TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Sistema conversacional para la alfabetización de la salud mental basado en la interacción humano-robot

Sistema conversacional integrado en Furhat

Autor

Juan Barrionuevo Valenzuela

Directores

Zoraida Callejas Carrión

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 24 de junio de 2024

Sistema conversacional para la alfabetización de la salud mental basado en la interacción humano-robot: Sistema conversacional integrado en Furhat

Juan Barrionuevo Valenzuela

Palabras clave: Chatbot, alfabetización de la salud mental, Furhat, RAG, LLM, chatbot basado en reglas

Resumen

La salud mental es un grave problema que sigue presente en nuestros días y afecta a una gran cantidad de personas. Para dar a conocer estos problemas, desmontar falsas creencias, reducir el estigma y dar a conocer conductas positivas es muy recomendable tener una buena alfabetización de la salud mental. Para ello, los chatbots juegan un rol fundamental ya que las personas suelen ser más abiertas para hablar de problemas con una máquina que con una persona profesional con tiempo limitado.

Para lograr concienciar a la población sobre la alfabetización de la salud mental se ha optado por crear un chatbot en Furhat que haga uso de tres métodos distintos para la creación de agentes conversacionales. Una parte basada en reglas que presenta las ventajas de que las respuestas están controladas, una parte libre para hablar de lo que sea y que el robot actúe como un amigo cercano y una parte entre medias que haga uso de Retrieval Augmented Generation (RAG) que tenga temas para ayudar a estudiantes a no sufrir estrés o ansiedad realizando buenas prácticas de estudios.

Este chatbot puede ser mejorado por profesionales de la salud mental, añadiendo material útil al conocimiento del robot de una manera fácil y accesible.

Conversational system for mental health literacy based on human-robot interaction: Conversational system integrated into Furhat.

Juan Barrionuevo Valenzuela

Keywords: Chatbot, mental health literacy, Furhat, RAG, LLM, rule based chatbot

Abstract

Mental health is a serious issue that remains prevalent today and continues to affect a large portion of the population. To raise awareness of these issues, debunk false beliefs, reduce stigma, and promote positive behaviors, it is highly recommended to have good mental health literacy. To achieve this, chatbots play a fundamental role, as people often prefer to talk about their personal problems with a machine rather than a real specialist with limited time.

To raise awareness and improve knowledge about mental health literacy, a FurhatSDK-based chatbot has been developed. It utilizes three different methods to create conversational agents. One part uses a rule-based chatbot, which ensures controlled responses; another part is a completely open-ended chatbot to discuss anything, making the robot act like a close friend; and finally, a method in between uses Retrieval Augmented Generation (RAG) with knowledge to help students and provide advice to avoid stress or anxiety by using good study techniques.

This chatbot can be further improved by mental health professionals, who can easily and accessibly add useful material to the robot's knowledge base.

Dña. **Zoraida Callejas Carrión**, Profesora del Área de Lenguajes y Sistemas Informáticos del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Sistema conversacional para la alfabetización de la salud mental basado en la interacción humano-robot: Sistema conversacional integrado en Furhat***, ha sido realizado bajo su supervisión por **Juan Barrionuevo Valenzuela**, 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 24 de junio de 2024.

La directora:

Zoraida Callejas Carrión

Agradecimientos

Primero me gustaría agradecer a Zoraida por proponerme este tema tan interesante como TFG del que no tenía mucho conocimiento y por haberme ayudado en todo momento en todo lo que he necesitado, haciendo que me interese más por los chatbots en general. Gracias a su apoyo y consejo todo este trabajo ha sido posible.

También quiero agradecer a todos mis compañeros y compañeras a los que he ido conociendo a lo largo de la carrera. Toda mi estancia en el grado no habría sido la misma sin ellos.

Además agradecer a mi familia y a mi pareja por todo su apoyo incondicional y por estar presentes siempre que lo he necesitado a lo largo de este trabajo, por darme ánimos y por hacerme creer que soy capaz de lograrlo.

Finalmente quiero dar las gracias a todos aquellos amigos y familia que ya no están pero que siempre confiaron en mí para conseguirlo.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

La salud mental es un estado de bienestar mental que ayuda a los individuos en su rutina diaria a afrontar mejor situaciones adversas y a contribuir positivamente a la sociedad [1]. Dentro de este concepto se encuentra la alfabetización de la salud mental, una idea muy importante y que es necesaria que el mundo comprenda para gozar de una buena estabilidad psicológica.

La alfabetización de la salud mental fue definida por Jorm como «los conocimientos y creencias sobre los trastornos mentales que ayudan a su reconocimiento, tratamiento o prevención» [2].

Es importante lograr una buena alfabetización de la salud en general ya que de esta manera es menos probable tener malestar y se reduce la posibilidad de encontrarse en situaciones adversas como serían por ejemplo [3]:

- Mayor tasa de readmisión en hospitales.
- Peor capacidad para recibir auto-ayuda.
- Peor entendimiento de las instrucciones médicas.
- Asistir con menor frecuencia a citas médicas.

Jorm identificó tres dimensiones principales de la alfabetización de la salud mental: reconocimiento, tratamiento y prevención.

Empezando por el reconocimiento, hay estudios que se han encargado de demostrar que no muchas personas tienen la capacidad de reconocer e identificar los diferentes trastornos mentales y tampoco entienden el significado de los distintos términos. Mostrando a personas situaciones de un individuo

con depresión solo el 39 % fue capaz de reconocer el trastorno y un 27 % para la situación de esquizofrenia. De hecho, el 11 % de los participantes en el estudio, para la situación de depresión, creían que tenía un problema físico. Asimismo, el término esquizofrenia suele ser poco entendido entre la población [2].

No solo hay problemas con el reconocimiento de las enfermedades, también es de vital importancia para la sociedad concienciarse acerca de los distintos trastornos para reducir el estigma existente. La mayor parte de ciudadanos europeos poseen una visión incorrecta de los pacientes que sufren trastornos. Esta visión conlleva al miedo y la exclusión, no aceptando a aquellos que sufren problemas en las comunidades puesto que son vistos como problemáticos. Aquellas personas con patologías mentales suelen ser tratadas como inferiores, creyendo que dependen de gente que les cuide y que decidan por ellas. De hecho, el rechazo a la población con problemas mentales es superior a la de aquellos con problemas físicos y suelen ser vistos como la causa de sus propios problemas [4].

Respecto al tratamiento, según la Organización Mundial de la Salud, casi la mitad de los europeos tienen habilidades inadecuadas y problemáticas acerca de su propia salud, salud mental incluida. Muchas de estas situaciones se deben a ser personas de bajos ingresos o migrantes. Este grupo de personas tiene peor acceso y usan menos los servicios para el cuidado y tratamiento [5].

Además, un gran porcentaje tanto en países desarrollados como no desarrollados no reciben el tratamiento necesario a pesar de sufrir trastornos mentales graves (hasta un 50 % en países desarrollados y hasta un 85 % en países sub-desarrollados)[6].

Finalmente la prevención, que, generalmente para problemas menores, tampoco recibe toda la atención por parte de la sociedad. Está demostrado por ejemplo que la actividad física es beneficiosa y sirve para prevenir posibles problemas pero vemos que la gente con una pobre alfabetización de la salud tiende a no hacer suficiente ejercicio [5] [7].

Por tanto, es crucial que la alfabetización en salud mental sea accesible para todos. Esta accesibilidad puede ser lograda por profesionales del ámbito pero su número es limitado (13 trabajadores de la salud mental cada 100.000 personas, incluso menor en países con menos recursos) [8] y lógicamente no tienen capacidad ni para estar siempre disponibles ni para centrarse en otros temas aparte del tratamiento (p.ej. la prevención o la alfabetización). Además es importante saber que no hay suficientes estudios acerca de la alfabetización de la salud mental, es significativo entonces tener medidas que se centren en este problema. Es también un tema muy primordial para los niños ya que una buena alfabetización de la salud mental sirve para descubrir cuanto antes posibles enfermedades mentales, puesto que la

mayoría de problemas de salud mental comienzan antes de los 25 años [9][10].

Es fundamental destacar que no todo el mundo puede permitirse un tratamiento con un psicólogo o no se atreven a asistir a las consultas por el estigma y vergüenza, adicionalmente, es necesario añadir que hay estudios que confirman que las personas son más sinceras con los ordenadores con temas sensibles como el consumo de drogas que con otras personas [11]. Por tanto, el potencial de las TIC es indiscutible ya que pueden estar disponibles en todo momento y presentan la ventaja de que la gente se sincera más con ellas al ser más anónimas [12]. Dentro de las TIC se encuentran los chatbots que, a diferencia de las guías, folletos y materiales multimedia de alfabetización de salud mental, permiten a los usuarios formular sus propias pregunas y hacerlo con sus propias palabras. En este trabajo estamos interesados en particular en la interacción humano-robot, que a través de un sistema conversacional permita presentar una interfaz fácil de usar que permite que la interacción humano-robot sea fluida, haciendo uso de inputs y outputs de distintos tipos como lenguaje o gestos. Esta interfaz permite que usuarios con poca experiencia con la tecnología, lectura o poca salud puedan usarlos eficazmente sin grandes complicaciones. Estos sistemas pueden realizar una comunicación más rica a través de la interacción oral en lenguaje natural y del uso de expresiones faciales, lo que les hace más cercanos a los ususarios, haciéndoles más propensos a ser usados continuamente y provocando que la gente se sienta más cómoda hablando de sus problemas [13].

Los sistemas conversacionales están progresando cada vez más y se está transicionando de un paradigma basado en reglas, en el que se definen previamente todos los posibles diálogos al uso de sistemas basados en inteligencia artificial (IA), los cuales entienden el lenguaje de manera más flexible pero para los que no es posible predecir sus respuestas. Por tanto, al elegir entre ambos paradigmas, se prima o bien el control sobre el comportamiento del chatbot o bien la flexibilidad de la interacción [14][15]. Existe por tanto el reto de combinar estos paradigmas para conseguir respuestas válidas y acertadas además de un entendimiento de la lengua más flexible. En este proyecto se busca la manera de desarrollar un sistema conversacional para la la alfabetización en salud mental que integre estos paradigmas de forma exitosa para distintos tipos de conversaciones de forma transparente al usuario a la vez que se integran estos paradigmas de manera conjunta, sin que el usuario sepa necesariamente que se usan métodos distintos dependiendo de su pregunta.

Por tanto, con el objetivo de conseguir concienciar a la sociedad sobre la salud mental se ha optado por realizar un sistema conversacional desplegado en un robot que aune tres enfoques diferentes de implementación:

- Chatbot basado en reglas: El sistema de diálogo sigue una serie de reglas que determinan el comportamiento del propio sistema, el chatbot

elige la respuesta que mejor se adapta a lo que ha dicho el usuario en base a un flujo de diálogo predefinido. Este tipo de chatbot tiene la ventaja de permitir un control total sobre la interacción, lo cual es de gran relevancia en dominios sensibles como la salud mental, donde es crucial proporcionar información fiable. La desventaja de este enfoque radica en que el desarrollador debe preveer todo lo que el usuario pueda decir al sistema para así evitar que el chatbot no sepa qué responder así como definir las posibles frases de respuesta del sistema. Esta desventaja resulta poco flexible ya que es difícil hacer un sistema basado en reglas que abarque un contexto muy grande [14].

- Chatbot con integración de grandes modelos de lenguage (LLM): Los LLMs permiten realizar un procesamiento del lenguaje más sofisticado para entender entradas del usuario más complejas y generar respuestas variadas. Sin embargo, presentan las limitaciones de que en ocasiones puede responder de manera genérica al no tener información o puede mostrar inconsistencias en las respuestas. Además, existe la posibilidad de que genere alucinaciones, proporcionando respuestas incoherentes. Asimismo, no se conoce con certeza la fuente exacta de la información que proporciona. [15].

- Chatbot basado en IA con RAG: Solventa gran parte de los problemas presentes en el enfoque anterior. La técnica Retrieval Augmented Generation permite dotar al modelo de un contexto concreto que permite que circunscriba sus respuestas a la base de conocimiento proporcionada, especializándose en un tema en concreto. El problema es que el contexto sea falso, provocando que el chatbot hable contando bulos. Esto se puede evitar proporcionando información fidedigna y aplicando de manera correcta técnicas de prompting para que el sistema sea transparente respecto a sus fuentes y no se salga del dominio considerado como contexto [16].

El uso de un chatbot basado en reglas aporta la posibilidad de tener respuestas completamente controladas, pudiendo responder a las preguntas más sensibles. Esto no es suficiente ya que la interacción humano-robot debe ser lo más flexible y fluida posible al tratar de simular una conversación tal y como se tendría entre personas. Es por ello que el chatbot que hace uso de LLM es de gran utilidad, su flexibilidad y entendimiento del lenguaje natural hacen que pueda entender y dar respuestas de todo tipo al usuario pero no siempre son fiables. En este trabajo combinamos todos los enfoques aplicando cada técnica al contexto de conversación que mejor se adapta a las capacidades y limitaciones de cada una.

Para lograr realizar el presente trabajo se va utilizar Furhat, un robot social que puede imitar la manera de actuar de las personas. Esto presenta la

ventaja que es más intuitivo y natural para interactuar con él lo cual supone el reto de que sus usuarios se van a drigir a él de manera más parecida a como lo harían con una persona, lo que implica que hay que diseñar con mucho cuidado sus capacidades convesracionales. No solo es capaz de imitar el habla y la apariencia de las personas, si no que también logra emular la comunicación no verbal como los gestos que realizan los seres humanos al interactuar unos con otros [17].

1.2. Objetivos

Los objetivos del proyecto son los siguientes:

- Crear un sistema conversacional para el robot Furhat que contribuya a la alfabetización de la salud mental haciendo uso de una interacción humano-robot de forma oral.
- Implementar una interacción que funcione por reglas (conversación guiada) acerca de los principales temas de alfabetización en salud mental, garantizando respuestas fidedignas.
- Implementar una interacción que use LLM con RAG (conversación abierta más limitada a una base de conocimiento), para hablar sobre temáticas contenidas en las guías de ayuda a estudiantes del Gabinete Psicopedagógico de la Universidad de Granada.
- Implementar una interacción que haga uso de LLM (conversación abierta) cuando la conversación se salga del ámbito de la salud mental, permitiendo una conversación más abierta con el robot.
- Gestionar los tres tipos de interacciones en un mismo sistema, garantizando una experiencia transparente al usuario. en la que no note que cada respuesta puede provenir de una interacción distinta.
- Apoyar la interacción oral con otras modalidades (expresiones faciales, luces) para una mejor comprensión de la interacción.
- Crear una aplicación web sencilla para poder añadir información adicional profesional a los conocimientos del chatbot basado en reglas.

Capítulo 2

Trabajos relacionados

En los últimos años ha cobrado relevancia el ámbito de la denominada "salud mental digital", donde se estudia el uso de herramientas TIC para favorecer el bienestar de las personas, uno de los lugares que reúne a expertos en este tema es «Digital Mental Health and Wellbeing Conference» [18]. Existen una gran variedad de sistemas digitales con este propósito.

Aplicaciones móviles y web

Presentan la ventaja de que se pueden utilizar en cualquier lugar teniendo acceso a un móvil o navegador. Tienen una interfaz que es generalmente intuitiva y son muy populares. Igualmente presentan problemas como la dificultad de mantener el interés del usuario, la necesidad de pagos adicionales para acceder al contenido total o que es un mercado grande sin regulación. Esto último provoca que no todas las aplicaciones que se pueden encontrar tanto en los móviles como la web estén verificadas por profesionales, haciendo que muchas no sean de fiar.

Cada aplicación posee distintos enfoques como terapia en línea, aplicaciones de meditación, de detección de síntomas o juegos de relajación. Existen un gran número de aplicaciones en estas tecnologías, algunas de ellas son:

- **MoodTools:** Se define a sí mismo como una aplicación de ayuda para la depresión. Contiene diferentes funciones dentro de la aplicación como tests para controlar la severidad de los síntomas, vídeos de meditación, charlas para conocer más acerca de este trastorno o un diario de pensamientos. También permite crear un plan de seguridad en caso de crisis. [19]
- **Headspace:** Se trata de una de las aplicaciones más descargadas y utilizadas en relación con la salud mental. Enseña a meditar, aliviar el

estrés, ayuda a dormir, a mejorar la productividad y la concentración... Tiene una interfaz gráfica muy llamativa. [20]

- **Happify:** Esta aplicación trata de ayudar a mantener un buen bienestar emocional. Ofrece una variedad de juegos relajantes además de guías para meditación y artículos sobre salud mental y mindfulness. [21]

- **BetterHelp:** Esta aplicación conecta al usuario con un terapeuta especializado. Tienen una gran variedad de personal disponible, pudiendo conectar por chat o llamada con especialistas en depresión, ansiedad, terapia de pareja, etc. [22]

- **Substance Abuse and Mental Health Services Administration (SAMHSA):** Esta agencia pretende concienciar a la población acerca de la salud mental y el consumo de sustancias. Su web explica los distintos trastornos, los riesgos de cada sustancia y ofrece ayuda a las personas que lo necesiten. [23]

- **7cups:** Este sitio web permite a los usuarios contactar con voluntarios para ser escuchados acerca de sus problemas. Adicionalmente ofrece ejercicios de estrategias de afrontamiento y foros en los que es posible contactar con gente con los mismos problemas o dispuesta a ayudar a los demás. [24]

Videojuegos

Este tipo de software puede abordar y tratar de concienciar a la población sobre distintos tipos de trastornos mentales. Pueden aprovechar la inmersión del usuario para presentar distintas narrativas y hacer uso de imágenes y audio para lograr tener impacto en la percepción del jugador acerca de la salud mental. Presentan límites al no ser un medio para todo el público, además de tener un alcance más limitado ya sea por la habilidad del jugador o por la necesidad de poseer una videoconsola o un ordenador de alto rendimiento. Algunos ejemplos que muestran el potencial de los videojuegos son:

- **Hellblade: Senua's Sacrifice:** Este videojuego de acción busca hacer comprender a los jugadores cómo se vive la psicosis y la esquizofrenia. La protagonista padece esa enfermedad de manera muy realista gracias a la implicación llevada a cabo por el estudio desarrollador del juego, que estuvo en continuo contacto con psiquiatras y psicólogos especializados en la psicosis y con pacientes diagnosticados tanto de psicosis como esquizofrenia. El juego ha sido aclamado por jugadores

que padecen la misma condición, explicando que el resto de la población que lo juegue será capaz de entender cómo se siente y se sufre el trastorno. [25]

- **Depression Quest:** Es un juego de toma de decisiones que dona todos sus ingresos a la Línea de Prevención del Suicidio y Crisis. Depression Quest hace que la persona que lo juegue se ponga en la piel de alguien que tiene depresión, haciendo ver que no están solos a aquellos que padezcan el trastorno y dando un enfoque más ilustrativo y profundo a aquellos que no lo padezcan. [26]

Realidad virtual

A pesar de su gran costo y difícil acceso para toda la sociedad, existen distintos usos de la realidad virtual en el ámbito de la salud mental. Con esta tecnología ha sido posible estudiar la adicción, hacer frente a fobias, crear tareas de atención para personas que muestren TDAH, ayuda para el tratamiento de la depresión y del estrés postraumático, etc [27] [28]. Tiene grandes posibilidades de uso debido a la inmersión que provoca al usuario, adaptándose a las situaciones requeridas del paciente. Por ejemplo, para el tratamiento de fobias, se pueden mostrar con mayor o menor intensidad la fobia, viendo la reacción del usuario. Es un sistema capaz de tomar datos en tiempo real y es un tratamiento atractivo al salirse de los estándares. A pesar de todo ello, no es una herramienta accesible para todo el mundo ya que puede producir mareos y es un método que no posee mucha evidencia.

