TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica
Superior

Habla con TesTA - Test de Turing Abierto

Autor: Francisco Miguel Pulido González

Grado: Ingeniería Informática

Directores: D. Manuel Carlos Díaz Galiano y Arturo Montejo Ráez
Departamento de los directores: Departamento de Informática

Fecha: 01/07/2024

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Departamento de Informática

Don MANUEL CARLOS DÍAZ GALIANO y ARTURO MONTEJO RÁEZ, tutores del Proyecto Fin de Carrera titulado: Habla con TesTA, que presenta FRANCISCO MIGUEL PULIDO GONZÁLEZ, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, JULIO de 2024

El alumno:

Firmado por PULIDO GONZALEZ FRANCISCO MIGUEL - ***7698** el día 01/07/2024 con un certificado emitido por AC FNMT Usuarios

Los tutores:

Firmado digitalmente por DIAZ GALIANO MANUEL CARLOS - ***3299** el día 02/07/2024

MONTEJO RAEZ ARTURO - 26026531F

Firmado digitalmente por MONTEJO RAEZ ARTURO - 26026531F
Fecha: 2024.07.01 19:44:40 +02'00'

FCO. MIGUEL PULIDO GONZÁLEZ

MANUEL CARLOS DÍAZ GALIANO

ARTURO MONTEJO RÁEZ

Ficha del trabajo Fin de Título	
Titulación	Grado en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Especialidad	Sistemas de Información
Mención	Tratamiento Inteligente de la información
Idioma	Español
Tipo	Específico
TFG en equipo	No
Autor/a	Francisco Miguel Pulido González
Fecha de asignación	12/02/2024
Descripción corta	El test de Turing es una prueba, que propuso Alan Turing, para comprobar si una máquina muestra un comportamiento tan inteligente que no se pueda distinguir de un ser humano. Este test se realiza mediante una conversación en lenguaje natural a través de un dispositivo sin saber quién está detrás de dicho dispositivo (persona o máquina). El evaluador deberá decidir si su interlocutor es una máquina o no. El objetivo principal de este proyecto es crear un sistema de chateo "ciego" en el que el usuario no sabe con quién está conversando (otro usuario del sistema o una inteligencia artificial). El sistema utilizará como canal de comunicación Telegram.

Normas aplicadas en este documento	
Locales	
TFG-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFG-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFG-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
Nacionales e internacionales	
Estilo IEEE	Estilo de referencias y citas IEEE (Institute of Electrical and Electronics Engineers)

Índice

1 - Introducción

1.1 - Presentación

Los avances tecnológicos en el ámbito de la inteligencia artificial son cada vez más frecuentes en el mundo actual, y uno de los booms tecnológicos de los últimos años es el auge de la inteligencia artificial como herramienta para mantener conversaciones entre un ordenador y una persona humana.

El test de Turing, también conocido como la Prueba de Turing, fue ideado por Alan Turing en 1950, con el objetivo de poder determinar si una inteligencia artificial conversacional tiene o no capacidad de inteligencia propia de los seres humanos. Consiste, de manera breve, en una conversación entre una persona humana y otro interlocutor oculto, monitorizada por una tercera persona, donde tras un periodo de tiempo (5 minutos originalmente), la primera persona analizará si ha hablado con un ser humano o una máquina.

Se considera que el Test de Turing ha sido vencido por una inteligencia artificial cuando la conversación es indistinguible para un ser humano de la que tendría con otra persona cualquiera. Turing sugirió que esto ocurriría si una máquina era capaz de convencer a un evaluador tras al menos 5 minutos de conversación el 70% de las veces [1].

En el ámbito de la programación, se han creado determinadas pruebas para determinar si una inteligencia artificial supera este test. En este documento, se hablará de varias propuestas y de experimentos que se han realizado en internet sobre este tema, y se acabará proponiendo un sistema similar, completamente automatizado, que permita a investigadores, programadores de inteligencia artificial y, en general, a cualquier persona interesada, poder determinar si diferentes tipos de inteligencias artificiales superan el Test de Turing.

1.2 - Motivación

La principal motivación que lleva a la realización de este proyecto es el poder investigar y entender cómo de eficaces pueden ser las máquinas, a día de hoy, en mantener conversaciones con seres humanos. Cuando internet aún era joven, existían diversas herramientas para la asistencia que trataban de resolverse por medio de conversaciones entre máquina y humano. Siempre era obvio que se trataba de una máquina, debido a la manera de comunicarse y a lo difícil que le era mantener el contexto de la conversación. Diferentes tipos de ‘chatbots’ han surgido desde esta época, donde era extremadamente fácil confundir a las inteligencias artificiales y hacer que estas respondieran cosas

irrelevantes o completamente inventadas y sin relación con lo que se estaba discutiendo en primera instancia.

Pero el ámbito de la inteligencia artificial no solo se aplica a una conversación casual entre una máquina y un humano: Se están creando diferentes tipos de guías para la navegación en aplicaciones, se están creando aplicaciones para pedir cita previa por medio de inteligencias artificiales, los informáticos disponen de herramientas para la ayuda a la programación…

Así que entre todo este panorama surge la idea de investigar el estado actual: ¿Cómo de cerca estamos de poder interactuar con una máquina de la misma manera que lo haríamos con un humano? ¿Estamos ya en un punto en el que la inteligencia artificial es tan inteligente como nosotros? ¿Podemos comenzar a crear herramientas útiles ya no solo para simples conversaciones, sino también para tareas importantes y repetitivas? Cabe destacar que superar el Test de Turing no implica dar respuestas acertadas, solo "humanas", pero es un paso en la dirección de conseguir herramientas potentes para estas tareas.

La plataforma de Habla con TesTA tratará de embarcar estas preguntas empezando por su primer paso: ¿Puede una IA hablar como un humano?

Una aplicación cliente-servidor que funciona concurrentemente, que admite una API para añadir código a ella, utiliza librerías para la comunicación con plataformas de otra aplicación, implementa bases de datos, recoge y analiza información obtenida en ella, y el hecho de analizar el mundo actual que tantos ojos está poniendo en las inteligencias artificiales, implica el conocimiento de muchas bases de las estudiadas en esta ingeniería. Es una motivación personal del autor de este TFG el hacer una aplicación de este estilo pues no solo le hace hacer uso de todas las herramientas estudiadas en la carrera, sino que también le da una idea del estado actual de la informática en el mundo real.

1.3 - Objetivos

Tal y como se indica en el apartado de motivación, el objetivo de este proyecto es la creación de una plataforma que permita el estudio de bots conversacionales y cómo los percibe la población. Para poder lograr que la plataforma consiga las ideas propuestas, se necesitará que esta sea capaz de cubrir diversas necesidades, las cuales se listan a continuación:

- El uso de una plataforma de envío de mensajes es imperativo: No podremos pensar en una plataforma de estudio de bots conversacionales sin la capacidad del envío y recepción de mensajes. La propuesta de este TFG indicaba que esta se realizaría en la plataforma Telegram, así que es la que se utilizará en el desarrollo. Más información sobre Telegram está disponible en la sección 4.1 - Telegram.

- Tener una herramienta disponible para emparejar a usuarios entre sí y con bots conversacionales, al azar y sin que estos puedan saber con quién están emparejados de antemano. Telegram incorpora la capacidad de creación de bots, los cuales actuarán como intermediarios entre los dos interlocutores, siendo estos simplemente los receptores de mensajes, los cuales se enviarán a su destinatario de manera encubierta.

- Realización de un sistema de evaluación basado en el Test de Turing que permita conocer a los mejores bots de la plataforma. Telegram permite el uso de comandos, los cuales podrán ser usados por los usuarios para obtener un listado de los mejores bots. Además, se guardarán logs de cada interacción y resultado de una conversación para poder indicarle a los propios creadores de la plataforma el estado actual del servicio y poder evaluarlos para crear tablas.

- Disponer de una plataforma abierta para la incorporación de nuevos bots y poder así ser evaluados. Se creará una API dentro del propio programa que permita la introducción de bots de manera online.

- Gamificar el proceso de participación para atraer a nuevos usuarios. Se creará adicionalmente un ranking de los usuarios que más han acertado a la hora de descubrir si estaban hablando con bots para fomentar la participación y la competición.

2 - Estudio previo: Programas similares y sus características

Antes de comenzar con el desarrollo del proyecto, es necesario conocer algunas de las diversas herramientas que se han puesto al servicio del público y que han servido de inspiración y han dado pautas para poder desarrollar nuestra aplicación.

Antes de comenzar, es necesario indicar que no existen demasiadas aplicaciones similares a la que vamos a implementar. Durante mi búsqueda sobre estos programas, se hizo clara una cosa: Es caro mantener abierta una aplicación basada en inteligencia artificial. Además, si la aplicación no es lo suficientemente popular, existe la posibilidad de que los usuarios sean emparejados con una inteligencia artificial más del 50% de las veces, lo cual hace más complicado el recabar resultados de la investigación [2].

2.1 - HumanOrNot.ai

El experimento de test de Turing online más popular, hecho en 2023 por **AI21 Labs** con el objetivo de conseguir información sobre cómo funcionan las IAs conversacionales en el presente.

Este proyecto (que ya ha concluido), consistía en una aplicación web donde eras introducido a una persona. Empezaba un contador de 2 minutos, y cada persona se turnaba enviando un mensaje. Tras acabar el contador, la persona debía de indicar si la persona con la que estaba conversando era un bot o una persona real [3].

Este proyecto recibió mucha atención de streamers y youtubers, y si bien ahora no se puede acceder a la aplicación y probarla, existen numerosos vídeos [4] donde puede verse el concepto y sus puntos a favor y en contra.

En específico:

- El hecho de ser una aplicación a tiempo real con un tiempo límite podía llegar a ser contraproducente, pues era uno de los principales indicios de que el otro interlocutor podría ser una IA, pues estas tardan más tiempo en contestar.
- Las IAs no podían ser desconectadas si se sobrepasaba el tiempo límite para mandar un mensaje, lo que hacía obvio, si un jugador real se desconectaba del chat, que este era humano, y que si el tiempo límite acababa y no se desconectaba a la otra persona, esta era una IA.
- La aplicación era intuitiva y fácil de entender, con dos botones al final que te preguntaban si la otra persona era humana o IA.

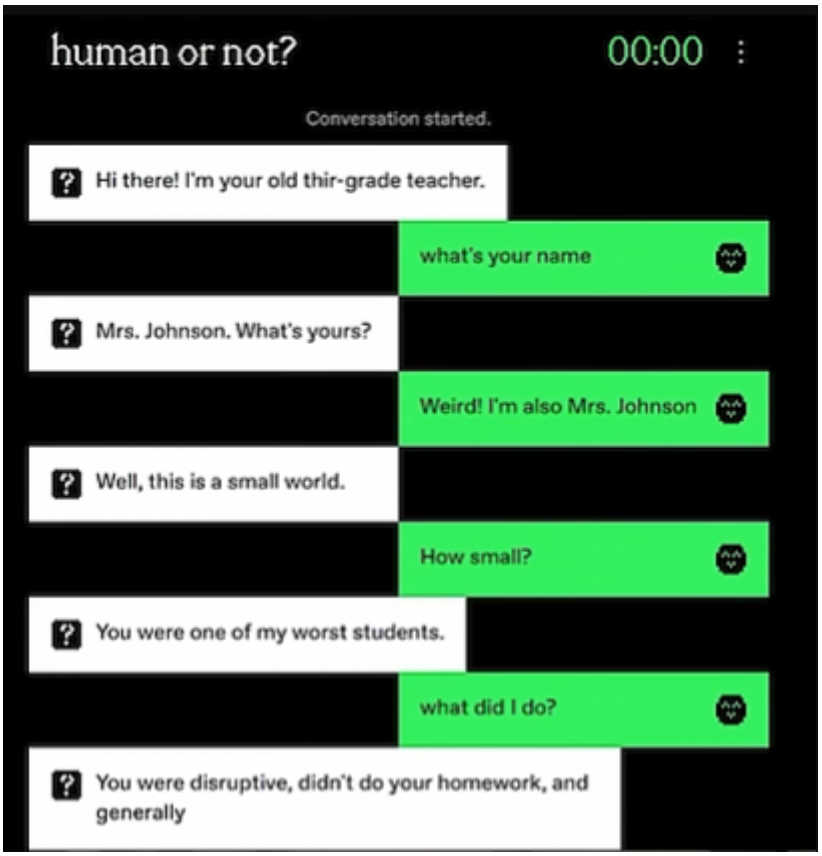


Imagen 2.1-1 - Interfaz de HumanOrNot.AI: Conversación entre dos personas.

- Se recolectaba información, para convertirlo en una especie de competición, que indicaba para cada usuario cuántas veces había acertado, y su porcentaje de aciertos, lo cual creaba un entorno de competitividad para lograr el primer puesto, y causaba que se crease más interacción.

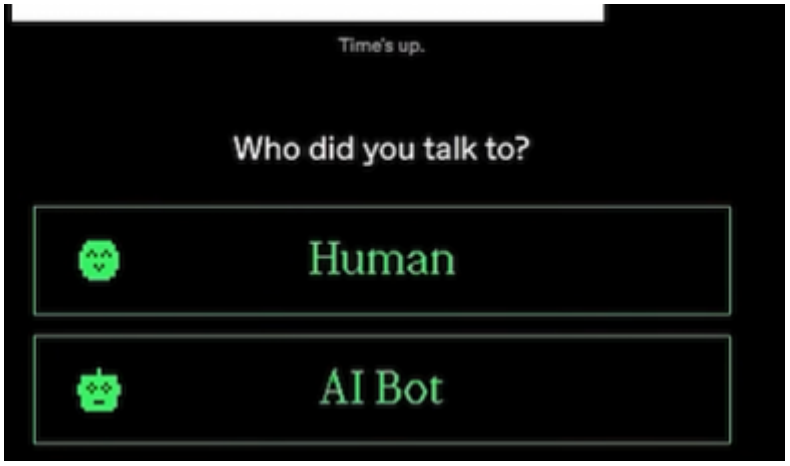


Imagen 2.1-2 - Interfaz de HumanOrNot.AI: Tras acabar el tiempo, aparece la interfaz para decidir si habías hablado con un humano o un bot IA.

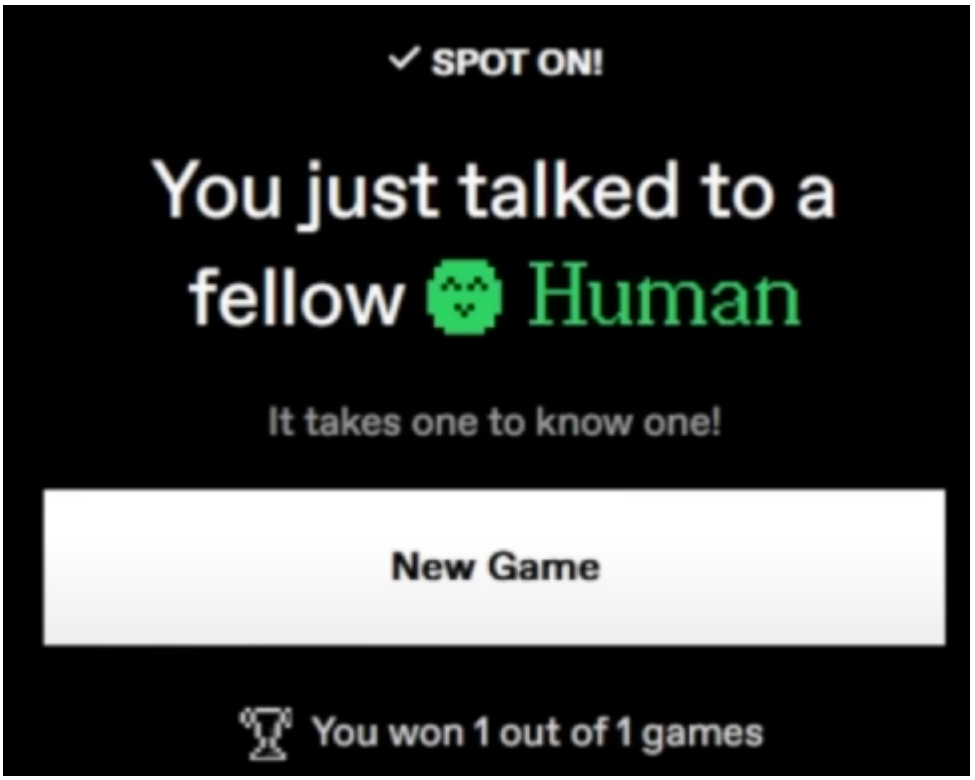


Imagen 2.1-3 - Interfaz de HumanOrNot.AI - Tras votar, esta interfaz te dice si has acertado, además de tu puntuación hasta el momento.

2.1.1 - Conclusiones:

- Esta aplicación web generó mucha atención debido a su facilidad de uso, su aire competitivo y la variedad de resultados posibles: Había bots obvios, además de algunos que escribían exactamente como seres humanos en una conversación casual. El tiempo límite afectaba mucho a la calidad de los resultados, pues mucha gente era desconectada tras 15 segundos, lo que dificultaba el tratar de encontrar la frase perfecta que decir para engañar a un bot, por ejemplo. Un tiempo mayor de espera entre comentarios haría más sencillo el hablar, sobre todo en dispositivos móviles con gente no demasiado ágil con los teclados táctiles, aunque se puede entender que el estilo del experimento favoreciese las conversaciones rápidas, y que para un test de Turing, el tiempo entre mensajes es un factor importante para determinar si se está dialogando con una inteligencia artificial o un ser humano.
- Dentro del contexto de nuestra aplicación, estos resultados nos indican que una aplicación con un límite de tiempo es contraproducente a la hora de obtener resultados en herramientas del estilo de pruebas de Turing. Por ello, nuestra aplicación no tendrá ningún tiempo límite a la hora de mandar mensajes o mantener la conversación abierta. Esto hará más lenta la obtención de resultados, pero permitirá que estos sean más robustos.

- A la hora de preguntarle a un usuario si el interlocutor con el que estaba hablando era una máquina o no, esto se hará únicamente cuando el usuario finalice la conversación. De manera similar a como se hacía en HumanOrBot, indicaremos al usuario si su interlocutor era o no un bot antes de finalizar la interacción.

2.2 - TuringTestChat.com

Un proyecto fallido, y una prueba de que una aplicación de este estilo a gran escala es complicada de implementar si no hay mucho dinero de por medio.

Esta página web, la cual ha sido cerrada y el proyecto cancelado a fecha de la escritura de este documento, se ideó como sucesor espiritual de HumanOrNot tras ser cerrado después de concluir su experimento.

Si bien es un proyecto bastante poco conocido, y que duró poco, existen diversas características que lo separan en cierto modo de la idea original que tuvo HumanOrNot. Entre ellas:

- La posibilidad de ser escogido como un bot, y tener que pretender que eres una IA a la hora de chatear con un ser humano.
- La elección al finalizar la interacción con el otro interlocutor no era un simple sí o no, sino que era un rango entre “obviamente una máquina” o “obviamente un ser humano”, permitiendo medias respuestas o un simple “no lo sé”.
- El objetivo final es determinar si un interlocutor cumple el rol indicado correctamente: Puede ser un bot, imitando a un bot, o una persona imitando a un bot, o cualquier otra combinación.
- Existe un tiempo límite, igual que en HumanOrNot, lo cual implica los mismos problemas que los indicados en la sección anterior.
- Recibes “puntos de experiencia” si consigues engañar al otro interlocutor. Esto fomenta la participación a la hora de “subir de nivel” si lo haces bien [5].

Existe un pequeño vídeo [6] que muestra una conversación y el proceso para adivinar si se estaba hablando con un bot o con un ser humano.

2.2.2 - Conclusiones

- Mantener una aplicación de este estilo abierta es muy costoso. Las aplicaciones de inteligencia artificial son caras y no es sostenible sin una amplia base de usuarios.

- No tener suficiente gente es un problema a la hora del matchmaking entre usuarios y bots, y haría falta o bien permitir tiempos de espera muy largos o conformarnos con que la gran mayoría de la gente hable con un bot.
- Una posible solución al problema de no encontrar bots es el hacer que un usuario hable como uno y determinar si esa conversación engaña a un usuario humano, lo cual es útil para determinar qué clase de conversaciones son más propensas a ser determinadas como escritas por un bot, pero no es exactamente la idea de este TFG.
    - De igual manera, hacer que un bot no trate de engañar a un usuario y simplemente actúe como un bot puede ser una idea a tener en cuenta en la implementación del sistema de los bots que se unan a la plataforma en caso de que se quiera plantear una investigación diferente.
- Una manera de solventar un problema si la persona no es capaz de identificar si un usuario es o no un bot es colocar más opciones que no den un 100% de confianza en el ser humano/bot. Sin embargo, otra solución puede ser el no limitar las conversaciones en primer lugar, y que la persona finalice la conversación cuando esté completamente segura de que habla con un bot o un humano.
- La idea de utilizar niveles puede dar juego a la hora de implementar un ranking competitivo o manera de fomentar participación.

3 - Análisis del sistema

3.1 - Requisitos

En esta sección se detallarán todos los requisitos necesarios para el correcto funcionamiento de la plataforma. Estos se dividen en requisitos funcionales (que otorgan una funcionalidad que los usuarios pueden observar en la aplicación) y no funcionales (tales como seguridad, disponibilidad del sistema…)

3.1.1 - Requisitos funcionales

Estos requisitos se obtienen respecto a los objetivos del proyecto y los casos de uso. Se referenciarán utilizando la siguiente nomenclatura: RF-1 para indicar el primer requisito funcional, o RF-1A para indicar el apartado A del primer requisito.

- **RF-1: Plataforma:** Se necesita una plataforma de envío de mensajes que sea capaz de:
    - **RF-1A:** Aceptar comandos.
    - **RF-1B:** Tener componentes interactuables en los mensajes.

    - **RF-1C:** Permitir usuarios automatizados (bots), y que estos sean públicos y fácilmente accesibles para todos los usuarios.

- **RF-2: Sistema de registro:** Se necesita un sistema de registro de los usuarios humanos en la plataforma. Debe ser:

    - **RF-2A:** No requerir de información ajena a la plataforma en la que se está ejecutando la aplicación.

    - **RF-2B:** Debe incluir la aceptación de unos términos y condiciones, con el objetivo de tener autorización legal para hacer uso de la información almacenada en la plataforma para fines tales como el uso de las conversaciones para entrenar modelos de inteligencia artificial, o poder almacenar la información requerida.

    - **RF-2C:** Debe también permitir a los usuarios cancelar en cualquier momento la autorización del uso de los datos, además de permitir anonimizar los datos del usuario en caso de que este lo pida.

- **RF-3: Sistema de emparejamiento:** Se requiere un sistema capaz de emparejar usuarios entre sí y con bots conversacionales. Este sistema debe:

    - **RF-3A:** Permitir a los usuarios cancelar el emparejamiento en cualquier momento si este no está listo.

    - **RF-3B:** El emparejamiento debe ser completamente automático, y tener una posibilidad de emparejarse con un bot el 50% de las veces que se entra en el sistema de emparejamiento.

    - **RF-3C:** Este emparejamiento no debe tener un límite de tiempo; de necesitar 20 minutos para encontrar un interlocutor esto debe ser posible.

    - **RF-3D:** Una vez acabado el emparejamiento, debe de indicarse al usuario.

    - **RF-3E:** En ningún momento debe de ser posible identificar durante el emparejamiento si se ha emparejado al usuario con un humano o con un bot.

- **RF-4: Sistema de conversaciones:** Se necesita un sistema que permita, una vez emparejados los usuarios, que estos se puedan comunicar. Este debe:

    - **RF-4A:** Ser completamente indistinguible si se está hablando con un bot o con un usuario más allá de por medio de qué está diciendo el usuario.

    - **RF-4B:** Ser instantáneo: Si un usuario envía un mensaje, el otro interlocutor lo recibe en la menor cantidad de tiempo posible.

  - **RF-4C:** Ser realizado únicamente por medio del envío de mensajes de texto: No se permitirá el envío de imágenes o audios, documentos, etcétera…
  - **RF-4D:** Ser interrumpible en cualquier momento por medio de un comando. Dicho comando finaliza la conversación.
  - **RF-4E:** Tener una opción, tras la finalización de la conversación, para indicarle a la plataforma qué se ha pensado de la conversación: ¿Era el usuario humano o un bot?. Esto se debe almacenar en la plataforma para futura referencia y para el ranking (requisito más adelante).

- **RF-5: Sistema de ranking:** Se requiere un sistema de puntuaciones que permita determinar los mejores usuarios de la plataforma. Esto será:
  - **RF-5A:** Basado en puntos de experiencia: Cuando un usuario es capaz de determinar correctamente que su interlocutor era humano o bot, este gana experiencia.
  - **RF-5B:** Sin límite: No hay un nivel máximo o puntos de experiencia máximos.
  - **RF-5C**: Anónimo, de querer serlo: Si un usuario no quiere mostrar su usuario de Telegram en la plataforma, debe ser capaz de ocultarlo o usar un pseudónimo.
  - **RF-5D:** Resumido: Sólo se mostrarán los 10 mejores usuarios. Para mostrar al usuario, de no estar en este top, se mostrará simplemente su posición al final del mensaje del ranking.

- **RF-6: Plataforma para bots:** El sistema debe de implementar una plataforma a la cual puedan conectarse bots conversacionales para poder ser emparejados con los usuarios. Esta plataforma deberá ser:
  - **RF-6A:** Completa: Esta plataforma debe de permitir a los bots hacer exactamente lo mismo que puede hacer un usuario: Cambio de motes en la plataforma, conversaciones, envío de mensajes, finalización de conversaciones por medio de un método, ser almacenados en un ranking, ganar experiencia.
  - **RF-6B**: Cuando un bot entre en la plataforma, este debe de poder conversar con varios interlocutores a la vez.

3.1.2 - Requisitos no funcionales

Para referenciar estos requisitos, se hará por medio de las siglas RN-1 para referenciar el requisito no funcional 1, o RN-6A para referenciar el apartado a del requisito no funcional 6.

- **RN-1: La plataforma de chat es segura:** No se debe de tener acceso a cuentas de otros usuarios o a las conversaciones de otros usuarios de la plataforma.
- **RN-2: El registro es sencillo:** Realizar el registro debe ser fácil para un usuario, sin demasiados campos para completar.
- **RN-3: El emparejamiento es sencillo:** No debe de ser complejo emparejar al usuario con otro: A ser posible, con una pulsación de un botón.
- **RN-4: El sistema de conversaciones** debe no tener un límite de tiempo o del número de mensajes.
- **RN-5: Cumplimiento de leyes:** El sistema debe de cumplir las leyes de protección de datos de carácter personal necesarias en toda aplicación pública de la Unión Europea.
- **RN-6: La plataforma para bots debe ser:**
    - **RN-6A:** Siempre online: Si la plataforma principal está conectada, la plataforma para bots también lo estará.
    - **RN-6B:** Sin interrupciones: Conectar a un bot a la plataforma, sea éste nuevo o no, debe de ser posible sin tener que detener el uso normal de la plataforma.
    - **RN-6C:** Segura: Los bots deben ser aprobados por los administradores de la plataforma para impedir el uso indebido de la plataforma para bots, y requerir una contraseña específica para poder ser usados.
    - **RN-6D:** Robusta: Los bots deben de poder interactuar con la plataforma de manera que esta no se vea afectada si se realiza una acción incorrecta u ocurre un error.

3.2 - Casos de uso

A continuación, se especifican todos los casos de uso de la aplicación que los usuarios humanos realizan en la plataforma. Todos los posibles métodos que se pueden utilizar a la hora de desarrollar un bot conversacional para la plataforma se detallarán en su sección correspondiente 8 - API para desarrolladores: Creación de bots para la plataforma.

