

Trabajo Fin de Máster Data Science y Big Data

Asistente Conversacional para Frigorífico Inteligente

Autor: Inmaculada Perea Fernández

Tutor: Fernando Sancho Caparrini

**Centro de Formación permanente
Universidad de Sevilla**

Sevilla, 2017

Trabajo Fin de Máster
Data Science y Big Data

Asistente Conversacional para Frigorífico Inteligente

Autor:

Inmaculada Perea Fernandez

Tutor:

Fernando Sancho Caparrini

Centro de Formación Permanente
Universidad de Sevilla
Sevilla, 2017

RESUMEN

En este trabajo se realiza una introducción a la construcción de chatbots. Se describirá la metodología seguida para construir un prototipo capaz de proporcionar una interfaz de lenguaje natural para un frigorífico inteligente. Se realizará un estudio del estado del arte de las principales tecnologías de procesado de lenguaje natural disponibles en la actualidad, y en especial de la tecnología IBM Watson, por la que se ha optado en este trabajo.

Junto a esta tecnología, el chatbot está implementado en Python y utiliza Slack como canal de comunicación, una base de datos local con los productos disponibles en el frigorífico, y una base de datos externa de recetas.

El chatbot proporciona recetas al usuario teniendo en cuenta sus preferencias, realiza sugerencias sobre platos y menús, y proporciona información relativa al contenido del frigorífico. También es capaz de reconocer alimentos en imágenes y proporcionar los ingredientes necesarios, así como su método de cocinado.

Por último, se realiza una evaluación de los resultados obtenidos, las dificultades encontradas durante la fase de análisis e implementación del prototipo, y algunos posibles escenarios de desarrollo futuros.

ÍNDICE

RESUMEN	v
ÍNDICE DE FIGURAS	ix
1 Introducción.....	11
1.1 <i>Motivación</i>	12
1.2 <i>Objetivos</i>	14
1.3 <i>Definición de requisitos</i>	14
1.4 <i>Organización de la memoria</i>	15
2 Estado del arte	16
2.1 <i>Frameworks para chatbots</i>	17
2.2 <i>APIs NLP</i>	18
2.2.1 Conversation (IBM Watson)	19
2.2.2 Lex (Amazon)	20
2.2.3 Api.ai (Google)	21
2.2.4 Luis (Microsoft).....	21
2.2.5 Wit.ai (Facebook)	22
2.2.6 SiriKit (Apple)	23
2.2.7 Criterios de selección	24
2.2.7.1 Rendimiento de la tecnología NLP	24
2.2.7.2 Cobertura de idiomas.....	27
2.2.7.3 Precio del servicio.....	27
2.2.7.4 Fortaleza de la compañía proveedora	29
2.2.7.5 Existencia de módulos predefinidos	29
2.2.7.6 Control de versiones	30
2.2.7.7 Flexibilidad, autonomía y accesibilidad	30
2.2.7.8 Integración	30
2.2.7.9 Reconocimiento y síntesis de voz.....	30
2.2.8 Conclusiones	31
2.3 IBM Watson	31
2.3.1 Introducción	31
2.3.2 Watson Developer Cloud (Bluemix).....	33
2.3.2.1 Servicio Natural Language Understanding	34
2.3.2.2 Discovery.....	35
2.3.2.3 Servicio Visual recognition	36
2.3.2.4 Personality Insights	37
2.3.2.5 Tone Analyzer	38
2.3.2.6 Language translator	38
2.3.2.7 Natural Language Classifier	39
2.3.2.8 Speech to Text	40
2.3.2.9 Text to Speech	41
2.3.2.10 Conversation.....	41
3 Trabajo realizado	45
3.1 <i>Arquitectura de la solución</i>	45
3.2 <i>Interfaz de usuario</i>	47
3.2.1 Selección de plataforma de mensajería	47
3.2.1.1 Facebook Messenger	47
3.2.1.2 Whatsapp	47

3.2.1.3	Telegram.....	47
3.2.1.4	Slack	48
3.2.1.5	Skype.....	48
3.2.1.6	Twitter	48
3.2.1.7	Conclusiones	49
3.2.2	Integración con Slack.....	49
3.3	<i>Base de datos ingredientes: productos del frigorífico</i>	50
3.4	<i>Base de datos de recetas de cocina</i>	52
3.4.1	Selección de la API de recetas.....	52
3.4.1.1	Spoonacular	52
3.4.1.2	Food2Fork	53
3.4.1.3	Edamam.....	54
3.4.1.4	Yummly.....	55
3.4.1.5	Conclusiones	56
3.5	<i>Gestión del diálogo</i>	57
3.5.1	Introducción a los sistemas de diálogo	57
3.5.2	Requisitos del diálogo.....	58
3.5.3	Configuración y entrenamiento del servicio Conversation	60
3.5.3.1	Intenciones	60
3.5.3.2	Entidades	61
3.5.3.3	VARIABLES DE CONTEXTO	62
3.5.3.4	Entidades de Sistema	63
3.5.4	Flujo de diálogo.....	64
3.5.4.1	Nodo <i>Welcome</i>	65
3.5.4.2	Intención <i>#greetings</i>	65
3.5.4.3	Intención <i>#capabilities</i>	66
3.5.4.4	Intención <i>#about_yumyum</i>	67
3.5.4.5	Intención <i>#interface</i>	68
3.5.4.6	Intención <i>#positive_reaction</i>	69
3.5.4.7	Intención <i>#needed_ingredients</i>	71
3.5.4.8	Intención <i>#negative_reactions</i>	76
3.5.4.9	Intención <i>#available_ingredients</i>	78
3.5.4.10	Intención <i>#get_recipe</i>	81
3.5.4.11	Intención <i>#suggest_dish</i>	89
3.5.4.12	Intención <i>#select_option</i>	97
3.5.4.13	Intención <i>#goodbyes</i>	99
3.5.4.14	Nodo <i>Anything else</i>	100
3.6	<i>Aplicación orquestadora</i>	101
3.6.1	Constantes.....	101
3.6.2	Variables de contexto.....	102
3.6.3	Inicialización de servicios	103
3.6.4	Integración con <i>Slack</i>	104
3.6.4.1	Método <i>parse_slack_output</i>	104
3.6.4.2	Método <i>send_response</i>	105
3.6.4.3	Método <i>download_file</i>	105
3.6.5	Integración con servicio <i>Food2Fork</i>	105
3.6.5.1	Método <i>food2fork_request</i>	105
3.6.5.2	Método <i>search_recipes</i>	106
3.6.5.3	Método <i>get_recipe_from_id</i>	106
3.6.5.4	Método <i>get_recipe_id</i>	106
3.6.5.5	Método <i>get_top_rated_recipe_options</i>	107
3.6.5.6	Método <i>get_trending_recipe_options</i>	107
3.6.6	Integración con base de datos de productos	107
3.6.6.1	Método <i>database_connection</i>	107

3.6.6.2	Método <i>fetch_content</i>	107
3.6.6.3	Método <i>get_top_expired_ingredients_from_db</i>	108
3.6.6.4	Método <i>get_db_information_about_ingredients</i>	108
3.6.7	Integración con Watson Visual Recognition.....	108
3.6.7.1	Método <i>get_food_from_image</i>	108
3.6.8	Gestión de la conversación.....	109
3.6.8.1	Método <i>handle_command</i>	109
3.6.8.2	Método <i>select_option</i>	109
3.6.8.3	Método <i>yum_suggestion</i>	110
3.6.8.4	Método <i>suggest_dish</i>	110
3.6.8.5	Método <i>get_db_summary</i>	110
3.6.8.6	Método <i>get_recipe</i>	111
3.6.8.7	Método <i>msg_to_conversation</i>	111
3.6.8.8	Método <i>update_local_context</i>	111
3.6.8.9	Método <i>update_conversation_context</i>	112
3.6.8.10	Método <i>get_ingredients</i>	112
3.6.8.11	Método <i>get_recipe_options_from_available_ingredients</i>	112
3.6.8.12	Método <i>get_ingredients_information</i>	112
3.6.8.13	Método <i>image_food_recognition</i>	113
4	Conclusiones	114
4.1	<i>Chatbots</i>	114
4.1.1	Beneficios.....	115
4.1.2	Riesgos	115
4.1.3	Oportunidades.....	115
4.1.4	Retos.....	116
4.2	<i>Watson</i>	116
4.3	<i>Resumen del trabajo realizado y cumplimiento de objetivos.</i>	119
4.4	<i>Dificultades encontradas durante el desarrollo</i>	120
4.5	<i>Evaluación del prototipo</i>	121
4.6	<i>Recomendaciones</i>	122
5	Líneas futuras.....	123
5.1	<i>Reconocimiento vocal</i>	123
5.2	<i>Funcionalidad lista de la compra</i>	123
5.3	<i>Integración con frigorífico IoT</i>	124
5.4	<i>Multi-idioma</i>	124
5.5	<i>Coach nutricional</i>	124
5.6	<i>Conversaciones personalizadas y análisis de sentimientos</i>	124
5.7	<i>Notificaciones productos perecederos</i>	124
5.8	<i>Cuadro de mando</i>	124
Apéndice I: Código	125
Referencias.....		136

ÍNDICE DE FIGURAS

Figura 1 - Evolución de las interfaces de usuario	12
Figura 2 - Ecosistema de chatbots	16
Figura 3 - Evolución temporal principales APIs de NLP	18
Figura 4 - Precisión de detección	24
Figura 5 - Precisión vs Sensibilidad	25
Figura 6 - Porcentaje de falsos positivos	25
Figura 7 - Velocidad de aprendizaje para cada plataforma	26
Figura 8 - Tiempo medio de respuesta (segundos)	26
Figura 9 - Rendimiento vs velocidad de respuesta	27
Figura 10 - Precio (USD) por cada 1K peticiones de texto	28
Figura 11 - Rendimiento vs precio	29
Figura 12 - Ecosistema de servicios Bluemix	33
Figura 13 - Catálogo de servicios IBM Watson Bluemix	33
Figura 14 - Ejemplo de extracción de entidades	35
Figura 15 - Ejemplo de extracción de roles (relación “sujeto-acción-objeto”)	35
Figura 16 - Ejemplo de extracción de relación entre entidades	35
Figura 17 - Flujo de trabajo con servicio Discovery	36
Figura 18 - Comparativa Watson Discovery y Watson NLU	36
Figura 19 - Ejemplo reconocimiento facial realizado con el servicio Visual Recognition	37
Figura 20 - Identificación rasgos psicológicos realizado con el servicio Personality Insights	37
Figura 21 - Ejemplo servicio Tone Analyzer	38
Figura 22 - Ejemplo Language Translator: La canción del pirata (José de Espronceda)	39
Figura 23 - Ejemplo CSV entrenamiento del servicio Natural Language Classifier	40
Figura 24 - Ejemplo del servicio Natural Language Classifier	40
Figura 25 - Arquitectura típica de una aplicación de conversación	41
Figura 26 – Watson Conversation: nodo de la conversación	42
Figura 27 – Watson Conversation: orden de evaluación de los nodos	43
Figura 28 – Watson Conversation: ejemplo de uso de slots	43
Figura 29 - Cuadro de mando básico proporcionado por el servicio de conversación	44
Figura 30 - Arquitectura de la solución	46
Figura 31 – Tabla de base de datos con los productos del frigorífico	51
Figura 32 - Vectores de ingredientes API Spoonacular	52

Figura 33 - Ontología de alimentos de Spoonacular	53
Figura 34 - Mapa de uso de la API Spoonacular	53
Figura 35 - Tarifas de uso para la API Spoonacular	53
Figura 36 - Fuentes de recetas de Food2Fork	54
Figura 37 - Planes de uso de la API Food2Fork	54
Figura 38 - Tarifas uso API Edamam	55
Figura 39 - Fuente de rectas de Yummly	55
Figura 40 - Tarifas ofrecidas por la API Yummly	56
Figura 41 - Esquema simplificado de la arquitectura de un sistema de diálogo	57
Figura 42 - Entradas adicionales a las interfaces de conversación	58
Figura 43 – Lista de intenciones servicio conversación de Watson	60
Figura 44 – Lista de entidades del servicio de conversación de Watson	62
Figura 45 – Entidades de sistema disponibles en el servicio de conversación de Watson	63
Figura 46 – Leyenda de los diagramas de flujo de la conversación	64
Figura 47 – Flujo de la conversación: nodo <i>Welcome</i>	65
Figura 48 – Intención <i>#greetings</i> : respuesta al usuario	66
Figura 49 – Nodo que gestiona la intención <i>#greetings</i>	66
Figura 50 – Intención <i>#capabilities</i> : respuesta al usuario	67
Figura 51 – Nodo intención <i>#capabilities</i>	67
Figura 52 – Nodos intención <i>#about_yumyum</i> y salto a <i>#capabilities</i>	68
Figura 53 – Intención <i>#about_yumyum</i> : respuesta	68
Figura 54 – Configuración de la intención <i>#about_yumyum</i>	68
Figura 55 – Nodo intención <i>#interface</i> y salto a <i>#capabilities</i>	69
Figura 56 – Intención <i>#interface</i> : respuesta al usuario	69
Figura 57 – Nodo intención <i>#interface</i>	69
Figura 58 – Intención <i>#positive_reaction</i> : respuesta	70
Figura 59 – Nodo intención <i>#positive_reaction</i>	70
Figura 60 - Flujo de la conversación para la intención <i>#needed_ingredients</i>	71
Figura 61 – Intención <i>needed_ingredients</i> : respuesta chatbot	72
Figura 62 – Intención <i>needed_ingredients</i> : varios tipos de respuestas del usuario	72
Figura 63 – Intención <i>needed_ingredients</i> : respuesta positiva	73
Figura 64 – Intención <i>needed_ingredients</i> : respuesta negativa	73
Figura 65 – Intención <i>needed_ingredients</i> : primera respuesta no válida	74
Figura 66 – Intención <i>needed_ingredients</i> : flujo de la conversación en el servicio Watson Conversation	75
Figura 67 - Flujo de la conversación para la intención <i>#negative_reactions</i>	76
Figura 68 – Intención <i>#negative_reaction</i> : respuesta (contador <3)	77
Figura 69 – Intención <i>#negative_reaction</i> : respuesta (contador >3)	77
Figura 70 – Intención <i>#negative_reaction</i> : flujo de diálogo en el servicio Watson Conversation	78
Figura 71 – Flujo de la conversación para la intención <i>#available_ingredients</i>	78

Figura 72 – Intención <code>#available_ingredients</code> : respuestas a preguntas sobre un determinado producto	79
Figura 73 – Intención <code>#available_ingredients</code> : resumen del contenido	79
Figura 74 – Intención <code>#available_ingredients</code> : configuración slot <code>@ingredients</code>	80
Figura 75 – Intención <code>#available_ingredients</code> : configuración servicio Watson Conversation	80
Figura 76 – Flujo de la conversación para la intención <code>#get_recipe</code>	81
Figura 77 - Intención <code>#get_recipe</code> : comprobación formato imagen	82
Figura 78 - Intención <code>#get_recipe</code> : respuestas tras petición servicio Visual Recognition	83
Figura 79 - Intención <code>#get_recipe</code> : receta a partir de una imagen	83
Figura 80 – Intención <code>#get_recipe</code> : receta a partir del nombre del plato	84
Figura 81 – Intención <code>#get_recipe</code> : respuesta si no encuentra recetas para la entrada del usuario	84
Figura 82 – Intención <code>#get_recipe</code> : solicitud del nombre de la receta o imagen	85
Figura 83 – Intención <code>#get_recipe</code> : sugerencia propia por falta de entrada del usuario	85
Figura 84 – Intención <code>#get_recipe</code> : check <code>@dish</code> y check <code>\$image</code>	87
Figura 85 – Intención <code>#get_recipe</code> : nodo “ask for <code>@dish name or photo</code> ”	88
Figura 86 – Intención <code>#get_recipe</code> : último nodo de la conversación	88
Figura 87 – Flujo de la conversación para la intención <code>#sugest_dish</code>	89
Figura 88 – Intención <code>#sugests_dish</code> : intolerancia válida	90
Figura 89 – Intención <code>#sugest_dish</code> : respuesta afirmativa intolerancias	90
Figura 90 – Intención <code>#sugest_dish</code> : ingredientes y/o tipo de cocina proporcionados (sin intolerancias)	91
Figura 91 – Intención <code>#sugest_dish</code> : petición no contiene tipo de cocina ni ingredientes	91
Figura 92 – Intención <code>#sugest_dish</code> : el usuario responde proporcionando tipo de cocina	92
Figura 93 – Intención <code>#sugest_dish</code> : sin preferencias respecto a ingredientes o tipo de cocina	93
Figura 94 – Intención <code>#sugest_dish</code> : respuesta afirmativa sobre preferencias ingredientes/tipo de cocina	93
Figura 95 - Intención <code>#sugest_dish</code> : nodos de la conversación	94
Figura 96 – Intención <code>#sugest_dish</code> : nodo “ <i>cuisine type or ingredients found</i> ”	96
Figura 97 – Intención <code>#sugest_dish</code> : nodo “ask for the <code>@ingredients</code> or <code>@cuisine_type</code> ”	97
Figura 98 – Intención <code>#select_option</code> : respuesta	98
Figura 99 – Nodo de la intención <code>#select_option</code>	98
Figura 100 – Intención <code>#goodbyes</code> : respuesta	99
Figura 101 – Nodo intención <code>#goodbyes</code>	99
Figura 102 – Nodo <i>Anything else</i> : respuesta	100
Figura 103 – Configuración del nodo <i>Anything else</i>	100
Figura 104 – Aplicación orquestadora: programa principal	101
Figura 105 – Aplicación orquestadora: constantes	102
Figura 106 – Aplicación orquestadora: Inicialización de las variables de contexto	103
Figura 107 – Aplicación orquestadora: peticiones al servicio Watson Conversation	103
Figura 108 – Aplicación orquestadora: Inicialización de servicios	104
Figura 109 – <i>Handlers</i> en Watson Conversation	119

1 INTRODUCCIÓN

El término **bot** es una abreviatura de la palabra robot. Los bots son programas informáticos que interactúan en la red simulando ser una persona. Los **chatbots** fueron uno de los primeros bots desarrollados, un chatbot es un sistema que entiende el lenguaje natural y tiene inteligencia sobre un cierto contexto de forma que puede interactuar con el usuario para resolver un determinado problema.

Los asistentes virtuales llevan operando en el mercado desde hace años, sin embargo, es ahora cuando cuentan con las plataformas y tecnologías ideales para su crecimiento y expansión. Según TechEmergence [1], los chatbots son hoy en día la aplicación de consumo número uno de inteligencia artificial, y se espera que durante los próximos cinco años siga siendo así. Gartner [2] también predice que en 2020 más del 85% de las interacciones con los clientes se gestionarán sin intervención humana.

La importancia que los gigantes tecnológicos como Google, Facebook, Microsoft, IBM y Amazon están dando a las tecnologías cognitivas y, más concretamente, a los chatbots es un fuerte indicador de que esta tecnología jugará un papel clave en el futuro.

Otro punto clave en la creciente popularidad de los chatbots, ha sido la importancia con la que actualmente cuentan las aplicaciones de mensajería, puesto que concentran la mayor cantidad de usuarios activos en internet¹. Con la ayuda de los chatbots los usuarios no tienen que buscar la información o llamar por teléfono, sino que pueden interactuar directamente con ellos por su canal favorito, escribiendo mensajes directos para llevar a cabo pedidos, compras, cambios de reservas y muchas otras operaciones.

Las tareas que desempeñan los chatbots son variadas, los hay que prestan soporte a los usuarios resolviendo dudas, brindando información, realizando acciones por ellos, o recopilando información. Sus aplicaciones son casi ilimitadas, aunque quizás la más extendida sea la del servicio de atención al cliente. Los chatbots eran simples al principio, a menudo se limitaban a ventanas emergentes del navegador web y solo podían enviar saludos directos a los visitantes del sitio web, pero gracias a las innovaciones en el área de la inteligencia artificial, las tecnologías cognitivas y procesado del lenguaje natural cada vez son capaces de desempeñar funciones de mayor complejidad y comprender mejor el lenguaje natural.

Lo que al principio podía parecer una moda pasajera o una estrategia de marketing, se está convirtiendo en una necesidad real, debido a que los chatbots tienen potencial para automatizar tareas que consumen mucho tiempo, dinero y mano de obra. Además proporcionan interfaces de lenguaje natural que están transformando la forma en que interactuamos con la tecnología, cambiando la experiencia del usuario y la forma de consumir los contenidos, haciendo que la interacción sea más simple y eficiente. Prueba de ello es que los usuarios de smartphones se descargan cada vez menos aplicaciones y tienden a pasar más tiempo en apps de mensajería instantánea. Podría decirse que las conversaciones son la nueva interfaz de usuario, y los bots son las nuevas aplicaciones.

¹ Por ejemplo Whatsapp cuenta con 1 billón de usuarios y Facebook Messenger con 800 millones de usuarios.

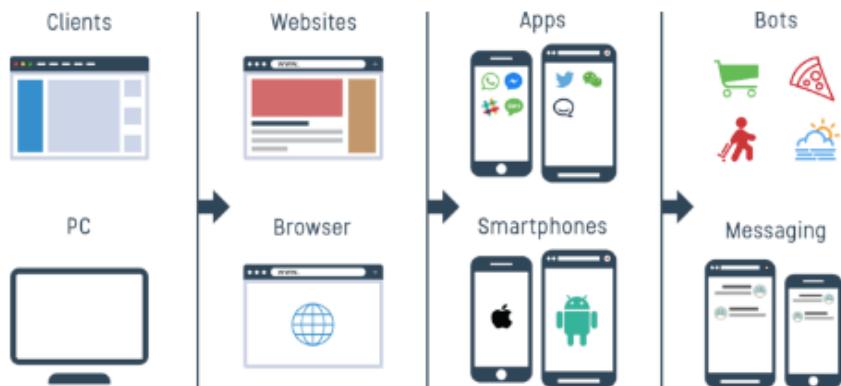


Figura 1 - Evolución de las interfaces de usuario

Sin embargo, la reputación que acompaña a los chabots no siempre es positiva, ya que muchos de ellos se emplean con fines poco éticos, como por ejemplo automatizar la propaganda, extraer información delicada del usuario, descifrar las passwords de entrada de páginas web, crear perfiles falsos en las redes sociales para conseguir likes o ganar sorteos, etc. Tampoco ha contribuido a mejorar su reputación algunos errores como Tay² de Microsoft o Barista³ de Starbucks. Sin embargo, creaciones como Siri, Cortana, Alexa o Google Now han hecho recuperar la confianza de los usuarios y nos han dado una idea de cómo sería comunicarse directamente con las máquinas y las ventajas que pueden proporcionar en un futuro cercano.

1.1 Motivación

Los chatbots han despertado el interés de las grandes empresas debido a las numerosas ventajas que conlleva su uso, a continuación enumeraremos algunas de las más destacadas:

- **Accesibilidad:**
 - Operan en cualquier dispositivo a través de las plataformas de mensajería. Mantener una conversación no requiere adaptación, los usuarios no tienen que aprender a utilizar una nueva aplicación o dispositivo.
 - El uso del texto y/o de la voz como interfaz facilita su acceso a personas con diferentes edades, formación técnica, o incluso con discapacidades.
 - A diferencia de las aplicaciones, los chatbots no se descargan, no es necesario actualizarlos y no ocupan espacio en la memoria del teléfono.
 - El usuario no necesita ser autenticado, pueden usarse directamente en la plataforma de chat.
 - Se pueden compartir fácilmente entre personas que ya están en una plataforma de chat e incluso se usan en grupos de chat.
- **Disponibilidad:** permiten que el usuario tenga acceso a la información en todo momento (24/7), ya que los chatbots no duermen, no sufren bajas, ni disfrutan de vacaciones, etc.
- **Interactividad:** permite notificar a los usuarios por medio de los chats sin esperar a que ellos inicien la conversación y se pongan en contacto.
- **Mejora la experiencia de usuario:** un estudio de One Reach asegura que el 64% de los clientes prefiere soluciones por texto antes que llamar al servicio de atención al cliente. Las interacciones

² Tras un día de su lanzamiento Tay comenzó a escribir mensajes racistas, y cargados de contenido sexual en respuesta a otros usuarios de Twitter.

³³ Su popularidad provocó que en algunas de las cafeterías con mayor afluencia se produjeran cuellos de botella y que algunos clientes se marcharan del establecimiento sin ser atendidos.

con texto, son más divertidas, íntimas y descriptivas que otras interfaces de usuario.

- **Comunicaciones personalizadas:** pueden reunir y monitorizar fácilmente los datos de los usuarios, por ejemplo, incluyendo cookies para conocer su comportamiento, de este modo se evita pasar al usuario por interminables encuestas o preguntas.
- **Escalabilidad:** pueden atender a varios usuarios al mismo tiempo. Esto también hace que la respuesta sea proporcionada en un tiempo mucho menor.
- **Ahorro de dinero:** la inversión que requiere no es elevada en comparación con el rendimiento que se deriva de su uso, ya que puede ahorrar costes en trabajo humano.

Las grandes compañías tienen preocupaciones tales como hacer que sus clientes comprendan más, captar nuevos clientes, promocionar sus productos o maximizar la presencia en lugares donde se encuentren potenciales clientes, en definitiva, aumentar sus ganancias. Los chatbots son herramientas muy útiles para complementar sus estrategias de marketing porque pueden ayudarles a conseguir dichos objetivos. Sin embargo, las grandes empresas no tienen la exclusividad en el uso de chatbots. En los últimos años las tecnologías para crear chatbots se han democratizado, y se están poniendo al alcance de cualquier persona con los conocimientos técnicos necesarios para construir su propio chatbot. Muchas de las plataformas de mensajería también facilitan su despliegue proporcionando frameworks o tecnologías para implementar chatbots y/u ofreciendo conectores e interfaces para comenzar a usarlos.

El objetivo principal de este trabajo es evaluar las tecnologías disponibles y construir un asistente de recetas de cocina conectado con una base de datos de los alimentos disponibles en nuestro hogar. Se evaluarán diferentes tecnologías para construcción de chatbots y se seleccionará la más interesante y prometedora para nuestro caso de uso y para el estudio realizado en este trabajo.

El asistente se encargará de buscar recetas a partir de los ingredientes que tiene a su disposición, o a partir de una foto de algún plato que hayamos visto, también buscará recetas de un determinado tipo de cocina (vegana, sin gluten, hipocalórica, etc.), alertará cuando detecte un alimento a punto de caducar, o cuando se esté agotando alguno de los productos considerados por el usuario como básicos.

El asistente trabajará por nosotros y nos aportará entre otros los siguientes beneficios:

- Ahorro de tiempo, porque ayuda a decidir qué cocinar y a organizar la compra.
- Mejora de la alimentación, porque ayuda a llevar una alimentación saludable sin consumir tantos precocinados y comida para llevar.
- Evita el desperdicio de alimentos porque optimiza la utilización de los ingredientes, alertando cuando estén próximos a caducar y proponiendo recetas que den prioridad a los alimentos con menor tiempo de vida.
- Ahorro de dinero porque evitará desperdicios de comida y abusar de la comida a domicilio o los precocinados, que suelen ser más caros que los productos frescos.
- Ayuda a descubrir nuevos sabores y platos gracias a su amplia gama de recetas.
- Ayuda a las personas con alergias o problemas alimentarios.
- Permite compartir recetas con otros usuarios gracias a su interfaz de mensajería.
- Ayuda a mejorar las habilidades culinarias y a descubrir nuevos trucos de cocina.

Existen muchas aplicaciones que ya ayudan a resolver estos problemas, pero ninguna de ellas ofrece una interfaz en lenguaje natural a través de una plataforma de mensajería, y que además utilice reconocimiento de imágenes para proporcionar las instrucciones con las que elaborar un plato del que sólo cuentas con una foto o un enlace a una web.

1.2 Objetivos

Los principales objetivos académicos que persigue este trabajo son:

- Introducción a los sistemas de diálogo.
- Introducción a la construcción de asistentes conversacionales.
- Comparativa de tecnologías de procesado de lenguaje natural disponibles en el mercado.
- Estudio teórico de los servicios *IBM Watson* disponibles en la plataforma cloud *Bluemix*.
- Análisis crítico de los servicios *IBM Watson Conversation* y *Watson Visual Recognition*.
- Implementación de un prototipo de asistente conversacional para frigorífico inteligente.
- Evaluación de los resultados: dificultades, inconvenientes y ventajas encontrados durante la fase de análisis e implementación del prototipo.

1.3 Definición de requisitos

Es necesario que la conversación sea abierta, es decir, permitiremos que el usuario escriba texto libre en todo momento, pero partiremos de un alcance inicialmente acotado.

En este sentido, los requisitos funcionales que impondremos al prototipo serán:

- **[RF01]** El asistente estará escuchando y saludará al usuario cuando éste se dirija a él por primera vez.
- **[RF02]** El asistente será capaz de entender la intención del usuario cuando éste se dirija a él usando lenguaje natural, y proporcionar la respuesta más adecuada.
- **[RF03]** El asistente podrá sugerir platos que cocinar según las preferencias del usuario en cuanto al tipo de cocina (española, francesa, china, italiana, vegana, baja en calorías, especiada, rápida...) o ingredientes a utilizar.
- **[RF04]** El asistente pedirá información al usuario sobre sus intolerancias alimenticias y las tendrá en cuenta en la sugerencia de platos que proporcione.
- **[RF05]** El asistente podrá realizar sugerencias de recetas propias, es decir sin ninguna entrada o información de preferencias del usuario. En este caso, dará prioridad a los ingredientes más próximos a caducar de los que disponga en su hogar, y a las recetas con más puntuación y más populares en su fuente de datos.
- **[RF06]** El asistente será capaz de clasificar e identificar a partir de una foto compartida por el usuario de qué plato se trata, y proporcionar una lista de los ingredientes del mismo, así como un enlace al método de cocinado.
- **[RF07]** El asistente proporcionará los ingredientes y el método de cocinado a partir del nombre de un plato dado por el usuario.
- **[RF08]** El asistente proporcionará al usuario información del contenido de su nevera y del estado de sus productos: cantidad disponible de un cierto producto, productos caducados y productos próximos a caducar.
- **[RF09]** Si el usuario lo solicita, el asistente proporcionará información sobre sus capacidades y sobre cómo dirigirse a él para realizar una petición.

1.4 Organización de la memoria

La memoria se ha dividido en los siguientes capítulos:

Capítulo 1. Introducción: donde se presenta una introducción general al documento y se exponen los objetivos y requisitos específicos del trabajo realizado.

Capítulo 2. Estado del arte: donde se realiza un estudio comparativo de los principales frameworks para creación de chatbots y de las APIs para procesado de lenguaje natural. Por último, se realizará una descripción de los servicios IBM Watson.

Capítulo 3. Trabajo realizado: donde se muestra una visión general del funcionamiento del prototipo, el canal de comunicación, los servicios externos que utiliza y las fuentes de datos seleccionadas, así como todo el detalle de su implementación y configuración.

Capítulo 4. Conclusiones: donde se presenta el resumen de los resultados obtenidos y de los aspectos más destacables durante la realización del proyecto.

Capítulo 5. Líneas futuras: donde se muestran algunas propuestas de mejora del prototipo, así como posibles escenarios de evolución en el futuro.

Referencias: contiene la bibliografía consultada para la realización del proyecto.

Anexo I. Código: contiene el código del prototipo.

2 ESTADO DEL ARTE

Actualmente existe una increíble cantidad de plataformas y herramientas para la creación de chatbots, con diferentes niveles de complejidad, capacidad expresiva y capacidad de integración.

A continuación veremos las herramientas, frameworks e implementaciones más extendidas en la actualidad, y seleccionaremos la que mejor se ajuste a nuestras necesidades.

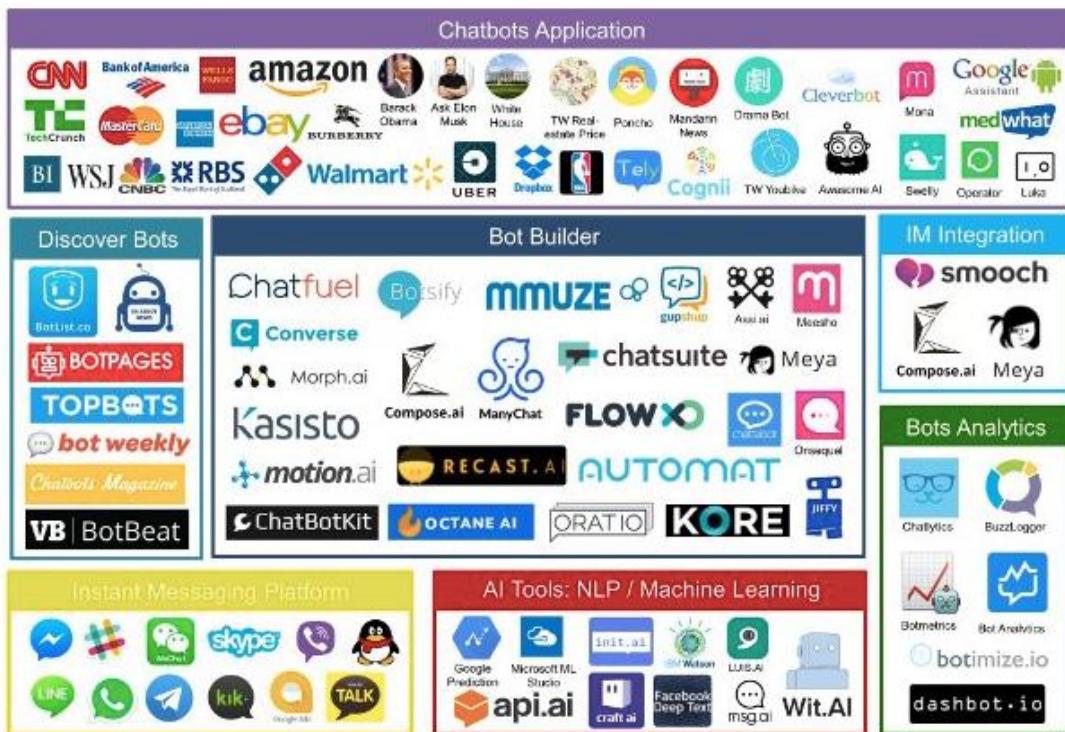


Figura 2 - Ecosistema de chatbots

Según el caso de uso abordado por el chatbot, algunas plataformas son más apropiadas que otras. La selección de la plataforma dependerá del tipo de chatbot que se desee construir y de cuál será su cometido. Una primera diferenciación a tener en cuenta es si debe tener fuertes habilidades de conversación o, por el contrario, está orientado a un objetivo específico y no necesita entender profundamente lo que el usuario dice o recordar todo el contexto de la conversación (chatbots de entretenimiento o chatbots para FAQs).

La mayoría de aplicaciones de mensajería instantánea han incorporado facilidades para el lanzamiento de chatbots en sus plataformas, ya sean para la distribución de contenidos, la atención al cliente, el comercio electrónico, etc. En torno a ellas han aparecido multitud de startups que están haciendo negocio ofreciendo servicios para la creación de bots en pocos pasos y sin necesidad de programar. Son una puerta abierta para los usuarios sin conocimientos técnicos, aunque por ahora tienen muchas limitaciones y se utilizan como un campo de exploración más que de negocio.

Construir un asistente robusto de inteligencia artificial todavía es bastante complicado. Los gigantes tecnológicos abordan este problema proporcionando soluciones de comprensión de lenguaje natural (NLU) que los desarrolladores podemos usar para aumentar nuestras aplicaciones con capacidades de lenguaje natural.

Podemos distinguir, por tanto, 2 tipos de plataformas o estrategias para la construcción de chatbots:

- Frameworks para chatbots
- APIs NLP

2.1 Frameworks para chatbots

Son plataformas orientadas a usuarios no técnicos. Por lo general, es fácil codificar un chatbot sin tener habilidades de programación y sin tener conocimientos de aprendizaje automático o de procesamiento de lenguaje natural. La idea clave es que el **usuario no tenga que preocuparse por los detalles técnicos**, y poder lanzar bots en tres o cuatro pasos en plataformas de mensajería o redes sociales como Facebook Messenger, Telegram, Slack o WeChat.

Hay una **gran cantidad de frameworks para chatbots**, algunos de los más extendidos son:

- **Chatfuel**: uno de los más usados del mercado, fue lanzado en 2015 por los desarrolladores rusos Dmitry Dumik y Artem Ptashnik. Hasta ahora, estos chatbots están disponibles únicamente para Facebook Messenger y Telegram. Existen chatbots implementados con este framework en los portales de noticias Forbes y TechCrunch.
- **ManyChat**: los chatbots generados con este framework tienen como objetivo invitar a los usuarios a suscribirse al sitio a través de Facebook Messenger y tener una comunicación interactiva con ellos. Existe una versión gratuita y otra de pago. Proporciona una interfaz drag and drop.
- **Massively**: es una plataforma de desarrollo de bot de código abierto, que permite implementarlos en cualquier plataforma de chat (Kik, Facebook Messenger, Skype, Telegram, etc.) o incrustados en un sitio web. El servicio de desarrollo es gratuito, pero la compañía también ofrece un servicio de soporte de pago a los clientes.
- **Motion.ai**: dispone de una interfaz gráfica para crear flujos de conversación. Proporciona una API para integración con otras aplicaciones y APIs de NLP poco potentes. También incorpora un cuadro de mando por defecto para visualizar los indicadores más relevantes de las conversaciones del bot.
- **Recast.ai**: plataforma colaborativa para construir, entrenar, desplegar y monitorizar bots. Es más que un framework para bots, porque dispone además de una potente API que proporciona funciones de procesamiento de texto, conversación (extracción de intención, entidades) y análisis de imágenes. Es gratuita para desarrolladores siempre que el código sea público en Github, pero también tiene tarifas para proyectos privados. Soporta múltiples idiomas (inglés, francés y anuncia que próximamente en español).

Las ventajas que presentan este tipo de plataformas es que tienen una **baja curva de aprendizaje** y por tanto se puede desarrollar un chatbot muy rápidamente.

Aunque todas estos framework parecen muy similares, existen importantes diferencias en la madurez, la usabilidad de la GUI y el poder de procesamiento del lenguaje natural. Suelen proporcionar una **interfaz gráfica** de usuario, a veces no muy intuitiva, y cuando la lógica del chatbot se vuelve más compleja se vuelve difícil de manejar.

Todas ellas están orientadas a realizar tareas muy concretas, el ejemplo más común es "pedir una pizza". Son ideales para bots simples, pero tienen **poca o ninguna capacidad de procesamiento del lenguaje natural**. Algunas de estas plataformas no pueden realizar extracción de información.

Este tipo de plataformas **no son válidas para proyectos comerciales a gran escala**. Las conversaciones no pueden ser muy complejas y, por lo general, no es posible integrar recursos externos, como componentes específicos de Procesamiento de Lenguaje Natural y Machine Learning. Sin embargo, son plataformas realmente buenas para proyectos a pequeña escala, por lo general para agregar rápidamente una funcionalidad de chatbot a una página, como muestra Chatfuel en Facebook.

2.2 APIs NLP

El **lenguaje natural** es un elemento **fundamental** para la construcción de **chatbots inteligentes** con los que el usuario pueda interaccionar libremente sin necesidad de seguir un conjunto de reglas básicas, y que a la vez sean capaces de proporcionar un rango amplio de posibles respuestas.

Desde un punto de vista conceptual, hay dos técnicas principales de programación del lenguaje natural que se han hecho populares con las tecnologías chatbot:

- **Procesamiento del lenguaje natural** (NLP): abarca varias disciplinas que abordan la interacción entre los sistemas informáticos y los lenguajes naturales humanos. Incluye varias subdisciplinas, como el análisis del discurso, la extracción de relaciones, la comprensión del lenguaje natural, y algunas otras áreas de análisis del lenguaje.
- **Comprensión del lenguaje natural** (NLU): es un subconjunto de NLP que se centra en la comprensión lectora y el análisis semántico.

El software NLP no busca palabras clave en el texto como un motor de búsqueda, sino que utiliza el conocimiento de la **estructura de oraciones, expresiones idiomáticas y patrones aprendidos** para intentar hacer coincidir lo que dice el usuario con una **intención** que ha sido clasificada. Esto se traduce en que el chatbot debe ser programado para identificar ciertas cosas que la gente quiere de él, y **actuar sobre ellas**. Cada vez que una intención se clasifica y se usa en una conversación, el chabot puede proporcionar una acción o respuesta rápida. La **inteligencia** del chatbot reside en el **entrenamiento** que se haga del mismo (intenciones, entidades, diálogos).

Sin embargo, el **lenguaje natural** no es natural para ninguna máquina, ya que es demasiado **desestructurado y ambiguo**. En consecuencia, el procesamiento del lenguaje natural es **computacionalmente costoso**. En contraposición, la curva de aprendizaje para usar una **API conversacional** es mucho **más simple**, ya que la estructura y la previsibilidad de las API REST proporcionan eficiencia y estabilidad.

Como consecuencia de lo anterior, la evolución de las plataformas chatbot y las plataformas de procesamiento de lenguaje natural han ido de la mano, pero mientras que las primeras están impulsadas principalmente por los proveedores de plataformas de mensajería como Facebook o WeChat, los principales avances en las tecnologías de procesamiento de lenguaje natural provienen de la **plataforma en la nube** y de **proveedores** de servicios como **Google, IBM o Amazon** que ofrecen sus servicios a través de APIs conversacionales.

Como puede observarse en la siguiente figura, en los dos últimos años se ha producido una gran proliferación de plataformas proveedoras de servicios de procesado de lenguaje natural que se ofrecen a través de una API en la nube, y encontramos que todas las grandes empresas tecnológicas han lanzado sus productos para competir en este mercado de soluciones NLP.

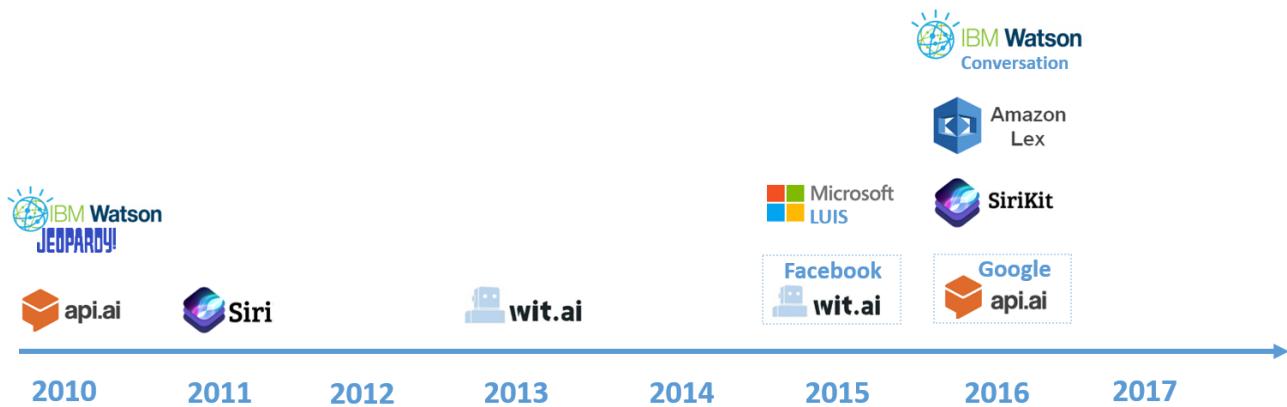


Figura 3 - Evolución temporal principales APIs de NLP

A continuación presentaremos las principales características de los proveedores de servicios NLP más

relevantes en la actualidad: Alexa (Amazon), Api.ai (Google), Luis (Microsoft), Sirikit (Apple), Watson (IBM) y Wit (Facebook). Pero antes de comenzar veamos algunos conceptos básicos de los sistemas de diálogo de lenguaje natural, y que se repetirán a lo largo del estudio comparativo que se realizará en este apartado.

Veámoslo con un ejemplo, supongamos la siguiente frase “*Quiero reservar un vuelo de Madrid a Sevilla para mañana a las 15 horas*”:

- **Intención:** ¿Cuál es el objetivo u objetivos de nuestra interacción?
 - Propósito de la entrada de un usuario.
 - Las intenciones pueden tener opcionalmente argumentos llamados *slots*.
 - En el ejemplo anterior, supongamos que nuestro chatbot gestiona llamadas de una agencia de viajes y que puede realizar funciones como informar sobre horarios, reservar vuelos y cancelar vuelos. En este caso, la intención es “*reservar vuelo*”.
- **Locución:** ¿Cómo transmite el usuario la intención?
 - Frases escritas o habladas que invocan la intención. Expresiones de muestra.
 - En el ejemplo anterior la locución sería la frase, tal cual la ha expresado el usuario. Otra posible locución sería “*Quiero un vuelo para Sevilla*” o “*Quiero ir mañana a Sevilla desde Madrid*”, etc.
- **Slot o entidad:** ¿Qué información necesitamos para lograr el objetivo?
 - Información de entrada necesaria para ejecutar la intención. Puede ser obligatorio u opcional.
 - Término u objeto que es relevante para la intención.
 - En el ejemplo, los slots podrían ser origen, destino, fecha y número de billetes, todos ellos obligatorios, porque si falta alguno de los datos no se podría ejecutar la acción que indica la intención.

2.2.1 Conversation (IBM Watson)

- Fue lanzado por **IBM** en **2016**, aunque en **2010** ya se habían dado a conocer sus habilidades al proclamarse vencedor frente a 2 dos humanos en el concurso Jeopardy.
- El Servicio de conversación⁴ de Watson (WCS) puede definir aspectos de **NLP** como **intenciones**, **entidades** y simular **conversaciones** enteras. Está construido sobre una red neuronal (mil millones de palabras de Wikipedia) que entiende intenciones, interpreta entidades y diálogos.
- Incorpora técnicas de **expansión lingüística** (fuzzy matching) que mejoran sustancialmente su rendimiento, incluso con datos de mala calidad (caso más común). Aunque estas técnicas no funcionan con la misma efectividad en todos los idiomas.
- Trabaja con **slots** (datos de entrada para satisfacer la intención).
- Dispone de una **interfaz gráfica** para la **definición de entidades e intenciones**, y también se pueden importar y exportar a CSV. Existe una interfaz gráfica para la definición del **flujo** de la **conversación**, aunque en este caso es poco manejable, no es intuitiva, no permite deshacer cambios ni llevar un control sobre ellos, salvo exportando el espacio de trabajo a JSON y versionando dicho JSON.
- La **configuración programática** también es posible si no se desea utilizar la interfaz gráfica.
- WCS se usa generalmente junto con otros servicios de Watson como *Natural Language Understanding* o *Knowledge Studio*, o con *Personality Insight* y *Tone Analyzer* para hacer un análisis de personalidad y de sentimientos.
- Para incorporar la **voz** a la conversación se puede utilizar en conjunto con los servicios **Speech To Text** y **Text To Speech** que realizan transcripción de voz a texto, y síntesis de voz, respectivamente.

⁴ <https://www.ibm.com/watson/services/conversation>

- Soporta **9 idiomas** diferentes: inglés, español, portugués, coreano, japonés, italiano, alemán, francés, árabe. No toda la funcionalidad está soportada en todos los idiomas, inglés es el idioma en el que está implementada toda la funcionalidad disponible. Además, soporta 3 idiomas más de forma experimental: chino, checo y holandés.
- Está accesible a través de la **plataforma de cloud** de IBM, *Bluemix*. Y se proporciona a través de una **interfaz HTTP REST**.
- Está disponible de **forma gratuita** por un mes, aunque con algunas **restricciones** y limitaciones de uso y respuesta. La **duración** estándar de la cuenta gratuita es de un **mes**, pero para algunas comunidades, como por ejemplo la universitaria, estas cuentas se extienden considerablemente en el tiempo (entre 6 y 12 meses). También dispone de **cuentas de pago por uso** y de **cuentas Premium**.
- Existe bastante **documentación** en inglés, y multitud de ejemplos, tanto en la web de IBM con en Github.
- Proporciona **SDKs** para diferentes lenguajes de programación y sistemas operativos, como son Node, Java, Python, iOS, Android, etc. También puede usarse la API de forma nativa o usando curl.
- Puede ser desplegado en **diferentes plataformas o canales**, como por ejemplo Facebook Messenger, Slack, Twitter, etc.
- Su **rendimiento** de acuerdo con un estudio realizado por Midbowser y Chatbots Journal [3], es la primera elección para la construcción de chatbots para el 61% de las empresas.

2.2.2 Lex (Amazon)

- Fue lanzado por **Amazon** en **2016**.
- Está construido con las mismas tecnologías de Deep learning que se utilizaron para Alexa.
- Amazon Lex es un servicio⁵ para crear interfaces de conversación en cualquier aplicación con voz y texto.
- Proporciona capacidades de **reconocimiento del habla** (convierte voz en texto) y **comprensión del lenguaje natural** para reconocer la intención del texto.
- Trabaja con **slots**.
- Para crear un chatbot, primero se deben definir las intenciones. Por cada intención, se deben añadir ejemplos de locuciones. Se debe definir el flujo de la conversación y, por último, la lógica de negocio necesaria para ejecutar la acción o respuesta.
- Disponible con tarifa de **pago por uso**.
- Accesible mediante **API REST**.
- **Integración** sencilla con muchos otros servicios de la plataforma de AWS:
 - *AWS Lambda*: para ejecutar código sin aprovisionar ni administrar servidores.
 - *Amazon Cognito*: para autenticar a los usuarios.
 - *Amazon Polly*: convierte el texto en voz.
 - *AWS Mobile Hub*: puede utilizarse para aprovisionar automáticamente bots.
- Soporte **multiplataforma**, se puede usar *Amazon API Gateway* y *AWS Lambda* para la integración de chatbots con servicios de mensajería como Facebook Messenger, Slack y Twilio SMS. Aunque usando AWS Lambda se puede integrar con cualquier otro sistema de mensajería que tenga una API.

