

RASA FRAMEWORK: ANÁLISIS E IMPLEMENTACIÓN DE UN CHATBOT

Trabajo final de grado



Alumno: Álvaro Castillo Cabero
Director: Patricia Picanyol Mercadal
Ponente: Joan Antoni Pastor Collado
Especialidad de ingeniería del software
28 de enero de 2020

AGRADECIMIENTOS

En primer lugar agradecer a mi ponente, Joan Antoni Pastor Collado, su disponibilidad y consejos brindados durante todos estos meses ya que han sido una pieza esencial del desarrollo del proyecto.

Además, me gustaría agradecer a mis responsables, Lucas y Patricia, y a mis compañeros de everis, que durante la elaboración del proyecto han sido bombardeados con un sinfín de preguntas, de las que siempre he recibido una respuesta útil.

Y finalmente a mi familia y pareja, por el apoyo y paciencia que han tenido escuchando mis dudas e intentando ayudarme en todo lo posible.

RESUMEN

Hoy en día existe una gran variedad de herramientas para crear chatbots, todas ellas con sus ventajas e inconvenientes. Creemos que sería muy útil usar una de estas herramientas para facilitar la consulta en tiempo real de la información de eventos corporativos, ya que estos eventos ayudan a motivar a los empleados de la compañía e incluso a captar nuevos clientes.

Partiendo de estas premisas, hemos decidido desarrollar un asistente conversacional que funcione a través de dos canales de texto y voz: Telegram y Google Assistant. Para hacer este asistente, se ha usado un framework de código abierto llamado Rasa y también se ha analizado el mercado para ver el posicionamiento de dicho framework frente a la competencia. Se han generado modelos del bot basados en Machine Learning que son capaces de responder a preguntas formuladas a través de lenguaje natural en castellano. Además, el asistente se ha implementado de tal forma que la información del evento se pueda introducir fácilmente a través de un libro de Excel.

Las conclusiones del proyecto son que Rasa es un framework con mucho potencial, pero al ser un producto joven todavía le falta pulir algunos detalles. El asistente creado es capaz de entender las 20 intenciones de usuario que se han considerado para este proyecto y responder mediante 21 acciones de respuesta (10 de ellas transaccionales usando recursos externos).

RESUM

Avui dia hi ha una gran varietat d'eines per a crear chatbots, totes elles amb els seus avantatges i inconvenients. Creiem que seria molt útil fer servir una d'aquestes eines per facilitar la consulta a temps real de la informació d'actes corporatius, ja que aquests esdeveniments ajuden a motivar els empleats de la companyia i fins i tot a captar nous clients.

Partint d'aquestes premisses, hem decidit desenvolupar un assistent conversacional que funcioni a través de dos canals de text i veu: Telegram i Google Assistant. Per fer aquest assistent, s'ha fet servir un framework de codi obert anomenat Rasa i també s'ha analitzat el mercat per veure el posicionament d'aquest framework enfront de la competència. S'han generat models del bot basats en Machine Learning que són capaços de respondre a preguntes formulades a través de llenguatge natural en castellà. A més, l'assistent s'ha implementat de tal manera que la informació de l'acte es pugui introduir fàcilment a través d'un llibre d'Excel.

Les conclusions del projecte són que Rasa és un framework amb molt potencial, però com és un producte jove encara li falta polir alguns detalls. L'assistent creat és capaç d'entendre les 20 intencions d'usuari que s'han considerat per a aquest projecte i respondre mitjançant 21 accions de resposta (10 d'elles transaccionals usant recursos externs).

ABSTRACT

Nowadays there is a great variety of tools to create chatbots, all of them with their advantages and disadvantages. We believe it would be very useful to use one of these tools to facilitate real-time consultation of corporate event information, because these events help to motivate company employees and even attract new customers.

Based on these premises, we have decided to develop a conversational assistant that uses two text and voice channels: Telegram and Google Assistant. In order to make this assistant, an open source framework called Rasa has been used and the market has also been analyzed to see the positioning of that framework against competitors. We have generated some bot models based on Machine Learning that are capable of answering questions formulated through natural language in Spanish. In addition, the chatbot has been implemented in such a way that the event information can be easily entered through an Excel workbook.

The conclusions of the project are that Rasa is a framework with a lot of potential, but since it is a young product it still needs to improve some details. The created assistant is able to understand the 20 user intents that have been considered for this project and respond through 21 response actions (10 of them transactional using external resources).

ÍNDICE

1.	CONTEXTO	10
1.1	CONTEXTUALIZACIÓN	10
1.2	CONCEPTOS PROPIOS	10
1.2	PROBLEMA A RESOLVER.....	13
1.3	PARTES INTERESADAS	13
2.	JUSTIFICACIÓN	14
3.	ALCANCE	15
3.1	OBJETIVOS GENÉRICOS Y REQUISITOS FUNCIONALES.....	15
3.2	SUB-OBJETIVOS.....	15
3.3	REQUISITOS NO FUNCIONALES	16
3.4	POSIBLES OBSTÁCULOS Y RIESGOS	16
4.	METODOLOGÍA	17
4.1	SCRUM.....	17
4.2	HERRAMIENTAS DE SEGUIMIENTO.....	18
4.3	HERRAMIENTAS DE VALIDACIÓN.....	19
5.	DATOS GLOBALES DEL PROYECTO	19
6.	DESCRIPCIÓN DE LAS TAREAS	20
6.1	FASE DE GESTIÓN DEL PROYECTO – F1	20
6.2	FASE DE ANÁLISIS DE MERCADO – F2.....	21
6.3	FASE DE DESARROLLO DEL ASISTENTE – F3	21
6.4	CONCLUSIONES – F4	22
7.	GANTT Y ESTIMACIÓN	24
8.	GESTIÓN DEL RIESGO	26
9.	PRESUPUESTO	27
9.1	IDENTIFICACIÓN Y ESTIMACIÓN DE LOS COSTES	27
9.1.1	Recursos humanos.....	27
9.1.2	Recursos materiales.....	29
9.1.3	Recursos software	29
9.1.4	Presupuesto total	29
9.2	CONTROL DE GESTIÓN	30
10.	INFORME DE SOSTENIBILIDAD.....	31
10.1	AUTOEVALUACIÓN.....	31
10.2	DIMENSIÓN ECONÓMICA	32
10.3	DIMENSIÓN AMBIENTAL.....	32

10.4	DIMENSIÓN SOCIAL	33
11.	ANÁLISIS DE MERCADO	34
11.1	DIALOGFLOW	35
11.2	WATSON	36
11.3	LEX	37
11.4	RASA	38
12.	FRAMEWORK Y LIBRERÍAS UTILIZADAS	46
12.1	DIAGRAMA DE PAQUETES USADOS	46
12.2	BOT DE TELEGRAM	47
12.3	SPACY	49
12.4	PANDAS	50
12.5	SMTPLIB	51
12.6	PDOC	53
12.7	OTRAS LIBRERÍAS	54
13.	ELEMENTOS DEL BOT	55
13.1	DIAGRAMA DE ARQUITECTURA	55
13.2	DIAGRAMA DE CLASES	56
13.3	BASE DE DATOS	57
13.4	DATOS ENTRENAMIENTO DEL NLU	59
13.4.1	INTENTS, ENTITIES Y SLOTS	59
13.4.2	SINÓNIMOS Y TABLAS DE BÚSQUEDA	60
13.4.3	EXPRESIONES REGULARES	61
13.5	HISTORIAS ENTRENAMIENTO DEL NLU	61
13.6	ACTIONS	62
13.7	DOMINIO DEL ASISTENTE	63
13.8	CONECTORES	64
14.	ANÁLISIS FUNCIONAL DEL BOT	65
14.1	PROCESADO DE UN MENSAJE	65
14.1.1	DIAGRAMA DE SECUENCIA DEL ASISTENTE	66
14.2	DIAGRAMA DE CASOS DE USO	69
14.3	DIAGRAMA DE ESTADOS	71
14.4	CONEXIÓN CON GOOGLE ASSISTANT	72
14.4.1	CONFIGURACIÓN DE LAS ACTIONS DE GOOGLE ASSISTANT	72
14.4.2	CREACIÓN DEL CONECTOR DE RASA	73

15.	PRUEBAS	75
15.1	PRUEBAS DE ESTRÉS	75
15.2	TESTS UNITARIOS	77
16.	INSTALACIÓN Y DESPLIEGUE DEL ASISTENTE	79
16.1	INSTALACIÓN DE RASA	79
16.2	DESPLIEGUE DEL ASISTENTE	81
17.	INTEGRACIÓN DE CONOCIMIENTOS	84
18.	IDENTIFICACIÓN DE LEYES Y REGULACIONES	85
19.	CONCLUSIONES Y RECOMENDACIONES DE MEJORA.....	86
19.1	CONCLUSIONES DE LA TECNOLOGÍA.....	86
19.2	CONCLUSIONES PERSONALES	86
19.3	POSIBLES MEJORAS	87
	REFERENCIAS	88

ÍNDICE DE FIGURAS

Figura 1. Concepto de chatbot (Fuente: chatbotslife.com).	11
Figura 2. Ejemplo de funcionamiento rasa (Fuente: medium.com).	13
Figura 3. Esquema conceptual Scrum (Fuente: openwebinars.net).	18
Figura 4. Diagrama de Gantt (Fuente: tomsplanner.es)	25
Figura 5. Logo dialogflow (Fuente: dialogflow.com)	35
Figura 6. Interfaz de usuario de Dialogflow (Fuente: dialogflow.com)	35
Figura 7. Logo Watson Assistant (Fuente: cloud.ibm.com)	36
Figura 8. Interfaz de usuario de Watson Assistant (Fuente: developer.ibm.com)	36
Figura 9. Logo Amazon Lex (Fuente: aws.amazon.com/lex)	37
Figura 10. Interfaz de usuario de Amazon Lex (Fuente: developer.here.com)	37
Figura 11. Logo Rasa (Fuente: rasa.com)	38
Figura 12. Interfaz de usuario de Rasa.	38
Figura 13. Diagrama de paquetes.	46
Figura 14. Logo Botfather (Fuente: core.telegram.org/bots)	47
Figura 15. Interacción con BotFather.	48
Figura 16. Ejemplo de tokenización de spaCy (Fuente: spacy.io)	49
Figura 17. Comandos instalación spaCy (Fuente: spacy.io)	49
Figura 18. Ejemplo de uso Pandas.	50
Figura 19. Permitir aplicaciones poco seguras	51
Figura 20. Código envío email con SMTP.	52
Figura 21. Función adjuntar imagen al correo.	52
Figura 22. Ejemplo documentación clase con pdoc.	53
Figura 23. Diagrama de Arquitectura.	55
Figura 24. Diagrama de clases.	56
Figura 25. Diferencias DBMS y Excel. (Fuente: exceltotal.com)	57
Figura 26. Ejemplo intents y entities RASA.	59
Figura 27. Ejemplo de uso de sinónimos.	60
Figura 28. Ejemplo tabla de búsqueda.	60
Figura 29. Ejemplo expresiones regulares.	61
Figura 30. Historias de entrenamiento.	61
Figura 31. Creación de un action.	62
Figura 32. Credenciales conector Telegram	64
Figura 33. Proceso de clasificación de un mensaje.	65
Figura 34. Diagrama secuencia Usuario y NLU	66
Figura 35. Diagrama de secuencia CORE.	67
Figura 36. Diagrama de secuencia creación de respuesta.	68
Figura 37. Diagrama de casos de uso del usuario.	69
Figura 38. Diagrama de casos de uso del administrador.	70
Figura 39. Diagrama de estados.	71
Figura 40. Contenido fichero actions.json.	72
Figura 41. Código conector Google Assistant.	74
Figura 42. Configuración del test de estrés.	75
Figura 43. Respuesta llamada test estrés.	75
Figura 44. Tiempo llamadas test estrés.	76
Figura 45. Usuarios concurrentes test estrés.	76
Figura 46. Ejecución tests unitarios.	78
Figura 47. Código tests unitarios.	78
Figura 48. Entrenamiento del modelo	81
Figura 49. Despliegue ngrok.	82

Figura 50. Despliegue servidor actions.82
Figura 51. Ejecutar asistente en terminal.83
Figura 52. Despliegue servidor modelo Rasa.83

ÍNDICE DE TABLAS

Tabla 1. Resumen de tareas.24
Tabla 2. Previsión horas por tarea.24
Tabla 3. Promedio salarial según rol (Fuente: indeed.com).....28
Tabla 4. Coste recursos humanos. (Fuentes: indeed.com y fib.com)28
Tabla 5. Coste recursos materiales. (Fuente: everis.com)29
Tabla 6. Presupuesto total del proyecto.30
Tabla 7. Comparativa Mercado39
Tabla 8. Tabla de puntuaciones comparativa tecnologías40
Tabla 9. Detalles hoja "info_evento".58
Tabla 10. Detalles hoja "charlas".58
Tabla 11. Tests unitarios realizados.77

1. CONTEXTO

1.1 CONTEXTUALIZACIÓN

El siguiente proyecto se trata de un **Trabajo Final de Grado** de la especialidad de Ingeniería de Software impartida en la Facultad de Informática de Barcelona, perteneciente a la Universidad Politécnica de Cataluña. Se trata de un TFG de modalidad B, ya que se desarrolla con un Convenio de Cooperación Educativa en la consultora internacional Everis.

«**Bots are the new apps**» (Della Cava, 2016). Hoy en día el uso de asistentes virtuales (chatbots) está en auge y cada vez son más las empresas que destinan estos softwares a tareas como por ejemplo, la atención al cliente. El secreto se halla en que estos asistentes aceleran las interacciones entre personas y servicios, mejorando la experiencia del cliente, ya que los clientes desconocen que están hablando con un programa informático. Al mismo tiempo, ofrecen a las empresas nuevas oportunidades para mejorar el proceso de compromiso de los clientes y además ayuda a mejorar la eficiencia operativa al reducir el costo típico del servicio al cliente.

La empresa con la que tengo el convenio activo, me invitó a descubrir un framework¹ de creación de asistentes virtuales llamado **Rasa** y documentar sus diferencias respecto otras opciones del mercado. En eso se centra una gran parte de mi proyecto, descubrir que ventajas aporta dicho framework sobre sus competidores y mostrar el resultado con una aplicación práctica de creación de un asistente usando esa tecnología, en este caso, destinado a la gestión de eventos corporativos.

A través de este proyecto, además, pretendo entender cómo funcionan los asistentes conversacionales, profundizar en ellos para ver cómo están creados y descubrir principales problemas con los que los ingenieros de software se encuentran a la hora de crear dicho software.

1.2 CONCEPTOS PROPIOS

Bot

Los bots son softwares creados haciendo uso de la inteligencia artificial y sirven para automatizar procesos que se ejecutan sin la necesidad de una intervención humana. Algunos ejemplos que encontramos de bots podrían servir para mostrar información del tiempo a los usuarios de una web o app o reservar habitación en un hotel o restaurante.

En este punto es importante explicar el test de Turing (Turing, 1950). En este test, el científico Alan Turing quería demostrar que no solo se podían mantener conversaciones máquina-humano, sino también comprobar si un ordenador puede convencer a un humano de que la conversación se está manteniendo con otro humano cuando en realidad se está hablando con un ordenador.

¹ Entorno de trabajo, conjunto de conceptos y prácticas para enfocar un problema.

Chatbot

Los **chatbots** – también llamados asistentes virtuales – son bots especializados y creados para mantener conversaciones y ofrecer respuestas preconcebidas. Por lo tanto, un chatbot es un software que utiliza mensajes estructurados para emitir respuestas desde una máquina hacia un interlocutor humano, véase figura 1.

Gracias a la inteligencia artificial, los chatbots son capaces de analizar el sentimiento de dicha conversación, lo que hace que el chatbot no solo se use para automatizar respuestas y contestaciones preestablecidas, sino para generar valiosos informes de reputación, análisis del sentimiento y engagement² con las marcas a través de las conversaciones que los clientes mantienen con ellas.

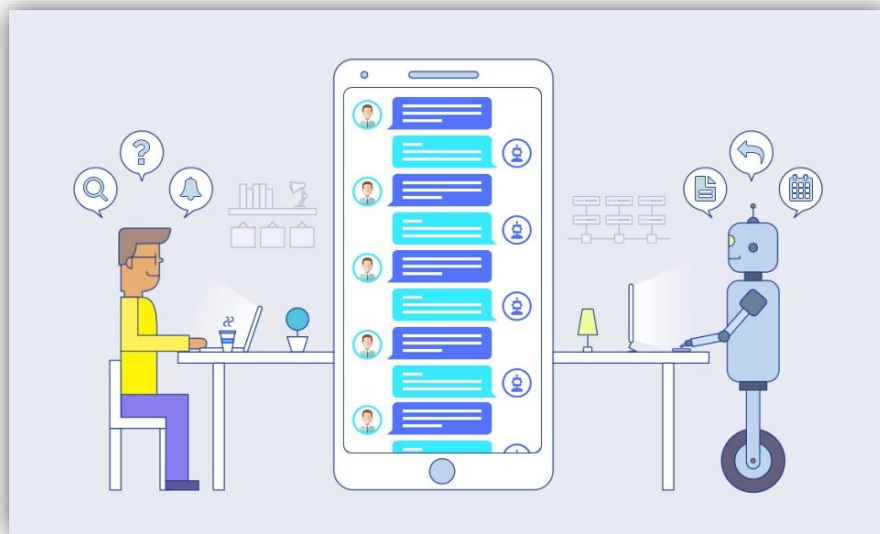


Figura 1. Concepto de chatbot (Fuente: chatbotslife.com).

NLP

Las siglas NLP hacen referencia a Natural Language Processing, es decir, el procesamiento del lenguaje natural. Se trata de una rama de la inteligencia artificial que ayuda a los ordenadores a entender, interpretar y manipular el lenguaje humano.

NLU

Estas siglas hacen referencia a Natural Language Understanding, se trata de un subconjunto de procesos NLP que tratan específicamente en convertir los datos de entrada en algo que la máquina pueda entender, para poder actuar en consecuencia.

Tal y como se comenta en (Bates, 1995), para los seres humanos no requiere mucho esfuerzo entender el lenguaje, pero para las máquinas es especialmente complicado entender insultos, juegos de palabras, faltas de ortografía, dobles sentidos, etc.

² Capacidad de una marca para crear relaciones sólidas y duraderas con sus usuarios.

NLG

Corresponde a la Generación de Lenguaje Natural (Natural Language Generation) y se trata del proceso contrario al NLU, es decir, cuando una máquina escribe o habla en nuestro lenguaje. En (Stent & Bangalore, 2001) se señala la importancia de enseñar a saltarse puntualmente las normas del lenguaje, para humanizar a la máquina.

Intent

Los intent o intentos son un conjunto de expresiones dichas por el usuario que significan una acción concreta. Por ejemplo, el intent “saludar” tendrá las siguientes expresiones: “Hola”, “Buenos días”, “Hey”, etc.

Para los ingenieros de software, a la hora de crear asistentes virtuales, este punto es el más difícil ya que nuestra lengua es sumamente compleja y en una conversación intervienen elementos difíciles de entender por una máquina, como por ejemplo el contexto y los modos de expresión o figuras retóricas – como por ejemplo la ironía –. En (Zapfen, 2018), se dice «**La lengua es una ciencia viva, varía en el tiempo y en el espacio**». Para mantener este control sobre la lengua, y poder humanizar a los robots, podemos contar con perfiles de lingüística en las grandes empresas. Así que para este proyecto, todos los intent serán revisados por un lingüista del departamento de Digital Experience de everis.

Entity

Una entity o entidad se trata de una información que es extraída de lo que nos dice un usuario. Por ejemplo, si nuestro chatbot le pregunta al usuario cómo se llama, se espera una respuesta con formato parecido a “Me llamo {nombre}”, donde nombre es la entidad.

Action

Una action o acción es la tarea que se espera que el bot realice. En la gran mayoría de los casos se tendrá que consultar una API³ externa. Una acción podría ser consultar el tiempo en un lugar concreto a partir de unos parámetros introducidos por el usuario.

Canal

El canal es la plataforma mediante la cual los usuarios podrán acceder al chatbot. Hay canales de texto y otros que integran voz y texto.

³ Conjunto de funciones y procedimientos que cumplen una o muchas funciones con el fin de ser utilizadas por otro software.

Ejemplos de canales pueden ser: telegram, whatsapp, twitter, slack, Messenger, Facebook, Google Assitant, Alexa.

En la Figura 2 que se muestra a continuación, podemos ver un diagrama de funcionamiento del framework, donde podemos apreciar como intervienen los conceptos listados anteriormente. Los módulos NLP y NLG se encuentran dentro de Rasa Stack.

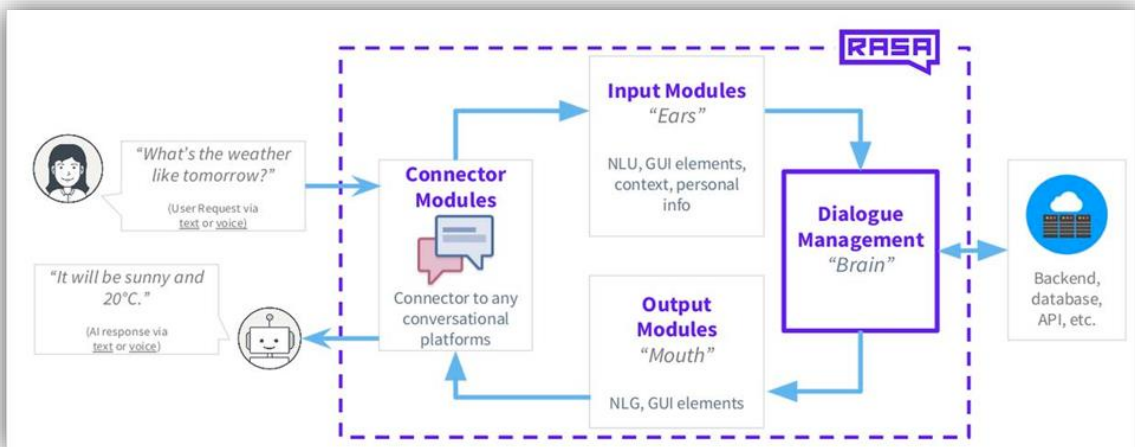


Figura 2. Ejemplo de funcionamiento rasa (Fuente: medium.com).

1.2 PROBLEMA A RESOLVER

El problema que se pretende resolver con este proyecto es el estudio de un nuevo framework para determinar en qué tipo de proyectos aportaría mejoras frente a las otras opciones existentes en el mercado.

Por otra parte, con el asistente resultante de aplicar la tecnología, se pretende ofrecer una solución para la gestión de eventos corporativos en la empresa donde se desarrolla el proyecto.

1.3 PARTES INTERESADAS

A continuación se describirán los actores implicados en el proyecto (a quien va dirigido el producto, quien lo usará y quien se beneficiará de sus resultados).

Everis

Una de las principales partes interesadas de este proyecto es la empresa. En concreto se quedará toda la documentación el departamento de Digital Experience⁴. Gracias a este proyecto se puede conseguir un empleado experto en este framework.

La empresa podrá usar el asistente de gestión de eventos para organizar sus propios eventos corporativos, y que pueda ser usado por sus empleados. Además, esta

⁴ Departamento encargado de la experiencia digital (Mobile, Digital Channels, Front, UX).

tecnología podría ser ofrecida por la compañía como un producto más a los clientes y obtener beneficios económicos.

Desarrollador

Al desarrollador del proyecto, le servirá para recibir un conocimiento profundo sobre el tema y poder convertirse en un experto en este framework dentro de la empresa. También le servirá para realizarlo como Trabajo Final de Grado y poder obtener el título universitario.

Clientes de la empresa

Otra de las partes interesadas son los clientes de Everis, ya que en los proyectos en los que Rasa Framework sea más óptimo que los demás, se podrá aplicar como solución a las empresas y mejorará la calidad del proyecto.

El ponente del proyecto

El ponente del proyecto, Joan Antoni Pastor Collado, que guiará al estudiante con el proyecto para realizarlo con éxito.

2. JUSTIFICACIÓN

Para desarrollar este proyecto se tendrá que diseñar una solución nueva en cuanto a tecnología, aprovechando la información proporcionada de otro proyecto interno, se podrán reutilizar funcionalidades dentro de este chatbot y así sacarle el máximo partido posible.

Se tendrá que diseñar la parte de tecnología, puesto que no hay proyectos anteriores en la empresa que usen esta tecnología llamada Rasa de código abierto⁵, y las diferencias con los otros frameworks son tan notables que se considera de un trabajo mucho mayor adaptarlas que crearlas desde cero.

La parte aprovechable es el conjunto de intents, entities, actions de un proyecto ya desarrollado con Dialogflow⁶. Este proyecto se presentó funcional en la dNextCon⁷, pero por motivos de poca previsión sobre la cantidad de peticiones que podíamos recibir, no estuvo funcional el 100% del tiempo.

⁵ Software que no utiliza encriptación y puede ser modificado por la comunidad de programadores.

⁶ Herramienta de creación de asistentes virtuales desarrollada por Google.

⁷ Evento corporativo de everis.

3. ALCANCE

Una vez hemos contextualizado nuestro proyecto y hemos justificado si vamos a usar una solución existente o adaptarla, vamos a definir el alcance de nuestro proyecto. Para poder hacerlo de la forma más fiel posible a la realidad, vamos a definir objetivos del proyecto – genéricos y sub-objetivos – y también identificar otros requisitos funcionales y no funcionales. Por último, vamos a intentar anticiparnos a los posibles obstáculos y riesgos que pueden aparecer durante el desarrollo del mismo.

3.1 OBJETIVOS GENÉRICOS Y REQUISITOS FUNCIONALES

El principal objetivo del proyecto es analizar un framework concreto, en este caso Rasa, comparando sus funcionalidades con los frameworks actuales y más conocidos como por ejemplo, Dialogflow, IBM Watson, Lex, etc.

También se pretende crear un chatbot usando esta tecnología y que use un documento en formato Excel como base de datos, de esta forma se puede modificar la información de una forma muy sencilla. Se quiere que el chatbot sea capaz de responder preguntas sobre un evento de empresa de forma rápida y fácil.

Las principales funcionalidades del asistente son:

- Consultar información básica del evento, eso quiere decir, saber el lugar del evento, la fecha y hora y cualquier modificación debido a incidencia que pueda haber en la organización del evento.
- Notificar la ubicación del evento con la ayuda de Google Maps⁸.
- Consultar las actividades que se van a desarrollar en cada sala durante todo el evento, también filtrar las charlas según diversos criterios (Ponente, título, fecha, etc.).
- Consultar las condiciones meteorológicas en tiempo real del evento.
- Poder hacer uso del asistente a través de Google Assistant y Telegram⁹.

