UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Sistema de recomendación conversacional basado en aspectos

Celia San Gregorio Moreno
Tutor: Alejandro Bellogín Kouki
Ponente: Iván Cantador Gutiérrez

Mayo de 2020

Sistema de recomendación conversacional basado en aspectos

AUTOR: Celia San Gregorio Moreno
TUTOR: Alejandro Bellogín Kouki
PONENTE: Iván Cantador Gutiérrez

Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo de 2020

Resumen (castellano)

Este Trabajo de Fin de Grado tiene como objetivo desarrollar un sistema conversacional (en concreto, un *chatbot*) que sea capaz de abordar dos casos de uso: permitir que el usuario encuentre opiniones sobre determinados aspectos de ítems que pertenecen a un dominio, y ofrecer recomendaciones de ítems según las preferencias del usuario hacia aspectos de dichos ítems.

El sistema está compuesto por tres elementos: un agente de Dialogflow integrado en un *bot* de Telegram, que se encarga de interactuar con el usuario y obtener sus preferencias a partir de mensajes de texto; un servicio web Spring Boot alojado en Heroku, que implementa la lógica interna del sistema y se comunica con el agente de Dialogflow a través de una interfaz REST; y una base de datos PostgreSQL que almacena información sobre el dominio y las preferencias del usuario, y es accesible desde una aplicación de Heroku.

Una vez integrados, todos los elementos descritos anteriormente satisfacen una serie de requisitos clave: poseer un diseño que permita realizar cambios de dominio sin necesidad de re-implementar toda la estructura del sistema, identificar *keywords* relacionadas con opiniones, características del dominio y cualidades asociadas a aspectos de ítems; analizar la reacción del usuario a los resultados ofrecidos por el sistema, y guiar la conversación con el fin de adquirir la información necesaria.

Este documento detalla las etapas que tuvieron lugar desde el comienzo del proyecto hasta su finalización. En primer lugar, ofrece una breve introducción a los sistemas conversacionales y analiza el estado del arte de las tecnologías de *chatbots* más populares. A continuación, describe las fases de diseño y desarrollo del proyecto, divididas en dos iteraciones con objetivos diferenciados. Por último, expone las pruebas realizadas sobre el sistema y presenta conclusiones y mejoras que pueden incorporarse en su estructura.

Palabras clave (castellano)

Sistema de recomendación, sistema conversacional, chatbot, opiniones de usuario, aspectos de ítem, servicio web, base de datos, Dialogflow, Heroku, Spring Boot, PostgreSQL, Java

Abstract (English)

This Bachelor Thesis aims to develop a conversational system (more specifically, a chatbot) capable of addressing two use cases: allowing a user to find opinions about particular aspects of items in a given domain, and providing item recommendations according to the user's preferences for item aspects.

The system is composed of three elements: a Dialogflow agent integrated in a Telegram bot, which is in charge of interacting with the user and obtaining user preferences from text messages; a Spring Boot web service hosted on Heroku, which implements the system's internal logic and communicates with the Dialogflow agent using a REST interface; and a PostgreSQL database, which stores information about the domain and the user's preferences, and is reachable from a Heroku app.

Once integrated, all the above elements satisfy a number of key requisites: following a design that enables domain changes without the need to re-implement the whole system structure, identifying keywords related to opinions, domain features and characteristics associated to given item aspects; analyzing the user's reaction to the results provided by the system, and guiding the conversation with the aim of acquiring necessary user input.

This document details the stages that took place from the start of the project to its completion. Firstly, it offers a brief introduction to conversational systems and reviews the state of the art on popular chatbot technologies. Next, it describes the design and development processes of the project, split into two iterations with specific objectives. Finally, it reports the evaluations performed on the system, and presents conclusions and improvements that may be done on the system.

Keywords (inglés)

Recommender system, conversational system, chatbot, user opinions, item aspects, web service, database, Dialogflow, Heroku, Spring Boot, PostgreSQL, Java

Agradecimientos

A mi tutor Alejandro y mi ponente Iván, por todo su apoyo y dedicación durante el desarrollo del proyecto, desde el principio hasta el final.

A todas las personas que me han acompañado y animado durante este viaje, y a aquellos que ya no están para celebrarlo.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

En la sociedad actual, la relevancia de los sistemas de recomendación conversacionales ha aumentado considerablemente durante los últimos años. Su atractivo se debe a dos factores principales:

- Son capaces de identificar los aspectos de ítems que resultan de interés para el usuario a través de preguntas elaboradas cuidadosamente [39].
- Permiten ofrecer servicio a un gran número de usuarios de forma simultánea. Sus tiempos de respuesta son rápidos, y pueden realizar búsquedas de datos en cuestión de milisegundos [30].

Sin embargo, el mayor reto a la hora de implementar un sistema de estas características consiste en interpretar correctamente el lenguaje que recibe como *input* (habla o texto). Además de ser capaz de comunicarse con el usuario como haría una persona humana [6], el sistema debe incorporar los siguientes mecanismos:

- Capacidad para analizar el significado de las palabras teniendo en cuenta el contexto de la conversación en curso.
- Identificación de los términos vinculados a aspectos sobre los que el usuario realiza consultas. Esta funcionalidad es particularmente crítica, ya que de ella depende la capacidad del sistema para ofrecer una respuesta ajustada a las necesidades del cliente..
- Métodos para redirigir la conversación en caso de recibir *input* que no se ajuste al ámbito especificado.

Para manejar las interacciones con personas reales, la mayor parte de sistemas conversacionales hacen uso de tecnologías de Procesamiento del Lenguaje Natural (NLP) [39]. Su objetivo principal consiste en manipular el lenguaje humano (sintaxis, conceptos y matices asociados a las palabras, ambigüedades), de forma que un programa interprete y comprenda su estructura de forma precisa [11]. El uso de NLP en los sistemas de recomendación que interaccionan con usuarios resulta beneficial tanto para el cliente como para las compañías, ya que una mayor satisfacción con las recomendaciones recibidas implica un aumento de confianza en el sistema y su evolución en el mercado [39].

1.1 Motivación

El mayor incentivo a la hora de implementar el proyecto fue la posibilidad de profundizar mis conocimientos en el mundo de los sistemas conversacionales, ámbito que se exploró brevemente en la asignatura de Desarrollo Automatizado de Software (Código 18778). Gran parte del atractivo se encontraba en los retos planteados en el apartado anterior: el sistema debía ser capaz de dialogar con el usuario e interpretar sus mensajes; no bastaba con pulsar una serie de botones o realizar preguntas de tipo sí/no.

Además de la oportunidad para trabajar con este tipo de sistemas, el proyecto exigía el uso de una serie de tecnologías con las que no estaba familiarizada. De este modo, no sólo iba a asimilar conceptos sobre el funcionamiento interno de un sistema conversacional, sino que aprendería a utilizar *frameworks* y herramientas punteras que podrían serme de utilidad una vez finalizase el grado.

1.2 Objetivos

El objetivo principal del proyecto consiste en desarrollar un *chatbot* que sea capaz de procesar los mensajes enviados por un usuario, con el fin de ofrecer opiniones o recomendaciones sobre un determinado dominio. Este objetivo engloba una serie de requisitos fundamentales sobre el sistema:

- Reconocer sobre qué objeto del dominio se desea obtener opiniones, así como el tipo de las mismas (positivas, negativas o ambas). La respuesta del *chatbot* debe ajustarse a estos parámetros.
- Identificar tanto aspectos de ítems pertenecientes al dominio como cualidades o adjetivos que el usuario decida expresar sobre ellos. Las recomendaciones que devuelve el *chatbot* deben tener en cuenta los intereses del usuario.
- Interpretar la reacción del usuario (favorable o no) ante las opiniones o recomendaciones que ha recibido sobre el dominio. En función de sus impresiones, el sistema debe actuar en consecuencia: por ejemplo, continuando la consulta o realizando una nueva.
- Guiar al usuario para que proporcione la información necesaria durante la conversación, y redirigirla en caso de recibir *input* fuera de ámbito.
- Poseer un diseño que permita realizar cambios de dominio sin necesidad de reconstruir su estructura desde cero, así como acceder rápidamente a la información disponible. Este último requisito nace de las estrictas restricciones de retardo que existen en una conversación entre dos personas.

1.3 Organización de la memoria

Este documento se ha dividido en los siguientes capítulos:

- **Estado del arte.** Describe los sistemas conversacionales de mayor relevancia en la actualidad, desde asistentes virtuales como Siri, Cortana, Amazon Alexa y Google Assistant hasta *chatbots* desarrollados para múltiples plataformas. También realiza un análisis de las tecnologías sobre las que se apoya el proyecto.
- **Diseño.** Incluye un esquema de interacción entre los componentes del proyecto, así como su proceso de diseño dividido en dos iteraciones. Cada iteración recoge los requisitos de los componentes y las técnicas utilizadas para cumplirlos.
- **Desarrollo.** De un modo similar al apartado de diseño, esta sección detalla la funcionalidad implementada en cada componente del proyecto durante las dos iteraciones. Cabe destacar que la fase de desarrollo fue la más larga y la que más desafíos planteó a la hora de satisfacer los requisitos de cada componente.
- **Pruebas y resultados.** Describe la batería de pruebas realizadas sobre el sistema conversacional y los resultados obtenidos al aplicarlas. Algunas de ellas facilitaron la detección temprana de errores y su consecuente resolución.
- **Conclusiones y trabajos futuros.** Incluye una reflexión sobre el aprendizaje, las dificultades y los retos asociados a la realización del proyecto, así como una serie de futuras implementaciones sobre el sistema.

2 Estado del arte

Como se ha expuesto en el apartado anterior, la relevancia de los sistemas conversacionales radica en su capacidad para interpretar el lenguaje humano y producir un resultado que se ajuste a los requerimientos del usuario. Actualmente existen dos categorías dentro de este tipo de sistemas: asistentes virtuales y *chatbots*. Ambos se describen con detalle en las secciones 2.1 y 2.2:

Por último, la sección 2.3 incluye las restricciones técnicas del proyecto y un estudio de las tecnologías utilizadas para implementarlo.

2.1 Asistentes virtuales

Con el avance de la tecnología NLP en los últimos años, Apple, Microsoft, Amazon y Google se han posicionado como cuatro compañías de referencia en lo relativo a sistemas conversacionales. Todas ellas han introducido sus propios asistentes virtuales en el mercado, denominados: Siri (Apple), Cortana (Microsoft), Alexa (Amazon) y Google Assistant.

2.1.1 Siri

Siri es un asistente capaz de interpretar el lenguaje hablado en 21 idiomas [3]. Sus inicios se remontan a 2010, momento en que Apple da a conocer el software como una aplicación independiente de su propia tienda. Un año después, toda su funcionalidad se incorpora permanentemente a los sistemas operativos iOS y macOS [31].

Siri procesa tanto preguntas en lenguaje hablado como comandos de voz [32], y su objetivo principal consiste en facilitar la gestión de una gran variedad de tareas [3] [4]. Entre otras, permite realizar llamadas y enviar mensajes de texto, ofrecer recomendaciones de música, e indicaciones para llegar a un destino, realizar transacciones de dinero y controlar dispositivos del hogar compatibles con su software. También permite su sincronización con servicios propios de Apple como Mail, Maps o Safari; de este modo, es capaz de determinar con mayor precisión las necesidades del usuario.

2.1.2 Cortana

Cortana es un asistente personal desarrollado y lanzado por Microsoft en 2014, que actualmente se encuentra incorporado en entornos Windows, Xbox One y Android, entre otros [7]. Permite recibir tanto comandos de voz como *input* en formato textual, y su funcionalidad se divide en *skills* o tareas que es capaz de facilitar al usuario [29].

De un modo similar a Siri, este asistente se apoya en los servicios de Microsoft para ofrecer una experiencia más ajustada a los requerimientos del usuario [36]. A través de aplicaciones como Outlook o Calendar, Cortana es capaz de extraer información relevante para el usuario a través de su correo electrónico (un ejemplo sería la fecha de una futura reunión), así como de crear, modificar y eliminar tanto eventos como alarmas. También hace uso de servicios como Bing Maps para mostrar previsiones de tiempo o indicaciones

sobre cómo llegar a un destino. Si el usuario realiza una consulta, Cortana utiliza el *input* recibido para ejecutar una búsqueda online a través de Bing.

Cabe destacar que el 28 de febrero de 2020 Microsoft anunció una serie de modificaciones en el diseño principal de Cortana: su objetivo de aquí en adelante consistiría en aumentar la productividad de sus usuarios, para lo cual algunas funcionalidades orientadas al ocio serían retiradas de su lista de *skills* [38].

2.1.3 Amazon Alexa

Este asistente virtual fue introducido en el mercado en 2014 [31], disponible exclusivamente en los altavoces propios de la compañía (Amazon Echo). Sin embargo, el aumento de su popularidad ha permitido la incorporación del asistente en otros dispositivos como televisiones, teléfonos móviles o incluso vehículos [33].

Alexa reconoce preguntas en lenguaje hablado y comandos de voz. De forma similar a Cortana, cuenta con un amplio repertorio de *skills* o tareas que facilita a sus consumidores. Entre todas ellas destacan [1] [34]: enviar y recibir tanto llamadas como mensajes, adquirir productos de Amazon o realizar donaciones a organizaciones benéficas, reproducir música a través de Bluetooth o aplicaciones de *streaming* oficiales, hacer uso de servicios de terceros como Uber o Just Eat, y controlar dispositivos del hogar compatibles con el asistente. Alexa dispone además de una utilidad denominada Alexa Guard, configurada para detectar sonidos específicos (por ejemplo, alarmas) y notificar al usuario de una posible emergencia.

2.1.4 Google Assistant

El asistente virtual de Google tuvo su debut en 2016 como extensión de Google Now [18]. Actualmente se distribuye como una aplicación independiente, disponible en altavoces Google Home y Google Nest, teléfonos y televisiones Android, e incluso terminales iOS [31].

Google Assistant es capaz de interpretar tanto *input* de voz como de texto. Del mismo modo que los asistentes expuestos en apartados anteriores, Google Assistant hace uso de servicios propios de la compañía y de terceros para ofrecer una experiencia centrada en las necesidades de cada usuario [35].

Entre las soluciones que puede ofrecer Google Assistant destacan las siguientes [35]: abrir y manejar aplicaciones mediante comandos de voz, reproducir un sonido de alarma en teléfonos compatibles para facilitar su localización, ignorando configuraciones de Silencio o No Molestar; hacer uso de la cámara en terminales móviles para traducir en vivo carteles y letreros, ofrecer recomendaciones de restaurantes y destinos vacacionales, mostrar predicciones del tiempo y calcular indicaciones para llegar a un determinado destino.

2.2 Chatbots

Del mismo modo que los asistentes virtuales, los chatbots se han diseñado como una interfaz que incorpora un producto o un servicio en su lógica interna. Los usuarios, a través de una conversación en lenguaje natural, interaccionan con dicho producto o servicio [37].

Sin embargo, existen dos diferencias con respecto a los asistentes virtuales:

- Los *chatbots* sólo procesan *input* en formato textual (mensajes procedentes del usuario).
- Su funcionalidad suele enfocarse hacia una conversación más corta que con un asistente virtual, y teniendo en cuenta objetivos concretos [40].

Con el uso cada vez más extendido de los teléfonos móviles, un tipo de aplicaciones que han mantenido su puesto en el ranking de popularidad han sido las dedicadas a mensajería [37]. Plataformas como Telegram, Facebook Messenger o Slack han implementado sus propios mecanismos para facilitar el desarrollo de chatbots, siendo éstos APIs, SDKs o librerías cliente. Ello ha permitido la proliferación de chatbots de diversas categorías, entre las que destacan [37] [40]:

- ***Chatbots* de entretenimiento.** Su software implementa mecanismos recreacionales como mini juegos, chistes, encuestas o aproximaciones a un diálogo casual. En este apartado merece especial mención Cleverbot [5], lanzado en 2006 en formato web con el fin de entablar conversaciones con personas humanas. Todo el input que recibe se utiliza para mejorar la experiencia de usuario y simular los patrones conversacionales presentes en los seres humanos.

- ***Chatbots* enfocados a negocio o marketing.** En la mayor parte de los casos, los chatbots que pertenecen a esta categoría son transaccionales [9]. El usuario puede realizar un número limitado de operaciones con ellos (por ejemplo, consultar el balance de una cuenta bancaria o solicitar recomendaciones de música), y la conversación se centra en sus propias necesidades y preferencias. Un caso de éxito fue Ralph [28], desarrollado por LEGO en Facebook Messenger para realizar recomendaciones y guiar los clientes en la compra de sus propios productos.

- ***Chatbots* enfocados a productividad y alcance de metas.** Su popularidad se debe a dos factores: la rapidez en el intercambio de información que facilita una conversación por chat, y el tratamiento personalizado que se realiza sobre cada usuario. Este tipo de *chatbots* elabora recordatorios, listas de tareas diarias y un seguimiento de objetivos centrado en las necesidades del usuario. Dentro de esta categoría destaca Lark [27], diseñado como un entrenador personal para mejorar las condiciones de salud de sus clientes.

- ***Chatbots* de atención al cliente.** Su objetivo principal consiste en atender de forma rápida y precisa a las preguntas más frecuentes que podrían realizar los usuarios sobre un determinado producto o servicio. Dado el patrón de consulta que lleva asignada su correspondiente respuesta, estos chatbots resultan relativamente sencillos de implementar. Uno de los más populares actualmente es Dom [12], desarrollado por Domino's Pizza para resolver dudas del usuario y procesar pedidos a domicilio en Norteamérica.

2.3 Estudio de las tecnologías utilizadas

A la hora de definir los componentes del sistema conversacional y sus flujos de información, debían considerarse las siguientes restricciones técnicas:

- El uso de un agente de Dialogflow para interaccionar con el usuario, integrado en un chatbot de Telegram que procesa mensajes de texto en inglés.
- La creación de un servicio web, preferiblemente en Java, y su despliegue en Heroku.
- El uso de una base de datos, preferiblemente en PostgreSQL, para almacenar toda la información relevante sobre el dominio.
- La capacidad del sistema para trabajar con aspectos de ítems pertenecientes a distintos dominios, sin necesidad de re-implementar toda su arquitectura.

