UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

MEJORA DE LA EFECTIVIDAD DE UN AGENTE VIRTUAL PARA APOYAR EL CUIDADO INFORMAL DE ADULTOS MAYORES

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

PEDRO PABLO BELMONTE LARREDONDA

PROFESOR GUÍA:
FRANCISCO GUTIÉRREZ FIGUEROA

MIEMBROS DE LA COMISIÓN:
NELSON BALOIAN TATARYAN
SERGIO OCHOA DELORENZI
JOSÉ PIQUER GARDNER

SANTIAGO DE CHILE
2021

Resumen

El "envejecimiento en el hogar" permite a los adultos mayores conservar cierta independencia, además de mantenerse en un ambiente familiar, lo cual puede ser muy beneficioso. Sin embargo, esto también conlleva una carga para los familiares más cercanos que asumen el rol de cuidadores informales. Esta carga para los familiares no se visibiliza demasiado, y carece de apoyo formal.

En este contexto surge la idea de desarrollar una herramienta que apoye el cuidado informal de adultos mayores. Esta herramienta, cuyo nombre es Hermes, fue desarrollada como parte del trabajo de título de Andrea Benavides, y es un concentrador de información útil para el cuidado del adulto mayor, además de integrar funciones de agenda y coordinación de actividades, usando un chatbot como intermediario.

Una de las funciones del chatbot es contactar a los usuarios del sistema para preguntarles si pueden hacerse cargo de una tarea que se ha agendado recientemente. Pensando en eso, se considera que sería útil que el chatbot tenga información de eventos pasados sobre cómo resultó intentar agendar a cada usuario a cada tarea en particular. Por esto se decide hacer un sistema de caracterización de usuarios, en base a sus interacciones pasadas con el chatbot.

Además, el chatbot desarrollado no se encuentra completamente funcional, y usa un software privado. Considerando esto, se propone hacer un nuevo chatbot con herramientas opensource para reemplazar al antiguo, y que se conecte con este nuevo sistema de caracterización.

El objetivo de este trabajo es entonces avanzar en el desarrollo de Hermes, creando un chatbot nuevo con otra tecnología, y desarrollando un sistema de caracterización para mejorar la efectividad y eficiencia de asignación de tareas en el futuro.

Ambos aspectos del trabajo fueron desarrollados con cierto nivel de éxito, pues al final del trabajo se tiene un nuevo chatbot conectado a la aplicación móvil que utiliza el backend del sistema, y que genera perfiles históricos con eventos que se crean cuando se le pregunta a un usuario si puede tomar una tarea en particular. Sin embargo, los objetivos no se pueden considerar completamente logrados, pues quedan pendientes algunas conexiones con el sistema legado y tanto el chatbot como el sistema de caracterización de usuarios se pueden mejorar. No se realiza una evaluación formal con usuarios reales, pues la usabilidad de la aplicación móvil ya había sido probada con usuarios reales usando interfaces en papel, pero se comenta la efectividad de la solución desarrollada como herramienta para lograr la mejora de la efectividad del sistema.

Tabla de Contenido

Índice de Ilustraciones

Capítulo 1

Introducción

A menudo, los adultos mayores son dependientes de otras personas para subsistir, debido al deterioro natural de sus condiciones físicas y psicológicas asociadas al envejecimiento [1]. Por distintos motivos, ya sean económicos, emocionales u otros, los adultos mayores frecuentemente prefieren envejecer en su hogar, en lugar de ser cuidados en una institución profesional [2]. Si bien existen muchos beneficios relacionados al "envejecimiento en el hogar", esta situación también conlleva una carga extra para la familia del adulto mayor, quienes usualmente asumen distintos roles como cuidadores informales (o cuidadores familiares) [3].

En la cultura chilena es común que los familiares cercanos al adulto mayor (usualmente, los hijos) asuman responsabilidades en el cuidado de este último [4]. Sin embargo, la realización de estas actividades puede volverse difícil para los cuidadores familiares por diversos motivos. Por ejemplo, si esos cuidadores viven en una región distinta a los adultos mayores, o si trabajan tiempo completo.

Las tecnologías de comunicación interpersonal, tales como los servicios de mensajería instantánea y de videollamadas, tienen el potencial de ser un apoyo importante y facilitar el cuidado y la comunicación con los adultos mayores. Sin embargo, a menudo las personas de la tercera edad carecen de iniciativa o conocimientos para adoptar estas tecnologías [5, 6]. Incluso si ellos son reacios a adoptar las nuevas tecnologías, éstas siguen siendo una herramienta poderosa para coordinar el trabajo conjunto de las personas que asumen distintos roles como cuidadores familiares.

Es en este contexto donde surge el desarrollo del sistema Hermes, el cual es un agente virtual que busca apoyar el cuidado informal de adultos mayores mientras envejecen en su hogar. Este sistema fue desarrollado por Andrea Benavides como parte de su trabajo de título [7, 8].

El sistema es un concentrador de la información del proceso de cuidado de un adulto mayor. Además, integra funciones de agenda y coordinación de las actividades de los participantes, usando un chatbot como intermediario. Algunos usuarios pueden comunicarse con el sistema mediante una aplicación móvil, mientras que otros —los que están menos comprometidos con el proceso de cuidado— pueden simplemente comunicarse con el bot mediante una aplicación de mensajería como Telegram.

Cuando se ingresa al sistema una tarea de cuidado del adulto mayor que requiere apoyo externo, el chatbot debe contactar a algunos usuarios para pedirles que se hagan cargo de dicha tarea. Los usuarios a contactar se deciden generando grupos arbitrarios, en los cuales el chatbot va a preguntar a todos los usuarios de un grupo y, en caso de no tener éxito se mueve al siguiente. El usuario puede decidir adjudicársela (es decir, asumir realizarla de manera voluntaria), o rechazar la petición. Si todos los potenciales cuidadores que fueron invitados rechazan la invitación, entonces el chatbot inicia una nueva ronda de invitaciones a personas del círculo familiar del adulto mayor, que aún no han sido contactadas por este agente.

Siguiendo ese esquema de funcionamiento, Hermes busca facilitar la asignación y coordinación de actividades de los cuidadores. Sin embargo, el chatbot implementado anteriormente es bastante básico y le faltan funcionalidades que son fundamentales para hacer más efectivo el comportamiento del sistema. Por ejemplo, para intentar asignar de mejor forma las tareas a los cuidadores, y balancear la carga de trabajo entre ellos.

1.1. Problema Abordado

Previamente, Hermes usa el sistema de BotCenter de Adereso[1] para implementar gran parte de la funcionalidad del chatbot. Este tecnología solo estaba disponible para el desarrollo de una prueba de concepto, pues el acceso a esta fue posible gracias a un favor de la empresa dueña de la misma, y su uso en la actualidad tiene un costo asociado. Debido a esto, ya no es factible seguir utilizando la tecnología de BotCenter[2], y por lo tanto, este trabajo de memoria se ve en la necesidad de rehacer el chatbot utilizando un nuevo sistema, preferentemente open-source, para así poder seguir mejorándolo con toda libertad.

Sumado a esto, el nuevo chatbot debe ayudar a mejorar la efectividad del sistema Hermes, identificando a quienes debería intentar asignarle una determinada tarea en un momento dado, usando para ello el historial pasado de comportamiento de los usuarios, pues el sistema originalmente usaba grupos arbitrarios para decidir a quién contactar primero. Además, debería intentar balancear la carga de trabajo entre los diferentes participantes, tratando de no sobrecargar a nadie.

Por todo los antes mencionado, esta memoria debe extender el trabajo previo [7, 8], rehaciendo el chatbot y generando un sistema que permita caracterizar usuarios. De esa manera, el sistema podrá volverse más efectivo en su labor de conseguir voluntarios para realizar las tareas asociadas al cuidado familiar de los adultos mayores. En ese sentido, el chatbot deberá actuar de manera diferenciada dependiendo del perfil del participante con el que interactúa, y para ello considerará la historia previa (comportamiento pasado) de cada usuario, a fin de hacer las interacciones más amenas y efectivas.

Con el fin de reducir al mínimo los cambios necesarios al sistema actual (es decir, a Hermes), el nuevo chatbot debe emular el comportamiento y las especificaciones de la API del agente que se usa actualmente. Luego, lo ideal es utilizar una herramienta para desarrollar

[1] https://www.adere.so/
[2] https://developers.botcenter.io/es/

bots que sea similar a la de BotCenter, y hacer una API intermedia para que las llamadas que debe hacer la aplicación móvil, sean las mismas que se hacen para interactuar con el bot actual.

Para incorporar la información del comportamiento pasado de un usuario, en el contexto de una conversación, el sistema debe tomar en consideración diversos aspectos, tales como: la frecuencia con la que el usuario contesta o ignora los mensajes, cuán a menudo asume las tareas que se le proponen, el tipo de tareas que usualmente lleva a cabo, si frecuentemente olvida o no sus tareas asignadas, o si es reacio a efectuar ciertas tareas, entre otros.

Para hacer uso de la información histórica de cuidado del adulto mayor, es necesario abordar algunos desafíos importantes. Hay que lograr registrar dicha historia en un formato que sea útil, y utilizarla para determinar cómo manejar la interacción con el chatbot de manera apropiada. En este sentido, el sistema debe ser capaz de responder diversas preguntas, tales como: "¿quién sería el miembro de la familia más apropiado para llevar a cabo esta tarea?", "¿qué usuarios tienen más posibilidades de aceptar realizar la tarea?", Si un usuario asume una tarea, "¿necesita ese usuario que el sistema le recuerde el compromiso asumido?", "¿usualmente ese usuario responde cuando se le pide algo? ¿responde sí o no? ¿ignora los mensajes?", y "¿a qué hora es más conveniente contactar a este usuario?", entre otras.

Tener una respuesta a estas preguntas permite asignar tareas de manera eficiente, y no agobiar a los usuarios del sistema. Sin embargo, responder estas preguntas no es sencillo, pues implica almacenar la información de manera inteligente, guardando más que simplemente los datos crudos de una conversación. Esto requiere detectar tendencias en el comportamiento del usuario, y codificarlas para su uso posterior en el sistema. El protocolo para esto debe ser definido con cuidado.

1.2. Objetivos

1.2.1. Objetivo general

Este trabajo de memoria pretende avanzar en el desarrollo de Hermes, generando un chatbot propio y un protocolo que permitirá mejorar la efectividad del sistema, a través de la mejora de la inteligencia de su chatbot mediante la caracterización de usuarios. Particularmente, dicho chatbot debe permitir mejorar la tasa de aceptación de tareas por parte de los cuidadores informales, al momento que dicho agente los invita a hacerse cargo de ellas. Así, el proceso futuro de decidir a quién invitar y cuándo invitar, deberá considerar el perfil de los participantes, y la carga de trabajo ya asumida por ellos.

1.2.2. Objetivos Específicos

Para alcanzar el objetivo general se han planteado los siguientes objetivos específicos:

1. Desarrollar un nuevo chatbot, usando herramientas open-source. Esto implica también

hacer cambios en la aplicación móvil para conectarse a este nuevo chatbot e implementar la conexión con la aplicación de mensajería Telegram para que lo usen los usuarios que no utilizan la aplicación móvil. Este objetivo se considera logrado si se tiene un chatbot al cual se pueda contactar a través de la aplicación móvil y a través de Telegram.

2. Generar un perfil para cada participante (cuidador informal) en base a la historia de las interacciones pasadas entre el chatbot y esa persona. Idealmente, este perfil debe permitir determinar el mejor horario para contactar a una persona, identificar su tiempo de respuesta promedio, o la probabilidad de que esa persona realice una cierta tarea. Se considera logrado si se tiene un sistema de perfiles que se puedan crear, modificar y leer que tenga información respecto a los intentos de asignación de tareas.

3. Desarrollar un sistema en la API de Hermes que permita mantener actualizado el perfil de usuario de todos los participantes, usando la información de interacciones que se va almacenando en el historial. Dicho sistema debe permitir elaborar otro sistema para sugerir candidatos para que asuman una cierta tarea. Este objetivo se considera logrado si la API de Hermes permite interactuar con los perfiles para crear, modificar y leerlos.

1.3. Solución Desarrollada

Para rehacer el chatbot sin cambiar su funcionamiento y diseño (es decir, que tenga los mismos flujos de conversación y efectue las mismas acciones que el original), se decidió usar una herramienta cuya forma de definir el comportamiento de un bot sea similar a la que se usa en BotCenter, pero que sea open-source. Para esto, una buena opción fue Botpress[3], pues permite modelar las conversaciones en base a nodos, usando una interfaz gráfica con drag-and-drop, y con los conceptos de intenciones y entidades, al igual que el sistema de BotCenter.

A pesar de que se quería hacer la menor cantidad de cambios posibles a la aplicación móvil y a la API, en realidad la conexión con el chatbot por parte de la aplicación no estaba completamente funcional, por lo cual en la aplicación se redefine la lógica completa para comunicarse con el chatbot y se hace una vista completamente nueva.

Sumado a esto, se crea un nuevo modelo en la API y la base de datos para la caracterización de usuarios, con sus endpoints correspondientes.

Además, inicialmente se pensó utilizar Whatsapp como aplicación externa de mensajería para contactar a los usuarios que no utilicen la aplicación móvil. Sin embargo, finalmente se decide utilizar Telegram por su facilidad a la hora de conectar chatbots y las herramientas que ofrece, pues superan a Whatsapp en muchos aspectos. Entre estas herramientas destacan la posibilidad de crear un bot sin un número de teléfono y el soporte para botones en el chat.

Para el sistema de caracterización se agrega un nuevo servicio al backend con sus endpoints correspondientes. Estos endpoints permiten ver, agregar y modificar eventos de un perfil, guardando información útil de cada evento. Además, usando estos eventos se genera información agregada como porcentajes y tiempos de respuesta.

[3] https://botpress.com/

El chatbot hace llamadas a la API para conectarse con este sistema de caracterización, guardando cada evento de intento de asignación con su información correspondiente.

1.4. Resultados

Si bien se implementan con éxito el chatbot y el sistema básico de caracterización, se considera que estos aún están incompletos, y no están listos para usarse en un ambiente productivo con usuarios reales.

El chatbot contiene todos los flujos requeridos, pero se podrían mejorar un poco para que la conversación sea más natural, mejorando la experiencia de usuario. Además, el sistema de asignación del backend todavía no está usando el chatbot nuevo para contactar a los usuarios. La aplicación móvil sí está conectada con el chatbot nuevo, y el chatbot puede usar Telegram para contactar a los usuarios si así se le indica.

En cuanto al sistema de caracterización, si bien es funcional y guarda datos que pueden ser útiles, es un poco básico y no se ha implementado un sistema de sugerencias de candidatos que lo use, por lo cual también se considera incompleto.

1.5. Estructura del Documento

En este documento se explica primero el marco teórico de este trabajo, entregando el contexto necesario para entender la necesidad a abordar en la memoria. Además se explican conceptos básicos de chatbots y en particular de Botpress, que es la herramienta utilizada para el desarrollo de este trabajo.

