y se lo tiene que tomar cada 8 horas por ejemplo, el nombre de lo que tiene que hacer y el ID del usuario al que pertenece.
- **Conversations:** En ella se guardaran la pregunta, la respuesta y la fecha de la conversación de cuando se hizo, además de su ID y de la ID del usuario a la que corresponde.



● Segundo Objetivo:

Tras crear la primera tabla de usuarios ya se puede crear las funcionalidades de crear cuenta e iniciar sesión. Además de que para que todo sea más visual y no esté todo apelmazado en un mismo html, este objetivo también conlleva que el inicio de sesión se haga en uno aparte y si es correcto se redirija al del chat al igual que cuando se cree una cuenta, pero si se produce un error es decir que no se han puesto bien los credenciales o que el usuario que se intenta crear ya existe aparezca un mensaje de error en el nuevo html y no redirija.

Lo primero de todo es crear el html, en el que se tendrán dos contenedores uno para el inicio de sesión y otro para crear cuenta cada uno con sus botones, y que pida el usuario y contraseña los cuales se le enviarán al chat.py donde se tratarán para comprobar los credenciales y redirigir al Chatbot, es un html muy sencillo pero al

modularizar el código es más fácil encontrar el error y solucionarlo o incluir nuevas modificaciones.

A continuación se tiene que crear la función en el chat.py al principio todo se hacía en una misma, ya que primero se obtenían los datos enviados por el formulario es decir el username y el password, después se verificaban las credenciales del usuario en la base de datos, conectando con ella y ejecutando un select de la tabla usuarios, con un condicional se comprueba si las credenciales son correctas, si lo son redirige a la conversación con el chat obteniendo el ID del usuario de la base de datos, y si las credenciales no son correctas significa para el programa que el usuario no existe por lo tanto lo crea en la otra parte de este condición, guarda su nuevo id, cierra la conexión y redirige al chat de este nuevo usuario.

Esto no tiene mucho sentido ya que si un usuario se equivoca con sus credenciales automáticamente le va a crear uno nuevo en vez de saltar el error. Por esto se dividió la función en dos, la función login y la función create.

Ahora la función login funciona de la siguiente forma, obtiene el username y el password de su formulario, crea la conexión con la base de datos, obtiene el usuario con estos credenciales de la base de datos, si existe significa que los credenciales están bien ambos por tanto redirige al chat de ese usuario, si no coinciden significa que los credenciales no son correctos por lo tanto crea un mensaje de error que se le pasa al html, para que se actualice y salte el mensaje en rojo.

El problema ahora es la función create ya que el login ya funciona a la perfección, el create también coge los credenciales de su formulario, conecta con la base de datos, obtiene el usuario y si existe uno con ese nombre y contraseña devuelve un error y si no lo encuentra crea un nuevo usuario y redirige al nuevo chat con este. El problema es que solo identifica que un usuario ya esté en la base de datos si coincide su nombre y contraseña y esto no puede ser así ya que no sería nada eficiente ni cómodo para los usuarios por lo tanto hay que modificarlo.

Esta modificación primero hay que hacerla desde la tabla de usuarios indicando que name es una variable única, además de que se creara una nueva función en el archivo database.py la cual comprueba que el nombre de usuario esté disponible, haciendo un select del nombre en la tabla usuarios si lo encuentra devuelve el nombre y si no un none, esta función se usará en el create para comprobar si el usuario con ese nombre ya existe.

Ahora el nuevo create será de esta forma, coge el usuario y contraseña del formulario, conecta con la base de datos, obtiene el usuario a través del nombre, si lo encuentra redirige al inicio ya que el usuario ya está creado con ese nombre dando un error de que se introduzca otro nombre de usuario, si no lo encuentra crea un nuevo usuario, insertándolo en la base de datos y redirigiendo al chat de este.

Además se creó un archivo css para el nuevo html del inicio de sesión ya que sin este no era para nada visual ni claro para ser utilizado por personas mayores aun así tiene que seguir arreglando y mejorando.

9.3. Conclusión

Teniendo ya en punto la base de datos y los usuarios, el inicio de sesión y la creación de usuarios, como ya están creadas las distintas tablas que se hicieron con el pensamiento de añadir nuevas funcionalidades el próximo sprint será de esto.

Además de que se sigue teniendo en cuenta que hay que añadir gran cantidad de preguntas y respuestas al JSON, estas se han ido desarrollando también en este sprint aunque no se ven reflejadas en el código ya que se ha preferido seguir investigando y obtener más, así que posiblemente en el siguiente ya habrá un mayor número de preguntas en el entrenamiento.

10. SPRINT 4

CREACIÓN DE LAS FUNCIONALIDADES

10.1. Introducción

En este sprint ya se tiene creada la base de datos con todas las tablas necesarias para crear nuevas funcionalidades por ello en esta entrega el objetivo va a ser crear esas nuevas funcionalidades para la página web y el Chatbot para que los usuarios puedan usar para más cosas el Chatbot, no sólo para consultas.

10.2. Implementación

- **Primer Objetivo:**

El primer objetivo en este caso fue crear un menú en el lateral derecho de la web, este es del tipo hamburguesa, es decir las tres típicas rayitas horizontales que si se pulsa sale el menú desplegable con las distintas opciones.

La implementación de este fue sencilla se hizo en el html principal del Chatbot, y no se tocó casi nada del css para hacerlo en un próximo sprint que será más centrado en la interfaz de usuario, la accesibilidad y seguridad.

En este menú aparecen los siguientes campos, añadir cita, añadir entrada al historial médico, añadir recordatorio y contacto, que se ha puesto por si en un futuro se quieren indicar datos del responsable pero actualmente no tiene ninguna funcionalidad ni hace ninguna actividad. Esta es la descripción de las opciones implementadas y cómo funcionan:

- **Añadir cita:**

Añade una nueva entrada a la tabla appointments, guardando el ID del usuario actual que está usando el Chatbot. Además, al pulsar el botón llevará a los usuarios a un formulario que tendrán que rellenar con la fecha, en el que aparecerá un calendario y seleccionarán el día y la hora además de escribir el lugar de la consulta. Estos datos se guardan en la tabla.

Su funcionamiento: para lograr esto se ha tenido que crear una nueva función en la base de datos que lo que hace es hacer un INSERT con sql en la tabla appointment comprueba que los datos insertados corresponden con los campos que hay que rellenar y si la inserción es exitosa nos devuelve un mensaje de confirmación.

A parte en el código Python del Chatbot hay que crear dos funciones más, una que nos redirigirá al nuevo html correspondiente al formulario de las citas, y la otra crea la conexión con la base de datos, consigue el id del usuario actual y recoge los datos del formulario enviando todos estos a la función de la base de datos para que la inserte en la tabla.

En el nuevo html se crea el formulario correspondiente a las citas, que pide la fecha y el lugar en este caso, y realizará la solicitud Ajax al servidor flask para guardar los datos.