3.2 SUB-OBJETIVOS

Además de los objetivos genéricos, nos hemos planteado algunos sub-objetivos, que consideramos complementarios sobre el desarrollo del asistente virtual.

Uno de ellos consiste en integrar el chatbot en un canal complementario que incluya texto y voz, como por ejemplo a través del asistente de Google. De esta forma, el usuario podrá controlar el asistente sin necesidad de ir escribiendo por chat, sino que directamente podrá hacer las preguntas y el asistente le contestará o le mostrará la información por la pantalla.

⁸ Aplicación de mapas de Google.

⁹ Plataforma de mensajería instantánea.

3.3 REQUISITOS NO FUNCIONALES

Como requisito adicional, queremos que el producto cumpla con los estándares de eficiencia, seguridad y usabilidad de la empresa, así como respetar el manual de identidad visual de la compañía.

El asistente tendrá una disponibilidad del 99,99% de las veces en que un usuario intente accederlo, es decir, se intentará que siempre pueda estar disponible para ser accedido por el usuario. También queremos que el asistente soporte un gran número de usuarios de forma simultánea, así que someteremos nuestro chatbot a tests de estrés para ver que no deja de estar operativo al recibir muchas conexiones de usuarios.

3.4 POSIBLES OBSTÁCULOS Y RIESGOS

«More than 70% of the IT projects suffer total failure, cost overruns, schedule overruns, or deliver fewer functions than promised» (Wallace & Keil, 2004).

Es una realidad que la gran mayoría de proyectos en el sector IT¹⁰ sufren problemas y tienen que recortar en funcionalidades, ampliar tiempo o cancelar el proyecto. Eso implica un aumento de coste y produce mala imagen de cara al cliente, por lo tanto, vamos a estructurar los posibles problemas que nos podemos encontrar durante el desarrollo.

Tiempo

Como el Trabajo Fin de Grado tiene una fecha límite fijada, nos encontramos con que cualquier tipo de problema, bug, espera en la obtención de licencias o estancamiento en la realización del proyecto puede dificultar poder llegar a tiempo para la entrega.

También tenemos que tener en cuenta que al utilizar técnicas de Machine Learning, se tendrá que generar un modelo y eso conlleva un tiempo adicional.

Framework

El éxito del proyecto dependerá en gran parte por un manejo fluido de la tecnología con la que vamos a trabajar. Se partirá de cero en el conocimiento de la tecnología pero se dispone de bastante información en la web oficial del framework y dispone de una gran comunidad detrás, con foros de ayuda y tutoriales. El riesgo puede aparecer si esa información no satisface alguna de las dudas, o se trabaja sobre conocimientos más avanzados sobre asistentes virtuales.

¹⁰ Tecnologías de la Información.

Presupuesto

Debido a que el presupuesto del que dispone el proyecto es nulo, es de vital importancia que los servicios y tecnologías utilizadas sean gratuitos o dispongan de pruebas gratuitas. Si alguna de las plataformas que se usan en el proyecto es de pago, se tendrá que adaptar algunas partes para incluir otras opciones gratuitas, sin llegar a perder mucho tiempo para evitar retrasar el proyecto.

4. METODOLOGÍA

Ya que se trata de un proyecto de casi cinco meses de duración y con mucho trabajo detrás, considero de vital importancia definir las bases que sirvan como guía para poder trabajar para poder acabar el proyecto sin prisas ni problemas de tiempo.

Tras estudiar diversas opciones de metodologías existentes, se ha decidido seguir una metodología ágil durante del desarrollo del proyecto. Esta metodología, definida en (Beck et al., 2001) por un grupo de ingenieros de software, nació de la crítica a las metodologías tradicionales, muy estructuradas y estrictas. El principal problema es que al desarrollar de una forma tan burocrática, lenta y con dificultad a cambios, podía afectar negativamente al resultado del proyecto, ya que no se le mostraba al cliente el resultado hasta que estaba muy avanzado. Al trabajar con una metodología basada en un desarrollo iterativo, nos proporciona flexibilidad para poder ir añadiendo cambios y también nos permitirá crear pequeños prototipos a medida que se añadan funcionalidades al chatbot. También nos ayuda a poder aplicar cambios a los requisitos y especificaciones que, debido a mi falta de experiencia, se hayan definido de forma errónea o diferente a la realidad.

Al principio de todo, se definirán los sprints¹¹ con una duración de una semana, de esta forma podremos realizar una reunión semanal con nuestro director del proyecto en la empresa. En estas reuniones se revisará el progreso de esa semana, obtendré feedback¹² sobre el estado del proyecto y también podremos definir las tareas para la siguiente iteración.

A continuación entraremos en detalle sobre la metodología y las herramientas de seguimiento empleadas para validar que se consiguen los objetivos.

4.1 SCRUM

Tras decidir que se trabajará con metodología ágil, debo decidir con que marco de trabajo implementar dicho desarrollo. Para este proyecto he considerado que la mejor opción es Scrum, ya que es la que más hemos usado durante la carrera y que considero que es la que mejor se adapta al proyecto que vamos a realizar.

¹¹ Intervalo prefijado durante el cual se crea un incremento de producto.

¹² Reacción, respuesta u opinión que nos da un interlocutor como retorno sobre un asunto determinado.

Esta metodología que significa “melé”, fue definida en (Takeuchi & Nonaka, 1986), la principal ventaja de esta metodología es que permite obtener resultados de forma rápida y así poder ir cambiando cosas del proyecto en función del feedback obtenido. En (Schwaber & Sutherland, 2010), se detalló una guía para explicar Scrum de manera clara y simple.

Las ideas principales de esta metodología las podemos ver en la Figura 3. El proyecto se divide en sprints, que tienen una duración de 1 a 4 semanas (en nuestro caso una semana) y en cada uno de estos sprints vamos a ir obteniendo un producto funcional. En el proyecto también se definirá un product backlog, que es una “lista” de todos los requisitos del proyecto ordenados por prioridad.

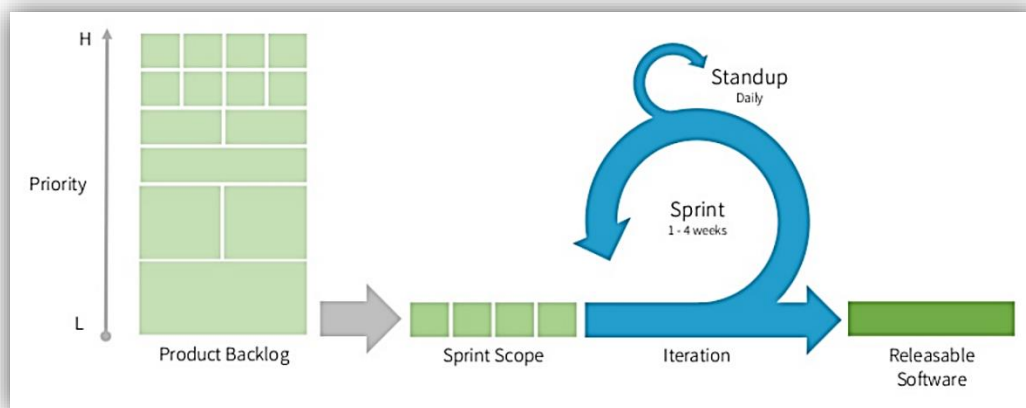


Figura 3. Esquema conceptual Scrum (Fuente: openwebinars.net).

4.2 HERRAMIENTAS DE SEGUIMIENTO

Es imprescindible definir una buena metodología de seguimiento y aunque se podría hacer un seguimiento del proyecto de forma manual, es aconsejable usar herramientas adecuadas que faciliten dicha gestión.

Para un correcto seguimiento de las tareas a realizar y los diferentes sprints, he decidido usar la herramienta Taiga. Esta herramienta de open source, ofrece tableros diferentes con funcionalidades específicas. El tablero Scrum de Taiga incluye un Burn Down Chart¹³ y otros datos de seguimiento, como los puntos de historia completados y los restantes para finalizar el sprint, además del tablero con los diferentes estados por los que pasan las tareas y las historias de usuario¹⁴. A través de la herramienta podremos ver si vamos cumpliendo los plazos que nos hemos marcado para cada tarea, y en función de ello, adaptar las funcionalidades. También nos será muy útil para dejar constancia de las incidencias que hemos sufrido y como nos han afectado en el desarrollo del proyecto. Para el control de versiones¹⁵ se usará Git¹⁶. En concreto trabajaremos con Github, herramienta que nos permite tener nuestros repositorios de Git en la nube. De esta

¹³ Gráfico que muestra el trabajo completado y restante.

¹⁴ Pequeñas descripciones de los requisitos de un cliente.

¹⁵ Sistema que permite registrar los cambios del código a medida que se modifica.

¹⁶ Software de control de versiones.

manera, podremos volver a una versión anterior que esté funcionando correctamente si encontramos un error de peso. Otra de las ventajas de Git es la facilidad que aporta en el trabajo colaborativo, pero en este proyecto, solo habrá un desarrollador y por lo tanto no se necesitará.

4.3 HERRAMIENTAS DE VALIDACIÓN

Validaremos nuestro proyecto a través de las reuniones con el director del proyecto y los demás compañeros de equipo. De esta forma podremos ir comprobando que estamos cumpliendo los requisitos definidos y que se ha hecho de la forma correcta. Trabajaremos haciendo pruebas simuladas e incluso se probará de aplicar en un evento real de la empresa, para ver si está disponible al ser usado por un número grande de usuarios.

Al trabajar cerca de compañeros del mismo departamento, especializados en chatbots, será más fácil poder resolver las dudas y los posibles problemas que puedan ir apareciendo a lo largo del proyecto.

Para validar los aspectos académico-administrativos se hará a través de la figura del ponente y el feedback del tutor de GEP en las demás entregas, ya que son quienes me podrán ayudar con ese aspecto en el marco de la universidad.

5. DATOS GLOBALES DEL PROYECTO

El proyecto se ha iniciado el día 16 de septiembre, coincidiendo con mi inicio de convenio en la empresa everis, las fechas de defensa¹⁷ puede variar en el rango del 23 a 29 de enero, ambos incluidos. Así que se ha tomado como referencia el último día, que coincide con la fecha de fin de convenio. Por lo tanto, se dispone de **un máximo de 92 días de trabajo**, destinados exclusivamente al proyecto, que equivalen a 735 horas de trabajo, aunque, si acabamos antes el proyecto, nos dedicaremos a revisarlo o trabajar sobre otros temas. No todas las horas del proyecto serán destinadas a realizar la memoria, pues según la normativa del TFG descrita en (Comissió Permanent, 2017) estipula que la memoria se tiene que entregar una semana antes de la fecha de defensa, así que contamos con un máximo de 87 días para la realización de esta. Tras llegar a la fecha de entrega, se dispone de una reserva por si hemos tenido que atrasar la preparación de la defensa debido a algún problema.

En los siguientes apartados del documento, se describirá en detalle la previsión de las tareas a realizar en estas fechas y las posibles desviaciones que puedan existir.

¹⁷ Exposición del trabajo por parte del alumno ante un tribunal académico.

6. DESCRIPCIÓN DE LAS TAREAS

En esta sección se muestran las diferentes fases que componen nuestro proyecto así como entrar en detalle sobre las diferentes tareas que componen las fases y sus dependencias. En cada fase se incluyen los recursos – humanos y materiales – que serán necesarios y al final de la sección, se incluye una la Tabla 1 que sirve como resumen de las tareas del proyecto.

Nuestro proyecto está compuesto por cuatro fases: fase de gestión, fase de análisis de mercado, fase de desarrollo del asistente y conclusiones. La Fase 2 y la Fase 3 pueden desarrollarse en paralelo, mientras que la Fase 4 tiene que esperar a que la F2 y la F3 acaben.

6.1 FASE DE GESTIÓN DEL PROYECTO – F1

En esta primera fase nos vamos a centrar en buscar documentación sobre la tecnología con la que vamos a trabajar, definir el contexto y alcance de nuestro proyecto, establecer una planificación temporal y detallar el presupuesto y sostenibilidad del proyecto. Esta primera fase está compuesta por las siguientes tareas:

- **T1 – Estudio de la documentación de Rasa Framework y posibilidades de la tecnología**, para tener una primera idea de qué nos ofrece Rasa y si es posible nuestro proyecto.
- **T2 – Definir contexto, justificación, alcance y metodología a seguir de nuestro proyecto**, ya que nos va a permitir organizar mejor el proyecto y contextualizar al lector de la memoria. ($T1 < T2$).
- **T3 – Definir las tareas a realizar y su estimación de tiempo**, para tener un calendario con las actividades a realizar y ver si podremos acabar el proyecto en la fecha límite. ($T2 < T3$).
- **T4 – Definir la gestión del riesgo**, para saber qué hacer si encontramos obstáculos en nuestro proyecto y tenemos que trazar planes alternativos. ($T3 < T4$).
- **T5 – Identificar y estimar costes, definir mecanismos para el control de gestión presupuestaria y realizar informe de sostenibilidad**, para obtener una idea del presupuesto que se debería destinar a un proyecto de estas características, y también, se realizará un informe de sostenibilidad sobre el proyecto. ($T3 < T5$).
- **T6 – Reuniones con ponente o con director TFG**, para ir comentando cada dos semanas el estado del proyecto y revisar si sigue la planificación de sprints prevista.

Recursos F1

- Humanos:
 - Profesor gestión de proyectos facultad.

- Software:
 - Tomsplanner¹⁸ para realizar los diagramas de Gantt.

6.2 FASE DE ANÁLISIS DE MERCADO – F2

En esta segunda fase, nos encontramos que ya nos hemos hecho una idea de la tecnología que tenemos entre manos y lo que podemos lograr con ella, así que vamos a compararlo con las otras opciones que encontramos en el mercado para ver en qué aspectos es mejor solución usar este framework y en cuáles no.

Esta fase consta de las siguientes tareas – coinciden con las tareas principales de un analista de mercados –:

- **T7 – Definir objetivos de investigación de mercado**, es decir, debemos tener claro que aspectos vamos a observar y comparar con las demás tecnologías. (T1<T7).
- **T8 – Fijar estrategias de selección e interpretación de los datos de las tecnologías ajenas y propia**, ya que antes de empezar a buscar los datos tenemos que plantearnos cómo los vamos a obtener (mirando documentación, instalando el software, etc.). Para cada tecnología puede ser de forma diferente. (T7 < T8).
- **T9 – Obtener datos y analizarlos**, esta fase es la más importante, ya que debemos obtener datos de calidad y analizar para ver si es cierto lo que hemos encontrado. (T8 < T9).
- **T10 – Organizar los resultados y estructurarlos**, ya que hoy en día es de vital importancia mostrar los datos de forma clara y entendedora, la intención es que gente no especializada en el tema, pueda entender los resultados. (T9 < T10).

Recursos F2

Esta fase no precisa de recursos específicos de software más allá del navegador web y alguna de las tecnologías que se quieren comparar, pero no se creará ningún asistente.

6.3 FASE DE DESARROLLO DEL ASISTENTE – F3

Entramos en la fase de desarrollo, esta es la que tendrá más peso en el proyecto ya que se trabaja con una tecnología nueva que no se tiene experiencia.

- **T11 – Validación del alcance definido de nuestro proyecto**. En este punto estudiaremos si vamos siguiendo la planificación correctamente o tenemos que recortar funcionalidades para poder llegar a la fecha de entrega. (F1 & F2 < T11).

¹⁸ Software para creación de diagramas de Gantt.

- **T12 – Instalación de Rasa Framework y realización de tutoriales**, serán básicos para aprender cómo funciona, poder empezar a tener una idea de la tecnología y preparar todo el entorno para realizar el proyecto.
- **T13 – Implementación e integración del chatbot**, consiste en realizar el asistente usando Rasa y tenerlo acabado y funcional antes de la fecha de entrega del mismo. Esta tarea es la más crítica, pues cualquier error o problema no detectado a tiempo nos puede retrasar e incluso evitar que se pueda acabar el proyecto. En esta tarea también se incluye la integración con los canales de comunicación, que idealmente serán dos: uno de texto con telegram y otro de voz con Google Assistant. (T11 & T12 < T13).
- **T14 – Pruebas del correcto funcionamiento del software**, que se harán tanto al instalar el software como al implementar el chatbot e integrarlo. Se irán haciendo cada vez que se añada nuevas funcionalidades y es muy importante que estas pruebas sean lo más exhaustivas posibles, ya que así evitaremos que se nos pueda colar algún error sin detectar que provoque fallos en nuestro asistente. (Paralelo a T12 & T13)

Recursos F3

- Materiales:
 - Móvil Xiaomi mi9 para testing del asistente en Telegram con procesador Qualcomm Snapdragon 855, 64 GB de memoria interna, 6GB de RAM y pantalla de 6,4”.
 - Google Assistant con pantalla integrada para testing del asistente en canal de voz.
- Software:
 - Anaconda¹⁹ para alojar el entorno de desarrollo.
 - Rasa Framework instalado en Anaconda.
- Humanos:
 - Lingüistas del equipo para recibir ayuda con la definición de los intent.
 - Experto en chatbots del equipo para consultar dudas sobre la implementación.

6.4 CONCLUSIONES – F4

Esta es la última fase del proyecto, en ella se analiza los resultados obtenidos anteriormente y se genera la documentación referente al proyecto. Esta documentación se irá elaborando en paralelo a las fases anteriores, para evitar tener que hacerla toda al final del proyecto. Al acabar la documentación, se empezará a preparar la defensa del proyecto.

¹⁹ Distribución de Python que funciona como un gestor de entorno.

- **T15 – Crear documentación del proyecto**, este punto pese a no ser crítico en cuanto a planificación, es muy importante para un correcto desarrollo del proyecto. Gracias a esta documentación se podrá mantener el software de forma óptima y entender por qué se han tomado ciertas decisiones.
En (Reilly, 2016), podemos encontrar los puntos más importantes del estándar de documentación de software desarrollado por el IEEE²⁰, así que intentaremos cernernos a este estándar para poder realizarla con la mayor calidad posible. (Paralela a F1, F2 y F3).
- **T16 – Preparar la defensa**, para la preparación de la defensa debemos acabar de dar estilo a la documentación que nos servirá como memoria, preparar unas diapositivas para usarlas de soporte en la defensa, y preparar la misma defensa. (F3 & T15 < T16).
- **T17 – Realizar la defensa**, cerraremos el proyecto realizando la defensa frente al tribunal. La fecha estimada es el día 29/01/2019 pero puede variar en el rango de días [23,29] de enero. (T16 < T17).
- **Reserva de 40h**, por si hemos tenido que atrasar la preparación de la defensa debido a algún problema de las tareas anteriores.

Recursos F4

Esta fase no precisa de recursos específicos de software más allá de Microsoft Office.

Recursos comunes en todas las fases del proyecto

- **Materiales:**
 - Ordenador Dell attitude E550 con procesador Intel Core i5-5300U, 128GB de disco duro y 8 GB de memoria RAM y pantalla de 15,6”.
 - Teclado Microsoft Wired Keyboard 200.
 - Ratón Targus AMU650.
 - Monitor AOC E2270SWN LCD con retroiluminación LED - 21.5’.
- **Software:**
 - Microsoft Office para realizar documentación y presentaciones.
 - Microsoft Outlook para comunicarse con el director y ponente del TFG.
 - Taiga para el seguimiento de las tareas y los sprints.
 - Git y GitLab para el control de versiones.
- **Humanos:**
 - Trabajador que desarrolla el proyecto con jornada de 8h diarias.
 - Director del proyecto en la empresa.
 - Ponente del proyecto en la facultad.

²⁰ Instituto de Ingeniería Eléctrica y Electrónica (Institute of Electrical and Electronics Engineers en inglés).

Resumen de tareas

Fase	Código	Tarea	Dependencias	Horas
F1	T1	Estudio de la documentación de Rasa Framework y posibilidades de la tecnología.	-	40
	T2	Definir contexto, justificación, alcance y metodología a seguir de nuestro proyecto.	T1	24,5
	T3	Definir las tareas a realizar y su estimación de tiempo.	T2	4,25
	T4	Definir la gestión del riesgo.	T3	4
	T5	Identificar y estimar costes, definir mecanismos para el control de gestión presupuestaria y realizar informe de sostenibilidad.	T4	9,25
	T6	Reuniones con ponente y con director TFG.	-	10
F2	T7	Definir objetivos de investigación de mercado	T1	8
	T8	Fijar estrategias de selección e interpretación de los datos de las tecnologías ajenas y propia.	T7	8
	T9	Obtener datos y analizarlos.	T8	20
	T10	Organizar los resultados y estructurarlos.	T9	8
F3	T11	Validación del alcance definido de nuestro proyecto.	F1 & F2	8
	T12	Instalación de Rasa Framework y realización de tutoriales.	-	40
	T13	Implementación e integración del chatbot.	T11 & T12	348
	T14	Pruebas del correcto funcionamiento del software.	-	54
F4	T15	Crear documentación.	-	65
	T16	Preparar la defensa.	F3 & T15	43
	T17	Realizar la defensa.	T16	1
		Reserva para problemas.	-	40
			TOTAL HORAS:	735

Tabla 1. Resumen de tareas.

7. GANTT Y ESTIMACIÓN

Para la estimación de horas, hemos intentado no sobrepasar el límite de 735 horas, que son las que disponemos en la empresa para realizar el proyecto. De esta forma, evitaremos tener que trabajar fuera del horario de trabajo. En la Tabla 2 podemos ver las horas que hemos planificado que nos llevará realizar cada tarea.

Fase	Código	Horas
F1	T1	40
	T2	24,5
	T3	4,25
	T4	4
	T5	9,25
	T6	10
F2	T7	8
	T8	8
	T9	20
	T10	8
F3	T11	8
	T12	40
	T13	348
	T14	54
F4	T15	65
	T16	43
	T17	1
	Reserva	40
TOTAL		735

Tabla 2. Previsión horas por tarea.

Gantt

Para el diagrama de Gantt que se muestra en la Figura 4 a continuación, hemos tenido en cuenta que solo se trabaja de lunes a domingo, que las tareas en paralelo no disponen de 8h al día cada una, que en las 735 horas ya están descontados los días festivos y que se acaba el proyecto el 29/01/2020 (Aunque la memoria se entrega unos días antes).

Hemos considerado que esa es la fecha fin del proyecto, ya que a nivel organizativo es la que se intentará elegir para hacer la defensa. De no ser así, simplemente es necesario adelantar la *T17 – Realizar defensa* y reducir ligeramente las horas reservadas a las desviaciones. Eso es posible, ya que la última tarea *T16 – Preparar defensa* finaliza el día 15 de enero y esta fecha es anterior a cualquier fecha posible de defensa.

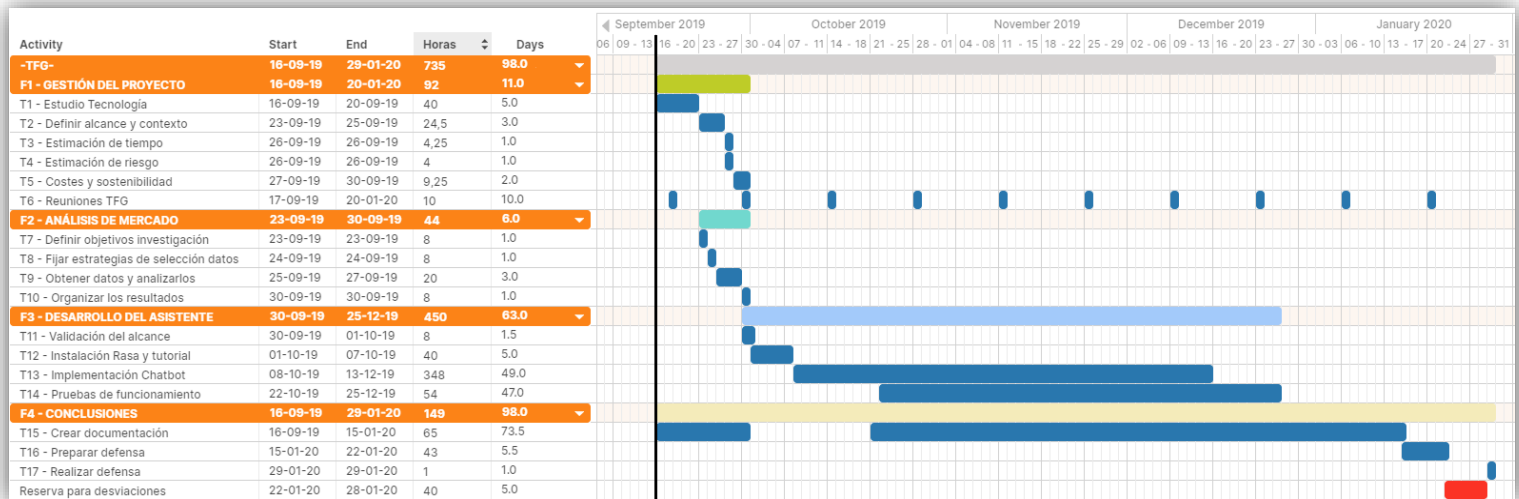


Figura 4. Diagrama de Gantt (Fuente: tomsplanner.es)

8. GESTIÓN DEL RIESGO

Hay tres motivos principales que pueden afectar en la entrega de nuestro proyecto: el tiempo, el framework y el presupuesto. A continuación los enumeramos y mostramos planes de acción. Si acabamos antes una tarea, empezaremos con la siguiente.

Framework

En el framework la única tarea que nos puede generar problemas es la integración del asistente con el canal de voz de Google Assistant y con la plataforma eVA de everis. Si nos encontramos con alguno de estos problemas, desistiremos en esa integración y usaremos únicamente como canal Telegram, donde el chatbot tendrá las mismas funcionalidades pero sin control de voz.

Tiempo

Si nos encontramos con problemas de tiempo, lo podemos solucionar usando las horas que hemos reservado para desviaciones, son un total de 40 horas. Si no tenemos suficiente con las 40 horas, nos tenemos que plantear que funcionalidades nos quedan por implementar y priorizar las más críticas. Si no es posible acabar con las críticas, debemos redefinir el proyecto cambiando las funcionalidades por otras más simples.

Presupuesto

Para problemas de presupuesto, solo nos queda buscar alternativas que sean gratis. Si no es posible, debemos ampliar las funcionalidades que se puedan desarrollar con herramientas gratuitas.

9. PRESUPUESTO

Ahora que hemos realizado la planificación del proyecto, debemos hacer un estudio de los costes, para saber si en relación a los ingresos que creemos que vamos a obtener con el proyecto, nos sale viable o no.

9.1 IDENTIFICACIÓN Y ESTIMACIÓN DE LOS COSTES

Para un buen cálculo del presupuesto que se destina al proyecto, primero de todo debemos identificar y estimar los costes relacionados con el proyecto, así que a continuación vamos a detallar las partidas²¹ que de las que se va a estimar su coste posteriormente.