Un claro ejemplo del uso que se le puede dar a la realidad virtual en el campo de la salud mental es XRHealth. Este sistema de realidad virtual trata de ayudar en el cuidado de la salud a los pacientes. A parte de juegos de memoria o de rehabilitación física también ofrece material educativo para la depresión, el estrés o los ataques de pánico. Asimismo ofrece ayudas para la atención, meditación y tratamiento de fobias como aracnofobia, acrofobia o agorafobia [29].

Chatbots

El uso de chatbots muestra grandes oportunidades tal y como se ha mencionado previamente en la introducción. Su facilidad de uso hace que tengan un gran atractivo y pueden ser utilizados para detectar trastornos, ayudar a la meditación y relajación o directamente actuar como apoyo psicológico en caso de ataque de pánico. La capacidad que tiene para el entendimiento del lenguaje natural los hace una herramienta idónea para todas las edades. El hecho de que funcione como conversación logra que la población se mantenga usándolo por mayores períodos de tiempo. Adicionalmente son

inmersivos al formar una conversación, ayudando a desarrollar habilidades sociales a los pacientes **??**. Las limitaciones radican en la dificultad de dar siempre respuestas de valor a las preguntas.

Algunos ejemplos de chatbots son:

- **Coach virtual de mindfulness:** Este chatbot llamado «Chris» consiste en una interacción con un profesional de meditación con el objetivo de que los usuarios practiquen el mindfulness y la meditación para reducir síntomas como la ansiedad, el estrés o mejorar las funciones cognitivas. Está implementado en web y el asistente virtual tiene distintas caras mostrando emociones durante las conversaciones para una mejor interacción no verbal.[30]

- **Ada:** A pesar de no ser específico de salud mental, Ada es una aplicación que hace uso de un chatbot para realizar diagnósticos. Comprueba los síntomas del usuario, busca las causas y da consejos para el auto cuidado además de explicar medidas preventivas. Es una de las aplicaciones móviles más utilizadas para la evaluación de síntomas de cualquier tipo de salud. [31]

- **Woebot:** Es un chatbot semiguiado (puede recibir texto escrito o respuestas predefinidas) especializado en el apoyo de la salud mental del usuario el cual tiene un buen récord de disminución de síntomas depresivos. Tiene una interacción muy humana que no se siente artificial, logrando así que el usuario se abra más para hablar de sus inquietudes. [32]

- **Mindspa:** Esta aplicación posee un chatbot guiado para casos de emergencia, charlas terapéuticas y apoyo en momentos de crisis. Además va más allá de la implementación de un chatbot y también ofrece cursos terapéuticos, analíticas, artículos para mejorar la salud mental. [33]

2.1. Comparativa

Se ha podido observar que hay una gran cantidad de medios con los que abordar la salud mental. A continuación se evaluarán las ventajas y las desventajas de cada uno.

	Ventajas	**Desventajas**
Aplicaciones móviles/web	Interfaz generalmente intuitiva, accesible, popular, se puede usar en cualquier lugar	Difícil mantener el interés del usuario, funcionalidades detrás de subscripciones extra, mercado grande sin regulación (conlleva a falta de verificación profesional en algunas aplicaciones)
Videojuegos	Inmersivo, Atractivo para la población joven, uso de música y visuales para mayor énfasis	Puede ser costoso, no es accesible para todo el mundo, presentan límites de los temas a tratar
Realidad virtual	Muy inmersivo, presenta situaciones reales interactivas, personalizable, captura de datos en tiempo real, atractivo	Muy costoso, puede producir mareos, no es muy accesible, sus métodos no tienen tanta evidencia, mantenimiento, manejo de datos sensibles
Chatbots	Comprende el lenguaje natural, facilidad de uso, conversaciones más humanas, interactivo, atractivo, los pacientes sienten confianza y cercanía, adherencia, accesible para todo el mundo, uso de situaciones reales (ej. entrevistas), no juzga los comentarios del usuario	No siempre pueden dar respuesta a la solicitud del usuario, posibilidad de respuestas genéricas, con IA respuestas fuera del control, manejo de datos sensibles

Tabla 2.1: Ventajas y desventajas de cada tecnología

Evidentemente a parte de las ventajas de la tabla las hay compartidas como anonimato y la disponibilidad.

A continuación la comparación se va a centrar en los chatbots mencionados previamente, ya que son los que guardan mayor similitud con el proyecto

y los que han mostrado unas ventajas más atractivas y accesibles para toda la población. En la comparativa «Furhat MHL» es el desarrollado en este TFG.

Se van a comparar según diferentes medidas.

- Comunicación no verbal: Si la poseen y la manera que la usan.
- Tipo de conversación: Si el robot solo permite que el usuario elija respuestas (guiada), si solo permite que el usuario escriba lo que quiera (abierta) o una mezcla de las dos (semi-guiada).
- Medio de comunicación: Cuál es el output, la manera de comunicarse, que utiliza el chatbot.
- Tema principal de conversación: En qué se centra o especializa el chatbot.

	Chris	**Ada**	**Woebot**	**Mindspa**	**Furhat MHL**
Comunicación no verbal	Gestos mediante imágenes	No	No	No	Gestos humanos, luz
Tipo de conversación	Semi-guiada	Guiada	Semiguiada	Guiada	Abierta con recomendaciones
Medio de comunicación	Texto	Texto	Texto y recursos audiovisuales	Texto y vídeo	Voz
Tema principal de conversación	Meditación y mindfulness	Detección de enfermedades	Estrés, ansiedad, depresión, autocuidado	Ansiedad, depresión, autoestima, autocuidado	Concienciación, acabar con el estigma y falsas creencias. Dar consejos

Tabla 2.2: Comparativa de chatbots

Se aprecia que los chatbots no se centran en la alfabetización si no que se especializan en en algún otro tema en concreto. Por ejemplo, realizan diagnósticos (Ada), hacen trabajo de un entrenador personal («Chris» o

Woebot) o directamente hacen de terapeuta (Mindspa). [32] El chatbot trabajado en este proyecto busca ser centrado en la alfabetización, actuando como un divulgador del tema. Sirve para conocer bien qué es la alfabetización de la salud mental y lograr obtener conocimientos de temas como distintos trastornos, mitos, dónde buscar ayuda o por qué es importante la propia alfabetización de la salud mental. Al contrario de los demás no busca centrarse solo en ayudar, si no en divulgar información relevante para acabar con el estigma de la salud mental además de sensibilizar a la población sobre conocer cómo actuar en caso de que necesiten ayuda ellos mismos o su entorno. El uso de gestos humanos le hace más cercano al usuario, además el uso de voz da mayor accesibilidad que los demás.

Capítulo 3

Propuesta

3.1. Planificación

Para la planificación se han dividido todas las tareas en grupos. El primero fue la preparación del proyecto

El primero fue la preparación para la realización del proyecto. Se puede apreciar un tiempo de preparación largo ya que se quería tener un conocimiento exhaustivo de todo lo necesario para un correcto desarrollo del proyecto. Estos conocimientos fueron:

- Estudiar el SDK de Furhat para conocer todas sus posibilidades. Para usar este software, es necesario tener conocimientos de Kotlin, lenguaje de programación del que no se estaba muy familiarizado. En adición, la documentación no está tan detallada como se esperaba, haciendo que fuese necesario probar ejemplos preexistentes creados por Furhat, estudiando el código continuamente y depurándolo. El estudio del SDK ayudó a la creación de un diagrama de flujo coherente con las posibilidades descubiertas.
- Para definir correctamente el alcance del proyecto fue necesario un estudio previo de múltiples artículos, guías, estudios tanto de profesionales de la salud mental, de la OMS, de universidades y de organismos oficiales. Este estudio permitió conocer en mayor profundidad distintas ideas sobre la salud mental, decidiendo en el proceso la información que se va a incorporar al sistema conversacional.
- El aprendizaje del resto de tecnologías también fue importante. Destacando en concreto el aprendizaje de los sistemas conversacionales del cual no se poseía experiencia y del que se ha necesitado tiempo para familiarizarse con los distintos modelos de lenguaje y sus paradigmas de implementación.

El siguiente paso fue el análisis y diseño de la solución. Se pensó cómo se debía llevar a cabo la interacción persona-robot, qué se iba a poder hacer en la conversación, cómo se va a utilizar la información que se va a incorporar al chatbot.

Con todo lo anterior completado se pudo comenzar la implementación. Se realizó de manera incremental, añadiendo poco a poco cada una de las ideas hasta lograr la interacción actual. Para el sistema conversacional, la primera interacción lograda fue la más básica, la que hace uso de LLM. Fue realizada de manera simple y se fue mejorando conforme se fue progresando en el desarrollo. Después se le incorporó la interacción basada en reglas y finalmente la que hace uso de LLM con RAG. La aplicación web también se creó incrementalmente, logrando un backend funcional, luego un frontend y finalmente un sistema de inicio de sesión.

Desde el momento que se podía tener una conversación fluida con el chatbot se estuvo sometiendo a pruebas exhaustivas probando con frases similares, mirando posibles gestos nuevos, implementando interacciones nuevas que podían ser relevantes, etc. Cada endpoint nuevo y cada interacción nueva eran probados en conjunto con el resto de la interacción para comprobar que era correcto y que se mantenía el funcionamiento en conjunto.

Asimismo la memoria se empezó a realizar concurrentemente junto al proyecto a partir del punto en el que se consideró que el sistema estaba lo suficientemente avanzado como para poder empezar a detallarlo.

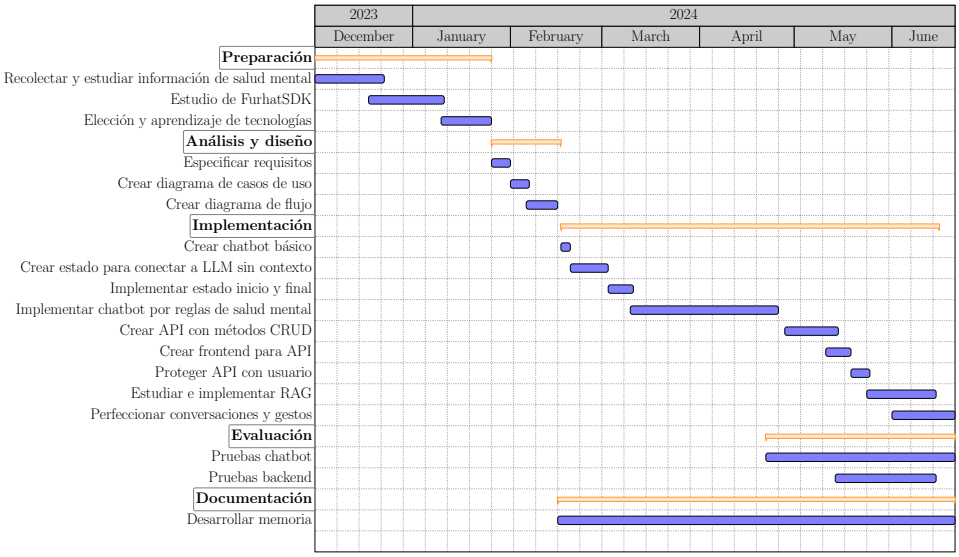


Figura 3.1: Diagrama de Gantt

3.2. Presupuesto

A continuación se detallarán todos los costes relacionados con la elaboración del Trabajo de Fin de Grado. Habría que diferenciar distintos grupos para los costes:

Personal: Se ha tomado como referencia el salario de un ingeniero informático junior. Teniendo en cuenta que el Trabajo de Fin de Grado consiste de 12 créditos y que cada crédito equivale a 25 horas, el trabajo consiste en 12 * 25 = 300 horas. Volviendo al salario de programador junior según la página de búsqueda de empleo Jooble.com [34] con una muestra de más de 15000 ofertas el salario medio consiste en 1957€ al mes o lo que es lo mismo 13,98€ la hora. Teniendo 300 horas y 13,98€/hora se multiplican los valores y se obtienen 4194€ para un trabajo de 300 horas. En este apartado sólo se ha considerado el coste del trabajo del desarrollador no se han tenido en cuenta las horas de asesoramiento con la tutora.

Material: En cuanto al material se ha hecho uso de un portátil Lenovo cuyo valor es de 900€. Además del portátil se hizo uso del robot físico de Furhat, facilitado por la tutora y la Universidad de Granada por lo que no se tuvo que gastar dinero en él para la realización de este proyecto. El robot Furhat puede ser adquirido eligiendo entre tres niveles, se ha optado por el más básico [35]. No se ha considerado el coste del despliegue del LLM y del servicio del servidor ya que están disponibles mediante el grupo de investigación de la tutora. Por tanto los costes del hardware son los siguientes:

Hardware	**Precio (€)**
Ordenador personal	900
Robot Furhat Basic	15000
Total	**15900**

Tabla 3.1: Presupuesto del hardware

Licencias software: Las licencias software empleadas en el proyecto han sido IntelliJ IDEA Ultimate para el desarrollo del chatbot con Kotlin y PyCharm Professional para la aplicación web. Estas licencias no han costado dinero ya que se disponía de la licencia de estudiante, la cual da la posibilidad de hacer uso durante un año estas tecnologías para fines educativos. Sin contar esta ventaja el coste de las licencias para recién graduados son: (para recién graduados resultan un 40 % más baratas) [36]

Licencia	Precio (€)
IntelliJ IDEA Ultimate	101,40
Pycharm Professional	59,40
Total	**160,80**

*Estos precios son hipotéticos al haber podido obtener la licencia gratis al ser estudiante.

Tabla 3.2: Presupuesto de las licencias

Por tanto, el presupuesto total sumando todas las secciones es el siguiente:

Categoría	Precio (€)
Recursos humanos	4194
Hardware	15900
Licencias software	160,80
Total	**20.254,8**

Tabla 3.3: Presupuesto total del proyecto

3.3. Análisis de requisitos

En este apartado se presentarán los requisitos funcionales y no funcionales del sistema que se ha desarrollado.

3.3.1. Requisitos funcionales

Los requisitos funcionales explican cómo debe comportarse el software, qué ofrece el sistema al usuario. En este proyecto se pueden dividir en dos partes, los requisitos funcionales del sistema conversacional y los de la aplicación web.

Requisitos funcionales del chatbot

ID	RF-1
Nombre	Indicar de qué hablar
Descripción	Los usuarios deben poder decidir si quieren hablar de salud mental o cambiar de tema.

Tabla 3.4: Requisito funcional RF-1 Indicar de qué hablar

ID	RF-2
Nombre	Preguntar sobre trastornos
Descripción	Los usuarios deben poder preguntar acerca de los síntomas, causas y tratamientos de los distintos trastornos.

Tabla 3.5: Requisito funcional RF-2 Preguntar sobre trastornos

ID	RF-3
Nombre	Preguntar sobre consejos de salud mental
Descripción	Los usuarios deben poder preguntar acerca de la auto-ayuda, los fármacos, las drogas, dónde encontrar ayuda o qué es la alfabetización

Tabla 3.6: Requisito funcional RF-3 Preguntar sobre consejos de salud mental

ID	RF-4
Nombre	Realizar test de mitos de la salud mental
Descripción	Los usuarios deben poder recibir preguntas acerca de mitos e ideas erróneas sobre la salud mental

Tabla 3.7: Requisito funcional RF-4 Realizar test de mitos de la salud mental

ID	RF-5
Nombre	Responder preguntas o frases típicas
Descripción	El sistema reaccionará a algunas frases o preguntas típicas como «eres hombre o mujer» o insultos.

Tabla 3.8: Requisito funcional RF-5 Preguntar sobre consejos de salud mental

ID	RF-6
Nombre	Preguntar sobre problemas de la vida universitaria
Descripción	Los usuarios deben poder preguntar sobre problemas tales como ansiedad ante los exámenes, cómo afrontar exámentes tipo test o cómo estudiar mejor y el sistema usará documentos del gabinete psicopedagógico para responder.

Tabla 3.9: Requisito funcional RF-6 Preguntar sobre problemas de la vida universitaria

ID	RF-7
Nombre	Preguntar sobre otros temas
Descripción	Los usuarios deben poder preguntar sobre otros temas en caso de no querer hablar de salud mental o problemas de la vida universitaria.

Tabla 3.10: Requisito funcional RF-7 Preguntar sobre otros temas

ID	RF-8
Nombre	Comenzar la conversación
Descripción	El sistema debe poder comenzar la conversación cuando al menos un usuario se haya acercado lo suficiente.

Tabla 3.11: Requisito funcional RF-8 Comenzar la conversación

ID	RF-9
Nombre	Terminar la conversación
Descripción	Los usuarios deben poder terminar la interacción con el chatbot en cualquier momento.

Tabla 3.12: Requisito funcional RF-9 Terminar la conversación

ID	RF-10
Nombre	Complementar la comunicación verbal
Descripción	El sistema complementará la comunicación oral con expresiones faciales.

Tabla 3.13: Requisito funcional RF-10 Complementar comunicación verbal

ID	RF-11
Nombre	Aclarar turno de palabra
Descripción	El sistema hará uso de luces LED para ayudar al usuario a discernir cuándo el sistema está escuchando y cuando está hablando.

Tabla 3.14: Requisito funcional RF-11 Aclarar turno de palabra

Requisitos funcionales de la aplicación web para la edición de preguntas y trastornos

ID	RF-12
Nombre	Iniciar sesión
Descripción	Los usuarios deben poder iniciar sesión en la aplicación con unos credenciales.

Tabla 3.15: Requisito funcional RF-12 Iniciar sesión

ID	RF-13
Nombre	Cerrar sesión
Descripción	Los usuarios deben poder cerrar sesión, acabando con la sesión actual.