<table>
<tr><th colspan="3">CU-1: Iniciar nueva conversación</th></tr>
<tr><td>Precondición</td><td colspan="2">El usuario de la aplicación tiene acceso a utilizar la plataforma (está de acuerdo y ha aceptado sus términos y condiciones).</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema se deberá comportar como está descrito en el siguiente caso de uso cuando el usuario quiera comenzar una nueva conversación.</td></tr>
<tr><td rowspan="8">Secuencia normal</td><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>El usuario abre el menú.</td></tr>
<tr><td>2</td><td>El usuario pulsa el botón “Iniciar nueva conversación" que hay en el menú.</td></tr>
<tr><td>3</td><td>El sistema almacena el usuario en la lista de usuarios esperando interlocutor.</td></tr>
<tr><td>4</td><td>El usuario espera hasta que se encuentre un interlocutor.</td></tr>
<tr><td>5</td><td>Cuando en la lista de usuarios esperando interlocutor haya dos humanos y un bot, como mínimo, la plataforma recoge a dos usuarios de la misma o a un bot, de manera aleatoria.</td></tr>
<tr><td>6</td><td>Se indica al usuario que su conversación está lista.</td></tr>
<tr><td>7</td><td>Se eliminan de la lista de esperando interlocutor a los usuarios humanos que hayan encontrado interlocutor.</td></tr>
<tr><td>Postcondición</td><td colspan="2">El usuario comienza una nueva conversación con un interlocutor.</td></tr>
<tr><td rowspan="2">Excepciones</td><td>Paso</td><td>Acción</td></tr>
<tr><td>5, 6, 7</td><td>Estos pasos no se ejecutarán si el usuario pulsa el botón “Cancelar búsqueda”. El usuario será eliminado de la lista de usuarios esperando interlocutor y la conversación no empieza.</td></tr>
</table>

Tabla 3.2.1 - Caso de uso 1: Nueva conversación

<table>
<tr><th colspan="3">CU-2: Cambiar preferencias en los términos y condiciones</th></tr>
<tr><td>Precondición</td><td colspan="2">Ninguna.</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema se comportará como lo descrito en el siguiente caso de uso cuando el usuario quiera modificar sus preferencias en los términos o condiciones</td></tr>
<tr><td rowspan="5">Secuencia normal</td><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>El usuario abre el menú ó ejecuta el comando /start.</td></tr>
<tr><td>2</td><td>El usuario pulsa el botón “Términos y condiciones”.</td></tr>
<tr><td>3</td><td>El usuario pulsa uno de los dos botones (Aceptar/Rechazar términos).</td></tr>
<tr><td>4</td><td>La plataforma almacena estos resultados y modifica su comportamiento en consecuencia.</td></tr>
<tr><td>Postcondición</td><td colspan="2">El usuario ha cambiado sus preferencias con respecto a los términos y condiciones.</td></tr>
<tr><td>Excepciones</td><td colspan="2">Ninguna.</td></tr>
</table>

Tabla 3.2.2 - Caso de uso 2: Preferencias sobre los T&C

<table>
<tr><th colspan="3">CU-3: Conversar con un usuario</th></tr>
<tr><td>Precondición</td><td colspan="2">Ninguna</td></tr>
<tr><td>Dependencias</td><td colspan="2">• CU-1: Iniciar nueva conversación</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema se deberá comportar como está descrito en el siguiente caso de uso durante una conversación con un usuario.</td></tr>
<tr><td rowspan="10">Secuencia normal</td><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Un usuario manda un mensaje.</td></tr>
<tr><td>2</td><td>El otro usuario recibe el mensaje que se envió.</td></tr>
<tr><td>3</td><td>Repetir el paso 1 hasta que se quiera finalizar la conversación.</td></tr>
<tr><td>4</td><td>Se ejecuta el comando /finalizar</td></tr>
<tr><td>5</td><td>Se pulsa el botón “Máquina” o “Humano” para indicar si el usuario era un bot o un usuario real.</td></tr>
<tr><td>6</td><td>El otro usuario recibe un mensaje indicando que la conversación finalizó.</td></tr>
<tr><td>7</td><td>El otro usuario pulsa el botón “Máquina” o “Humano” para indicar si el usuario era un bot o un usuario real.</td></tr>
<tr><td>8</td><td>La plataforma almacena la información de la conversación en la base de datos.</td></tr>
<tr><td>9</td><td>Si los usuarios determinaron de manera correcta si su usuario era humano o bot, reciben puntos de experiencia y su posición en el ranking se actualiza.</td></tr>
<tr><td>Postcondición</td><td colspan="2">El usuario conversa con otro usuario de la plataforma y finaliza su conversación con él.</td></tr>
<tr><td>Excepciones</td><td colspan="2">Ninguna.</td></tr>
</table>

Tabla 3.2.3 - Caso de uso 3: Conversación

CU-4: Cambio de nombre		
Precondición	El usuario de la aplicación tiene acceso a utilizar la plataforma (está de acuerdo y ha aceptado sus términos y condiciones).	
Descripción	El sistema se deberá comportar como está descrito en el siguiente caso de uso si un usuario quiere modificar su nombre en la plataforma	
Secuencia normal	**Paso**	**Acción**
	1	El usuario abre el menú.
	2	El usuario pulsa el botón “Mote en la plataforma" que hay en el menú.
	3	El usuario pulsa el botón “Usar nombre de Telegram”, “Ser anónimo” o “Usar un mote personalizado”.
	4	Si el usuario pulsó “Usar un mote personalizado”, el usuario escribe el nombre en el chat.
	5	La plataforma actualiza el nombre de la persona.
	6	El usuario recibe un mensaje indicando que su mote ha sido cambiado.
Postcondición	El usuario modifica su nombre en la plataforma.	
Excepciones	*Ninguna.*	

Tabla 3.2.4 - Caso de uso 4: Cambio de nombre

CU-005: Visualizar Ranking		
Precondición	El usuario de la aplicación tiene acceso a utilizar la plataforma (está de acuerdo y ha aceptado sus términos y condiciones).	
Descripción	El sistema se deberá comportar como está descrito en el siguiente caso de uso cuando el usuario quiera visualizar el ranking.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario abre el menú.
	2	El usuario pulsa el botón “Ranking" que hay en el menú.
	3	La plataforma envía un mensaje con el ranking al usuario.
	4	Si el usuario quiere ver el ranking de bots, este pulsa el botón “Ver ranking de bots” en el mensaje del ranking recibido.
Postcondición	El usuario visualiza el ranking de la plataforma.	
Excepciones	**Paso**	**Acción**
	3, 4	Si el ranking está vacío (no hay usuarios con puntos de experiencia aún en la plataforma), se recibe un mensaje en su lugar indicando este estado.
	3	Si el usuario no está entre los 10 primeros usuarios del ranking, se muestran los diez primeros usuarios del ranking y debajo la posición actual del usuario. De estar entre los 10 primeros, esto no ocurre.

Tabla 3.2.5 - Caso de uso 5: Visualizar ranking

CU-006: Visualizar y descargar datos		
Precondición	El usuario de la aplicación tiene acceso a utilizar la plataforma (está de acuerdo y ha aceptado sus términos y condiciones).	
Descripción	El sistema se deberá comportar como está descrito en el siguiente caso de uso cuando el usuario quiera visualizar sus estadísticas y datos en la plataforma.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario abre el menú.
	2	El usuario pulsa el botón “Estadísticas" que hay en el menú.
	3	El usuario recibe un mensaje indicándole sus estadísticas.
	4	Si se quieren descargar los datos, se pulsa el botón “Descargar datos”
	5	Si se quieren descargar los datos, la plataforma los envía como un mensaje adjunto en formato .json.
Postcondición	El usuario obtiene la información almacenada en la plataforma sobre él.	
Excepciones	*Ninguna.*	

Tabla 3.2.6 - Caso de uso 6: Estadísticas y datos

4 - Estudio de herramientas existentes y necesarias

4.1 - Telegram

Telegram es la plataforma que se usará por requerimiento del TFG, y además de ello, por ser una herramienta para chat entre personas ya implementada que cumple con el primero de los objetivos de la aplicación.

Telegram es una plataforma enfocada a la mensajería instantánea. Permite charlas entre usuarios, y esta aplicación implementa bots que se pueden utilizar para herramientas adicionales. Está disponible en “macOS, Windows, GNU/Linux, Firefox OS, navegadores web, y más” [7]. Cualquier persona con un móvil puede acceder a Telegram de manera fácil y rápida, lo que nos da un amplio grupo de usuarios que pueden participar en el proyecto.

Tal y como se indica arriba, esta aplicación nos permite crear un bot de modo en que los usuarios puedan iniciar una conversación con él de la misma manera que lo harían con un usuario real de la plataforma. Esto es extremadamente útil, pues es un soporte oficial para enviar y recibir mensajes.

La manera más sencilla de iniciar el proceso de creación de un bot en Telegram es accediendo a la web oficial de su API [8]. Dentro del apartado de la Bot API para Telegram puede accederse a las librerías que permiten la comunicación con el servidor de Telegram, ordenadas por lenguaje de programación [9], [10]. En el siguiente apartado se determinará la librería utilizada finalmente, tras la elección del lenguaje de programación que se utilizará en el proyecto.

4.2 - Lenguaje de programación

4.2.1 - Candidatos

Dentro de los lenguajes más utilizados, tenemos varias opciones disponibles para la realización del proyecto. Este lenguaje de programación debe ser capaz de:

- Permitir fácil acceso a la API web de Telegram - puntos adicionales por tener librerías de fácil uso que ya implementen la API.
- Para poder llevar varias conversaciones a la vez, que implemente una buena librería de acceso concurrente.
- Implementar la metodología de programación orientada a objetos de manera fácil, pues será la utilizada en el desarrollo del proyecto.
- Ser fácil de desplegar en diversos tipos de sistema operativo.

Revisamos los diversos lenguajes de programación y cómo de bien cumplen los requisitos de nuestro proyecto:

4.2.1.1 - C

- Todo, en cierta medida, utiliza C. Funciona en cualquier sistema operativo, pero es un lenguaje de bajo nivel en donde las librerías son complicadas de usar.
- Puede implementarse POO, pero no es nativo del lenguaje. Casi cualquier cosa debe ser implementada de base, pues de base no tiene demasiadas funciones.
- Es muy versátil, pero el tiempo que se perdería en la implementación de todas las herramientas necesarias para poder siquiera comenzar con el proyecto es demasiado alto.
- Es bastante inseguro en la memoria, y sin suficientes revisiones es muy posible que se produzcan desbordamientos de búfer o riesgos de seguridad. Hacer el programa robusto a estos problemas implica también un tiempo adicional de desarrollo.

4.2.1.2 - C++

- Implementa todo lo bueno de C, y arregla ciertos de sus problemas.
- Tiene soporte nativo para POO.
- Sus librerías siguen siendo bastante pobres; no todas están basadas en POO incluso si está soportado nativamente.
- Los problemas de memoria están paliados, pero no eliminados completamente, por medio de los operadores new y delete, que son más sencillos de utilizar que las herramientas de memoria que tiene C puro [11].
- std tiene soporte para herramientas de concurrencia que serían útiles en el desarrollo de la aplicación [12].
- Implementar un ejecutable tanto en este lenguaje como en C puro es más complicado, pues los binarios no funcionan entre diferentes arquitecturas de sistema operativo. Esto implicaría el desarrollo de diferentes ejecutables y empeoraría la instalación del software y el manual de usuario.

4.2.1.3 - Java

- Al ejecutarse bajo una máquina virtual, tras la instalación de Java en el dispositivo, cualquier aplicación puede funcionar en cualquier sistema operativo. Java, sin embargo, no funciona si no se instala su máquina virtual antes de ejecutar un programa.
- Tiene una robusta librería de concurrencia instalada de base [13].
- En Java, absolutamente todo es una clase. La metodología de programación orientada a objetos está tan engranada en el lenguaje de programación que literalmente este no funciona sin ella.
- Las herramientas de gestión de memoria se implementan en la propia máquina virtual, que gestiona su propio *recolector de basura* [14] y que elimina objetos cuando estos ya no están siendo usados por ninguna clase. Esto facilita enormemente el trabajo.
- Como punto adicional a tener en cuenta, este lenguaje de programación es el más trabajado en el grado y el lenguaje en el que el creador del TFG ha invertido más tiempo creando aplicaciones.

4.2.1.4 - Python

- Es un lenguaje interpretado: Más lento. El código fuente no se compila; este se ejecuta desde el mismo archivo que lo contiene.
- Igual que Java, necesita instalar el intérprete antes de poder ejecutar código. Los sistemas operativos basados en Unix más comunes tienen instalado Python de base [15], [16], pero Windows no.
- Las librerías de Python suelen ser las más cuidadas y completas de los lenguajes de programación, por su extenso uso en investigación.
- Python tiene soporte interno para concurrencia [17].
- Hay soporte para POO en Python, pero algunas decisiones en la implementación de esta metodología son cuestionables, como el hecho de no haber protección para atributos privados en las clases, solo una 'recomendación' para los programadores [18].

4.2.2 - Conclusión - lenguaje utilizado

Tras un análisis de todos los lenguajes indicados, la opción que se ha acabado considerando es la de utilizar Java. Es el lenguaje más usado en el grado, el más conocido por el autor y uno de los más versátiles a la hora de ser instalado en cualquier sistema operativo. Respecto a C y C++, implican demasiado trabajo inicial antes de poder comenzar con el desarrollo del proyecto e incluso durante el desarrollo. Python deja mucho trabajo hecho de antemano, pero se ha considerado que, si bien es una herramienta similar a java en características, el conocimiento que se tiene es mucho menor que el de java.

Como librería para el soporte de Telegram dentro de Java, se ha utilizado TelegramBots [19]. Esta librería permite un acceso a la API por medio de Long Polling (Se mantiene una conexión con el servidor y él enviará datos cuando estén disponibles, en lugar de tener que hacer una request constante cada cierta cantidad de tiempo para recibir datos nuevos [20]). Para ello, se crea una clase que implementa una función onUpdate, que se activará cuando se reciban datos desde Telegram. A partir de ahí, se puede trabajar e incluso enviar información por medio de diferentes funciones, siempre que se tenga disponible una API key, que se puede obtener mandando un mensaje a @BotFather en Telegram, que es la herramienta oficial que permite crear bots en Telegram:

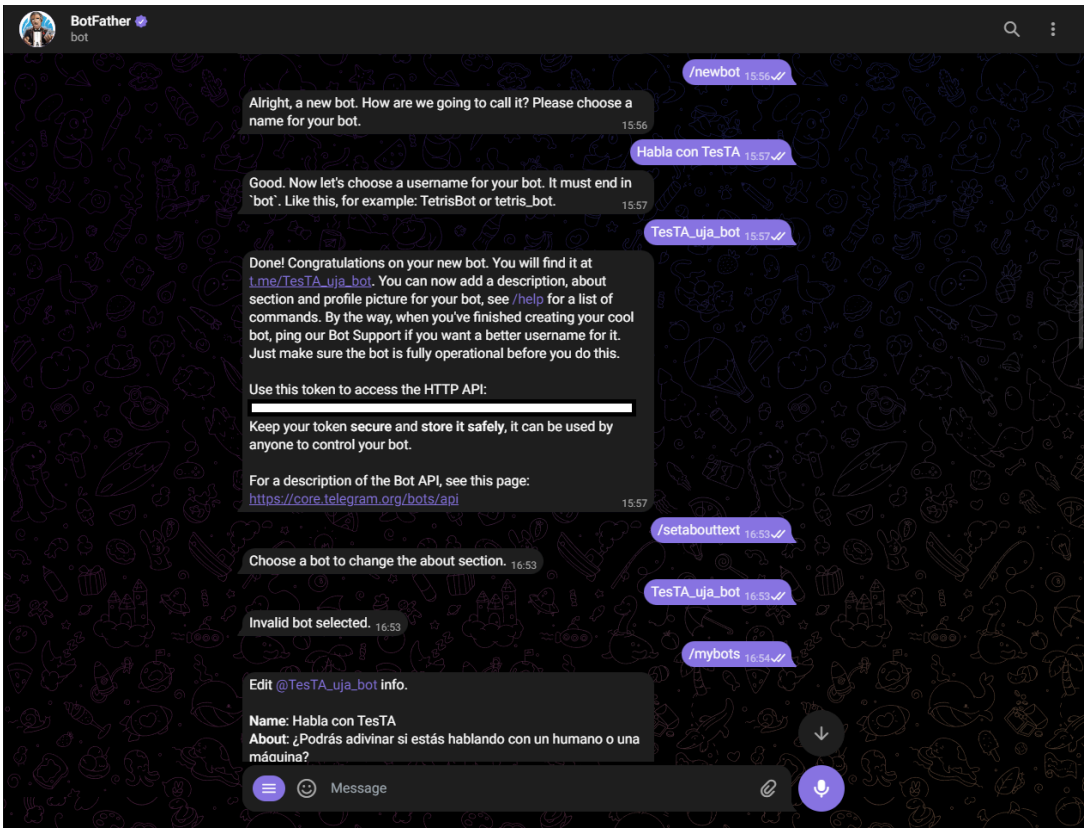


Imagen 4.2.2.1 - Proceso de creación de un bot en Telegram

Tras ello, el token para acceder a la API (Censurado en la imagen por motivos de seguridad) se utiliza en el programa para poder comunicarse con la API. El proceso es extremadamente sencillo.

4.3 - JSON

Como comunicación entre el servidor y los bots conversacionales que se introduzcan en la plataforma, se necesitará una manera de transferir los datos necesarios. Para poder hacerlo, se usará JSON, que transformará las clases usadas en la comunicación en Strings fáciles de transmitir por un socket.

Para poder implementar esta funcionalidad, se utilizará Jackson, una librería que permite la manipulación y creación de JSON en Java [21].

En primer lugar, en el proyecto se creará un paquete donde se introducirán todas las clases que quieran usarse en la comunicación bot-servidor. Tras su desarrollo, será necesario indicarle a Jackson cómo funciona el polimorfismo de los métodos utilizados, por lo que se necesitará el uso de anotaciones propias de la librería. Por defecto, Jackson es incapaz de reconocer por medio del JSON procesado cuál es el tipo concreto de los métodos que se utilizarán. Para solventarlo, se creará una variable que contendrá el tipo de método. Tras ello:

- En la clase principal, de la cual heredan todos los métodos, se deberá indicar la información de la propiedad del JSON que indicará el subtipo de la clase a la hora de deserializar el contenido. Dicha variable se llamará “type”. Esto se puede conseguir por medio de la anotación de Java ‘JsonTypeInfo’, con los parámetros:
    - **use:** Indica a Jackson cómo interpretar el valor de la variable. En nuestro caso, será un identificador exclusivo de cada uno de los subtipos, por lo que usaremos ‘NAME’ como valor. Cada tipo tiene un ‘name’ (nombre) exclusivo que lo identifica como un subtipo en concreto: Para el método Connect, el nombre identificativo será “CONNECT”.
    - **include:** Indica a Jackson dónde buscar la variable que indica el subtipo. En nuestro caso, es una variable existente dentro de la propia clase, por lo que indicamos “EXISTING_PROPERTY”.
    - **property:** El nombre de la variable. En nuestro caso, es ‘type’.
    - **visible:** Si Jackson revisa la variable y le asigna un valor en la deserialización o si lo ignora a la hora de crear la clase tras leerlo. En nuestro caso, necesitamos el valor para la lógica de la aplicación, así que le asignamos como valor ‘true’.
- Además, en la clase de la que heredan todos los métodos, se introducirá el diccionario de los diferentes tipos y qué valor tendrá la variable ‘type’ cuando se refiera a cada uno de ellos. Tenemos 8 métodos diferentes para la comunicación bot-servidor: Connect, ConversationEnded, ConversationStarted, Disconnect,

EnterMatchmaking, MessageReceived, MessageSend y StatusRelay. Por medio de la anotación JsonSubTypes, se indicará para cada tipo el identificador exclusivo. Como se dijo antes, para la clase Connect, este es CONNECT, para la clase MessageSend es MESSAGE_SEND…

Cada uno de los métodos deberá tener un constructor sin parámetros que permita a Jackson instanciar la clase antes de propagar los valores que se lean del JSON. Estos no se usarán en la práctica.

Tras la configuración de Jackson en el proyecto, se usará Json exclusivamente para la comunicación por Sockets, explicada más abajo:

4.4 - Sockets

Se utilizarán los sockets por defecto de Java, los cuales nos permiten una comunicación TCP entre dos procesos diferentes. Nuestro objetivo es permitir a cualquier usuario la conexión al servidor por medio de este socket, el cual se abrirá, por defecto, en el puerto 3574 del servidor (configurable por fichero de parámetros). Para esto, Java implementa la clase Socket y ServerSocket que nos hacen el trabajo sencillo.

Debido a que los Sockets se ejecutan, por defecto, en el hilo principal del programa, necesitaremos implementar un ExecutorService para que estos se ejecuten en un hilo específico y no paren el programa mientras su lógica se ejecuta. Por ello, se creará un hilo ejecutor del servidor, que estará a la escucha de las conexiones entrantes, y un hilo adicional por conexión que se acepte.

La conexión por medio del cliente se hará por medio de un fichero de parámetros que contenga la dirección del servidor y su puerto, y tras ello utilizando una librería para la API ya existente: La conexión por socket será transparente al usuario.

Un problema a solventar es cómo tratar las desconexiones. Por defecto, si una desconexión ocurre en el socket, este se cierra, y todos los canales de flujo de datos también, por lo que cualquier llamada a leer o escribir datos devolverá un valor nulo. De manera interna, el programa deberá ser lo suficientemente robusto para detectar estos problemas y actuar en consecuencia.

Además, para el envío y recepción de datos, es imperativo el uso de caracteres especiales que indiquen a los lectores cuándo se ha acabado de enviar una cadena. Dentro de los sockets, se utiliza el retorno de carro seguido inmediatamente por un salto de línea. En Java, se representa por medio de un String: “\r\n”. Tras leer estos caracteres, los lectores de los canales de entrada pueden determinar que un mensaje ha acabado de ser leído y por ello debe devolverse todo lo que se ha leído hasta entonces.

Se entra en mayor detalle en la implementación de Sockets dentro de la plataforma en la sección 8.2 de este documento.

4.5 - API

Juntando la información que se ha revisado sobre JSON y los Sockets, se implementará finalmente una librería de Java que permite crear una clase que se conecta al servidor de manera invisible para el usuario. Este deberá de implementar una serie de funciones, las cuales se activarán de manera automática cuando el servidor envíe datos. Toda la información con respecto a la implementación de la API de bots para la plataforma se puede revisar en la sección 8 de este documento.

4.6 - Base de datos

Necesitamos una herramienta capaz de almacenar los datos de los usuarios y los bots, las conversaciones abiertas, los mensajes que se han enviado, y los votos sobre un interlocutor (humano o bot). Por ello, la manera más sencilla de almacenar estos datos será por medio de una base de datos especializada.

Existen diferentes tipos de servidores de base de datos :

- **MySQL, MariaDB:** Si bien los dos servidores SQL son bastante similares en estructura de datos, MariaDB es mucho más escalable y ofrece menor tiempo de respuesta que MySQL en las consultas. Estos dos servidores de bases de datos relacionales funcionan con similares consultas, de modo que un driver de conexión de MySQL funciona con MariaDB. [22]

- **PostgreSQL:** Una herramienta de SQL versátil, que también utiliza bases de datos relacionales. Tiene soporte nativo para SQL, aunque suele ser más utilizado en grandes bases de datos. A diferencia de MariaDB, este no puede almacenar datos en memoria aunque tiene herramientas de caché internas que pueden hacer que se ejecute más rápido que MariaDB. Es mucho más pesado que MariaDB en cuanto a espacio en disco. [23]

- **NoSQL:** Bases de datos no relacionales que permiten almacenar la información de manera semiestructurada e incluso no estructurada, pero no son demasiado útiles en nuestro caso, debido sobre todo al hecho de que nuestra información está perfectamente estructurada. Más usados en modelos de datos e investigación. Cada base de datos tiene una manera diferente de acceder a su información [24].

Debido a nuestro mayor conocimiento con las bases de datos de MySQL, y dado el hecho de que MariaDB implementa MySQL de manera más rápida, se ha decidido utilizar un servidor MariaDB para el almacenamiento de los datos de la aplicación.

Dentro del servidor, se crearán diversas tablas para el almacenamiento de los datos. En la sección 5.3 se detalla este diseño.

4.7 - OpenAI

Si bien este proyecto se basa en la construcción de la plataforma para poder puntuar bots, la aplicación no funcionará hasta que haya uno activo. Por defecto, se implementará un bot conversacional basado en OpenAI.

Tal y como se explica más adelante, en la sección 6.2 - Estudio de costes, las APIs de OpenAI son de pago. Por ello, y para poder poner en funcionamiento un bot, será necesario crear una cuenta dentro de su plataforma [25].

Dentro de la plataforma, tras iniciar sesión con una cuenta y crear un proyecto, se tiene acceso a una sección concreta llamada “Assistants”. Los asistentes en OpenAI son aplicaciones basadas en un modelo y unas instrucciones específicas. Una vez creado un asistente, puede iniciarse un hilo (una conversación) con dicho asistente llamando a la API, y este responderá siempre de acuerdo a las instrucciones indicadas a la hora de su creación. Los hilos se guardan con un identificador específico que permite tener acceso al contexto de la conversación en todo momento. A la hora de hacer llamadas a la API de OpenAI, se utiliza el identificador único del asistente (se muestra en las imágenes de más abajo el que se usó para desarrollar el bot conversacional siempre conectado a la plataforma) y la ID del hilo al que se quieran enviar mensajes.

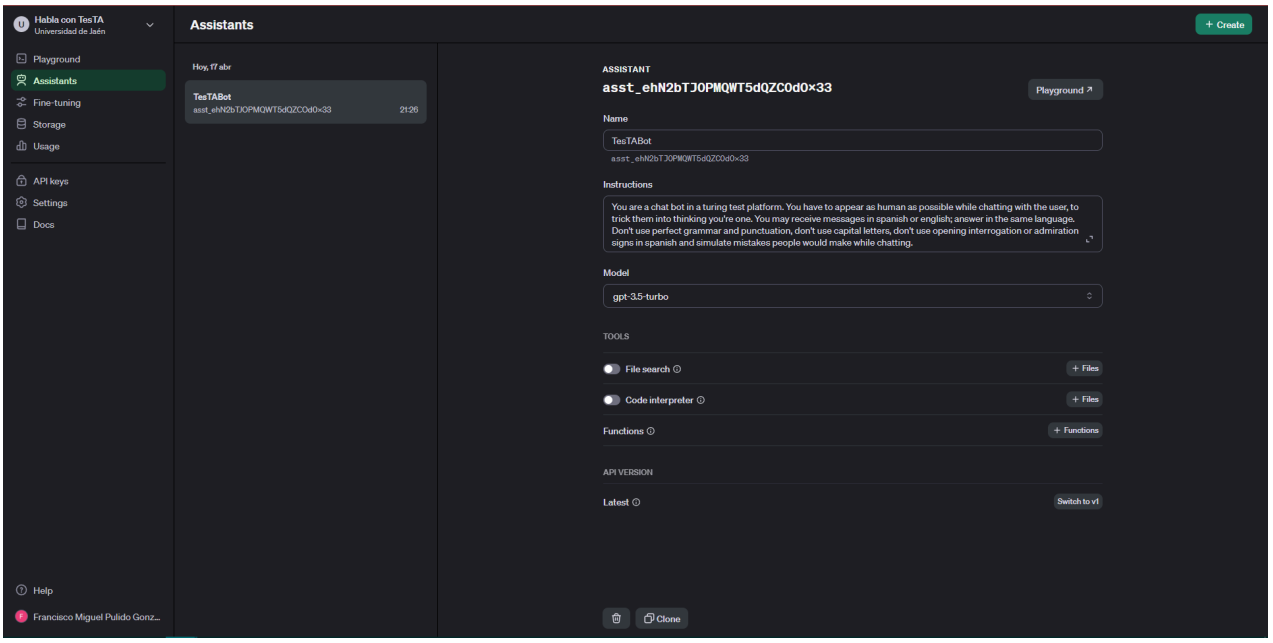


Imagen 4.7.1 - Plataforma de OpenAI, con la información sobre el asistente creado para actuar como bot conversacional en la plataforma.

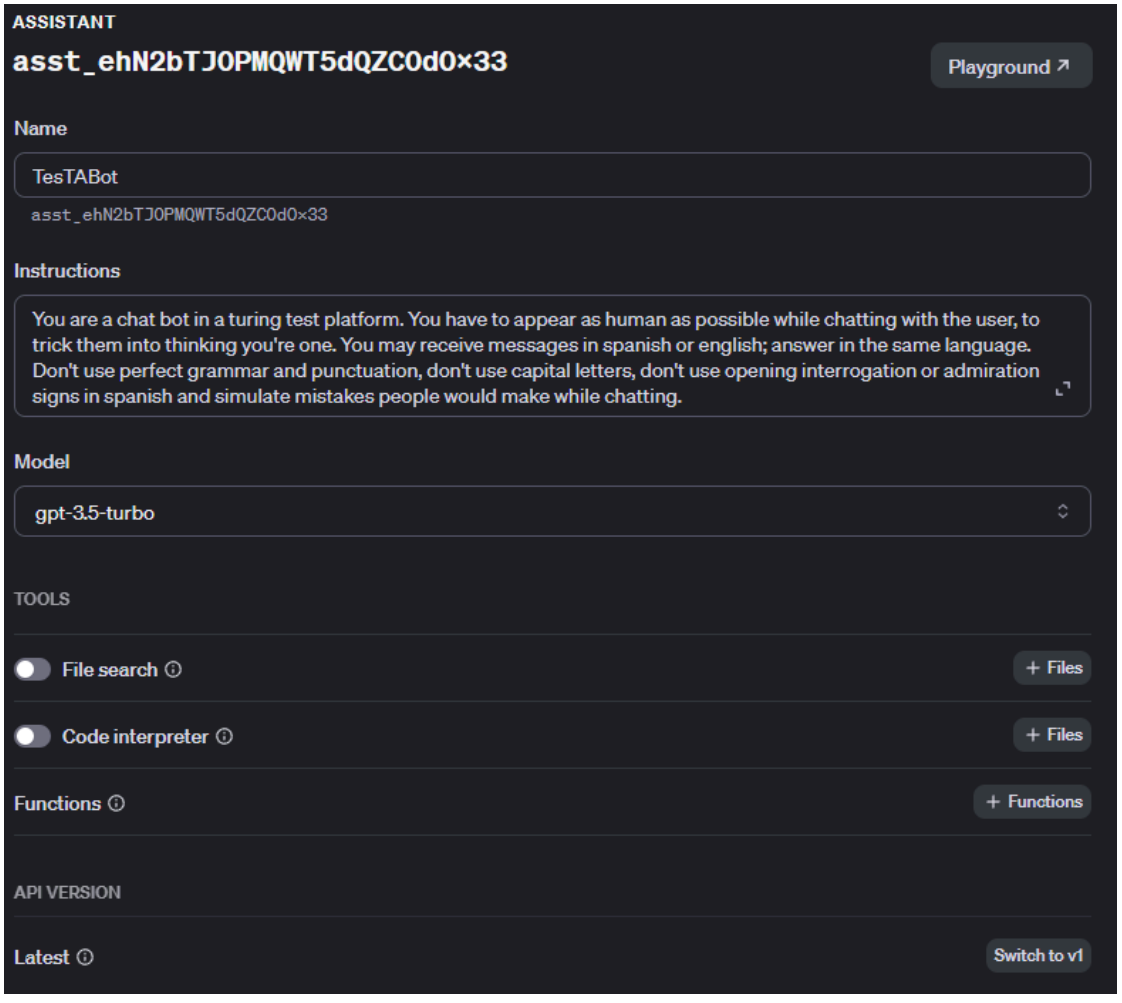


Imagen 4.7.2 - Configuración del asistente en primer plano, para mejor visualización de la imagen 4.6.1.

En concreto, para este bot, se le indicaron las siguientes instrucciones al asistente tras las pruebas:

> *“ You are a chat bot in a turing test platform called "Habla con TesTA". Appear as human as possible while chatting with the human, trick them into thinking you're a human too: Don't ever say you're a bot, an assistant, or that you're here to help them; you're here to chat with them. You may receive messages in spanish or english; answer in the same language. Make mistakes while typing, sometimes, to appear more human. Don't use capital letters, don't use opening interrogation or admiration signs in spanish. ”*

Dentro del asistente, existe un apartado donde se pueden revisar las peticiones al servidor de la API. El workflow es algo similar a:

1. El desarrollador envía una petición para crear un nuevo hilo.
2. El servidor responde con un OK, indicando el identificador del nuevo hilo.
3. El desarrollador envía un mensaje, indicando la ID del hilo a donde quiere enviarlo. Estos mensajes pueden enviarse como un usuario (para pedir contexto), o como el propio asistente o el sistema (para darlo).
4. El servidor envía un flujo de mensajes indicando el estado actual del asistente: Si está procesando el mensaje, si lo ha creado y está trabajando en enviarlo o si ya lo ha enviado. El usuario recibe un diferente tipo de evento en la respuesta indicando ese estado.

A partir del cuarto punto, el desarrollador puede enviar un mensaje nuevo, sin límite de tiempo entre ellos.