A continuación, se explica el sistema legado, con lo que tenía implementado y lo que tenía propuesto, para dar más contexto a lo que se desarrolla en este trabajo. Se comenta sobre los componentes que tiene y qué cosas hay que considerar para conectar los nuevos componentes.

Luego, se comenta la concepción de la solución, en donde se define qué debe contener la solución y cómo se planea resolver cada problema, explicando a grandes rasgos la arquitectura y el diseño.

Posteriormente, se explica en detalle el sistema implementado, desde el punto de vista del chatbot, el frontend y el backend. Se muestran los distintos flujos y las partes del sistema, y lo que ocurre con cada situación.

Finalmente se comenta sobre los resultados del trabajo, sobre su utilidad y nivel de éxito, y sobre lo que se podría mejorar a futuro para tener un sistema que realmente resuelva el problema de mejorar la efectividad y eficiencia de Hermes.

Capítulo 2

Marco Teórico

A continuación se contextualiza el problema a abordar, considerando para qué sirve el sistema Hermes. También se explica a grandes rasgos los chatbots y como funcionan particularmente en la herramienta Botpress. Finalmente se explica la utilidad que podría tener un sistema de caractarización de usuarios y cómo se podría concebir y utilizar.

2.1. Cuidado informal de adultos mayores

El cuidado informal de adultos mayores no es un proceso estructurado, no tiene un flujo de trabajo bien definido. Los participantes del cuidado del adulto mayor se coordinan según su situación, contexto y capacidades actuales de manera arbitraria. Además, el trabajo de los cuidadores informales tiende a ser invisible [9] y carece de apoyo formal [10].

Este trabajo típicamente recae sobre familiares cercanos al adulto mayor [3, 11], quienes a menudo también pueden ser adultos mayores o presentar problemas de salud que les complican aún más el cuidado de otro adulto mayor [10].

En este contexto, resulta útil distribuir roles y responsabilidades entre los integrantes del grupo familiar para llevar a cabo las actividades de cuidado de forma más eficiente y efectiva. Se considera útil también centralizar la comunicación y coordinación entre los cuidadores en un sistema colaborativo que permita facilitar estas tareas.

Este sistema debería permitir a los participantes tener claro en todo momento las necesidades que se deben satisfacer en relación al cuidado del adulto mayor y quién es la persona encargada de llevar a cabo las actividades requeridas.

Con esto en mente surge la idea del sistema Hermes [7, 8], pensado para resolver estos problemas aplicando tecnología. Uno de los elementos requeridos en la concepción de este sistema es un chatbot, mediante el cual los cuidadores pueden generar requerimientos para satisfacer las actividades necesarias en el cuidado del adulto mayor; y que también sirve como mediador para asignar tareas preguntándole a los usuarios si desean tomar la responsabilidad de una de estas.

2.2. Chatbots

Un chatbot es un sistema computacional cognitivo que emula conversaciones humanas para proveer servicios de información, transaccionales y conversacionales [12]. Los chatbots permiten automatizar procesos que normalmente requieren interactuar con personas. Un uso típico de estos es ser la primera capa de comunicación para descongestionar la atención al cliente resolviendo las dudas o problemas más básicos. De esta forma gestionan la mayor cantidad de requerimientos, reservando para los ejecutivos aquellos que realmente requieren intervención humana.

A continuación se describe cómo se trabaja desarrollando un chatbot utilizando la herramienta de Botpress [13], que es la herramienta utilizada en este trabajo para crear el chatbot.

2.2.1. Anatomía de un bot

El ciclo de vida de un bot de Botpress se puede resumir de manera bastante simple:

1. Recibe un mensaje de un **canal** de mensajería
2. **Procesa** el mensaje para entenderlo
3. **Decide** qué responder al usuario mediante el canal

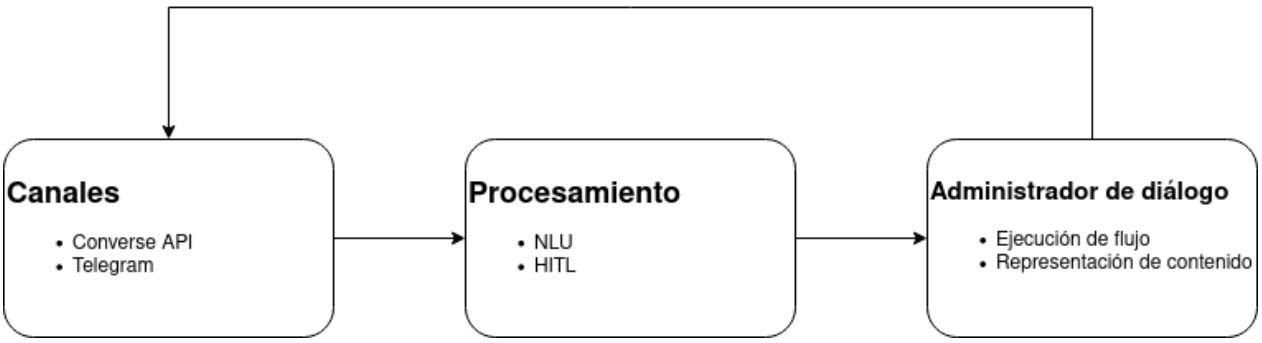


Figura 2.1: Ciclo de vida básico de un bot de Botpress.

2.2.1.1. Canales

En este trabajo se utilizan los canales Converse API y Telegram para comunicarse con el bot. Converse API permite enviar y recibir mensajes mediante peticiones HTTP y es lo que se usa en la aplicación móvil. Telegram es el canal que permite usar la aplicación de mensajería Telegram para hablar con el bot.

2.2.1.2. Procesamiento

Natural Language Understanding (o NLU) permite al bot procesar el mensaje del usuario de manera inteligente para convertirlo en datos estructurados. Las tareas que se pueden

realizar mediante la herramienta de NLU de Botpress son:

- **Reconocimiento de intenciones:** Reconocer qué quiere hacer el usuario mediante intenciones previamente definidas y ejemplificadas (mediante los ejemplos se entrena el modelo para el reconocimiento)
- **Extracción de entidades:** Extraer información estructurada a partir de los mensajes como fecha, hora, nombres, etc
- **Etiquetado de slots:** Identificar parámetros (como fecha o lugar) para rellenar entidades
- **Identificación de lenguaje:** Saber el idioma que se está usando

Human in the loop (o HITL) permite informar a un humano cuando se necesita atención extra durante una conversación, pues el chatbot no puede encargarse por sí solo. Esta herramienta no se usa en este trabajo por lo cual no se profundizará más en ella.

2.2.1.3. Administrador de diálogo

Una vez que el bot ha procesado los datos de un mensaje recibido por un usuario debe decidir qué hacer a continuación. El administrador de diálogo de Botpress se compone de flujos que contienen nodos de conversación. En cada uno de estos nodos se pueden poner condiciones para ir a otros nodos (o flujos), ejecutar código, presentar opciones, o simplemente responder un mensaje y avanzar a algún nodo en particular.

Así, cuando el bot recibe un mensaje se entra al nodo inicial del flujo principal y se avanza desde ahí según cómo se vaya desarrollando la conversación. Los mensajes que entrega el bot son representados a medida según el canal que se esté usando.

2.3. Caracterización de Usuarios

Si bien las personas tienden a tener desconfianza en los chatbots, es posible mejorar esta confianza si este trata al usuario de manera personalizada según el individuo con el cual se está comunicando [14].

Sumado a esto, en este sistema es importante también considerar la carga de trabajo de cada uno de los participantes, para no sobrecargar demasiado a una sola persona. Es decir, el chatbot debería evitar intentar asignarle una tarea a un cuidador que ya tiene muchas otras tareas asignadas.

Además, sería útil que el sistema busque la mayor efectividad posible a la hora de intentar convencer al usuario, considerando por ejemplo la hora a la cual es mejor preguntarle o qué tan seguido debería insistir.

En resumen, es prudente crear un sistema de caracterización de usuarios que permita evaluar el comportamiento de estos para generar información útil para mejorar la efectividad del

sistema Hermes. Un sistema como este permitiría generar algoritmos que mejoren la efectividad del sistema y la experiencia de los cuidadores informales.

Existen trabajos previos que tratan de perfilar personas en base a sus interacciones en un chat [15, 16], a partir de los cuales se puede obtener inspiración. De todas maneras es necesario idear un chatbot enfocado en abordar este problema particular, con las características propias de este, buscando hacerlo más efectivo a partir del uso de información histórica de los usuarios para ajustar su comportamiento.

Capítulo 3

Análisis del Sistema Legado

A continuación se presenta la estructura y el comportamiento básico del sistema legado. Además, se muestra la necesidad de implementar un nuevo chatbot y ajustar la solución en ese sentido.

3.1. Estructura general del sistema legado

Tal como se mencionó antes, el sistema Hermes fue diseñado para apoyar la coordinación de actividades en favor de los adultos mayores, por parte de los miembros de su comunidad familiar principalmente. Sin embargo, tal como se muestra en la Figura 3.1, en este proceso también participan actores externos, como por ejemplo cuidadores ocasionales (vecinos o amigos del adulto mayor) y también el personal médico que le hace seguimiento al adulto mayor, cuando corresponde.

Como se observa en la figura 3.1, los cuidadores principales y los adultos mayores que requieren cuidados utilizan la aplicación móvil, aunque para estos últimos es opcional. En la aplicación móvil se pueden crear recordatorios, revisarlos, iniciar una conversación con otros usuarios, etc. El backend luego trata de asignar tareas contactando a los usuarios ya sea mediante la aplicación móvil o mediante la aplicación de mensajería externa.

3.2. Front-end de Hermes

La estructura de las vistas de la aplicación móvil es bastante simple. La pantalla principal que se observa en la Figura 3.2 contiene enlaces a un “Panel de noticias” donde se pueden ver los eventos, y los chats con los demás cuidadores y el chatbot. En el caso de los chats, hay una diferencia entre los usuarios de que los que usan la aplicación pueden ser contactados mediante un chat directo en esta y los otros solo tienen la opción para llamar por teléfono.

En la vista de la Figura 3.2, la opción “Hermes” lleva a la conversación con el chatbot mientras que cada nombre que aparece abajo permite comunicarse con esa persona. Si las personas también usan la aplicación se puede abrir un chat directo a través de esta mientras que las personas que no usan la aplicación aparecen oscurecidas y solo se puede usar el teléfono.

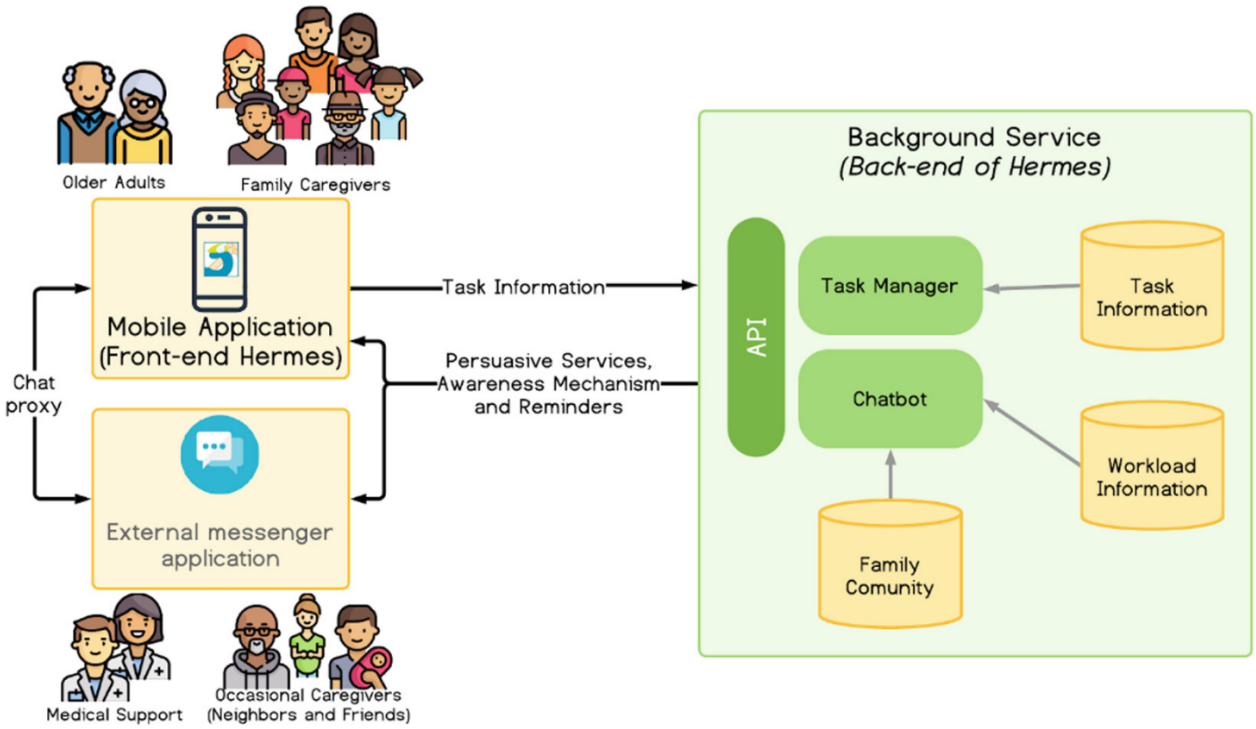


Figura 3.1: Estructura del sistema legado (Hermes), según lo indicado en [7, 8]

Al tocar el ícono de menu hamburguesa de la esquina superior izquierda aparecen las opciones de la Figura 3.3. Las opciones de agregar cita médica, tarea y recordatorio aún no son funcionales, y otras como la de chat grupal aún no están completas. Estas opciones no son tan relevantes para este trabajo.

Al tocar el nombre de una persona que use la aplicación móvil se abre un chat para hablar directamente a través de la app. Este chat es bastante simple, como se observa en la Figura 3.4. La vista del chat con el chatbot debería ser básicamente igual que con los otros usuarios, pero al inicio de este trabajo no está funcional.

3.3. Back-end de Hermes

Como se observa en la Figura 3.1, el back-end de Hermes cuenta con tres componentes: el Task Manager, el chatbot y la bases de datos. La API que permite realizar todas las acciones desde la aplicación móvil y con la cual se conecta el chatbot para agregar tareas, consultar la lista de contactos y otras cosas está implementada en Tornado Web Server [17].

Figura 3.2: Pantalla principal de la aplicación móvil.

3.4. Comportamiento del chatbot legado

El chatbot legado, reportado en [7, 8], contacta a los usuarios cuando se necesita realizar una tarea en favor de un adulto mayor, según lo indicado por el administrador de tareas, usando potencialmente algún criterio para ello. Para determinar de manera más efectiva a qué usuario contactar, esta versión del sistema no saca provecho del comportamiento registrado de cada usuario, sino que crea grupos arbitrarios para empezar a contactar.