Tabla 3.16: Requisito funcional RF-13 Cerrar sesión

ID	RF-14
Nombre	Listar trastornos
Descripción	Los usuarios deben poder obtener la lista completa de los trastornos almacenados en el sistema.

Tabla 3.17: Requisito funcional RF-14 Listar trastornos

ID	RF-15
Nombre	Añadir trastornos
Descripción	Los usuarios deben poder añadir trastornos que consideren necesarios a la lista.

Tabla 3.18: Requisito funcional RF-15 Añadir trastornos

ID	RF-16
Nombre	Borrar trastornos
Descripción	Los usuarios deben poder borrar trastornos que no sean necesarios o estén mal de la lista.

Tabla 3.19: Requisito funcional RF-16 Borrar trastornos

ID	RF-17
Nombre	Editar trastornos
Descripción	Los usuarios deben poder editar trastornos ya creados para añadir más información o corregir errores.

Tabla 3.20: Requisito funcional RF-17 Editar trastornos

ID	RF-18
Nombre	Listar preguntas
Descripción	Los usuarios deben poder obtener la lista completa de las preguntas almacenadas en el sistema.

Tabla 3.21: Requisito funcional RF-18 Listar preguntas

ID	RF-19
Nombre	Borrar preguntas
Descripción	Los usuarios deben poder borrar preguntas que no sean necesarias o estén mal de la lista.

Tabla 3.22: Requisito funcional RF-19 Borrar preguntas

ID	RF-20
Nombre	Editar preguntas
Descripción	Los usuarios deben poder editar preguntas ya creadas para añadir más información o corregir errores.

Tabla 3.23: Requisito funcional RF-20 Editar preguntas

3.3.2. Requisitos no funcionales

También conocidos como requisitos de calidad, detallan las características de funcionamiento del sistema. Son restricciones impuestas para asegurar la mayor calidad posible del software.

ID	RNF-1
Nombre	Usabilidad
Descripción	Tanto el flujo de la conversación del chatbot como la aplicación web serán fáciles de usar. Por parte del chatbot la conversación será clara y se sabrá qué hacer en cada situación y por la parte de la aplicación web tendrá una interfaz simple e intuitiva para los usuarios.

Tabla 3.24: Requisito no funcional RNF-1 Usabilidad

ID	RNF-2
Nombre	Seguridad y privacidad
Descripción	El sistema no guardará información personal del usuario y garantizará su privacidad total.

Tabla 3.25: Requisito no funcional RNF-2 Seguridad y privacidad

ID	RNF-3
Nombre	Fiabilidad
Descripción	Tanto el chatbot como la aplicación web deben funcionar de manera consistente y dejar claro en todo momento qué es lo que está ocurriendo en el sistema. En caso del chatbot debe dejar claro el estado en el que se encuentra y responder consistentemente a las frases que recibe.

Tabla 3.26: Requisito no funcional RNF-3 Fiabilidad

ID	RNF-4
Nombre	Rendimiento
Descripción	El chatbot deberá dar respuestas en un tiempo óptimo que se asemeje al de una conversación normal con otra persona. La aplicación web deberá funcionar de manera óptima y no presentar retrasos con el manejo de información ni con su navegación.

Tabla 3.27: Requisito no funcional RNF-4 Rendimiento

ID	RNF-5
Nombre	Compatibilidad
Descripción	La aplicación web será usable en distintos navegadores y dispositivos.

Tabla 3.28: Requisito no funcional RNF-5 Compatibilidad

ID	RNF-6
Nombre	Escalabilidad
Descripción	Tanto el chatbot como la aplicación web deben de ser capaces de manejar un número de usuarios alto sin afectar al rendimiento.

Tabla 3.29: Requisito no funcional RNF-6 Escalabilidad

ID	RNF-7
Nombre	Idioma
Descripción	El idioma de la aplicación web será el español y las respuestas del chatbot también.

Tabla 3.30: Requisito no funcional RNF-7 Idioma

ID	RNF-8
Nombre	Mantenibilidad
Descripción	El sistema entero deberá ser modular para asegurar que sea fácil de mantener y actualizar. En caso de añadir nuevas funcionalidades el resto no deben verse afectadas.

Tabla 3.31: Requisito no funcional RNF-8 Mantenibilidad

3.4. Casos de uso

Para el modelo de los casos de uso se tienen dos actores principales diferentes que interactúan con distintas partes del sistema. Por un lado es posible diferenciar al usuario de la aplicación web que es el profesional de la salud mental que tiene el conocimiento experto para incorporar al chatbot información sobre trastornos y preguntas de salud mental. Por otro lado se encuentra el usuario normal que interactúa con el chatbot y que busca concienciarse acerca de la salud mental o que simplemente quiere tener una conversación con el robot.

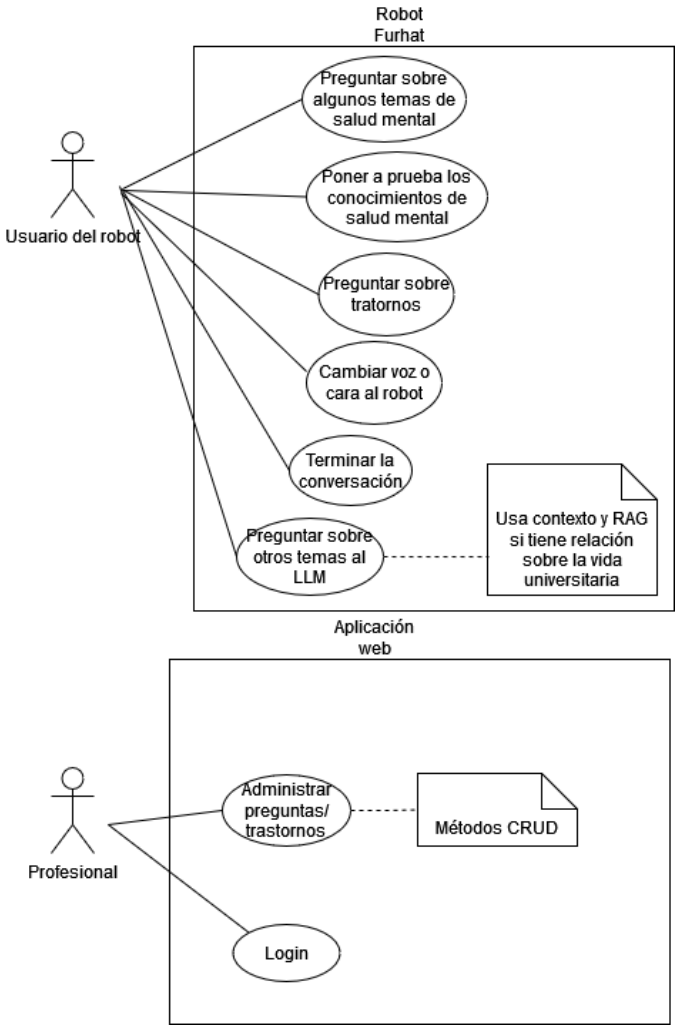


Figura 3.2: Diagrama de casos de uso

3.4.1. Descripción de los casos de uso

A continuación se describen los casos de uso con la siguiente información:

- **Actores:** Entidades externas al sistema que interactúan con él.
- **Referencias:** Aquellos requisitos funcionales que se incluyen en el caso de uso.
- **Precondición:** Condición que se tiene que cumplir en el sistema para que pueda dar lugar el caso de uso.
- **Poscondición:** Condición que se cumple tras completar el caso de uso.

- Propósito: Breve explicación del caso de uso

- Curso normal de eventos: Flujo de acciones que ocurren en el caso de uso.

- Curso alternativo de eventos: Flujo de acciones alternativas que ocurren a partir de cierto punto en el caso de uso.

Caso de uso	Administrar preguntas/trastornos	CU-01
Actores	Profesional	
Referencias	RF-14, RF-15, RF-16, RF- 17, RF-18, RF-19 RF-20	
Precondición	Estar logueado en la aplicación	
Poscondición	El usuario logra realizar operaciones CRUD sobre preguntas y/o trastornos	
Propósito	Administrar la base de conocimiento de algunas partes del chatbot	
Curso normal de eventos	1. El usuario profesional decide que lista visualizar: trastornos o preguntas 2. El sistema muestra la lista pertinente. 3. El usuario elige si ver, borrar o editar de la lista o añadir un elemento. 4. El sistema procesa la petición y realiza lo que el usuario pide. 5. El usuario ve los cambios que ha realizado.	

Tabla 3.32: Caso de uso CU-01: Administrar preguntas/trastornos

Caso de uso	Login	CU-02
Actores	Profesional	
Referencias	RF-12	
Precondición	-	
Poscondición	El usuario se loguea en la aplicación y puede acceder a todas las funciones del sistema. Adicionalmente se crea un JWT (JSON Web Token) para poder hacer uso de la aplicación.	
Propósito	Iniciar sesión en la aplicación para acceder al resto de funcionalidades.	
Curso normal de eventos	1. El usuario profesional ingresa en los campos de la pantalla de inicio de sesión su correo y contraseña. 2. El sistema busca al usuario y crea un JWT a partir de los datos del usuario. 3. El usuario recibe el JWT y gana acceso al resto de la aplicación.	
Curso alterno de eventos	2.a El sistema busca al usuario y no lo encuentra. 1. El usuario recibe un mensaje de credenciales erróneos.	

Tabla 3.33: Caso de uso CU-02: Login

Caso de uso	Preguntar sobre ayuda universitaria u otros temas al LLM	CU_03
Actores	Usuario del robot	
Referencias	RF-1, RF-6, RF-7, RF-10, RF-11	
Precondición	El robot está iniciado y está entablando una conversación con al menos un usuario.	
Poscondición	El robot envía la respuesta de la pregunta al usuario.	
Propósito	Preguntar sobre un tema que no está relacionado con la salud mental y recibir una respuesta usando el LLM con o sin RAG	
Curso normal de eventos	1. El usuario dice que no quiere hablar de salud mental. 2. El sistema ofrece nuevos temas de conversación y espera una respuesta. 3. El usuario formula una pregunta. 4. El sistema procesa la pregunta y decide qué tipo de pregunta es. 5. El sistema clasifica la pregunta como 'RAG' (pregunta relacionada con estudios) y realiza una petición al LLM con la pregunta del usuario y el contexto. 6. El usuario recibe una respuesta a su pregunta según el contexto.	
Curso alterno de eventos	5.a El sistema clasifica la pregunta como 'Otro' (pregunta sin relación a ayuda para estudios ni salud mental) y realiza una petición al LLM con solo la pregunta del usuario. 1. El usuario recibe una respuesta a su pregunta según los conocimientos previos del LLM. 5.b El sistema clasifica la pregunta como 'MHL' (pregunta relacionada con la salud mental). 1. El usuario es informado que se puede realizar un cambio de estado para hablar de salud mental. 5.c El sistema no puede clasificar la pregunta por un error del servidor. 1. El usuario es informado que hubo un error. 2. El sistema se va al estado de dormir.	

Tabla 3.34: Caso de uso CU_03: Preguntar al LLM

Caso de uso	Preguntar sobre algunos temas de salud mental	CU_04
Actores	Usuario del robot	
Referencias	RF-1, RF-3, RF-10, RF-11	
Precondición	El robot está iniciado y está entablando una conversación con al menos un usuario.	
Poscondición	El robot envía la respuesta de la pregunta al usuario.	
Propósito	Recibir respuesta sobre un tema relacionado con la salud mental.	
Curso normal de eventos	1. El usuario dice que quiere hablar de salud mental. 2. El sistema responde preguntando que qué quiere saber y da ideas. 3. El usuario formula una pregunta. 4. El sistema procesa la pregunta y decide qué tipo de pregunta es. 5. El sistema clasifica la pregunta como 'MHL' (pregunta relacionada con vida universitaria) 6. El sistema encuentra una posible respuesta entre su conocimiento. 7. El usuario recibe una respuesta coherente y relacionada con su pregunta.	
Curso alterno de eventos	5.a El sistema clasifica la pregunta y no ve relación con salud mental. 1. El usuario es informado que hubo un cambio de estado para hablar de temas sin relación con salud mental. 5.b El sistema no puede clasificar la pregunta por un error del servidor. 1. El usuario es informado que hubo un error. 2. El sistema se va al estado de dormir. 6.a El sistema no encuentra una posible respuesta. 1. El usuario es informado que el sistema no entendió su pregunta.	

Tabla 3.35: Caso de uso CU_04: Preguntar sobre salud mental

Caso de uso	Poner a prueba los conocimientos de salud mental	CU_05
Actores	Usuario del robot	
Referencias	RF-4, RF-10, RF-11	
Precondición	El sistema está en el estado de salud mental y el usuario ha pedido poner a prueba sus conocimientos de salud mental.	
Poscondición	-	
Propósito	Recibir preguntas sobre salud mental para aprender más.	
Curso normal de eventos	1. El sistema elige una pregunta del banco de preguntas. 2. El usuario recibe la pregunta y formula una respuesta. 3. El sistema procesa la respuesta como correcta o incorrecta y da información adicional. 4. El sistema pregunta si quiere otra pregunta. 5. El usuario confirma y recibe otra pregunta.	
Curso alterno de eventos	4.a Al sistema no le quedan más preguntas por hacer y dice al usuario si quiere empezar de nuevo. 1. Si el usuario confirma se vuelve al curso normal de eventos, si rechaza vuelve al estado de recibir preguntas de salud mental. 5.a El usuario rechaza recibir una pregunta más. 1. El sistema vuelve al estado de recibir preguntas de salud mental del usuario.	

Tabla 3.36: Caso de uso CU_05: Poner a prueba los conocimientos de salud mental

Caso de uso	Preguntar sobre trastornos	CU_06
Actores	Usuario del robot	
Referencias	RF-2, RF-10, RF-11	
Precondición	El sistema está en el estado de salud mental.	
Poscondición	El robot envía la información requerida del trastorno al usuario.	
Propósito	Recibir información acerca de las causas, tratamientos o síntomas de un trastorno.	
Curso normal de eventos	1. El usuario pregunta sobre un trastorno. 2. El sistema responde preguntando si quiere saber las causas, síntomas o tratamientos. 3. El usuario responde con uno de ellos. 4. El sistema dice la información relevante a la petición del usuario.	
Curso alterno de eventos	1.a El usuario pregunta directamente por las causas, síntomas o tratamientos de un trastorno. 1. El sistema busca el trastorno y responde según la petición. 1.b El usuario pregunta directamente por las causas, síntomas o tratamientos sin mencionar un trastorno. 1. El sistema responde con la información del último trastorno hablado o diciendo que le diga el trastorno si no se habló antes de alguno.	

Tabla 3.37: Caso de uso CU_06: Preguntar sobre trastornos

Caso de uso	Cambiar voz o cara al robot	CU_07
Actores	Usuario del robot	
Referencias	RF-5, RF-10, RF-11	
Precondición	El sistema está inicializado y en una conversación	
Poscondición	El sistema cambia la voz o la cara al robot.	
Propósito	Cambiar el tipo de voz o cara al robot para que el usuario se sienta más cómodo.	
Curso normal de eventos	1. El usuario pregunta acerca del género al robot. 2. El sistema explica que no tiene género pero ofrece la posibilidad de cambiar la voz y/o la cara. 3. El usuario elije las dos, una de ellas o ninguna. 4. El sistema realiza la acción que pide el usuario.	
Curso alterno de eventos	1.a El usuario pide directamente que cambie la voz. 1. El sistema procede y cambia su voz. 1.b El usuario pide directamente que cambie la cara. 1. El sistema procede y cambia su cara.	

Tabla 3.38: Caso de uso CU_07: Cambiar voz o cara al robot

Caso de uso	Terminar la conversación	CU_08
Actores	Usuario del robot	
Referencias	RF-9, RF-10, RF-11	
Precondición	El sistema está inicializado y en una conversación	
Poscondición	El sistema termina la interacción con el usuario y se va a dormir.	
Propósito	Finalizar la conversación con el usuario.	
Curso normal de eventos	1. El usuario abandona el área de la interacción. 2. El sistema espera un momento dando información que no ve a ningún usuario. 3. El sistema pasado un tiempo avisa que la interacción acabó y se duerme.	
Curso alterno de eventos	1.a El usuario se despide por medio de lenguaje natural. 1. El sistema manda la despedida de vuelta, espera un momento y se duerme.	

Tabla 3.39: Caso de uso CU_08: Terminar la conversación

3.5. Diseño

Con los próximos diagramas se visualizará el funcionamiento completo del sistema conversacional. Los recursos utilizados en la conversación han sido obtenidos de instituciones de reconocida solvencia en el ámbito de la salud mental como la Administración de Salud Mental y Abuso de Sustancias[1] (SAMHSA por sus siglas en inglés) y de artículos de autores como Anthony Jorm para la parte de la interacción por reglas y del gabinete psicopedagógico de la UGR [37] para la parte de LLM con RAG. La interacción por reglas corresponde a la conversación de salud mental, la interacción de LLM con RAG corresponde a los estudios y exámenes y la interacción de solo LLM se utiliza para todo lo demás. Estas tres partes están conectadas entre sí, decidiendo cuál es la que puede dar la mejor respuesta al usuario. La explicación de cómo funciona internamente se define en la implementación del chatbot 3.6.2 y del backend 3.6.3.