- **Añadir recordatorio:**

Añade una nueva entrada a la tabla reminders, guardando el ID del usuario actual que está usando el Chatbot. Además, al pulsar el botón, llevará a los usuarios a un formulario que tendrán que rellenar con la fecha, en el que aparecerá un calendario y seleccionarán el día y la hora además de escribir el nombre que le quieran poner al recordatorio y la frecuencia pensando en que si es tomarse un medicamento sea de la siguiente forma: nombre, Paracetamol; fecha, 04/07/2023 16:53 y frecuencia, cada 8 horas. Estos datos se guardan en la tabla.

Su funcionamiento: para lograr esto se ha tenido que crear una nueva función en la base de datos que lo que hace es hacer un INSERT con sql en la tabla reminders comprueba que los datos insertados corresponden con los campos que hay que rellenar y si la inserción es exitosa nos devuelve un mensaje de confirmación.

Aparte en el código Python del Chatbot hay que crear dos funciones más, una que nos redirigirá al nuevo html correspondiente al formulario de los recordatorios, y la otra crea la conexión con la base de datos, consigue el id del usuario actual y recoge los datos del formulario enviando todos estos a la función de la base de datos para que la inserte en la tabla.

En el nuevo html se crea el formulario correspondiente a los reminders, que pide la fecha, el nombre y la frecuencia en este caso, y realizará la solicitud Ajax al servidor flask para guardar los datos.

- **Añadir entrada al historial médico:**

Añade una nueva entrada a la tabla history, guardando el ID del usuario actual que está usando el Chatbot. Además, al pulsar el botón llevará a los usuarios a un formulario que tendrán que rellenar un cuadro de texto sobre la nueva actualización sobre su historial médico. Estos datos se guardan en la tabla.

Su funcionamiento: para lograr esto se ha tenido que crear una nueva función en la base de datos que lo que hace es hacer un INSERT con sql en la tabla history, comprueba que los datos insertados corresponden con los campos que hay que rellenar y si la inserción es exitosa nos devuelve un mensaje de confirmación.

A parte en el código python del Chatbot hay que crear dos funciones más, una que nos redirigirá al nuevo html correspondiente al formulario del historial, y la otra crea la conexión con la base de datos, consigue el id del usuario actual y recoge los datos del formulario enviando todos estos a la función de la base de datos para que la inserte en la tabla.

En el nuevo html se crea el formulario correspondiente al historial, que pide el texto correspondiente a la nueva entrada en este caso, y realizará la solicitud Ajax al servidor flask para guardar los datos.

● Segundo Objetivo:

Ahora que los usuarios ya pueden ingresar sus próximas citas, algunos recordatorios incluso actualizar su historial médico, lo que se quiere conseguir ahora es cubrir la necesidad de estos de poder visualizar pues las citas que tienen pendientes o recordatorios que todavía están vigentes o ver su historial médico al completo.

Ya que estas funcionalidades se van a usar más frecuentemente se ha decidido que en vez de estar en el menú lateral de la página web se sitúan los tres botones debajo del panel de escritura de las preguntas para el Chatbot y que al pulsar cada uno de estos botones se muestre por la conversación la información correspondiente al botón que se ha pulsado, si por ejemplo se ha pulsado “mis citas” pues que se muestre las que tienen una fecha posterior a la actual al igual con los recordatorios y cuando se pulse “mi historial médico” que muestre al usuario toda la información guardada en su historial médico por el chat.

Por lo tanto se han creado estas tres nuevas funcionalidades, Mis Citas, Mis Recordatorios y Mi Historial Médico. A continuación se va a explicar cada una de ellas y cómo se ha conseguido su funcionamiento. Actualmente la forma en la que se muestran es muy sencilla, en el siguiente sprint se profundizará más en ello para que sea más visual y vistoso.

- Mis Citas:

Al pulsar este botón se mostrarán las citas en el chat del bot, da igual que ya se haya hablado con él aparecerá todo debajo de la conversación incluso si ya ha mostrado más información correspondiente a los otros botones. Mostrará todas las citas correspondientes al usuario que está usando el Chatbot y cuya fecha sea mayor que la actual, es decir las citas anteriores a la fecha actual no aparecerán, sí estarán guardadas en la base de datos para tener la información por si hiciera falta en algún caso pero no aparecen al pulsar el botón.

Su funcionamiento es un poco más complejo, lo principal es crear la función de la base de datos que saca los datos que se requieren de la tabla appointment. Esta

lo primero que hace es coger la hora local y guardarla en una variable, guardándola en el mismo formato que está la hora de la tabla citas para poder compararlas más tarde, después sacará los datos de la fecha y el lugar de la tabla appointment donde el usuario es el actual que está usando el Chatbot y la fecha sea mayor a la actual es decir sean citas próximas no antiguas, esto devuelve en un map de citas.

En el fichero Python del Chatbot también se deberá incluir una función, en la que se conecte con la base de datos, se guarde el ID de la conversación se llame a la función que se había creado anteriormente para obtener las citas, con esto se recorrerá el map de citas creando el string para cada una de ellas con la información que se quiere que se muestre por pantalla este string es el que se le envía al html.

En el html se debe crear el nuevo botón para “Mis Citas” además de crear nuevo código JS en la parte de script del html para recibir el string que se ha creado con anterioridad en el archivo Python y mandarlo como respuesta al pulsar el botón para que así se muestre en la conversación con el Chatbot.

- **Mis Recordatorios:**

Al pulsar se mostrarán los recordatorios en el chat del bot, da igual que ya se haya hablado con él aparecerá todo debajo de la conversación incluso si ya ha mostrado más información correspondiente a los otros botones. Mostrará todos los recordatorios correspondientes al usuario que está usando el Chatbot y cuya fecha sea mayor que la actual, es decir los recordatorios anteriores a la fecha actual no aparecerán, sí estarán guardados en la base de datos para tener la información por si hiciera falta en algún caso pero no aparecen al pulsar el botón.

Su funcionamiento es un poco más complejo, lo principal es crear la función de la base de datos que saca los datos que se requieren de la tabla reminders. Esta lo primero que hace es coger la hora local y guardarla en una variable, guardándola en el mismo formato que está la hora de la tabla reminders para poder compararlas más tarde, después sacará los datos de la fecha, el nombre y la frecuencia de la tabla reminders donde el usuario es el actual que está usando el Chatbot y la fecha sea mayor a la actual es decir sean recordatorios próximos no antiguos, esto devuelve en un map de recordatorios.

En el fichero Python del Chatbot también se deberá incluir una función, en la que se conecte con la base de datos, se guarde el ID de la conversación se llame a la función que se había creado anteriormente para obtener los recordatorios, con esto se recorrerá el map de los recordatorios creando el string para cada uno de ellos con la información que se quiere que se muestre por pantalla este string es el que se le envía al html.

