Chatbot para la Sede Electrónica de la Dirección General de Policía

– –

Chatbot for Electronic Headquarters of the Spanish National Police

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
CURSO 2021–2022

Rebeca Herranz Gómez

Directores

Luis Javier García Villalba
Ana Lucila Sandoval Orozco

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2022

Agradecimientos

A mis directores del TFG, que han confiado en mí desde el principio para realizar este proyecto. También a Luis, Sandra y Daniel por ofrecerme un apoyo continuo durante todas las semanas, ayudándome con las dudas que tenía sobre el código y documentación.

A todo el grupo de policías que me han explicado los trámites dentro de la policía, siempre me han estado ofreciendo su ayuda y dándome su feedback.

A mi familia y a María que siempre han estado ahi, apoyándome y haciendo un poco más fácil mi vida estudiantil.

A Álvaro Castillo, quién ha dado un soporte increíble sobre la herramienta de Rasa Stack.

A la Facultad de Informática de la Universidad Complutense de Madrid por habernos ofrecido herramientas online en momentos de COVID-19 con los que he podido avanzar en el trabajo sin tener que acudir presencialmente.

Índice General

Índice de Figuras

Índice de Tablas

Listings

Lista de Acrónimos

A.L.I.C.E	*Artificial LinguisticInternet Computer Entity*
AI	*Artificial Intelligence*
AIML	*Artificial Intelligence Markup Language*
COVID-19	*Coronavirus*
DIET	*Dual Intent Entity Transformer*
DL	*Deep Learning*
IA	*Inteligencia Artificial*
LIJ	*Long Island Jewish*
LMS	*Learning Management System*
ML	*Machine Learning*
NLP	*Natural Language Proccessing*
NLU	*Natural Language Understanding*
SDK	*Software Development Kit*

VLE *Virtual Learning Environment*

Abstract

Nowadays, there is a great number of conversational assistants in many different institutions, such as Siri or Alexa. That is the reason why it was deemed necessary to develop a chatbot which helped users with their doubts about the procedures that can be carried out in the environment of the National Police (Policía Nacional) Electronic Headquarters. Within this context, it is decided to focus this End of Degree Dissertation on developing a conversational assistant capable of answering simple questions regarding immigration matters, and renewal or application for an ID card or Passport. Before programming the assistant, a previous analysis and study of the tools available for the development of chatbots in order to find the most adequate application. The framework chosen is Rasa Stack, generated by Machine Learning algorithms capable of answering simple questions formulated through natural language. Additionally, the chatbot has been published in two text channels: Telegram, and the web server provided by the tool BotFront. The result of this project is a conversational assistant developed with Rasa Framework, which is capable of answering documentation-related questions, as well as helping users carry out the procedures that are available on the National Police Electronic Headquarters. Moreover, this End of Degree Dissertation serves as a manual for all the functionalities that Rasa Stack has regarding the programming of a chatbot.

Keywords: Chatbot, Conversational Assistant, Electronic Office, Entities, ID, Immigration Matters, Machine Learning, Natural Language Processing, Passport, Policía Nacional, Rasa.

Resumen

Actualmente hay una gran diversidad de asistentes conversacionales en diferentes instituciones, como por ejemplo Siri o Alexa. Por ello se vio necesario desarrollar un chatbot que ayudará a los usuarios a resolver sus dudas sobre los trámites que se encuentran dentro de la Sede Electrónica de la Policía Nacional. Con este contexto, se decide enfocar el Trabajo Final de Grado en desarrollar un asistente conversacional que pueda responder preguntas sencillas relacionadas con los procesos de extranjería, renovación y solicitud del dni o pasaporte. A la hora de programar el asistente, se ha realizado un estudio previo de herramientas dedicadas al diseño de chatbots con el fin de encontrar la aplicación idónea. Finalmente, el framework que se utiliza es Rasa Stack, generado por algoritmos de Machine Learning capaces de responder a mensajes formulados a través del lenguaje natural. Por otro lado, se ha publicado el asistente en dos canales de texto: Telegram y el servidor web ofrecido por la herramienta BotFront. En resultado de este proyecto es un asistente conversacional desarrollado con Rasa Framework, capaz de responder a cuestiones sobre documentación o guiar al usuario en los trámites que brinda la Policía Nacional. Además, este Trabajo Fin de Grado explica como un manual todas las funcionalidades que tiene Rasa Stack con el fin de programar un chatbot.

Keywords: Aprendizaje Automático, Asistente conversacional, Chatbot, DNI, Entidades, Extranjería, Pasaporte, Policía Nacional, Procesado del Lenguaje Natural, Rasa, Sede Electrónica.

Capítulo 1

Introducción

1.1. Motivación

Los procesos sobre la Sede Electrónica del Cuerpo Nacional de Policía han ido evolucionando y sufrido cambios debido a la pandemia por el *Coronavirus* (COVID-19). Por ello, los ciudadanos se encuentran con más preguntas a la hora de necesitar algunos de los puntos que brinda esta página web.

La policía presenció esta elevación de preguntas, y por ello se decidió que sería de utilidad crear un chatbot que ayudara al ciudadano con las dudas que tuviera sobre estos trámites. Por tanto la motivación de este proyecto reside en responder dichas dudas que presente el usuario mediante un asistente conversacional inteligente.