El agente de Dialogflow se correspondería con la capa del sistema visible al exterior: un *chatbot* que actuaría como interfaz entre el usuario y el servicio ofrecido. Este agente se desarrollaría a través de la plataforma online Dialogflow, diseñada por Google para crear sistemas conversacionales capaces de procesar el lenguaje natural [16]. Todos los sistemas implementados mediante Dialogflow deben incorporar una serie de componentes que manejan distintos aspectos de una conversación:

- **Entity types.** Los tipos de entidades o *entity types* representan palabras clave que un agente es capaz de identificar. Por ejemplo: fechas, valores numéricos como precios, o aspectos de ítems. En este último caso, un aspecto de ítem en el ámbito de una película podría ser el nombre completo de su director.
- **Intents.** Los *intents* detectan las intenciones del usuario para realizar cambios de tema durante su turno de conversación. En Dialogflow, los *intents* llevan asociados una serie de sub-componentes: frases de entrenamiento, parámetros, contextos, eventos, acciones y respuestas.
- **Frases de entrenamiento.** Las frases de entrenamiento representan ejemplos de mensajes que un usuario podría enviar al sistema. Cuando el *input* del usuario coincide con frases de entrenamiento asociadas a un *intent*, éste se activa y Dialogflow muestra la respuesta correspondiente. Es posible anotar uno o más términos de las frases de entrenamiento utilizando parámetros.
- **Parámetros.** Asocia una palabra o un conjunto de palabras de una frase de entrenamiento a un tipo de entidad. De este modo, indica al agente qué términos debe identificar como palabras clave cuando reciba *input* del usuario. La representación textual del término identificado se almacena en el parámetro; esta información puede mostrarse en una respuesta del agente o enviarse a un componente externo como un servicio web.
- **Contexto.** Representan la información adicional que rodea una conversación, y permite al agente determinar sobre lo que está hablando el usuario. En Dialogflow, los contextos sirven de transición entre un *intent* y otro, y permiten guiar el diálogo hacia un fin específico. Existen dos tipos de contextos: contextos de salida y contextos de entrada. Los primeros se activan cuando se lanza un *intent*; por su parte, los contextos de entrada restringen la activación de ciertos *intents*: sólo pueden lanzarse si el *input* del usuario coincide con las frases de entrenamiento y si el correspondiente contexto de salida está activo en ese momento.
- **Eventos.** Los eventos representan una forma alternativa de activar *intents*. Permiten lanzarlos directamente y en función de determinados sucesos, sin necesidad de comprobar si las frases de entrenamiento coinciden con el *input* del usuario.

- ***Fulfillment.*** Este término hace referencia a la capacidad del agente para comunicarse con un servicio web o *webhook*. La opción de *fulfillment* puede habilitarse en los *intents*. De este modo, cuando un *intent* con *fulfillment* habilitado se activa, Dialogflow envía una solicitud al servicio web con toda la información contenida dicho *intent* (contextos, parámetros, tipos de entidades, etc.). Después de enviar la petición, esperará la llegada de una respuesta desde el servicio web.
- **Acciones y respuestas.** Cuando Dialogflow activa un *intent,* desencadena una serie de procesos: acciones, *fullfilment* (si está habilitado) y respuestas. Las acciones consisten en un campo de texto que se adjunta en las peticiones al servicio web con el fin de ejecutar lógica específica. Por su parte, las respuestas son mensajes de texto o de voz que el agente muestra de vuelta al usuario. Estas respuestas pueden consistir en mensajes predeterminados o datos procedentes del servicio web.

En cuanto al servicio web y la base de datos, se realizó un estudio sobre los mecanismos de gestión de bases de datos disponibles en Heroku, así como de *frameworks* de servicios web aptos para cumplir con las restricciones técnicas.

En el primer caso, el mayor interés se centraba sobre la posibilidad de incorporar una base de datos PostgreSQL como parte de una aplicación alojada en Heroku. Gracias al *add-on* Heroku Postgres [24], disponible de forma nativa en esta plataforma, se pudo llevar a cabo la funcionalidad deseada.

En cuanto a *frameworks* para desarrollar servicios web, se barajaron tres candidatos: JAX-WS, Dropwizard y Spring Boot. De entre todos ellos, se escogió finalmente Spring Boot. Esta decisión se fundamentó en una serie de factores muy beneficiosos para el desarrollo del proyecto:

- Su sencilla configuración de despliegue en Heroku (muy cercana a "*one-click*").
- La calidad de la documentación y los tutoriales disponibles para este *framework*.
- La curva de aprendizaje moderada para el ámbito de este proyecto.
- El hecho de que Spring Boot está comenzando a marcar tendencia en el desarrollo de servicios web dentro de las empresas tecnológicas.
- El enfoque de Spring Boot a construir servicios web con componentes configurados automáticamente mediante ficheros de configuración y anotaciones Java como @Autowired, así como con el uso de mecanismos como escaneo de componentes o inyección de dependencias.
- La disponibilidad de librerías que permiten la serialización/deserialización a formato JSON a través de anotaciones (como Jackson [10], por ejemplo), que facilitan considerablemente el proceso de transformación de datos JSON a clases Java y viceversa.
- El uso por defecto de HikariCP [44], un pool de conexiones JDBC considerado como uno de los más rápidos y eficientes que existen en la actualidad. La velocidad del pool de conexiones es un requisito de vital importancia para este proyecto, ya que las peticiones que envía Dialogflow a un servicio web poseen un timeout de 5 segundos.
- La integración de herramientas de *building* automatizado como Maven o Gradle, necesarias para desplegar y ejecutar el servicio web en Heroku.

3 Diseño

Una vez concluido el estudio de *frameworks* y gestión de bases de datos expuesto en el apartado anterior, se definieron tres componentes dentro del sistema conversacional:

1. Un agente de Dialogflow que reside en un bot de Telegram y recibe *input* en inglés.
2. Una base de datos PostgreSQL alojada en una aplicación de Heroku.
3. Un servicio web desarrollado con Spring Boot y desplegado en otra aplicación de Heroku.

Todos ellos deben realizar las funciones adecuadas para recuperar opiniones sobre un dominio a demanda del usuario, así como ofrecer recomendaciones en función de preferencias especificadas a lo largo de la conversación. Aunque es posible configurar el sistema para permitir su funcionamiento con distintos dominios, se estableció como caso práctico el uso de hoteles.

Con el objetivo de estructurar el diseño y desarrollo del sistema, así como de generar entregas funcionales e incrementales, se realizaron dos iteraciones: la primera cubrió todos los mecanismos necesarios para mostrar opiniones de hoteles al usuario; por su parte, la segunda iteración incorporó el modelo de recomendaciones en base a preferencias.

Los siguientes apartados describen con un enfoque *top-down* el proceso de diseño. El apartado 3.1 muestra un esquema de interacción simplificado. A continuación, los apartados 3.2 y 3.3 realizan un análisis en detalle de los componentes en cada incremento.

3.1 Esquema general de interacción

Cada elemento del sistema conversacional posee determinados flujos de interacción, que se muestran de forma esquemática en la Figura 3-1:

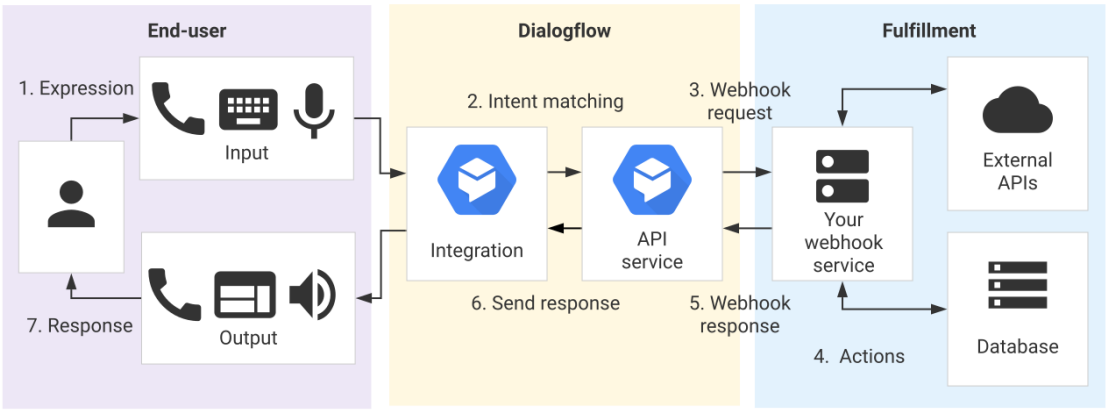


Figura 3-1: Componentes del sistema conversacional. Fuente: [16].

Dado que el agente de Dialogflow se encuentra integrado en un bot de Telegram, el usuario debe iniciar dicho bot para comenzar una conversación. Una vez se ha realizado este paso, el agente muestra la siguiente frase de presentación:

```
"Welcome! I can search for hotels that suit your needs, as well as hotel
opinions. What do you want me to search about?"
```

El usuario puede realizar dos consultas en este momento: obtener todos los hoteles que cumplan una serie de preferencias, o buscar opiniones sobre ellos. Diferenciamos, por tanto, dos casos de uso principales:

- Buscar opiniones.
- Obtener recomendaciones ajustadas a preferencias.

3.1.1 Caso de uso A: Buscar opiniones

Si el usuario desea consultar las opiniones asociadas a un hotel o cadena de hoteles determinados, responderá al agente con un mensaje de este tipo:

```
"Get all opinions from AC Hotels"
```

Dialogflow reconocerá el cambio de tema en la conversación y enviará una petición al servicio web de Spring Boot a través de su API. La petición contiene datos en formato JSON que serán deserializados y procesados por el servicio, con el fin de generar una consulta a la base de datos que reside en otra aplicación web. Cuando recupere la información solicitada, generará una respuesta en formato JSON que envía de vuelta a Dialogflow

Dialogflow extraerá la información que recibe desde el servicio web y la mostrará al usuario en forma de mensaje de respuesta procedente del bot de Telegram.

3.1.2 Caso de uso B: Recomendaciones ajustadas a preferencias

Si el usuario decide compartir ciertos aspectos de ítems que le interesan sobre un dominio, su respuesta puede asemejarse a este *input*:

```
"Show me hotels with clean bathrooms and great wifi"
```

El flujo de acción en este caso es similar al descrito anteriormente: Dialogflow detectará la intención del usuario para referirse al hecho de recibir recomendaciones, y enviará una solicitud al servicio web. Este servicio obtendrá los aspectos de interés («*bathrooms*», «*wifi*») y cómo de exigente es el criterio del usuario en función de los adjetivos utilizados («*clean*», « *great*»). A partir de esta información, generará una consulta a la base de datos PotsgreSQL, de la que obtendrá una lista de hoteles que cumplen con los requisitos del usuario. Finalmente, la lista se enviará de vuelta a Dialogflow en formato JSON.

Dialogflow interpretará los datos recibidos y los mostrará como un mensaje emitido desde el bot de Telegram.

En ambos casos de uso, las interacciones posteriores entre usuario y bot siguen el mismo esquema de acción: el usuario escribe un mensaje, que Dialogflow interpreta y utiliza para enviar una petición al servicio web. El servicio, a su vez, procesa los datos contenidos en la solicitud y devuelve una respuesta a Dialogflow. Cuando el agente recibe dicha respuesta, la muestra a través del bot de Telegram.

3.2 Diseño en la primera iteración

3.2.1 Agente de Dialogflow: HotelReviewProvider

Antes de implementar el agente, se realizó un estudio de la documentación [16] y diversos tutoriales de Google [19] con el objetivo de asimilar los conceptos básicos sobre el flujo de conversación, las *keywords* de interés que podría enviar el usuario, y el modo de interacción entre esta plataforma y un servicio web externo.

Una vez concluido el estudio, se diseñó un diagrama de flujo (ver Anexo A) para analizar los posibles cambios de tema y la gestión de situaciones anómalas o *edge cases*. Una situación anómala puede tener lugar si el agente no recibe la información requerida por parte del usuario, o si procesa un *input* que no está relacionado con la conversación actual.

Durante la primera iteración, el objetivo principal fue añadir mecanismos en el agente de Dialogflow para que detectase cuándo el usuario deseaba obtener opiniones sobre hoteles, así como su reacción ante ellas. En este último caso, si el usuario expresaba su insatisfacción con los resultados, el agente debía permitirle realizar una nueva búsqueda.

Con el fin de detectar la intención del usuario para buscar opiniones, se definieron una serie de palabras clave que Dialogflow debía identificar en un mensaje de entrada:

- **Keywords relacionadas con nombres y cadenas de hoteles.** Dialogflow debía asociar palabras como «*Asheville Downtown*» o «*AC Hotels*» a un nombre y una cadena de hoteles, respectivamente.
- **Keywords relacionadas con opiniones.** Abarcan desde palabras como «*opinion*» o «*assumption*», hasta expresiones más elaboradas como «*What do people think about (<hotel_name>|<hotel_chain>)?*». Todas ellas, una vez identificadas, debían referirse al hecho de buscar impresiones sobre hoteles.
- **Keywords relacionadas con tipos de opiniones.** Un usuario podría buscar opiniones positivas, negativas o de ambos tipos sobre un dominio, con lo que surgió la necesidad de asociar palabras como «*positive*» o «*optimistic*» a puntos de vista favorables, y términos como «*negative*» o «*unfavorable*» a opiniones adversas. Si no se expresaba la demanda de opiniones positivas o negativas, Dialogflow asumiría que el usuario deseaba ver todos los puntos de vista.
- **Keywords relacionadas con reacciones al resultado ofrecido.** Cuando el usuario recibiese la lista de opiniones desde el agente, podría reaccionar de dos maneras: aceptando el resultado o rechazándolo. Si lo aceptaba, Dialogflow debía asociar este sentimiento a expresiones como «*I like this*» o «*I'm satisfied with the result*». En caso contrario, el rechazo de opiniones iría ligado a mensajes como «*I hate this*» o «*I'm not satisfied with the result*».
- **Keywords relacionadas con una respuesta afirmativa o negativa.** Si el usuario no quedaba convencido con las opiniones recibidas, el agente debería ofrecerle la posibilidad de realizar otra búsqueda. Para ello, era necesario conocer si se recibía *input* afirmativo («*yes*», «*sure*», «*okay*») o negativo («*no*», «*not interested*»), y si dicho *input* venía después de ofrecer un nuevo intento al usuario.

Todas las *keywords* descritas anteriormente podrían aparecer en mensajes que no contengan necesariamente las mismas palabras en el mismo orden. Por tanto, se definieron una serie de expresiones que el agente de Dialogflow almacenaría como frases de entrenamiento, ya que el usuario podría enviar este tipo de mensajes al sistema. La Tabla 3-1 muestra varios ejemplos:

Frases de entrenamiento	
Tipo	**Contenido**
Buscar opiniones	«*Get positive opinions of AC Hotels*»
Buscar opiniones	«*What are people's thoughts about Asheville Downtown?*»
Buscar opiniones	«*Display only negative opinions about Hilton Hotels*»
Reacción al resultado	«*Thanks, I love these*»
Reacción al resultado	«*I don't like this information*»

Tabla 3-1: Frases de entrenamiento del agente HotelReviewProvider.

A continuación, se establecieron las acciones que debía tomar el agente según el *input* recibido, así como las respuestas que devolvería al usuario. Cuando se solicitase una búsqueda de opiniones sobre un hotel, el agente interaccionaría con el servicio web y devolvería una lista con todas las reseñas asociadas al mismo. Si el usuario envía mensajes sobre su conformidad con los resultados, Dialogflow daría por concluida la conversación. En caso contrario, el agente ofrecería la posibilidad de realizar una nueva búsqueda. Si el usuario accede a ello, comenzaría una nueva interacción; si no, el bot de Telegram concluiría el diálogo.

Por último, se definieron mecanismos para redirigir la conversación en caso de producirse situaciones anómalas, como mensajes fuera de ámbito al iniciar el diálogo con el agente o después de recibir la lista de opiniones.

3.2.2 Base de datos de hoteles y opiniones

Ante la imposibilidad de almacenar datos estructurados en Dialogflow sobre hoteles y sus opiniones asociadas, se definieron los siguientes requisitos para crear una base de datos PostgreSQL con esta información:

- Un hotel posee un identificador único, un nombre y una cadena a la que pertenece.
- Un hotel puede tener de 0 a N opiniones diferentes.
- Una opinión está compuesta por un identificador único, un sentimiento (positivo, negativo o "todos"), y su contenido en formato textual.
- Una opinión concreta está asociada a un único hotel.

Estos requisitos se reflejan en la Figura 3-2 mostrada a continuación:

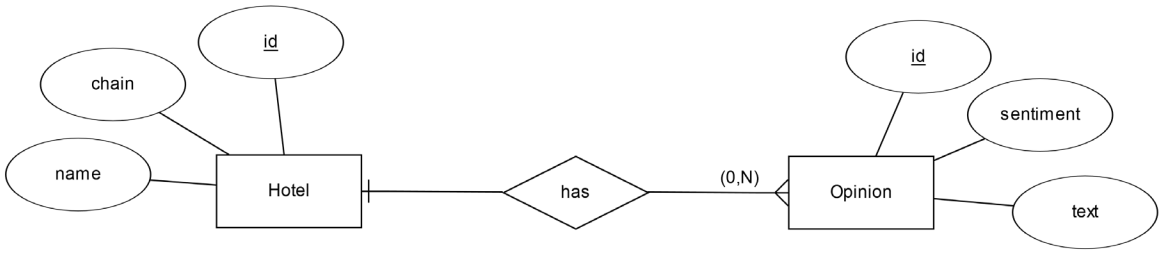


Figura 3-2: Diagrama ER de la base de datos.

Los hoteles y opiniones se representan como entidades que poseen tres atributos, de los cuales *id* es un atributo clave. La relación entre un hotel y sus correspondientes opiniones es de tipo (1, N).

3.2.3 Servicio web: hotel-review-provider

Para definir los aspectos clave del último componente, se siguieron diversas guías de Heroku [23] y Spring Boot [42], y se consultó documentación con el fin de conocer el funcionamiento de estas herramientas. Una vez realizada esta primera toma de contacto, comenzó el proceso de diseño.

Durante la primera iteración, el servicio web se encargaría de ejecutar una sucesión de tareas:

- Recibir una solicitud o Webhook Request [20] procedente de Dialogflow. Dado que la información viene representada en formato JSON, el servicio debe ser capaz de interpretarla y traducirla a objetos Java.
- Utilizar los datos de la Webhook Request para obtener el nombre y/o la cadena de hoteles que resulta de interés para el usuario.
- Realizar una consulta a la base de datos sobre las opiniones relacionadas con el hotel o cadena de hoteles pertinente.
- Encapsular el resultado de la consulta en una respuesta o Webhook Response [20], en formato JSON, y enviarla al agente de Dialogflow.

La Figura 3-3 muestra un esquema de los módulos que se diseñaron como parte del servicio web, así como la interacción entre ellos:

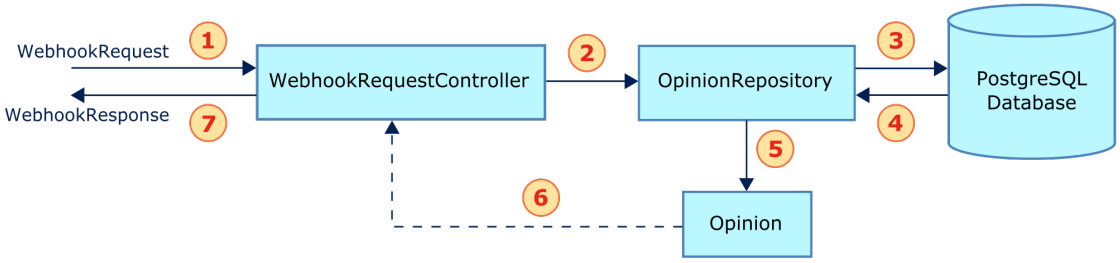


Figura 3-3: Componentes principales de hotel-review-provider.

