Facultad de Ciencias

Diseño e implementación de un asistente inteligente para la gestión de eventos y servicios básicos sobre Google Cloud Platform

(Design and implementation of a smart assistant for event and service management in Google Cloud Platform)

Trabajo de Fin de Máster

para acceder al

MÁSTER EN INGENIERÍA INFORMÁTICA

Autor: Jesús Durán Hernández

Director\es: Marta Elena Zorrilla Pantaleón

Co-director: Fernando López Saiz

Septiembre - 2018

Tabla de contenido

Tabla de ilustraciones

Tabla de tablas

1. Resumen y *Abstract*

1.1. Resumen

Los grandes avances en Inteligencia Artificial en los últimos años han repercutido, entre otros, en dos campos de conocimiento. Uno de ellos es el Procesamiento de Lenguaje Natural, que se centra en estudiar mecanismos eficaces computacionalmente para la comunicación entre personas y máquinas por medio de lenguajes naturales. Y otro, el Aprendizaje Automático, que desarrolla técnicas para permitir a las computadoras aprender a partir de patrones extraídos de los datos con los que se entrenan.

A raíz de estos progresos, surge una tecnología conocida como *chatbot*, la cual utiliza tanto el Procesamiento de Lenguaje Natural como el Aprendizaje Automático. Los *chatbots* son sistemas capaces de mantener conversaciones con humanos de forma eficaz, y aprenden a medida que tienen estas conversaciones, por lo que la experiencia del usuario tiende a mejorar con el paso del tiempo.

Grandes empresas tecnológicas han desarrollado sus propios *chatbots* o asistentes y, además los han integrado en otros productos para facilitar su uso o los han comercializado como un producto independiente. Algunos ejemplos de esta situación son Cortana, el asistente de Microsoft; Siri, el de Apple; Alexa el de Amazon; y Google Assistant, el de Google.

A menor nivel, muchas empresas tienen su propio asistente para resolver casos de negocio como pueden ser: dar soporte a determinadas aplicaciones, llevar a cabo transacciones bancarias o hacer pedidos de comida, entre otros muchos.

En concreto, este Trabajo Fin de Máster tiene como objetivo la implementación de un *chatbot* que sea capaz de proporcionar información sobre el catálogo de aplicaciones y servicios de la empresa EvenBytes S.L., así como de gestionar citas de manera autónoma.

El desarrollo del trabajo se llevará a cabo utilizando varios servicios de Google Cloud Platform, como Dialogflow y Google Cloud Functions, integrándose además con el Calendario de Google para la gestión de eventos. También, se hará uso de Google App Engine para proporcionar al cliente una aplicación con la que poder configurar el asistente.

Palabras clave: *chatbot*, asistente virtual, Google Cloud Platform, Dialogflow,Procesamiento de Lenguaje Natural.

1.2. Abstract

The great advances in the Artificial Intelligence arena have fostered, among others, two fields of knowledge in the last years, such as Natural Language Processing, which focuses

on studying computationally efficient methods of communication between people and machines by using natural languages and Machine Learning, which develops new techniques that allow computers to learn from patterns that were built from the know facts.

As a result of these advances, a new technology known as chatbot has emerged. It uses both Natural Language Processing and Machine Learning to interact with humans in a efficient way, simulating the human behaviour. Chatbots learn while they talk, hence user experience always tends to improve.

Large technology companies have developed general purpose chatbots or virtual assistants to both create a new product or integrate it in an existing one, so it can help users. These kinds of assistants can complete any user request. A few examples are Microsoft's Cortana, Apple's Siri, Amazon's Alexa or Google's Google Assistant.

At a lower scale, many companies have their own specific-purpose assistant to help in different business cases such as user support, conducting bank transactions or food ordering, among others.

Specifically, the goal of this project is to develop a chatbot that answers user's questions about the apps and services of the EvenBytes S.L. company, as well as manages appointments. Its development will be performed using several Google Cloud Platform services such as Dialogflow and Google Cloud Functions, and Google Calendar as the event manager. Furthermore, Google App Engine will be used to provide the client with an application to be able to tune up chatbot settings.

Palabras clave: chatbot, virtual assistant, Google Cloud Platform, Dialogflow, Natural Language Processing

2. Introducción

2.1. Contexto de la empresa y motivación

EvenBytes S.L. es una empresa TI (Tecnologías de la Información) fundada en el año 2014, con sede en la localidad de Santa Cruz de Bezana, Comunidad Autónoma de Cantabria, España, y en ella trabajan profesionales con más de 15 años de experiencia en el sector.

Una de sus señas de identidad es su calificación como Google Cloud Partner, que la acredita como experta en la nube de Google (Google Cloud Platform), y en el conjunto de servicios en la nube de Google para empresas (G Suite). Esto conlleva una serie de responsabilidades, entre otras, tener que estar al tanto de las nuevas herramientas y servicios que pone a disposición Google, para identificar las que ofrezcan mayores y mejores posibilidades y, a partir de esto, investigar sobre cómo pueden ser utilizadas para el beneficio de la propia empresa y de sus clientes.

En septiembre de 2016, Google adquirió API.AI; pasó a llamarse Dialogflow en octubre de 2017. Desde entonces se ha ido desarrollando e integrando con otros servicios del entorno Google Cloud Platform, proporcionando un conjunto más diverso de posibilidades para los desarrolladores.

Con la aparición de esta herramienta, se han identificado nuevos casos de negocio en los que se pueda utilizar, como lo es en el supuesto del que se ha de interaccionar con usuarios. Los dos casos más frecuentes son: soporte de aplicaciones y planificación de reuniones, y ambos se realizan a través del correo electrónico. Dado que las aplicaciones tienen un servicio propio de *ticketing*, documentación y video tutoriales en YouTube, hemos optado por utilizarlo para la gestión de eventos y consultas básicas sobre los diferentes servicios que comercializa la empresa.

2.2. Objetivos

Los objetivos principales del proyecto son proporcionar a los usuarios información sobre el catálogo de servicios de la empresa de una forma sencilla, así como gestionar las citas con los potenciales clientes de forma automática. Se pretende conseguir una disponibilidad total de la empresa con respecto a sus clientes, especialmente con aquellos que residen en zonas geográficas con una gran diferencia horaria respecto a la nuestra; mejorar el tiempo de respuesta, para que se reduzca a cero; y ahorrar en diferentes recursos que se puedan invertir en otras tareas y/o proyectos.

Para cumplir este objetivo, se ha desarrollado un asistente virtual capaz de entablar conversaciones con los usuarios y guiarlos para obtener la información que desean de la forma más eficiente y rápida. Además, se proporciona una aplicación web para configurar ciertos parámetros del asistente, así como la información que utilizará éste para resolver las consultas de los usuarios.

Dado que algunas de las tecnologías con las que se va a trabajar no se han utilizado antes, o son nuevas en la empresa, el desarrollo del proyecto se va a iniciar con una fase de investigación sobre las posibilidades que ofrecen tanto Dialogflow como Google Cloud Functions para su futuro uso en diferentes proyectos.

2.3. Antecedentes

En 1950 Alan Turing publicó el artículo "*Computing Machinery and intelligence*" [27] en el que se cuestionaba si las máquinas son capaces de pensar. En él describe en qué consiste "*The Imitation Game*", una prueba en la que una persona (interrogador) habla con otras dos, un hombre y una mujer, y ha de determinar cuál de las dos es el hombre y cuál la mujer. Posteriormente, se cuestiona si sustituir al hombre por una máquina que simule su comportamiento tendría efecto en el porcentaje de acierto del interrogador. Esto es lo que se conoce como el Test de Turing, en el cual se considerará que una máquina es capaz de superarlo si el interrogador no puede determinar cuál de las dos entidades es la máquina y cuál la persona.

En 1966, Joseph Weizenbaum creó ELIZA [28], un *chatbot* que identificaba diferentes patrones en las consultas y construía respuestas en base a sus palabras claves.

En la década de 1990 aparecieron Dr. Sbaitso y A.L.I.C.E. (*Artificial Linguistic Internet Computer Entity*) [2]. El primero, mejorado respecto a ELIZA por la utilización de voz digital. El segundo, aplica reglas heurísticas para el reconocimiento de patrones; ha sido reconocido con tres premios Loebner [26] en 2000, 2001 y 2004.

En 2001 surge Smarterchild [21], que se integró en aplicaciones de mensajería y redes SMS. Además, estaba conectado a múltiples bases de datos que le permitían responder preguntas relacionadas con el clima, con películas de cine, con noticias de actualidad, con deportes, etc.

En esta última década, grandes empresas tecnológicas han desarrollado sus propios *chatbots*: Siri (Apple) [6], Alexa (Amazon) [3] y Google Assistant (Google) [11] son algunas de ellas.

En la actualidad, muchas organizaciones optan por desarrollar sus propios *chatbots* para proporcionar servicios personalizados a sus clientes, realizar experimentos sociales [20] o intentar descubrir nuevas aplicaciones que puedan llegar a tener interés en la sociedad [1]. Incluso organizando concursos, como en el caso de Alexa Prize [23], que potencian la investigación en estos temas.

Las tendencias actuales del mercado y las previsiones para los próximos años, tanto sobre la presencia de los *chatbots* [10] como sobre la inversión que se va a realizar en esta tecnología [19], hacen pensar que van a estar cada vez más presentes en nuestro día a día.

2.4. Conceptos técnicos

En este apartado se definirán los conceptos técnicos que se utilizan en el presente documento:

- Computación *serverless*: es un modelo de computación en la nube en la que el proveedor se encarga de la parte relacionada con la administración de recursos. El objetivo es que el cliente centre todos sus recursos en el desarrollo de aplicaciones. Sus principales características son escalabilidad automática y plan de pago por uso.

- Computación en la nube: es un modelo de computación en el cual los servicios que se ofrecen (servidores, bases de datos, software...) se encuentran accesibles a través de Internet. Se distinguen tres tipos de servicios:

    - *Infrastructure as a Service (IaaS):* proporciona los recursos (servidores, almacenamiento, red…) que son gestionados por el cliente.

    - *Platform as a Service (PaaS)*: este tipo de servicios proporcionan una plataforma para desarrollar aplicaciones sin necesidad de que el cliente invierta tiempo y recursos en gestionar la infraestructura que está por debajo.

    - *Software as a Service (SaaS):* son aquellas aplicaciones que se utilizan a través de Internet, como es el caso de Twitter, Google Drive y GMail.

- *Single Page Application* (*SPA*): son aplicaciones que están constituidas por una sola página, la cual se reescribe en función del estado de la aplicación y no es necesario realizar una llamada al servidor cada vez que se cambia de página y se actualiza el contenido que se muestra al usuario.

3. Desarrollo del *chatbot*

En este apartado se describen las distintas fases del proyecto. Se comienza por su planificación y definición de requisitos, y se continúa con el diseño detallado de su arquitectura para conseguir su implementación y el despliegue de la solución.

3.1. Planificación del proyecto

Como herramienta de planificación se ha creado un Diagrama de Gantt utilizando un Google Spreadsheet. En él se han definido las distintas fases del proyecto, las fechas de comienzo y final de cada una, así como las del proyecto global.

Cada mes se ha realizado una revisión del *planning* para comprobar que el proyecto seguía avanzando en los plazos establecidos y, en el caso de que no fuera así, redefinir el tiempo asignado a cada fase para poder cumplir el plazo final.

Como se puede ver en la ilustración 1, se han definido varias fases en el proyecto: Análisis, Arquitectura, Diseño, Desarrollo, Despliegue, junto con las tareas de Documentación, y de Preparación del Trabajo Fin de Máster, necesarias para la presentación del presente trabajo.

Para la fase de desarrollo se ha utilizado la metodología Scrum, dividiéndola en tres *sprints* o iteraciones. Cada uno tiene varios pasos:

1. Reunión inicial con un representante de la empresa, en la cual se deciden las historias de usuario que se van a llevar a cabo.

2. Desarrollo y actualización del estado de las tareas (pendientes, en progreso, en pruebas, o completadas).

3. Pruebas del producto.

4. Revisión del *sprint* para corregir posibles errores en la planificación de futuras iteraciones.

Además, se realizan reuniones frecuentemente (varias veces a la semana) para informar al cliente sobre el estado de las tareas comenzadas y los problemas que han podido surgir durante su desarrollo.

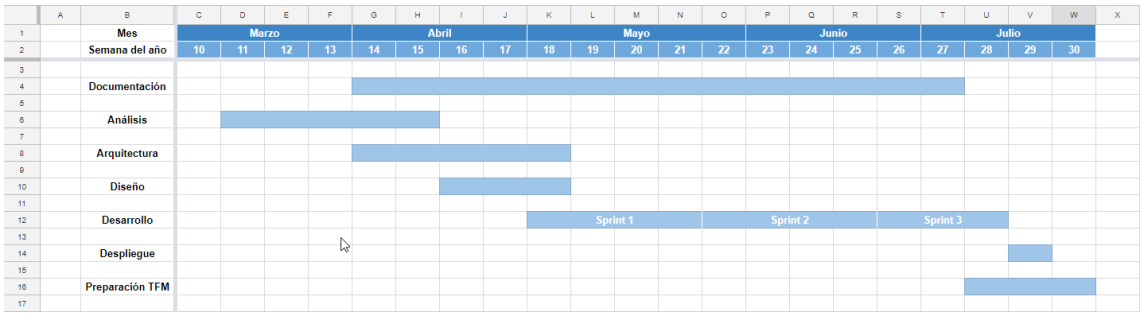


Ilustración 1. Planificación del proyecto.

3.2. Servicios utilizados

Google Cloud Platform es el conjunto de servicios de computación en la nube ofrecido por Google. En particular, los que se han utilizado en este proyecto han sido Dialogflow, Google Cloud Functions, Google App Engine, Google Cloud Datastore y Google Stackdriver. A continuación, describimos resumidamente sus principales características.

3.2.1. Dialogflow

Dialogflow es una herramienta que permite construir agentes conversacionales que interaccionan con los usuarios, bien de forma escrita o bien de forma oral. Se pueden utilizar en tu propia web (son ampliamente utilizados en soporte al usuario), o tu aplicación móvil e integrarlo con otros servicios como Google Assistant, Facebook Messenger y más plataformas de mensajería y/o conversación [7].

3.2.2. Google Cloud Functions

Google Cloud Functions es un servicio de computación *serverless* que forma parte de Google Cloud Platform y está orientado a eventos. Son microservicios que tienen un solo propósito, no son aplicaciones completas [15]. Sus características más destacadas son:

- No es necesario gestionar servidores: simplemente se despliega el código y Google lo gestiona por ti.
- Se paga solo por el tiempo en que tu código está en ejecución.
- Escala automáticamente.
- Ejecuta código en respuesta a eventos: se puede invocar desde Google Cloud Platform, Firebase, Google Assistant y mediante peticiones HTTP.
- Soporta Node.js y Python, junto con sus librerías nativas.
- Se puede conectar a servicios de Google Cloud Platform y a servicios externos.

Uno de los casos de uso más frecuentes es su integración con servicios que utilizan la Inteligencia Artificial de Google, como es el caso de este proyecto; su utilización para realizar procesamiento de datos en tiempo real; y el desarrollo de una *Application Programming Interface* (API) o de la parte de servidor de tu aplicación [16].

3.2.3. Google App Engine

Google App Engine es un servicio *PaaS* que cuenta con las características de computación *serverless* en la nube de Google, anteriormente descritas, y que proporciona una plataforma totalmente gestionada para que el usuario pueda centrar todos sus recursos en el desarrollo de aplicaciones.

Soporta el desarrollo de aplicaciones en Node.js, Java, Ruby, C#, Go, Python y PHP; también lo hace con cualquier librería o *framework* por medio de un contenedor Docker.

Además, permite el versionado de aplicaciones, de forma que el desarrollador pueda tener diferentes entornos para realizar pruebas de las actualizaciones de su aplicación y de la distribución del tráfico entre varias versiones; por ejemplo, para que las actualizaciones de la aplicación se realicen de forma incremental [14].

Google App Engine proporciona dos entornos*: standard* y *flexible*. En nuestro caso, se ha optado por el entorno *standard* principalmente porque la aplicación de configuración no se va a utilizar de forma continuada, y en este entorno puede correr de forma gratuita o a un coste muy bajo, ya que el gasto se mide en horas de uso de instancias de CPU, que pueden llegar a escalar a cero y no generar gasto alguno [12].

3.2.4. Google Cloud Datastore

Google Cloud Datastore es una base de datos NoSQL de tipo documental. En el símil con el modelo relacional, las tablas serían tipos (*kinds*); las filas, entidades (*entities*); las columnas, propiedades (*properties*); y la *Primary Key* sería una *Key* única por entidad.

Las características de esta base de datos son:

- Transacciones atómicas
- Alta disponibilidad en lecturas y escrituras debido a la redundancia que utiliza Google en sus *datacenters*.
- Gran rendimiento en cambios bruscos de tráfico gracias a su arquitectura distribuida y a su escalabilidad automática.
- Múltiples opciones en lenguajes de consulta: de forma natural se adapta a lenguajes orientados a objetos, pero también provee un lenguaje de sintaxis SQL.

- Encriptación automática de los datos antes de ser escritos en disco y posterior desencriptación en las operaciones de lectura.

- Es gestionada por Google, sin tiempo de mantenimiento al recibir actualizaciones.

Además, soporta *multitenancy*. Esto es, almacenar los datos de diferentes organizaciones de forma separada, en particiones denominadas "silos de información" [13].

3.2.5. Google Stackdriver

Google Stackdriver es un servicio de monitorización que incluye la gestión de *logs*, el reporte de errores y un servicio para poder hacer *debug* de tu aplicación. Además, genera automáticamente métricas de rendimiento de la aplicación, soportando la integración con aplicaciones alojadas en Google Cloud Platform o en cualquier proveedor externo [17].

En este proyecto se utilizará para monitorizar tanto la parte de servidor de la aplicación de configuración, como las Cloud Functions que utiliza el agente de Dialogflow.

3.3. Análisis

3.3.1. Requisitos no funcionales

Los requisitos no funcionales sobre la disponibilidad temporal de los servicios vienen marcados por los SLAs (*Service Level Agreement*) de los servicios de Google Cloud Platform utilizados, y son:

- La aplicación web de configuración que utiliza App Engine ha de estar disponible el 99.95% del tiempo.

- Los microservicios alojados en las Cloud Functions tienen un 99.5% de tiempo de disponibilidad.

- La base de datos en Cloud Datastore debe estar disponible el 99.9% del tiempo.