En el html se debe crear el nuevo botón para “Mis Recordatorios” además de crear nuevo código JS en la parte de script del html para recibir el string que se ha creado con anterioridad en el archivo Python y mandarlo como respuesta al pulsar el botón para que así se muestra en la conversación con el Chatbot.

- **Mis Historial Médico:**

Al pulsarlo se mostrará el historial médico en el chat del bot, da igual que ya se haya hablado con él aparecerá todo debajo de la conversación incluso si ya ha mostrado más información correspondiente a los otros botones. Mostrará todo el historial médico correspondiente al usuario que está usando el Chatbot.

Su funcionamiento es un poco más complejo, lo principal es crear la función de la base de datos que saca los datos que se requieren de la tabla history. En esta lo que se hace es seleccionar las entradas de texto de la tabla history en las que el ID del usuario es igual al ID del usuario actual que está usando el Chatbot, así devolverá un map del historial médico de dicho usuario

En el fichero Python del Chatbot también se deberá incluir una función, en la que se conecte con la base de datos, se guarde el ID de la conversación se llame a la función que se había creado anteriormente para obtener el historial médico, con esto se recorrerá el map del historial creando el string para cada una de las entradas con la información que se quiere que se muestre por pantalla este string es el que se le envía al html.

En el html se debe crear el nuevo botón para “Mi Historial Médico” además de crear nuevo código JS en la parte de script del html para recibir el string que se ha creado con anterioridad en el archivo Python y mandarlo como respuesta al pulsar el botón para que así se muestre en la conversación con el Chatbot.

● **Pequeños Arreglos:**

Además de todo esto también se ha hecho una pequeña mejor al obtener el id del usuario para acceder a su conversación con el Chatbot cuando se iniciaba sesión, ya que en la url se mostraba anteriormente el id de usuario y ahora se ha modificado para que sea más seguro y no se muestre.

10.3. Conclusión

En este sprint se han conseguido todos los objetivos que se esperaban, teniendo prácticamente toda la parte del código Python y de la base de datos hecha, pudiendo así enfocar los próximos sprints en la parte visual del html, para que la página web sea más sencilla y accesible para todo tipo de usuarios.

Todo esto se hará a través de los archivos css previamente creados en sprints anteriores, pero añadiendo mucha más complejidad para que no sea una página web tan básica.

Además, habrá que comenzar a poner todas las preguntas y respuestas en el entrenamiento del JSON para ir modificando y especializando el Chatbot a que si sea para un uso orientado a personas en situaciones de dependencia.

11. SPRINT 5

ACCESIBILIDAD, INTERFAZ DE USUARIO

11.1. Introducción

El objetivo de este sprint es finalizar el proceso de implementación del Chatbot y dejarlo todo listo, es decir en esta semana se va a trabajar sobre los html y css para crear una verdadera interfaz de usuario, tanto en la página de inicio como en la landpage de la conversación como en los formularios.

Además de llenar el fichero con las preguntas creadas a partir de una exhaustiva búsqueda en medios médicos y de ayuda a personas con dependencia para que el Chatbot sea más específico y de respuestas con mejores resultados.

11.2. Implementación

- **Primer Objetivo: Mejora de la IU de la página de inicio**

En el código proporcionado, se han realizado los siguientes cambios y mejoras en los archivos HTML y CSS del formulario de inicio de sesión y de creación de cuenta:

Cambios en el archivo HTML:

1. Se ha definido la estructura básica de la página utilizando las etiquetas ``, `` y ``.
2. Se ha agregado el encabezado `` que contiene la sección de texto de bienvenida y un efecto de onda de fondo.
3. Se ha creado un contenedor principal `

- 4. Se ha agregado un contenedor `

- 5. Se ha agregado un contenedor `

- 6. Se han añadido etiquetas `` para los campos de entrada de nombre de usuario y contraseña, así como para los mensajes de error.

7. Se han definido campos de entrada `<input>` para capturar el nombre de usuario y contraseña, con los atributos `id` y `name` para identificar los campos en el servidor.
8. Se ha agregado un botón de envío `<input type="submit">` en cada formulario para enviar los datos al servidor.
9. Se han utilizado bloques de código `{% if ... %} ... {% endif %}` para mostrar mensajes de error personalizados en caso de que existan.

Cambios en el archivo CSS:

1. Se ha definido un estilo para el encabezado `<header>` que incluye un fondo degradado y una imagen de fondo.
2. Se ha establecido el estilo de los textos dentro del encabezado, como el tamaño de fuente y la alineación.
3. Se ha creado una clase `.wave` para aplicar un efecto de onda en la parte inferior del encabezado.
4. Se ha definido un estilo para el `<body>` que incluye el color de fondo y la fuente.
5. Se ha utilizado la clase `.container` para alinear los formularios en el centro de la página.
6. Se ha creado una clase `.inicio-container` para dar estilo al contenedor del formulario de inicio de sesión.
7. Se ha creado una clase `.crear-container` para dar estilo al contenedor del formulario de creación de cuenta.
8. Se han definido estilos para los títulos `<h2>` de cada formulario, como el margen superior y la alineación.
9. Se ha establecido el estilo de las etiquetas `<label>` y los campos de entrada `<input>`, como el ancho, el margen y el borde.
10. Se ha aplicado un estilo al botón de envío `<input type="submit">`, incluyendo el color de fondo y el cambio de color al pasar el cursor por encima.
11. Se ha definido un estilo para los mensajes de error `.error-message`, incluyendo el color y el margen inferior.

Estos cambios se han realizado con el propósito de mejorar la apariencia visual y la usabilidad de los formularios de inicio de sesión y creación de cuenta en el

Chatbot. Se ha aplicado un diseño atractivo con un encabezado llamativo y se han personalizado los estilos de los elementos para crear una experiencia de usuario más agradable. Los mensajes de error personalizados ayudan a los usuarios a identificar y solucionar problemas al interactuar con los formularios.

- **Segundo Objetivo: Mejora de la IU de la página de principal**

Cambios en el archivo HTML:

1. Se ha añadido un encabezado `<header>` que contiene el título del Chatbot y un efecto de onda de fondo.
2. Se ha creado un contenedor principal `<div class="container">` que alberga el contenido principal del Chatbot.
3. Se ha añadido un contenedor `<div id="conversation" class="conversation">` para mostrar la conversación entre el usuario y el Chatbot.
4. Se ha creado un formulario `<form id="chat-form" class="bottom-bar">` que permite al usuario enviar preguntas al Chatbot.
5. Se ha añadido un campo de entrada de texto `<input type="text" class="question-input" id="question" placeholder="Escribe tu pregunta" />` donde el usuario puede escribir sus preguntas.
6. Se ha agregado un botón de envío `<button type="submit" class="submit-button">Enviar</button>` que permite al usuario enviar la pregunta al Chatbot.
7. Se han creado tres botones adicionales `<button>` dentro de un contenedor `<div class="button-container">`. Estos botones se utilizan para realizar acciones específicas, como ver citas, recordatorios e historial médico.
8. Se ha añadido un botón de menú `<div class="menu-toggle">` que permite al usuario desplegar o contraer el menú.
9. Se ha creado un menú desplegable `<div class="menu">` que contiene una lista de opciones de navegación.
10. Se han añadido enlaces `<a>` dentro de los elementos de lista `` para cada opción de navegación.
11. Se ha agregado un bloque de derechos de autor `<div class="copyright">` que muestra el nombre del autor y los derechos de autor.