Por otra parte, en la implementación de este componente no hay una evaluación de la efectividad de las invitaciones que el sistema envía a los usuarios, para que estos realicen una tarea en favor de los adultos mayores. El sistema tampoco realiza cambios de estrategias en función de los resultados de invitaciones pasadas. Por lo tanto, existe una necesidad de mejorar la efectividad del sistema, generando un nuevo protocolo de comportamiento para el chatbot.

Este protocolo debe permitirle al sistema actuar de forma más inteligente a la hora de intentar asignar tareas a los usuarios. Para ello, el sistema debe tratar de asignarle las tareas correctas, a las personas correctas, y además contactarlas de forma adecuada en un momento potencialmente oportuno. En este sentido, el sistema también debe considerar la carga de trabajo ya asumida por estas personas, no solo para no sobrecargarlos, sino también para aumentar su efectividad en la asignación (potencial) de tareas, pues alguien que ya está sobrecargado probablemente no es un buen candidato para asumir nuevas tareas.

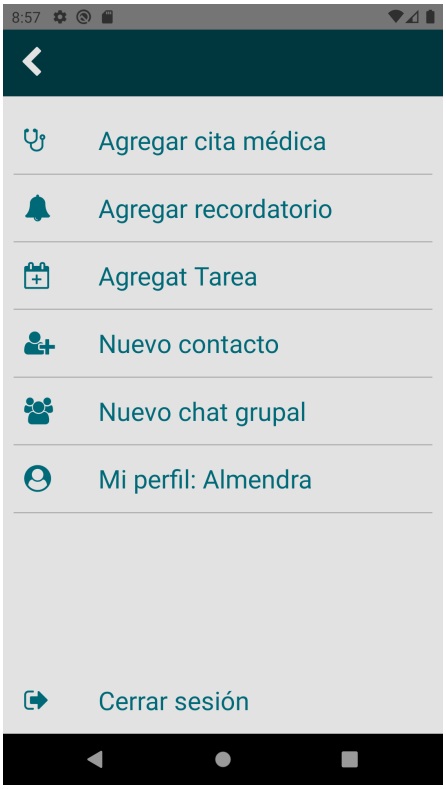


Figura 3.3: Menu hamburguesa.

Así, el sistema debe ser capaz de encontrar un buen balance en la asignación potencial de tareas a cuidadores. Para esto, primero es necesario idear un método de caracterización de usuarios, de manera que el sistema pueda analizar dicha caracterización y así decidir cómo proceder adecuadamente al momento de buscar un cuidador para apoyar la realización de una cierta actividad.

Dado todo lo anterior, y tal como se mencionó en la sección de objetivos de la memoria, el propósito de este trabajo es diseñar un nuevo comportamiento para el chatbot que apoya al sistema Hermes. Particularmente, uno que sea más efectivo en la asignación de tareas y que no limite el desarrollo de este asistente virtual de cara al futuro.

3.5. Procesos a apoyar por el Chatbot

Tal como se mencionó antes, el chatbot debe ser capaz de llevar a cabo los flujos para obtener información del usuario (Figura 3.5), agregar una actividad a la agenda (Figura 3.6) y asignar tareas a usuarios que podrían hacerse cargo (Figura 3.7). Estos flujos no estaban completamente funcionales en el sistema legado. A continuación se indican los flujos de conversación propuestos que debe implementar el chatbot.

Figura 3.4: Chat con otro usuario de la app.

3.5.1. Iniciar Sesión

Este flujo se lleva a cabo cuando un usuario se va a conectar al chatbot mediante la aplicación externa de mensajería. Permite asociar el chat con un usuario en la base de datos. El chatbot le pregunta al usuario por su correo y contraseña, y le consulta al backend para ver si es correcto y poder marcar como iniciada la sesión. En la Figura 3.5 se observa un diagrama de este flujo.

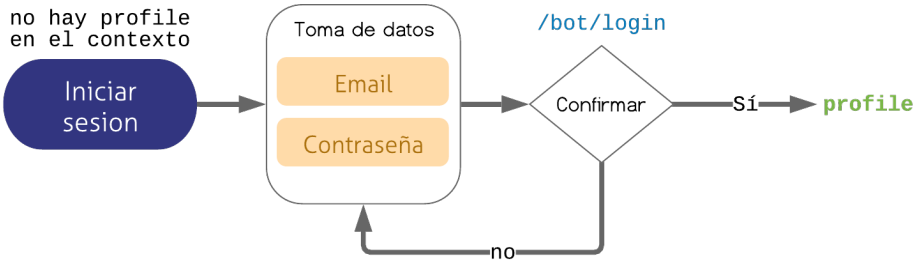


Figura 3.5: Flujo propuesto del chatbot para obtener la información del usuario.

3.5.2. Agregar actividad

Este flujo se lleva a cabo cuando un usuario desea agendar una actividad o recordatorio. El acceso a este flujo es solo mediante la app móvil, es decir, no está disponible en la aplicación externa de mensajería. Dependiendo de la intención detectada por el chatbot se decide qué tipo de actividad es la que se quiere agregar. Cada actividad tiene ligeras diferencias en el proceso de toma de datos y en los textos que responde el chatbot, así como en la acción que se ejecuta al terminar el flujo. El usuario indica qué tipo de actividad es la que quiere agendar, y a continuación el chatbot le hace una serie de preguntas para obtener datos de paciente, asunto, fecha y hora, etc. Al final si se confirman los datos se agrega la nueva actividad y se inicia el ciclo de asignación. En la Figura 3.6 se observa un diagrama de este flujo. En este diagrama se observan también los endpoints de la API que se llaman cuando se llegan a algunos nodos del flujo (/bot/activity, /bot/assignment, etc).

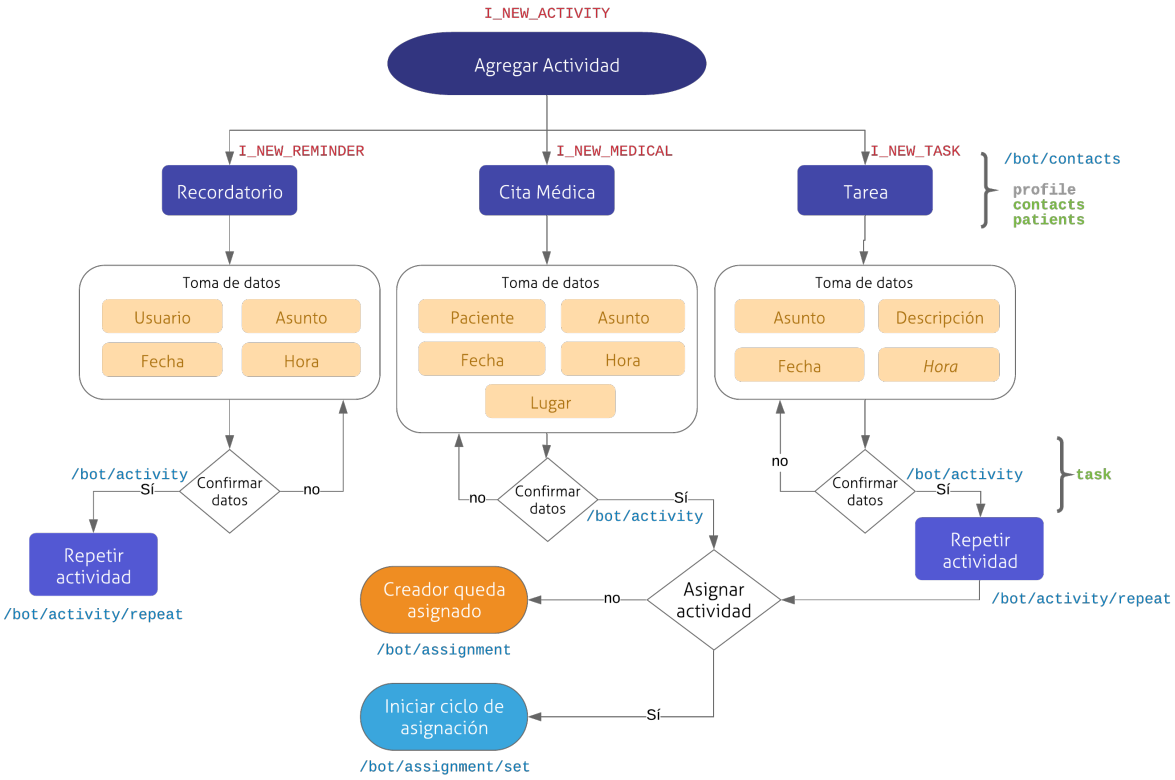


Figura 3.6: Flujo propuesto del chatbot para agregar una actividad.

3.5.3. Asignar actividad

Este flujo se lleva a cabo cuando se generó una actividad que necesita ser asingada a un usuario. El chatbot debe contactar usuarios para intentar adjudicarles la actividad. Se contacta a los usuarios por grupos y se actúa acorde a la respuesta de estos hasta que se logre asignar a un usuario a la actividad. En realidad este flujo es responsabilidad del backend, que le indica al chatbot a quién preguntarle si puede hacerse cargo de la tarea. Como se observa en la Figura 3.7, se intenta asignar por grupos, y si nadie en el primer grupo toma

la tarea pasa al siguiente grupo, en donde si de nuevo nadie lo toma se inicia la notificación de urgencia para intentarlo una vez más. Si nadie toma la tarea, el creador queda asignado.

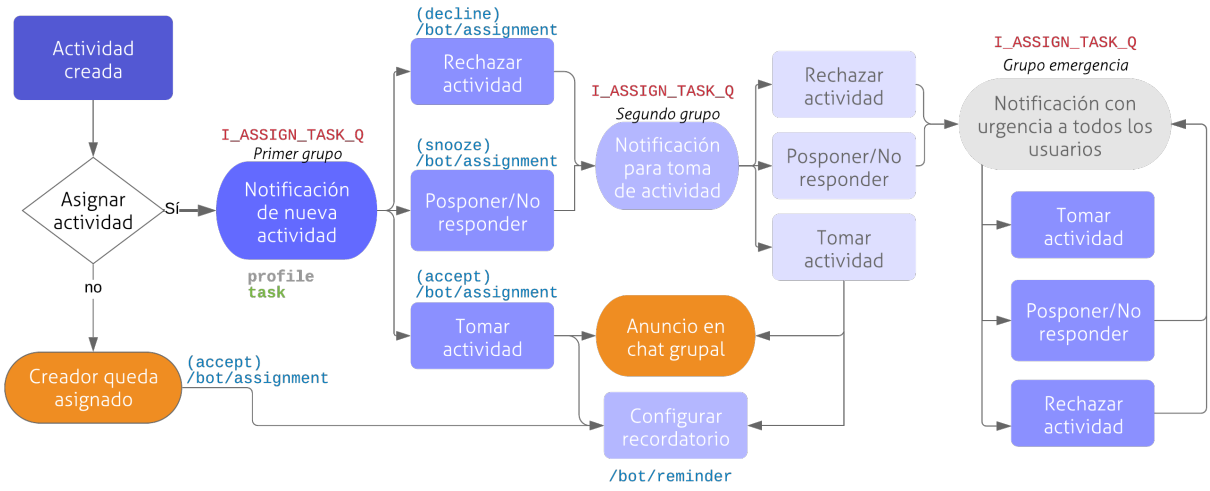


Figura 3.7: Flujo propuesto del chatbot para asignar una actividad a un usuario.

3.6. Dependencia de tecnología de terceros

La otra arista de este trabajo de memoria (adicional a la mejora de la efectividad del chatbot), es la implementación misma de un nuevo componente de este tipo. Como se mencionó antes, no se puede seguir usando el chatbot actual pues la infraestructura para el funcionamiento de éste fue proporcionada como un favor por parte de la empresa BotCenter. Además, era factible de utilizarla como parte de una prueba de concepto.

Debido a eso, se llegó a la conclusión de que es necesario reemplazar en la arquitectura de Hermes, todo lo relacionado al chatbot, generando una solución que pueda evolucionar de cara al futuro. El sistema actualmente implementado usaba algunos criterios no aleatorios para determinar el usuario a contactar, los cuales pueden servir como un acercamiento inicial a la nueva solución.

El desafío más grande yace ahora en rehacer el chatbot de manera que la aplicación siga funcionando igual que siempre, pero con un bot propio, el cual se pueda seguir usando y modificando en el futuro. Es decir, es necesario crear un chatbot nuevo, con una herramienta diferente, y una API para comunicarse con éste. El nuevo chatbot debería usar la misma especificación de endpoints de la API de BotCenter, para que el resto de la arquitectura siga funcionando sin necesidad de cambiar código fuente de Hermes.

Una vez completada esta tarea, se podrá estudiar cómo mejorar la efectividad del sistema, caracterizando a los usuarios en base a la información obtenida a través del chatbot. Elaborar un sistema inteligente, que defina la mejor forma de proceder cuando se le entregan los datos necesarios, puede ser muy complejo y escapa al alcance de este trabajo de memoria. Por este motivo, el objetivo será desarrollar un sistema que permita recopilar los datos y darles un

formato útil (aún por definir), de forma que se tenga todo lo necesario para el siguiente paso; es decir, determinar el nuevo comportamiento del chatbot de cara al usuario final.

3.7. Interacción con la API de BotCenter

La API de BotCenter se usa como se especifica en el Código 3.1. El endpoint recibe el id del bot, el mensaje que se le envía y datos contextuales. En el anexo se muestran ejemplos del mensaje que se envía (Código A.1), y de la respuesta que se recibe (Código A.2).

Código 3.1: Especificación de la API del chatbot

```
url = 'https://api.botcenter.io/api/v1/bot/run'

headers = {
  ApiKey: api_key
}

body = {
  bot_id:str,
  message:str,
  data:str(json)
}

response = request(url, {
  method:'POST',
  headers: headers,
  body: body
})
```

Capítulo 4

Concepción de la Solución

En esta sección se explica a grandes rasgos cómo se pretende resolver los problemas y qué es lo que se necesita hacer para implementar la solución.

4.1. Backend

La arquitectura básica del back-end propuesta se puede observar en la Figura 4.1. Allí se puede ver que cada vez que hay una comunicación relevante con el chatbot, se llama a la API del sistema de caracterización, para allí actualizar datos o realizar consultas.