Además, hay otra serie de requisitos:

- El idioma principal del *chatbot* será el inglés.

- La aplicación de configuración también estará en inglés.

- Las ejecuciones de las Cloud Functions deben completarse por debajo de 5 segundos.

3.3.2. Requisitos funcionales

Para definir los requisitos funcionales del *chatbot* se han establecido roles e historias de usuario, esto es, las acciones que puede realizar cada actor.

Roles:

- Administrador del *chatbot*: es el que gestiona el asistente (añadir nuevas funcionalidades, entrenarlo…).
- Mánager del *chatbot*: es el encargado de configurar el asistente para su negocio.
- Usuario del *chatbot*: usuario final que interacciona con el asistente.

Historias de usuario:

- Como mánager:
    - puedo definir el calendario en el que se gestionan las citas,
    - puedo definir los productos disponibles en mi negocio, especificando nombre, descripción, precio, cantidad y sinónimos del nombre del producto,
    - puedo definir los servicios que ofrezco en mi negocio, detallando el nombre, descripción, duración (en minutos), precio y sinónimos del nombre del servicio,
    - puedo definir el horario de mi negocio, y
    - puedo definir eventos especiales en el calendario para indicar que no se pueden concertar citas en ese periodo de tiempo.
- Como administrador:
    - puedo entrenar al asistente,
    - puedo evaluar la eficiencia del asistente, y
    - puedo modificar el asistente añadiendo, modificando o eliminando sus funcionalidades.
- Como usuario:
    - puedo consultar los productos y servicios que oferta la empresa,

- puedo concertar una cita con la empresa, ya sea para un servicio de los ofrecidos en la empresa o por otras circunstancias,
- puedo modificar la fecha de una cita establecida, y
- puedo consultar la disponibilidad para concertar citas en una fecha o rango de fechas concretos.

3.3.3. Investigación

Debido a que tanto Dialogflow como Google Cloud Functions eran tecnologías que no habían sido utilizadas previamente en la empresa, iniciamos una fase de investigación de las mismas, consultando la documentación, ejemplos de agentes ya completados y desarrollando un agente que es capaz de gestionar consultas sobre el clima, y que las resuelve utilizando una API externa.

Esto nos permitió obtener el conocimiento necesario sobre Dialogflow y sus componentes (intentos, entidades, etc), la conexión con las Cloud Functions y el uso de ellas para conectarse a servicios externos.

3.4. Arquitectura

En la ilustración 2 se muestra la arquitectura de la solución que se ha implementado; la constituyen seis módulos: el *chatbot*, los microservicios que utiliza, la aplicación web de configuración, una base de datos para almacenar esta información, las Google Apps for Work y otra base de datos para almacenar las conversaciones.

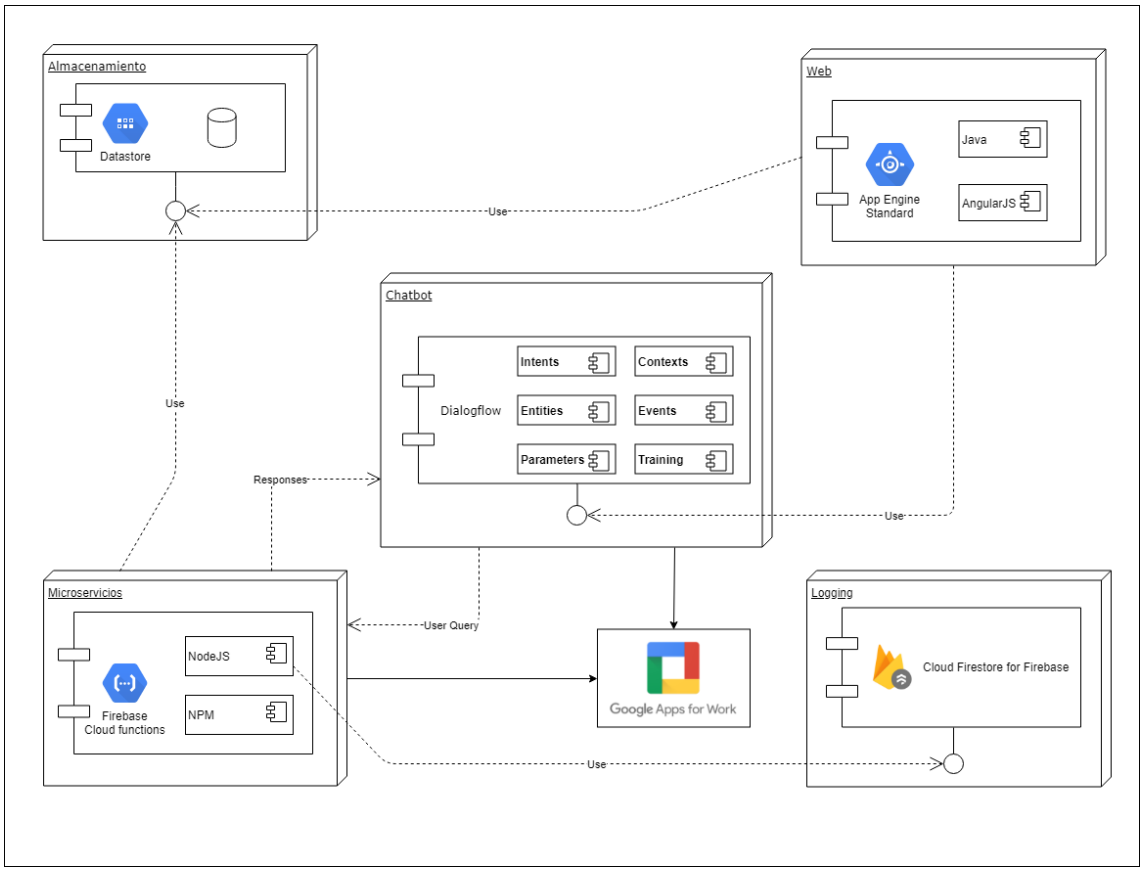


Ilustración 2. Arquitectura de la solución.

A continuación, se muestra un ejemplo de interacción entre los componentes que utiliza el agente. En primer lugar, el usuario realiza una consulta utilizando cualquiera de las plataformas integradas. Después, el agente lanza la llamada correspondiente, que se comunica con las Cloud Functions a través de una petición HTTP. Éstas hacen uso del Datastore, y envían una respuesta al agente (Dialogflow), que es la que se muestra al usuario.

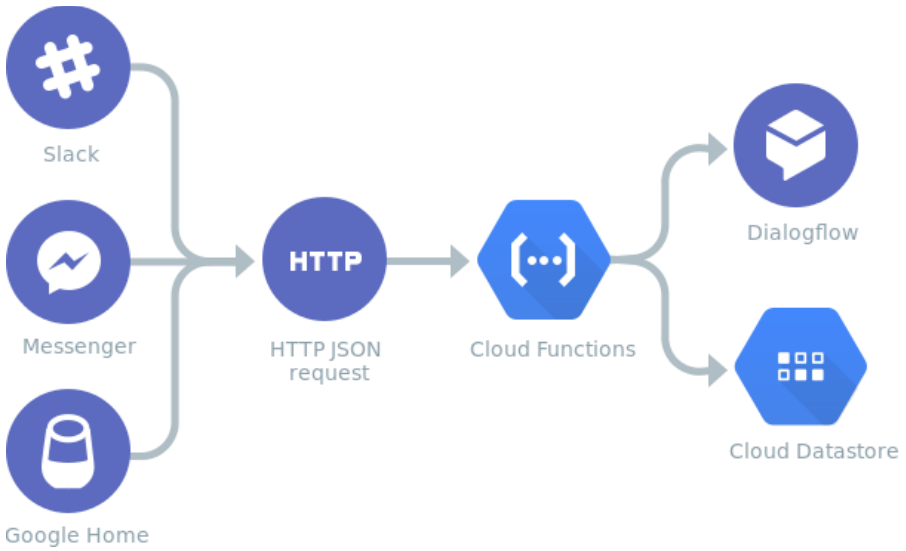


Ilustración 3. Flujo de ejecución de una consulta [16].

3.4.1. *Chatbot*

El *chatbot* es el elemento principal generado utilizando Dialogflow, que es el que proporciona una plataforma para interactuar con los usuarios. Está formado por varios componentes:

- Intentos: representan una acción a realizar por el agente. Pueden ejecutarse porque la consulta del usuario coincide con una de las frases definidas en él, o porque se ha lanzado desde código a través de un evento. En él se definen los parámetros que tiene, los contextos que utiliza y el comportamiento que va a tener, el cual puede ser "estático" (envía una respuesta de texto) o "dinámico" (realiza un procesamiento de los datos que generan un resultado y después envía una respuesta al usuario).

  Además, hay un tipo de intentos, llamados intentos de continuación (*follow-up intents*), que son los que nos permiten definir un flujo de conversación, ya que siguen a un intento "padre". Por ejemplo, cuando el usuario está a punto de completar una operación, una buena práctica es mostrar la información que se va a utilizar junto a una pregunta para confirmar que se desea realizar esa operación. En Dialogflow, tendríamos un intento *A* que recoge los parámetros necesarios y realiza la pregunta, y dos intentos de seguimiento para capturar la respuesta: uno con varias formas de confirmar la operación y otro para rechazar la operación. Estos dos intentos serían "hijos" del intento *A*, y solo se lanzarían cuando se detectara una de las frases que tiene configurada en una respuesta a la ejecución de su intento "padre". Nunca se ejecutaría si su "padre" no lo ha hecho.

- Entidades: sirven para extraer información de la consulta del usuario. Hay entidades definidas por el sistema; por ejemplo, la entidad @sys.date detecta cuando un usuario introduce una fecha en su consulta; y otras definidas por el desarrollador.

  A la hora de crear una entidad, hay que distinguir tres tipos:

  - Entidades enumeradas: son una lista de entradas que solo pueden tener un valor concreto.

  - Entidades con mapeos: son una lista de entradas que asocian varios valores a un valor de referencia.

  - Entidades compuestas: son las formadas a partir de dos o más entidades, ya sean definidas por el sistema o por el desarrollador.

- Parámetros: objetos que guardan información útil en los intentos. Tienen un nombre único, un valor, un valor por defecto opcional, y la posibilidad de

definirlos como obligatorios. Es posible definir mensajes de aviso que se utilizarán en caso de que un parámetro obligatorio no tenga valor cuando el intento se lance para que el usuario introduzca la información necesaria.

Su valor puede venir definido por un contexto, un evento o ser obtenido de la consulta del usuario gracias a las técnicas de Procesamiento de Lenguaje Natural que aplica Dialogflow.

- Contextos: objetos que permiten almacenar información dentro de una misma sesión para utilizarla a lo largo de la conversación. Se almacenan en un objeto JSON que se pasa entre los intentos, y que tiene un tiempo de validez.

- Eventos: se utilizan para lanzar intentos desde código, sin que el usuario haga ninguna consulta. Se pueden pasar parámetros a través de ellos al intento que se va a ejecutar.

- Respuestas: el objeto que se utiliza para enviar una respuesta está compuesto por dos elementos. Uno es un texto simple que se convertirá a audio. El otro es la respuesta que se mostrará al usuario en la pantalla. Esta puede ser también un texto simple o un mensaje enriquecido.

  Estos mensajes pueden ser imágenes, audios, sugerencias o cartas (que generalmente están compuestas por una imagen, un título, un texto y botones para realizar alguna acción). En la ilustración 4 se muestra un mensaje de texto simple, una carta y un mensaje que consta de dos sugerencias.

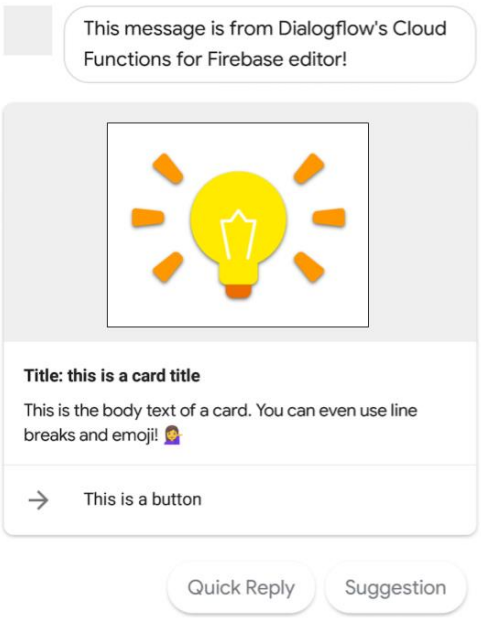


Ilustración 4. Ejemplos de mensajes enriquecidos.

- Entrenamiento: los agentes creados en Dialogflow están aprendiendo siempre. Para gestionar las consultas introducidas existe una sección de entrenamiento en la herramienta, en la que se muestran todas las consultas realizadas, el comportamiento que ha tenido el *chatbot* (intento ejecutado, parámetros detectados en la consulta…) y la posibilidad de confirmar que la detección ha funcionado correctamente, o de modificar si se ha hecho de forma incorrecta, o de eliminar la consulta para que no la tenga en cuenta el agente.

  El aprendizaje de los agentes se realiza utilizando algoritmos de *Machine Learning* que se van actualizando en base a las frases de entrenamiento y a los modelos de lenguaje configurados en los intentos, por lo que terminan siendo algoritmos personalizados para un agente concreto. Estos algoritmos se utilizan para entender las consultas en lenguaje natural, extraer información estructurada y lanzar el intento correspondiente a cada consulta [8].

3.4.2. Microservicios

Este componente es utilizado por el agente para realizar el procesamiento de la información en la mayoría de los intentos. Recibe los datos a través de una petición HTTP a una URL que se configura en el Dialogflow y envía una respuesta con un formato concreto. Tiene 256 MB de memoria reservada para su ejecución. Utiliza información almacenada en el Datastore, y se conecta al Google Calendar configurado a través de su API. Para ello, es necesario realizar un proceso de autenticación previo utilizando el protocolo OAuth 2.0 [18, 24].

Todos los servicios externos de los que hacen uso los microservicios son accesibles utilizando librerías de cliente gestionadas en NPM (*Node Package Manager*), y el código corre en el entorno Node.js 6 [22].

3.4.3. Aplicación web de configuración

Con el objetivo de configurar el *chatbot* de una forma sencilla e intuitiva, se ha creado esta aplicación de configuración. Desde ella el responsable del negocio configura los parámetros que utilizará el asistente para gestionar las citas y el catálogo de productos y servicios que se ofrecen.

Esta aplicación está alojada en Google App Engine Standard, y utiliza Java8 como entorno de ejecución en la parte de servidor y una SPA (*Single Page Application*) desarrollada en AngularJS 1.6.7 [5, 25] en la parte de cliente.

La parte de servidor se ha configurado para que realice un escalado automático de las instancias. Estas instancias son del tipo F1, y tienen un límite de memoria de 128 MB y de CPU de 600MHz.

3.4.4. Base de datos

Para almacenar la información que introduce el cliente, y que será utilizada por el asistente, se ha utilizado una base de datos en Cloud Datastore. Tiene cuatro *kinds* (símil de las tablas en el modelo relacional): productos, servicios, información del usuario y entidades.

En lo referido a los productos, se almacenan el nombre, la descripción, el precio, la cantidad y los sinónimos. En este caso, dado que los productos son productos software, la cantidad hace referencia al número de licencias.

La información almacenada de los servicios es similar a la de los productos. Solo difieren en que los servicios tienen una duración (en minutos) en vez de una cantidad. En ambos *kinds*, los IDs de las *entities* (símil de las filas) son autogenerados.

En cuanto a la información del usuario, se almacena el email, el ID del Calendario de Google y la zona horaria que ha seleccionado, un objeto de configuración del horario del negocio en formato JSON y un *refresh token*. Este *token* permite generar nuevos *access token* y utilizarlos para personificar al usuario y realizar acciones en su nombre de forma *offline* (sin que esté presente).

Por último, el *kind* Entidades, que almacena el ID y un "nombre" de las entidades "Productos" y "Servicios" creadas en Dialogflow. Cuando el cliente modifica los datos de algún producto o servicio, se actualizan las entidades en el Dialogflow con los nuevos valores. Para ello hay que enviar una petición HTTP a la API de Dialogflow en la que hay que indicar el "nombre" de la entidad a actualizar.

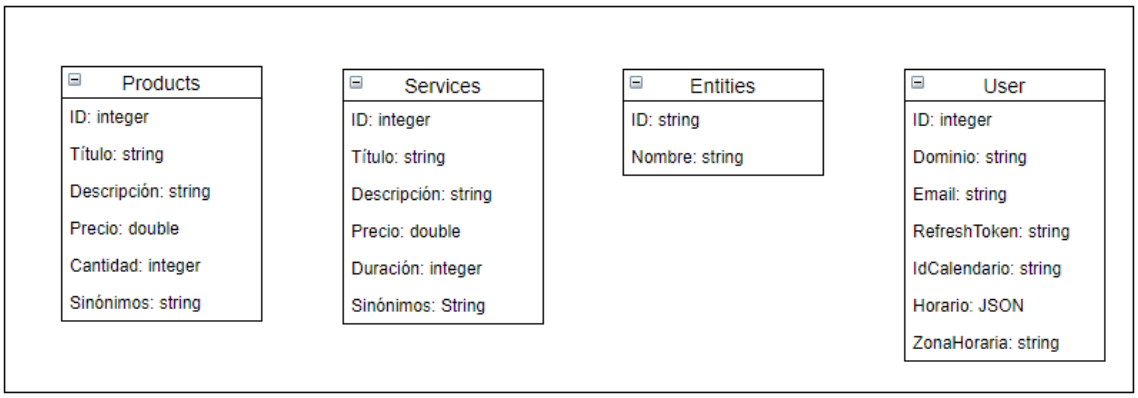


Ilustración 5. Arquitectura de la base de datos.

3.5. Diseño

La fase de diseño consistirá en detallar los diferentes flujos de conversación que puede tener el agente con los usuarios finales y, a partir de esto; realizar unos *wireframes* que sirvan para tener un boceto del diseño de la aplicación web de configuración del asistente.

3.5.1. Flujos de conversación

Para poder definir los intentos y las entidades en el agente de Dialogflow, así como el código que será necesario en cada uno de ellos, hay que especificar cuáles son los flujos de conversación que puede generar el usuario. Cada flujo se corresponderá con una de las funcionalidades del *chatbot*, como, por ejemplo, que el usuario pueda editar una cita. En los siguientes apartados se describirán los diseños realizados, junto con imágenes que ejemplifican diferentes conversaciones entre un usuario (en amarillo) y el agente (en verde).