9.1.1 Recursos humanos

En este apartado, se estimaran todos los costes relacionados con los roles que toman los trabajadores que participan en cada una de las tareas que hemos identificado en apartados anteriores así como los cálculos de coste estimado.

Los roles que se contemplan en este apartado son los siguientes:

- Jefe de proyecto: Responsable diferente del proyecto en la empresa. Se encarga de toda la gestión y de vender el producto a los clientes.
- Arquitecto: Se encarga de diseñar el sistema, requisitos, tomar las decisiones técnicas. Otra función que desarrolla es la de documentar la parte técnica del proyecto.
- Analista: Se encarga de coordinar, planear y recomendar opciones de software y sistemas, para cumplir los requerimientos del proyecto. También elabora parte de la documentación del proyecto.
- Desarrollador: Se encarga de desarrollar y mantener el sistema de software a partir de la documentación que recibe del analista y el arquitecto.
- Tester: Se encarga de llegar a cabo pruebas para evaluar el funcionamiento general del software. Los testers detectan errores y los comunican o sugieren formas de mejorarlo.

²¹ Gastos derivados de una actividad económica.

En la Tabla 3 que se muestra a continuación, podemos ver el promedio salarial que se ha tomado como referencia para cada rol.

Rol	Salario bruto + S.S promedio anual	Salario bruto + S.S promedio anual	Salario bruto + S.S promedio hora
Jefe de proyecto	35.600 €	48.060 €	27 €
Arquitecto	36.546 €	49.337 €	28 €
Analista	29.197 €	39.416 €	22 €
Desarrollador	25.112 €	33.901 €	19 €
Tester	25.563 €	34.510 €	20 €

Tabla 3. Promedio salarial según rol (Fuente: indeed.com)

En la **¡Error! No se encuentra el origen de la referencia.**4 que se encuentra a continuación, mostraremos dos cálculos del coste estimado, uno con el coste de los salarios teniendo en cuenta los diferentes roles que participan – el que se tomará en cuenta en este proyecto – y otro con el teniendo en cuenta el salario de becario que realiza todas las funciones. En (Equip Deganal, 2019), se ha define un salario de becario de 9€ la hora, pero ese valor no es lo que le cuesta realmente a la empresa, debido a que allí no se incluye el coste de la seguridad social. El coste real para la empresa es aproximadamente de 12€ la hora, y ese será el salario que tomaremos como referencia

Tarea	Rol	Horas	Coste estimado	Coste estimado becario
Estudio de la documentación de Rasa Framework y posibilidades de la tecnología.	Analista	40	894 €	480 €
Definir contexto, justificación, alcance y metodología a seguir de nuestro proyecto.	Jefe de proyecto	24,5	668 €	294 €
Definir las tareas a realizar y su estimación de tiempo.	Jefe de proyecto	4,25	116 €	51 €
Definir la gestión del riesgo.	Jefe de proyecto	4	109 €	48 €
Identificar y estimar costes, definir mecanismos para el control de gestión presupuestaria y realizar informe de sostenibilidad.	Jefe de proyecto	9,25	252 €	111 €
Reuniones con ponente y con director TFG.	Arquitecto / Analista / Jefe de proyecto	10	272 €	120 €
Definir objetivos de investigación de mercado	Jefe de proyecto	8	218 €	96 €
Fijar estrategias de selección e interpretación de los datos de las tecnologías ajenas y propia.	Arquitecto	8	224 €	96 €
Obtener datos y analizarlos.	Desarrollador	20	384 €	240 €
Organizar los resultados y estructurarlos.	Desarrollador	8	154 €	96 €
Validación del alcance definido de nuestro proyecto.	Analista	8	179 €	96 €
Instalación de Rasa Framework y realización de tutoriales.	Desarrollador	40	769 €	480 €
Implementación e integración del chatbot.	Desarrollador	348	6.688 €	4.176 €
User Acceptance Test (UAT)	Tester	54	1.056 €	648 €
Crear documentación.	Arquitecto / Analista / Jefe de proyecto	65	1.818 €	780 €
Preparar la defensa.	Jefe de proyecto	43	1.172 €	516 €
Realizar la defensa.	Jefe de proyecto	1	27 €	12 €
COSTE TOTAL			14.999 €	8.340 €

Tabla 4. Coste recursos humanos. (Fuentes: indeed.com y fib.com)

para el becario.

9.1.2 Recursos materiales

Los recursos humanos descritos en el apartado anterior, utilizan para realizar su trabajo, los recursos materiales que se detallan en la Tabla 5.

Para calcular la amortización se ha dividido el valor del producto entre los meses de vida útil y posteriormente, se ha multiplicado por los meses de proyecto. En algunos recursos, no se ha estimado amortización puesto que se alquilan a la empresa. El proyecto paga 100€ mensuales por persona para alquilar el material y espacio de trabajo.

	Precio	Unidades	Vida útil	Amortización	Coste total
Móvil Xiaomi mi9	435 €	1	36 meses (3 años)	48,33 €	48,33 €
Google Assistant	99 €	1	48 meses (4 años)	8,25 €	8,25 €
Ordenador Dell atitude E550	150€ alquiler mensual	1	Alquiler		600 €
Teclado Microsoft Wired Keyboard 200		1			
Ratón Targus AMU650		1			
Monitor AOC E2270SWN		1			
Espacio de trabajo (luz, aire acondicionado...)		1			
				TOTAL	656,58 €

Tabla 5. Coste recursos materiales. (Fuente: everis.com)

9.1.3 Recursos software

Para los recursos software, nos encontramos que todas las herramientas que se van a usar en este apartado tienen coste 0, ya que trabajaremos con tecnologías open source.

Las herramientas de pago que trae configuradas el ordenador, están incluidas en el alquiler mensual que se paga por el ordenador y el espacio de trabajo, por lo tanto, no se ha considerado oportuno añadirlas aquí.

9.1.4 Presupuesto total

Por lo tanto, tras detallar el presupuesto de los diferentes recursos que aparecen en nuestro proyecto, en la **¡Error! No se encuentra el origen de la referencia.**6 podemos ver el presupuesto total del proyecto – el presupuesto que consideramos es el de los roles – y se ha añadido una contingencia del 15% que nos servirá para cubrir obstáculos no previstos.

El único imprevisto que hemos encontrado en nuestra planificación es la desviación temporal, que la consideramos probable debido al desconocimiento de la tecnología. Creemos que esta desviación nos puede suponer un máximo de 40h, que ya las teníamos reservadas para este probable problema.

TIPO	PRESUPUESTO ROLES	PRESUPUESTO BECARIO
RECURSOS HUMANOS	14.999 €	8.340 €
RECURSOS MATERIALES	657 €	657 €
RECURSOS SOFTWARE	0 €	0 €
CONTINGENCIA (15%)	2.348 €	1.349 €
IMPREVISTOS (40h)	931 €	480 €
TOTAL	18.935 €	10.826 €

Tabla 6. Presupuesto total del proyecto.

9.2 CONTROL DE GESTIÓN

Para el control de gestión, debemos centrarlos principalmente en los recursos humanos, puesto que son los que tienen más peso en el presupuesto del proyecto.

En nuestro caso, los recursos materiales no necesitaran un seguimiento, ya que están relacionados con el tiempo. Eso es debido a que al ser un alquiler mensual, el coste del espacio de trabajo y material está relacionado con el tiempo que lo usemos. Por lo tanto, como se comentaba previamente, no necesitan un seguimiento especial ya que se hará a nivel temporal en los recursos humanos.

Los indicadores de este control de seguimiento serán las horas que se han destinado a cada tarea, si vemos que nos estamos excediendo en el tiempo, tendremos que empezar a usar tiempo dedicado a imprevistos. Por otra parte, podemos planificar puntos de control en algunas fases del proyecto, para ver si el presupuesto planteado es fiel a la realidad.

Si nos falla presupuesto que no está relacionado con las horas de trabajo – por ejemplo si tenemos que comprar material – lo usaremos del presupuesto destinado a contingencias, que estará repartido entre las diferentes fases para evitar excederlo.

10. INFORME DE SOSTENIBILIDAD

10.1 AUTOEVALUACIÓN

La sostenibilidad está presente en todos los proyectos que se desarrollan cada día, sin embargo, tal y como se comenta en (Penzenstadler, 2013), el alcance de dicha sostenibilidad debe ser de forma parecida en cada una de las tres dimensiones de la sostenibilidad – económica, ambiental y social –, así que no es correcto centrarse en una de las dimensiones a costa de perjudicar a alguna de las otras dos.

A continuación, vamos a detallar los puntos fuertes y los débiles que he observado tras contestar la encuesta sobre sostenibilidad. Uno de los grandes puntos fuertes se trata de que desde la universidad se ha trabajado muy en serio para que los alumnos entiendan la importancia de trabajar en la sostenibilidad de un proyecto, en las asignaturas comunes se han visto aspectos de sostenibilidad, pero además, cursé la optativa de ASMI²², donde se pueden ver con más detalle temas de sostenibilidad.

Pero no me considero un experto, ya que pocas veces he valorado la sostenibilidad en un proyecto de la universidad y desconozco herramientas para controlar el grado de sostenibilidad de un proyecto. Otro punto débil para valorar la sostenibilidad del proyecto es que desconozco los principios deontológicos²³ relacionados con la sostenibilidad.

Por otro lado, soy consciente de los problemas medioambientales por los que estamos pasando y de la importancia que se le tiene que dar a la sostenibilidad en todo proyecto. Eso sumado al conocimiento de las consecuencias tanto directas como indirectas que tienen los productos tecnológicos, hacen que mi ambición para que este proyecto sea lo máximo sostenible en las tres dimensiones sea mayor.

Para concluir esta autoevaluación, pienso que pese a que mi conocimiento sobre sostenibilidad no sea muy avanzado, soy consciente de la importancia de la sostenibilidad en un proyecto, así que se va a aprender con mayor profundidad la gestión de la sostenibilidad en este proyecto.

²² Aspectos Sociales y Medioambientales de la Informática.

²³ Principios generales de las normas éticas de conducta del profesional.

10.2 DIMENSIÓN ECONÓMICA

Como se ha comentado en apartados anteriores, hemos de realizar una evaluación de costes, para ver si nuestro proyecto puesto en producción (PPP) será viable económicamente. A simple vista podemos apreciar que el grueso de los costes se encuentra en los recursos humanos, hay costes materiales pero tienen poca relevancia en el total del proyecto. Por último, observamos que no hay costes relacionados con el software, ya que uno de los requisitos del proyecto era trabajar sin software de pago. En los otros proyectos que hay en el mercado, vemos que las empresas destinan presupuesto a los recursos de software, comprando las versiones de empresa, así que, nosotros mejoramos respecto a las demás opciones usando software gratuito, ya que las funcionalidades que vamos a usar no requieren la versión de pago.

10.3 DIMENSIÓN AMBIENTAL

Tras hacer un análisis de la sostenibilidad ambiental, encontramos que el proyecto tiene un impacto muy bajo. Eso es debido a que el principal causante es el consumo eléctrico de los recursos materiales. Los recursos materiales que se encuentran en la infraestructura de la oficina, tales como calefacción aire acondicionado y electricidad, se gestionan de forma excelente por parte de everis.

El ordenador es proporcionado por everis y va rotando entre el personal, es decir, al finalizar la beca se entrega para que sea usado por otro becario. Sólo se ha considerado usar un único ordenador ya que eso ayudará a reducir el impacto ambiental del proyecto, ya que no es necesario usar más ordenadores.

La parte de documentación sí que tiene un impacto ambiental, ya que la memoria final del proyecto debe ser impresa y además hacer varias copias. Se estima una cantidad de entre 60 y 120 páginas por memoria.

Por otra parte, al poder reusar código de las librerías de Rasa, se ha ahorrado tiempo y recursos, reduciendo así el consumo de electricidad. Cuando el proyecto esté en funcionamiento, se deberá usar un equipo antiguo o el propio equipo para ejecutar el bot, ya que los requisitos computacionales de la tecnología son bajos.

Respecto otras soluciones existentes, la principal mejora es que este proyecto se puede ejecutar únicamente en una única máquina, mientras que otras soluciones como por ejemplo Dialogflow, precisan de ejecutarse en un servidor externo, además de en la propia máquina.

10.4 DIMENSIÓN SOCIAL

Este proyecto, de forma personal me va a aportar conocimiento sobre la gestión de un proyecto complejo, también me ayudará para aprender nuevas tecnologías para la creación de asistentes virtuales y muchos otros conceptos que no se han visto de forma práctica en la universidad.

Actualmente, a nivel de nuestra empresa no se ha trabajado nunca con Rasa y no hay ninguna solución funcional parecida para gestionar los eventos que se realizan. Así que nuestra solución, va a servir para ayudar a los asistentes de estos eventos, y por lo tanto, ayudar también a la empresa a mejorar su imagen a ojos de sus empleados y clientes.

Además del caso práctico que se ha elegido para este proyecto, la parte fundamental es la tecnología, de forma que cada empresa la puede adaptar a su caso particular y sacarle el máximo partido posible.

Tras analizar el uso que se le va a dar al proyecto, creo profundamente que sí que existe una necesidad real del proyecto, ya que se va a dotar a la empresa del conocimiento en una nueva tecnología que puede abrir puertas para futuros proyectos.

11. ANÁLISIS DE MERCADO

En este apartado, se analizará el contexto en el que se encuentra nuestro proyecto y eso implica analizar de forma exhaustiva los problemas o productos similares existentes en el mercado. De la misma manera, debemos identificar y comparar las tecnologías y soluciones que se pueden usar en nuestro problema.

En el sector hay multitud de frameworks, y cada vez van apareciendo más ya que las grandes empresas ven una oportunidad de negocio en el mundo de los asistentes conversacionales. En (Gavilán, 2018), se muestran los siguientes motivos por los cuales todas las empresas están empezando a considerar implementar un asistente:

- **Avances en tecnología de lenguaje natural:** El autor comenta que uno de los factores el auge de los asistentes conversacionales son los avances en materia de inteligencia artificial y en todo lo que tiene que ver con entendimiento y procesamiento del lenguaje natural. Aunque se afirma que se puede avanzar más, el autor también comenta que la tecnología ya es suficientemente madura como para permitir usos comerciales y a gran escala.
- **Disponibilidad de la tecnología:** Este es otro de los puntos que señala el autor, pues el coste para acceder a las tecnologías es muy bajo o nulo.
- **Impacto en el negocio:** El autor concluye el análisis indicando que todo lo anterior adquiere sentido porque las organizaciones y empresas que hacen uso de asistentes conversacionales pueden obtener un gran valor de las soluciones que ofrecen, en materia de experiencia de cliente, eficiencia, reducción de costes, etc.

Para nuestro proyecto, hemos decidido comparar las funcionalidades y posibilidades de varios frameworks cuya funcionalidad es la de crear asistentes. En concreto hemos comparado tres de las grandes opciones del mercado; Dialogflow, Watson y Lex, con la que tenemos pensado descubrir, Rasa.

11.1 DIALOGFLOW



Figura 5. Logo dialogflow (Fuente: dialogflow.com)

En (Gelfenbeyn, 2016) encontramos el anuncio de Google sobre la adquisición de api.ai en 2016, la plataforma de donde surgió el proyecto conocido como Dialogflow. No es de extrañar su buen funcionamiento, ya que Dialogflow es la solución del gigante Google a la necesidad del mercado de poder desarrollar asistentes de manera sencilla y a bajo costo.

Este se trata de uno de los frameworks más populares para la creación de asistentes. Su popularidad es debida principalmente por dos factores: la empresa que tiene detrás y la facilidad de su uso. El factor del bajo costo y su facilidad de uso es una de sus claves del éxito, ya que anteriormente estos frameworks tenían unos precios de licencia bastante altos. En cambio, Google permite usar sus herramientas de forma gratuita para un uso básico.

No obstante, si va a ser usado por bastante gente, su coste irá en función de las interacciones que reciba. La tarifa más reducida se encuentra en los asistentes que no tienen audio y el coste tras pasar el saldo inicial gratuito (200 USD) es de 0,002 USD por solicitud.

Este bajo coste ha supuesto que empresas como IBM que disponen de otros frameworks de creación de asistentes, se vean obligadas a bajar sus precios para poder competir contra Dialogflow.

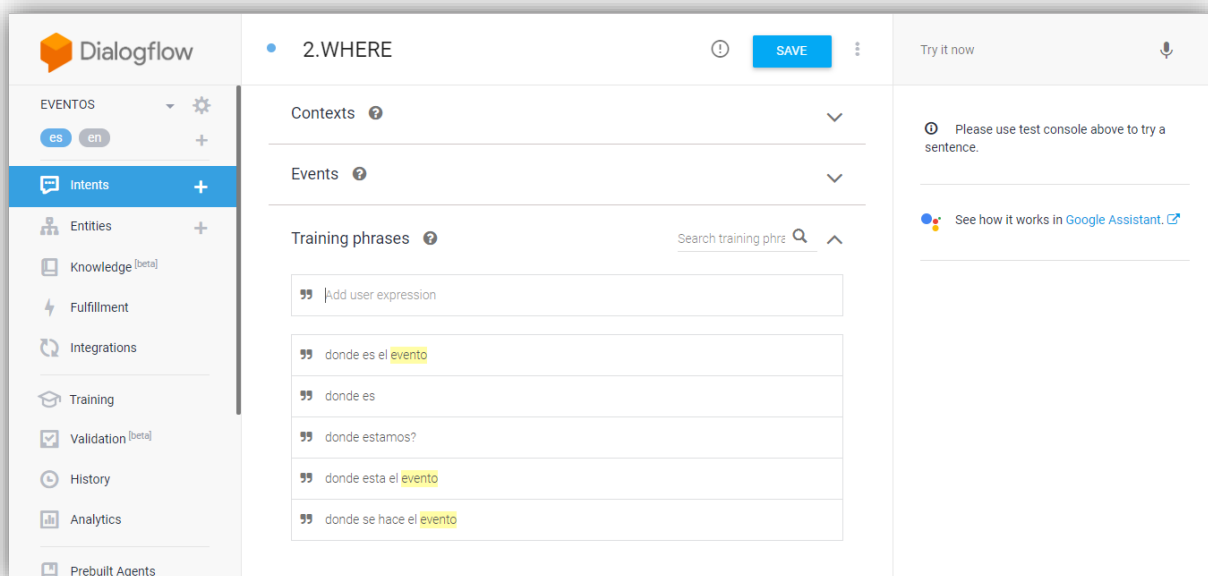


Figura 6. Interfaz de usuario de Dialogflow (Fuente: dialogflow.com)

11.2 WATSON



Figura 7. Logo Watson Assistant (Fuente: cloud.ibm.com)

Watson Assistant es el framework de creación de asistentes desarrollado por IBM hace poco menos de 2 años. Actualmente se puede encontrar dentro del IBM Cloud.

Una de las claves del éxito de este asistente es que brinda a los desarrolladores la posibilidad de aislar la información que obtiene su asistente en una nube privada. De esta manera se pueden proteger los conocimientos resultantes de la interacción del usuario con el asistente. Los usuarios pueden interactuar con la aplicación del desarrollador a través de una variedad predefinida de canales de texto y voz. También se les ofrece la posibilidad de personalizar la interfaz para que no parezca que el asistente trabaja con la plataforma de Inteligencia Artificial Watson.

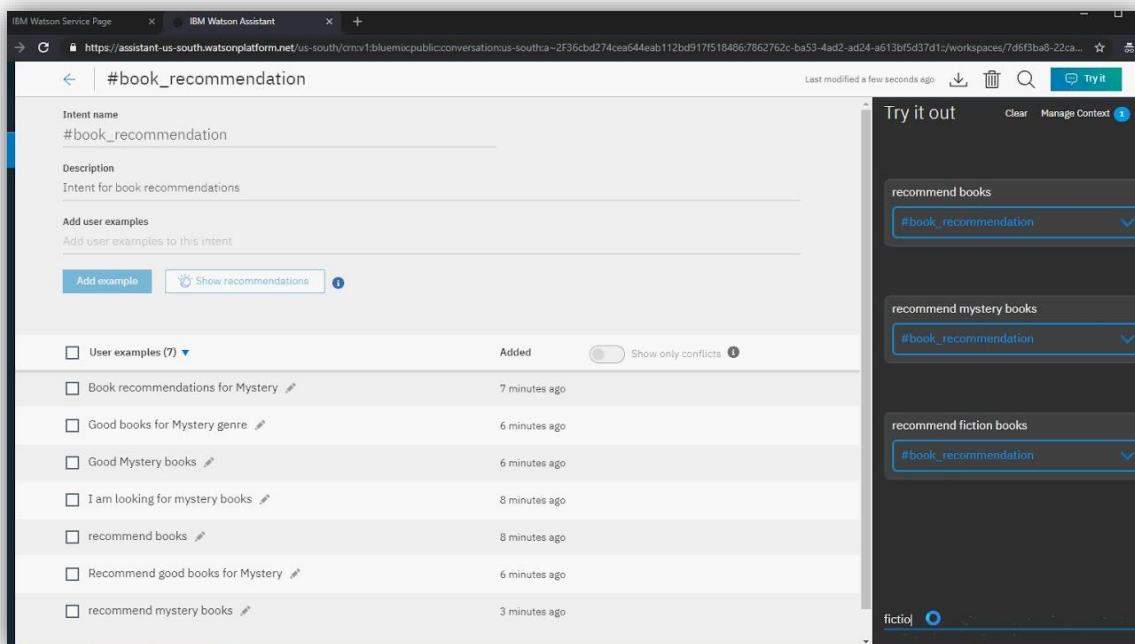


Figura 8. Interfaz de usuario de Watson Assistant (Fuente: developer.ibm.com)

11.3 LEX



Figura 9. Logo Amazon Lex (Fuente: aws.amazon.com/lex)

Amazon Lex es un servicio de AWS que se utiliza para construir asistentes conversacionales que se pueden usar mediante texto y voz (hasta 15 segundos de entrada de voz). Tiene mucha importancia en el mercado, ya que de él se nutre el asistente de Amazon llamado Alexa, presente en todos sus dispositivos echo.

Otra de las características es que, como nos tiene acostumbrados Amazon, se permite la integración con otros servicios de Amazon de forma muy sencilla pero nos dificulta realizar esta integración con productos externos. Pese a ello, cuenta con bastantes entornos de programación (Node.js, Python, .NET, Java, Javascript, PHP y Ruby).

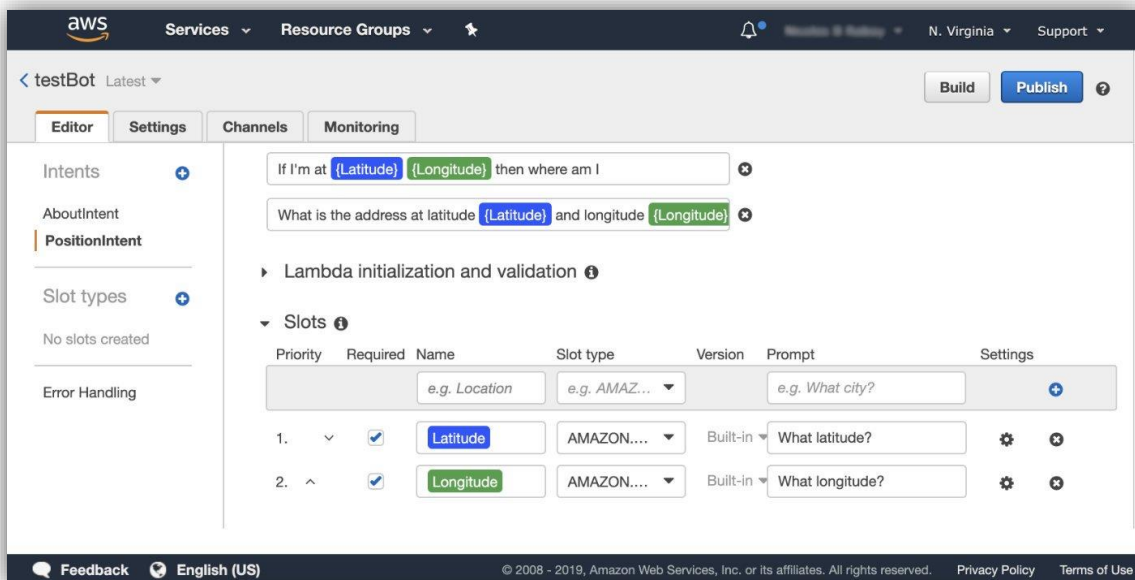


Figura 10. Interfaz de usuario de Amazon Lex (Fuente: developer.amazon.com)

11.4 RASA



Figura 11. Logo Rasa (Fuente: rasa.com)

Rasa es una empresa de software, con central en Berlin que ofrece soluciones para crear chatbots de forma Open-Source y en Python. Al ser Open-Source proporciona muchas ventajas respecto a otras alternativas, ya que los datos del usuario que usa el asistente no pasan por terceros, y al disponer del framework localmente instalado, aunque el software deje de mantenerse, los chatbots pueden funcionar sin problemas.

El framework de Rasa consta de dos partes:

- **RASA NLU:** Esta parte se encarga del lenguaje natural, para ello se nutre de los idiomas disponibles en la librería spaCy. Su misión es realizar el procesado del mensaje y su posteriormente transformarlo en datos con estructura concreta.
- **RASA Core:** Esta parte se encarga de la gestión del diálogo, se conecta con RASA NLU. El componente decide qué acciones tomar en cada momento haciendo uso de un modelo de machine learning creado con tensorflow y keras.

Pese a que existe la versión Rasa X que cuenta con interfaz gráfica para poder crear los elementos del asistente como en las otras opciones descritas anteriormente, **para este proyecto hemos usado la versión normal**, así que todos los elementos los hemos creado en formato markdown, tal y como se muestra en la siguiente Figura.

```

nlu.md
34  ## intent:filtro_ponente
35  - Dame las charlas hechas por [Mickael](nombre_ponente)
36  - Charlas de [Lucas](nombre_ponente)
37  - Dame las charlas hechas por [Álvaro](nombre_ponente)
38  - Charlas donde el ponente es [Maria](nombre_ponente)
39  - Charlas en las que habla [Javi](nombre_ponente)
40
41  ##intent:filtro_categoria
42  - Charlas sobre [Innovación](nombre_categoria)
43  - Charlas sobre [Cursos](nombre_categoria)
44  - Charlas sobre [Test](nombre_categoria)
45  - Charlas sobre [Ecología](nombre_categoria)
46  - Charlas que hablen sobre [Sostenibilidad](nombre_categoria)
47
48  ##intent:filtro_fecha
49  - Charlas del día [13/12/1995](formato_fecha)
50  - Charlas del día [25/08/1987](formato_fecha)
51  - Charlas del día [14/06/2019](formato_fecha)
52  - Charlas del día [11/11/2019](formato_fecha)
53  - Charlas del día [](formato_fecha)
54  - Solo las que se hacen el [13/12/1995](formato_fecha)
55  - Solo las que se hacen el [25/08/1987](formato_fecha)
56  - Solo las que se hacen el [14/06/2019](formato_fecha)
57  - Solo las que se hacen el [](formato_fecha)
58
    
```

Figura 12. Interfaz de usuario de Rasa.