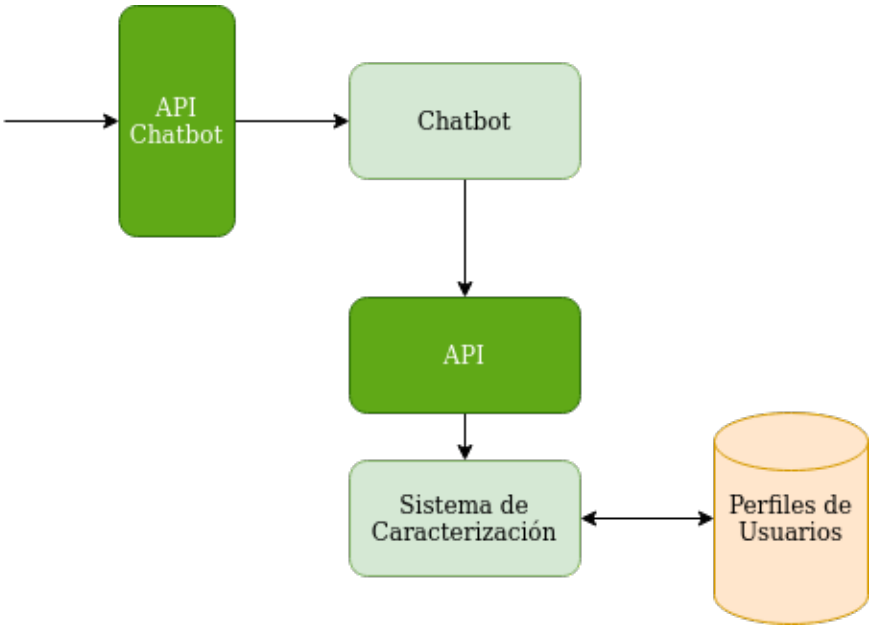


Figura 4.1: Arquitectura básica del back-end

El chatbot debe entonces interactuar con un sistema de caracterización. Este sistema debe incluir al menos las siguientes acciones:

- Crear un perfil de usuario para cada participante

- Actualizar un perfil usando datos de conversaciones
- Entregar estadísticas del perfil respecto a tiempos de respuesta y tipos de respuesta

Como el objetivo principal de este trabajo es mejorar la efectividad de Hermes y no cambiar demasiado su funcionamiento, la solución desarrollada busca evitar (dentro de lo posible) la modificación de cualquier cosa que no tuviera que ver con el funcionamiento mismo del chatbot y la identificación de candidatos para la asignación de tareas por parte de éste. Considerando eso, se ideó un nuevo servicio en el back-end de Hermes, que queda disponible a través de la API.

Pese a que inicialmente se pensó desarrollar una API completamente aparte para el sistema de caracterización, dado que esto se relaciona de manera estrecha con los demás componentes de la API se decide incluir el sistema de caracterización como un componente más en el back-end ya existente.

El nuevo servicio del back-end debe poder realizar la caracterización de usuarios (perfilamiento) de forma independiente de la operación de Hermes. De esta manera, lo único que se tiene que modificar del sistema actual, es la función de coordinación de usuarios del chatbot, para que dicha función sea realizada a través de servicios del nuevo sistema.

Finalmente, se decide utilizar Cloud Firestore[1] para guardar los datos principalmente por 2 motivos:

1. La API de Hermes ya utilizaba esta herramienta
2. La flexibilidad para almacenar los datos permite editar y mejorar la definición de los perfiles fácilmente en el futuro

Así, los perfiles de usuarios se piensan como una lista de eventos —donde cada evento es un intento de asignación de una tarea— y datos estadísticos obtenidos a partir de los datos agregados de los estos y recalculados con cada nuevo evento.

4.2. Nuevo Chatbot

Para rehacer el chatbot sin cambiar su funcionamiento y diseño, se decidió usar la herramienta Botpress, pues su forma de definir el comportamiento de un bot es similar a la que se usa en BotCenter. Botpress permite modelar las conversaciones en base a nodos, usando una interfaz gráfica con drag-and-drop, y con los conceptos de intenciones y entidades, al igual que el sistema de BotCenter.

El chatbot debe ser capaz de llevar a cabo los flujos para obtener información del usuario, agregar una actividad a la agenda, y asignar tareas a usuarios que podrían hacerse cargo. En la Sección 3.5 se describen en detalle estos procesos.

[1] https://firebase.google.com/docs/firestore

Botpress, al igual que el sistema de BotCenter, permite definir el comportamiento del bot en base a intenciones (intents) y entidades (entities). Las intenciones y entidades propuestas en el sistema legado son:

Intenciones

- I_NEW_ACTIVITY: Agendar actividad (cualquier tipo)
- I_NEW_REMINDER: Agendar recordatorio
- I_NEW_TASK: Agendar tarea
- I_NEW_MEDICAL: Agendar cita médica

Entidades

- taskDate: Fecha y hora

Definiendo estas intenciones y entidades, a continuación se debe entrenar el chatbot, entregándole ejemplos para cada una de las intenciones. Esto se hace en Botpress de la misma manera que en el sistema de BotCenter. Sin embargo, como se verá más adelante, se crearon algunas intenciones nuevas y finalmente no se usó taskDate.

Además, se necesita una base de datos en la cual se puedan guardar mensajes y datos de usuarios y que se pueda consultar. Para esto se puede usar PostgreSQL porque Botpress tiene soporte para conectarse directamente con una base de datos PostgreSQL.

4.3. Frontend

Teniendo un chatbot con el cual se puede interactuar mediante una API, es necesario también agregar las vistas y lógica a la aplicación móvil. Para esto es necesario crear una vista nueva basada en la vista del chat con otros usuarios y también un servicio para ser usado por esta que consulte la API del chatbot.

Además, para que no se borre la conversación cada vez que se cierra el chat es necesario tener una forma para consultar la base de datos de mensajes. Para esto, se decide hacer un micro servicio cuya única función es entregar historiales de mensajes según los parámetros que se le entreguen. El servicio del chatbot en la aplicación móvil puede usar este micro servicio para ir cargando los mensajes dinámicamente conforme se va haciendo scroll hacia arriba.

Si bien se pueden reutilizar casi todos los elementos visuales de la vista de chat, es necesario hacer algunas modificaciones tales como agregar botones para cuando el chatbot ofrece opciones y cambiar toda la lógica para cargar mensajes. La aplicación cuenta con un modelo de mensajes que fue pensado para el sistema legado, así que hay que transformar los datos que se envían y reciben del chatbot para adaptarlos a estos mensajes.

Capítulo 5

Diseño de la Solución

A continuación se explica en detalle la implementación de los sistemas mencionados desde el punto de vista del chatbot, el frontend y el backend.

5.1. Chatbot

Experimentando con Botpress se llegó a la conclusión de que tiene capacidades muy similares a BotCenter y, definiendo las intenciones (con sus datos de entrenamiento correspondientes), entidades y flujos, se puede emular el comportamiento deseado del chatbot previo, y hacer las mismas llamadas al backend en los casos correspondientes.

Definir los flujos del bot es en realidad un poco más fácil que en BotCenter, pues se puede usar una interfaz con drag-and-drop para llevar el flujo de la conversación, mientras que BotCenter utiliza un lenguaje similar a Lisp.

No obstante, no basta con usar la interfaz gráfica de Botpress. Para definir comportamientos un poco más complejos Botpress ofrece herramientas como actions y hooks, los cuales deben ser escritos usando el lenguaje Javascript.

Las actions se pueden adjuntar a los nodos de los flujos de conversación para realizar acciones tales como hacer peticiones HTTP (a la API de Hermes), guardar datos temporales, cambiar el flujo o el contexto de la conversación según condiciones más complejas, etc.

Los hooks pueden definirse para ser llamados cuando ocurre algún evento. Por ejemplo, para hacer algo antes o después de procesar el mensaje del usuario o redirigir la conversación a Telegram.

En las secciones siguientes se explican las intenciones y entidades del chatbot, los flujos de conversación de este y cómo se usan las acciones y hooks para manejar comportamientos más complejos en los flujos de conversación.

5.1.1. Intenciones y entidades

A continuación se explican las distintas intenciones creadas y sus datos de entrenamiento.

5.1.1.1. I_NEW_ACTIVITY

Esta intención indica que se desea agendar una actividad, ya sea cita médica, recordatorio o tarea. En la Figura 5.1 se observan los ejemplos entregados para el entrenamiento. Usando estos ejemplos se genera un modelo para reconocer esta intención.

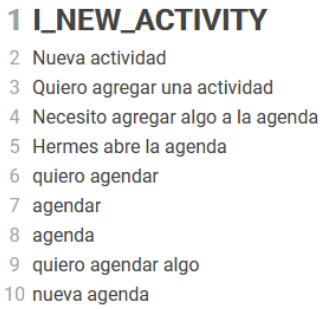


Figura 5.1: Definición de la intención de crear actividad, como se ve en la interfaz de Botpress.

5.1.1.2. I_NEW_MEDICAL

Esta intención indica que se desea agendar una cita médica. En la Figura 5.2 se observan los ejemplos entregados para el entrenamiento.

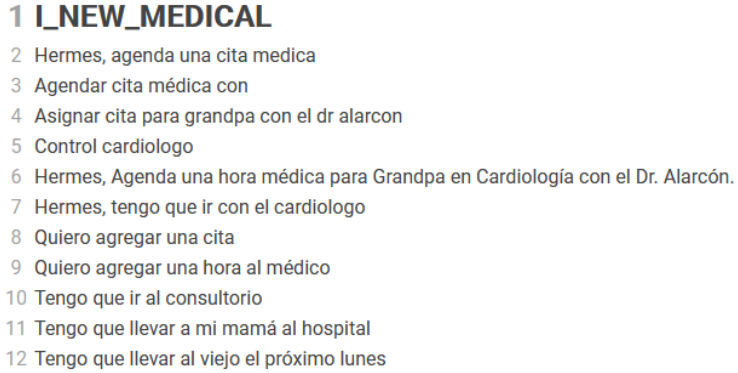


Figura 5.2: Definición de la intención de crear cita médica, como se ve en la interfaz de Botpress.

5.1.1.3. I_NEW_REMINDER

Esta intención indica que se desea agendar un recordatorio. En la Figura 5.3 se observan los ejemplos entregados para el entrenamiento.

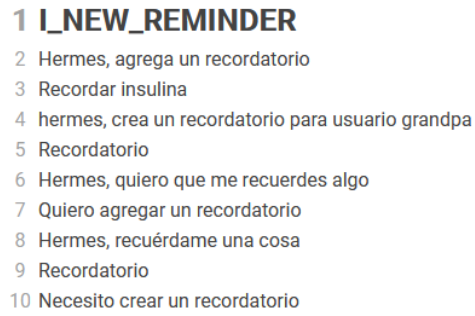


Figura 5.3: Definición de la intención de crear recordatorio, como se ve en la interfaz de Botpress.

5.1.1.4. I_NEW_TASK

Esta intención indica que se desea agendar una tarea. En la Figura 5.4 se observan los ejemplos entregados para el entrenamiento.

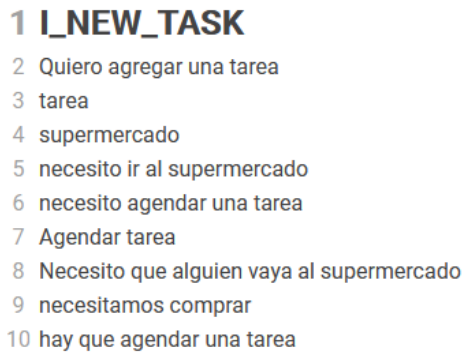


Figura 5.4: Definición de la intención de crear una tarea, como se ve en la interfaz de Botpress.

5.1.1.5. I_SET_TIME

Se usa cuando se necesita indicar una fecha y hora. Como se observa en la Figura 5.5, en vez de usar ejemplos de entrenamiento se usa un reconocimiento exacto del string “i_set_time” porque esta intención solo es declarada en el código cuando se le pregunta al usuario por el día y la hora. Se usa para rellenar un slot “time” que usa la entidad time de Botpress. Esta entidad propia de Botpress hizo innecesario crear la entidad taskDate definida en el sistema legado.

Figura 5.5: Definición de la intención de indicar fecha y hora, como se ve en la interfaz de Botpress. Rellena un slot llamado “time”.

5.1.1.6. I_SET_LOCATION

Se usa cuando se necesita indicar una ubicación. Rellena un slot “location”. En la Figura 5.6 se observan los ejemplos entregados para el entrenamiento.

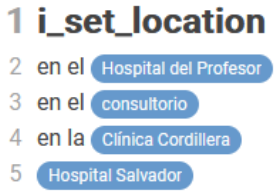


Figura 5.6: Definición de la intención de indicar ubicación, como se ve en la interfaz de Botpress. Rellena un slot llamado “location”.

5.1.2. Flujos

Botpress usa una interfaz de drag and drop para definir los flujos de conversación principales. A continuación se detallan los flujos implementados.

5.1.2.1. Main

El flujo Main es el flujo inicial a donde llegan las conversaciones nuevas. Siempre se entra al nodo “entry” y se ejecuta la acción que decide qué hacer a continuación, dependiendo de si debe redirigir al flujo de Login, al nodo New_activity.

En este flujo también se define el nodo New_activity, que espera recibir la intención correspondiente a un tipo de actividad o la intención general I_NEW_ACTIVITY. Si el caso es este último entonces pregunta directamente el tipo mediante el nodo de alternativas. Si se reconoce la intención de agendar alguno de los tipos de tareas se redirige al flujo correspondiente. Este flujo se observa en la Figura 5.7.

5.1.2.2. Cita-medica

Cita-medica es el flujo al que se llega cuando la intención es agendar una cita médica. Como se observa en la Figura 5.8, es bastante más complejo que el flujo Main por la cantidad de nodos, pero en realidad es solo una secuencia de preguntas y cuando se llega al final se

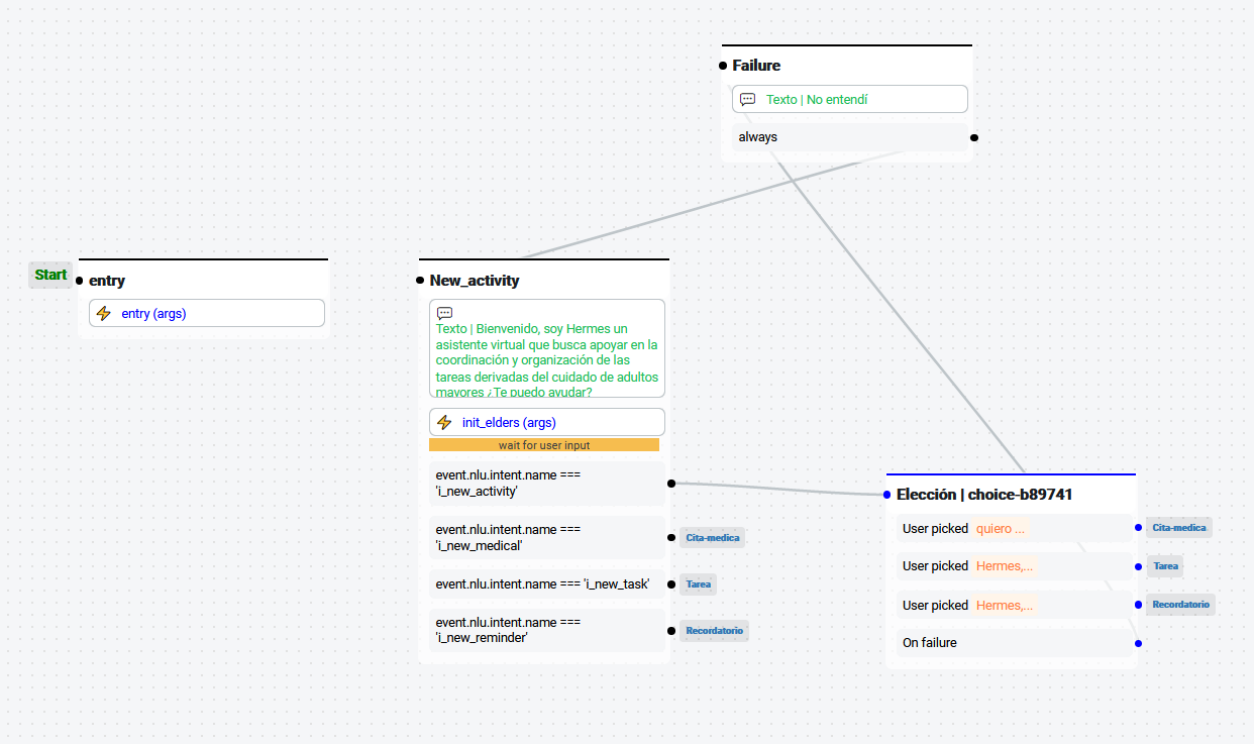


Figura 5.7: Interfaz de Botpress del flujo principal del chatbot, donde caen conversaciones nuevas.

pregunta si se desea cambiar un dato.