⁵ <https://aws.amazon.com/lex>

- **SDKs** soportados para creación de bots: iOS, Android, Java, JavaScript, Python, CLI, .NET, Ruby on Rails, PHP, Go y CPP.
- **Sólo disponible en inglés estadounidense.**

2.2.3 Api.ai (Google)

- Fue **lanzada en 2010** y **adquirido por Google en 2016**.
- Recientemente se ha cambiado el nombre a **Dialogflow**.
- Plataforma⁶ con capacidades de NLP y NLU. Tiene funcionalidades como **detección de intenciones y entidades, análisis de sentimiento, clasificación de contenido y gráficos de relación**.
- Incluye capacidades tales como **reconocimiento de voz** y un conjunto de herramientas de gestión.
- Trabaja con **slots**, detectando si le falta información necesaria, en cuyo caso preguntará por los campos obligatorios hasta que el usuario complete la información. Sin embargo, no puede modelar que una intención se pueda detectar solo si un contexto determinado no está presente.
- Proporciona técnicas de **expansión automática**. Esto permite comprender la variación en las entidades sin que tenga que introducir una lista completa. Por ejemplo, si tenemos una intención que diga "Me gusta @color" y en la entidad de color, tenemos azul, verde y rojo. Si un usuario dice "Me gusta el marrón", comprenderá que "marrón" es un color.
- Proporciona **webhook**, y se puede decidir qué intentos van a llamar al webhook, esto puede resultar útil para cambiar los contextos y la respuesta del chatbot, porque se podría pasar la información de la intención detectada a un servicio web y obtener un resultado del mismo.
- Proporciona integración con **diferentes plataformas de mensajería** como Facebook Messenger, Slack, Twitter, Telegram, etc. Es particularmente popular dentro de la comunidad Slack.
- Disponible en Google Cloud a través de una **API REST**.
- Plataforma **gratuita**, solo necesita una cuenta Google.
- Proporciona **SDK** para diferentes sistemas operativos y lenguajes de programación: Android, iOS, Cordova, HTML, JavaScript, Node.js, .NET, Unity, Xamarin, C++, Python, Ruby, PHP, Epson Moverio, Botkit y Java.
- Soporta **13 idiomas**: portugués brasileño, chino, inglés, holandés, francés, alemán, italiano, japonés, coreano, portugués, ruso, español y ucraniano.
- Existe gran cantidad de **documentación** y ejemplos de la comunidad de desarrolladores.
- Integración: es posible interactuar con otros sistemas, por ejemplo:
 - Es posible conectarla con *Google Home* creando acciones.
 - También puede integrarse con otros productos de la competencia, como Alexa o Cortana.

2.2.4 Luis (Microsoft)

- Fue **lanzada por Microsoft en 2015**.
- El Servicio⁷ de Inteligencia de Comprensión del Lenguaje de Microsoft (Luis) es un componente de los Servicios Cognitivos de Microsoft (MCS) enfocados en crear y procesar modelos de lenguaje natural. Permite crear aplicaciones inteligentes que puedan comprender el lenguaje humano y reaccionar en consecuencia a las solicitudes de los usuarios.
- Trabaja con **intenciones, entidades** y el desarrollador debe entrenarlo con ejemplos de enunciados para detectar las intenciones.

⁶ <https://api.ai>

⁷ <https://www.luis.ai>

- Proporciona una **interfaz de usuario** para ayudar a los desarrolladores a crear intenciones, entidades, pero no proporciona una representación visual de los flujos de conversación. Tampoco permite administrar los parámetros de contexto a través de la interfaz de usuario.
- Proporciona un conjunto de **modelos de idioma ya preparados** que se pueden usar directamente. Incluye un conjunto de entidades pre-construidas, pero cuyo comportamiento no se puede modificar. Algunas de estas entidades son números de teléfono, emails, urls, enciclopedia, geografía, edad, dinero, temperatura, hora, números, ordinales, dimensiones, edad, etc.
- LUIS también admite las **entidades compuestas**. Se trata de una agrupación de entidades en una sola entidad predefinida. Por ejemplo, en "2 billetes de adultos en primera clase para el vuelo LA hasta NY". 2 es el número de billetes, *adulto* es el tipo de billete, y *primera* es la clase del billete, pero ambas entidades (tipo de billete y clase) forman parte de la entidad compuesta *compra de billetes*.
- LUIS proporciona **aprendizaje activo**, porque examina todas las palabras con las que ha sido entrenado y para las que tenga una mayor incertidumbre y solicita ayuda para que sean etiquetadas.
- Se puede integrar con el servicio de **reconocimiento de voz** de **Microsoft Cognitive Service** para añadir voz a la conversación. Se puede usar junto con otras API de procesamiento de texto en MCS, como el análisis lingüístico y el análisis de texto.
- Soporta **10 idiomas** diferentes: inglés, francés, italiano, alemán, español, portugués, japonés, coreano, chino y holandés.
- Dispone de **cuentas gratuitas con restricciones** y 170 euros de crédito. Cuentas para empresas y cuentas de **pago por uso**.
- Está accesible a través de *Microsoft Azure* mediante una interfaz **API REST**.
- Los **dominios** para crear **bots** son más **restringidos** que en otras plataformas de la competencia, requiere la creación de aplicaciones UWP (*Plataforma Universal de Windows*).
- Se puede **configurar programáticamente**.

2.2.5 Wit.ai (Facebook)

- Fue lanzada en **2013**, y adquirida por **Facebook** en **2015**.
- Wit.ai es la plataforma⁸ detrás de las capacidades de **NLP** y **NLU** del servicio de mensajería Facebook Messenger.
- Proporciona una **interfaz gráfica** de usuario para ayudar a los desarrolladores a crear **intenciones** y **entidades**. También consta de una GUI de desarrollador que incluye una representación visual de los **flujos de conversación**, invocaciones de lógica de negocios, variables de contexto, saltos, y lógica de bifurcación.
- Soporta **entidades jerárquicas**: por ejemplo, en la frase "*de Los Ángeles a Nueva York*", tanto *LA* como *Nueva York* son entidades de ubicación, pero puede distinguir aún más, por ejemplo, entre una ubicación de destino (LA) y una ubicación (Nueva York).
- **No soporta slots**, por lo que debe invocar la lógica de negocio después de cada interacción para recopilar cualquier información faltante que el usuario no haya mencionado.
- Proporciona dominios o **entidades reutilizables** predefinidos.
- Proporciona además un conjunto de **herramientas para monitorizar las interacciones** entre los usuarios y la plataforma.
- Soporta **11 idiomas** actualmente: inglés, francés, alemán, italiano, holandés, español, polaco,

⁸ <https://wit.ai>

sueco, portugués, estonio y ruso, aunque en beta soporta 28 idiomas más.

- Está accesible a través de una **interfaz HTTP REST**.
- **SDKs** para los siguientes lenguajes de programación: Node.js, Python, Ruby, etc.
- Wit.ai es **gratis**, incluso para uso comercial, y sin límite de peticiones.
- La **integración** con otros sistemas de mensajería **no está clara**, todos los ejemplos documentados invocan mensajería de **Facebook Messenger**.

2.2.6 SiriKit (Apple)

- Siri es el **sistema de reconocimiento de voz** de **Apple** que utiliza procesamiento del lenguaje natural para responder preguntas, hacer recomendaciones y realizar acciones mediante la delegación de solicitudes hacia un conjunto de servicios que ha ido aumentando con el tiempo.
- Fue **lanzado** en el **2011**, pero hasta **2016** no se ha **liberado Sirikit**, una interfaz para que la comunidad de desarrolladores pueda integrar algunas de sus funcionalidades en sus aplicaciones. Con anterioridad solo era posible usarla con las aplicaciones que venían instaladas en el sistema.
- La principal diferencia y limitación con respecto al resto de plataformas que estamos evaluando es que SiriKit **no es** una plataforma de procesamiento de lenguaje natural **de propósito general**.
- SiriKit está **restringido a** un conjunto de **dominios** y tareas particulares con flujos de interacción claramente definidos. Los dominios permitidos son: mensajería, llamadas de voz sobre IP, pagos, ejercicios de entrenamiento, búsqueda de fotos, encargar un recorrido (es decir, pedir un Uber, un taxi...), carplay, etc. Algunas de las aplicaciones que utilizamos con frecuencia no están cubiertas, por ejemplo, música, podcasts, estadísticas deportivas, correo electrónico, recordatorios, listas de tareas, etc.
- Dentro de cada dominio hay un conjunto de intenciones que representan las tareas específicas que el usuario puede lograr con Sirikit. Por ejemplo, dentro del dominio de mensajería, existen intenciones de enviar un mensaje, buscar mensajes y establecer atributos en un mensaje. Sirikit **determina la intención** del usuario y decide si una consulta de voz en particular debe manejarse por sí misma o enviarse a una aplicación de un tercero.
- Toda experiencia de SiriKit se compone de cuatro pasos:
 - **Habla:** es traducida a texto para capturar las instrucciones.
 - **Intención:** se genera desde el habla y define qué hemos de realizar.
 - **Acción:** aquello que se envía a través de una extensión a la aplicación que tiene que generar dicha acción.
 - **Respuesta:** aquella que la aplicación devuelve, y que Sirikit nos trasladará como feedback de vuelta a nuestra acción.
- Para hacer todo este proceso posible, Apple ha creado dos nuevos tipos de extensiones para el sistema:
 - **Extensión Intent:** controlador o vía de comunicación entre la aplicación y Sirikit. Es posible tener varias instancias. Implementa los métodos de resolución, respuesta y manejo de todo el proceso.
 - **Extensión Intent UI:** proporciona una experiencia de usuario personalizada en cada una de las respuestas que partan de nuestra aplicación hacia Sirikit.
- Soporta **8 idiomas**: inglés, alemán, francés, japonés, español, italiano, chino y coreano.
- No está claro el coste por uso, parece que se ofrece de forma gratuita para las aplicaciones que estén dentro de los dominios de uso de Sirikit y que acepten la política de uso.

2.2.7 Criterios de selección

2.2.7.1 Rendimiento de la tecnología NLP

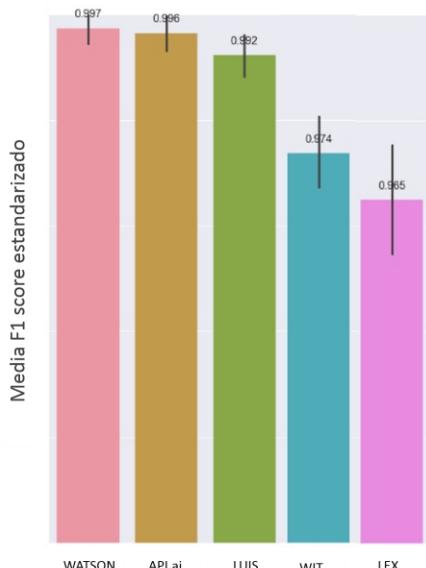
Para la evaluación del rendimiento de las plataformas bajo estudio nos hemos basado en el trabajo realizado en agosto de este año por la compañía Norte Americana *Intento* [4], donde se han utilizado datos que cubren la mayor variedad de casos, y que están disponibles públicamente en Github⁹.

Algunas cuestiones importantes de este estudio son:

- Características del dataset utilizado: 7 intenciones (consultar el tiempo, reservar un restaurante, poner música, añadir canción a playlist y buscar cartelera), 15.600 ejemplos y 340 símbolos.
- Para Amazon Lex ha sido necesario reducir el conjunto de entrenamiento porque la API tiene una limitación de 200.000 símbolos.
- La evaluación se ha realizado en un idioma: inglés.

Las métricas de rendimiento (predicción de intenciones, falsos positivos, tiempo de aprendizaje y tiempo de respuesta) obtenidas para las plataformas evaluadas han sido:

Predicción de intenciones



*Las barras negras indican el intervalo de confianza

Figura 4 - Precisión de detección

La plataforma que presenta mayor precisión es Watson, seguida muy de cerca de API.ai y LUIS. Si enfrentamos precisión y sensibilidad obtenemos la siguiente gráfica:

⁹ <https://github.com/snipsco/nlu-benchmark>

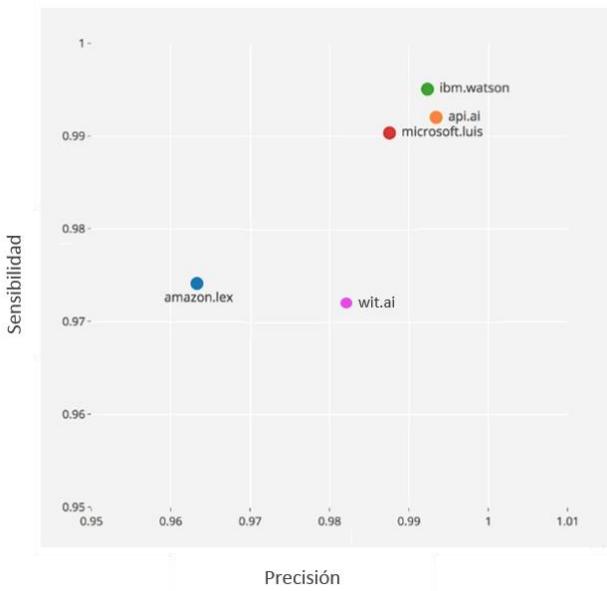


Figura 5 - Precisión vs Sensibilidad

Que muestra que nuevamente Watson obtiene mejor relación precisión / sensibilidad, seguido muy de cerca de API.ai y LUIS. En consecuencia, desde el punto de vista de la precisión en la detección de intenciones, las mejores plataformas son IBM Watson, API.ai y Microsoft LUIS.

Hay intenciones, como por ejemplo la del pronóstico meteorológico (getWeather), para la que todas las plataformas funcionan bien, pero para las intenciones más complicadas de detectar tener un buen producto es importante.

Falsos positivos

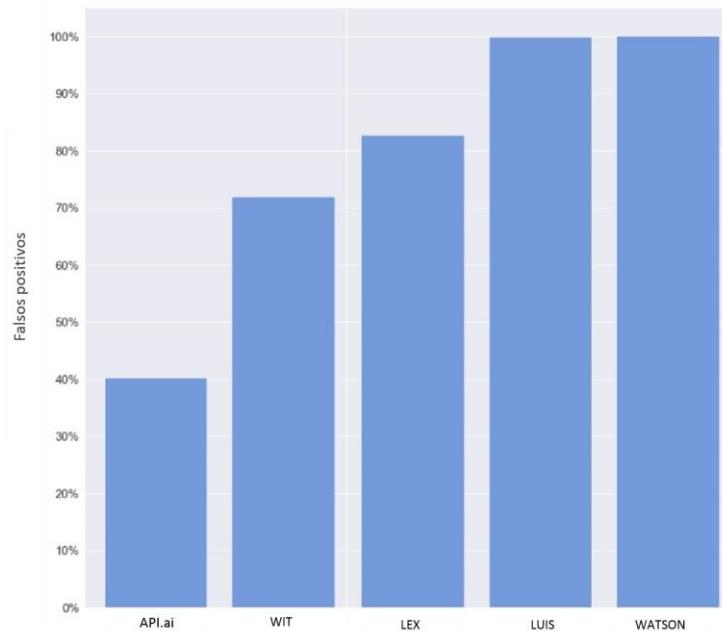


Figura 6 - Porcentaje de falsos positivos

Sólo API.ai es bueno detectando las peticiones del usuario para las que no ha sido entrenado. IBM Watson y Microsoft LUIS intentan mapear la petición de usuario con alguna de las intenciones del conjunto de datos con el que han sido entrenados.

Velocidad de aprendizaje

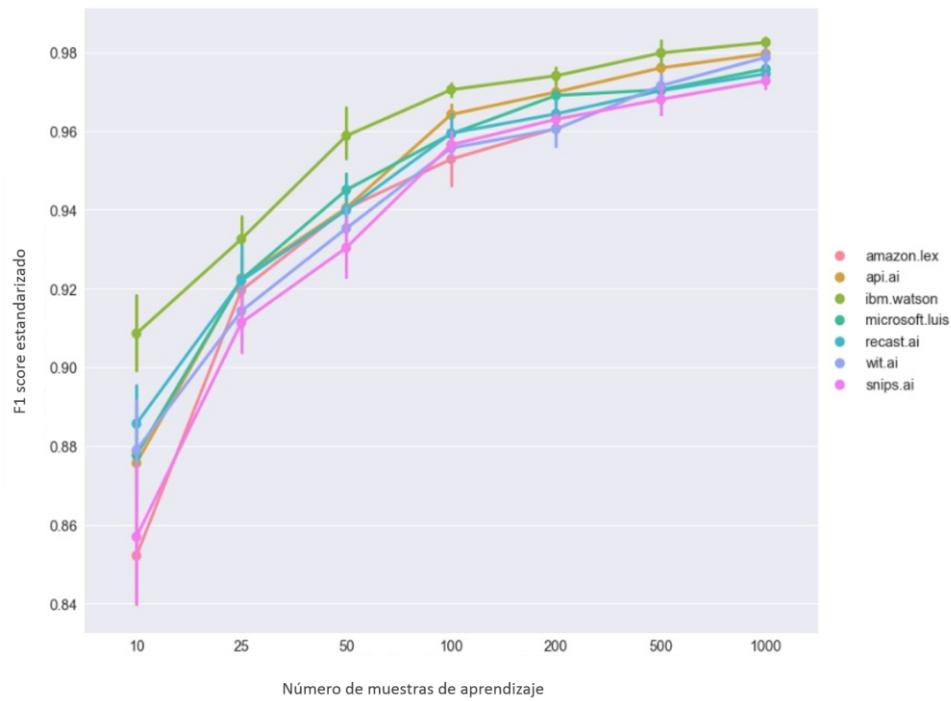


Figura 7 - Velocidad de aprendizaje para cada plataforma

Observamos que la plataforma que presenta mayor velocidad de aprendizaje es Watson. Especialmente para muestras de entrenamiento menores a 100 locuciones, donde IBM Watson es bastante superior al resto de plataformas. Para muestras mayores que 100 locuciones API.ai se aproxima mucho a Watson.

Tiempo de respuesta

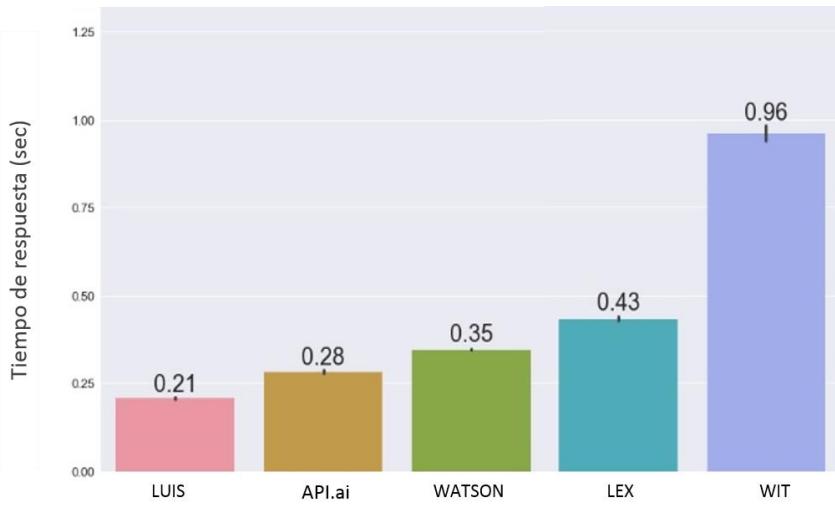


Figura 8 - Tiempo medio de respuesta (segundos)

En este caso, LUIS presenta la mejor velocidad de respuesta, está seguido muy de cerca por API.ai y Watson. Wit.ai sin embargo muestra el peor comportamiento, ya que su tiempo de respuesta es 4 veces mayor que el de LUIS o API.ai.

Veamos a continuación cómo se comporta cada plataforma al enfrentar tiempo de respuesta y precisión.



Figura 9 - Rendimiento vs velocidad de respuesta

2.2.7.2 Cobertura de idiomas

- **Watson (IBM)**: soporta **9** idiomas diferentes (inglés, español, portugués, coreano, japonés, italiano, alemán, francés, árabe) y **3** idiomas **experimentales** (chino, checo y holandés).
- **Lex (Amazon)**: sólo disponible en **1 único idioma**, el inglés estadounidense.
- **Api.ai (Google)**: soporta **12** idiomas (portugués, chino, inglés, holandés, francés, alemán, italiano, japonés, coreano, ruso, español y ucraniano). También soporta idiomas locales como el chino cantonés, francés canadiense, inglés US, inglés UK, inglés canadiense, inglés australiano o portugués brasileño.
- **Luis (Microsoft)**: soporta **10** idiomas (inglés, francés, italiano, alemán, español, portugués, japonés, coreano, chino y holandés).
- **Wit.ai (Facebook)**: soporta **11** idiomas actualmente (inglés, francés, alemán, italiano, holandés, español, polaco, sueco, portugués, estonio y ruso) aunque en **beta** soporta otros **28** idiomas más.
- **Sirikit (Apple)**: soporta **8** idiomas (inglés, alemán, francés, japonés, español, italiano, chino y coreano)

Salvo excepciones como **Lex de Amazon**, que presenta **serias limitaciones** en cuanto a soporte de idiomas (sólo está disponible para inglés), el resto de plataformas soportan todos los idiomas mayoritarios. **Destacan** especialmente por su cobertura **Wit.ai y Api.ai**. Sería necesario profundizar en el estudio de la cobertura lingüística real de cada idioma, puesto que para inglés todas parecen tener su óptimo de rendimiento, mientras que para el resto de idiomas no está tan experimentado y a medida que profundizamos en su uso encontramos limitaciones funcionales o una **disminución del rendimiento con respecto al proporcionado** para inglés.

2.2.7.3 Precio del servicio

- **Watson (IBM)**: Dispone de cuentas de prueba gratuitas, cuentas de **pago por uso** y cuentas Premium.
 - Las **cuentas gratuitas** tienen restricciones funcionales (limitado a 25 intenciones y 25 entidades) y de uso (límite de 10.000 llamadas a la API al mes). Por defecto tienen una duración de un mes, pero para la comunidad universitaria o para los desarrolladores de hackathones o de sus programas de marketing tienen una duración que oscila entre 6 y 12 meses.
 - **Pago por uso**: el servicio **Convesation** tiene un coste de 0.0025 USD por cada llamada a la API, sin límite de llamadas. Si se añade reconocimiento y síntesis de **voz** a la aplicación, el precio se eleva porque hay que añadir los costes del uso de los servicios **Speech To Text**

(1.000 minutos gratis al mes y a partir de ahí los precios oscilan entre 0.02 USD y 0.01 USD por minuto según tramos de consumo) y **Text to Speech** (1 millón de caracteres gratis al mes, a partir de ahí la tarifa es de 0.02 USD por cada 1.000 caracteres que se solicite sintetizar).

- **Lex (Amazon)**: dispone de cuentas de prueba y cuentas de **pago por uso** (cantidad de solicitudes de voz y texto procesadas). El precio por solicitud de **voz** es de 0,004 USD y el precio de cada solicitud de **texto** es de 0,00075 USD.
- **Api.ai (Google)**: gratuito.
- **Luis (Microsoft)**: dispone de cuentas gratuitas con restricciones y 170 euros de crédito. Cuentas para empresas, y cuentas de **pago por uso** con un coste de 0.000633 USD por llamada a la API, siendo las 10.000 primeras solicitudes del mes gratuitas.
- **Wit.ai (Facebook)**: gratuito.
- **Sirikit (Apple)**: No está claro el coste por uso, parece que se ofrece de forma gratuita para las aplicaciones que estén dentro de los dominios de uso.

Es difícil estimar una cifra final con la que comparar para las plataformas de pago por uso, ya que cada una de ellas tiene una tarificación distinta y en la arquitectura de la solución final puede que sea necesario añadir costes adicionales como el de añadir o no voz.

El número de llamadas a estas APIs dependerá en gran medida del número de usuarios, y del tipo de peticiones que estos realicen.

Además, no solo hay que contar el consumo una vez la aplicación está en producción, sino que en el proceso de construcción, entrenamiento y pruebas antes de lanzarla también será necesario realizar peticiones a la API y esto conlleva un coste asociado.

Con respecto a este criterio de selección parece que la plataforma que destaca es Api.ai en relación calidad precio, puesto que ofrece uno de los mejores rendimientos y además es gratuita.



Figura 10 - Precio (USD) por cada 1K peticiones de texto



Figura 11 - Rendimiento vs precio

2.2.7.4 Fortaleza de la compañía proveedora

Todas las compañías proveedoras que estamos evaluando son gigantes tecnológicos, fuertes y consolidados. Rivales que compiten por el dominio tecnológico, el mercado y los clientes. Cada uno de ellos compite con su estrategia y con campañas de marketing en las que han usado la inteligencia artificial para hacer sus sistemas publicitarios más efectivos.

IBM ganó notoriedad por haber salido victorioso con Watson frente a 2 humanos en el concurso **Jeopardy**, pero en los años siguientes, el espacio de inteligencia artificial, y más concretamente el procesado de lenguaje natural, ha visto entradas importantes de compañías como **Google** y **Amazon**, que tienen una **gran cantidad de datos** de usuarios y clientes para alimentar sus sistemas.

El modelo de negocio que se está construyendo alrededor del procesado de lenguaje natural y reconocimiento vocal pasa por ofrecer este como un **servicio en la nube**, y actualmente la nube está **dominada por Amazon Web Services y Microsoft Azure**.

2.2.7.5 Existencia de módulos predefinidos

Evaluaremos en esta sección si la plataforma dispone de módulos genéricos o de uso común ya precargados en el sistema para que el desarrollador pueda iniciar su proyecto con más rapidez.

- **Watson (IBM)**: no dispone de intenciones pre-construidas, lo más parecido son las variables de sistema definidas para manejar, por ejemplo, monedas, fechas, horas, localizaciones, número, porcentajes, etc., y en beta aún para persona¹⁰. También permite importar intenciones o entidades en formato CSV, lo que se puede aprovechar para incorporar intenciones genéricas como las de saludar, despedirse, etc.
- **Lex (Amazon)**: La librería *Alexa Skill Kit* proporciona intenciones pre-construidas para acciones generales, como detener, cancelar y pedir ayuda. También existen entidades ya construidas para los datos más comunes como emails, porcentajes, números de teléfono, unidades de medida, fechas, horas, etc.¹¹
- **Api.ai (Google)**: proporciona entidades predefinidas para fechas, horas, números, colores, etc. con el objetivo¹².
- **Luis (Microsoft)**: Proporciona una amplia gama¹³ de entidades pre-construidas disponibles en la mayoría de idiomas. También incorpora dominios predefinidos que proporcionan entidades e

¹⁰ <https://console.bluemix.net/docs/services/conversation/system-entities.html#system-entity-details>

¹¹ <http://docs.aws.amazon.com/lex/latest/dg/howitworks-builtins.html>

¹² <https://dialogflow.com/docs/entities#system-mapping>

¹³ <https://docs.microsoft.com/en-us/azure/cognitive-services/luis/pre-builtentities>

intenciones, algunos de estos dominios son calendario, cámara, comunicación, música, etc. Además, también permite utilizar las intenciones y entidades de Cortana en nuestra aplicación.

- **Wit.ai (Facebook):** tiene modelos pre-construidos que pueden reconocer entidades e intenciones, como lugares, correos electrónicos, calendarios, etc.
- **Sirikit (Apple):** no se han encontrado elementos predefinidos, pero al tratarse de una plataforma con dominios más restringidos el usuario no tendrá que proporcionar tantos ejemplos a la hora de entrenar para que la aplicación comience a funcionar con las intenciones específicas.

2.2.7.6 Control de versiones

El control de versiones es importante y más aún cuando se trabaja en equipo. Resulta imprescindible tener archivados los cambios que se realizan sobre nuestro chatbot, así como las intenciones, entidades y el flujo de la conversación que hemos definido para él, con un histórico de los cambios que se han realizado a lo largo del tiempo para poder recuperar versiones específicas más adelante.

Tanto **AWS Lex¹⁴** como **Luis¹⁵** incorporan control de versiones, en el resto de plataformas no hay disponible mecanismos de este tipo, la única opción pasa por llevar el control de forma manual exportando los espacios de trabajo y salvando las distintas versiones que se van generando. Tanto en Wit.ai como en Watson se puede exportar a formato JSON. Este JSON contiene las intenciones, las entidades y el flujo de la conversación que se ha definido para nuestra aplicación.

2.2.7.7 Flexibilidad, autonomía y accesibilidad

Salvo **Sirikit** que está **restringida** a los **dominios** que se enumeraron en el apartado 2.2.6, el resto de plataformas NLP evaluadas trabajan de forma similar, proporcionando flexibilidad para la construcción de chatbots en cualquier campo o ámbito de uso. Permiten usar intenciones y entidades predefinidas, así como crear otras específicas que se adapten mejor al caso de uso para el que el chatbot sea destinado.

2.2.7.8 Integración

Presentan **limitaciones** en este aspecto **Sirikit**, que sólo se integra con sistemas iOS, **Luis**, que necesita la creación de aplicaciones UWP (Plataforma Universal de Windows), y **Watson**, que no parece integrarse con facilidad en otros sistemas de mensajería diferentes a Facebook.

2.2.7.9 Reconocimiento y síntesis de voz

Muchos chats tienen una función de mensaje de voz. La razón es obvia, escribir con el teclado a veces puede resultar tedioso y es más fácil y rápido expresarse usando la voz. Sin embargo, hay ocasiones en que las circunstancias te obligan a guardar silencio y otros en los que necesitas realizar una búsqueda en tu historial de chat para recuperar información que no puedes encontrar en los mensajes de voz. Por estas razones nuestro chatbot debe poder comunicarse tanto en texto como en voz.

Veamos en más detalle cómo se adapta a este requisito las plataformas de NLP que estamos estudiando:

- **Watson (IBM):** combinado con los servicios **Watson Speech to Text** y **Tex to Speech**, es posible incorporar voz en las conversaciones de nuestro chatbot. Speech to Text transcribe la voz a texto, y Text to Speech sintetiza la voz a partir de un texto como entrada. El servicio Conversation recibe texto como entrada y devuelve texto.
- **Lex (Amazon):** Amazon Lex comprende tanto texto como voz, pero si la respuesta de nuestro chatbot es también usando voz, será necesario utilizar el servicio **Polly** de Amazon, que transforma texto en voz.

¹⁴ <http://docs.aws.amazon.com/lex/latest/dg/versioning-aliases.html>

¹⁵ <https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-how-to-manage-versions>

- **Api.ai (Google)**: El servicio de reconocimiento de voz de API.ai está en desuso, para incorporar voz a la conversación hay que incorporar servicios para reconocimiento y síntesis de voz. Se puede usar la **API Speech de Google Cloud**, que convierte audio en texto aplicando potentes modelos de redes neuronales en una API fácil de usar. La API reconoce más de 80 idiomas y variantes. No se ha encontrado ninguna API para síntesis de voz en los servicios de la nube de Google.
- **Luis (Microsoft)**: Se puede integrar con el servicio **Bing Speech API** de los servicios cognitivos de Microsoft para añadir voz a la conversación. Este servicio convierte voz en texto y viceversa.
- **Wit.ai (Facebook)**: Wit.ai entiende tanto texto como voz. Admite voz en tiempo real y audios. No se ha encontrado en la documentación si dispone de métodos en su API para sintetizar voz.
- **Sirikit (Apple)**: comprende tanto texto como voz, proporciona un método llamado **AVSpeechSynthesizer** para sintetizar voz.

2.2.8 Conclusiones

Los chatbots son una tendencia ascendente. Cuando son de calidad pueden aumentar el compromiso con los usuarios, brindar una mejor experiencia y ahorrar costes. Sin embargo, hacerlo bien no es trivial.

Actualmente, hay una gran cantidad de plataformas que pueden ayudar en la creación de chatbots. Algunas de estas plataformas, como las que se mencionan en el apartado 2.1, han surgido para beneficiarse del boom momentáneo del mercado, algunos son muy básicos y cubren casos de uso demasiado específicos, por lo que difícilmente pueden ser escalables. Son buenas soluciones para experimentar con la experiencia del usuario final, pero no para entornos productivos.

Para la construcción de chatbots ambiciosos, capaces de manejar conversaciones complejas y realizar acciones, son más apropiadas, por su estabilidad, rendimiento y flexibilidad, plataformas como las mencionadas en el apartado 2.2, que proveen servicios para incorporar las técnicas de procesado de lenguaje natural en nuestro chatbot.

Watson tiene el mejor rendimiento de detección de intenciones, especialmente en conjuntos de datos de entrenamiento más pequeños. Como desventaja, su precio es el más elevado de todas las plataformas evaluadas en este estudio. Wit.ai es gratuito y tiene la mejor cobertura de idioma, pero presenta un rendimiento y tiempo de respuesta peor que los de sus competidores. API.ai también es gratuito y tiene uno de los mejores resultados tanto en rendimiento como en tiempo de respuesta. LUIS también tiene un rendimiento de alto nivel, y un tiempo de respuesta menor que otros con los que ha sido comparado. Lex tiene buen rendimiento, pero solo está disponibles en inglés. Sirikit está restringido sólo a ciertos dominios de uso y no aporta flexibilidad en la construcción de chatbots.

2.3 IBM Watson

2.3.1 Introducción

Como se ha comentado en el apartado anterior, Watson es una tecnología de computación cognitiva de IBM. Lee y entiende el lenguaje natural de las personas. Es capaz de responder a preguntas complejas en pocos segundos a partir de su capacidad para analizar grandes volúmenes de datos, tanto estructurados (bases de datos) como no estructurados, incluyendo texto (como correos electrónicos, blogs, redes sociales, etc.), imágenes y voz. Ante una pregunta, formula hipótesis y escoge la respuesta en la que tiene un mayor nivel de confianza. IBM Watson se está utilizando en diferentes sectores como son la sanidad, finanzas o educación.

El sistema IBM Watson fue presentado en enero de 2011 en el concurso de televisión *Jeopardy!*, en el que ganó a los dos mejores concursantes de la historia de este programa, siendo capaz de responder a preguntas formuladas en lenguaje natural.

Inicialmente, IBM creó Watson como un gran sistema único de inteligencia artificial, pero ante la falta de resultados de negocio (o la lentitud con la que éstos se obtendrían) y las altas inversiones que requería, decidió en 2015 trocearla en una variedad de componentes para usos concretos. Esto dio lugar a los siguientes productos y servicios:

- **Watson Knowledge Studio:** permite a los desarrolladores y expertos en la materia colaborar y crear componentes de anotadores personalizados para sectores concretos. Estos anotadores pueden identificar menciones y relaciones en datos no estructurados. Los componentes de los anotadores se pueden desplegar directamente en *IBM Watson Explorer* y en *Discovery*, de *Watson Developer Cloud*.
- **Watson Explorer:** es la plataforma de análisis de contenido y búsqueda cognitiva. Permite el rastreo, ingestión, indexación, y búsqueda de grandes conjuntos de datos estructurados y no estructurados. También facilita el análisis de dichos datos y la creación de aplicaciones de exploración.
- **Agente virtual Watson:** es el asistente conversacional pre-entrenado y construido para diferentes sectores. Es necesario configurarlo para que se adapte a las necesidades específicas del entorno donde se va a desplegar.
- **Watson Analytics:** facilita la exploración y el análisis de datos mediante un sistema de visualización y predicción analítica. No es una herramienta de recolección de datos, proporciona diferentes opciones para subir, o extraer de base de datos, la información que queremos analizar. Permite refinar la información antes de analizarla, por ejemplo, agrupando, jerarquizando, renombrando, etc. La herramienta sugiere las visualizaciones que mejor se adapten a nuestros datos de entre todas las disponibles.

IBM también ha creado APIs para Watson, de modo que se puedan desarrollar soluciones y servicios especializados sobre la plataforma. Estas APIs están disponibles a través del *Watson Developer Cloud*, que es la gama de servicios cognitivos de *Bluemix*.

Bluemix es la plataforma cloud de IBM que proporciona acceso a desarrolladores móviles y web al software y servicios de IBM a través de APIs. A diferencia de los productos de Watson mencionados con anterioridad (agente virtual, analytics, etc.), los servicios de Watson disponibles en *Bluemix* no están pre-construidos, necesitan ser entrenados introduciendo la respuesta a multitud de posibles situaciones.

Lo que ofrece Watson *Bluemix* son APIs, por tanto, siempre será necesaria una aplicación o entorno que orqueste dichos servicios. *Bluemix* proporciona SDKs para iOS, Android, Java, Node.js, Python, Scala, Unity, Ruby, la comunidad de desarrolladores también ha creado SDKs para Go, Scala, R, PHP y Genexus. Se puede usar cualquier lenguaje como orquestador, porque al final se trata de una llamada a un servicio HTTP REST. Las SDKs facilitan el trabajo de los desarrolladores, pero tienen la desventaja de que incorporan más lentamente las nuevas funcionalidades que van siendo incorporadas a la API.

El entorno de IBM *Bluemix* cuenta con un amplio catálogo de servicios, no sólo los de Watson. Podemos encontrar servicios para persistencia, seguridad, devOps, internet de las cosas, infraestructura de red en la nube, análisis de datos, servidores de aplicación, etc.



Figura 12 - Ecosistema de servicios Bluemix

2.3.2 Watson Developer Cloud (Bluemix)

Los servicios que ofrece se pueden agrupar como sigue:

- **Conocimiento y descubrimiento:** *Natural Language Understanding* y *Discovery*.
- **Visión:** *Visual recognition*.
- **Empatía:** *Personality insight* y *Tone Analyzer*.
- **Lenguaje:** *Language translator* y *Natural Language Classifier*.
- **Voz:** *Speech to Text* y *Text to Speech*.

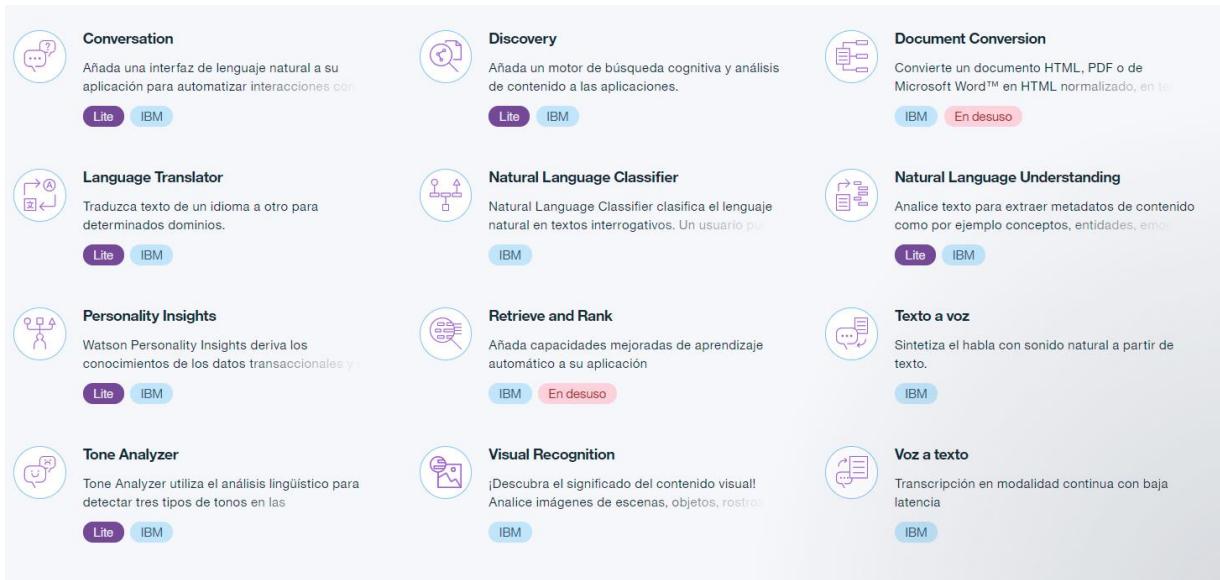


Figura 13 - Catálogo de servicios IBM Watson Bluemix

Además de los servicios mencionados en este apartado también existen servicios experimentales, en versión beta disponible para ser usados¹⁶.

Podemos ampliar la información de cada uno de los servicios que se ha mencionado consultando las siguientes fuentes:

¹⁶ <https://console.bluemix.net/catalog/labs/>

- Documentación de la web: API reference, demo, etc.
- Comunidad de desarrolladores de Watson¹⁷.
- Blogs de Watson¹⁸.
- Starter kits: ejemplos de código (demos, github)¹⁹.
- IBM developerWorks Cognitive Area²⁰.
- Learn cognitive: Courses and learning path .
- Architectures references.
- Watson console²¹.
- IBM Cloud Garage Method - Architecture Center²².
- Github²³.
- Academy²⁴.
- Canal de youtube²⁵.
- Redbooks²⁶.
- Releases notes de cada versión²⁷.

A continuación presentaremos un breve un resumen de cada uno de los servicios listados anteriormente, haciendo más énfasis en aquellos que se utilizarán en este trabajo: *Conversation* y *Visual Recognition*.

2.3.2.1 Servicio Natural Language Understanding

- Realiza análisis de texto a través del procesamiento del lenguaje natural para extraer significado del contenido. Extrae **conceptos**, **entidades**, **palabras clave**, **sentimiento** (positivo/negativo), **categorías**, **conceptos**, **roles**, etc.
- Los datos para ser analizados se pueden proporcionar en formato HTML o en texto plano.
- El tipo de contenido que se desea extraer del texto se especifica mediante los denominados “**enrichments**” en la llamada al servicio. Hay enrichments como el de la emoción que sólo está disponible en inglés.
- La tarificación de este servicio también tiene en cuenta los enrichments consultados.
- Con *Watson Knowledge Studio* se pueden crear modelos de anotación personalizados para obtener resultados adaptados a un dominio específico.

One year ago, several hours before cities across the United States started their annual fireworks displays, a different type of fireworks were set off at the European Center for Nuclear Research (CERN) in Switzerland. At 9:00 a.m., physicists announced to the world that they had found something they had been searching for for nearly 50 years: the elusive Higgs boson. Today, on the anniversary of its discovery, are we any closer to figuring out what that particle's true identity is? The Higgs boson is popularly referred to as "the God particle," perhaps because of its role in giving other particles their mass. However, it's not the boson itself that gives mass. Back in 1964, Peter Higgs proposed a theory that described a universal field (similar to an electric or a magnetic field) that particles interacted with.

ENTITY	RELEVANCE	SENTIMENT	TYPE	LINKED DATA
Peter Higgs	0.98893	positive	Person	DBpedia Yago
European Center for Nuclear Research	0.69407	neutral	Organization	Website Lat:46.23,Lon:6.06 DBpedia Yago OpenCyc GeoNames
United States	0.461032	neutral	Country	Website DBpedia Yago OpenCyc CIA Factbook
Switzerland	0.445847	neutral	Country	Website Lat:46.83,Lon:8.33 DBpedia Yago OpenCyc CIA Factbook

¹⁷ <https://developer.ibm.com/watson/>

¹⁸ <https://www.ibm.com/blogs/watson/>

¹⁹ <https://www.ibm.com/watson/developercloud/starter-kits.html>

²⁰ <https://www.ibm.com/developerworks/learn/cognitive/index.html>

²¹ <https://console.bluemix.net/developer/watson/dashboard>

²² <https://www.ibm.com/devops/method/category/architectures>

²³ <https://github.com/watson-developer-cloud>

²⁴ <https://www.watson-academy.info/>

²⁵ <https://www.youtube.com/user/IBMWatsonSolutions>

²⁶ <http://www.redbooks.ibm.com/Redbooks.nsf/domains/watson>

Figura 14 - Ejemplo de extracción de entidades

Sentence				
Yesterday [dumb Bob] [destroyed] [my fancy iPhone] [in beautiful Denver, Colorado].				
Parsing	Sentiment Analysis			
Subject: [dumb Bob]	Relational (subject): dumb			
Action: [destroyed]	Relational (object): fancy			
Object: [my fancy iPhone]	Directional: destroyed			
Location: [in beautiful Denver, Colorado]	Relational (location): beautiful			
Entity Extraction				
ENTITY	PART	TYPE	SUBTYPES	LINKED DATA
Bob	subject	Person		
iPhone	object	Technology		
Denver	location	City	Location, AdministrativeDivision, PlaceWithNeighborhoods, USCounty, CityTown	Website DBpedia
Colorado	location	StateOrCountry		

Figura 15 - Ejemplo de extracción de roles (relación “sujeto-acción-objeto”)



Figura 16 - Ejemplo de extracción de relación entre entidades

2.3.2.2 Discovery

- Motor de búsqueda cognitiva y análisis de contenido. Identifica patrones, tendencias y otros conocimientos útiles a partir de datos estructurados y no estructurados (el formato de datos admitido es PDF, HTML, JSON y Microsoft Word).
- Los documentos se incorporan al sistema mediante data crawling, haciendo uso de la interfaz gráfica o programáticamente. IBM no proporciona un servicio propio de data crawling, por lo que hay que integrarlo con algún otro producto del mercado que realice esta función.
- También proporciona funciones para conversión de documentos, por ejemplo, de PDF a texto, etc.
- Extrae palabras clave, e indexa por rasgos cognitivos (palabras clave, conceptos, jerarquías, sentimiento, relación, etc.).
- Permite realizar consultas en lenguaje natural.
- Un típico caso de uso es la búsqueda corporativa.
- En este servicio se realiza persistencia, por lo que también tendrá asociado un coste por almacenamiento.
- *Watson NLU* está integrado en *Discovery*, por lo que la principal diferencia entre ambos es que *Discovery* es una solución integral que incluye crawling, conversión de documentos, ingestión, almacenamiento de datos y un lenguaje simplificado de consultas.
- *Watson Discovery* incorpora también *Watson Discovery News*, que es un conjunto de fuentes de noticias en inglés que se actualiza diariamente.



Figura 17 - Flujo de trabajo con servicio Discovery

La siguiente tabla resume y compara ambos servicios:

Criterio	Watson Discovery	Watson NLU
Descripción	Servicio de extremo a extremo Ingiere datos, enriquece y analiza contenido	Servicio de análisis de texto Extracción de contenido
Almacenamiento	Almacenamiento en la nube	Sin persistencia
Formato de datos	PDF, HTML, JSON y Microsoft Word	HTML y texto plano
Fuente de datos	Cualquier fuente (archivo, base de datos, SharePoint, CMIS, etc.) Integración con rastreadores datos web	URL archivo de texto o texto plano
Interfaces	API y web tooling	API
Precio	Almacenamiento, número de enrichments, Queries y modelos personalizados	Volumen de datos, número de enrichments y modelos personalizados

Figura 18 - Comparativa Watson Discovery y Watson NLU

2.3.2.3 Servicio Visual recognition

- El servicio de reconocimiento visual utiliza algoritmos de aprendizaje profundo para identificar escenas, objetos, caras y otros contenidos en las imágenes.
- La API ofrece tres capas distintas:
 - **Capacidades listas para usar:** etiquetado general y reconocimiento facial.
 - **Modelos pre-construidos:** modelo para detectar comida y color.
 - **Modelos personalizados:** pueden ser creados y entrenados para reconocer contenido específico.
- También funciona como clasificador, porque podemos entrenar el servicio en base a colecciones de imágenes positivas (cumplen la condición que se desea clasificar) y negativas (no cumplen la condición), y de este modo, al introducir una nueva clase, encuentra los patrones comunes y lo clasifica en función de la similaridad.
- En el reconocimiento facial, además, si la persona es una celebridad puede etiquetarlo y ofrecer información más detallada.

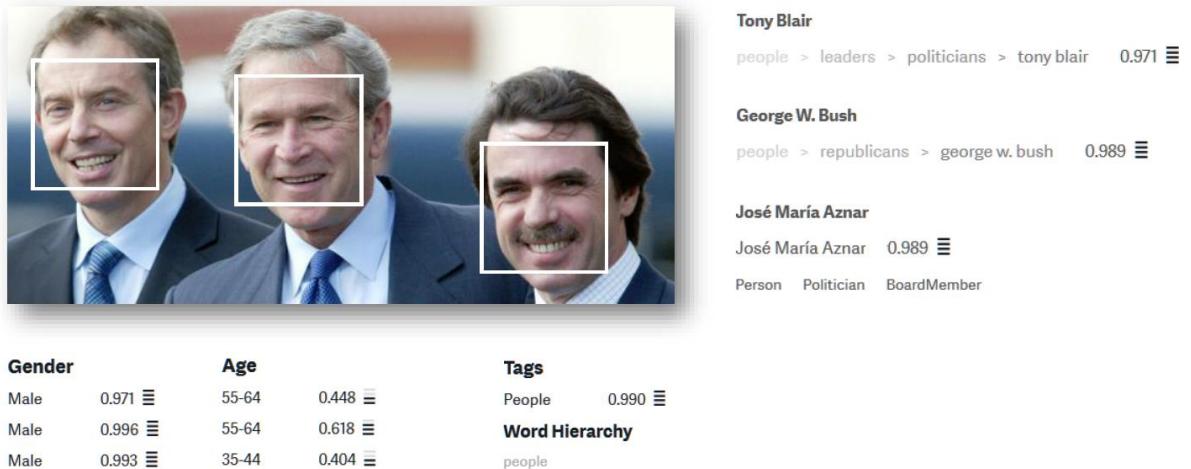


Figura 19 - Ejemplo reconocimiento facial realizado con el servicio Visual Recognition

2.3.2.4 Personality Insights

- Predice las características, necesidades y valores de la personalidad a través del texto escrito.
- El servicio analiza el contenido que recibe y devuelve un perfil de personalidad para el autor de la entrada.
- La clasificación de perfiles en *IBM Watson Personality Insights* se basa en varios estudios y modelos en campos como la psicología, la psicolingüística y el marketing. Infiere características de personalidad basadas en tres modelos principales:
 - **Big Five:** representa el modelo más utilizado para describir cómo una persona se relaciona con el mundo. El modelo incluye cinco dimensiones principales; amabilidad, conciencia, extraversión, rango emocional y apertura. Cada dimensión tiene seis facetas que caracterizan aún más a un individuo según la dimensión.
 - **Necesidades:** describen qué aspectos de un producto se asemejan a la persona. El modelo incluye doce necesidades características: entusiasmo, armonía, curiosidad, aspiración, cercanía, autoexpresión, libertad, amor, practicidad, estabilidad, desafío y estructura.
 - **Valores:** describen los factores motivadores que influyen en la toma de decisiones de una persona. El modelo incluye cinco valores: autotranscendencia / ayudar a los demás, conservación / tradición, hedonismo / disfrutar de la vida, autorealimentación / lograr el éxito y abierto al cambio / emoción.
- Actualmente soporta los siguientes idiomas de entrada: árabe, inglés, japonés, coreano y español.