Obtener información genérica

En la ilustración 6 se muestra una conversación de un usuario consultando la información del negocio y, posteriormente, la de un producto y la de un servicio. El flujo de conversación para obtener información genérica engloba también los flujos de consultar un producto (sombreado en azul) y un servicio (sombreado en naranja), los cuales pueden ser lanzados directamente realizando la consulta 2 para obtener información de un servicio o la 3 para un producto, sin necesidad de pasar por la consulta 1. Asimismo, la consulta de un servicio contiene el flujo de creación de un evento (sombreado en morado).

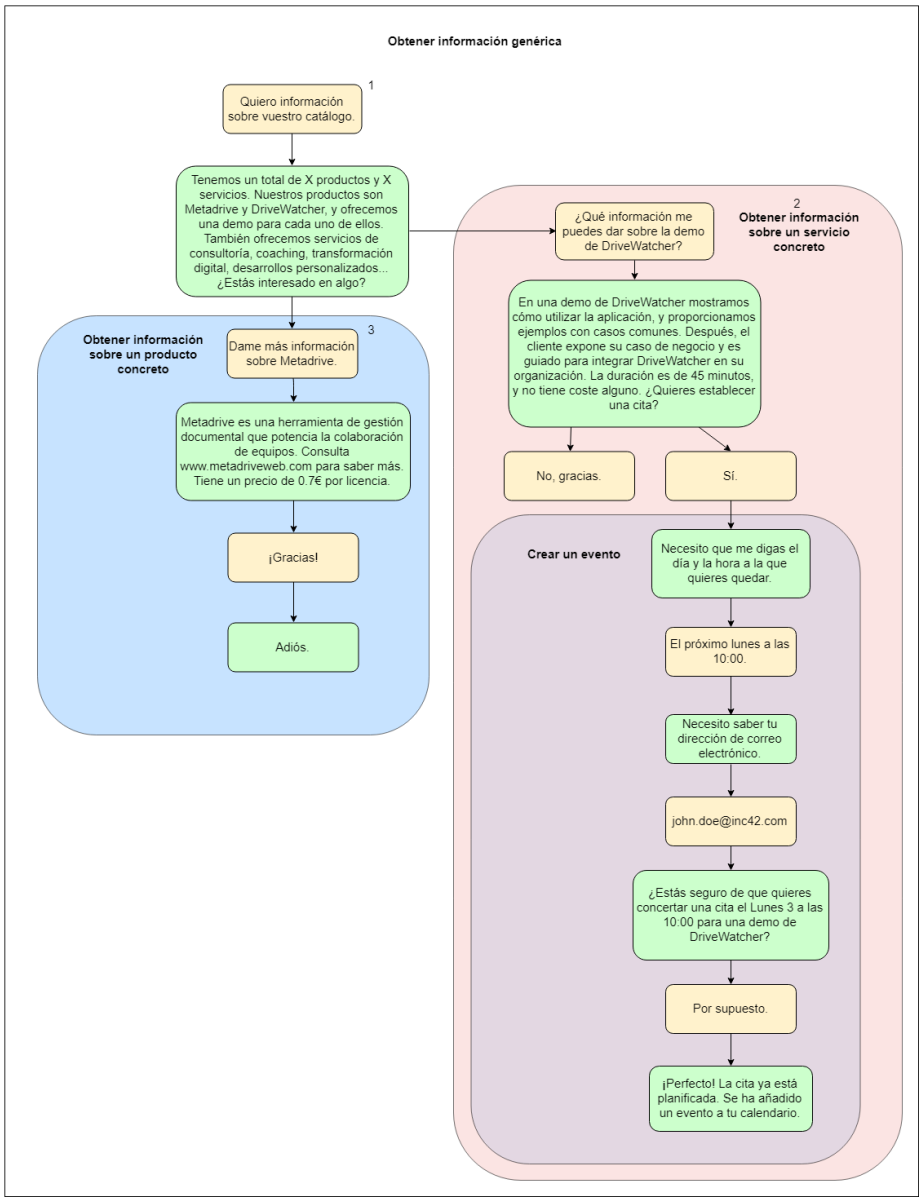


Ilustración 6. Flujo de conversación de la consulta del catálogo.

Concertar una cita

En este caso la conversación sirve para que el usuario obtenga una reunión con el negocio. Para ello el agente recogerá los datos necesarios: fecha y hora del evento, correo electrónico del usuario y un asunto para la reunión. Es necesario comprobar que la fecha es válida: que sea posterior al día actual, que se encuentre en el horario de disponibilidad del negocio, y que no haya ningún otro evento creado. En caso de que no sea posible la reunión, el usuario será informado y podrá sugerir otra fecha o consultar la disponibilidad para que se muestren distintas sugerencias de fechas. Como se puede observar, este flujo también contiene el flujo de creación de un evento (sombreado en morado).

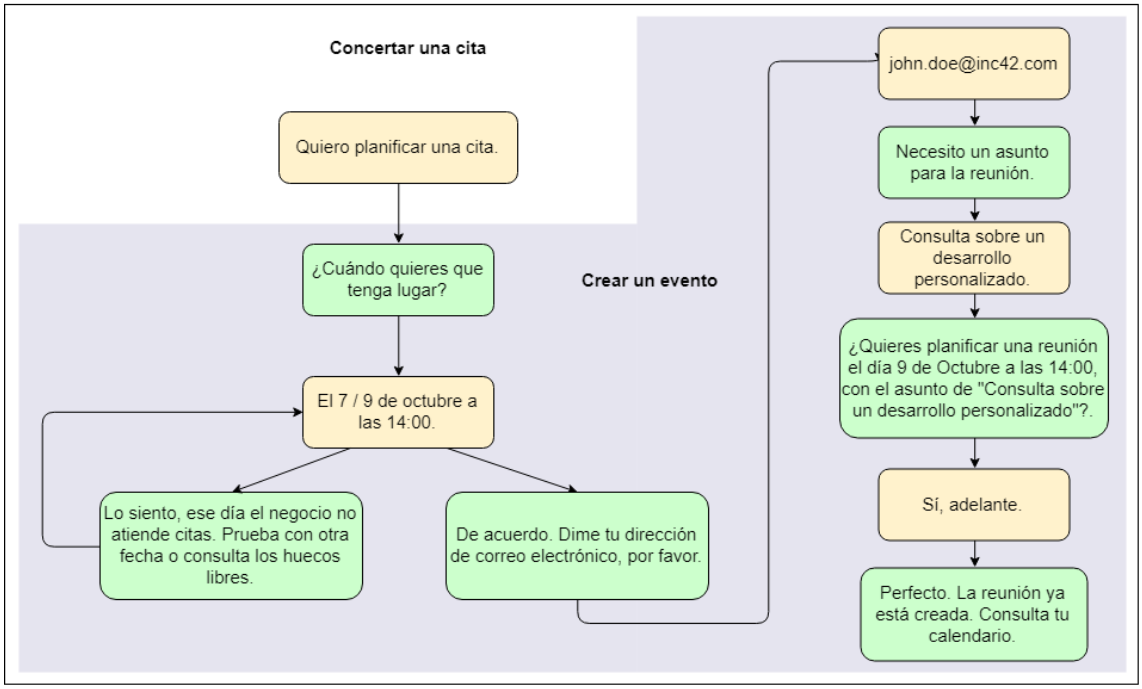


Ilustración 7. Flujo de conversación para concertar una cita.

Editar una cita existente

Este flujo de conversación es un ejemplo en el cual el usuario ya ha acordado previamente una cita con el negocio, y desea modificar la fecha en la que tendrá lugar. El agente le pide que se identifique, le muestra las citas que tiene asignadas, y el usuario selecciona cuál de ellas es la que desea cambiar. A continuación, el usuario ha de proponer fechas que sean válidas y en las que el negocio no haya creado ningún evento.

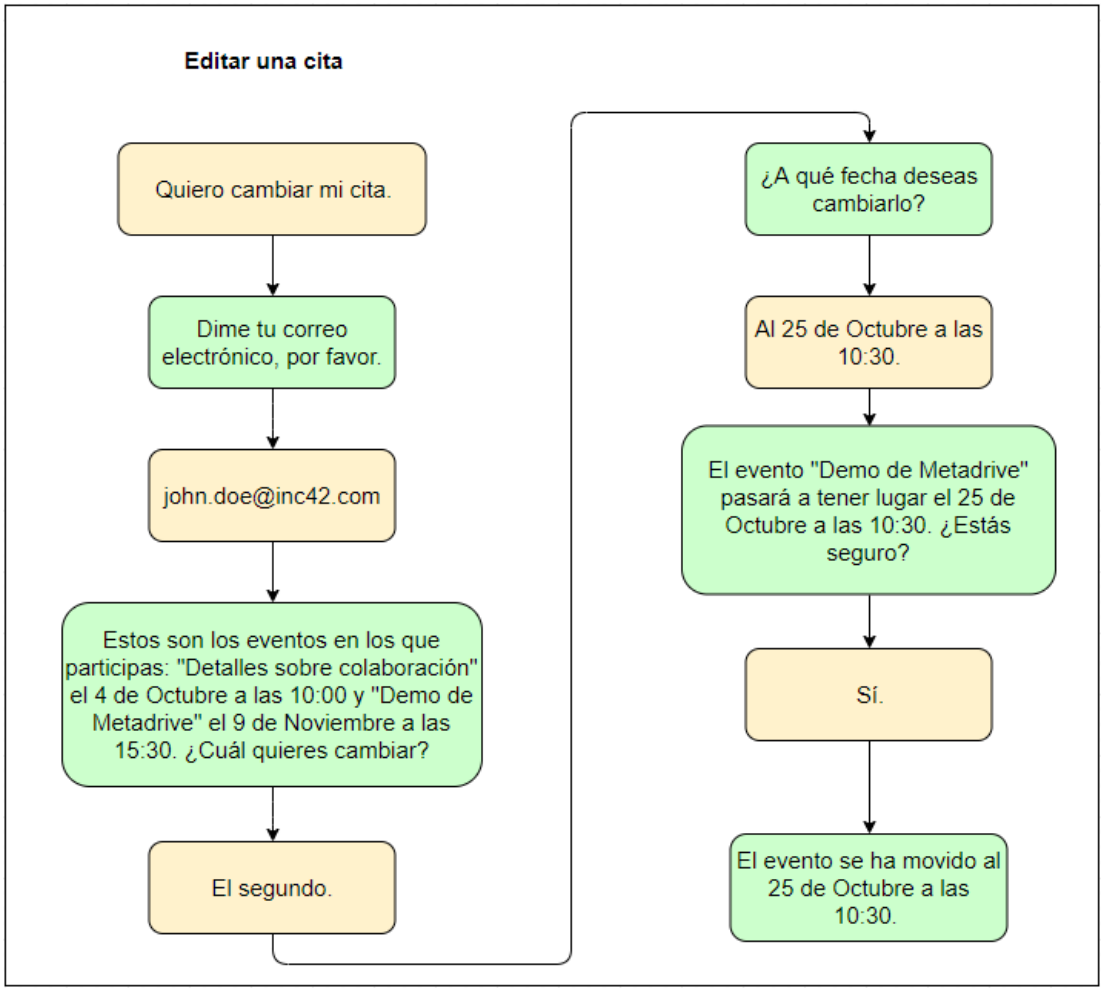


Ilustración 8. Flujo de conversación para editar una cita existente.

Consultar disponibilidad

Uno de los problemas que el usuario se puede encontrar a la hora de planificar un evento es el de elegir una fecha en la cual tanto él como el responsable del negocio estén disponibles. Por esta razón, podrá pedirle al agente que le muestre la disponibilidad del negocio. El agente responderá con los tres primeros espacios, si los hubiere, permitiendo al usuario seleccionar uno de ellos para reservarlo. El usuario también tiene la posibilidad de pedir más resultados al agente, que le devolverá los tres siguientes, hasta que ya no haya más espacios para las fechas indicadas.

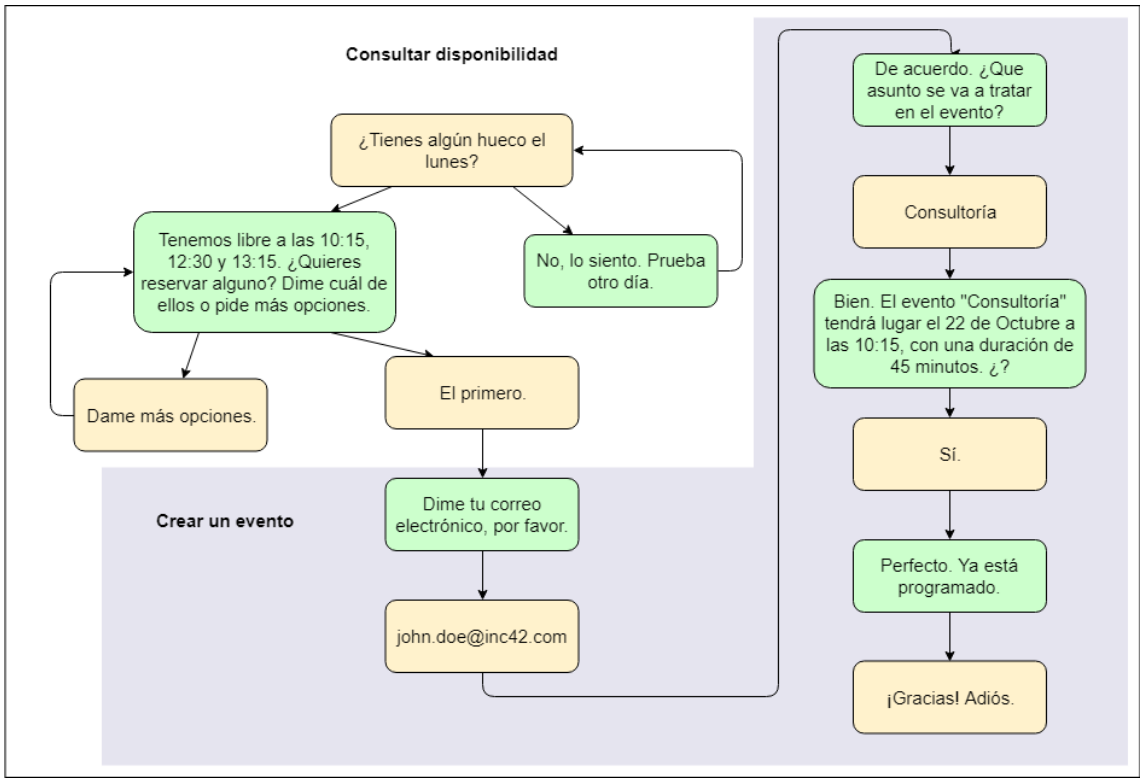


Ilustración 9. Flujo de conversación para consultar la disponibilidad.

3.5.2. Aplicación de configuración

A continuación, se mostrará en varios *wireframes* el diseño de la aplicación que permite al cliente gestionar la información que utilizará el asistente.

En primer lugar, se describe la gestión del catálogo de productos y servicios. La interfaz será la misma en ambos casos, variando simplemente la información que se muestra de cada objeto. El cliente podrá añadir nuevos objetos, y editar y eliminar los existentes.

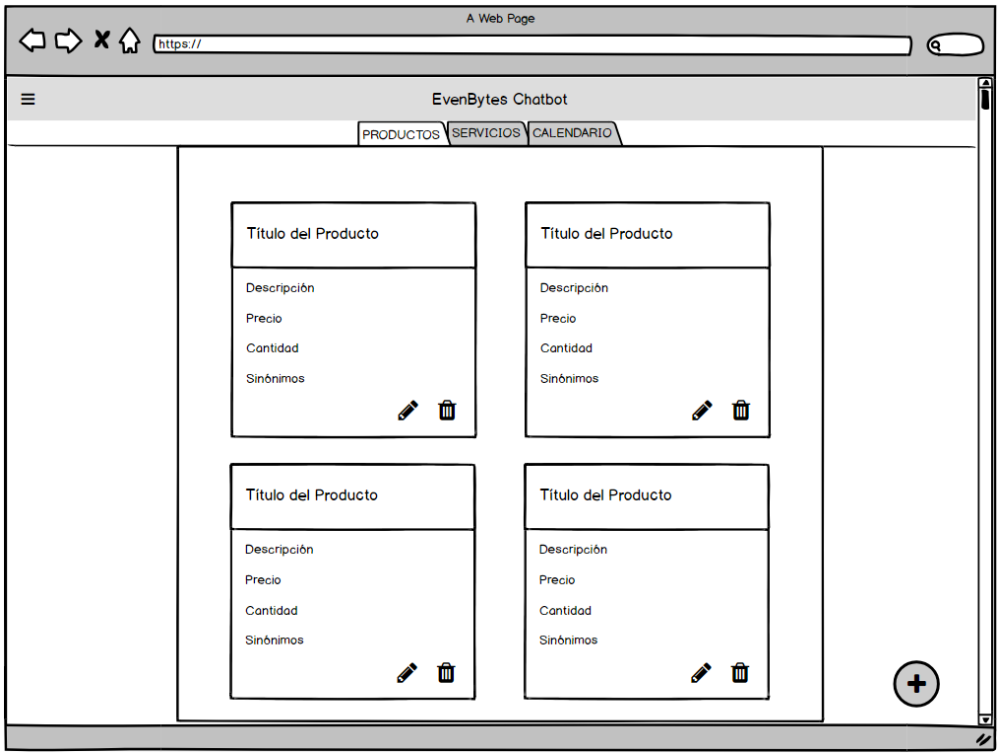


Ilustración 10. Diseño de la gestión de productos.

En caso de que el cliente edite o cree un objeto, se mostrará un formulario en una ventana emergente con los campos correspondientes.