Para solventar el problema de recepción de solicitudes, se definió un controlador REST [8] denominado WebhookRequestController, que se asociaría a una determinada ruta. Cuando llegaran peticiones desde Dialogflow a esta ruta, el controlador las interceptaría y traduciría su contenido a objetos Java. Para ello, se utilizarían mecanismos incorporados en Spring Boot (como librerías, por ejemplo) capaces de realizar *mappings* de datos JSON a objetos en este lenguaje.

Una vez obtenidos los objetos Java de interés, era necesario ejecutar consultas sobre la base de datos y almacenar el resultado. Spring Boot ofrecía la posibilidad de definir dos tipos de objetos muy útiles para este fin [43]:

- **Entidades.** Modelan el contenido de una tabla o el resultado de una consulta a una base de datos.
- **Repositorios.** Se encargan de configurar la conexión a la base de datos, realizar consultas y transformar el resultado a las entidades presentes en el servicio web.

Teniendo esta información en cuenta, se diseñó la entidad Opinion, que representaría las opiniones asociadas a hoteles, así como un repositorio denominado OpinionRepository, que accedería a la base de datos definida en el apartado anterior y traduciría el resultado a objetos de tipo Opinion.

Finalmente, el servicio web utilizaría una serie de objetos Java para modelar una respuesta o Webhook Request, y haría uso del mismo mecanismo de Spring Boot para convertirla a formato JSON. A través del controlador REST, la respuesta se enviaría de vuelta a Dialogflow.

3.3 Diseño en la segunda iteración

3.3.1 Agente de Dialogflow: PreferenceCollector

En la segunda iteración, el objetivo principal consistió en dotar al agente con mecanismos para identificar cuándo el usuario deseaba obtener hoteles acordes a sus preferencias. El Anexo A recoge el diagrama de flujo que se diseñó en esta fase, con los posibles *inputs* procedentes del usuario y su gestión por parte del agente.

A la hora de identificar palabras clave, debía tenerse en cuenta la siguiente especificación: una preferencia consiste en un aspecto o en un aspecto acompañado de un adjetivo o un rango. Para implementar este requisito en el agente de Dialogflow, se definieron las siguientes palabras clave:

- **Keywords relacionadas con aspectos.** Especifican características del dominio. En este caso, debían hacer referencia a servicios e instalaciones ofrecidos por hoteles, como «*bedroom*», «*wifi*» o «*stars*».
- **Keywords relacionadas con adjetivos.** Califican las expectativas del usuario hacia un aspecto determinado. Dentro de la categoría de los adjetivos, existen diferencias en la intensidad con la que se pretende expresar un concepto: por ejemplo, un usuario indica un mayor grado de exigencia sobre la comida de un hotel si solicita «*excellent food*» en vez de «*decent food*». Por tanto, surgió la necesidad de identificar adjetivos como «*excellent*», «*nice*» o «*decent*», así como de asociarlos a una clasificación numérica de 1 a 4. A modo de ranking, los números más pequeños hacen referencia a una mejor calidad de los servicios e instalaciones de un hotel; los más altos expresan un descenso en esta calidad.
- **Keywords relacionadas con rangos.** Durante la conversación, un usuario podría solicitar hoteles con un precio por noche menor a una cantidad especificada. De este modo, Dialogflow debía identificar expresiones como «*above*», «*less than*» o «*higher or equal*», y asociarlas a los términos "mayor que", "menor que", "mayor o igual que" y "menor o igual que".
- **Keywords relacionadas con una preferencia completa.** Dado que una preferencia consiste en un aspecto o un aspecto acompañado de un adjetivo o un rango, era necesario englobar las *keywords* de los apartados anteriores en una única expresión para conservar la correspondencia entre aspectos y adjetivos/rangos. Por tanto, se definió un tipo de palabra clave que almacenaba combinaciones de otras *keywords* siguiendo el requisito inicial. Entre sus diferentes valores se encontrarían «*great wifi*», «*spotless bedroom*» o «*price below 120€*».

De igual modo que en la primera iteración, las palabras clave podrían aparecer en mensajes escritos de formas distintas, con lo que se establecieron una serie de frases de entrenamiento que el agente debía reconocer. La Tabla 3-2 muestra algunos ejemplos:

Frases de entrenamiento	
Tipo	**Contenido**
Buscar hoteles	«*Get hotels with more than 3 stars*»
Buscar hoteles	«*Show me hotels with gym and great wifi*»
Buscar hoteles	«*Search for hotels that have decent food, and cost below 90 euros/night*»

Tabla 3-2: Frases de entrenamiento del agente PreferenceCollector.

Cuando el agente recibiese una solicitud para encontrar hoteles acordes a ciertas preferencias, interaccionaría con el servicio web y devolvería tanto los aspectos solicitados por el usuario como una lista con los resultados. Si la lista está vacía, el agente ofrecería realizar una nueva búsqueda. En caso contrario, esperaría al *input* del usuario sobre su satisfacción con los resultados. Si el usuario expresa su agrado con expresiones definidas en el apartado 3.2.1, el agente concluiría la conversación; si no, permitiría continuar la búsqueda actual o realizar una nueva. Tanto en el caso de continuar búsquedas como comenzar otras desde cero, el usuario podría aceptar o rechazar ambas propuestas.

Finalmente, en esta iteración también se definieron mecanismos para redirigir el diálogo si el agente recibía *input* anómalo por parte del usuario (mensajes o peticiones fuera de ámbito).

3.3.2 Base de datos con aspectos y preferencias de usuario

Además de los requisitos expuestos en el apartado 3.2.2, surgió la necesidad de añadir nuevos elementos en la base de datos, descritos a continuación:

- Un hotel puede tener asociado entre 1 y N aspectos distintos. Cada uno de ellos posee un valor, que puede ser de dos tipos:
  - Puntuación o ranking numérico entre 1 y 4, siendo 1 la categoría más alta y 4 la más baja.
  - Valor real de un aspecto, como el precio del hotel o su número de estrellas.
- Cada aspecto posee un identificador único y un nombre. Un aspecto individual puede asociarse a M hoteles distintos.
- Un usuario puede tener entre 0 y N preferencias sobre aspectos distintos. Cada preferencia posee un rango (opcional), un valor y un peso. El valor puede ser un ranking numérico de 1 a 4, un precio o un número de estrellas; todos ellos son estimados por el usuario. Por su parte, los pesos determinan su interés hacia determinados aspectos.
- Cada usuario se diferencia de los demás mediante un identificador único. Un número M de usuarios pueden expresar su afinidad por un aspecto individual.

La Figura 3-4 muestra los nuevos requisitos traducidos a un diagrama ER.

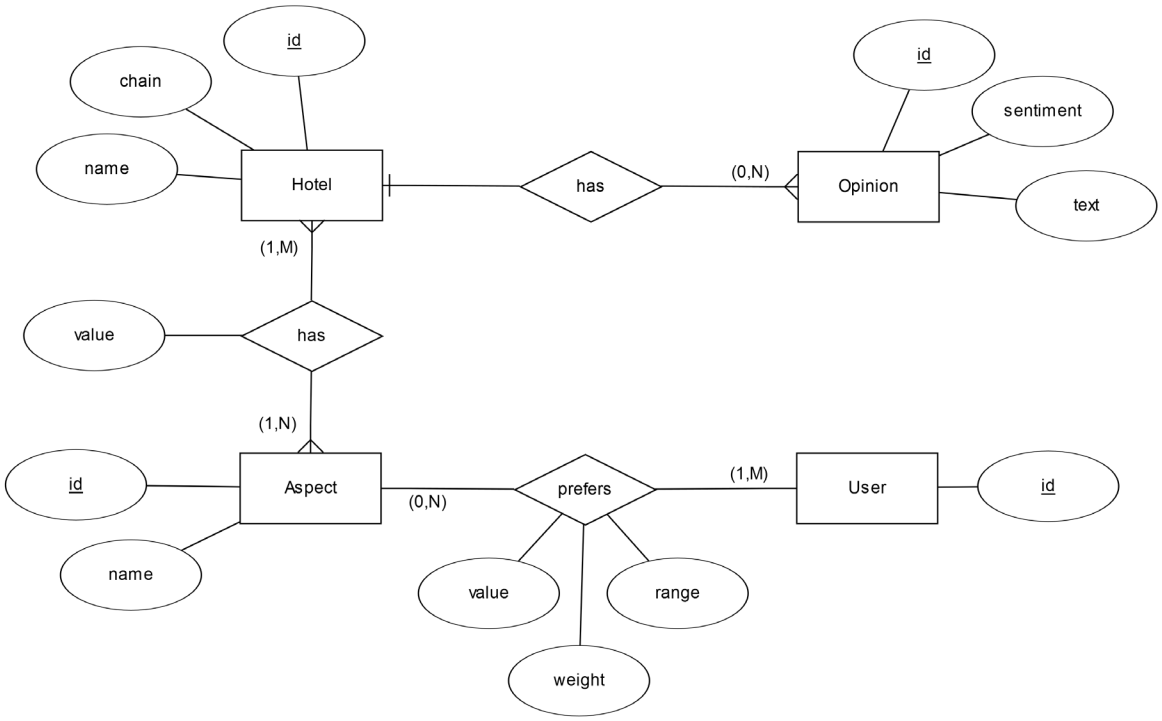


Figura 3-4: Diagrama ER completo.

3.3.3 Servicio web: preference-collector

Durante la segunda iteración, el servicio web debía apoyarse sobre los mecanismos implementados en el apartado 3.2.3 para realizar las nuevas tareas descritas a continuación:

- Una vez recibida la Webhook Request y traducida a objetos Java, se debía identificar de forma única al usuario con el que se estaba conversando en ese momento. De este modo, el sistema sería capaz de asignar y recuperar sus preferencias de forma precisa.
- Además de identificar al usuario, el servicio web debía obtener todas las preferencias contenidas en una solicitud. A este requisito se le añade el hecho de que existen varios tipos de preferencias que un usuario puede expresar. Según este tipo, se recogería un valor distinto para cada preferencia. La Tabla 3-3 muestra las posibles variaciones y el valor resultante que se asocia al aspecto (ranking de 1 a 4, precio o número estrellas).
- Por otro lado, el sistema debía asignar pesos a las preferencias para registrar posibles cambios en los intereses del usuario.
- Finalmente, se requería mantener una lista de aspectos con el fin de comprobar si cualquier preferencia realizada por el usuario se ajustaba al ámbito del dominio o no.

Preferencias de usuario			
Tipo	**Contenido**	**Aspecto**	**Valor**
Aspecto individual	«*Get hotels with gym*»	«*gym*»	3
Aspecto y adjetivo	«*Show me hotels with spotless bathrooms*»	«*bathrooms*»	1
Rango de precio	«*Search for hotels below 90€ per night*»	«*price*»	< 90
Rango de estrellas	«*Display hotels with more than 3 stars*»	«*stars*»	> 3
Estrellas (individual)	«*I want 5-star hotels*»	«*stars*»	5

Tabla 3-3: Tipos de preferencias expresadas por el usuario.

Dentro de los posibles tipos de preferencias que podría formular un usuario, a los aspectos que no van acompañados de adjetivos se les asignaría por defecto la puntuación 3 sobre 4. En cuanto a los rangos, sólo el precio y las estrellas de un hotel podrían expresarse de este modo. Su valor final incluiría el símbolo apropiado de entre los siguientes: >, >=, < o <=.

Todos los requisitos expuestos anteriormente se tradujeron a una serie de módulos nuevos, mostrados en la Figura 3-5:

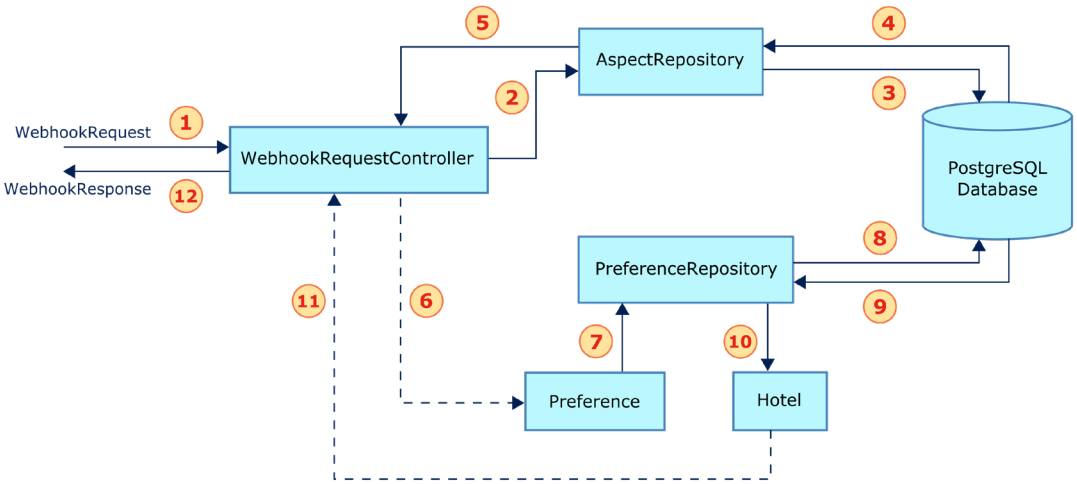


Figura 3-5: Componentes principales de preference-collector.

Del mismo modo que en el apartado 3.2.3, el controlador REST denominado como WebhookRequestController recibiría peticiones desde Dialogflow. La información de interés contenida en ellas se transformaría a objetos Java a través de mecanismos auxiliares como librerías.

En este caso, si el usuario expresa sus preferencias sobre determinados aspectos, el servicio web comprobaría primero si dichos aspectos se ajustan al ámbito del dominio. A través del repositorio AspectRepository, realizaría una consulta a la base de datos, de la que obtendría los nombres de todos los aspectos definidos hasta el momento.

A continuación, el servicio web analizaría cada preferencia del usuario y la clasificaría según los tipos recogidos en la Tabla 3-3. Una vez identificado su tipo, crearía un nuevo objeto de tipo Preference con el identificador, el nombre y el valor correspondiente al aspecto (ranking 1-4 o valor numérico de precio/estrellas). Cada objeto Preference se añadiría a una lista, que podría estar vacía o contener elementos de consultas previas.

El repositorio PreferenceRepository utilizaría la lista de preferencias para recuperar hoteles de la base de datos cuyos aspectos cumplan con los requisitos especificados. Cada uno de ellos se modelaría como un objeto de tipo Hotel, con su correspondiente identificador, nombre y la cadena a la que pertenece; y se insertaría en una nueva lista.

Finalmente, la lista completa de hoteles se devolvería a WebhookRequestController, que a su vez encapsularía esta información en una respuesta en formato JSON. Al igual que en el apartado 3.2.3, esta respuesta se enviaría de vuelta a Dialogflow.

3.3.4 Herramientas complementarias

Dado que el sistema conversacional debía adaptarse a diversos dominios sin que fuese necesario su completo rediseño, se definieron dos herramientas encargadas de facilitar la agregación de keywords y frases de entrenamiento en el agente de Dialogflow. Estas utilidades recibieron el nombre de **aspectItemsParser** y **training-phrase-uploader**, respectivamente.

La herramienta aspectItemsParser debía recibir ficheros CSV con vocabulario de aspectos, procedentes del Information Retrieval Group (UAM) [25]. A continuación, procesaría los ficheros para generar un CSV de salida con la estructura especificada por Dialogflow [17]. De este modo, el usuario podría importar su contenido en el agente a través de la consola de Dialogflow.

Por otro lado, la aplicación training-phrase-uploader debía procesar un fichero con frases de entrenamiento asociadas a *intents*, y comunicarse con Dialogflow para añadir estas frases en el agente. En este caso, sería necesario utilizar la API de Dialogflow y obtener un token de acceso que autorizase las operaciones realizadas por training-phrase-uploader. Esta problemática y su solución se abordan en el apartado de desarrollo del proyecto.

4 Desarrollo

Tal y como se ha expuesto en el apartado anterior, los procesos de diseño y desarrollo se estructuraron en dos iteraciones con objetivos definidos. En las secciones siguientes se describirá la implementación del agente de Dialogflow, la base de datos y el servicio web para cumplir con los requisitos de búsqueda de opiniones y recomendación de hoteles en base a preferencias del usuario.

4.1 Desarrollo en la primera iteración

4.1.1 Agente de Dialogflow: HotelReviewProvider

En primer lugar, se realizó un estudio de todos los agentes predeterminados que ofrecía Dialogflow como punto de partida para desarrolladores. Cada uno de ellos incorporaba una configuración sencilla, adaptada a las diferentes necesidades de los clientes (ofrecer noticias, recordatorios, realizar operaciones bancarias, etc.). En base a este análisis, se implementaron todos los componentes de Dialogflow [16] necesarios para modelar un agente capaz de ofrecer opiniones sobre hoteles:

- Tipos de entidades o *entity types*.
- *Intents*, frases de entrenamiento, *Input/Output contexts* y flujo de conversación.
- Comunicación con el servicio web.
- Integración en aplicaciones de mensajería (Telegram).

4.1.1.1 Tipos de entidades

Durante la primera iteración se definieron entidades de mapeo, compuestas por una *keyword* y una lista de sinónimos asociados a ella. Todas las entidades se listan a continuación:

- **@hotel-chain.** Corresponde a la cadena a la que pertenecen los hoteles. Se definieron dos entidades de este tipo: «*AC Hotels*» y «*Hilton Hotels*».
- **@hotel-name.** Corresponde al nombre del hotel. Por ejemplo, «*Asheville Downtown*».
- **@opinion.** Representa las distintas palabras con las que un usuario puede referirse al concepto de opinión: «*impression*», «*thoughts*», etc.
- **@opinion-sentiment.** Corresponde a los distintos tipos de opiniones que puede solicitar un usuario: «*Positive*», «*Negative*» o «*All*» (ambos tipos).
- **@result.** Representa la respuesta de un usuario ante las opiniones recibidas: si está satisfecho con ellas o no le han resultado útiles.

4.1.1.2 Intents

La Figura 4-1 muestra un esquema de todos los *intents* que se definieron en la primera iteración, así como las relaciones entre ellos. Cada *intent* lleva asociados ciertos sub-componentes definidos en la sección 2.3, que se describirán con detalle en los siguientes apartados.