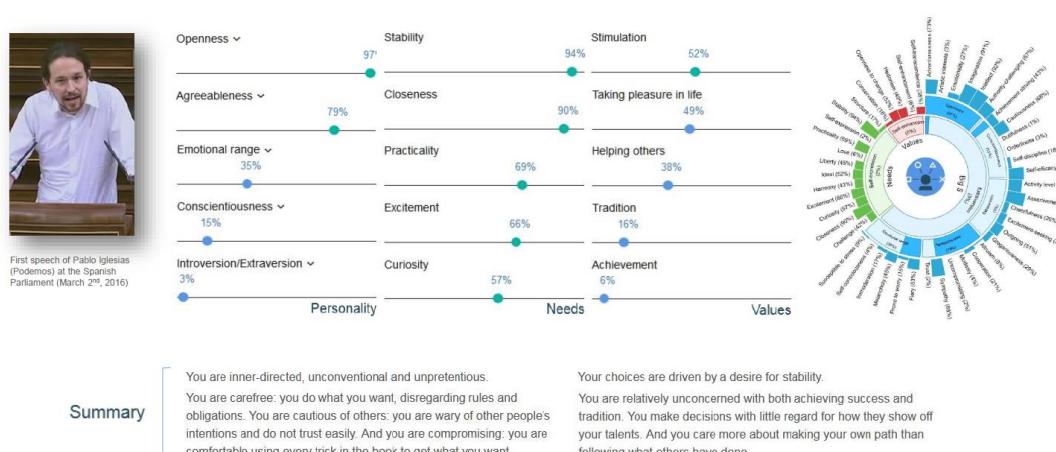


Figura 20 - Identificación rasgos psicológicos realizado con el servicio Personality Insights

2.3.2.5 Tone Analyzer

- El servicio *Tone Analyzer* utiliza análisis lingüístico para detectar tonos emocionales y de lenguaje en textos escritos.
- Puede analizar el tono en los niveles de documento y oración.
- Detecta tres tipos de tonos:
 - **Emoción:** ira, asco, miedo, alegría y tristeza.
 - **Actitudes sociales:** franqueza, diligencia, extroversión, simpatía y rango emocional.
 - **Estilos de lenguaje:** analítico, seguro y vacilante.
- Acepta el formato de entrada JSON, texto plano o HTML.
- Acepta hasta 128 KB de texto (equivale aproximadamente a 1000 oraciones).
- El servicio devuelve resultados JSON que informan el tono de su entrada.
- Actualmente soporta sólo inglés y francés como idiomas de entrada.
- Algunos casos de uso típicos son análisis de sentimiento en redes sociales, marketing personalizado, respuestas personalizadas en chatbots, evaluación del compromiso de los clientes, etc.

Hi Team,

The times are difficult! Our sales have been disappointing for the past three quarters for our data analytics product suite. We have a competitive data analytics product suite in the industry. However, we are not doing a good job at selling it, and this is really frustrating. We are missing critical sales opportunities. We cannot blame the economy for our lack of execution. Our clients are hungry for analytical tools to improve their business outcomes. In fact, it is in times such as this, our clients want to get the insights they need to turn their businesses around. It is disheartening to see that we are failing at closing deals, in such a hungry market. Let's buckle up and execute.

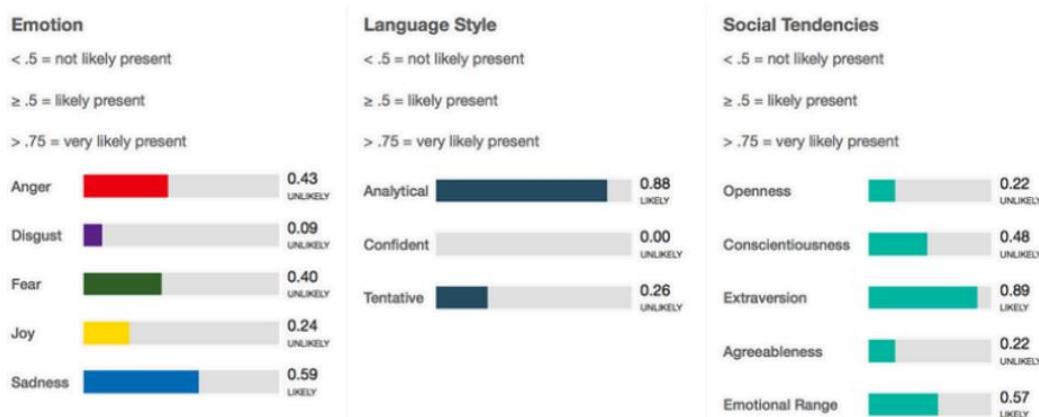


Figura 21 - Ejemplo servicio Tone Analyzer

2.3.2.6 Language translator

- Identifica 62 idiomas diferentes a partir del texto que recibe como entrada. Por ejemplo, si se introduce una palabra en inglés, como por ejemplo “Hello”, el servicio puede identificarlo como inglés.
- Convierte la entrada de texto en un idioma a otro idioma de destino utilizando los modelos específicos del dominio. Por ejemplo, traduce el saludo en inglés “Hello” a su equivalente en español, “Hola”.
- Evalúa la calidad de la traducción usando el estándar BLEU (*BiLingual Evaluation Understudy*).

- También permite traducir terminología especializada, es posible personalizar el modelo lingüístico para optimizarlo según nuestras necesidades, agregando un glosario de términos personalizado, o corpus, en un idioma de origen y destino.
- Los modelos lingüísticos que se proporcionan con el servicio pueden realizar traducciones entre los siguientes dominios e idiomas:
 - **Noticias:** dirigido a artículos de noticias y transcripciones. Traduce de inglés a árabe, portugués, francés, alemán, italiano, japonés, coreano y español, y a la inversa. También puede traducir el español al francés y viceversa.
 - **Conversacional:** dirigido a coloquialismos conversacionales. Traduce de inglés a árabe, portugués, francés, italiano y español y viceversa.
 - **Patentes:** dirigido a la terminología técnica y legal. Traduce el portugués, chino, coreano y español al inglés.

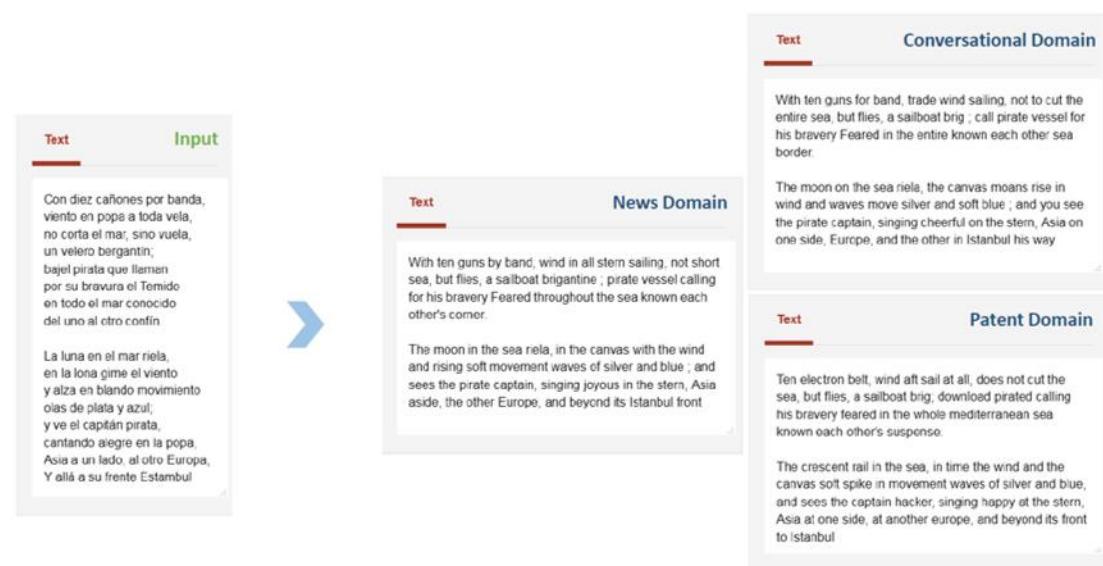


Figura 22 - Ejemplo Language Translator: La canción del pirata (José de Espronceda)

2.3.2.7 Natural Language Classifier

- *Natural Language Classifier* aplica técnicas de aprendizaje profundo para hacer predicciones sobre las mejores clases predefinidas para oraciones cortas o frases.
- Se crea una instancia de clasificador al proporcionar un conjunto de series representativas y un conjunto de una o varias clases correctas para cada uno como parte del entrenamiento.
- Despues del entrenamiento, el nuevo clasificador puede aceptar nuevas preguntas o frases y devolver los resultados con mayor coincidencia con un valor de probabilidad para cada coincidencia.
- Los datos de entrenamiento se pueden proporcionar en formato CSV, donde una fila representa un registro del ejemplo. Cada registro tiene dos o más columnas, la primera columna es el texto representativo que hay que clasificar, las columnas adicionales son clases que se aplican a dicho texto.
- En la imagen siguiente se muestra un archivo CSV que tiene cuatro registros. Cada registro del ejemplo incluye la entrada de texto y una clase, separados por una coma:

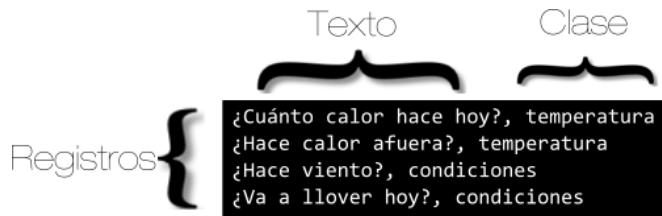


Figura 23 - Ejemplo CSV entrenamiento del servicio Natural Language Classifier

- Actualmente soporta los siguientes idiomas: inglés, árabe, francés, alemán, japonés, coreano, italiano, portugués y español.
- Los datos de entrenamiento deben tener al menos cinco registros (filas) y no pueden superar los 15000 registros. Y además la longitud máxima total de un valor de texto es 1024 caracteres.
- Una posible aplicación de este servicio podría ser dar soporte al cliente. Por ejemplo, se podría utilizar para emprender acciones predictivas, como direccionamiento de las preguntas de los usuarios a la persona correcta, o clasificación de problemas según su gravedad.

Ask a question about the weather

Watch the Natural Language Classifier categorize your weather-related question. In this demo, the classifier is trained to determine whether the question is related to `temperature` OR `conditions`. The output includes the top classification and a confidence score.

Ask

Output

Results
JSON

Natural Language Classifier is `100%` confident that the question submitted is talking about `temperature`.

Figura 24 - Ejemplo del servicio Natural Language Classifier

2.3.2.8 Speech to Text

- Este servicio transcribe la voz, es decir, convierte la voz en texto.
- Ofrece tres interfaces para reconocimiento de voz: WebSocket, interfaz HTTP REST, e interfaz HTTP asíncrona que proporciona llamadas sin bloqueo al servicio.
- El servicio también proporciona una interfaz de personalización que permite ampliar el vocabulario de un modelo base con terminología específica de un dominio.
- Admite los siguientes formatos de audio como entrada: Ogg, WebM con el códec Opus o Vorbis, MP3 o MPEG, WAV, FLAC, PCM, audio mu-law (o u-law) y audio básico.
- Dispone de un modelo banda ancha (16KHz) y otro de banda estrecha (8KHz).
- Permite la transmisión de hasta 100 MB de audio como un flujo continuo o como una entrega única.
- Reconoce las contribuciones de cada hablante a una conversación de múltiples participantes.
- Identifica palabras clave con un nivel de confianza definido por el usuario.
- Puede solicitar resultados de transcripción alternativos e intermedios. El servicio indica los resultados finales en los que tiene la mayor confianza.
- Dispone de una opción que permite censurar palabras (sólo disponible en inglés).

- Permite la conversión inteligente de fechas, horas, números, números de teléfono y valores de moneda a formatos convencionales más legibles (sólo disponible en inglés).
- Soporta los siguientes idiomas: portugués, francés, japonés, chino, árabe, español e inglés.
- Algunas de las principales aplicaciones de este servicio podrían ser: control por voz, transcripción de reuniones, llamadas, dictado de mensajes de correo electrónico y notas.

2.3.2.9 Text to Speech

- El servicio convierte texto escrito en voz. Genera salida de audio sintetizada completa con la entonación y cadencia adecuadas al idioma correspondiente.
- El servicio soporta los siguientes idiomas: portugués, inglés, francés, alemán, italiano, japonés y español.
- Ofrece al menos una voz masculina o femenina, a veces ambas, para cada idioma.
- Proporciona interfaces HTTP REST y WebSocket.
- Produce audio de salida en formato Ogg, WebM con el códec Opus o Vorbis, WAV, FLAC, MP3 (MPEG), 116 (PCM), mulaw o formato básico.
- Acepta texto sin formato o texto marcado con el Lenguaje de marcado de síntesis de voz basado en XML (SSML).
- Extiende SSML al agregar un elemento de transformación de voz que le permite expandir el rango de voces posibles controlando aspectos como el tono, la frecuencia y el timbre.
- Para la interfaz WebSocket admite el elemento <mark> de SSML, que permite sincronizar el texto de entrada y el audio resultante.
- Proporciona una interfaz de personalización que permite definir pronunciaciones con el alfabeto fonético internacional (IPA) o la representación fonética simbólica de IBM (SPR) para especificar cómo pronunciar palabras inusuales.

2.3.2.10 Conversation

El servicio *Watson Conversation* proporciona la comprensión de lenguaje natural y el flujo de la conversación para simular conversaciones humanas.

La siguiente figura muestra la arquitectura general de una solución que integra el servicio *Watson Conversation*:

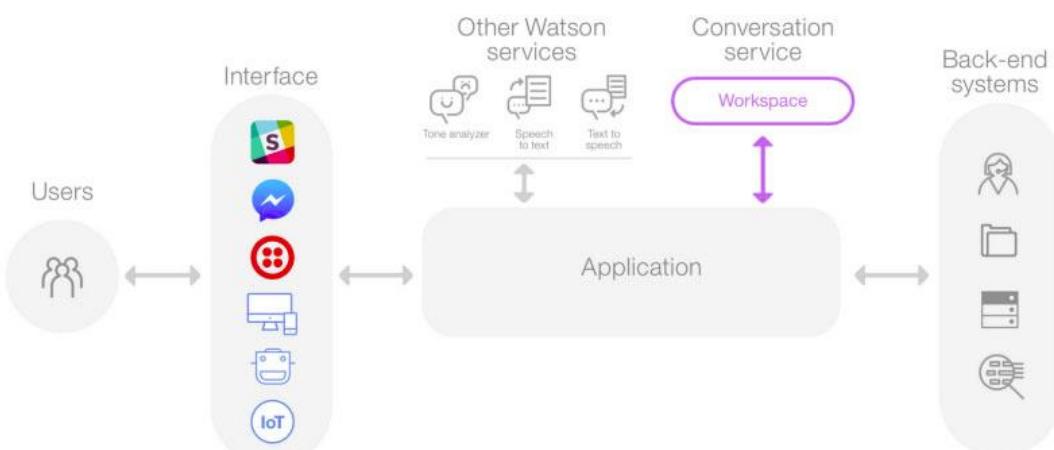


Figura 25 - Arquitectura típica de una aplicación de conversación

- 1) Los usuarios interactúan con la aplicación a través de las interfaces elegidas (servicios de mensajería, ventana de chat dentro de un sitio web, o incluso interfaces de audio cuando se combinan con los servicios Watson Speech to Text).
- 2) La aplicación envía la entrada del usuario al servicio de conversación.
- 3) El servicio de conversación interpreta dicha entrada e identifica los conceptos para los que fue entrenado.
- 4) En función de los conceptos que identifica, maneja el flujo de conversación.
- 5) Reúne la información que necesita consultando diferentes fuentes de datos.
- 6) Finalmente proporciona la información al usuario.

La aplicación se conecta a un espacio de trabajo del servicio de conversación. El espacio de trabajo es un contenedor para los artefactos que definen el flujo de conversación y que proporciona la comprensión del lenguaje natural.

Es posible definir múltiples espacios de trabajo en cada instancia de servicio Watson Conversation. Cada espacio de trabajo es entrenado en un determinado idioma para reconocer ciertos conceptos y para dirigir el flujo de conversación que rige la interacción con usuario.

2.3.2.10.1 Intenciones, entidades y diálogo

El servicio Watson Conversation trabaja con 3 conceptos: las intenciones, las entidades y los diálogos.

Las **intenciones** hacen referencia a las acciones que puede querer hacer el usuario, por ejemplo: “quiero sacar dinero”.

El segundo concepto, las **entidades**, se refiere a los datos que necesitaremos para definir la acción y definir exactamente qué hacer, por ejemplo, sacar dinero... ¿dónde?, ¿a mi alrededor?, ¿en el cajero más cercano?, ¿en el más barato?, ¿en otro sitio al que voy a ir dentro de media hora?

Y, por último, el **diálogo**, define el flujo del diálogo por medio de nodos y preguntas que se desencadenen cuando el bot necesite más información para saber qué acción llevar a cabo.

2.3.2.10.2 Flujo de la conversación

El diálogo utiliza las intenciones y entidades que se identifican en la entrada del usuario, además del contexto de la aplicación, para interactuar con el usuario y finalmente proporcionar una respuesta útil.

El diálogo se representa **gráficamente** en la herramienta como un árbol. Cada intención se procesa en una rama de este árbol. Una rama se compone de uno o varios nodos, y cada nodo del diálogo contiene, como mínimo, una condición y una respuesta.

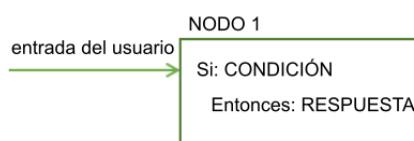


Figura 26 – Watson Conversation: nodo de la conversación

- **Condición:** especifica la información (intención, entidad, o variable contexto) que debe aparecer en la entrada del usuario para activar el nodo.
- **Respuesta:** expresión que el servicio utiliza para responder al usuario. Existen 3 tipos:
 - Respuesta de texto simple.
 - Varias respuestas condicionadas, cada una se activa por una condición diferente.
 - Respuesta compleja, se puede utilizar el editor de JSON para especificar la respuesta en la propiedad "output":{ }.

El servicio procesa el diálogo de arriba a abajo. A medida que baja por el árbol, si el servicio encuentra una condición que se cumple, activa dicho nodo. Luego, se mueve de izquierda a derecha en el nodo

activado para comparar la entrada del usuario con las condiciones de los nodos hijos. Cuando consulta los nodos hijo, se mueve de nuevo de arriba a abajo.

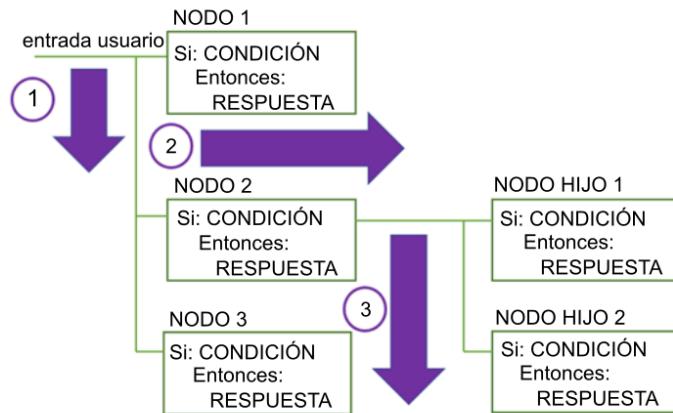


Figura 27 – Watson Conversation: orden de evaluación de los nodos

Por tanto, **el orden de las ramas es importante**, porque los nodos se evalúan de arriba a abajo. Se utiliza el primer nodo base cuya condición coincide con la entrada, y los nodos que hay por debajo del árbol no se activan.

Después de ofrecer la respuesta configurada, se debe indicar al servicio una de las siguientes acciones:

- **Esperar una entrada** de usuario: El servicio espera a que el usuario especifique una nueva entrada que obtiene la respuesta.
- **Ir a otro nodo** del diálogo: omite la espera de información del usuario, y salta directamente a un nodo de diálogo diferente.

2.3.2.10.3 Slots

Esta es una funcionalidad que permite recopilar varios fragmentos de información del usuario en un mismo nodo. La información se solicita hasta que es proporcionada, intercambiando las respuestas correspondientes en cada caso. Si los slots son obligatorios no se sale del nodo de slot hasta que todos los slots están llenos.

Los detalles que el usuario proporciona por adelantado se guardan, y el servicio sólo solicita los que no se hayan especificado.

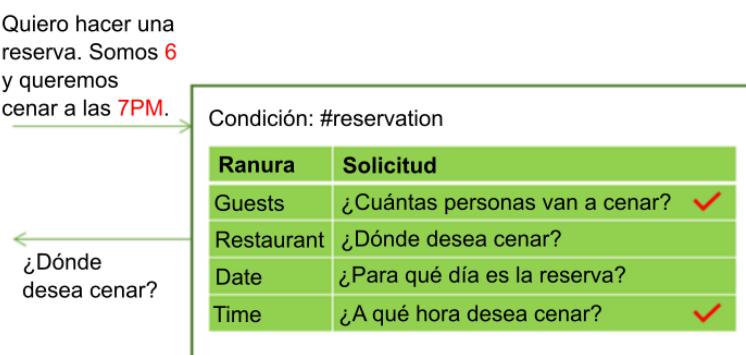


Figura 28 – Watson Conversation: ejemplo de uso de slots

El uso de slots reduce el número de nodos necesarios para completar la información necesaria, y genera un flujo de diálogo más natural entre el usuario y el servicio. Además, son más fáciles de administrar que tratar de recopilar la información mediante el uso de muchos nodos separados.

2.3.2.10.4 Fuzzy matching

Esta funcionalidad mejora la capacidad del servicio para reconocer términos de entrada con una sintaxis similar a la entidad, pero sin requerir una coincidencia exacta. Hay tres componentes para la coincidencia aproximada:

- **Stemming:** es un método para reducir una palabra a su raíz. Esto aumentará el *recall*, que es una medida sobre el número de documentos que se pueden encontrar con una consulta. Por ejemplo, una consulta sobre "bibliotecas" también encuentra documentos en los que solo aparezca "bibliotecario" porque la raíz de las dos palabras es la misma ("bibliotec").
- **Errores ortográficos:** la función puede asignar la entrada del usuario a la entidad correspondiente a pesar de la presencia de errores ortográficos o ligeras diferencias sintácticas. Por ejemplo, si define "jirafa" como sinónimo de una entidad animal, y la entrada del usuario contiene los términos "jirafas" o "girafa", el emparejamiento difuso es capaz de asignar correctamente el término a la entidad animal.
- **Concordancia parcial:** sugiere automáticamente sinónimos basados en subcadenas presentes en las entidades definidas por el usuario, y asigna una puntuación de confianza inferior en comparación con la coincidencia de entidad exacta. Actualmente, esta característica utiliza solo palabras estándar del diccionario de inglés y no utiliza sinónimos definidos por el usuario.

2.3.2.10.5 Analytics Dashboard

El servicio *Watson Conversation* también proporciona un cuadro de mando básico con las métricas más relevantes del uso del servicio como, por ejemplo, tiempo de conversación, intenciones y entidades más importantes, uso de nodos, etc. También registra las conversaciones y hace búsquedas por intenciones o entidades.

Para el plan Lite sólo es posible visualizar los últimos 7 días. Para otros planes se puede visualizar información de días previos.

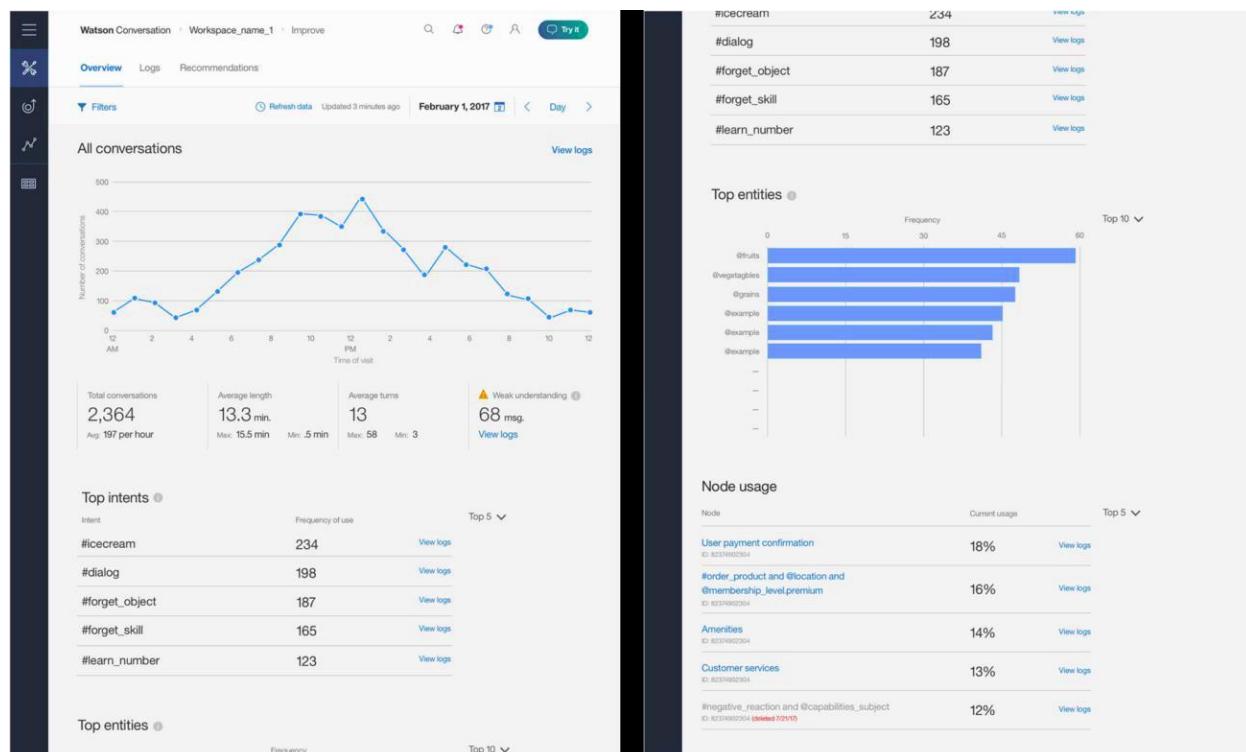


Figura 29 - Cuadro de mando básico proporcionado por el servicio de conversación

3 TRABAJO REALIZADO

A continuación se presenta un resumen del trabajo realizado y la metodología empleada, así como las fuentes de datos utilizadas, las herramientas seleccionadas para la implementación del prototipo, y los detalles técnicos de integración y entrenamiento de cada parte.

Los pasos que se han seguido para la construcción del prototipo se pueden resumir como sigue:

- 1) Definición de requisitos funcionales.
- 2) Elección y estudio teórico de la tecnología de procesado de lenguaje natural.
- 3) Familiarización práctica con la tecnología seleccionada.
- 4) Definición de la arquitectura del prototipo.
- 5) Elección e integración de la interfaz de usuario.
- 6) Elección e integración de las fuentes de datos externas.
- 7) Modelado de fuentes de datos internas (frigorífico).
- 8) Definición de intenciones, entidades y variables de contexto de la conversación.
- 9) Diseño del flujo de la conversación.
- 10) Entrenamiento de intenciones y entidades.
- 11) Implementación del flujo de la conversación.
- 12) Implementación de la aplicación orquestadora.
- 13) Pruebas y correcciones.

Los puntos 1 y 2 ya han sido descritos con anterioridad en los apartados 1.3 y 2, respectivamente. Tras esta introducción, se describirán el resto de pasos y se ofrecerá una muestra de los resultados obtenidos. En la primera parte (apartado 3.1) se describe la arquitectura de la solución propuesta. En el apartado 3.2 se realiza la selección de la interfaz de usuario, una descripción de sus principales características, y su integración con el resto de servicios.

A continuación, se describen las fuentes de datos utilizadas: una base de datos interna (mostrada en el apartado 3.3) con los productos del frigorífico de cada usuario, y una base de datos externa para obtención de recetas (mostrada en el apartado 3.4), a la que se accede mediante una interfaz HTTP.

Después, en el apartado 3.5, se realizará una introducción a los sistemas de diálogo, que particularizaremos para nuestro caso concreto, y donde detallaremos requisitos de la conversación, así como las intenciones, entidades, variables de contexto y flujo del diálogo que han sido implementados.

Para finalizar este capítulo, se presentará la aplicación orquestadora (3.6), que es la encargada de coordinar todas las piezas que componen el chatbot, realizando las consultas necesarias a las fuentes de datos, las peticiones a los servicios externos, y el intercambio de mensajes a través de la interfaz de usuario.

3.1 Arquitectura de la solución

La siguiente figura muestra un esquema de la arquitectura a alto nivel del prototipo implementado para el asistente conversacional:

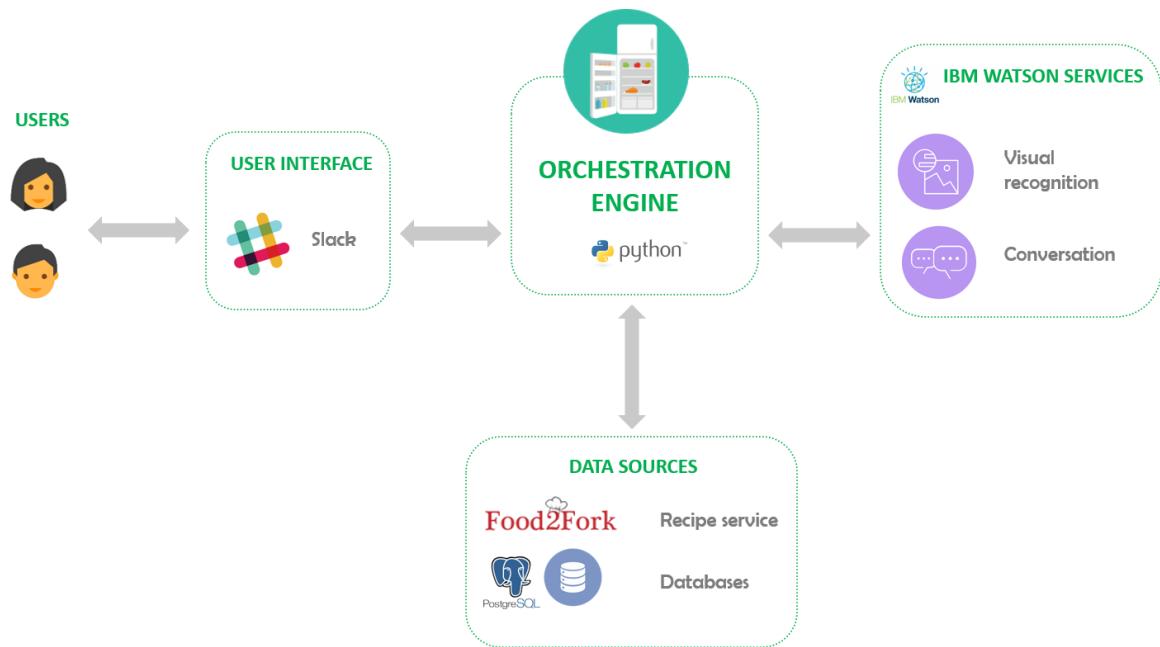


Figura 30 - Arquitectura de la solución

Se compone de los siguientes bloques funcionales:

- **User interface**: medio a través del cual los usuarios podrán comunicarse en **tiempo real** con el asistente. La comunicación está soportada por la plataforma de mensajería Slack a través WebSockets²⁸. La comunicación se realizará utilizando la interfaz web de Slack a través de texto, también será posible compartir ficheros con imágenes de platos para los que el asistente intentará reconocer y proporcionar su receta.
- **Data sources**: este bloque representa las fuentes de datos que utiliza el asistente para componer la respuesta que ofrecerá al usuario. El prototipo consulta principalmente dos fuentes de datos:
 - Base de datos local: base de datos con una tabla en la que se persisten todos los productos que se almacenan en el frigorífico o despensa del usuario.
 - Servicio de recetas en la nube (Food2Fork): que proporciona todo tipo de recetas clasificadas socialmente que han sido extraídas previamente de blogs de cocina y otros sitios web.
- **IBM Watson services**: servicios IBM Watson utilizados por el prototipo. Como indicamos en el capítulo anterior, se utilizan los siguientes servicios:
 - **Visual recognition**: identifica y clasifica escenas, objetos, comida, personas, etc. a partir de la imagen que se le proporcione como entrada.
 - **Conversation**: proporciona la comprensión del lenguaje natural y simula una conversación humana.
- **Orchestration engine**: es la pieza central de la arquitectura, encargada de orquestar las peticiones a los servicios, el acceso de base de datos, y la comunicación con el usuario en tiempo real. Con toda esta información esta pieza compone y hace llegar al usuario la respuesta más adecuada a la consulta o petición que realizó.

²⁸ WebSocket es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP

3.2 Interfaz de usuario

3.2.1 Selección de plataforma de mensajería

Utilizaremos una plataforma de mensajería como interfaz de usuario. Cada plataforma sirve para diferentes propósitos, atrae a grupos demográficos diferentes, y tiene sus propias interfaces y capacidades. Las diferencias entre ellas son sutiles, y radican principalmente en aspectos como el área geográfica donde se utilizan, edad de sus usuarios, estilo de comunicación, capacidades de mensajería, servicios disponibles, integración, comunidad de desarrolladores, etc.

Coemnzaremos haciendo un repaso de las principales características de las plataformas de mensajería más populares en España, para finalmente seleccionar la que mejor se adapte a nuestras necesidades, atendiendo principalmente a criterios como: integración y facilidad para despliegue de chatbots, grupo demográfico al que atraen, estilo de la comunicación, comunidad de desarrolladores, posibilidad de comunicaciones en grupo, y precio.

3.2.1.1 Facebook Messenger

Sin duda, se trata del gigante de las redes sociales, cuenta actualmente con 1.200 millones de usuarios activos.

Messenger se usa principalmente con fines promocionales y de entretenimiento. Actualmente los chatbots y las aplicaciones conversacionales no están permitidos en grupos, pero esto parece que será así por poco tiempo, ya que Facebook está apostando fuertemente por la tecnología chatbot, no sólo por su reciente adquisición de Wit.ai, sino porque ha asegurado que más de 10.000 desarrolladores se encuentran ya trabajando en los chatbots de Facebook Messenger.

Este año además ha anunciado que quiere convertir Facebook Messenger en una “chatbots store”, es decir, una plataforma de soporte y ventas a las marcas para que, desde allí, sin necesidad de instalar una aplicación, el usuario pueda conectar con las marcas mediante chatbots y consumir los productos y servicios que éstas ofrecen.

3.2.1.2 Whatsapp

Se trata de la app de mensajería por excelencia en España y Latinoamérica, y cuenta con más de 1.200 millones de usuarios activos. WhatsApp es propiedad de Facebook desde 2014 y junto con Messenger, brinda a Facebook una cobertura dominante de mensajes en la mayoría de los países del mundo. Sin embargo, se está demorando en adaptarse al auge de los chatbots, porque aún no está abierto a chatbots o aplicaciones conversacionales.

No dispone de un framework oficial para chatbots, aunque se especula que llegará pronto. Actualmente la única vía para crear un chatbot en Whatsapp es desarrollar nuestro propio framework o trabajar con agencias especializadas que dispongan de su propio motor de chatbots en WhatsApp, como ChatBotChocolate, que es la agencia líder con un sistema propio (SONI) para la creación de chatbots en esta aplicación. La primera opción (crear nuestro propio framework) es compleja y se sale del alcance de este trabajo, la segunda opción (trabajar con agencias) tiene un coste elevado.

3.2.1.3 Telegram

Es el gran competidor de WhatsApp, nació casi como un clon de ella, pero es comparativamente muchos más pequeña, con aproximadamente 100 millones de usuarios, sin embargo, su base de usuarios está creciendo.

Telegram además ofrece privacidad en la comunicación de una manera que otras plataformas de mensajería evitan, por lo que se ha convertido en la plataforma de referencia para aquellos que necesitan seguridad en sus comunicaciones.

Esta plataforma posiblemente sea la mejor adaptada a la tecnología chatbot por las siguientes razones:

- Ha ofrecido API públicas y acceso bot durante gran parte de su historia.
- Las fuentes de sus aplicaciones cliente son de código abierto, y cuenta con un protocolo abierto y flexible.
- En este canal encontramos ya cientos de bots con diferentes propósitos.
- Cuenta el sistema de pago *In-Chat Payments*, similar al de iTunes o Google Play, y que lleva el modelo de compra a las conversaciones con bots.
- Su fundador incentiva a los desarrolladores a crear bots de alta calidad, premiando con 25 mil dólares a aquellos que la compañía considere interesante.

3.2.1.4 Slack

Se trata de una aplicación de mensajería corporativa interna. Está libre de anuncios, y se orienta especialmente en torno a canales y grupos, y no tanto a la comunicación uno a uno.

Es la aplicación de mensajería favorita para desplegar chatbots, y está invadido por ellos. De hecho, lleva tiempo apostando por los bots:

- Es uno los primeros canales de mensajería en el apoyo de bots y ha servido como banco de pruebas para muchos de ellos.
- Ha invertido en compañías que despliegan bots en su plataforma, y recientemente también en tecnología de procesamiento de lenguaje natural.
- Su gran apuesta ha sido *SlackBot*, un bot que ayuda a organizar reuniones y recordatorios de tareas no cumplidas.

Esta aplicación de mensajería cuenta actualmente con unos 6 millones de usuarios activos, muy a la cola de Whatsapp o Facebook Messenger.

3.2.1.5 Skype

Pertenece a Microsoft, actualmente cuenta con más de 320 millones de usuarios.

Es una herramienta multiplataforma de gran utilidad en las empresas, que permite hacer llamadas, videoconferencias, mensajes instantáneos de texto, y enviar archivos de cualquier tipo.

Desde que fue adquirida por Microsoft se están haciendo grandes esfuerzos por aproximarse al mercado de los chatbots:

- Dispone de un framework que facilita el desarrollo de chatbots.
- Microsoft ha presentado **Cortana**, una aplicación de inteligencia artificial que se integra con Skype, y que podrá realizar reservas de vuelos, hoteles, restaurantes, etc.
- Permite bots dentro de grupos.

Históricamente se trata de una plataforma de comunicación principalmente basada en audio y video. A largo plazo, aprovechar las capacidades de video-bot de Skype podría brindar oportunidades interesantes que otras plataformas no ofrecen.

3.2.1.6 Twitter

Cuenta con más de 300 millones de usuarios activos. La comunicación en Twitter es principalmente pública, lo que la convierte en una buena herramienta para la comunicación masiva, muy popular en la comunidad tecnológica. Pero sus capacidades de mensajería son comparativamente limitadas, ya que se basa en mensajes únicos de longitud restringida, sin soporte real para nada más que imágenes.

Con respecto a la integración con bots, Twitter está incorporando bots como *DJ Lazy Set*, que permite crear listas de reproducción en Spotify con solo darle el nombre de un artista o grupo que te guste.

3.2.1.7 Conclusiones

Todas las plataformas que se han evaluado son gratuitas, aunque Slack y Skype ofrecen cuentas de pago para entornos empresariales, con una serie de servicios adicionales que no están presentes en la cuenta gratuita.

Whatsapp es la plataforma ideal para dar difusión rápidamente a un mensaje debido a su popularidad y a la gran cantidad de usuarios con los que cuenta, sin embargo, en el despliegue de chatbots es la que mayores limitaciones presenta, puesto que actualmente no está abierta al despliegue de chatbots.

Skype está entrando también en el negocio de los chatbots, por ejemplo, con Cortana, pero aún está bastante lejos de alcanzar a Facebook, Telegram o Slack en cuanto a integración con chatbots se refiere. Además, se trata de una plataforma históricamente orientada a llamadas y vídeo, y no tanto a texto.

Facebook, Telegram y Slack son las plataformas más consolidadas en la tecnología chatbot. Sin embargo, Telegram y Slack ofrecen una mayor flexibilidad de integración, y una comunidad de desarrolladores de chatbots mucho más consolidada. Facebook está abriéndose a las tecnologías chatbot a pasos agigantados, preparándose para hacer caja en los próximos años.

Twitter no se adapta al tipo de comunicación que vamos a implementar en este trabajo, ya que no es propiamente un chat de mensajería, sino un sistema de comunicación de uno a muchos, en el que se publican mensajes sin esperar necesariamente respuesta, a menos que se involucre a alguien, y si hay respuesta es entonces cuando se inicia una conversación.

Por tanto, en nuestro estudio las plataformas que parecen adaptarse mejor a las necesidades de nuestro chatbot son Telegram y Slack porque, aunque no disponen de una cantidad de usuarios elevada, son pioneros en la integración con chatbots.

Telegram cuenta con más usuarios que Slack, sin embargo, para usar Telegram se necesita un número de teléfono asociado, mientras que Slack presenta las ventajas de ser multiplataforma y no necesitar un número de teléfono, sólo hay que crear una cuenta y unirse a cualquiera de sus canales de bots para poder desplegar nuestro chatbot sin coste.

Otro motivo que nos hace inclinar la balanza hacia Slack es que es la plataforma de referencia para desplegar chatbots implementados con la tecnología cognitiva IBM Watson. IBM es uno de sus principales clientes, puesto que esta es su plataforma de mensajería corporativa. Prueba de toda esta apuesta por Slack se puede contrastar con la gran cantidad de tutoriales, comunidad de desarrollo, y código disponible que facilitan el despliegue de bots construidos con Watson en Slack.

3.2.2 Integración con Slack

Slack es una plataforma de comunicación en equipo que permite la comunicación síncrona y asíncrona en diferentes dispositivos. Integra diferentes servicios en una sola aplicación de mensajería, como por ejemplo Dropbox, Twitter, Github, Google drive, WordPress, etc. De este modo, los usuarios de la plataforma pueden disfrutar de todos estos servicios desde un único punto de acceso.

Slack ofrece salas de chat organizadas por temas, así como grupos privados y mensajes directos. También proporciona búsquedas por los chats. La comunicación se puede realizar a través de texto y también proporciona llamadas de voz y vídeo.

Una de las claves de su triunfo es su API [5], que es actualizada constantemente para integrar nuevas funcionalidades.

Para la integración con Slack la aplicación orquestadora (ver apartado 3.5.4.12) utiliza el SDK de Python, concretamente la librería *SlackClient*²⁹. Se trata de un paquete que facilita el uso de la API de Slack, y que también proporciona conexión con la API RTM (Real Time Messaging) de Slack. Permite, por ejemplo, administrar las conexiones de mensajería en tiempo real a través de websocket, o gestionar el estado en los canales a los que el usuario bot esté asociado.

²⁹ <http://slackapi.github.io/python-slackclient/>

Para la integración se han seguido los pasos descritos en los siguientes tutoriales [6] y [7], que se pueden resumir como sigue:

- 1) Registro en Slack web en el espacio de trabajo para desarrolladores (“*Bot developer Hangout*”)³⁰
- 2) Crear una nueva aplicación en Slack.
- 3) Crear el *bot user* y configurarlo con el nombre que aparecerá en pantalla.
- 4) Obtener el token de acceso a la API.
- 5) Instalar la librería Python *slackclient*.
- 6) Descargar el script *startebot*³¹ de Python. Este script utiliza realiza una petición al método de autenticación e imprime el bot ID.
- 7) Configurar el nombre del bot y el token.
- 8) Ejecutar el script y guardar el bot ID.
- 9) Configurar el ID y el token obtenido en los pasos anteriores en la aplicación que controla nuestro bot.
- 10) Conexión al espacio de trabajo *Bot developer Hangout*, y creación de un canal público.
- 11) Añadir la aplicación creada al nuevo canal como si se tratara de cualquier otro usuario.
- 12) Inicio de la conversación en Slack con nuestro bot.

3.3 Base de datos ingredientes: productos del frigorífico

Como se ha indicado en los requisitos de nuestro proyecto, se utilizará una base de datos para llevar el control de los productos contenidos en el frigorífico al que sirve como interfaz nuestro chatbot. En primer lugar, decidimos utilizar una base de datos relacional, principalmente por los siguientes motivos:

- Los datos deben ser consistentes sin dar posibilidad al error.
- Un frigorífico tiene un tamaño limitado y, por tanto, el volumen de datos que manejará la aplicación no será muy alto. Aunque la aplicación maneje distintos frigoríficos de diferentes usuarios la cantidad de datos no se espera que sea elevada.
- No se espera que la estructura de datos que se maneja varíe.
- La información que almacenará está estructurada.

Actualmente existen diferentes gestores de base de datos relacionales, los más populares son: MySQL, PostgreSQL, Oracle, DB2 y Microsoft SQL Server. Descartamos los gestores comerciales (Oracle, DB2 y Microsoft SQL Server) por su coste. Por tanto, la decisión está entre gestor MySQL y PostgreSQL. Aunque ninguno de los dos es perfecto, y además tienen características similares, podría concluirse que MySQL es más rápido, mientras que PostgreSQL es más rico en características.

Finalmente, nos hemos decantado por el uso de PostgreSQL, principalmente por una mayor familiarización con su uso, y dado que se va a implementar un prototipo y realizar una prueba de concepto, optimizaremos los esfuerzos en esta parte para dar más peso al procesado de lenguaje natural y al uso, entrenamiento y configuración de las APIs de Watson.

La base de datos actual consta de una única tabla llamada *products*, que tiene los siguientes atributos:

- **id**: id único de registro. Clave primaria.
- **name**: nombre del producto (egg, apple, etc.).
- **registered**: fecha y hora de registro de entrada del producto.

³⁰ <http://dev4slack.xoxco.com/>

³¹ <https://github.com/mattmakai/slack-starterbot/blob/master/starterbot.py>

- **modified:** fecha y hora en la que se ha modificado el producto, por ejemplo, para modificar la cantidad o cualquier otro parámetro.
- **expiration_date:** fecha de caducidad del producto.
- **quantity:** cantidad en gramos disponible del producto.

La siguiente figura muestra una imagen del contenido de la tabla que se ha utilizado para el desarrollo y las pruebas de la aplicación desarrollada.

id bigint	name character varying(50)	registered timestamp with time zone	modified timestamp with time zone	expiration_date timestamp with time zone	quantity double precision
1	onion	2017-10-01 01:21:36.738+02	2017-10-01 01:21:36.738+02	2017-10-18 01:21:36.738+02	230
2	lettuce	2017-10-01 01:21:36.843+02	2017-10-01 01:21:36.843+02	2018-02-01 00:21:36.738+01	158
3	milk	2017-10-01 01:21:36.867+02	2017-10-01 01:21:36.867+02	2017-10-18 01:21:36.738+02	100
4	mayonnaise	2017-10-01 01:21:36.891+02	2017-10-01 01:21:36.891+02	2018-04-01 01:21:36.738+02	150
5	olive	2017-10-01 01:21:36.914+02	2017-10-01 01:21:36.914+02	2018-10-18 01:21:36.738+02	150
6	ketchup	2017-10-01 01:21:36.937+02	2017-10-01 01:21:36.937+02	2018-09-18 01:21:36.738+02	100
7	ham	2017-10-01 01:21:36.961+02	2017-10-01 01:21:36.961+02	2018-01-11 00:21:36.738+01	250
8	cheese	2017-10-01 01:21:36.984+02	2017-10-01 01:21:36.984+02	2018-02-18 00:21:36.738+01	120
9	cream	2017-10-01 01:21:37.009+02	2017-10-01 01:21:37.009+02	2018-10-10 01:21:36.738+02	80
10	juice	2017-10-01 01:21:37.035+02	2017-10-01 01:21:37.035+02	2018-03-19 00:21:36.738+01	50
11	york ham	2017-10-01 01:21:37.059+02	2017-10-01 01:21:37.059+02	2018-01-13 00:21:36.738+01	200
12	tomato	2017-10-01 01:21:37.092+02	2017-10-01 01:21:37.092+02	2018-01-06 00:21:36.738+01	1000
13	eggplant	2017-10-01 01:21:37.125+02	2017-10-01 01:21:37.125+02	2018-01-20 00:21:36.738+01	500
14	beer	2017-10-01 01:21:37.149+02	2017-10-01 01:21:37.149+02	2018-10-20 01:21:36.738+02	400
15	coke	2017-10-01 01:21:37.172+02	2017-10-01 01:21:37.172+02	2018-03-20 00:21:36.738+01	200
16	apple	2017-10-01 01:21:37.196+02	2017-10-01 01:21:37.196+02	2018-02-20 00:21:36.738+01	500
17	plum	2017-10-01 01:21:37.224+02	2017-10-01 01:21:37.224+02	2018-10-21 01:21:36.738+02	400
18	orange	2017-10-01 01:21:37.248+02	2017-10-01 01:21:37.248+02	2018-03-21 00:21:36.738+01	1500
19	cucumber	2017-10-01 01:21:37.272+02	2017-10-01 01:21:37.272+02	2018-03-21 00:21:36.738+01	500
21	chicken	2017-10-01 01:21:37.272+02	2017-10-01 01:21:37.272+02	2018-01-10 00:21:36.738+01	900
22	salmon	2017-10-01 01:21:37.272+02	2017-10-01 01:21:37.272+02	2018-01-11 00:21:36.738+01	500
23	butter	2017-10-01 01:21:37.272+02	2017-10-01 01:21:37.272+02	2018-10-22 01:21:36.738+02	50
24	egg	2017-10-01 01:21:37.272+02	2017-10-01 01:21:37.272+02	2018-01-17 00:21:36.738+01	300
25	carrot	2017-10-01 01:21:37.272+02	2017-10-01 01:21:37.272+02	2018-01-02 00:21:36.738+01	600
26	zucchini	2017-10-01 01:21:37.272+02	2017-10-01 01:21:37.272+02	2018-10-22 01:21:36.738+02	500
28	mushroom	2017-10-01 01:21:37.272+02	2017-10-01 01:21:37.272+02	2018-02-03 00:21:36.738+01	500
29	banana	2017-10-01 01:21:37.272+02	2017-10-01 01:21:37.272+02	2018-02-10 00:21:36.738+01	700
30	spinach	2017-10-01 01:21:37.272+02	2017-10-01 01:21:37.272+02	2018-02-01 00:21:36.738+01	360

Figura 31 – Tabla de base de datos con los productos del frigorífico

En resumen, hemos utilizado una base de datos simplificada y muy enfocada a la funcionalidad que se ha desarrollado. Se trata sólo de una primera versión, y está pensada para actualizarse manualmente. Es decir, el usuario tendrá que dar de alta los nuevos productos, introducir su fecha de caducidad, dar de baja los productos que saque del frigorífico, o actualizar la cantidad del producto cuando lo utilice. Lo ideal, y donde se espera una evolución en un futuro próximo, es a que el frigorífico detecte los productos de forma autónoma y su cantidad sin necesidad de intervención humana. En el apartado 5.3 se explica esta línea de mejora.

En futuras versiones se podría realizar las siguientes mejoras:

- Añadir una tabla con la información de los usuarios: nombre, alias o diminutivo con el que el chatbot debe dirigirse a él, platos preferidos, intolerancias alimenticias, etc.
- Añadir al modelo una tabla que identifique los frigoríficos: nombre, id del usuario o usuarios dueño del frigorífico³², ubicación en la que se encuentra, etc.
- Añadir identificador de frigorífico a la tabla *products*. Esto, junto con la tabla anterior de usuarios permitiría que un mismo usuario pueda referirse en la conversación a distintos frigoríficos.
- Añadir una tabla con un inventario de productos y todas sus características, como por ejemplo sus calorías. En la tabla *products* se utilizará el *id* de este inventario en lugar del propio nombre del producto. Esto nos permite que no haya inconsistencias ni duplicados de productos, así como homogeneizar la nomenclatura para referirse a ellos.