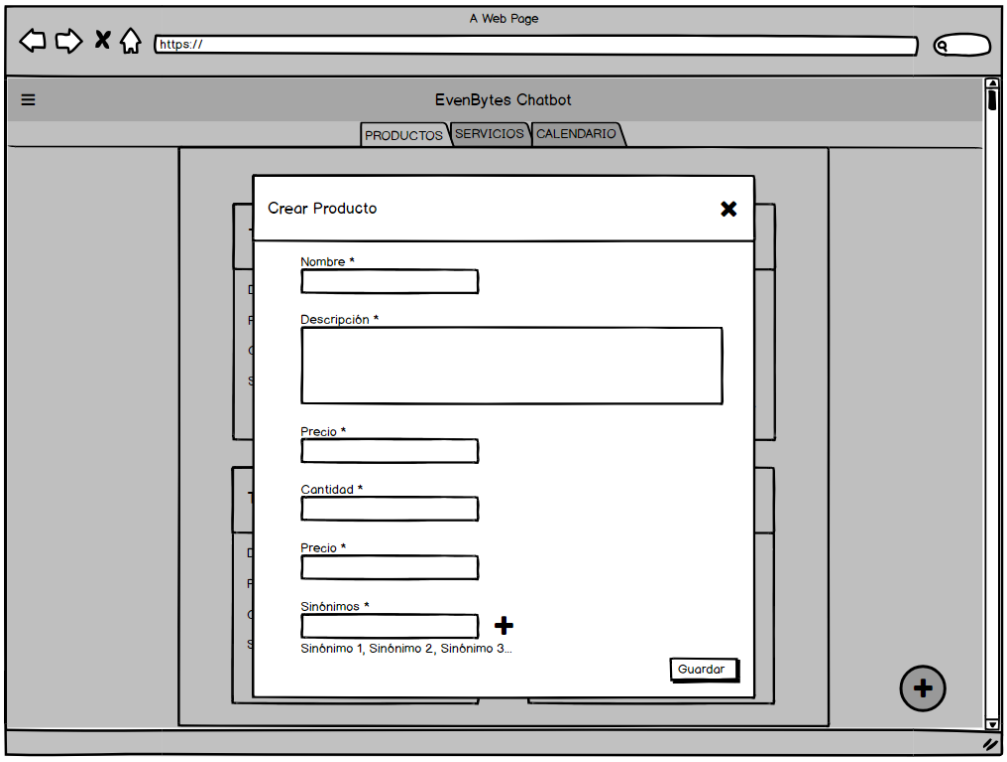


Ilustración 11. Diseño de la gestión de servicios.

En la pestaña de configuración del calendario, el cliente podrá seleccionar cuál de sus Calendarios de Google quiere utilizar para gestionar las citas, su zona horaria, y el horario del negocio. En el último caso, el usuario podrá configurar el horario de apertura y cierre de cada día de la semana. En caso de que no todos los días tengan el mismo horario, podrá añadir nuevas configuraciones para cubrir todas las combinaciones posibles.

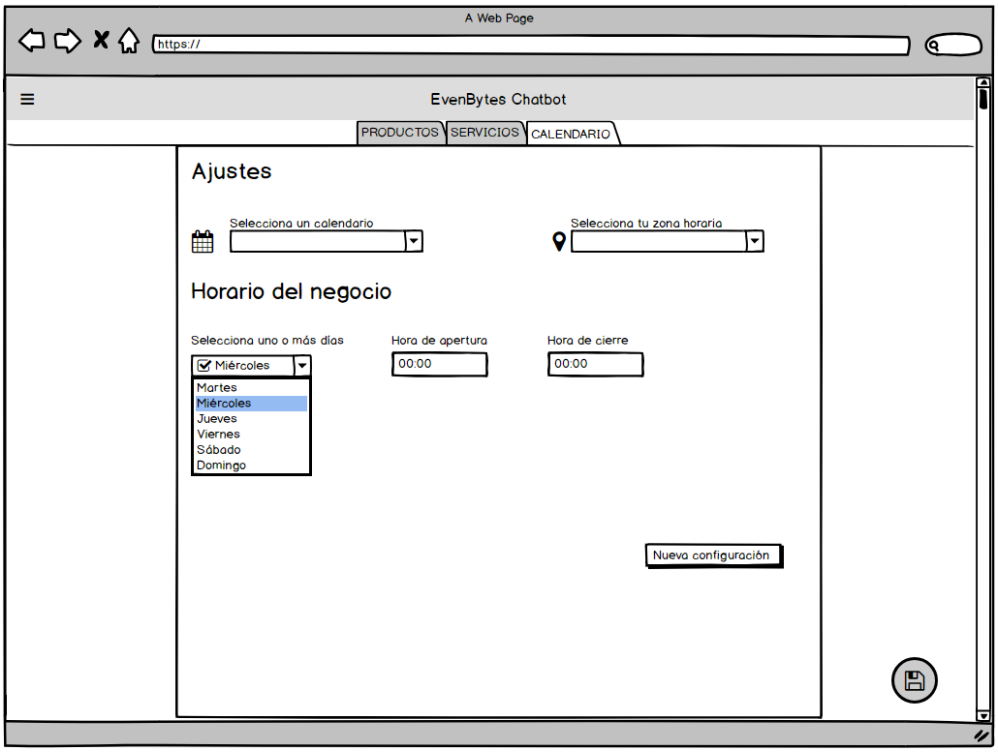


Ilustración 12. Diseño de la configuración del calendario.

Para manipular la información almacenada en la base de datos, se hará uso de servicios REST, descritos en la tabla 1:

Método	**URL**	**Parámetros**
GET	/user	IdUsuario
POST	/user	IdUsuario, IdCalendario, Horario, ZonaHoraria
GET	/products	
POST	/products	IdProducto, Precio, Cantidad, Descripción, Sinónimos
DELETE	/products	IdProducto

GET	/services	
POST	/services	IdServicio, Precio, Duración, Descripción, Sinónimos
DELETE	/services	IdServicio
GET	/entities	
POST	/entities	IdEntidad, Nombre
POST	/oauth2callback	Código

Tabla 1. Servicios REST.

3.6. Desarrollo e instalación

3.6.1. Dialogflow

Microservicios

Para que los intentos puedan utilizar los microservicios, es necesario definirlos previamente en la configuración del agente. Las Cloud Functions que se van a utilizar son del tipo Funciones HTTP. Es decir, son funciones que, al ser desplegadas, proveen de una URL para invocarlas.

Cuando se ejecute un intento que tiene activada la opción de utilizar los microservicios, Dialogflow enviará una petición POST a esta URL, que se encargará de gestionar la información recibida, generar una respuesta y enviarla.

Por otro lado, el despliegue de las Cloud Functions, se realizará con la herramienta Google Cloud SDK, utilizando el comando $gcloud\ functions\ deploy\ <nombre>\ --trigger-http$, en el que el parámetro nombre será el nombre de la función principal, que se encarga de gestionar la petición recibida, autentificar al usuario y dirigir la petición a la función que se corresponde al intento que envía la petición. La opción $--trigger-http$ indica que la función será accesible a través de peticiones HTTP.

En cuanto al código, las Cloud Functions tienen dos archivos principales, que son el *package.json* e *index.js*. El primero es un objeto JSON en el cual se especifica información del proyecto, como una descripción, la dirección del repositorio y el tipo de licencia. También almacena las dependencias del código, especificando el paquete y la versión, para que al desplegarlo se actualicen automáticamente y se pueda ejecutar el código sin problemas. En este proyecto los paquetes que se utilizarán serán la librería de cliente de Google Cloud Datastore, una librería de Dialogflow para gestionar más fácilmente las

peticiones del agente, la librería de las Google APIs para autentificar y realizar llamadas a las APIs de Google, y las librerías de *Moment* y *Moment Timezone* que facilitan la manipulación de fechas.

Ilustración 13. Archivo package.json.

En el archivo *index.js* está la función principal, que es la que se encarga de gestionar las peticiones HTTP que llegan.

Primero, autentifica al usuario utilizando el servicio de OAuth2 de Google. Para ello se crea un objeto OAuth2Client con las credenciales de la aplicación (*clientId*, *clientSecret* y *redirectUri*). Para almacenar y utilizar estos datos se ha creado un servicio (*credentials*), evitando así tener las credenciales de la aplicación especificadas en el fichero principal, que se puede reutilizar para otras aplicaciones. Después, se añade el *refreshToken* que genera la aplicación de configuración cuando el cliente inicia sesión.

Una vez que el objeto de autentificación está creado, se idea un objeto *WebhookClient*, que representa la consulta del agente. A continuación, se genera un mapa que tiene como claves los nombres de los intentos, y como valor la función que se ha de ejecutar para cada uno.

Por último, se llama a la función *refreshAccessToken*, que genera un *accessToken* y lo almacena en el objeto de autentificación. Este *token* es el que sirve para ejecutar las llamadas autentificadas a las APIs de Google. Cuando esta función ha terminado, se ejecuta la función *handleRequest* del objeto *agent*, que lanza la función correspondiente al intento que ha capturado la consulta del usuario. Ésta se encarga de procesar la información recibida, generar una respuesta y enviarla.

```
const {google} = require('googleapis');
const OAuth2Client = google.auth.OAuth2;
const {WebhookClient} = require('dialogflow-fulfillment');
const credentials = require('./credentials_service.js');

function mapAgent(req, res) {

    const oAuthClient = new OAuth2Client(credentials.getClientId(), credentials.getClientSecret(), credentials.getRedirectUris()[0]);
    oAuthClient.setCredentials({
        refresh_token: refreshToken
    });

    const agent = new WebhookClient({
        request: req,
        response: res
    });

    const intentMap = new Map();

    intentMap.set('appointDate', appointDate);
    intentMap.set('askGenericInfo', askGenericInfo);
    intentMap.set('queryFreeSpots - more - more', queryFreeSpotsMuchMoreResults);
    intentMap.set('queryFreeSpots - select.number', queryFreeSpotsSelectNumber);
    intentMap.set('queryFreeSpots - more - select.number', queryFreeSpotsSelectNumber);
    //...all the intents with their corresponding function

    oAuthClient.refreshAccessToken().then(() => {
        agent.handleRequest(intentMap);
    });
}
```

Ilustración 14. Función principal de las Cloud Functions.

Entidades

El *chatbot* utilizará tres entidades personalizadas, todas entidades con mapeos, independientemente de algunas entidades del sistema:

1. Tipo: tendrá dos entradas para detectar cuándo el usuario quiere consultar los productos y cuándo los servicios. En la ilustración 15 se muestra la configuración.

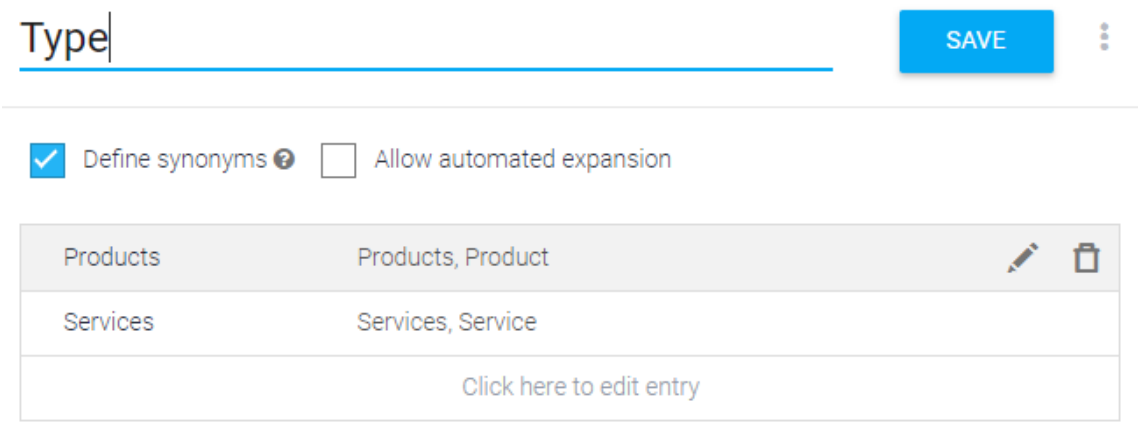


Ilustración 15. Entidad "Tipo".

2. Productos: esta entidad tendrá una entrada por cada producto definido por el mánager del asistente a través de la aplicación de configuración, en la que introducirá su nombre y los sinónimos.

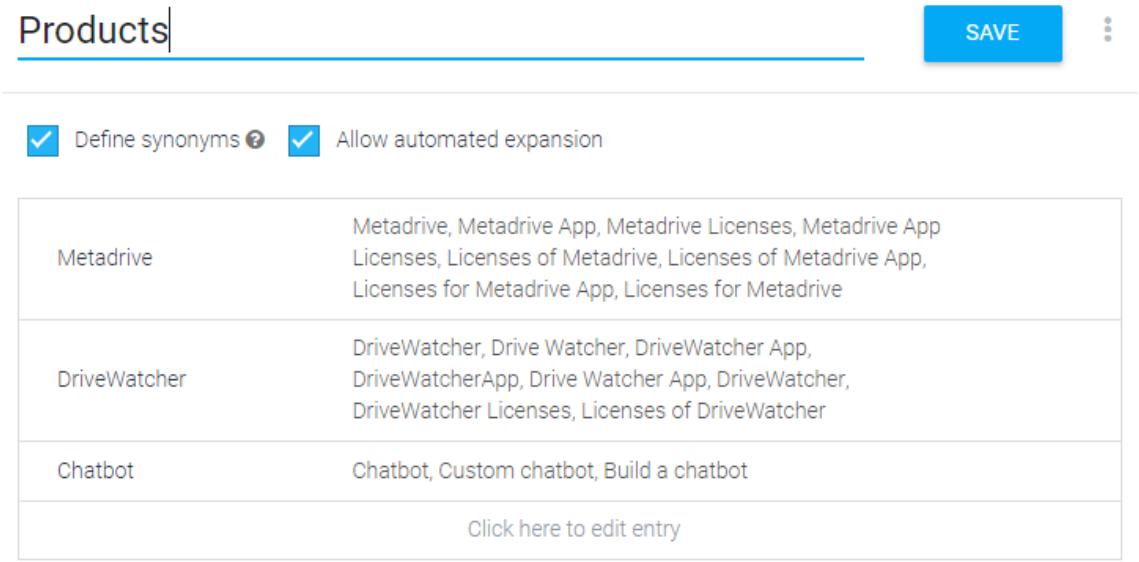


Ilustración 16. Entidad "Productos".

3. Servicios: misma configuración que la entidad “Productos”, pero con los servicios definidos en el negocio.

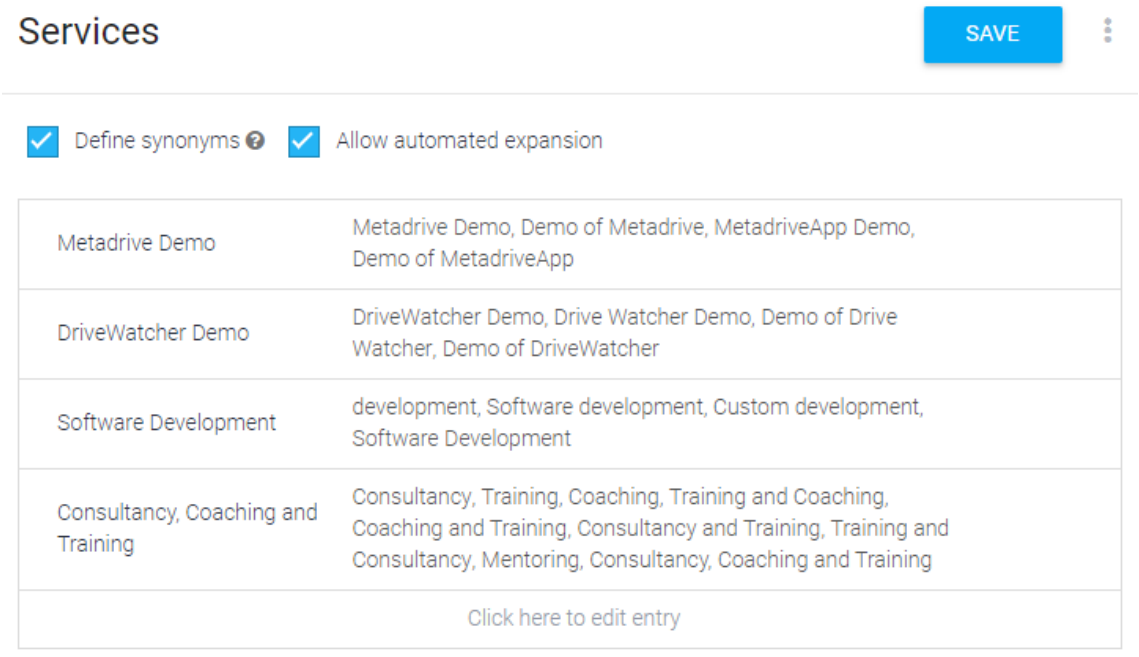


Ilustración 17. Entidad "Servicios".

Las entidades “Productos” y “Servicios” se actualizarán automáticamente cada vez que el mánager realice un cambio en la información almacenada en la base de datos, de forma que pueda añadir fácilmente nuevos elementos al catálogo del negocio. Además, tienen activada la opción “*Allow automated expansion*” (permitir expansión automatizada), para que, si el agente detecta nuevas entradas de las entidades, las añada automáticamente.

Intentos

En la ilustración 18 se pueden observar los intentos "padres" del agente, entre los cuales se encuentran dos intentos especiales, los únicos que no utilizan las Cloud Functions: el intento de bienvenida (*Default Welcome Intent*) y el intento de error (*Default Fallback Intent*). Esta circunstancia se debe a que el comportamiento de estos intentos no requiere ningún procesamiento de información, simplemente se configuran las respuestas que utilizará el agente en la herramienta Dialogflow.