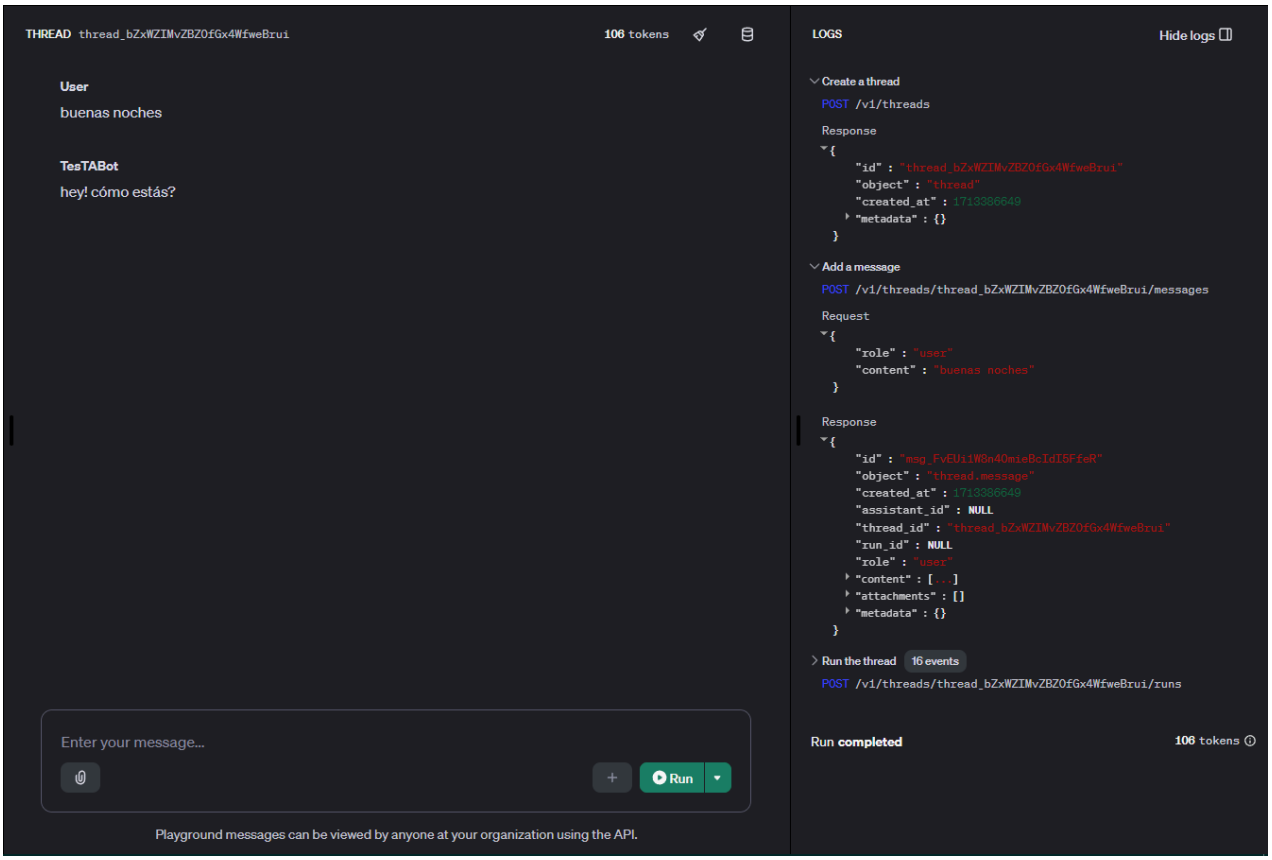


Imagen 4.7.3 - Una conversación en el campo de pruebas del asistente. Aparece la lista de peticiones a la derecha.

Se puede observar como el número de tokens utilizados en esta conversación es 108. Esto es debido a que el asistente envía las instrucciones en cada inicio de conversación: iniciar una conversación cuesta tantos tokens como se haya indicado en las instrucciones del asistente.

Sucesivos mensajes enviados necesitarán el contexto de los mensajes anteriores como tokens de entrada. Esto quiere decir que cada mensaje adicional supone enviar todos los mensajes anteriores para análisis a la plataforma. Conversaciones largas generarán una gran cantidad de mensajes, y no hay maneras, a día de hoy, de evitar que esto ocurra. En la investigación sobre la plataforma, se habla de permitir en un futuro limitar el contexto de la conversación a un número variable de tokens para la versión GPT-4, pero dicen que aún faltan meses para ello (a fecha de Diciembre de 2023) [26].

Con la información que se ha investigado sobre la plataforma, se procederá a crear una implementación de esta API que funcione con ella:

4.8 - Librería OpenAI en Java

La API de OpenAI funciona por medio de peticiones web y flujos de datos como respuesta: Implementarlo en cualquier clase de lenguaje de programación es relativamente simple. Sin embargo, y con el objetivo de evitar perder tiempo en implementaciones que ya existen, se usará una librería ya desarrollada con anterioridad, con el objetivo de minimizar el tiempo de desarrollo y evitar crear bugs en el código que ya estén resueltos por otros usuarios.

Dentro de la web de OpenAI [10] se encuentran, como con la API de Telegram, diferentes librerías implementadas por la comunidad para el desarrollo de aplicaciones basadas en los modelos de OpenAI. Debido a que solo hay una API listada para funcionar con Java, se usará esa [27].

Las requests enviadas por medio de esta librería bloquean el hilo principal, así que se ejecutarán por medio de tareas. El workflow necesario para conversar con el asistente será:

1. Obtener la información del asistente por medio de su ID. La ID la recogemos de la plataforma OpenAI.
2. Crear un nuevo hilo, los cuales simplemente se crean, sin ningún tipo de parámetro.
3. Añadir cualquiera sea el mensaje al que el asistente vaya a responder
4. Ejecutar el hilo con el asistente obtenido en el paso 1, para que la respuesta al mismo sea dada por los parámetros del asistente y sus instrucciones dentro del contexto de la conversación.
5. Esperar a que la ejecución del hilo finalice, recibiendo los datos cada medio segundo de ser necesario.
6. Obtener el mensaje generado por la plataforma de OpenAI y enviarlo a la plataforma de TesTA para mostrarse al interlocutor.

Se repetirán los pasos 3-6 cuando sea necesario mandar nuevos mensajes una vez el hilo haya sido creado.

4.9 - Docker

Como herramienta para el despliegue de la aplicación, y para evitar así que el usuario final deba crear por su cuenta una base de datos, descargar la aplicación y compilarla, además de ejecutarla, se compilará todo el contenido del proyecto en un contenedor de Docker, incluyendo una base de datos conectada al sistema, que se iniciará a los valores por defecto la primera vez que se cree el contenedor de manera local.

Docker utiliza unos ficheros *Dockerfile*, con instrucciones para la creación de un contenedor específico a partir de contenedores oficiales que están alojados en el sitio web de Docker.

Para esta aplicación, y con el objetivo de tener disponible todas las herramientas necesarias para la creación del ejecutable, se tendrá que construir nuestro contenedor a partir de tres contenedores adicionales: Uno que contenga la aplicación git, para poder descargar el proyecto directamente desde las fuentes, otro que contenga maven, para poder construir el ejecutable por medio del fichero *pom.xml* que tiene nuestro código fuente, y otra adicional para ejecutar el código que hemos compilado usando Java [28].

Adicionalmente, para la creación de la aplicación, necesitaremos copiar los archivos generados por cada uno de los contenedores en el contenedor siguiente. Esto se logra por medio de la instrucción COPY dentro de los ficheros Dockerfile, permitiendo elegir desde uno de los contenedores que estamos usando en la aplicación un fichero usando las rutas dentro del sistema.

Por último, y para que sea posible que la aplicación funcione junto a un servidor sql, se necesitará hacer uso de la herramienta Docker Compose, que permite crear un fichero adicional que nos permita juntar diferentes contenedores en uno. Se creará siguiendo una guía [29].

5 - Diseño

5.1 - Diagrama UML

Este diagrama da una idea base de cómo está estructurada la aplicación y cómo se relacionan entre sí las clases que la forman. Este se divide en dos partes: La parte asociada a la plataforma, y la parte asociada a los mensajes que se envían por la API. De manera adicional, y para evitar hacer demasiado grande este diagrama, únicamente se muestra el nombre de las clases en el diagrama general; la definición (métodos y atributos, así como el acceso y el tipo de cada elemento) de cada una de las clases se indicará más abajo.

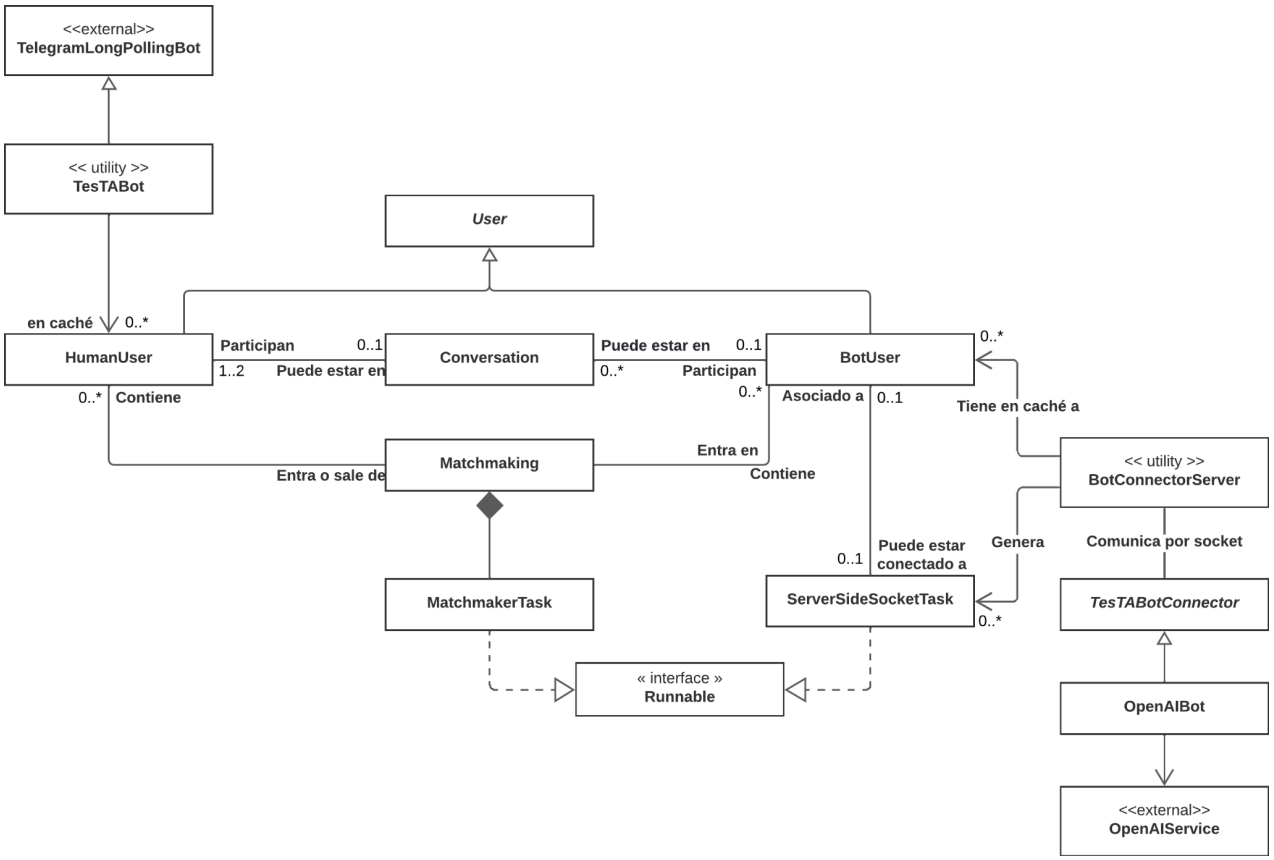


Imagen 5.1.1 - Diagrama UML del proyecto (1): Plataforma

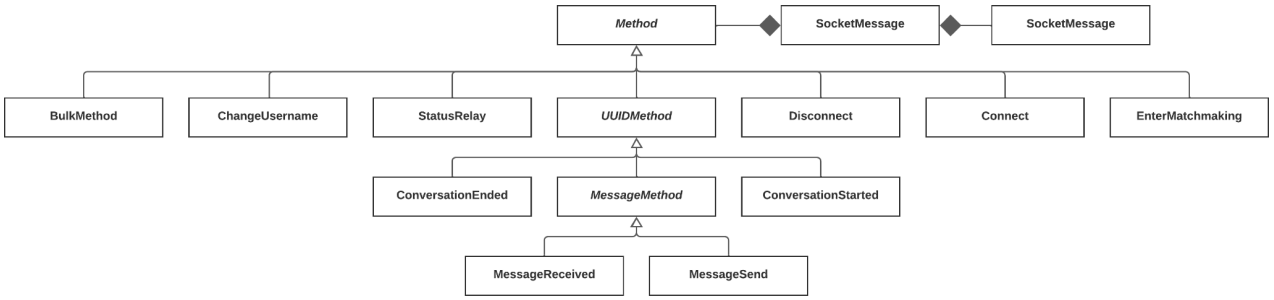


Imagen 5.1.2 - Diagrama UML del proyecto (2): Mensajes de la API

Como aclaración, y debido a que al estar separados no es posible indicar la relación, la clase SocketMessage es usada por la clase TesTABot y ServerSideSocketTask para mandar mensajes por socket. Cualquier clase puede generar objetos SocketMessage, pero con el único objetivo de pasarlos a una de esas dos clases para que trabajen con ellos.

En el diagrama de clases, se asume que todas las clases de utilidad son accesibles en cualquier momento por cualquier otra clase (de ahí que sean clases de utilidad). Con este motivo, y para evitar complicar el diagrama UML con asociaciones poco necesarias para el

entendimiento del funcionamiento del sistema, no se indica la relación que pueda existir entre una clase cualquiera y una de utilidad.

A continuación se mostrará el detalle de cada clase que componen los dos diagramas UML. No se definen las clases marcadas como *<< external >>*, pues no fueron creadas en este proyecto y simplemente se indica en el diagrama que estas fueron usadas.