En la tabla que se muestra a continuación, podemos ver el resumen de las diferentes opciones que hemos valorado y las empresas propietarias.

Framework	Empresa detrás
Dialogflow	Google
Watson	IBM
Lex	Amazon
Rasa	--

Tabla 7. Comparativa Mercado

Son varios los aspectos que hemos tenido en cuenta para analizar las diferentes tecnologías:

- **Facilidad curva de aprendizaje:** Este concepto se refiere al grado de éxito obtenido durante el aprendizaje en el transcurso del tiempo.
- **Soporte de la comunidad:** En este aspecto, se valora si el framework tiene una comunidad detrás dispuesta a ayudar, por ejemplo, respondiendo dudas en algún foro.
- **Documentación existente:** Se valora la cantidad y calidad de la documentación que existe tanto oficial como creada por la comunidad.
- **Madurez tecnológica:** Concepto derivado del que fue desarrollado por la NASA en la década de los 70 para cuantificar en una escala del 1 al 9 el grado de evolución de una tecnología. El nivel 1 es la observación y reporte de principios y el nivel 9 se considera una tecnología probada con éxito en misiones.
- **Capacidad de personalización:** Consiste en la capacidad de la tecnología para adaptarse a la solución particular del cliente. En concreto, se cuantifica la capacidad del asistente de realizar acciones personalizadas más allá de devolver texto predefinido.
- **Actualización de software:** Consiste en analizar si los desarrolladores introducen cambios y añaden funciones nuevas a la tecnología.

Al ser difícil dar valor numérico a los aspectos anteriores, los cuantificaremos en una **escala del 1 al 10** y luego haremos la **media ponderada** para quedarnos con la tecnología con mayor puntuación.

Para nuestro proyecto, hemos considerado más importantes algunos aspectos que otros, así que hemos **ponderado cada aspecto de forma diferente**. En concreto, se ha dado más valor a la facilidad de aprendizaje, la comunidad y la documentación ya que no se tiene experiencia previa en la creación de chatbots.

En la tabla que se muestra a continuación, se puede observar los resultados del estudio:

	Dialogflow	Watson	Lex	Rasa
Aprendizaje (20%)	7	7	6,5	5
Comunidad (20%)	4	2	5	9
Documentación (20%)	6,5	9	7	9
Madurez (15%)	8	10	6,5	4
Personalización (15%)	6,5	6	5,5	7
Actualización (10%)	7	8	7	8
TOTAL	6,375	6,8	6,2	7,05

Tabla 8. Tabla de puntuaciones comparativa tecnologías

Con los resultados de la tabla, obtenidos teniendo en cuenta los pesos distribuidos según nuestro criterio de importancia, se ha concluido que **Rasa es una buena elección para desarrollar nuestro proyecto**, ya que se consolida como una opción muy buena para iniciarse en el mundo de los asistentes y además, es una tecnología de código abierto en constante actualización.

En las siguientes páginas, procederemos a justificar cada puntuación de la tabla.

Dialogflow

- **Aprendizaje (7):** Este es uno de los puntos en los que destaca el framework, ya que cómo casi todas las herramientas de google, tiene una interfaz muy intuitiva. Le hemos dado la mayor puntuación en este aspecto ya que hemos valorado muy positivamente que se pueda tener un asistente funcional sin tener que usar código, simplemente usando la interfaz que nos proporciona Dialogflow. Cierto es que para funcionalidades complejas, más allá de un bot de FAQs, sí que necesita la implementación de código, pero la plataforma resulta muy intuitiva.
- **Comunidad (4):** Este es el punto en el que flojea el framework, ya que dispone de un foro público pero este es complicado de localizar –se encuentra dentro de support.google.com– y además no es demasiado activo, así que las respuestas que se ofrecen son pocas. La solución que tienen que tomar los desarrolladores, es acceder a un foro externo como por ejemplo Stackoverflow.
- **Documentación (6,5):** Hemos observado que pese a no tener foro, sí que dispone de documentación oficial –no muy extensa– de cómo funciona la herramienta, además, al ser tan conocido se puede encontrar mucha información por internet.
- **Madurez (8):** La madurez es el punto más robusto del framework. Hemos considerado oportuno darle esta puntuación ya que estamos ante un framework maduro. Lleva años en el mercado siendo usado por grandes empresas y no solo Dialogflow como lo conocemos actualmente, sino que también su predecesor api.ai se usó como solución para desarrollar muchos asistentes conversacionales.
- **Personalización (6,5):** En este punto Dialogflow no destaca mucho, ya que todo se desarrolla desde su plataforma y la personalización se vuelve limitada. Pese a ello, se pueden programar eventos transaccionales con código en diversos lenguajes de programación.
- **Actualización (7):** Pese a que no hay grandes versiones –hasta la fecha han sacado únicamente la versión 1 y la 2– sí que es cierto que cada 10 días suelen sacar pequeñas versiones que eliminan funcionalidades obsoletas o añaden mejoras. Por ello, creemos que el nivel de actualización es muy correcto, pero no el mejor de todas las opciones disponibles.

Watson

- **Aprendizaje (7):** La herramienta es muy intuitiva y sencilla. Visualmente se pueden encontrar todos los elementos necesarios y salvo que se quiera hacer algún asistente muy complejo, se puede desarrollar sin demasiados conocimientos.
- **Comunidad (2):** No existe ningún foro de la tecnología, y en foros para resolver dudas genéricas como por ejemplo Stack Overflow²⁴, hay muy pocas dudas sobre el framework. El único canal de dudas es un Slack interno de IBM, al que es difícil acceder desde fuera de la compañía.
- **Documentación (9):** Como se comenta en el punto anterior, la documentación no está pública y desde fuera de la compañía es complicado acceder a ella. Pero una vez se ha accedido a esta documentación a través de Slack o de la plataforma, esta es extensa y se encuentran múltiples ejemplos y vídeos de cualquier parte de la creación de un asistente.
- **Madurez (10):** La tecnología de Watson es muy madura, de echo consideramos que es la más madura de las que se comparan en esta sección.
En (Fernández, 2018) se hace un análisis de la historia de los chatbots y en el, se menciona que el primer asistente de IBM se desarrolló en 1996 y fue llamado “Deep Blue” y en 2004 surgió Watson, mucho antes que sus competidores. Desde 2004 se ha ido usando esta tecnología y mejorando cada vez más. También usa *Watson Conversation* y *Watson Virtual Agent*, otras tecnologías²⁵ de IBM que han sido ampliamente usadas en otros ámbitos.
- **Personalización (6):** Watson no destaca por tener una gran personalización dentro de la propia interfaz, pero en cambio ofrece APIs para cada componente del asistente, por lo tanto, se puede integrar en diferentes canales haciendo uso de estas APIs o se puede consultar el NLU desde una aplicación externa.
Por otra parte, en la última versión del asistente, se puede configurar un archivo de información del bot y en caso de no tener definido el intent del usuario, buscará esa información en el archivo proporcionado.
- **Actualización (8):** Otro punto fuerte del framework es su constante actualización, pues cada 10 días más o menos van actualizando problemas y añadiendo funcionalidades nuevas. Al tratarse de pequeños cambios, las APIs no han sufrido una renovación total, así que únicamente está disponible la V1 y V2.

²⁴ Sitio web para preguntar y resolver problemas de programación en diferentes lenguajes.

²⁵ Servicios cognitivos de IBM.

Lex

- **Aprendizaje (6,5):** Para este punto, hemos tenido en cuenta que el framework se usa vía interfaz web. Pero esta interfaz no resulta tan intuitiva como la de las otras opciones, ni tampoco el uso de Lambdas. Por ello, se considera que es relativamente sencillo hacer un bot con Lex, pero no tanto como en Dialogflow o Watson.
- **Comunidad (5):** La comunidad de Amazon Lex es muy pobre, dispone de un foro público que se puede encontrar en la siguiente dirección web: <https://forums.aws.amazon.com/forum.jspa?forumID=251>, pero es muy poco activo (Aunque más que el de Dialogflow). Por ejemplo, tenemos preguntas del día 2 de diciembre, y la siguiente es del día 16. Por otra parte, casi todas las preguntas están sin responder, pese a tener más de 1000 visitas.
- **Documentación (7):** La documentación de Lex es correcta, en la página de AWS, podemos encontrar una sección de documentación de Lex. Encontramos una guía de desarrollador con información sobre cómo empezar, uso de funciones Lambda²⁶, el despliegue del bot, seguridad, uso de las APIs y ejemplos de bots.
- **Madurez (6,5):** Lex se encuentra en un estado de Madurez bastante bajo. Eso es debido a que el producto salió el 19 de abril de 2017, en (Barr, 2017) podemos ver el anuncio de salida. También se ha considerado razonable darle una nota más baja en este aspecto que su competidor Dialogflow, ya que Lex ha nacido de 0 y en cambio Dialogflow viene de api.ai.
- **Personalización (5,5):** En cuanto a la personalización, podemos decir que Lex nos ofrece un nivel bajo de personalización. El motivo de esta puntuación es que se trata de un framework básico y cerrado, ya que Amazon siempre intenta proteger sus productos de conexiones con terceros. Por otra parte, no dispone de muchos canales de comunicación además de Alexa.
- **Actualización (7):** La actualización es decente, sobre todo en los últimos meses han empezado a trabajar en nuevas actualizaciones. Entre ellas destaca el soporte para realizar análisis de sentimientos y el Fallback intent, que proporciona una respuesta cuando la entrada que recibe el asistente es desconocida.

²⁶ Funciones personalizadas que se activan en respuesta de un evento.

Rasa

- **Aprendizaje (5):** El aprendizaje de esta tecnología no es sencillo. El motivo es que el asistente no se desarrolla usando ninguna interfaz gráfica, sino que se modifican los ficheros directamente. Los ficheros que hay que saber manejar para empezar a hacer un asistente sencillo son los que definen la información del bot y se encuentran en formato markdown²⁷ y los de configuración del asistente y los canales que son ficheros en formato yml. Por otra parte, también hay que saber desplegar el servidor, ya que se tiene que hacer con una serie de comandos a través de la terminal.
- **Comunidad (9):** Sin duda alguna, la ilusión que aporta el equipo a este framework es latente en cada aspecto de la tecnología, y esta ilusión se traslada a la comunidad. Rasa tiene un foro (forum.rasa.com) donde se pueden publicar dudas o preguntas y los tiempos de respuesta son muy cortos, en torno a los 10-15 minutos. En el foro, participan tanto el equipo de desarrolladores como usuarios que usen el framework.
- **Documentación (9):** Este es el punto que más me ha gustado del framework, pues la gente que hay detrás de Rasa se preocupa en documentar cada cambio, realizar tutoriales e incluso hay una masterclass en youtube donde semanalmente se explican conceptos de la tecnología y se muestran ejemplos. A diferencia de las demás opciones, se aprecia que los desarrolladores se preocupan para que la gente que se inicia en el mundo de los asistentes conversacionales pueda entender cómo funciona su producto.
- **Madurez (4):** El punto negativo de Rasa es la madurez tecnológica. El problema que tiene es que es un producto muy nuevo ya que la primera versión salió en 2017, en (Nichol, 2017) podemos ver uno de los primeros anuncios de uno de los creadores en el blog. Al ser un producto bastante nuevo, hay funcionalidades que no funcionan como deberían en algunos casos y poco a poco se van solucionando mediante parches. Un ejemplo de ello, es los problemas derivados de usar únicamente el NLU de Rasa sin usar todo el paquete, ya que las descargas.
- **Personalización (7):** El grado de personalización de la tecnología es bastante alto, ya que al poder manipular todos los ficheros es sencillo modificar configuraciones o añadir nuevas opciones. Por ejemplo, podemos añadir nuevos conectores, modificar las pipelines de configuración, editar la configuración de idioma del asistente, etc.

²⁷ Lenguaje de marcado que facilita el formatear un texto.

Por otra parte, el uso de Python como lenguaje de programación para las acciones personalizadas es una ventaja, ya que es un lenguaje muy versátil con una gran cantidad de librerías disponibles.

- **Actualización (8):** Otro de los puntos donde destaca esta tecnología, pues al ser relativamente nueva en el mercado, constantemente están sacando actualizaciones y parches para corregir problemas. Una de las ventajas es que se trata de un software de código abierto, así que hay una comunidad de programadores bastante extensa que va sacando versiones y mejoras. Recientemente han sacado la versión de Rasa X, que incluye una interfaz gráfica.

12. FRAMEWORK Y LIBRERÍAS UTILIZADAS

En esta sección, vamos a comentar que frameworks hemos usado para crear el asistente –adicionalmente al framework de Rasa, que hemos descrito en la sección del Análisis de mercado– y también que librerías se han usado tanto en el código cómo internamente por Rasa.

12.1 DIAGRAMA DE PAQUETES USADOS

Para poner en contexto al lector, se ha elaborado un diagrama de paquetes, con las diferentes librerías, paquetes usados y dependencias entre estos. Este diagrama nos será útil para tener una visión más general de las dependencias entre librerías y paquetes de nuestro asistente.

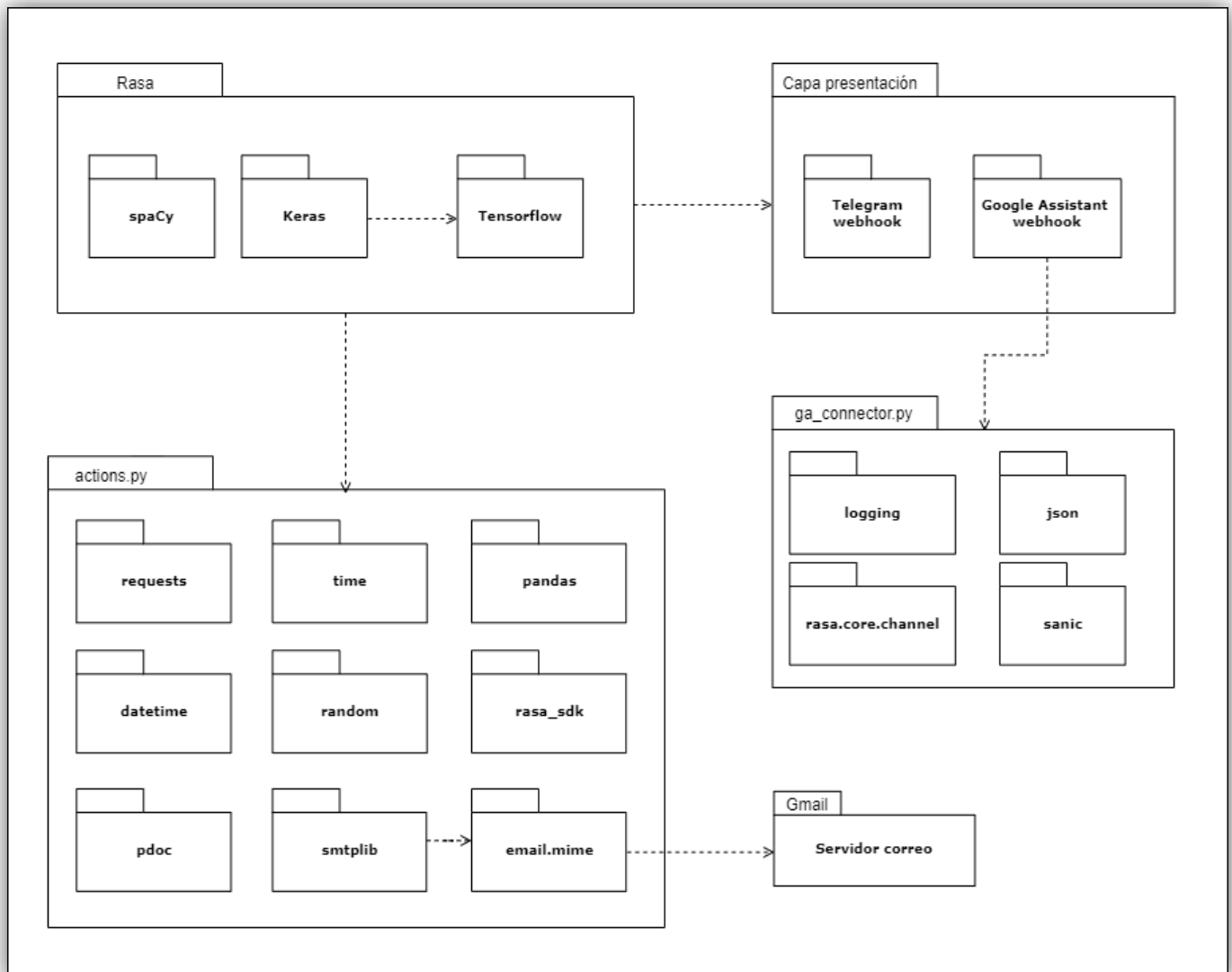


Figura 13. Diagrama de paquetes.

12.2 BOT DE TELEGRAM

Se ha decidido usar *Telegram Messenger*, ya que tras analizar el estudio (Sutikno et al., 2016) publicado en el *International Journal of Electrical and Computer Engineering (IJECE)*, donde se comparan las principales plataformas de mensajería instantánea gratuitas que podemos encontrar en el mercado –Whatsapp, Viber y telegram–, nos ha parecido conveniente usar esta plataforma debido a tres motivos: Es una plataforma open-source, su nivel de seguridad es el más alto y la facilidad de creación y uso de los bots.

Para crear un bot en telegram es muy sencillo, simplemente hace falta hablar con un Bot creado por telegram llamado BotFather (<https://telegram.me/BotFather>). Con este bot se puede gestionar los demás de forma sencilla, ya que nos permite crear bots y configurarlos: modificar imagen de perfil, descripción, información sobre el bot, lista de comandos, etc.



Figura 14. Logo Botfather (Fuente: core.telegram.org/bots)

Una vez hemos iniciado la conversación con el bot y hemos seguido los pasos para crear el asistente, este nos proporciona un token –que no expira– de acceso al bot. Este token se trata de una cadena de caracteres que nos permite controlar el asistente mediante llamadas HTTP a la API. En la Figura mostrada a continuación, podemos ver la interacción del usuario con el bot y todas las posibles configuraciones.

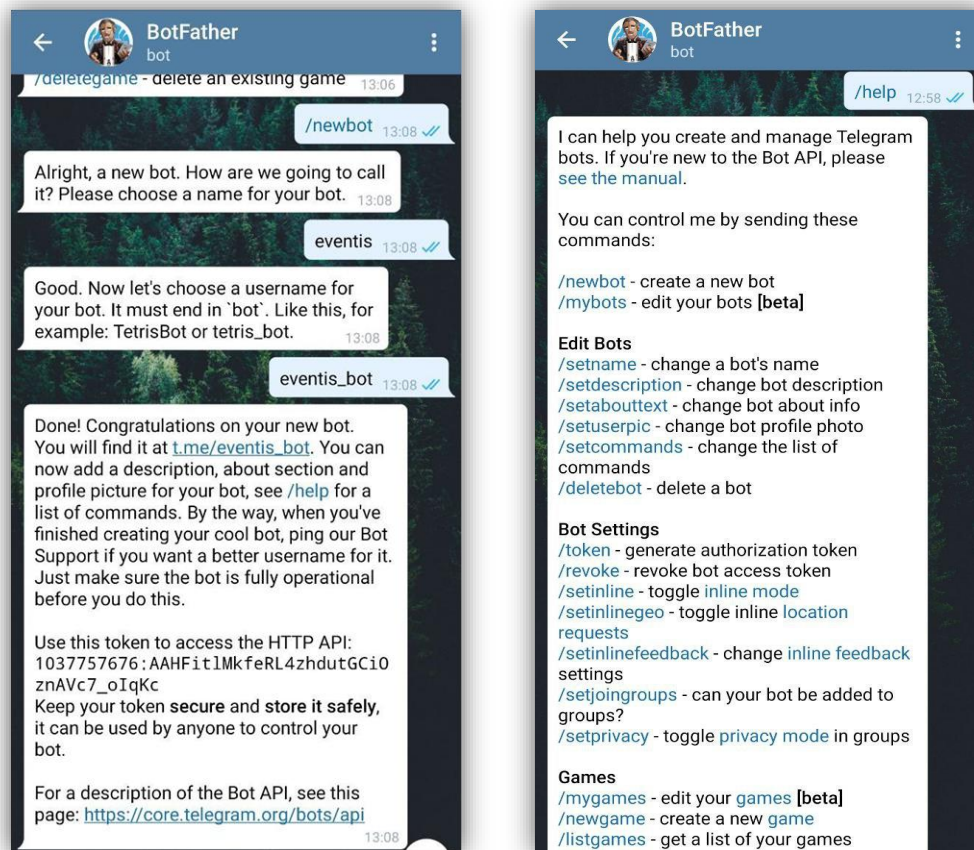


Figura 15. Interacci3n con BotFather

En las secciones que explicamos m1s adelante, mostraremos el uso que hacemos de esta API para conectar el bot de telegram con Rasa.

12.3 SPACY

spaCy es una librería de Python que se encarga del procesamiento de lenguaje natural. La librería es usada internamente por Rasa, ya que se nutre de ella para procesar el lenguaje natural del usuario.

En (Nasser, Al Omran, & Treude, 2017), se hace una comparativa de diferentes librerías para NLP (Google's SyntaxNet, the Stanford CoreNLP Suite, the NLTK Python library y spaCy) y se concluye en que, en general, spaCy proporciona mejor experiencia que las demás, con un 90% de acierto. Además, spaCy ha sido diseñada dando prioridad a su eficiencia y rapidez, además incluye una gran variedad de funcionalidades, a continuación mostramos las que se han usado en el proyecto:

- **Tokenización:** Antes de la interpretación de los mensajes, se usa la librería para hacer el análisis léxico de cada palabra.

```
Apple está buscando comprar una startup del Reino Unido por mil millones de dólares.  
Apple PROPN nsubj  
está AUX aux  
buscando VERB ROOT  
comprar VERB xcomp  
una DET det  
startup NOUN obj  
del ADP case  
Reino PROPN nmod  
Unido PROPN flat  
por ADP case  
mil NUM nummod  
millones NOUN obl  
de ADP case  
dólares NOUN nmod  
. PUNCT punct
```

Figura 16. Ejemplo de tokenización de spaCy (Fuente: spacy.io)

- **Dispone de vectores pre-entrenados:** Los llamados “Word Embeddings”, que consiste en la conversión de palabras a vectores de números reales, partiendo del supuesto que palabras que se encuentran en un espacio parecido, deben tener algún tipo de relación.
- **Soporte multidioma:** La librería dispone de gran variedad de paquetes de idiomas, en este caso llamados corpus. Para el proyecto, se ha descargado el idioma de su página web²⁸ y usado el corpus en castellano en su versión más ligera. Para descargarlo e instalarlo, sólo hace falta ejecutar un comando de Python:

```
$ python -m spacy download es_core_news_md
```

Figura 17. Comandos instalación spaCy (Fuente: spacy.io)

²⁸ El corpus se ha descargado de <https://spacy.io/models/es>

12.4 PANDAS

Pandas es una librería de Python destinada al análisis y tratamiento de datos. Pandas proporciona unas estructuras de datos propias que permite trabajar con ellos de forma muy sencilla y eficiente, además de permitir una sencilla visualización de los datos. En (McKinney & Team, 2015) se describen todos los tipos de estructuras que nos ofrece y las funciones asociadas a estas, pero en este caso solo no sería útil usar dos de ellas. Las principales estructuras que nos servirían son:

- **Series:** Se trata de vectores unidimensionales con indexación (con etiqueta o índice). Estas estructuras pueden generarse a través de los diccionarios.
- **DataFrame:** Es el tipo de estructura que hemos usado en nuestro proyecto, al tener nuestros datos almacenados en un documento de Excel, nos ha servido para poder convertir estos datos en tablas y realizar las consultas sobre ellas.

En la Figura que se muestra a continuación, vamos a ver un ejemplo básico de cómo hemos usado pandas en nuestro proyecto (el código real es más complejo).

```
import pandas as pd 1
df = pd.read_excel('charlas.xlsx', 'charlas') 2
df['Titulo'][0] 3
seleccion = df.loc[df['Categoría']=='gustos] 4
seleccion2 = seleccion.reset_index(drop=True)
```

Figura 18. Ejemplo de uso Pandas.

- 1) Para usar pandas, lo primero que debemos hacer es instalar e importar la librería, por convenio de la comunidad de desarrolladores se usa pd como alias.
- 2) Debemos construir el DataFrame sobre el que vamos a operar, en este caso leemos la hoja "Charlas" del archivo de Excel "charlas.xlsx". Se ha usado la función `read_excel()` pero también cuenta con funciones como `read_csv()` para leer diferentes tipos de archivos.
- 3) Por último, ya podemos operar sobre nuestro DataFrame, en este caso podemos tratarlo como una matriz donde al primer índice se accede por nombre de columna y la segunda dimensión es el número de fila.
- 4) También hemos hecho divisiones de los DataFrames según una búsqueda, conservando únicamente las columnas que tienen el valor que nos interesa. Se ha tenido que hacer un reset del número de fila ya que se mantenían los del DataFrame original y daba problemas al iterar por ese número.

12.5 SMTPLIB

Esta se trata de una librería de Python que sirve para enviar correos electrónicos haciendo uso del protocolo SMTP, pudiendo añadir código HTML.

El Protocolo simple de transferencia de correo (SMTP en inglés), se encarga de establecer una conexión entre los servidores de correo electrónico.

Antes de todo, debemos permitir el acceso de aplicaciones poco seguras a nuestra cuenta de Google, ya que por defecto Google no permite el inicio de sesión a través de smtp ya que lo tiene etiquetado como “menos seguro”. Para ello solo debemos otorgar permisos de acceso a las aplicaciones menos seguras en la web correspondiente²⁹.