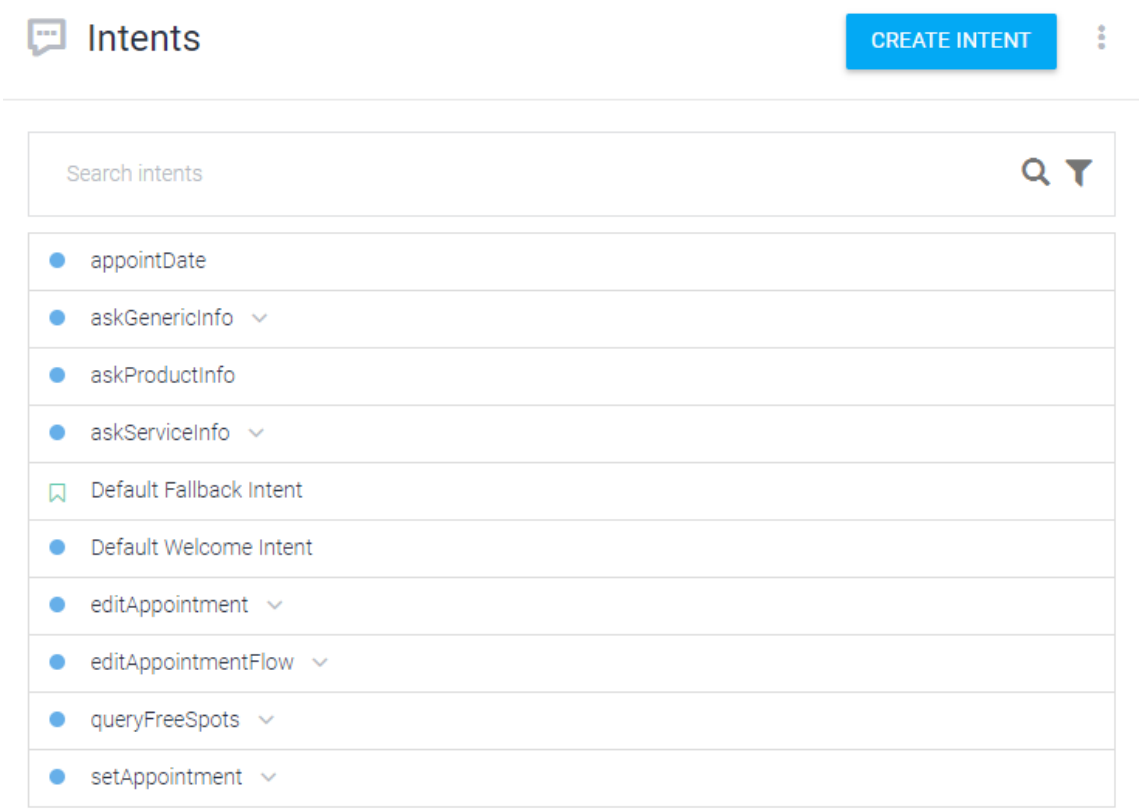


Ilustración 18. Intentos del agente.

Intentos de bienvenida y de error

En primer lugar, el intento de bienvenida, que se lanza cuando el usuario comienza la conversación con cualquier tipo de saludo, y que está configurado con cuatro posibles respuestas. En algunas de ellas (las más utilizadas), aporta información sobre qué puede hacer el usuario, con el objetivo de guiarle en su siguiente consulta.

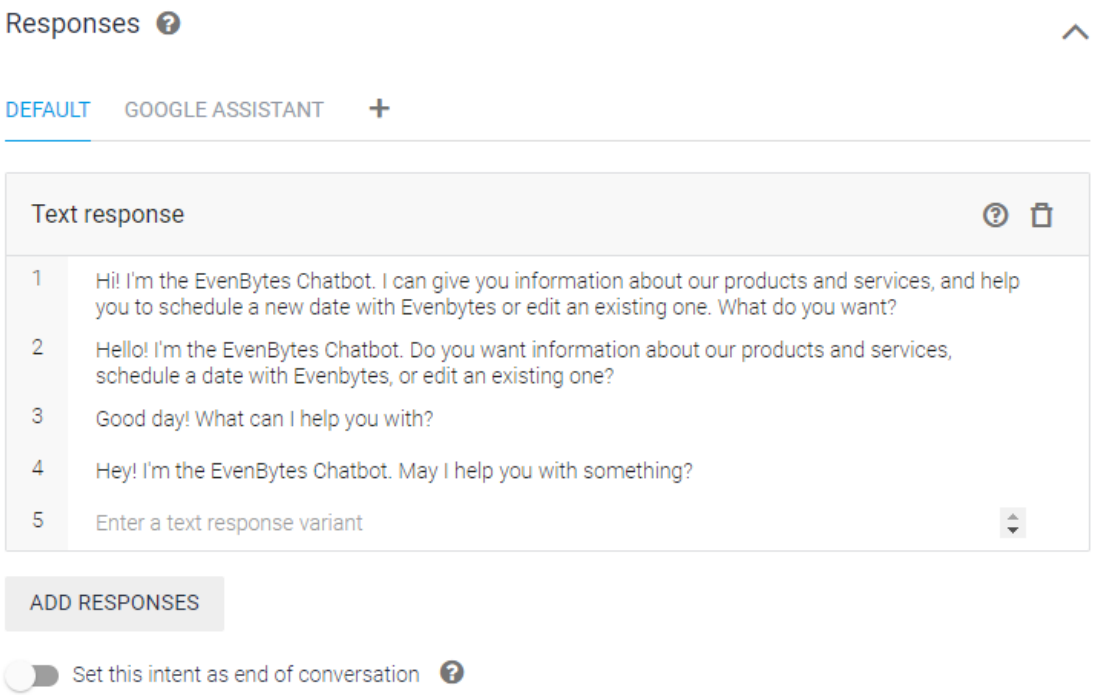


Ilustración 19. Respuestas al intento de bienvenida.

En segundo lugar, el intento de error. Cada vez que el usuario realiza una consulta, Dialogflow utiliza técnicas de Aprendizaje Automático y Procesamiento de Lenguaje Natural para asignar una puntuación entre 0 y 1 a cada intento en base a la similitud entre la consulta y las frases definidas en la configuración del intento.

En la configuración del agente hay un apartado de Ajustes de Aprendizaje Automático en el que se selecciona qué técnica utilizar, el umbral de confianza para lanzar el intento de error y activar o desactivar la corrección de errores ortográficos en la consulta del usuario. En nuestro agente, elegimos que utilice técnicas de Aprendizaje Automático, establecemos el umbral de confianza en 0.9 y activamos la corrección de errores.

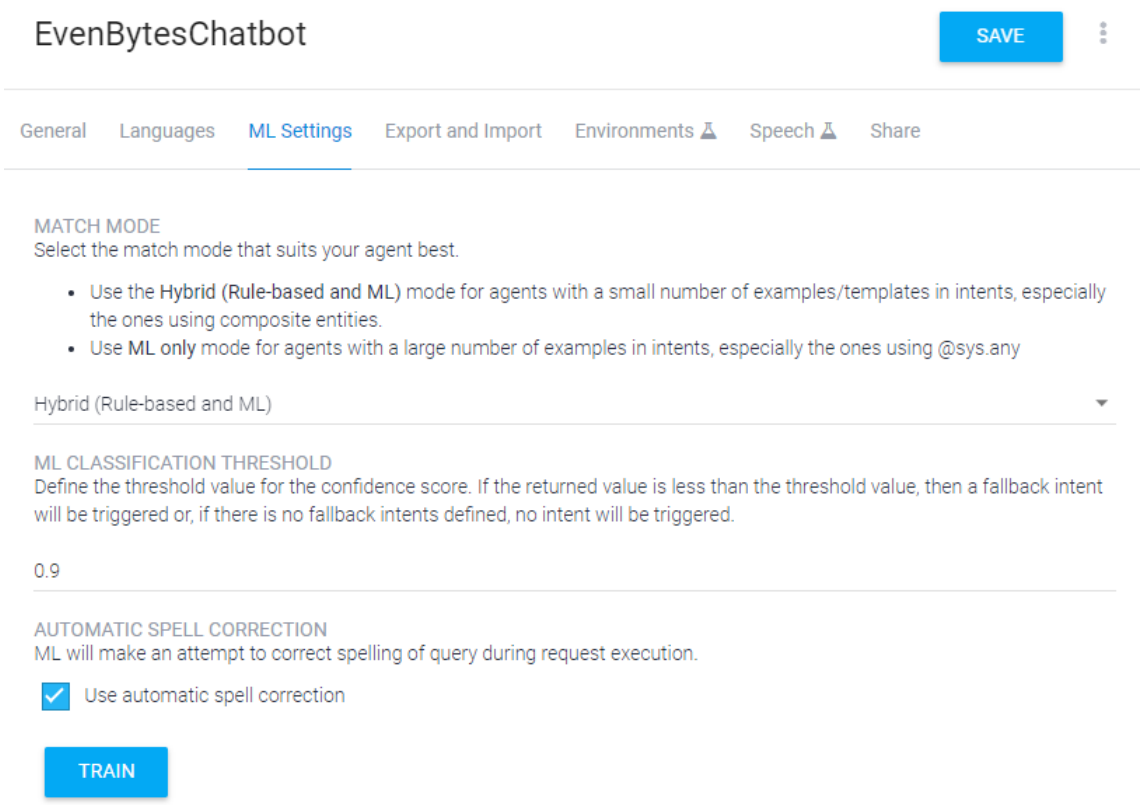


Ilustración 20. Ajustes de aprendizaje automático.

En caso de que una consulta no obtenga una puntuación mayor o igual a 0.9 en algún intento, se lanza automáticamente el intento de error. En él se definen varias respuestas en las que se indica al usuario las consultas que puede realizar. Siguiendo las buenas prácticas de Dialogflow, en los intentos como éste (que no necesitan utilizar los microservicios) se definen varias respuestas para que el agente pueda contestar de varias formas diferentes. La respuesta elegida en cada ejecución es seleccionada de forma aleatoria.

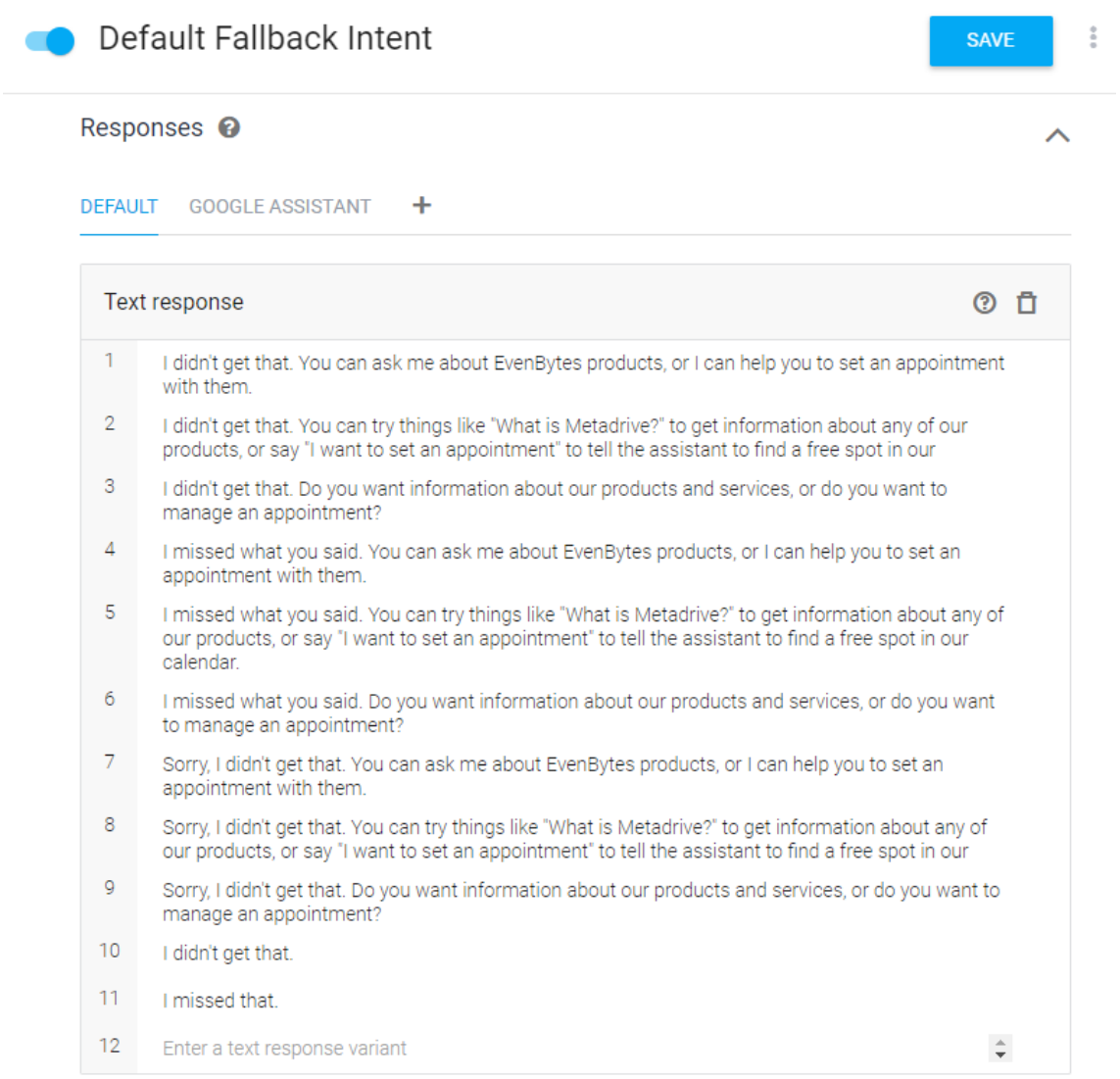


Ilustración 21. Configuración del intento de error.

Una vez iniciada la conversación, hay varios flujos de conversación que el usuario puede lanzar: pedir información genérica de la empresa, consultar los productos, consultar los servicios, programar una cita, editar una cita ya existente y consultar los huecos libres para un rango de fechas.

Obtener información genérica

Dado que este flujo de conversación tiene varias ramas, la estructura será un intento padre que tiene varios intentos *follow-up* en cada nivel.

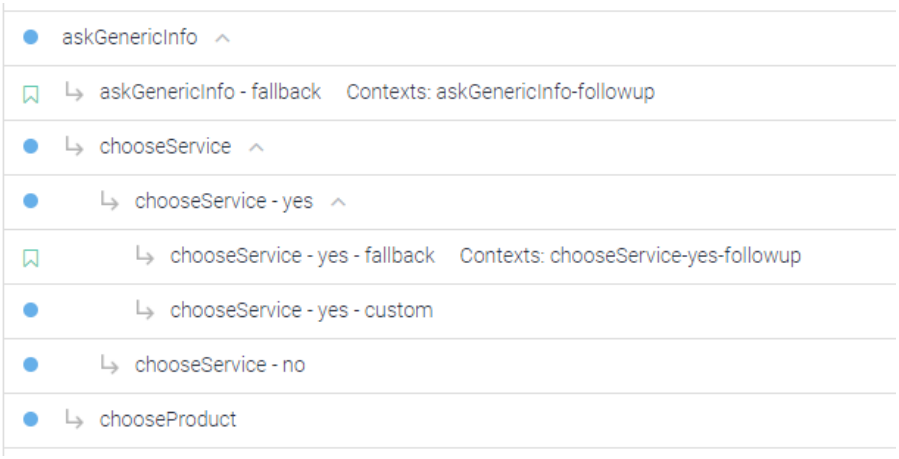


Ilustración 22. Intentos para obtener información genérica.

En el primero, *askGenericInfo*, se configuran expresiones que el usuario puede utilizar para pedir información acerca de la empresa. El usuario puede especificar si quiere información sólo de los productos, solo de los servicios, u obtener el catálogo completo. Este intento se conecta a los microservicios que, tras consultar la base de datos, generan una respuesta con la información apropiada a la consulta del usuario.

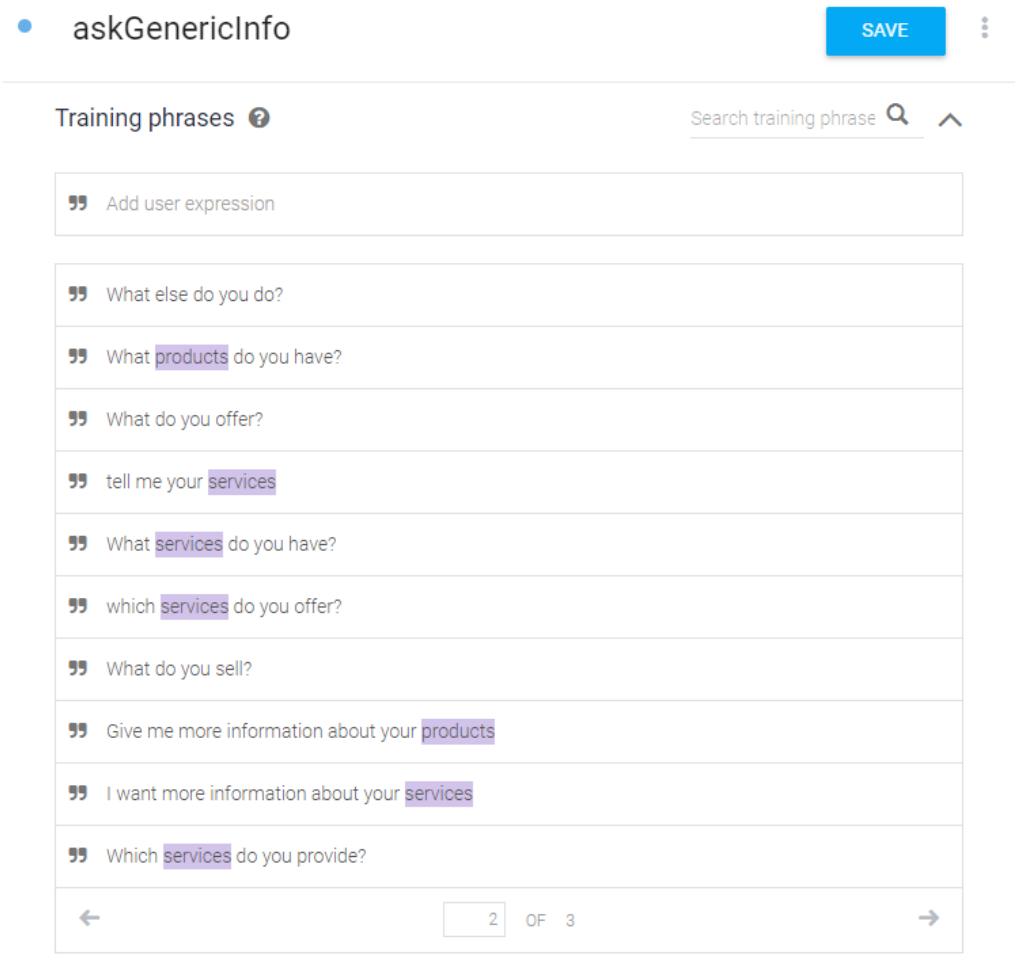


Ilustración 23. Consultas para obtener información.

Cuando el intento se lanza, se envía la petición HTTP a las Cloud Functions, y es gestionado por la función principal. Ésta detecta el intento que la envía, y ejecuta dicha función. En ella se accede al parámetro *type*, cuya entidad será *Type*, definida previamente. Esto significa que el valor solo puede ser "*Products*", "*Services"*, o ninguno de ellos, dependiendo de si el usuario ha introducido en la consulta cualquiera de los sinónimos definidos para alguna de las entradas de la entidad. Se comprueba el valor recibido y, en función de él, se genera el mensaje al usuario incluyendo la información relevante a la consulta. Para crear los mensajes de respuesta se ha desarrollado otro servicio (*msgService*).