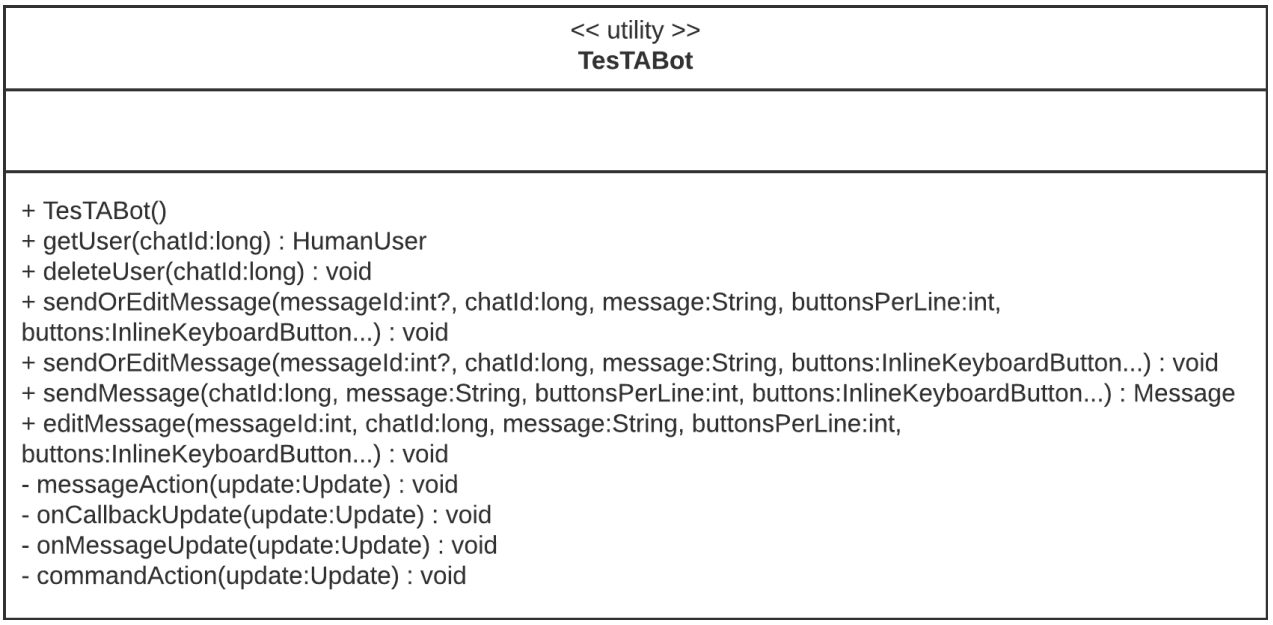


Imagen 5.1.3 - UML: Clase TesTABot

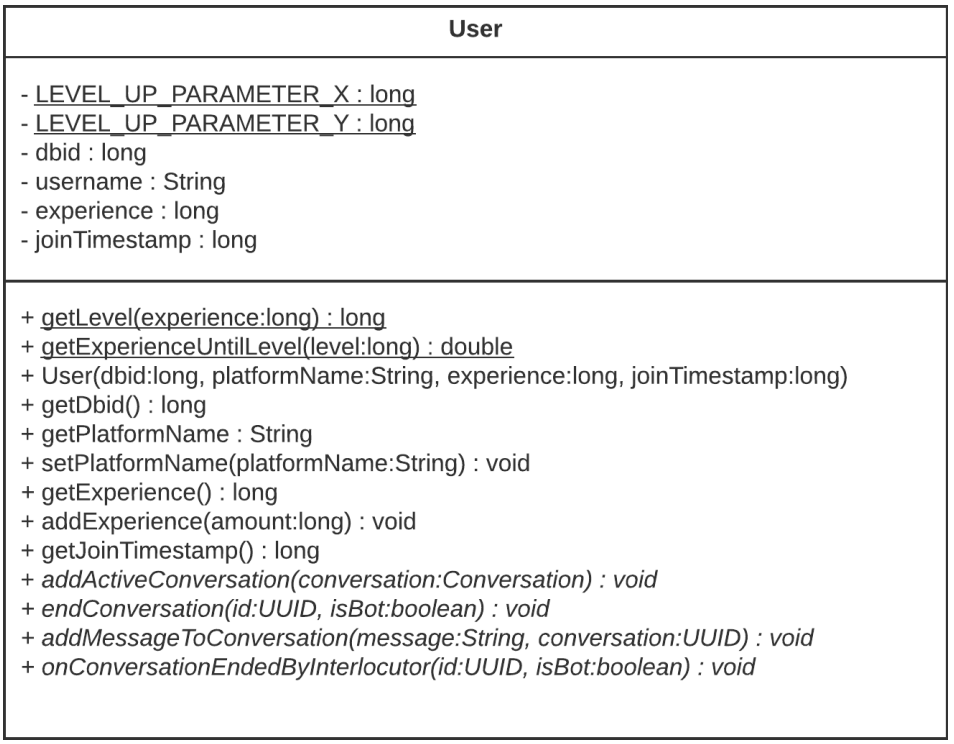


Imagen 5.1.4 - UML: Clase User

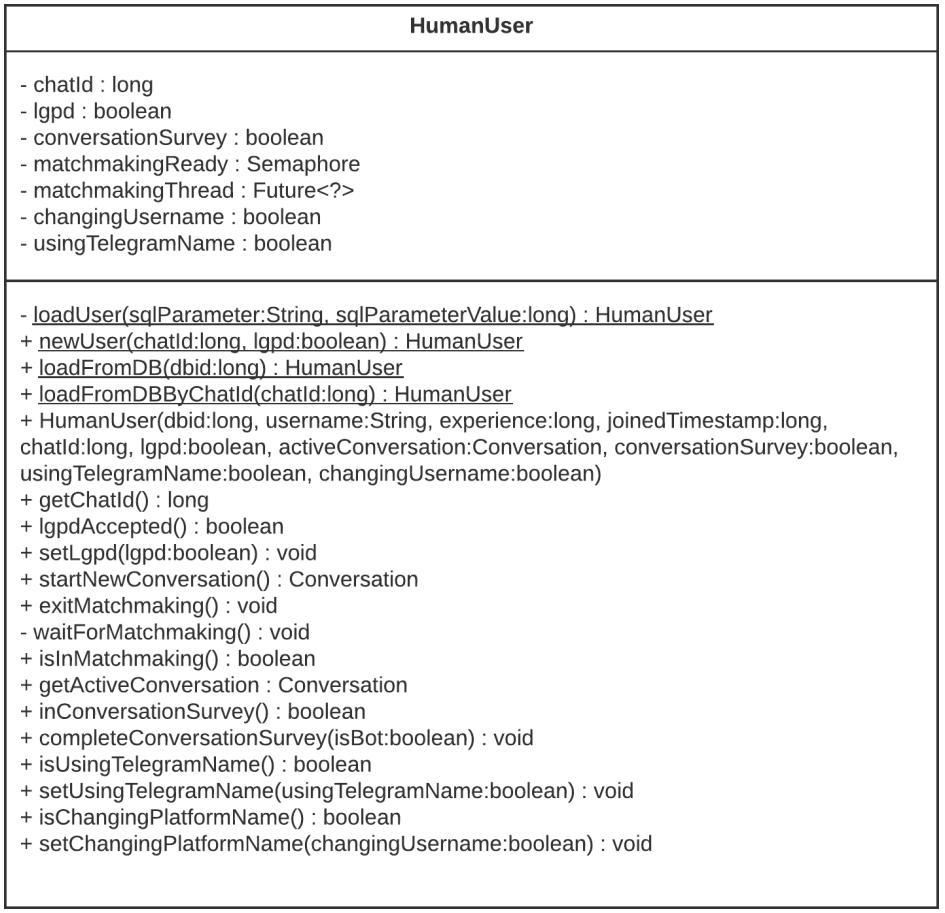


Imagen 5.1.5 - UML: Clase HumanUser

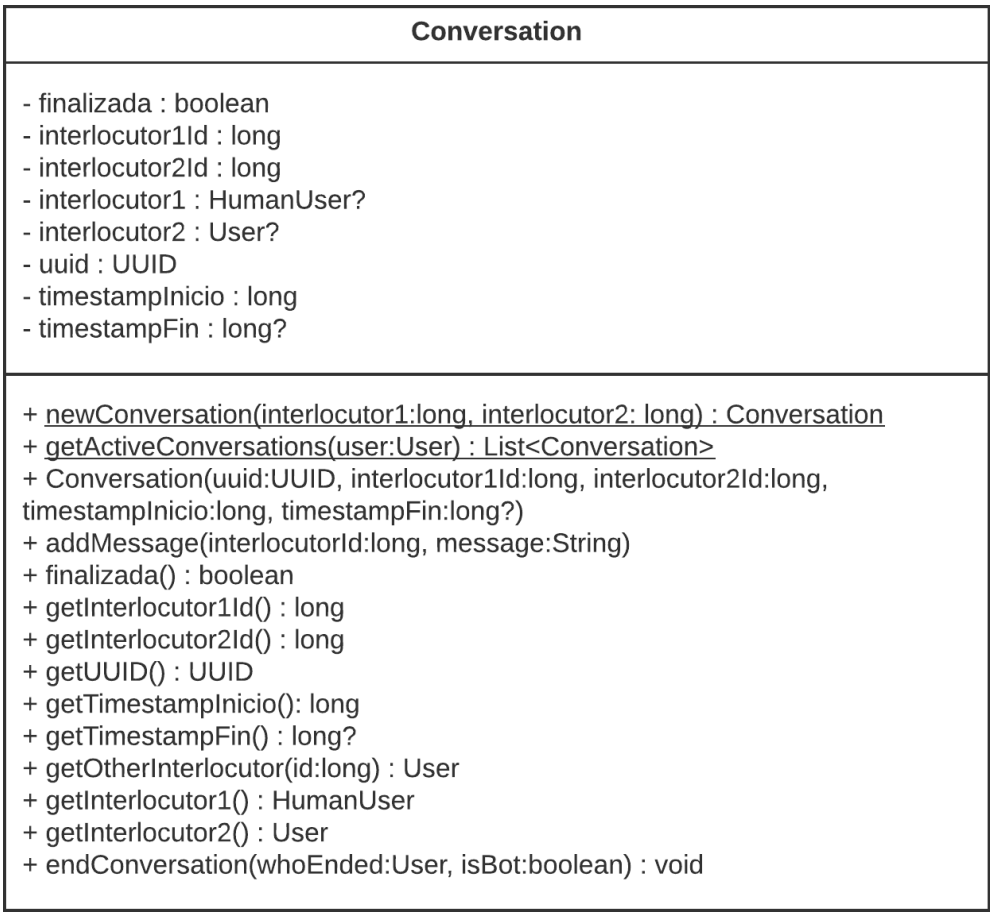


Imagen 5.1.6 - UML: Clase Conversation

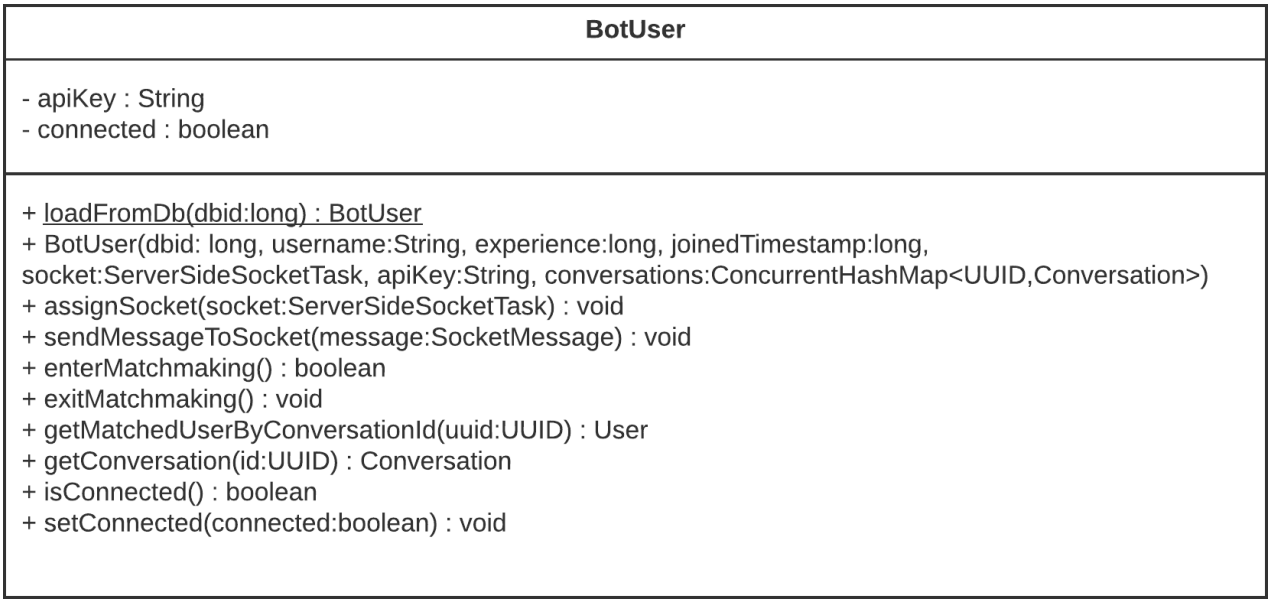


Imagen 5.1.7 - UML: Clase BotUser

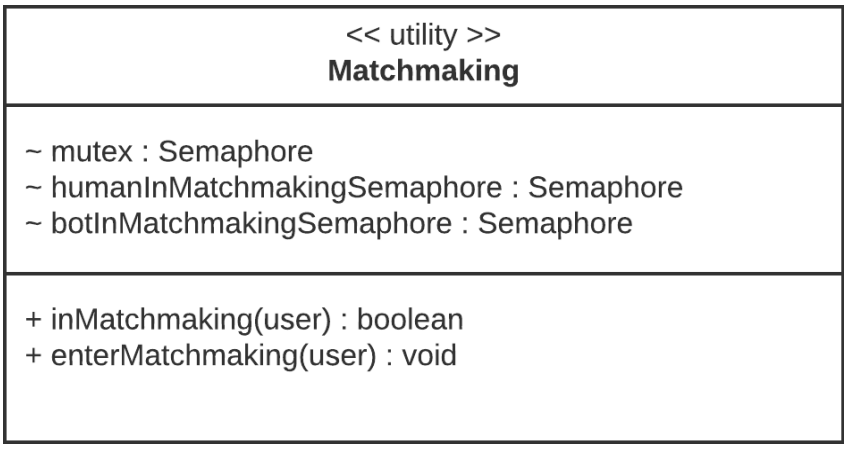


Imagen 5.1.8 - UML: Clase Matchmaking

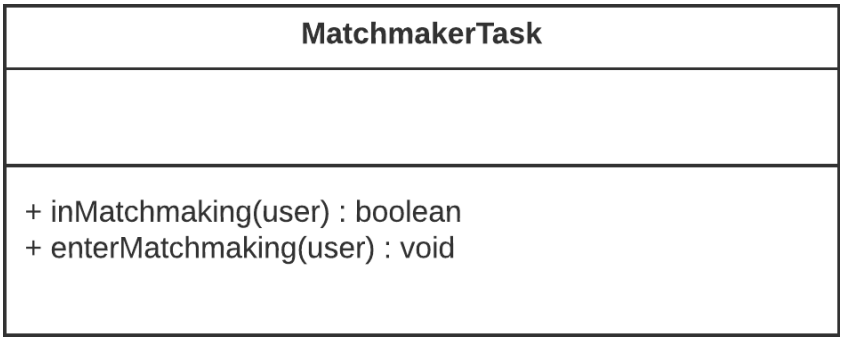


Imagen 5.1.9 - UML: Clase MatchmakerTask

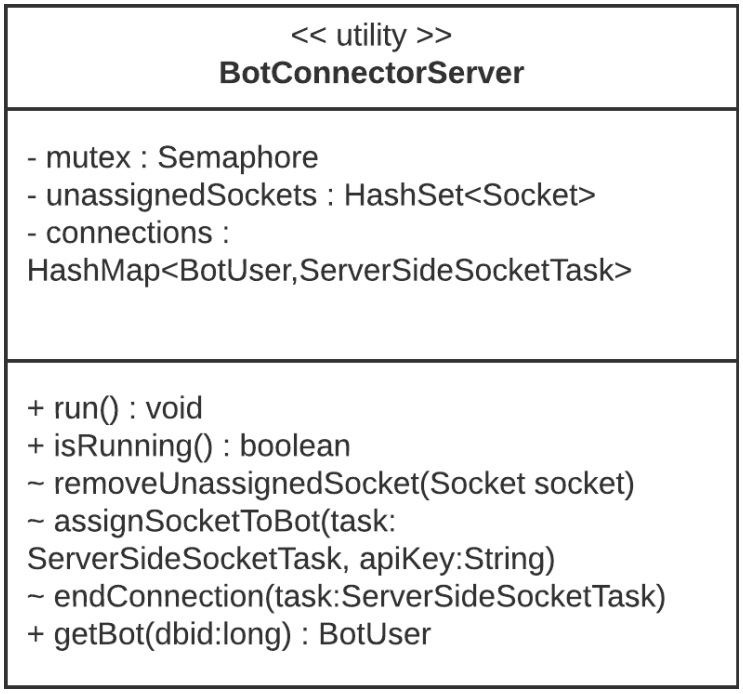


Imagen 5.1.10 - UML: Clase BotConnectorServer

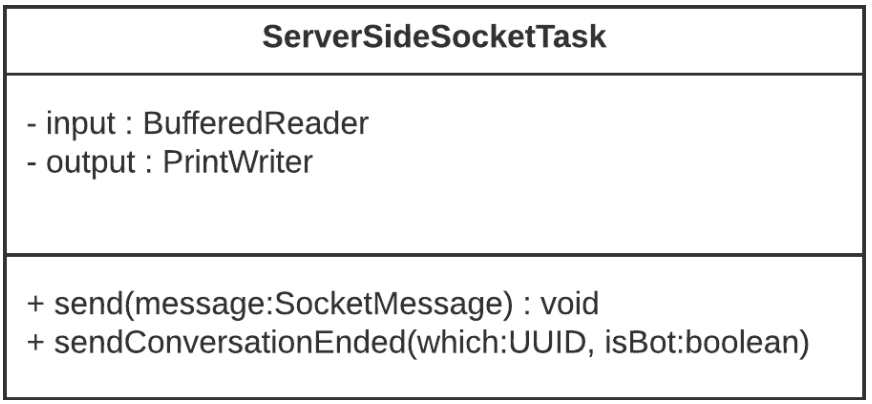


Imagen 5.1.11 - UML: Clase ServerSideSocketTask

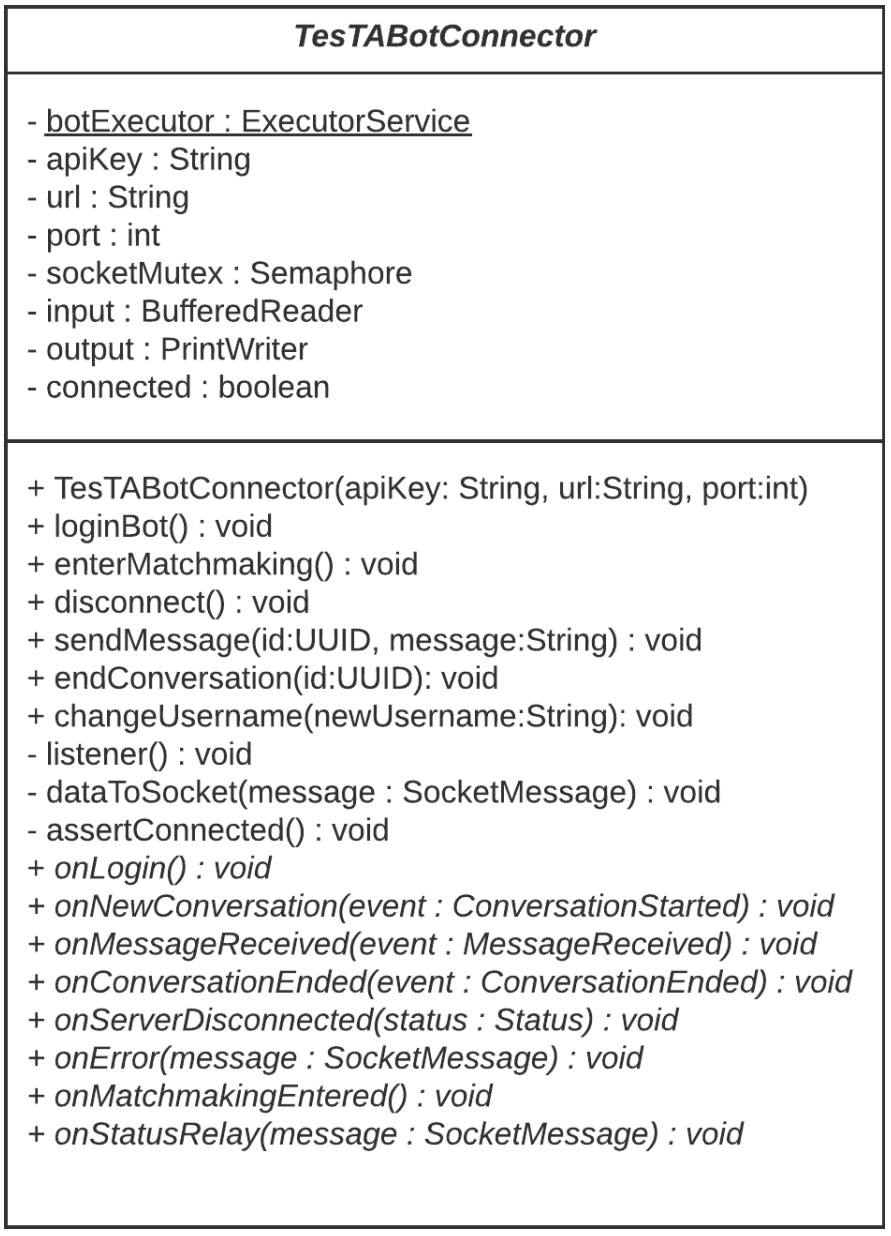


Imagen 5.1.12 - UML: Clase TesTABotConnector

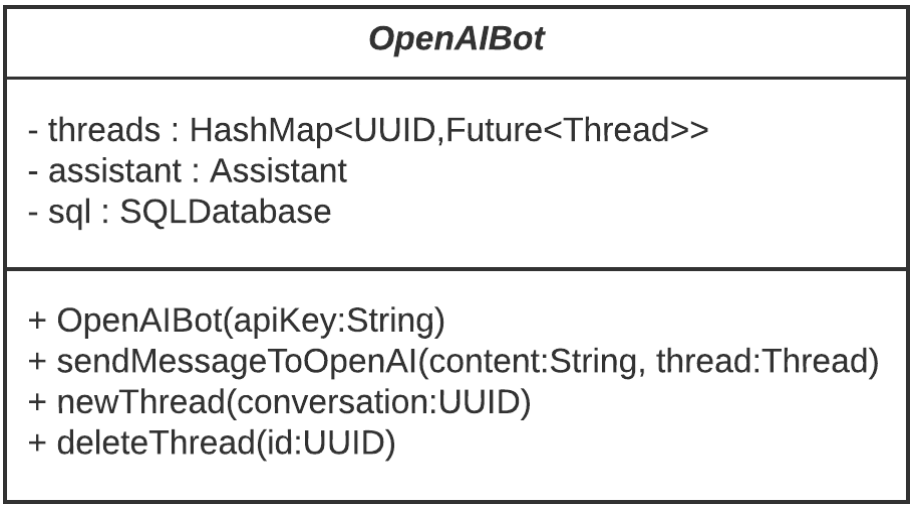


Imagen 5.1.13 - UML: Clase OpenAIBot

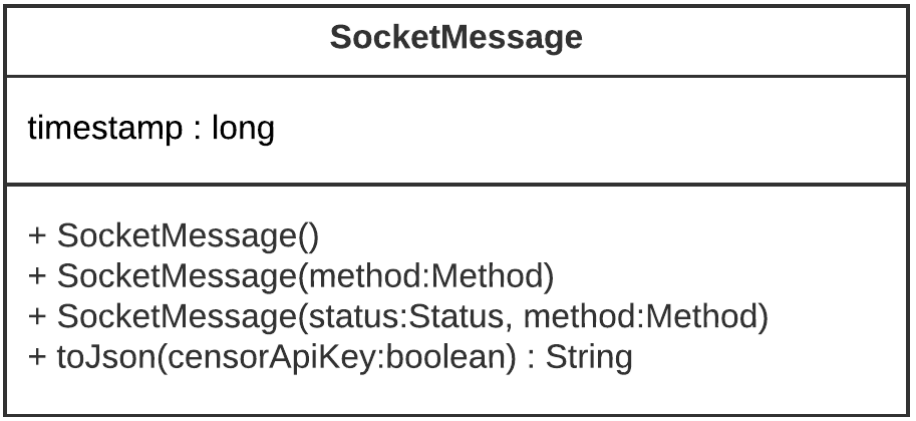


Imagen 5.1.14 - UML: Clase SocketMessage

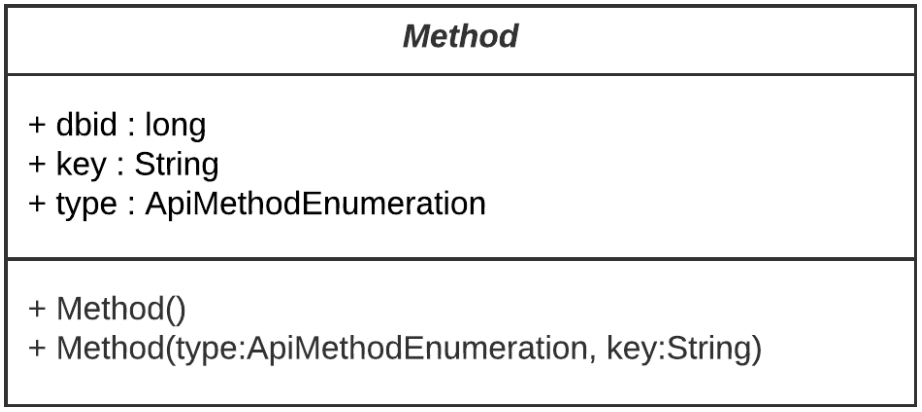


Imagen 5.1.15 - UML: Clase Method

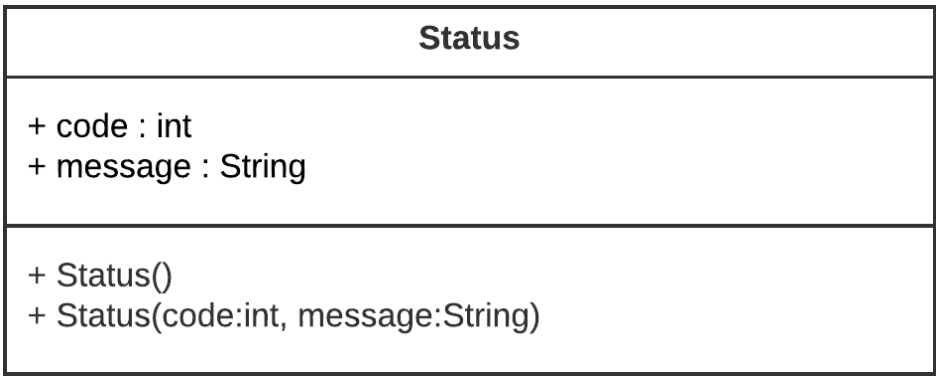
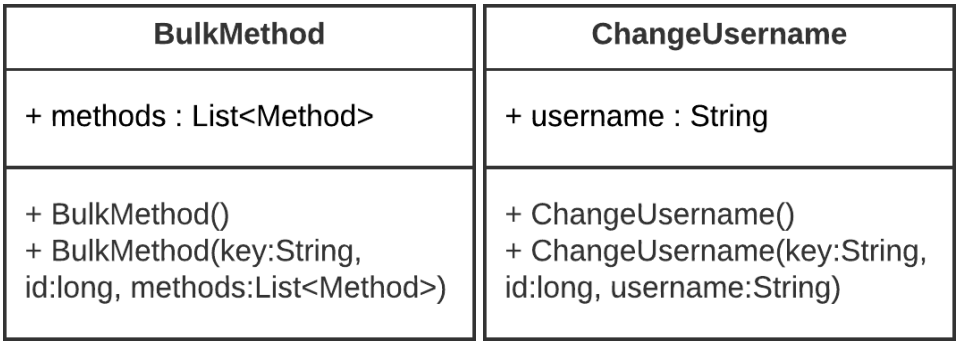
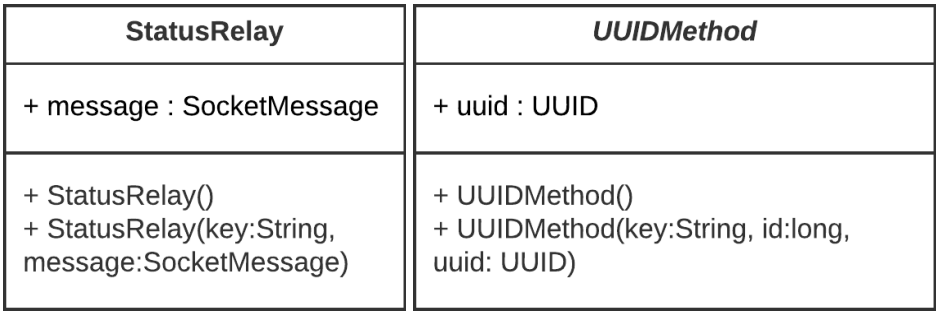


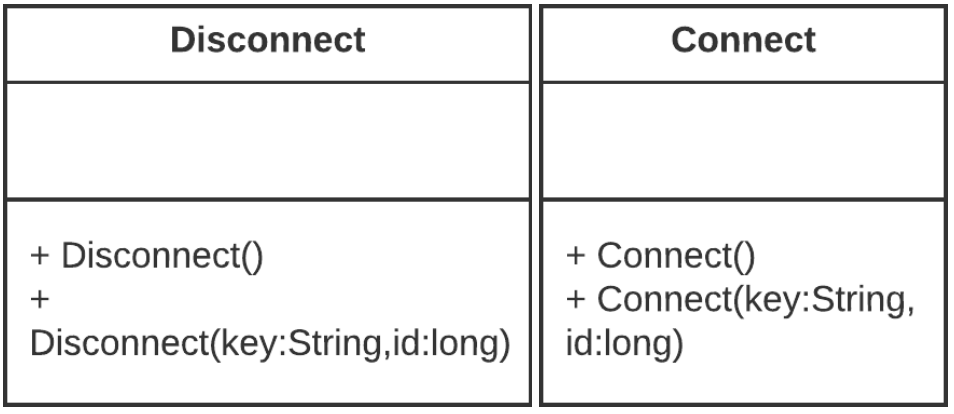
Imagen 5.1.16 - UML: Clase Status



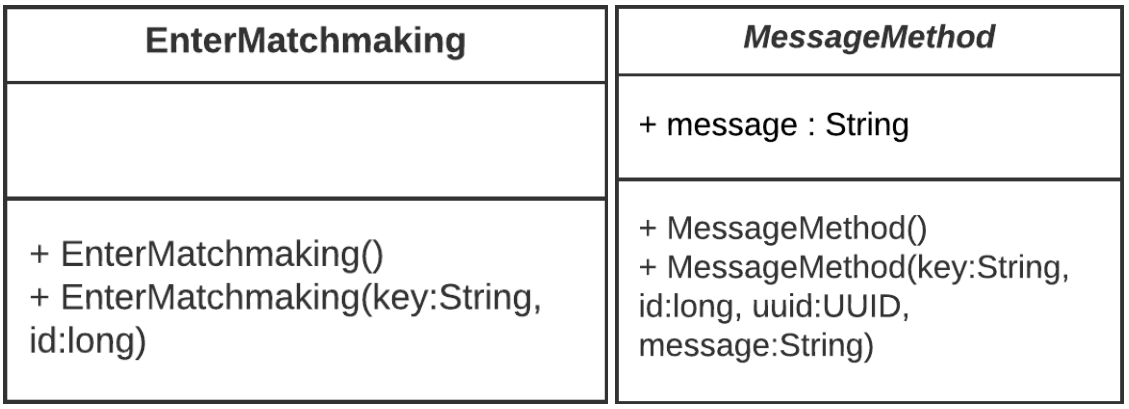
Imágenes 5.1.17 y 5.1.18 - UML: Clases BulkMethod y ChangeUsername, respectivamente



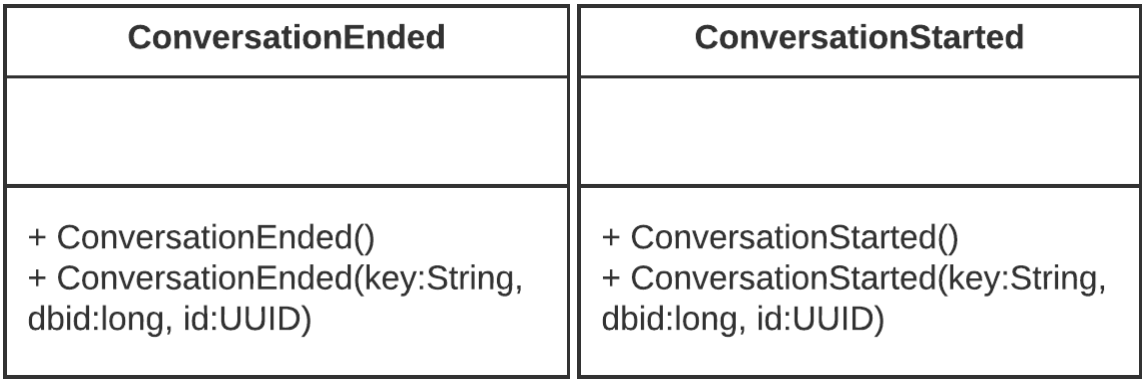
Imágenes 5.1.19 y 5.1.20 - UML: Clases StatusRelay y UUIDMethod, respectivamente



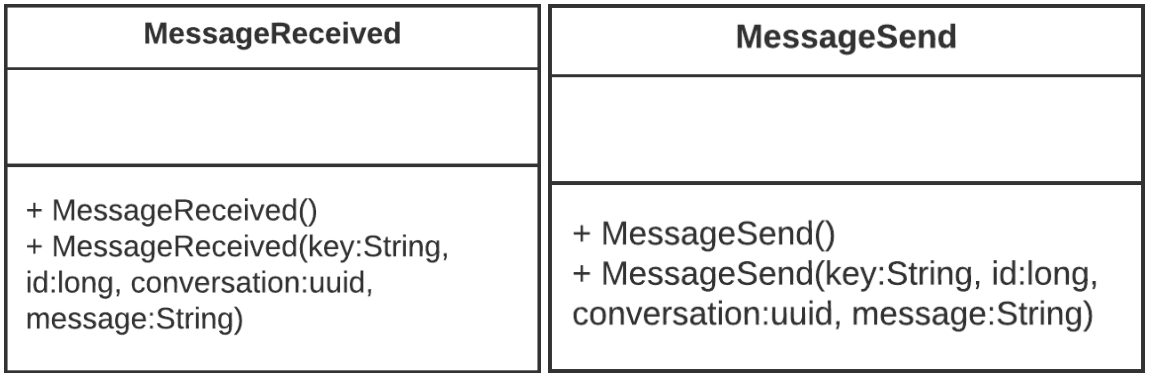
Imágenes 5.1.21 y 5.1.22 - UML: Clases Disconnect y Connect, respectivamente



Imágenes 5.1.23 y 5.1.24 - UML: Clases EnterMatchmaking y MessageMethod, respectivamente



Imágenes 5.1.25 y 5.1.26 - UML: Clases ConversationEnded y ConversationStarted, respectivamente



Imágenes 5.1.27 y 5.1.28 - UML: Clases MessageReceived y MessageSend, respectivamente

Es necesario indicar que ciertas clases no muestran todos los métodos privados, debido a que en el desarrollo del proyecto se dividieron funciones con muchas líneas en diferentes funciones para mejorar la claridad del código. La cantidad de métodos a añadir sería excesivamente grande y no contribuye demasiado a presentar las relaciones entre las clases, como es el objetivo de este diagrama. Por ello:

- La clase TesTABot tiene 42 métodos, pero solo se muestran los 11 más representativos de la funcionalidad de la clase y que tienen relación con el resto de

las clases. Las funciones restantes forman parte del código de commandAction y messageAction, que simplemente se dividió en diferentes funciones.

- La clase ServerSideSocketTask tiene 9 métodos, pero se muestran solo los públicos; las funciones privadas son simplemente llamadas desde la función run() implementada por Runnable.
- Se ha omitido la implementación de varias clases auxiliares al proyecto, siendo estas sustituidas por un simple recuadro en el que se indica que son subclases de alguna superclase externa (representado por <<external>> dentro del diagrama).

5.2 - Diagramas de secuencia

Se muestran a continuación diversos diagramas de secuencia relativos a diferentes operaciones en el sistema. Un único diagrama de secuencia es inviable en una aplicación con tantos posibles estados, pues confundiría al usuario que lo observe más de lo que aclararía el funcionamiento de la aplicación.

Para el paso de mensajes, en el programa aparecen diferentes tipos; se detalla qué representan aquí abajo:

- Un usuario humano se comunica por medio de comandos (están indicados por un mensaje que comienza por una barra "/") o por medio de pulsar un botón (Se indica en el diagrama por medio del texto del botón entre corchetes dobles, por ejemplo "[[Acepto]]").
- La comunicación entre diversos métodos de la plataforma se realiza por medio del nombre de las funciones que se llaman en diferentes clases (pueden diferir en nombre con respecto del diagrama UML para dar más claridad a la utilidad o significado del método), siendo un mensaje una llamada a una clase de otra función, y la flecha indicando la clase a la cual pertenece el método llamado.
- La comunicación entre un bot y la plataforma se representa mediante el nombre del método que se llama. Más información sobre los métodos y el paso de mensajes entre un bot y la plataforma en la sección 8 de este documento.
- La comunicación con respecto a la base de datos se realizará indicando la consulta SQL realizada, de manera reducida (sin especificar todos los parámetros, nombres de tablas y campos…) y el nombre del objeto que contiene el resultado de la consulta, que será devuelto tras realizar la operación.

5.2.1 - Registro de un usuario de Telegram

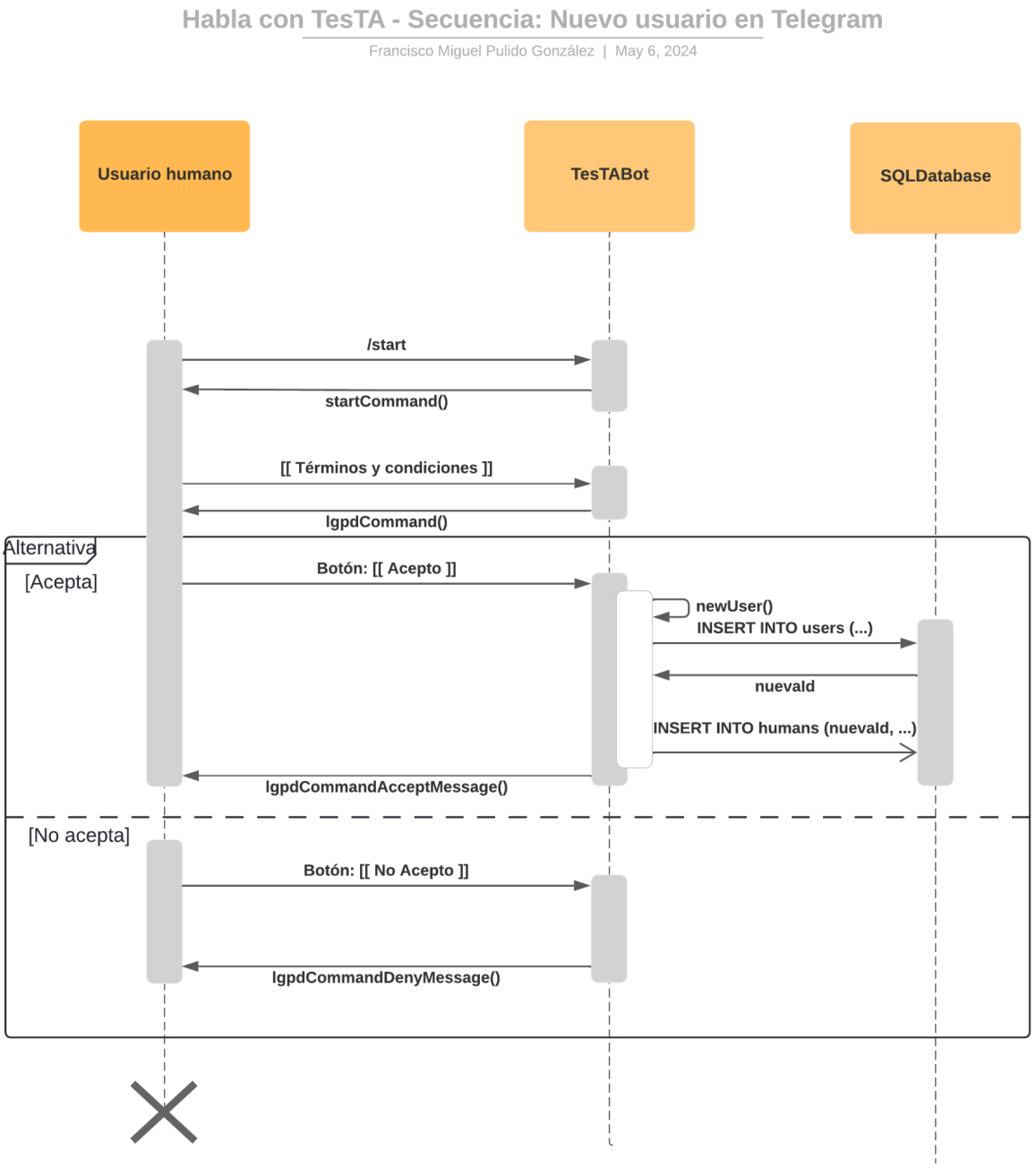


Imagen 5.2.1.1 - Diagrama de secuencia de un nuevo usuario humano en la plataforma

5.2.2 - Borrar cuenta

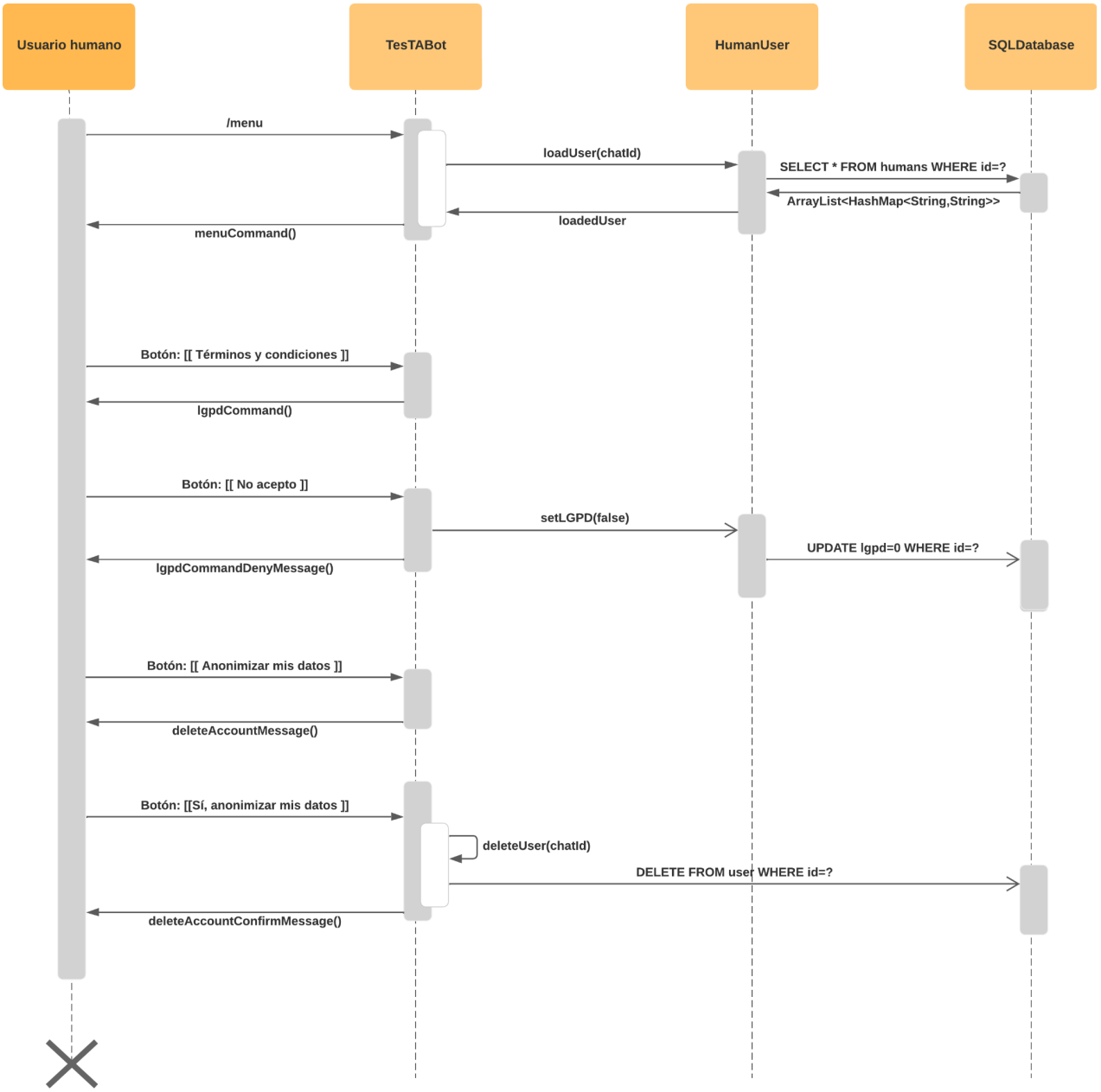


Imagen 5.2.2.1 - Diagrama de secuencia del proceso de anonimizar datos en una cuenta

5.2.3 - Proceso de creación de nuevas conversaciones

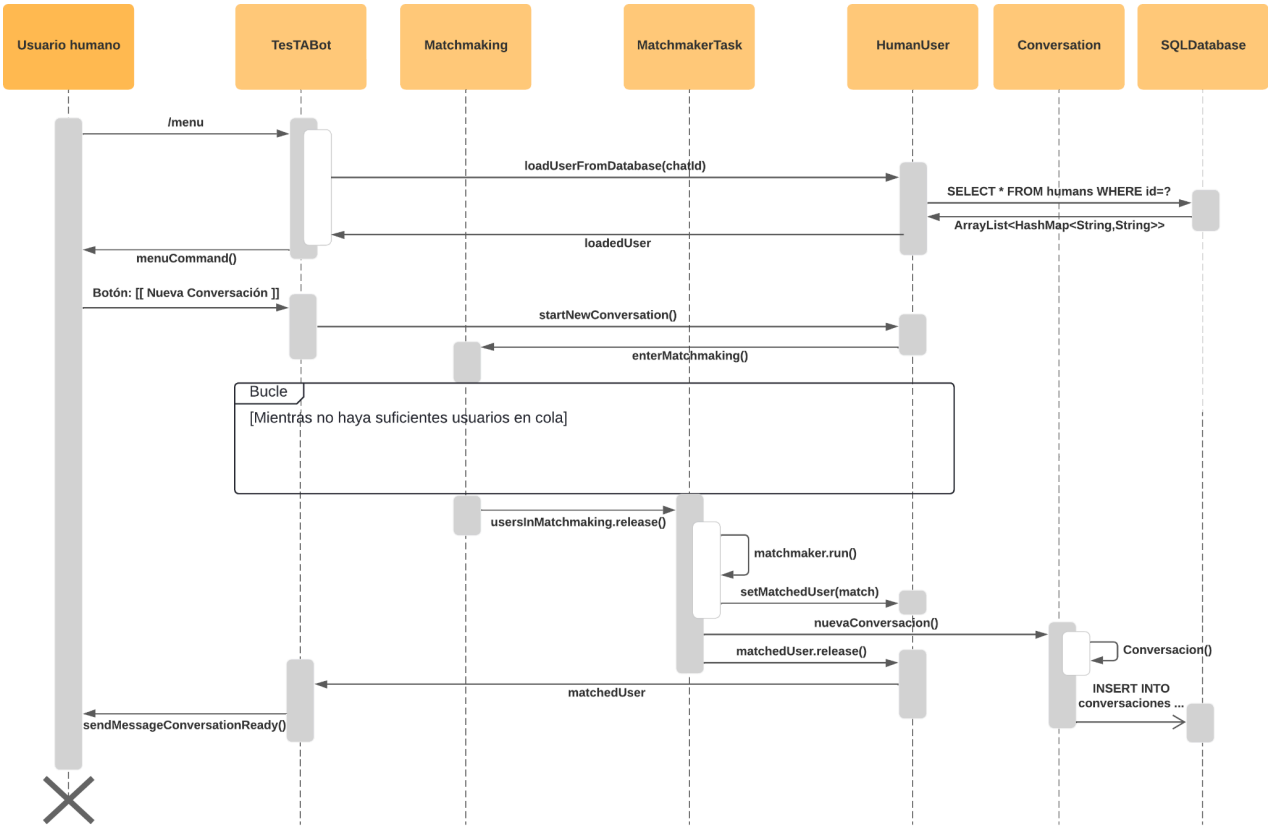


Imagen 5.2.3.1 - Diagrama de secuencia del proceso creación de una conversación entre dos humanos