Para el diseño de la interacción con el robot se ha seguido el siguiente diagrama de flujo:

[1] https://www.samhsa.gov/

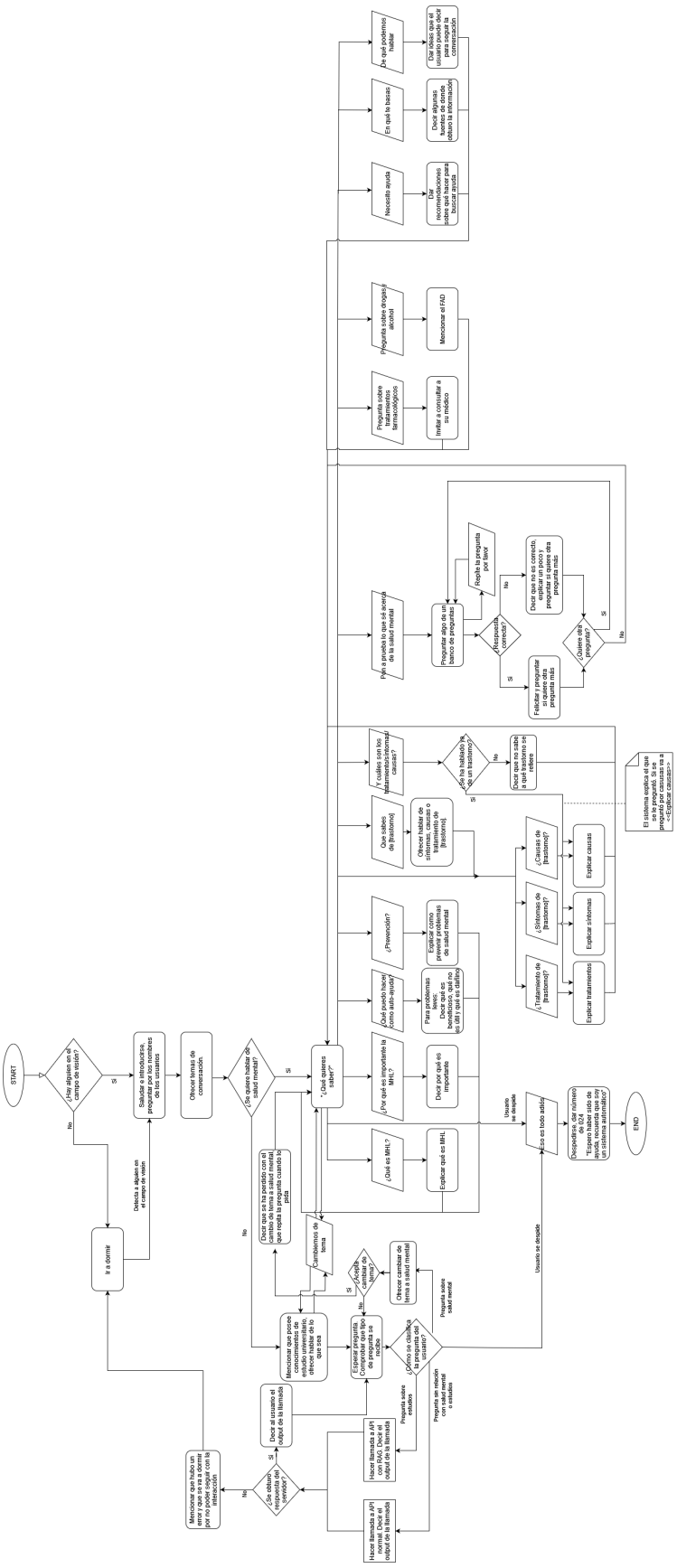


Figura 3.3: Diagrama de flujo

También se ha realizado el siguiente diagrama de estados:

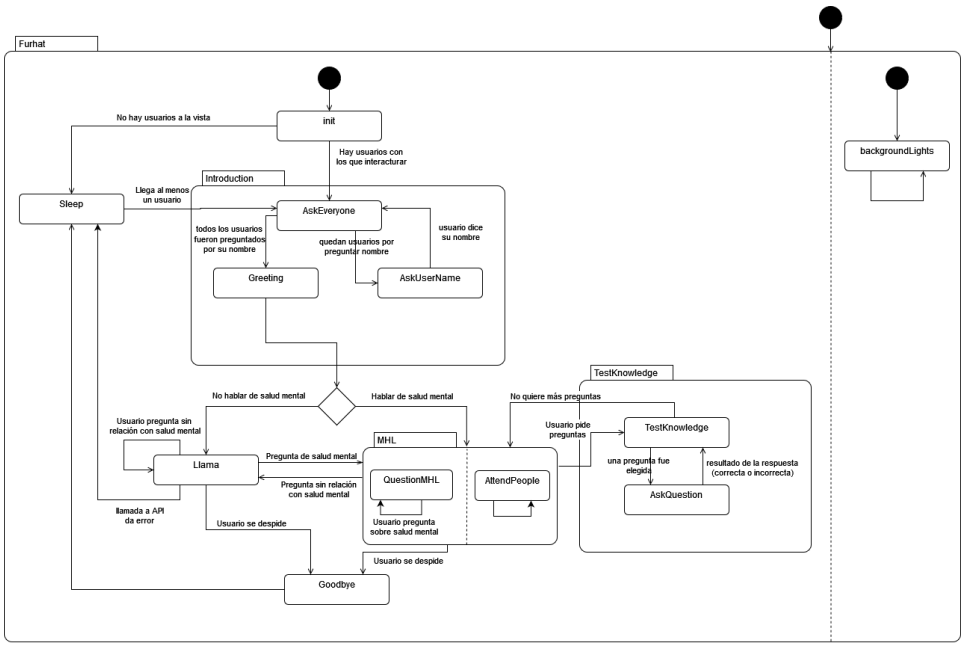


Figura 3.4: Diagrama de estados

A continuación se detallarán los posibles estados del robot, explicando en mayor profundidad el diagrama de flujo 3.3 y el de estados 3.4:

El primer estado en el que se encuentra el sistema al ejecutarlo es el estado *init*. Si no detecta usuarios se dirige al estado *Sleep*. Este último estado realiza un gesto en el que está permanentemente con la cabeza agachada y los ojos cerrados, si en algún momento alguien entra en el campo de visión realizará el gesto de despertarse, haciendo entender al usuario que su presencia ha sido detectada y que va a comenzar la interacción.

El siguiente paso es introducirse a las personas que quieren interactuar con el robot. Para ello, hace uso de tres estados distintos. En el primer estado, *AskEveryone*, saluda, dice su nombre y expresa interés en conocer los nombres de los usuarios. Por cada usuario que no supere el límite de usuarios de la interacción pregunta el nombre llamando a otro estado (*AskUserName*) en el que observa a una de las personas para dar a entender a quién se refiere. Les dice frases de saludo utilizando su nombre como por ejemplo «encantado de conocerte [nombre]» y pasa al siguiente. Una vez que se ha realizado la presentación de los nombres, el robot avanza al estado de introducir su funcionalidad, llamado *Greeting*. Aquí menciona los nombres de todos los participantes para que le presten más atención y explica que puede resolver dudas de salud mental y si están dispuestos a hablar de ello.

Si los usuarios responden negativamente el robot se dirige al estado *Llama*. En esta condición el robot explica que es capaz de resolver más dudas a parte de salud mental, expresa que tiene conocimientos acerca de otros temas como ayudas para estudiar y exámentes para que los universitarios no sufran de estrés o ansiedad y espera una pregunta. En este estado se lleva a cabo la técnica de enrutamiento que será detallada en la implementación. Con esta técnica el sistema decide si debe hacer uso de la interacción de LLM con RAG, de solo LLM o por reglas. En el caso de solo LLM implica que al sistema no le ha parecido que la pregunta del usuario tenga relación con los temas de los que tiene conocimiento, aquí funciona como sería realizar una consulta a cualquier LLM como ChatGPT o Gemini. Si se tiene que usar LLM con RAG significa que la pregunta tuvo relación por ejemplo con ansiedad en exámenes o técnicas de estudio. Se espera a la respuesta del modelo al que se le envía el contexto del gabinete psicopedagógico y contesta según los datos del gabinete. Este contexto es suministrado mediante archivos de texto simple y el uso de la técnica RAG. En el caso de entender que la pregunta es de salud mental, el robot pregunta al usuario si quiere cambiar de tema y hablar de salud mental. Si se responde que sí se cambia a la interacción por reglas y si responde negativamente pide disculpas por no haber entendido la pregunta bien y pide que se le reformule la pregunta de otra manera.

En cambio, si los usuarios responden afirmativamente a la pregunta de si quieren hablar de salud mental se utiliza la interacción por reglas que es el estado *QuestionMHL*. Aquí ofrece múltiples ideas y el robot invita a que se le pregunte «de qué podemos hablar» para dar más detalles de las reglas de la interacción y que el usuario sepa mejor qué puede preguntar. El nombre del usuario que pregunte es dicho aleatoriamente al principio de cada respuesta para parecer más cercano pero sin repetirlo siempre, mirándole al principio. En este estado responde utilizando por ejemplo la definición de Jorm de la alfabetización de salud mental o la información de los distintos trastornos según SAMHSA. Aquí también se utiliza la técnica de enrutamiento para poder saber si es necesario cambiar el estado a *Llama*. Además, hay otro estado en ejecución paralela, *AttendPeople*, que se encarga de alternar la atención del robot con los usuarios para mirar a todo el mundo y simular que está hablando a un público y no solo a una persona.

Una de las respuestas más interesantes es cuando el usuario pide que se pongan a prueba sus conocimientos de la salud mental. Ahí el robot cambia al estado en el que él es quien pregunta (*TestKnowledge*) en la interacción y el usuario responde. Se introduce el juego y se elige una pregunta aleatoria, con la pregunta se cambia al estado *AskQuestion* para realizar la pregunta a un usuario aleatorio, escuchar la respuesta y determinar si es correcta o incorrecta. Una vez respondido el usuario vuelve al estado *TestKnowledge*, dice si es correcto o incorrecto y pregunta si quiere seguir jugando. Si no quiere

se vuelve al estado *QuestionMHL*, si quiere se sigue jugando. *TestKnowledge* administra las preguntas y lleva el flujo de la conversación en cambio *AskQuestion* realiza la pregunta y determina si la respuesta del usuario es correcta o no.

En cualquier momento de la realización de preguntas se le puede decir adiós al robot y éste terminará la interacción explicando que él no es ningún profesional y que tiene número de ayuda. Espera un rato para que el usuario pueda abandonar la zona de interacción a tiempo y se va al estado *dormir*.

Durante toda la interacción paralelamente hay un estado encargado del manejo de las luces de Furhat. Estas luces hacen más fácil comprender si el robot está hablando (luz roja), está escuchando (luz verde) o está simplemente parado (sin luz).

En ocasiones si es acorde a la interacción el robot muestra tristeza, felicidad para parecer más empático, asombro, etc.

Retrieval-Augmented Generation

Es trascendental comprender qué es Retrieval-Augmented Generation, abreviado como RAG. Es una técnica que ayuda a lograr que los modelos de lenguaje grandes como las familias LLama o GPT se circunscriban a una base de conocimiento, consiguiendo que sean más transparentes y respondan con información relevante y actualizada. Se reducen así las desventajas de los LLM como el razonamiento de caja negra y las posibles alucinaciones. Para ello, al modelo se le administran documentos que constituyen la base de conocimiento que empleará para contestar a las preguntas, cuya comprensión puede servirse de su gran capacidad para el procesamiento del lenguaje [38].

Haciendo uso de LangChain [39], que es un framework para el manejo y gestión de LLM, los pasos relacionados con RAG son los siguientes:

- **Loading:** Consiste en cargar todos los archivos que son el contexto para el LLM, su base de conocimiento. Estos archivos son convertidos en *Documents* un tipo de dato con el que Langchain es capaz de trabajar.
- **Splitting:** En este paso se trata de hacer *chunking*. Esta técnica consiste en separar los documentos en trozos más pequeños ya que son muy largos y no caben como contexto para el modelo de datos, que tiene un máximo de tokens disponible. También sería útil aunque el modelo fuese capaz de procesar todo el texto porque la información se divide en secciones pequeñas más rápidas de procesar comparadas con el texto entero. Es importante además de seleccionar un tamaño de la división elegir la cantidad de caracteres que coincidirán entre textos

(overlapping). De esta manera se asegura que los textos no pierdan significado al principio de cada división, evitando dejar una frase a medio empezar al principio.

- **Storing:** Una vez se tienen divididos los textos en trozos más pequeños es el momento de almacenarlos para buscar entre ellos. Para ello se realiza una técnica de embedding en cada trozo y se guardan en una base de datos vectorial. Para buscar el resultado que tiene más parecido con la pregunta se realiza la misma técnica de embedding sobre la pregunta y se utiliza una función de búsqueda de similaridad, encontrando los trozos con el embedding que más se aproxime al embedding de la pregunta. La técnica de embedding transforma los textos en vectores de datos *float*, que son datos más manejables que largas cantidades de texto.

  Esta búsqueda se puede realizar de distintas maneras y en el proyecto se usa la conocida como similitud coseno. Al ser vectores de datos tipo *float* es posible efectuar esta medida de similitud. La similitud coseno se puede usar para comparar vectores sin importar el tamaño de cada uno, lo cual es una gran ventaja ya que no todos los vectores van a tener el mismo tamaño. El valor de similitud coseno es un número del -1 al 1 donde 1 es que los contenidos del vector son idénticos, 0 es que son completamente diferentes y -1 es que son opuestos. Lo que se hace es comparar el coseno del ángulo formado entre los dos vectores, por ejemplo, si el ángulo es de cero grados el coseno es uno, en cambio, si son ortogonales el coseno es 0. A continuación se muestra la fórmula de similitud coseno: [40][41]

$$\text{Similitud Coseno(A, B)} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

- **Querying:** Es el último paso. Se conecta al modelo al que se le quiere hacer la pregunta administrándole un *prompt* que explique su rol y cómo actuar. Finalmente se le introduce el contexto en el que buscar y la pregunta a responder. Este último paso produce una respuesta coherente usando solo como conocimiento los documentos suministrados.

El diagrama de la arquitectura del sistema es el siguiente:

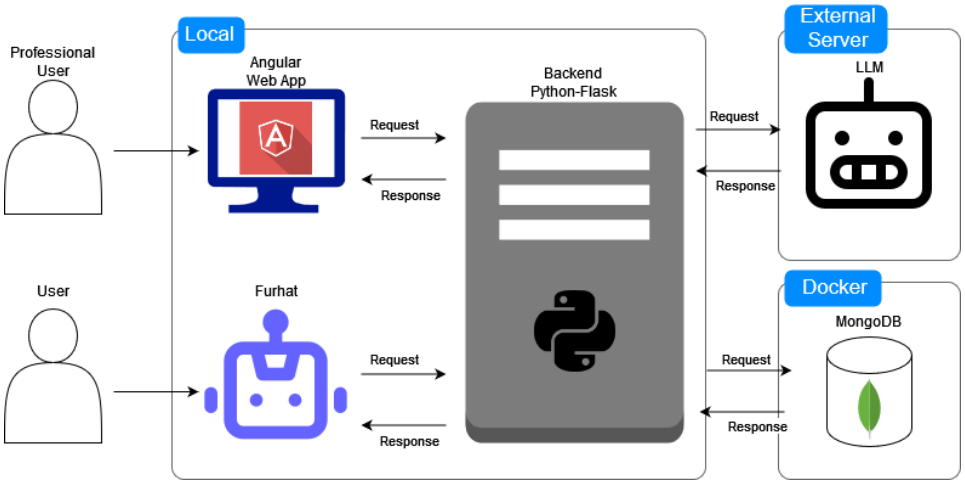


Figura 3.5: Diagrama de la arquitectura del software

En este diagrama es fácil ver como los diferentes componentes que forman el sistema interactúan entre sí. Se logra ver que se tienen dos tipos de usuarios que se comunican con el sistema a través de medios diferentes, una aplicación web y el robot Furhat. Estos dos componentes realizan peticiones al backend y éste obtiene la información que necesita para la respuesta pidiéndola al LLM que está alojado en un servidor externo o a la base de datos que está contenerizada mediante Docker.

La aplicación web es una herramienta se ha concebido en el TFG como un complemento, siendo el desarrollo de las capaciades conversacionales el objetivo principal del mismo.

Se ha realizado el siguiente diagrama de secuencia para entender mejor qué sucede dentro del estado *Llama*, que hace uso del LLM. Tal y como se observa en el diagrama de la figura 3.5, es posible detectar que se respeta esa relación y que cada componente del sistema pide la información necesaria a otro hasta llegar al LLM. También se aprecia cómo se lleva a cabo el enrutamiento para la elección la interacción necesaria que hace falta en cada situación.

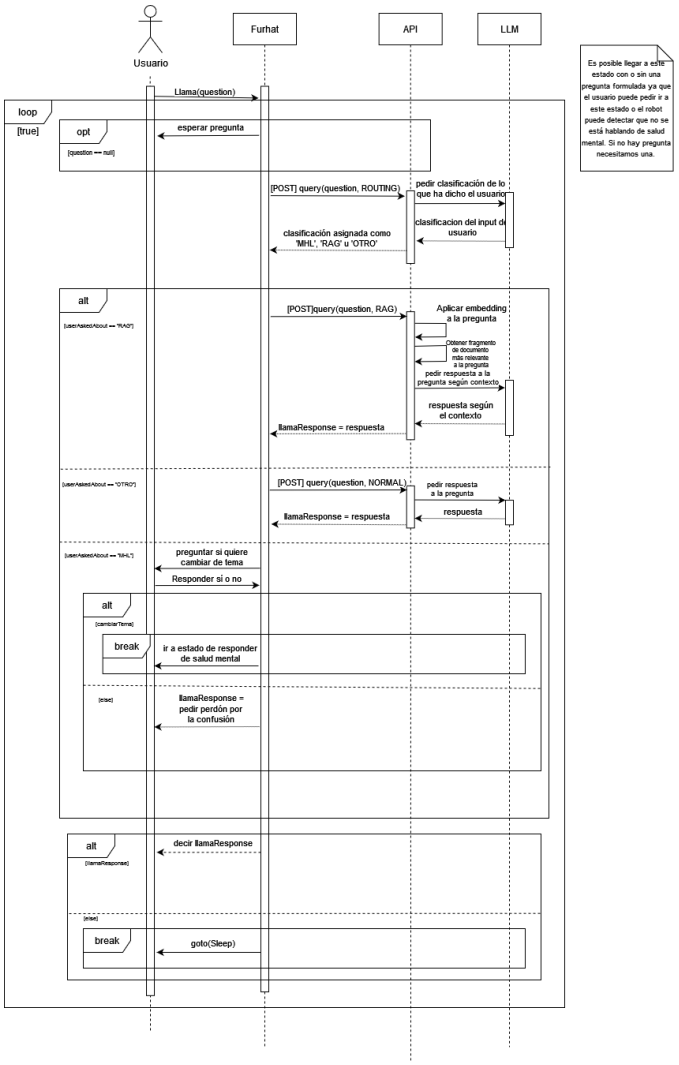


Figura 3.6: Diagrama de secuencia del estado que hace uso de LLM

Si se entró rechazando la pregunta del sistema que se dice en la bienvenida, el robot espera a que se le formule una pregunta. En cambio, si se accede a este estado por medio de *QuestionMHL*, la pregunta es procesada

automáticamente gracias al enrutamiento el cual decidió que, estando en la situación de responder preguntas de salud mental, la pregunta debería ser procesada por otro de los modelos de interacción.

3.6. Implementación

3.6.1. Tecnologías usadas

A continuación se detallarán las distintas tecnologías utilizadas para el desarrollo del proyecto, estas tecnologías fueron elegidas entre otras posibles opciones.

Chatbot: FurhatSDK

Este SDK está basado en Kotlin y es utilizado para conectar y crear la interacción del robot Furhat. Tiene un funcionamiento muy específico y está todo lo necesario para crear una interacción (llamada *skill*) detallado en su documentación. Con este SDK la interacción fluye entre estados en los que dependiendo de lo que dice el usuario el sistema actúa de una manera u otra, ya sea llamando a otro estado, respondiendo con frases y gestos o usando eventos [42].