Cambios en el archivo CSS:

1. Se ha definido el estilo del encabezado `<header>`, incluyendo el fondo degradado, la imagen de fondo y el estilo del texto.
2. Se ha establecido el estilo global del cuerpo `<body>`, incluyendo la fuente, el color de fondo y el margen.
3. Se ha creado una clase `.conversation` para dar estilo al contenedor de la conversación, incluyendo el tamaño, el fondo y el borde.
4. Se han definido estilos para los mensajes dentro de la conversación, como el margen inferior, el relleno y el color de fondo para diferenciar entre los mensajes del usuario y del Chatbot.
5. Se ha creado una clase `.container.bottom-bar` para dar estilo al contenedor inferior del Chatbot, que contiene el formulario de entrada de texto y los botones adicionales.
6. Se han definido estilos para el formulario de entrada de texto y el botón de envío, como el tamaño, el margen, el color de fondo y el color del texto.
7. Se ha creado una clase `.button-container` para dar estilo al contenedor de botones adicionales, estableciendo el flujo flex y el margen entre los botones.
8. Se han definido estilos para los botones adicionales, como el tamaño, el relleno, el color de fondo, el color del texto y el margen derecho.
9. Se han definido estilos para el botón de menú `.menu-toggle` y sus elementos internos (`hamburger`, `hamburger:before`, `hamburger:after`) para crear un efecto de hamburguesa que se transforma al hacer clic.
10. Se ha creado una clase `.menú` para dar estilo al menú desplegable, estableciendo su posición, ancho, altura y fondo.
11. Se han definido estilos para la lista de opciones de navegación `.menu ul`, incluyendo el tipo de lista, el relleno y los márgenes.
12. Se ha establecido el estilo de los elementos de lista `.menu ul li`, como el margen inferior, el color del texto y el tamaño de fuente.
13. Se han definido estilos para los enlaces dentro de los elementos de lista `.menu ul li a`, estableciendo el color y la decoración del texto al pasar el cursor sobre ellos.

Estos cambios se han realizado con el objetivo de mejorar la apariencia visual y la interacción del Chatbot. Se ha aplicado un diseño atractivo con un encabezado llamativo y se han personalizado los estilos de los elementos para crear una experiencia de usuario agradable. Los botones adicionales ofrecen funcionalidades específicas para acceder a información relevante, como citas, recordatorios e historial médico. Al hacer clic en el botón de menú, se despliega un menú que muestra las opciones de navegación disponibles. Los estilos aplicados al menú y a los elementos de lista ayudan a que el menú sea visualmente atractivo y fácil de usar.

- **Tercer Objetivo: UI de los formularios y preguntas del entrenamiento**

En este último objetivo se han arreglado los formularios de las citas, de los recordatorios y del historial médico aunque ya estaban creados correctamente sus respectivos html no tenían estilos por lo tanto se veían muy mal y aunque fuera un simple formulario a las personas mayores o con alguna dependencia les podría costar usarlos correctamente por eso se ha creado un estilo general para todos los formularios aunque es muy simple cualquier persona que lo vea sabrá que campos tiene que rellenar, como debe hacer y cómo guardar los datos sin ningún tipo de problema ya que es muy sencillo siguiendo la gama de colores del resto de estilos usados en la página principal y en la de inicio de sesión y creación de cuenta.

Además tras todas estas semanas poco a poco se ha ido recolectando información sobre temas médicos interesantes para poder hacer mas preguntas y repuestas en el JSON de entrenamiento que finalmente han sido implementados en este último Sprint.

11.3. Conclusión

Aunque ya esté terminada esta versión “beta” del Chatbot, se le pueden implementar muchas más funcionalidades y mejorarlo en muchos más aspectos esto es solo la base de una aplicación que podría ser muy grande y ayudar a mucha gente ya que todavía no hay nada por el estilo tan específico para este ámbito.

12. INTERFAZ DE USUARIO

A continuación se muestran algunas de las imágenes más importantes de la interfaz de usuario tales como la página de inicio, la página principal del chat, el menú desplegable y las opciones para añadir citas, recordatorios o entradas al historial médico.

Página de inicio



Iniciar Sesión

Nombre de Usuario:

Contraseña:

Iniciar sesión

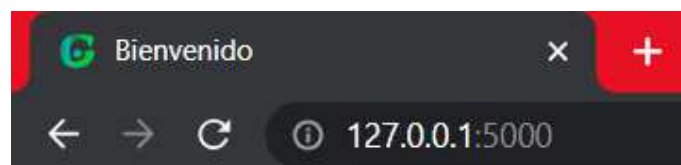
Crear Cuenta

Nombre de Usuario:

Contraseña:

Crear cuenta

Detalles del título de la página de inicio



Cuadros de inicio de sesión (izquierda) y de crear cuenta (derecha)

Iniciar Sesión

Nombre de Usuario:

Contraseña:

Iniciar sesión

Crear Cuenta

Nombre de Usuario:

Contraseña:

Crear cuenta

Introducción de credenciales

Iniciar Sesión	Crear Cuenta
Nombre de Usuario: <input type="text" value="Claudia"/>	Nombre de Usuario: <input type="text" value="Alejandra"/>
Contraseña: <input type="password" value="...."/>	Contraseña: <input type="password" value="....."/>
<input type="button" value="Iniciar sesión"/>	<input type="button" value="Crear cuenta"/>