Lo primero que se hace es seleccionar al paciente. Si hay solo un adulto mayor que requiere cuidados en el grupo familiar se selecciona automáticamente ese sin necesidad de preguntar al usuario. Además, el usuario puede elegir al paciente antes de que el chatbot pregunte si por ejemplo en lugar de decir algo como “quiero agendar una cita médica” dice “quiero agendar una cita médica para Jaime”. Esto guarda en una variable temporal al paciente.

Posteriormente, se pregunta por el asunto, el día y la hora y el lugar. Gracias a la entidad “time” de Botpress, el chatbot es inteligente para reconocer el día y la hora, de manera que si por ejemplo hoy es Miércoles y se quiere agendar una cita médica para el Lunes de la próxima semana a las 10 de la mañana se puede escribir “el próximo Lunes a las 10 de la mañana” en vez de escribir con algún formato definido. Además, como se definió la intención I_SET_LOCATION usando un slot para el lugar, si el usuario escribe algo como “en el Hospital Salvador” el lugar queda guardado solamente como “Hospital Salvador”.

Finalmente, se pide confirmar si está todo correcto y, en caso de que no lo esté, se pueden hacer modificaciones dando opciones y volviendo a preguntar según lo que se elija.

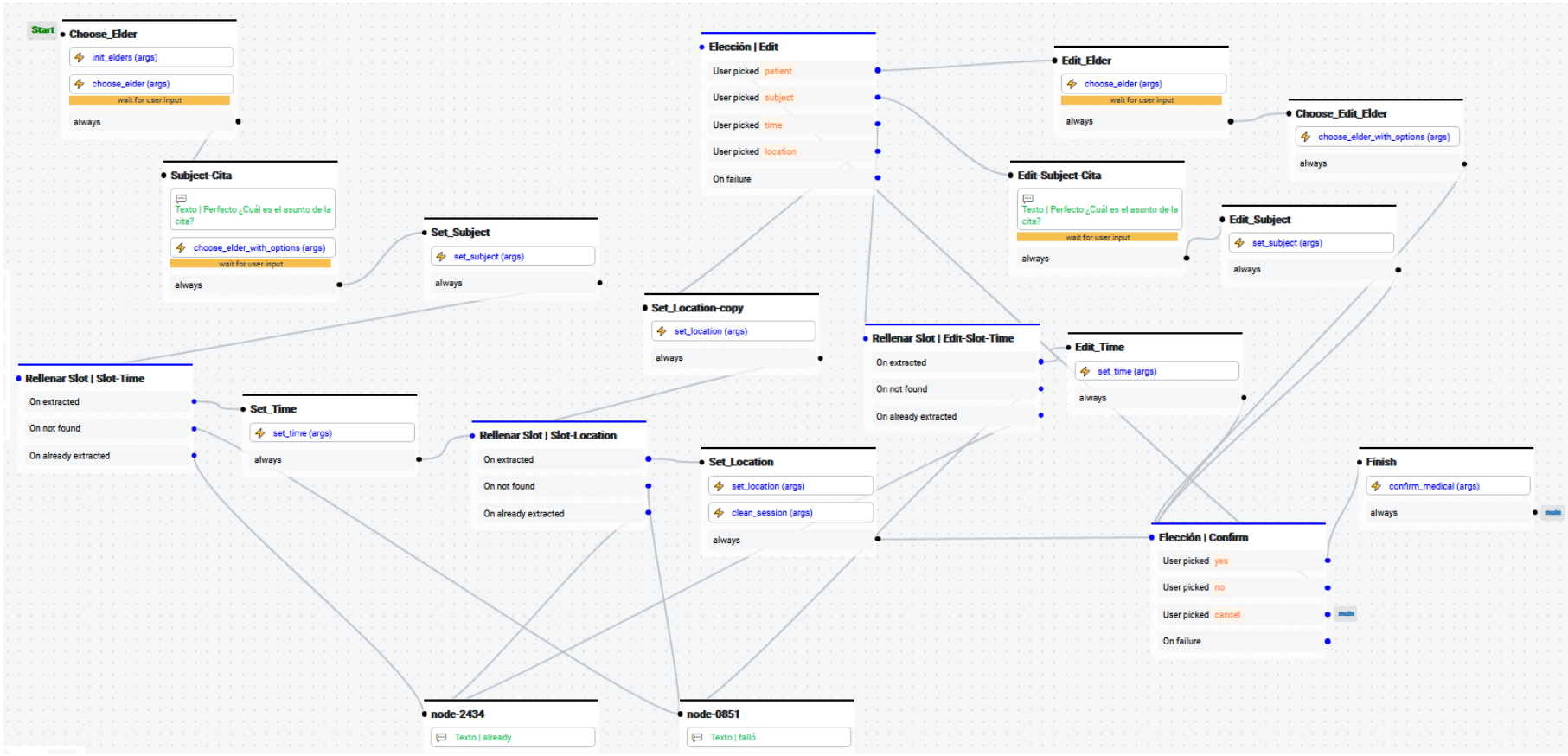


Figura 5.8: Interfaz de Botpress del flujo de creación de cita médica.

5.1.2.3. Recordatorio

El flujo Recordatorio es el flujo al que se llega cuando la intención es agregar un recordatorio. Como se observa en la Figura 5.9, es similar al flujo para agendar cita médica aunque un poco menos complejo.

Lo primero que se hace es seleccionar para quién es el recordatorio, de manera muy similar a como se selecciona un paciente en el flujo de agendar cita médica. Luego se indica el asunto y el día y la hora, para nuevamente llegar a la confirmación, en donde se puede elegir editar un dato.

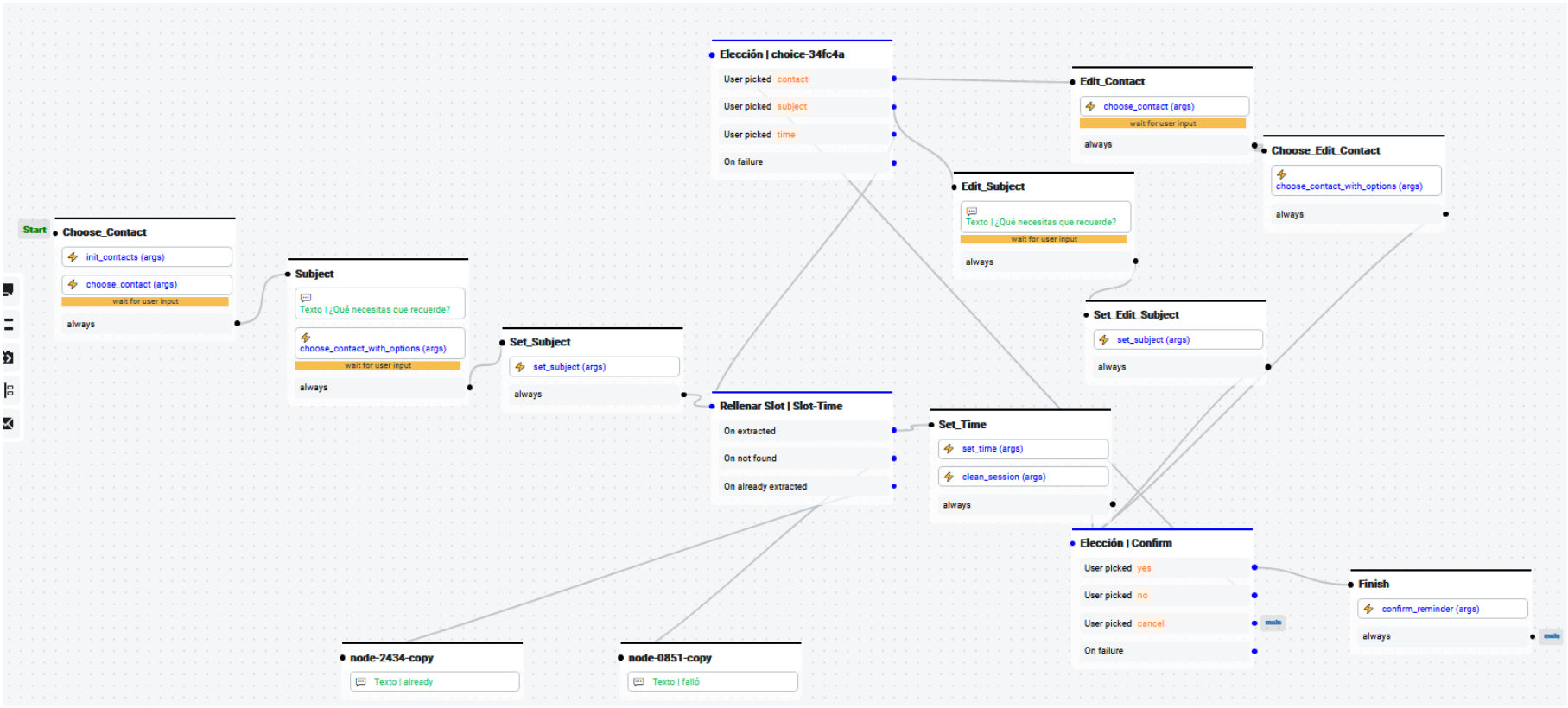


Figura 5.9: Interfaz de Botpress del flujo de creación de recordatorio.

5.1.2.4. Tarea

El flujo Tarea es el flujo al que se llega cuando la intención es agregar una tarea. Como se observa en la Figura 5.10, es similar a los flujos anteriores. Se indica secuencialmente asunto, descripción y fecha y hora. Al final se puede editar o confirmar.

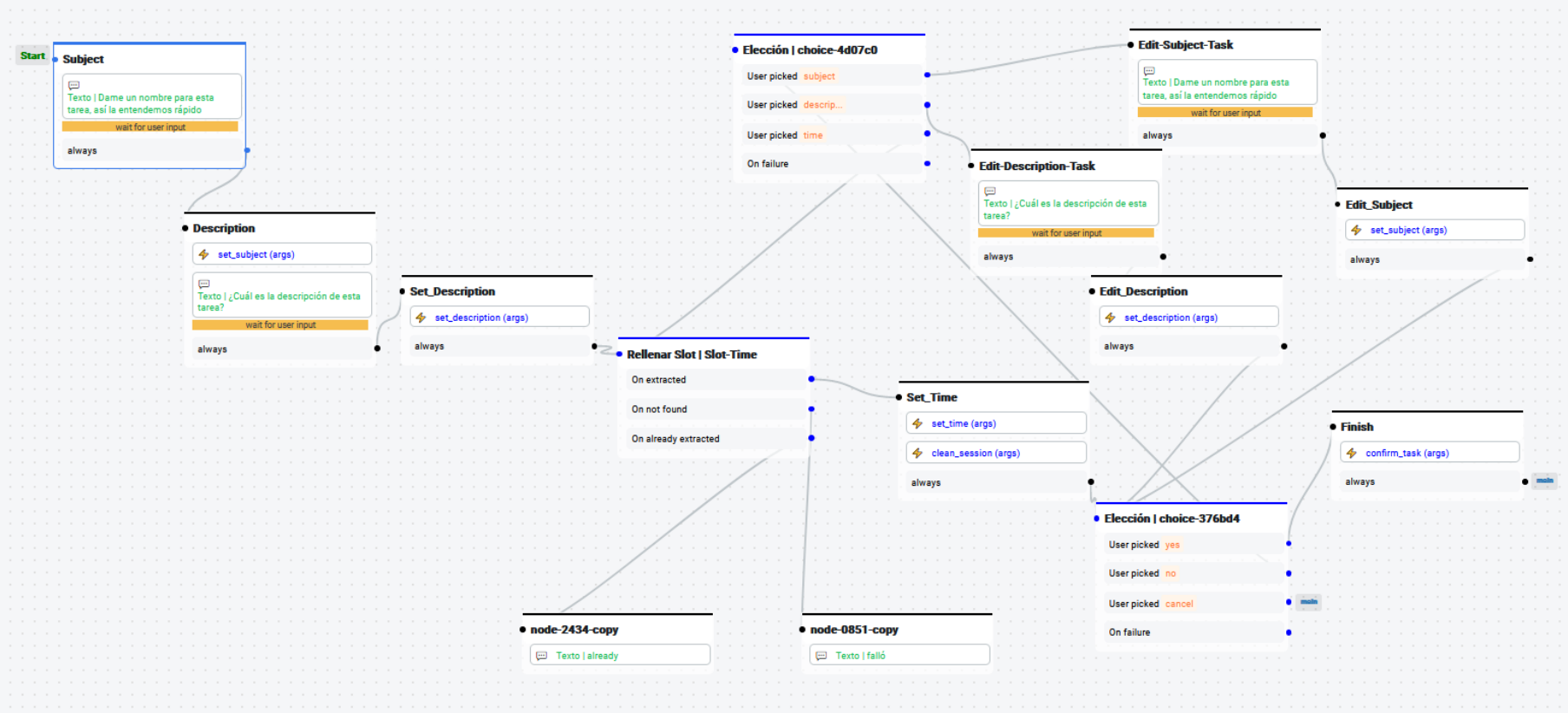


Figura 5.10: Interfaz de Botpress del flujo de creación de tarea.