El motivo por el que se ha escogido usar esta tecnología en lugar de por ejemplo Rasa es para probar qué tal funciona el SDK en sí.

Backend: Python-Flask

Se ha decidido utilizar Python con el framework Flask para la creación del backend. Este backend es utilizado para gestionar los datos almacenados en la base de datos que necesita utilizar el chatbot, es decir, para gestionar las preguntas y los trastornos que conoce el bot. También se usa por la posibilidad de usar LangChain, framework que facilita el desarrollo de aplicaciones que hacen uso de modelos de lenguaje grande (LLM). Como el objetivo es utilizar LLM para ciertas partes del chatbot, Python es la mejor opción ya que dispone de bibliotecas especializadas para el tratamiento de LLM con gran comunidad como pueden ser LangChain y LlamaIndex.

Flask es ligero, simple y minimalista con una gran comunidad activa y documentación actualizada. Al ser simple es sencillo de aprender y desarrollar rápidamente una aplicación web. Tiene muy buen rendimiento para proyectos de esta envergadura.

Otro posible framework es Django pero la aplicación realizada no es tan grande y compleja como para necesitar un framework más potente y mucho

más pesado que Flask.

Base de datos: MongoDB

Para la base de datos se ha optado por MongoDB. Es una base de datos NoSQL la cual guarda los datos en documentos BSON. Presenta bastantes ventajas como su soporte para JSON, sus consultas rápidas y su simpleza para su uso web en una API que trata con datos de tipo JSON.[43] Además, la familiaridad del desarrollador con esta tecnología fue un punto muy a favor para su uso.

Al principio se hacía uso de archivos JSON simulando una base de datos por lo que escalar esa funcionalidad e integrar MongoDB a partir del trabajo previo realizado fue una tarea más sencilla.

Frontend: Angular

Angular es un framework utilizado para desarrollar aplicaciones web mantenido por Google. Está desarrollado en Typescript. Tiene una arquitectura basada en componentes lo cual hace que sea una labor sencilla el mantenimiento y la reutilización de código, manteniendo un desarrollo modular. Es un framework eficiente y con una gran comunidad y bibliotecas disponibles para su uso. La documentación es muy extensa y actualizada. También tiene un CLI (Command Line Interface) que hace que crear nuevos componentes y construir el proyecto en sí sea más fácil de llevar.[44]

Finalmente, también se optó por este framework ya que tiene disponible para su uso la biblioteca Angular Material la cual sigue las especificaciones de Material Design de Google. Ayuda a tener unos componentes consistentes, personalizables y agradables a la vista [45].

Control de versiones: GitHub

GitHub ofrece la posibilidad de alojar proyectos software. Permite entre otras funcionalidades ver y gestionar todos los cambios que se han ido incorporando al código y tenerlo siempre disponible en la nube [46].

Todo el sistema se ha ido subiendo a GitHub progresivamente conforme se ha ido avanzando. Tanto el código del robot, del backend y del frontend se fueron trabajando con la ayuda de GitHub.

Docker

Docker es una plataforma de software la cual permite crear, desplegar, ejecutar y gestionar contenedores. Un contenedor es un componente que contiene todo lo necesario para ejecutar un código en un entorno cualquiera de manera aislada [47].

Ha sido utilizado para desplegar un servicio de base de datos MongoDB en un contenedor. De esta manera simplemente creando y ejecutando el contenedor se realiza toda la configuración de la base de datos automáticamente. Así se facilita la portabilidad, escalabilidad y replicabilidad del entorno de desarrollo.

LLM: Llama3

Hay muchos modelos disponibles y se ha elegido de entre todos ellos a Llama3. Es un modelo reciente y abierto con el que se han obtenido buenos resultados en otras tareas del grupo de investigación. Es superior a su antecesor Llama2 en distintos benchmark y ha sido entrenado con 15 billones de tokens de datos hasta el año 2023, lo que le confiere un conocimiento considerablemente actualizado [48].

3.6.2. Implementación del chatbot

En el siguiente apartado se detallará el código del chatbot, su estructura y algunos aspectos interesantes a destacar.

El código se encuentra divido en varias partes.

Dentro de la carpeta *flow* se encuentra la interacción del robot en su totalidad. Es decir, es posible observar aquí todos los estados a los que el robot puede acceder durante una interacción.

Dentro de *models* están definidos los modelos de datos de los objetos recuperados del backend al ejecutar la aplicación.

La carpeta *nlu* (Natural Language Understanding) presenta todas las entidades y los *intents*. Los *intent* son ejemplos de lo que un usuario podría decir al robot y las entidades representan un objeto o un concepto de la interacción (se pueden usar como sinónimos de palabras y también dentro de los *intent* para darles más sentido). Aquí está definido todo lo que el robot puede esperar que el usuario diga.

En *settings* están todas las funciones útiles, los gestos personalizados y los parámetros configurables del robot.

Finalmente, el archivo *main.kt* es el lugar donde está definida la clase

main de la *skill*. Es la función que comienza a ejecutar la *skill* la cual ejecuta el primer estado definido en la función *start()* que es *Init*. Una instancia de *Flow* es creada y es la encargada de llevar el flujo entero de la interacción.

```
class MentalhealthliteracyfurhatSkill : Skill() {
    override fun start() {
        Flow().run(Init)
    }
}

fun main(args: Array<String>) {
    Skill.main(args)
}
```

El estado init inicializa todos los parámetros necesarios de la interacción que hacen falta al comienzo y dependiendo si hay gente delante comienza el flujo con el usuario o se va a «dormir» a la espera de que alguien aparezca en el rango de visión. También este estado se encarga de precargar todos los *intents* que necesitan información de la base de datos, de esta manera cuando hacen falta ya están almacenados en el conocimiento del robot y puede entender y responder correctamente acerca de esos temas. Finalmente realiza la parte más lenta del programa que es ejecutar el endpoint init perteneciente al RAG. Este endpoint es detallado en la implementación del backend.

En adición, en este estado también se comienza la ejecución paralela para el control de los LEDs del robot. Los LEDs se iluminarán verde cuando el usuario pueda hablar y serán rojos cuando el robot esté hablando, haciendo que el flujo de la conversación y el turno de palabra sea más fácil de entender.

La mayor parte de los estados heredan de un estado padre que se encuentra detallado en *parent.kt*. Este estado padre proporciona diferentes funciones para que los estados actúen de manera consistente. Por ejemplo, en este estado está especificado qué hacer cuando ocurren eventos tales como cuando un usuario se sale del área de conversación, cuando entra o cuando no recibe input (una frase) por parte del usuario al cabo de un tiempo. También se añaden las interacciones comunes que tienen sentido que estén presentes en todos los estados ya que son algo que se puede preguntar en cualquier instante. Esto incluye respuestas a ciertas preguntas como por ejemplo «cuál es tu género» ya que es una duda que puede surgir en cualquier momento de la conversación o reacciones a insultos. Ante los insultos reacciona con sorpresa y tristeza y si es insultado repedidamente terminará la interacción, yéndose a dormir. Para el género ofrece la oportunidad de cambiar su voz o apariencia (por defecto es voz y apariencia femenina) a la del sexo opuesto.

Una vez se entra en el primer estado después del estado *Init* se ejecutan los fragmentos de código (handlers) en el siguiente orden:

init: Este handler se ejecuta una vez por programa. Una vez ejecutado no se vuelve a ejecutar aunque se vuelva a entrar en el estado.

OnEntry: Al entrar en el estado se ejecuta lo que esté dentro.

OnReentry: Si se vuelve a entrar al estado sin salir de él y este handler está definido se ejecutará en lugar de *OnEntry*, en caso de no estar definido se ejecuta de nuevo *OnEntry*.

OnResponse: Cuando el usuario habla el robot decide qué *OnResponse* se ejecuta dependiendo de lo que haya dicho el usuario. Cada uno de estos handlers tiene uno o más *intents* asociados (o ninguno para la respuesta por defecto) y elige el que encuentre que supera cierto índice de confianza. Siguen la sintaxis OnResponse¡Intent¿donde Intent es la lista de frases con la configuración que puede decir el usuario para activar el handler, para la respuesta por defecto se omite «Intent», dejando OnResponse sólo.

OnExit: Cuando se va a salir del estado se ejecuta esta parte del código.

También hay más handlers que se ejecutan al cabo de cierto tiempo (*OnTime*), cuando ocurre un evento (*OnEvent*)... Con todos estos handlers se puede crear eficazmente una conversación con sentido.

Conociendo los handlers se puede explicar el funcionamiento interno más detalladamente. En un curso normal de acción para la interacción por reglas se hacen llamadas al backend usando *init* si procede, luego, en *OnEntry* y *OnReentry* se hace que hable el robot introduciendo que va a comenzar a escuchar y espera a que un usuario hable. En caso de no hablar tras un tiempo determinado el handler *OnNoResponse* se activa, en cambio, si habla se activa el handler *OnResponse* correspondiente. Si ninguno de los handlers definidos supera el índice de confianza se utiliza la respuesta por defecto. Al final de cada handler se puede añadir la función *reentry* para volver al principio del estado, activando *onReentry* o se puede llamar a otro estado. Las respuestas a los mensajes del usuario pueden usar eventos en común y llamarlos con raise(Evento) donde «Evento» es el nombre del evento a utilizar.

Para la interacción con el LLM, la respuesta del usuario siempre activa el handler *OnResponse* por defecto. Éste activa el evento de respuesta que hace uso del LLM que realiza la llamada de enrutamiento para conocer de qué se está hablando y actúa en consecuencia de ello cambiando al uso de la interacción por reglas o usando los endpoints de LLM con o sin RAG, después vuelve a reentrar al estado para escuchar la próxima pregunta.

Los gestos también son una parte fundamental en un chatbot para lograr tener una conversación más humana con elementos no verbales. Existe una

biblioteca con gestos predefinidos o se pueden crear gestos propios. Para crear gestos propios hay que definir la duración del gesto y las partes de la cara que se quieren mover, con qué intensidad y en qué momento en el tiempo dentro del intervalo. Hay un gran número de parámetros para modificar ya sea en los ojos, las cejas, la boca, el cuello... Por ejemplo, se ha creado para los accesos a las llamadas a la API que tardan más tiempo un gesto de pensar, para que el robot parezca más humano y no se mantenga mirando a la nada varios segundos mientras procesa una respuesta el backend. Los gestos predefinidos son también interesantes ya que no hay que preocuparse por crear un gesto que exprese tristeza o alegría puesto que hay varios ya creados para esas emociones.

Tal y como se ha mencionado previamente, el color de los LEDs se controla paralelamente pero no es el único proceso controlado en paralelo. En el momento que el robot habla sobre salud mental se encarga otro proceso paralelamente de ir cambiando la atención entre los usuarios, mirándolos y aparentando que está hablando a todo el mundo. Tiene una probabilidad de girar toda la cabeza, quedarse mirando y atender a un nuevo usuario aleatorio o simplemente echarle un vistazo con los ojos. De esta manera es más natural y humano y no está fijamente mirando sólo al ususario que formula la pregunta. Tanto con los gestos, el color de los LEDs y los turnos de atención se consigue una comunicación no verbal útil que ayuda al usuario a entender lo que ocurre de mejor manera.

Para la comunicación con el backend se ha utilizado OkHttp, creando una función que se encarga de preparar el body si hace falta, hacer petición GET, POST, dependiendo de lo que haga falta y todo lo necesario para su correcto funcionamiento.

Limitaciones

Gran parte de los problemas enfrentados y superados en la implementación del chatbot fueron debido a la escasa comunidad que tiene FurhatSDK. Se depende en su totalidad solo de la documentación oficial y de algunos ejemplos subidos a GitHub por los mismos creadores. No se ha encontrado ni una comunidad activa ni foros de dudas por ejemplo.

Un ejemplo de estos problemas fue la utilización de sinónimos para entidades más complejas. De normal una entidad tiene un valor *string* asociado que es su «significado». Con otro tipo de entidad llamado *GenericEnumEntity*, el robot es capaz de almacenar como significado más valores y no solo de tipo *string*. Es decir, se puede tener guardado un equipo de fútbol y tener guardado además en la misma entidad la cantidad de copas que tiene (un entero), sus jugadores (una lista de *string*), etc. Esta función, al menos hasta donde se ha comprobado, no soporta sinónimos por lo que el usuario

debería decir el nombre de la entidad entera (o muy parecida) para que el robot la entienda. Con sinónimos se permite referir a una entidad de muchas maneras pero con esta entidad simplemente no se ha encontrado la posibilidad. Por ejemplo, se podría crear una entidad y poner de ejemplo «la selección española» y como sinónimo «La Roja» y el robot entendería que se está hablando de lo mismo a pesar de ser una palabra muy distina. Con este tipo de entidad no es posible hacer de ese tipo de función pero se gana la posibilidad de almacenar más datos.

Otro problema es el uso de *stemming*. Esta función hace que el robot sea capaz de identificar palabras similares como iguales. Es decir, el robot es capaz de entender palabras como por ejemplo cambio - cambios - cambiante como palabras que significan lo mismo pero esta función no existe en español. La solución inmediata fue añadir directamente todas las palabras necesarias como sinónimos, aumentando de esta manera el vocabulario y conocimiento del robot.

Otro detalle encontrado es el orden de las respuestas. *OnResponse* es el fragmento del código que se ejecuta al escuchar cierto *intent* por parte del usuario. Estas respuestas se comprueban por orden y la primera que supere el índice de confianza es la que se ejecuta. Tal vez sea más lento comprobar cuál tiene la respuesta más alta pero es la baza más segura para dar la respuesta más correcta a lo que dice el usuario si hay dos respuestas que podrían ser válidas.

Otro apunte a destacar son los *intents* predefinidos. Están disponibles en una gran variedad de idiomas pero, dependiendo del lenguaje que use el robot, son más o menos los ejemplos usados. Un ejemplo es el *intent* de despedida, llamado «Goodbye». En dicho *intent* mirando la librería que lo implementa se contempla que consiste de la siguiente lista:

```
1     adios
2     hasta luego
3     chao
4     me despido
5     hasta pronto
6     nos vemos
```

Estos ejemplos claramente son insuficientes para una lengua tan rica como el español, en la que una misma palabra (adiós), se puede decir de una infinidad de maneras distintas. Por tanto, todos los *intents* precargados han sido ignorados y se hicieron unos más extensos (añadiendo por ejemplo en este caso frases como «me tengo que ir» o «hasta la próxima»).

Con el problema anterior es posible ahora explicar un nuevo problema. Hay una manera de hacer que el robot acepte solo respuestas de sí o no y almacenar la respuesta en un booleano. Este handler se llama *askYN* y utiliza los intents preexistentes «Yes» y «No». El intent «Yes» tiene bastantes

ejemplos pero «No» tan solo presenta en español cinco maneras de negarse. No tiene en cuenta si se dice «qué va», «negativo», «en absoluto»... Esto hace que la respuesta dependiendo de la negación o afirmación del usuario falle aunque sea en la lengua usada algo válido.

Otro pequeño problema enfrentado ha sido la recuperación de contexto. Si un evento ocurre mientras el robot está hablando se puede parar su habla si es necesario y retomarla cuando se considere oportuno. Hasta este punto está bien, el problema es que el robot puede estar escuchando y no hablando. Es posible parar de escuchar pero no es posible retomar la escucha, se pierde el punto de la conversación por donde iba el robot. La solución encontrada ha sido volver a entrar al estado donde se hizo la escucha hablando de manera natural y humana, como si el robot hubiese perdido el hilo de la conversación y sea necesario empezar por el último estado en el que estaba.

3.6.3. Implementación del backend

El propósito de esta API es actuar como el conocimiento del chatbot acerca de diferentes temas. Todo lo que sabe el chatbot acerca de trastornos (como por ejemplo depresión o ansiedad) está almacenado aquí, además de las preguntas que es capaz de formular con sus respuestas y explicación. Adicionalmente también se utiliza para conectar con el LLM y poder hacer uso de la parte del RAG y de otros tipos de peticiones.

El backend contiene las siguientes clases, a las cuales el chatbot accede a ellas mediante métodos CRUD si es necesario.

- **User:** Esta clase tiene un campo email y otro password la cual se almacena encriptada para mayor seguridad. Esta clase se utiliza para obtener un JWT con el que poder acceder al resto de métodos de la aplicación.

  Esta clase solo tiene un único endpoint el cual es usado para iniciar sesión:

  POST /user/login Genera un JWT para ganar acceso al resto de endpoints que están protegidos.

- **Disorder:** Esta clase contiene los campos id, causes, disorder, symptoms, synonyms, treatment. Todos los datos son usados por el chatbot para ganar el «conocimiento» acerca del susodicho trastorno.

  Disorder hace uso de todos los métodos CRUD con su debido manejo de errores:

  GET /disorders Devuelve una lista con todos los campos de cada objeto *disorder*.

GET /disorders/disorder_id Devuelve el objeto *disorder* al que le corresponde el id dado en *disorder_id*.

POST /disorders Devuelve el objeto creado con este método usando los valores proporcionados en el esquema.

DELETE /disorders/disorder_id Borra de la base de datos el objeto cuyo id corresponde a *disorder_id* si existe.

PUT /disorders/disorder_id Devuelve el objeto previamente existente cuyo id es *disorder_id* modificado según los parámetros del esquema.

GET /disorders/furhat Devuelve la lista completa de trastornos en un formato que el chatbot puede entender sin necesidad de procesar la salida del endpoint de la lista. Este endpoint no tiene uso útil ya que como se explicó previamente, *GenericEnumEntity* no acepta sinónimos.

- **Question:** Esta clase presenta los campos question, answer y explanation. Con esta información el chatbot es capaz de obtener la información y lanzar preguntas al usuario, saber si son correctas y explicar algo más acerca de la pregunta.

  Question presenta todos los métodos CRUD. Todos tienen manejo de errores:

  GET /questions Devuelve una lista con todos los campos de cada objeto *question*.

  GET /questions/question_id Devuelve el objeto *question* al que le corresponde el id dado en *question_id*.

  POST /questions Devuelve el objeto creado con este método usando los valores proporcionados en el esquema.

  DELETE /questions/question_id Borra de la base de datos el objeto cuyo id corresponde a *question_id* si existe.

  PUT /questions/question_id Devuelve el objeto previamente existente cuyo id es *question_id* modificado según los parámetros del esquema.