Como se puede observar en la ilustración 24, esta función hace uso de las *Promises*, las cuales representan una operación asíncrona que puede completarse o terminar con un error. Aunque en este caso no se realiza ninguna función asíncrona, es un requisito de la librería de Dialogflow que todas las funciones utilicen *Promises*.

```
5   function askGenericInfo(agent) {
6       return new Promise((resolve, reject) => {
7           const type = agent.parameters["type"];
8
9           switch (type.toLowerCase()) {
10              case "products":
11                  agent.add(msgService.getProductsMsg(PRODUCTS_DATASTORE));
12                  break;
13              case "services":
14                  agent.add(msgService.getServicesMsg(SERVICES_DATASTORE));
15                  break;
16              default:
17                  agent.add(msgService.getCatalogMsg(PRODUCTS_DATASTORE, SERVICES_DATASTORE));
18          }
19          resolve();
20      });
21  }
```

Ilustración 24. Código para obtener información genérica.

De este intento padre salen tres intentos de *follow-up*, que se lanzan después de que el agente haya enviado una respuesta: uno, para gestionar cuando el usuario quiere más detalles acerca de un servicio (*chooseService*); otro, para cuando se trata de un producto (*chooseProduct*); y un tercero, por si la respuesta del usuario no es sobre los productos o servicios del catálogo. En estos intentos el usuario simplemente introduce el nombre de alguno de los productos o servicios que ha citado el agente en respuesta a su primera consulta.

En el caso de que el usuario pregunte por un producto, el agente proporciona detalles sobre él, junto con el precio por licencia. Sin embargo, si el usuario consulta un servicio, el agente, además de describirlo le ofrece la posibilidad de solicitar una cita con el fin de negociar la contratación de dicho servicio. Por esta razón el intento *chooseService* tiene más intentos de seguimiento. En ellos se gestiona que el usuario acceda a concertar la cita o no hacerlo, y, en caso afirmativo, a la propuesta de una hora por parte del usuario.

Para que la información que recoge el intento *chooseService* llegue al intento en el que se especifica la hora de la cita, se hacen uso de los contextos. En la ilustración 25 se

pueden observar los parámetros del último intento: la fecha y la hora se recogen de la consulta introducida por el usuario (referenciados con el símbolo “$” junto con el nombre del parámetro), y el servicio seleccionado se obtiene de un contexto que se pasa al intento. Dado que el objeto del contexto es un JSON, el valor se obtiene accediendo a una propiedad del objeto. Para ello se utiliza la forma “#<contexto>.<propiedad>”.

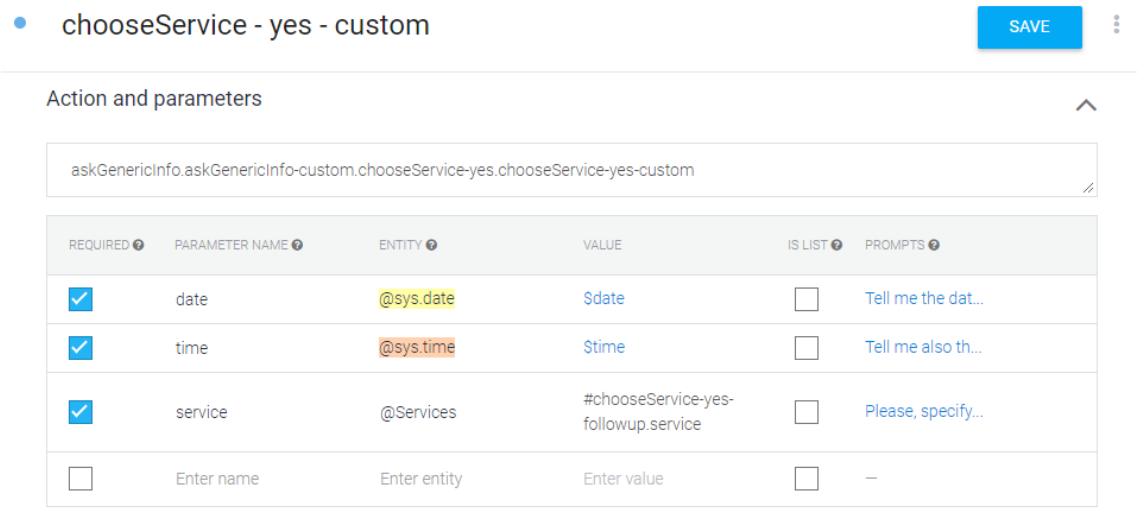


Ilustración 25. Parámetros del intento "chooseService-yes-custom".

En este intento se procesan estos parámetros: utilizando el día y hora capturados para crear un objeto de tipo fecha y, por otro lado, el nombre del servicio elegido se busca entre los servicios configurados (ya que también se contempla el caso en que el usuario especifique un asunto diferente). En caso de que haya seleccionado uno de los servicios existentes, se obtiene de la base de datos la información sobre el mismo para especificar la duración del evento a crear. Si ha optado por especificar otro asunto, la duración será la establecida por defecto (45 minutos). Por último, se crea un objeto de tipo evento, que sirve para lanzar otro intento automáticamente, y se envía como respuesta. Se especifica el nombre del evento y los parámetros que se envían, que serán los datos del evento a crear (fecha, duración y asunto).

```
function askServiceInfoConfirm(agent) {
    return new Promise((resolve, reject) => {
        const newDate = agent.parameters["date"];
        const newTime = agent.parameters["time"];
        const timeString = moment(newTime);

        const dateObject = moment(newDate);
        dateObject.set({
            hour: timeString.get('hour'),
            minute: timeString.get('minute')
        });

        let found = false,
            i = 0,
            service = agent.parameters["service"],
            duration;
        while (!found && i < SERVICES_DATASTORE.length) {
            if (SERVICES_DATASTORE[i].Name == service) {
                found = true;
                duration = SERVICES_DATASTORE[i].Duration;
            }
            i++;
        }

        const event = {
            name: "createAppointment",
            parameters: {
                date: dateObject,
                reason: agent.parameters["service"],
                duration: duration
            },
            languageCode: "en"
        };

        agent.setFollowupEvent(event);
        resolve();
    });
}
```

Ilustración 26. Código que lanza la creación de un evento.

Por otro lado, como se puede observar en la fase de diseño de este flujo, hay dos intentos padres que se encargan de capturar consultas que se centran directamente en un producto (*askProductInfo*) o en un servicio (*askServiceInfo*). Éstas, a su vez, lanzan los intentos *chooseProduct* o *chooseService* respectivamente. Algunos ejemplos de estas consultas son “What can you tell me about Metadrive?”, “Give me some information about your consultancy services” y “I need a custom development”.

Concertar una cita

Este flujo se inicia con una consulta del usuario en la cual especifica que quiere crear una cita, lanzando el intento *appointDate*. Para ello se introducen múltiples expresiones que puede utilizar el usuario.

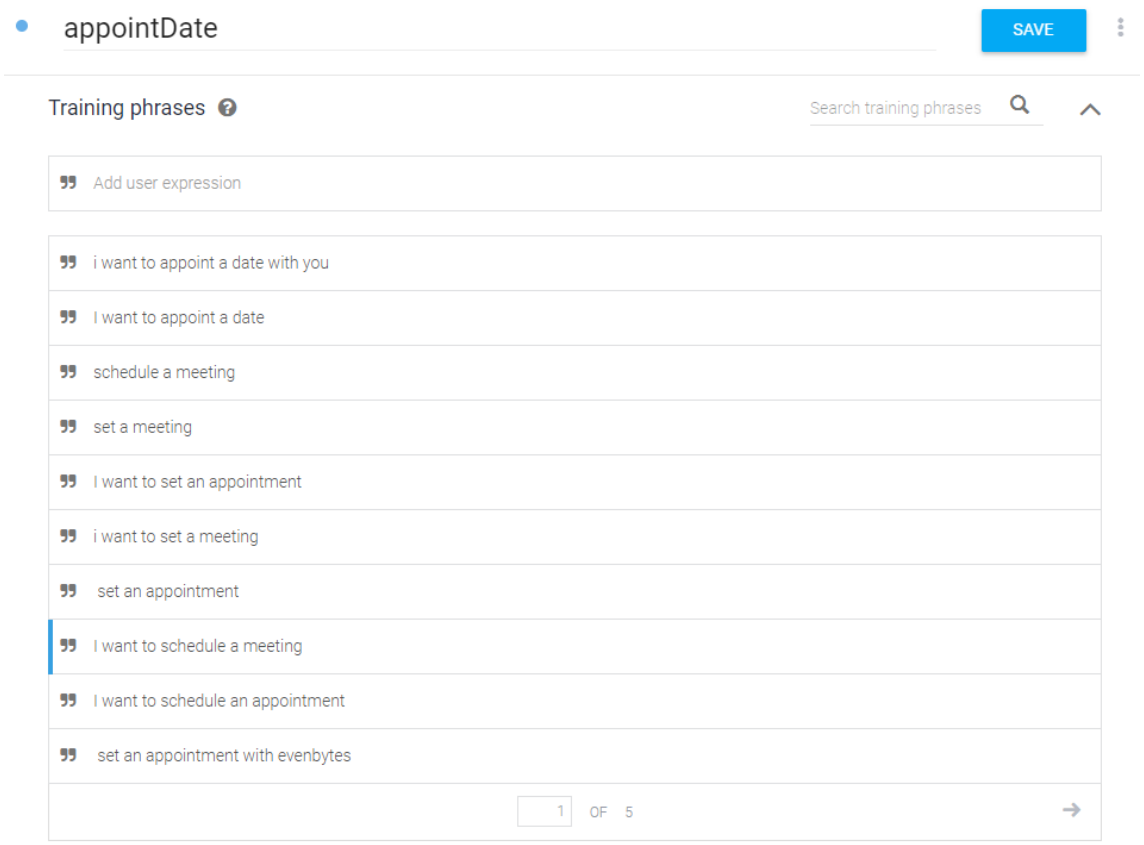


Ilustración 27. Expresiones para concertar una cita.

Después, en los microservicios, se lanza otro intento (*setAppointment*) que obtiene la información necesaria (fecha y hora, email del usuario y asunto), y crea el evento en el calendario. Este intento también se utiliza cuando el usuario selecciona un servicio y acepta crear una cita en el flujo de conversación “Obtener información genérica”. Para ello se configuran en el intento los eventos que pueden lanzarlo, especificando los nombres de los mismos.

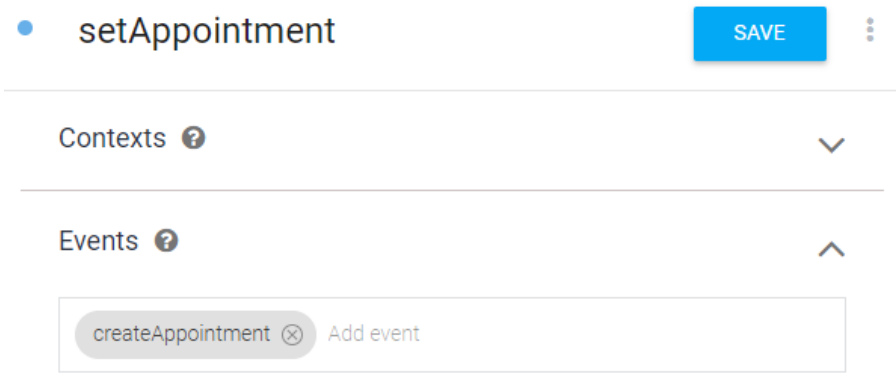


Ilustración 28. Eventos que lanzan el intento "setAppointment".

Además, se definen cuatro parámetros que toman su valor del evento que lo lanza por código. Tres de ellos son obligatorios, para los cuales se definen una serie de frases

(columna *Prompts*) para pedir la información al usuario en caso de que no haya podido ser obtenida del evento.

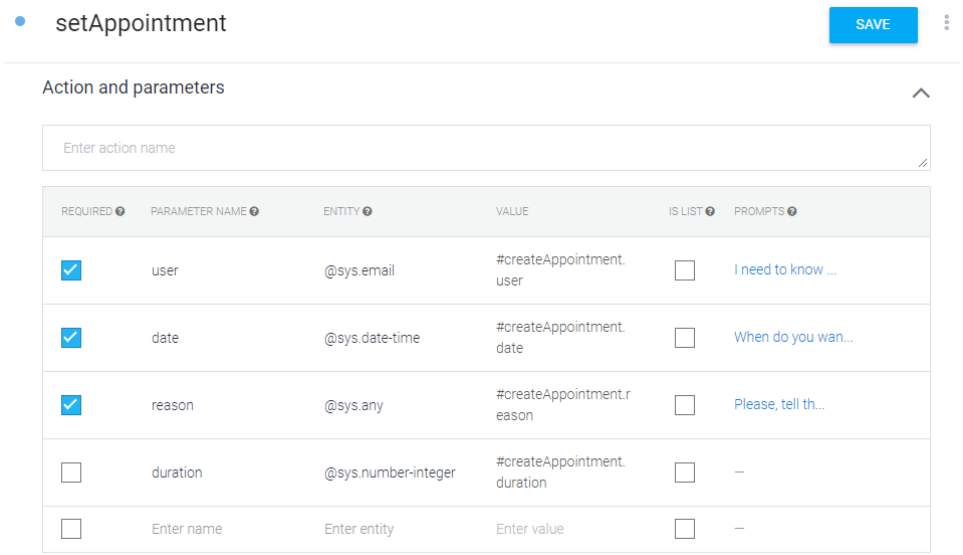


Ilustración 29. Configuración de los parámetros de una cita.

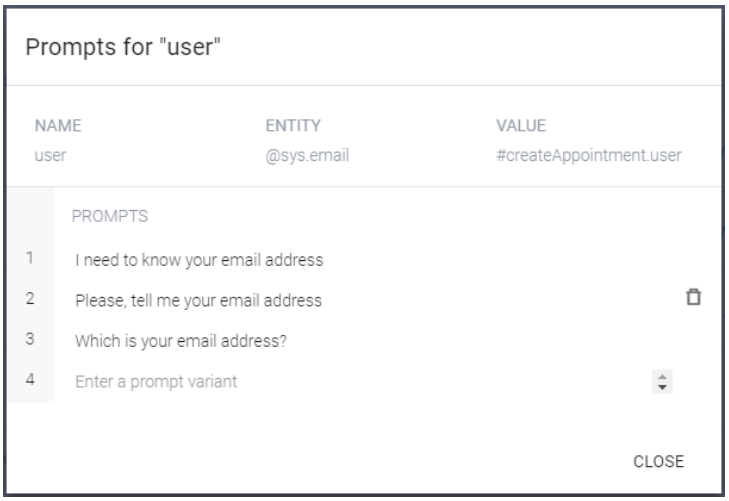
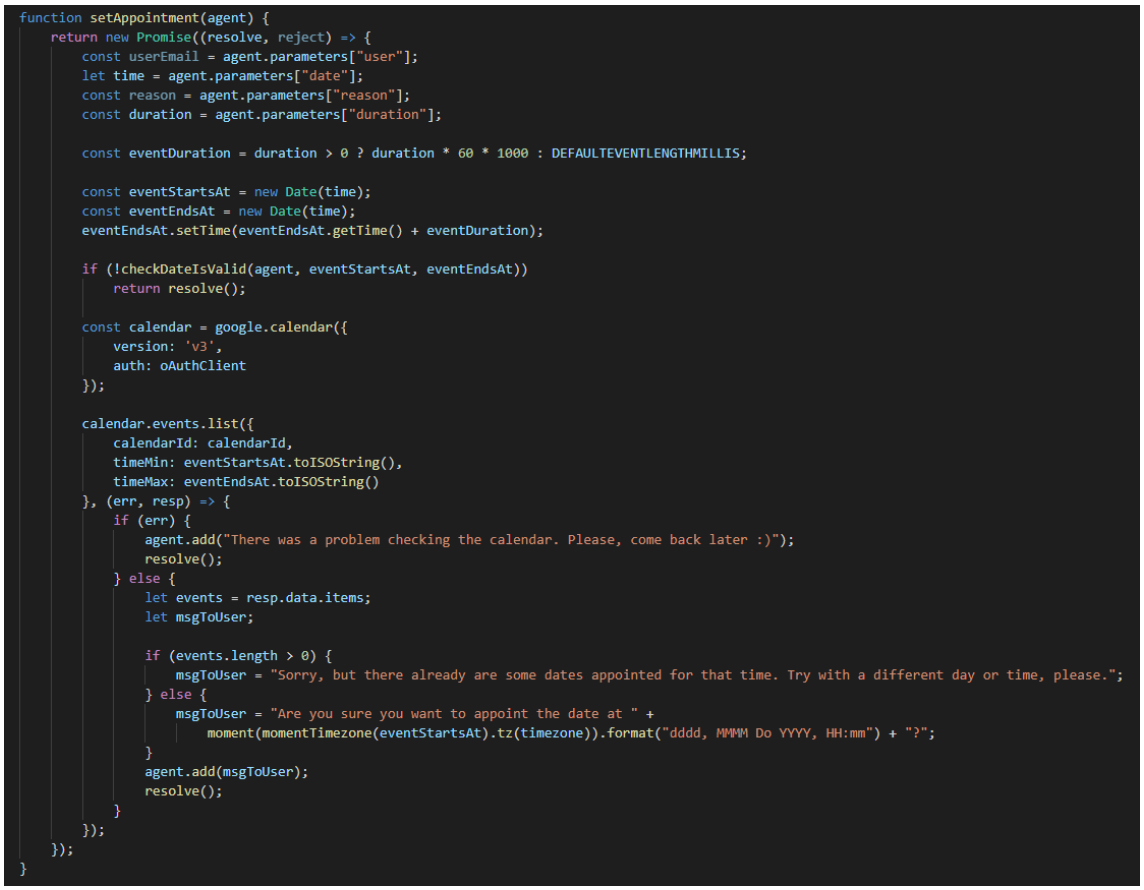


Ilustración 30. Ejemplo de configuración de los propmts.