³² Notar que un mismo frigorífico puede tener varios dueños, por ejemplo, en una familia, o compañeros de piso que comparten frigorífico.

- Añadir otra tabla con el histórico de todos los productos de cada frigorífico, es decir, una vez se dan de baja en la tabla *products* (contenido actual del frigorífico) se almacenan en esta otra tabla, junto con la fecha en la que se introdujo en el frigorífico, la fecha en la que salió de él, la fecha de caducidad y la cantidad. Esto puede ser interesante para calcular estadísticas de consumo (por ejemplo, los productos más consumidos por el usuario, la cantidad de comida que tira el usuario, etc.).

Otra mejora interesante sería disponer de una interfaz gráfica de usuario que facilitase la actualización de la base de datos.

3.4 Base de datos de recetas de cocina

Necesitaremos un servicio externo que cuente con una base de datos de recetas y que proporcione recetas a partir de ciertos ingredientes o tipos de cocina.

3.4.1 Selección de la API de recetas

Presentaremos a continuación un breve resumen de las APIs evaluadas para este fin.

3.4.1.1 Spoonacular

- API de nutrición, recetas y alimentación publicada en *Mashape*.
- Dispone de más de 365.000 recetas y 86.000 productos alimenticios.
- Encuentra recetas por: nombre, a partir de unos ingredientes dados (requeridos o por exclusión), basadas en dietas especiales (vegano, vegetariano, sin gluten, sin lactosa...), o con ciertos requisitos nutricionales (bajo en calorías, rico en hierro, etc.).
- Proporciona 3 vectores por cada búsqueda: ingredientes de la búsqueda, ingredientes usados, ingredientes que faltan. Esto es muy útil para buscar recetas que utilicen exclusivamente los ingredientes de los que dispones.

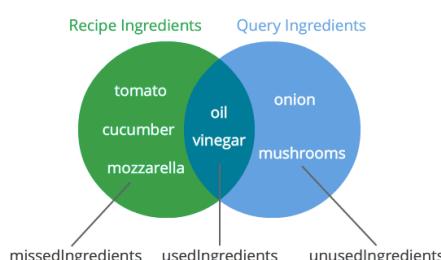


Figura 32 - Vectores de ingredientes API Spoonacular

- Además de las búsquedas, proporciona una amplia gama de funciones útiles para cualquier aplicación de cocina y alimentación. Algunas de las más interesantes son: cálculo de la información nutricional, análisis del coste de una receta, clasificación de recetas en tipos y cocinas, cálculo de plan de comidas completo, recuperación de curiosidades y chistes sobre alimentos, obtención de ingredientes sustitutos, etc.
- Usa una ontología alimentaria con la que es posible realizar búsquedas semánticas, por ejemplo, si buscamos recetas con vegetales, entiende que puede ser cualquier vegetal, como por ejemplo brócoli, zanahoria, etc. También permite buscar recetas utilizando lenguaje natural, por ejemplo, "brownies sin gluten" o "cupcakes veganos bajos en grasa".

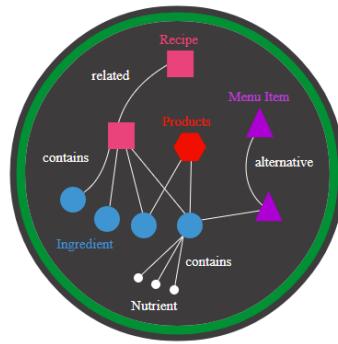


Figura 33 - Ontología de alimentos de Spoonacular

- La API es ampliamente utilizada por universidades, escuelas, empresas y hackathons en todo el mundo.

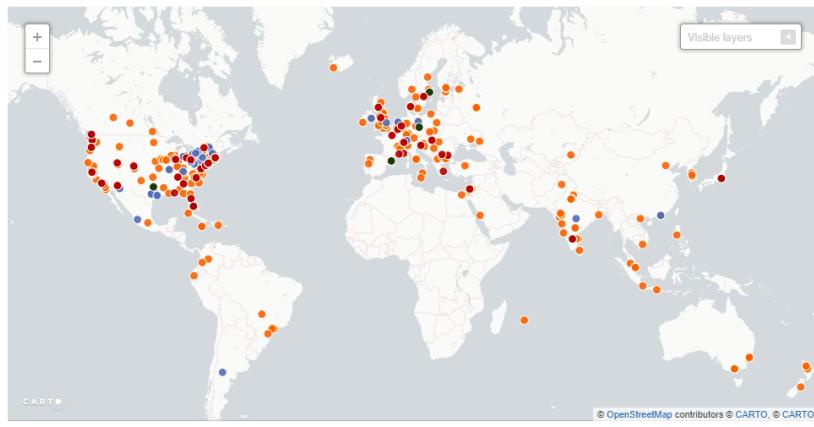


Figura 34 - Mapa de uso de la API Spoonacular

- La API se puede utilizar en muchos lenguajes de programación diferentes. En Mashape hay wrappers para Java, Node, PHP, Python, Objective-C, Ruby y .Net. También están disponibles SDKs para Android, Java, Ruby, Angular y PHP.
- Ofrece diferentes planes de uso, todos ellos de pago, por lo que el alta en el servicio requiere registrar una tarjeta de crédito.

	BASIC \$ 0.00	PRO \$ 29.00	ULTRA \$ 99.00	MEGA \$ 999.00
② requests	50 / daily \$ 0.007 per extra	500 / daily \$ 0.004 per extra	2,500 / daily \$ 0.004 per extra	30,000 / daily \$ 0.001 per extra
② results	500 / daily \$ 0.003 per extra	5,000 / daily \$ 0.003 per extra	25,000 / daily \$ 0.003 per extra	100,000 / daily \$ 0.001 per extra
② tinyrequests	500 / daily \$ 0.003 per extra	5,000 / daily \$ 0.001 per extra	50,000 / daily \$ 0.001 per extra	200,000 / daily \$ 0.001 per extra
② On-Demand API Endpoints			✓	✓
② Support			✓	✓

Figura 35 - Tarifas de uso para la API Spoonacular

3.4.1.2 Food2Fork

- Food2Fork es un sitio web de recetas. Su base de datos se actualiza constantemente porque usa como fuente blogs y sitios web de cocina en constante uso. Algunos de los lugares donde se recogen recetas son los siguientes:

101 Cookbooks	All Recipes	A Spicy Perspective	Back to Her Roots	BBC Food
BBC Good Food	Big Girls Small Kitchen	Bon Appetit	Bunky Cooks	Chow
Closet Cooking	Cookie and Kate	Cookin Canuck	Cookstr	Delishhh
Eats Well With Others	Elana's Pantry	Epicurious	Fine Dining Lovers	Food Network
Food Republic	Framed Cooks	Healthy Delicious	Homesick Texan	Jamie Oliver
Lisa's Kitchen	My Baking Addiction	Naturally Ella	Panini Happy	Pastry Affair
PBS Food	Picky Palate	Pillsbury Baking	Real Simple	Serious Eats
Simply Recipes	Smitten Kitchen	Steamy Kitchen	Tasty Kitchen	The Pioneer Woman
Two Peas and Their Pod	Vintage Mixer	What's Gaby Cooking	Whats Gaby Cooking	

Figura 36 - Fuentes de recetas de Food2Fork

- Food2Fork ofrece una API que expone su funcionalidad de búsqueda de recetas por ingredientes, por nombre, y por tipo de cocina.
- La API da acceso a su base de datos de recetas clasificadas socialmente, es decir, los resultados se ordenan según los gustos de las personas que han cocinado dichas recetas y que han dejado su opinión en los blogs o en los sitios web donde fue publicada.
- La API admite llamadas GET y POST a través de HTTP, y las respuestas tienen formato JSON.
- El servicio está disponible con distintos planes de uso. No es necesario registrar tarjeta bancaria para usar la **cuenta gratuita**, pero existe una limitación de 500 llamadas al día en dicho plan.

Plan	API Calls Per Day	Cost Per Month
Free	500	\$0
Basic	2500	\$50
Pro	10,000	\$150
Ultimate	150,000	\$750

Figura 37 - Planes de uso de la API Food2Fork

3.4.1.3 Edamam

- Está disponible en inglés y **también en español**, aunque este último de forma experimental.
- Dispone de 1,5 millones de recetas en inglés y 200.000 en español.
- Utiliza más de 500 fuentes de recetas (principales sitios de cocina y blogs) y las ordena según su relevancia y popularidad.
- Dispone de una API para **análisis nutricional** y otra para búsqueda de recetas.
- Permite búsqueda de recetas por calorías, preferencias de dieta y por restricciones de alergias.
- Ofrece desglose nutricional detallado de cada receta con más de 25 nutrientes.
- La API está disponible en diferentes tarifas. Existe una **cuenta gratuita** y no es necesario registrar tarjeta de crédito.

	Developer	Startup	Enterprise
Pricing	Free	Free for startups*	Based on usage
API calls limits	5,000 month	Unlimited	Unlimited
Throttling	5 minute	15 minute	none
Recipe data (image, ingredients, title)	✓	✓	✓
Diet, Health and Allergy filters	10 filters ⓘ	10 filters ⓘ	30+ filters ⓘ
Recipe caching	Not allowed	Not allowed	Custom
Recipes in Spanish	✓	✓	✓
Full Nutritional Details	Basic nutrition ⓘ	✓	✓
Glycemic Index			✓
Results per call	up to 100 results	up to 100 results	up to 1000 results

Figura 38 - Tarifas uso API Edamam

3.4.1.4 Yummly

- Herramienta para descubrir, buscar, compartir y organizar recetas.
- Dispone de más de 2 millones de recetas agregadas de las mejores fuentes (blogs y sitios web de recetas).

Serious Eats	Food Network	Kalyn's Kitchen	Post Punk Kitchen	My Baking Addiction
The Pioneer Woman	Taste of Home	Not Without Salt	Real Simple	Honey and Jam
The Shiksa in the Kitchen	David Lebovitz	A Year of Slow Cooking	Baked Bree	Spoon Fork Bacon
I Adore Food!	Leite's Culinaria	Skinnytaste	Gluten Free Girl	Picky Palate
Smitten Kitchen	Simply Recipes	The Kitchn	Gluten Free Goddess	How Sweet It Is
AllRecipes	La Tartine Gourmande	Steamy Kitchen	Roost	Sprouted Kitchen
Epicurious	Big Girls Small Kitchen	Joy the Baker	Mommy's Kitchen	Lottie + Doof
Martha Stewart	Food52	Saveur Magazine	Shape Magazine	The British Larder
MyRecipes	Food.com	Rachael Ray Every Day	Orangette	Pictures & Pancakes
Naturally Ella	Homesick Texan	Williams-Sonoma	Real Mom Kitchen	Steamy Kitchen
Once Upon A Chef	Food Republic	Cook Taste Eat	Salad Pride	
Chow	101 Cookbooks	Turntable Kitchen	The Naptime Chef	

Figura 39 - Fuente de recetas de Yummly

- Permite búsquedas de recetas por ingredientes (requeridos o por exclusión), tipo de cocina, tipo de dieta, valor nutritivo (nivel de colesterol, calorías, vitaminas, grasas, etc.), restricciones alérgicas, sabores (salado, picante, etc.).
- Búsquedas por jerarquías de alimentos, por ejemplo, si excluyes las frutas la receta no tendrá ni manzanas, ni naranjas, etc.
- Se puede ordenar las recetas devueltas según una valoración interna calculada por Yummly en función de la popularidad de la receta.
- Además de un buscador web ofrece una HTTP API para obtener y buscar recetas. La respuesta de la API la devuelve en formato JSON con bastante detalle en cuanto a ingredientes necesarios, valor nutritivo e instrucciones para su cocinado.
- La API está disponible en diferentes tarifas. Existe la posibilidad de probarla con la cuenta *trial*, pero sólo está disponible durante 14 días. También ofrece cuentas especiales para estudiantes o para uso en hackathones, pero todas ellas requieren registro de tarjeta de crédito.

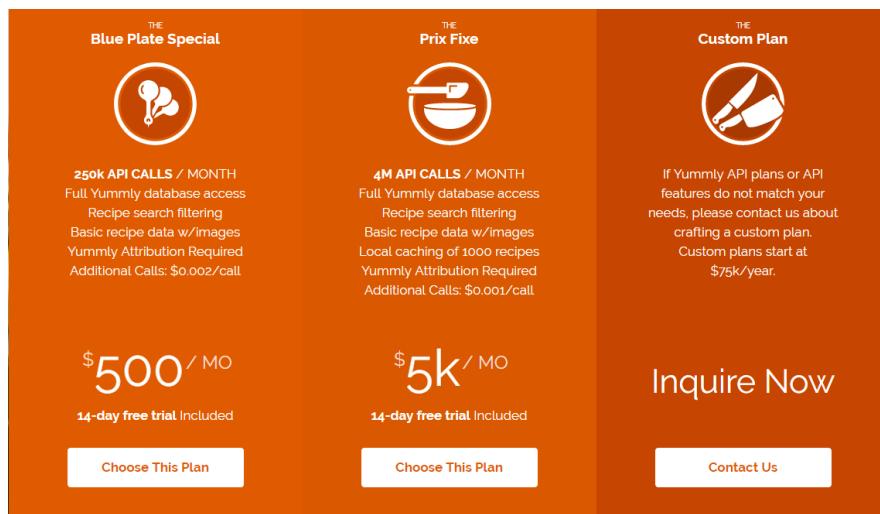


Figura 40 - Tarifas ofrecidas por la API Yummly

3.4.1.5 Conclusiones

Además de las APIs ya mencionadas, también se han evaluado otras que no funcionan o no están disponibles, como por ejemplo, Kraft Recipe, Pearson Kitchen manager o Recipe Bridge. Y otras APIs como Bigoven o Recipal que no proporcionan recetas a partir de ingredientes.

No ha sido sencillo encontrar una API gratuita que se adapte a las necesidades, ya que las que ofrecen más funcionalidad o una mayor riqueza de recetas (Spoonacular o Yummly) no son gratuitas. Otra limitación importante que se ha encontrado en este estudio ha sido el hecho de que ninguna proporcione soporte multiidioma, todas están disponibles en inglés, y no se ha encontrado ninguna con recetas en español, al menos que no sea de forma experimental (Edamam). Este ha sido un factor determinante a la hora de seleccionar el idioma de nuestro chatbot, ya que descartamos realizar traducción para no introducir ruido.

El precio es otro elemento diferenciador importante en este tipo de servicios, puesto que hay pocas APIs disponibles de forma gratuita con riqueza funcional y sin limitaciones significativas en cuanto a número de llamadas diarias.

Edamam dispone de recetas en español de forma experimental y es gratuita sin registro de tarjeta de crédito. Sin embargo, se ha descartado su uso porque en las pruebas realizadas se ha comprobado que esta API no responde. Se han hecho diferentes pruebas utilizando un cliente REST (Postman y Web Restful) y también con Python realizando llamadas HTTP, pero en ningún caso la API devuelve respuesta, sólo se ha obtenido respuesta con el buscador web disponible en su página.

Spoonacular es, sin duda, la más completa y con mejores resultados, prueba de ello es su amplio número de usuarios entre la comunidad de desarrolladores. Sin embargo, su cuenta gratuita, además de presentar limitaciones importantes en cuanto a número de llamadas y resultados, requiere registro de tarjeta de crédito. Por este motivo se ha descartado su uso.

Yummly también es muy rica funcionalmente y aporta gran variedad de opciones de filtrado de recetas, así como una extensa base de datos. Proporciona cuentas para estudiantes y cuentas trial de 15 días con restricciones de uso, pero para cualquiera de estas tarifas es necesario registrar la tarjeta de crédito, y por este motivo se ha descartado su uso.

Food2Fork cuenta con una gran variedad de recetas en inglés de todo tipo de cocina y procedencia. Proporciona las características básicas deseadas, aunque presenta limitaciones en cuanto a funcionalidad si se compara, por ejemplo, con Spoonacular o Yummly. La principal ventaja que presenta esta API frente al resto es que dispone de cuenta gratuita sin registro de tarjeta de crédito, y aunque presenta limitaciones en cuanto a cantidad de llamadas, el número permitido al día es suficiente para la construcción de un prototipo. En consecuencia, se ha decidido utilizar Food2Fork para nuestra implementación.

3.5 Gestión del diálogo

3.5.1 Introducción a los sistemas de diálogo

Típicamente, un sistema de diálogo tiene la siguiente estructura:

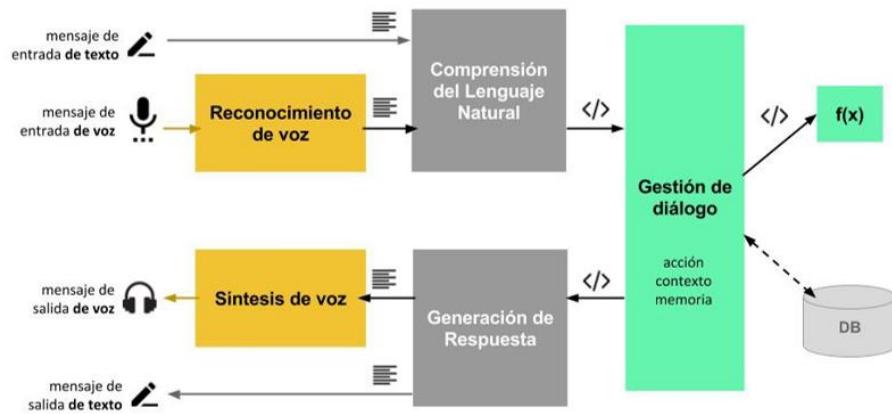


Figura 41 - Esquema simplificado de la arquitectura de un sistema de diálogo

- **Reconocimiento de voz:** reconoce las palabras habladas por el usuario y traduce los mensajes de voz a texto.
- **Comprensión del Lenguaje Natural:** comprende qué quiere el usuario. Interpreta las palabras, es decir, descubre lo que el usuario quiso decir al hablar o escribir estas palabras.
- **Gestión del diálogo:** encargada de entender el estado actual del diálogo y actuar en consecuencia. Formula la respuesta o, si el mensaje no fue claro o está incompleto, interactúa con el usuario para buscar aclaración y obtener la información requerida.
- **Generación de respuestas:** devuelve la información o ejecuta alguna acción, según lo que haya sido solicitado el usuario.
- **Síntesis de voz:** traduce los mensajes de texto a voz.

Por supuesto, es un esquema muy simplificado y hay mucho más detrás de una conversación natural, porque las personas transmiten mucho más que el significado de las palabras pronunciadas como, por ejemplo, su estado emocional y aspectos de su personalidad. Además, en las conversaciones cara a cara, los comportamientos no verbales, como la expresión facial, gestos y la postura del cuerpo, también transmiten significado. Esto podría modelarse en las interfaces de conversación mediante el uso de dispositivos inteligentes dotados con sensores y actuadores que recopilen datos sobre el usuario, el medio ambiente, la ubicación, el movimiento, la orientación y las bioseñales (ej. ritmo cardíaco) del usuario. Hay una amplia gama de métodos para su modelado, y en la actualidad es una línea de trabajo activa, aunque por supuesto se sale de los objetivos del proyecto que estamos desarrollando.

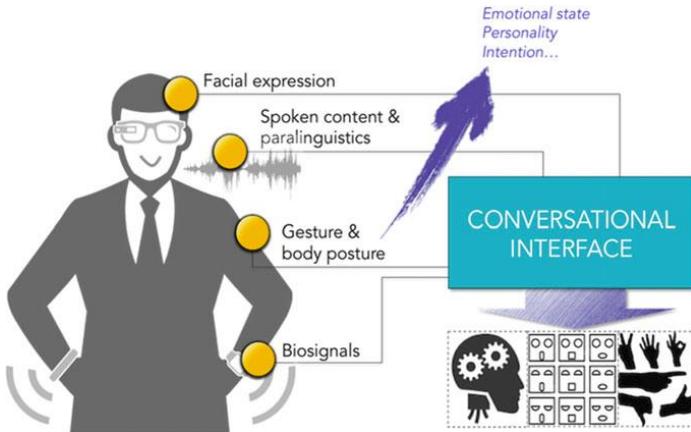


Figura 42 - Entradas adicionales a las interfaces de conversación

En la arquitectura propuesta para la implementación de nuestro prototipo (ver Figura 30) la comprensión del lenguaje natural y la gestión del diálogo son responsabilidad del servicio de conversación de Watson. Este servicio hay que entrenarlo, proporcionando las intenciones y entidades, y además es necesario diseñar el flujo de la conversación, y orquestar todas las llamadas a servicios externos, bases de datos e interfaces de mensajería para componer la respuesta final que será devuelta al usuario.

Antes de pasar a describir el entrenamiento, configuración y diseño del flujo de diálogo que se ha realizado en este trabajo, veamos los requisitos que impondremos a la conversación y algunas cuestiones importantes a tener en cuenta durante su desarrollo.

3.5.2 Requisitos del diálogo

Para una comunicación efectiva, es importante gestionar el diálogo de forma adecuada. Básicamente, implementar un sistema de diálogo requiere:

- Reconocer las palabras introducidas por los usuarios (intenciones, entidades).
- Interpretarlas dentro de un contexto.
- Proporcionar respuestas coherentes dentro del contexto del diálogo y consultando las fuentes adecuadas.

Sin embargo, la realidad es mucho más compleja. En el diálogo no se puede asumir una comunicación perfecta, es decir, admitir que todas las locuciones que intercambien los interlocutores se percibirán e interpretarán de forma correcta. Enumeraremos algunas situaciones a las que el chatbot podría enfrentarse en una conversación real y para las que deberíamos entrenarlo:

- El usuario no siempre proporcionará toda la información que necesita el chatbot, por tanto, éste debe ser capaz de evaluar la información que necesita y solicitar al usuario la información que le falte.
- El usuario podría pedir algo al chatbot para lo que no esté entrenado o no sepa interpretar con la suficiente fiabilidad.
- El usuario podría necesitar rectificar alguna elección que hizo con anterioridad.
- El usuario puede interrumpir al chatbot para rectificar, corregir algún error, o proporcionar información adicional.
- El usuario puede no tener claro lo que quiere o necesita, y el chatbot debería guiarle o ayudarle a elegir.
- El usuario puede gastar bromas, realizar comentarios sarcásticos, maliciosos, o incluso insultar y enfadarse con el chatbot.
- El usuario puede proporcionar información irrelevante, cambiar radicalmente su propósito, o abandonar repentinamente la conversación.

Todas estas situaciones se han formalizado en los sistemas de diálogo como sigue:

- **Grounding:** se refiere al conjunto de cosas que son mutuamente creídas por las personas que participan en la conversación. En el contexto de los chatbots se trata de que el chatbot reconozca

al usuario que ha entendido lo que acaba de decir. Por ejemplo, si el usuario pregunta "¿Puedes ayudarme a encontrar un taxi?", el chatbot responde "¡Por supuesto que puedo ayudarle a encontrar un taxi!".

- **Slots:** cuando un usuario solicita algo, es posible que el chatbot no tenga suficiente información para procesar la solicitud. Por ejemplo, cuando el usuario dice "Estoy buscando un restaurante", y el chatbot no puede dar una ubicación para el restaurante porque no sabe dónde o qué tipo de restaurante está buscando el usuario. Una solución ante estas incertidumbres es definir ciertas entidades como obligatorias, y antes de resolver la intención el chatbot debe solicitar al usuario la información requerida que éste no haya facilitado.
- **Contexto:** en una conversación se debe recopilar y administrar la información del contexto, que contiene toda la información a corto y largo plazo necesaria para manejar una conversación. Por ejemplo, qué acaba de preguntar el usuario, qué entidades necesitan ser completadas, qué pregunta acabo de hacerle al usuario, etc.
- **Iniciativa o turno de palabra:** cuando dos personas se comunican, una de ellas tiene la iniciativa, y es la que dirige la conversación en ese momento. Además, a lo largo de una conversación se van intercambiando el turno de palabra. Una aproximación simple sería que un participante habla, y el otro espera hasta que ese turno ha sido completado para comenzar a hablar. Pero en la práctica saber cuándo intervenir para un chatbot es más complejo, porque existen situaciones en las que se producen silencios, y otras en las que existen ligeros solapamientos.
- **Cambio de contexto:** el patrón de la conversación no es predecible. Puede ocurrir que una conversación fluya a la siguiente sin que la conversación previa se termine por completo. Por ejemplo, imaginemos que estamos pidiendo a un chatbot la reserva de un vuelo, y que nos pregunta "¿A dónde?", y nosotros de repente cambiemos de contexto y respondamos "¿Sabes qué? Odio viajar ¿Me puedes encontrar un restaurante? He cambiado de idea".
- **Reparación conversacional:** los errores en la comunicación son habituales en las conversaciones y pueden darse en diferentes dominios, por ejemplo, entre hombres y mujeres, entre personas con diferentes antecedentes culturales, generacionales, organizacionales, etc. Es importante distinguir fallos como los que se dan cuando el oyente es incapaz de interpretar lo que dijo el hablante, y malentendidos donde la interpretación del oyente es diferente a lo que intentó transmitir el hablante. La primera es más sencilla de manejar, porque se puede pedir al hablante que repita o reformule. Sin embargo, los malentendidos son más difíciles de reparar, ya que puede no ser inmediatamente obvio que ha ocurrido un malentendido. Es importante destacar que la detección y el tratamiento de la falta de comunicación es un requisito para ambos participantes mientras colaboran para lograr el entendimiento mutuo.

A raíz de todo lo anterior, y para mejorar la calidad del diálogo con nuestro chatbot, mostramos los requisitos que imponemos para la definición del flujo de diálogo de nuestro prototipo:

- Diálogo simple.
- Debe requerir el mínimo aprendizaje por parte del usuario.
- Personalización de la conversación.
- Registrar diálogos.
- Capturar y manejar el contexto del diálogo.
- Extraer información relevante.
- Determinar cuándo participar.
- Capacidad de resolución de problemas.
- La lógica de procesamiento del lenguaje natural debe ser única e independiente a la plataforma en la que se despliegue.

Otras características deseables de nuestro prototipo, pero que sobrepasan el alcance de este trabajo, son las que se enumeran a continuación. La mayoría de ellas son características avanzadas, que hoy en día suponen un reto para los sistemas de diálogo actuales, y para los cuales existen líneas de investigación abiertas:

- Adaptar el lenguaje: en función del entorno, la importancia del tema, si se conoce o no al interlocutor, etc.
- Fallos de entendimiento.
- Flexibilidad y adaptación.
- Aprendizaje de la experiencia.
- Adaptación al usuario: perfiles personales. Adaptación a destrezas de los usuarios (p.ej. noveles vs. expertos), adaptación a necesidades especiales (no nativos, discapacitados, niños y ancianos)
- Sistemas multilingües.
- Inteligencia emocional /sistema empático: sistemas capaces de detectar el estado emocional de los usuarios, modificar su comportamiento de acuerdo al mismo, dar respuestas emocionales.
- Dotar de personalidad al sistema de diálogo.
- Detectar ironías, bromas del interlocutor.

3.5.3 Configuración y entrenamiento del servicio Conversation

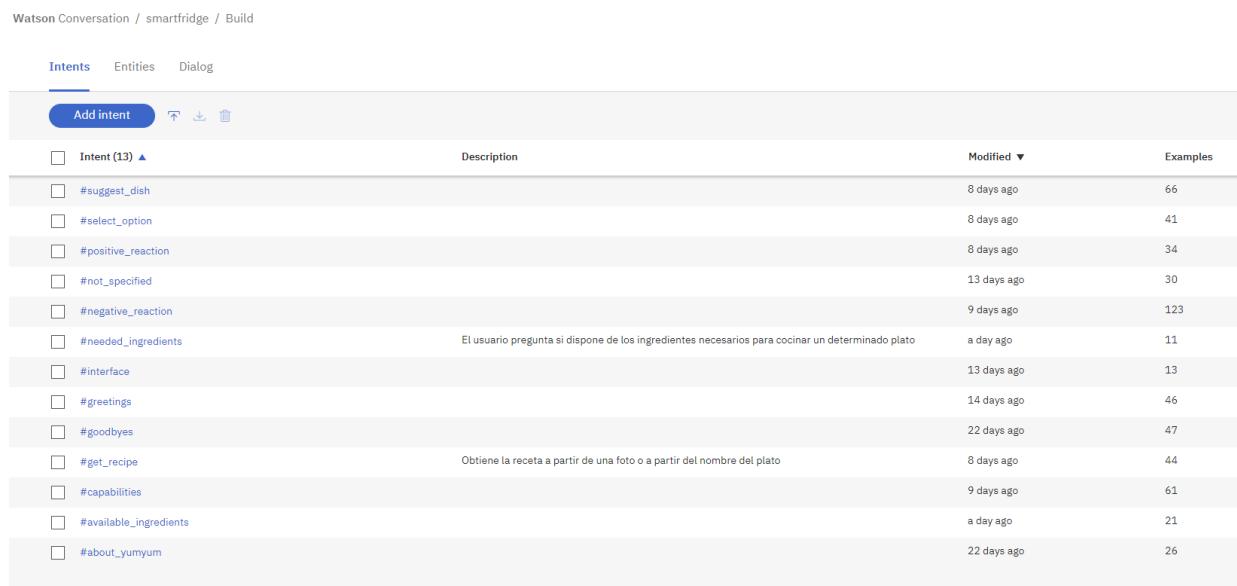
3.5.3.1 Intenciones

Una *intención*, tal y como su propio nombre indica es el propósito o finalidad que persigue el usuario al dirigirse al chatbot, y que debe ser interpretada correctamente para proporcionar la respuesta más adecuada.

Elegir la intención correcta para la entrada de un usuario es el primer paso para proporcionar una respuesta útil. Para hacer posible la interpretación de la intención es necesario proporcionar muchos ejemplos de entradas de usuario, es decir, locuciones con las que creemos que los usuarios se dirigirán al chatbot, e indicar con qué intenciones se corresponden.

En la herramienta gráfica que proporciona el servicio de conversación de Watson, el nombre de una intención siempre tiene como prefijo el carácter #. Es necesario definir una intención para cada tipo de solicitud de usuario a la que el chatbot dé soporte.

La siguiente imagen muestra las intenciones que han sido añadidas al workspace del servicio de conversación de Watson que utilizamos en nuestro prototipo.



The screenshot shows the Watson Conversation service interface with the 'Intents' tab selected. There are 13 intents listed:

Intent	Description	Modified	Examples
#suggest_dish		8 days ago	66
#select_option		8 days ago	41
#positive_reaction		8 days ago	34
#not_specified		13 days ago	30
#negative_reaction		9 days ago	123
#needed_ingredients	El usuario pregunta si dispone de los ingredientes necesarios para cocinar un determinado plato	a day ago	11
#interface		13 days ago	13
#greetings		14 days ago	46
#goodbyes		22 days ago	47
#get_recipe	Obtiene la receta a partir de una foto o a partir del nombre del plato	8 days ago	44
#capabilities		9 days ago	61
#available_ingredients		a day ago	21
#about_yumyum		22 days ago	26

Figura 43 – Lista de intenciones servicio conversación de Watson

La lista de intenciones, con una breve explicación de cada una, junto con algunas de las locuciones que han sido utilizadas para su entrenamiento, son:

- **#about_yumyum:** conocer los gustos, edad, procedencia,... del chatbot.
- **#available_ingredients:** conocer los ingredientes disponibles en la actualidad y aquellos que están a próximos a caducar o ya han caducado. Obtener información sobre un determinado alimento, por ejemplo, saber cuándo caduca o qué cantidad hay disponible.
- **#capabilities:** conocer las habilidades y la utilidad del chatbot, qué sabe hacer, cómo puede ayudarle, etc.
- **#get_recipe:** obtener una receta a partir de una foto o a partir del nombre del plato.
- **#goodbyes:** terminar la conversación y despedirse.
- **#greetings:** presentarse, saludar.
- **#interface:** preguntar sobre los idiomas que soporta el chatbot y cómo debe dirigirse a él.
- **#negative_reaction:** expresar descontento, enfado, insultos y descalificaciones.
- **#not_specified:** expresar ambigüedad, indecisión, e indefinición, o en definitiva trasladar al chatbot la responsabilidad de decidir.
- **#positive_reaction:** transmitir satisfacción, agradecimiento y gratitud.
- **#suggest_dish:** solicitar sugerencias de recetas a partir de un tipo de cocina, una lista de ingredientes, o lo deja a la elección del chatbot, para que sea este quien elija una receta en función de los ingredientes disponibles o las recetas más populares socialmente.
- **#needed_ingredients:** informarse sobre disponibilidad de los ingredientes requeridos para cocinar un determinado plato.
- **#select_option:** seleccionar una opción de entre las que han sido propuestas por el chatbot.

3.5.3.2 Entidades

Una *entidad* representa un objeto relevante para las intenciones que ofrece un contexto específico para dicha intención.

En la herramienta, el nombre de una entidad siempre tiene como prefijo el carácter @. Para preparar el espacio para que reconozca sus entidades, debe obtener una lista de los posibles valores para cada entidad y sinónimos que puedan especificar los usuarios.

Una entidad representa un término o un objeto en la entrada del usuario que proporciona aclaraciones o un contexto específico para una determinada intención. Si las intenciones representan verbos (algo que un usuario quiere hacer), las entidades representan nombres (como el objeto o el contexto de una acción). Las entidades permiten que una sola intención represente varias acciones concretas.

Básicamente, si desea capturar una solicitud o llevar a cabo una acción, utilice una intención. Si desea capturar información que puede afectar a la forma de responder a una solicitud o acción, utilice una entidad.

My entities		
	Values	Modified
<input type="checkbox"/> Entity (7) ▾		Modified ▾
<input type="checkbox"/> @cuisine_type	african, american, argentine, asian, austrian, basque, belgian, breakfast, british, cajun, californian, canadian, carib...	a few seconds ago
<input type="checkbox"/> @dish	legume, hotpot, burgers, pasta, seafood, tacos, fruit, vegetable, meat	a few seconds ago
<input type="checkbox"/> @ingredients	apple, apricot, banana, beer, bread, butter, carrot, cheese, chicken, chocolate, chorizo, clam, coke, corn, cream, cu...	a few seconds ago
<input type="checkbox"/> @intolerances	soy free, Dairy free, gluten free, egg free, Yeast free, nut free, sugar free, alcohol free	a few seconds ago
<input type="checkbox"/> @menu_choice	lunch, breakfast, dinner, dessert, repast, brunch	a few seconds ago
<input type="checkbox"/> @option	6, 5, 1, 2, 4, 3	a few seconds ago
<input type="checkbox"/> @response_types	positive, negative, uncertain	a few seconds ago

Figura 44 – Lista de entidades del servicio de conversación de Watson

A continuación se muestra una lista de las entidades utilizadas en el prototipo junto con una breve explicación de qué representa cada una:

- **@cuisine_type:** tipo de cocina (mexicana, mediterránea, vegana, etc.).
- **@dish:** nombre del plato para el que se solicita la receta.
- **@ingredients:** ingredientes con los que preparar la receta solicitada.
- **@intolerances:** intolerancias alimenticias proporcionadas por el usuario (celíaco, intolerancia a la lactosa, etc.).
- **@menu_choice:** parte del día en la que se prevé que se consumirá el plato proporcionado.
- **@option:** opción seleccionada por el usuario de entre las 6 opciones proporcionadas.
- **@response_types:** tipo de respuesta proporcionada por el usuario (afirmativa, negativa o indeterminada).

3.5.3.3 Variables de contexto

El diálogo no tiene estado, esto significa que no retiene información entre un intercambio con el usuario y el siguiente. La aplicación orquestadora es la responsable de mantener y dar continuidad a la información que necesita.

La aplicación puede pasar información al diálogo, y el diálogo también puede actualizar esta información y pasársela a la aplicación, algo que se realiza mediante las variables de contexto.

Una variable de contexto es una variable que el usuario define en un nodo y para la que puede especificar un valor predeterminado. Otros nodos, o también otra la lógica de aplicación, pueden establecer o cambiar posteriormente el valor de dicha variable de contexto.

En el servicio de conversación de Watson se puede definir una condición sobre valores de variables de contexto haciendo referencia a una variable de contexto desde una condición de nodo de diálogo para determinar si se debe ejecutar dicho nodo. También se puede hacer referencia a una variable de contexto desde condiciones de respuesta del nodo del diálogo para mostrar distintas respuestas, en función de un valor especificado por un servicio externo o por el usuario.

Las variables de contexto utilizadas en el prototipo, junto con una breve explicación, son:

- **\$search_recipe** (True/False): indica si la aplicación debe proporcionar la receta de un determinado plato que recibe como entrada en la variable **\$dish**.

- **\$image_recipe** (True/False): indica que el usuario ha solicitado la receta a partir de una imagen que ha intercambiado en el chat.
- **\$suggest_dish** (True/False): indica si la aplicación debe proporcionar una receta a partir de entradas como una lista de ingredientes (**\$ingredients**), el nombre del plato (**\$dish**) así como las intolerancias alimenticias que hayan sido introducidas durante la petición (**\$intolerances**)
- **\$yum_sugest** (True/False): indica si la aplicación debe proporcionar una lista de sugerencias sin esperar más entradas por parte del usuario.
- **\$summary** (True/False): indica si se debe proporcionar un resumen del contenido de la base de datos al usuario.
- **\$option**: número de opción seleccionada por el usuario de entre las 6 que fueron propuestas.
- **\$cuisine_type**: tipo de cocina con los que realizar la búsqueda.
- **\$ingredients**: lista de ingredientes con los que hacer la búsqueda de la receta.
- **\$intolerances**: intolerancias con las que realizar la búsqueda.
- **\$dish**: plato para el que se debe proporcionar la receta.
- **\$counter**: contador para gestionar acciones que se repiten durante el flujo de la conversación.
- **\$insult_counter**: contador global de reacciones negativas por parte del usuario.

3.5.3.4 Entidades de Sistema

El servicio de conversación de Watson proporciona una serie de entidades del sistema, que son entidades comunes que puede utilizar para cualquier aplicación. La habilitación de una entidad del sistema permite incorporarla en el espacio de trabajo con los datos de entrenamiento más comunes del lenguaje.

The screenshot shows the 'Entities' tab in the Watson Assistant interface. Under the 'System entities' tab, there is a list of entities with their descriptions and status (On/Off). The entities listed are:

Name	Description	Status
@sys-currency	Extracts currency values from user examples including the amount and the unit. (20 cents)	Off
@sys-date	Extracts date mentions (Friday)	On
@sys-location BETA	The @sys-location system entity extracts place names (country, state/province, city, town, etc.) from the user's input. (Boston)	Off
@sys-number	Extracts numbers mentioned from user examples as digits or written as numbers. (21)	On
@sys-percentage	Extracts amounts from user examples including the number and the % sign. (15%)	Off
@sys-person BETA	The @sys-person system entity extracts names from the user's input. (Anna)	On
@sys-time	Extracts time mentions (at 10)	On

Figura 45 – Entidades de sistema disponibles en el servicio de conversación de Watson

- **@sys-number**: detecta los números que se han escrito utilizando números o palabras. En cualquiera de los casos, se devuelve un valor numérico, incluidos números enteros, fracciones decimales o incluso números escritos como palabras. Los formatos reconocidos son: “20 céntimos”, “cinco euros”, “10\$”. Reconoce tanto el valor numérico (entero o decimal) como las unidades (dólares, euros, etc.).
- **@sys-date**: extrae menciones como “viernes”, “hoy” o “1 de noviembre”. El valor de esta entidad guarda la fecha inferida correspondiente como una serie de caracteres en el formato “aaaa-MM-dd”, por ejemplo “2016-11-21”. El sistema cumplimenta los elementos que faltan de una fecha (como el año para “21 de noviembre”) con valores de la fecha actual.

- **@sys-time:** extrae menciones como *2pm*, *4* o *15:30*. El valor de esta entidad almacena la hora como una serie de caracteres en el formato "*HH:mm:ss*" (por ejemplo, "*13:00:00*"). Las menciones a fecha y hora, como por ejemplo "*ahora*" o "*dentro de dos horas*" se extraen como dos menciones de entidades separadas, una **@sys-date** y una **@sys-time**.
- **@sys-location:** extrae nombres de lugares (país, estado/provincia, ciudad, pueblo, etc.) a partir de la entrada del usuario.
- **@sys-percentage:** detecta porcentajes expresados en una expresión con el símbolo de porcentaje, o escritos con la palabra *por ciento*. En cualquiera de los casos, se devuelve un valor numérico. Los formatos reconocidos son: *15%*, *10 por ciento*.
- **@sys-person:** extrae nombres de la entrada del usuario. Los nombres son reconocidos individualmente, de modo que "*Will*" no se trata igual que "*William*", y viceversa. Algunos de los formatos reconocidos son: "*Will*", "*Jane Doe*" o "*Vijay*".

3.5.4 Flujo de diálogo

Un *diálogo* en Watson es un flujo de conversación que define la forma en que la aplicación responde cuando reconoce las intenciones y entidades definidas. Se trata de la pieza más importante en el gestor del diálogo. En esencia, se trata de un conjunto de reglas donde ayudamos a modelar la conversación y el comportamiento del chatbot. Es el verdadero cerebro de un asistente.

Watson proporciona una herramienta gráfica con la que se puede definir el flujo que debe seguir la conversación y las respuestas que proporcionar en base a las intenciones y entidades que son reconocidas a partir de la entrada del usuario.

En los siguientes apartados se detallará el comportamiento que ha sido definido para cada una de las intenciones del prototipo. Para una mejor comprensión se han incluido diagramas que muestran de forma simplificada el flujo que sigue la conversación para cada intención, junto con una explicación funcional en texto de las diferentes alternativas del flujo. Por último, para las intenciones que no tienen un número excesivo de nodos se han incluido además los nodos del diálogo en la herramienta gráfica del servicio de conversación de Watson con la configuración de cada nodo.

Con el fin de facilitar la comprensión de las siguientes subsecciones, en la siguiente figura se muestra la leyenda de los diagramas de flujo que se mostrarán como soporte visual.

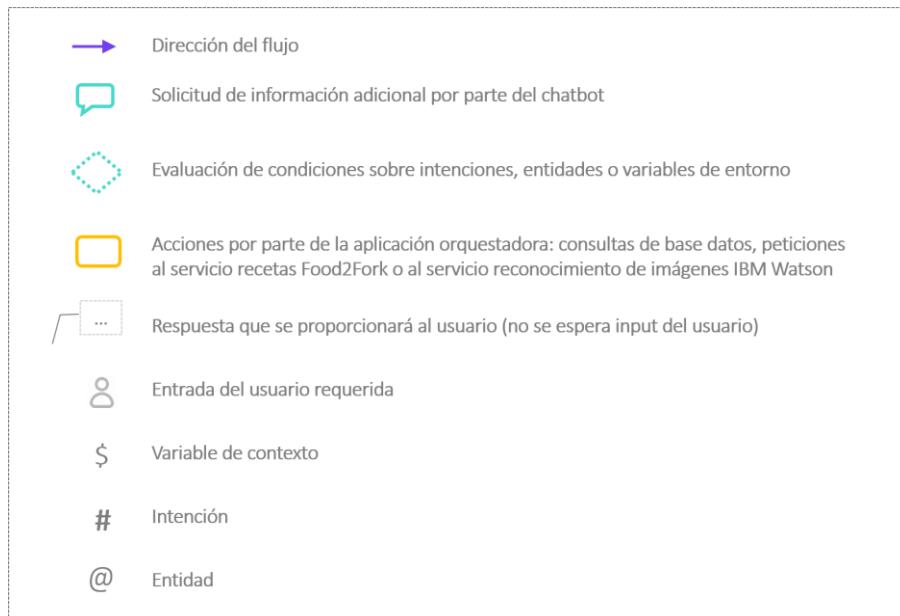


Figura 46 – Leyenda de los diagramas de flujo de la conversación

3.5.4.1 Nodo Welcome

Se trata de un nodo especial existente en todos los flujos de diálogo de Watson. Este nodo indica que el usuario ha iniciado una nueva conversación. En este caso resetaremos todas las variables de contexto, no se proporciona respuesta al usuario y quedamos a la espera de recibir entrada del usuario.

En la imagen siguiente se muestra el nodo *Welcome* que ha sido definido utilizando la interfaz de Watson para la configuración del diálogo, y que consta de 3 partes:

- 1) Condición que se evalúa para este nodo.
- 2) Respuesta en caso de cumplirse la condición impuesta en 1).
- 3) Acción tras la respuesta, que en este nodo en concreto será quedarse a la espera de la entrada del usuario (*wait for user input*).

The screenshot shows the Watson Assistant interface with the 'Welcome' node selected. At the top, there's a title bar with 'Welcome', a 'Customize' button, and a close button. Below the title bar, the first section is titled 'If bot recognizes:' with a condition 'welcome'. Underneath this, the second section is titled 'Then respond with:' and contains a JSON code block. The third section is titled 'And finally' with a 'Wait for user input' option.

```
1 {
2   "context": {
3     "dish": null,
4     "option": null,
5     "counter": 0,
6     "summary": "false",
7     "yum_sugest": "false",
8     "ingredients": null,
9     "cuisine_type": null,
10    "image_recipe": "false",
11    "intolerances": null,
12    "suggest_dish": "false",
13    "search_recipe": "false",
14    "insult_counter": 0
15  },
```

Figura 47 – Flujo de la conversación: nodo *Welcome*

3.5.4.2 Intención #greetings

El nodo que controla esta intención, en caso de satisfacer la condición y detectar que el usuario está saludando, realiza las siguientes acciones:

- 1) Resetea las variables de contexto que activan acciones en la aplicación orquestadora (*\$summary*, *\$yum_sugest*, *\$image_recipe*, *\$suggest_dish* y *\$search_recipe*).
- 2) Responde con un saludo al usuario. Hay dos posibles respuestas, que con la configuración actual usa de forma secuencial, pero que también se podría haber configurado para usarlas de forma aleatoria.



Figura 48 – Intención #greetings: respuesta al usuario

- 3) Se queda a la espera de una nueva entrada del usuario.

```

1 {
2   "context": {
3     "summary": "false",
4     "yum_sugest": "false",
5     "image_recipe": "false",
6     "suggest_dish": "false",
7     "search_recipe": "false"
8   },
9   "output": {
10     "text": {
11       "values": [
12         "Hello, I'm yumyumBot, can I help you?",
13         "Hi, What can I do for you?"
14       ],
15       "selection_policy": "sequential"
16     }
17   }
18 }
  
```

Figura 49 – Nodo que gestiona la intención #greetings

3.5.4.3 Intención #capabilities

Al detectar que el usuario solicita información sobre las capacidades o habilidades del chatbot se activará este nodo, y realizará las siguientes acciones:

- 1) Resetear las variables de contexto que activan acciones en la aplicación orquestadora.
- 2) Responder al usuario explicando cómo puede ayudarle y cuáles son sus habilidades.

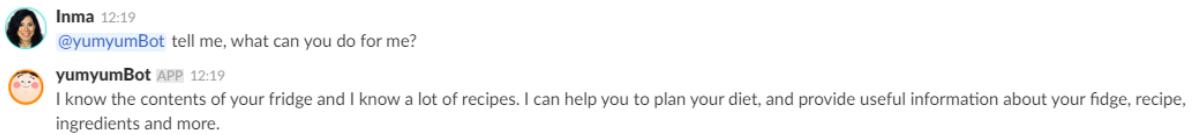


Figura 50 – Intención *#capabilities*: respuesta al usuario

- 3) Quedarse a la espera de nueva entrada del usuario.

#Capabilities

If bot recognizes:
#capabilities ⊖ ⊕

```

1 {
2   "context": {
3     "summary": "false",
4     "yum_sugest": "false",
5     "image_recipe": "false",
6     "suggest_dish": "false",
7     "search_recipe": "false"
8   },
9   "output": {
10    "text": [
11      "values": [
12        "I know the contents of your fridge and I know a lot of recipes. I can
13        help you to plan your diet, and provide useful information about your fidge,
14        recipe, ingredients and more."
15      ],
16    ]
17  }
18}

```

Then respond with:

And finally

Wait for user input ✓

Figura 51 – Nodo intención *#capabilities*

3.5.4.4 Intención *#about_yumyum*

Al detectar que el usuario solicita información “personal” sobre el chatbot se activará este nodo, y realizará las siguientes acciones:

- 1) Resetear las variables de contexto que activan acciones en la aplicación orquestadora.
- 2) Responder al usuario presentándose, es decir, diciendo su nombre y que es un bot para ayudarle (ver Figura 54).
- 3) Saltar a la intención “*#capabilities*” para completar la información. Se salta a la intención sin evaluar la condición, es decir, directamente se proporciona la respuesta correspondiente al usuario.

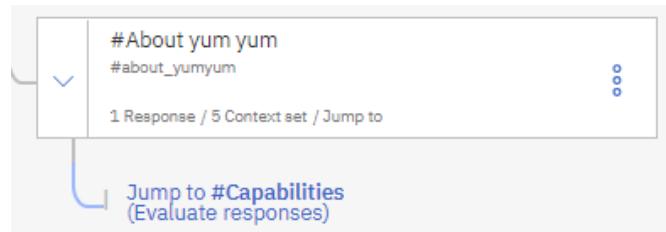


Figura 52 – Nodos intención `#about_yumyum` y salto a `#capabilities`

Inma 12:26
 @yumyumBot Who you are? Where do you come from?
 yumyumBot APP 12:26
 I am yum yum bot, I am here to help you.
 I know the contents of your fridge and I know a lot of recipes. I can help you to plan your diet, and provide useful information about your fidge, recipe, ingredients and more.

Figura 53 – Intención `#about_yumyum`: respuesta

- 4) Quedarse a la espera de nueva entrada del usuario.

The screenshot shows the configuration for the '#about_yumyum' intent. It includes fields for 'If bot recognizes:' containing '#about_yumyum' and 'Then respond with:' containing a JSON payload. The JSON payload is as follows:

```

1 {
2   "context": {
3     "summary": "false",
4     "yum_sugest": "false",
5     "image_recipe": "false",
6     "suggest_dish": "false",
7     "search_recipe": "false"
8   },
9   "output": {
10    "text": {
11      "values": [
12        "I am yum yum bot, I am here to help you"
13      ],
14      "selection_policy": "sequential"
15    }
  }

```

At the bottom, there is a 'Jump to...' button followed by '#Capabilities (Response)'.