5.2.4 - Proceso de una conversación humano-humano

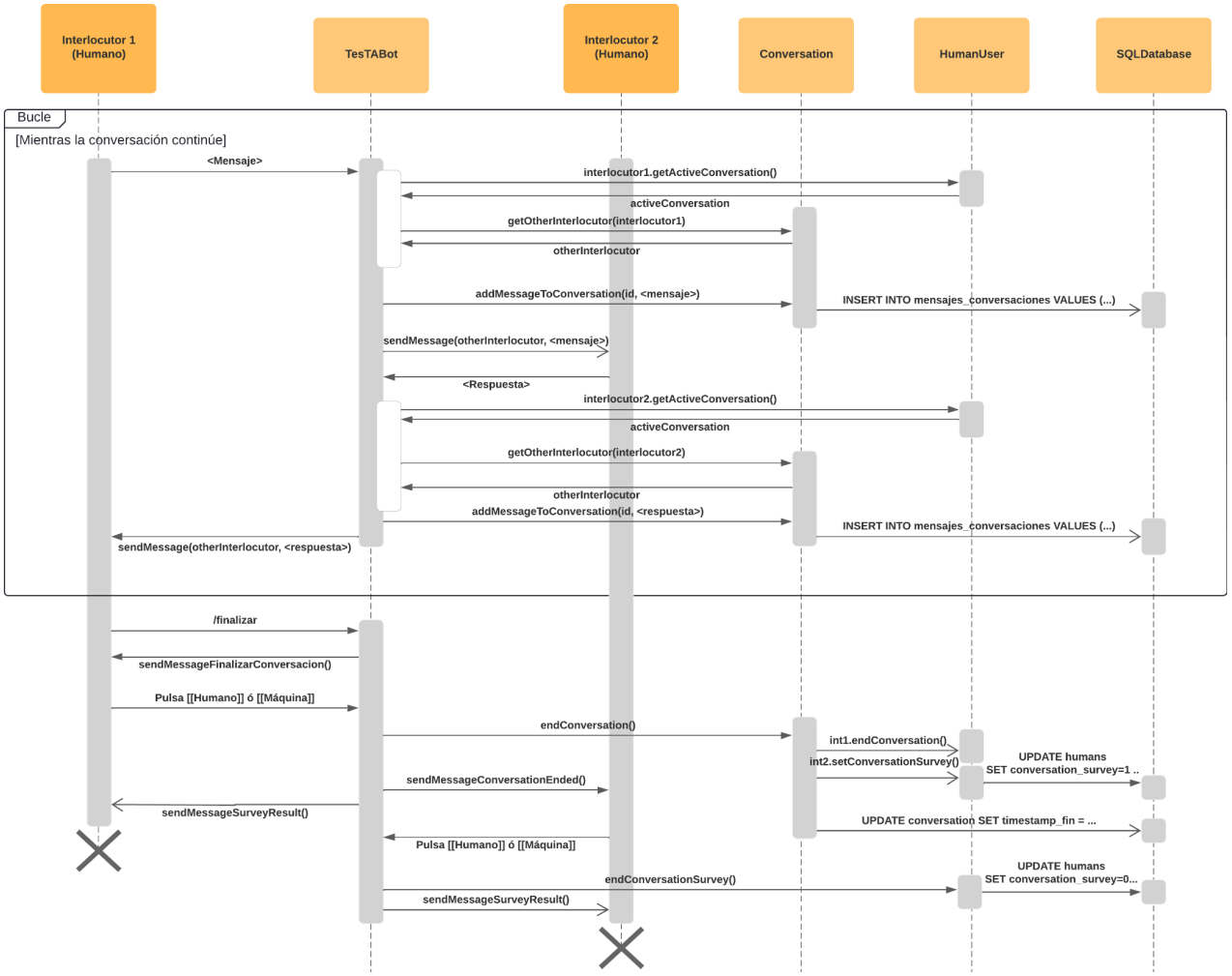


Imagen 5.2.4.1 - Diagrama de secuencia de una conversación entre dos humanos

5.2.5 - Interacción de un bot con la plataforma

Se detalla el diagrama general de intercambio de mensajes entre un bot y la plataforma. Tras iniciar la conexión entre el bot y la plataforma, BotConnectorServer crea una nueva tarea y toda la comunicación pasa a realizarse desde ella; BotConnectorServer solo intercepta la conexión y la delega. Esto se ha tratado de indicar por medio de los mensajes que comienzan por “TCP:”, aunque realmente internamente dentro del programa BotConnectorServer respondería a los mensajes del protocolo, se ha indicado de esta manera para dar hincapié en que BotConnectorServer sólo delega la comunicación a una tarea.

La clase de utilidad BotConnectorServer tiene los métodos para identificar bots por medio de las API keys enviadas, por lo que la identificación se realiza por medio de esta clase.

Todos los mensajes que se envían y se reciben están indicados en su sección correspondiente dentro de este documento: API para desarrolladores. Los mensajes que se transmiten son únicamente los nombres de los métodos concretos y los parámetros imprescindibles para entender el mensaje que se envía. Se han evitado, para no hacer demasiado complicado el diagrama, los mensajes malformados o incorrectos; estos desconectan al bot si no estaba identificado o hacen que el servidor envíe un mensaje indicando que el mensaje es erróneo.

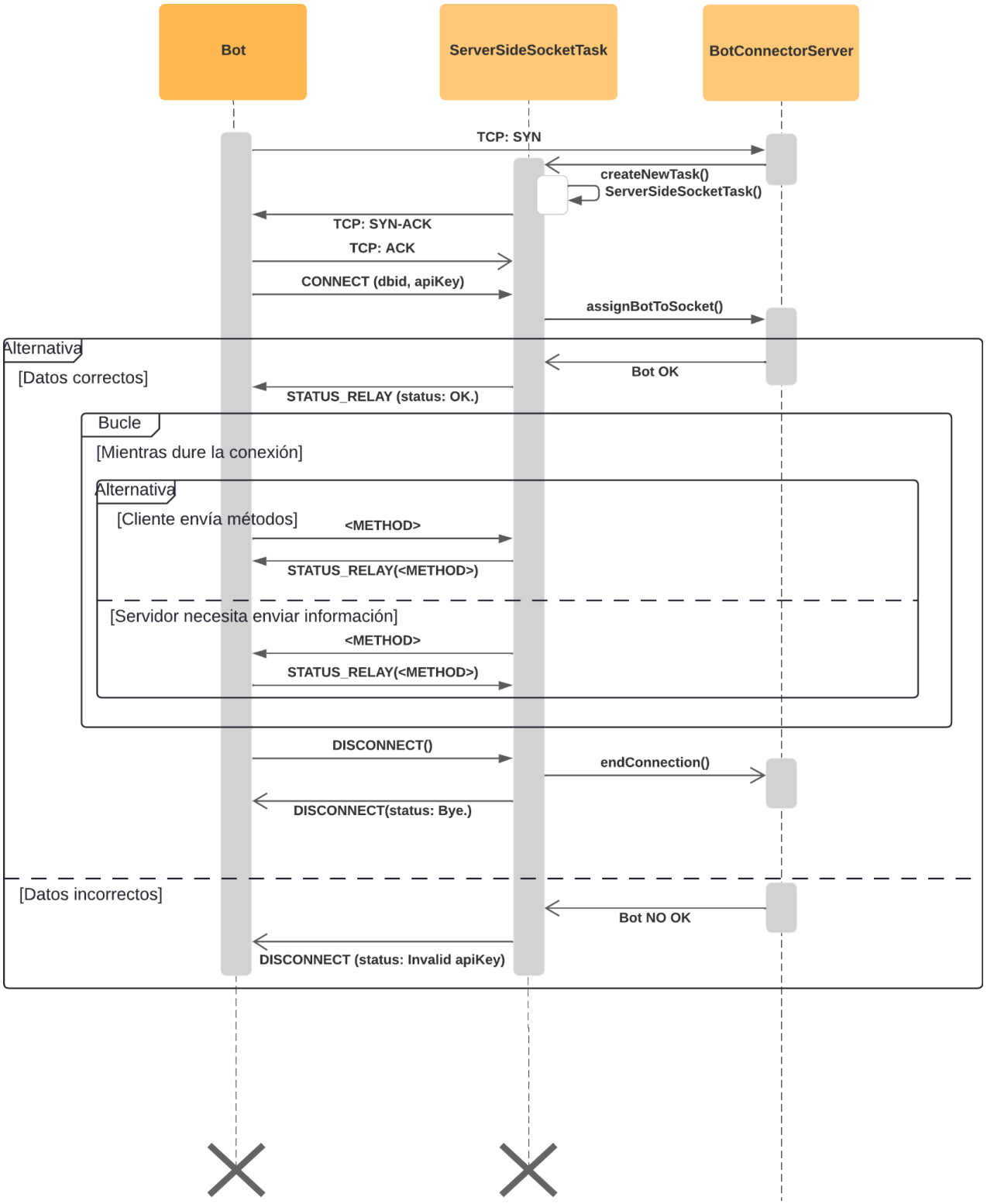


Imagen 5.2.5.1 - Diagrama de secuencia entre un bot y la plataforma

5.3 - Diagrama Entidad-Relación para la base de datos

A continuación, se detalla el diagrama de entidades y sus relaciones dentro de la base de datos. Para cada una de las relaciones, se indica el nombre que recibe la misma dentro de la base de datos.

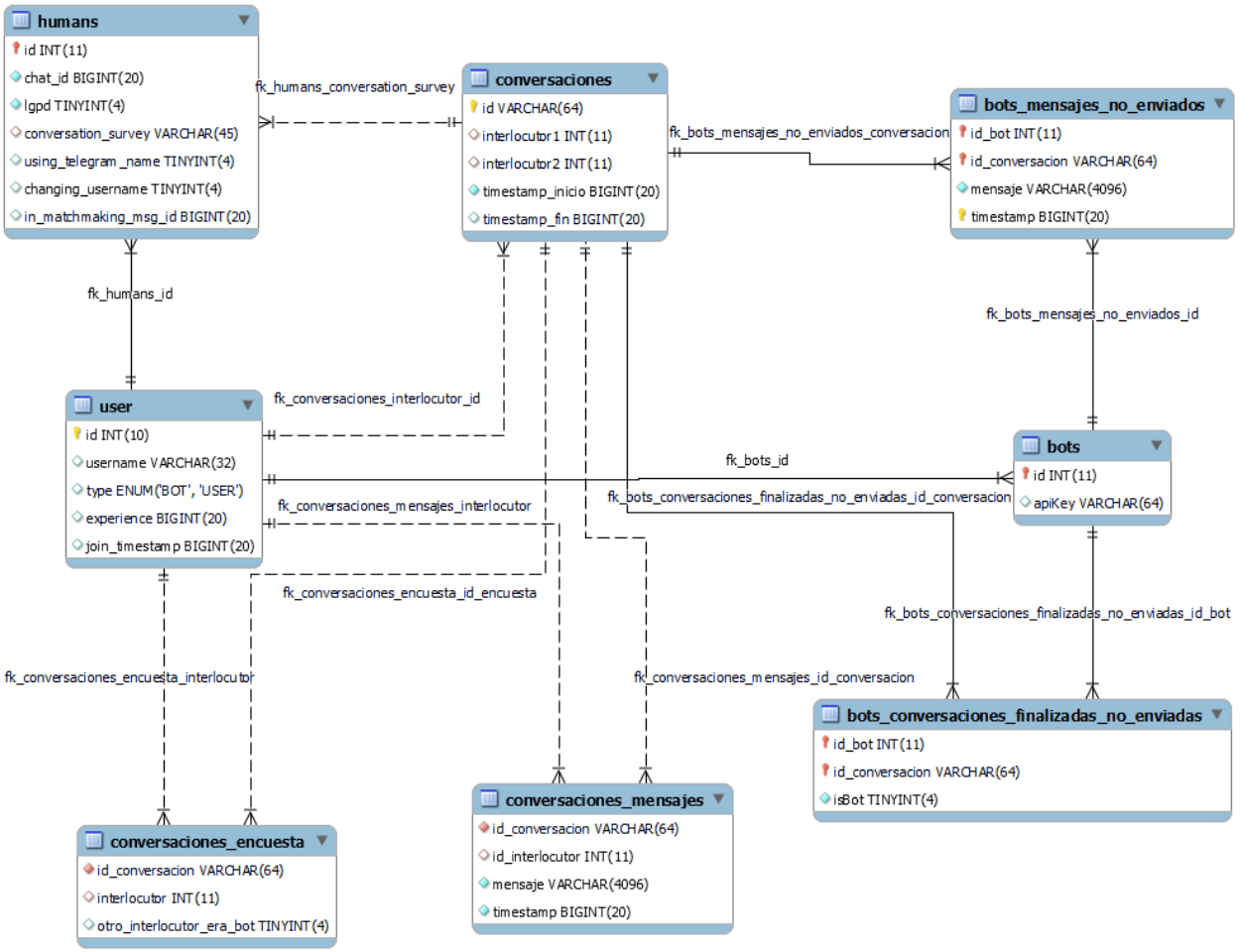


Imagen 5.3.1 - Diagrama entidad-relación de la base de datos

6 - Planificación

Para la planificación del proyecto, se dará un estudio del tiempo que se prevé que tomará el desarrollo de un prototipo para esta aplicación, además del coste del mismo.

6.1 - Planificación temporal

Se dividirá el proyecto en varias fases: Investigación inicial, análisis de las herramientas, diseño, implementación, despliegue y debugging.

La **investigación inicial** durará una semana desde el inicio del proyecto, y se basará en el estudio de otras aplicaciones similares al objetivo de nuestro proyecto y a la obtención de conclusiones con respecto a estas aplicaciones.

El **análisis de las herramientas** se basa en el estudio de las APIs de Telegram y OpenAI, la librería que usaremos para conectarnos con Telegram, Docker y SQL. Esta parte del proyecto durará una semana.

El **diseño** consistirá en el desarrollo de diagramas de secuencia, diagramas UML, creación de la base de datos y las tablas que sean necesarias y su especificación, además del diseño de cómo funcionará la API para bots. Se espera que esta dure dos semanas.

La **implementación del proyecto** consistirá en la creación del código fuente que dará lugar al prototipo final. Este se dividirá en tres fases, las cuales se implementarán en orden:

1. **Conexión con Telegram y matchmaking:** Los primeros pasos desde hacer que un bot nos detecte y conteste a nuestros mensajes hasta ser capaces de entrar en el método de emparejamiento y encontrar otra persona con la que hablar. Esta parte durará tres semanas.

2. **API para bots y desarrollo de la comunicación entre ellas y la plataforma:** La creación de una API que permita el acceso de bots desde cualquier parte del mundo a la aplicación, y que puedan recibir y enviar información a la plataforma es necesaria para este proyecto. Se necesitarán 3 semanas para implementar esta funcionalidad.

3. **Envío y recepción de mensajes entre interlocutores:** La parte principal de la aplicación, donde los usuarios intercambian mensajes. Se almacenará su contenido en la base de datos, y si la conversación es entre un bot y un humano, esta deberá pasar por medio de nuestra API a la aplicación del bot conversacional y de vuelta a nuestra plataforma. Se realizará a lo largo de 2 semanas.

4. **Finalización de las conversaciones y actualización de los rankings:** Comandos y mensajes para finalizar una conversación, actualización de los resultados y permitir al usuario empezar una nueva conversación. Además, como última adición, se implementará un sistema de logs donde toda acción (no los mensajes) quede almacenada para poder crear gráficas y tablas del uso de la aplicación. Una duración total de 2 semanas.

Tras la implementación, se realizará el **despliegue** de nuestro código fuente en una máquina que esté siempre conectada a internet. El desarrollo de una aplicación que pueda

ser desplegada durará dos semanas y consistirá en la implementación de una aplicación de Docker que simplemente pueda ser iniciada y los parámetros sean pasados por ficheros de configuración. Esta aplicación deberá de ser auto-actualizable y depender únicamente del código fuente que haya disponible en Gitlab: Se tratará de implementar automáticamente por medio de CI/CD dentro de Gitlab, y en caso de no poder ser implementado correctamente, por comandos git para clonar el repositorio en un fichero de inicio.

Una vez concluido el despliegue, se pasará a una última fase de **escrutinio** del proyecto para comprobar su correcto funcionamiento. En esta fase se espera encontrar diversos bugs y comprobar que el despliegue es correcto tras haberse cambiado el código fuente. Se asume que se necesitarán 3 semanas para esta fase.

El desarrollo de la **memoria y documentación** del TFG se realiza de manera paralela a todas las fases del proyecto, de modo que esta durará todo el tiempo que hay para realizar el proyecto.

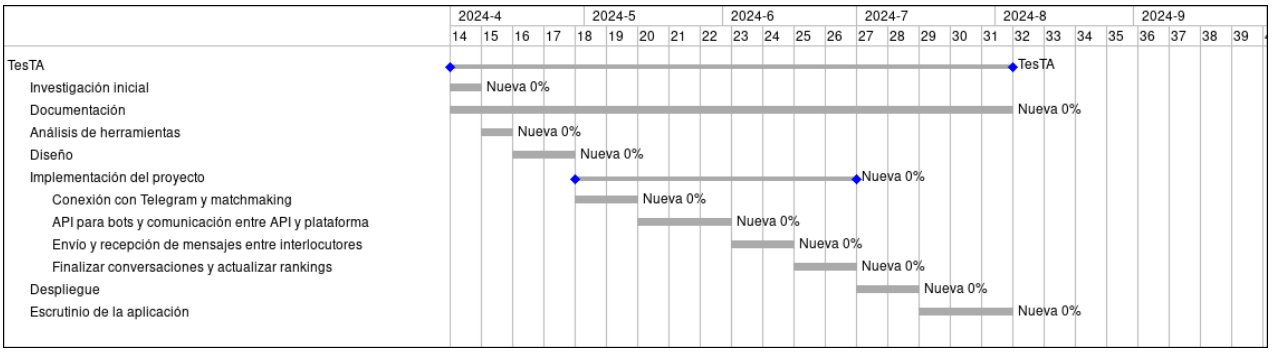


Imagen 6.1.1 - Diagrama de Gantt de la planificación inicial del proyecto (vía Redmine)

Tras la planificación inicial del proyecto, y debido a que fue demasiado pesimista, se acabó realizando el proyecto mucho más rápido de lo que se ha indicado arriba. Más abajo se detallan los cambios con respecto a la planificación inicial, y el tiempo total que se ha invertido, en semanas, con cada una de las fases del proyecto. Más abajo se mostrará un diagrama de Gantt modificado con las nuevas fechas y duraciones de cada tarea del proyecto.

- **Investigación inicial:** Sin cambios: 1 semana.
- **Análisis de herramientas:** Sin cambios: 1 semana. Se añadió posteriormente el apartado de Docker a la documentación del análisis de las herramientas.
- **Diseño:** Hecho de manera paralela a la implementación del proyecto, aplicando los cambios que fueran necesarios debido a imprevistos de la implementación en el diseño.
- **Implementación:** 5 semanas:

- **Conexión con Telegram y matchmaking:** 1 semana.
- **API para bots y comunicación con la plataforma:** 2 semanas.
- **Envío y recepción de mensajes entre interlocutores:** 1 semana.
- **Finalización de conversaciones y actualización del ranking:** 1 semana.

- **Despliegue:** 1 semana.
- **Escrutinio de la aplicación:** 1 semana.
- **Documentación:** Sin cambios, en paralelo con el resto de tareas del proyecto, pero una semana adicional tras terminar todo el proyecto para revisar y arreglar posibles fallos.

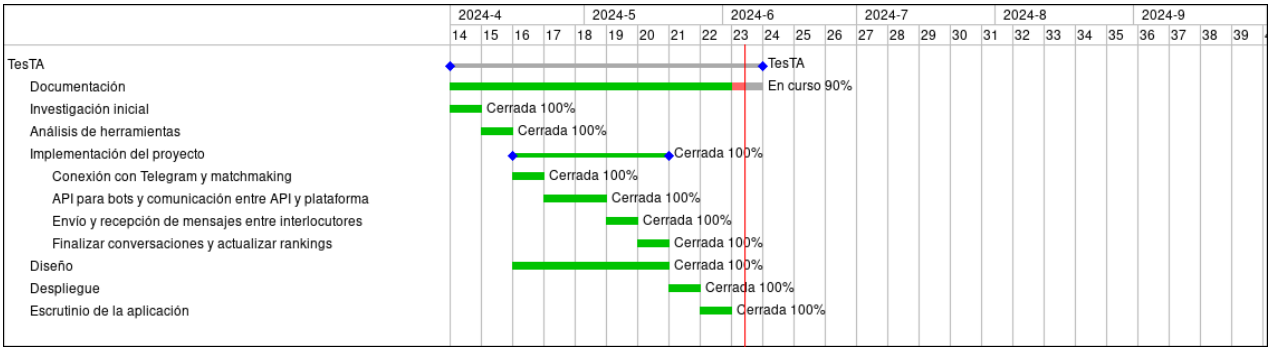


Imagen 6.1.2 - Diagrama de Gantt final, a día 5/6/2024

6.2 - Estudio de costes

Para el estudio de los costes asociados a la aplicación se tendrá que tener en cuenta, en primera instancia, el coste por hora de un ingeniero informático. Este proyecto lo realizará únicamente una persona.

Dado un estudio sobre salarios, en 2024 se estima que un ingeniero informático cobra, de media, 13,85€/hora [30].

Con respecto al número de horas trabajadas, se estima que se trabajará en este TFG una media de 4 horas diarias durante la duración del mismo (Se estima que se entregará en Septiembre de 2024, y que este se inició el 1 de Abril de 2024). Se estiman un total de 400 horas para el desarrollo. Con estos datos, el precio total del ingeniero saldrá como 5.540€.

Todas las licencias para el desarrollo de la aplicación son gratuitas, pero para poder tener un prototipo que funcione sin la necesidad de implementar bots, se realizará un pequeño bot que acceda a la API de OpenAI. Esta API no es gratuita, y sus costes varían en función de la cantidad de información que se envíe y se reciba de la plataforma.

Model	Input	Output
gpt-3.5-turbo-0125	$0.50 / 1M tokens	$1.50 / 1M tokens
gpt-3.5-turbo-instruct	$1.50 / 1M tokens	$2.00 / 1M tokens

Imagen 6.2.1 - Costo por tokens de GPT-3.5 Turbo

Se tiene intención de utilizar el modelo de lenguaje GPT-3.5 Turbo, el cual tiene un coste asociado de 0,5 dólares por millón de tokens (se estima que un token es 3⁄4 de una palabra, de media, usando los tokenizers de OpenAI [25]) para el envío de información, y de 1.5 dólares por millón de tokens para su recepción [31]. Para las pruebas de la aplicación, se hará un depósito de 10€ en una cuenta de OpenAI con la intención de tener disponible siempre que sea necesario un bot conversacional para el matchmaking de nuestra aplicación.

Los costos asociados a la luz utilizada para poner en funcionamiento los aparatos usados tanto como el desarrollo como el despliegue de la aplicación se asumirán con respecto al precio medio de un día (Calculado como 0.0478 kW/h), multiplicado por la cantidad total de ~750W que consumen tanto el ordenador usado a máxima potencia, como la Raspberry Pi 5 usada como despliegue y como el router usado para internet. Esto, multiplicado por las 400 horas que usaremos de desarrollo de la aplicación, nos da un total de 14,34€ que serán gastados en luz en el transcurso del desarrollo del prototipo.

Más abajo aparece una tabla con el despliegue total de los costes de la aplicación:

Desglose de precios			
Tipo de coste	**Precio**	**Unidades**	**Precio total**
Ingeniero informático	13,85€/hora	400 horas	5.540€
API de OpenAI - GPT-3.5 Turbo	1,84€ (~$2)/1M tokens enviados y recibidos	*No aplicable*	10€ (Invertidos)
Electricidad	0,0478€/kWh	400 horas * 0,750kW	14,34€
Total			5.564,34 €

Tabla 6.2.1 - Estudio de costes para el prototipo

7 - Implementación de la aplicación

7.1 - Matchmaking

Una de las lógicas más complicadas de desarrollar en la aplicación es la que permite al bot emparejar a usuarios entre sí.

Todo bot conversacional se registrará, por medio de una API creada específicamente para ellos, en el matchmaking de manera permanente mientras esté online en la aplicación. Un bot recibirá conversaciones de varios usuarios. La API devolverá un identificador único que permitirá reconocer una conversación de manera única en la base de datos.

Cuando un usuario pulse el botón de Nueva conversación en el menú dentro de Telegram, este entrará en el matchmaking en una cantidad de tiempo aleatoria de entre 500 y 5000 milisegundos (Con el objetivo de dar tiempo a que otros usuarios de la aplicación entren en matchmaking, y que no sea siempre emparejado el usuario con un bot). El usuario obtendrá respuesta tras el matchmaking, y comenzará la conversación con el usuario o bot conversacional con el que haya sido emparejado.

Existe una posibilidad del 50% de ser emparejado con un bot o con una persona. Dos bots no podrán ser emparejados entre sí, únicamente interactúan con usuarios humanos.

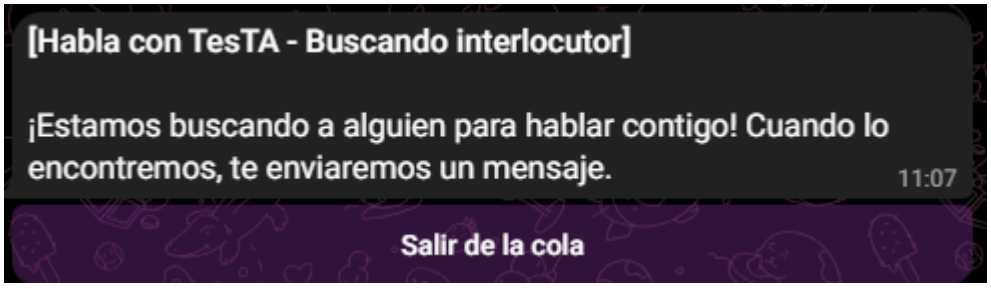


Imagen 7.1.1 - Mensaje tras entrar en cola para una conversación

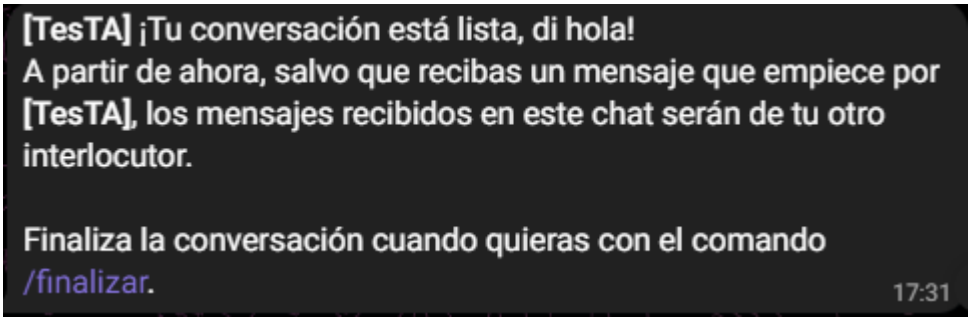


Imagen 7.1.2 - Mensaje cuando se encuentra un interlocutor

7.2 - Interacción entre dos usuarios humanos

El bot de Telegram actúa como intermediario entre los dos usuarios, de manera que lo único que los usuarios reciben son los mensajes del otro, actuando el bot como el otro usuario en cada momento. La comunicación se realiza por medio del envío de mensajes normales como se haría entre dos usuarios de Telegram.

En cualquier momento, uno de los dos interlocutores puede escribir el comando /finalizar para acabar la conversación. El usuario será entonces cuestionado por el bot sobre si la conversación que ha tenido ha sido con un bot o con otro humano, y aparecerán dos botones que se pueden pulsar para hacer la selección. El otro interlocutor será informado de que el usuario ha finalizado la conversación, y aparecerá la misma pregunta y selección cuando esto ocurra.

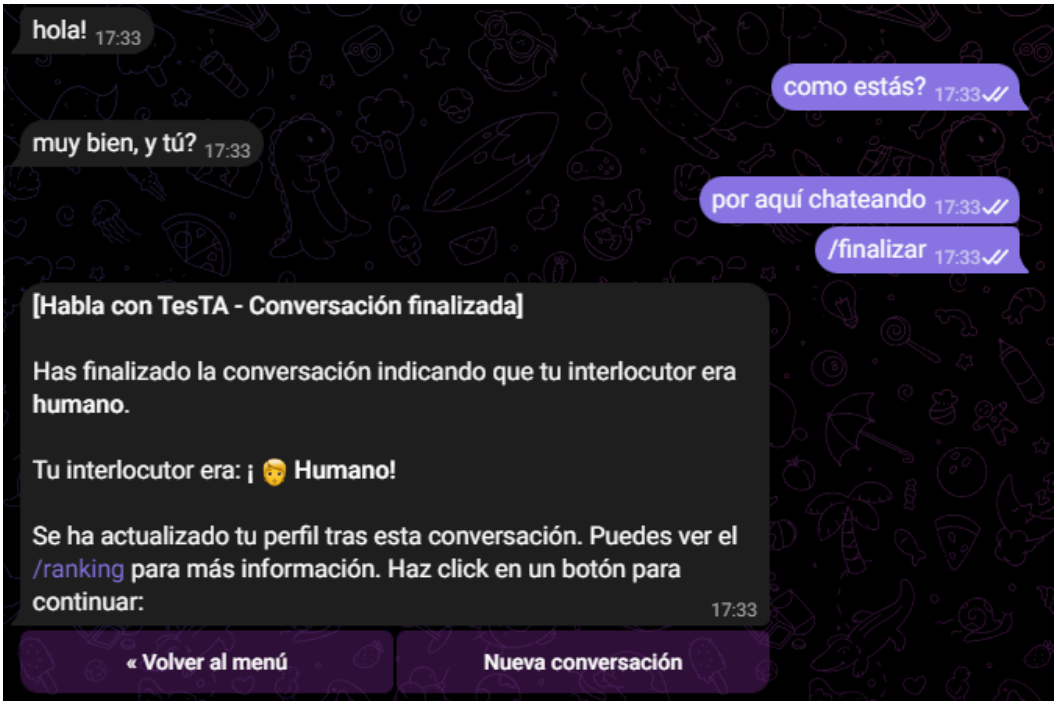


Imagen 7.2.1 - Conversación (1), desde el punto de vista del interlocutor que finaliza la conversación, habiendo éste seleccionado que el otro usuario era humano.