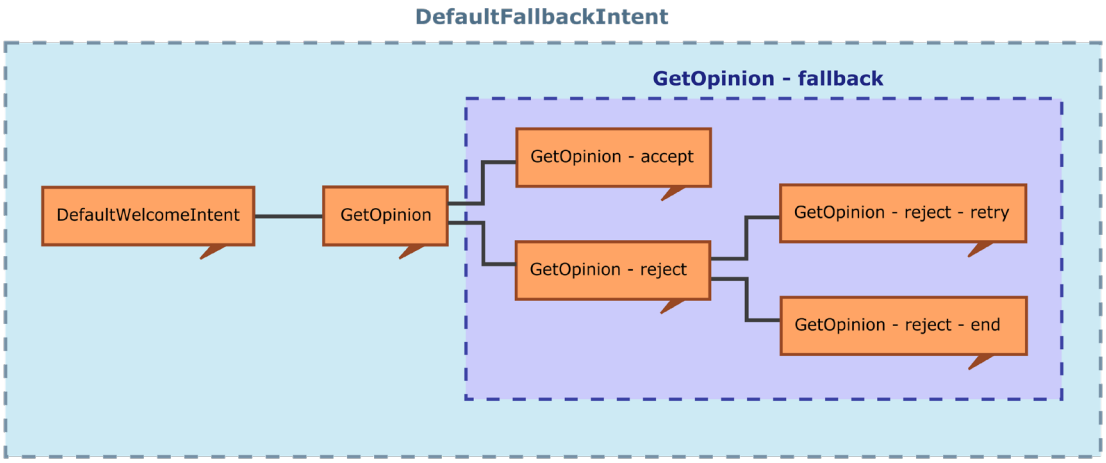


Figura 4-1: *Intents* de HotelReviewProvider.

DefaultWelcomeIntent

Este *intent* se lanza cuando el usuario realiza cualquiera de las siguientes acciones:

- Iniciar el bot de Telegram con el comando `/start`, que activa el *intent* mediante un evento asociado al mismo.
- Escribir un mensaje de saludo como «*hello*» o «*hi there*», que activa el *intent* cuando el texto del usuario coincide con alguna de las frases de entrenamiento asociadas. Todas ellas son expresiones de saludo en inglés o ciertos emoticonos.

Como consecuencia de activar el *intent*, Dialogflow responde con este mensaje:

```
"Welcome! I can search for hotels that suit your needs, as well as hotel
opinions. What do you want me to search about?"
```

DefaultFallbackIntent

Este *intent* se activa cuando el usuario envía un mensaje fuera de ámbito durante la conversación (por ejemplo, «*I want to book an hotel*»). En este caso, Dialogflow envía la siguiente respuesta para redirigir el diálogo hacia *inputs* del usuario conocidos:

```
"Sorry, which kind of hotel or opinions about hotels do you want to
search for?"
```

GetOpinion

Si el usuario solicita opiniones sobre hoteles o cadenas de hoteles, Dialogflow activará GetOpinion. Sus frases de entrenamiento están diseñadas para identificar este tipo de consultas (ver Tabla 3-1), y tienen etiquetas asociadas para capturar las entidades de interés: **@opinion**, **@hotel-name** y **@hotel-chain**. El valor de **@hotel-chain** es obligatorio: si el usuario no proporciona una cadena de hoteles, Dialogflow preguntará por este valor hasta que lo reciba. Sólo entonces concluirá la captura de información.

Por otro lado, GetOpinion activa dos contextos de salida: `GetOpinion-followup` y `GetOpinion-fallback`. Los contextos sirven de transición entre *intents*: los que tengan

como entrada `GetOpinion-followup` o `GetOpinion-fallback` se lanzarán si estos contextos están activos y si el *input* del usuario coincide con las frases de entrenamiento.

Finalmente, GetOpinion almacena los valores de las entidades en parámetros y crea una Webhook Request en formato JSON, donde inserta estos parámetros. A través de la URL del servicio web (ver apartado 4.1.1.3), Dialogflow envía la petición y recibe la lista de opiniones desde el servicio. Esta lista, seguida de la pregunta: `"What do you think?"`, se mostrará como un mensaje en el bot de Telegram.

GetOpinion – accept

Este *intent* es de tipo *follow-up*: para activarse, el *intent* padre GetOpinion debe haberse lanzado previamente. Esto ocurre porque GetOpinion – accept lleva asociado el contexto de entrada `GetOpinion-followup`, que a su vez es el contexto de salida de GetOpinion. De este modo, los contextos sirven para definir las etapas de una conversación.

Si `GetOpinion-followup` está activo (con lo que previamente se han solicitado opiniones sobre hoteles) y el usuario expresa su satisfacción con los resultados, este *input* coincidirá con las frases de entrenamiento asociadas a GetOpinion – accept (ver Tabla 3-1). El *intent* se activará, y almacenará las impresiones del usuario en la entidad **@result**. Por último, el agente responderá con un mensaje predeterminado (`"Glad I could help!"`) y dará por finalizada la conversación.

GetOpinion – reject

Al igual que GetOpinion – accept, este *follow-up intent* depende de GetOpinion, ya que lleva asociado el contexto de entrada `GetOpinion-followup`.

Su funcionamiento es muy similar a GetOpinion – accept, aunque presenta dos diferencias:

- Se lanza cuando `GetOpinion-followup` está activo y el usuario expresa su inconformidad con las opiniones devueltas por el bot de Telegram.
- Este *intent* activa el contexto de salida `GetOpinion-reject-followup`, y permite al usuario realizar una nueva búsqueda con el mensaje: `"Would you like to see other reviews or opinions?"`.

GetOpinion – reject – retry

Este *intent* de tipo *follow-up* se activa cuando se cumplen tres condiciones:

- GetOpinion y GetOpinion – reject se han lanzado previamente.
- El contexto `GetOpinion-reject-followup` está activo, como consecuencia de que el usuario ha mostrado su inconformidad con las opiniones mostradas.
- El usuario, ante el mensaje `"Would you like to see other reviews or opinions?"` responde afirmativamente («*OK*», «*Sure*», etc.). El contexto `GetOpinion-reject-followup` permite al agente determinar que el usuario da su consentimiento para empezar una nueva búsqueda. De lo contrario, un simple «*OK*» en la conversación resultaría ambiguo.

Si se cumplen todas las condiciones, el agente enviará el siguiente mensaje al usuario:

`"Which reviews or opinions would you like to see?"`

Cuando se vuelva a realizar una consulta sobre opiniones de hoteles, Dialogflow activará el *intent* GetOpinion y comenzará otro nuevo diálogo.

GetOpinion – reject – end

Este *follow-up intent* se activa también si se cumplen tres condiciones:

- GetOpinion y GetOpinion – reject se han lanzado previamente.
- El contexto `GetOpinion-reject-followup` está activo, como consecuencia de que el usuario ha mostrado su inconformidad con las opinions mostradas.
- El usuario, ante el mensaje `"Would you like to see other reviews or opinions?"` responde negativamente («*No thanks*», «*Not now*», etc).

Dadas estas circunstancias, el agente responde con un mensaje predeterminado (`"Okay. Have a nice day!"`) y da por finalizada la conversación.

GetOpinionFallback

De un modo similar a DefaultFallbackIntent, GetOpinionFallback se activa si el usuario envía un mensaje fuera de ámbito tras recibir las opiniones sobre hoteles. Además de esta condición, el contexto `GetOpinion-fallback` debe estar activo (es decir, se ha realizado una búsqueda de opiniones previamente).

En este caso, el bot de Telegram mostrará el siguiente mensaje para redirigir la conversación:

```
"Sorry, what do you think about the opinions?"
```

4.1.1.3 Comunicación con el servicio web e integración en Telegram

Para establecer un *endpoint REST* entre el agente de Dialogflow y el servicio web de Spring Boot, se modificó la configuración del agente para incluir la URL https://hotel-review-provider.herokuapp.com como punto de comunicación bidireccional para ambos componentes.

Por otro lado, para integrar el agente en Telegram se siguieron varios pasos. En primer lugar, se creó un nuevo *chatbot* en el gestor de bots *BotFather* [41]. A continuación, se asignó el nombre HotelReviewProvider a dicho *chatbot*, y se utilizó el token de acceso generado por *BotFather* para autorizar la integración del agente en Telegram.

4.1.2 Base de datos de hoteles y opiniones

Una vez definido el agente de Dialogflow, se creó una nueva aplicación en la plataforma Heroku, denominada hotel-review-db-server. Como parte de su funcionalidad se incluyó el *add-on* Heroku Postgres [24], que aloja un administrador de bases de datos PostgreSQL y genera unas credenciales de acceso a la propia BD.

Tras desplegar la aplicación, se configuró el gestor pgAdmin para crear una conexión a la base de datos remota y añadir dos tablas: **hotels** y **hotel_opinions**. Su estructura se muestra a continuación:

Hotels		
hotel_id **[PK]**	hotel_name	hotel_chain

Tabla 4-1: Contenido de hotels.

hotel_opinions			
opinion_id **[PK]**	hotel_id **[FK]**	opinion_sentiment	opinion_text

Tabla 4-2: Contenido de hotel_opinions.

La tabla **hotels** almacena los nombres de los hoteles definidos en el agente de Dialogflow, así como las cadenas a las que pertenecen (*AC Hotels* o *Hilton Hotels*). El ID de cada hotel es un contador que se incrementa según se añaden nuevas entradas en la tabla.

Por su parte, la tabla **hotel_opinions** almacena opiniones relacionadas con los hoteles. El campo *opinion_id* es un contador que identifica de forma única cada opinión, mientras que *hotel_id* se utiliza como clave foránea para asociar una opinión a un hotel concreto. Por último, *opinion_sentiment* y *opinion_text* hacen referencia al tipo (*Positive/Negative/All*) y texto de la opinión, respectivamente.

Tras definir las tablas y la relación que existe entre ellas, se añadieron los mismos nombres y cadenas de hoteles disponibles en el agente, así como una serie de opiniones ficticias.

4.1.3 Servicio web: hotel-review-provider

Antes de implementar el servicio web, se consultó tanto documentación como tutoriales de Heroku [23] y Spring Boot [42] con el objetivo de conocer el funcionamiento de estas herramientas. A continuación, se creó la aplicación hotel-review-provider en Heroku para alojar el servicio web, y se configuró un nuevo proyecto en Eclipse con Maven y el plugin Spring Tools [26], que contendría toda la funcionalidad del servicio.

Sobre este proyecto se añadieron tres ficheros de configuración: un fichero POM [2] que contiene todas las dependencias necesarias para que Maven pueda crear un ejecutable, y los archivos application.properties y system.properties, que configuran elementos como el pool de conexiones HikariCP. El contenido de todos ellos se recoge en el Anexo B.

Tal y como se expuso en el apartado 3.2.3, el servicio debía incluir una serie de módulos principales:

- Clases Java para modelar Webhook Requests y Webhook Responses.
- WebhookRequestController, un controlador REST para interceptar peticiones desde Dialogflow y enviar respuestas.
- Entidades y Repositorios.

La Figura 4-2 muestra la estructura de paquetes del proyecto, que se describirá en las siguientes secciones. La dependencia de todos ellos sobre *application* se debe a que Spring Boot utiliza por defecto este paquete como base para buscar los componentes de un servicio web (aunque es posible crear configuraciones personalizadas).

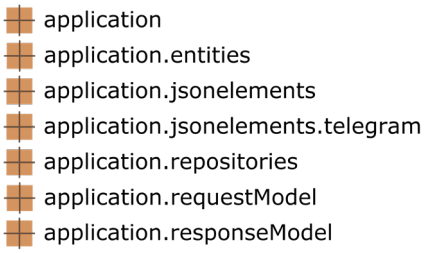


Figura 4-2: Paquetes del servicio web hotel-review-provider.

4.1.3.1 Paquete application

Contiene la clase **Application**, que se encarga de ejecutar la aplicación Spring Boot y de escanear y cargar todos los componentes necesarios: controladores REST, clases Java que modelan una Webhook Request o Webhook Response, entidades, repositorios y servicios.

4.1.3.2 Paquetes application.jsonelements y application.jsonelements.telegram

El paquete *application.jsonelements* contiene las clases que representan los elementos JSON de una Webhook Request y/o una Webhook Response. Todas ellas carecen de constructor e incluyen los atributos especificados en la documentación de Dialogflow [16], así como sus *getters* y *setters*. Además, presentan una serie de anotaciones:

- **@Component.** Se incluye a nivel de definición de clase, e identifica los componentes que pertenecen a una aplicación Spring Boot. Con esta anotación, el framework es capaz de encontrar las clases Java e inicializarlas automáticamente, sin necesidad de un constructor explícito.
- **@JsonInclude.** A nivel de definición de clase, indica que todas sus propiedades deben serializarse/deserializarse. Sin embargo, se ha añadido una restricción adicional para que sólo se consideren los valores distintos de NULL.
- **@JsonIgnoreProperties.** A nivel de definición de clase y con la restricción `ignoreUnknown = true`, indica que debe ignorarse cualquier propiedad que no forme parte de la clase Java. De este modo, se asegura el funcionamiento del servicio web ante definiciones de nuevos campos en la Webhook Request.
- **@JsonProperty.** Indica que el atributo de una clase Java se corresponde con un elemento JSON. Se ha utilizado en *getters* y *setters* para agilizar el proceso de serialización y deserialización de propiedades.

Algunos elementos de la Webhook Request/Response pueden incluir un número variable de pares clave-valor, y no se garantiza que el nombre de las claves sea siempre el mismo. Para solucionar este problema, se tuvo en cuenta la estructura en árbol de los datos JSON y se modeló este tipo de propiedades como objetos JsonNode, propios de la librería Jackson.

Por otro lado, el paquete *application.jsonelements* contiene la clase **Payload**, que define las respuestas personalizadas que pueden enviar los servicios compatibles con Dialogflow [21]. Actualmente se incluye el *payload* propio de Telegram en el paquete *application.jsonelements.telegram*, que modela dos elementos JSON interpretados por esta aplicación:

- Texto formateado en estilo Markdown o HTML.
- *Inline buttons*, o botones que contienen un texto decorativo e información que puede enviarse al agente de Dialogflow.

4.1.3.3 Paquete application.entities

Incluye la clase **Opinion**, que define una serie de propiedades, *getters* y *setters*. A diferencia de las clases que modelan una Webhook Request/Response, Opinion se ha diseñado como una entidad: posee un constructor con argumentos y atributos que representan cada columna de la tabla hotel_opinions (ver Tabla 4-2).

4.1.3.4 Paquete application.repositories

Este paquete contiene la interfaz **OpinionRepository** y una clase que la implementa: **JDBCOpinionRepository**. La interfaz modela un repositorio u objeto de acceso a la base de datos, y su objetivo es abstraer los detalles de implementación de estas operaciones. Define tres métodos:

- **count().** Devuelve el número de registros contenidos en hotel_opinions.
- **findAll().** Devuelve todas las opiniones de la tabla hotel_opinions.
- **findByHotelInfo().** Devuelve opiniones asociadas a un hotel concreto o a una cadena de hoteles. Las opiniones pueden ser positivas, negativas o de ambos tipos.

La clase **JDBCOpinionRepository** implementa estos métodos a través de *JdbcTemplate*, un objeto de Spring Boot que hace uso del driver JDBC y la pool de conexiones HikariCP (configurada en el Anexo B) para automatizar el acceso a la base de datos y la ejecución de consultas. De este modo, el código de los métodos se simplifica considerablemente, ya que no es necesario manejar conexiones de forma explícita.

Una vez realizadas las consultas, se utiliza el objeto *RowMapper* de Spring Boot para convertir los registros de la tabla hotel_opinions a objetos de tipo Opinion.

4.1.3.5 Paquete application.requestModel

Este paquete contiene las clases **WebhookRequest**, **WebhookRequestController** y **ResponseGeneratorService**. La primera representa las peticiones que envía el agente, y contiene todas las anotaciones y tipos de datos definidos en los paquetes *application.jsonelements* y *application.jsonelements.telegram*.

Por otro lado, ResponseGeneratorService modela un servicio que abstrae los accesos a la base de datos a través del repositorio OpinionRepository. De este modo, se realiza un *separation of concerns* entre el controlador REST y el manejo de la base de datos.

ResponseGeneratorService define tres métodos:

- **generateWebhookResponse().** Crea una nueva Webhook Response con los datos especificados.
- **checkRequestParameters().** Dada una Webhook Request recibida como argumento, comprueba si el *intent* del que proviene es GetOpinion. Si esto ocurre, invoca a getOpinionsFromHotel() para recuperar las opiniones solicitadas. En caso contrario, genera una Webhook Response predefinida (*placeholder*).
- **getOpinionsFromHotel().** Dado un hotel específico o una cadena de hoteles, sumado a un tipo de opinión, genera un *payload* de Telegram con dos elementos:
  - Una imagen distinta en función de la cadena (AC Hotels o Hilton Hotels).
  - El texto de las opiniones que se han recuperado, en formato Markdown.

  El *payload* personalizado se encapsula en una Webhook Response que se envía de vuelta al agente, quien muestra la imagen y el texto al usuario.

Finalmente, el controlador REST **WebhookRequestController** recibe solicitudes del agente en la URL base hotel-review-provider.herokuapp.com (*endpoint* "/"), e invoca al método checkRequestParameters() del servicio para generar una respuesta. A través de la anotación @ResponseBody de Spring Boot, la respuesta se serializa en una estructura JSON y se envía al agente a través del protocolo HTTP automáticamente.

4.1.3.6 Paquete application.responseModel

Del mismo modo que con WebhookRequest, se definió la clase WebhookResponse para modelar las respuestas que se devolverían al agente. Esta clase comparte anotaciones Jackson y atributos propios de las peticiones, además de contener elementos específicos.

4.2 Desarrollo en la segunda iteración

Una vez concluida la primera iteración, el agente HotelReviewProvider, la aplicación hotel-review-db-server y el servicio web hotel-review-provider se congelaron en una versión estable con la funcionalidad desarrollada durante esa etapa. Sobre ellos se crearon nuevos componentes, que incluirían tanto el código y la configuración definidos hasta el momento como los cambios añadidos durante la segunda iteración.

4.2.1 Agente de Dialogflow: PreferenceCollector

A partir de HotelReviewProvider se definió un nuevo agente denominado PreferenceCollector, que contaba con las mismas entidades, *intents* y frases de entrenamiento que su homónimo. Dado que el objetivo de esta iteración consistía en buscar hoteles en base a preferencias de usuario, se añadieron nuevos componentes de Dialogflow que se expondrán en las siguientes secciones.

4.2.1.1 Tipos de entidades

Dentro de PreferenceCollector se definieron una serie de entidades de mapeo (*keyword* principal y sinónimos asociados a esta palabra clave), que representaban diversos conceptos:

- **@aspect.** Corresponde a los aspectos pertenecientes al dominio. En el caso de hoteles, algunos aspectos válidos serían «*bedroom*» o «*cafeteria*».
- **@adjetive-ranking.** Corresponde a puntuaciones numéricas de 1 a 4. Cada una de ellas representa las expectativas del usuario sobre un aspecto, y tiene asociada una serie de sinónimos como «*upscale*» (1), «*great*» (2), «*good*» (3), o «*passable*» (4), respectivamente.
- **@range.** Identifica los rangos de precio y estrellas de hoteles que puede solicitar un usuario: «*less than*», «*greater than*», «*greater than or equal to*», y «*less than or equal to*».
- **@stars.** Corresponde al número de estrellas de un hotel, entre 1 y 5.