Fuera de las clases se presentan otros endpoints que hacen uso de los modelos de lenguaje grande (LLM):

POST /rag Recibe como parámetro de entrada un *string* (que es la petición del usuario) y devuelve como salida otro *string* que es la respuesta que da el LLM que se ha usado. Usa un contexto que son varios documentos obtenidos del gabinete psicopedagógico de la UGR [37], siendo

capaz de dar información actualizada a universitarios para estudiar mejor y no sufrir estrés o ansiedad con los exámenes o la vida universitaria. Corresponde a la última parte del RAG, conocida como querying. Este endpoint da error si no se ha ejecutado previamente /init

POST /init Inicializa todo lo necesario para que el endpoint /rag pueda funcionar correctamente. El código que se ejecuta en este endpoint puede estar en /rag pero no es óptimo ejecutarlo constantemente. Teniéndolo separado se logra mayor eficiencia en las respuestas. Realiza las partes del RAG conocidas como Loading, Splitting y Storing las cuales fueron explicadas previamente. Es decir, se obtiene tras su ejecución una base de datos vectorial de los documentos que hacen de contexto.

POST /normal_query Recibe la petición del usuario en *string* y devuelve una respuesta. Se comporta de la misma manera que sería mandar un mensaje normal a un LLM en la web.

POST /routing Recibe una petición del usuario en *string* y devuelve 'MHL', 'RAG' u 'OTRO'. Este endpoint es utilizado para determinar de qué está hablando el usuario. El LLM clasifica la petición del usuario para saber si está hablando del tema principal que está definido por reglas (MHL), de temas que necesita contexto para responder haciendo uso de /rag (RAG) o de cualquier otra cosa que no necesita contexto, que corresponde al endpoint /normal_query (OTRO). Con este endpoint es posible cambiar entre uno de los tres modelos usados sin que el usuario tenga que pedir explícitamente que se cambie el modelo usado. Para evitar posibles confusiones en la respuesta del modelo es interesante considerar que responda palabras completamente ajenas a cualquier conversación que pueda decir. Por ejemplo, una buena idea sería usar el nombre de localidades granadinas, ya que la probabilidad de que diga «Armilla» en otro contexto es bastante baja. Si se usa esa misma localidad para que decida si se está hablando de salud mental en la respuesta del modelo se busca la palabra «Armilla» y si se encuentra se devuelve el nombre 'MHL' para mayor consistencia.

Es importante destacar que se tiene que buscar la palabra, no esperar una coincidencia de que el output sea igual a la palabra. Esto se debe a la inconsistencia de los modelos en los que, aunque les digas que respondan con solo una palabra, a veces pueden fallar y decir por ejemplo «Armilla porque se está hablando de salud mental».

Para lograr el cometido de estos endpoints se ha hecho uso del framework LangChain. Gracias a esto es posible también preparar un RAG con varias líneas de código.

Lo esencial es tener un *loader* para cargar los archivos que se quieren utilizar como contexto, una herramienta para la división de los archivos en trozos más pequeños, la base de datos vectorizada y un *retriever*. El *retriever* se encarga del tipo de búsqueda que hay que hacer a la hora de comparar la pregunta del usuario con la base de datos que actúa como conocimiento y cuántos archivos similares debe encontrar. En el código se le ha marcado que devuelva un archivo (lógicamente el más similar de la lista) y ninguno más ya que con dos o más se puede superar el máximo de tokens permitidos por el modelo LLM.

Después se prepara el modelo con la URL, el máximo de tokens de la respuesta y la temperatura (cómo de creativo puede ser en su respuesta el modelo). Se prepara un prompt siguiendo buenas prácticas como por ejemplo separar las instrucciones con símbolos o siendo lo más descriptivo y detallista posible. Para el prompt se ha hecho uso de la técnica *zero-shot*. Esta estrategia hace saber al modelo qué es lo que tiene que responder exactamente dejando un «espacio en blanco» en el prompt que el sistema vea que tiene que rellenar. Finalmente, es mejor especificar al modelo sus límites y lo que puede hacer que lo que no debe hacer [49].

El siguiente ejemplo es el utilizado para la técnica de RAG. Se pueden observar las distintas técnicas mencionadas. Las partes del prompt *context* y *question* son partes del texto que son completadas más adelante en la ejecución del código. Actúan como variables vacías a las que se les da un valor en el momento que se obtenga:

```
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            """Usando la informacion que se encuentra en el contexto,
                da una respuesta razonable a la pregunta. Responde a
                la pregunta sin dar informacion extra, se conciso y
                claro. Eres un robot amigable que proporciona
                informacion de manera informal a estudiantes de la
                universidad para solventar problemas de ansiedad en
                los examenes y tecnicas de estudio. Comienza la
                respuesta con frases del estilo, <<segun mis datos>>.

            Si no puedes dar una respuesta a partir del contexto, di
                que no conoces la respuesta.""",
        ),
        (
            "user",
            """Contexto:
            {context}
            ---
            Aqui esta la pregunta que tienes que responder.

            Pregunta: {question}

            Respuesta:
            """,
        ),
    ]
)
```

Se ve que está explicado con detalle su rol en la parte *system* y en *user* se entiende que es el input que recibe por parte del usuario. El hecho de que «Respuesta: » no tenga nada es la técnica *zero-shot*. El modelo ve al final ese campo vacío y se encarga de rellenarlo. De igual forma se pueden apreciar distintas separaciones en el texto como por ejemplo tres guiones una vez dado el contexto, dando a entender que el contexto ha acabado. Este modelo si no recibe la parte «Comienza la respuesta con frases del estilo, "según mis datos".» responde mencionando el contexto con frases como «el contexto proporcionado menciona...» lo cual no tiene sentido decir a un usuario que haga uso del robot y que no tiene conocimiento de un contexto suministrado en el momento. Si se le suministra en lugar de esa frase una en negativo, «no menciones que tienes contexto» hay ocasiones en las que no es consistente. Es por ello que la frase que está en el prompt actual tiene más valor que una negativa o directamente nada.

Al ser estos modelos no deterministas, lograr respuestas consistentes no es tarea sencilla y es necesario probar distintos prompts con contenidos variados.

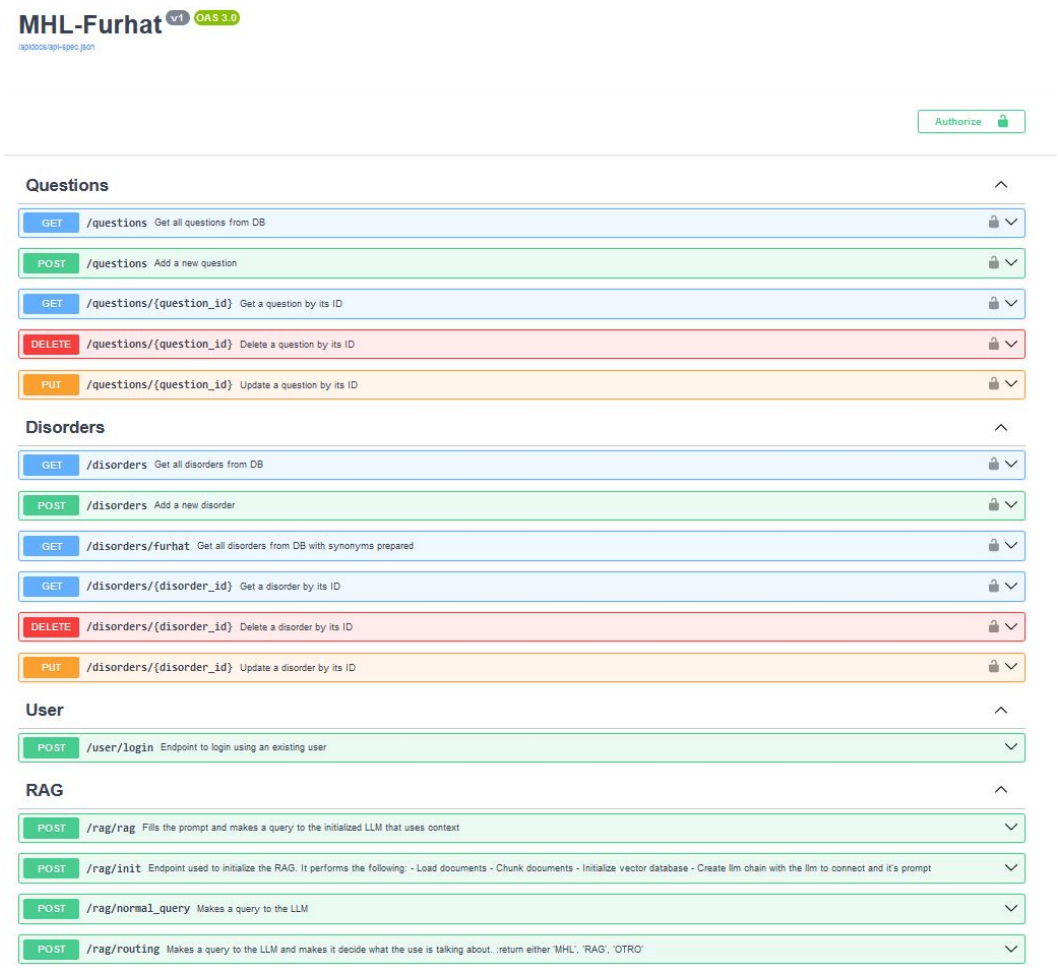


Figura 3.7: Endpoints del backend en Swagger

Para la conexión con el LLM se utiliza un modelo facilitado por la tutora. Sin embargo, también se ha investigado cómo ejecutar un modelo localmente. Es posible utilizar Ollama para ello. Consiste en una herramienta que permite ejecutar modelos de lenguaje grandes desde la máquina local, instalándolos y ejecutándolos con simples líneas de comandos. Está disponible tanto para Windows, Linux y MacOS. El problema de esta solución es que se necesita un computador muy potente para evitar respuestas lentas y cuello de botella.

Para tener un modelo descargado localmente:

```
$ ollama serve
$ ollama pull <modelo_a_descargar>
```

Usando *ollama serve* se hace que comience la aplicación y con *ollama pull* se obtiene el modelo deseado en el ordenador. Ahora solo falta usar la biblioteca de LangChain de Ollama en lugar de usar la función ChatOpenAI. En el código sería sustituir este fragmento:

```
llm = ChatOpenAI(
    temperature=0,
    openai_api_base=f"http://{MODEL_URL}/v1",
    max_tokens=1000,
    streaming=False
)
```

Por ejemplo por el siguiente:

```
llm = Ollama(
    model="llama2"
)
```

Si se hizo uso de *ollama pull llama2* el código debe funcionar pero con el modelo local en lugar de un modelo externo al ordenador personal donde se ejecuta el código. Esta opción fue descartada para el proyecto ya que con el portátil en posesión no es posible tener tiempos de respuesta satisfactorios para un chatbot.

3.6.4. Implementación del frontend

Este frontend ha sido implementado para poder gestionar las preguntas y los trastornos con una interfaz simple e intuitiva. Así, cualquier profesional de la salud mental puede gestionar los datos con los que trata el robot.

Tal y como se detalló previamente, para la implementación del frontend se ha utilizado Angular. Al realizar el comando *ng run*, el archivo *main.ts* arranca una estancia de *AppComponent* y la inicializa como el componente raíz. A partir de *AppComponent* se cargan **dinámicamente** los componentes que hacen falta en cada momento definidos en *app.routes.ts*. Dependiendo de la ruta en la que se encuentre el usuario, Angular sabe qué componentes hacen falta. Un servicio es utilizado, entre otros, para la comunicación con el backend y el componente muestra la información y la usa.

Se hace uso del patrón de diseño inyección de dependencias en los servicios creados en Angular, con esto se logra modularidad y reusabilidad. Los servicios pueden ser encontrados en la carpeta *services* y los componentes en distintas carpetas (*questions, login, home, disorders*). Los modelos de los objetos utilizados en el proyecto se ubican en *models* donde están definidos los contenidos de cada objeto y una clase que actúa como lista.

Se han protegido los distintos accesos a cada componente en caso de no

tener la sesión iniciada, redirigiendo siempre a la pantalla de inicio de sesión e impidiendo la visualización de nada más a no ser que se esté logueado. Se ha conseguido esta funcionalidad con el uso de un *guard*. Un *guard* se utiliza para controlar el acceso del usuario a distintas partes de la aplicación, este en concreto comprueba que se tiene un token guardado y listo para ser usado.

Además, es necesario tener en cuenta que no sólo los componentes del frontend están protegidos, también están protegidos los endpoints del backend por tanto es necesario crear un interceptor para que a las llamadas al backend se le añada un header con la sintaxis necesaria del token. El token se almacena en el almacenamiento local, se obtiene de ahí y se le da el formato necesario para enviarlo en la cabecera de la petición al backend.

Para presentar la información de manera responsive y atractiva se ha usado SCSS y Angular Material. SCSS presenta algunas diferencias a CSS como *nesting* que hace más fácil de leer el archivo o variables con las que mantener por ejemplo colores de manera consistente en todo el proyecto. Angular Material ofrece una lista grande de componentes responsive y consistentes, además de muy personalizables.

La comunicación con el backend se realiza en los servicios. Dentro de cada servicio se inicializa otro servicio de Angular llamado HttpClient que proporciona métodos especiales para realizar peticiones al backend fácilmente. Cada una de estas llamadas hacen captura de errores y se conectan al servicio de notificaciones para avisar al usuario de que algo ha ido mal, indicando el error pertinente. No todos los servicios comunican con el backend, por ejemplo, existe el servicio de notificaciones, que es utilizado para crear *snackbars* que muestran información relevante cuando sea necesario.

3.7. Despliegue

Para desplegar el código en el robot de Furhat es necesario crear un archivo *.skill* y añadirlo al catálogo de skills del robot. Para lograr este cometido tanto el robot como el ordenador donde está la skill tienen que estar conectados a la misma red. El siguiente paso sería generar el archivo. Usando el IDE IntelliJ IDEA hay que navegar en la pestaña *Gradle* a la parte shadowJar y ejecutarlo. Esto generará el archivo *.skill*. Todo esto está relatado con más detalle en el apéndice A.1.1

3.8. Evaluación

En esta sección se van a mostrar distintos flujos de conversación que es posible tener con el robot, tanto la parte RAG, como las reglas y las queries sin contexto. Es posible crear logs de las conversaciones usando una clase llamada dialogLogger. Con estos logs se aprecia que es posible mantener conversaciones con el robot según lo deseado pero estos logs contienen demasiada información (momento exacto que se empezó a hablar, cuando paró de hablar, estado, si se ha guardado la pregunta como audio en el ordenador, etc.) y son difíciles de visualizar como una conversación, por tanto, se ha optado por mostrar la interfaz gráfica que es más fácil de entender.

Este primer ejemplo es una muestra de la parte de salud mental, es decir, la basada en reglas.

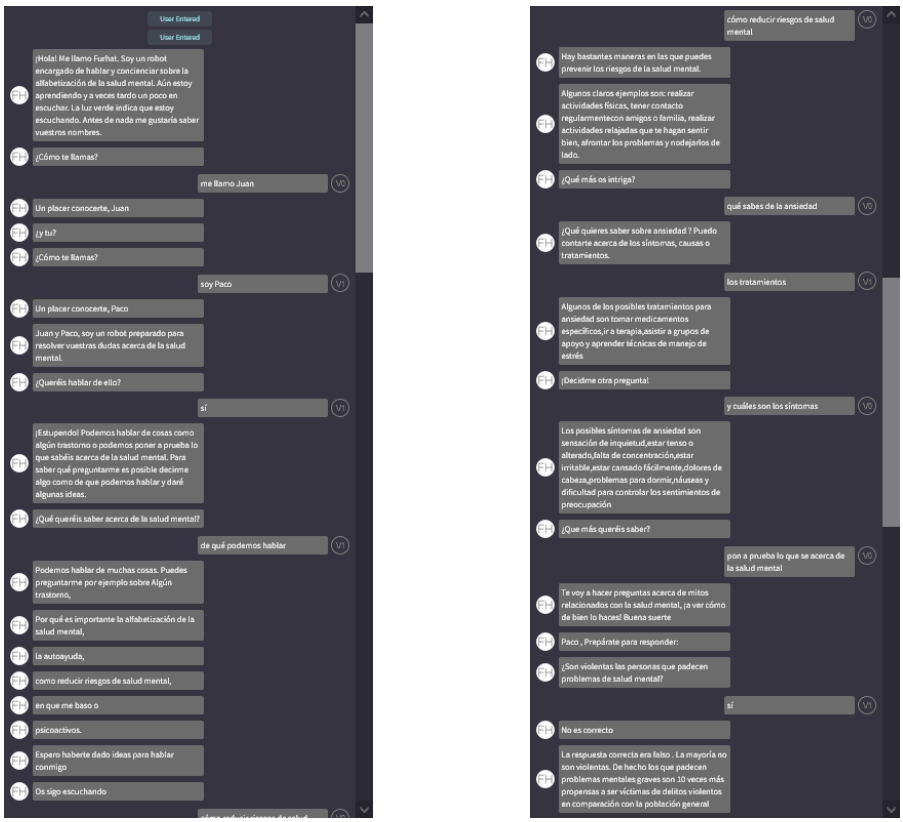


Figura 3.8: Interacción con chatbot por reglas

En estas dos primeras imágenes 3.8 se puede observar como se pasa del estado de presentación al de preguntas de salud mental al aceptar los usuarios hablar del tema. Ahí dentro el chatbot guía dando consejos de qué preguntar y responde correctamente según lo que se le pregunte. También

se nota que mantiene el contexto sin olvidar los nombres de cada usuario y recordando que se realizó una pregunta acerca del trastorno ansiedad, por lo que es posible preguntarle «y cuáles son los síntomas» sin necesidad de especificar el trastorno porque sabe que se habló de uno previamente.