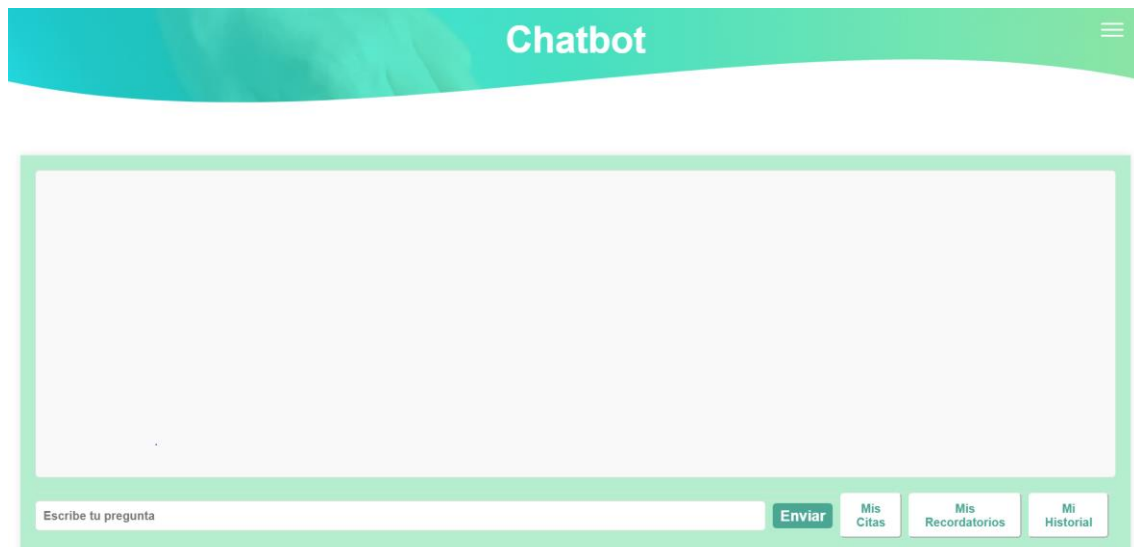
Introducción errónea de los credenciales

Iniciar Sesión	Crear Cuenta
Nombre de Usuario: <input type="text"/>	Nombre de Usuario: <input type="text"/>
Contraseña: <input type="password"/>	Contraseña: <input type="password"/>
El usuario no existe. Comprueba los credenciales.	El usuario ya existe. Por favor, elige otro nombre de usuario.
<input type="button" value="Iniciar sesión"/>	<input type="button" value="Crear cuenta"/>

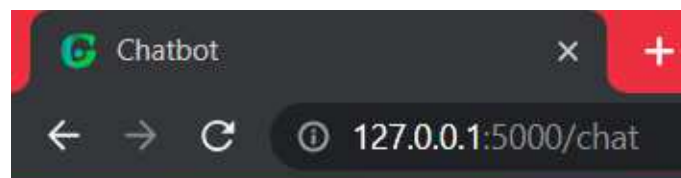
Falta de un campo

Iniciar Sesión	Crear Cuenta
Nombre de Usuario: <input type="text" value="Claudia"/>	Nombre de Usuario: <input type="text"/>
Contraseña: <input type="password"/>	Contraseña: <input type="password"/>
<div>! Completa este campo</div> <input type="button" value="Iniciar sesión"/>	<div>! Completa este campo</div> <input type="button" value="Crear cuenta"/>

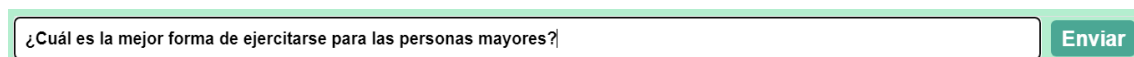
Página principal



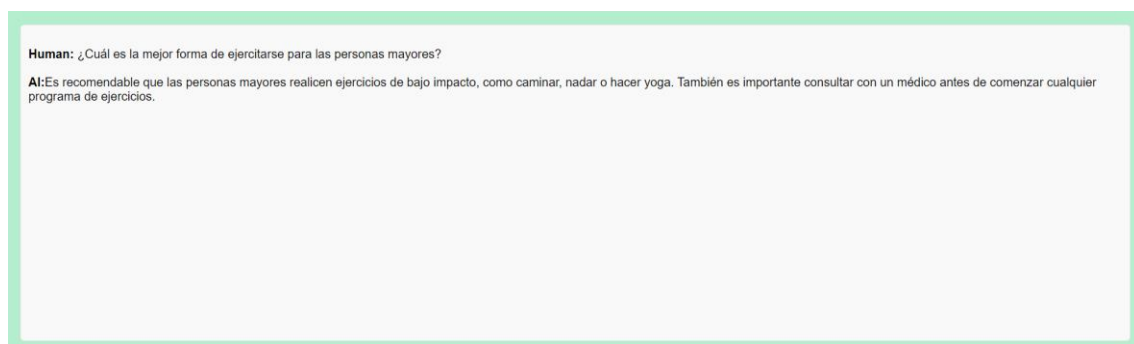
Detalles del título de la página principal



Escritura de preguntas

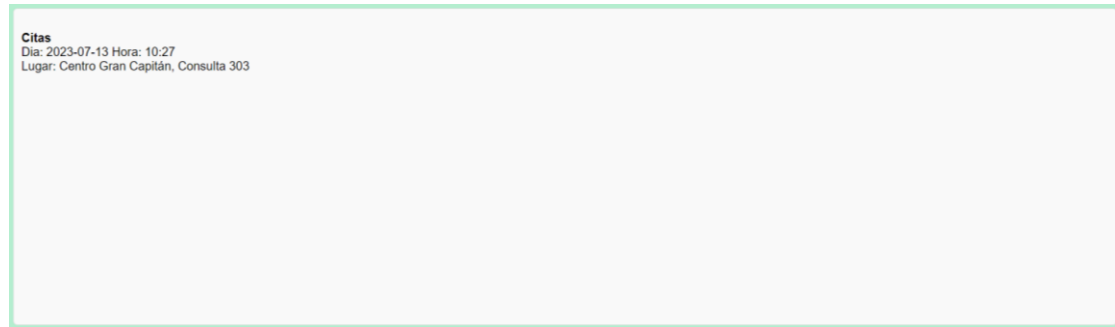
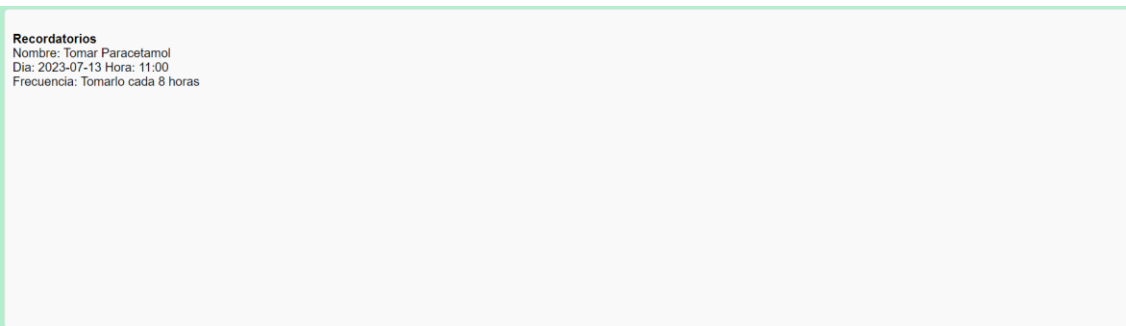
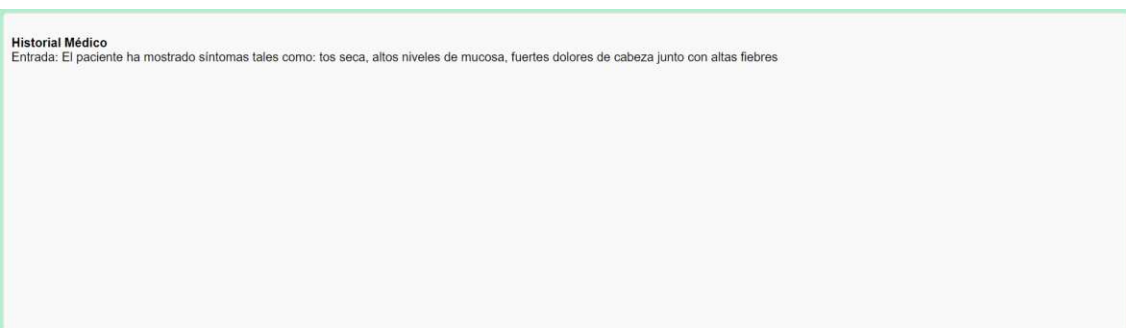