Además de entidades de mapeo, se añadió una entidad de lista denominada **@preference**. En este caso, en vez de definir sinónimos, incluía los posibles valores que podía asignarle Dialogflow en función del *input* del usuario: un aspecto (@aspect) precedido o seguido por un adjetivo (@adjetive-ranking), un rango (@range) y un precio (indicado por la entidad predefinida @sys:unit-currency) o un rango y un número de estrellas (@stars).

4.2.1.2 Intents

Para implementar los nuevos requisitos en el agente, se definieron una serie de *intents* recogidos en la Figura 4-3. Todos ellos se exponen en las siguientes secciones.

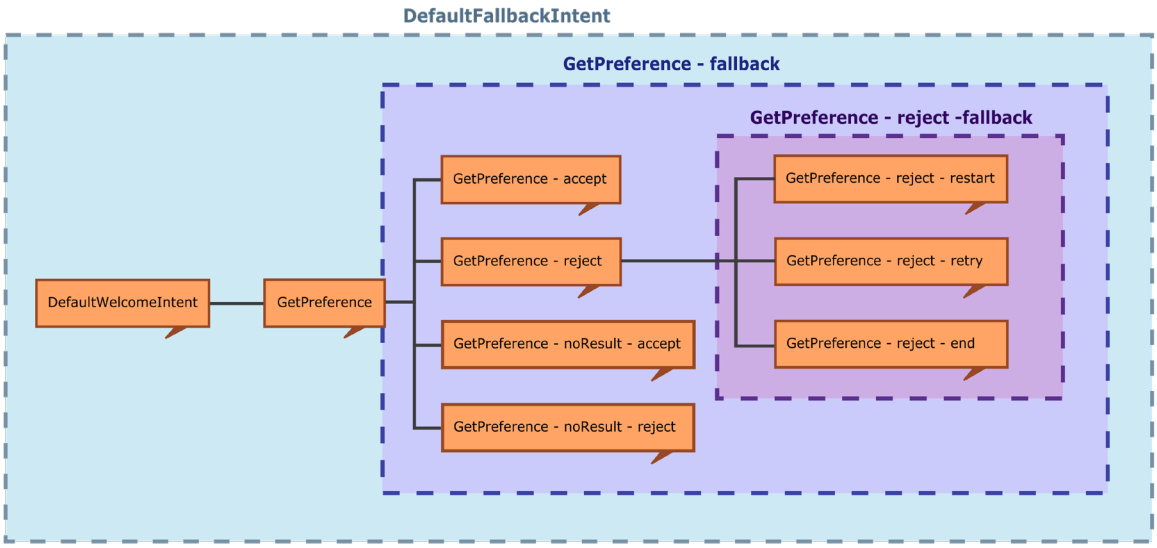


Figura 4-3: *Intents* de PreferenceCollector.

GetPreference

Tras recibir el mensaje de bienvenida como resultado de iniciar el bot (ver apartado 4.1.1.2), Dialogflow activará GetPreference si el usuario solicita hoteles acordes a sus intereses. Este *intent* incluye frases de entrenamiento similares a las mostradas en la Tabla 3-2; todas ellas se han etiquetado para capturar entidades relacionadas con aspectos, adjetivos y/o rangos de precio por los que un usuario podría preguntar.

De forma similar a GetOpinion, el agente almacena el valor de las entidades en parámetros y activa los contextos de salida `GetPreference-followup` y `GetPreference-fallback`. Ambos contextos servirán para guiar la conversación hacia estados conocidos.

Por último, el agente inserta los parámetros en una petición JSON que envía al servicio web. La respuesta que reciba se mostrará a través del bot de Telegram.

GetPreference – noResult – reject

Este *intent* es de tipo *follow-up*: posee el contexto de entrada `GetPreference-followup`, que sólo se activa si GetPreference se ha lanzado previamente. Es decir, si el usuario ha solicitado una búsqueda de hoteles.

Durante la búsqueda es posible que el servicio web no encuentre resultados acordes a las preferencias del usuario. En este caso, el agente mostrará la siguiente respuesta:

`"I didn't find any hotels. :( Would you like to try another search?"`

Si el usuario responde negativamente («*No thanks*», «*Not now*», etc.), el bot devolverá un mensaje predeterminado (`"Okay. Have a nice day!"`) y finalizará la conversación.

GetPreference – noResult – accept

Este *follow-up intent* posee una estructura muy similar a GetPreference – noResult – reject. Incluye el contexto de entrada `GetPreference-followup`, que se activa si el usuario ha efectuado una búsqueda de hoteles previamente.

A diferencia de GetPreference – noResult – reject, este *intent* se lanza cuando el usuario responde afirmativamente («*Yes*», «*Sure*», etc.) a la propuesta de realizar una nueva búsqueda de hoteles. En ese momento, el agente envía una petición al servicio web para eliminar las preferencias actuales del usuario y comenzar una consulta desde cero.

GetPreference – accept

GetPreference – accept es un *follow-up intent* que lleva asociado el contexto de entrada `GetPreference-followup`. Por tanto, se lanza cuando se cumplen dos condiciones:

- Previamente, el usuario ha solicitado realizar una búsqueda de hoteles (momento en que se activan GetPreference y sus dos contextos).
- El usuario expresa su satisfacción con los resultados obtenidos (por ejemplo, «*I like this*» o «*This is useful*»).

Si se verifican ambas condiciones, el agente activará el *intent* y mostrará un mensaje predeterminado (`"Glad I could help!"`). Por último, finalizará la conversación.

GetPreference – reject

De forma similar a GetPreference – accept, este *follow-up intent* tiene asociado el contexto de entrada `GetPreference-followup` y se lanza cuando se cumplen dos condiciones:

- Previamente, el usuario ha realizado una búsqueda de hoteles.
- Ante los resultados obtenidos, el usuario expresa su rechazo (por ejemplo, «*I don't like this*»).

Si se verifican ambas condiciones, Dialogflow activará el *intent* y dos contextos de salida: `GetPreference-reject-followup` y `GetPreference-reject-fallback`. Como el usuario no ha quedado satisfecho con los resultados, el bot de Telegram propondrá tres alternativas:

```
"What sounds best to you: continue searching, starting over, or finishing?"
```

En este momento, el usuario puede seguir añadiendo preferencias a su búsqueda, comenzar una nueva o terminar. Cada opción disponible lanzará *intents* diferentes.

GetPreferenceFallback

Tras realizar una búsqueda de hoteles, Dialogflow activará GetPreferenceFallback si el usuario no expresa su satisfacción o rechazo hacia los resultados recibidos. Este *intent* posee el contexto de entrada `GetPreference-fallback` (con lo que GetPreference debe haberse activado previamente), y mostrará el siguiente mensaje a través del bot de Telegram para reconducir la conversación:

```
"Sorry, I didn't get that. Can you try again?"
```

GetPreference – reject – restart

Partiendo de la situación en que se ha solicitado una búsqueda de hoteles previamente, GetPreference – reject – restart entra en juego si se cumple lo siguiente:

- El usuario ha mostrado su insatisfacción con los resultados, con lo que el contexto de entrada de este *follow-up intent* (`GetPreference-reject-followup`) se encuentra activo.
- Ante las tres acciones propuestas por el bot (continuar, volver a empezar o finalizar), el usuario elige volver a empezar con mensajes como «*Start over*».

Si se cumplen las condiciones, Dialogflow activará GetPreference – reject – restart. Este *intent* enviará una Webhook Request al servicio web para eliminar las preferencias del usuario de la base de datos, y propondrá la realización de una nueva búsqueda mediante el siguiente mensaje:

```
"What kind of hotels would you like to see?"
```

GetPreference – reject – retry

Este *follow-up intent* es muy similar a GetPreference – reject – restart, aunque difiere en una de las condiciones que lo activan: ante las tres acciones propuestas por el bot (continuar, volver a empezar o finalizar), el usuario escoge volver a empezar. Un ejemplo de mensaje que puede coincidir con las frases de entrenamiento del *intent* sería: «*I want to continue searching*».

Al lanzarse GetPreference – reject – retry las preferencias se mantienen en la base de datos, y el agente responde con un mensaje para seguir recabando información:

```
"Alright. What else are you interested in?"
```

GetPreference – reject – end

GetPreference – reject – end es un *follow-up intent* muy similar a los dos anteriores, pero cubre el caso en que el usuario decide concluir la búsqueda con *input* como «*I want to finish*».

Cuando se lanza GetPreference – reject – end, el agente responde con un mensaje predeterminado (`"Okay. Have a nice day!"`) y finaliza la conversación.

GetPreference – reject – fallback

GetPreference – reject – fallback se encarga de gestionar *input* anómalo cuando el agente ofrece reanudar, restablecer o finalizar una búsqueda previa. Posee el contexto de entrada `GetPreference-reject-fallback`, con lo que sólo se activará en caso de que el usuario haya solicitado una búsqueda de hoteles y no esté satisfecho con los resultados.

Cuando se lanza este *intent* como consecuencia de una respuesta fuera de ámbito, el agente responde de la siguiente manera para redirigir la conversación:

```
"Sorry, would you like to continue searching, start over, or finish?"
```

4.2.1.3 Comunicación con el servicio web e integración en Telegram

Para comunicar el agente de Dialogflow con el nuevo servicio web, descrito en el apartado 4.2.3, se incluyó la URL https://preference-collector.herokuapp.com como *endpoint* REST entre ambos componentes.

La integración del agente en Telegram consistió en los mismos pasos expuestos en el apartado 4.1.1.3. La única diferencia con respecto a este proceso fue el nombre que se asignó al nuevo bot de Telegram: PreferenceCollector.

4.2.2 Base de datos con aspectos y preferencias del usuario

Del mismo modo que con el agente de Dialogflow, se creó una nueva aplicación de Heroku denominada preference-collector-db-server, que incluía el *add-on* Heroku Postgres [24] y las tablas descritas en el apartado 4.1.2.

Sobre esta versión de la base de datos se definieron nuevas tablas para modelar los aspectos y preferencias del usuario sobre hoteles. Todas ellas se muestran a continuación:

aspects	
aspect_id **[PK]**	aspect_name

Tabla 4-3: Contenido de aspects.

hotel_aspects		
hotel_id **[FK]**	aspect_id **[FK]**	aspect_value

Tabla 4-4: Contenido de hotel_aspects.

user_preferences				
aspect_id **[FK]**	user_id **[PK]**	preference_value	preference_range	preference_weight

Tabla 4-5: Contenido de user_preferences.

La tabla **aspects** define todos los aspectos relacionados con el dominio. Cada aspecto posee un nombre distinto, y se identifica de manera única a través de *aspect_id*, un contador que se incrementa según se añaden entradas a la tabla.

Por su parte, **hotel_aspects** almacena los aspectos asociados a diversos hoteles. Los registros de la tabla se distinguen entre sí mediante distintas combinaciones de *hotel_id* y *aspect_id*. Estos campos referencian el hotel y el aspecto que posee dicho hotel, respectivamente. El campo *aspect_value* puede almacenar una clasificación numérica entre 1 y 4, el precio del hotel o su número de estrellas.

Finalmente, **user_preferences** modela las preferencias del usuario sobre aspectos específicos. Cada registro de la tabla posee un *user_id* o identificador único de usuario, que se corresponde con el campo Session ID de la Webhook Request. Este ID de sesión se genera en el agente de Dialogflow cada vez que un usuario distinto interacciona con él. Por otro lado, *aspect_id* referencia el aspecto de interés para el usuario, *preference_value* almacena el ranking de 1 a 4, precio o número de estrellas que el agente recibe como *input*;

preference_range contiene los símbolos “>”, “<”, “>=”, “<=” o una cadena vacía si el usuario no especifica rangos; y *preference_weight* es un entero que se incrementa cuando el usuario menciona una o más veces un aspecto determinado.

4.2.3 Servicio web: preference-collector

Tras implementar la funcionalidad del agente y las nuevas tablas de la base de datos, se creó un servicio web denominado preference-collector. Este servicio, además de contener todos los componentes desarrollados para hotel-review-provider, incorpora funcionalidad adicional expuesta en el apartado 3.3.3:

- Controlador REST WebhookRequestController, capaz de capturar campos de la Webhook Request como Session ID (para identificar de manera única al usuario), y las preferencias especificadas.
- Entidades y repositorios para manejar las preferencias del usuario y obtener los hoteles que mejor se adapten a sus intereses.
- Lógica adicional para determinar los aspectos sobre los que el usuario expresa su interés, así como los valores asociados (puntuación numérica de 1 a 4, precio o número de estrellas).

Sobre la estructura original de paquetes, se añadieron nuevas clases Java y otro paquete adicional, denominado *application.utils*. La Figura 4-4 muestra la estructura completa del proyecto:

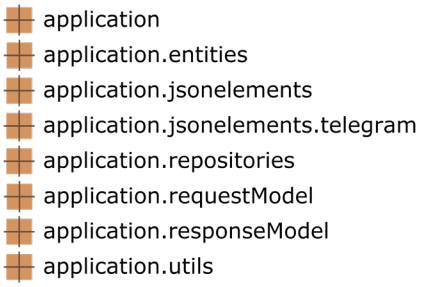


Figura 4-4: Paquetes del servicio web preference-collector.

Los siguientes apartados recogen los paquetes que han sufrido cambios, así como las clases implementadas durante la segunda iteración.

4.2.3.1 Paquete application.entities

En *application.entities* se definieron tres nuevas clases: **Aspect**, **Hotel** y **Preference**. Las dos primeras incluyen un constructor y propiedades que modelan las tablas aspects y hotels, respectivamente (ver Tabla 4-1 y Tabla 4-3). Todas las propiedades tienen asociadas sus correspondientes *getters* y *setters*.

Por su parte, la clase Preference modela los registros obtenidos al realizar consultas a la base de datos sobre preferencias del usuario. Sus propiedades incluyen tanto el contenido de la tabla user_preferences (ver Tabla 4-5) como el campo aspect_name de la Tabla 4-3, así como un constructor y *getters*/*setters*.

4.2.3.2 Paquete application.repositories

En este paquete se añadieron las interfaces **AspectRepository** y **PreferenceRepository**, así como dos clases encargadas de implementar sus métodos: **JDBCAspectRepository** y **JDBCPreferenceRepository**.

AspectRepository abstrae los accesos a la base de datos para obtener información de la tabla aspects. Define cuatro métodos:

- **findAll().** Devuelve todos los aspectos de la tabla aspects.
- **findAllNames().** Devuelve los nombres de todos los aspectos presentes en la tabla aspects.
- **findAspectID().** Dado el nombre de un aspecto, obtiene su ID asociado.
- **findAspectName().** Dado el ID de un aspecto, obtiene su nombre correspondiente.

Por otro lado, PreferenceRepository oculta las operaciones sobre la tabla user_preferences y define cuatro métodos:

- **getPreferencesFromUser().** Dado un Session ID, devuelve la lista de preferencias asociadas al usuario correspondiente. Si no encuentra resultados (porque es la primera vez que el usuario hace este tipo de consulta, por ejemplo), devuelve una lista vacía.
- **clearPreferences().** Dado un Session ID, elimina todas las preferencias asociadas al usuario correspondiente.
- **batchInsertPreferences().** Inserta una lista de preferencias de usuario en la tabla user_preferences.
- **getHotelsByUserPreferences().** Dada una lista de preferencias, obtiene todos los hoteles que cumplen con los requisitos especificados.

Del mismo modo que con el resto de repositorios, las clases JDBCAspectRepository y JDBCPreferenceRepository utilizan el driver JDBC, el objeto JdbcTemplate y la pool de conexiones HikariCP. JDBCAspectRepository hace uso de RowMapper en el método findAll() para transformar el resultado de la consulta en una lista de objetos Aspect. En el resto de los casos, devuelve un objeto de tipo String o Integer, o una lista de Strings.

En cuanto a JDBCPreferenceRepository, esta clase utiliza el método batchUpdate() de JdbcTemplate para insertar y eliminar las preferencias del usuario. En la inserción, utiliza el objeto PreparedStatement de Spring Boot para especificar qué campos se van a insertar en user_preferences y en qué orden. También se sirve de RowMapper para obtener las preferencias del usuario, así como los hoteles que cumplen con sus expectativas.

4.2.3.3 Paquete application.requestModel

Dentro de *application.requestModel* se modificó el servicio **ResponseGeneratorService** y se añadió otro nuevo: **PreferenceEvaluatorService**.

A continuación, se listan los cambios realizados sobre ResponseGeneratorService:

- **checkRequestParameters().** Dada una Webhook Request recibida como argumento, comprueba si el *intent* del que proviene es GetOpinion, GetPreference, GetPreference – noResult o GetPreference – reject – restart. El caso de GetOpinion se recoge en el apartado 4.1.3.5. Si el *intent* es GetPreference, obtiene las preferencias del usuario a través de PreferenceEvaluatorService, e invoca al método

getResultsByUserPreferences(). En los dos últimos casos, elimina las preferencias del usuario e invoca a getMessagesForNewPreferenceSearch().

- **getResultsByUserPreferences().** Este método, definido durante la segunda iteración, hace uso de PreferenceRepository para obtener hoteles acordes a los intereses del usuario. Las preferencias y la lista de hoteles se encapsulan en un *payload* de Telegram, que a su vez se inserta en una Webhook Response dirigida al agente. Si PreferenceRepository no encuentra hoteles, el *payload* contendrá un mensaje informativo y una propuesta para comenzar una nueva búsqueda.
- **getMessagesForNewPreferenceSearch().** Este nuevo método se invoca cuando GetPreference – noResult – accept o GetPreference – reject – restart están activos, momento en que el agente ha propuesto al usuario realizar una nueva búsqueda y éste ha accedido. Por tanto, crea un *payload* de Telegram con mensajes predeterminados para preguntar al usuario sobre sus intereses.

Por otro lado, PreferenceEvaluatorService se creó con el fin de modularizar el controlador REST abstrayendo los procesos de análisis de preferencias y consultas a la base de datos. Define una serie de métodos:

- **initializeAspectList().** Obtiene todos los aspectos relacionados con el dominio desde la base de datos.
- **getUserID().** Obtiene el Session ID que identifica a un usuario de forma única.
- **updateUserPreferences().** Este método invoca a getUserID() y hace uso de PreferenceRepository para obtener todas las preferencias del usuario (o una lista vacía si es la primera vez que se realiza una búsqueda). A continuación, analiza los parámetros de la Webhook Request para determinar qué *keywords* ha detectado el agente de Dialogflow (ver apartado 4.2.1.1): aspectos individuales (**@aspects**), número de estrellas (**@stars**), o preferencias (**@preference**). En cada caso, extrae el nombre de los aspectos y sus valores asociados. Si un aspecto va acompañado de un rango, invoca a parseTextualRange() para obtener el símbolo correspondiente. Todas las preferencias se añaden a una lista a través de llamadas a updatePreferenceList(). Tras actualizar los intereses del usuario, el método registra los cambios en la base de datos.
- **parseTextualRange().** Convierte las *keywords* de la entidad **@range** (ver apartado 4.2.1.1) en los símbolos “<”, “>”, “<=” o “>=”.
- **updatePreferenceList().** Este método se invoca cada vez que el servicio detecta preferencias en los parámetros de la Webhook Request. Si el usuario ha especificado el mismo aspecto más de una vez, se actualiza su valor asociado y se incrementa el peso de la preferencia. En caso contrario, se crea un objeto de tipo Preference y se añade a la lista de intereses del usuario.