Figura 54 – Configuración de la intención `#about_yumyum`

3.5.4.5 Intención `#interface`

Al detectar que el usuario solicita información sobre cómo comunicarse con el chatbot se activará este nodo, y realizará las siguientes acciones:

- 1) Resetear las variables de contexto que activan acciones en la aplicación orquestadora.
- 2) Responder al usuario explicando que sólo entiende inglés (ver Figura 57).

- 3) Saltar a la intención “#capabilities” para completar la información. Se salta a la intención sin evaluar la condición, es decir, directamente se proporciona la respuesta correspondiente al usuario.

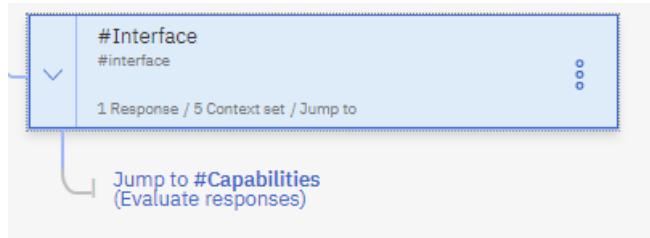


Figura 55 – Nodo intención #interface y salto a #capabilities

Inma 12:36
 @yumyumBot Do you understand me? What languages do you speak?
 yumyumBot APP 12:36
 I only know how to speak in English
 I know the contents of your fridge and I know a lot of recipes. I can help you to plan your diet, and provide useful information about your fidge, recipe, ingredients and more.

Figura 56 – Intención #interface: respuesta al usuario

- 4) Quedarse a la espera de una nueva entrada por parte del usuario.

```

1 {
2   "context": {
3     "summary": "false",
4     "yum_sugest": "false",
5     "image_recipe": "false",
6     "suggest_dish": "false",
7     "search_recipe": "false"
8   },
9   "output": {
10     "text": {
11       "values": [
12         "I only know how to speak in English"
13       ],
14       "selection_policy": "sequential"
15     }
16   }
}
  
```

Figura 57 – Nodo intención #interface

3.5.4.6 Intención #positive_reaction

Al detectar que el usuario reacciona de forma positiva, agradeciendo la ayuda del chatbot, se activará este nodo, y realizará las siguientes acciones:

- 1) Resetear las variables de contexto que activan acciones en la aplicación orquestadora.
- 2) Responder al usuario expresando alegría y participando en el agradecimiento mostrado por parte del usuario (ver Figura 57). En la respuesta se utilizan emoticonos que refuerzan la comunicación con elementos visuales, que consiguen hacer más fluidas, menos programadas y más divertidas la

conversación.

 **yumyumBot** APP 12:40
Ok, good choice! The Smashed Chickpea & Avocado Salad recipe below:
To cook this dish you need the following ingredients:
- 1 (15 ounce) can chickpeas or garbanzo beans (I use Bush's Garbanzo Beans)
- 1 large ripe avocado
- 1/4 cup fresh cilantro, chopped
- 2 tablespoons chopped green onion
- Juice from 1 lime
- Salt and pepper, to taste
- Bread of your choice (I use whole wheat bread)
- Fresh spinach leaves or other sandwich toppings: lettuce, tomato slices, sprouts, etc.
Here, you can find the **method of cooking**: <http://www.twopeasandtheirpod.com/smashed-chickpea-avocado-salad-sandwich/>
 **Smashed Chickpea Avocado Salad | Two Peas & Their Pod**
Smashed chickpea salad made with chickpeas, avocado, cilantro, green onion, and lime juice. This quick, easy, and healthy salad makes a great sandwich or dip.

 **Inma** 12:41
@yumyumBot Wow, great! That's so good

 **yumyumBot** APP 12:41
Nice to help you! 

Figura 58 – Intención #positive_reaction: respuesta

- 3) Quedarse a la espera de una nueva entrada por parte del usuario.

#Positive reactions  

If bot recognizes:
#positive_reaction  

Then respond with:

```
1 {
2   "context": {
3     "summary": "false",
4     "yum_sugest": "false",
5     "image_recipe": "false",
6     "suggest_dish": "false",
7     "search_recipe": "false"
8   },
9   "output": {
10     "text": {
11       "values": [
12         "Glad to help you! :success:",
13         "Nice to help you! :success:",
14         "Charmed! :success:"
15       ],
16     }
17   }
18 }
```

And finally
Wait for user input 

Figura 59 – Nodo intención #positive_reaction

3.5.4.7 Intención #needed_ingredients

El flujo de la conversación para esta intención se ha diseñado para que se comporte del siguiente modo:

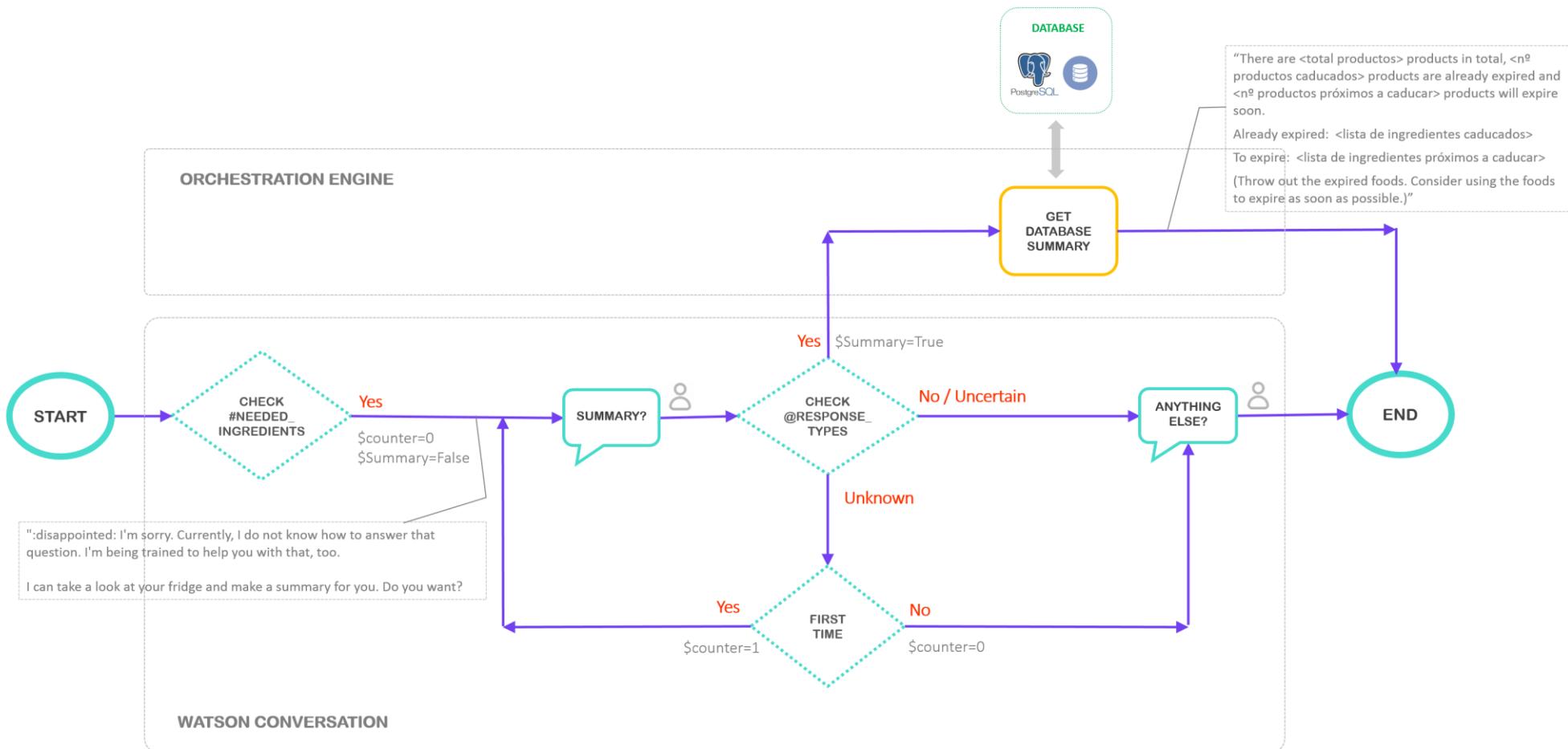


Figura 60 - Flujo de la conversación para la intención #needed_ingredients

- 1) Se detecta que la petición del usuario corresponde a la intención `#needed_ingredients`, es decir, el usuario ha preguntado si dispone de los ingredientes necesarios para cocinar una cierta receta.
- 2) El chatbot responde al usuario diciendo que actualmente no puede contestar a esa petición y luego ofrece realizar un resumen del contenido del frigorífico (productos totales, productos caducados, productos próximos a caducar, otras recomendaciones y comentarios en función del contenido).

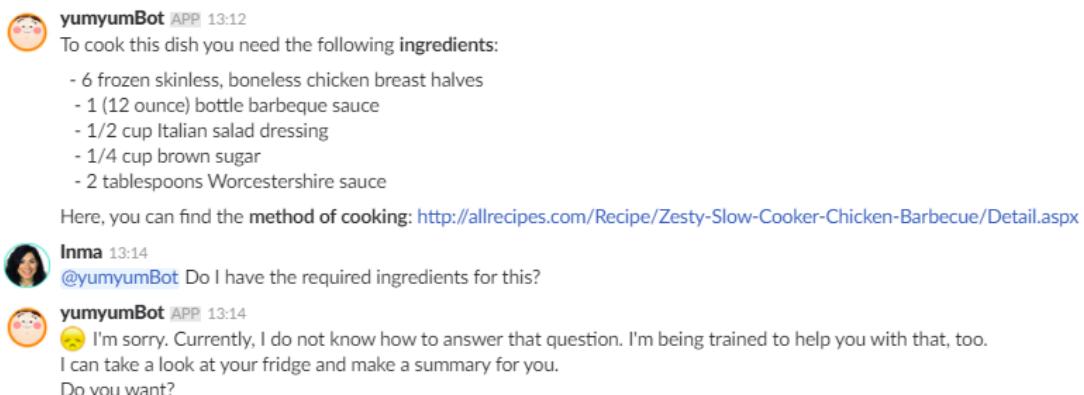


Figura 61 – Intención `needed_ingredients`: respuesta chatbot

- 3) El chatbot espera la respuesta del usuario respecto a si le interesa el resumen del contenido de la nevera.
- 4) Una vez el usuario proporciona respuesta, esta se evalúa para extraer de ella la entidad `@response_types`. En función de la respuesta proporcionada por el usuario el flujo de la conversación toma caminos diferentes:

POSITIVE	UNKNOWN + NEGATIVE	UNCERTAIN
<p> I'm sorry. Currently, I do not know how to answer that question. I'm being trained to help you with that, too. I can take a look at your fridge and make a summary for you. Do you want?</p> <p>Inma 1:51 p.m. @yumyumBot yes, tell me</p> <p> There are 30 products in total, 4 products are already expired and 4 products will expire soon.</p> <p> Already expired: carrot, milk, onion, tomato.</p> <p> To expire: chicken, ham, salmon, york ham.</p> <p>Throw out the expired foods. Consider using the foods to expire as soon as possible.</p>	<p> @yumyumBot do i have yo buy something?</p> <p>yumyumBot APP 1:53 p.m. 😔 I'm sorry. Currently, I do not know how to answer that question. I'm being trained to help you with that, too. I can take a look at your fridge and make a summary for you. Do you want?</p> <p>Inma 1:53 p.m. @yumyumBot mmm</p> <p>yumyumBot APP 1:53 p.m. Please, tell me, do you want a summary of the available products?</p> <p>Inma 1:54 p.m. @yumyumBot i dont</p> <p>yumyumBot APP 1:54 p.m. Can I help you with anything else?</p>	<p>Inma 2:05 p.m. @yumyumBot maybe</p> <p>yumyumBot APP 2:05 p.m. Can I help you with anything else?</p> <p>Inma 2:06 p.m. @yumyumBot need additional ingredients?</p> <p>yumyumBot APP 2:06 p.m. 😔 I'm sorry. Currently, I do not know how to answer that question. I'm being trained to help you with that, too. I can take a look at your fridge and make a summary for you. Do you want?</p> <p>Inma 2:07 p.m. @yumyumBot i am not sure</p> <p>yumyumBot APP 2:07 p.m. Can I help you with anything else?</p>

Figura 62 – Intención `#needed_ingredients`: varios tipos de respuestas del usuario

▪ Respuesta **positiva** (sí, vale, por supuesto, etc.):

- Se pone la variable de contexto `$summary = True`, para indicar a la aplicación orquestadora que debe proporcionar el resumen de productos.
- La aplicación orquestadora consulta en base de datos los productos y analiza la información extraída para presentar un resumen con el total de productos, los productos que ya han caducado (fecha caducidad < fecha actual) y los productos próximos a caducar en los próximos `DAYS_TO_EXPIRE` días (por defecto igual a una semana).

- Envía la respuesta al usuario.
- Finaliza la atención de esta intención y se queda a la espera de nuevas entradas del usuario.

 **Inma** 13:50
@yumyumBot do i have the required ingredients?

 **yumyumBot** APP 13:50 ☆
😔 I'm sorry. Currently, I do not know how to answer that question. I'm being trained to help you with that, too. I can take a look at your fridge and make a summary for you. Do you want?

 **Inma** 13:51
@yumyumBot yes, tell me

 **yumyumBot** APP 13:51
There are 30 products in total, 4 products are already expired and 4 products will expire soon.

♻️ **Already expired:** carrot, milk, onion, tomato.

⏰ **To expire:** chicken, ham, salmon, york ham.

Throw out the expired foods. Consider using the foods to expire as soon as possible.

Figura 63 – Intención *#needed_ingredients*: respuesta positiva

- Respuesta negativa (no, nada, no me interesa, etc.) o indefinida (no sé, no lo tengo claro, no sabría qué decirte, etc.)
 - Se finaliza la atención de esta intención preguntando al usuario si puede ayudarle con cualquier otra cosa.
 - Se queda a la espera de respuesta por parte del usuario para identificar nuevas intenciones.

 **Inma** 13:14
@yumyumBot Do I have the required ingredients for this?

 **yumyumBot** APP 13:14 ☆
😔 I'm sorry. Currently, I do not know how to answer that question. I'm being trained to help you with that, too. I can take a look at your fridge and make a summary for you.
Do you want?

 **Inma** 13:15
@yumyumBot no

 **yumyumBot** APP 13:15
Can I help you with anything else?

Figura 64 – Intención *#needed_ingredients*: respuesta negativa

- **Respuesta no válida:**
 - Si es la primera vez que el usuario proporciona una respuesta inválida ($$counter==0$) vuelve a preguntar al usuario una última vez si está interesado en obtener un resumen del contenido del frigorífico. Aumenta el contador ($$counter==1$) y vuelve al paso 4 para volver a evaluar la respuesta del usuario.

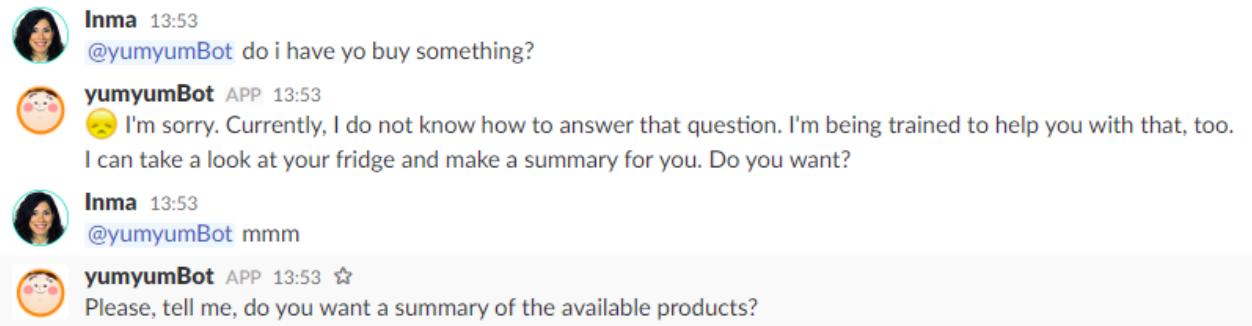
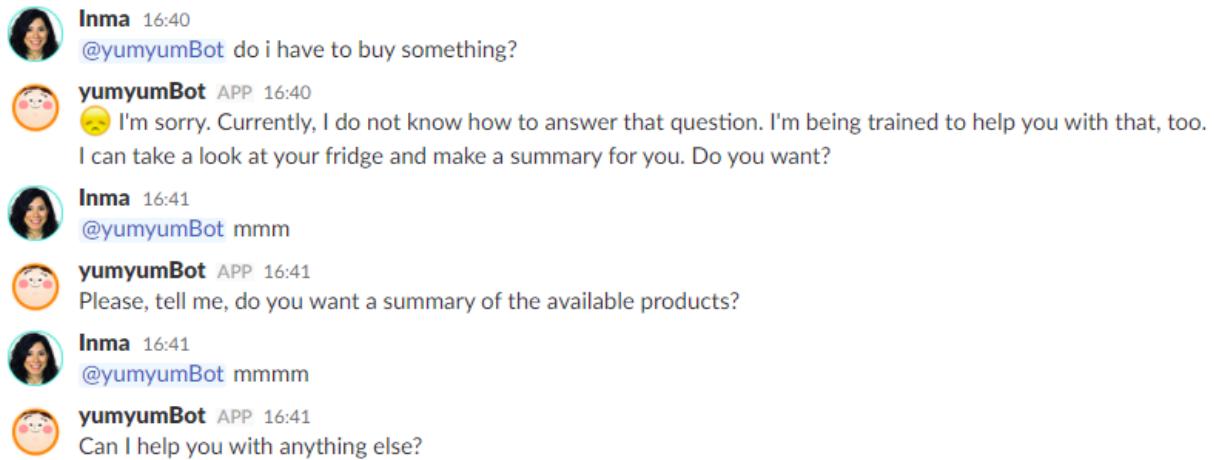


Figura 65 – Intención `#needed_ingredients`: primera respuesta no válida

- Si no es la primera vez que proporciona respuestas inválidas ($\$counter==1$) se finaliza la atención de esta intención preguntando al usuario si puede ayudarle en alguna otra cosa y se queda a la espera de respuesta por parte del usuario y se resetea el contador ($(\$counter=0)$).



El flujo del diálogo descrito anteriormente se traduce en el servicio Watson Conversation como sigue:



Figura 66 – Intención `#needed_ingredients`: flujo de la conversación en el servicio Watson Conversation

3.5.4.8 Intención #negative_reactions

El flujo de la conversación al detectar esta intención se ha diseñado para que se comporte del siguiente modo:

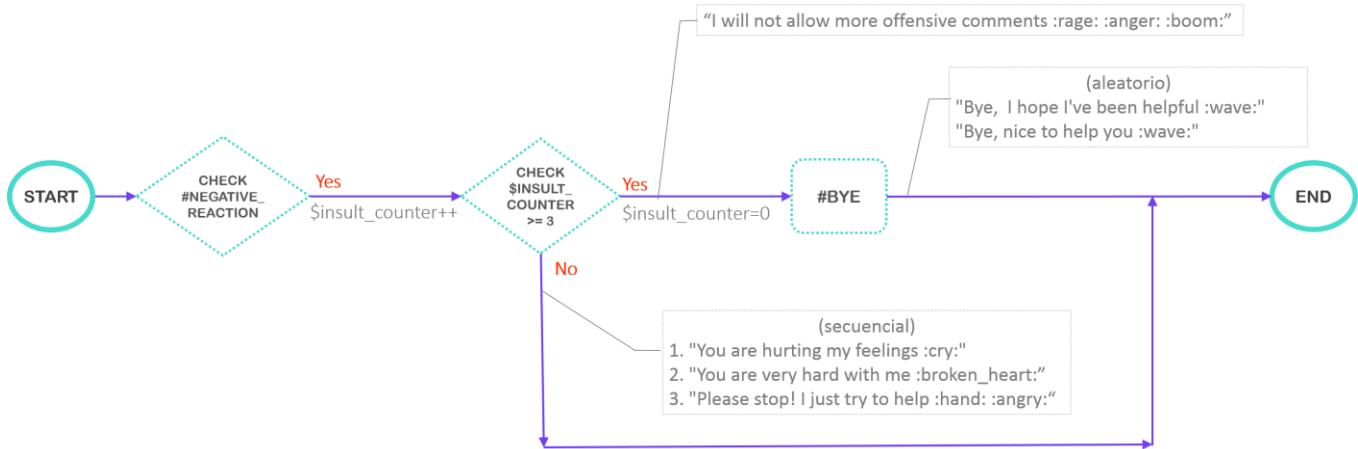


Figura 67 - Flujo de la conversación para la intención #negative_reactions

- 1) Se detecta que la petición del usuario corresponde a la intención “#negative_reaction”, es decir, el usuario expresa su descontento, es irrespetuoso o incluso insulta al chatbot. En ese caso se incrementa el contador de reacciones negativas (`$insult_counter++`)³³ para contabilizar cuántas veces se ha detectado esta intención dentro de la conversación.
- 2) Se evalúa el valor del contador de reacciones negativas para determinar cómo responder al usuario:
 - Si el contador es menor que 3, es decir, el chatbot ha recibido 3 reacciones negativas o menos (notar que el valor inicial del contador es igual a 0) por parte del usuario se responde con una de las 3 respuestas que se muestran en el diagrama de la Figura 67. Estas respuestas se van seleccionando en el mismo orden en el que se muestra³⁴. Una vez devuelve la respuesta correspondiente finaliza la atención de esta intención.

³³ Notar que se ha utilizado un contador específico (`$insult_counter`) y no el contador que utilizamos en el resto de intenciones para llevar el control de repeticiones de una determinada acción (`$counter`). Esto es así porque en el caso de la intención #negative_reaction nos interesa la suma global de reacciones negativas durante toda la conversación, y no sólo las repeticiones dentro de una misma intención. El contador `$counter` sin embargo se resetea cada vez que se detecta la intención en la que se utiliza o al salir de la condición que hace que se incremente. Y sin embargo, en el caso de la intención #negative_reaction si el usuario solicita otra funcionalidad (intención) este contador no debe resetearse, debe almacenarse el valor e incrementarse la siguiente vez que sea detectada la intención #negative_reaction.

³⁴ En el servicio de conversación de Watson cuando se proporciona una lista con varias respuestas posibles se puede especificar el orden en el que se deben mostrar al usuario. Existen dos posibilidades: secuencial, en cuyo caso se usan en el orden en el que se hayan introducido, y aleatorio, para seleccionar una de ellas de forma aleatoria.

 Inma 17:41 Hey! @yumyumBot, are you there?
 yumyumBot APP 17:41 Hello, I'm yumyumBot, can I help you?
 Inma 17:42 @yumyumBot let's try it, the last time you wasted my time
 yumyumBot APP 17:42 You are hurting my feelings 😢
 Inma 17:42 @yumyumBot come on, search for me the meatballs recipe
 yumyumBot APP 17:42 🌟 Good choice! I'll find the meatballs recipe for you, give me a second 🕒
 I found Albondigas Soup. To cook this you need the following ingredients:

- 2 Tbsp olive oil
- 1 large onion, chopped
- 1 large garlic clove, minced
- 3 quarts of chicken stock or beef stock OR water OR a mixture of both (we usually use half stock half water as the meatballs will create their own stock)*
- 1/2 cup of tomato sauce
- 1/2 lb of string beans, strings and ends removed, cut into 1 inch pieces
- 2 large carrots, peeled and sliced
- 1/3 cup of raw white rice
- 1 pound ground beef
- 1/4 cup of chopped fresh mint leaves
- 1/4 cup of chopped parsley
- 1 raw egg
- 1 1/2 teaspoon salt
- 1/4 teaspoon black pepper
- A dash of cayenne (optional)
- 1 1/2 cup of frozen or fresh peas
- 1 teaspoon of dried oregano, crumbled, or 1 Tbsp fresh chopped oregano
- Salt and pepper
- 1/2 cup chopped fresh cilantro

Here, you can find the method of cooking: http://www.simplyrecipes.com/recipes/albondigas_soup/

Figura 68 – Intención #negative_reaction: respuesta (contador <3)

- Si por el contrario el contador de reacciones negativas es mayor o igual a 3 esto quiere decir que el chatbot ha recibido más de 3 reacciones negativas por parte del usuario, en ese caso el chatbot avisa que se va a finalizar la conversación, y después salta al nodo de despedida (#goodbyes). Saltará a este nodo sin evaluar la condición, sólo para devolver la respuesta correspondiente. Una vez devuelta la respuesta, el chatbot quedará a la espera de otra entrada del usuario para detectar una nueva intención.

 Inma 17:43 @yumyumBot Do I have all the needed ingredients?
 yumyumBot APP 17:43 😢 I'm sorry. Currently, I do not know how to answer that question. I'm being trained to help you with that, too.
 I can take a look at your fridge and make a summary for you. Do you want?
 Inma 17:44 @yumyumBot no, i don't
 yumyumBot APP 17:44 Can I help you with anything else?
 Inma 17:44 @yumyumBot You're wasting my time with nonsense questions
 yumyumBot APP 17:44 You are very hard with me 💔
 Inma 17:45 @yumyumBot I speak to you in this way because my patience is running out
 yumyumBot APP 17:45 Please stop! I just try to help 🙏😢
 Inma 17:45 @yumyumBot you are not helping me, I think you are useless
 yumyumBot APP 17:45 I will not allow more offensive comments 😡😡😡
 Bye, I hope I've been helpful 🙏

Figura 69 – Intención #negative_reaction: respuesta (contador >3)

En el servicio Watson Conversation, el flujo del diálogo descrito anteriormente se traduce como sigue:

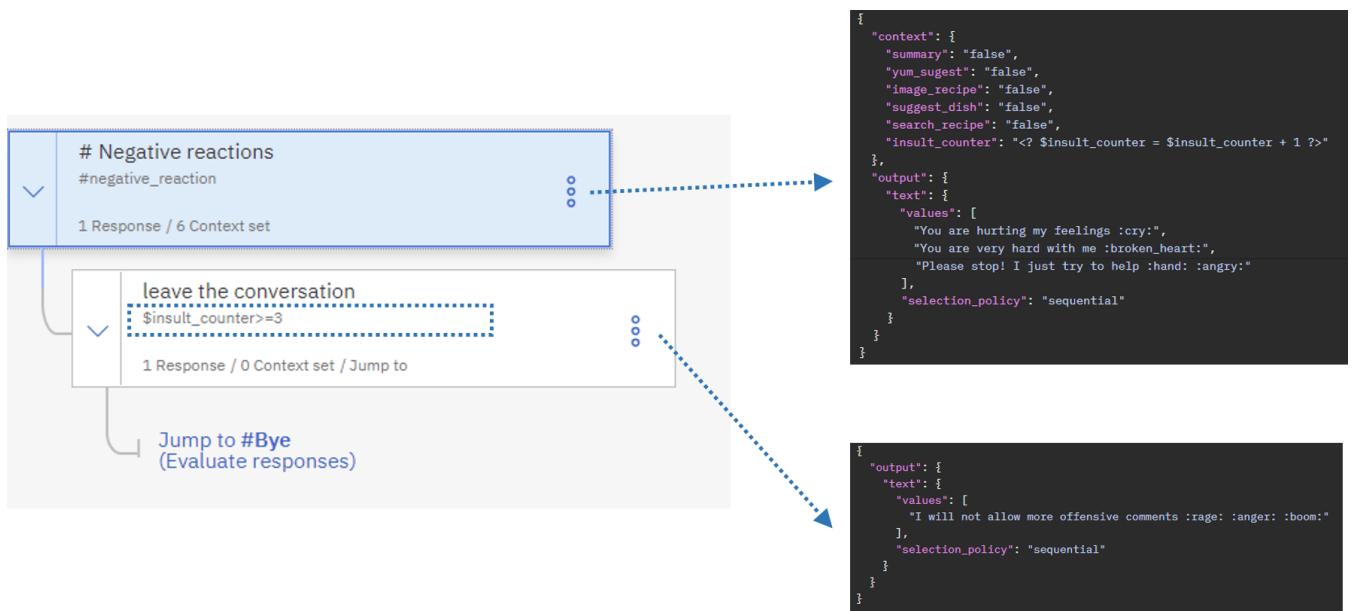


Figura 70 – Intención #negative_reaction: flujo de diálogo en el servicio Watson Conversation

3.5.4.9 Intención #available_ingredients

El flujo de la conversación para esta intención se ha diseñado para que se comporte del siguiente modo:

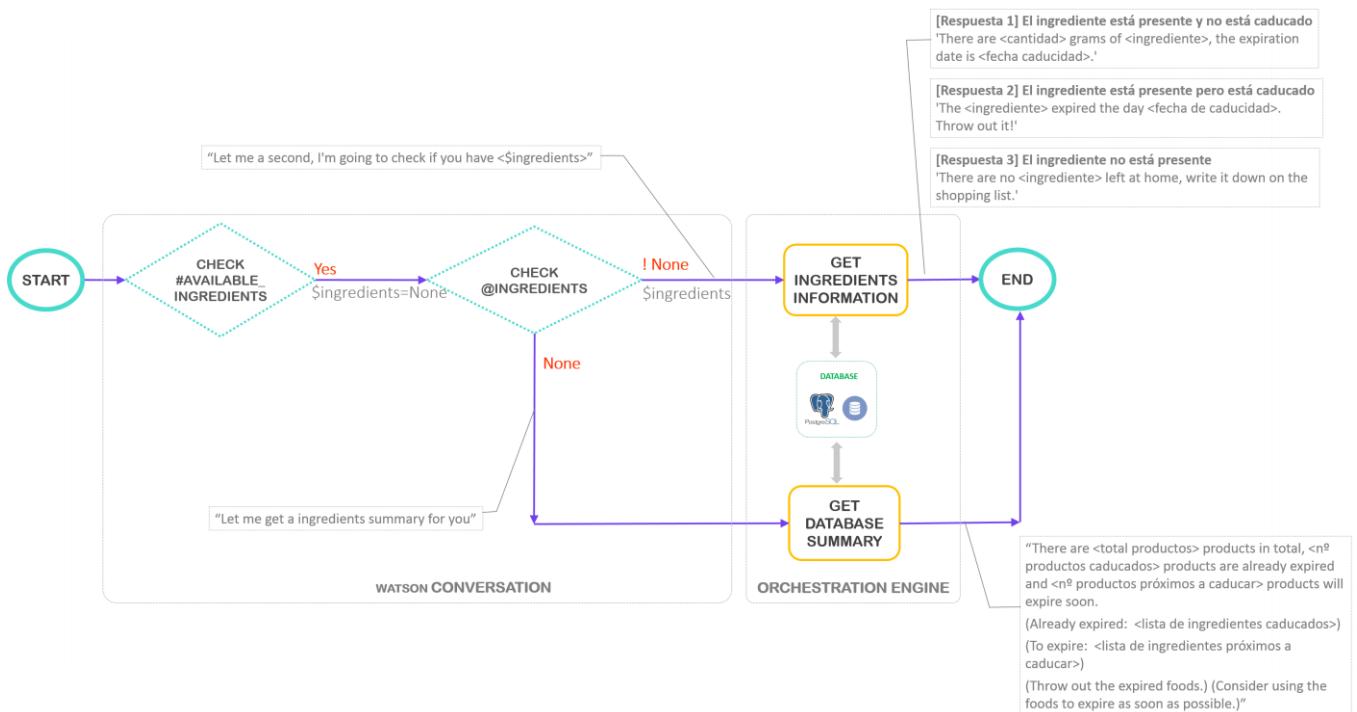


Figura 71 – Flujo de la conversación para la intención #available_ingredients

- 1) Se detecta la intención #available_ingredients, es decir, el usuario se interesa por el contenido de su frigorífico o pregunta por la existencia de un ingrediente en particular. Al satisfacer la condición (intención #available_ingredients) se limpia la variable de contexto \$ingredients para tomar el nuevo valor que el usuario, opcionalmente, podría proporcionar.

2) Se comprueba si la petición del usuario proporciona información relativa a un producto o ingrediente en concreto (@*ingredients*). Existen dos posibilidades:

- Se ha preguntado por un ingrediente en concreto (@*ingredients* != None):
 - i. El servicio de conversación responde al usuario con una frase para indicar que va a consultar la información del producto que ha solicitado ("Give me a second, I'm going to check if you have <\$*ingredients*>").
 - ii. La aplicación orquestadora consulta la base de datos y extrae toda la información referente a los ingredientes por los que se ha preguntado.
 - iii. La aplicación orquestadora construye la respuesta de forma dinámica utilizando la información que acaba de extraer, y la devuelve al usuario. Existen 3 tipos de respuestas en función de si el ingrediente por el que ha preguntado está presente o no, y en el caso de que esté presente la respuesta también varía en función de si está caducado o no.



Figura 72 – Intención #available_ingredients: respuestas a preguntas sobre un determinado producto

- El usuario no ha preguntado por un producto en concreto (@*ingredients* == None), por tanto, se ofrece un resumen de todos los productos disponibles:
 - i. El servicio de conversación responde al usuario con una frase para indicar que va a consultar la información disponible ("Let me get a ingredients summary for you").
 - ii. La aplicación orquestadora consulta en base de datos la información de todos los productos disponibles, junto con su fecha de caducidad.
 - iii. La aplicación orquestadora construye de forma dinámica la respuesta con la información extraída y responde al usuario.

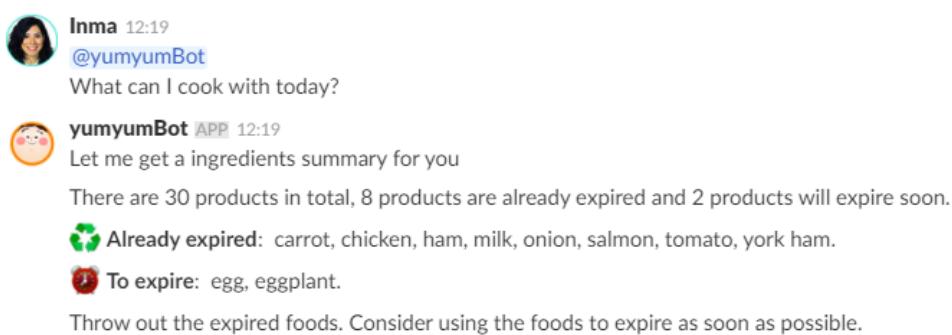


Figura 73 – Intención #available_ingredients: resumen del contenido

En la siguiente figura se muestra el flujo de la conversación tal y como se ha definido en la interfaz gráfica de Watson Conversation. En el nodo que se ha nombrado como “*particular ingredient*” se hace uso de los slots del servicio. En este nodo se comprueba la existencia de la entidad @*ingredients* que se corresponde con un slot opcional, y en el caso de que se encuentre en la petición del usuario se almacena su valor en la variable de contexto \$*ingredients*.

Check for	Save it as	If not present, ask	Type
1 @ingredients	\$ingredients	Enter a prompt	Optional

Figura 74 – Intención #available_ingredients: configuración slot @ingredients

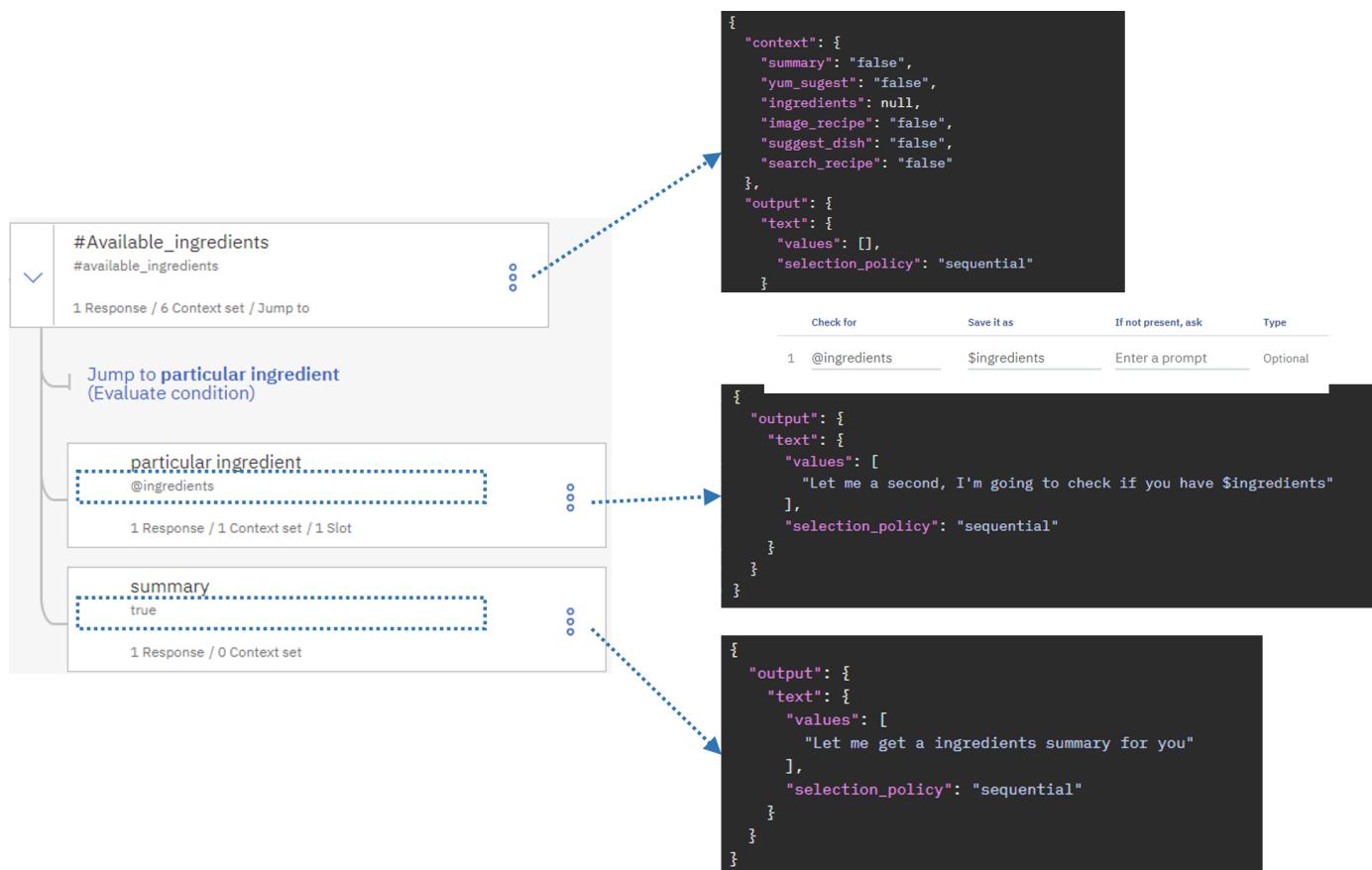


Figura 75 – Intención #available_ingredients: configuración servicio Watson Conversation

Notar que en la gestión de esta intención no es necesario utilizar la variable de contexto \$summary porque en la aplicación orquestadora se comprueba el valor de la variable de contexto \$ingredients para decidir si debe proporcionar un resumen general (\$ingredients==None) o un resumen de un ingrediente particular (\$ingredients!=None).

3.5.4.10 Intención #get_recipe

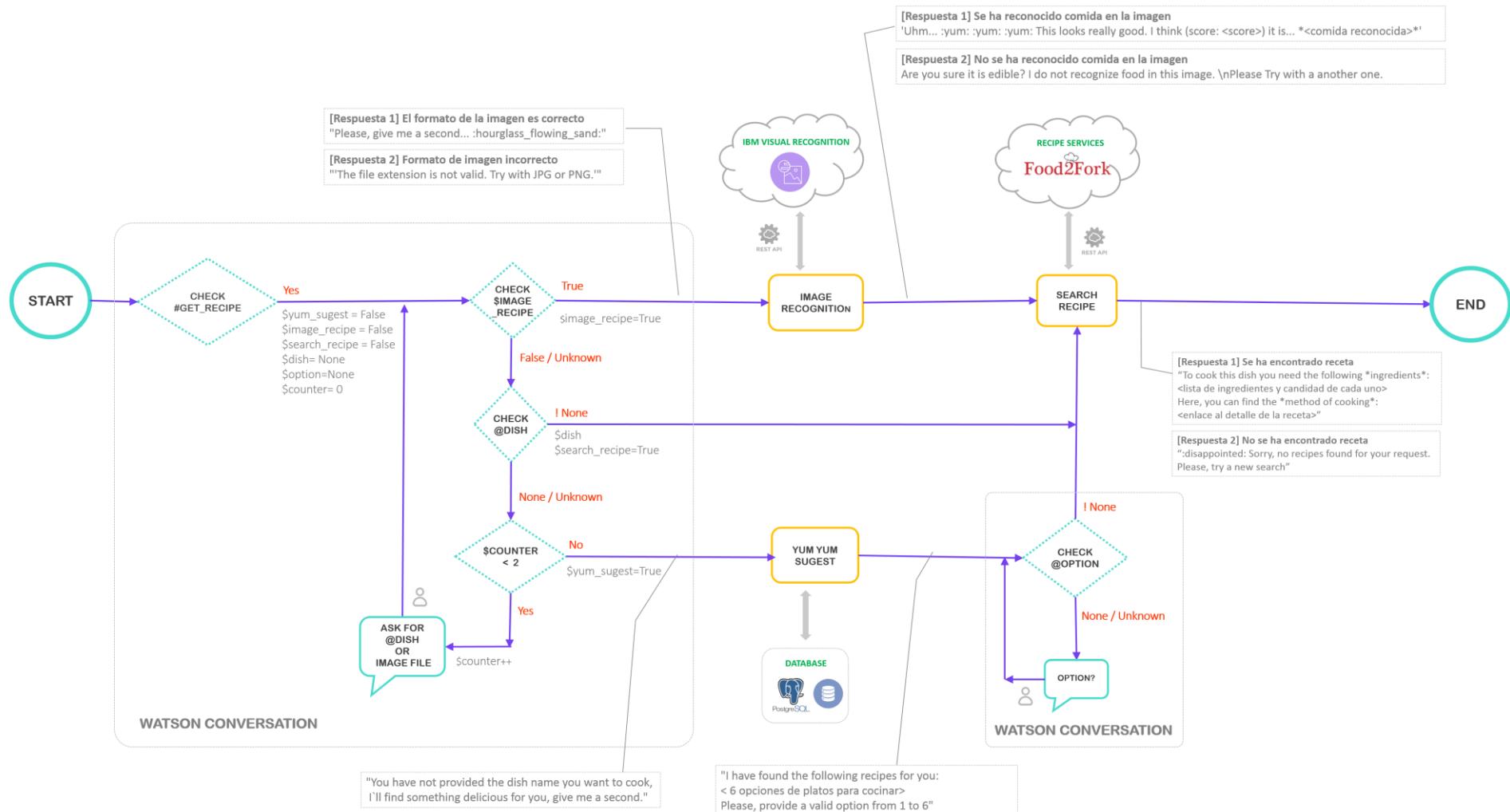


Figura 76 – Flujo de la conversación para la intención #get_recipe

El flujo de la conversación para esta intención se ha diseñado para que se comporte del siguiente modo:

- 1) Se reconoce la intención `#get_recipe`, es decir, el usuario quiere encontrar una receta (ingredientes y el método de cocinado).
- 2) Todas las variables de contexto que se utilizan en este flujo de conversación se limpian.
- 3) Se comprueba en primer lugar si en la petición se ha proporcionado una imagen de algún plato de comida (`$image_recipe`) para el que se está solicitando la receta. Existen dos opciones:
 - a. Sí se ha proporcionado una imagen. En ese caso, la aplicación orquestadora pone la variable de contexto `$image_recipe=True`, y se encarga de gestionar la conversación. Los pasos que sigue a continuación:
 - i. Comprueba si la imagen tiene una extensión válida³⁵. Si no es así, muestra un aviso al usuario (“*The file extension is not valid. Try with JPG or PNG.*”) y finaliza la atención de esta intención. Si por el contrario la imagen tiene un formato válido muestra un mensaje al usuario para indicar que se mantenga a la espera (“*Please, give me a second... :hourglass_flowing_sand:*”) y continúa con el resto de acciones que se listan a continuación.



Figura 77 - Intención `#get_recipe`: comprobación formato imagen

- ii. La aplicación orquestadora realiza una petición al servicio *Watson Visual Recognition*, envía como entrada la imagen que ha proporcionado el usuario y selecciona el modelo de comida que viene ya entrenado por defecto en este servicio. Si el servicio no reconoce comida en la imagen que recibe como entrada, se responde al usuario indicando que no se ha reconocido nada comestible, y que debe intentarlo con otra imagen. En caso contrario (sí se reconoce comida), se responde al usuario con el plato detectado y la probabilidad con la que se ha reconocido, después se continúa con el resto de acciones que se describen a continuación para obtener la receta.

³⁵ El servicio de reconocimiento de imágenes de IBM Watson solo soporta archivos PNG y JPG, la aplicación orquestadora comprueba que estas son las extensiones de los ficheros que proporciona el usuario como entrada.

IMAGEN CON COMIDA

 **Inma** 7:59 p. m.
ha compartido una imagen:
@yumyumBot look at this, tell me the
recipe



 **yumyumBot** APP 8:00 p. m.
Please, give me a second... 

Uhm... 😊 😊 😊 This looks really good. I think (score: 1) it is... pavlova

IMAGEN NO CONTIENE COMIDA

 **Inma** 8:19 p. m.
ha compartido una imagen:
@yumyumBot, tell me the recipe of this
photo



 **yumyumBot** APP 8:19 p. m.
Please, give me a second... 

Are you sure it is edible? I do not
recognize food in this image.
Please Try with a another one.

Figura 78 - Intención #get_recipe: respuestas tras petición servicio Visual Recognition

- iii. La aplicación orquestadora en este caso (se ha proporcionado una imagen en formato válido y que contiene comida) realiza una consulta al servicio *Food2Fork* enviando como entrada el nombre del plato que ha sido reconocido en el paso anterior para obtener todos los detalles de la receta (ingredientes y proceso de cocinado). Si se obtiene una receta válida se responde al usuario con la lista de ingredientes y un enlace al procedimiento de cocinado. Si no se obtiene receta válida se contesta igualmente al usuario indicando que no se han encontrado recetas. En ambos casos (con receta válida o no) se finaliza la atención de esta intención y el asistente se queda a la espera de una nueva entrada por parte del usuario.

 **yumyumBot** APP 15:52
Please, give me a second... 

Uhm... 😊 😊 😊 This looks really good. I think (score: 1) it is... pavlova

To cook this dish you need the following ingredients:

- 4 egg whites
- 225g golden caster sugar
- 1 tsp cornflour
- 1 tsp white wine vinegar
- vanilla pod
- 284ml carton double cream
- 1 lemon , zested
- 450g berries , raspberries, strawberries, or blackberries or a mixture
- 1 tbsp icing sugar

Here, you can find the **method of cooking**: <http://www.bbcgoodfood.com/recipes/4258/pavlova>

Figura 79 - Intención #get_recipe: receta a partir de una imagen

- b. No se ha proporcionado imagen, entonces se comprueba si se ha proporcionado el nombre del plato (@dish) para el que se busca la receta. Existen 2 posibilidades también en este paso:
 - i. Sí se ha proporcionado un nombre de un plato. En ese caso el servicio Watson

Conversation modifica el valor de dos variables de contexto, `$search_recipe=True`, y en `$dish` guarda el literal³⁶ del plato que ha introducido el usuario. Con esta información y la variable `search_recipe` a `True`, la aplicación orquestadora entiende que debe buscar la receta a partir de un nombre de plato, y se encarga de realizar la llamada correspondiente al servicio `Food2Fork`. Esta llamada se realiza del mismo modo que en el paso anteriormente descrito (a.iii), enviando como entrada el nombre del plato (`$dish`). Finalmente se responde al usuario con la receta si se encontró, o se informa de que no se ha encontrado receta.

Inma 18:12
@yumyumBot, please, tell me the tabbouleh recipe

yumyumBot APP 18:12
Good choice! I'll find the tabbouleh recipe for you, give me a second

To cook this dish you need the following ingredients:

- 1 cup quinoa, rinsed well
- 1/2 teaspoon kosher salt plus more
- 2 tablespoons fresh lemon juice
- 1 garlic clove, minced
- 1/2 cup extra-virgin olive oil
- Freshly ground black pepper
- 1 large English hothouse cucumber or 2 Persian cucumbers, cut into 1/4-inch pieces
- 1 pint cherry tomatoes, halved
- 2/3 cup chopped flat-leaf parsley
- 1/2 cup chopped fresh mint
- 2 scallions, thinly sliced

Here, you can find the method of cooking: <http://www.bonappetit.com/recipes/2012/06/quinoa-tabbouleh>

ba Bon Appétit
Quinoa Tabbouleh Recipe
The classic Middle Eastern Tabbouleh salad gets an of-the-moment upgrade with the (158 kB) ▾

Figura 80 – Intención `#get_recipe`: receta a partir del nombre del plato

Sin embargo, la respuesta cuando no encuentra la receta en el servicio de recetas será la que se muestra a continuación:

Inma 16:31
@yumyumBot how can I cook the fabada?

yumyumBot APP 16:31
Sorry, no recipes found for your request. Please, try a new search

Figura 81 – Intención `#get_recipe`: respuesta si no encuentra recetas para la entrada del usuario

³⁶ Se guarda el literal porque puede ocurrir que el plato forme parte de una super clase, por ejemplo, la superclase podría ser “legumbres”, y el plato proporcionado podría ser “lentejas”, sinónimo de “legumbres”. A la hora de buscar la receta debemos buscar el literal, es decir “lentejas” y no “legumbres”

- ii. *No se ha proporcionado un nombre de plato válido.* El servicio de conversación en este caso pedirá al usuario la información que necesita (nombre del plato o imagen). Si no se obtiene información válida se repite la petición nuevamente una última vez. Si el usuario proporciona información válida se actúa del modo ya descrito en los puntos anteriores (“a” si proporciona imagen, “b.i” si se proporciona nombre de plato).