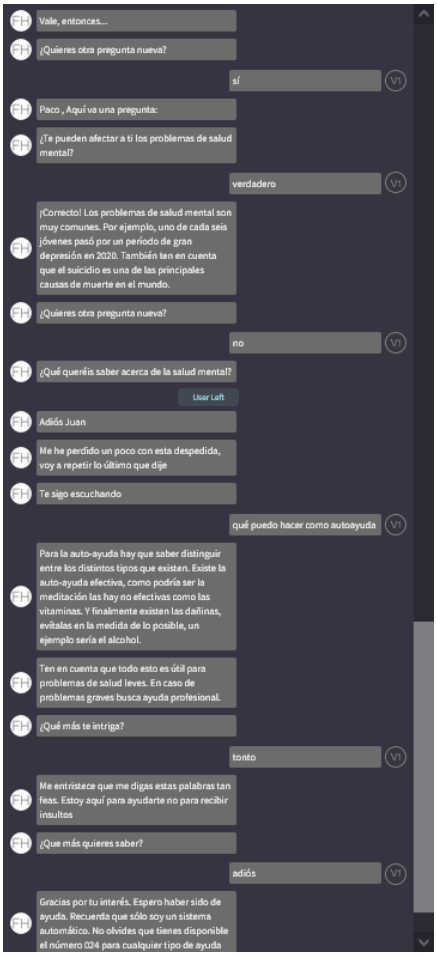
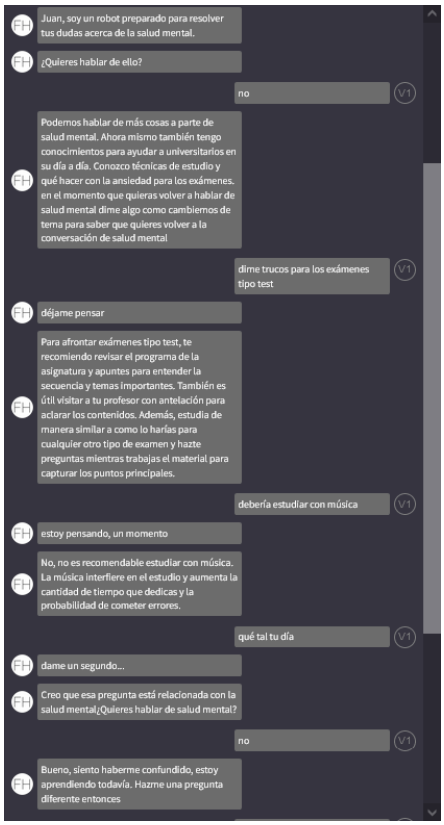


Figura 3.9: Final de la interacción del chatbot por reglas

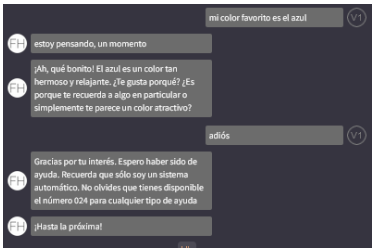
En esta última imagen 3.9 lo más interesante que se aprecia son algunas de las utilidades del estado padre del que heredan la mayoría de estados. En un momento de la comunicación uno de los usuarios abandona el área de interacción y Furhat detiene la conversación para despedirse y vuelve a referirse al otro usuario hablando ya en singular ya que sólo queda él. También se ve que detecta algunos insultos y reacciona negativamente a ellos y que es posible despedirse para terminar la conversación.

Las siguientes imágenes enseñan un ejemplo en el que no se está de acuerdo con que se quiera hablar de salud mental, lo que hace que el robot

vaya al estado de llamadas al LLM, permitiendo tener conversaciones más amplias:



(a) Interacción haciendo uso de RAG



(b) Llamada al LLM que no hace uso de RAG

Figura 3.10: Interacción con chatbot con llamadas a LLM

En la imagen que hace uso de RAG 3.10a es posible ver que responde según un contexto y todo sobre el tema para afrontar mejor los estudios y no tener estrés ni ansiedad durante el curso. Adicionalmente, se observa la técnica de enrutamiento que tiene lugar durante toda la interacción con el robot, tanto si se habla de salud mental como no, detectando si tiene que usar el sistema por reglas, el RAG o el prompt normal. En la otra imagen 3.10b se observa que se puede hablar de cualquier tema con el robot, siempre va a tener una respuesta disponible aunque no tenga relación con ningún tema. La variedad del lenguaje juega un rol importante aquí, siempre que se pone a pensar elige una frase aleatoria que generalmente es distinta a la utilizada previamente.

Para poder probar cada estado rápidamente se definieron botones que van a los estados más importantes para agilizar el proceso. Estos botones en FurhatSDK son un estado parcial y aparecen en la interfaz web de Furhat, disponible en http://localhost:8080/ y sirven para depurar con mayor agilidad. Para depurar por ejemplo un gesto:

```
1    onButton("Doubt Gesture", color = Color.Red, section = Section.
         RIGHT) {
2        furhat.gesture(Doubt)
3    }
```

Para provocar la ejecución de un estado se cambia la línea del gesto por *goto(el_estado_a_ejecutar)*.

Pruebas backend

Es importante asegurar que los endpoints funcionan como se desea. Para depurar y probar todos y cada uno de los endpoints se ha utilizado tanto la herramienta Swagger UI como Postman. El primer endpoint a ejecutar sería *http://localhost:5000/user/login* para poder acceder al resto ya que están protegidos. Si se intenta ejecutar cualquier otro endpoint sin tener el JWT añadido en los headers se recibirá un mensaje de respuesta 401 con el mensaje *"msg": "Missing Authorization Header"*. Una vez se tiene acceso se puede probar a ejecutar cada endpoint que funciona satisfactoriamente, se pueden intentar provocar errores como no incluir un parámetro obligatorio (error 422), intentar usar una ID inexistente (error 404)...

Para hacer uso de Postman es necesario descargarlo y ejecutarlo. Se elige en la pantalla el método HTTP del que trata la petición y se escribe la dirección del endpoint. Es necesario añadir en la pestaña Authorization el tipo Bearer Token y poner el token generado.

Si se quiere usar Swagger este presenta la ventaja que no es necesario conocer los endpoints ya que es una herramienta que sirve también para documentar por lo tanto se pueden ver todos los endpoints disponibles con su descripción, parámetros necesarios, el método HTTP que son y el header se aplica automáticamente sobre todos los endpoints sin necesidad de ser configurado. Con el backend ejecutado se utiliza Swagger desde *http://localhost:5000/apidocs/swagger*. Para usar el token se hace click en el botón Authorize y se escribe: Bearer «el token generado». A continuación se muestra cómo utilizar Swagger:

En primer lugar se obtiene el JWT ejecutando el endpoint del login con las credenciales.

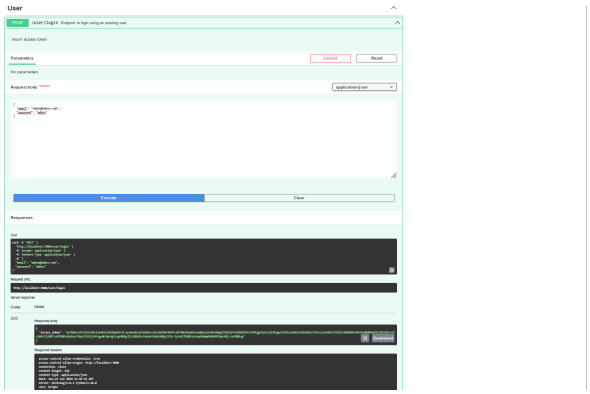


Figura 3.11: Endpoint login

Luego, se añade a todos los headers haciendo click en el botón Authorize y rellenando el campo.

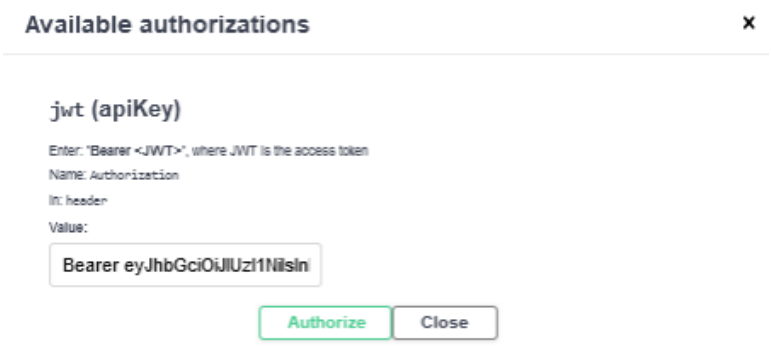


Figura 3.12: Ventana emergente de la autorización

Finalmente, se utiliza el endpoint que se quiera, se observa en la imagen que el endpoint quye hace uso de RAG responde en base a unos datos suministrados previamente. (/init se ejecutó con antelación).

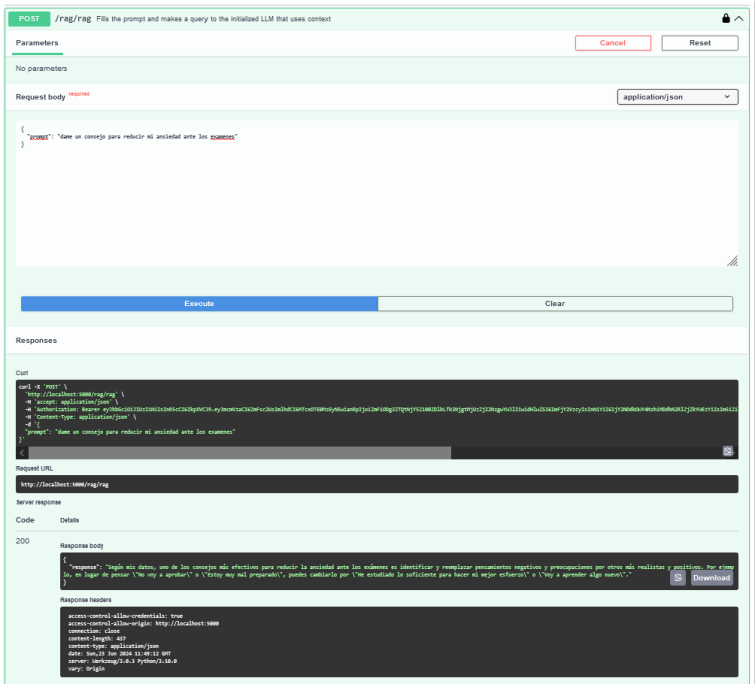


Figura 3.13: Endpoint que hace uso de LLM con RAG

Pruebas frontend

Finalmente, para las pruebas del frontend se probó todo lo que se podía hacer en cada página y se protegieron los botones hasta que cada formulario fuese válido. Si ocurre un problema se muestra mediante una *snackbar* con el código de error. Por ejemplo, en las siguientes imágenes se puede ver que el botón para continuar no está disponible. Esto ocurre en todos los demás formularios.

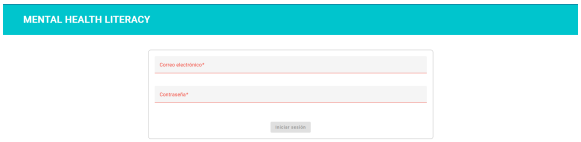


Figura 3.14: Página login de la web

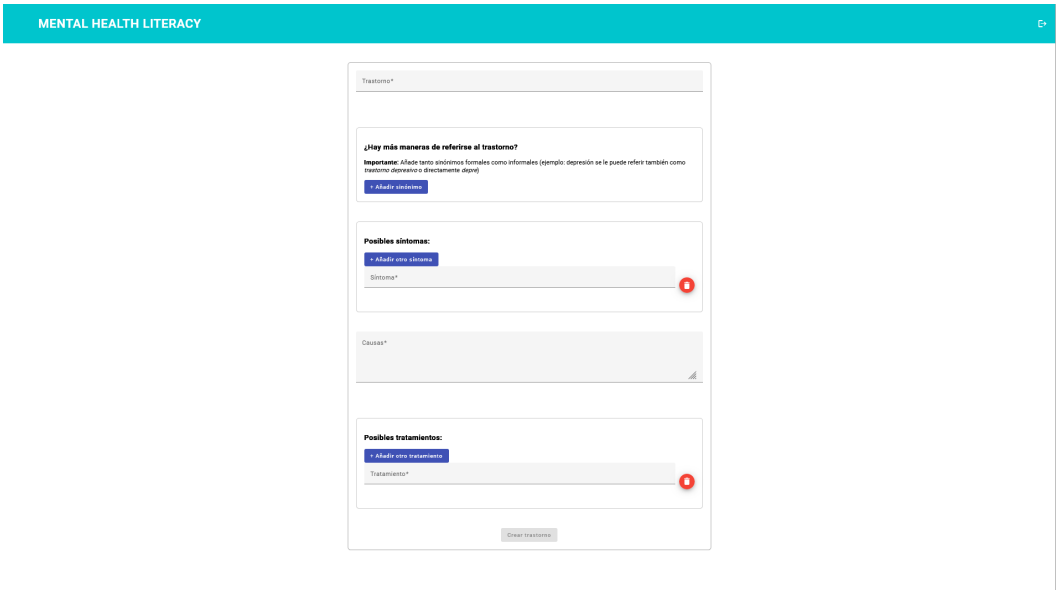


Figura 3.15: Añadir trastorno

En la última imagen 3.15, es posible añadir dinámicamente más información pulsando en los botones de añadir, y también se puede borrar con el botón correspondiente pero no se permite borrar todos ya que son campos obligatorios a excepción de los campos de sinónimos. Esto ha sido probado para que el usuario no provoque errores enviando trastornos no válidos.

Adicionalmente todas las rutas no existentes redirigen a login si no se inició sesión o a home si se tiene una sesión iniciada.

Capítulo 4

Conclusiones

Se ha probado que una sólida alfabetización en salud mental ayuda a mejorar los hábitos que promueven el bienestar psicológico, facilita la búsqueda de ayuda a tiempo y reduce el estigma y la desconfianza acerca de las personas que sufren problemas de salud mental. Por estos motivos, es crucial promover una buena alfabetización mental y los sistemas conversacionales, ya que estos pueden ser de gran ayuda al permitir a los usuarios hacer preguntas con lenguaje natural, usando sus propias palabras sin sentirse juzgados. Adicionalmente pueden ayudar a complementar la falta de personal, sirviendo de material de consulta siempre disponible.

En este TFG, se ha desarrollado un sistema conversacional para la interacción humano-robot que busca la alfabetización en salud mental y que integra diferentes enfoques del paradigma actual de desarrollo de sistemas. El sistema es perfectamente funcional e interactúa con sus distintas partes para lograr un equilibrio entre el control de las respuestas y la flexibilidad de la interacción en función del tema de conversación. Se han cumplimentado todos los objetivos propuestos de manera satisfactoria.

Ha sido muy enriquecedor aprender sobre los distintos enfoques de chatbot e implementarlos todos juntos en un mismo sistema, práctica que no es muy común en el ámbito.

Finalmente se debe recalcar que se han superado una gran cantidad de retos de los cuales se ha podido aprender en gran medida, además de lograr adentrarse en el área de los agentes conversacionales que hacen uso de los grades modelos de lenguaje, un ámbito con una gran proyección actual.

4.1. Trabajo futuro

Es posible realizar incrementos en el trabajo presente, creando funcionalidades nuevas y mejorando las ya existentes.

Una de las primeras mejoras que se puede pensar es mejorar el tiempo de respuesta de las llamadas al LLM. Ya que el servidor es externo y no depende directamente del código implementado, la parte a mejorar podría ser la búsqueda de un modelo de embedding mejor que el utilizado en el código. Esto puede mejorar el tiempo de generación del embedding, la precisión, contenido o tamaño del vector... Para este tema de benchmarking de embeddings resulta de gran interés observar las aportaciones de HuggingFace. Tanto la *leaderboard* como el artículo encontrados en `https://huggingface.co/spaces/mteb/leaderboard` son de bastante interés para encontrar el modelo de embedding que mejor se adapte. [50]

A pesar de que se ha tratado de añadir toda la flexiblidad posible en las entradas y salidas del chatbot, otra posible mejora puede ser añadir más variedad de conversaciones y nuevas interacciones a la parte del chatbot basado en reglas. Así se lograría tener más frases nuevas que decir, dando mayor variedad y unicidad a la conversación. Sería entonces de esta manera difícil tener la misma interacción con las mismas respuestas si se le añaden mucha variedad a su lenguaje, cada usuario recibiría respuestas más únicas.

Mejorar el procesamiento de lenguaje natural añadiendo más entidades, sinónimos e *intents* puede ser interesante para tener un robot por reglas más robusto y fiable, admitiendo mayor variedad de lenguaje que el usuario pueda decir para interactuar con él.

Para terminar, es también posible añadir la posibilidad de que el usuario profesional pueda añadir más conocimiento a parte de preguntas y trastornos a través de la aplicación web. Por ejemplo se podría añadir también psicoactivos o medicamentos, que son partes que no tienen tanto uso actualmente en el chatbot pero que pueden dar cabida a más conversaciones relacionadas con la salud mental.

Todas estas ideas pueden mejorar el proyecto, haciendo que dé lugar a una interacción más fluida, con menos errores posibles y un mayor número de conversaciones.

Bibliografía

[1] World Health Organization. Salud mental: fortalecer nuestra respuesta, 2022. URL https://www.who.int/es/news-room/fact-sheets/detail/mental-health-strengthening-our-response.

[2] A. F. Jorm. Mental health literacy: Public knowledge and beliefs about mental disorders. *British Journal of Psychiatry*, 177(5):396–401, 2000. doi: 10.1192/bjp.177.5.396.

[3] Australian Institute of Health and Welfare. Health literacy, 2022. URL https://www.aihw.gov.au/reports/australias-health/health-literacy.

[4] Patrick W. Corrigan and Amy C. Watson. Understanding the impact of stigma on people with mental illness. *World Psychiatry: Official Journal of the World Psychiatric Association (WPA)*, 1(1):16–20, 2002.

[5] World Health Organization. *Strengthening Health Literacy in Europe: Implementation of the European Health Literacy Survey Project*. Publications of the WHO Regional Office for Europe, Copenhagen, Denmark, 2013. ISBN 9789289000154.

[6] K. Demyttenaere, R. Bruffaerts, J. Posada-Villa, I. Gasquet, V. Kovess, J. P. Lepine, M. C. Angermeyer, S. Bernert, G. de Girolamo, P. Morosini, G. Polidori, T. Kikkawa, N. Kawakami, Y. Ono, T. Takeshima, H. Uda, E. G. Karam, J. A. Fayyad, A. N. Karam, Z. N. Mneimneh, and WHO World Mental Health Survey Consortium. Prevalence, severity, and unmet need for treatment of mental disorders in the world health organization world mental health surveys. *JAMA*, 291(21):2581–2590, 2004. doi: 10.1001/jama.291.21.2581.

[7] Anthony F. Jorm. The concept of mental health literacy. In Orkan Okan, Ullrich Bauer, Diane Levin-Zamir, Paulo Pinheiro, and Kristine Sørensen, editors, *International Handbook of Health Literacy: Research, Practice and Policy Across the Life-span*, pages 53–66. Policy Press, 2019. doi: 10.51952/9781447344520.ch004.

[8] World Health Organization. *Mental Health Atlas 2020*. World Health Organization, Geneva, Switzerland, 2021. URL `https://www.who.int/publications/i/item/9789240036703`.

[9] Marco Solmi, Joaquim Radua, Miriam Olivola, Enrico Croce, Livia Soardo, Gonzalo Salazar de Pablo, Jae Il Shin, James B. Kirkbride, Peter Jones, Jae Han Kim, Jong Yeob Kim, Andrè F. Carvalho, Mary V. Seeman, Christoph U. Correll, and Paolo Fusar-Poli. Age at onset of mental disorders worldwide: large-scale meta-analysis of 192 epidemiological studies. *Molecular Psychiatry*, 27(1):281–295, Jan 2022. ISSN 1476-5578. doi: 10.1038/s41380-021-01161-7. URL `https://doi.org/10.1038/s41380-021-01161-7`.