En este intento se comprueba que la fecha que ha especificado el usuario es válida y que hay espacio en ella para crear el evento. Para obtener esta información se realiza una consulta al calendario del negocio y se comprueba que no hay eventos creados en la franja de tiempo en la que se va a crear dicho evento. Esta consulta se realiza a través de las Google APIs con el objeto de autentificación generado previamente.



```
function setAppointment(agent) {
    return new Promise((resolve, reject) => {
        const userEmail = agent.parameters["user"];
        let time = agent.parameters["date"];
        const reason = agent.parameters["reason"];
        const duration = agent.parameters["duration"];

        const eventDuration = duration > 0 ? duration * 60 * 1000 : DEFAULTEVENTLENGTHMILLIS;

        const eventStartsAt = new Date(time);
        const eventEndsAt = new Date(time);
        eventEndsAt.setTime(eventEndsAt.getTime() + eventDuration);

        if (!checkDateIsValid(agent, eventStartsAt, eventEndsAt))
            return resolve();

        const calendar = google.calendar({
            version: 'v3',
            auth: oAuthClient
        });

        calendar.events.list({
            calendarId: calendarId,
            timeMin: eventStartsAt.toISOString(),
            timeMax: eventEndsAt.toISOString()
        }, (err, resp) => {
            if (err) {
                agent.add("There was a problem checking the calendar. Please, come back later :)");
                resolve();
            } else {
                let events = resp.data.items;
                let msgToUser;

                if (events.length > 0) {
                    msgToUser = "Sorry, but there already are some dates appointed for that time. Try with a different day or time, please.";
                } else {
                    msgToUser = "Are you sure you want to appoint the date at " +
                        moment(momentTimezone(eventStartsAt).tz(timezone)).format("dddd, MMMM Do YYYY, HH:mm") + "?";
                }
                agent.add(msgToUser);
                resolve();
            }
        });
    });
}
```

Ilustración 31. Comprobación de validez de fecha para crear un evento.

Después, tras mostrar al usuario la información del evento y obtener la confirmación, se pasa a crearlo.

```
function setAppointmentConfirm(agent) {
    return new Promise((resolve, reject) => {
        const userEmail = agent.parameters["user"];
        let time = agent.parameters["date"];
        const reason = agent.parameters["reason"];
        let duration = agent.parameters["duration"];

        const eventStartsAt = new Date(time);
        const eventEndsAt = new Date(time);
        eventEndsAt.setTime(eventEndsAt.getTime() + duration);

        const eventToInsert = {
            summary: reason,
            start: {
                dateTime: eventStartsAt.toISOString(),
                timeZone: DEFAULTTIMEZONE
            },
            end: {
                dateTime: eventEndsAt.toISOString(),
                timeZone: DEFAULTTIMEZONE
            },
            attendees: [{
                email: userCalendar,
                responseStatus: "accepted"
            }, {
                email: userEmail,
                responseStatus: "accepted"
            }]
        };

        const calendar = google.calendar({
            version: 'v3',
            auth: oAuthClient
        });

        calendar.events.insert({
            calendarId: calendarId,
            resource: eventToInsert,
            sendNotifications: true
        }, (err, event) => {
            if (err) {
                agent.add('The event could not be created. Please, come back later :)');
            } else {
                agent.add('The date was appointed succesfully. Thanks for using EvenBytes Chatbot :)');
            }
            resolve();
        });
    });
}
```

Ilustración 32. Creación de un evento utilizando Google Calendar API.

Editar una cita existente

Una vez que el usuario ha planificado una cita con el negocio, tiene la posibilidad de moverla a otra fecha si así lo desea. Para ello, el agente le solicita sus datos de contacto, y le muestra los eventos en los cuales es uno de los participantes.

Los eventos que el usuario ha planificado previamente se obtienen realizando una consulta a la API de Google Calendar, pasando como parámetro de búsqueda la dirección de correo electrónico del usuario. La lista de eventos que se obtiene se añade como parámetro del contexto del intento, para que al seleccionar uno de ellos el usuario, se obtenga la información de esta lista, sin necesidad de realizar otra llamada a la API.

```
function editAppointment(agent) {
    return new Promise((resolve, reject) => {
        const user = agent.parameters["user"];

        const calendar = google.calendar({
            version: 'v3',
            auth: oAuthClient
        });
        const now = new Date();

        calendar.events.list({
            calendarId: calendarId,
            timeMin: now.toISOString(),
            q: user,
            fields: "items(id, summary, start, end)"
        }, (err, resp) => {
            if (err) {
                agent.add('Events could not be retrieved. Try again later.');
                resolve();
            } else {
                const eventIds = resp.data.items.map(function (e) {
                    return e.id;
                });

                agent.setContext({
                    "name": "editAppointment-followup",
                    "lifespan": 5,
                    "parameters": {
                        "events": JSON.stringify(eventIds)
                    }
                });

                agent.add(msgService.generateEventsListMsg(resp.data.items, timezone));
                resolve();
            }
        });
    });
}
```

Ilustración 33. Obtención de los eventos del usuario.

Una vez que el usuario ha seleccionado el evento que desea editar, y ha introducido una fecha válida a la que moverle, se realiza la actualización de éste.

```
function editAppointmentFlowConfirm(agent) {
    return new Promise((resolve, reject) => {
        const eventId = agent.parameters["eventId"];
        const newStart = new Date(agent.parameters["newStart"]);
        const newEnd = new Date(agent.parameters["newEnd"]);

        const calendar = google.calendar({
            version: 'v3',
            auth: oAuthClient
        });

        const newEventBody = {
            start: {
                dateTime: newStart.toISOString(),
                timeZone: DEFAULTTIMEZONE
            },
            end: {
                dateTime: newEnd.toISOString(),
                timeZone: DEFAULTTIMEZONE
            }
        };

        calendar.events.patch({
            calendarId: calendarId,
            eventId: eventId,
            resource: newEventBody
        }, (err, event) => {
            if (err) {
                agent.add('The event could not be updated. Please, try again later.');
            } else {
                agent.add('The date was updated succesfully. Thanks for using EvenBytes Chatbot!');
            }
            resolve();
        });
    });
}
```

Ilustración 34. Actualización de un evento.

Consultar disponibilidad

Con el objetivo de gestionar los posibles casos de esta consulta, se crea un intento padre con tres *follow-up*. Éstos gestionan si el usuario solicita más propuestas, selecciona una de ellas, o si la consulta no coincide con ninguna de las establecidas en estos dos intentos. Además del intento para solicitar más propuestas (*queryFreeSpots - more*) también tendrá otros intentos de seguimiento, de forma que el usuario pueda solicitar más resultados o seleccionar uno de ellos hasta que ya no haya más propuestas.

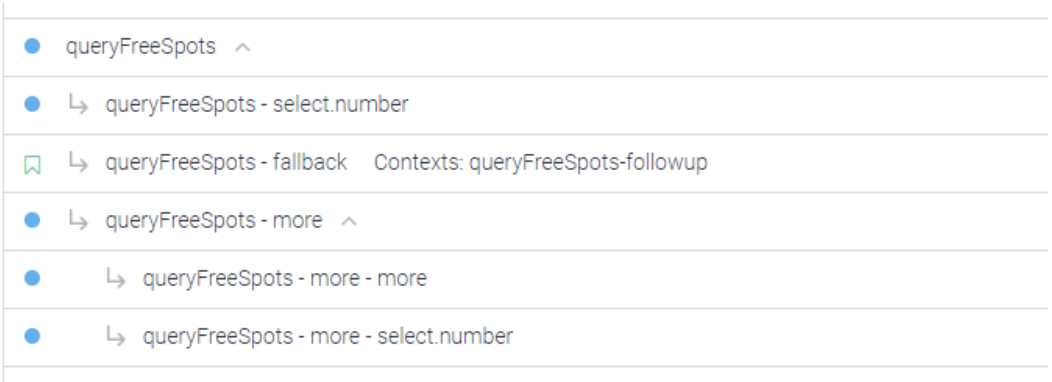


Ilustración 35. Intentos para consultar la disponibilidad.

En caso de que el usuario seleccione uno de ellos, se lanzará el intento para crear una cita, enviando como parámetro del evento la fecha y hora de la propuesta seleccionada. Después, el agente le pedirá al usuario el resto de datos necesarios para crear la reunión

(dirección de correo electrónico y asunto de la reunión), de acuerdo a lo especificado en el flujo de conversación para crear una cita.

Algunas de las consultas que el usuario puede realizar para obtener información sobre la disponibilidad del negocio están mostradas en la ilustración 36.

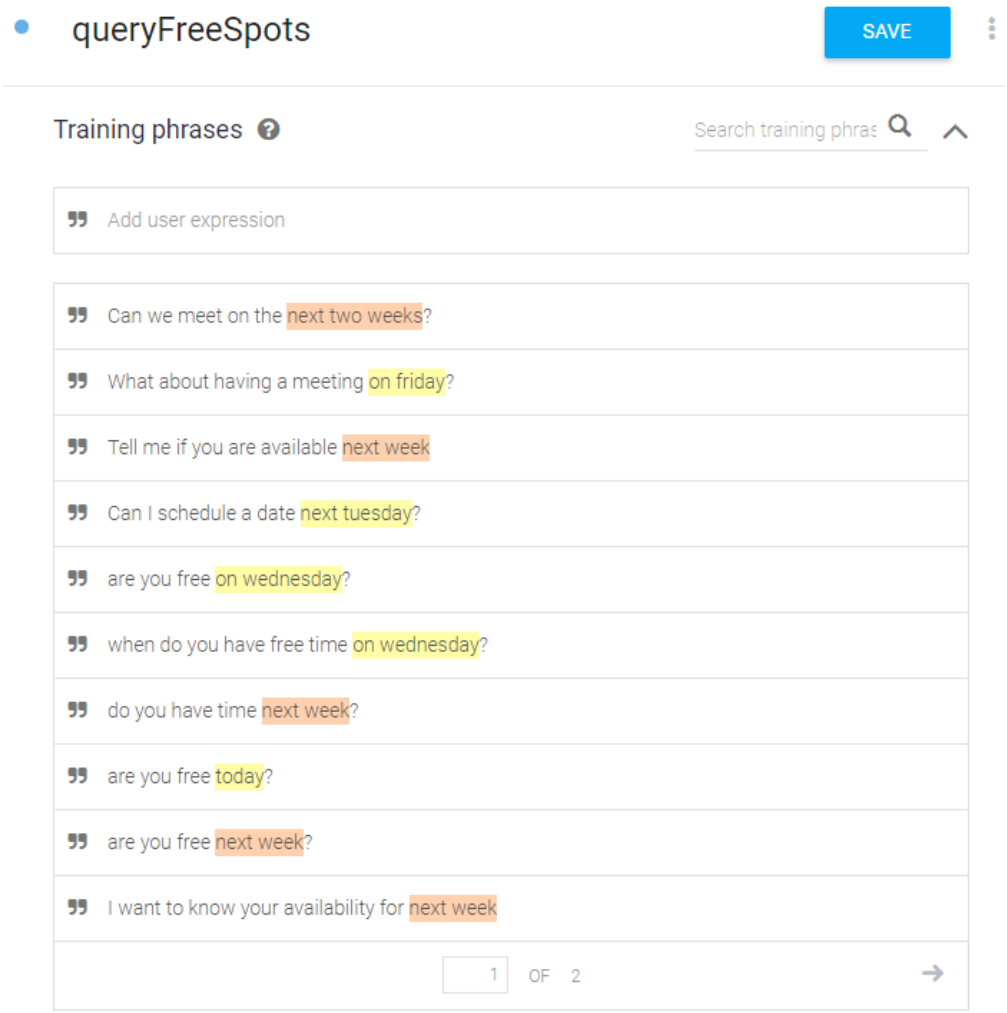


Ilustración 36. Ejemplos de consultas de disponibilidad.

Agentes Externos

Dialogflow ofrece la posibilidad de importar cualquier agente ya construido, ya sea un agente construido por Google o por cualquier desarrollador que lo utilice. Algunos ejemplos son agentes para consultar el clima, gestionar alarmas o utilizar un convertidor de moneda.

Entre esos agentes se puede encontrar un agente que sirve para gestionar consultas habituales de los usuarios del tipo “How old are you?”, “Where do you work?” o “Talk to me”. Este agente se llama *Small Talk*, y permite configurar varias respuestas a un catálogo de preguntas como las anteriormente mencionadas. Se clasifican en varios

grupos: preguntas sobre el agente, de cortesía, expresiones sobre emociones, de saludo y despedida, sobre el propio usuario, de confirmación y otro tipo de expresiones.

Dado que son consultas que los usuarios realizan frecuentemente, y que las respuestas están predefinidas, se ha importado este agente para gestionar estas conversaciones y se ha configurado a través de un formulario en la herramienta.

3.6.2. Aplicación de configuración

Se ha desarrollado una aplicación para configurar la información que utiliza el *chatbot* de acuerdo a los *wireframes* diseñados. Ésta se ha implementado con HTML5, CSS3, y AngularJS, además de componentes de AngularJS Material [4].

Como se puede observar en las imágenes 37, 38 y 40, la aplicación tiene una parte común, que es la cabecera. En ella se muestra el nombre del *chatbot* y una serie de pestañas que permiten navegar entre diferentes componentes. Para ello se han utilizado las directivas *md-toolbar* y *md-nav-bar* de AngularJS Material.

En la gestión de productos y de servicios se sigue el mismo diseño: uso de *cards* para mostrar el catálogo y modificar o eliminar los elementos. Además, se proporciona un botón flotante en la parte inferior derecha de la pantalla que permite la creación de nuevos elementos.

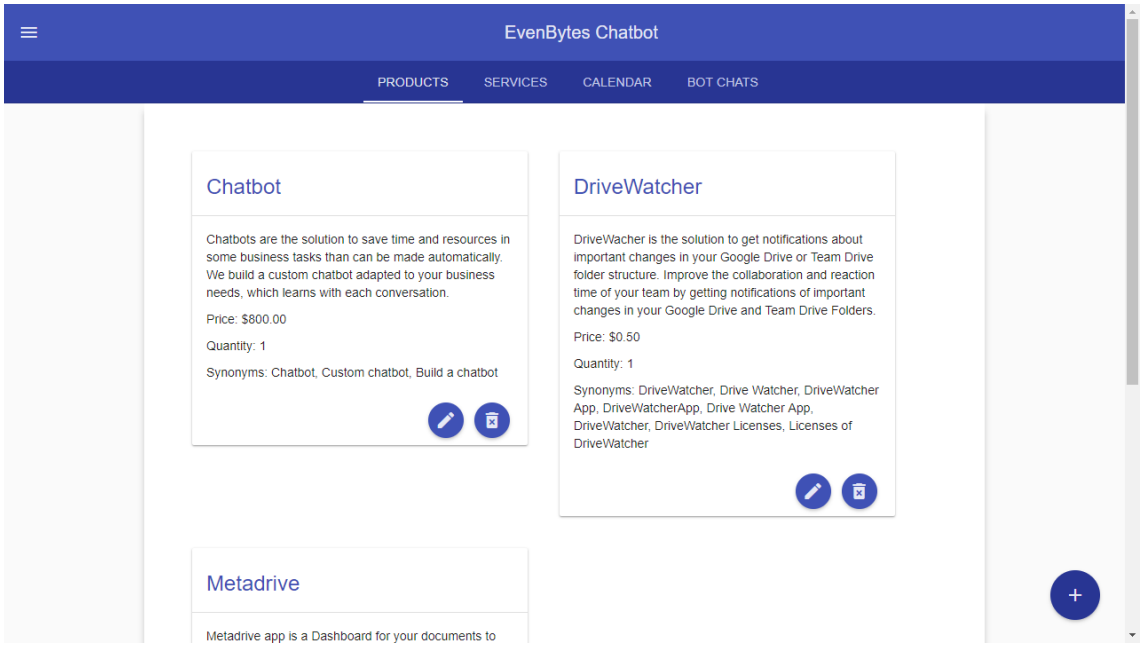


Ilustración 37. Implementación: página de productos.

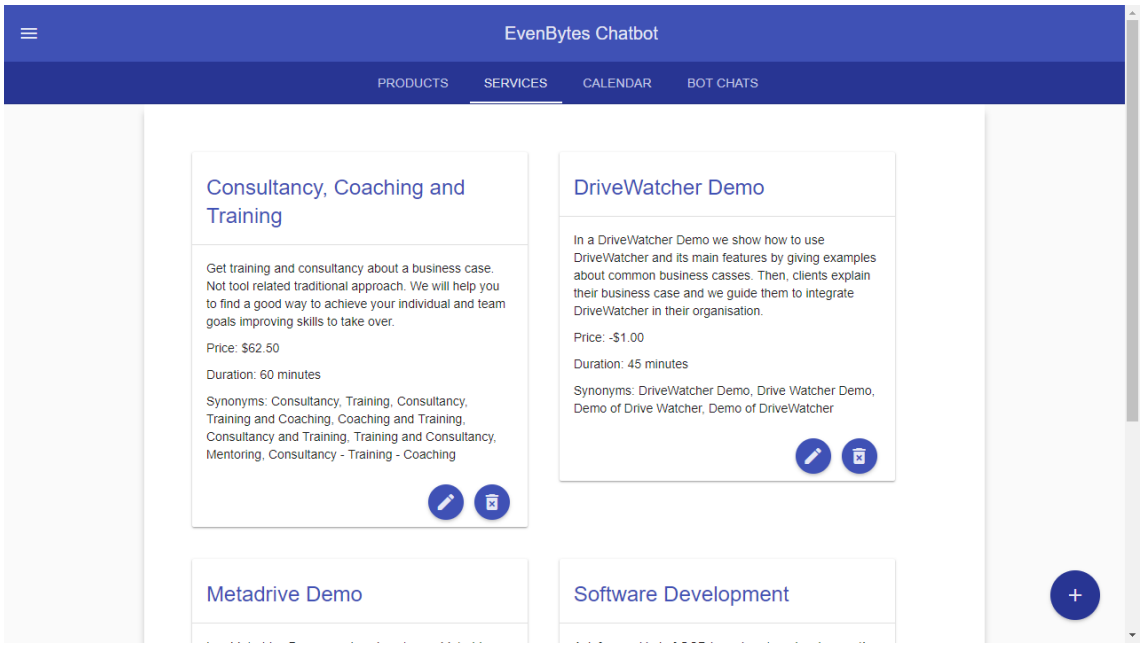


Ilustración 38. Implementación: página de servicios.

Cuando el usuario de la aplicación selecciona crear un nuevo producto o servicio, ésta muestra un formulario en una ventana emergente. Dicho formulario está contenido en un elemento *dialog*, que sirve para gestionar la ventana emergente, y cuenta con todos los campos que caracterizan a los productos o servicios. En ambos casos son todos obligatorios excepto el campo de sinónimos.

Si el usuario selecciona la opción de editar un elemento existente, este formulario tendrá los campos completados con los datos del elemento correspondiente.