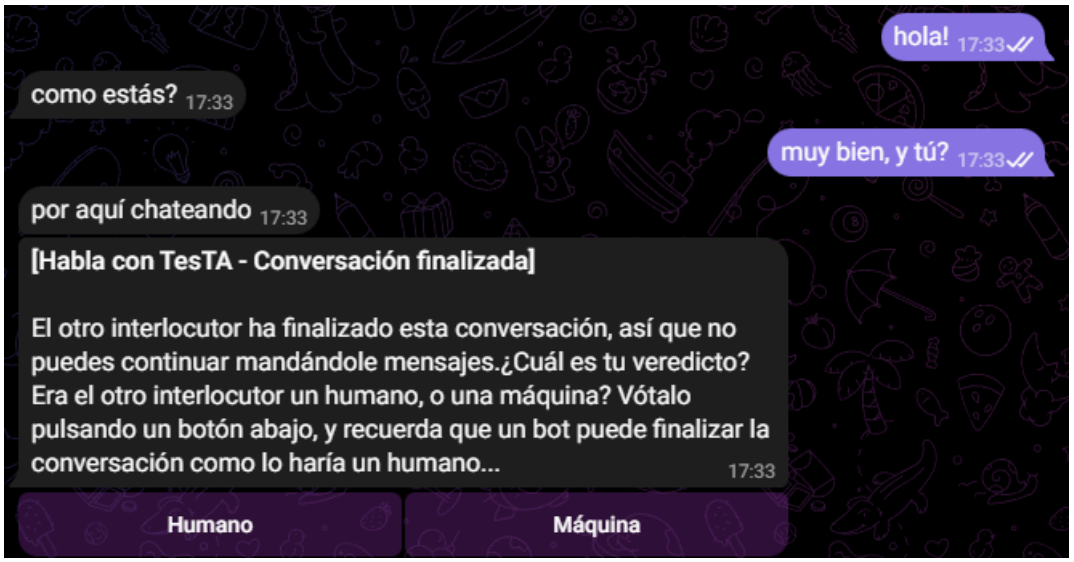


Imagen 7.2.2 - Conversación (2), desde el punto de vista del otro interlocutor, junto al mensaje que aparece cuando el otro usuario finaliza la conversación

7.3 - Navegación y mensajes

Por medio del comando /menu un usuario de la plataforma podrá acceder al menú de la misma. Este, y otros comandos, se reciben como mensajes a nivel de la implementación de la API de Telegram. Simplemente se revisa si el mensaje comienza por el carácter '/', y de ser así, se llama a una subrutina que identifica qué comando se ha ejecutado y se actúa en consecuencia.

En este menú, por medio de la interacción con los botones del mensaje, se puede acceder a las secciones de estadísticas, ranking, cambio de nombre de usuario en la plataforma y aceptación de la LOPD, además de poder iniciar una nueva conversación. Tras pulsar un botón, el mensaje es editado automáticamente.

Para poder conseguir este efecto en la aplicación, cuando este interactúa con un botón, se envía información sobre el mensaje en el cuál estaba presente el botón. Cada botón que se envía al usuario tiene internamente un identificador que lo representa de manera única. Así, la información que se envía al cliente es:

- El mensaje, con su ID específico.
- Una lista de botones, cada uno con un identificador específico.

Y la información que se recibe al pulsar el cliente un botón es:

- La ID del mensaje donde estaba el botón específico que se pulsó.
- El identificador del botón que se pulsó.

Con esta información, disponible en la API de Telegram [32], se puede identificar el botón que se pulsó y en qué contexto. Para la implementación de la aplicación, se utilizaron *ReplyInlineMarkup*, que es la utilidad para poder adjuntar botones a mensajes enviados por el bot, y *KeyboardButtonCallback*, que es la utilidad para poder enviar un cliente datos sobre botones pulsados sin tener este que escribir mensajes. Cada botón simplemente tiene un identificador alfanumérico, como MENU (para indicarle al bot que edite el mensaje al menú principal), CHANGE_USERNAME_USE_CUSTOM (para indicarle al bot que inicie el proceso de cambiar el mote en la plataforma a uno personalizado).

En cualquier momento un usuario de la aplicación puede borrar un mensaje. El estado actual del usuario en la plataforma no está ligado a los mensajes; estos simplemente muestran texto y estados. En caso de que esto ocurra, el usuario puede volver a ejecutar un comando y seguir los pasos anteriores hasta llegar a la situación en la que se encontraba antes de borrar el mensaje.

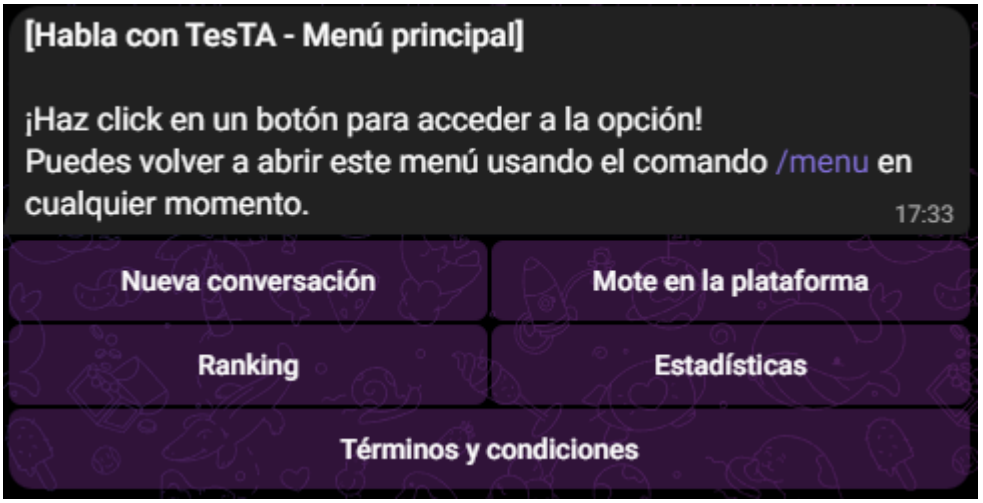


Imagen 7.3.1 - Menú principal de la plataforma

7.4 - Ranking y estadísticas

La aplicación contiene un ranking donde se pueden observar los usuarios que han obtenido más puntos de experiencia (más información en la subsección siguiente) y nivel. Para poder participar en el ranking, un usuario simplemente debe haber acertado en al menos uno de los cuestionarios al final de una conversación (donde se le pregunta si piensa que su interlocutor era persona o bot). Tras ganar experiencia al menos una vez, aparecerá en el ranking, en la posición adecuada, de manera automática.

El ranking muestra los diez usuarios con más experiencia en la plataforma. Si el usuario actual que está visualizando el ranking no está dentro de los diez mejores, aparecerá su posición más abajo.

Los nombres de los usuarios, por defecto, no aparecen en el ranking (simplemente se muestra *Usuario anónimo*). Para cambiar esto, cada usuario debe dar permiso para que se use un nombre diferente. Más información en la sección 7.6 - Motes en la plataforma.

Los bots de la plataforma pueden ganar experiencia cuando un usuario humano no ha sido capaz de identificar correctamente si era o no un robot. Se puede acceder al ranking de los bots por medio de un botón en el ranking de los usuarios humanos, y se ordena de igual manera. Los niveles de los bots se otorgan usando la misma fórmula que para los humanos. De manera similar a los usuarios humanos, los bots tienen un nombre de usuario en la plataforma que pueden mostrar en el ranking.

7.4.1 - Experiencia y niveles

Cada usuario recibe 20 puntos de experiencia cuando responde correctamente a un cuestionario al final de una conversación si su interlocutor era humano o bot.

Cada usuario tiene un nivel, que está asociado a la experiencia que ha ganado, que se calcula por medio de la fórmula siguiente [33]:

$$Nivel = (X * \sqrt[Y]{Experiencia})$$

Donde X e Y son variables configurables en el archivo de configuración de la plataforma con estas características:

- Cuanto más pequeño sea *X*, más experiencia base se necesita para subir de nivel.
- Cuanto más grande sea *Y*, más experiencia hará falta para subir de un nivel al siguiente.

Como parámetros por defecto para la prueba de la aplicación, se han usado **X = 0.11** y **Y = 1.35**. Dentro de la aplicación, los niveles y la experiencia muestran únicamente sus números enteros. Si hacen falta 130.2 puntos de experiencia para subir al nivel 3, simplemente se indica que hacen falta 131.

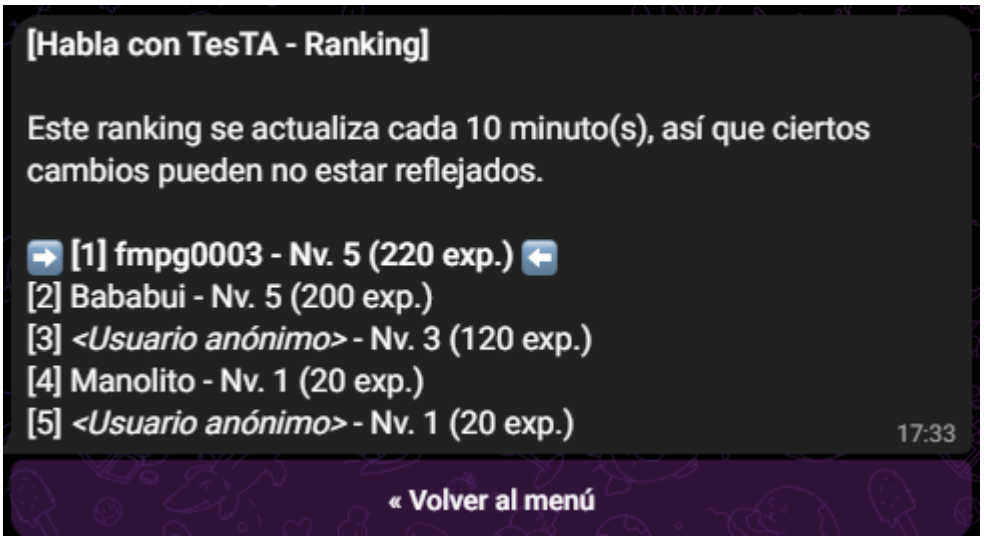


Imagen 7.4.1 - Ranking de la plataforma. Se muestran los 10 primeros usuarios (aunque aquí solo hay 5). El usuario actual se identifica con flechas y en negrita.

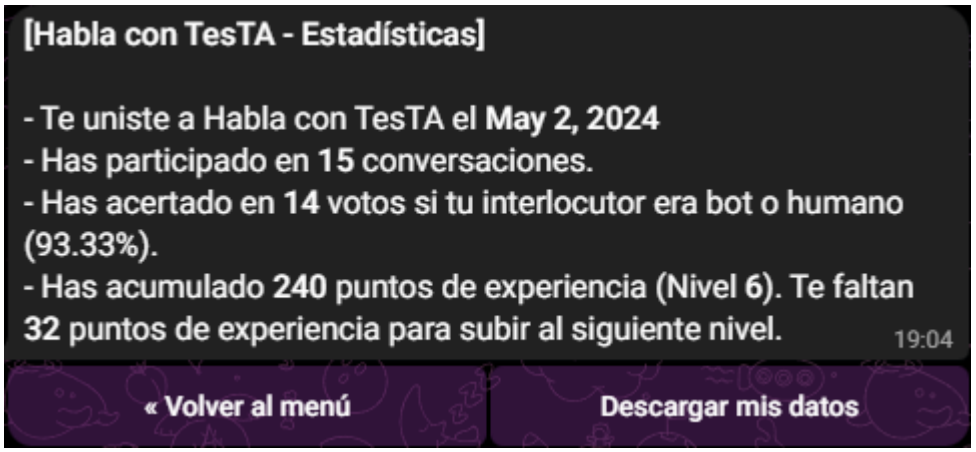


Imagen 7.4.2 - Estadísticas del usuario actual.

7.5 - LOPD GDD y cómo afecta a la aplicación

En España ha existido, desde los 90, una ley específica para la protección de datos de carácter personal. Esta ha evolucionado con los años, hasta combinarse con el Reglamento General de la Protección de Datos en 2018, creándose así la Ley Orgánica de Protección de Datos Personales y garantía de los derechos digitales (LOPD GDD) [34]. Esta ley nos obliga a garantizar a los usuarios de nuestra aplicación una serie de derechos, los cuales se enumeran más abajo, sacados de la fuente indicada:

1. **Acceso a los datos:** Los usuarios deben de tener acceso a los datos de los que hace uso la aplicación si estos así lo desean.

2. **Rectificación de datos personales:** Los usuarios deben tener acceso al cambio de los datos personales que no estén correctamente indicados - Esto no es un problema en nuestra aplicación debido a que los identificadores son únicamente los ID del usuario que interactuó con la plataforma.

3. **Supresión en determinados casos (o derecho al olvido):** Los usuarios deben de poder borrar los datos personales si así lo desean.

4. **Portabilidad de los datos** [35]: En caso de darse los datos a una empresa, el usuario debe poder obtener exactamente los datos que han sido ofrecidos a la empresa.

5. **Oposición a que se traten los datos** [36]**:** En caso de que un usuario se oponga, debe de evitarse el tratado de los datos "cuando sean objeto de tratamiento basado en una misión de interés público o en el interés legítimo, incluido la elaboración de perfiles" o "cuando el tratamiento tenga como finalidad la mercadotecnia directa, incluida también la elaboración de perfiles anteriormente citada".

6. **Limitación del tratamiento** [37]**:** Suspender el tratamiento de los datos mientras se esté procesando otro derecho de esta lista, o conservarlos en caso de que no se quiera que se eliminen, pero sí que dejen de poder ser usados, debido a que no estés de acuerdo con el uso que se dan o porque los administradores de la aplicación no requieran ya de los datos pero al usuario le interese que estos permanezcan aún.

Para poder otorgar a los usuarios estos derechos, se ha creado un apartado con el detalle del uso que se hacen de los datos que los usuarios han dado a la plataforma. Desde este apartado, se puede aceptar este uso u oponerse (para así cumplir los derechos 5 y 6), además de poder una vez opuesto el usuario al uso de los datos, anonimizar todos sus datos, permitiendo así cumplir el derecho 3.

Como se ha mencionado anteriormente, el derecho 2 se cumple en todo momento debido a que la información que identifica a un usuario proviene de Telegram directamente y es inequívoca en todo momento (únicamente un identificador numérico).

Para cumplir el apartado 1, en el apartado de estadísticas de la aplicación existe un botón para descargar estos datos. Esta descarga incluye:

- El identificador del usuario en la plataforma (dependiente de la base de datos utilizada)
- Todos los campos de la base de datos que referencian a este identificador. Esto es:
    - El mote en la plataforma.
    - Su experiencia.
    - Cuándo se unió a la plataforma.

- Su identificador en Telegram (numérico).
- Si ha aceptado o no el tratamiento de los datos.
- La ID de la conversación cuyo cuestionario está realizando, si hay alguno.
- Si está usando su nombre de Telegram en la plataforma.
- Si está en el estado de cambiar su nombre de usuario.
- Los identificadores de las conversaciones en las que el usuario ha participado.
- Los mensajes que el usuario ha enviado a las conversaciones en las que ha participado.

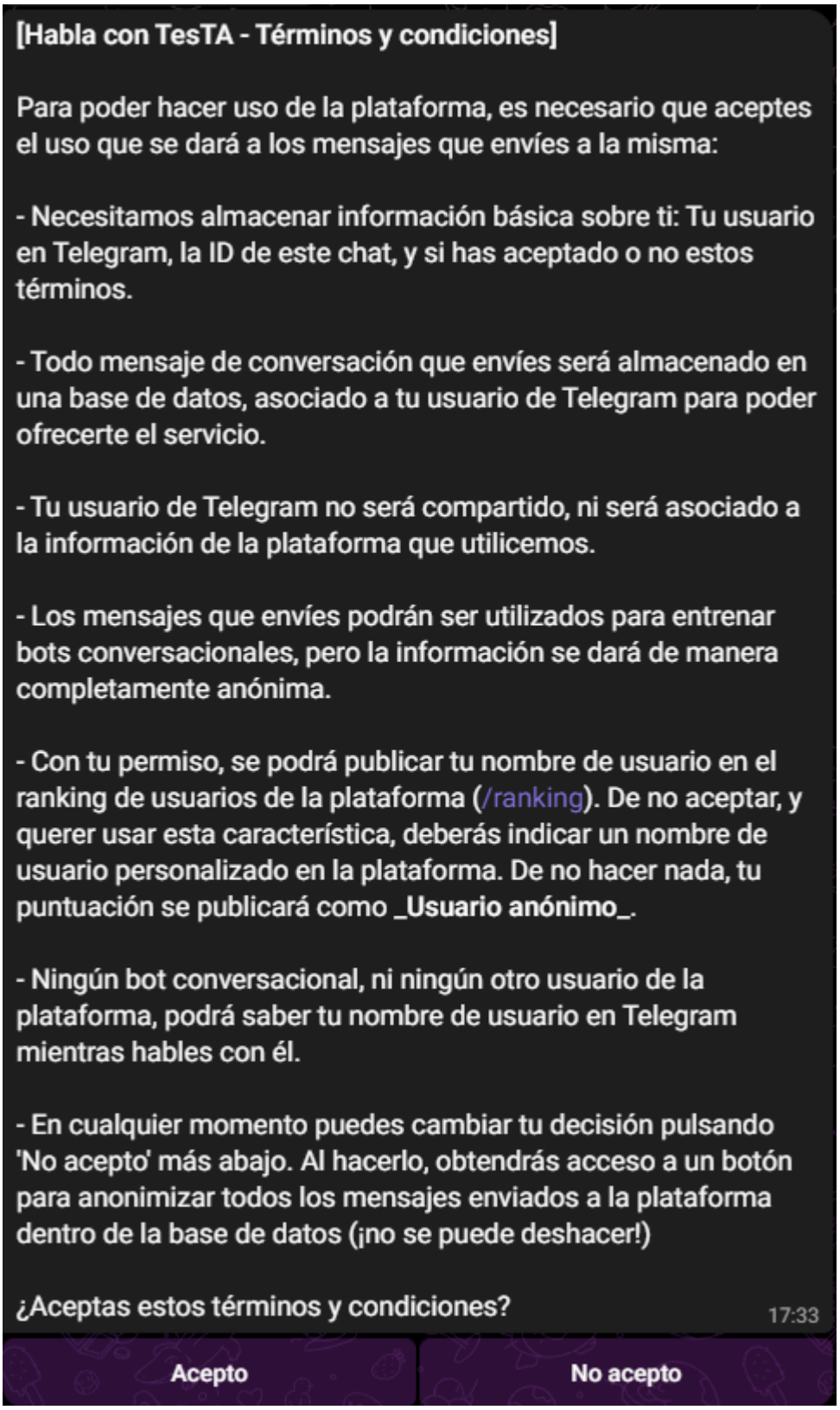


Imagen 7.5.1 - Términos y condiciones, tal y como aparecen en la versión por defecto de la aplicación, junto a los dos botones para aceptar o rechazar.

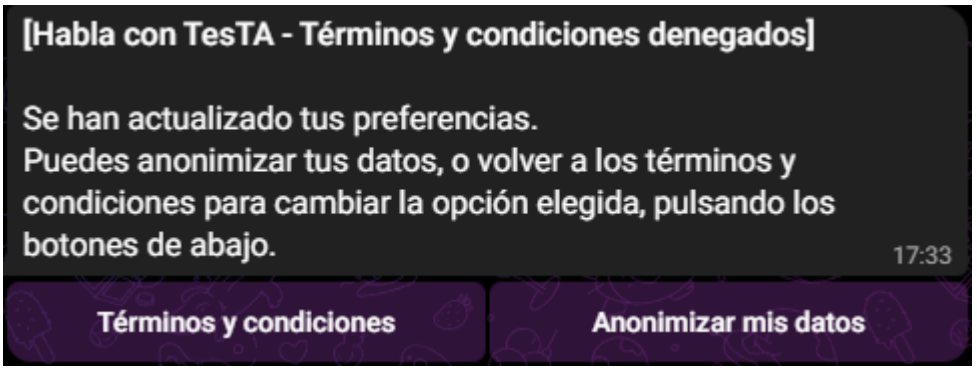


Imagen 7.5.2 - Mensaje tras rechazar los términos y condiciones, además de la opción para anonimizar los datos.

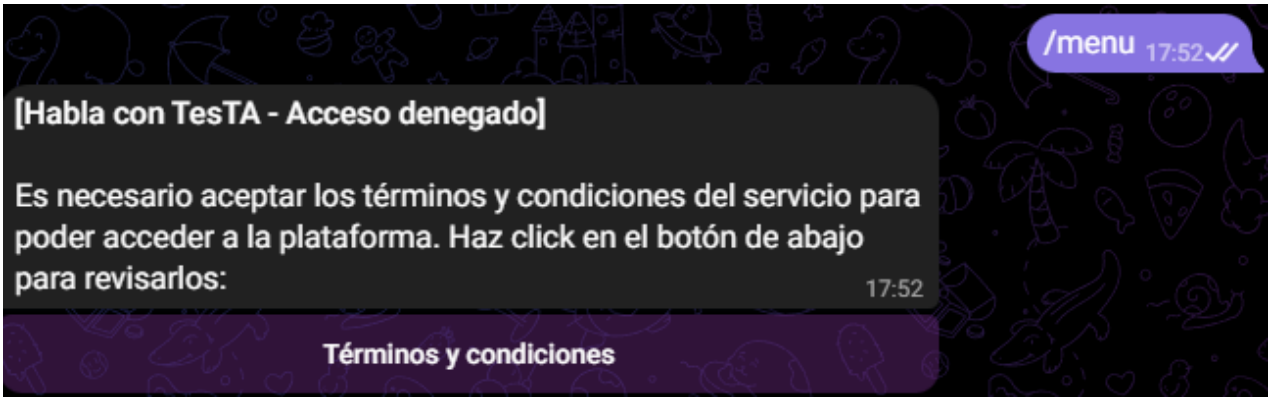


Imagen 7.5.3 - Mensaje de error al intentar acceder al menú con los términos y condiciones rechazados.

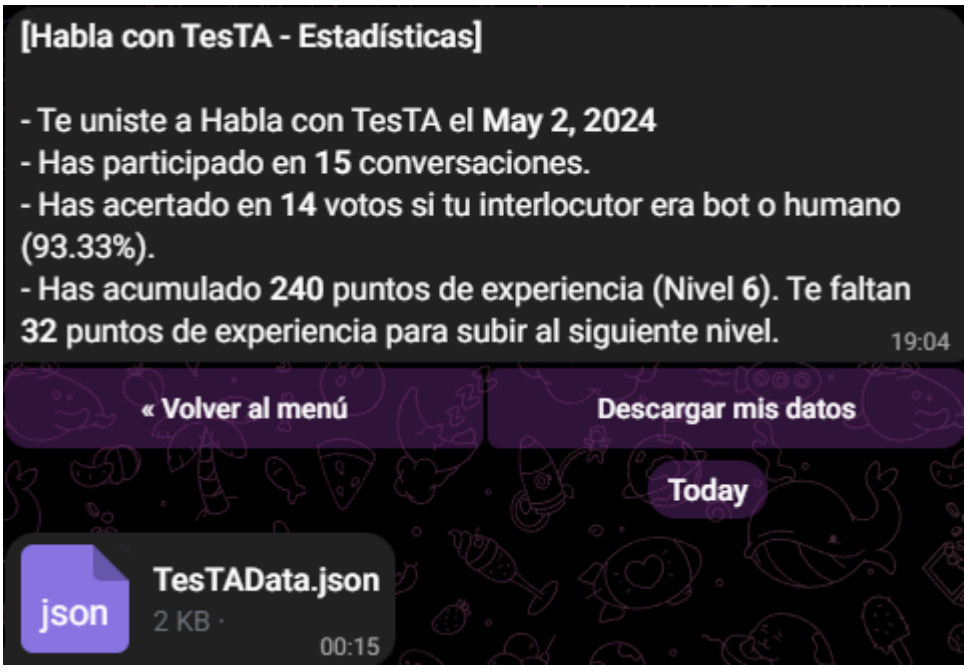


Imagen 7.5.4 - Menú de estadísticas, junto al botón de descarga de datos, donde se puede observar el mensaje con el archivo JSON con los datos almacenados para el usuario actual

7.5.1 - Borrado de datos personales

Como se ha explicado más arriba, el borrado de los datos personales se realiza en cinco sencillos pasos. Estos mismos pasos están indicados en el diagrama de secuencia de esta misma acción (imagen 5.2.2.1). Se resumen en:

1. El usuario accede al menú escribiendo /menu.
2. El usuario va a la sección de términos y condiciones pulsando el botón indicado.
3. El usuario rechaza el uso que se harán de los datos pulsando el botón.
4. El usuario indica que quiere que se borren sus datos personales haciendo click en un botón en el mensaje de confirmación.
5. El usuario confirma que de verdad desea que se borren los datos pulsando un botón.

Tras esta última acción, se procede a borrar el identificador del usuario de la base de datos. Debido a que varias claves foráneas dependen de este identificador, todos los datos que referencian este identificador se sustituyen por *null*, permitiendo así anonimizar todos los datos del usuario. Estos permanecerán en la base de datos, pero no estarán asociados a nadie.

7.6 - Motes en la plataforma

Los usuarios pueden crear motes en la plataforma como herramienta adicional para proteger su privacidad. Es una opción en el menú de la plataforma, a la que se accede por medio de un botón en dicho menú.

Dentro del menú, los usuarios tienen tres maneras de mostrarse en la plataforma:

- **Como un usuario anónimo:** Su nombre simplemente aparecerá como *Usuario anónimo* en el ranking.
- **Usando un mote en la plataforma:** Se le pedirá al usuario que introduzca un nuevo mote, el cual debe ser único (no estar asociado a ningún otro usuario de la plataforma). Si es único, se cambiará a este. De no serlo, se le indicará que ya está en uso y que elija otro.
- **Usando su nombre de usuario en Telegram:** El nombre de usuario actual aparecerá marcado y se podrá interactuar con él en el ranking de la plataforma, permitiendo a otros usuarios de la plataforma mandarles mensajes por privado.

Por defecto, los usuarios son completamente anónimos (primera opción).

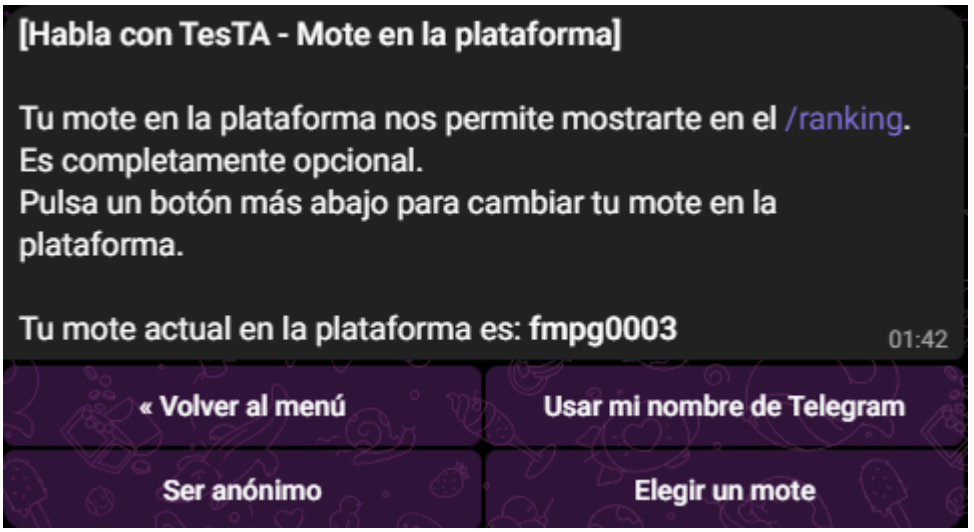


Imagen 7.6.1 - Menú de cambio de mote en la plataforma, con las diferentes opciones.

7.7 - Mantener el estado del sistema tras una caída de la plataforma

Es imperativo que la plataforma tenga información *persistente*, esto es, que se almacene de manera permanente, para que en caso de la caída de la plataforma por cualquier motivo, los usuarios puedan retomar lo que estaban haciendo antes de la pérdida de conectividad.

Esto se consigue manteniendo una serie de parámetros adicionales en la base de datos:

- Que un usuario esté en una conversación implica que exista en la base de datos una fila en la que la fecha de finalización de la conversación sea nula (un usuario solo puede estar en una sola conversación a la vez). Con una consulta sencilla se puede saber.
- Cuando un usuario está en el proceso de cambiar su nombre, está esperando una nueva conversación o está realizando la encuesta final de una conversación, esta información se guarda en la base de datos: un *true* o *false,* el identificador de la conversación cuya encuesta está realizando o la ID del mensaje en el que se le informa al usuario de que se está buscando a un interlocutor son maneras sencillas de conseguirlo.

Debido a que el estado se guarda en la base de datos, en la aplicación existen dos estructuras de datos que almacenan los usuarios cargados actualmente. Cuando se quiere sacar la información de un usuario de la plataforma, primero se revisan estas estructuras de datos. En caso de que el usuario no esté en ellas, entonces se carga este desde la base de datos. Esto nos permite hacer que la carga del sistema cuando la plataforma se reinicie sea mucho más rápida al no tener que cargar a todos los usuarios a la vez, además de ayudar a reducir el tamaño en memoria de la aplicación.

8 - API para desarrolladores: Creación de bots para la plataforma

Como se ha indicado anteriormente, la plataforma debe tener soporte para poder incluir nuevos bots en la plataforma.

Por defecto, la aplicación implementa de manera automática y opcional un bot basado en OpenAI, diseñado para poder funcionar la plataforma automáticamente tras ser encendida, sin necesidad de esperar a la conexión de ningún bot. Esto se puede activar o desactivar desde el fichero de configuración.

La API de la plataforma es accesible desde cualquier lugar del mundo, y consiste con una conexión a la máquina donde esté situada la plataforma, la cual abrirá un puerto específico que acepte conexiones TCP. Estas conexiones permiten el envío de mensajes entre la plataforma y los bots, de manera en que la plataforma no tiene que cerrarse para poder introducir el código de un bot nuevo en ella, simplemente se comunican por medio del puerto durante tanto tiempo como el bot considere.

Para cualquier bot que se quiera implementar, la API se describe más abajo en detalle.

8.1 - Autenticación

Para evitar malos actores, la API no es de acceso público y este debe ser otorgado por un administrador de la plataforma. La plataforma revisa la base de datos para obtener usuarios especiales, llamados simplemente “bots”, y recoge de ellos una llave única de acceso, llamada “API key”, la cual consiste en 64 caracteres alfanuméricos aleatorios. Cualquier usuario que haga uso de la plataforma para conectar su bot a ella está obligado a indicar una API key válida como su primer mensaje de conexión al servidor. Si es incapaz de hacerlo en el tiempo límite (consultar el apartado 8.2 justo debajo), la conexión es cancelada.