[10] Zoraida Callejas y David Griol. Dialog systems a perspective from language, logic and computation: A perspective from language, logic and computation. pages 220–224. 01 2021. ISBN 978-3-030-61437-9. doi: 10.1007/978-3-030-61438-6.

[11] Anmol N. Vaidyam, Hannah Wisniewski, John D. Halamka, Madhu S. Kashavan, and John B. Torous. Chatbots and conversational agents in mental health: A review of the psychiatric landscape. *The Canadian Journal of Psychiatry*, 64(7):456–464, 2019. doi: 10.1177/0706743719828977.

[12] G. M. Lucas, A. Rizzo, J. Gratch, S. Scherer, G. Stratou, J. Boberg, and L.-P. Morency. Reporting mental health symptoms: Breaking down barriers to care with virtual human interviewers. *Frontiers in Robotics and AI*, 4:51, 2017. doi: 10.3389/frobt.2017.00051.

[13] Kerstin Denecke, Alaa Abd-Alrazaq, and Mowafa Househ. *Artificial Intelligence for Chatbots in Mental Health: Opportunities and Challenges*, pages 115–128. Springer International Publishing, Cham, 2021. ISBN 978-3-030-67303-1. doi: 10.1007/978-3-030-67303-1_10. URL `https://doi.org/10.1007/978-3-030-67303-1_10`.

[14] Michael McTear. *Rule-Based Dialogue Systems: Architecture, Methods, and Tools*, pages 43–70. Springer International Publishing, Cham, 2021. ISBN 978-3-031-02176-3. doi: 10.1007/978-3-031-02176-3_2. URL `https://doi.org/10.1007/978-3-031-02176-3_2`.

[15] Michael McTear. *End-to-End Neural Dialogue Systems*, pages 125–158. Springer International Publishing, Cham, 2021. ISBN 978-3-031-02176-3. doi: 10.1007/978-3-031-02176-3_5. URL `https://doi.org/10.1007/978-3-031-02176-3_5`.

[16] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau

Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021. URL https://arxiv.org/abs/2005.11401.

[17] Alisys Robotics. Así es furhat: el robot que redefine la comunicación máquina-robot, 2024. URL https://alisysrobotics.com/es/blog/asi-es-furhat-el-robot-que-redefine-la-comunicacion-maquina-robot.

[18] Ulster University. Second international digital mental health and wellbeing conference, 2024. URL https://www.ulster.ac.uk/conference/digital-mental-health-and-wellbeing.

[19] MoodTools. Moodtools: Depression aid, 2024. URL https://moodtools.org/.

[20] Headspace. Headspace: Meditación y mindfulness, 2024. URL https://www.headspace.com/es.

[21] Happify. Happify: Science-based activities and games for stress and anxiety, 2024. URL https://www.happify.com/.

[22] BetterHelp. Betterhelp: Online counseling therapy. https://www.betterhelp.com/, 2024.

[23] Substance Abuse and Mental Health Services Administration. Substance abuse and mental health services administration, 2024. URL https://www.samhsa.gov.

[24] 7 Cups of Tea. 7 cups. URL https://www.7cups.com/es/.

[25] Joseph Fordham and Christopher Ball. Framing mental health within digital games: An exploratory case study of hellblade. *JMIR Mental Health*, 6(4), 2019. doi: 10.2196/12432. URL https://doi.org/10.2196/12432.

[26] Zoe Quinn. Depression quest: an interactive (non)fiction about living with depression, 2013. URL www.depressionquest.com.

[27] Imogen H. Bell, Jennifer Nicholas, Mario Alvarez-Jimenez, Andrew Thompson, and Lucia Valmaggia. Virtual reality as a clinical tool in mental health research and practice. *Dialogues in Clinical Neuroscience*, 22(2):169–177, 2020. doi: 10.31887/DCNS.2020.22.2/lvalmaggia. URL https://doi.org/10.31887/DCNS.2020.22.2/lvalmaggia.

[28] Sascha Keutel. The application of xr in mental health care. *Healthcare in Europe*, 2022. URL https://healthcare-in-europe.com/en/news/application-of-xr-in-mental-health-care.html.

[29] XR Health. Catalog of experiences, 2024. URL https://www.xr.health/experiences/.

[30] Eva Hudlicka. Virtual training and coaching of health behavior: Example from mindfulness meditation training. *Patient Education and Counseling*, 2013. doi: 10.1016/j.pec.2013.05.007. URL http://dx.doi.org/10.1016/j.pec.2013.05.007. Article in press.

[31] Ada Health. Ada: Tu compañero de salud, 2024. URL https://ada.com/es/.

[32] Md. Rokonuzzaman Haque and Saira Rubya. An overview of chatbot-based mobile mental health apps: Insights from app description and user reviews. *JMIR mHealth and uHealth*, 11, 2023. doi: 10.2196/44838. URL https://doi.org/10.2196/44838.

[33] Mindspa. Mindspa: Tu compañero de salud mental, 2024. URL https://mindspa.me/es/.

[34] Salario de programador junior, 2024. URL https://es.jooble.org/salary/programador-junior.

[35] Furhat Robotics. Furhat pricing, 2024. URL https://furhatrobotics.com/get-furhat/.

[36] JetBrains. Free educational licenses, 2024. URL https://www.jetbrains.com/community/education/#students.

[37] Universidad de Granada - Vicerrectorado de Estudiantes y Vida Universitaria. Gabinete psicopedagógico - principales problemas y desafíos, 2024. URL https://ve.ugr.es/secretariados-y-unidades/orientacion/recursos/coleccion-temas.

[38] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024.

[39] LangChain, Inc. Langchain documentation, 2024. URL https://python.langchain.com/v0.2/docs/introduction/.

[40] Omar Osanseviero. Sentence embeddings: Exploring distance between embeddings, 2024. URL https://osanseviero.github.io/hackerllama/blog/posts/sentence_embeddings/#distance-between-embeddings.

[41] Spencer Porter. Understanding cosine similarity and word embeddings, 2024. URL https://spencerporter2.medium.com/understanding-cosine-similarity-and-word-embeddings-dbf19362a3c.

[42] Furhat Robotics. *Furhat Developer Documentation*, 2024. URL `https://docs.furhat.io/`.

[43] MongoDB, Inc. Mongodb: The developer data platform, 2024. URL `https://www.mongodb.com/`.

[44] Google. Angular documentation, 2024. URL `https://v17.angular.io/docs`.

[45] Google. Angular material, 2024. URL `https://material.angular.io/`.

[46] GitHub. About github and git, 2024. URL `https://docs.github.com/en/enterprise-server@3.10/get-started/start-your-journey/about-github-and-git`.

[47] IBM. ¿qué es docker?, 2024. URL `https://www.ibm.com/es-es/topics/docker`.

[48] AI@Meta. Llama 3 model card. 2024. URL `https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md`.

[49] OpenAI. Best practices for prompt engineering with the openai api, 2024. URL `https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api`.

[50] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022. doi: 10.48550/ARXIV.2210.07316. URL `https://arxiv.org/abs/2210.07316`.

[51] ngrok - unified application delivery platform for developers, 2024. URL `https://ngrok.com/`.

Apéndice A

Manual de usuario

A.1. Introducción

Este manual de usuario detallará como ejecutar localmente el proyecto. A continuación se detallará como, a partir del código fuente, se despliega de manera local el programa en su totalidad.

A.1.1. Chatbot

Ya que en la documentación de Furhat se recomienda encarecidamente, se hará uso de IntelliJ IDEA y se configurará desde ahí

Prerrequisitos

Para utilizar el robot de Furhat es necesario instalar el siguiente software:

- IntelliJ IDEA: La versión *Community Edition* es suficiente.
- FurhatSDK Desktop: Esta aplicación es necesaria si se desea ejecutar la *skill* del robot de manera virtual en lugar del robot físico.

Configuración Virtual

En caso de ejecutar la versión virtual del robot lo primero necesario es iniciar FurhatSDK Desktop. Para ello hay que tener una cuenta en `https://furhat.io/` y generar gratuitamente un *Furhat API token* que hay que introducir al ejecutar el software.

La clave se genera desde la pestaña *settings* situada en el botón *My account*.

Con el token introducido y el SDK instalado completamente se presenta la siguiente interfaz:

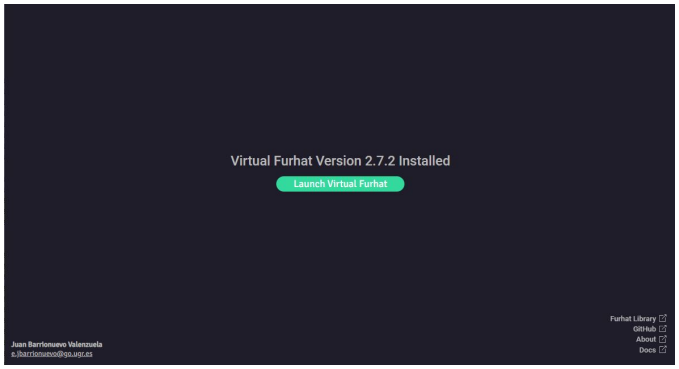


Figura A.1: SDK pantalla de bienvenida

Hay que pulsar sobre *Launch Virtual Furhat* y así se inicia el robot virtualmente. Tras pulsar el botón se abre otra ventana en la que está el robot virtual y otra ventana con la interfaz del SDK.

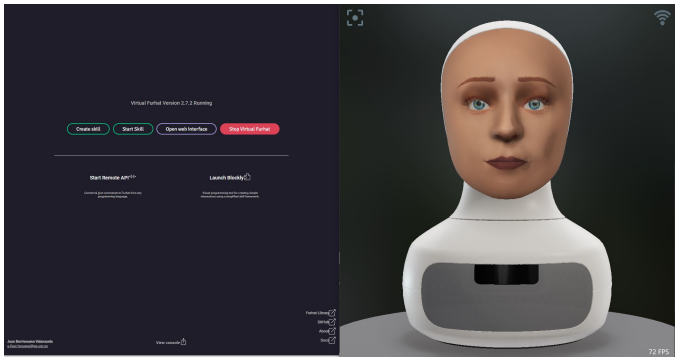


Figura A.2: SDK tras ser ejecutado

También es posible acceder a la dirección `http://localhost:8080/` e iniciar sesión con la contraseña *admin* para usar la interfaz web que permite crear usuarios virtuales con los que el robot virtual interactúa, elegir la *skill* que ejecutar y cambiar diversos parámetros manualmente como la voz o la cara.

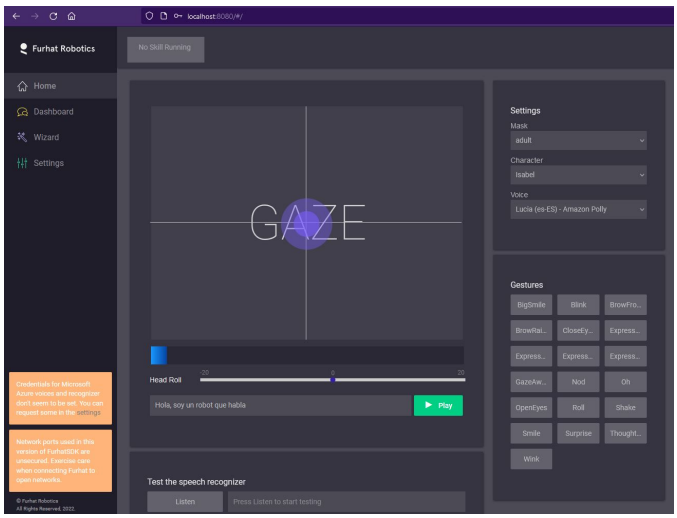


Figura A.3: Página de bienvenida tras iniciar sesión en la interfaz web

Ahora desde IntelliJ IDEA hay que ejecutar *main.kt* y se ejecutará la *skill* en el robot virtual.

Es necesario desde la interfaz virtual crear un usuario virtual usando doble click sobre la ventana para que el robot detecte que hay alguien delante suya.

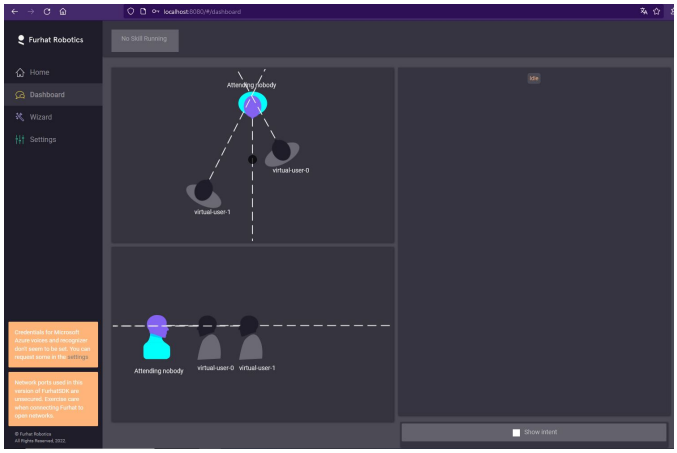


Figura A.4: Sección Dashboard de la página web con dos usuarios virtuales añadidos

Configuración Física

Para la configuración física es necesario tener el robot conectado a la misma red que el ordenador que tiene la *skill*. Hay que generar el archivo .skill ejecutando shadowJar desde la pestaña *Gradle* de IntelliJ IDEA.

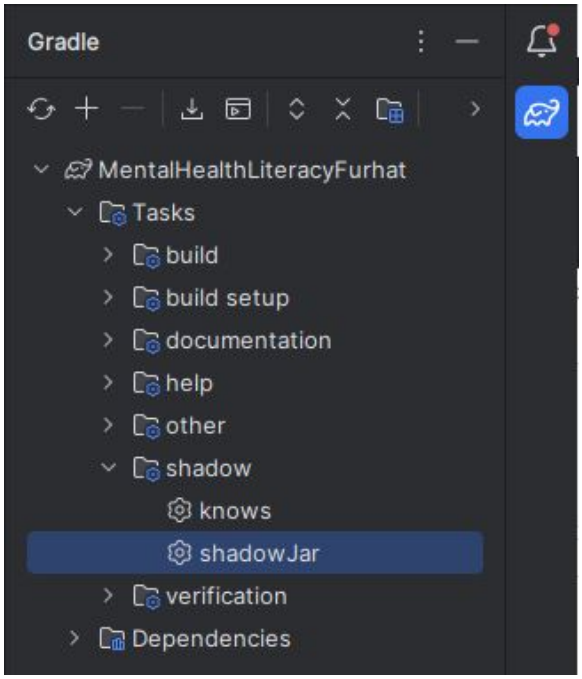


Figura A.5: Localización del archivo shadowJar a ejecutar

La ejecución de shadowJar genera un archivo .skill en carpeta del proyecto /build/libs.

Desde la IP que se puede consultar desde la configuración de la pantalla del robot se accede a ella de la misma manera que se accedió a la interfaz web virtualmente y desde ahí se sube al robot. En la lista de skills ahora aparece la recién descargada al robot.

Es muy importante tener en cuenta que el robot virtual es capaz de conectar con el backend haciendo uso de localhost ya que típicamente van a estar alojados en el mismo ordenador. En cambio, el robot físico es una máquina ajena al ordenador donde está alojado el backend por lo que es necesario cambiar la URL a la que conecta. Para ello se hizo uso de ngrok, un servicio que permite alojar gratuitamente (si es para *pre-release* o aplicaciones internas como es el caso) servidores locales en internet sin necesidad de un dominio propio. [51]

Para utilizarlo hay que crear una cuenta en su página web, descargar el

servicio y ejecutar lo siguiente por la línea de comandos (el token se obtiene en la web):

```
$ ngrok config add-authtoken <tu-authToken>
$ ngrok http http://localhost:5000
```

El token es personal y se obtiene en la misma web y el puerto es 5000 ya que es el puerto donde está por defecto ejecutado el servidor, en caso de cambiarlo es necesario poner el puerto nuevo.

A.1.2. Backend

Prerrequisitos

Para hacer uso del backend en un ordenador localmente es necesario tener instalado:

- Python 3.10
- Docker: Además de tenerlo instalado también hay que ejecutarlo.

Configuración

A continuación es necesario inicializar el entorno virtual del proyecto.

```
$ python -m venv venv
```

Después activamos el entorno virtual ejecutando el script de activación.

En Unix:

```
$ source venv\bin\activate
```

En Windows:

```
$ venv\Scripts\activate
```

Ahora procedemos a instalar los requisitos de la aplicación los cuales están detallados en el archivo *requirements.txt*.

```
$ pip install -r requirements.txt
```

A continuación en la carpeta raíz hay que crear un archivo llamado *.env* y configurar las variables de entorno necesarias. En el caso del desarrollador su archivo era el siguiente:

```
#
# Docker volume (a folder) where we will store the DB
#
MONGO_VOLUME=~/mongo

#
# SQL Server external & internal port (external:internal)
#
MONGO_PORT_MAPPING=27017:27017

#
# URL FOR LLM
#
MODEL_URL="Introducir aqui la URL del LLM a usar"
```

Una vez tenemos todo esto preparado hay que iniciar y ejecutar el contenedor el cual está definido en el archivo *docker-compose.yml*

```
        $ docker-compose up -d
```

Podemos ver que ha ido bien y que el docker está creado y se está ejecutando viendo la lista de contenedores y ver que el docker con nombre *mongo* está en ejecución.

```
        $ docker ps
```

Para un entorno de pruebas, se puede crear un usuario con email *admin@admin.com* con contraseña *admin* con el siguiente comando generado a partir de una función del código fuente.

```
        $ flask --app flaskr populate_user_db
```

Ahora tras la ejecución de ese comando se acaba de generar un usuario en la base de datos. Ya solo falta ejecutar la aplicación ya que todas las dependencias fueron instaladas en un paso previo.

```
        $ flask --app flaskr run
```

Veremos que la aplicación se ejecuta en el puerto 3000 y la base de datos en el 27017 (o el que se pusiese en .env).

A.1.3. Frontend

Prerrequisitos

Para el frontend hay que instalar lo siguiente:

- Node.js y npm

Configuración

Con Node.js y npm instalados solo hace falta ejecutar el siguiente comando desde el directorio raíz del proyecto:

```
1        $ npm install
```

Con el comando previo se instalarán todas las dependencias necesarias para el proyecto. Ahora sólo falta ejecutar la aplicación que se puede haciendo uso el CLI de Angular de nuevo desde el directorio raíz.

```
1        $ ng serve
```