Inma 16:53
@yumyumBot, please, look for the recipe

yumyumBot APP 16:53
I can take your recipe request. Please, tell me the dish name or show me a picture of the dish you want

Inma 16:54
@yumyumBot, I meant the chocolate cake recipe

yumyumBot APP 16:54
Good choice! I'll find the chocolate cake recipe for you, give me a second

To cook this dish you need the following ingredients:

- 2 cups sugar
- 1 3/4 cups all-purpose flour
- 3/4 cup unsweetened cocoa powder
- 2 tablespoons King Arthur Flour Black Cocoa, optional
- 2 teaspoons baking soda
- 1 teaspoon baking powder
- 1 teaspoon kosher salt
- 2 eggs
- 1 cup buttermilk
- 1 cup strong black coffee (I used Green Mountain Coffees Vermont Country Blend)
- 1/2 cup vegetable oil

Here, you can find the method of cooking: <http://www.mybakingaddiction.com/the-best-chocolate-cake-recipe/>

My Baking Addiction
The Best Chocolate Cake | My Baking Addiction
A rich chocolate cake infused with cocoa and coffee creating a chocolate lover's dream come true.
12 oct. 2011 (50 KB) ▾

Figura 82 – Intención #get_recipe: solicitud del nombre de la receta o imagen

Si, por el contrario, después de 2 intentos no obtenemos la información necesaria el asistente realiza una sugerencia propia. Dará a elegir al usuario entre 6 platos diferentes: 2 de ellos utilizando los ingredientes más próximos a caducar disponibles en casa, otros 2 platos con las recetas más puntuadas en el servicio de recetas, y otros 2 con las recetas que son tendencia en el servicio de recetas. Estas 6 opciones son presentadas al usuario y se le pide que elija una de ellas. Una vez el usuario selecciona una de ellas, el asistente busca la receta correspondiente y responde al usuario con todos los detalles (ingredientes y procedimiento de cocinado).

Figura 83 – Intención #get_recipe: sugerencia propia por falta de entrada del usuario

A continuación se muestra la configuración de esta intención en el servicio Watson Conversation junto con una breve explicación de las acciones realizadas por cada nodo:

- Nodo "check \$image_recipe": se activa si en la petición hay una imagen, en ese caso la variable \$image_recipe se pone a True.

Inma 17:01
@yumyumBot I want to get kssls recipe

yumyumBot APP 17:01
I can take your recipe request. Please, tell me the dish name or show me a picture of the dish you want

Inma 17:01
@yumyumBot c

yumyumBot APP 17:01
Try again, please provide the dish to get the recipe

Inma 17:02
@yumyumBot mmmm

yumyumBot APP 17:02
You have not provided the dish name you want to cook, I'll find something delicious for you, give me a second.
I have found the following recipes for you:
[1] : Smashed Chickpea & Avocado Salad Sandwich
[2] : Leftover Easter Sandwich
[3] : Jalapeno Popper Grilled Cheese Sandwich
[4] : Perfect Iced Coffee
[5] : No-Bake Chocolate Peanut Butter Pretzel Cookies
[6] : All Recipes
Please, provide a valid option from 1 to 6

Inma 17:03
@yumyumBot, number 3 sounds good

yumyumBot APP 17:03
Ok, good choice! The Jalapeno Popper Grilled Cheese Sandwich recipe below:
To cook this dish you need the following ingredients:
- 2 jalapeno peppers, cut in half lengthwise and seeded
- 2 slices sour dough bread
- 1 tablespoon butter, room temperature
- 2 tablespoons cream cheese, room temperature
- 1/2 cup jack and cheddar cheese, shredded
- 1 tablespoon tortilla chips, crumbled

Here, you can find the method of cooking: <http://www.closetcooking.com/2011/04/jalapeno-popper-grilled-cheese-sandwich.html>

- Nodo "*check @dish in the request*": se activa si hay un nombre de plato válido en la petición, en cuyo caso se activa la variable \$search_recipe.
- Nodo "*ask for @dish name or photo*": se activa si las condiciones de los nodos anteriores no se han satisfecho. Es decir, en la petición no se incluía ni imagen ni nombre de plato.

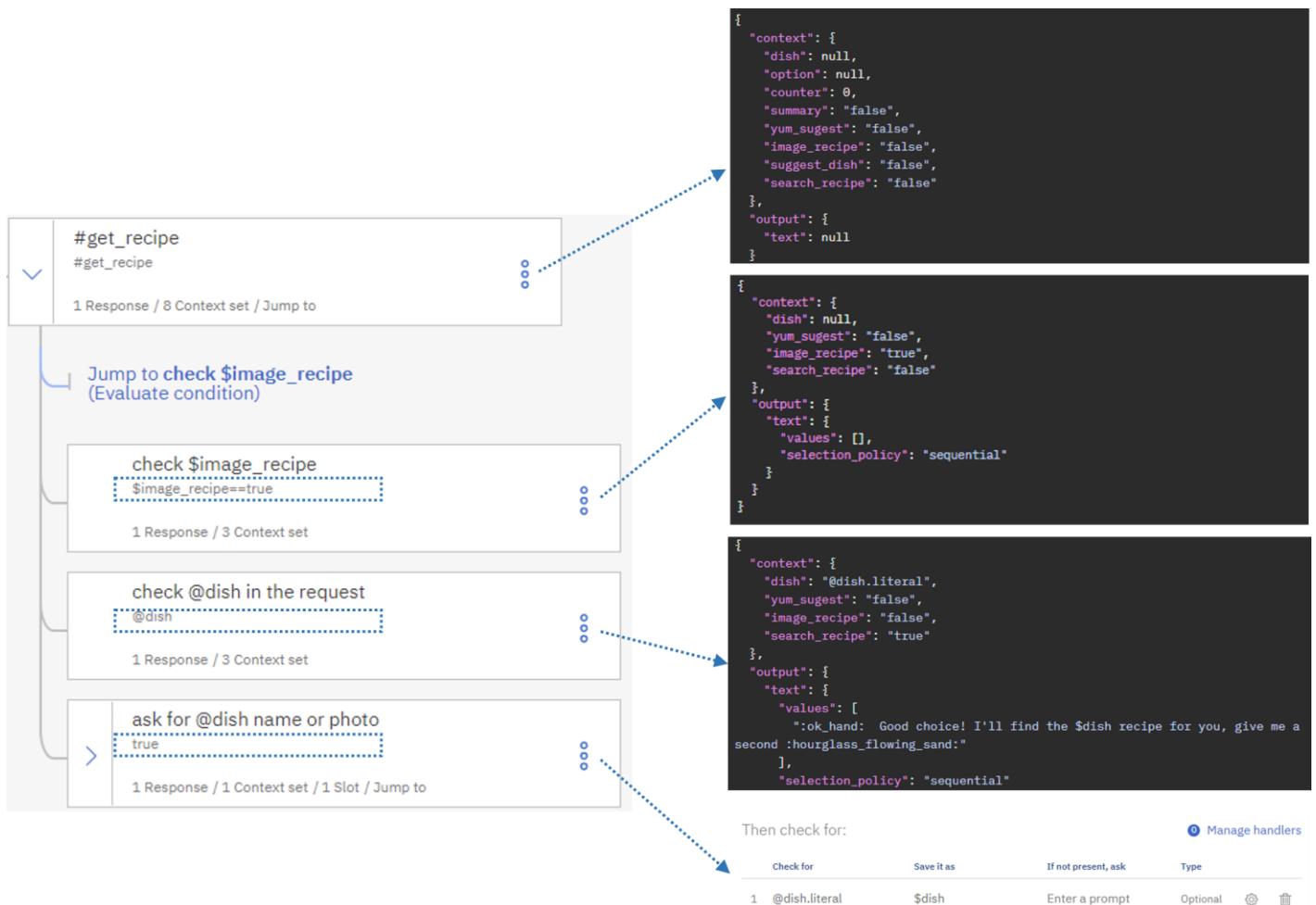


Figura 84 – Intención #get_recipe: check @dish y check \$image

A su vez, el nodo “*ask for @dish name or photo*” tiene el siguiente flujo y los siguientes nodos hijos:

- Nodo “*No dish or image provided (counter==0)*”: se activa si no se ha proporcionado imagen o nombre de plato. En caso de satisfacer las condiciones se solicita la información necesaria al usuario, se aumenta el contador, y se salta al nodo que evalua las entradas (*check \$image_recipe*).
- Nodo “*No dish or image provided (counter==1)*”: se activa si no se ha proporcionado imagen o nombre de plato y ya ha sido solicitado una vez. En caso de satisfacer la condición se solicita una última vez la información al usuario, se aumenta el contador, y se salta al nodo que evalua las entradas del usuario.
- Nodo “*make suggestion (counter>=2)*”: se activa si después de dos intentos (*counter>=2*) la información de entrada no es válida. En ese caso se pone la variable de contexto *\$yum_sugest=True* y se responde al usuario diciendo que se va a hacer una sugerencia propia.

Notar que en los nodos hijos de este nodo se distingue entre valores del contador 0 y 1 porque la respuesta en cada caso es diferente, en el primer caso es “*I can take your recipe request. Please, tell me the dish name or show me a picture of the dish you want*”, mientras que en el segundo intent la respuesta es “*Try again, please provide the dish to get the recipe*”.

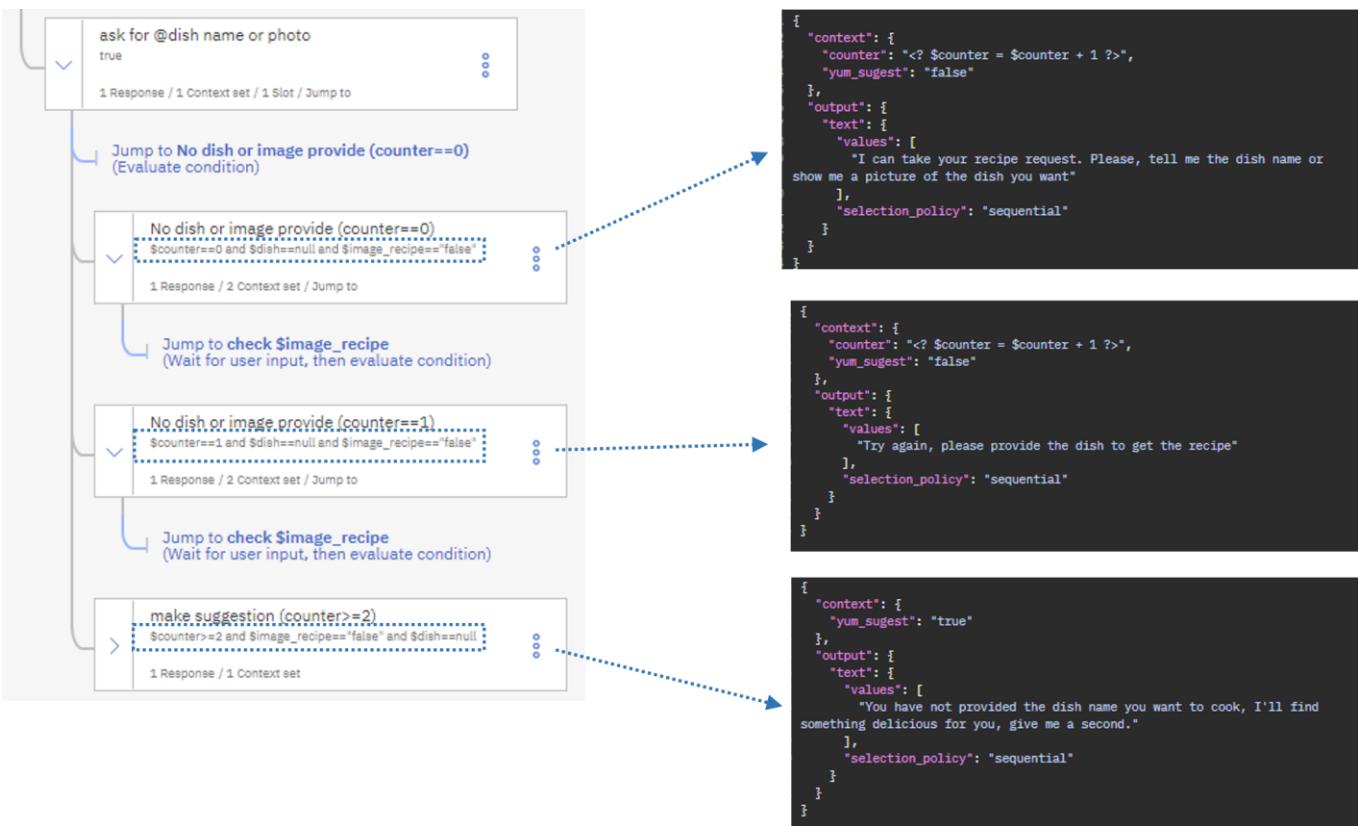


Figura 85 – Intención #get_recipe: nodo “ask for @dish name or photo”

Y finalmente, a su vez el nodo “*make suggestion (counter>=2)*” se encargará de proporcionar una sugerencia propia al usuario sin necesidad de entrada por su parte, para más tarde capturar la opción favorita del usuario. Este nodo ha sido definido como sigue en el servicio Watson Conversation. Este nodo consta de 2 nodos hijos:

- Nodo “*check option*”: se ha definido la entidad @option como obligatoria, y no sale del nodo hasta que no se proporciona una respuesta válida. Como mejora habría que utilizar un contador para forzar a salir tras un número de intentos máximo. Al salir de este nodo se salta al nodo “*reset counter*”.
- Nodo “*reset counter*”: se activa siempre (condición es igual a True) y se ocupa de resetear la variable \$counter. Este nodo también se podría haber creado como hijo de “*check option*”.

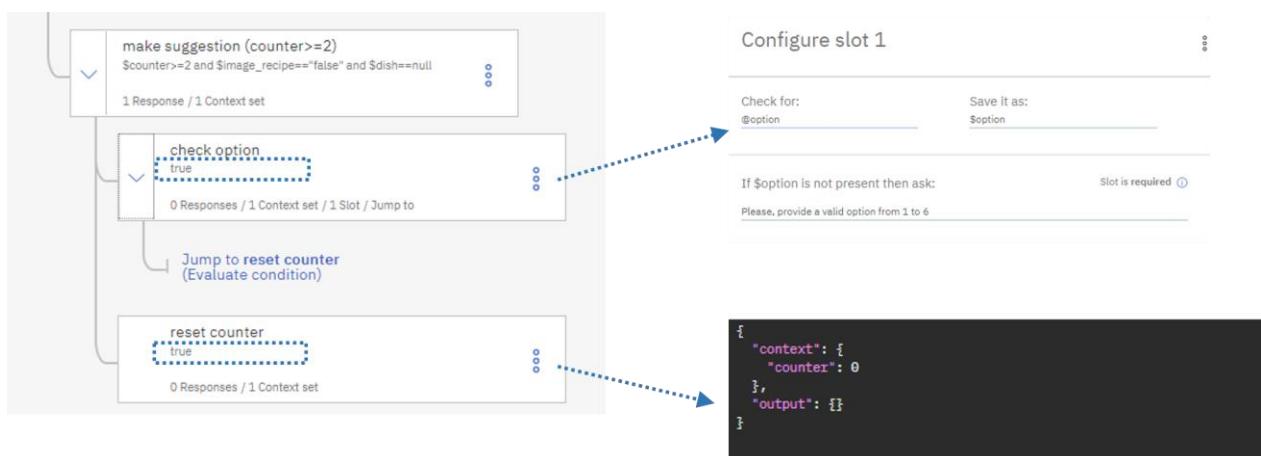


Figura 86 – Intención #get_recipe: último nodo de la conversación

3.5.4.11 Intención #suggest_dish

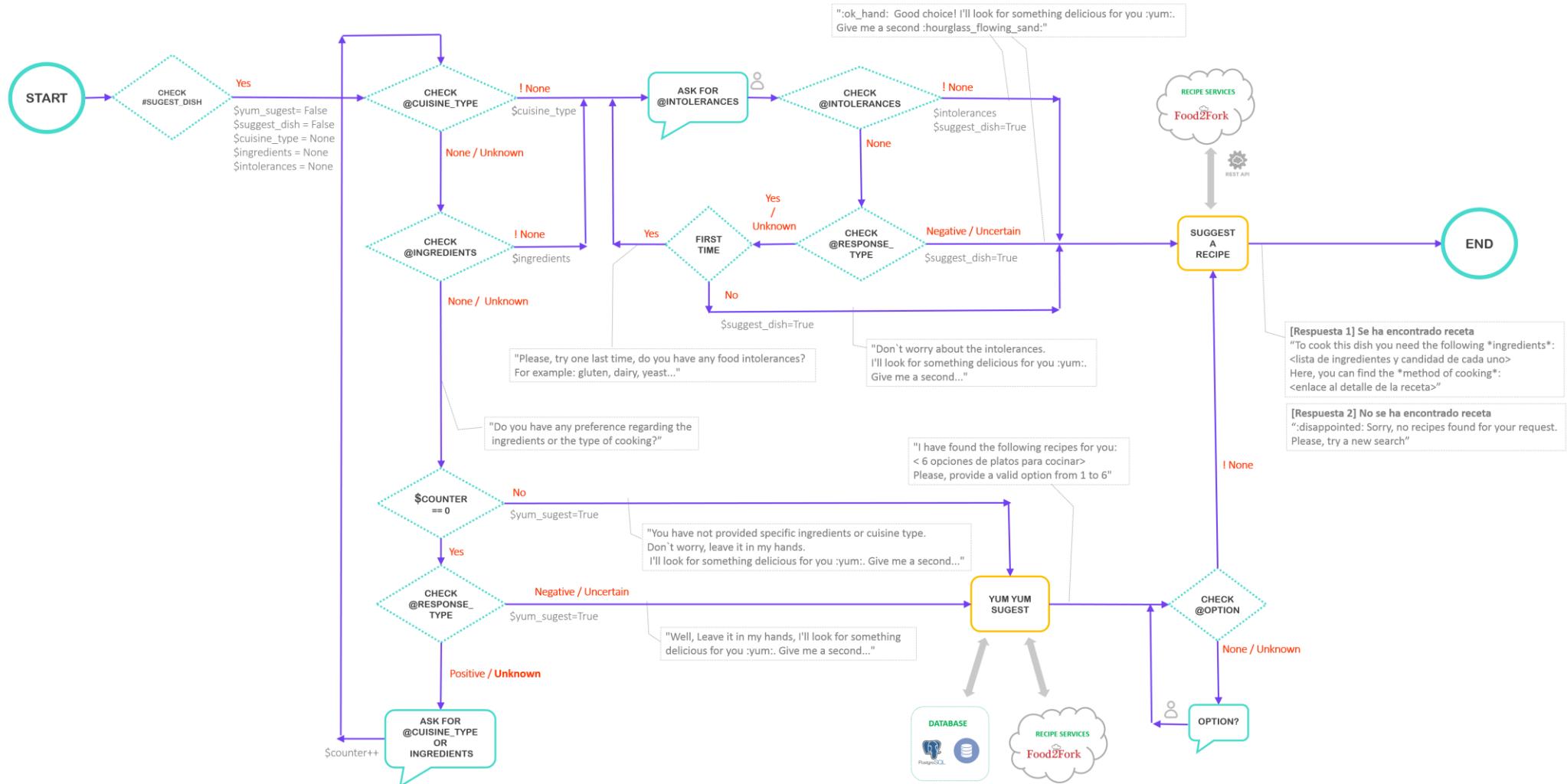


Figura 87 – Flujo de la conversación para la intención #suggest_dish

El flujo de la conversación para esta intención se ha diseñado para que se comporte del siguiente modo:

- 1) Se detecta la intención `#suggest_dish`, es decir, el usuario está solicitando una sugerencia de recetas al chatbot. En este caso, se resetean las variables que activan acciones en la aplicación orquestadora y todas las que son utilizadas en esta intención.
- 2) Se comprueba si la petición del usuario contiene información sobre ingredientes concretos o sobre un cierto tipo de cocina. Existen 2 opciones posibles en este punto de la conversación:
 - a. La petición contiene información sobre un determinado tipo de cocina (`@cuisine_type`) o ingredientes (`@ingredients`), en cuyo caso se realizan las siguientes acciones:
 - i. Se almacena el valor proporcionado en la variable de contexto correspondiente (`$cuisine_type` o `$ingredients`). Puede ocurrir que el usuario proporcione ambos valores, en ese caso se almacenara tanto tipo de cocina como ingredientes.
 - ii. El chatbot pregunta si el usuario tiene intolerancias alimentarias y evalúa la respuesta.
 1. Si el usuario responde con una intolerancia válida se almacena en la variable de contexto `$intolerances`, y además se pone la variable `$sugest_dish` a True para indicar a la aplicación orquestadora que debe buscar recetas con el tipo de cocina, ingredientes e intolerancias proporcionados. En el siguiente ejemplo no se han encontrado recetas por lo restringido de la búsqueda, también se le indicaría al usuario.

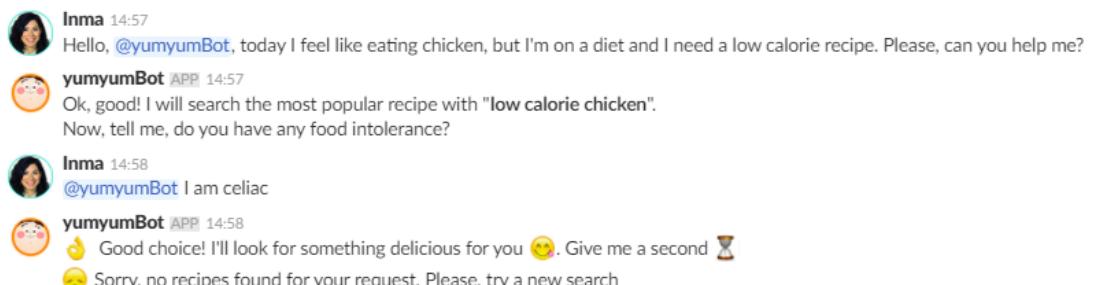


Figura 88 – Intención `#sugets_dish`: intoleracia válida

2. Si el usuario responde afirmativamente pero no proporciona intolerancia válida, o si proporciona una respuesta desconocida, el chatbot vuelve a solicitar la información de intolerancias una última vez, junto con un ejemplo para facilitar la tarea al usuario.

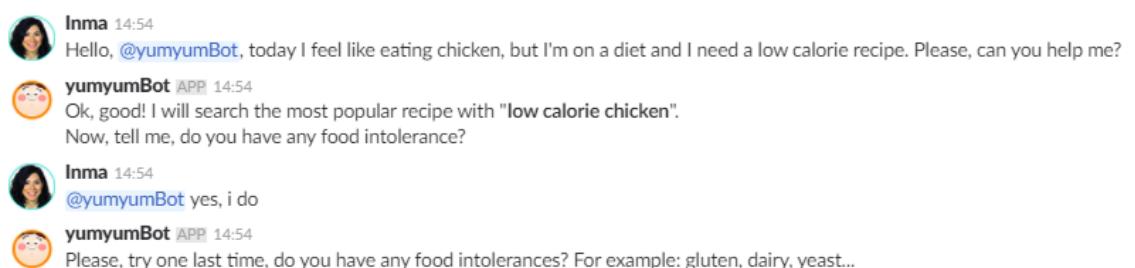


Figura 89 – Intención `#sugest_dish`: respuesta afirmativa intolerancias

3. Si el usuario responde negativamente, o no precisa respuesta, o es la segunda vez que se le ha preguntado por las intolerancias, directamente se activa la variable `$sugest_dish` para que la aplicación orquestadora se ocupe de buscar recetas con los ingredientes y tipo de cocina proporcionados, esta vez `$intolerances` será `None`.

Inma 14:42 Hello, @yumyumBot, today I feel like eating chicken, but I'm on a diet and I need a low calorie recipe. Please, can you help me?

yumyumBot APP 14:42 Ok, good! I will search the most popular recipe with "low calorie chicken". Now, tell me, do you have any food intolerance?

Inma 14:42 @yumyumBot no, i don't

yumyumBot APP 14:42 🍽️ Good choice! I'll look for something delicious for you 😊. Give me a second 🕒

I found Low-Calorie Teriyaki Chicken and Vegetables. To cook this you need the following ingredients:

- cups cups
- 1 Tablespoon 1 Tablespoon
- 2 teaspoons 2 teaspoons
- cups cups
- 8 pieces 8 pieces
- 4 whole 4 whole
- 4 whole 4 whole
- 1 whole 1 whole

Here, you can find the method of cooking: <http://tastykitchen.com/recipes/main-courses/low-calorie-teriyaki-chicken-and-vegetables/>



Figura 90 – Intención #sugest_dish: ingredientes y/o tipo de cocina proporcionados (sin intolerancias)

- iii. La aplicación orquestadora, en función de la información de entrada, realiza la búsqueda en el servicio *Food2Fork* y construye la respuesta. Esta respuesta se envía al usuario, y se finaliza la atención de la intención. Si la búsqueda del usuario no da resultados se indica también y se recomienda que pruebe con otra búsqueda.
- b. La petición no contiene información sobre tipo de cocina (@cuisine_type) ni ingredientes (@ingredients). En este caso, el servicio de conversación pregunta si tiene preferencias con respecto a ingredientes o tipo de cocina, y luego evalúa la respuesta.

Inma 15:05 Hello @yumyumBot, tell me what to cook today

yumyumBot APP 15:05 🍽️ Do you have any preference regarding the ingredients or the type of cooking?

Figura 91 – Intención #sugest_dish: petición no contiene tipo de cocina ni ingredientes

En función de la respuesta proporcionada por el usuario la conversación tomará caminos distintos. Existen 3 posibilidades:

- i. El usuario responde proporcionando el tipo de cocina o ingredientes válidos. En ese caso se procede del mismo modo que ya se explicó en el punto a.

 **Inma** 15:05
 Hello @yumyumBot, tell me what to cook today

 **yumyumBot** APP 15:05
 Do you have any preference regarding the ingredients or the type of cooking?

 **Inma** 15:07
 @yumyumBot yes, I like mexican food

 **yumyumBot** APP 15:07
 I have catched the cuisine type chosen is mexican. Now, tell me, do you have any food intolerance?

 **Inma** 15:08
 @yumyumBot no

 **yumyumBot** APP 15:08
 🌮 Good choice! I'll look for something delicious for you 😊. Give me a second ⏳
 I found Mexican Baked Eggs. To cook this you need the following ingredients:

- 1 tablespoon corn oil
- 1 small onion, diced
- 2 cloves garlic, chopped
- 1 teaspoon cumin, toasted and ground
- 1 teaspoon chipotle chili powder
- 2 jalapeno peppers, diced
- 1 (28 ounce) can diced tomatoes or 4 cups diced fresh tomatoes
- 1 (19 ounce) can black beans, rinsed and drained
- 1 teaspoon oregano
- salt and pepper to taste
- 1 handful cilantro, chopped
- 4 eggs
- 1/2 cup grated cheese such as jack and cheddar

Here, you can find the method of cooking: <http://www.closetcooking.com/2011/09/mexican-baked-eggs.html>

Figura 92 – Intención #sugest_dish: el usuario responde proporcionando tipo de cocina

- ii. El usuario responde negativamente (no, no tengo preferencias, etc.), o responde de forma incierta (no sé, no lo tengo claro, etc.). En ambos casos se pone a *True* la variable de contexto *\$yum_sugget* para que la aplicación orquestadora busque sugerencias libres, es decir, sin restringirse a ciertos ingredientes o tipo de cocina. La aplicación orquestadora proporcionará 6 opciones de platos (2 con los ingredientes disponibles más próximos a caducar, 2 con las recetas más votadas y otras 2 con las recetas más populares en la actualidad) y luego pide al usuario que seleccione una de ellas. Una vez el usuario selecciona una opción válida, la aplicación orquestadora realiza la búsqueda de la receta para el plato seleccionado y compone la respuesta para enviársela al usuario. En ese momento finaliza la atención de esta intención.

 **Inma** 15:12
 Hello @yumyumBot, tell me what to cook today
 **yumyumBot APP** 15:12
 Do you have any preference regarding the ingredients or the type of cooking?
 **Inma** 15:12
 @yumyumBot no idea
 **yumyumBot APP** 15:12
 Well, Leave it in my hands, I'll look for something delicious for you 😊. Give me a second...
 I have found the following recipes for you:
 [1] : Smashed Chickpea & Avocado Salad Sandwich
 [2] : Leftover Easter Sandwich
 [3] : Perfect Iced Coffee
 [4] : Jalapeno Popper Grilled Cheese Sandwich
 [5] : No-Bake Chocolate Peanut Butter Pretzel Cookies
 [6] : All Recipes
 Please, provide a valid option from 1 to 6
 **Inma** 15:13
 @yumyumBot number 2 of course
 **yumyumBot APP** 15:14
 Ok, good choice! The Leftover Easter Sandwich recipe below:
 I found **Leftover Easter Sandwich**. To cook this you need the following **ingredients**:
 - 12 whole Hard Boiled Eggs
 - 1/2 cup Mayonnaise
 - 3 Tablespoons Grainy Dijon Mustard
 - Salt And Pepper, to taste
 - Several Dashes Worcestershire Sauce
 - Leftover Baked Ham, Sliced
 - Kaiser Rolls Or Other Bread
 - Extra Mayonnaise And Dijon, For Spreading
 - Swiss Cheese Or Other Cheese Slices
 - Thinly Sliced Red Onion
 - Avocado Slices
 - Sliced Tomatoes
 - Lettuce, Spinach, Or Arugula
 Here, you can find the **method of cooking**: <http://thepioneerwoman.com/cooking/2013/04/easter-leftover-sandwich/>

Figura 93 – Intención #sugest_dish: sin preferencias respecto a ingredientes o tipo de cocina

- iii. El usuario responde con una afirmación pero sin proporcionar el tipo de cocina o ingredientes, o no es posible interpretar la respuesta, en este caso, se vuelve a solicitar la información de ingredientes o tipo de cocina una última vez. Se aumenta el contador, y se vuelve a evaluar la nueva respuesta. Si la nueva respuesta contiene ingredientes o tipo de cocina válidos, se procede como se ha explicado en el punto a. Si no se proporciona ingredientes o tipo de cocina ya no se pregunta más, se ofrece una sugerencia libre con 6 opciones del mismo modo que en el punto descrito anteriormente (ii).

 **Inma** 15:17
 @yumyumBot, what do I eat today?
 **yumyumBot APP** 15:17
 Do you have any preference regarding the ingredients or the type of cooking?
 **Inma** 15:18
 @yumyumBot yes
 **yumyumBot APP** 15:18
 Please, provide the type of cooking (i.e. mexican, french, etc.) or the ingredients to use.

Figura 94 – Intención #sugest_dish: respuesta afirmativa sobre preferencias ingredientes/tipo de cocina

A continuación se muestra la configuración del flujo que se acaba de describir en la interfaz de Watson Conversation (ver Figura 95)

- Nodo "check ingredients / cuisine type (included request)": utiliza los slots del servicio Watson conversation para capturar las entidades @cuisine_type y/o @ingredients, y si están presentes los

almacena en la variable de contexto correspondiente y responde informando de que los ha encontrado.

- Nodo "*Cuisine type or ingredients found*": evalúa si alguna de las variables de contexto `$ingredients` o `$cuisine_type` contiene información. Si satisface la condición pregunta por las intolerancias.
- Nodo "*ask for the @ingredients or @cuisine_type*": los slots no están llenos, se pregunta al usuario por dicha información y se evalúa la nueva respuesta.

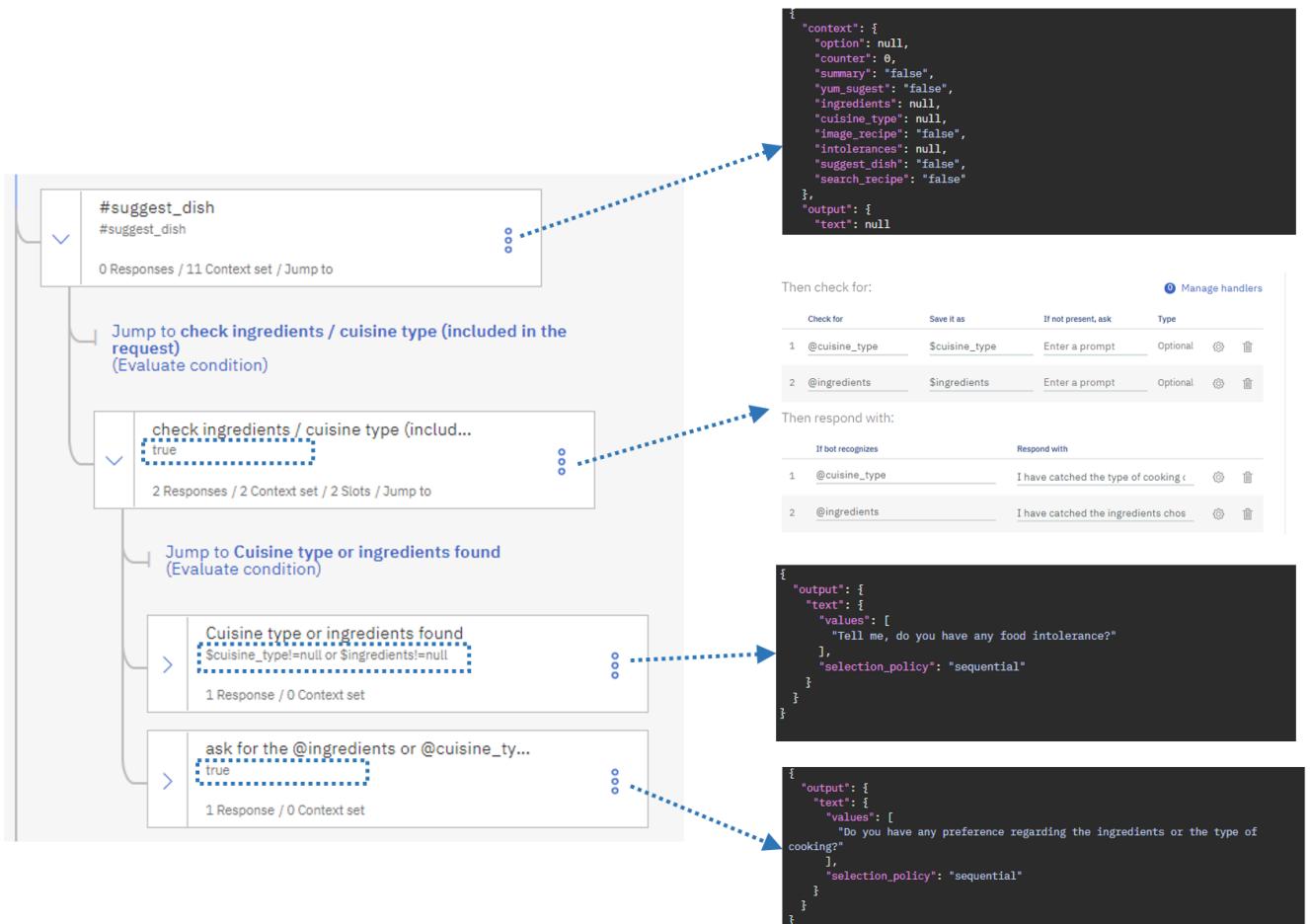


Figura 95 - Intención `#sugest_dish`: nodos de la conversación

A continuación veremos la configuración de cada uno de los 2 nodos anteriores en más detalle.

Comenzaremos con el nodo "*Cuisine type or ingredients found*". Este nodo tiene los siguientes nodos hijos (ver Figura 96):

- Nodo "*intolerances found*": es el primero en evaluarse, y comprueba si la respuesta del usuario contiene un valor de `@intolerances` válido. Si es así, responde al usuario diciendo que va a buscar una receta con la información proporcionada, y pone la variable `$sugest_dish` a `True` para indicar a la aplicación orquestadora que debe buscar la receta con la información de entrada (ingredientes, tipo de cocina e intolerancias).
- Nodo "*negative / uncertain response*": este nodo evalúa si la respuesta del usuario es negativa, o incierta, es decir, el usuario responde que no tiene intolerancias o que lo desconoce. Si se satisface la condición, se responde al usuario y se pone la variable `$sugest_dish` a `True`.
- Nodo "*Unknown response (first time)*": en este nodo entra si la respuesta es desconocida o afirmativa (notar que el resto de valores posibles de la entidad `@response_type` ya han sido evaluados en los

nodos previos) y además es la primera vez que se ha preguntado por las intolerancias. Si se cumple todo lo anterior se aumenta el contador, se pregunta una última vez al usuario por las intolerancias, y se salta al nodo que evalúa la respuesta.

- Nodo “*make a suggestion*”: a este nodo se entra si ya se ha preguntado más de una vez por intolerancias sin éxito. En ese caso se responde al usuario diciendo que se va a proporcionar una receta, y se activa la variable `$sugest_dish` para indicar a la aplicación orquestadora que debe buscar la receta con las entradas que no estén a *None* (ingredientes y/o tipo de cocina).

El nodo “*ask for the @ingredients or @cuisine_type*” contiene los siguientes nodos hijos (ver Figura 97):

- Nodo “*cuisine type found*”: evalúa la respuesta en busca de la entidad `@cuisine_type`. Si está presente almacena su valor en la variable `$cuisine_type`. Después pregunta al usuario por intolerancias, y salta al nodo “*intolerances found*” para evaluar la respuesta.
- Nodo “*ingredients found*”: evalúa la respuesta en busca de la entidad `@ingredients`. Si está presente almacena su valor en la variable `$ingredients`. Después pregunta al usuario por intolerancias, y salta al nodo “*intolerances found*” para evaluar la respuesta.
- Nodo “*positive response*”: a este nodo se entra cuando el usuario responde que sí tiene preferencias de ingredientes o tipo de cocina, pero no ha proporcionado valores válidos, por lo que se vuelve a preguntar una última vez por dicha información.
- Nodo “*negative / uncertain response*”: este nodo se satisface si el usuario responde de forma negativa o incierta a la pregunta de si tiene preferencias de ingredientes o tipo de cocina. En ese caso, se proporcionará una recomendación propia con 6 opciones, entre las que el usuario debe seleccionar una. Para activar esta acción en la aplicación orquestadora la variable `$yum_sugest` se pone a *True* y se responde al usuario informando de los siguientes pasos.
- Nodo “*unknown response (first time)*”: a este nodo se entra si ninguno de los anteriores se ha satisfecho, y si además es la primera vez que se ha preguntado por los ingredientes o el tipo de cocina (`$counter==0`). En este caso, se pide una última vez la información necesaria al usuario y se incrementa el contador.
- Nodo “*make a suggestion*”: se activa si el contador es mayor que 0 y si no se ha encontrado ingredientes o tipo de cocina válidos. En este caso se informa al usuario, y se activa la variable `$yum_sugest`, para que la aplicación orquestadora proporcione 6 opciones de platos al usuario y pida que elija una de ellas.

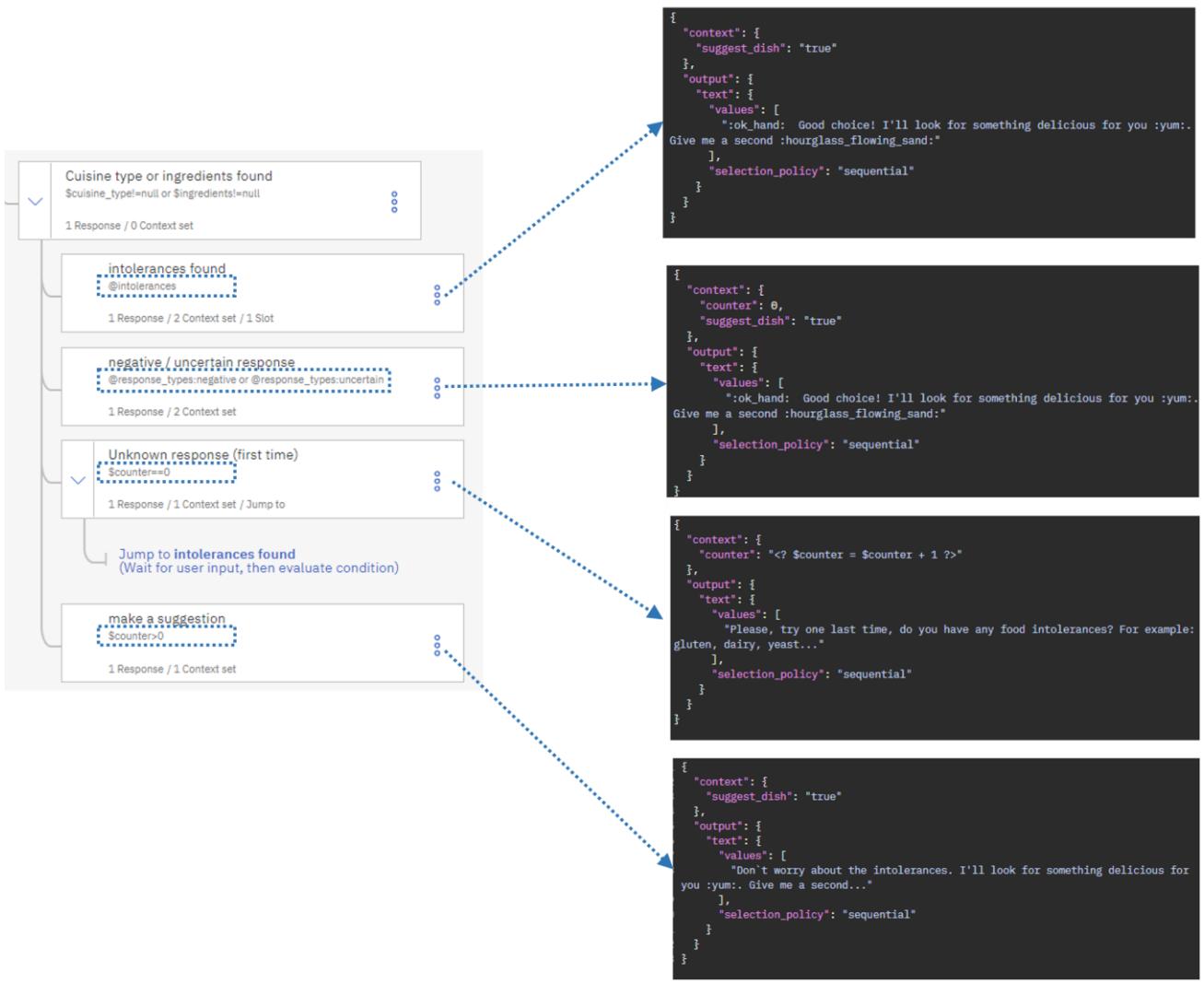


Figura 96 – Intención #sugest_dish: nodo “cuisine type or ingredients found”

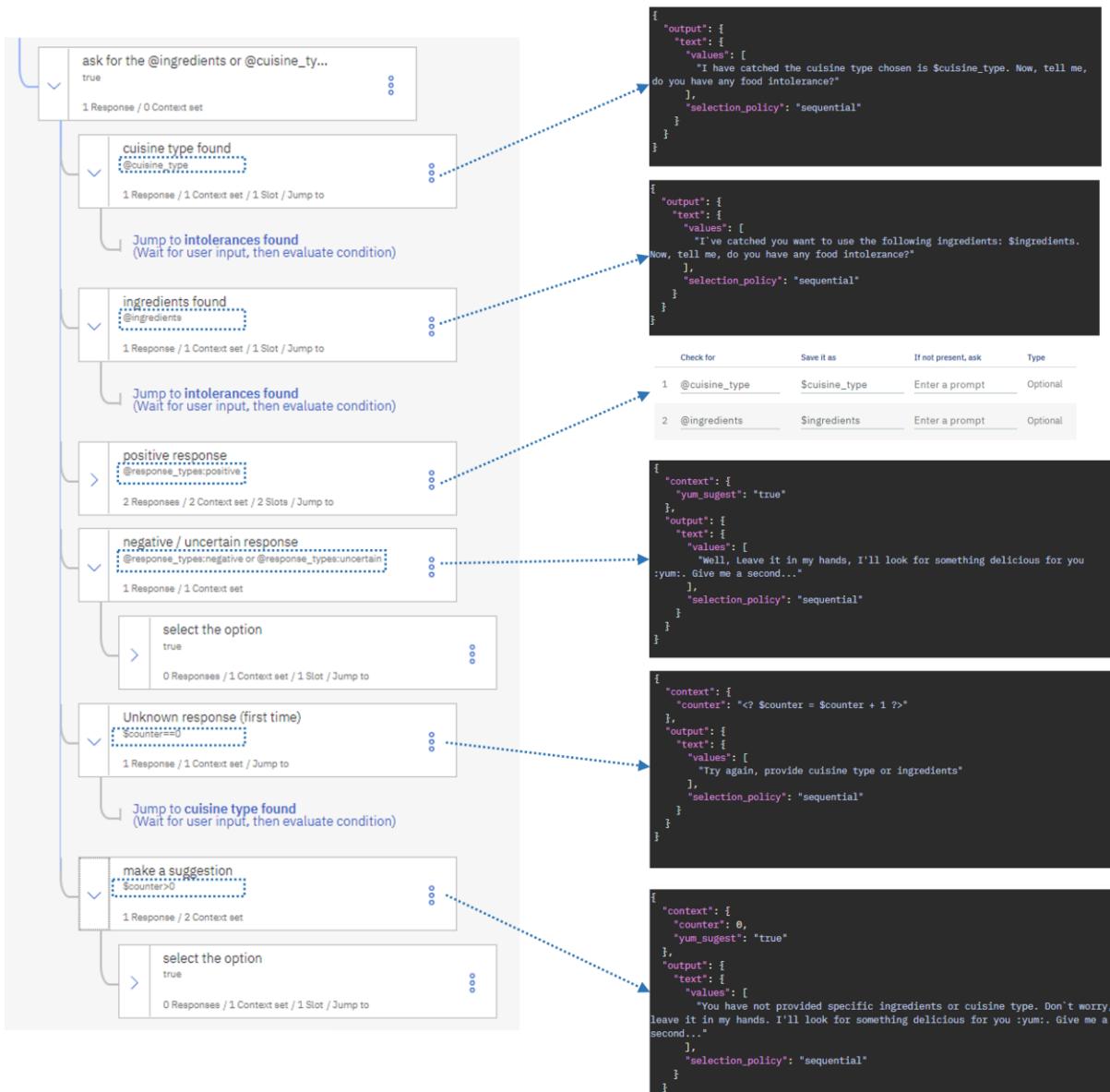


Figura 97 – Intención #sugest_dish: nodo “ask for the @ingredients or @cuisine_type”

3.5.4.12 Intención #select_option

Al igual que en el resto, cuando se cumple la condición que activa este nodo (el usuario desea seleccionar una opción entre la lista propuesta) se resetean las variables de contexto que activan acciones en la aplicación orquestadora.

En este caso, la mayor parte de la gestión recae del lado de la aplicación orquestadora. La respuesta al usuario se maneja en la aplicación orquestadora, ya que ésta es la que dispone de la información relativa a la lista de opciones proporcionadas al usuario, y entre las que éste seleccionará una.

 **yummumBot** APP 16:58
 You have not provided the dish name you want to cook, I'll find something delicious for you, give me a second.
 I have found the following recipes for you:
 [1] : Eggplant Parmesan
 [2] : Crispy Baked Eggplant Fries with Marinara Dipping Sauce (aka Eggplant Parmesan Fries!)
 [3] : Perfect Iced Coffee
 [4] : Jalapeno Popper Grilled Cheese Sandwich
 [5] : No-Bake Chocolate Peanut Butter Pretzel Cookies
 [6] : All Recipes
 Please, provide a valid option from 1 to 6

 **Inma** 16:59
 @yummumBot number 2

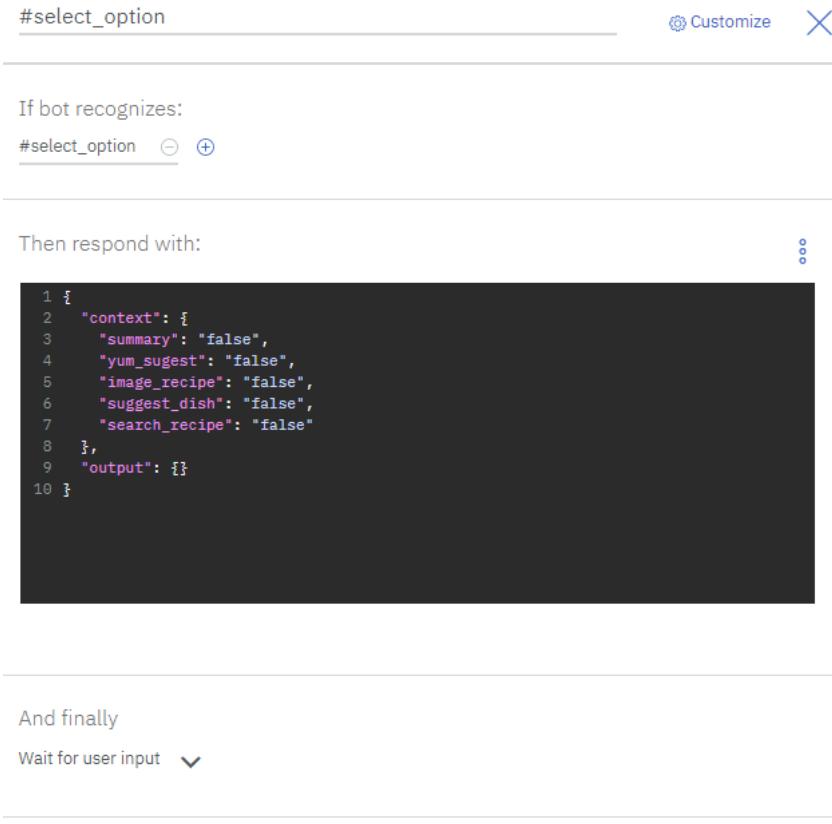
 Ok, good choice! The Crispy Baked Eggplant Fries with Marinara Dipping Sauce (aka Eggplant Parmesan Fries!) recipe below:
 To cook this dish you need the following ingredients:
 - 1 medium eggplant, cut into 1/4 inch thick 'fries'
 - 1/2 cup flour
 - 2 eggs, lightly beaten
 - 3/4 cup panko breadcrumbs
 - 1/4 cup parmesano reggiano (parmesan), grated
 - 1 teaspoon Italian seasoning
 - salt and pepper to taste

Here, you can find the method of cooking: <http://www.closetcooking.com/2012/09/crispy-baked-eggplant-fries-with.html>

Figura 98 – Intención `#select_option`: respuesta

Esta intención habitualmente se detectará como parte final de las intenciones `#sugest_dish` (sección 3.5.4.11) o `#get_recipe` (sección 3.5.4.10), pero también se ha añadido como un nodo más del diálogo porque podría ocurrir que el usuario realice algún comentario positivo o negativo sobre las opciones proporcionadas (`#negative_reaction` o `#positive_reaction`) y que tras dicho comentario sea cuando seleccione la opción, la lista de opciones se almacena hasta que se elige una opción, se finaliza o inicia una conversación.

Finalmente, una vez que se proporciona respuesta al usuario se queda a la espera de una nueva entrada.