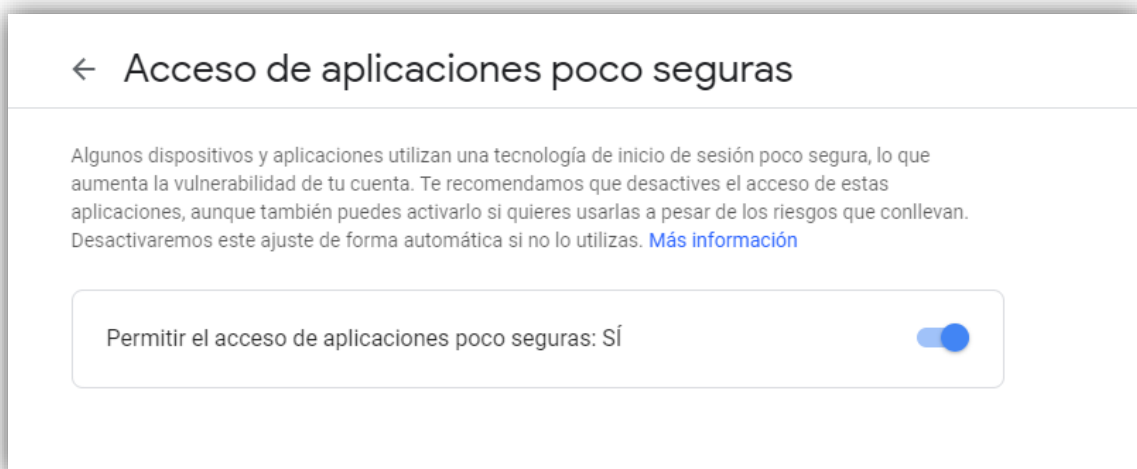


Figura 19. Permitir aplicaciones poco seguras (Fuente: myaccount.google.com/lesssecureapps)

Una vez hemos dado permiso, debemos hacer los siguientes pasos:

- 1) Crear un objeto del mensaje del tipo MIMEMultipart³⁰ donde almacenaremos la información del mensaje.
- 2) Configurar los parámetros de inicio de sesión.
- 3) Definir el mensaje que se va a enviar.
- 4) Añadir el cuerpo del mensaje al objeto creado.
- 5) Creamos conexión con el servidor, en este caso hemos usado la dirección 'smtp.gmail.com: 587' ya que es el servidor SMTP gratuito de Google. Si fuese necesario, se podría usar un servidor SMTP propio.
- 6) Iniciamos sesión en el servidor de Google anterior.
- 7) Enviamos el mensaje a través del servidor.
- 8) Cerramos la conexión.

²⁹ Para permitir el acceso acceder a <https://myaccount.google.com/lesssecureapps>

³⁰ Parte de Multipurpose Internet Mail Extension

```

# importamos las librerías necesarias
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
import smtplib

# Creamos instancia del objeto del mensaje 1
msg = MIMEMultipart()

#Configuramos los parámetros de inicio de sesión 2
user = "emisor@gmail.com"
password = "tucontraseña"

#Configuramos los parámetros del mensaje 3
msg['From'] = user
msg['To'] = "destinatario@gmail.com"
msg['Subject'] = "Escribir asunto del correo"
message = "Escribir mensaje aquí"

# Añadimos el cuerpo del mensaje al objeto mensaje 4
msg.attach(MIMEText(message, 'plain'))

#Creamos servidor 5
server = smtplib.SMTP('smtp.gmail.com: 587')
server.starttls()

# Iniciamos sesión en el servidor de correo 6
server.login(msg['From'], password)

# Enviamos el mensaje a través del servidor de correo 7
server.sendmail(msg['From'], msg['To'], msg.as_string())

# Cerramos la conexión 8
server.quit()
    
```

Figura 20. Código envío email con SMTP.

En nuestro caso hemos decidido usar la versión sin archivos adjuntos ni html, pero añadir estas opciones es sencillo:

- Archivo adjunto: Se debe importar la librería MIMEImage (en el caso de que sea una imagen) y luego añadir el archivo adjunto al objeto msg, para ello se debe usar la función `attach(archivo_adjunto)`.

```

# Adjuntar imagen al correo
msg.attach(MIMEImage(file("google.jpg").read()))
    
```

Figura 21. Función adjuntar imagen al correo.

- HTML: Para ello, simplemente tenemos que poner el código HTML dentro del mensaje y usar la función `attach()` del paso 4 sustituyendo "plain" por "text/html".

12.6 PDOC

La librería `pdoc` de Python permite documentar código ya que proporciona tipos, funciones y una interfaz que se ejecuta sobre línea de comandos para acceder a la documentación pública de los módulos del proyecto y para presentarla en un formato amigable y estándar de la industria. Es el más adecuado para proyectos pequeños y medianos con Python. Una de las ventajas de `pdoc` es que nos permite expandir el código de una función o clase para ver el contenido sin tener que ir a los ficheros.

La documentación también se puede generar en formato HTML, que es el que se ha usado en este proyecto. Así que, se han generado dos ficheros html con la documentación de los archivos `actions.py` y `ga_connector.py`.

Para generar la documentación debemos usar el comando `pdoc --html actions.py --force` y `pdoc --html ga_connector.py --force`.

The screenshot shows the pdoc documentation for the `ActionInscripción` class. It includes the class definition, a description, a link to expand source code, a list of ancestors, and a list of methods. The `name` method is expanded to show its implementation, which returns the string `"action_enviar_email"`. The `run` method is also shown with its implementation, which sets variables for the message and SMTP port.

```
class ActionInscripción (*args, **kwargs)
    Class that defines the transactional action used for the user to sign up for a talk.
    ▶ EXPAND SOURCE CODE

Ancestors
    rasa_sdk.interfaces.Action

Methods
    def name(self)
        In this function the name that our action will have is defined. We must include it in the
        domain.yml file
        ▼ EXPAND SOURCE CODE
        def name(self) -> Text:
            """
            In this function the name that our action will have is defined. We must include it in the `do
            """
            return "action_enviar_email"

    def run(self, dispatcher, tracker, domain)
        Here the message is sent, to do that we need to set some variables:
        msg['From'] = Sender of the message
        msg['To'] = Receiver of the message
        msg['Subject'] = Subject of the message
        'smtp.gmail.com: 587' = SMTP port provided by Gmail.
        ▶ EXPAND SOURCE CODE
```

Figura 22. Ejemplo documentación clase con `pdoc`

12.7 OTRAS LIBRERÍAS

Tras comentar las principales librerías y frameworks usados en el Proyecto, vamos a hablar de otras que se usan pero no entraremos tanto en detalle para este Proyecto.

Keras

Keras se trata de un framework de Redes Neuronales de código abierto escrito en Python. La particularidad de este framework es que soporta múltiples motores computacionales a nivel de back-end.

En nuestro proyecto se usa una red neuronal implementada en Keras para seleccionar la siguiente acción a realizar por el bot. Por defecto viene configurado como una red neuronal del tipo LSTM, así que no hemos considerado oportuno modificar esta configuración.

Tensorflow

TensorFlow es una biblioteca de código abierto que se utiliza para tareas de aprendizaje automático. Internamente, las redes neuronales de TensorFlow, realizan operaciones sobre arrays de datos de más de una dimensión, llamados “tensores”.

En nuestro proyecto se ha usado TensorFlow para hacer el entrenamiento de nuestro asistente.

Rasa_sdk

Esta librería proporciona las herramientas que se necesitan para incluir acciones transaccionales a nuestro asistente.

Primero de todo, nos permite importar la clase *Action*, que se trata de la clase base de cualquier acción transaccional. Estas acciones tienen el método *name()* y *run()*.

También nos permite acceder a elementos como el Tracker, el Dispatcher y el dominio.

Requests

Mediante esta librería podemos hacer peticiones HTTP a través de código, la ventaja de usarlo es que facilita el formato de los parámetros de la petición, ya que se hace mediante elementos de la librería. En nuestro asistente, la hemos usado para realizar las peticiones GET para saber el clima actual en la ciudad del evento.

13. ELEMENTOS DEL BOT

13.1 DIAGRAMA DE ARQUITECTURA

En el siguiente diagrama, se muestra una vista física de la arquitectura de Rasa y en particular de nuestro asistente. En él, podemos ver los diferentes módulos de nuestro asistente (Canal, bot, Trainer y Actions) y dentro de cada módulo los componentes que lo integran y sus relaciones.

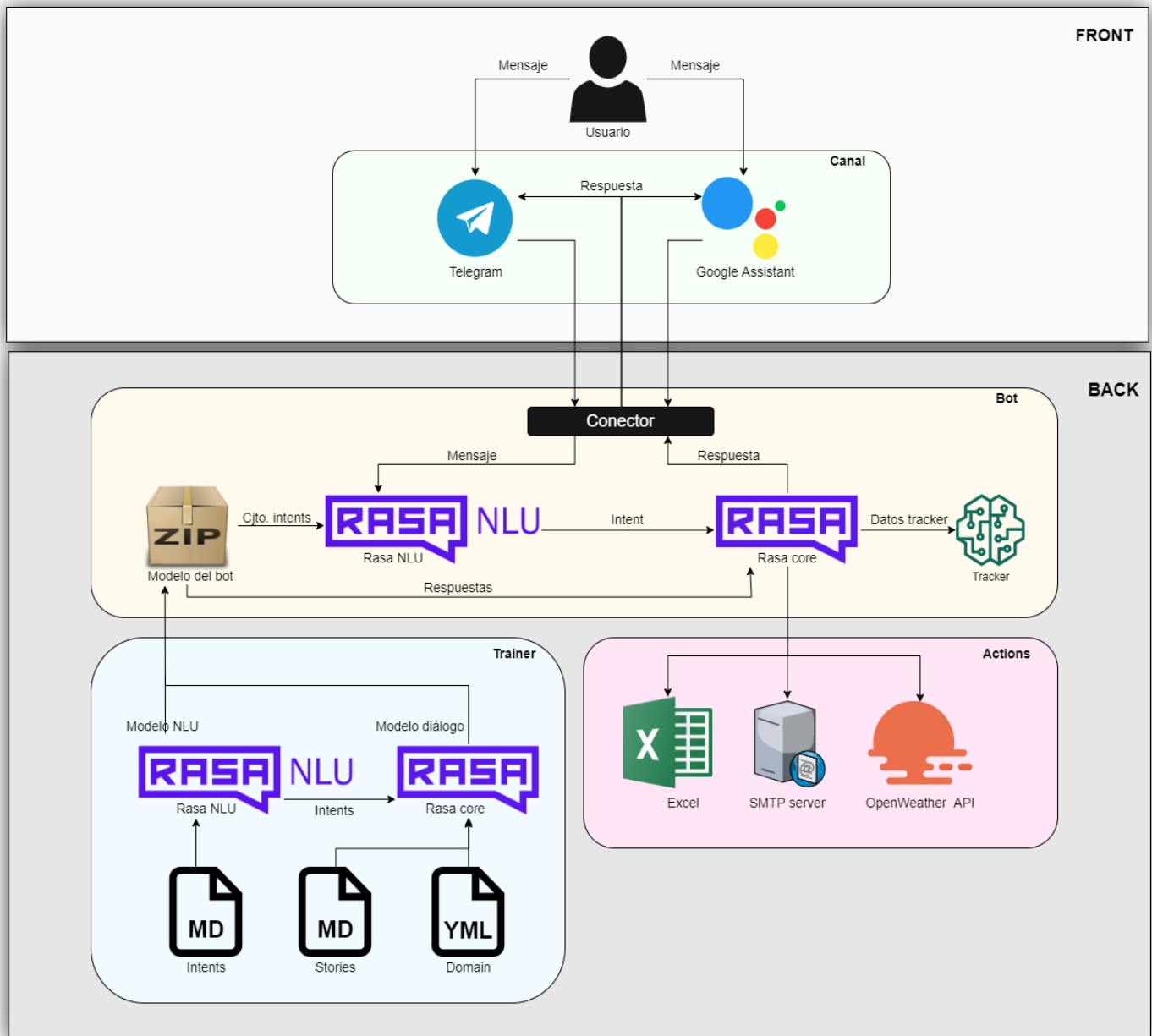


Figura 23. Diagrama de Arquitectura.

Tracker: Elemento encargado de mantener actualizado el estado del dialogo entre el usuario y el asistente, aquí se guardan por ejemplo los slots (variables) que se recogen de la conversación.

13.2 DIAGRAMA DE CLASES

En esta sección mostramos el diagrama con las clases que forman nuestro sistema. A continuación, procedemos a explicar por encima el sistema:

Este diagrama nos muestra que por una ubicación y varias fechas tenemos un evento. Una charla tiene una fecha, una sala y un evento asociado, además de una categoría.

La ubicación del evento (el edificio) contiene las salas de las charlas, si no está en esta ubicación no se puede crear la charla.

Los empleados son invitados o no al evento, en caso de serlo, pueden participar como Oyentes o como Ponentes a las charlas, aunque para una charla no pueden ser ambas cosas a la vez.

Todos los datos son *String*, menos la fecha y la hora, que son de tipo *Date* y el num_empleado y duración que son de tipo *Integer*.

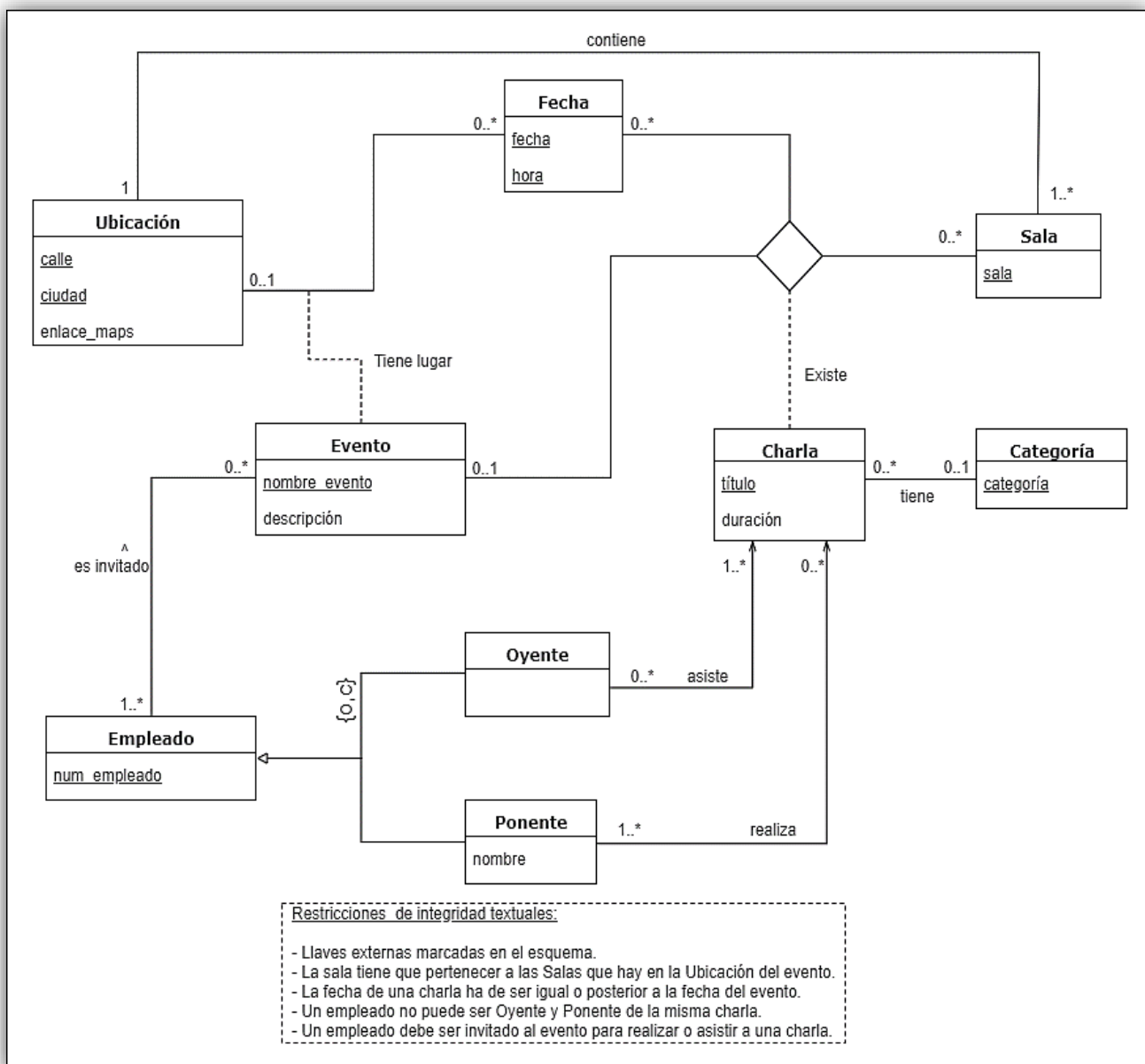


Figura 24. Diagrama de clases.

13.3 BASE DE DATOS

Para el tratamiento de nuestros datos, hemos valorado diferentes alternativas: usar una base de datos relacional, una no relacional, usar un fichero JSON con toda la información, etc. Pero hemos considerado que para el alcance de este proyecto nos basta con guardar nuestros datos en **varias tablas de un archivo de Excel**.

Nos hemos decantado por esta opción ya que Excel ofrece una interfaz muy intuitiva y sencilla. También nos permite realizar el reconocimiento automático de nuevas filas y columnas.

Como la finalidad del asistente es ir actualizando los datos para cada evento corporativo, hemos considerado oportuno que **pueda ser actualizado por una persona sin conocimientos sobre bases de datos o archivos JSON**.

Es muy importante y debemos tener claro que **Excel no se trata de un sistema de gestión de bases de datos (DBMS)**, ya que no tiene un servicio para controlar la manipulación de datos, sino que pueden ser modificados directamente en la aplicación por el usuario. En la Figura que se muestra a continuación, podemos verlo con más detalles.

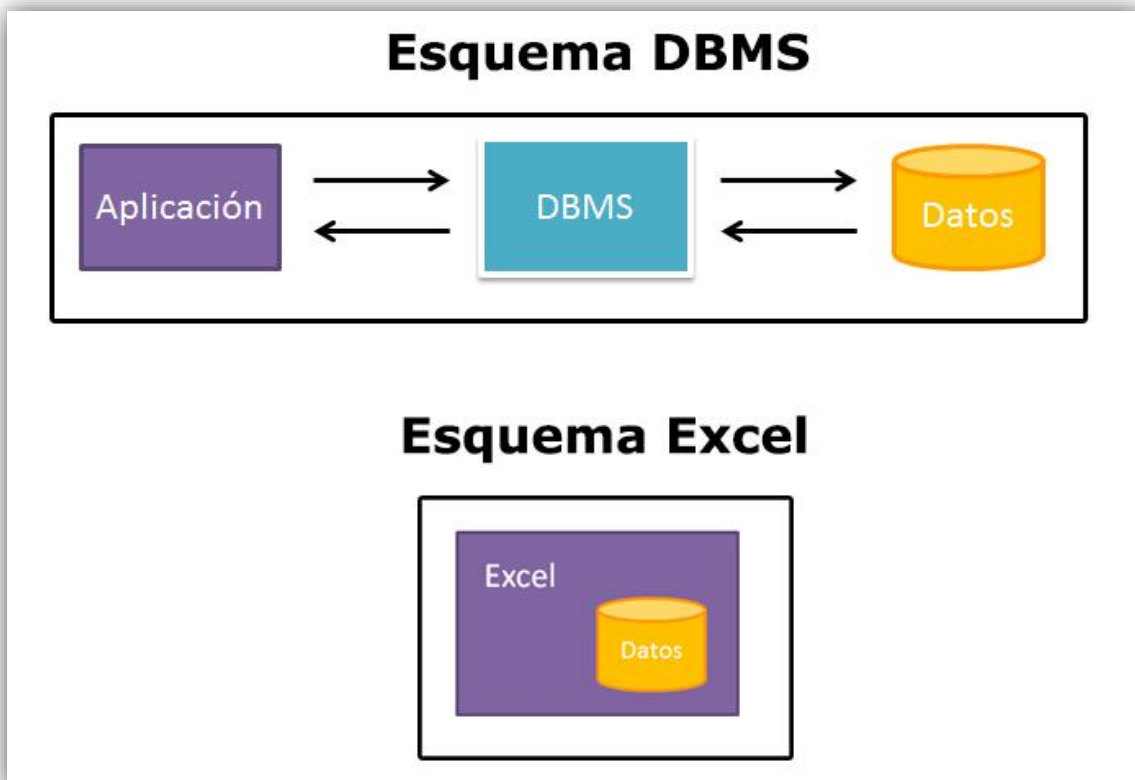


Figura 25. Diferencias DBMS y Excel. (Fuente: exceltotal.com)

Aunque para el proyecto nos sirve a la perfección, debemos tener en cuenta algunas desventajas que existen al usar este software para almacenar los datos:

- Solo un usuario puede acceder a modificar la información al mismo tiempo.
- La escalabilidad del programa no es muy buena, si tenemos mucha información empezará a ralentizarse.
- No es posible establecer un nivel de seguridad avanzado con el fin de proteger los datos almacenados.

En nuestro Excel hemos definido dos Hojas de trabajo, una llamada info_evento, con información del evento actual y otra llamada charlas, con información de las conferencias.

A continuación mostramos los campos de cada una de las hojas del Excel:

Campo	Detalles
Nombre_evento	Nombre del evento actual.
Fechas	Fechas en qué transcurre el evento.
Hora	Hora de inicio del evento.
Descripción	Detalles sobre la temática del evento o datos de interés.
Calle	Ubicación del evento, meramente informativo.
Ciudad	Ciudad donde se realiza el evento. De aquí se consulta el clima.
Enlace_maps	Enlace al mapa de Google con la ubicación.

Tabla 9. Detalles hoja "info_evento".

Campo	Detalles
Fecha	Fecha en que se realiza la charla, con formato DD/MM/AAAA.
Hora	Hora en que empieza la charla, en formato 24 horas.
Sala	Sala donde tiene lugar la charla.
Persona	Persona encargada de realizar la charla. Hay un filtro de charlas por persona.
Título	Título descriptivo de la charla.
Categoría	Categoría de la charla. Existe un filtro de charlas por categoría.

Tabla 10. Detalles hoja "charlas".

13.4 DATOS ENTRENAMIENTO DEL NLU

13.4.1 INTENTS, ENTITIES Y SLOTS

A través del componente NLU, nuestro asistente debe entender qué está diciendo el usuario, así que de cada mensaje que recibe el bot se extrae el intent asociado y, si existen, las entidades del mensaje.

Los intent son las intenciones que tiene el usuario al interactuar con nuestro asistente, para cada intención, un usuario puede expresar con diferentes palabras – llamadas utterances – que debemos asignar a ese intent. Por ejemplo, una persona para expresar el intent saludo puede hacerlo de muchas formas, por ejemplo *hola*, *buenas*, *qué tal* o *hey*, entre otras.

Por otra parte, hay mensajes que contienen entidades que es preciso extraer del mensaje, ya que nuestro asistente puede usar dichas entidades para personalizar la respuesta. Por ejemplo, si tenemos un intent asociado a consultar el tiempo en una ciudad, una de las utterance podría ser “*Cuál es el tiempo en {nombre_ciudad}*”, donde nombre_ciudad es una entity que servirá para buscar el tiempo de esa ciudad en una API externa.

En RASA, los intents y entidades se encuentran definidos dentro de una carpeta llamada data, dentro de esta carpeta encontramos un fichero en formato markdown llamado *nlu.md*. Allí es donde definiremos nuestros intents y entidades. En la figura que se muestra a continuación, podemos ver un ejemplo de cómo se define. Los elementos que hay en azul es el nombre de las entities.

```

nlu.md
1  ## intent:saludo
2  - Buenos días
3  - Buenas tardes
4  - Buenas noches
5  - Hola
6  - Qué tal?
7  - Empezar
8  - Qué pasa?
9  - Saludos
10
11 ## intent:ayudame
12 - ¿Como funcionas?
13 - ¿Como va esto?
14 - ¿Como funciona esto?
15 - ayuda
16 - ¿Me puedes ayudar?
17 - ¿Cuáles son tus funciones?
18
19 ## intent:programacion_personalizada
20 - ¿A qué charlas debería ir?
21 - a que charlas tengo que ir?
22 - Personaliza mi horario
23 - Dame un horario personalizado
24 - dime solo las charlas interesantes
25
26 ## intent:filtro_ponente
27 - Dame las charlas hechas por [Mickael](nombre_ponente)
28 - Charlas de [Lucas](nombre_ponente)
29 - Dame las charlas hechas por [Álvaro](nombre_ponente)
30 - Charlas donde el ponente es [María](nombre_ponente)
31 - Charlas en las que habla [Javi](nombre_ponente)
32

```

Figura 26. Ejemplo intents y entities RASA.

Por otra parte, también encontramos unos elementos llamados *slots*. El concepto es parecido al de las *entities*, pero a diferencia de ellas, los slots son la memoria de nuestro asistente y se almacenan como variables clave – valor. Un ejemplo de uso de estos elementos podría ser para almacenar el nombre del usuario que está hablando con el bot. Así se puede recuperar posteriormente y se pueden personalizar los mensajes.

Dentro del fichero *nlu.md*, los *slots* se definen de la misma forma que las *entities*, pero es en el fichero de dominio del asistente donde se especifica que son *slots*.

13.4.2 SINÓNIMOS Y TABLAS DE BÚSQUEDA

Otra de las opciones para trabajar con los intents es el uso de sinónimos y tablas de búsqueda, mediante estos elementos podemos ayudar a que nuestro asistente entienda al usuario.

Sinónimos: Podemos definir que dos entidades tienen el mismo valor, así que van a ser tratadas como sinónimos. Estos elementos suelen ser usados para tratar abreviaturas o cuando se usará para la entrada de una *Action*.

Existen dos formas de definir los sinónimos, desde el propio intent o como un elemento diferente. Veamos un ejemplo, para referirnos a los Estados Unidos, se puede hacer de diversas formas, así que podemos definir cada una de ellas con el uso de sinónimos y evitar que nuestro asistente se confunda.

```
##intent:buscar
- Calcula la distancia del evento con [EEUU](pais_estados:USA) <!-- sinónimos, método 1-->

##synonym: EEUU
- Estados Unidos <!-- sinónimos, método 2-->
- United States
- Estados Unidos de América
- EUA
- US
```

Figura 27. Ejemplo de uso de sinónimos.

Tablas de búsqueda: También llamadas lookup tables, nos sirven para reducir el número de ejemplos que debemos aportar al asistente, ya que nos permiten definir palabras que pertenezcan al mismo grupo aunque tengan significados diferentes. Estas tablas se pueden importar desde fichero o directamente como un elemento propio, veamos un ejemplo con los platos del catering.

```
##intent:comida_consulta
- En el catering se incluye [verdura](platos_comida)?
- Para comer tenemos [pan](platos_acompañar)?
-

##lookup: platos_comida
- sushi
- pescado
- comida vegana-
- pizza
- pasta
- pollo tikka masala

##lookup: platos_acompañar
| data/comida/acompanar.txt <!-- Se importa desde fichero -->
```

Figura 28. Ejemplo tabla de búsqueda.

13.4.3 EXPRESIONES REGULARES

Las expresiones regulares sirven a nuestro asistente para comprender entidades del mensaje que siguen algún patrón. Se definen mediante el uso de la etiqueta *regex*, y tienen un formato estandarizado. Muchas veces se usa este tipo de elementos para expresiones como códigos postales, fechas o valores numéricos.