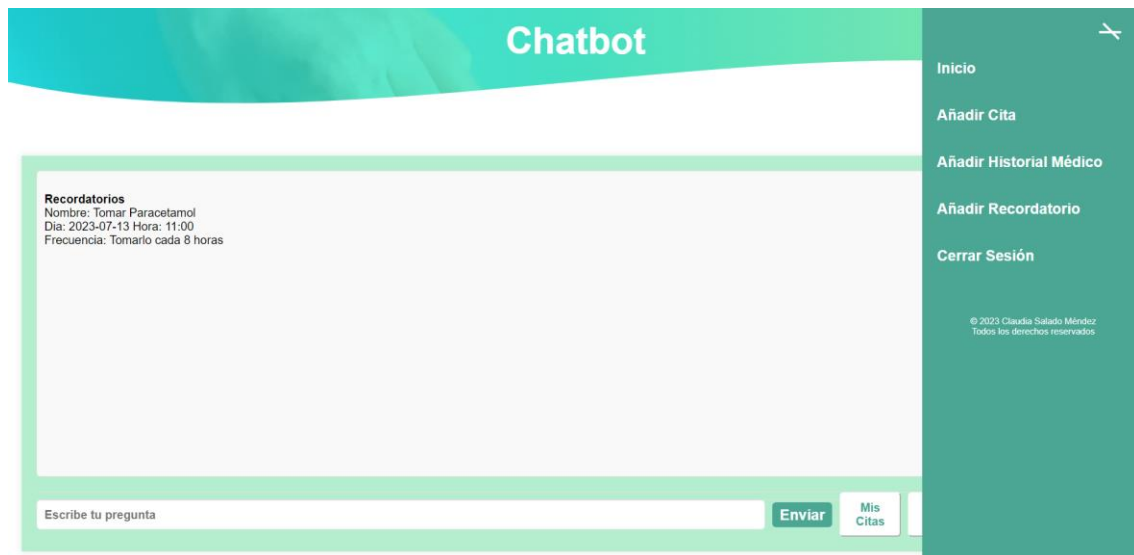
Conversación



Human: ¿Cuál es la mejor forma de ejercitarse para las personas mayores?

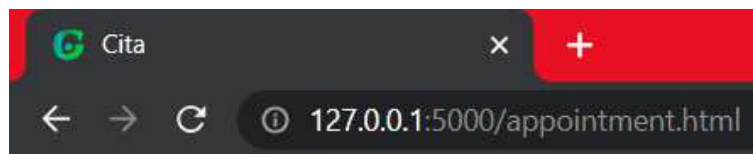
AI: Es recomendable que las personas mayores realicen ejercicios de bajo impacto, como caminar, nadar o hacer yoga. También es importante consultar con un médico antes de comenzar cualquier programa de ejercicios.

Botones para mostrar contenidos:**Muestra de las citas al pulsar el botón “Mis Citas”****Muestra de los recordatorios al pulsar el botón “Mis Recordatorios”****Muestra del historial médico al pulsar el botón “Mi Historial Médico”**

Menú desplegable**Página principal con el menú desplegado**

Cuadro para añadir cita

A screenshot of a web form titled "Añade una Cita Médica" (Add a Medical Appointment). The form is centered on a dark gray background. It consists of a light green rectangular box containing two input fields: "Lugar:" (Location) and "Fecha:" (Date). The "Fecha:" field includes a date picker icon. Below the input fields is a green button labeled "Enviar" (Send).

Detalle del título para añadir cita**Cuadro para añadir cita completado**

A screenshot of the "Añadir una Cita Médica" form, now filled out. The title "Añade una Cita Médica" is at the top. The "Lugar:" field contains the text "Centro Gran Capitán, Consulta 303". The "Fecha:" field shows the date and time "13/07/2023 10:27". A green button labeled "Enviar" is at the bottom.

Desplegable fecha en el cuadro para añadir cita

Lugar:

Centro Gran Capitán, Consulta 303

Fecha:

13/07/2023 10:27

julio de 2023 ▾ ↑ ↓

L	M	X	J	V	S	D
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

Borrar Hoy

10	27
11	28
12	29
13	30
14	31
15	32
16	33

Aviso de campo sin completar en el cuadro de añadir cita

Añade una Cita Médica

Lugar:

Fecha:

13/07/2023 02:00

Completa este campo

Enviar

Cuadro para añadir recordatorio

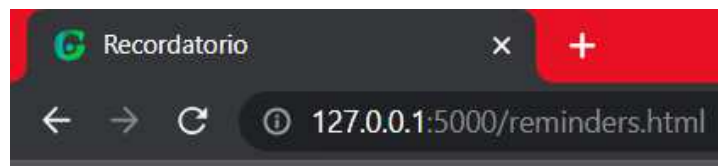
Añade un Recordatorio

Nombre:

Fecha:

Frecuencia:

Enviar

Detalle del título para añadir recordatorio**Cuadro para añadir recordatorio completado**

Añade un Recordatorio

Nombre:

Fecha:

Frecuencia:

Enviar

Desplegable fecha en el cuadro para añadir recordatorio

Añade un Recordatorio

Nombre:

Tomar Paracetamol

Fecha:

13/07/2023 11:00

julio de 2023 ↑ ↓

L	M	X	J	V	S	D	11	00
26	27	28	29	30	1	2	12	
3	4	5	6	7	8	9	13	01
10	11	12	13	14	15	16	14	02
17	18	19	20	21	22	23	15	03
24	25	26	27	28	29	30	16	04
31	1	2	3	4	5	6	17	05

Borrar Hoy

Aviso de campo sin completar en el cuadro para añadir recordatorio

Añade un Recordatorio

Nombre:

Tomar Paracetamol

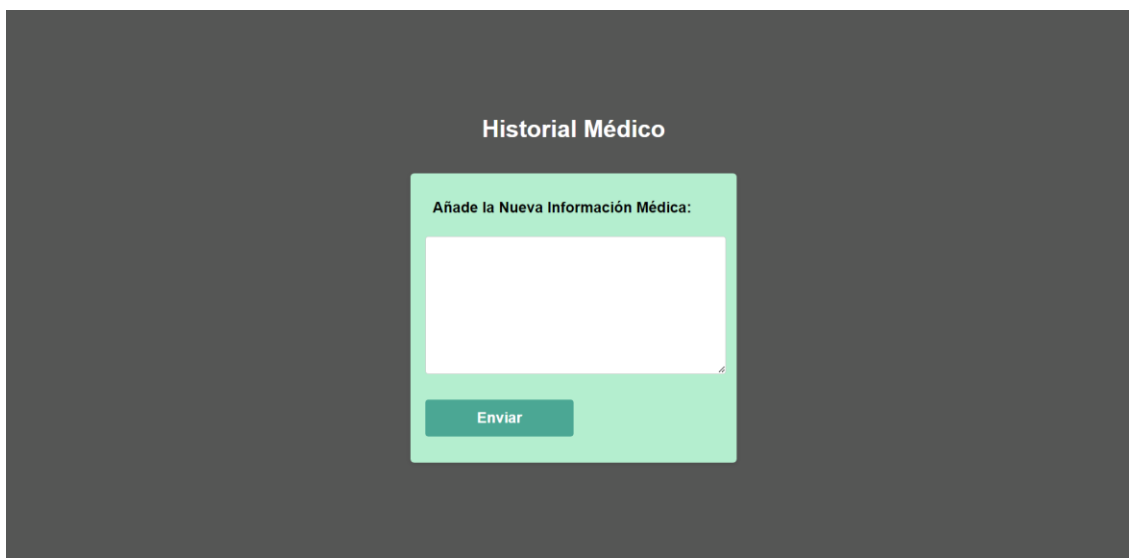
Fecha:

dd/mm/aaaa --:--

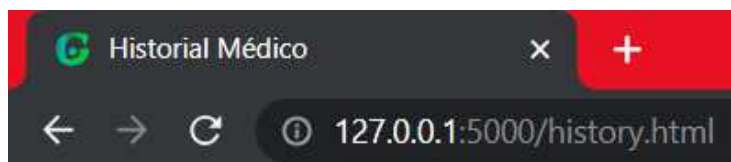
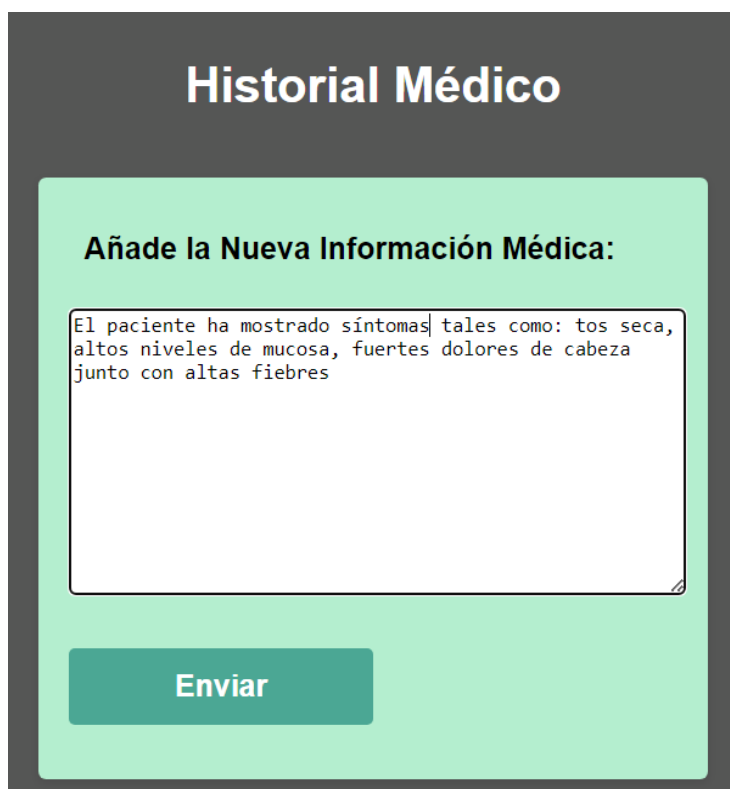
Frecuencia  Completa este campo

Tomarlo cada 8 horas

Enviar

Cuadro para añadir entrada al historial médico

The screenshot shows a web interface with a dark gray background. At the top center, the text "Historial Médico" is displayed in white. Below it is a light green rectangular box. Inside this box, the text "Añade la Nueva Información Médica:" is at the top. Below the text is a large, empty white rectangular text input field. At the bottom of the green box is a teal button with the word "Enviar" in white text.

Detalle del título para añadir entrada al historial médico**Cuadro para añadir entrada al historial médico completado**

The screenshot shows the same web interface as before, but the text input field is now filled with the text: "El paciente ha mostrado síntomas tales como: tos seca, altos niveles de mucosa, fuertes dolores de cabeza junto con altas fiebres". The teal "Enviar" button remains at the bottom of the green box.

Aviso de campo sin completar en el cuadro para añadir entrada al historial

Historial Médico

Añade la Nueva Información Médica:

!

Completa este campo

Enviar

13. CONCLUSIÓN FINAL

En este Trabajo de Fin de Grado se ha desarrollado un Chatbot personalizado que ofrece una experiencia interactiva y útil para los usuarios. A lo largo de este proyecto se han abordado diversos aspectos relacionados con la implementación de un Chatbot desde el diseño de la interfaz de usuario hasta la integración con tecnologías de procesamiento de lenguaje natural.

En cuanto a la implementación del Chatbot, se ha utilizado un enfoque basado en reglas y aprendizaje automático para permitir respuestas inteligentes y contextualizadas. Se ha utilizado el lenguaje de programación Python y el framework Flask para crear el backend del Chatbot, mientras que el frontend se ha desarrollado utilizando HTML, CSS y JavaScript.

Durante el proceso de desarrollo se han seguido metodologías ágiles lo que ha permitido una iteración continua y una mejora constante del sistema. Se han aplicado técnicas de prueba exhaustivas para garantizar el correcto funcionamiento y la fiabilidad del Chatbot.

Además, al diseñar la interfaz de usuario, se han tenido en cuenta aspectos de usabilidad y accesibilidad con el objetivo de brindar una experiencia amigable e intuitiva para los usuarios.

En resumen, este Trabajo de Fin de Grado ha permitido la creación de un Chatbot funcional y eficiente, capaz de interactuar con los usuarios y proporcionar respuestas relevantes y útiles. El desarrollo de este proyecto ha supuesto un aprendizaje profundo en el campo de la inteligencia artificial y el procesamiento del lenguaje natural, así como una oportunidad para aplicar los conocimientos adquiridos a lo largo de la carrera.