4.2.3.4 Paquete application.utils

Este paquete define una serie de enumeraciones con el objetivo de conseguir mayor generalidad y mantenibilidad en el código fuente. Estas enumeraciones ocultan los nombres de columnas y tablas de la base de datos, cadenas de hoteles, tipos de opiniones y entidades e *intents* pertenecientes al agente de Dialogflow. De este modo, se evita el uso de *hard-coded data* y se reduce el número de cambios que sería necesario realizar al incorporar un dominio distinto (como películas, por ejemplo).

4.2.4 Herramientas complementarias

Tal y como se expuso en el apartado 3.3.4, se diseñaron dos utilidades para facilitar la incorporación de grandes volúmenes de entidades y frases de entrenamiento en Dialogflow: **aspectItemsParser** y **training-phrase-uploader**. Las siguientes secciones describen su proceso de desarrollo, así como los tipos de ficheros que interpretan.

4.2.4.1 Script aspectItemsParser

Se trata de un script en lenguaje Python que procesa ficheros de vocabulario de aspectos pertenecientes al Information Retrieval Group (UAM) [25]. Estos ficheros se encuentran en formato CSV, y todas sus filas contienen dos datos: el primero representa un aspecto, y el segundo indica un posible sinónimo.

La tarea principal del script consiste en asociar a cada aspecto una lista de sinónimos que va confeccionando según lee cada fila. De este modo, genera un fichero CSV de salida con el formato especificado por Dialogflow [17], listo para importar su contenido en el agente.

El Anexo B recoge un ejemplo de CSV de entrada, así como de salida.

4.2.4.2 Aplicación training-phrase-uploader

Antes de implementar la aplicación, era necesario autorizar sus operaciones sobre el agente de Dialogflow. Para ello, se activó la API de la plataforma sobre PreferenceCollector y su proyecto de Google Cloud asociado. A continuación, se creó una cuenta de servicio [22] sobre la que operaría training-phrase-uploader, que poseía permisos de administrador sobre el agente y unas credenciales para autorizar el uso de la API. Finalmente, se configuraron tanto el SDK de Google Cloud [14] como Eclipse para que la aplicación fuese capaz de acceder a la API desde el equipo de desarrollo.

En cuanto a su funcionalidad, la aplicación procesa ficheros JSON de entrada que contienen los siguientes elementos:

- **projectId.** ID de proyecto de Google Cloud asociado al agente, que lo identifica de forma única.
- **intents.** Turnos de conversación definidos en el agente (por ejemplo, buscar opiniones sobre hoteles). Dentro de ellos se encuentran las frases de entrenamiento que identifican la intención del usuario para referirse a un tema concreto.
- **trainingPhrases.** Almacena las frases de entrenamiento asociadas a un *intent*. Cada una de ellas está dividida en partes, que contienen texto o *keywords* específicas del dominio. Las *keywords* van etiquetadas en partes individuales.
- **languageCode.** Idioma en que están escritas las frases de entrenamiento.

El Anexo B muestra un ejemplo de estos ficheros de entrada.

A través de la librería Jackson, training-phrase-uploader transforma los elementos del fichero en objetos Java con la información del proyecto, los *intents* y las frases de entrenamiento. A continuación, utiliza el *project-id* para acceder al agente de Dialogflow y añadir las frases de entrenamiento asociadas a cada *intent*. Cabe destacar que la API de Dialogflow sobrescribe las nuevas frases con las que existían anteriormente y, a día de hoy. no es posible modificar este comportamiento.

5 Pruebas y resultados

Durante la primera y segunda iteración, así como al concluir cada una de ellas, se realizó una batería de pruebas sobre el sistema con el fin de verificar su correcto funcionamiento. Los siguientes apartados describen su ejecución, así como los resultados obtenidos.

5.1 Pruebas durante la primera iteración

En paralelo con el proceso de desarrollo, se iniciaron numerosas conversaciones de prueba tanto en la consola de Dialogflow como en el bot de Telegram. Su fin principal consistió en verificar el flujo habitual de una conversación entre el agente y el usuario, así como introducir *input* anómalo en distintas fases del diálogo para estudiar la respuesta del agente. Gracias a una prueba de este tipo se descubrió que, cuando el bot devolvía las opiniones sobre hoteles y el usuario enviaba un mensaje fuera de ámbito, el agente respondía con el mensaje predeterminado asociado a DefaultFallbackIntent:

```
"Sorry, which kind of hotel review or opinion do you want to search for?"
```

De este modo, se añadió el *intent* GetOpinionFallback para manejar input anómalo después de que el usuario solicitase opiniones sobre hoteles.

Por otro lado, se confeccionaron diversos scripts cURL con Webhook Requests diseñadas para solicitar opiniones sobre una cadena de hoteles u hoteles específicos. De este modo, fue posible automatizar y facilitar el envío de peticiones complejas al agente de Dialogflow con propósitos de *debugging* y validación. En total se definieron cuatro scripts que obtuvieron los resultados deseados: dos de ellos pedían opiniones positivas, y los dos restantes pedían opiniones negativas.

5.2 Pruebas durante la segunda iteración

Del mismo modo que en la primera iteración, se ejecutó un gran número de conversaciones entre el agente y el usuario para explorar tanto casos favorables como el manejo de *input* anómalo. En una de las pruebas se descubrió la necesidad de interaccionar con el servicio web cuando el usuario accedía a realizar una nueva búsqueda, ya que el agente de Dialogflow no podía eliminar los registros de la base de datos directamente. Por ello, los *intents* GetPreference – noResult – accept y GetPreference – reject – restart se modificaron para enviar una Webhook Request al servicio web y mostrar el *payload* recibido.

Una vez finalizada la fase de desarrollo, se creó una serie de scripts cURL para verificar que el agente devolvía los hoteles adecuados en función de ciertas preferencias de usuario:

- **Script 1:** «*Hotels with pool and great Chinese food*»
- **Script 2:** «*Hotels with more than 3 stars and spotless bathrooms*»
- **Script 3:** «*Hotels with nice food service, and price below 120€*»
- **Script 4:** «4-star *hotels with food service and rooms that are clean*»

Todos los scripts mencionados anteriormente devolvieron los aspectos sobre los que el usuario había expresado su interés, así como la lista de hoteles asociada.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Este proyecto poseía una curva de aprendizaje considerable, ya que utilizaba herramientas muy diferentes entre sí para propósitos específicos (Dialogflow, Heroku, Spring Boot). Todas ellas debían interactuar de una determinada manera y proporcionar un resultado rápidamente, ya que el retardo en una conversación debía ser reducido. Precisamente por la complejidad de las herramientas y las restricciones preliminares, desarrollar el sistema conversacional ha conllevado la asimilación y puesta en práctica de una gran variedad de conceptos, muchos de ellos punteros.

La parte más desafiante del proyecto consistió en el diseño del sistema conversacional, puesto que la conversación entre el agente y el usuario debía resultar natural e informativa, abstrayendo detalles innecesarios. Adicionalmente, el *backend* del sistema debía adaptarse a diversos dominios sin necesidad de implementar su lógica desde cero. Por tanto, esta fase del proyecto se llevó a cabo de manera cuidadosa, considerando diversas estrategias de desarrollo y aplicando técnicas de programación consolidadas a lo largo del grado.

Considero que todo el tiempo invertido en el sistema conversacional ha sido de gran utilidad para profundizar sobre plataformas y *frameworks* relevantes en la actualidad, y que ha concluido de forma satisfactoria. Se ha implementado un software estable, cuya funcionalidad puede extenderse de manera sencilla con nuevos requisitos.

6.2 Trabajo futuro

Durante la fase de desarrollo del proyecto surgieron ideas para futuras implementaciones, que se describen a continuación:

- Transformar la lista que obtiene el servicio web con los nombres de todos los aspectos asociados al dominio en un árbol. Este árbol contendría nodos hoja con aspectos independientes (por ejemplo, «*stars*» o «*price*»), y nodos padre que representarían conceptos relacionados (por ejemplo, un nodo padre denominado «*services*», con sub-nodos «*gym*» y «*wifi*»).
- Capacidad del agente para procesar consultas que excluyan ciertos aspectos (por ejemplo, «*get hotels with no pool*»), y recuperar hoteles que no los incluyan en sus prestaciones.
- Transformar el modelo de conversación de forma que los usuarios puedan identificarse en el sistema (por ejemplo, con email y contraseña), y éste recuerde sus preferencias más allá de una interacción completa. A esta idea se le suma la capacidad del servicio web para decrementar pesos según cambien los intereses del usuario.
- Utilizar la entidad @result definida en el agente de Dialogflow para tomar rutas más específicas en el diálogo: el usuario expresa que ya ha visitado los hoteles o que desea conocer más sobre ciertos aspectos, decide reestructurar su consulta, etc.
- Expandir la base de conocimiento del agente (tipos de entidades, *intents*, frases de entrenamiento, contextos, etc.) con el objetivo de investigar e implementar nuevos casos de uso.

Referencias

[1] Amazon.com, Inc., (2020). "Alexa and Alexa Device FAQs" [online] https://www.amazon.com/gp/help/customer/display.html/ref=hp_left_v4_sib?ie=UTF8&nodeId=201602230 [Accedido el 6 de Marzo de 2020]

[2] The Apache Software Foundation, (2020). "Introduction to the POM" [online] https://maven.apache.org/guides/introduction/introduction-to-the-pom.html [Accedido el 25 de Marzo de 2020]

[3] Apple, Inc., (2020). "Siri does more than ever. Even before you ask." [online] https://www.apple.com/siri/ [Accedido el 5 de Marzo de 2020]

[4] Apple, Inc., (2020). "Your home at your command" [online] https://www.apple.com/ios/home/ [Accedido el 5 de Marzo de 2020]

[5] R. Carpenter, (2020). "Cleverbot" [online] https://www.cleverbot.com/ [Accedido el 8 de Marzo de 2020]

[6] K. Christakopoulou, F. Radlinski, K. Hofmann, (2016). "Towards Conversational Recommender Systems", in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.815-824.

[7] "Cortana" (s.f). En Wikipedia. https://en.wikipedia.org/wiki/Cortana [Accedido en 5 de Marzo de 2020]

[8] P. Dutta, (2020). "Quick Guide to Spring Controllers" [online] https://www.baeldung.com/spring-controllers [Accedido el 18 de Marzo de 2020]

[9] S. Faatz, (2018). "Transaction or Knowledge Chatbot? Moving Beyond the Siri Syndrome" [online] https://www.telerik.com/blogs/transactional-or-knowledge-chatbot-moving-beyond-siri-syndrome [Accedido el 8 de Marzo de 2020]

[10] FasterXML, LLC., (2020). "Jackson" [repositorio Github] https://github.com/FasterXML/jackson [Accedido el 15 de Marzo de 2020]

[11] M. J. Garbade, (2018). "A Simple Introduction to Natural Language Processing" [online] https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32 [Accedido el 2 de Marzo de 2020]

[12] N. Gilliland, (2016). "Domino's introduces 'Dom the Pizza Bot' for Facebook Messenger" [online] https://econsultancy.com/domino-s-introduces-dom-the-pizza-bot-for-facebook-messenger/ [Accedido el 8 de Marzo de 2020]

[13] Google, LLC, (2020). "Build an agent from scratch using best practices" [online] https://cloud.google.com/dialogflow/docs/tutorials/build-an-agent/ [Accedido el 14 de Abril de 2020]

[14] Google, LLC, (2020). "Cloud SDK" [online] https://cloud.google.com/sdk [Accedido el 30 de Marzo de 2020]

[15] Google, LLC, (2020). "Conversation design" [online] https://designguidelines.withgoogle.com/conversation/conversation-design/welcome.html [Accedido el 14 de Abril de 2020]

[16] Google, LLC, (2020). "Dialogflow documentation" [online] https://cloud.google.com/dialogflow/docs/ [Accedido el 12 de Marzo de 2020]

[17] Google, LLC, (2020). "Export and import entities" [online] https://cloud.google.com/dialogflow/docs/entities-export [Accedido el 22 de Marzo de 2020]

[18] Google, LLC, (2020). "Google Now. The right information at just the right time" [online] https://www.google.com/intl/en-GB/landing/now/ [Accedido el 8 de Marzo de 2020]

[19] Google, LLC, (2020). “Introduction videos” [online] https://cloud.google.com/dialogflow/docs/video [Accedido el 15 de Marzo de 2020]
[20] Google, LLC, (2020). “Package google.cloud.dialogflow.v2” [online] https://cloud.google.com/dialogflow/docs/reference/rpc/google.cloud.dialogflow.v2 [Accedido el 18 de Marzo de 2020]
[21] Google, LLC, (2020). “Rich response messages” [online] https://cloud.google.com/dialogflow/docs/intents-rich-messages [Accedido el 26 de Marzo de 2020]
[22] Google, LLC, (2020). “Service accounts” [online] https://cloud.google.com/iam/docs/service-accounts [Accedido el 30 de Marzo de 2020]
[23] Heroku.com, (2020). “Getting Started on Heroku with Java” [online] https://devcenter.heroku.com/articles/getting-started-with-java [Accedido el 18 de Marzo de 2020]
[24] Heroku.com, (2020). “Heroku Postgres” [online] https://www.heroku.com/postgres [Accedido el 15 de Marzo de 2020]
[25] Information Retrieval Group, (s.f.). “Introduction” [online] http://ir.ii.uam.es/ [Accedido el 22 de Marzo de 2020]
[26] M. Lippert, (2018). “Spring Tools for Eclipse IDE” [online] https://www.eclipse.org/community/eclipse_newsletter/2018/february/springboot.php [Accedido el 25 de Marzo de 2020]
[27] L. Lovett, (2020). “Lark Health rolls out behavioral health coaching targeted at patients at risk for chronic conditions” [online] https://www.mobihealthnews.com/news/lark-health-rolls-out-behavioral-health-coaching-targeted-patients-risk-chronic-conditions [Accedido el 8 de Marzo de 2020]
[28] J. McCarthy, (2017). “Meet Lego’s Facebook Messenger chatbot Ralph, a helpful alternative to bricks and mortar” [online] https://www.thedrum.com/news/2017/11/23/meet-lego-s-facebook-messenger-chatbot-ralph-helpful-alternative-bricks-and-mortar [Accedido el 8 de Marzo de 2020]
[29] Microsoft Corporation (2019). “Cortana Skills Kit” [online] https://docs.microsoft.com/en-us/cortana/skills/overview [Accedido en 5 de Marzo de 2020]
[30] B. Morgan, (2017). “What Is a Chatbot, and Why Is It Important for Customer Experience?” [online] https://www.forbes.com/sites/blakemorgan/2017/03/09/what-is-a-chatbot-and-why-is-it-important-for-customer-experience/ [Accedido el 2 de Marzo de 2020]
[31] B. O’Boyle, (2019). “Google Assistant vs. Alexa vs. Siri: Battle of the personal assistants” [online] https://www.pocket-lint.com/smart-home/buyers-guides/124938-google-assistant-vs-alexa-vs-siri-personal-assistants [Accedido el 5 de Marzo de 2020]
[32] B. O’Boyle, (2019). “What is Siri and how does Siri work?” [online] https://www.pocket-lint.com/apps/news/apple/112346-what-is-siri-apple-s-personal-voice-assistant-explained [Accedido el 5 de Marzo de 2020]
[33] B. O’Boyle, (2020). “What is Alexa and what can Amazon Echo do?” [online] https://www.pocket-lint.com/smart-home/news/amazon/138846-what-is-alexa-how-does-it-work-and-what-can-amazons-alexa-do [Accedido el 6 de Marzo de 2020]
[34] D. Priest, (2016). “Here’s everything the Amazon Echo can do” [online] https://www.cnet.com/pictures/what-can-amazon-echo-and-alexa-do-pictures/ [Accedido el 6 de Marzo de 2020]
[35] M. Prospero, K. Kozuch (2020). “The best Google Assistant skills in 2020” [online] https://www.tomsguide.com/round-up/best-google-assistant-features [Accedido el 8 de Marzo de 2020]

[36] E. Ravenscraft, (2015). “Everything You Can Ask Cortana to Do in Windows 10” [online] https://lifehacker.com/everything-you-can-ask-cortana-to-do-in-windows-10-1721725525 [Accedido el 5 de Marzo de 2020]
[37] A. Shevat, (2017). “Designing Bots: Creating Conversational Experiences” O'Reilly Media, Inc. 348pp.
[38] A. Shuman, (2020). “Cortana in the upcoming Windows 10 release: focused on your productivity with enhanced security and privacy” [online] https://blogs.windows.com/windowsexperience/2020/02/28/cortana-in-the-upcoming-windows-10-release-focused-on-your-productivity-with-enhanced-security-and-privacy/ [Accedido el 5 de Marzo de 2020]
[39] Y. Sun, Y. Zhang, (2018). “Conversational Recommender System”, in the 41st International ACM SIGIR Conference o Research & Development in Information Retrieval, p.235-244.
[40] P. Surmenok, (2016). “Chatbot Architecture” [online] https://medium.com/@surmenok/chatbot-architecture-496f5bf820ed [Accedido el 8 de Marzo de 2020]
[41] Telegram FZ-LLC, (s.f.) “Bots: an introduction for developers” [online] https://core.telegram.org/bots#6-botfather [Accedido el 24 de Marzo de 2020]
[42] VMWare, Inc., (2020). “Spring | Guides” [online] https://spring.io/guides [Accedido el 18 de Marzo de 2020]
[43] VMWare, Inc., (2020). “Spring Data JDBC – Reference Documentation” [online] https://docs.spring.io/spring-data/jdbc/docs/current/reference/html/#jdbc.repositories [Accedido el 18 de Marzo de 2020]
[44] B. Wooldridge, (2020). “HikariCP” [repositorio Github] https://github.com/brettwooldridge/HikariCP [Accedido el 15 de Marzo de 2020]

Glosario

Agente	Software de Dialogflow cuya tarea principal consiste en mantener conversaciones con usuarios. Dentro de esta tarea principal, es capaz de realizar otras operaciones como recoger información o enviar peticiones a un servicio web.
Anotación Java	Comienzan por el símbolo '@' y añaden directrices adicionales a una aplicación Java. Existen tanto anotaciones nativas como otras definidas por librerías o *frameworks*.
Asistente virtual	Software que ofrece una gran variedad de servicios a los usuarios con el fin de automatizar o facilitar ciertas tareas. Es capaz de interpretar *input* en formato textual o hablado.
Chatbot	Software cuyas características y objetivos son muy similares a las que incorporan los asistentes virtuales, aunque sólo es capaz de procesar mensajes de texto.
Contexto	En el ámbito de una conversación, específicamente entre un agente de Dialogflow y un usuario, representa la información que rodea las interacciones entre ambos. Esta información aporta un significado adicional e implícito al *input* recibido, y permite determinar a qué se refiere un usuario.
Contexto de entrada	Contexto de Dialogflow identificado por un nombre, que puede asignarse a *intents* determinados. Cuando un *intent* lleva asociado un contexto de entrada, se lanzará si se cumplen dos condiciones: un contexto de salida con el mismo nombre se ha activado previamente, y el *input* del usuario coincide con las frases de entrenamiento registradas en el *intent*.
Contexto de salida	Contexto de Dialogflow identificado por un nombre, que puede configurarse para ser activado cuando se lanza un *intent* concreto.
Dialogflow	Plataforma de Google, LLC que permite diseñar y construir sistemas conversacionales capaces de interpretar el lenguaje natural.
Entidad de mapeo	Tipo de entidad compuesta por una *keyword* que actúa como valor de referencia, y una lista de sinónimos asociados a ella.
Entidad de lista	Tipo de entidad compuesta por una enumeración de valores independientes; todos ellos actúan como posibles valores que pueden ser detectados por el agente de Dialogflow.
Framework	Conjunto de herramientas que permiten desarrollar software siguiendo una serie de estándares y criterios específicos.