5.1.2.5. Login

Cuando se inicia una conversación con el chatbot a través de la aplicación Telegram, el nodo inicial del flujo principal primero redirige al flujo Login, en donde se chequea si en ese chat se ha iniciado sesión con algún usuario. Si no se ha iniciado sesión entonces se ofrece la opción para hacerlo, en donde el usuario tiene que indicar correo y contraseña. Si el inicio de sesión en el backend de Hermes es exitoso se redirige al nodo Success, en donde posteriormente se redirige mediante al flujo y nodo que corresponda definido al inicio de la conversación y almacenado en una variable temporal. Si el usuario ya había iniciado sesión entonces se redirige directamente al nodo Success sin necesidad de volver a indicar correo y contraseña. Este flujo se puede ver en la Figura 5.11.

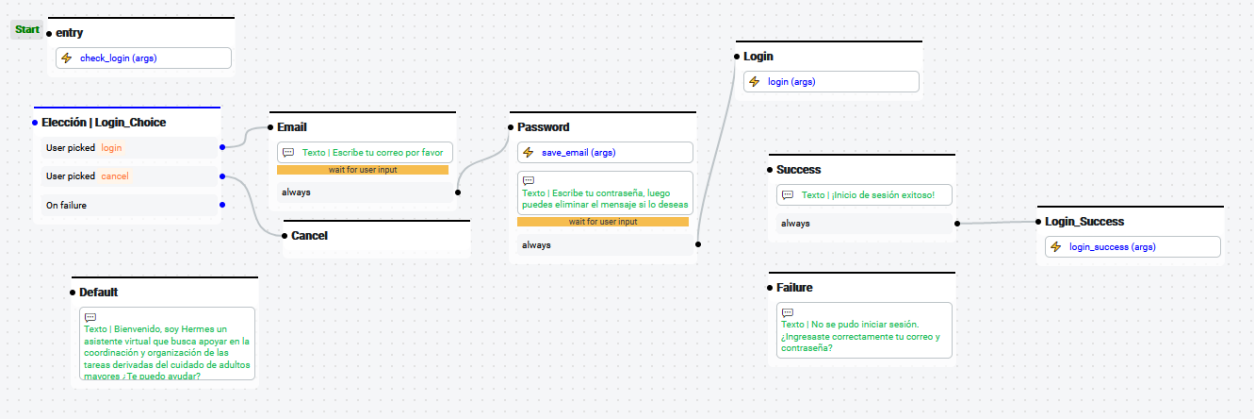


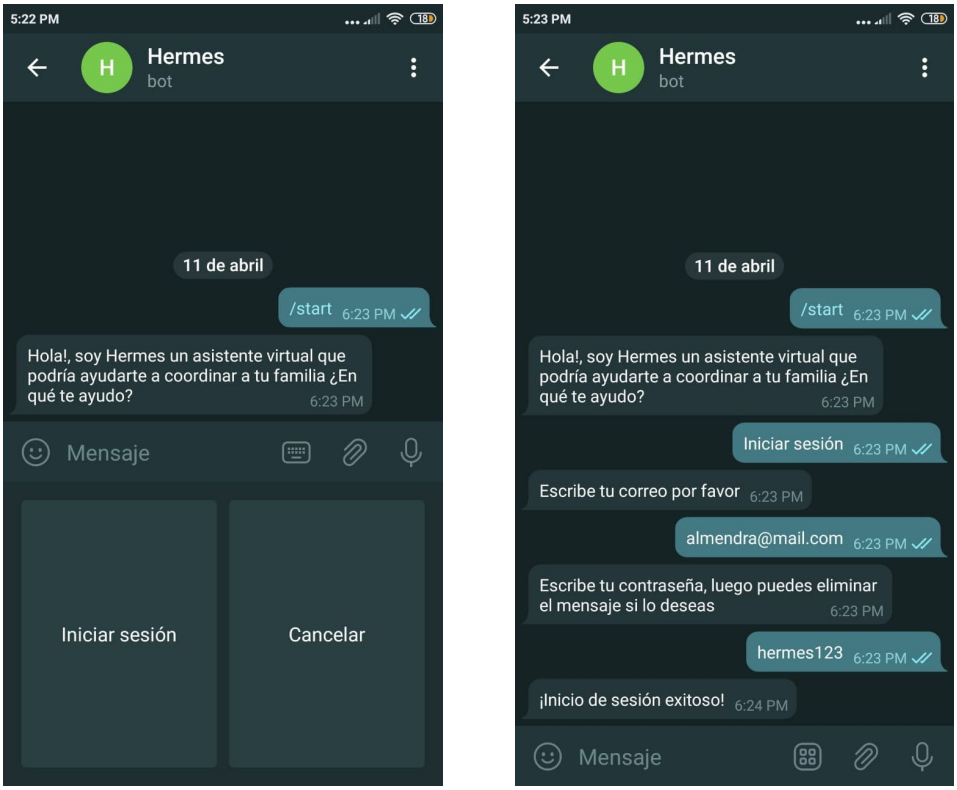
Figura 5.11: Interfaz de Botpress del flujo de inicio de sesión.

5.1.2.6. Assign

El flujo Assign es el que se usa para preguntar al usuario si desea tomar una tarea. Si se quiere contactar a un usuario que usa Telegram, primero este debe haber iniciado sesión con el flujo que se muestra en la Figura 5.12. Ahí se observa que, al iniciar una conversación con el bot por primera vez, este ofrece un botón para iniciar sesión (a), y luego pregunta por correo y contraseña (b). Si el inicio de sesión es exitoso se le informa al usuario y a partir de ese momento el chatbot reconocerá las conversaciones desde Telegram con ese usuario.

El flujo Assign se activa mediante un texto especial que simula ser un usuario y le entrega los datos necesarios al bot. Usa acciones para obtener los datos de la tarea y actualiza el historial del usuario según su respuesta. Tiene nodos desconectados porque parte de la lógica se define en el código de las acciones, en vez de directamente en la interfaz gráfica del flujo. Este flujo se puede observar en la Figura 5.13.

Lo que ve el usuario cuando interactúa con este flujo es lo siguiente: recibe un mensaje que le informa de la actividad que se requiere agendar y le ofrece las opciones "Sí", "Si nadie más puedez "No". Una vez que el usuario selecciona una opción se ejecutan las acciones correspondientes. Este flujo se puede ver en la Figura 5.14.



(a) Inicio

(b) Continuación

Figura 5.12: Flujo de inicio de sesión desde Telegram.

5.1.3. Acciones y Hooks

Las acciones y hooks de Botpress son las herramientas que permiten diseñar comportamientos más complejos tales como redirecciones a nodos y flujos con condiciones complejas, interacción con APIs y almacenamiento de datos utilizando el lenguaje de programación Javascript.

Las acciones pueden ejecutarse por ejemplo al ingresar a un nodo de conversación, y se le pueden entregar parámetros. Un ejemplo muy simple sería la acción que se ejecuta al final del flujo de Login si el inicio de sesión fue exitoso, en donde solamente se redirige al flujo y nodo definido al inicio de la conversación en temp.targetFlow y temp.targetNode. Esta acción se puede ver en el Código 5.1. En el anexo se pueden ver ejemplos un poco más complejos de acciones.

Los hooks se ejecutan cuando ocurren ciertos eventos, como antes de procesar un mensaje o después de responder. En este caso, se usa un hook con "Before Incoming Middleware" para revisar si el mensaje recibido tiene el identificador especial que indica que hay que continuar la conversación en Telegram. Si ese es el caso entonces se modifican algunos parámetros del evento para que parezca que el mensaje se recibió a través de ese canal. El código de este hook se encuentra en el anexo, en el Código B.3.

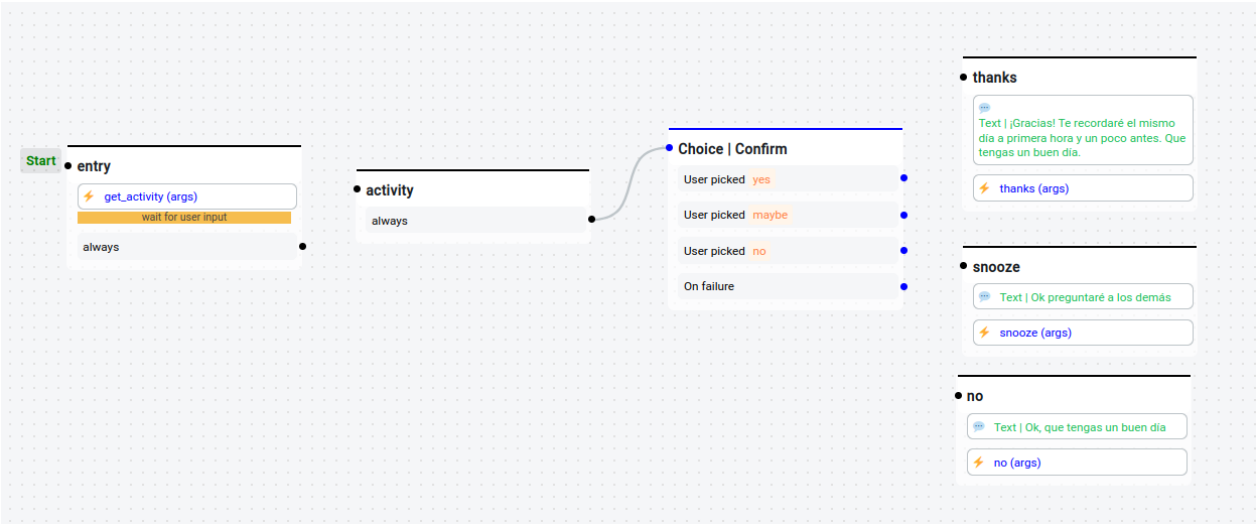


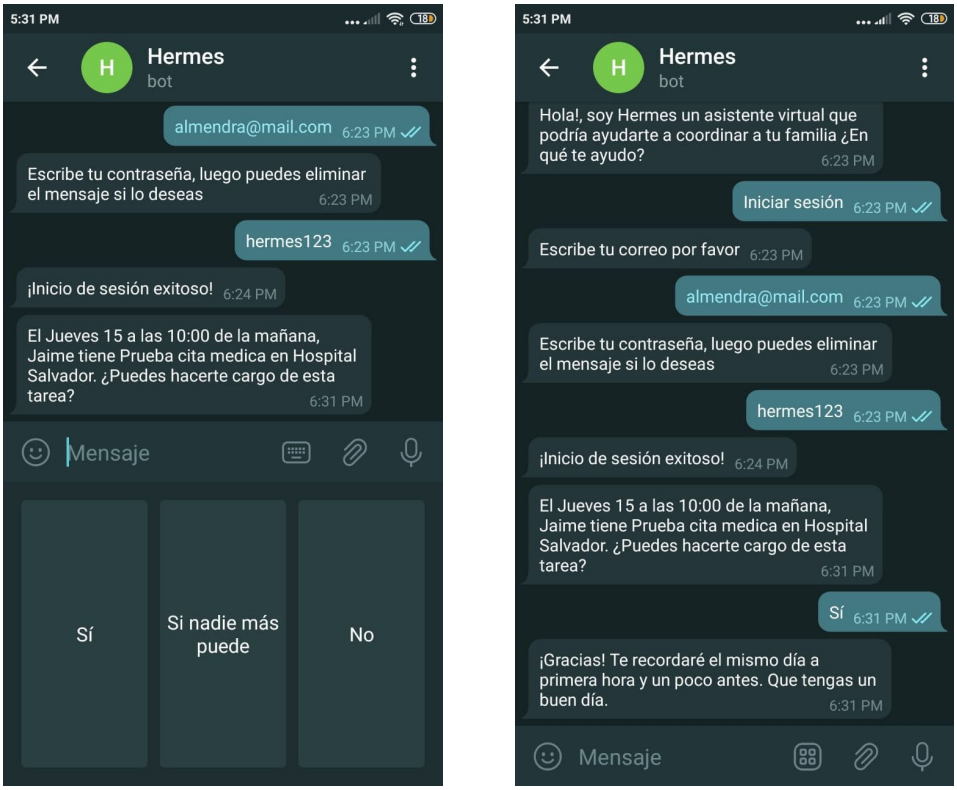
Figura 5.13: Interfaz de Botpress del flujo de asignación.

Código 5.1: login_success. Acción que se ejecuta al final del flujo de Login.

```
function action(
  bp: typeof sdk,
  event: sdk.IO.IncomingEvent,
  args: any,
  { user, temp, session } = event.state) {
  /**
   * Action to run when login is succesful
   * @title Login success
   * @category Custom
   * @author Pedro Belmonte
   */
  const myAction = async () => {
    const sessionId = bp.dialog.createId(event)
    await bp.dialog.jumpTo(sessionId, event, temp.targetFlow, temp.targetNode)
  }
  return myAction()
}
```

5.2. Frontend

A continuación se explica el trabajo realizado en cuanto a frontend, en donde primero se hizo una prueba de concepto para probar la comunicación con la API del chatbot y posteriormente se implementó en la aplicación móvil.



(a) Inicio

(b) Continuación

Figura 5.14: Asignación de una tarea desde Telegram.

5.2.1. Prueba de concepto

Teniendo el chatbot disponible, es necesario probarlo y conectarlo con la aplicación móvil. Dado que inicialmente no se tenía acceso al código de la aplicación se decidió hacer una prueba de concepto del uso de la API del chatbot mediante una pequeña aplicación web desarrollada en React [18]. Se escogió React principalmente porque se sabía de antemano que la aplicación móvil está implementada en React Native [19] en donde, si bien hay diferencias con React, la lógica para comunicarse con una API y mostrar la información en una vista es prácticamente igual, pero es un poco más rápido crear una prueba de concepto en React que hacerlo directamente en React Native.

La única vista de esta pequeña aplicación web se puede observar en la Figura 5.15. En este ejemplo en particular el flujo de la conversación fue el siguiente: Usuario dijo que quería agendar, pero no especificó el tipo de agenda así que el bot le ofreció las opciones mediante botones. El usuario hizo click en la opción de Cita médica por lo cual el bot procede a preguntar asunto, fecha, hora y lugar. Al final, el bot pregunta si está todo correcto y ofrece confirmar, cambiar algo o cancelar mediante botones. El usuario hace click en el botón para cambiar un dato y luego cuando el bot le pregunta cuál el usuario selecciona fecha y hora. Luego de que el usuario informa la nueva fecha y hora se vuelve al punto de confirmación.