The screenshot shows the Rasa NLU configuration interface with the following details:

- Node Type:** #select_option
- Customize:** A button with a gear icon.
- Cancel:** A red X button.
- If bot recognizes:**
 - #select_option
 - ⊖
 - ⊕
- Then respond with:**

```

1 {
2   "context": {
3     "summary": "false",
4     "yum_sugest": "false",
5     "image_recipe": "false",
6     "suggest_dish": "false",
7     "search_recipe": "false"
8   },
9   "output": {}
10 }

```
- And finally:**
 - Wait for user input
 - ⌄

Figura 99 – Nodo de la intención `#select_option`

3.5.4.13 Intención #goodbyes

Esta intención es obligatoria en cualquier chatbot, debe tener un nodo del diálogo que gestione cómo despedirse del usuario. Las acciones que realiza este nodo son las siguientes:

- 1) Resetear las variables de contexto que activan acciones en la aplicación orquestadora. Y además resetea el contador de reacciones negativas (*\$insult_counter*).
- 2) Responder al usuario despidiéndose. Se han programado varias respuestas disponibles que se seleccionan de forma secuencial.

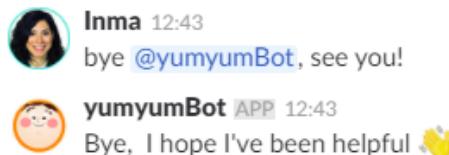


Figura 100 – Intención #goodbyes: respuesta

- 3) Finalmente se queda a la espera de una nueva respuesta por parte del usuario, se ha decidido así porque la conexión de Slack no se cierra. El mismo usuario u otro usuario podría iniciar una nueva conversación en cualquier otro momento. Otros chatbots podrían aprovechar esta intención para desconectar la plataforma de mensajería o incluso si se trata de un asistente con reconocimiento de voz colgar el teléfono, transferir la llamada, etc.

#Bye

If bot recognizes:
#goodbyes

Then respond with:

```
1 {
2   "context": {
3     "summary": "false",
4     "yum_sugest": "false",
5     "image_recipe": "false",
6     "suggest_dish": "false",
7     "search_recipe": "false",
8     "insult_counter": 0
9   },
10  "output": {
11    "text": {
12      "values": [
13        "Bye, I hope I've been helpful :wave:",
14        "Bye, nice to help you :wave:"
15      ],
16    }
17  }
18}
```

And finally

Wait for user input

Figura 101 – Nodo intención #goodbyes

3.5.4.14 Nodo Anything else

La misión de este nodo de la conversación es gestionar las peticiones del usuario para las que el chatbot no está entrenado o no puede clasificar en el resto de las intenciones ya mencionadas. Por este motivo este nodo es el último en evaluarse siempre en una conversación, si no ha entrado en el resto de nodos lo hará en este.

Las acciones que realiza este nodo son:

- 1) Resetear las variables de contexto que activan acciones en la aplicación orquestadora.
- 2) Responder al usuario solicitando que reformule su consulta porque no le ha entendido.



Figura 102 – Nodo Anything else: respuesta

- 3) Esperar una nueva entrada del usuario.

The screenshot shows the configuration of an 'Anything else' node in Node-RED. The node has the following settings:

- Name:** Anything else
- If bot recognizes:** anything_else
- Then respond with:**

```
1 {
2   "context": {
3     "summary": "false",
4     "yum_sugest": "false",
5     "image_recipe": "false",
6     "suggest_dish": "false",
7     "search_recipe": "false"
8   },
9   "output": {
10     "text": {
11       "values": [
12         "I didn't understand. You can try rephrasing.",
13         "Can you reword your request? I'm not understanding.",
14         "I didn't get your meaning."
15       ],
16     }
17   }
18 }
```
- And finally:** Wait for user input

Figura 103 – Configuración del nodo Anything else

3.6 Aplicación orquestadora

La aplicación orquestadora se ha implementado en Python, y será la encargada de coordinar todos los servicios, interfaces y fuentes de información que necesite nuestro chatbot para funcionar. Se ocupa de gestionar el canal de comunicación, las conexiones con las distintas bases de datos (productos y recetas), las peticiones al servicio de reconocimiento de imágenes, así como de intervenir en la conversación y procesar las acciones correspondientes a la funcionalidad solicitada por el usuario.

La aplicación orquestadora realiza las siguientes tareas:

- 1) Crear un objeto de la clase *SmartFridge*. Esta clase contiene todos los métodos necesarios para la gestión de la comunicación y gestión del diálogo. Estos métodos serán descritos en los siguientes apartados.
- 2) Establecer la conexión en el canal de comunicación (Slack).
- 3) Leer el contenido recibido en el canal de comunicación cada *READ_WEBSOCKET_DELAY* segundos.
- 4) Filtrar los mensajes dirigidos al chatbot.
- 5) Enviar el mensaje recibido al servicio *Watson Conversation* para obtener de éste la intención del mensaje, las entidades, las variables de contexto, o incluso la respuesta que se debe enviar al usuario si aplica.
- 6) Atender la petición según la intención que ha sido detectada en el paso anterior. La aplicación orquestadora utilizará la información proporcionada por el servicio Watson conversation y la combinará con el resto de servicios para componer o completar la respuesta adecuada al usuario en cada caso.

El código de la función principal de la aplicación orquestadora es el siguiente:

```
if __name__ == "__main__":
    smartfridge=SmartFridge()

    if smartfridge.slack_client.rtm_connect():
        print("smartfridge connected and running!")
        while True:
            command, channel = smartfridge.parse_slack_output(smartfridge.slack_client.rtm_read())
            if command and channel:
                smartfridge.handle_command(command, channel)
                time.sleep(READ_WEBSOCKET_DELAY)
            else:
                print("Connection failed. Invalid Slack token or bot ID?")
    else:
        print("Connection failed. Invalid Slack token or bot ID?")
```

Figura 104 – Aplicación orquestadora: programa principal

Pasamos a describir los principales bloques funcionales en los que se divide la aplicación. Realizaremos una descripción funcional, para más detalles se puede consultar el código que está disponible en el *Apéndice I: Código*.

3.6.1 Constantes

En este apartado echaremos un vistazo a las constantes que utiliza la aplicación orquestadora.

Se han utilizado variables de contexto que se recuperan al iniciar la aplicación para después asignarlas a la constante correspondiente. Se ha elegido esta operativa para evitar incluir información comprometida en el código, como por ejemplo las credenciales de los servicios de *Watson* o el servicio *Food2Fork*.

Las constantes se pueden agrupar como sigue:

- **Slack:** Identificador de la aplicación Slack (*BOT_ID*), token de seguridad para utilizar la API RTM de Slack (*SLACK_BOT_TOKEN*), cadena de texto para filtrar mensajes dirigidos al bot (*AT_BOT*), y retardo en segundos para leer los mensajes del canal de comunicación (*READ_WEBSOCKET_DELAY*).

- **Food2Fork:** clave de acceso para las peticiones al servicio *Food2Fork* (*FOOD2FORK_KEY*).
- **Base de datos de productos:** cadena de texto con los datos de conexión a la base de datos (*DB_STRING_CONNECTION*).
- **Servicio Watson Visual Recognition:** versión (*VISUAL_RECOGNITION_VERSION*), url base (*VISUAL_RECOGNITION_URL*), y clave (*VISUAL_RECOGNITION_KEY*) para uso de la API del servicio *Watson Visual Recognition*.
- **Servicio Watson Conversation:** versión (*CONVERSATION_VERSION*), usuario (*CONVERSATION_USERNAME*), password (*CONVERSATION_PASSWORD*) e identificador de workspace (*CONVERSATION_WORKSPACE*) del servicio *Watson Conversation*.
- **Otras:** número de días restantes para considerar un producto próximo a caducar (*DAYS_TO_EXPIRE*), número de opciones de platos entre las que el usuario podrá elegir (*TOTAL_NUMBER_OPTIONS*).

```
# Slack bot id
BOT_ID = os.environ.get("BOT_ID")
# Slack token
SLACK_BOT_TOKEN=os.environ.get('SLACK_BOT_TOKEN')
# String to identify messages with bot as recipient
AT_BOT = "<@{} + BOT_ID + ">"
# Delay between reading from firehose
READ_WEBSOCKET_DELAY = 1

# Food2fork API key
FOOD2FORK_KEY=os.environ.get('FOOD2FORK_KEY')

# Postgres database connection string
DB_STRING_CONNECTION = "host='localhost' dbname='smartfridge' user='postgres' password='postgres'"

# Watson Visual Recognition version (to ensure backward compatibility)
VISUAL_RECOGNITION_VERSION = '2017-10-15'
# Watson Visual Recognition base url
VISUAL_RECOGNITION_URL = 'https://gateway-a.watsonplatform.net/visual-recognition/api'
# Watson Visual Recognition API key
VISUAL_RECOGNITION_KEY = os.environ.get('VISUAL_RECOGNITION_KEY')

# Watson Conversation version (to ensure backward compatibility)
CONVERSATION_VERSION = '2017-09-23'
# Watson Conversation username
CONVERSATION_USERNAME = os.environ.get('CONVERSATION_USERNAME')
# Watson Conversation password
CONVERSATION_PASSWORD = os.environ.get('CONVERSATION_PASSWORD')
# Watson Conversation workspace identifier
CONVERSATION_WORKSPACE = os.environ.get('CONVERSATION_WORKSPACE')
# Watson Conversation base url
CONVERSATION_URL = 'https://gateway.watsonplatform.net/conversation/api'

# Number of remaining days to consider a product as next to expire
DAYS_TO_EXPIRE = 7
# Number of dish options provided to the user
TOTAL_NUMBER_OPTIONS = 6
```

Figura 105 – Aplicación orquestadora: constantes

3.6.2 Variables de contexto

Tal y como se explicó en el apartado 3.5.3.3, la aplicación orquestadora maneja y consulta las variables de contexto, que son compartidas con el servicio *Watson Conversation* con el objetivo de mantener el estado de la conversación entre un intercambio con el usuario y otro posterior.

Las variables de contexto se almacenan en un diccionario cuya clave es el nombre de la variable y su valor el valor de la variable en cuestión. Estas son inicializadas cuando la aplicación orquestadora comienza su ejecución.

```

    self.context['search_recipe'] = False
    self.context['image_recipe'] = False
    self.context['suggest_dish'] = False
    self.context['yum_sugest'] = False
    self.context['summary'] = False
    self.context['option'] = None
    self.context['cuisine_type'] = None
    self.context['ingredients'] = None
    self.context['intolerances'] = None
    self.context['dish'] = None
    self.context['counter'] = 0
    self.context['insult_counter'] = 0

```

Figura 106 – Aplicación orquestadora: Inicialización de las variables de contexto

Los cambios de valor de las variables de contexto se realizan casi íntegramente en el servicio *Watson Conversation*. El uso de las variables de contexto en la aplicación orquestadora es principalmente de consulta. Es decir, a lo largo de la conversación, la aplicación orquestadora consultará los valores de estas variables para activar o desactivar acciones y/o búsquedas, y para elegir el momento adecuado para enviar respuestas a la interfaz de usuario.

Sólo existe una excepción, y es para la variable de contexto *\$image_recipe*, en este caso es la aplicación orquestadora la encargada de ponerla a *True* cuando recibe una imagen válida a través de la interfaz de usuario.

La actualización de las variables de contexto seguirá principalmente el siguiente flujo:

- 1) La aplicación orquestadora envía las variables de contexto en cada solicitud al servicio *Watson Conversation*.
- 2) El servicio *Watson Conversation* consulta los valores de las variables de contexto en cada nodo de la conversación, realiza las acciones correspondientes y actualiza las variables si aplica.
- 3) El servicio *Watson Conversation* envía respuesta a la petición realizada en 1) con la intención y entidades detectadas, la respuesta y las variables de contexto actualizadas.
- 4) La aplicación orquestadora actualiza el diccionario que contiene las variables de contexto con los nuevos valores y realiza las acciones correspondientes según las intenciones, entidades y variables de contexto recibidas como respuesta.

```

response = self.conversation.message(workspace_id=CONVERSATION_WORKSPACE,
                                       message_input=message,
                                       context=self.context,
                                       alternate_intents=False)

self.update_local_context(response['context'])
self.intents = response['intents']
self.entities = response['entities']

```

Figura 107 – Aplicación orquestadora: peticiones al servicio Watson Conversation

3.6.3 Inicialización de servicios

Al inicio de su ejecución, la aplicación orquestadora se encargará de inicializar los servicios y conexiones necesarias para el correcto funcionamiento de nuestro chatbot.

Estos servicios son: *SlackClient*, *Watson Conversation*, *Watson Visual Recognition*, *Food2Fork*. También se inicializará la conexión con la base datos de productos. La aplicación orquestadora utiliza la SDK disponible para Python de cada uno de estos servicios para proporcionar las credenciales requeridas y obtener así una instancia de cada uno de ellos. Esta instancia se utilizará durante toda la ejecución de la aplicación y no será necesario volver a proporcionar las credenciales nuevamente.

En el caso del servicio *Food2Fork*, no existe SDK para Python, las peticiones se realizarán utilizando el método *get* de la librería *requests*, y las credenciales viajarán en cada petición. Se ha implementado un

método (*food2fork_request*) que facilita esta tarea y que se puede consultar en más detalle en el apartado 3.6.5 o en el Apéndice I: Código.

```
# Services initialization

# Slack client instance
self.slack_client = SlackClient(SLACK_BOT_TOKEN)

# Watson Conversation service instance
self.conversation = Conversation(version = CONVERSATION_VERSION,
                                   username = CONVERSATION_USERNAME,
                                   password = CONVERSATION_PASSWORD,
                                   url = CONVERSATION_URL)

# Watson Visual Recognition service instance
self.visual_recognition = VisualRecognition(version=VISUAL_RECOGNITION_VERSION,
                                              url=VISUAL_RECOGNITION_URL,
                                              api_key=VISUAL_RECOGNITION_KEY)

# Database connection
self.database_connection()
```

Figura 108 – Aplicación orquestadora: Inicialización de servicios

3.6.4 Integración con *Slack*

Las tareas que realiza la aplicación orquestadora para gestionar el canal de comunicación se pueden resumir como sigue:

- 1) Conexión al canal de comunicación en tiempo real proporcionado por Slack mediante RTM Websockets.
- 2) Lectura continua del flujo *RTM WebSocket*. Recibe un JSON en el que están contenidos todos los mensajes entrantes.
- 3) Filtra los mensajes dirigidos al chatbot mediante su *BOT_ID*.
- 4) Descarga imágenes si las hay entre los mensajes entrantes.
- 5) Envía respuesta correspondiente al canal.

Para todas estas tareas se han definido los métodos que se describen a continuación.

3.6.4.1 Método *parse_slack_output*

Un canal de Slack puede tener un elevado número de miembros, y todos ellos pueden dirigirse al bot si éste también ha sido añadido a dicho canal. Para evitar saturación del buffer de mensajes sólo se atenderán los mensajes dirigidos al chatbot y que, por tanto, contengan la mención @yummyBot.

<i>parse_slack_output</i> (self, slack_rtm_output, download_path='download/food.jpg')	
Funcionalidad	Busca las menciones al bot (@BOT_ID) de entre la lista de mensajes de entrada y filtra aquellos que van dirigidos al chatbot.
Parámetros	<ul style="list-style-type: none">· slack_rtm_output: lista de mensajes entrantes en formato JSON.· download_path: path donde almacenar las descargas de imágenes contenidas en los mensajes. Su valor por defecto es 'download/food.jpg'.

Devuelve	<ul style="list-style-type: none"> · command: cadena de texto con el mensaje que ha introducido el usuario. O cadena de texto con la palabra “photo” para indicar que se ha descargado una imagen. O un texto indicando que se ha producido un error de formato de imagen si esta no es <i>PNG</i> o <i>JPG</i>. · channel: identificador del canal de comunicación.
-----------------	--

3.6.4.2 Método `send_response`

<i>send_response (self, response, channel)</i>	
Funcionalidad	Envía al canal de comunicación establecido el texto que recibe como entrada (<i>response</i>).
Parámetros	<ul style="list-style-type: none"> · response: texto para enviar al canal comunicación de Slack. · channel: identificador del canal al que se debe enviar el mensaje.
Devuelve	Texto con la respuesta de la petición HTTP.

3.6.4.3 Método `download_file`

<i>download_file (self, url, local_filename, basedir)</i>	
Funcionalidad	Descarga un fichero compartido en Slack en el path local (<i>local_filename</i>) que se indica como entrada.
Parámetros	<ul style="list-style-type: none"> · url: url privada de Slack donde se encuentra el archivo compartido a descargar. Es una url privada, por tanto es necesario usar el token de Slack en la descarga. · local_filename: ruta relativa local donde almacenar el archivo que se va a descargar. · basedir: directorio relativo donde se almacenará la descarga. Si no existe se crea.
Devuelve	<i>True</i> si la descarga se ha realizado correctamente, o <i>False</i> si se ha producido algún error en el proceso.

3.6.5 Integración con servicio *Food2Fork*

La aplicación orquestadora utiliza el servicio de recetas *Food2Fork* para todas las acciones relacionadas con búsquedas de platos y recetas a partir de ciertos ingredientes o tipo de cocina, intolerancias, etc.

La aplicación orquestadora realizará las peticiones HTTP oportunas para:

- Buscar platos que contengan ciertos ingredientes.
- Buscar platos pertenecientes a cierto tipo de cocina.
- Buscar platos para ciertas intolerancias alimentarias.
- Buscar los ingredientes y el método de cocinado de un determinado plato.
- Buscar las recetas más populares.

Para todas estas tareas se han definido los métodos que se describen a continuación.

3.6.5.1 Método `food2fork_request`

<i>food2fork_request (self, endpoint, **kwargs)</i>	
Funcionalidad	Método para facilitar la construcción de la url a la que se realizará la petición HTTP a la API del servicio Food2Fork.
Parámetros	<ul style="list-style-type: none"> · endpoint: url base para peticiones a la API de Food2Fork. Será igual a

	<p>http://food2fork.com/api/search para búsquedas, y http://food2fork.com/api/get para consulta de recetas.</p> <ul style="list-style-type: none"> · kwargs: parámetros de la petición HTTP. Deben ser proporcionados de la forma clave-valor.
Devuelve	Devuelve una cadena de texto con la url correspondiente.

3.6.5.2 Método `search_recipes`

<i>search_recipes (self, query, sortBy='r')</i>	
Funcionalidad	Realiza petición HTTP a la API del servicio Food2Fork para buscar recetas que cumplan los términos de búsqueda contenidos en el texto que recibe como parámetro.
Parámetros	<ul style="list-style-type: none"> · query: texto de la búsqueda (por ejemplo: “chicken”, “mexican food”, o “gluten-free”). · sortBy: especifica el orden en el que se deseaba recuperar las recetas de la búsqueda. Puede tomar dos posibles valores ‘r’ (rating) y ‘t’ (trendingness). El valor por defecto es ‘r’ (rating), en este caso las recetas se ordenan según su puntuación en las redes sociales. Mientras que con el valor ‘t’ (trendingness) se ordenan según la rapidez con la que las recetas más recientes están ganando popularidad.
Devuelve	Devuelve la lista de recetas encontradas por el servicio Food2Fork para la búsqueda ejecutada en formato JSON.

3.6.5.3 Método `get_recipe_from_id`

<i>get_recipe_from_id (self, recipeId):</i>	
Funcionalidad	Realiza petición HTTP en el servicio Food2Fork para buscar toda la información (ingredientes, método de cocinado, fuente, url de la fuente, etc.) de una receta a partir de un identificador e receta dado como entrada.
Parámetros	recipeId: Identificador de receta interno de Food2fork para el que se quiere obtener toda su información.
Devuelve	Devuelve un JSON con toda la información de respuesta del servicio Food2Fork para la búsqueda realizada, o <i>None</i> en caso de que no encuentre coincidencias.

3.6.5.4 Método `get_recipe_id`

<i>get_recipe_id (self, query, sortBy='r'): </i>	
Funcionalidad	Obtiene el identificador de la receta más puntuada o más popular para la búsqueda dada.
Parámetros	<ul style="list-style-type: none"> · query: texto de la búsqueda (por ejemplo: “chicken”, “mexican food”, o “gluten-free”). · sortBy: especifica el orden en el que se deseaba recuperar las recetas de la búsqueda. Puede tomar dos posibles valores ‘r’ (rating) y ‘t’ (trendingness). El valor por defecto es ‘r’ (rating), en este caso las recetas se ordenan según su puntuación en las redes sociales. Mientras que con el valor ‘t’ (trendingness) se ordenan según la rapidez con la que las recetas más recientes están ganando popularidad.
Devuelve	Devuelve una cadena de texto con el identificador interno de Food2Fork o <i>None</i> si no se encontraron resultados para la búsqueda realizada.

3.6.5.5 Método `get_top_rated_recipe_options`

<code>get_top_rated_recipe_options (self, n_options=1)</code>	
Funcionalidad	Busca las recetas más puntuadas en el servicio Food2Fork.
Parámetros	<code>n_options</code> : número de recetas a devolver. El valor por defecto es igual a 1.
Devuelve	<p>Lista con los nombres de las <code>n_options</code> recetas más puntuadas. Notar que en este caso devuelve sólo los nombres de los platos (title).</p> <p>También almacena en el diccionario <code>option_dic</code> cada una de los nombres de las recetas proporcionado como clave del diccionario junto con su identificador de receta como valor.</p>

3.6.5.6 Método `get_trending_recipe_options`

<code>get_trending_recipe_options (self, n_options=1)</code>	
Funcionalidad	Busca las recetas más recientes que han ganado más popularidad en el servicio Food2Fork.
Parámetros	<code>n_options</code> : número de recetas a devolver. El valor por defecto es igual a 1.
Devuelve	<p>Lista con los nombres de las <code>n_options</code> recetas más populares. Notar que en este caso devuelve sólo los nombres de los platos (title).</p> <p>También almacena en el diccionario <code>option_dic</code> cada una de los nombres de las recetas proporcionado como clave del diccionario junto con su identificador de receta como valor.</p>

3.6.6 Integración con base de datos de productos

La aplicación orquestadora necesita conocer el contenido del frigorífico en todo momento para poder proporcionar sugerencias con los productos más próximos a caducar, o para contestar a las preguntas del usuario respecto a su contenido (existencia o no de productos, productos próximos a caducar, productos caducados, etc.).

Tal y como se explicó en el apartado 3.3, el frigorífico se ha modelado como una base de datos de productos, de modo que para este cometido la aplicación orquestadora se conectará a dicha base de datos y así podrá conocer su contenido.

A continuación se detallarán los métodos que se han implementado para encapsular las acciones que la aplicación orquestadora realizará con la base de datos de productos.

3.6.6.1 Método `database_connection`

<code>database_connection (self, str_db_connection)</code>	
Funcionalidad	Crea una conexión a la base de datos que se especifica como entrada.
Parámetros	<code>str_db_connection</code> : cadena de texto con los datos de conexión (nombre de la base de datos y credenciales). Por ejemplo: " <code>host='localhost' dbname='smartfridge' user='postgres' password='postgres'</code> ".
Devuelve	Devuelve un objeto <code>cursor</code> para realizar queries.

3.6.6.2 Método `fetch_content`

<code>fetch_content (self, query)</code>	
Funcionalidad	Ejecuta la consulta SQL que recibe como entrada y devuelve todos los nombres de productos recuperados de la base de datos. Notar que sólo devuelve los nombres de los

	productos, no recupera el resto de información como por ejemplo fecha de caducidad, cantidad (gr).
Parámetros	query : cadena de texto con la consulta SQL a ejecutar en la base de datos.
Devuelve	Lista con los nombres de los productos que cumplen la consulta realizada.

3.6.6.3 Método `get_top_expired_ingredients_from_db`

get_top_expired_ingredients_from_db (self, n_ingredients=2)	
Funcionalidad	Obtiene los productos más próximos a caducar y para los que hay más cantidad de la base de datos.
Parámetros	n_ingredients : número de productos a recuperar, por defecto son 2.
Devuelve	Lista de los “ n_ingredients ” productos más próximos a caducar.

3.6.6.4 Método `get_db_information_about_ingredients`

get_db_information_about_ingredients (self, ingredients)	
Funcionalidad	Recupera la información de fecha de caducidad y cantidad de un determinado ingrediente contenido en la base de datos.
Parámetros	ingredients : nombre del producto para el que se desea realizar la consulta.
Devuelve	Devuelve la lista de records recuperada de la base datos para la información consultada.

3.6.7 Integración con Watson Visual Recognition

La aplicación orquestadora necesita reconocer comida en las imágenes que comparte el usuario, como paso previo para proporcionar su receta. Para ello la aplicación orquestadora hace uso del servicio *Watson Visual Recognition* (ver apartado 2.3.2.3) y de su modelo ”*food*”, que ya ha sido pre-entrenado para reconocer comida y alimentos con bastante precisión.

Se ha implementado un método con esta finalidad que se detalla a continuación.

3.6.7.1 Método `get_food_from_image`

get_food_from_image (self, image_path)	
Funcionalidad	Realiza una petición al servicio Watson Visual Recognition proporcionando una imagen cuyo path recibe como entrada, y devuelve el nombre del alimento o plato que reconoce en dicha imagen.
Parámetros	image_path : path donde se encuentra la imagen para la que se realiza la consulta.
Devuelve	<ul style="list-style-type: none"> · food: plato o alimento que ha sido reconocido con mayor confianza en la imagen proporcionada. Si el servicio no reconoce comida en la imagen devuelve “<i>non-food</i>”. · score: probabilidad con la que ha sido reconocida. <p>Si se produce un fallo durante la petición o lectura de la imagen se devuelve <i>None</i> para ambas variables de salida.</p>

3.6.8 Gestión de la conversación

La aplicación orquestadora es la encargada de coordinar las peticiones a los diferentes servicios y fuentes de información para componer y proporcionar la respuesta más apropiada al usuario.

Los métodos que se describen en este apartado se utilizan para la construcción de dichas respuestas, y a su vez, hacen uso de los métodos descritos anteriormente.

3.6.8.1 Método `handle_command`

<i>handle_command (self, command)</i>	
Funcionalidad	<p>Este método se encarga de proporcionar la respuesta adecuada al usuario para una petición dada. Las acciones que realiza son las siguientes:</p> <ul style="list-style-type: none">· Recibe las peticiones del usuario a través del canal de comunicación (Slack).· Envía el mensaje recibido al servicio <i>Watson Conversation</i> y recupera las intenciones, entidades, variables de contexto y respuesta asociadas a dicho mensaje.· Gestiona las llamadas al resto de métodos y servicios en función de las intenciones detectadas y los valores de las variables de contexto.· Realiza las acciones que correspondan (buscar receta, buscar sugerencia, reconocer una imagen, buscar en base datos de productos, informar de un error, etc.), y obtiene o completa la respuesta que finalmente será proporcionada al usuario.· Por último, envía la respuesta correspondiente al usuario a través de Slack.
Parámetros	<p><i>command</i>: cadena de texto con la petición recibida a través de Slack por el usuario.</p> <p>En el caso en el que el usuario comparta una imagen, en el paso previo de lectura del canal de comunicación “<i>command</i>” se etiqueta como “<i>photo</i>”, para indicar que debe leer la imagen del directorio de descargas, y reconocerla con ayuda del servicio <i>Watson Visual Recognition</i>. Del mismo modo, si la imagen compartida tiene un formato incompatible, se marca para indicar al usuario que debe aportar un formato diferente. En el resto de casos en “<i>commad</i>” se recibe la petición tal cual el usuario ha escrito en Slack.</p>
Devuelve	No devuelve ningún valor.

3.6.8.2 Método `select_option`

<i>select_option (self)</i>	
Funcionalidad	<p>Recupera de la lista la opción seleccionada por el usuario durante la conversación (intención <code>#select_option</code> y variable de contexto <code>\$option</code>).</p>
Parámetros	<p>Busca la receta para la opción seleccionada.</p> <p>Construye la respuesta que finalmente se enviará al usuario, con los detalles de ingredientes y el método de cocinado.</p>
Devuelve	No recibe parámetros, la opción elegida por el usuario se encuentra en la variable de contexto <code>\$option</code> . Y la lista de opciones se encuentra en una variable interna de la clase (<code>recipe_options</code>).

3.6.8.3 Método `yum_suggestion`

<i>yum_suggestion (self, n_options=6)</i>	
Funcionalidad	<p>Ofrece una lista con <i>n_options</i> sugerencias de platos al usuario.</p> <p>Por defecto <i>n_options</i> es 6, que se distribuyen de la siguiente manera:</p> <ul style="list-style-type: none"> · 2 platos que contengan 2 de los ingredientes más próximos a caducar de la base de datos de productos. A igualdad de fecha de caducidad de productos prioriza los productos con más cantidad. · Los 2 platos más puntuados en la base de datos de recetas (Food2Fork) · Los 2 platos que han ganado más popularidad recientemente en el servicio Food2Fork. <p>Si no se encuentran 2 platos con los productos próximos a caducar, se completa hasta llegar a <i>n_options</i> con los platos mejor puntuados y con los que han ganado más popularidad. Las opciones se muestran junto con un número para facilitar la elección al usuario y se guardan en la lista interna de la clase (<i>recipe_options</i>) para poder recuperarlas cuando el usuario haga su elección.</p>
Parámetros	<i>n_options</i> : número de sugerencias de platos que se ofrecerán al usuario.
Devuelve	Devuelve una cadena de texto con la lista de opciones. Y una frase para indicar al usuario que seleccione una de ellas.

3.6.8.4 Método `suggest_dish`

<i>suggest_dish (self)</i>	
Funcionalidad	<p>Este método se invoca cuando la variable de contexto <i>\$suggest_dish</i> se pone a <i>True</i>. Esta variable se activa cuando se detecta la intención <i>#suggest_dish</i> y además el usuario proporciona un tipo de cocina o lista de ingredientes válidos.</p>
Parámetros	Busca de recetas para el tipo de cocina (<i>\$cuisine_type</i>), ingredientes (<i>\$ingredients</i>) e intolerancias (<i>\$intolerances</i>) proporcionados por el usuario y contenidos en las correspondientes variables de contexto.
Devuelve	Compone la respuesta con la información solicitada

<i>get_db_summary (self)</i>	
Funcionalidad	Recupera de base datos: todos los productos, los productos caducados y los productos que van a caducar en los próximos “ <i>“DAYS_TO_EXPIRE”</i> días.
Parámetros	Procesa toda la información y compone una respuesta con el resumen del estado de los productos.
Devuelve	No recibe parámetros de entrada.

3.6.8.5 Método `get_db_summary`

<i>get_db_summary (self)</i>	
Funcionalidad	Devuelve una cadena de texto con el resumen del estado de los productos disponibles, y otra parte del mensaje que es variable según los siguientes criterios:
Parámetros	Recupera de base datos: todos los productos, los productos caducados y los productos que van a caducar en los próximos “ <i>“DAYS_TO_EXPIRE”</i> días.
Devuelve	Procesa toda la información y compone una respuesta con el resumen del estado de los productos.

	<ul style="list-style-type: none"> · No hay ningún producto en la nevera: recomienda hacer la compra y pedir comida a domicilio. · No hay productos caducados: felicita al usuario. · Sí hay productos caducados: indica cuáles son y recomienda tirarlos. · Hay productos próximos a caducar: indica cuáles son y recomienda usarlos cuanto antes. · No hay productos próximos a caducar: felicita al usuario.
--	--

3.6.8.6 Método `get_recipe`

<code>get_recipe (self)</code>	
Funcionalidad	<p>Este método se invoca cuando la variable de contexto <code>\$search_recipe</code> es igual a <code>True</code>. Recordemos que esta variable se activa cuando se detecta la intención <code>#get_recipe</code> y el usuario además proporciona el nombre de un plato válido (entidad <code>@dish</code>).</p> <p>Comprueba que <code>\$dish</code> no está vacío y realiza una consulta al servicio Food2Fork para obtener la receta de dicho plato (ingredientes y método de cocinado).</p>
Parámetros	No recibe parámetros de entrada. El plato para el que debe buscar la receta se encuentra en la variable de contexto <code>\$dish</code> .
Devuelve	Devuelve una cadena de texto con los detalles de la recta (lista de ingredientes y enlace al método de cocinado). Si no se encuentra receta para la búsqueda se devuelve un mensaje distinto indicándolo.

3.6.8.7 Método `msg_to_conversation`

<code>msg_to_conversation (self, input_message)</code>	
Funcionalidad	<p>Hace uso del método <code>message</code> (disponible en la SDK para Python) para enviar un mensaje al servicio Watson Conversation junto con el contexto de la conversación.</p> <p>Procesa la respuesta del servicio Watson Conversation para extraer: intenciones, entidades, respuesta y variables de contexto actualizadas.</p>
Parámetros	<code>input_message</code> : cadena de texto con el mensaje que se quiere enviar al servicio Watson Conversation.
Devuelve	<ul style="list-style-type: none"> · <code>response_text</code>: cadena de texto con la respuesta devuelta por el servicio Watson conversation para el mensaje de entrada (<code>input_message</code>). · <code>intent</code>: intención detectada por el servicio con más probabilidad. · <code>entity</code>: entidad detectada por el servicio.

3.6.8.8 Método `update_local_context`

<code>update_local_context (self, context)</code>	
Funcionalidad	Actualiza las variables de contexto con los valores que se proporcionan a la entrada.
Parámetros	<code>context</code> : diccionario con los nuevos valores de las variables de contexto. Las claves son los nombres de las variables y los valores son los valores que toman éstas.
Devuelve	No devuelve nada, se actualiza la variable interna de la clase (<code>context</code>)

3.6.8.9 Método `update_conversation_context`

<i>update_conversation_context (self)</i>	
Funcionalidad	Actualiza las variables de contexto del servicio Watson Conversation con los valores que maneja localmente la aplicación orquestadora.
Parámetros	No tiene parámetros de entrada, utiliza el diccionario local context que contiene las variables de contexto.
Devuelve	No devuelve ningún valor.

3.6.8.10 Método `get_ingredients`

<i>get_ingredients (self, recipeId)</i>	
Funcionalidad	<p>Utiliza el método <code>get_recipe_from_id</code> para obtener toda la información de la receta con identificador <code>recipeId</code>.</p> <p>Procesa la respuesta para extraer la información de ingredientes, url donde se encuentran los detalles del método de cocinado y el título de la receta.</p> <p>Construye la respuesta correspondiente en función de la información anterior.</p>
Parámetros	<code>recipeId</code> : identificador interno del servicio Food2Fork para el que se quiere obtener información de ingredientes
Devuelve	<p>Cadena de texto que contiene el nombre del plato, la lista de ingredientes junto con sus cantidades, y la url donde se encuentran los detalles del método de cocinado.</p> <p>Si no se encuentra la receta para el identificador proporcionado se devuelve un mensaje indicándolo y sugiriendo una nueva búsqueda.</p>

3.6.8.11 Método `get_recipe_options_from_available_ingredients`

<i>get_recipe_options_from_available_ingredients (self, n_options=2, n_ingredients=2)</i>	
Funcionalidad	<p>Utiliza el método <code>get_top_expired_ingredients_from_db</code> para obtener los “<code>n_ingredients</code>” productos más próximos a caducar.</p> <p>Busca recetas que contengan los “<code>n_ingredients</code>” productos recuperados.</p> <p>Devuelve una lista con los nombres de “<code>n_options</code>” recetas encontradas.</p>
Parámetros	<ul style="list-style-type: none"> · <code>n_options</code>: número de opciones de platos que se devolverá en la lista. · <code>n_ingredients</code>: número de ingredientes más próximos a caducar que se recuperarán de base de datos y que debe contener los platos que se devuelvan.
Devuelve	Lista con los nombres de los platos encontrados.

3.6.8.12 Método `get_ingredients_information`

<i>get_ingredients_information (self, ingredients)</i>	
Funcionalidad	<p>Utiliza el método <code>get_db_information_about_ingredients</code> para obtener la información de un determinado producto (<code>ingredients</code>) en la base de datos de productos.</p> <p>Procesa la información extraída de base de datos y compone la respuesta que se enviará al usuario con la fecha caducidad y cantidad disponible del producto consultado.</p>
Parámetros	<code>ingredients</code> : producto para el que se quiere obtener información
Devuelve	La respuesta es una cadena de texto con la información del producto, pero existen 3 posibles opciones:

	<p>El producto existe en la base de datos y no está caducado, entonces devuelve una cadena de texto con la información del producto (fecha de caducidad y cantidad disponible).</p> <p>El producto existe, pero está caducado, devuelve una cadena de texto avisando de este hecho y proponiendo que tire los alimentos en mal estado.</p> <p>El producto no está disponible, en la cadena de texto se informa de esto y se sugiere que se apunte en la lista de la compra.</p>
--	---

3.6.8.13 Método *image_food_recognition*

<i>image_food_recognition(self)</i>	
Funcionalidad	<p>Utiliza el método <i>get_food_from_image</i> (3.6.7.1) para realizar una petición al servicio <i>Visual Recognition</i> y obtener la comida que puede identificarse en la imagen, junto con su score.</p> <p>Busca la receta para el plato de comida que ha sido reconocido en la imagen.</p> <p>Compone la respuesta que finalmente se enviará al usuario.</p>
Parámetros	No recibe parámetros de entrada
Devuelve	<p>Devuelve una cadena de texto con la respuesta que se enviará al usuario.</p> <ul style="list-style-type: none"> · Si no reconoce nada comestible en la imagen se avisa al usuario y se le sugiere una nueva búsqueda. · Si reconoce algo comestible se responde al usuario con el plato encontrado en la imagen y con la receta de dicho plato.

4 CONCLUSIONES

EN este capítulo presentamos un breve resumen del trabajo realizado, se evaluará el prototipo y se describirán las dificultades encontradas durante su desarrollo. Para finalizar se reflexionará sobre los resultados obtenidos y las principales enseñanzas adquiridas.

4.1 Chatbots

Los chatbots basados en reglas siguen un flujo determinado de interacciones y, generalmente, se limitan al entendimiento de oraciones con construcciones o comandos específicos. Éste es el tipo más común de chatbots. Son eficientes para realizar tareas sencillas como indicar la hora, proporcionar información básica de un producto, capturar datos del cliente o promocionar un producto, pero son muy limitados y no entienden realmente al usuario. Se limitan a seguir una serie de reglas que, a medida que aumenta la complejidad de los procesos, se vuelven demasiado complicadas de seguir, tanto para usuarios como para programadores.

Todos nos hemos encontrado con este tipo de chatbots al llamar, por ejemplo, a nuestro operador de telefonía, y enfrentarnos a una interminable serie de menús (“pulse..., diga...”) que buscan determinar a qué área dirigir nuestra llamada para acabar poniéndonos una grabación preparada, o redirigirnos a un operador si el resto de intentos no ha tenido éxito (“no le he entendido, por favor, reformule su consulta”). Esta interacción nos ha llevado a creer que los chatbots son poco inteligentes, deficientes, se confunden con facilidad, o su funcionalidad es limitada.

El éxito de adopción de un chatbot depende, por tanto, del grado en que sea capaz de comprender al usuario y de manejar la conversación. También está directamente relacionado con su valor añadido, es decir la utilidad que proporcione y las necesidades de los usuarios que sea capaz de resolver de forma eficiente y sencilla. Esto nos indica que el grado de éxito de un chatbot se encuentra directamente relacionado con el grado de “inteligencia” que tiene.

Los chatbots que utilizan Inteligencia Artificial (IA) tienen como objetivo realizar un verdadero entendimiento del lenguaje. Son capaces de entender una petición del usuario a partir del contexto, y comprender un mensaje sin importar que tenga errores ortográficos. También permiten que el usuario se exprese libremente, sin limitarlo a una construcción determinada, pero como contrapartida son mucho más complejos en su programación.

Actualmente existen muchas herramientas y APIs que facilitan la implementación de chatbots. Sobre algunas de las más populares ya hemos hablado en el apartado 2. Esta proliferación de frameworks puede llevarnos a engaño, y hacernos pensar que construir un chatbot es cosa de niños, y que cualquiera puede crear uno sin tener conocimientos del tema. La realidad, sin embargo, es muy diferente, yo misma subestimé este trabajo en su inicio porque me había enfrentado previamente a un problema más acotado, y en el que implementé un prototipo con peticiones más limitadas, respuestas predeterminadas y sin consultas de fuentes de datos externas o uso de otro tipo de servicios. Pero la complejidad aumenta en gran medida cuando se añade inteligencia al chatbot para que responda a cuestiones y formulaciones muy diferentes por parte del usuario, y que además construya la respuesta de forma dinámica, consultando fuentes externas, etc. En este caso, la conversación puede tener multitud de posibilidades, y conseguir resultados sólo aceptables requiere un gran esfuerzo en el diseño del flujo de la conversación, así como mucho entrenamiento y depuración con ejemplos reales. Deja de convertirse en un juego de niños, y se convierte en un largo camino en el que entran en juego otras habilidades mucho más avanzadas y específicas del campo del procesado del lenguaje natural, programación, lógica, imaginación e intuición.

4.1.1 Beneficios

Ya comentamos algunos de los beneficios del uso de chatbots en el apartado 1.1. A continuación haremos un repaso de los que consideramos más importantes.

Los chatbots que incorporen Inteligencia Artificial y estén bien programados ofrecerán, entre otros, los siguientes beneficios con respecto a las centralitas de atención al cliente actuales:

- Serán capaces de ofrecer respuestas mucho más acertadas de las que pueda ofrecer una persona, más propensa a los olvidos, a los fallos, o a dar mensajes ambiguos.
- Un mismo chatbot puede responder multitud de peticiones simultáneamente, con el respectivo ahorro de tiempo y dinero.
- Se pueden implementar chatbots que soporten múltiples idiomas.
- Pueden estar disponibles tanto en interfaces vocales como textuales. Estas últimas, según un estudio de One Reach, son las preferidas por el 64% de los clientes frente a las llamadas al servicio de atención al cliente.
- Su disponibilidad es total, 24x7, los 365 días del año. No se cansan, no disfrutan de vacaciones, ni causan bajas.
- Implementan mecanismos de comunicación asíncrona, por si el usuario prefiere interrumpir la conversación o recuperar la respuesta en otro momento

4.1.2 Riesgos

En la implementación y adopción de chatbots nos enfrentamos entre otros a los siguientes riesgos o dificultades:

Mala reputación: la fama que acompaña en ocasiones a los chatbots dificulta su adopción. Muchos usuarios asocian los chatbot con centralitas virtuales, y con interminables conversaciones pidiendo que pulsen tal o cual tecla, o que repitan lo que han dicho porque no les han entendido, música de espera, en definitiva, falta de efectividad para conseguir sus objetivos cuando se comunican con máquinas. También han aumentado esta mala reputación algunos fallos virales de chatbots, como por ejemplo, la fiesta privada que montó Alexa de madrugada en Hamburgo³⁷; Tay, la robot racista de Microsoft³⁸, o la confusión de Siri con el himno de Bulgaria y el “despacito” de Luis Fonsi³⁹.

- **Desconfianza:** algunos usuarios han vivido o escuchado alguna experiencia negativa con chatbots poco éticos, creados, por ejemplo, para ganar seguidores en Twitter, robar credenciales o cazar información personal para captación de clientes. Esto provoca desconfianza entre los usuarios, que recelan de los chatbots y de si estos tienen otro fin oculto menos honesto.
- **Deshumanización:** muchos detractores miran con reticencia el uso de chatbots en la atención al cliente porque creen que esta tecnología deshumanizará la comunicación. Otros argumentan que sustituirán a los humanos y que les arrebatarán sus puestos de trabajo.
- **El usuario no encuentra utilidad:** los chatbots generalistas han provocado una caída de interés. Por ejemplo, el usuario común no sabe para qué utilizar asistentes como Siri, salvo para cuestiones básicas o como novedad.

4.1.3 Oportunidades

Los chatbots que usan inteligencia artificial pueden ser una oportunidad para mejorar la atención al cliente, que actualmente se realiza mayoritariamente a través de centralitas virtuales, y de las que

³⁷ http://www.abc.es/recreo/abci-policia-irrumpe-fiesta-alex-a-asistente-virtual-amazon-monto-si-sola-20171121659_noticia.html

³⁸ http://www.bbc.com/mundo/noticias/2016/03/160325_tecnologia_microsoft_tay_bot_adolescente_inteligencia_artificial_racista_xenofoba_lb

³⁹ <https://androidphoria.com/novedades/siri-himno-bulgaria-despacito>

numerosos estudios⁴⁰ confirman el descontento entre sus usuarios. Ya existen casos de uso para chatbots en servicios de atención al cliente como, por ejemplo:

- **TacoBot:** chatbot de la cadena de restaurantes Taco Bell, que ayuda a los clientes a conocer el menú y realizar su pedido.
- **Telepizza:** chatbot de la cadena de comida rápida española que resuelve dudas de los clientes.
- **KLM:** chatbot de la compañía aérea holandesa que permite a los usuarios obtener información sobre sus reservas.

Estos casos de uso pueden proporcionar una puerta de entrada para que las empresas brinden más servicios a través de chatbots y para que los usuarios comprueben cómo pueden ayudarles.

Además, los avances tecnológicos y el cambio en las interfaces de comunicación han provocado que las conversaciones en persona sean menos habituales que por teléfono, chats o redes sociales, y ha generado que las personas cada vez más interactúen con máquinas.

4.1.4 Retos

Ya hablamos en el apartado 3.5.1 de algunos de los principales retos a los que se enfrentan en la actualidad los sistemas de diálogo. Podemos dividir los retos actuales en los siguientes grupos:

- **Tecnología:** El aumento de la capacidad de procesamiento ha propiciado grandes avances en el campo del procesamiento del lenguaje natural, y esto ha favorecido la aparición de chatbots. Pero aún no se ha logrado alcanzar los niveles de rendimiento humano en lo que respecta a la comprensión del lenguaje natural. Se presentan, por tanto, grandes desafíos para esta tecnología en los próximos años, del mismo modo que presentaban, en sus orígenes para los smartphones.
- **Expectativas:** El uso que hacemos hoy en día de los asistentes virtuales es aún muy básico. Se espera mucho más de ellos. La mayoría de nosotros esperamos poder hablar con ellos tal y como lo hacemos con un familiar o un amigo. Sin embargo, para que esto suceda aún queda mucho. Se está haciendo un gran esfuerzo, pero los resultados prácticos no son tan espectaculares como se muestran en las películas de Hollywood. Esto entra en conflicto con las expectativas y ambición de las grandes compañías que participan en la carrera por ser los primeros en incorporar las tecnologías más novedosas en sus canales de ventas, y que les permitan atraer nuevos clientes y aumentar así sus ganancias.
- **Datos:** son excepciones los casos en los que se dispone de datos de calidad y ejemplos de conversaciones reales que puedan integrarse con el chatbot para ofrecer una experiencia satisfactoria.

4.2 Watson

En este apartado haremos un resumen de los aspectos más destacables de lo que ha supuesto la experiencia con Watson durante el desarrollo de este trabajo.

La principal cuestión a subrayar es que Watson es una tecnología que se entrena, no se programa. Se trata de un conjunto de herramientas o servicios a los que hay que entrenar introduciendo ejemplos y respuestas a multitud de posibles situaciones. A los servicios Watson se accede mediante interfaces y, por tanto, siempre habrá una aplicación que implementar para que orqueste todas las llamadas a estos servicios, a sus fuentes de datos, etc.

La curva de aprendizaje con Watson no es muy elevada, hay muchos videos, documentación, ejemplos y código compartido en Github que facilita su uso. También hay una amplia comunidad de usuarios y desarrolladores que prueban la funcionalidad que ofrece Watson y con los que consultar dudas o compartir código, trucos, etc.

⁴⁰ Una reciente investigación de Aspect pone de manifiesto un dato llamativo, y es que un 42% de los millennials preferiría limpiar aseos antes que contactar con un contact center.

Watson ofrece funcionalidades ya creadas listas para ser usadas, y algoritmos de uso común ya implementados. Esto permite centrarnos en el verdadero problema, sin preocuparnos de implementar código de bajo nivel. Con un poco de optimización y entrenamiento se puede obtener un producto mínimo de forma rápida, lo que supone su principal ventaja frente al uso de algoritmos propios. Sin embargo, también puede limitarnos en desarrollos de mayor envergadura, ya que el comportamiento de Watson no puede ser modificado, normalmente sus servicios disponen de diferentes parámetros de configuración con los que adaptarlos a cada problema, pero no siempre esto es suficiente. Las APIs de Watson son genéricas, y sus algoritmos no pueden alcanzar la precisión de los modelos personalizados, especialmente para trabajos de mayor envergadura.

Durante el trámite de este trabajo he encontrado numerosas restricciones y carencias, las más importantes se enumeran a continuación, separadas por bloques funcionales para una mejor comprensión:

1) Interfaz gráfica para implementación del flujo de diálogo

Esta es la parte del servicio Watson Conversation menos operativa, más limitada y con peor rendimiento. Algunas de las razones encontradas son:

- La interfaz gráfica no es intuitiva, es necesario leer la documentación en profundidad, y revisar ejemplos para entender cómo funciona.
- No permite el trabajo colaborativo, que para proyectos de cierta envergadura es imprescindible. Tampoco dispone de un mecanismo de versionado que facilite el trabajo.
- No es posible deshacer cambios efectuados en el flujo de la conversación. Por ejemplo, no se puede eliminar un nodo recién insertado, restaurar un texto anterior en el editor JSON, volver al texto de la respuesta anterior para un cierto slot, etc. Esto dificulta la implementación, porque si nos equivocamos y hacemos un cambio que introduce una regresión en la conversación, restaurar el estado anterior es casi misión imposible. La única opción pasa por descargar el JSON del workspace antes de empezar con una nueva iteración en la implementación.
- Copiar, pegar, duplicar o mover nodos de la conversación a veces provoca comportamientos erráticos que obligan a salir del workspace y volver a entrar para poder continuar con la implementación del flujo de la conversación. También hemos encontrado situaciones en las que desaparecen los botones de *volver* y *guardar*. En otras ocasiones, tras eliminar un nodo, y aunque no se ha informado de un error, la operación no se efectúa, y el nodo continúa activo. Todo esto unido a la dificultad para versionar y deshacer cambios provoca una cierta psicosis en el desarrollador, y obliga a estar constantemente realizando backups del workspace si quiere evitar perder el trabajo realizado.
- Las variables de entorno no se validan ni deben ser definidas antes de ser usadas. Esto provoca que un error en el nombrado de la variable, por ejemplo, quitando una letra, sea interpretado por Watson como una variable diferente, ya que la interfaz no autodetecta errores de este tipo. Esto dificulta la depuración, porque hay que revisar uno a uno los editores JSON de cada nodo para comprobar si la variable es correcta o no.
- Es necesario una conexión a internet para poder trabajar, porque la conversación se configura directamente sobre el espacio de trabajo de la instancia de Watson Conversation, y por tanto no es posible trabajar en local y liberar los cambios cuando ya están probados.
- No dispone de un sistema de backups del workspace. La única alternativa es exportar a JSON el workspace y versionar este JSON.
- El número de nodos para conversaciones reales es bastante grande, el flujo se vuelve complejo y se hace inmanejable conforme aumentan las intenciones a comprobar, condiciones de salida, etc. En flujos de conversación de cierta envergadura no es operativo.
- El flujo de la conversación no es reutilizable.
- No se valida el desbalanceo de ejemplos en las intenciones. Se han introducido muchos más ejemplos en una de las intenciones, y se ha comprobado que en este caso esa intención será más probablemente detectada que el resto.