Heroku	Plataforma que permite desplegar y alojar aplicaciones web en la nube.
Heroku Postgres	*Add-on* o extensión de Heroku que permite asociar un gestor de base de datos PostgreSQL a una aplicación web.
HikariCP	Framework que permite a las aplicaciones Java crear un pool de conexiones JDBC de alto rendimiento.
Intent	En el marco de un agente de Dialogflow, se trata de un conjunto de frases de entrenamiento y contextos de entrada/salida que permite detectar la intención del usuario para referirse a un tema específico (por ejemplo, solicitar la previsión del tiempo).
Jackson	Librería que permite transformar objetos Java a otros formatos de datos como JSON, XML o YAML, entre otros; así como realizar el proceso inverso.
Parámetro	En el marco de un agente de Dialogflow, se trata de datos que se insertan en una Webhook Request o Webhook Response. Cada parámetro almacena el valor de un tipo de entidad (una keyword) que el agente ha identificado en los mensajes del usuario.
Pool de conexiones	Conjunto de conexiones a una base de datos que se mantienen abiertas. Este comportamiento tiene como objetivos la reutilización de dichas conexiones por parte de los clientes y la optimización de recursos.
PostgreSQL	Sistema de gestión de bases de datos relacionales distribuido como código *open-source*.
Spring Boot	Framework especializado en la construcción de aplicaciones web Java. Su diseño abstrae detalles de bajo nivel y de configuración al desarrollador, permitiéndole centrar sus esfuerzos en implementar la solución.
Telegram	Aplicación de mensajería instantánea y VOIP que permite a los usuarios comunicarse entre sí e interaccionar con *chatbots*.
Tipo de entidad	*Keyword* o información de interés presente en los mensajes del usuario. Dialogflow define entidades de varios tipos, como de mapeo o de lista.
Webhook Request	Petición HTTPS POST que el agente de Dialogflow puede enviar a un servicio web concreto. Su contenido es una estructura JSON con información sobre el *intent* que se ha lanzado y los tipos de entidades detectados, entre otros.
Webhook Response	Respuesta HTTPS que recibe el agente de Dialogflow desde el servicio web. Su contenido es una estructura JSON con posibles

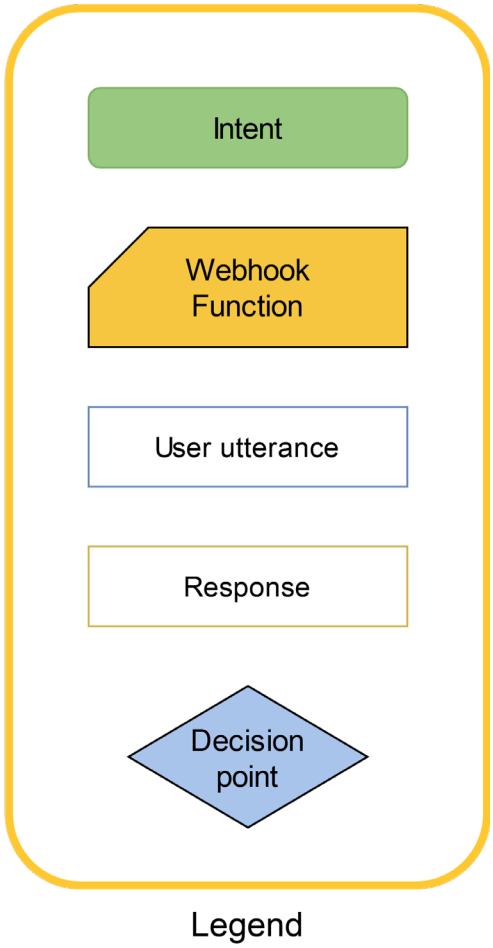
mensajes dirigidos al usuario y otros valores para controlar la conversación (por ejemplo, nombres de contextos de salida que deben activarse en el agente).

Anexos

A Diagramas de flujo

Durante el proceso de diseño se crearon dos diagramas de flujo para definir la conversación que correspondía a cada escenario planteado: recuperar opiniones sobre hoteles y buscar hoteles acordes a las preferencias del usuario.

A continuación, se adjunta la leyenda de ambos diagramas:

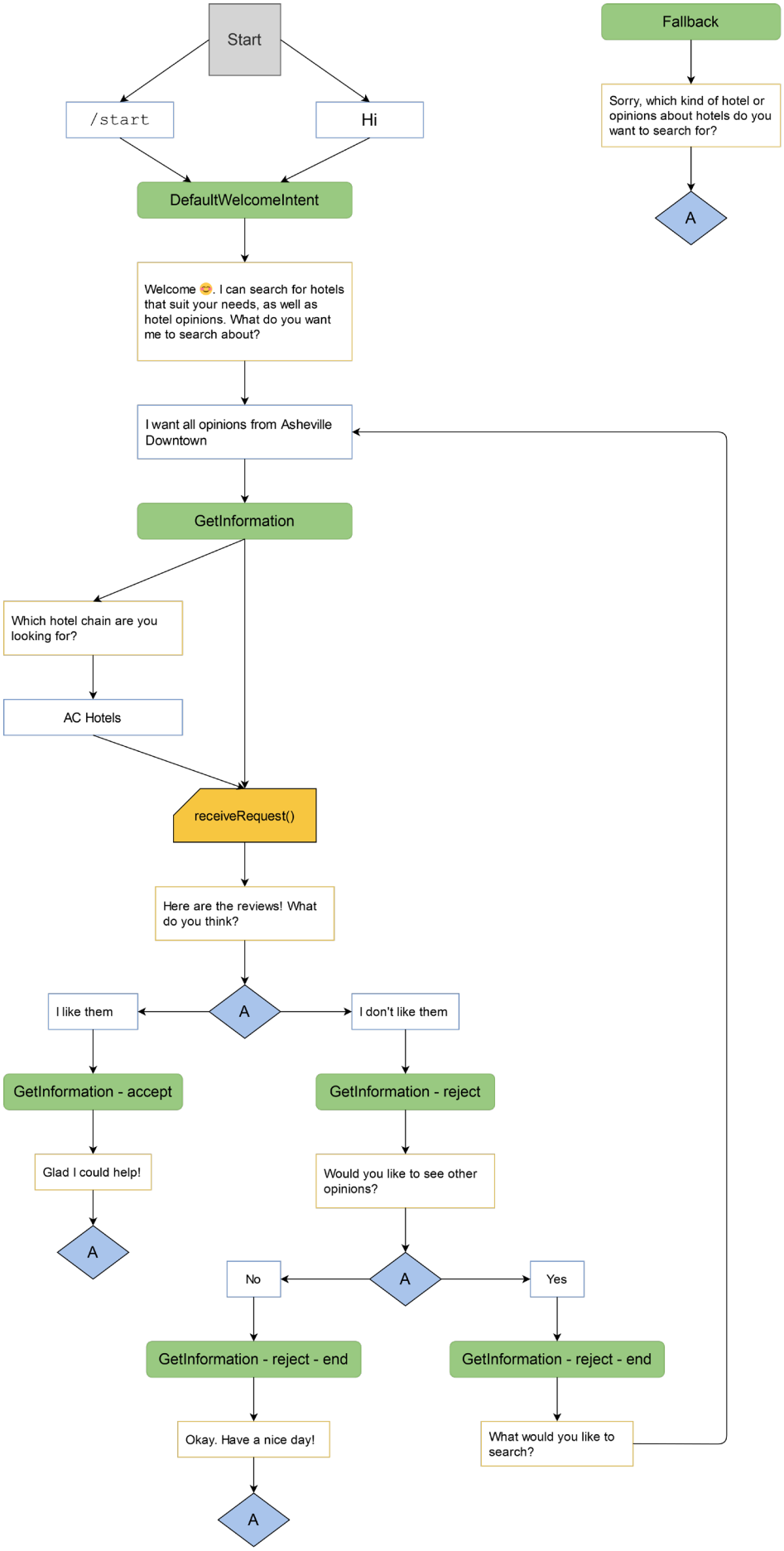


Fuente: [16]

Todos ellos se construyeron teniendo en cuenta las prácticas recomendadas por Google [15], y siguieron proceso de diseño especificado en un tutorial de Dialogflow para construir un agente desde cero [13].

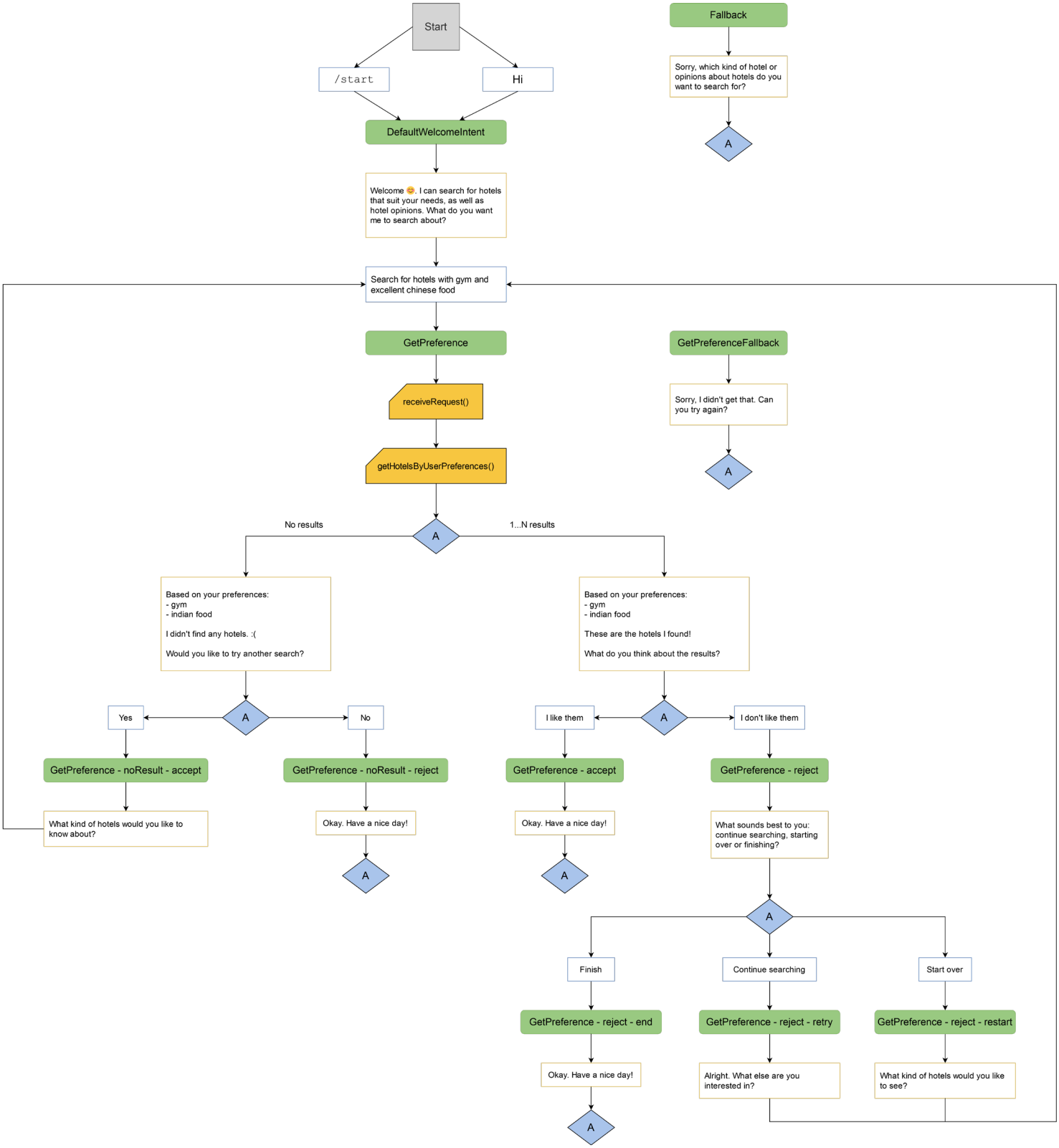
i. Diagrama de flujo en la primera iteración

En la siguiente página se muestra la evolución del diálogo entre el agente de Dialogflow HotelReviewProvider y el usuario final, con sus *intents* y respuestas correspondientes.


Start
/start
Hi
DefaultWelcomeIntent
Welcome 😊. I can search for hotels that suit your needs, as well as hotel opinions. What do you want me to search about?
I want all opinions from Asheville Downtown
GetInformation
Which hotel chain are you looking for?
AC Hotels
receiveRequest()
Here are the reviews! What do you think?
A
I like them
I don't like them
GetInformation - accept
Glad I could help!
A
GetInformation - reject
Would you like to see other opinions?
A
No
Yes
GetInformation - reject - end
Okay. Have a nice day!
A
GetInformation - reject - end
What would you like to search?
Fallback
Sorry, which kind of hotel or opinions about hotels do you want to search for?
A

ii. Diagrama de flujo en la segunda iteración

A continuación, se muestra la evolución del diálogo entre el agente de Dialogflow PreferenceCollector y el usuario final, con sus *intents* y respuestas correspondientes.



B Ficheros de entrada del sistema

Las siguientes secciones muestran el contenido de diversos ficheros de entrada que el sistema es capaz de procesar, así como una descripción de su estructura.

i. Fichero application.properties

```
# Database configuration
# NOTE: Spring 2.x onwards uses HikariCP as default datasource
spring.datasource.url=${JDBC_DATABASE_URL}
spring.datasource.username=${JDBC_DATABASE_USERNAME}
spring.datasource.password=${JDBC_DATABASE_PASSWORD}

spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.hikari.connection-timeout=10000
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=10

# Application's default port (environment variable)
server.port=${PORT:5000}

# Jackson set to ignore null JSON fields
spring.jackson.default-property-inclusion=NON_NULL

# Logging info
logging.level.org.springframework=INFO
```

Este archivo de configuración define parámetros que aplican a varios aspectos del servicio web:

- **Base de datos.** Especifica la ruta completa a la base de datos alojada en Heroku, compuesta por variables de entorno que almacenan la URL de acceso, el usuario y la contraseña necesarios para operar sobre ella. También define parámetros como el driver JDBC que utilizará el servicio web, el *timeout* y el tamaño del pool de conexiones permitido por HikariCP.
- **Jackson.** A la hora de transformar objetos Java a datos JSON y viceversa, el parámetro `spring.jackson.default-property-inclusion` establece que sólo se tendrán en cuenta los elementos distintos a NULL.
- **Aplicación de Spring Boot.** El fichero indica un puerto por defecto para recibir Webhook Requests (definido en la variable `$PORT` o 5000 en caso contrario), así como el tipo de mensajes que Spring Boot volcará en sus logs.

ii. Fichero system.properties

```
# Java language version
java.runtime.version=11
```

Fuerza a Heroku a utilizar OpenJDK 11 a la hora de desplegar la aplicación. De este modo, se evitan discordancias entre el proceso de *build* de Maven (configurado sobre la versión 11 de Java) y el *deploy* en Heroku.

iii. CSV de entrada para AspectItemsParser

El siguiente fragmento muestra un CSV de entrada resumido, procedente de Information Retrieval Group [25]:

```
amenities,amenities
amenities,services
atmosphere,ambiance
atmosphere,atmosphere
atmosphere,light
bar,bar
bar,bartender
bathrooms,bathrooms
bathrooms,bathtub
bedrooms,bed
bedrooms,bedrooms
bedrooms,pillow
```

Cada fila contiene dos entradas. La primera representa una *keyword* o tipo de entidad que va a incorporarse en el agente de Dialogflow; la segunda corresponde a un posible sinónimo asociado a esa *keyword.*

iv. CSV de salida producido por AspectItemsParser

A continuación se adjuntan las primeras 15 líneas de un CSV que puede generar AspectItemsParser a partir del *input* mostrado en el apartado anterior:

```
amenities,amenities,amenity,services
atmosphere,atmosphere,ambiance,ambiances,ambience,ambiences
bar,bar,bars,bartender,bartenders
bathrooms,bathrooms,bath,bathroom,baths,bathtub,bathtubs,shampoo
bedrooms,bedrooms,bed,bedroom,beds,pillow,pillows,sheet,sheets,sleep
booking,booking,book,reservation,reservations,reserve
breakfast,breakfast,breakfasts,morning,mornings,toast,toasts
building,building,architecture,architectures,buildings,decor
checking,checking,check,check in,check ins,check out,check outs
cleanliness,cleanliness,clean,cleaned,cleaning,smell
coffee,coffee,cafe,cafes,coffees,tea,teas
dinner,dinner,dinners,evening,evenings,night,nights
drinks,drinks,beer,beers,drink,wine,wines
events,events,activities,activity,event,parties,party,trip,trips
facilities,facilities,equipment,facility
```

Este fichero sigue el formato indicado por Dialogflow para importar entidades de mapeo en un agente [17]. Cada fila posee una *keyword* (resaltada en negrita) y una lista de sinónimos que se corresponden con esa palabra clave. La propia *keyword* se incluye dos veces para indicar a Dialogflow que podría aparecer en mensajes procedentes del usuario.

v. Fichero JSON de entrada para training-phrase-uploader

El archivo, adjunto en la página siguiente, contiene todos los campos descritos en el apartado 4.2.4.2:

- **Datos del proyecto.** Indica el ID del proyecto asociado a PreferenceCollector, y el idioma en que se adjuntan las frases de entrenamiento (en este caso, inglés).
- **Intents.** Se trata de una lista con dos elementos: GetOpinion y GetPreference. Por cada *intent* se debe especificar su ID y su nombre asignado.
- **Training phrases.** GetOpinion incluye una frase etiquetada con la entidad @hotel-chain («*Get all positive opinions from AC Hotels*»), mientras que GetPreference lleva asociada la frase («*Search for hotels with excellent Chinese food*»). En el segundo caso, se pretende asignar a @preference el valor «*excellent Chinese food*»).