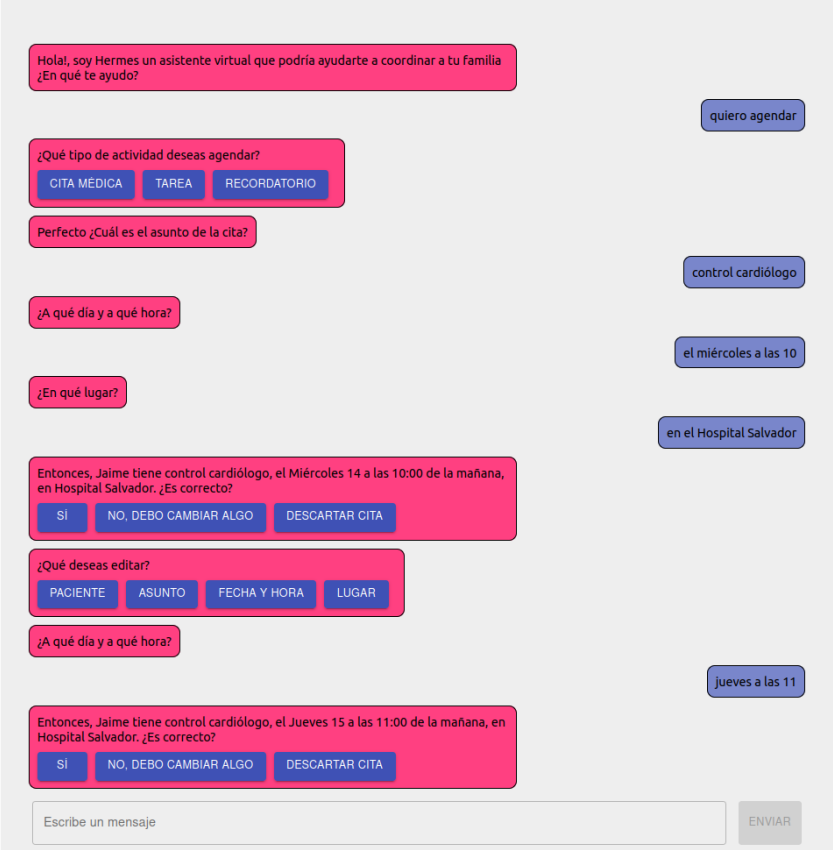


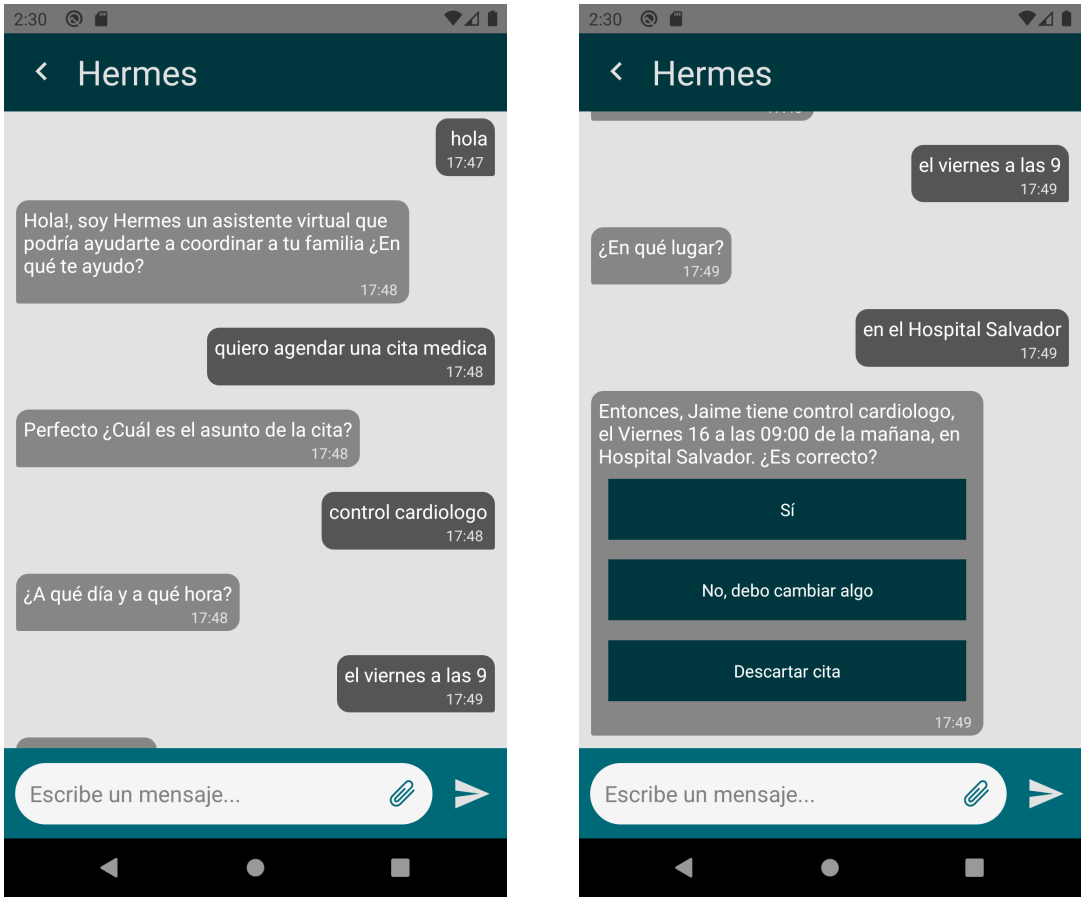
Figura 5.15: Prueba de concepto para probar la API del chatbot.

5.2.2. Aplicación móvil

La lógica de la interacción con la API implementada en la prueba de concepto se reutiliza en la aplicación móvil, pero también es necesario adaptarse a la arquitectura de esta. Para esto se implementa primero un servicio para realizar peticiones HTTP a cualquier API usando Axios [20].

A continuación, se implementa un servicio para conectarse a la API del chatbot y el micro servicio de historial de mensajes. Este usa el servicio de Axios en donde se definen las distintas funciones para ser usadas por la nueva vista, tales como el formateo de los mensajes para que cumplan con el modelo definido previamente en la aplicación, el envío y recepción de mensajes con la API del chatbot y cargar los mensajes antiguos.

Teniendo esto, se diseña una nueva vista muy similar a la vista de los otros chats pero con una lógica diferente usando el servicio antes mencionado. Además, es necesario adaptar el modelo de mensajes de la aplicación para dar soporte a los botones cuando el chatbot ofrece opciones. Esta vista se observa en la Figura 5.16. En la imagen se observa el mismo flujo que en la Figura 5.15 pero en la aplicación móvil, hasta el punto de confirmación, con la diferencia de que se indicó en el primer mensaje la intención de una cita médica, por lo cual el bot no preguntó qué se desea agendar.



(a) Inicio (b) Continuación

Figura 5.16: Vista del chatbot en la aplicación móvil.

5.3. Backend

Las siguientes secciones explican el trabajo realizado en backend, en donde se incluye el micro servicio implementado para obtener el historial de mensajes para poder mostrarlos en la aplicación y el sistema de caracterización de usuarios.

5.3.1. Historial de mensajes

Como se mencionó en la Sección 4.3, se implementa un micro servicio para consultar la base de datos de mensajes con el chatbot. Este servicio se implementa en Flask [21] porque es útil para hacer aplicaciones web extremadamente pequeñas de manera simple y rápida. Solo se define un endpoint, cuya ruta es /api/events/<user>/<offset> y entrega 20 mensajes del usuario con el offset indicado (esto es, si el offset es 0 entrega los últimos 20 mensajes, y si el offset es 20 entrega los 20 mensajes anteriores a esos). Este endpoint simplemente hace una consulta SQL a la base de datos del bot.

Para cada mensaje, el endpoint del micro servicio entrega bastante información como por ejemplo el texto, el tipo de mensaje, el canal, si fue enviado por el bot o por el usuario, etc. Para el caso de la aplicación móvil se usan algunos de estos datos para poder dar formato al

mensaje según el modelo definido previamente en la aplicación.

5.3.2. Sistema de Caracterización de Usuarios

Como se mencionó anteriormente, el sistema de caracterización queda dentro de la API previamente desarrollada en Tornado Web Server. Además, los datos quedan guardados en Cloud Firestore.

Para mantener el orden del código, se sigue el mismo formato que para los demás servicios. Esto significa agregar 3 archivos nuevos:

- **firebase_user_history.py:** Servicio para conectar con la colección en Cloud Firestore. Tiene setters para crear, modificar y borrar perfiles de usuarios y un getter para obtener la información.
- **db_user_history.py:** Clase para definir los objetos que representan los perfiles y permite crear, modificar y eliminar perfiles, agregar eventos, obtener información del perfil, etc. Se conecta con el servicio de **firebase_user_history.py**.
- **user_history.py**: Handler para recibir peticiones HTTP. Se conecta con la clase definida en **db_user_history.py**.

Con esto, el chatbot debe comunicarse con los endpoints definidos en user_history.py para crear eventos cuando se contacta a un usuario, y para actualizarlos cuando el usuario responde. Los métodos HTTP disponibles son GET, POST y PUT.

5.3.2.1. POST

El método POST se usa para agregar eventos (incompletos) al historial de un usuario. El chatbot lo usa cuando inicia el contacto con el usuario, y guarda datos de la actividad y el timestamp del momento en que se inició la conversación. También inicializa el momento de la respuesta (response_date) como nulo y el valor accepted como falso. En el Código 5.2 se ve un ejemplo de los datos que se envían en este caso.

Código 5.2: Ejemplo body para hacer POST al endpoint /bot/history

```
{
  "user": "1221600",
  "event": {
    "activity_id": "8tdlMjTYgAzAhEpy9GOy",
    "activity_subject": "Prueba Tarea",
    "type": "task",
    "task_date": "2021-04-16T17:00:00-04:00",
    "first_contact_date": "2021-04-11T19:24:09.497Z",
    "response_date": null,
```

```
    "accepted": false,
    "user": "1221600"
    }
}
```

5.3.2.2. PUT

El método PUT se usa para modificar un evento existente usando el id de la actividad como identificador del evento. El chatbot lo usa cuando el usuario responde si es que va a tomar la tarea, y cambia el valor de accepted si corresponde y el de response_date. Además le agrega el valor time_elapsed, que indica en segundos cuánto tiempo pasó entre que se inició el contacto y que el usuario respondió. En el Código 5.3 se ve un ejemplo de los datos que se envían en este caso.

Código 5.3: Ejemplo body para hacer PUT al endpoint /bot/history

```
{
  "activity_id": "8tdlMjTYgAzAhEpy9GOy",
  "activity_subject": "Prueba Tarea",
  "type": "task",
  "task_date": "2021-04-16T17:00:00-04:00",
  "first_contact_date": "2021-04-11T19:24:09.497Z",
  "response_date": "2021-04-11T19:30:51.299Z",
  "time_elapsed": 401.802
  "accepted": true,
  "user": "1221600"
}
```

5.3.2.3. GET

El método GET se usa para obtener los datos de un usuario. Entrega una lista con todos los eventos del usuario, en donde cada evento tiene información del tipo de tarea, fechas, cuánto tardó en responder, etc. Además, como se observa en el Código 5.4, también se incluyen datos de porcentaje de aceptación de tareas y el promedio y la mediana del tiempo de respuesta de este usuario.

Código 5.4: Ejemplo respuesta al usar el método GET en el endpoint /bot/history?user=1221600. Se omite el arreglo de eventos para abreviar, pero el valor de cada uno de los eventos es como lo que se ve en los Códigos 5.2 y 5.3.

```
{
  "user_id": "1221600",
  "events": [...],
  "acceptance_rate": 0.5,
```

```
  "response_times_statistics": {
    "mean": 335.69366666666673,
    "median": 401.802
  }
}
```

Capítulo 6

Análisis de la Solución

Dado que la naturaleza de este trabajo estaba más enfocada en volver a implementar funcionalidades y obtener más datos útiles de un sistema cuya usabilidad ya había sido evaluada con usuarios reales mediante prototipos en papel [7, 8], la evaluación va también más enfocada en reflexionar si todo lo implementado es realmente útil para mejorar el sistema de Hermes.

6.1. Chatbot

Al inicio de este trabajo existía un chatbot desarrollado en la plataforma de BotCenter, pues se permitió usar esta plataforma de manera gratuita como un favor para Andrea Benavides. Cuando se comienza con el trabajo, este chatbot no está funcionando correctamente así que no se puede probar bien su funcionamiento deseado. Sin embargo, gracias a la documentación de Andrea se logra entender lo que se necesita para desarrollar el chatbot nuevo.

Este nuevo chatbot implementado usando Botpress tiene algunas ventajas interesantes con respecto al otro. Estas ventajas incluyen:

- La interfaz gráfica para diseñar flujos de conversación de Botpress (a diferencia del lenguaje similar a Lisp de BotCenter) que permite una curva de aprendizaje algo menos pronunciada si otra persona comienza a trabajar en el chatbot y permite entender e imaginar más fácilmente los flujos de conversación.
- Reconocimiento inteligente de fechas y horas que permite mayor flexibilidad en los formatos.
- Al usar el lenguaje de programación Javascript se simplifican las llamadas a la API de Hermes.
- Botpress incluye soporte para Telegram y otros canales de comunicación de una manera bastante simple, lo que permite agregar más canales en el futuro sin problema.

Dicho esto, el sistema originalmente se pensó para conectarse con el sistema de BotCenter, por lo cual hubo que hacer adaptaciones de interfaces en ambos sentidos que se podrían mejorar con un poco más de tiempo. Además, el código que inicia el ciclo de asignación no se

ha adaptado todavía para usar este chatbot nuevo, razón por la cual las pruebas de asignación de tareas para el sistema de caracterización se hicieron haciendo llamadas manuales al endpoint de la API del chatbot. Finalmente, algunos aspectos de los flujos de conversación pueden trabajarse más para mejorar la experiencia de usuario, manejando más casos borde o mejorando el manejo de estados para que el usuario no reciba respuestas extrañas o se pierda el flujo.

En resumen, el trabajo realizado en el chatbot es una mejora en cuanto a lo que se tenía antes de iniciar este trabajo y es una buena base para continuar avanzando, pero no se puede considerar completamente listo.

6.2. Sistema de Caracterización de Usuarios

Como se vio en la Sección 5.3.2, el sistema de caracterización guarda datos de la interacción entre el chatbot y un usuario cuando trata de asignarle una tarea. Si bien estos datos ya permitirían hacer un poco más inteligente el ciclo de asignación de tareas (por ejemplo priorizando ciertas horas para iniciar el contacto, tomando en consideración el tiempo promedio que tarda en responder un usuario en particular o considerando el porcentaje de aceptación de tareas de los usuarios), la verdad es que estos datos aún siguen siendo bastante básicos. Esto es en parte porque, como se vio en la Sección 5.1.2, el flujo de conversación para asignar una tarea a un usuario es extremadamente simple: se informa de la tarea y se pregunta si la quiere tomar y el usuario responde "Sí", "Si nadie más puede" o "No". Sin embargo, sí se podrían agregar algunas cosas más al perfil como por ejemplo precalcular horarios en donde es más probable que el usuario responda que sí.

Dicho esto, el sistema de caracterización implementado cumple la función básica de mantener un historial de interacciones, y es muy flexible de manera que si se hacen modificaciones al flujo o se idea otro parámetro que se podría guardar es bastante simple modificarlo para que soporte estos cambios.

Finalmente, al inicio se quería hacer también un sistema básico de asignación que propusiera a quién debería contactar el chatbot dada una tarea en particular, usando este sistema de asignación. Sin embargo, esto no se alcanzó a implementar.