En general, creo que se ha intentado enfocar la interfaz del flujo de la conversación para que sea implementado de forma gráfica y facilitar su uso, pero creo que el efecto es el inverso, porque este planteamiento dificulta y limita el desarrollo de proyectos de cierta envergadura o en los que colaboren varios desarrolladores, que suele ser lo habitual. Creo que daría mayor flexibilidad si se pudiese incrustar código en la propia herramienta y no tener que usar nodos obligatoriamente, o si existiera una forma de definir la conversación programáticamente.

2) Slots

El uso de slot nos ayuda a asegurarnos de que la información necesaria es proporcionada, y si no es así, se solicita hasta que lo sea, intercambiando respuestas adecuadas en cada caso. Si los slots son obligatorios no se sale del nodo de slot hasta que todos los slots están llenos.

Esta funcionalidad se añadió recientemente, está disponible sólo en algunos idiomas, y aún está en beta, y se aprecia en su funcionamiento que aún no está muy depurada.

Concretamente en este trabajo he intentado usar los slots para optimizar el número de nodos de la conversación, pero no he podido explotar esta funcionalidad como imaginaba en un principio, ya que he encontrado muchas limitaciones cuando se utiliza con entidades opcionales, es decir, entidades que no tienen que ser obligatoriamente aportadas por el usuario para poder proporcionar una respuesta. En este caso he encontrado las siguientes limitaciones y dificultades:

- Es necesario añadir nodos adicionales a los nodos de slot para comprobar si las entidades correspondientes están llenas. Es decir, nodos que comprueban si el nodo de slot detectó la entidad en la entrada del usuario y la guardó en la variable de contexto correspondiente. Con lo cual no se consigue optimizar el número de nodos ni tampoco el flujo.
- Como alternativa pensé utilizar nodos de slots diferentes para cada variable, pero comprobé que esto no era viable porque la respuesta del usuario puede contener información de otras entidades, y en ese caso estaría dejando sin capturar dicha información, o incluso preguntando repetidamente al usuario por dicha información aun cuando ya ha sido proporcionada.
- Tampoco he encontrado la forma de inicializar variables en los nodos de slots, es decir añadir configuración que deba ejecutarse siempre al entrar en el nodo, independientemente de si se detecta o no una determinada entidad.
- También encontré en las pruebas que realicé que es necesario añadir una frase en la condición del nodo de slot "*If \$variable is not present then ask*". Cuando la dejaba vacía no funcionaba correctamente.

3) Handlers

Si mientras se está atendiendo una intención se detecta otra intención distinta, no se atiende la nueva hasta finalizar la primera. Watson dispone de mecanismos para manejar esta situación, los llamados "handlers", pero esta funcionalidad está algo limitada, porque solo está disponible en los nodos que manejan slots. Además, sólo permite alterar el flujo de la intención para dar una determinada respuesta al usuario en caso de que el slot no esté lleno. Después de responder al usuario la conversación sigue su curso, y sólo es posible alterarla utilizando valores ficticios con los que llenar los slots para así forzar que el flujo salga del nodo slot, y a continuación evaluar en el resto de nodos la existencia de valores ficticios.

Manage handlers for "check option"

Handlers are how your bot will respond when the user's answer to a prompt is not found. These handlers will be checked before trying the "Not found" responses in a slot.

If answer to any prompt is not found and:

If bot recognizes	Respond with	
1 Enter an intent, entity or context vari...	Enter a response...	 
 Add handler		

Figura 109 – *Handlers* en Watson Conversation

En general se aprecia que el servicio Watson Conversation está en una etapa inicial, resuelve bien los problemas básicos de la implementación de chatbots, pero aún tiene mucho margen de mejora. Necesita consolidarse y solucionar con solvencia los problemas comunes y otros más avanzados de la implementación de los sistemas de diálogo.

Fruto de lo anterior, y también de sus campañas estratégicas y de marketing, se realizan constantes actualizaciones del servicio. Cada poco tiempo se libera una nueva versión con nueva funcionalidad, mejoras, correcciones, etc. Normalmente las nuevas funcionalidades se publican como beta y para inglés, y conforme se va depurando se añade a otros idiomas. Todos los servicios suelen requerir un parámetro de entrada llamado “versión”, y sirve para asegurar la retrocompatibilidad. En este parámetro el desarrollador debe proporcionar la fecha de implementación de su aplicación, para que cuando la API evolucione no sea necesario cambiar el código, automáticamente se comprobará dicha fecha y se adaptarán los parámetros de la llamada al servicio.

Las grandes campañas de marketing de IBM, y la popularidad que ha ganado Watson en los últimos años, y especialmente tras su victoria en Jeopardy!, han contribuido a que se exageren las expectativas sobre la tecnología IBM Watson. Prueba de ello es su precio, mucho mayor que el resto de APIs comerciales.

4.3 Resumen del trabajo realizado y cumplimiento de objetivos

Con este trabajo se han logrado los objetivos fijados al inicio (apartado 1.2). Repasemos estos objetivos, y la metodología seguida para cumplirlos.

En primer lugar, se planteó realizar una introducción a los sistemas de diálogo, y más concretamente a la construcción de asistentes conversacionales o chatbots. Para llevar a cabo este objetivo, en el apartado 3.5.1 se realizó un estudio de la estructura típica de los sistemas de diálogo, se formalizaron sus requisitos y se enumeraron algunas de las principales líneas de investigación abiertas en la actualidad. Durante todo el trabajo se han tenido presentes estas cuestiones en el diseño e implementación del prototipo, y se ha profundizado en ellos para materializarlos en el caso de uso propuesto en este trabajo: un asistente virtual para frigorífico inteligente.

El siguiente objetivo planteado fue el estudio de los principales servicios e interfaces de procesado de lenguaje natural existentes en la actualidad. En el apartado 2 se recoge el estudio comparativo realizado con respecto a los siguientes criterios: rendimiento, cobertura de idiomas, precio, fortaleza de la compañía proveedora, existencia de módulos predefinidos, control de versiones, flexibilidad, autonomía, accesibilidad, integración, y reconocimiento y síntesis de voz. Como conclusión de dicho estudio, se obtiene que *API.ai*, *Luis* y *Watson* tienen características similares, y que cualquiera de ellas puede ser adecuada para los requisitos impuestos en este trabajo. Finalmente se selecciona Watson porque es ligeramente mejor en el reconocimiento de intenciones, y aunque presenta la desventaja de tener el precio más elevado, no nos preocuparemos de eso en este trabajo, ya que disponemos de una cuenta académica gratuita que nos servirá para evaluar si realmente merece la pena pagar el precio que se pide por su uso.

Otro objetivo que se planteó fue el estudio teórico de los servicios IBM Watson disponibles en la plataforma cloud Bluemix, y cuyo detalle puede encontrarse en el apartado 2.3.

A continuación, para materializar los resultados del estudio teórico realizado hasta ese momento, y poder evaluar los aspectos a tener en cuenta en la construcción de un chatbot, se propuso desarrollar un prototipo de asistente conversacional para frigorífico inteligente. Inicialmente se impusieron 9 requisitos funcionales (ver apartado 1.3), y se exigió además que la conversación fuese abierta. Todos estos requisitos han quedado cubiertos con la definición, entrenamiento e implementación de las intenciones descritas en los apartados 3.5.3 y 3.5.4.

Finalmente, en las próximas secciones haremos un repaso de las principales dificultades encontradas durante la ejecución de este trabajo, evaluaremos los resultados obtenidos y realizaremos un análisis crítico de los servicios Watson utilizados en la construcción del prototipo (apartado 4.2).

4.4 Dificultades encontradas durante el desarrollo

Además de las ya mencionadas limitaciones experimentadas en el uso de Watson (ver apartado 4.2), y las dificultades habituales a las que nos enfrentamos en la construcción de cualquier chatbot que utilice IA, y que dé libertad al usuario para expresarse, en este apartado enumeraremos otras dificultades y limitaciones más específicas de nuestro caso de uso.

Las dificultades más destacables, bien porque son las que han requerido más esfuerzo para su resolución, o porque han tenido más impacto en los resultados finales del prototipo son las siguientes.

La limitación más importante la he encontrado con el servicio de recetas **Food2Fork**, principalmente por los siguientes motivos:

- Solo está disponible en inglés, esto ha forzado a que el chatbot sea implementado en este idioma.
- El servicio no es estable, en el transcurso de la implementación del prototipo he experimentado varias caídas del sistema que bloquean la obtención de resultados tanto con la API como con la interfaz web. Este es un problema serio, porque durante el tiempo que el servicio esté caído el chatbot no podrá proporcionar recetas a los usuarios.
- La base de datos de recetas no es muy rica, muchas recetas que he buscado no están disponibles.
- Las recetas no se actualizan con frecuencia, durante el desarrollo de este trabajo no se ha encontrado nuevas recetas en las búsquedas. Las recetas más populares o las mejor puntuadas siempre son las mismas.
- El sistema de búsqueda no es potente. Hace búsquedas literales del texto que recibe como entrada, si tenemos la suerte de encontrar una receta que contenga las palabras introducidas habremos obtenido la receta buscada, pero si no contiene esas palabras en concreto, no devolverá resultados. Tampoco permite búsquedas por ingredientes, incluir o excluir ciertos ingredientes, o búsquedas por intolerancias, tipos de cocina, etc. Tampoco cuenta con una taxonomía de alimentos como en el caso de Spoonacular.
- No es un servicio muy usado y la documentación es escasa.

Con respecto a **Slack**, en general la experiencia ha sido muy positiva, se aprecia que es una plataforma muy volcada en el desarrollo de chatbots, con una comunidad de desarrolladores muy amplia, y que cuenta con una gran cantidad de chatbots ya desplegados. Sin embargo, para la descarga de imágenes vía API no hay métodos en la SDK de Python que faciliten el trabajo. La descarga de las imágenes que comparte el usuario con nuestro chatbot se ha resuelto de una forma algo artesanal, tuve que buscar la forma de llevarlo a cabo analizando y experimentando con el JSON que se obtiene como respuesta del envío de mensajes al canal.

Otro problema al que me he enfrentado ha sido la falta de conversaciones reales con las que entrenar. Esto ha obligado a entrenar con ejemplos inventados por mí, en un idioma que no domino.

Todas estas cuestiones, en mayor o menor medida, han tenido impacto en la solución final y en los resultados obtenidos. Algunas limitaciones, como por ejemplo la de Slack con la descarga de imágenes, son transparentes para el usuario porque he podido resolverlo en el código. Pero todas las relativas al servicio de recetas

Food2Fork han restado calidad al prototipo, y han limitado sus posibilidades y funcionalidades. Por ejemplo, hubiera resultado muy interesante que el chatbot pudiera buscar recetas que contengan exclusivamente ingredientes con los que cuenta el usuario en su nevera, sin necesidad de utilizar otros ingredientes adicionales con los que no cuenta. Pero esto no es posible actualmente si usamos *Food2Fork*, porque el servicio devuelve recetas con ciertos ingredientes que facilita como entrada, pero no garantiza que tenga que usar otros de los que no dispongo.

Con respecto al idioma en el que se ha implementado el chatbot, ha supuesto un mayor esfuerzo para el entrenamiento de entidades e intenciones, y ha hecho que los resultados no sean tan ricos como si se hubiera implementado de mi idioma nativo, en el que puedo aportar una mayor riqueza de matices, y ejemplos, más expresivos y fluidos. En contrapartida, la ventaja de haberlo implementado en inglés es que los servicios de Watson son más ricos en funcionalidad y más precisos para este idioma que para el resto.

4.5 Evaluación del prototipo

Los resultados obtenidos con este trabajo suponen una forma novedosa y divertida de ayudar a mejorar la dieta de las personas que conversan con **yumyumBot**. Su uso permite al usuario ahorrar tiempo planificando su dieta o la lista de la compra, evitar desperdicio de alimentos, descubrir nuevos sabores, trucos y técnicas de cocina. Y a la vez, facilita estar en contacto con otras personas, y compartir inquietudes culinarias.

Existen aplicaciones que ayudan a resolver estos problemas, pero este prototipo ofrece además una interfaz en lenguaje natural a través de una plataforma de mensajería móvil o de escritorio. Otro elemento diferenciador respecto a otros chatbots o aplicaciones es que **yumyumBot** utiliza reconocimiento de imágenes para proporcionar las instrucciones con las que elaborar un plato del que sólo contamos con una foto.

Este prototipo es ambicioso para un trabajo fin de máster, porque obtener buenos resultados cuando se da libertad al usuario en la conversación requiere bastante esfuerzo, y un buen diseño del flujo de la conversación, ya que una conversación real tiene infinidad de posibilidades y debemos contemplar la mayoría de ellas en nuestro diseño si queremos conseguir una experiencia de usuario satisfactoria.

Los resultados obtenidos son buenos, el chatbot es operativo y las respuestas proporcionadas son adecuadas. Y además proporciona bastantes funciones interesantes para el día a día del usuario. Sin embargo, aún tiene mucho margen de mejora, los resultados en el reconocimiento de intenciones y entidades pueden mejorarse con más entrenamiento y con ejemplos de conversaciones reales. Por ejemplo, las entidades `@dish` e `@ingredients` deben ser entrenadas con más ejemplos para que el chatbot sea capaz de interceptarlas.

Otra mejora interesante sería que la entidad `@ingredients` fuera una lista de valores, que sí está soportado en Watson Conversation, pero actualmente el chatbot sólo captura uno de ellos, aunque haya reconocido otros en la entrada del usuario, porque no es una lista sino un único elemento.

El flujo de la conversación necesita ser puesto a prueba por diferentes usuarios para explorar diferentes caminos, y reconfigurarlo en caso de que se detecten comportamientos no deseados o imprevistos en el diseño.

Con respecto a la base de datos local de productos (contenido del frigorífico) sí que encontramos mucho margen de mejora, ya que actualmente es muy manual: el usuario debe actualizar cada vez que introduzca un nuevo producto, modifique su cantidad o elimine de su interior. Lo ideal sería integrar dicha base de datos con un frigorífico real, con sensores capaces de detectar automáticamente la existencia o no de un cierto producto, así como de la cantidad disponible y de su fecha de caducidad.

En cuanto a las limitaciones en las búsquedas de recetas con el servicio *Food2Fork*, existen varias alternativas posibles, una de ellas es utilizar un servicio de recetas diferente, más robusto como puede ser *Spoonacular*. Otra opción, mucho más costosa, podría ser recopilar recetas de diferentes fuentes para crear nuestra propia base de datos de recetas.

Hay algunos aspectos avanzados de la conversación que no están resueltos en el prototipo, bien por limitaciones en el servicio Watson Conversation, o bien porque son líneas de investigación abiertas y requerirían un estudio completo solo para ellas como, por ejemplo, ofrecer la posibilidad de atender dos intenciones a la vez, sin necesidad de finalizar con una de ellas para comenzar con la otra, o la capacidad de captar dobles sentidos, bromas, etc.

4.6 Recomendaciones

Para finalizar, se ofrecen algunas recomendaciones, fruto de la experiencia que ha aportado enfrentarnos a la construcción de un chatbot y a la resolución de los problemas mencionados con anterioridad:

- Es importante comprender que no todos los chatbots son iguales, y que antes de lanzarnos a implementar el nuestro hay que analizar y acotar el problema que se pretende resolver, la información necesaria para poder proporcionar la respuesta, la complejidad de la conversación, los usuarios a los que va dirigido, los canales de comunicación y la experiencia de usuario que se quiere conseguir. Todo esto tendrá un impacto en el esfuerzo necesario para su implementación. Este esfuerzo no debe ser subestimado, porque según los expertos, el desarrollo de un chatbot universal que comprenda el lenguaje natural y evalúe el contexto lleva años de trabajo para un equipo de programadores experimentados. E incluso en este caso, dichos programas deberían mejorarse y reentrenarse constantemente durante el servicio.
- Cada paso del diálogo tiene miles de posibles respuestas por parte del usuario, por muy buen trabajo de diseño del diálogo que hayamos hecho es imposible conocer todas las respuestas o caminos que puede tomar la conversación. Por ello es vital registrar qué responde el usuario en qué pasos, a fin de reprogramar (entrenar) nuestro chatbot para que pueda procesar esas respuestas y tomar decisiones.
- Es necesario controlar el tiempo de respuesta y reducir al mínimo el número de preguntas dirigidas al usuario.
- Los elementos visuales refuerzan la comunicación, por ejemplo, la inclusión de iconos o gifs en las respuestas proporcionan conversaciones más fluidas y menos programadas.
- Una técnica muy extendida para facilitar el trabajo con chatbots es la de sugerir posibles respuestas al usuario para que, de forma casi inconsciente, elija una y sepamos donde dirigir la conversación.
- Tener siempre presentes los aspectos legales que puedan afectar a la operativa de nuestro chatbot, por ejemplo en el tratamiento y almacenamiento de información personal, en las comunicaciones comerciales, o en la responsabilidad asumida por chatbots que facilitan información financiera, médica, etc.
- El chatbot será la imagen de cara al público, habrá que dotarlo de personalidad atractiva y de los valores de la empresa a la que representa. Involucrar diferentes perfiles en la construcción del chatbot puede ser una buena estrategia, ya que esta tarea no es exclusivamente de los perfiles técnicos, con conocimientos de computación.

5 LÍNEAS FUTURAS

S

ería muy interesante abordar en trabajos posteriores las ideas que se muestran a continuación. Con estas propuestas se lograría explotar en mayor medida las posibilidades que ofrece nuestro chatbot, así como reforzar las debilidades detectadas en el prototipo actual.

5.1 Reconocimiento vocal

La forma natural que tenemos de comunicarnos es mediante una conversación hablada. Por tanto, añadir a nuestro prototipo la capacidad de interpretar la voz, y también de sintetizarla, aportaría una gran mejora en cuanto a usabilidad, haría más sencilla la interacción con el chatbot, especialmente para las personas de mayor edad, que suelen sentirse menos cómodas utilizando texto para comunicarse. Además, generalmente, este medio es más rápido que el textual.

Para llevar a cabo lo anterior sería necesario:

- **Transcribir la voz:** Watson Conversation necesita texto como entrada, por tanto, es necesario convertir previamente la voz a texto. Se podría usar por ejemplo Watson *Speech to Text*, que tiene a nuestra disposición varios modelos en inglés para esta finalidad.
- **Sintetizar la voz:** la respuesta final que se proporciona al usuario también sería mediante voz. Podríamos usar para ello el servicio Watson *Text to Speech*, que convierte texto en voz.
- **Canal de comunicación** que soporte voz. Slack permite realizar llamadas de voz entre usuarios, y también podrían usarse grabaciones a través de sus chats.
- **Grabaciones** de conversaciones reales con las que entrenar el servicio de transcripción.

5.2 Funcionalidad lista de la compra

Se podría añadir una nueva funcionalidad que autocomplete la lista de la compra del usuario con los ingredientes que haya que reponer, bien porque han caducado, están acabándose, o ya se han terminado. Imaginemos que el usuario ha planificado una receta y no dispone de ciertos ingredientes, el chatbot también podría encargarse de incluirlos en la lista para que el usuario se libere de esta tarea.

Hasta ahora todas las interacciones propuestas con el chatbot consisten en búsquedas de información y presentación de resultados mediante respuestas. Podríamos ir un paso más allá y convertir el chatbot en lo que actualmente se conoce como *chatOps*⁴¹, chatbots capaces de realizar operaciones por nosotros, es decir tareas específicas que hayan sido solicitadas durante la conversación, y son muy populares en el mundo IT y en devOps. Por ejemplo, el chatbot podría realizar el pedido online de forma automática al supermercado utilizando la lista de la compra que ha elaborado.

Para esto se necesitaría:

- Usar la funcionalidad de notas que hay disponible en Slack, o integrar el chatbot con alguna herramienta o aplicación de notas online, como *Google Keep* o *Evernote*, y de este modo compartir la lista con el usuario, y que éste también pueda modificarla.
- Integrar el chatbot con los supermercados online y pasarelas de pago.
- Añadir mecanismos que, a partir de una receta, sean capaces de extraer los ingredientes de los que ya dispone el usuario, y aquellos de los que no dispone en suficiente cantidad. Con *Spoonacular* este

⁴¹ https://www.gmv.com/blog_gmv/the-power-of-a-good-conversation/

trabajo se facilitaría, porque es capaz de proporcionar un vector de los ingredientes necesarios para cada receta.

5.3 Integración con frigorífico IoT

Se propone integrar el prototipo con un frigorífico real, dotado de sensores o de sistemas de reconocimiento visual capaces de disponer en todo momento de la información detallada sobre los productos que guarda en su interior (cantidad, características, fecha de caducidad, etc.)

Con esto se consigue que la base de datos de productos se actualice automáticamente cada vez que el usuario introduzca un nuevo producto o consuma cierta cantidad de un producto.

5.4 Multi-idioma

Con esta mejora se pretende que el chatbot esté disponible en diferentes idiomas, y no solo en inglés. Existen varias aproximaciones para ello. Una de ellas es añadir un traductor a la entrada y a la salida del usuario, pero esto introduciría ruido. Lo ideal sería implementar el chatbot en cada idioma, y añadir una primera etapa que detecte el idioma del usuario y elija qué instancia del chatbot se debe usar. El servicio Watson *Translate* puede facilitar la implementación de este módulo de detección del lenguaje.

5.5 Coach nutricional

Se podría añadir funcionalidad que ayude al usuario a llevar una dieta sana, que le eduje y que coopere con él para cumplir sus objetivos nutricionales, bien porque necesite seguir ciertas pautas o restricciones a causa de alguna enfermedad o intolerancia, o para mejorar su rendimiento deportivo, perder o ganar peso, etc.

El chatbot ofrecería las mejores recomendaciones de dietas en cada caso, resolvería las dudas del usuario, ayudaría a consumir las calorías recomendadas, y motivaría al usuario para cumplir sus objetivos.

Para llevar esto a cabo se necesitaría disponer del valor calórico y nutricional de los alimentos, y de una amplia base de datos de dietas que cumplan los criterios requeridos.

5.6 Conversaciones personalizadas y análisis de sentimientos

Con esta mejora se propone la extracción de información emocional a partir de las expresiones del usuario. Es decir, extraer de la entrada del usuario su opinión sobre determinadas cuestiones, o su estado de ánimo. Esto es algo que los humanos realizamos de forma continua durante la conversación, y que nos permite tomar decisiones más concretas, y diferentes a las que tomariamos sin dicha información. Esto puede ayudarnos a mejorar los lazos con el usuario, ganar su confianza y dirigirnos a él de formas diferentes según su estado de ánimo.

5.7 Notificaciones productos perecederos

Consiste en añadir notificaciones proactivas cuando el chatbot detecte que un producto está próximo a caducar. En el prototipo actual ya se ofrecen informes de productos perecederos, distinguiendo los que están en buenas condiciones, los próximos a caducar, y los que ya han caducado, pero lo que aquí se propone es avisar al usuario proactivamente sin tener que esperar a que éste pregunte.

Para esto se necesita ejecutar una comprobación diaria del estado de los productos, y que notifique al usuario las conclusiones, por ejemplo, cuando inicie la conversación.

5.8 Cuadro de mando

Se trata de explotar la información disponible a través de informes online. Por ejemplo, se podrían mostrar evoluciones temporales de los productos más consumidos, de las calorías consumidas, de los productos que más desperdicia, del dinero que gasta en alimentos, de las recetas más utilizadas, etc.

APÉNDICE I: CÓDIGO

El código de la aplicación se puede encontrar a continuación y también está disponible en [Github](#).

```
1. #!/usr/bin/python
2. # coding=utf-8
3.
4. ##### LIBRARIES #####
5. import pprint
6. import os
7. import time
8. import datetime
9. import psycopg2
10. import requests
11. import urllib.request
12. import json
13. from slackclient import SlackClient
14. from watson_developer_cloud import VisualRecognitionV3 as VisualRecognition
15. from watson_developer_cloud import ConversationV1 as Conversation
16.
17. ##### CONSTANTS #####
18. # Slack bot id
19. BOT_ID = os.environ.get("BOT_ID")
20. # Slack token
21. SLACK_BOT_TOKEN=os.environ.get('SLACK_BOT_TOKEN')
22. # String to identify messages with bot as recipient
23. AT_BOT = "<@" + BOT_ID + ">"
24. # Delay between reading from firehose
25. READ_WEBSOCKET_DELAY = 1
26.
27. # Food2fork API key
28. FOOD2FORK_KEY=os.environ.get('FOOD2FORK_KEY')
29.
30. # Postgres database connection string
31. DB_STRING_CONNECTION = "host='localhost' dbname='smartfridge' user='postgres' password='postgres'"
32.
33. # Watson Visual Recognition version (to ensure backward compatibility)
34. VISUAL_RECOGNITION_VERSION = '2017-10-15'
35. # Watson Visual Recognition base url
36. VISUAL_RECOGNITION_URL = 'https://gateway-a.watsonplatform.net/visual-recognition/api'
37. # Watson Visual Recognition API key
38. VISUAL_RECOGNITION_KEY = os.environ.get('VISUAL_RECOGNITION_KEY')
39.
40. # Watson Conversation version (to ensure backward compatibility)
41. CONVERSATION_VERSION = '2017-09-23'
42. # Watson Conversation username
43. CONVERSATION_USERNAME = os.environ.get('CONVERSATION_USERNAME')
44. # Watson Conversation password
45. CONVERSATION_PASSWORD = os.environ.get('CONVERSATION_PASSWORD')
46. # Watson Conversation workspace identifier
47. CONVERSATION_WORKSPACE = os.environ.get('CONVERSATION_WORKSPACE')
48. # Watson Conversation base url
49. CONVERSATION_URL = 'https://gateway.watsonplatform.net/conversation/api'
50.
51. # Number of remaining days to consider a product as next to expire
52. DAYS_TO_EXPIRE = 7
53. # Number of dish options provided to the user
54. TOTAL_NUMBER_OPTIONS = 6
55.
56.
```

```

57. class SmartFridge():
58.     def __init__(self):
59.         self.context = {}
60.         self.option_dict = {}
61.         self.intents = []
62.         self.entities = []
63.         self.recipe_options = []
64.         self.database_cursor = None
65.
66.         # Reset environment variables
67.         self.context['search_recipe'] = False
68.         self.context['image_recipe'] = False
69.         self.context['suggest_dish'] = False
70.         self.context['yum_sugest'] = False
71.         self.context['summary'] = False
72.         self.context['option'] = None
73.         self.context['cuisine_type'] = None
74.         self.context['ingredients'] = None
75.         self.context['intolerances'] = None
76.         self.context['dish'] = None
77.         self.context['counter'] = 0
78.         self.context['insult_counter'] = 0
79.
80.
81.         # Services initialization
82.         # Slack client instance
83.         self.slack_client = SlackClient(SLACK_BOT_TOKEN)
84.
85.         # Watson Conversation service instance
86.         self.conversation = Conversation(version = CONVERSATION_VERSION,
87.                                         username = CONVERSATION_USERNAME,
88.                                         password = CONVERSATION_PASSWORD,
89.                                         url = CONVERSATION_URL)
90.
91.         # Watson Visual Recognition service instance
92.         self.visual_recognition = VisualRecognition(version=VISUAL_RECOGNITION_VERSION,
93.                                                       url=VISUAL_RECOGNITION_URL,
94.                                                       api_key=VISUAL_RECOGNITION_KEY)
95.         # Database connection
96.         self.database_connection(DB_STRING_CONNECTION)
97.
98.
99. ###### CONVERSATION MANAGEMENT ######
100.        # Receives commands directed at the bot and determines if they
101.        # are valid commands. If so, then acts on the commands. If not,
102.        # returns back what it needs for clarification.
103.        def handle_command(self, command):
104.
105.            response = "Not sure what you mean. Please reword your request"
106.
107.            print('\n\nINPUT = {}\n'.format(command))
108.
109.            # Processing of the response
110.            if command == 'download_file_format_error':
111.                response = 'The file extension is not valid. Try with JPG or PNG.'
112.            # Food image recognition
113.            elif command.startswith('photo'):
114.                self.send_response('Please, give me a second... :hourglass_flowing_sand:')
115.                self.context['image_recipe'] = "true"
116.                response_text, intent, entity=self.update_conversation_context()
117.                response = self.image_food_recognition()
118.            else:
119.                response_text, intent, entity = self.msg_to_conversation(command)
120.                print('intent = {}'.format(intent))
121.                print('entity = {}'.format(entity))
122.

```

```

123.         # A suggestion is provided to the user because the required information
124.         # is not provided by the user after several attempts $yum_sugest
125.         if(self.context['yum_sugest'] == 'true'):
126.             self.send_response(response_text)
127.             response = self.yum_suggestion()
128.
129.         # $suggest_dish
130.         elif(self.context['suggest_dish'] == 'true'):
131.             self.send_response(response_text)
132.             response = self.suggest_dish()
133.
134.         # $summary
135.         elif(self.context['summary'] == 'true'):
136.             self.send_response(response_text)
137.             response = self.get_db_summary()
138.
139.         # $search_recipe
140.         elif(self.context['search_recipe'] == 'true'):
141.             self.send_response(response_text)
142.             response = self.get_recipe()
143.
144.         # #get_recipe
145.         elif intent == 'get_recipe':
146.             if(self.context['search_recipe'] == 'true'):
147.                 response = self.get_recipe()
148.             elif(self.context['yum_sugest'] == 'true'):
149.                 response = self.yum_suggestion()
150.             else:
151.                 response= response_text
152.
153.         # #suggest_dish
154.         elif intent=='sugest_dish':
155.             if(self.context['suggest_dish'] == 'true'):
156.                 response = self.suggest_dish()
157.             elif(self.context['yum_sugest'] == 'true'):
158.                 response = self.yum_suggestion()
159.             else:
160.                 response = response_text
161.
162.         # #available_ingredients
163.         elif intent=='available_ingredients':
164.             ingredients = self.context['ingredients']
165.             if ingredients != None:
166.                 self.send_response(response_text)
167.                 response = self.get_ingredients_information(ingredients)
168.             else:
169.                 self.send_response(response_text)
170.                 response = self.get_db_summary()
171.
172.         # #needed_ingredients
173.         elif intent == 'needed_ingredients':
174.             response = response_text
175.
176.         # #select_option
177.         elif intent == 'select_option':
178.             option_response = self.select_option()
179.             if option_response != '':
180.                 response = option_response
181.             else:
182.                 response = response_text
183.
184.         # #negative_reaction
185.         elif intent == 'negative_reaction':
186.             response = response_text
187.
188.

```

```

189.         # #anaything_else
190.     else:
191.         response = response_text
192.
193.     # Send the corresponding response to the user interface (slack)
194.     self.send_response(response)
195.
196.
197. def select_option(self):
198.     response = ''
199.     if (self.context['option']!= None) and (len(self.recipe_options) > 0):
200.         index = self.parse_to_valid_index(self.context['option'])
201.         if index != None:
202.             selection = self.recipe_options[index]
203.             response = 'Ok, good choice! The {} recipe below: '.format(selection)
204.             response = response + '\n' + self.get_ingredients(self.option_dict[selection])
205.             self.recipe_options = []
206.     return response
207.
208.
209. def parse_to_valid_index(self, opt):
210.     value = None
211.     if opt in range(1, TOTAL_NUMBER_OPTIONS+1):
212.         value = opt - 1
213.     else:
214.         value = None
215.     return value
216.
217.
218. # Prioritizes the use of ingredients that are about to expire
219. # Excludes expired products
220. def yum_suggestion(self, n_options=6):
221.     header = 'I have found the following recipes for you:'
222.     response = ''
223.     footer = 'Please, provide a valid option from 1 to 6'
224.
225.     self.recipe_options= []
226.     self.context['yum_sugest'] = False
227.
228.     if n_options > 0:
229.         available_ingredients_options = []
230.         top_rated_options = []
231.         trending_options = []
232.
233.         # n_recipes recipe suggestion by using the n_ingredients with closest expiration date
234.         # and biggest quantity
235.         available_ingredients_options=self.get_recipe_options_from_available_ingredients(n_options=2,
236.                                         n_ingredients=2)
237.
238.         top_rated_options = self.get_top_rated_recipe_options(
239.             (n_options=((TOTAL_NUMBER_OPTIONS - len(available_ingredients_options)) // 2)
240.              + ((TOTAL_NUMBER_OPTIONS - len(available_ingredients_options)) % 2)))
241.
242.         trending_options = self.get_trending_recipe_options(
243.             n_options=(TOTAL_NUMBER_OPTIONS - len(available_ingredients_options)) // 2))
244.
245.         self.recipe_options = available_ingredients_options + top_rated_options + trending_options
246.         for i, recipe in enumerate(self.recipe_options[:n_options]):
247.             response = response + '\n' + '[{0}] : {1}'.format(i+1, recipe)
248.
249.     if response == '':
250.         response = ':disappointed: Sorry, no recipes found for your request. Please, try a new search'
251.     else:
252.         response = header + '\n' + response + '\n' + footer
253.     return response

```

```

254.
255.     def suggest_dish(self):
256.         print('ingredients={0}, cuisine type={1}, intolerances={2}'.format(self.context['ingredients'],
257.                                         self.context['cuisine_type'],
258.                                         self.context['intolerances']))
259.
260.         query = ''
261.         response = ':disappointed: Sorry, no recipes found for your request. Please, try a new search'
262.
263.         if (self.context['suggest_dish']):
264.             if self.context['ingredients'] != None:
265.                 query = query + self.context['ingredients']
266.             if self.context['cuisine_type'] != None:
267.                 query = query + ' ' + self.context['cuisine_type']
268.             if self.context['intolerances'] != None:
269.                 query = query + ' ' + self.context['intolerances']
270.             if query != '':
271.                 print('Buscando receta para: <> {} <>'.format(query))
272.                 response = self.get_ingredients(self.get_recipe_id(query))
273.         return response
274.
275.
276.     def get_db_summary(self):
277.         all_products = []
278.         expired_products = []
279.         products_to_expire = []
280.         header = ''
281.         recap = ''
282.
283.         # All products
284.         query_all = "SELECT name " \
285.                     "FROM products " \
286.                     "ORDER by name"
287.         all_products = self.fetch_content(query_all)
288.
289.         # Expired products
290.         query_expired = "SELECT name " \
291.                         "FROM products " \
292.                         "WHERE date(expiration_date)<=current_date " \
293.                         "ORDER by name"
294.         expired_products = self.fetch_content(query_expired)
295.
296.         # Products to expire in next DAYS_TO_EXPIRE days
297.         query_to_expire = "SELECT name " \
298.                         "FROM products " \
299.                         "WHERE date(expiration_date)>current_date " \
300.                         "AND date(expiration_date)<=current_date + interval '{} days' " \
301.                         "ORDER by name".format(DAYS_TO_EXPIRE)
302.
303.         products_to_expire= self.fetch_content(query_to_expire)
304.
305.         if len(all_products)>0:
306.             header = '\n\nThere are {0} products in total, ' \
307.                   '{1} products are already expired and {2} products will expire soon. ' \
308.                   format(len(all_products), len(expired_products), len(products_to_expire))
309.
310.             if(len(expired_products)>0):
311.                 footer = '\n\nThrow out the expired foods. '
312.                 recap = recap + '\n\n' + ':recycle: *Already expired*: {0}.' \
313.                         .format(', '.join(map(str, expired_products)))
314.             else:
315.                 recap = recap + '\n\n' + 'Great! :clap: There are no expired products.'
316.             if(len(products_to_expire)>0):
317.                 footer = footer + 'Consider using the foods to expire as soon as possible.'
318.                 recap = recap + '\n\n' + ':alarm_clock: *To expire*: {0}.' \
319.                         .format(', '.join(map(str, products_to_expire)))

```

```

320.     else:
321.         recap = recap + '\n\n' +
322.         'Congrats! :v: There are no products to expire in the next {0} days.'.format(DAYS_TO_EXPIRE)
323.     else:
324.         recap= ':-1: You have the fridge EMPTY! :disappointed_relieved:' \
325.             'Make the purchase if you do not want to starve. ' \
326.             'Today you will have to order food at home.'
327.     return(header + recap + footer)
328.
329.
330. def image_food_recognition(self):
331.     food, score = self.get_food_from_image('./download/food.jpg')
332.     if (food != 'non-food'):
333.         response = 'Uhm... :yum: :yum: This looks really good.
334.                                     I think (score: {1}) it is... *{0}*'\ \
335.             .format(food, score)
336.         self.send_response(response)
337.         response = '\n' + smartfridge.get_ingredients(self.get_recipe_id(food))
338.     else:
339.         response = 'Are you sure it is edible? I do not recognize food in this image.
340.                                     \nPlease, try with another one.'
341.
342.     return response
343.
344.
345. def get_recipe(self):
346.     recipe = ':disappointed: Sorry, no recipes found for your request. Please, try a new search'
347.     print('dish = {}'.format(self.context['dish']))
348.     print('search_recipe = {}'.format(self.context['search_recipe']))
349.     if (self.context['dish'] != None and self.context['search_recipe']):
350.         print('Buscando receta para: <> {}'.format(self.context['dish']))
351.         recipe=self.get_ingredients(self.get_recipe_id(self.context['dish']))
352.     return recipe
353.
354.
355. def get_ingredients(self, recipeId):
356.     print('recipe id = {}'.format(recipeId))
357.     response = ':disappointed: Sorry, no recipes found for your request. Please, try a new search'
358.     if recipeId != None:
359.         recipe = self.get_recipe_from_id(recipeId)
360.         if recipe and 'recipe' in recipe:
361.             ingredients=[]
362.             source = '\nHere, you can find the *method of cooking*: {}.\n'.format(recipe['recipe']['source_url'])
363.             for ingredient in (recipe['recipe']['ingredients']):
364.                 ingredients.append(ingredient)
365.                 str_ingredients= '\nI found *{}*. To cook this you need the following *ingredients*:\n'.format(recipe['recipe']['title']) + \
366.                               '\n - {}'.format('\n - '.join(map(str, ingredients)))
367.             response = str_ingredients + '\n' + source
368.     return response
369.
370.
371.
372.
373.
374. def get_ingredients_information(self, ingredients):
375.     info = ""
376.     try:
377.         records = self.get_db_information_about_ingredients(ingredients)
378.         if len(records)>0:
379.             for r in records:
380.                 # Not expired product
381.                 if r[1].date() >= datetime.datetime.now().date():
382.                     info = info + '\n' + 'There are {0} grams of {1}, the expiration date is {2}'.\
383.                         format(round(r[2], 0), r[0], r[1].strftime("%d/%m/%Y"))
384.                 # Expired product
385.                 else:

```

```

386.             info = info + '\n' + 'The {0} expired the day {1}. Throw out it!'\
387.                             .format(r[0], r[1].strftime("%d/%m/%Y"))
388.         else:
389.             info = 'There are no {} left at home, write it down on the shopping list.'\
390.                               .format(ingredients)
391.     except:
392.         info = 'Sorry, we are having technical problems, please try again.'
393.
394.     return info
395.
396.
397. def get_top_rated_recipe(self):
398.     return self.get_ingredients(self.get_recipe_id(''))
399.
400. def get_trending_recipe(self):
401.     return self.get_ingredients(self.get_recipe_id('', 't'))
402.
403.
404. def get_recipe_options_from_available_ingredients(self, n_options=2, n_ingredients=2):
405.     ingredients = []
406.     options = []
407.     ingredients = self.get_top_expired_ingredients_from_db(n_ingredients)
408.
409.     if len(ingredients) > 0:
410.         query = ', '.join(map(str, ingredients))
411.         recipes = self.search_recipes(query)
412.
413.     if recipes and 'recipes' in recipes:
414.         for recipe in recipes['recipes'][:n_options]:
415.             options.append(recipe['title'])
416.             self.option_dict[recipe['title']] = recipe['recipe_id']
417.     return options
418.
419.
420. def update_conversation_context(self):
421.     print('update_conversation_context')
422.     return self.msg_to_conversation('')
423.
424.
425. def msg_to_conversation(self, input_message):
426.     message = {}
427.     intent = ''
428.     entity = ''
429.     response_text = ''
430.
431.     if (input_message != ''):
432.         message['text'] = input_message
433.
434.     response = self.conversation.message(workspace_id=CONVERSATION_WORKSPACE,
435.                                         message_input=message,
436.                                         context=self.context,
437.                                         alternate_intents=False)
438.
439.     self.update_local_context(response['context'])
440.     self.intents = response['intents']
441.     self.entities = response['entities']
442.
443.     # Print intent and entity
444.     if (len(self.intents) > 0 and len(self.entities) > 0):
445.         print('#{0} ({@1}:{2})'.format(self.intents[0]['intent'], self.entities[0]['entity'],
446.                                         self.entities[0]['value']))
447.         intent = self.intents[0]['intent']
448.         entity = self.entities[0]['entity']
449.     elif (len(self.intents) > 0):
450.         print('#{0}'.format(self.intents[0]['intent']))
451.         intent = self.intents[0]['intent']

```

```

452.     elif (len(self.entities) > 0):
453.         print('@{0}:{1}'.format(self.entities[0]['entity'], self.entities[0]['value']))
454.         entity = self.entities[0]['entity']
455.
456.         if (response["output"] and response["output"]["text"]):
457.             for r in response["output"]["text"]:
458.                 response_text = response_text + '\n' + r
459.         return response_text, intent, entity
460.
461.
462.     def update_local_context(self, context):
463.         self.context = context
464.         for key, value in self.context.items():
465.             print('{0} = {1}'.format(key, value))
466.         print('\n')
467.
468.
469. ##### VISUAL RECOGNITION #####
470.     def get_food_from_image(self, image_path):
471.         with open(image_path, 'rb') as image_file:
472.             vr_response = self.visual_recognition.classify(images_file=image_file, classifier_ids=['food'])
473.             if vr_response['images'] and len(vr_response['images']) > 0:
474.                 image = vr_response['images'][0]
475.                 if image['classifiers'] and len(image['classifiers']) > 0:
476.                     classifier = image['classifiers'][0]
477.                     if classifier['classes'] and len(classifier['classes']) > 0:
478.                         food = classifier['classes'][0]['class']
479.                         score = classifier['classes'][0]['score']
480.                         return food, score
481.         return None, None
482.
483.
484.
485. ##### SLACK #####
486. """ The Slack Real Time Messaging API is an events firehose. This parsing function returns None unless a
487. message is directed at the Bot, based on its ID.
488.
489.     def parse_slack_output(self, slack_rtm_output, download_path='download/food.jpg'):
490.         output_list = slack_rtm_output
491.         if output_list and len(output_list) > 0:
492.             for output in output_list:
493.                 if output and 'text' in output and AT_BOT in output['text']:
494.                     # return text after the @ mention, whitespace removed
495.                     return output['text'].split(AT_BOT)[1].strip().lower(), output['channel']
496.                 elif output and 'file' in output and 'url_private_download' in output['file']:
497.                     down_url = output['file']['url_private_download']
498.                     extension = os.path.splitext(down_url)[1][1:].strip().lower()
499.                     if extension in ['jpg', 'png']:
500.                         self.download_file(down_url, download_path, 'download')
501.                         return 'photo', output['channel']
502.                     else:
503.                         return 'download_file_format_error', output['channel']
504.         return None, None
505.
506.
507. # Provide response
508.     def send_response(self, response):
509.         result = self.slack_client.api_call("chat.postMessage",
510.                                             channel=channel,
511.                                             text=response,
512.                                             as_user=True)
513.
514.     return result
515.
516.
517.

```

```

518.
519.     def download_file(self, url, local_filename, basedir):
520.         try:
521.             os.stat(basedir)
522.         except:
523.             os.mkdir(basedir)
524.         try:
525.             print('Saving to {}'.format(local_filename))
526.             headers = {'Authorization': 'Bearer ' + os.environ.get('SLACK_BOT_TOKEN')}
527.             r = requests.get(url, headers=headers)
528.             with open(local_filename, 'wb') as f:
529.                 for chunk in r.iter_content(chunk_size=1024):
530.                     if chunk:
531.                         f.write(chunk)
532.             except:
533.                 return False
534.         return True
535.
536.
537.
538.
539. ##### FOOD2FORK #####
540.     def food2fork_request(self, endpoint, **kwargs):
541.         data = {'key': FOOD2FORK_KEY}
542.         for key, value in kwargs.items():
543.             data[key] = value
544.         return endpoint + '?' + urllib.parse.urlencode(data)
545.
546.
547.     def search_recipes(self, query, sortBy='r'):
548.         endpoint = 'http://food2fork.com/api/search'
549.         url = self.food2fork_request(endpoint, q=query, sort=sortBy)
550.         print(url)
551.         return requests.get(url).json()
552.
553.
554.     def get_recipe_from_id(self, recipeId):
555.         endpoint = 'http://food2fork.com/api/get'
556.         try:
557.             url = self.food2fork_request(endpoint, rId=recipeId)
558.             print(url)
559.             return requests.get(url).json()
560.         except Exception as inst:
561.             print(inst)
562.             return None
563.
564.
565.     def get_recipe_id(self, query, sortBy='r'):
566.         recipes=self.search_recipes(query, sortBy)
567.         if recipes and 'recipes' in recipes and len(recipes['recipes'])>0:
568.             return(recipes['recipes'][0]['recipe_id'])
569.         else:
570.             return None
571.
572.
573.     def get_top_rated_recipe_options(self, n_options=1):
574.         options = []
575.         recipes = self.search_recipes('')
576.         if recipes and 'recipes' in recipes:
577.             for recipe in recipes['recipes'][:n_options]:
578.                 options.append(recipe['title'])
579.                 self.option_dict[recipe['title']] = recipe['recipe_id']
580.         return options
581.
582.
583.

```

```

584.     def get_trending_recipe_options(self, n_options=1):
585.         options = []
586.         recipes = self.search_recipes('', sortBy='t')
587.
588.         if recipes and 'recipes' in recipes:
589.             for recipe in recipes['recipes'][::n_options]:
590.                 options.append(recipe['title'])
591.                 self.option_dict[recipe['title']] = recipe['recipe_id']
592.
593.         return options
594.
595.
596.
597. ##### POSTGRES DATABASE #####
598. def database_connection(self, str_db_connection):
599.     print('Connecting to database ... ')
600.     # get a connection, if a connect cannot be made an exception will be raised here
601.     db_conn = psycopg2.connect(str_db_connection)
602.     # conn.cursor will return a cursor object, you can use this cursor to perform queries
603.     self.database_cursor = db_conn.cursor()
604.
605.
606.
607. def fetch_content(self, query):
608.     # execute Query
609.     self.database_cursor.execute(query)
610.
611.     # retrieve the records from the database
612.     records = self.database_cursor.fetchall()
613.     record_list=[]
614.     for r in records:
615.         record_list.append(r[0])
616.     return(record_list)
617.
618.
619.
620. # Obtain the n_ingredients products with the closest expiration date and that are in more quantity
621. def get_top_expired_ingredients_from_db(self, n_ingredients=2):
622.     ingredients = []
623.     db_query = 'SELECT name ' \
624.                 'FROM products ' \
625.                 'WHERE date(expiration_date) > current_date ' \
626.                 'ORDER by expiration_date ASC, quantity DESC ' \
627.                 'LIMIT {}'.format(n_ingredients)
628.
629.     ingredients = self.fetch_content(db_query)
630.
631.     return ingredients
632.
633.
634.
635. def get_db_information_about_ingredients(self, ingredients):
636.     query = "SELECT name, expiration_date, quantity " \
637.             "FROM products " \
638.             "WHERE name like '%{}%'".format(ingredients)
639.
640.     self.database_cursor.execute(query)
641.     records = self.database_cursor.fetchall()
642.
643.     return records
644.
645.
646.
647.
648.
649.
```

```
650.  
651. ##### MAIN #####  
652. if __name__ == "__main__":  
653.  
654.     smartfridge=SmartFridge()  
655.  
656.     if smartfridge.slack_client.rtm_connect():  
657.         print("smartfridge connected and running!")  
658.         while True:  
659.             command, channel = smartfridge.parse_slack_output(smartfridge.slack_client.rtm_read())  
660.             if command and channel:  
661.                 smartfridge.handle_command(command)  
662.                 time.sleep(READ_WEBSOCKET_DELAY)  
663.             else:  
664.                 print("Connection failed. Invalid Slack token or bot ID?")
```

REFERENCIAS

- [1] TechEmergence, «AI and consumer technology,» [En línea]. Available: <https://www.slideshare.net/TechEmergence/ai-founders-and-executives-predict-5year-trends-on-consumer-tech>.
- [2] Gartner, «The road to enterprise AI,» [En línea]. Available: https://www.gartner.com/imagesrv/media-products/pdf/rage_frameworks/rage-frameworks-1-34JHQ0K.pdf.
- [3] Minbowser, «Global Chatbot Trends Report – 2017,» [En línea]. Available: <http://mindbowser.com/chatbot-market-survey-2017/>.
- [4] Intento, «NLU / Intent Detection Benchmark by Intento, August 2017,» Agosto 2017. [En línea]. Available: <https://www.slideshare.net/KonstantinSavenkov/nlu-intent-detection-benchmark-by-intento-august-2017>.
- [5] S. team, «API Slack,» 2017. [En línea]. Available: <https://api.slack.com/>.
- [6] Z. Walchuk, «Easy Slack integration for Watson Conversation,» 2016. [En línea]. Available: <https://www.ibm.com/blogs/watson/2016/10/easy-slack-integration-watson-conversation/>.
- [7] M. Makai, «How to Build Your First Slack Bot with Python,» 2017. [En línea]. Available: <https://www.fullstackpython.com/blog/build-first-slack-bot-python.html>.
- [8] J. C. Fernandez, «Así funciona SiriKit,» [En línea]. Available: <https://applecoding.com/guias/asi-funciona-sirikit>.
- [9] 1. B. O. A. I. INSIGHTS, «COMPARING IBM WATSON TO ITS RIVALS,» [En línea]. Available: https://www.jyu.fi/it/tutkimus/muistiot/comparing_IBM_Watson.pdf.
- [10] M. McTear, Z. Callejas y D. Griol, *The Conversational interface. Talking to smart devices*, Springer, 2016.