Este trabajo sienta las bases para futuras mejoras y ampliaciones del Chatbot, tanto en términos de funcionalidad como de escalabilidad. Se espera que este proyecto sea una herramienta útil y práctica en diversos contextos, como la atención al cliente, la asistencia médica y la interacción con sistemas de información. En última instancia, el Chatbot desarrollado en este Trabajo de Fin de Grado demuestra el potencial y las posibilidades de la inteligencia artificial aplicada a la comunicación y la interacción humano-máquina.

En lo que respecta a mí como autora de este trabajo, a pesar de los problemas iniciales con la implementación de la herramienta y una pérdida de tiempo sustancial, he aprendido muchísimo de muchos campos de la informática. Además, al haber tenido que buscar información relacionada con la ayuda a personas dependientes, he adquirido una formación sobre recomendaciones sanitarias y respuesta en situaciones de emergencia que me va a servir de ahora en adelante.

La elaboración de este Trabajo Fin de Grado en estos más de cuatro meses ha sido un tanto complicada al tener que compaginarlo con las prácticas de empresa que estoy realizando actualmente pero también he aprendido a organizarme, a establecer horarios, a hacer una programación e ir cumpliendo plazos y también a hacer algún que otro sacrificio. Realmente estoy muy orgullosa de lo que he conseguido porque he mejorado en muchos aspectos importantes tanto para el mundo laboral (que ya estoy experimentando) como a nivel personal. Espero poder transmitir todo esto a los lectores de esta Memoria.

14. VÍAS FUTURAS

Existen varias mejoras y funcionalidades que se podrían implementar en este proyecto en el futuro para ampliar la utilidad del Chatbot y mejorar la experiencia del usuario. A continuación, se presentan algunas ideas y posibilidades:

1. **Mejora del procesamiento del lenguaje natural:** Se puede trabajar en la mejora de los algoritmos de procesamiento del lenguaje natural utilizados por el Chatbot, con el objetivo de comprender y responder de manera más precisa y contextualizada las preguntas y solicitudes de los usuarios.
2. **Integración con sistemas externos:** Se puede explorar la integración del Chatbot con sistemas externos, como bases de datos de conocimiento, APIs de servicios web o sistemas de información específicos. Esto permitiría al Chatbot acceder a información actualizada y proporcionar respuestas más completas y precisas.
3. **Expansión de la base de conocimientos:** Se puede trabajar en la ampliación de la base de conocimientos del Chatbot, incorporando nuevas categorías de información y respuestas. Esto implicaría la recopilación y curación de datos relevantes para mejorar la capacidad de respuesta del Chatbot en diferentes dominios.
4. **Personalización y adaptabilidad:** Se puede implementar funcionalidades que permitan al Chatbot aprender y adaptarse a los usuarios individuales. Esto podría incluir la capacidad de recordar preferencias, historial de conversaciones y adaptar las respuestas en función de las necesidades y características de cada usuario.
5. **Integración con plataformas de mensajería:** Se puede explorar la integración del Chatbot con plataformas de mensajería populares, como WhatsApp, Facebook Messenger o Slack. Esto permitiría a los usuarios interactuar con el Chatbot a través de sus canales de comunicación preferidos.
6. **Funcionalidades de voz:** Se podría agregar la capacidad de interactuar con el Chatbot a través de comandos de voz. Esto requeriría la implementación de tecnologías de reconocimiento de voz y procesamiento del habla.
7. **Mejoras en la interfaz de usuario:** Se podría trabajar en la mejora y optimización de la interfaz de usuario del Chatbot, con el objetivo de hacerla más atractiva, intuitiva y fácil de usar. Esto incluiría mejoras en el diseño, la navegación y la usabilidad general.

8. Soporte multilingüe: Se podría desarrollar la capacidad de comprensión y respuesta en varios idiomas, lo que permitiría a los usuarios interactuar con el Chatbot en su idioma preferido.

Estas son solo algunas ideas para posibles mejoras y funcionalidades futuras del proyecto del Chatbot. La implementación de estas mejoras dependerá de las necesidades específicas del proyecto y de los recursos disponibles.

15. BIBLIOGRAFÍA

Para la elaboración de este Trabajo Fin de Grado se ha empleado la siguiente bibliografía:

- Relativo a **OpenAssistant**:

<https://open-assistant.io/es>

<https://huggingface.co/OpenAssistant>

<https://stackoverflow.com/questions/40465979/change-docker-native-images-location-on-windows-10-pro>

<https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch?quick-deploy=false>

<https://docs.nvidia.com/datacenter/cloud-native/#containers-and-nvidia-gpus>

<https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html#install-guide>

- Relativo a **OpenAI**:

<https://platform.openai.com/docs/api-reference>

<https://platform.openai.com/docs/introduction>

- Relativo a **ChatterBot**:

<https://chatterbot.readthedocs.io/en/stable/>

<https://github.com/gunthercox/ChatterBot>

- Relativo a **Rasa**:

<https://rasa.com/>

<https://rasa.com/docs/rasa/>

- Relativo a **Dialogflow**:

<https://cloud.google.com/dialogflow?hl=es-419>

- Relativo a **OpenChatKit**:

<https://huggingface.co/spaces/togethercomputer/OpenChatKit>

<https://github.com/facebookresearch/faiss/blob/main/INSTALL.md>

<https://conda.io/projects/conda/en/latest/user-guide/install/index.html>

Prácticamente toda la bibliografía de la parte de implementación ha sido relativa a la instalación de la herramienta a usar y para la búsqueda de problemas en esta. El resto de implementación lo he realizado a partir de los conocimientos adquiridos en las distintas asignaturas que he cursado en el Grado de Informática y no he necesitado apenas información adicional.

En cuanto a la información relacionada con ayuda a dependencia y salud, he recurrido principalmente a las web oficiales del Servicio Andaluz de Salud y del Ministerio de Sanidad y otros canales de difusión de estas administraciones. También he utilizado otras fuentes en mayor medida. Las principales webs consultadas son las siguientes:

- Ministerio de Sanidad (Gobierno de España)
<https://www.mscbs.gob.es/>
- Servicio Andaluz de Salud (Junta de Andalucía)
<https://www.sspa.juntadeandalucia.es/servicioandaluzdesalud/>
- Consejería de Salud y Consumo (Junta de Andalucía)
<https://www.juntadeandalucia.es/organismos/saludyconsumo/>
- Agencia de Servicios Sociales y Dependencia (Junta de Andalucía)
<https://www.juntadeandalucia.es/agenciadeserviciosocialesydependencia/>
- Sociedad Española de Medicina Interna
<https://www.fesemi.org/>
- Medline Plus
<https://medlineplus.gov/>
- Mayo Clinic
<https://www.mayoclinic.org/>
- Centers for Disease Control and Prevention
<https://www.cdc.gov/>
- World Health Organization
<https://www.who.int/>