En nuestro proyecto las usamos para capturar la fecha con separadores “/” o el número de empleado, que no tiene límite de dígitos numéricos.

```
##intent:filtro_fecha
- Charlas del día [13/12/1995](formato_fecha)
- Solo las que se hacen el [14/06/2019](formato_fecha)

## intent:contest_empleado
- mi número de empleado es [1](numero_employado)
- mi identificación en la empresa es [8](numero_employado)

## regex:formato_fecha
- ^[0-9]*/[0-9]*/[0-9]*

## regex:numero_employado
- ^[0-9]*
```

Figura 29. Ejemplo expresiones regulares.

13.5 HISTORIAS ENTRENAMIENTO DEL NLU

Las historias de Rasa forman parte de los datos de entrenamiento utilizados para entrenar los modelos de gestión de diálogo de Rasa.

Una historia es una representación de una conversación entre un usuario y un asistente, convertida a un formato específico donde las entradas del usuario se expresan como los intents correspondientes (y entidades cuando es necesario) mientras que las respuestas de un asistente se expresan como nombres de acción o respuesta correspondientes.

Las acciones del usuario se definen con un “*” delante del nombre del intent, mientras que las del bot se hace con “-” delante de la respuesta. Aquí es donde se puede definir comportamiento específico para ciertas entidades. Siempre es el usuario el que inicia el flujo.

```
## obteniendo programacion personalizada 2
* programacion_personalizada
- utter_pregnumempleado
* contest_employado{"numero_employado":"2"}
- action_planificacion_personalizada

## obteniendo programacion personalizada general
* programacion_personalizada
- utter_pregnumempleado
* contest_employado
- action_planificacion_personalizada

## consultando siguiente charla
* siguiente_charla
- action_siguiente_charla

## obteniendo clima
* dame_clima
- action_clima

## inscribirse a una charla y email
* inscribirse_charla
- action_enviar_email
```

Figura 30. Historias de entrenamiento.

13.6 ACTIONS

Las acciones son funciones escritas en Python en el fichero *actions.py* que sirven para que el bot sea capaz de tener un comportamiento personalizado. Al ser código en Python, permite muchas funcionalidades y es aquí donde se debe desarrollar la lógica de mensajes del bot (más allá de las plantillas para devolver un mensaje estático).

Cada action se define de la misma forma:

- 1) Se crea una clase del tipo (Action).
- 2) Se define una función *name(self)* -> *Text*: y esta función retorna el nombre del action que se ha creado.
- 3) Se define la función *run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any])* -> *List[Dict[Text, Any]]*: Aquí es donde irá la lógica de nuestra función.
- 4) Se envía el mensaje a través del dispatcher, con la función *utter_message("mensaje")*. Este mensaje es lo que devolverá el chatbot al usuario.
- 5) Se retorna un conjunto vacío. Aquí se podría poner las variables que se quisieran mantener.

Un ejemplo básico del action para obtener un número aleatorio:

```
class ActionRandomNumber(Action):
    def name(self) -> Text:
        return "action_random_number"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        seed(randint(0,1000))
        randnum = randint(0,1000)
        dispatcher.utter_message("Claro!! Tu número aleatorio es: {}".format(randnum))
        return []
```

Figura 31. Creación de un action.

Listado de actions implementadas:

- **action_saludo** -> Al iniciarse el asistente, saluda ofreciendo información del Excel del evento.
- **action_random_number** -> Devuelve un número aleatorio.
- **action_todas_charlas** -> devuelve información de todas las charlas del evento, para cada charla se muestra: Fecha, hora, sala, título, ponente y categoría.
- **action_planificacion_personalizada** -> Personaliza las charlas según los gustos, en función del número de empleado sólo se muestran las charlas de una categoría concreta.
- **action_filtro_ponente** -> Se filtran las charlas por el nombre del ponente.
- **action_filtro_categoria** -> Se filtran las charlas por el nombre de categoría.
- **action_filtro_fecha** -> Se filtran las charlas por la fecha de realización.
- **action_siguiete_charla** -> Consulta la siguiente charla a partir de la hora actual.
- **action_clima** -> Consulta el clima en la ubicación del evento (Haciendo uso de OpenWeatherAPI).
- **action_enviar_email** -> Envía un correo electrónico de confirmación haciendo uso del protocolo SMTP al inscribirse a una charla.

13.7 DOMINIO DEL ASISTENTE

El dominio define el universo en el que opera el asistente. En el fichero *domain.yml* se especifican *los intents, entities, slots y actions* que el chatbot contiene. Opcionalmente, y en nuestro caso se ha usado, también puede incluir plantillas para las respuestas del bot.

Primero de todo, listamos los intents que deben coincidir con los del fichero *nlu.md*:

```
domain.yml
1  intents:
2    - saludo
3    - bot_dudas
4    - despedida
5    - aleatorio
6    - charlas
7    - programacion_personalizada
8    - filtro_ponente
9    - filtro_categoria
10   ...
```

En segundo lugar, las entities y slots que usaremos en esos intents. Los slots deben contener la categoría (normalmente usaremos *unfeaturized* o *text*):

```
12  entities:
13    - numero_empleado
14    - nombre_ponente
15    - nombre_categoria
16    - formato_fecha
17
18  slots:
19    formato_fecha:
20      type: unfeaturized
21    nombre_categoria:
22      type: unfeaturized
23    nombre_ponente:
24      type: unfeaturized
25    numero_empleado:
26      type: unfeaturized
27
```

A continuación, definimos las plantillas de respuesta del bot (este elemento es opcional, ya que se puede responder todo desde las actions personalizadas). Pueden tener más de una respuesta y en tal caso se usará una al azar.

```
28  templates:
29    utter_dudas:
30      - text: Soy un bot, hecho por un becario de everis usando Rasa Framework.
31      - text: Soy un duende encerrado en el ordandor... ¡Ayúdame a saliiiiir! jejeje era broma, soy un bot.
32    utter_ayuda:
33      - text: Me puedes preguntar lo que quieras sobre el evento, te responderé encantado!
34    utter_pregnumpleado:
35      - text: ¿Cuál es tu número de empleado?
36      - text: ¿Me puedes decir tu número de empleado porfavor?
37    utter_despedida:
38      - text: ¡Hasta luego!
39      - text: ¡Nos vemos pronto!
40    ...
```

Por último, listamos las actions y también las plantillas que usa nuestro bot.

```
42  actions:
43    - utter_dudas
44    - utter_ayuda
45    - action_siguiete_charla
46    - action_clima
47    - action_todas_charlas
48    - action_enviar_email
49    ...
```

13.8 CONECTORES

Los conectores permiten que nuestro asistente se comunique con los canales de entrada y salida de información. Rasa dispone de 9 conectores predefinidos, además permite la creación de conectores personalizados, como hemos tenido que hacer en nuestro caso con el conector del canal de Google Assistant, que entraremos en detalles más adelante.

Los conectores que trae por defecto Rasa son:

- Páginas web
- Facebook Messenger
- Slack
- Telegram
- Twilio
- Microsoft Bot Framework
- Cisco Webex Teams
- RocketChat (parecido a Slack)
- Mattermost (parecido a Slack)
- Conectores personalizados

Para cada conector que se quiera usar, hay que añadir las credenciales oportunas en el fichero *credentials.yml*. Hay que tener en cuenta que no siempre serán los mismos campos de credenciales, ya que cada canal puede tener diferente número de tokens de acceso.

En nuestro caso hemos usado Telegram, así que en el fichero de las credenciales hemos tenido que añadir las credenciales que nos ha proporcionado el Bot Father en el momento de crear el bot:

```
telegram:  
  access_token: "1037757676:AAHFitlMkfeRL4zhdutGciOznAVc7_oIqKc"  
  verify: "eventis_bot"  
  webhook_url: "https://b6cba815.ngrok.io/webhooks/telegram/webhook"
```

Figura 32. Credenciales conector Telegram

Para que la url del webhook sea accesible desde fuera (ya que el servidor está funcionando en nuestro localhost³¹) debemos hacer público dicho localhost mediante una herramienta como ngrok. En el despliegue del asistente entraremos en más detalle, pero la primera parte del webhook del conector será el túnel hacia el puerto 5005 de nuestro localhost.

³¹ Nombre reservado para el ordenador o dispositivo que se está usando. Se puede usar para acceder a recursos localmente como si estuviesen colgados en internet.

14. ANÁLISIS FUNCIONAL DEL BOT

14.1 PROCESADO DE UN MENSAJE

En el diagrama que se muestra a continuación, podemos observar cómo se realiza el procesamiento de un mensaje por parte del NLU y posteriormente como se obtiene la respuesta en el Core.

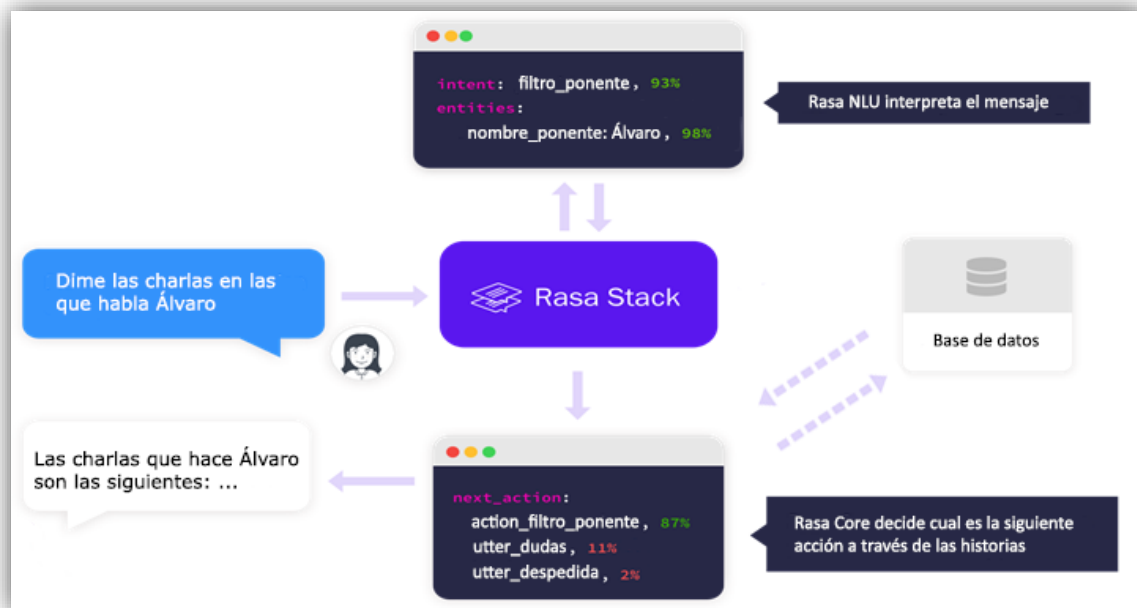


Figura 33. Proceso de clasificación de un mensaje.

1. En primer lugar, se envía a Rasa Stack (Rasa Core + Rasa NLU) la conversación del usuario, en este caso “Dime las charlas en las que habla Álvaro”. Este stack hace la función de Broker u Orquestador, y se encarga de gestionar las comunicaciones entre las capas.
2. El componente NLU del asistente, descifra el intent del usuario y extrae las características del mensaje, es decir, las entidades y slots asociados. Esta parte tiene unos % de “entendimiento”, -también llamado grado de confianza de la predicción- que si son inferiores al 40% (en nuestro caso, ya que este parámetro es ajustable), se mostrará un mensaje conforme el asistente no ha entendido al usuario. Estos mensajes se han definido en la template *utter_default* del archivo *domain.yml*.
3. Por otra parte, el componente Rasa Core, analiza qué debe hacer tras recibir ese intent, en función de las historias que se han definido en su entrenamiento. Ofrece diferentes opciones pero usará la que tenga un mayor % de coincidencia. En esta parte es donde se ejecuta la acción, así que pueda que se consulten recursos externos, en este ejemplo, la base de datos de charlas.

4. Por último, el asistente devuelve la respuesta al usuario por el mismo canal que la entrada.

14.1.1 DIAGRAMA DE SECUENCIA DEL ASISTENTE

Para modelar esta interacción entre el usuario y el sistema durante el momento comprendido entre que el usuario envía el mensaje hasta que recibe respuesta por parte del asistente, hemos considerado oportuno mostrarlo a través de un diagrama de secuencia. Debido a que se trata de un diagrama demasiado extenso para ponerlo todo junto, lo hemos dividido en varias partes:

- 1) Diagrama con la parte de interacción del usuario y detalles de la parte de NLU.
- 2) Diagrama con la parte del CORE (haciendo énfasis en el Tracker).
- 3) Proceso de creación del mensaje.

Interacción usuario + NLU

En esta primera parte, observamos cómo el usuario al introducir un mensaje se envía al bróker, que es el encargado de gestionar las comunicaciones. Este bróker envía la información al NLU, encargado de procesar el lenguaje natural. El NLU, haciendo uso del Interpreter, obtiene el intent asociado al texto y las entidades o slots del mensaje (Si el intent los requiere).

Posteriormente, esta información vuelve al Broker que la distribuye nuevamente hacia el CORE.

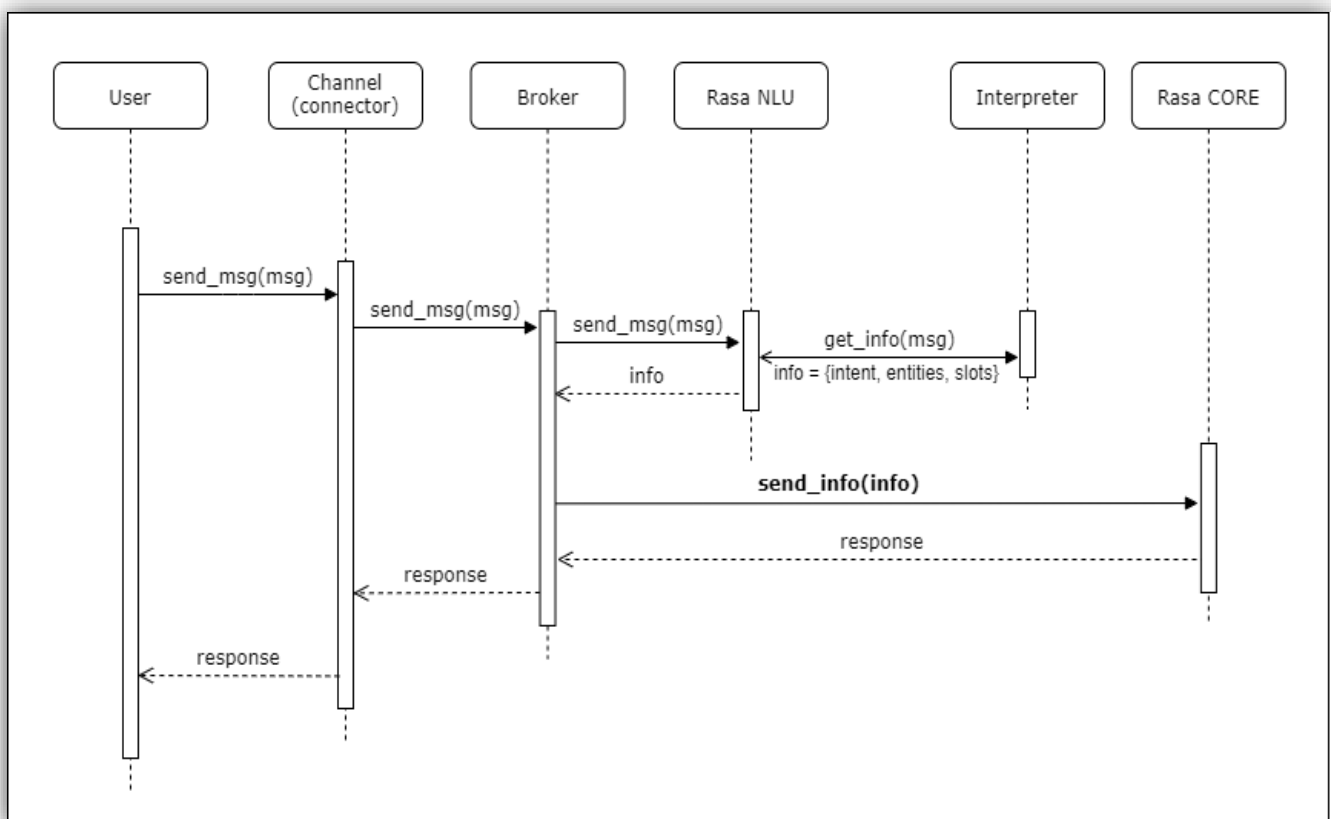


Figura 34. Diagrama secuencia Usuario y NLU

Rasa CORE (continuación función send_info(info))

En este segundo bloque desarrollamos la parte del diagrama que comprende el **RASA Core** + el **tracker** (elemento que mantiene consistente el estado de la conversación).

El Core envía la información al **tracker**, que almacena esta información conforme se ha recibido un mensaje. A continuación, envía la información al elemento **Policy**, encargado de decidir qué acción debe tomar el bot (llamada *next*). Esta elección se realiza a partir de las historias de entrenamiento.

Acto seguido, el **tracker** guarda la información relativa a la decisión tomada y se ejecuta la acción (*función do_action(info)*)

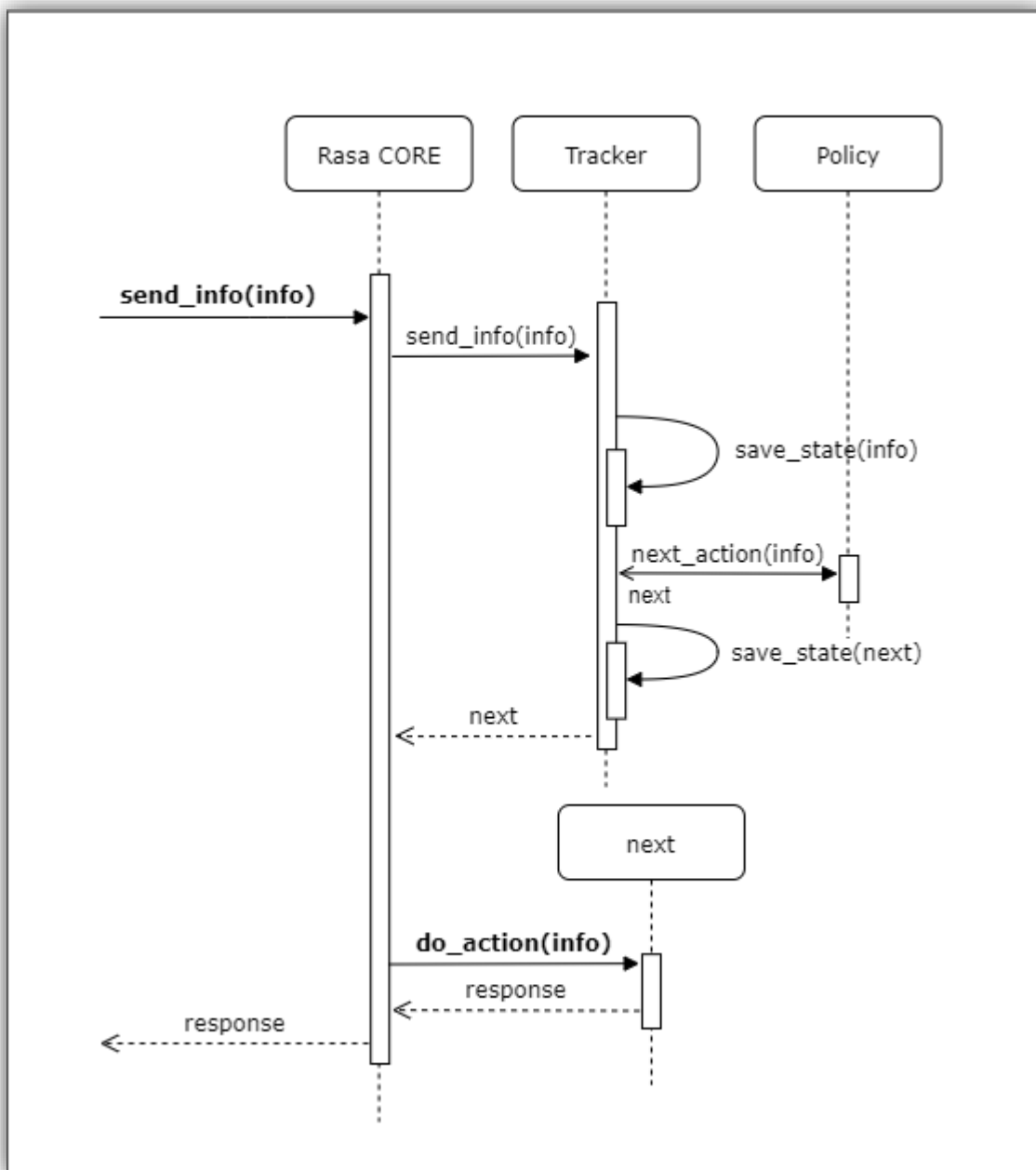


Figura 35. Diagrama de secuencia CORE.

Creación del mensaje

En esta parte, se desarrolla la función `do_action(info)`, invocada por el **CORE**. Esta función es la encargada de llevar a cabo la acción de ese intent y retornar el mensaje de respuesta.

El comportamiento de esta función puede variar según el tipo de acción que se produzca:

- **Action:** En este caso la acción se trata de una acción transaccional, definida en el fichero `actions.py`. Esta acción hará uso de algún recurso externo, generando la respuesta en función de ese recurso.
- **Template:** En este caso, se genera la respuesta a través de la plantilla que tiene asociado ese intent. Esta plantilla se encuentra definida en el fichero `domain.yml`.

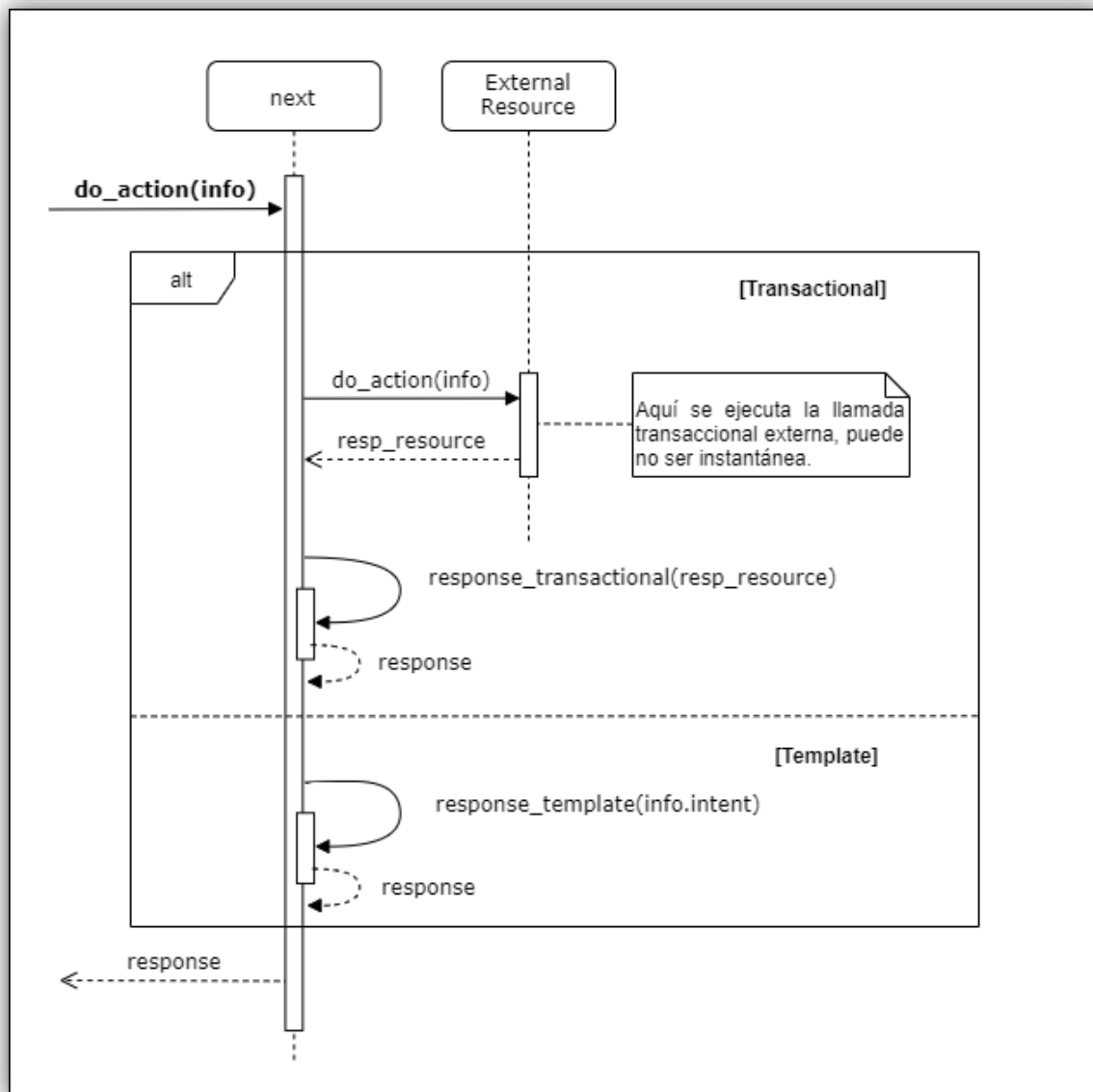


Figura 36. Diagrama de secuencia creación de respuesta.

14.2 DIAGRAMA DE CASOS DE USO

El siguiente diagrama que se ha hecho es el diagrama de casos de uso, mediante este diagrama se pretende especificar el comportamiento del sistema mediante la interacción con los usuarios y con otras partes de dicho sistema. En nuestro caso hemos detectado dos actores implicados en el sistema, el usuario y el administrador del evento.

El primero de ellos, el usuario, lleva a cabo una serie de funcionalidades (se muestran en la Figura que hay a continuación). Y a su vez, dos de estas funcionalidades incluyen otra más, que es la de introducir número de empleado.