Se concluye que, similar al chatbot, el sistema de asignación es una buena base para avanzar, pero no cumple lo que se hubiera querido al comenzar el trabajo.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusión

Para los adultos mayores, envejecer en el hogar puede tener aspectos muy positivos como son el ambiente familiar y conservar cierta independencia. Sin embargo, esto también conlleva una carga para sus familiares más cercanos que a menudo son los que ejercen el rol de cuidadores informales. El trabajo de los cuidadores informales tiende a ser invisible y carecer de apoyo formal.

Es posible aplicar tecnología para apoyar el cuidado de adultos mayores, para poder organizar de manera más eficiente y efectiva el trabajo de los cuidadores informales. Si bien las tecnologías de comunicación como aplicaciones de mensajería y videollamadas pueden ser muy útiles para apoyar el cuidado y la comunicación con adultos mayores, es común que las personas de la tercera edad sean reacios a adoptar estas tecnologías. No obstante, la tecnología sigue siendo una herramienta poderosa para coordinar de mejor manera el trabajo de los cuidadores informales.

En este contexto surge la idea de Hermes, una herramienta que pretende facilitar la coordinación de los cuidadores informales integrando un calendario, un sistema de comunicación y un asistente virtual. Este sistema fue desarrollado como parte de un trabajo de título anterior, pero no está completo. Considerando el sistema desarrollado, surge la idea de mejorar la inteligencia del asistente virtual, para mejorar su efectividad.

Este trabajo pretende apoyar esa mejora, creando un chatbot nuevo y un sistema de caracterización basado en el historial de interacciones entre el chatbot y los usuarios cuando se trata de repartir tareas entre los cuidadores informales.

Para esto se desarrolla un chatbot nuevo utilizando Botpress, que entrega muchas herramientas para desarrollo de chatbots y es open-source. Este chatbot implementa los flujos que se habían propuesto anteriormente en el desarrollo de Hermes, y se conecta con la aplicación móvil existente mediante una nueva vista implementada en esta.

Además, se implementa un sistema de caracterización de usuarios, que guarda información de cada vez que el chatbot le pregunta a un usuario si se va a encargar de realizar una tarea en particular. Este sistema de caracterización de usuarios guarda cada uno de estos eventos

con información respecto a tiempos de respuesta, horarios, tipo de actividad, asunto de la actividad y si se aceptó o se rechazó. El sistema entrega además datos agregados respecto a esta información.

Si bien se logra implementar un chatbot nuevo y un sistema de caracterización básico, los resultados finales no son tan completos como se quería en un inicio, quedando pendientes algunas mejoras al chatbot y algunas conexiones entre distintos sistemas, además de que el sistema de caracterización todavía no se está usando para mejorar el ciclo de asignación.

Por todo lo dicho, los objetivos propuestos no se consideran completamente logrados. No obstante, el trabajo realizado no se pierde, pues sirve como base para seguir construyendo mejoras al sistema de Hermes.

7.2. Trabajo Futuro

Como se mencionó anteriormente, quedan pendientes algunas posibles mejoras al chatbot, como por ejemplo mejor manejo de casos bordes y estados, para hacer flujos más naturales. Además queda pendiente hacer algunas conexiones entre el backend de Hermes y el chatbot nuevo para que efectivamente se use en el ciclo de asignación.

Sumado a esto, el sistema de caracterización es bastante flexible y extensible, lo que permitirá agregar nuevos parámetros de caracterización y elaborar sistemas más complejos que sugieran un ciclo de asignación de tareas más óptimo.

Vale mencionar también que la aplicación móvil y el backend de Hermes tienen mucho espacio para mejorar todavía, tanto en experiencia de usuario como en la calidad del código. Esto podría incluir por ejemplo usar estándares más modernos de React Native o hacer que los sistemas sean menos dependientes entre sí, lo cual haría más fácil también conectar el chatbot nuevo y el sistema de caracterización, además de cualquier otro componente que tenga sentido agregar.

Finalmente, sería interesante aplicar inteligencia artificial en el flujo de asignación o en la forma en la que se comunica el chatbot para que sea más natural o se comporte distinto dependiendo de con quién está hablando.

Bibliografía

[1] W. H. Organization, "Ageing and health," 2018.

[2] J. L. Wiles, A. Leibing, N. Guberman, J. Reeve, and R. E. S. Allen, "The Meaning of "Aging in Place" to Older People," *The Gerontologist*, vol. 52, pp. 357–366, 10 2011.

[3] F. J. Gutierrez and S. F. Ochoa, "It takes at least two to tango: Understanding the cooperative nature of elderly caregiving in latin america," in *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW '17, (New York, NY, USA), p. 1618–1630, Association for Computing Machinery, 2017.

[4] F. J. Gutierrez, S. F. Ochoa, and J. Vassileva, "Identifying opportunities to support family caregiving in chile," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, (New York, NY, USA), p. 2112–2118, Association for Computing Machinery, 2016.

[5] H. Gelderblom, T. van Dyk, and J. van Biljon, "Mobile phone adoption: Do existing models adequately capture the actual usage of older adults?," in *Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, SAICSIT '10, (New York, NY, USA), p. 67–74, Association for Computing Machinery, 2010.

[6] F. J. Gutierrez and S. F. Ochoa, "Mom, i do have a family! attitudes, agreements, and expectations on the interaction with chilean older adults," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work Social Computing*, CSCW '16, (New York, NY, USA), p. 1402–1411, Association for Computing Machinery, 2016.

[7] A. Benavides, F. J. Gutierrez, and S. F. Ochoa, "Hermes: A digital assistant for coordinating invisible work in family elderly caregiving scenarios," in *Human Aspects of IT for the Aged Population. Healthy and Active Aging* (Q. Gao and J. Zhou, eds.), (Cham), pp. 437–450, Springer International Publishing, 2020.

[8] A. Benavides, "Hermes no solo es un mensajero: Agente virtual para apoyar el cuidado informal de adultos mayores," 2020.

[9] Y. Chen, K. Cheng, C. Tang, K. A. Siek, and J. E. Bardram, "The invisible work of health providers," *Interactions*, vol. 21, p. 74–77, Sept. 2014.

[10] M. Schorch, L. Wan, D. W. Randall, and V. Wulf, "Designing for those who are overlooked: Insider perspectives on care practices and cooperative work of elderly informal caregivers," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work amp; Social Computing*, CSCW '16, (New York, NY, USA), p. 787–799, Association for Computing Machinery, 2016.

[11] P. S. T. H. S. P. Papastavrou E, Kalokerinou A, "Caring for a relative with dementia:

family caregiver burden," *Journal of advanced nursing*, vol. 58, p. 446–457, 2007.

[12] A. Bozzon, "Enterprise crowd computing for human aided chatbots," in *Proceedings of the 1st International Workshop on Software Engineering for Cognitive Services*, SE4COG '18, (New York, NY, USA), p. 29–30, Association for Computing Machinery, 2018.

[13] "Guides and references for all you need to know about botpress." https://botpress.com/docs. Última visita: Abril 2021.

[14] L. Müller, J. Mattke, C. Maier, T. Weitzel, and H. Graser, "Chatbot acceptance: A latent profile analysis on individuals' trust in conversational agents," in *Proceedings of the 2019 on Computers and People Research Conference*, SIGMIS-CPR '19, (New York, NY, USA), p. 35–42, Association for Computing Machinery, 2019.

[15] E. A. Rissola, S. A. Bahrainian, and F. Crestani, "Personality recognition in conversations using capsule neural networks," in *IEEE/WIC/ACM International Conference on Web Intelligence*, WI '19, (New York, NY, USA), p. 180–187, Association for Computing Machinery, 2019.

[16] N. Tepper, A. Hashavit, M. Barnea, I. Ronen, and L. Leiba, "Collabot: Personalized group chat summarization," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, (New York, NY, USA), p. 771–774, Association for Computing Machinery, 2018.

[17] "Tornado web server - tornado 6.1 documentation." https://www.tornadoweb.org/en/stable/. Última visita: Abril 2021.

[18] "React - a javascript library for building user interfaces." https://reactjs.org/. Última visita: Abril 2021.

[19] "React native - learn once, write anywhere." https://reactnative.dev/. Última visita: Abril 2021.

[20] "Axios - promise based http client for the browser and node.js." https://github.com/axios/axios. Última visita: Abril 2021.

[21] "Welcome to flask - flask documentation (1.1.x)." https://flask.palletsprojects.com/en/1.1.x/. Última visita: Marzo 2021.

Anexo A

Chatbot Legado

A.1. Ejemplos de interacción con la API del Chatbot de BotCenter

A continuación hay ejemplos de mensajes y respuestas de la API del chatbot de BotCenter, para entender cómo utilizarla.

Código A.1: Ejemplo del mensaje que se envía a la API del chatbot de BotCenter

```
body = {
  "bot_id":str,
  "message":"Hola Hermes",
  "data":{
    "profile":{
      "id":"1001588",
      "name":"Tata Hermes",
      "email":"test@ann.cl",
      "phone":"+56987456321",
      "family":[
        "1001588"
      ],
      "user_type":"patient",
      "usertoken":str,
      "password":"hermes123",
      "invitation_accepted":True
    },
    "contacts":[
      {
        "id":"1001608",
        "name":"Helper tata",
        "email":"test4@ann.cl",
        "phone":"+56987456321",
        "family":[
          "1001588"
```

```
            ],
            "user_type":"helper",
            "invitation_accepted":True
        }
    ]
  }
}
```

Código A.2: Ejemplo de la respuesta que entrega la API del chatbot de BotCenter

```
response.json = {
  "next_node":"None",
  "message":"NO_ENTENDI",
  "session_id":str,
  "data":{
    "contacts":[
      {
        "name":"Helper tata",
        "user_type":"helper",
        "email":"test4@ann.cl",
        "invitation_accepted":True,
        "family":[
          "1001588"
        ],
        "id":"1001608",
        "phone":"+56987456321"
      }
    ],
    "profile":{
      "password":"hermes123",
      "name":"Tata Hermes",
      "user_type":"patient",
      "email":"test@ann.cl",
      "invitation_accepted":True,
      "family":[
        "1001588"
      ],
      "id":"1001588",
      "phone":"+56987456321",
      "usertoken":str
    },
    "entities":[

    ],
    "metadata":{
```

```
        "emails":[

        ],
        "phones":[

        ]
      },
      "intents":[

      ],
      "intent":"INTENT_NOT_FOUND",
      "nodes_visited":[
        "entry"
      ]
    },
    "state":"BOT_ENDED"
}
```

Anexo B

Chatbot Nuevo

B.1. Acciones y Hooks

A continuación hay ejemplos de acciones y hooks.

Código B.1: login. Acción para iniciar sesión. Se conecta con el backend y guarda los datos en la base de datos de Botpress

```
function action(bp: typeof sdk, event: sdk.IO.IncomingEvent, args: any, { user, temp, session } = event.state) {
  /** Your code starts below */

  /**
   * Login to bff
   * @title Login
   * @category Custom
   * @author Pedro Belmonte
   */
  const myAction = async () => {
    const axios = require('axios')

    const password = event.payload.text

    const userData = await bp.users.getAttributes(event.channel, event.target)

    const url = 'https://bff-hermes.herokuapp.com/bot/login'

    const params = `email=${userData.email}&password=${password}`

    const config = {
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
      }
    }

    const sessionId = bp.dialog.createId(event)
```

```
28
29      axios
30        .post(url, params, config)
31        .then(async res => {
32          user.firebaseId = res.data.profile.id
33          user.familyId = res.data.profile.family[0]
34          user.telegramTarget = event.target
35          user.email = userData.email
36
37          bp.users
38            .getOrCreateUser('api', `${res.data.profile.id}-telegram-${res.data.profile.family[0]}`)
39            .then(apiUserPromise => {
40              const apiUser = apiUserPromise.result
41              bp.logger.forBot(event.botId).info(JSON.stringify(apiUser))
42              apiUser.attributes = {
43                firebaseId: res.data.profile.id,
44                familyId: res.data.profile.family[0],
45                telegramTarget: event.target,
46                email: userData.email
47              }
48              bp.logger.forBot(event.botId).info(JSON.stringify(apiUser))
49            })
50
51          await bp.dialog.jumpTo(sessionId, event, 'Login.flow.json', 'Success')
52          await bp.dialog.processEvent(sessionId, event)
53        })
54        .catch(async err => {
55          bp.logger.forBot(event.botId).info(err)
56          await bp.dialog.jumpTo(sessionId, event, 'Login.flow.json', 'Failure')
57          await bp.dialog.processEvent(sessionId, event)
58        })
59    }
60
61    return myAction()
62
63    /** Your code ends here */
64  }
```

Código B.2: choose_elder. Acción para guardar la información del paciente para una cita médica. Si ya se indicó entonces solo lo guarda, si no, genera un nodo que entrega las opciones con botones.

```
1  function action(bp: typeof sdk, event: sdk.IO.IncomingEvent, args: any, { user, temp, session } = event.state) {
2    /** Your code starts below */
3
4    /**
```

```
   * Select an elder from list
   * @title Choose elder
   * @category Custom
   * @author Pedro Belmonte
   * @param {string} text - Text said by user
   */
  const myAction = async text => {
    let patient = null
    user.elders.forEach((item, index) => {
      if (text.search(new RegExp(item, 'i')) !== -1) {
        patient = index
      }
    })
    if (patient != null) {
      temp.options = false
      temp.chosenPatient = patient
      temp.chosenPatientString = user.elders[patient].name
      const sessionId = bp.dialog.createId(event)
      await bp.dialog.jumpTo(sessionId, event, 'Cita-medica.flow.json', 'Subject-Cita')
    } else {
      temp.options = true
      const choices = user.elders.map((item, index) => {
        return {
          title: item.name,
          value: index.toString()
        }
      })
      const data = {
        text: '¿De quién es esta cita?',
        choices: choices,
        typing: false
      }
      const payloads = await bp.cms.renderElement('builtin_single-choice', data, event)
      await bp.events.replyToEvent(event, payloads)
    }
  }

  return myAction(args.text)

  /** Your code ends here */
}
```

Código B.3: to_telegram_target. Hook que se ejecuta antes de procesar un mensaje. Se usa para que cuando se inicia una conversación mediante la API pero se quiere que la conversación continue en Telegram se use un identificador especial. Pensado para iniciar la conversación de asignación de tareas.

```
function hook(bp: typeof sdk, event: sdk.IO.IncomingEvent) {
  /** Your code starts below */

  if (event.target.includes('telegram')) {

    event.target = event.state.user.telegramTarget
    event.type = 'message'
    event.channel = 'telegram'

    const date = new Date()

    event.payload = {
      from: {
        id: parseInt(event.state.user.telegramTarget),
        is_bot: false
      },
      chat: {
        id: parseInt(event.state.user.telegramTarget),
        type: 'private'
      },
      date: date.getTime(),
      text: event.payload.text
    }
  }

  /** Your code ends here */
}
```