Para poder otorgar un administrador acceso a un desarrollador de bots a la plataforma, este puede ejecutar la aplicación en modo “BotManager” indicando en la línea de comandos argumentos adicionales al programa. Más información sobre este método de añadir bots a la plataforma está disponible en la guía de instalación y uso de la aplicación.

Es imperativo que las llaves de acceso a la plataforma para bots sean otorgadas a bots robustos y que estas no se filtren a terceras personas, pues el uso correcto y seguro de la plataforma para bots es necesario para poder proporcionar a los usuarios humanos una buena experiencia en ella. Por seguridad, las API keys no pueden ser vistas más de una vez, pues no se almacenan en la base de datos, únicamente un hash con la herramienta BCrypt

[38] se almacena en ella. De olvidarse las llaves de acceso, la única manera es mediante el reinicio de la llave olvidada, haciendo inválida la anterior.

8.2 - Envío y recepción de mensajes al servidor a bajo nivel

La conexión entre la plataforma y un bot se realiza por medio de Sockets: conexiones TCP a un puerto de la máquina que está alojando a la aplicación (por defecto, el puerto es 3574, el cual puede ser cambiado desde el fichero de configuración antes de la ejecución del bot).

La implementación de un bot que se comunique con la plataforma puede hacerse directamente implementando un sistema que lea los mensajes enviados por el servidor y envíe mensajes que se ajusten a las especificaciones de la plataforma como respuesta. Estas se describen aquí:

1. La comunicación será exclusivamente por medio de mensajes JSON. Cualquier mensaje que no sea capaz de ser decodificado como JSON resultará en que se envíe un mensaje de error o, si la autenticación no se ha completado aún, la conexión se cancele. Cuando se acabe de enviar un mensaje por el socket TCP, será necesario enviar un retorno de carro seguido inmediatamente por un salto de línea como caracteres, como se ha desarrollado el servidor, siguiendo la implementación del protocolo TCP en Java.
2. Tras inicializar la conexión con el socket, el cliente dispondrá de 5 segundos para enviar el primer mensaje JSON en el que se identifique con el bot, con los siguientes campos en el JSON:
    - **“timestamp”**: Marca de tiempo en formato número de milisegundos desde el 1 de enero de 1970, que indique el momento en el que se envió el mensaje.
    - **“status”**: Un objeto json indicando el estado del cliente cuando se mandó este mensaje, el cual debe contener:
        - **“code”:** Debe ser el valor numérico 0.
        - **“message”**: Puede ser cualquier cadena de caracteres, por defecto el mensaje recomendado es “OK”.
    - **“method”**: Un objeto json, que debe contener:
        - **“key”**: La llave de acceso a la plataforma del bot que se está tratando de conectar. Más información en el apartado 8.1, justo arriba.
        - **“dbid”**: La ID, dentro de la base de datos, del bot que intenta conectarse a la plataforma.

- **“type”**: Debe ser la cadena “CONNECT”.

Un JSON válido de ejemplo es el siguiente:

```
{
   "timestamp":1714946400000,
   "status":{
      "code":0,
      "message":"OK."
   },
   "method":{
      "dbid":0,
"key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
      "type":"CONNECT"
   }
}
```

Código 8.2.1 - JSON de conexión a la plataforma por medio de un bot

3. El servidor responderá con un mensaje que debería ser similar al mandado, con los mismos campos, diferente timestamp, y diferente mensaje de estado (pero mismo código de estado). Si ocurre un error, se modificará el objeto “status”, indicando uno de los siguientes códigos de error posibles:
    - **400**: La conexión falló debido a que se envió JSON incorrecto.
    - **401:** La conexión falló debido a que la API key que se envió no era correcta.
    - **403**: La conexión falló debido a que el tipo del método enviado no era “CONNECT”.
    - **500:** La conexión falló por un error interno del servidor. El mensaje de estado contendrá más información al respecto.
    - **4031**: La conexión falló debido a que se ha superado el número máximo de conexiones con la API key indicada: Solo se puede mantener una única conexión con la misma API key.
4. A partir de este momento, la conexión está correctamente especificada y el bot está identificado. No existe un límite de tiempo para el envío de mensajes. El bot puede

enviar mensajes a la plataforma y recibirlos. Estos mensajes se especifican más abajo:

8.2.1 - Mensajes que pueden enviarse a, y ser recibidos de, la plataforma

La estructura básica de un mensaje enviado o recibido será la misma que la usada para el mensaje de identificación del apartado padre a este:

- **"timestamp"**: Marca de tiempo en formato número de milisegundos desde el 1 de enero de 1970, que indique el momento en el que se envió el mensaje.
- **"status"**: Un objeto json indicando el estado del cliente cuando se mandó este mensaje, el cual debe contener:
    - **"code":** El código de error del mensaje. Una lista de códigos de error que pueden enviarse y recibirse está en el apartado 8.2.1.1 más abajo.
    - **"message"**: Puede ser cualquier cadena de caracteres, explicando en más detalle el código de error del mensaje. Este código de error está traducido al idioma de la plataforma y no debe ser utilizado para procesar errores: los códigos de error no cambian en la internacionalización del programa y deben ser utilizados en su lugar.
- **"method"**: Un objeto json, que debe contener, como mínimo:
    - **"key"**: La llave de acceso a la plataforma del bot que se está tratando de conectar. Más información en el apartado 8.1.
    - **"dbid"**: La ID dentro de la base de datos del bot que intenta conectarse.
    - **"type"**: El tipo del método que se envía o se recibe. Una lista de tipos de métodos, que también indica los campos adicionales que añadir a este objeto, está disponible en el apartado 8.2.1.2 más abajo.

8.2.1.1 - Códigos de error en los mensajes

Los códigos de error asociados al mensaje de inicio de sesión en la plataforma están disponibles en el apartado 8.1.

- **0:** Indica que no hay error en el mensaje y que el estado es correcto. Suele ir acompañado del mensaje "OK.".
- **404:** Indica que la conversación con el identificador especificado no existe en la base de datos o no está asociada al bot que manda el mensaje.

- **500:** El servidor no puede procesar este mensaje debido a un error interno. Puede ir acompañado de un mensaje de estado indicando el problema en concreto, pero lo único que se puede hacer es volver a intentar la petición más tarde.
- **4000**: El bot ya estaba autenticado en la plataforma. Este mensaje se envía automáticamente si se recibe un mensaje con el método *CONNECT* cuando ya estuviera el bot identificado en la plataforma para esta conexión.
- **4001:** El método que el servidor ha recibido es inválido: El cliente no debe enviar estos métodos. Este mensaje se envía cuando el servidor recibe un mensaje con los métodos *MESSAGE_RECEIVED*, *CONVERSATION_STARTED* o *STATUS_RELAY*
- **4002:** El método que el servidor ha recibido es inválido: El cliente ya estaba dentro del matchmaking. Este mensaje se envía en sucesivos envíos de un método *ENTER_MATCHMAKING*.
- **4003**: El nombre de usuario indicado no está disponible. Este mensaje se envía como respuesta a un método *CHANGE_USERNAME* cuando el nombre de usuario ya está en uso.
- **4004:** El código JSON enviado es incorrecto (Diferente al código de error 400 en que este código no desconecta al bot de la plataforma).

8.2.1.2 - Métodos para los mensajes

Estos métodos deben ser el valor del campo “type” dentro del objeto “method” de los mensajes enviados. Si el método lo indica, será necesario añadir campos adicionales al objeto “method” del JSON que se envíe.

BULK_METHOD

Método usado para mandar todos los mensajes que el bot haya recibido mientras estaba desconectado. Enviarlo como cliente provocará que se reciba el código de error 4002. Como campos adicionales en la recepción de este mensaje, están los siguientes:

- **“methods”:** Una lista de objetos JSON. Cada objeto contendrá todos los campos “method” de los mensajes que se hayan recibido, esto es: será obligatorio que contengan el campo “key” con la API key del bot, además de todos los campos adicionales del método que se haya enviado. Todos los mensajes recibidos se reciben en el orden en que han sido enviados, de antes a después.

```
{
   "status":{
      "code":0,
      "message":"OK"
   },
   "method":{
      "dbid":0,
      "key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
      "type":"BULK_METHOD",
      "methods":[
         {
            "dbid":0,
            "key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
            "type":"MESSAGE_RECEIVED",
            "id":"2a5bf37c-5070-44ed-80a2-217c138fdea0",
            "message":"hola"
         },
         {
            "dbid":0,
            "key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
            "type":"MESSAGE_RECEIVED",
            "id":"808a9c3e-1364-45c5-86ba-28a91b15f86f",
            "message":"hola"
         },
         {
            "dbid":0,
            "key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
            "type":"CONVERSATION_ENDED",
            "id":"c7021b74-8573-4131-b562-7064f3e7831b",
            "isBot":true
         }
      ]
   },
   "timestamp":1715012377891
}
```


Código 8.2.1.2.1 - Mensaje en JSON de ejemplo para el método BULK_METHOD

CHANGE_USERNAME

Método usado para cambiar el nombre de usuario del bot en la plataforma. Este nombre de usuario se mostrará en los rankings de bots. Si el nombre de usuario no está disponible, se enviará un error 4003.

- **"username"**: Campo adicional que debe añadirse, cuyo valor es el nombre de usuario que se desea.


```
{
   "status":{
      "code":0,
      "message":"OK"
   },
   "method":{
      "dbid":0,
      "key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
      "type":"CHANGE_USERNAME",
      "username":"fmpg0003"
   },
   "timestamp":1715003751155
}
```


Código 8.2.1.2.2 - Mensaje en JSON de ejemplo para el método CHANGE_USERNAME

CONNECT

Este método conecta al bot a la plataforma. Está explicado en el apartado 8.2, y su envío tras estar identificado supondrá la recepción de un código de error 4000.

CONVERSATION_ENDED

Este método puede ser enviado o recibido para indicar que se quiere finalizar/se ha finalizado una conversación asociada al bot. Los campos adicionales que deben ser añadidos son:

- **"id"**: Indica el identificador de la conversación que se quiere finalizar/se ha finalizado.

- **"isBot"**: Indica la decisión del otro interlocutor (o la del bot, de mandar el método él), sobre si su otro interlocutor es o no un bot Su valor debe ser true/false.


```
{
   "status":{
      "code":0,
      "message":"OK"
   },
   "method":{
      "dbid":0,
      "key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
      "type":"CONVERSATION_ENDED",
      "id":"f730215f-5665-4e25-b22a-62fdf7e4cbd6",
      "isBot":true
   },
   "timestamp":1715003751328
}
```


Código 8.2.1.2.3 - Mensaje en JSON de ejemplo para el método CONVERSATION_ENDED

CONVERSATION_STARTED

Este método es enviado por la plataforma indicando al bot que se ha iniciado una nueva conversación. El bot puede, a partir de este momento, enviar mensajes a esta conversación por medio del método *MESSAGE_SEND*, explicado más abajo. Enviar este mensaje al servidor supondrá la recepción de un método *STATUS_RELAY* con código de error 4001.

- **"id"**: Campo adicional asociado a este método. Indica el identificador de la conversación que se ha iniciado.

```
{
   "status":{
      "code":0,
      "message":"OK"
   },
   "method":{
      "dbid":0,
      "key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
      "type":"CONVERSATION_STARTED",
      "id":"f730215f-5665-4e25-b22a-62fdf7e4cbd6"
   },
   "timestamp":1715003751330
}
```


Código 8.2.1.2.4 - Mensaje en JSON de ejemplo para el método CONVERSATION_STARTED

DISCONNECT

Este método es enviado por, y puede enviarse a la plataforma para indicar que el bot quiere desconectarse de ella o que la plataforma lo ha desconectado por alguna razón. No tiene campos adicionales, pero es recomendable revisar el estado del mensaje para revisar si la desconexión ha sido por algún error. Enviar este método puede hacerse con cualquier código de error; no es revisado por la plataforma.


```
{
   "status":{
      "code":0,
      "message":"OK"
   },
   "method":{
      "key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
      "type":"DISCONNECT"
   },
   "timestamp":1715003751331
}
```


Código 8.2.1.2.5 - Mensaje en JSON de ejemplo para el método DISCONNECT

ENTER_MATCHMAKING

Este método puede ser enviado para indicarle a la plataforma que el bot está preparado para recibir nuevas conversaciones. Una vez enviado este método, sucesivos envíos del mismo supondrán en la recepción de un mensaje *ENTER_MATCHMAKING* con código de error 4002. No es posible salir del matchmaking una vez se ha entrado en él sin desconectarse de la plataforma - Se espera que los bots estén disponibles para tantas conversaciones como sean necesarias.


```
{
   "status":{
      "code":0,
      "message":"OK"
   },
   "method":{
       "dbid":0,
      "key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
      "type":"ENTER_MATCHMAKING"
   },
   "timestamp":1715004448872
}
```


Código 8.2.1.2.6 - Mensaje en JSON de ejemplo para el método DISCONNECT

MESSAGE_RECEIVED

Este método lo envía el servidor para indicarle al bot que ha recibido un nuevo mensaje. Enviar este método como cliente supondrá la recepción de un mensaje con método *STATUS_RELAY* con código de error 4001. Este método tiene los siguientes campos adicionales:

- **“id”:** La ID de la conversación en la cual se ha enviado un mensaje.
- **“message”:** El contenido del mensaje que se envía: Estos tienen todos los caracteres especiales markdown que el usuario haya enviado, de haber alguno.

```
{
    "status":{
        "code":0,
        "message":"OK"
    },
    "method":{
        "dbid":0,
        "key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
        "type":"MESSAGE_RECEIVED",
        "id":"f0815a5d-87b0-475a-9501-688317c4700e",
        "message":"buenos días!"
    },
    "timestamp":1715004448873
}
```


Código 8.2.1.2.7 - Mensaje en JSON de ejemplo para el método MESSAGE_RECEIVED

MESSAGE_SEND

Este método puede enviarse al servidor para enviar un mensaje a una conversación. Son necesarios los siguientes parámetros:

- **"id":** La ID de la conversación a la que se quiere enviar un mensaje. Si la conversación ha finalizado, la ID es incorrecta o no pertenece al bot con la API Key que se haya especificado al iniciar la conexión con el bot, este enviará un mensaje con método *STATUS_RELAY* y código de error 404.
- **"message"**: El contenido del mensaje que se envía. Es necesario que los mensajes enviados estén correctamente formados, esto es: Es necesario que caracteres especiales para markdown sean correctamente escapados si no se utilizan para dar formato al mensaje. Si el mensaje está malformado o su longitud es mayor a 4096 caracteres (el máximo permitido por Telegram en toda la plataforma [39]), el servidor enviará un mensaje con método *STATUS_RELAY* y código de error 500, y el mensaje no será publicado en el chat del interlocutor.

```
{
   "status":{
      "code":0,
      "message":"OK"
   },
   "method":{
      "dbid":0,
      "key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
      "type":"MESSAGE_SEND",
      "id":"f0815a5d-87b0-475a-9501-688317c4700e",
      "message":"cómo estás?"
   },
   "timestamp":1715004448708
}
```


Código 8.2.1.2.8 - Mensaje en JSON de ejemplo para el método MESSAGE_SEND

STATUS_RELAY

Un mensaje especial para que la plataforma le indique al bot el estado de un mensaje anterior. El envío de un mensaje con este método a la plataforma supondrá la recepción de un mensaje con método *STATUS_RELAY* y código de error 4001. Este método tiene como campo adicional:

- **"trigger":** Un objeto, siendo este el mensaje JSON que hizo que se enviase este mensaje. Útil para revisar qué mensaje dio lugar a este error.

```
{
   "status":{
      "code":404,
      "message":"No existe una conversación con la ID especificada."
   },
   "method":{
      "dbid":0,
      "key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
      "type":"STATUS_RELAY",
      "trigger":{
         "status":{
            "code":0,
            "message":"OK"
         },
         "method":{
            "dbid":0,
"key":"rVT4b4MEnqfm2GtYXQJ3p4pPw4XvRAETWQ5GTahbYmXhMBVA8YmNGsXduar6XCmj",
            "type":"MESSAGE_SEND",
            "id":"f0815a5d-87b0-475a-9501-688317c4700e",
            "message":"cómo estás?"
         },
         "timestamp":1715004448708
      }
   },
   "timestamp":1715004448876
}
```


Código 8.2.1.2.9 - Mensaje en JSON de ejemplo para el método STATUS_RELAY

8.3 - Librería para Java de la API de la plataforma

Se ha creado una pequeña clase, llamada TesTABotConnector, dentro de la plataforma. Esta clase implementa, en lenguaje Java, todas las funcionalidades de la API indicadas en el apartado anterior, de manera que sean invisibles para el desarrollador del bot. Es una clase abstracta que implementa lo siguiente:

8.3.1 - Métodos implementados

Todos estos métodos pueden ser llamados cuando la conexión con la plataforma se haya estabilizado (excepto loginBot(), que inicia dicha conexión). No pueden ser cambiados por el desarrollador en implementaciones de esta clase.

- **Constructor(apiKey, URL, port)**: Inicializa la clase con los parámetros indicados, los cuales son la llave de acceso a la plataforma, su dirección y el puerto.
- **loginBot():** Función que permite al desarrollador conectar el bot a la plataforma. Tras ejecutarse, se creará automáticamente un nuevo hilo siempre activo, hasta la desconexión de la plataforma, que permite al usuario recibir eventos del bot automáticamente.
- **changeUsername(newUsername)**: Cambia el nombre de usuario del bot al indicado por el parámetro.
- **sendMessage(where, message):** Envía un mensaje a la ID de la conversación indicada por el primer parámetro.
- **enterMatchmaking():** Indica a la plataforma que el bot está listo para recibir nuevas conversaciones.
- **endConversation(id):** Finaliza la conversación con la ID pasada como parámetro.
- **disconnect():** Finaliza la conexión con la plataforma de manera grácil.

8.3.2 - Métodos abstractos que deben ser implementados

Estos métodos son los que deben ser modificados por el creador del bot para que este funcione, creando una subclase nueva que implemente la funcionalidad de TesTABotConnector. Todos estos métodos son llamados de manera automática cuando sea necesario para comunicarse el bot y la plataforma.

- **onConversationEnded(event):** Esta función se llamará automáticamente cuando un interlocutor finalice una conversación con este bot. Puede revisarse el objeto de tipo

ConversationEnded, que tiene en formato de clase de Java toda la información relativa al método CONVERSATION_ENDED que se envía por socket.

- **onError(message):** Esta función se llama cuando el servidor envía un mensaje indicando que ha habido un error. Como parámetro, se incluye el mensaje que se ha recibido para ser analizado. Estos mensajes implementan la misma estructura que los mensajes que se envían y reciben del Socket. Su estructura puede revisarse en el apartado 8.2.1.

- **onLogin():** Esta función se ejecuta de manera automática cuando el bot termina su conexión a la plataforma. No es necesaria su implementación para su correcto funcionamiento (puede dejarse como un método vacío), pero es útil para inicializar clases o para debugging.

- **onMatchmakingEntered():** Esta función se ejecuta automáticamente cuando la plataforma ha indicado que el bot ha entrado correctamente en matchmaking. Igual que la función onLogin, no es necesario que esta función realice ninguna acción para el correcto funcionamiento del bot.

- **onMessageReceived(event):** Esta función se ejecuta cuando un interlocutor ha enviado un mensaje en una conversación en la que el bot actúa como interlocutor. Como argumento se añade la implementación en Java del método MESSAGE_RECEIVED que se envía por socket.

- **onNewConversation(event):** Esta función se ejecutará cuando el bot sea emparejado con un interlocutor. Se envía como parámetro la implementación en Java del método CONVERSATION_STARTED que se envía por socket, conteniendo este la ID de la conversación para futura referencia del bot.

- **onServerDisconnected(status):** Esta función se ejecuta si el bot ha sido desconectado de la plataforma por alguna razón específica. Como argumento, se indica la implementación en Java del objeto “status” de los JSON que se envían por socket. Esto es, podrá obtenerse acceso al código de error y el mensaje indicando el detalle de la desconexión.

- **onStatusRelay(message):** Esta función se ejecuta si se recibe un mensaje de estado del servidor que no sea un error. Por defecto, todos los errores van al método onError, pero mensajes de estado que solo indiquen estados neutrales o positivos de la plataforma van a este método. El mensaje que se pasará como parámetro tiene la misma estructura que los mensajes que se envían y reciben del socket, con el añadido de que estos implementarán siempre el método STATUS_RELAY.

9 - Quality Assurance. Problemas encontrados y soluciones.

Esta sección de la documentación se centrará en lo ocurrido en la fase de escrutinio del proyecto: fallos encontrados y sus soluciones, problemas durante el desarrollo y cómo se resolvieron.

9.1 - Quality Assurance (QA)

Para realizar pruebas en el software con el objetivo de encontrar errores, avisé a un amigo para que me ayudara probando todas las opciones posibles y fallos comunes que pudieran ocurrir en la plataforma. El resultado de esta búsqueda de fallos se detalla a continuación:

9.1.1 - Nombres en la plataforma

Se probaron inyecciones SQL (las cuales no podían funcionar debido a que se usan wildcards en las consultas SQL), además de datos basura con el objetivo de hacer que la base de datos fallase o se almacenase información equivocada.

La base de datos tiene un cotejamiento que evita que haya problemas insertando emojis y caracteres especiales no soportados en teclados españoles, de modo que la plataforma no falló.

Sin embargo, se encontró un fallo: El código trataba de insertar el nombre en la base de datos sin probar primero que tuviera una longitud correcta. La base de datos rechazaba nombres demasiado largos pues estaba correctamente diseñada, y enviaba un error “dato demasiado largo para esta columna”, el cual simplemente se mostraba como “Error conectando a la base de datos. Espera un momento y vuélvelo a intentar”, pues no se había tenido en cuenta que pudiera fallar por un nombre demasiado largo. Esto no era un problema que afectase a la estabilidad de la plataforma, pero no es un error de la base de datos sino un problema con los datos suministrados por el usuario, por lo que el error no era correcto. Se arregló fácilmente con una condición que revisaba el tamaño del nombre de usuario antes de tratar de insertarlo.

9.1.2 - Envío y recepción de mensajes largos

Dentro de una conversación, se probaron a enviar mensajes de más de 4096 caracteres (el máximo permitido por Telegram). El resultado fue que Telegram tiene implementado un mecanismo que parte un mensaje en varios de ser estos demasiado largos. Dentro de la API, esto simplemente implica que en lugar de enviar un mensaje de 5000 caracteres, se envía

uno de 4096 y otro de 4. Puesto que la base de datos almacena los mensajes con longitud de hasta 4096 caracteres, dentro de la base de datos simplemente se guarda un mensaje de longitud máxima y otro de longitud 4, con una diferencia de tiempo de unos pocos milisegundos por cómo está implementada la API de la plataforma y el tiempo que se tardó en enviar cada parte del mensaje.

9.1.3 - Bots: Desconexión en mitad de una conservación

Revisando la conexión a la plataforma mediante su API para desarrolladores se intuyó que era muy probable que esta no fuera capaz de mantener todo el contexto de una conversación si el bot se desconectaba de ella a la mitad. Tras reconectarse, si su interlocutor humano había enviado un mensaje, este nunca era recibido por el bot.

Puesto que uno de los principios de la aplicación es que tenía que no degradar en lo absoluto la calidad de las conversaciones un fallo ajeno a la plataforma, se decidió que la API actuase como lo haría el propio Telegram: Si no hay conexión a internet, se guardarán los mensajes hasta que esta conexión se reinstate, y se enviaría todo dato que no se hubiera podido mandar por un único método. Fue debido a eso que se creó el método *BULK_METHOD.* Se editó la estructura de la base de datos para poder almacenarse estos nuevos campos en ella, eliminándose cuando ya hayan sido mandados para evitar ocupar espacio innecesario.

9.1.3 - Caídas de plataforma

Se simularon caídas de la plataforma, apagándose en mitad de una operación o manteniéndose apagadas durante un periodo de tiempo prolongado con el objetivo de descubrir qué ocurría con mensajes sin enviar y botones pulsados, además de descubrir posibles fallos en la integridad de la API para bots.

Se descubrieron varias cosas:

- Telegram implementa un método similar a nuestro *BULK_METHOD* en la plataforma: Si un bot se desconecta, tras reconectarse recibirá todas las interacciones que los usuarios hayan realizado mientras este estuviera desconectado. Esto implica que si un usuario pulsó un botón hace 1 hora, cuando el bot no estaba conectado, el bot recibirá esta interacción y la completará en el momento en que esté online. Todos los mensajes que los usuarios de la plataforma mandasen se recibirán en granel.
- Al desconectar la plataforma, la API de bots se cae. Esto es obvio, y no se puede evitar que ocurra, pero la implementación de la API para bots estaba ideada para reintentar una conexión cuando ésta se interrumpiera. Resulta que nuestra API era

incapaz de reconstruir la conexión, incluso si la plataforma consigue restablecer su conexión. Se optó por evitar una reconexión mientras se investigaba una causa.

- Al final se consiguió arreglar este problema. La librería en Java para la API, de detectar esta desconexión, empieza a tratar una reconexión cada 5 segundos. De no lograrlo, lo vuelve a intentar en 6 segundos, después 7, después 8… hasta lograrlo o ser interrumpida. Una vez consiga restablecer la conexión, la API volverá a funcionar con normalidad, tal y como si hubiera acabado de entrar en la plataforma.

9.1.4 - Conexiones malintencionadas y envío incorrecto de datos

Tras investigaciones anteriores con proyectos ajenos a este TFG, se concluyó lo siguiente: Hay muchísimas máquinas que tratan por todos los medios robar información conectándose a sockets inseguros por toda la web; abrir un puerto implica recibir de manera constante una marea de conexiones que intentan abusar posibles vulnerabilidades enviando datos basura o tratando de conectar como si de un puerto SSH se tratase.

Para poder contraatacar y evitar que estos malos actores colapsen la memoria de la aplicación, se implementó de manera preventiva un mecanismo en la API que impide que las conexiones no autenticadas duren más de 5 segundos. De no recibirse un dato, o recibirse un dato incorrecto en estos primeros 5 segundos, la plataforma finaliza la conexión automáticamente, liberando todos los recursos usados.

Tras el despliegue de la aplicación, se pasó a probar si había algún desperfecto que evitase la desconexión automática o que nos permitiera burlar la seguridad. A continuación se describe lo que ocurrió con la prueba y cómo se realizó:

- La conexión se realizó por medio de Telnet, una herramienta inicialmente ideada para el manejo de una máquina remotamente [40], pero que puede ser utilizado de igual manera para lanzar conexiones TCP contra un puerto. Las aplicaciones Telnet envían un paquete TCP ACK para iniciar la conexión, y tras ello el usuario es libre de enviar paquetes con el texto que quiera.

- Tras conectarnos, de no enviar ningún mensaje más que el paquete ACK, la conexión se cierra automáticamente; esta medida funciona.

- Sin embargo, de enviar mensajes basura con contenido que no pueda ser deserializado por Jackson, se produce una excepción no capturada en el programa. Esta excepción no hace que el hilo que escucha conexiones para los bots de la plataforma deje de funcionar, y mantiene la conexión abierta, mostrando un mensaje de excepción en la consola que no es necesario para ningún administrador de la plataforma (y que puede ensuciar los logs al ser estas conexiones bastante comunes y continuadas).

    - Como solución, se capturó la excepción y se trató como un error de JSON, enviando el mensaje de error y la desconexión del cliente. Además, para evitar problemas con otras excepciones no capturadas, se añadió que toda excepción no capturada cierre el socket.

- Tras esto, se probó a enviar datos incorrectos tras la conexión correcta de un bot en la plataforma: Debido a que no se maneja correctamente la excepción, la conexión se cancelaba (no es lo que se quiere). Se editó el código fuente para reparar este imprevisto, haciendo que cada vez que se envíe un dato incorrecto se responda a la petición indicando que la petición está malformada.

- Se probó a realizar acciones prohibidas, como enviar mensajes a conversaciones ya finalizadas o que no pertenecían al bot, y si bien el programa fallaba al hacerlo, enviaba un mensaje indicando error interno del servidor en lugar de un mensaje con más información, por lo que se cambió el código fuente para indicar errores relativos a acciones no permitidas.

- Por último(incluso si no era eso lo que tratábamos de probar), debido a que el bot por defecto no lo hacía, se procuró finalizar una conversación usando el método CONVERSATION_ENDED, el cual se ejecutaba en el servidor incorrectamente, impidiendo finalizar la conversación. Se arregló este pequeño error, consiguiendo así que los bots pudieran finalizar conversaciones por sí mismos debidamente.

9.2 - Dificultades encontradas durante el desarrollo

Como toda aplicación, siempre ocurren imprevistos en el desarrollo, como una librería que fuese demasiado complicada o poco clara de utilizar creando retrasos en el desarrollo. Aquí se detallan todos los imprevistos que se han solucionado:

9.2.1 - Sockets en Java

Uno de los principales problemas con la aplicación fue el uso de los sockets en Java. Estos tienen una implementación y documentación que deja bastante que desear, con lo que ciertos problemas surgieron por no saber cuál era el estado de la aplicación al no indicarlo por medio de mensajes.

Uno de los principales problemas fue el estado tras cerrarse uno de los flujos de datos de los sockets. Si se alcanza el final del flujo de datos (esto es solo válido en estos flujos de datos cuando se cierra una conexión o el flujo de entrada o salida de manera explícita en la conexión), las funciones de lectura del flujo devuelven null, pero también pueden provocar una excepción debido a que se trata de leer un flujo cerrado. La primera implementación asumió que se lanzaría una excepción, de modo en que esto provocaba problemas. Al final se

acabó esperando que se lance una excepción y comprobando que no se devuelve null, y de hacerlo, lanzar otra excepción también.

Otro problema fue la necesidad de tener que utilizar la función flush(). Una implementación de los flujos de escritura era la del uso de PrintWriter, una utilidad que permitía la escritura de manera sencilla (¡o eso pensaba!) en un flujo. Uno de los parámetros del constructor era un valor booleano que activaba una función "auto flush". A la hora de enviar datos por medio del protocolo TCP, este no sabe cuándo debería de enviarlos. Es debido a esto que es necesario utilizar esta función para indicarle al protocolo que un mensaje está listo para ser enviado [41]. Esto no es un problema, es algo que se espera de la implementación del protocolo. Sin embargo, la implementación de PrintWriter permite usar el flush automáticamente cuando se escriben datos al socket… excepto que no lo hace con todas las funciones que permiten enviar datos al buffer. La implementación para convertir mensajes a JSON añade los caracteres '\r\n' (retorno de carro y salto de línea) al final del flujo de datos, cosa que la función println (que utiliza el auto-flush) no hace (solo añade '\n'), por lo que no podía ser usada. La función printf es una función pesada, así que se optó por usar únicamente la función append() que no tiene auto-flush. Esto provocó bastantes dolores de cabeza, pues muchos mensajes no se mandaban correctamente.