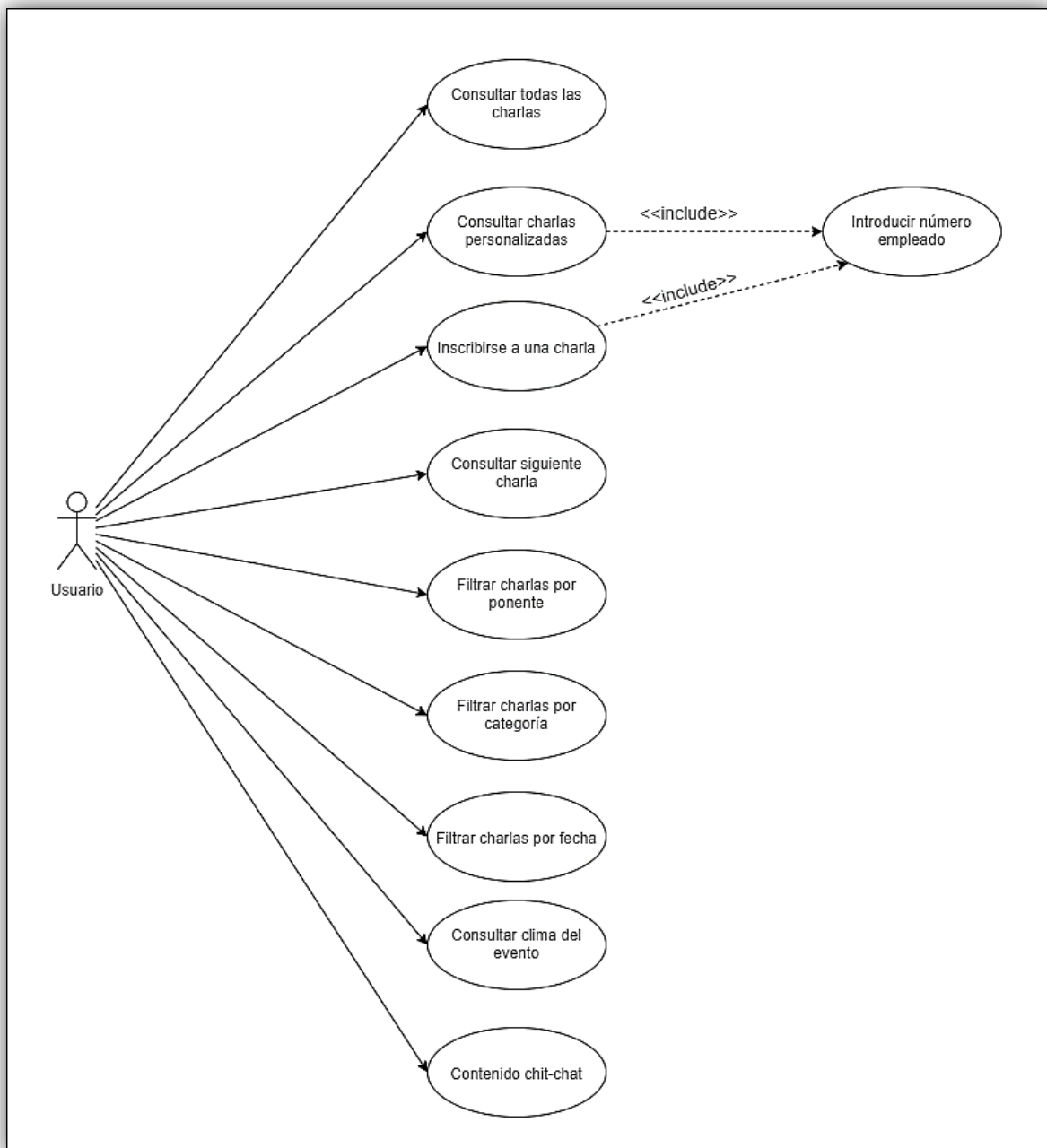


Figura 37. Diagrama de casos de uso del usuario.

El otro actor implicado es el Administrador del evento. Este actor solo hace uso de dos funcionalidades del asistente, más destinadas a la gestión. Estas funcionalidades son la de introducir la información del evento (descripción, lugar, fecha, enlace mapas) y la de introducir todas las charlas que se llevan a cabo en dicho evento con sus detalles. Debemos recordar que el administrador del evento no tiene por qué tener un perfil técnico, así que esta introducción de la información del evento y las charlas se hará a través de un libro de Excel.

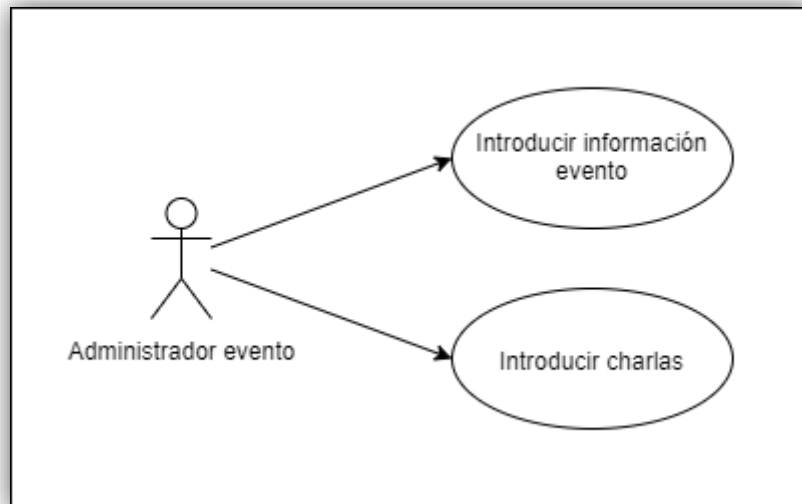


Figura 38. Diagrama de casos de uso del administrador.

14.3 DIAGRAMA DE ESTADOS

El diagrama de estados nos mostrará los estados por los que pasa la interacción usuario-sistema. En amarillo encontramos los estados que corresponden al asistente y en verde los del usuario.

Primero de todo, el asistente saluda y permanece a la espera de la acción del usuario. Una vez se ha llevado a cabo la acción, el bot la ejecutará si no es la acción de salir, y volverá a permanecer a la espera. En el caso de que la acción sea la de salir procederá a despedirse y finalizar la conversación.

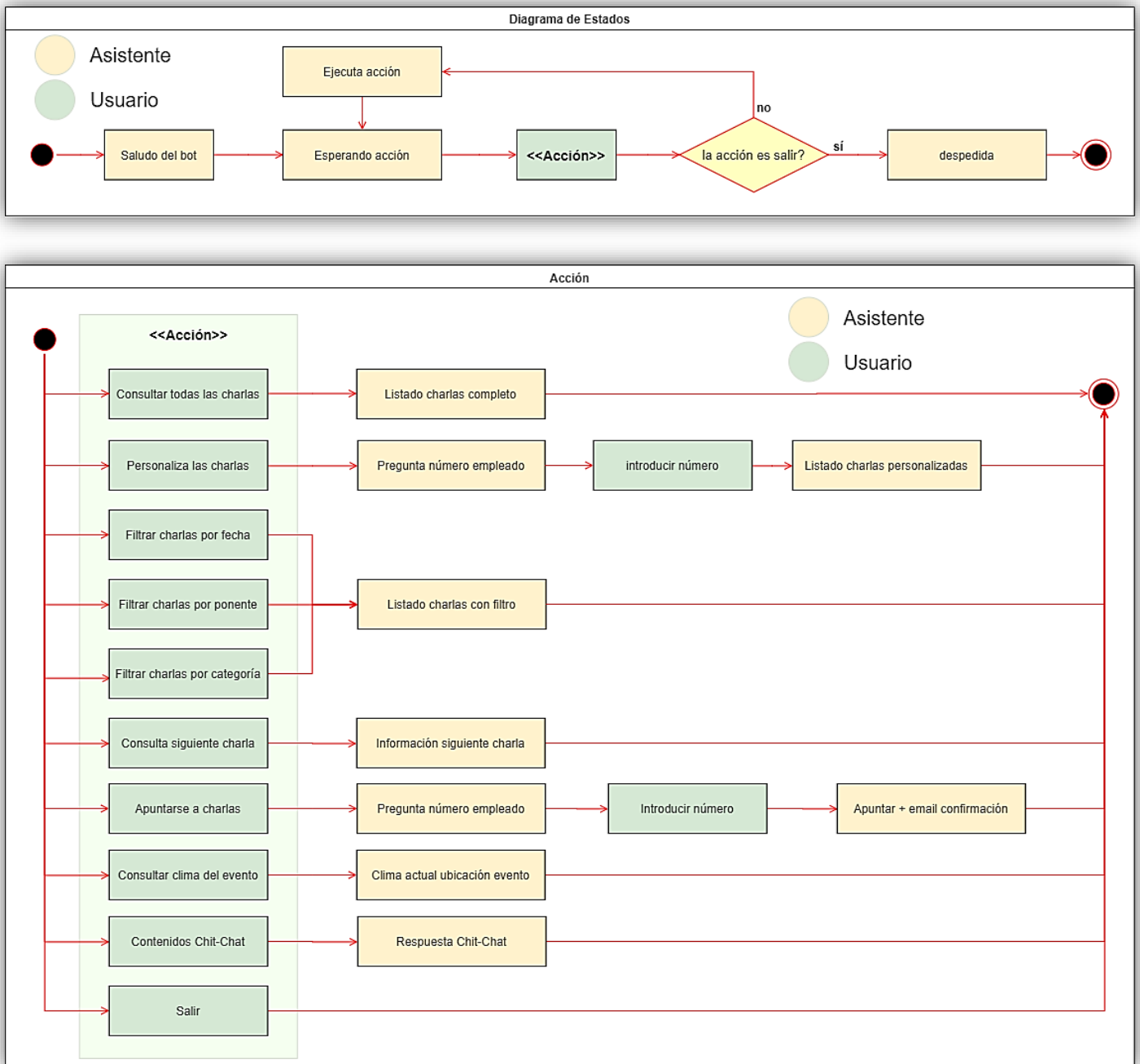


Figura 39. Diagrama de estados.

14.4 CONEXIÓN CON GOOGLE ASSISTANT

Otra parte a destacar del proyecto es la conexión con Google Assistant, ya que, como hemos comentado en la sección de conectores, Rasa incluye diversos conectores predefinidos de las aplicaciones de mensajería más populares, pero Google Assistant no es uno de ellos, por lo tanto, lo tendremos que crear nosotros.

14.4.1 CONFIGURACIÓN DE LAS ACTIONS DE GOOGLE ASSISTANT

Primero de todo hay que crear una custom action a través de la interfaz de Google Assistant, de la categoría *Actions SDK*. Debido a que Rasa va a procesar por su cuenta el entendimiento del lenguaje natural y la gestión del diálogo, google actions solo será responsable de dos cosas: Entender que el usuario ha invocado la acción de abrir nuestro asistente y dejar pasar todos los inputs que haga el usuario y las respuestas del asistente en el otro sentido.

Tras iniciar nuestra action haciendo uso de *gactions init*, procedemos a configurar el archivo *actions.json* que se ha generado en el directorio del proyecto. En este fichero debemos configurar las dos actions, la primera contendrá el trigger para empezar la conexión y la otra capturará el texto del usuario. En segundo lugar, debemos indicar la url del webhook que se usará para establecer la conexión entre Google Assistant y Rasa.

```
1 {
2   "actions": [
3     {
4       "description": "Default Welcome Intent",
5       "name": "MAIN",
6       "fulfillment": {
7         "conversationName": "welcome"
8       },
9       "intent": {
10        "name": "actions.intent.MAIN",
11        "trigger": {
12          "queryPatterns":["hablar con eventos everis"]
13        }
14      }
15    },
16    {
17      "description": "Rasa Intent",
18      "name": "TEXT",
19      "fulfillment": {
20        "conversationName": "rasa_intent"
21      },
22      "intent": {
23        "name": "actions.intent.TEXT",
24        "trigger": {
25          "queryPatterns":[]
26        }
27      }
28    }
29  ],
30  "conversations": {
31    "welcome": {
32      "name": "welcome",
33      "url": "https://67b6e794.ngrok.io/webhooks/google_assistant/webhook",
34      "fulfillmentApiVersion": 2
35    },
36    "rasa_intent": {
37      "name": "rasa_intent",
38      "url": "https://67b6e794.ngrok.io/webhooks/google_assistant/webhook",
39      "fulfillmentApiVersion": 2
40    }
41  },
42  "locale": "es"
43 }
```

Figura 40. Contenido fichero *actions.json*

14.4.2 CREACIÓN DEL CONECTOR DE RASA

Ahora que hemos creado las acciones que inician la conexión y nos envían la información del usuario, debemos generar el código necesario para establecer esa conexión y recibir esos mensajes desde Rasa.

- 1) Dentro del directorio del proyecto debemos crear un fichero llamado *'ga_connector.py'*, aquí es donde escribiremos el código.
- 2) Empezaremos el código importando las librerías necesarias, en este caso son:
 - a. Logging (para debugar y encontrar posibles errores).
 - b. Sanic (para crear el webhook de conexión).
 - c. Librerías de rasa core para los canales.
 - d. Json para tratar los datos.
- 3) Luego, creamos una clase llamada `GoogleAssistant` que hereda la clase `InputChannel` de rasa core, esta clase tiene dos funciones: *name* y *blueprint*. La clase **name**, definirá el prefijo del webhook, en nuestro caso `google_assistant`.
- 4) Ahora, vamos a rellenar la otra función, la *blueprint*. Esta función se encarga de definir el webhook de Google Assistant, además de obtener las respuestas del bot y enviarlas de vuelta a Google Assistant.
- 5) Ahora, debemos definir las rutas del webhook, aquí incluiremos dos rutas:
 - a. Primero definimos la *health route*, que contiene peticiones **GET a la ruta raíz**, devolviendo el mensaje "200 OK" confirmando que la conexión funciona correctamente.
 - b. En segundo lugar, definimos la *receive route*, que recibirá peticiones **POST a la ruta /webhook**, una vez que se recibe la solicitud POST, el conector toma la carga útil³² de la solicitud. Esta carga contiene los detalles sobre la entrada del usuario enviada por Google Assistant. Estos detalles incluyen la identificación del usuario, el action (los que se han definido en la sección anterior) y el mensaje en formato de texto. Estos detalles se asignan a las variables correspondientes.
 - c. Si la action es una entrada de usuario normal, entonces debemos inicializar el canal de salida –`CollectingOutputChannel` en este caso– y pasarlo a Rasa junto con la entrada del usuario y su identificación.
 - d. Rasa Core hará una predicción sobre cómo debe responder el asistente y finalmente, el mensaje de respuesta se incorpora a un json que se envía de nuevo a Google Assistant.

³² Traducción al español del término "payload".

```

1  import logging
2  import json
3  from sanic import Blueprint, response
4  from sanic.request import Request
5  from typing import Text, Optional, List, Dict, Any
6
7  from rasa.core.channels.channel import UserMessage, OutputChannel
8  from rasa.core.channels.channel import InputChannel
9  from rasa.core.channels.channel import CollectingOutputChannel
10
11 logger = logging.getLogger(__name__)
12
13
14
15 class GoogleConnector(InputChannel):
16
17     @classmethod
18     def name(cls):
19         return "google_assistant"
20
21
22     def blueprint(self, on_new_message):
23
24         google_webhook = Blueprint('google_webhook', __name__)
25
26         @google_webhook.route("/", methods=['GET'])
27         async def health(request):
28             return response.json({"status": "ok"})
29
30         @google_webhook.route("/webhook", methods=['POST'])
31         async def receive(request):
32             payload = request.json
33             intent = payload['inputs'][0]['intent']
34             text = payload['inputs'][0]['rawInputs'][0]['query']
35
36             if intent == 'actions.intent.MAIN':
37                 message = "Bienvenido a eventis. Puedes comenzar saludando. :)"
38             else:
39                 out = CollectingOutputChannel()
40                 await on_new_message(UserMessage(text, out))
41                 responses = [m["text"] for m in out.messages]
42                 message = responses[0]
43
44             r = {
45                 "expectUserResponse": 'true',
46                 "expectedInputs": [
47                     {
48                         "possibleIntents": [
49                             {
50                                 "intent": "actions.intent.TEXT"
51                             }
52                         ],
53                         "inputPrompt": {
54                             "richInitialPrompt": {
55                                 "items": [
56                                     {
57                                         "simpleResponse": {
58                                             "textToSpeech": message,
59                                             "displayText": message
60                                         }
61                                     ]
62                                 }
63                             }
64                         ]
65                     }
66                 ]
67             }
68
69             return response.json(r)
70
71         return google_webhook
    
```

Figura 41. Código conector Google Assistant

15. PRUEBAS

15.1 PRUEBAS DE ESTRÉS

Hemos realizado pruebas para saber cómo reacciona nuestro asistente en función del número de usuarios que lo utilicen al mismo tiempo. Para ello, hemos usado una extensión de Google Chrome llamada RESTful Stress, que nos permite realizar llamadas HTTP (GET, POST, PUT y DELETE) a nuestro endpoint de la API. Los resultados de estas pruebas son producto de las capacidades de nuestra máquina, ya que se ejecuta en local, así que si destinamos una máquina más potente los resultados mejorarán.

Primero de todo hemos configurado el test que vamos a realizar, aquí hemos indicado que será una llamada POST a la URL `http://localhost:5005/webhooks/rest/webhook`. Esta URL recibe un usuario y un texto, y hace todo el proceso de procesamiento del mensaje, devolviendo la respuesta del bot. Hemos configurado el text con 6500 iteraciones y en el cuerpo del mensaje le hemos enviado el texto `"dame el clima del evento"`, por lo tanto, haremos el test de la acción transaccional que consulta la API y devuelve el clima del evento. Para poder realizar el test, debemos desplegar el servidor activando CORS, con el comando `rasa run -m models --enable-api --cors "*" --debug`.

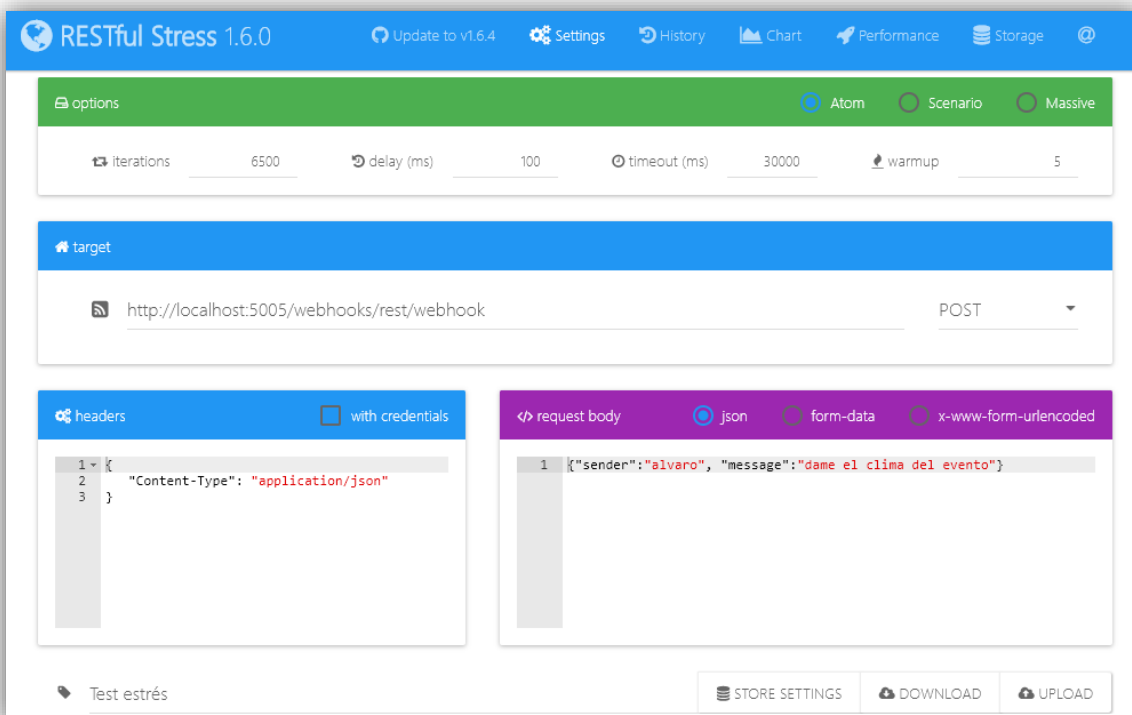


Figura 42. Configuración del test de estrés.

Aunque en el test no se muestran las repuestas, sino que simplemente el rendimiento, mostramos a continuación un ejemplo de la respuesta que obtendríamos.

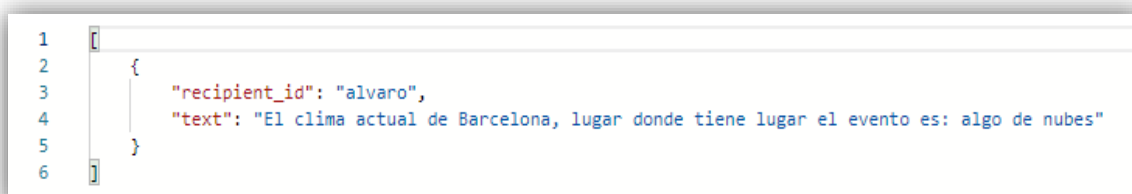


Figura 43. Respuesta llamada test estrés.

En cuanto a resultados del tiempo de respuesta, obtenemos que para 1650 iteraciones, de media se ha tardado 2861 ms en obtener la respuesta tras hacer la llamada, por otra parte, la llamada que más tiempo ha durado han sido 30000 ms y la que menos 841 ms.

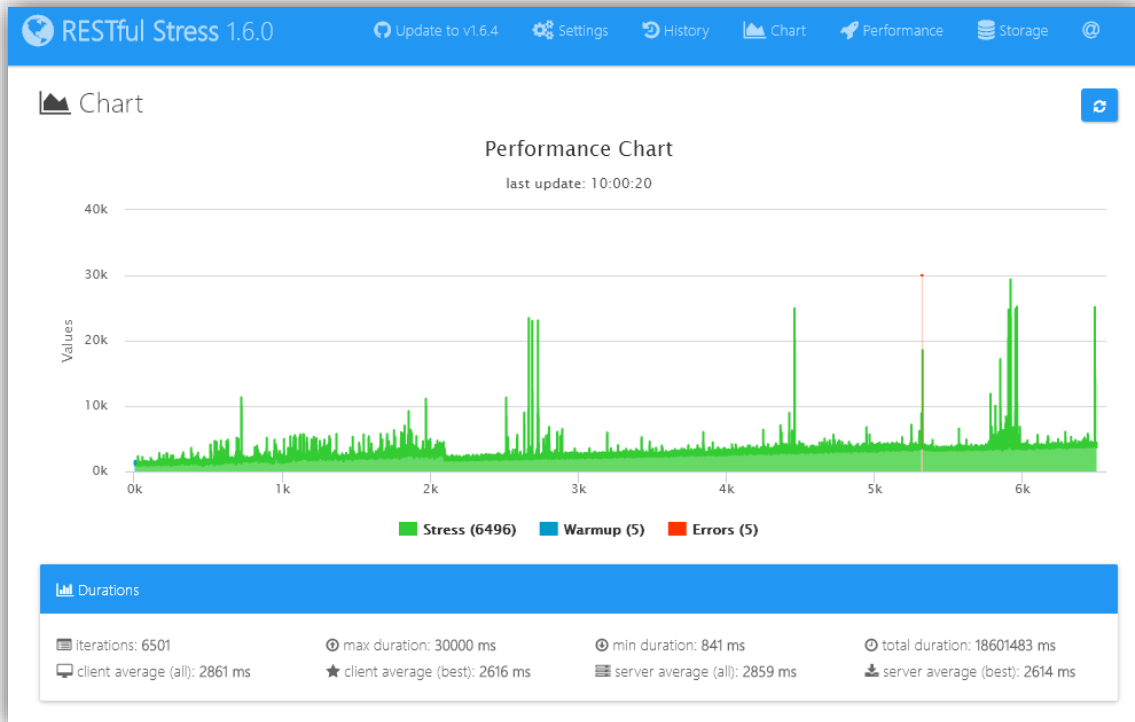


Figura 44. Tiempo llamadas test estrés.

Por lo que hace a los resultados que tienen que ver con los usuarios que pueden usar nuestro chatbot a la vez, nos encontramos que nuestro asistente puede ser usado por 3 usuarios concurrentes, en la gran mayoría de su tiempo. Aunque, aplicando paralelismo, se podría subir a los 4 usuarios a la vez.

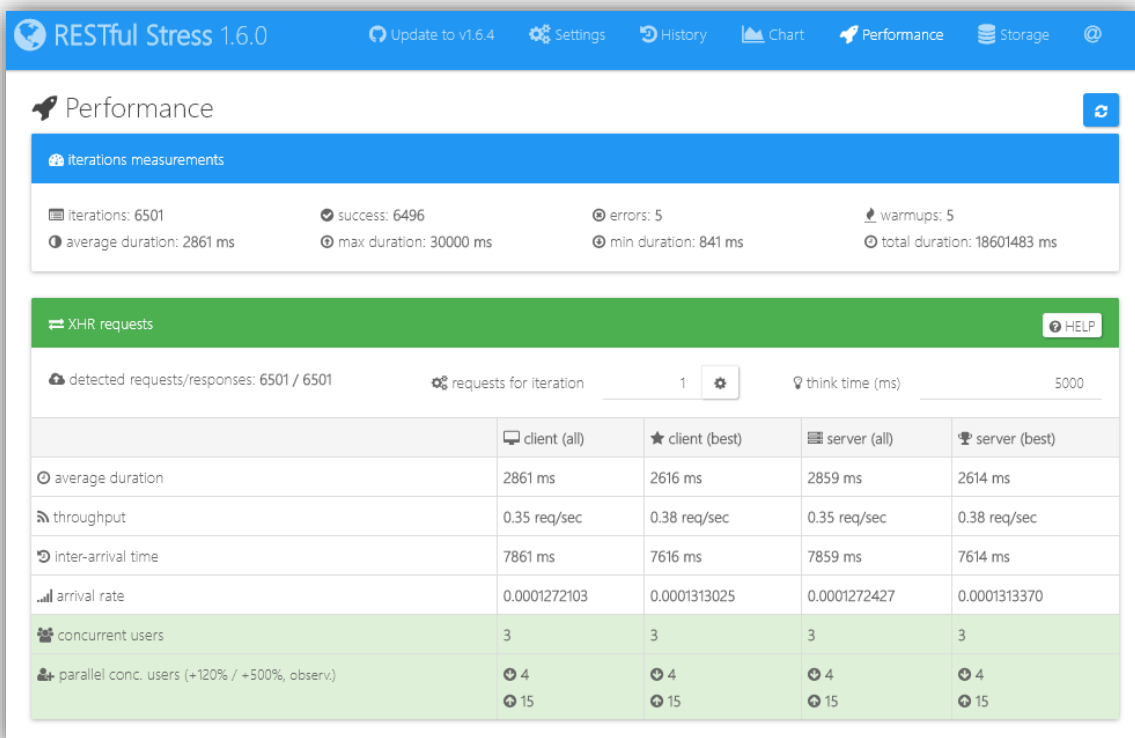


Figura 45. Usuarios concurrentes test estrés.