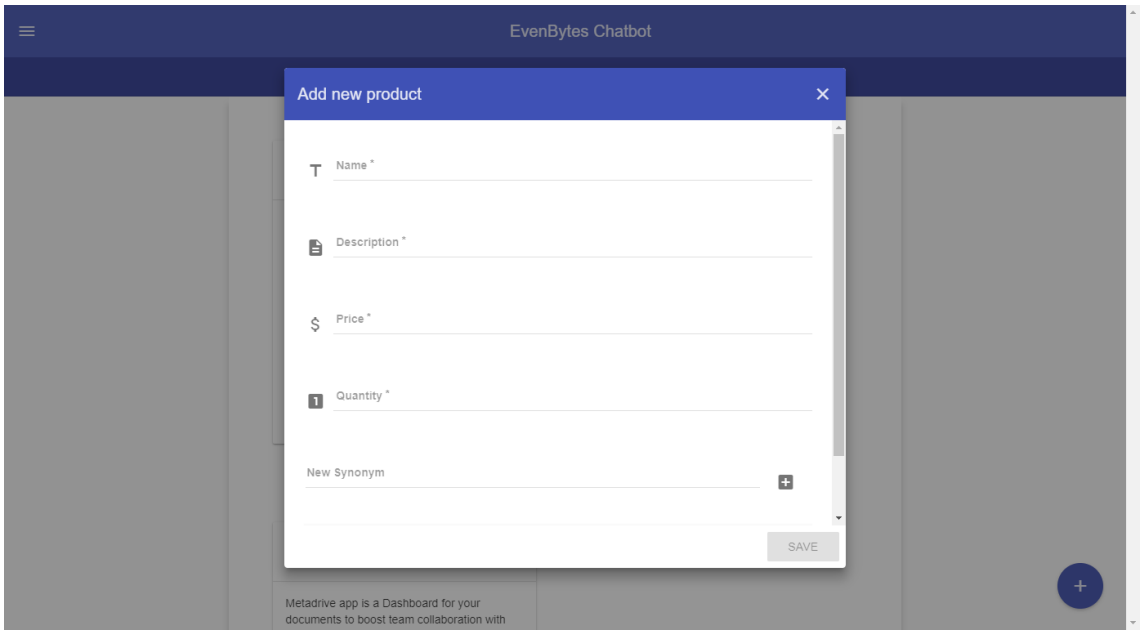


Ilustración 39. Implementación: creación de un nuevo producto.

En cambio, como se puede observar en la ilustración 40, la interfaz para gestionar los ajustes del calendario difiere de las anteriores. En primer lugar, se muestra una sección que permite seleccionar el calendario (que utilizará el asistente para la gestión de eventos) y la zona horaria del negocio.

En segundo lugar, la configuración sobre el horario de disponibilidad del negocio. El usuario puede crear hasta siete configuraciones diferentes (una por día de la semana). En ellas establece el horario de apertura y de cierre en los días seleccionados de la lista, pudiendo elegir entre horario continuo u horario partido. En caso de que el usuario elija un día ya seleccionado en otra configuración, éste se elimina de la configuración previa y se añade a la actual, de forma que cada día de la semana solo tenga un horario definido.

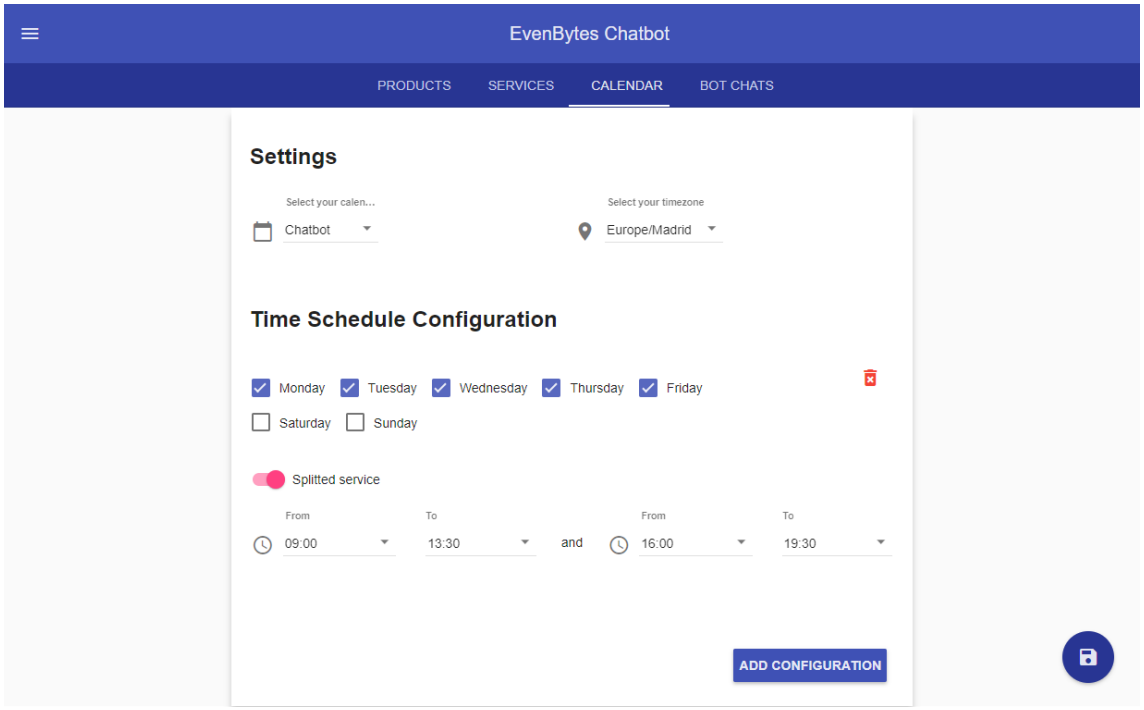


Ilustración 40. Implementación: página de calendario.

Al acceder a la aplicación por primera vez, el usuario (responsable del negocio) concede permisos para gestionar información de su perfil (dirección de correo electrónico, ID de usuario, nombre…), sus calendarios y servicios de Google Cloud Platform (la API de Dialogflow), todo ello de manera *offline*. Es decir, la aplicación puede realizar operaciones sobre estos servicios simulando ser el usuario.

Para que las llamadas a la API de Dialogflow se puedan completar, será necesario que el usuario tenga permisos en este servicio. Esta característica se gestionará desde la sección *Cloud IAM* de Google Cloud Platform, la cual permite configurar los roles de los usuarios en un proyecto. En este caso se utilizará el rol "Dialogflow API admin", que es el único rol que permite modificar los intentos y entidades del agente.

3.6.3. Instalación

Entre las integraciones de Dialogflow, se encuentra su instalación en un sitio web como un *iframe*, para lo cual proporciona el correspondiente código HTML necesario.

```
<iframe
    allow="microphone;"
    width="350"
    height="430"
    src="https://console.dialogflow.com/api-client/demo/embedded/
">
</iframe>
```

Ilustración 41. Código HTML para integrarlo en la web corporativa.

La integración en Google Assistant no requerirá desarrollo. Simplemente, una vez que se haya publicado el proyecto en Actions on Google (la plataforma de construcción de asistentes en Google Assistant) el usuario podrá invocar el *chatbot* enviando el mensaje "Talk to EvenBytes Chatbot", y así comenzará a hablar con el asistente. De esta forma, cualquier usuario que disponga de un dispositivo Android podrá utilizar dicho asistente.

4. Conclusiones y trabajo futuro

4.1. Conclusiones

El objetivo principal del proyecto era desarrollar un *chatbot* o asistente virtual que se utilizara para gestionar reuniones con clientes y proporcionar información básica sobre los servicios de la empresa EvenBytes S.L. Además, este asistente tenía que ser fácilmente configurable por medio de una aplicación web que se conectase a él a través de su API. Para ello, se han utilizado diversos servicios de Google Cloud Platform: Dialogflow y Google Cloud Functions para el asistente, y Google App Engine para desplegar la aplicación de configuración del mismo, junto con Google Cloud Datastore para almacenar la información necesaria. De esta forma, el mánager del *chatbot* solo necesita iniciar sesión con su cuenta de Google en la aplicación de configuración para actualizar la información que desea que utilice el *chatbot* en sus conversaciones con los usuarios.

El *chatbot* se ha integrado en la página web de la empresa a través de un *iframe*. Con el fin de facilitar la interacción con los usuarios, también se podrá utilizar a través de Google Assistant, una vez que haya superado el proceso de validación de Google.

En el plano personal, el desarrollo de este proyecto me ha permitido conocer tecnologías novedosas como son Dialogflow y Cloud Functions, utilizarlas en un caso de uso real, e investigar cómo y para qué pueden ser utilizadas en futuros proyectos. Además, las Cloud Functions usan el entorno Node.js 6, con el cual apenas había tenido la oportunidad de trabajar hasta ahora, junto con NPM (Node Package Manager) para la gestión de librerías. Por otro lado, también he tenido la oportunidad de ampliar mis conocimientos y experimentar en servicios ya conocidos como Google App Engine, Cloud Datastore y Stackdriver, así como las APIs de Google; en tecnologías como HTML, CSS, AngularJS y Java; y en metodologías ágiles de gestionar proyectos como Scrum.

4.2. Trabajo futuro

A continuación, se exponen funcionalidades que se podrían incorporar al *chatbot*.

Soporte multi idioma

El asistente ha sido diseñado para entablar la conversación en inglés. A pesar de que la mayoría de los clientes son extranjeros, y la comunicación se produce en inglés, una tarea futurible sería configurar el *chatbot* para ser capaz de comunicarse en español, y así atender a cualquier cliente en este idioma.

Integración en distintas plataformas

Los requisitos del proyecto exigían que el *chatbot* se incorporara a la página web de la empresa, utilizando la integración en la web a través de un *iframe*, e integrarlo en el

Google Assistant, pudiéndose utilizar desde cualquier dispositivo Android y desde Google Home.

Dialogflow permite desplegarlo en múltiples plataformas, entre las que destacan Twitter, Facebook Messenger, Telegram, Skype, Twilio y Slack. Habría que analizar cuáles de ellas son de interés para la empresa y realizar la correspondiente integración.

Añadir nuevos casos de negocio

Sería interesante, también, que el asistente sea entrenado para algún caso de negocio nuevo, como puede ser el de dar soporte de las aplicaciones de la empresa, consultar la facturación de un cliente o suscribir a los usuarios a las campañas de correo electrónico de la misma.

Copia de seguridad de las conversaciones

Esta funcionalidad conlleva almacenar en una base de datos tanto las consultas de los usuarios como las respuestas del asistente, de forma que el cliente pueda acceder a todas las conversaciones a través de la aplicación de configuración.

Tras comparar las bases de datos disponibles en Google Cloud Platform, se ha elegido la base de datos Cloud Firestore for Firebase para guardar esta información por la eficiencia de su modelo de datos, y porque uno de los casos de uso más frecuente de esta base de datos es el almacenamiento de chats.

Su modelo de datos se basa en acumular documentos, que contienen un conjunto de elementos clave-valor. Estos documentos están organizados en colecciones [9].

Para almacenar las conversaciones del asistente se crearía una colección por ID de sesión en Dialogflow, y un documento popara cada mensaje. En él se guardaría el origen (usuario o asistente), la fecha y hora de envío, y el texto del mensaje.

Informe de consultas fuera de catálogo

En los *chatbots* de propósito específico como éste, hay que gestionar de alguna forma el comportamiento cuando el usuario realiza una consulta que no está definida. El asistente se ha diseñado para que envíe un mensaje al usuario con las posibles consultas que le puede hacer. Sin embargo, puede haber consultas de productos y/o servicios que no estén en el catálogo de la empresa.

Esta funcionalidad almacenaría todas estas consultas, y se generaría un reporte que se enviaría al cliente. También, sería accesible a través de la aplicación de configuración, para que pudiese tomar las decisiones que considere oportunas.

Definición de eventos especiales en el calendario

El Calendario de Google permite crear una serie de eventos especiales, como por ejemplo “Fuera de la Oficina”. Esta funcionalidad permitiría crear nuevos eventos especiales para evitar que se asignen citas los días que contengan estos eventos, y personalizar el mensaje de respuesta que el asistente utilizaría.

5. Bibliografía

[1] ABU SHAWAR, B.; ATWELL, E. 2007. *Chatbots: Are they Really Useful?* [Consulta: 2 octubre 2018]. LDV Forum, vol. 22. Disponible en: https://pdfs.semanticscholar.org/8d82/84bfba7ebcb4e2575d864ec7c16ea6a168f0.pdf

[2] ABU SHAWAR, B.; ATWELL, E. 2015. *ALICE Chatbot: Trials and Outputs*. [Consulta: 4 octubre 2018]. Computación y Sistemas, vol. 19, no. 4, pp.625-632. DOI 10.13053/CyS-19-4-2326.

[3] AMAZON ALEXA. 2018. [Consulta: 4 octubre 2018]. Disponible en: https://developer.amazon.com/alexa

[4] ANGULARJS MATERIAL. 2018. [Consulta: 2 octubre 2018]. Disponible en: https://material.angularjs.org

[5] ANGULARJS. 2018. *AngularJS -- Superheroic JavaScript MVW Framework*. [Consulta: 2 octubre 2018]. Disponible en: https://angularjs.org/

[6] APPLE INC. 2018. *Siri - Apple*. [Consulta: 4 octubre 2018]. Disponible en: https://www.apple.com/siri/

[7] DIALOGFLOW. [Consulta: 19 septiembre 2018]. Disponible en: https://dialogflow.com/.

[8] DIALOGFLOW. *Machine Learning*. [Consulta: 24 septiembre 2018]. Disponible en: https://dialogflow.com/docs/agents/machine-learning.

[9] FIREBASE. 2018. *Cloud Firestore Data model*. [Consulta: 24 septiembre 2018]. Disponible en: https://firebase.google.com/docs/firestore/data-model.

[10] GARTNER. 2018. *Gartner Says 25 Percent of Customer Service Operations Will Use Virtual Customer Assistants by 2020*. [Consulta: 4 octubre 2018]. Disponible en: https://www.gartner.com/en/newsroom/press-releases/2018-02-19-gartner-says-25-percent-of-customer-service-operations-will-use-virtual-customer-assistants-by-2020

[11] GOOGLE ASSISTANT. [Consulta: 4 octubre 2018]. Disponible en: https://assistant.google.com/

[12] GOOGLE CLOUD. 2018. *Choosing an App Engine Environment | App Engine Documentation*. [Consulta: 19 septiembre 2018]. Disponible en: https://cloud.google.com/appengine/docs/the-appengine-environments.

[13] GOOGLE CLOUD. 2018. *Google Cloud Datastore Overview | Cloud Datastore Documentation*. [Consulta: 6 septiembre 2018]. Disponible en: https://cloud.google.com/datastore/docs/concepts/overview.

[14] GOOGLE CLOUD. *App Engine - Build Scalable Web & Mobile Backends in Any Language | App Engine*. [Consulta: 24 agosto 2018]. Disponible en: https://cloud.google.com/appengine/.

[15] GOOGLE CLOUD. *Cloud Functions - Event-driven Serverless Computing | Cloud Functions*. [Consulta: 13 septiembre 2018]. Disponible en: https://cloud.google.com/functions/.

[16] GOOGLE CLOUD. *Cloud Functions - Intelligent Applications*. [Consulta: 19 septiembre 2018]. Disponible en: https://cloud.google.com/functions/use-cases/intelligent-applications.

[17] GOOGLE CLOUD. *Stackdriver - Hybrid Monitoring | Stackdriver*. [Consulta: 19 septiembre 2018]. Disponible en: https://cloud.google.com/stackdriver/.

[18] GOOGLE IDENTITY PLATFORM. 2018. *Using OAuth 2.0 to Access Google APIs*. [Consulta: 8 agosto 2018]. Disponible en: https://developers.google.com/identity/protocols/OAuth2.

[19] GRAND VIEW RESEARCH. 2017. *Chatbot Market Size To Reach $1.25 Billion by 2025 | CAGR: 24.3%*. [Consulta: 4 octubre 2018]. Disponible en: https://www.grandviewresearch.com/press-release/global-chatbot-market

[20] MEDHI THIES, I.; MENON, N.; MAGAPU, S. SUBRAMONY, M.; O'NEILL, J. 2017. *How do you want your chatbot? An exploratory Wizard-of-Oz study with young, urban Indians*. [Consulta: 2 octubre 2018]. *Lecture notes in Computer Science*, vol. 10513. Disponible en: https://pdfs.semanticscholar.org/e24b/275073915d008038294f5ee98a625259ce7c.pdf

[21] MOTHERBOARD. 2016. *A History of SmarterChild*. [Consulta: 4 octubre 2018]. Disponible en: https://motherboard.vice.com/en_us/article/jpgpey/a-history-of-smarterchild

[22] NODE.JS. [Consulta: 2 octubre 2018]. Disponible en: https://nodejs.org

[23] RAM, A.; PRASAD, R.; KHATRI, C.; VENKATESH, A.; GABRIEL, R.; LIU, Q.; NUNN, J.; HEDAYATNIA, B.; CHENG, M.; NAGAR, A.; KING, E. 2018. *Conversational AI: The Science Behind the Alexa Prize*. [Consulta: 3 octubre 2018]. *arXiv preprint arXiv:1801.03604*.

[24] *RFC 6749 - The OAuth 2.0 Authorization Framework*. [Consulta: 19 septiembre 2018]. Disponible en: https://tools.ietf.org/html/rfc6749.

[25] SESHADRI, S.; GREEN, B. 2014. *AngularJS: Up and Running*. Estados Unidos: O'Reilly Media, Inc. ISBN 9781491901946.

[26] THE LOEBNER PRIZE. [Consulta: 4 octubre 2018]. Disponible en: https://www.ocf.berkeley.edu/~arihuang/academic/research/loebner.html

[27] TURING, A. M. 1950. *I.—COMPUTING MACHINERY AND INTELLIGENCE*. Manchester: *Victoria University*. [Consulta: 3 octubre 2018]. Mind, Vol. LIX, Issue 236, pp. 433–460. Disponible en: https://doi.org/10.1093/mind/LIX.236.433

[28] WEIZENBAUM, J. 1966. *ELIZA - a computer program for the study of natural language communication between man and machine*. Cambridge, Massachusetts: Massachusetts Institute of Technology (MIT) [Consulta: 3 octubre 2018]. Commun. ACM, 9, pp. 36-45. Disponible en: https://web.stanford.edu/class/linguist238/p36-weizenabaum.pdf