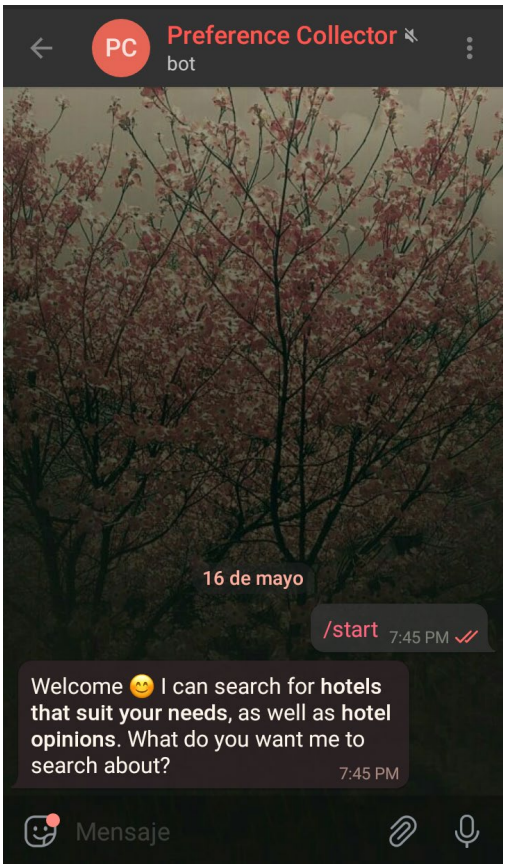
```
{
"projectId": "preferencecollector-dpgxal",
"languageCode": "en",
"intents": [
    {
        "intentId": "124a0b4e-5af0-48f2-b1b0-29a7b9c5c2df",
        "displayName": "GetOpinion",
        "trainingPhrases": [
            {
                "parts": [
                    {
                        "text": "Get all positive opinions from "
                    },
                    {
                        "text": "AC Hotels",
                        "entityType": "@hotel-chain",
                        "alias": "hotel-chain"
                    }
                ]
            }
        ]
    },
    {
        "intentId": "1a09c885-427c-465f-abf4-df9a4b6b249e",
        "displayName": "GetPreference",
        "trainingPhrases": [
            {
                "parts": [
                    {
                        "text": "Search for hotels with "
                    },
                    {
                        "text": "excellent chinese food",
                        "entityType": "@preference",
                        "alias": "preference"
                    }
                ]
            }
        ]
    }
]
}
```

C Ejemplos de conversaciones con el bot de Telegram

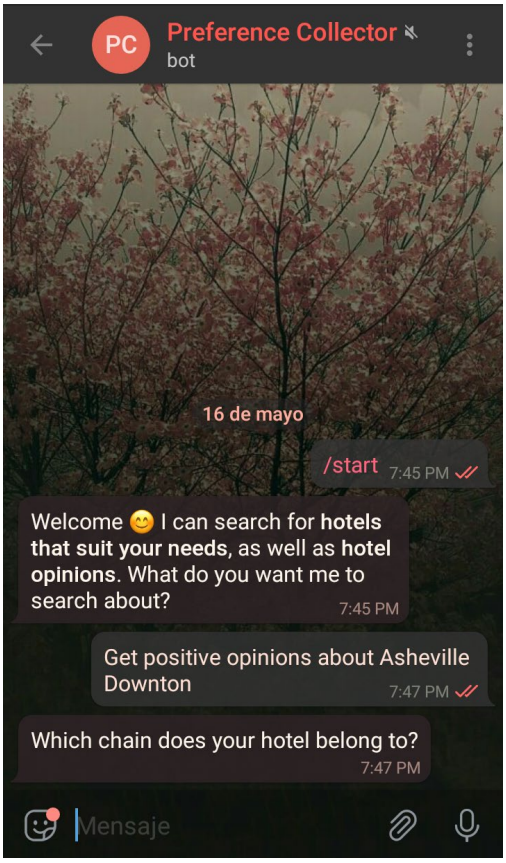
Esta sección incluye dos ejemplos de conversaciones entre el agente de Dialogflow y un usuario final, que abordan los casos de uso descritos en el apartado 3.1. Durante el primer diálogo, el usuario desea recibir opiniones sobre un hotel que pertenece a una cadena determinada. En la segunda interacción, el usuario realiza una consulta para obtener hoteles que se ajusten a sus necesidades.

i. Ejemplo 1: Recibir opiniones sobre un hotel concreto

El usuario inicia la conversación con el *bot* de Telegram mediante el comando `/start`. En ese momento, el agente muestra un mensaje de bienvenida.



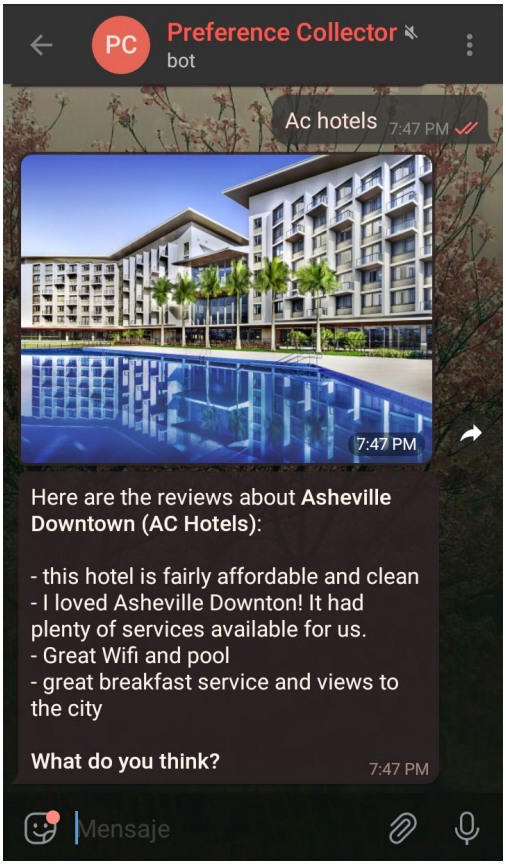
A continuación, el usuario solicita obtener opiniones positivas sobre Asheville Downtown. Dado que el agente necesita saber a qué cadena pertenece este hotel para realizar sus operaciones, guía al usuario para que facilite la información requerida.

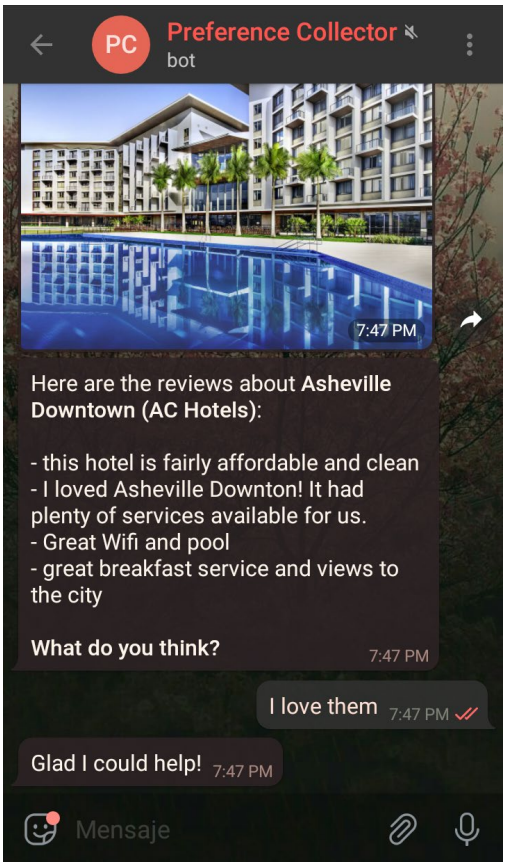


Consecuentemente, el usuario indica la cadena de hoteles vinculada a Asheville Downtown: AC Hotels. En este punto de la conversación, el agente asocia a los tipos de entidades **@hotel-name**, **@hotel-chain** y **@opinion-sentiment** los valores «*Asheville Downtown*», «*AC Hotels*» y «*Positive*», respectivamente. Estos valores se insertan en una Webhook Request que Dialogflow enviará al servicio web. Cuando reciba la Webhook Response correspondiente, el agente adjuntará su contenido en un mensaje dirigido al usuario.

Este mensaje contiene varios elementos:

- Una imagen descriptiva.
- El nombre y la cadena del hotel sobre los que el usuario ha solicitado opiniones, así como la lista de reseñas (en este caso, positivas).
- La pregunta `"What do you think?"`, con el fin de que el usuario indique sus impresiones acerca de los resultados obtenidos.

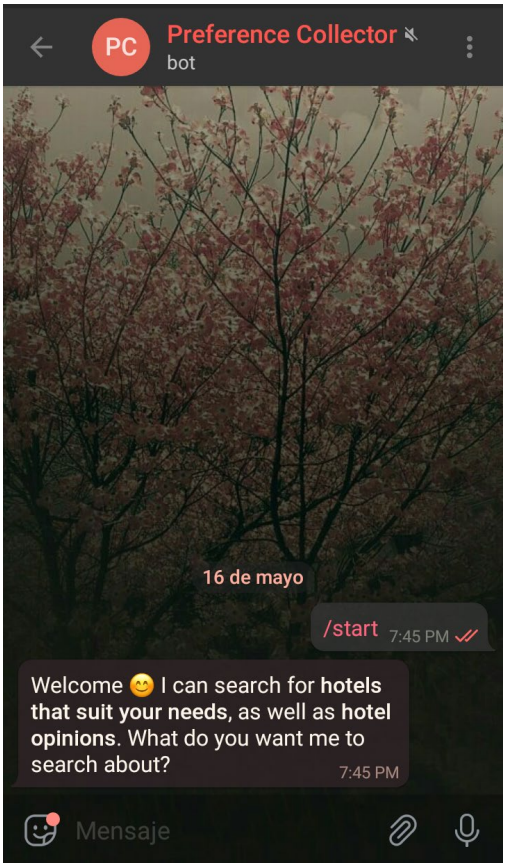

PC
Preference Collector
bot
Ac hotels 7:47 PM
7:47 PM
Here are the reviews about Asheville Downtown (AC Hotels):
- this hotel is fairly affordable and clean
- I loved Asheville Downton! It had plenty of services available for us.
- Great Wifi and pool
- great breakfast service and views to the city
What do you think? 7:47 PM
Mensaje


PC
Preference Collector
bot
7:47 PM
Here are the reviews about Asheville Downtown (AC Hotels):
- this hotel is fairly affordable and clean
- I loved Asheville Downton! It had plenty of services available for us.
- Great Wifi and pool
- great breakfast service and views to the city
What do you think? 7:47 PM
I love them 7:47 PM
Glad I could help! 7:47 PM
Mensaje

Como el usuario expresa su satisfacción con las opiniones que ha recibido, el agente envía una respuesta predeterminada y finaliza la conversación.

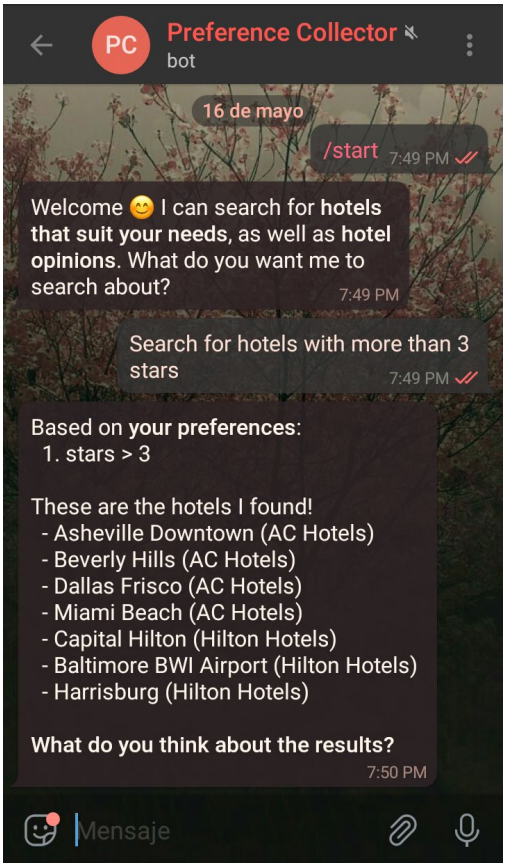
ii. Ejemplo 2: Obtener hoteles según preferencias de usuario

Del mismo modo que en el ejemplo anterior, el usuario invoca al agente de Dialogflow mediante el comando `/start`, y recibe su mensaje de bienvenida.

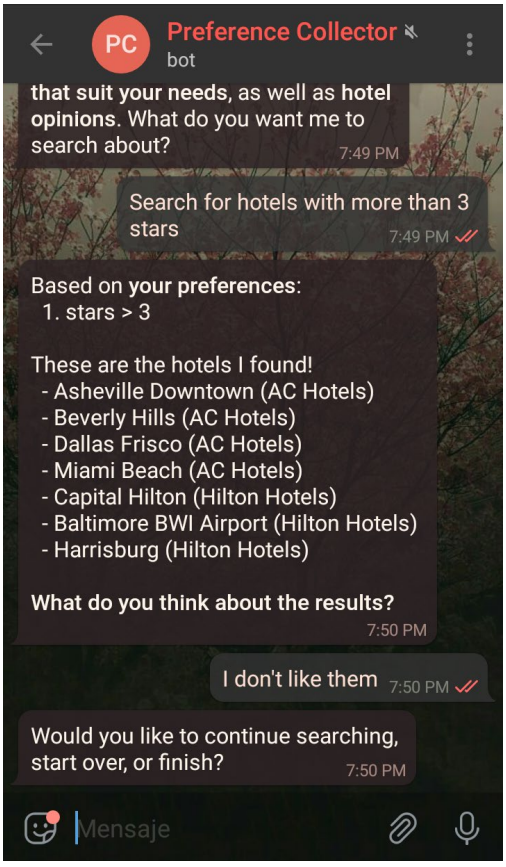


En este caso, el usuario realiza una consulta para obtener hoteles de más de tres estrellas. A continuación, el agente de Dialogflow extrae el tipo de entidad asociado al aspecto que ha indicado el usuario (en este caso, **@stars**), así como su valor. Esta información se inserta en una Webhook Request dirigida al servicio web, quien enviará su correspondiente Webhook Response de vuelta al agente. Cuando esto ocurra, el agente mostrará un mensaje en el chat con el siguiente contenido:

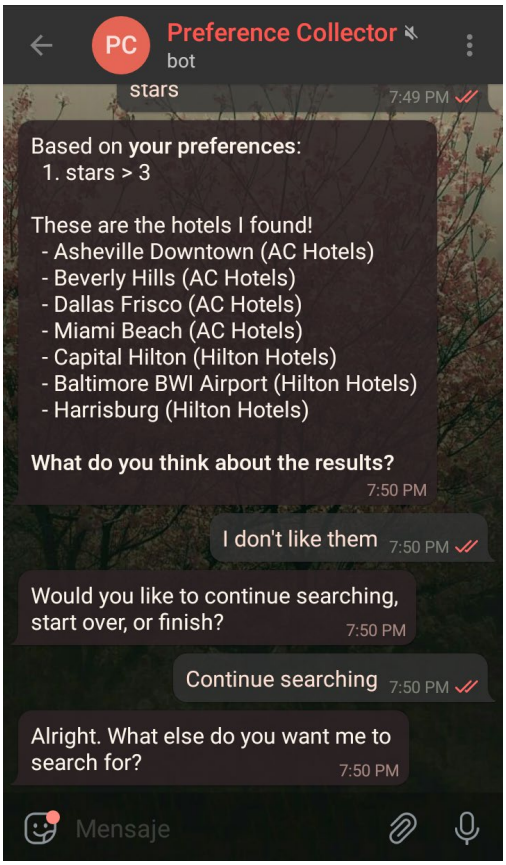
- Aspectos solicitados.
- Hoteles que se ajustan a las preferencias del usuario: incluyen los aspectos solicitados, y el valor de dichos aspectos cumple con las expectativas del usuario (en este caso, todos los hoteles poseen tres o más estrellas).
- La pregunta `"What do you think about the results?"`, con el fin de que el usuario comparta sus impresiones acerca de los resultados obtenidos.



Sin embargo, el usuario no se muestra satisfecho con los resultados.

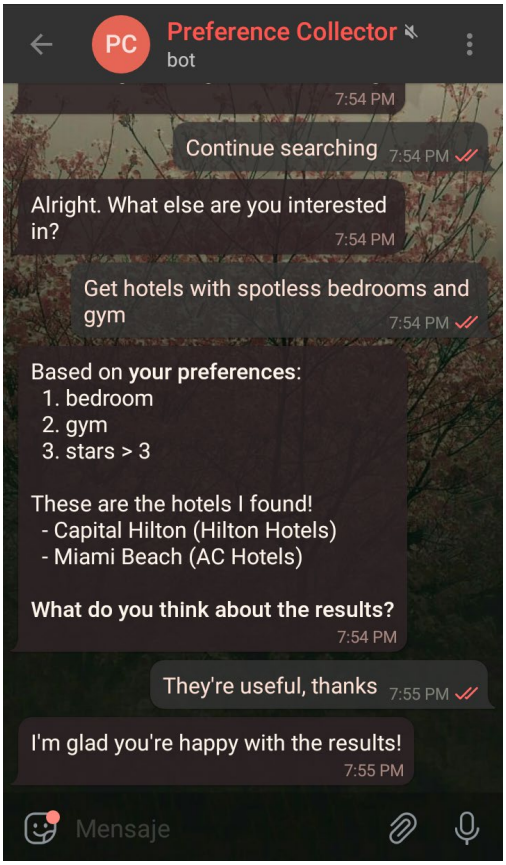


Ante esta situación, el agente de Dialogflow ofrece continuar la búsqueda de hoteles, realizar una nueva o finalizar el diálogo. El usuario escoge continuar la búsqueda, con lo que el agente solicita nuevos aspectos para añadirlos a la consulta en curso.



El usuario limita la búsqueda a hoteles que, además de ser calificados con tres estrellas o más, posean habitaciones impecables y un gimnasio. De nuevo, Dialogflow extrae los tipos de entidades y sus valores asociados a partir del *input* del usuario, y adjunta la información en una Webhook Request que enviará al servicio web.

Cuando el agente reciba la respuesta desde el servicio, mostrará un nuevo mensaje con los aspectos especificados por el usuario, la lista de hoteles que se ajustan a sus preferencias, y la pregunta "`What do you think about the results?`".



Como el usuario responde de forma positiva a los resultados obtenidos, el agente envía una respuesta predeterminada y da por concluida la conversación.