15.2 TESTS UNITARIOS

Se han realizado tests unitarios con la librería *unittest* de Python, nos ha servido para comprobar que todas las acciones transaccionales funcionaban correctamente, en total se han hecho 11 que explicamos a continuación:

Test	Detalles
test_API_clima	Se comprueba si la consulta a la API se realiza correctamente, para ello se ejecuta una llamada y se mira si la respuesta no está vacía (No se puede mirar por valor, ya que el clima varía constantemente).
test_envio_email_error	Se comprueba si el envío de email genera alguna excepción, en este caso solo se comprueba si falla (Con excepción controlada), ya que si no falla se habrá enviado correctamente.
test_filtro_cantidad	Se comprueba que los filtros funcionan correctamente, para ello se mira si se selecciona el número de hileras correctas para un ejemplo.
test_filtro_valor	Se comprueba que las hileras de los filtros tienen los valores correctos, para ello se ejecuta el filtro y se validan todos los valores por los que se ha filtrado.
test_info_evento	Se comprueba si se lee correctamente la información del evento, en este caso, solo se mira si se lee la descripción correctamente.
test_planif_personal	Se comprueba la planificación personalizada, para ello se valida si se obtiene el número de hileras que tocan del Excel de prueba.
test_planif_personal_error	Se comprueba que si hay un error lo detecta, para ello se etiqueta como que se espera que tenga un fallo.
test_random	Se comprueba si genera correctamente números aleatorios, para ello se generan dos y se valida si son diferentes. Puede que sean iguales, pero al ser en el rango [0,1000] es muy complicado.
test_sig_charla	Se comprueba que la siguiente charla se obtiene de forma correcta, para ello se mira si el título que se obtiene es el que tocaría.
test_ultima_charla	Se comprueba si se selecciona correctamente la siguiente charla si se encuentra en la última posición.
test_todas_charlas	Se comprueba si se leen todas las charlas del Excel, se valida a través del número de charlas leídas.

Tabla 11. Tests unitarios realizados.

Para ejecutar los tests simplemente hace falta llamar al método *main()* de la librería *unittest* desde nuestro *main*. Así, al ejecutar el fichero se harán los tests.

```

Anaconda Powershell Prompt (anaconda3)
(ventorno) (base) PS C:\Users\acastica\chatbot\eventis> python unit_tests.py -v
test_API_clima (__main__.TestActions) ... ok
test_envio_email_error (__main__.TestActions) ... expected failure
test_filtro_cantidad (__main__.TestActions) ... ok
test_filtro_valor (__main__.TestActions) ... ok
test_info_evento (__main__.TestActions) ... ok
test_planif_personal (__main__.TestActions) ... ok
test_planif_personal_error (__main__.TestActions) ... expected failure
test_random (__main__.TestActions) ... ok
test_sig_charla (__main__.TestActions) ... ok
test_todas_charlas (__main__.TestActions) ... ok
test_ultima_charla (__main__.TestActions) ... ok

-----
Ran 11 tests in 1.297s

OK (expected failures=2)
    
```

Figura 46. Ejecución tests unitarios.

Tras ejecutar el fichero que contenía los tests unitarios, se han pasado correctamente en un tiempo de 1.297s, además te marca en que tests se esperaba el fallo. A continuación mostraremos un fragmento de código de alguno de los tests realizados. Algunos de ellos ya contienen la integración con las actions y otros solo las funcionalidades simples.

```

def test_filtro_valor(self):
    fecha = "15/11/2019"
    seleccion = df.loc[df['Fecha']==fecha]
    seleccion = seleccion.reset_index(drop=True)
    self.assertEqual(seleccion['Fecha'][0], fecha, 'No se filtra correctamente por un valor concreto')

def test_API_clima(self):
    response = actionclima.getclima("Barcelona")
    vacio = None
    self.assertNotEqual(clime, vacio, 'No se obtiene nada de consultar la API')

@unittest.expectedFailure #Se espera que falle
def test_envio_email_error(self):
    with self.assertRaises(Exception) as context:
        actionmail.sendmail("noexiste@gmail.com")

    self.assertFalse('Excepción de email' in str(context.exception))
    
```

Figura 47. Código tests unitarios.

16. INSTALACIÓN Y DESPLIEGUE DEL ASISTENTE

16.1 INSTALACIÓN DE RASA

Primero de todo, debemos instalar **Python**, ya que el framework está basado en este lenguaje. Por compatibilidad con tensorflow en este proyecto se ha usado la versión **3.7.4 de 64 bits**, ya que las otras impedían la instalación por incompatibilidad de versiones a octubre de 2019.



Una vez instalado Python, se ha considerado muy oportuno instalar un **entorno virtual**, esto nos permitirá poder instalar todos los paquetes necesarios de forma más limpia sin comprometer nuestro sistema con conflictos de dependencias. Aunque en este proyecto no nos ha afectado, también permite instalar paquetes sin permiso de administrador. A continuación mostramos la creación y activación del entorno.

```
PS C:\Users\Alvaro\chatbot> python -m venv --system-site-packages ./venv
PS C:\Users\Alvaro\chatbot> .\venv\Scripts\activate
(venv) PS C:\Users\Alvaro\chatbot>
```

Además, para poder instalar rasa sin errores debemos instalar *visual c++ build tools* en versión 14 o superior y también ejecutar el comando `pip install -U setuptools` que nos permite tener actualizada la librería de instalación de paquetes de Python.

Finalmente, podemos proceder a instalar **Rasa** mediante `pip33`. A continuación mostramos una parte del proceso de instalación.

```
(venv) PS C:\Users\Alvaro\chatbot> pip install rasa
Collecting rasa
  Downloading https://files.pythonhosted.org/packages/6e/b4/dad07c728501a12a7df62977ee1d17ed3551c2b1a0a8bc2e21575ef74557/rasa-1.6.1-py3-none-any.whl (559kB)
    |#####| 563kB 1.7MB/s
Collecting ruamel.yaml~=0.15.0 (from rasa)
  Downloading https://files.pythonhosted.org/packages/9a/ee/55cd64bbff971c181e2d9e1c13aba9a27fd4cd2bee545dbe90c44427c757/ruamel.yaml-0.15.100.tar.gz (318kB)
    |#####| 327kB 2.2MB/s
Collecting python-socketio>=4.3.1 (from rasa)
  Downloading https://files.pythonhosted.org/packages/11/e0/71c90fbfc534108d0869ff5f92c16e4bc9b1347439b9022e815c03ff4193/python_socketio-4.4.0-py2.py3-none-any.whl (51kB)
    |#####| 51kB 544kB/s
Collecting questionnaire>=1.1.0 (from rasa)
```

³³ Sistema de gestión de paquetes utilizado para instalar y administrar paquetes en Python

Ahora ya tenemos el framework instalado, así que debemos crear un proyecto donde definiremos nuestro chatbot. Para ello, ejecutamos el comando `rasa init --no-prompt`. Este comando **nos genera un proyecto de un bot simple**, con los ficheros necesarios para que funcione, además, con el flag de no prompt, evitamos que se nos vayan haciendo preguntas durante la creación del proyecto.

```

Anaconda Powershell Prompt (anaconda3)
(ventorino) (base) PS C:\Users\acastica\chatbot\new_bot> rasa init --no-prompt
Welcome to Rasa!

To get started quickly, an initial project will be created.
If you need some help, check out the documentation at https://rasa.com/docs/rasa.

Created project directory at 'C:\Users\acastica\chatbot\new_bot'.
Finished creating project structure.
Training an initial model...
Training Core model...
2020-01-18 19:40:27      abs1 - Entry Point [tensor2tensor.envs.tic_tac_toe_env:TicTacToeEnv] registered with id [T2TEnv-TicTacToeEnv-v0]
Processed Story Blocks: 100%|          | 5/5 [00:00<?, ?it/s, # trackers=1]
Processed Story Blocks: 100%|          | 5/5 [00:00<00:00, 159.86it/s, # trackers=5]
Processed Story Blocks: 100%|          | 5/5 [00:00<00:00, 159.97it/s, # trackers=20]
Processed Story Blocks: 100%|          | 5/5 [00:00<00:00, 160.06it/s, # trackers=24]
Processed trackers: 100%|          | 5/5 [00:00<?, ?it/s, # actions=16]
Processed actions: 16it [00:00, 512.15it/s, # examples=16]
Processed trackers: 100%|          | 231/231 [00:01<00:00, 129.68it/s, # actions=126]
Model: "sequential"
    
```

Nombre	Fecha de modifica...	Tipo	Tamaño
__pycache__	18/01/2020 19:39	Carpeta de archivos	
data	18/01/2020 19:39	Carpeta de archivos	
models	18/01/2020 19:41	Carpeta de archivos	
__init__.py	16/10/2019 9:49	Archivo PY	0 KB
actions.py	16/10/2019 9:49	Archivo PY	1 KB
config.py	16/10/2019 9:49	Archivo YML	1 KB
credentials.py	16/10/2019 9:49	Archivo YML	1 KB
domain.py	16/10/2019 9:49	Archivo YML	1 KB
endpoints.py	16/10/2019 9:49	Archivo YML	2 KB

A partir de aquí, podemos empezar a modificar los ficheros tal y como se ha descrito en la sección de elementos del bot, para personalizar toda la información de nuestro asistente.


```
ngrok by @inconshreveable

Session Status      online
Session Expires    7 hours, 52 minutes
Version             2.3.35
Region              United States (us)
Web Interface        http://127.0.0.1:4040
Forwarding           http://da6b436b.ngrok.io -> http://localhost:5005
Forwarding           https://da6b436b.ngrok.io -> http://localhost:5005

Connections
  ttl    opn    rt1    rt5    p50    p90
   3      0    0.04   0.01   5.18   6.63

HTTP Requests
-----
POST /webhooks/telegram/webhook 200 OK
POST /webhooks/telegram/webhook 200 OK
POST /webhooks/telegram/webhook 200 OK
```

Figura 49. Despliegue ngrok.

Una vez tenemos expuesto nuestro localhost, procederemos a desplegar el servidor, o en nuestro caso, los servidores.

En primer lugar, debemos desplegar **un primer servidor para las acciones transaccionales**, ya que nuestro asistente hará uso de diversos endpoints para acceder a ellas. Para desplegar este servidor, debemos ejecutar el comando `rasa run actions`. En este servidor, también se nos mostrará la información de las diferentes llamadas que recibe.

```
(ventorno) (base) PS C:\Users\acastica\chatbot\eventis> rasa run actions
2020-01-19 13:47:07 rasa_sdk.endpoint - Starting action endpoint server...
2020-01-19 13:47:08 rasa_sdk.executor - Registered function for 'action_saludo'.
2020-01-19 13:47:08 rasa_sdk.executor - Registered function for 'action_random_number'.
2020-01-19 13:47:08 rasa_sdk.executor - Registered function for 'action_todas_charlas'.
2020-01-19 13:47:08 rasa_sdk.executor - Registered function for 'action_planificacion_personalizada'.
2020-01-19 13:47:08 rasa_sdk.executor - Registered function for 'action_filtro_ponente'.
2020-01-19 13:47:08 rasa_sdk.executor - Registered function for 'action_filtro_categoria'.
2020-01-19 13:47:08 rasa_sdk.executor - Registered function for 'action_filtro_fecha'.
2020-01-19 13:47:08 rasa_sdk.executor - Registered function for 'action_siguiente_charla'.
2020-01-19 13:47:08 rasa_sdk.executor - Registered function for 'action_clima'.
2020-01-19 13:47:08 rasa_sdk.executor - Registered function for 'action_enviar_email'.
2020-01-19 13:47:09 rasa_sdk.endpoint - Action endpoint is up and running on http ('0.0.0.0', 5055)
127.0.0.1 - - [2020-01-19 13:50:12] "POST /webhook HTTP/1.1" 200 624 0.046005
```

Figura 50. Despliegue servidor actions.

En el caso de que usemos la pipeline DucklingHTTPExtractor para la extracción de entidades por parte del NLP (en nuestro proyecto no lo hacemos), debemos asegurarnos de iniciar también un servidor local para ello. Así que, debemos ejecutar bajo un container de docker el servidor, con el comando `docker run -p 8000:8000 rasa/duckling`.

Al desplegar este primer servidor, ya podríamos usar nuestro asistente mediante la terminal, para ello ejecutamos el comando *rasa Shell*.

```
(ventorno) (base) PS C:\Users\acastica\chatbot\eventis> rasa shell
2020-01-19 14:31:30      root - Connecting to channel 'cmdline' which was specified by the '--connector' argument.
Any other channels will be ignored. To connect to all given channels, omit the '--connector' argument.
2020-01-19 14:31:30      root - Starting Rasa server on http://localhost:5005
2020-01-19 14:31:42      abs1 - Entry Point [tensor2tensor.envs.tic_tac_toe_env:TicTacToeEnv] registered with id [T
2TEnv-TicTacToeEnv-v0]
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input -> Hola
Bienvenido!! El siguiente evento corporativo es Conócenos.

Este es un evento para descubrir los diferentes proyectos que se llevan a cabo dentro de la empresa everis.

El evento se realiza el 29 y 30 de enero de 2020, empezando a las 09:00:00 en la calle Av. Diagonal, 605 - 4ª planta de
Barcelona, si quieres la ubicación exacta accede a: https://goo.gl/maps/1vA9yt34gwrEo51j6.

¡Pregúntame lo que quieras!
Your input -> dame el clima del evento
El clima actual de Barcelona, lugar donde tiene lugar el evento es: algo de nubes
Your input -> /stop
2020-01-19 14:37:54      root - Killing Sanic server now.
(ventorno) (base) PS C:\Users\acastica\chatbot\eventis>
```

Figura 51. Ejecutar asistente en terminal.

Si queremos usar nuestro asistente en un canal externo (Telegram y Google Assistant en nuestro caso), debemos desplegar **un segundo servidor para el modelo de Rasa**, a través de este servidor, recibiremos las peticiones HTTP por parte del canal que procesará el modelo. Para ello, ejecutaremos el comando *rasa run --enable-api*. Con ese flag permitimos utilizar la API de Rasa³⁴.

```
(ventorno) (base) PS C:\Users\acastica\chatbot\eventis> rasa run --enable-api
2020-01-19 13:47:40      root - Starting Rasa server on http://localhost:5005
2020-01-19 13:47:52      abs1 - Entry Point [tensor2tensor.envs.tic_tac_toe_env:TicTacToeEnv]
```

Figura 52. Despliegue servidor modelo Rasa.

Si nuestro asistente se ejecuta sobre Google Assistant, debemos ejecutar dos últimos comandos, *gactions update --action_package action.json --project {PROJECT_ID}* y *gactions test --action_package action.json --project {PROJECT_ID}*. El id del proyecto nos lo proporciona Google al crear el action.

Con estos pasos, ya podríamos usar nuestro asistente en los canales que hayamos definido, recordamos que solo podrá ser accesible durante 8 horas antes de tener que actualizar la URL de nuestro localhost.

³⁴ Métodos definidos en <https://rasa.com/docs/rasa/api/http-api/>

17. INTEGRACIÓN DE CONOCIMIENTOS

En el proyecto se han integrado diversas disciplinas que se han trabajado durante los estudios de grado. A continuación vamos a mostrar las diversas disciplinas que incluye nuestro proyecto:

- **Programación:** Se ha aplicado para programar en Python las diferentes acciones transaccionales que realiza nuestro chatbot, más allá de contestar con una frase predefinida.
- **Estructuras de Datos y Algoritmos:** Se han aplicado algoritmos de búsqueda y ordenación para buscar entre las charlas y encontrar la siguiente o la primera que cumple algún requisito.
- **Aplicaciones y Servicios Web:** Se han realizado llamadas GET a un servicio externo del tiempo para obtener el clima actual del evento y también al crear el conector con el canal de Google Assistant. Por otra parte, se han hecho tests de estrés mediante métodos POST a la API de nuestro bot.
- **Redes de Computadores:** Se han usado protocolos de transferencia simple entre diferentes computadoras, para enviar los correos electrónicos al usuario.
- **Arquitectura del Software:** Se ha desarrollado, se mantiene y se evalúa un sistema software complejo y se han valorado las necesidades del usuario al usar el asistente, adaptando la implementación del software a estas necesidades. También se incluyen los diagramas necesarios.
- **Ingeniería de Requisitos:** Se ha valorado los objetivos del proyecto, las partes interesadas y se han realizado diagramas de flujo.

18. IDENTIFICACIÓN DE LEYES Y REGULACIONES

Tras realizar un estudio sobre las posibles leyes y regulaciones que pueden afectar a nuestro proyecto, hemos concluido hay dos leyes que nos afectan en el proyecto:

La primera es la **Ley Orgánica de Protección de Datos de Carácter Personal (LOPD)**, ley Orgánica 15/1999, de 13 de diciembre– ya que nuestro asistente almacena información de carácter personal al guardar el número de empleado para consultar las preferencias del usuario y así personalizar su programación y también al guardar el número de empleado al inscribirse en una charla.

Para cumplir esta ley, informaremos al usuario al inicio de la conversación con el asistente sobre el uso que se hace de sus datos y cómo se almacenan en nuestra aplicación.

La segunda ley es la **Ley de Propiedad intelectual (LPI)** –Ley 2/2019, de 1 de marzo– ya que *la totalidad del contenido de este proyecto, entendiendo por contenido a título meramente enunciativo, textos, imágenes, ficheros, fotografías, logotipos, gráficos, marcas, iconos, combinaciones de colores, o cualquier otro elemento, su estructura y diseño, la selección y forma de presentación de los materiales incluidos en la misma, y el software, links y demás contenidos audiovisuales o sonoros, así como su diseño gráfico y códigos fuentes necesarios para su funcionamiento, acceso y utilización, están protegidos por derechos de propiedad industrial e intelectual, titularidad de everis.*³⁵ Pese a ello, el estudiante tendrá derecho a la propiedad intelectual e industrial del trabajo, si así lo solicita, según se encuentra estipulado por la cláusula tercera del convenio de cooperación educativa para la realización de prácticas académicas externas con número 19-210-25.

Podemos concluir que el estudio de la regulación asociada a nuestro proyecto tiene en cuenta el total las leyes citadas previamente.

³⁵ Fragmento del aviso legal de everis: <https://www.everis.com/spain/es/aviso-legal>

19. CONCLUSIONES Y RECOMENDACIONES DE MEJORA

19.1 CONCLUSIONES DE LA TECNOLOGÍA

Tras usar este framework durante cuatro meses, podemos concluir que *Rasa tiene un equipo con ganas de hacer las cosas bien, pero que al ser un producto joven todavía tiene algunas funcionalidades con gran potencial de mejora.*

La parte en la que destaca el asistente es por su comunidad y documentación, ya que para adentrarse en el mundo de los chatbots te ofrece una gran cantidad de documentación para gente que no está familiarizada con los chatbots. Aunque una de las cosas que hemos observado es que la documentación se basa, en gran parte, en tutoriales de cómo realizar algunas cosas. En ese aspecto hemos echado en falta documentación más técnica de cómo funciona internamente el framework.

La parte en la que más floja se encuentra la tecnología es la integración con funcionalidades que no son out of the box³⁶, como por ejemplo canales específicos o exportación del NLU en diferentes formatos, ya que genera bastantes problemas.

Por otra parte, al ejecutarse en localhost nos brinda un control total del servidor, cosa que nos ha parecido muy interesante y valoramos positivamente en cuanto a privacidad.

Hemos detectado que ngrok, el software que nos permite crear una URL que sirve de túnel para hacer público nuestro localhost, solo nos permite 40 usuarios concurrentes por minuto en versión gratis y 120 en la versión Business, la versión más cara de las que disponen. Por lo tanto, *concluimos que ngrok es poco escalable y no es una buena solución a la larga*, ya que es normal que más de 120 usuarios utilicen el bot en un minuto. Otro problema de ngrok, es que el dominio caduca cada 8 horas y se tiene que modificar la configuración del bot.

Durante el desarrollo del proyecto, se ha cancelado la integración del asistente con la plataforma de everis llamada eVA. Los motivos que se han tenido en cuenta son dos: el primero de todos es la necesidad de crear una arquitectura compleja para hacer la conexión –con el tiempo y coste que esto supone– y el segundo motivo es la confidencialidad que rodea a la plataforma, ya que se trata de un activo interno de la empresa y no se podría dar detalles sobre su funcionamiento durante el proyecto.

19.2 CONCLUSIONES PERSONALES

A nivel personal, en este proyecto he aprendido a llevar a cabo la actividad de desarrollo de un proyecto de software como se hace a nivel profesional, también a trabajar con roles diferentes y luego combinar los resultados, y a gestionar un proyecto en temas de planificación, plazos, presentaciones orales y documentación.

Por otra parte, y muy importante, me ha servido para aprender una nueva tecnología y un campo de la informática nuevo. Gracias al proyecto he podido conocer cómo funciona un chatbot y cómo se implementa, también a usar muchas herramientas de desarrollo de software que me serán útiles para mi vida profesional.

³⁶ Funcionalidad que se puede usar sin ninguna instalación especial.

En cuanto a gestión, hemos realizado una buena planificación del tiempo del proyecto, ya que no hemos sufrido apenas cambios respecto la planificación inicial. Tan solo hemos sufrido dos desviaciones: La primera de ellas se encuentra en la fase de documentación, ya que inicialmente se planificó que se documentaría durante todo el proyecto paralelamente a la implementación del asistente, pero finalmente no ha podido ser así, ya que durante dos semanas y media tras generar la documentación de GEP, solo se ha trabajado en la instalación e implementación del chatbot.

La segunda desviación se encuentra en la fase de desarrollo del asistente, ya que debido a haberle dedicado el tiempo completo durante las dos primeras semanas de octubre, hemos podido finalizar antes de tiempo la implementación del asistente y las pruebas de funcionamiento.

Estas desviaciones han sido fruto de que, sobre todo en el primer contacto con la tecnología, he preferido dedicar el 100% del tiempo disponible a ello para evitar distracciones y perder el seguimiento de lo que hacía. Posteriormente, se ha vuelto a documentar el proyecto. Pese a ello, no nos han afectado negativamente ni al coste del proyecto ni a los objetivos, ya que este cambio nos ha permitido poder acabar antes la fase de implementación y dedicarle tiempo completo a documentar lo que faltaba del proyecto, en lugar de dedicarle la mitad del tiempo a cada tarea.

19.3 POSIBLES MEJORAS

- Utilizar otra herramienta para sustituir a ngrok, ya que este nos limita los usuarios concurrentes y además, cada 8 horas tenemos que actualizar el dominio. Para solucionar esto tendríamos que encontrar un dominio que no tenga caducidad (por ejemplo aws de amazon) y usar ese dominio como túnel a nuestro localhost.
- Personalizar la salida según el canal, de tal forma que el asistente varíe su respuesta en función del canal que se está usando.
- Automatizar el despliegue del asistente, para no tener que ejecutar todos los comandos manualmente.
- Añadir diversos idiomas a nuestro bot, aunque seguramente esta funcionalidad nos implique tener diversos modelos del NLU del asistente.

REFERENCIAS

- Barr, J. (2017). Amazon Lex – Now Generally Available | AWS News Blog. Retrieved December 16, 2019, from <https://aws.amazon.com/es/blogs/aws/amazon-lex-now-generally-available/>
- Bates, M. (1995). *Models of natural language understanding* (Vol. 92).
- Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... Thomas, D. (2001). *Manifesto for Agile Software Development Twelve Principles of Agile Software*. Retrieved from <http://www.agilemanifesto.org>
- Comissió Permanent. (2017). *NORMATIVA DEL TREBALL FINAL DE GRAU DEL GRAU EN ENGINYERIA INFORMÀTICA DE LA FACULTAT D'INFORMÀTICA DE BARCELONA*.
- Equip Deganal. (2019). *Acord 26/04/2019 Increment remuneració estudiants en pràctiques 2019/2020*.
- Fernández, C. (2018). Historia de los Chatbots. Retrieved December 16, 2019, from <https://www.denoticias.es/tecnologia/revolucion-digital/historia-chatbots.html>
- Gavilán, I. (2018). *La carrera digital*. Retrieved from https://books.google.es/books?hl=es&lr=&id=o-KaDwAAQBAJ&oi=fnd&pg=PT8&dq=auge+de+los+chatbots&ots=vAtSH_EvPZ&sig=1wcKvRmZZ5jK8fEKrKBNRUKqH6Y#v=onepage&q&f=false
- Gelfenbeyn, I. (2016). API.AI is joining Google! Retrieved December 9, 2019, from <https://blog.dialogflow.com/post/joining-google/>
- Marco della Cava. (2016). Microsoft CEO Nadella: “Bots are the new apps.” *USA Today*. Retrieved from <https://eu.usatoday.com/story/tech/news/2016/03/30/microsoft-ceo-nadella-bots-new-apps/82431672/>
- McKinney, W., & Team, P. D. (2015). Pandas - Powerful Python Data Analysis Toolkit. *Pandas - Powerful Python Data Analysis Toolkit*, 1625. Retrieved from <https://pandas.pydata.org/pandas-docs/stable/>
- Nasser, F., Al Omran, A., & Treude, C. (2017). *Choosing an NLP library for analyzing software documentation: a systematic literature review and a series of experiments*. 187–197. <https://doi.org/10.1109/MSR.2017.42>
- Nichol, A. (2017). We don't know how to build conversational software yet. Retrieved December 10, 2019, from <https://blog.rasa.com/we-dont-know-how-to-build-conversational-software-yet/>
- Penzenstadler, B. (2013). What does Sustainability mean in and for Software Engineering? *Proceedings of the 1st International Conference on ICT for Sustainability (ICT4S)*, (May), 9–12.
- Reilly, A. D. (2016). Standards for software documentation. Retrieved September 30, 2019, from <http://www.tcworld.info/e-magazine/translation-and-localization/article/standards-for-software-documentation/>
- Schwaber, K., & Sutherland, J. (2010). *The Scrum Guide™ The Definitive Guide to Scrum: The Rules of the Game*.

- Stent, A., & Bangalore, S. (2001). Natural language generation in interactive systems. In *Natural Language Generation in Interactive Systems* (Vol. 9781107010). <https://doi.org/10.1017/CBO9780511844492>
- Sutikno, T., Handayani, L., Stiawan, D., Riyadi, A., Much, I., & Subroto, I. (2016). WhatsApp, Viber and Telegram: which is the Best for Instant Messaging? *International Journal of Electrical and Computer Engineering (IJECE)*, 6(3), 909–914. <https://doi.org/10.11591/ijece.v6i3.10271>
- Takeuchi, H., & Nonaka, I. (1986). The New New Product Development Game. *Harvard Business Review*. Retrieved from <https://hbr.org/1986/01/the-new-new-product-development-game>
- Turing, A. M. (1950). COMPUTING MACHINERY AND INTELLIGENCE. In *Computing Machinery and Intelligence. Mind* (Vol. 49).
- Wallace, L., & Keil, M. (2004). Software project risks and their effect on outcomes. In *Communications of the ACM* (pp. 68–75).
- Zapfen, A. L. (2018). ¿Por qué somos expertos en lingüística además de desarrolladores tecnológicos? - Gus Chat. Retrieved September 24, 2019, from <https://gus.chat/por-que-somos-expertos-en-linguistica-ademas-de-desarrolladores-tecnologicos/>