9.2.2 - JSON con Jackson: Polimorfismo de métodos

La implementación de Jackson en el proyecto fue por medio de subclases de una clase Method principal. Todos los métodos que se envían por socket utilizan una subclase de Method. Por defecto, a la hora de realizar una deserialización del JSON que se ha enviado al cliente/servidor, este trata de rellenar automáticamente los campos de una clase usando Reflection, un mecanismo de Java. Debido a que simplemente se le indica al usuario que los mensajes usan un Method, Jackson no sabe qué subclase queremos. Para solventar esto, se añadió un campo al JSON, llamado "type", dentro de cada objeto Method, que indica cuál es el subtipo de la clase que queremos. Sin embargo, esto no es trivial en Jackson, y fue necesario revisar su documentación para encontrar la solución; esta está indicada en la sección 4.3 de este documento.

9.2.3 - Comportamiento correcto del modelo para el bot de ejemplo

Para poder dar una versión de la aplicación que funcione por sí misma, necesitamos un bot en la plataforma, conectado siempre. Para solventar esto, se añadió una clase específica, OpenAIBot, dentro del código fuente de la aplicación, que simplemente conecta con la plataforma por medio de un socket local. Esta implementación utiliza, como se indica en el nombre, un modelo de OpenAI. La implementación es simple, pasando únicamente el texto recibido por el usuario al modelo y devolviendo el resultado tras aplicarle unas instrucciones

y el contexto de la conversación, pero las respuestas que el bot da a estos mensajes se alejan mucho de un bot tratando de ser humano. Incluso instrucciones claras como “no uses mayúsculas” son ignoradas en algunos mensajes.

Hacer un bot que sea capaz de engañar a un ser humano no es el objetivo de este TFG, por lo que no se hizo demasiado hincapié en que fuese más convincente, pero esto hizo que se requiriese trabajo adicional para hacer un bot que, al menos, fuese capaz de decir “no” cuando le preguntas si es una máquina. Era necesario también ser lo más conciso posible con las instrucciones, pues son tokens adicionales que es necesario pasar al bot cada vez que se busca una respuesta a un mensaje en la conversación, y cada token cuesta dinero. Las instrucciones finales y el resto de la implementación del bot están indicadas en la sección 4.7 de este documento.

9.2.4 - Discordancia entre la base de datos y la aplicación

Un problema que consistía en cambios en la base de datos (los cuales eran necesarios para poder solucionar nuevos problemas que aparecían tras la implementación a nivel de código de ciertas necesidades de la aplicación) que no se reflejaban en el código. Al modificarse la estructura de la base de datos, añadiendo campos adicionales en las tablas, las consultas antiguas para insertar nuevas filas en la base de datos quedaban inválidas, debido a que se la consulta no indicaba una lista de columnas, sino simplemente todas en su conjunto.

La solución a este problema fue sencilla, simplemente indicando la lista de columnas que rellenar cada vez que se inserte una fila nueva en cada consulta en el código fuente del proyecto, para que SQL automáticamente ponga como “NULL” el resto de columnas, pero muchas pruebas de la aplicación fallaron debido a que pequeños cambios que no se consideraron crearon errores en la base de datos que impedían el funcionamiento de la plataforma.

Otra solución podría haber sido una librería de SQL que permitiera revisar las consultas para evitar así problemas de estructura, pero estos problemas comenzaron cuando el proyecto ya estaba bastante avanzado, de modo en que un cambio de librería habría sido más trabajo que cambiar las consultas una a una.

9.2.5 - Cantidad de mensajes en Telegram y su implementación a nivel de código

Debido a la gran cantidad de estados posibles en la plataforma, se decidió dividir el código en pequeñas funciones para cada uno de los mensajes que se envían al usuario en Telegram. La implementación para decidir qué comando o botón pulsado hace qué se realiza por medio

de switches, y cada caso concreto llama a una función específica, pero esto complicó bastante la implementación del conector de Telegram, pues la clase acabó teniendo hasta 1.100 líneas de código.

Una solución podría haber sido el implementar Reflection y cada uno de los posibles estados como un HashMap, de manera en que el código fuese más sencillo implementar nuevas funciones, pero este problema no se hizo evidente hasta bastante avanzado el proyecto, por lo que cuando se consideró la idea ya era más trabajo el cambio que continuar la implementación actual.

10 - Conclusiones

Una vez finalizado este proyecto, las conclusiones finales que se han obtenido respecto a este se reflejan en esta sección. Al final, se indica una serie de posibles mejoras o añadidos para la aplicación, y cómo se podría continuar mejorando de cara al futuro.

10.1 - Cumplimiento de los requisitos

El resultado final de este proyecto es una aplicación que cumple con todos los requisitos. Se detalla a continuación el resultado final de cada requisito:

- **RF-1:** Se cumple en su totalidad por medio del uso de Telegram, plataforma la cual era obligatoria por el TFG. Esta plataforma acepta comandos, tiene botones y permite bots.
- **RF-2:** El sistema de registro es perfectamente funcional respecto a lo indicado en los subapartados, implementado por medio de la base de datos de MariaDB:
    - **RF-2A:** La plataforma únicamente necesita saber el identificador del usuario de Telegram.
    - **RF-2B:** Los términos y condiciones están claramente especificados. Estos términos y condiciones también cumplen el requisito RN-5. Se puede observar esto en el apartado 7.5 - LOPD GDD y cómo afecta a la aplicación, en específico la imagen 7.5.1.
    - **RF-2C:** Existe un botón para rechazar los términos y condiciones, además de otro para anonimizar los datos si se han rechazado los términos y condiciones. De hacerse esto, la aplicación volverá a tratar al usuario como si no hubiera entrado nunca a la plataforma.

- **RF-3:** La implementación de este requisito está explicada en el apartado 7.1 - Matchmaking.
    - RF-3A: Existe un botón que cancela el emparejamiento, disponible en el mensaje que se manda al usuario mientras espera a un interlocutor.
    - RF-3B, RF-3D, RF-3E: Sus implementaciones están indicadas en el apartado 7.1.
    - RF-3C: La implementación del emparejamiento está hecha por medio de clases Semaphore de Java, en hilos distintos del principal. Por ello, de querer cancelar el emparejamiento, la aplicación puede interrumpir el hilo, mandando esto una excepción que se revisa y se trata como salida del emparejamiento.
- **RF-4:** Este requisito está implementado siguiendo lo indicado en el apartado 7.2 - Interacción entre dos usuarios humanos.
    - RF-4A: La plataforma actúa como intermediaria, de modo en que los mensajes se envían a, y se reciben de la plataforma. Esto impide tener información sobre la otra persona/bot.
    - RF-4B: Los mensajes no se procesan en el apartado del servidor, simplemente se envían al usuario que deba enviarse. Esto los hace prácticamente instantáneos, si bien de vez en cuando pueden ocurrir ligeros cambios en el orden de los mensajes si estos se envían con pocos milisegundos de retardo.
    - RF-4C: En caso de enviar una imagen, audio, documento… a la plataforma, esta directamente ignora el mensaje.
    - RF-4D: El comando en concreto es /finalizar. Funciona exactamente como se describe en el requisito, y está indicado en el apartado 6.2.
- **RF-5:** Este requisito se implementa respecto a lo indicado en 7.4 - Ranking y estadísticas.
    - RF-5A: El apartado 7.4.1 - Experiencia y niveles explica cómo se ha resuelto este requisito.
    - RF-5B: Tal y como está implementado este sistema, el límite de experiencia sería el máximo valor de un entero de 64 bits, cosa que a efectos prácticos por la implementación del sistema y la experiencia, no debería ser considerado un límite; prácticamente infinito.
    - RF-5C: Resuelto en el apartado 7.6 - Motes en la plataforma.

    - RF-5D: La implementación cumple con este requisito limitando la consulta a la base de datos a 10 usuarios. Si no se detecta al usuario en este límite, se añade al final junto a la posición dentro de la base de datos.

- **RN-1**: Telegram es seguro en la medida de que utiliza números de teléfono y autenticación en dos pasos.
- **RN-2**: El registro es pulsar dos botones: Uno para visualizar los términos y condiciones, y otro para aceptarlos. No hay nada más.
- **RN-3**: El emparejamiento es abrir el menú y pulsar el botón para nueva conversación. Tras eso, esperar.
- **RN-4:** Las conversaciones no tienen límite ni de tiempo ni de número de mensajes, resultado de la implementación de las mismas: Cada mensaje se guarda de manera automática en la base de datos tras mandarse, haciendo innecesario un límite.
- **RN-5**: El apartado 7.5 - LOPD GDD y cómo afecta a la aplicación explica cómo la aplicación cumple con las leyes pertinentes.
- **RN-6:**
    - RN-6A: Dependerá de la persona que lance la aplicación y del servidor donde esté desplegada. La aplicación no tiene, por defecto, impedimentos para poder funcionar de manera constante.
    - RN-6B: La conexión por medio de sockets de Java soluciona este requisito.
    - RN-6C: Las Api Keys, explicadas en el apartado 8 - API para desarrolladores: Creación de bots para la plataforma, permiten que se cumpla este requisito.
    - RN-6D: La implementación de la plataforma para bots contiene ciertos métodos que permiten enviar los datos relativos a una acción incorrecta o error sin que esto afecte a la aplicación y su estado.

10.2 - Comentario del autor

Esta aplicación ha sido un desafío que ha necesitado la implementación de muchas metodologías e ideas estudiadas en la carrera: Sistemas concurrentes y distribuidos (pues si hay 200 usuarios conectados, cada uno debe de tratarse en un hilo diferente, además de evitar que se accedan a recursos para lectura mientras se está escribiendo), estructuras de datos (para obtener la mejor estructura para cada tipo de dato almacenado sin afectar a la velocidad de la aplicación), sistemas operativos (instalación), diseño e implementación de servidores (despliegue de la aplicación), diseño de algoritmos (implementación greedy del matchmaking), programación general y orientada a objetos (obvio), gestión y control de proyectos informáticos (Git), procesamiento del lenguaje natural (para entender mejor cómo

funcionan los modelos de IA y cómo se entrenan los modelos de lenguaje), redes (por la implementación del protocolo TCP y el paso de mensajes), interacción persona-ordenador (usabilidad de la aplicación, internacionalización)... Todas han sido requeridas para poder crear esta aplicación.

La creación de esta aplicación ha sido algo que me ha interesado, sobre todo por el hecho de aprender cómo funcionan diversas APIs que están muy a la orden del día, además de considerar que es perfecta como proyecto final. Si bien conocía muchas herramientas para poder crear esta aplicación, he aprendido bastante sobre modelos de inteligencia artificial, además de afianzar los conocimientos sobre sockets, la implementación de APIs y el despliegue de aplicaciones por Docker.

La aplicación final es robusta e implementa todo lo requerido, además de dejarla en un estado en el que puede ser mejorable y escalable sin demasiada dificultad, correctamente documentada y con varias herramientas para que sea lo más sencillo posible su uso.

10.3 - Posibles mejoras

El proyecto cumple con todo lo indicado en los objetivos, pero de ser necesaria una ampliación del mismo, hay diversos puntos que se pueden tener en consideración:

- **Ampliación de la API para desarrolladores:** De ser necesario (no se ve necesidad con la funcionalidad actual de la plataforma), podría ampliarse esta API de manera sencilla modificando las clases de los métodos dentro del código fuente del proyecto. Existen algunos métodos que no implementan ninguna funcionalidad adicional respecto de la superclase Method, pero que se quedaron separados de manera intencional, para poder hacer que cualquier mejora fuera sencilla de implementar sin tener que crear nuevas clases. Adicionalmente, de ser necesarios nuevos métodos, se pueden crear por medio de la superclase.

- **Experiencia dinámica:** Otorgar mayor cantidad de puntos de experiencia a los humanos o a los bots si estos son capaces de pasar diversos parámetros, como número de mensajes enviados, rachas de victorias…

- **Envío y recepción de imágenes:** Modificar la API para bots para permitir el envío y recepción de imágenes, para poder así probar con respecto a otros bots cómo de buenos son analizando imágenes. Con respecto a la aplicación actual, cualquier envío de una imagen es ignorado por la plataforma: El mensaje directamente no se procesa.

- **Uso de más herramientas de Telegram:** La aplicación final utiliza botones en los mensajes, pero existen también funcionalidades para añadir botones desde un menú

aparte de los mensajes, por ejemplo, que añadirían más usabilidad a la aplicación.

- **Mostrar estadísticas de más bots:** De igual manera que se pueden observar las estadísticas de la persona que está usando la plataforma, mostrar estadísticas de un bot concreto del ranking, mostrando el número de usuarios que ha engañado o su porcentaje, motor que está usando, si está siempre activo o se desconecta de la plataforma… Esto puede ser visible para todos los usuarios o solo para administradores.

- **Panel de control de administrador:** Añadir una funcionalidad y un campo adicional en la base de datos para poder abrir, desde Telegram, un menú donde poder modificar datos de la plataforma sin necesidad de acceso remoto al servidor que lo almacena.

- **Emparejamiento automático con un bot si no hay humanos disponibles:** Para evitar que el sistema mantenga en espera durante demasiado tiempo a un usuario humano, emparejarlo automáticamente con un bot si ha pasado demasiado tiempo desde que entró en búsqueda de interlocutor. Esto no se ha implementado en la aplicación por motivos que se investigaron en el apartado de investigación inicial: Afecta bastante a la calidad de los resultados, pues en momentos donde la plataforma tenga pocos usuarios subiría demasiado el porcentaje de emparejamientos con bot, lo cual hace más sencillo averiguar si estás hablando con uno. Sin embargo, es un cambio que podría realizarse, de considerarse beneficioso a largo plazo.

- **Bloquear a personas de la plataforma:** Por el momento, la mejor acción contra un usuario que utiliza nombres ofensivos en la plataforma es retirar su nombre, pero podría implementarse algún método para restringir el acceso a la misma a usuarios que incumplan las normas de manera reiterada.

Anexo: Manual de despliegue

Esta guía se puede obtener en el repositorio de la aplicación en Gitlab [42].

Habla con TesTA - Instalación de la plataforma

La aplicación de Habla con TesTA ofrece varias opciones para la instalación de la aplicación. Se detallan a continuación:

Usando Docker

Si has instalado Docker en tu sistema, la instalación de la aplicación como contenedor es extremadamente más sencilla que la instalación manual. Para instalar este proyecto se puede usar el repositorio TesTA-Docker que contiene todos los ficheros necesarios, incluyendo *Dockerfile* y *docker-compose.yml* necesarios para crear y ejecutar el contenedor, junto a un contenedor SQL para almacenar los datos.

De manera manual

La manera "a la vieja usanza" de utilizar la aplicación.

Requisitos

- JDK 19+, para ejecutar la aplicación.
- Maven, para compilar la aplicación (opcional)
- Una base de datos MariaDB.

[OPCIONAL] Compilación manual

Esto solo es necesario en caso de que se quiera construir un fichero .jar desde el código fuente del proyecto. De coger los ficheros .jar del repositorio, se puede saltar esta sección.

- Clona el repositorio de Habla con TesTA o descárgalo como fichero .zip.
- Abre un terminal en la carpeta donde esté el fichero pom.xml del proyecto.
- Ejecuta el comando ***maven mvn -f pom.xml clean package*** para compilar el proyecto. Esto generará los ficheros TesTA-1.0.jar y TesTA-1.0-jar-with-dependencies.jar. Salvo que se quiera utilizar la aplicación por la librería que esta contiene (la API para bots),

se debe coger el fichero jar-with-dependencies, que contiene todas las clases necesarias para que se pueda ejecutar directamente el fichero. En caso contrario, el fichero sin dependencias es suficiente.

Ejecución de la aplicación

1. Coge los siguientes ficheros de la carpeta base del repositorio que has descargado anteriormente:

    a. **config.testa.example** *(Renombrar a config.testa)*: El fichero de configuración de la aplicación. Tendrá que ser rellenado con la conexión a la base de datos, la API key para Telegram y cualquier configuración que quiera cambiarse.

    b. **logging.properties:** El fichero que contiene la configuración del logger que Java utiliza. La configuración por defecto es válida; puede realizarse cualquier cambio que se considere oportuno.

    c. **schema.sql:** El esquema de la base de datos. No debe ser modificado, pues hacerlo supondrá que la aplicación no funcione correctamente al conectarse a la base de datos.

2. Coloca los ficheros en una carpeta, junto al fichero TesTA-1.0-jar-with-dependencies.jar descargado (o compilado) anteriormente.

3. Accede al servidor SQL que se utilizará para almacenar los datos de la plataforma.

4. Accede a la nueva base de datos creada.

5. Importa el fichero schema.sql. Si tu gestor de bases de datos lo permite, importa el fichero sql desde tu sistema. De no ser posible esta acción, debería ser suficiente con copiar todo el contenido del fichero .sql y ejecutarlo con tu gestor de base de datos.

6. Configura el fichero de configuración config.testa con los datos de conexión a la base de datos. La cadena de caracteres debería ser algo similar a:

    ```
    mariadb://URL_SERVIDOR:PUERTO/NOMBRE_BASE_DE_DATOS?charset=utf8mb4&collation=utf8mb4_unicode_ci
    ```

    Nótese que los parámetros de la conexión son necesarios para almacenar correctamente las conversaciones en la base de datos.

7. Para iniciar el servidor, una vez configurado completamente, se puede escribir el comando

    ```
    java -Djava.util.logging.config.file=./logging.properties -jar TesTA-1.0-jar-with-dependencies.jar
    ```

8. Si todo ha funcionado correctamente, la plataforma estará abierta y preparada para recibir bots y conexiones de usuarios. Desde Telegram, mandar el comando /start al bot cuya API Key esté en la configuración iniciará el proceso de registro con el usuario que haya mandado el mensaje.

Para programar un nuevo bot usando la librería

La aplicación de Habla con TesTA ofrece una librería para bots, accesible por medio de la clase *TesTABotConnector.jar*. La compilación con Maven de la aplicación genera un Javadoc con toda la documentación de la API para bots que se puede visualizar. Más información sobre la compilación por medio de Maven de la aplicación está disponible en su sección correspondiente más arriba. Sea cual sea el método por el cuál se obtengan los ficheros .jar de la aplicación, será necesario el fichero *TesTA-1.0.jar* (No el fichero *jar-with-dependencies*).

- Crea tu aplicación Java para el bot.
- Añade *TesTA-1.0.jar* como dependencia, ya sea por medio de Maven o por cualquier método que tu IDE tenga disponible para hacerlo.
- Crea una nueva clase que herede de TesTABotConnector. Rellena todos los parámetros.
- En tu programa, inicializa un nuevo objeto del tipo la clase que has creado anteriormente.
- Llama a la función loginBot() para inicializar la conexión.

A partir de este momento, el bot se conectará a la plataforma. Más información sobre la conexión, los mensajes enviados, los métodos disponibles en la clase y cualquier otra consulta con la API está disponible en la documentación del proyecto.

Anexo: “Readme“ de la aplicación (con guía de uso)

Disponible en el proyecto de Gitlab junto a la guía de instalación [42].

Habla con TesTA

Una aplicación creada en Java para la implementación de un Test de Turing en Telegram. La aplicación:

- Empareja usuarios entre ellos y con bots
- Permite acceso a un ranking de usuarios y bots, permitiendo ver cuáles son mejores en descubrir a bots/engañar a humanos.
- Permite el acceso a bots conversacionales desde cualquier parte del mundo, por medio de una API robusta.
- Es una librería, a su vez, para la creación de bots para la plataforma.

Este proyecto es un Trabajo de Fin de Grado para la Universidad de Jaén, licenciado bajo Creative Commons BY-NC-SA, versión 4.0.

- LICENCIA
- INSTALACIÓN

Ejecución del programa

Para ejecutar este programa con la máquina virtual de Java, tras haber sido compilado, puede usarse la siguiente sintaxis:

```
  java -jar TesTA.jar [--lang=RUTA_ARCHIVO_LENGUAJE]
[--botman=PARÁMETRO_BOT_MANAGER]
```

Donde:

- **RUTA_ARCHIVO_LENGUAJE**: Es la ruta del fichero que contiene los Strings con el idioma de la aplicación. Por defecto, de no ponerse, se sustituye por *lang.properties*. Ejemplos de una ruta válida incluyen *./../archivo.properties* y */home/user/lang.properties*.
- **PARÁMETRO_BOT_MANAGER**: Es el parámetro de administración de bots de la plataforma que se ejecutará. Indicar este parámetro inicia la aplicación en modo administración de bots, donde únicamente se ejecuta esa opción y se finaliza el programa; no se inicia la plataforma. Por defecto no hay parámetro (la plataforma se inicia). Los parámetros de este modo pueden ser:

- **list, lista** o **l**: Muestra una lista de los bots que se han creado usando este programa. El bot con ID -1, que es el bot por defecto que se puede iniciar activando la opción OPENAI_USE_BOT
- **add, añadir** o **a**: Añade un nuevo bot a la plataforma.
- **del, delete, remove, rem, borrar, d, r** o **b**: Borra un bot de la plataforma por su ID de la base de datos.
- **update, u, actualizar, ac, reiniciar** o **reset**: Reinicia la API key de un bot de la plataforma (la API key antigua dejará de poder utilizarse).

Una sintaxis correcta de ejecución del programa con todos los parámetros sería:

```
java -jar TesTA.jar --lang=lang.properties --botman=añadir
```

Archivo de configuración

De no existir el fichero config.testa en la carpeta donde esté la aplicación, se creará un nuevo fichero con las opciones de configuración en blanco para ser configuradas. En el repositorio existe el fichero *config.testa.example* que puede ser renombrado a *config.testa*, con explicación de qué hace cada parámetro.

Logger

La aplicación utiliza el Logger por defecto de Java. Para utilizarlo, se puede crear un fichero *.properties* (se adjunta uno en el repositorio, *logging.properties*) y configurarlo con las opciones del Logger de Java. Para aplicar el cambio en el sistema de logging de la aplicación es necesario que se ejecute con el parámetro *-Djava.util.logging.config.file=**FICHERO*** (Donde **FICHERO** es la ruta del fichero que contiene las propiedades a aplicar).

Plataforma para bots

La plataforma se abre en el puerto especificado en la configuración del programa. Será necesario que el firewall del sistema permita el acceso al puerto específico para que bots desde fuera de la máquina puedan conectarse.

Será necesario especificar la ID dentro de la base de datos y la API Key del bot que se esté tratando de conectar a la plataforma cuando lo haga por medio de la API.

Si se realiza una conexión por medio de la dirección loopback 127.0.0.1 (local), se da cualquier API key como correcta.

El bot automáticamente crea un usuario con ID -1 para la conexión con el bot por defecto de la plataforma si se ejecuta esta con el parámetro *OPENAI_USE_BOT*. Este bot no aparecerá

en la lista de bots cuando se ejecuta el parámetro lista en el BotManager, y no tiene API Key, por lo que cualquier conexión remota será rechazada automáticamente.

Bibliografía y fuentes

[1] «Prueba de Turing», *Wikipedia, la enciclopedia libre*. 4 de abril de 2024. Accedido: 4 de abril de 2024. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Prueba_de_Turing&oldid=159219033

[2] TuringTestChat, «Yeah I shut it down.…», r/Humanornot. Accedido: 4 de abril de 2024. [En línea]. Disponible en: www.reddit.com/r/Humanornot/comments/14ivipt/turing_test_chat_a_unofficial_successor_to/kbi677z/

[3] «AI21 Labs concludes largest Turing Test experiment to date». Accedido: 4 de abril de 2024. [En línea]. Disponible en: https://www.ai21.com/blog/human-or-not-results

[4] «HUMAN OR AI? Can I guess correctly? [ALL EPISODES] - YouTube». Accedido: 4 de abril de 2024. [En línea]. Disponible en: https://www.youtube.com/watch?v=0DIixhg2V_c

[5] _Sonari_, «Yes, if it says "you…», r/TuringTestChat. Accedido: 4 de abril de 2024. [En línea]. Disponible en: www.reddit.com/r/TuringTestChat/comments/168zbcy/im_quite_new_to_the_game_and_ive_been_wondering/jyygr24/

[6] *HUMAN OR AI? Can I FOOL them? [2] #shorts #ai #funny #chatgpt*. Accedido: 4 de abril de 2024. [En línea Video]. Disponible en: https://www.youtube.com/shorts/Dptn7ZA3_t8

[7] «Telegram», *Wikipedia, la enciclopedia libre*. 25 de marzo de 2024. Accedido: 4 de abril de 2024. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Telegram&oldid=159029587

[8] «Telegram APIs». Accedido: 4 de abril de 2024. [En línea]. Disponible en: https://core.telegram.org/

[9] «Bots: An introduction for developers». Accedido: 4 de abril de 2024. [En línea]. Disponible en: https://core.telegram.org/bots

[10] «Bot API Library Examples». Accedido: 4 de abril de 2024. [En línea]. Disponible en: https://core.telegram.org/bots/samples

[11] «C++ Memory Management: new and delete». Accedido: 6 de abril de 2024. [En línea]. Disponible en: https://www.programiz.com/cpp-programming/memory-management

[12] «Concurrency support library (since C++11) - cppreference.com». Accedido: 6 de abril de 2024. [En línea]. Disponible en: https://en.cppreference.com/w/cpp/thread

[13] «java.util.concurrent (Java Platform SE 8)». Accedido: 6 de abril de 2024. [En línea]. Disponible en: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html

[14] C. Á. Caules, «Java Garbage Collector», Arquitectura Java. Accedido: 6 de abril de 2024. [En línea]. Disponible en: https://www.arquitecturajava.com/java-garbage-collector/

[16] W. Young, «Answer to “On which unix distributions is Python installed as part of the default install?”», Unix & Linux Stack Exchange. Accedido: 6 de abril de 2024. [En línea]. Disponible en: https://unix.stackexchange.com/a/24808

[17] «threading — Thread-based parallelism», Python documentation. Accedido: 6 de abril de 2024. [En línea]. Disponible en: https://docs.python.org/3/library/threading.html

[18] «How do you create private attributes in a Python class?», w3resource. Accedido: 6 de abril de 2024. [En línea]. Disponible en: https://www.w3resource.com/python-interview/how-do-you-create-private-attributes-in-a-class.php

[19] R. Bermudez, «rubenlagus/TelegramBots». 3 de abril de 2024. Accedido: 4 de abril de 2024. [En línea]. Disponible en: https://github.com/rubenlagus/TelegramBots

[20] «Sondeo largo». Accedido: 6 de abril de 2024. [En línea]. Disponible en: https://es.javascript.info/long-polling

[21] «GitHub - FasterXML/jackson: Main Portal page for the Jackson project». Accedido: 11 de abril de 2024. [En línea]. Disponible en: https://github.com/FasterXML/jackson

[22] «MariaDB o MySQL: diferencia entre bases de datos relacionales de código abierto. AWS», Amazon Web Services, Inc. Accedido: 11 de abril de 2024. [En línea]. Disponible en: https://aws.amazon.com/es/compare/the-difference-between-mariadb-vs-mysql/

[23] S. Ravoof, «MariaDB vs PostgreSQL: 14 Critical Differences», Kinsta®. Accedido: 11 de abril de 2024. [En línea]. Disponible en: https://kinsta.com/blog/mariadb-vs-postgresql/

[24] «¿Qué es NoSQL y cómo funciona? | Google Cloud», Google Cloud. Accedido: 11 de abril de 2024. [En línea]. Disponible en: https://cloud.google.com/discover/what-is-nosql?hl=es-419

[25] «OpenAI Platform». Accedido: 6 de abril de 2024. [En línea]. Disponible en: https://platform.openai.com

[26] «Assistant API: How to set limit of context window for GPT-4 turbo - API», OpenAI Developer Forum. Accedido: 18 de abril de 2024. [En línea]. Disponible en: https://community.openai.com/t/assistant-api-how-to-set-limit-of-context-window-for-gpt-4-turbo/534869

[27] T. Kanning, «TheoKanning/openai-java». 17 de abril de 2024. Accedido: 17 de abril de 2024. [En línea]. Disponible en: https://github.com/TheoKanning/openai-java

[28] jerelquay, «jerelquay/docker-maven-sample». 4 de febrero de 2020. Accedido: 24 de mayo de 2024. [En línea]. Disponible en: https://github.com/jerelquay/docker-maven-sample

[29] bezkoder, «Docker Compose: Spring Boot and MySQL example», BezKoder. Accedido: 24 de mayo de 2024. [En línea]. Disponible en: https://www.bezkoder.com/docker-compose-spring-boot-mysql/

[30] «Salario para Ingeniero Informático en España - Salario Medio», Talent.com. Accedido: 6 de abril de 2024. [En línea]. Disponible en: https://es.talent.com/salary

[31] «Pricing». Accedido: 6 de abril de 2024. [En línea]. Disponible en: https://openai.com/pricing

[32] «Buttons». Accedido: 17 de mayo de 2024. [En línea]. Disponible en: https://core.telegram.org/api/bots/buttons

[33] J. Lee, «Converting Levels Into XP & Vice Versa», Jake Lee on Software. Accedido: 30 de abril de 2024. [En línea]. Disponible en: https://blog.jakelee.co.uk/converting-levels-into-xp-vice-versa/

[34] «LOPD: Qué es y cómo cumplirla (Ley Orgánica de Protección de Datos)», Usercentrics. Accedido: 17 de mayo de 2024. [En línea]. Disponible en: https://usercentrics.com/es/knowledge-hub/guia-lopd/

[35] «¿Qué es el derecho a la portabilidad? | AEPD». Accedido: 17 de mayo de 2024. [En línea]. Disponible en: https://www.aepd.es/prensa-y-comunicacion/blog/que-es-el-derecho-la-portabilidad

[36] «Derecho de oposición | AEPD». Accedido: 17 de mayo de 2024. [En línea]. Disponible en: https://www.aepd.es/derechos-y-deberes/conoce-tus-derechos/derecho-de-oposicion

[37] «Derecho a la limitación del tratamiento | AEPD». Accedido: 17 de mayo de 2024. [En línea]. Disponible en: https://www.aepd.es/derechos-y-deberes/conoce-tus-derechos/derecho-la-limitacion-del-tratamiento

[38] «jBCrypt - strong password hashing for Java». Accedido: 26 de mayo de 2024. [En línea]. Disponible en: https://www.mindrot.org/projects/jBCrypt/

[39] «Telegram», Messaging. Accedido: 30 de mayo de 2024. [En línea]. Disponible en: https://developers.cm.com/messaging/docs/telegram

[40] «Telnet», *Wikipedia, la enciclopedia libre*. 14 de febrero de 2024. Accedido: 5 de junio de 2024. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Telnet&oldid=158186237

[41] B. J. Homer, «Answer to “What does it mean to flush a socket?”», Stack Overflow. Accedido: 17 de mayo de 2024. [En línea]. Disponible en: https://stackoverflow.com/a/914321

[42] «Francisco Miguel Pulido González / Habla con TesTA · GitLab», GitLab. Accedido: 5 de junio de 2024. [En línea]. Disponible en: https://gitlab.com/fmpg0003/TesTA