Un chatbot, o bot conversacional, es un programa informático que interactúa con personas a través de un chat textual o mediante voz, ofreciendo respuestas y soluciones rápidas a tareas repetitivas y preguntas comunes susceptibles de ser automatizadas. Llevan existiendo desde los principios de la programación pero gracias a los avances realizados dentro del mundo de la inteligencia artificial durante estos últimos años, han hecho resurgir el interés en ellos debido a las nuevas técnicas de reconocimiento y análisis del lenguaje natural.

Con este contexto se puede dar uso a dos campos de la inteligencia artificial: *Natural Language Understanding* (NLU) y *Natural Language Proccessing* (NLP). Estos dos componentes aportan resultados concretos sobre las cuestiones que realizan los clientes, comprendiendo el habla humana a medida que se reproduce y además capacita a un sistema de comprender el significado e intención que encierra una frase. Actualmente existen varias aplicaciones que hacen uso de estos campos con el fin de diseñar un asistente conversacional. El ejemplo que se ve en este trabajo es Rasa, framework sencillo y completo con todas las funcionalidades necesarias.

Pero primero, antes de empezar a programar, se necesita extraer todas las dudas posibles encontradas por foros y además de la información depositada en la misma Sede Electrónica, para su posterior análisis y recopilación en un *dataset* , con el fin de entrenar al chatbot.

1.2. Objeto de la Investigación

El objetivo de este proyecto es el diseño e implementación de un chatbot basado en las preguntas realizadas con anterioridad tanto en foros o dudas comunes suministradas por la Policía. Con este elemento se pretende mejorar y agilizar los procesos que pertenecen a la Sede Electrónica, y a su vez, ayudar al ciudadano, respondiendo las cuestiones que le puedan surgir mientras visita la web. Por tanto, la clave es hallar cómo se va acelerar las interacciones entre personas y servicios, mejorando así la experiencia del cliente. Al mismo tiempo, se quiere ofrecer a la Policía una opción para mejorar el proceso de compromiso con los clientes y además beneficia la eficiencia operativa al reducir el costo típico del servicio al cliente.

De manera paralela, se tiene la intención de clasificar las preguntas y respuestas que se extraigan con el fin de obtener un número de expresiones y palabras que creen un vocabulario para luego entrenar el chatbot. A través de este proyecto, además, se pretende entender como funcionan los asistentes conversacionales, analizar como es su estructura y encontrar las principales dificultades que los ingenieros de informáticos se encuentran a la hora de desarrollar dicho software.

1.3. Plan de Trabajo

El desarrollo de este trabajo se ha realizado en tres fases:

1. **Investigación**:

   Durante los primeros tres meses se llevó a cabo un periodo de adaptación al contexto del trabajo y de adquisición de conocimientos necesarios para comenzar con el posterior desarrollo. Al comienzo, se realizó una reunión general en la que se explicó el punto de partida del trabajo, cuales iban a ser los objetivos que lograr y cuales iban a ser los conocimientos necesarios. Tras esto, se acordaron numerosas reuniones con el fin de realizar un seguimiento del progreso en la investigación y para resolver las posibles dudas que surgieran. Otro motivo por el cual se concertaron dichas reuniones fue para poder explicar los fundamentos básicos de las aplicaciones con la que se pueden programar asistentes. En las reuniones también se explicaron numerosas herramientas que facilitarían la labor de investigación. Una de estas herramientas fue *Google Scholar*, la que el equipo usaría para buscar artículos científicos sobre investigaciones anteriores realizadas sobre el mismo campo de estudio. Finalmente, una vez obtenidas varias conclusiones acerca los requerimientos del proyecto, se comenzó a dar prioridad al desarrollo.

2. **Desarrollo**:

   Una vez se adquirieron los conocimientos necesarios para poder empezar a crear una versión sólida del proyecto, se decidió comenzar con la codificación de la propuesta. La fase de investigación no cesó, ya que era imprescindible saber sobre conceptos avanzados del lenguaje de programación de *Python* y manejo con la herramientas Rasa Stack. Durante esta fase se construyeron también los datatsets de entrenamiento, basados en información extraída de trabajos leídos durante la fase anterior.

3. **Experimentación**:

Finalmente se realizaron los prototipos de la idea inicial, comenzó el proceso de experimentación, en el qué se realizaron pruebas en servidores web para visualizar el aspecto del chatbot. También se realizaron varias pruebas que consistieron en ir comprobando si el asistente respondía correctamente a todos los trámites que abarcaba el proyecto. Por otra parte se hicieron pruebas de estrés con la herramienta de RESTful. Finalmente, se realizó una comparación de los resultados obtenidos derivados de la fase de experimentación y con los adquiridos en otros trabajos de investigación.

1.4. Estructura del Trabajo

El resto del trabajo está organizado en 6 capítulos y 1 anexo con la estructura que se comenta a continuación:

El Capítulo 2 se explica la historia de los asistentes conversacionales, además de describir la relación actual que se mantiene con la inteligencia artificial. Por otro lado se aclaran conceptos que son elementales para comprender el funcionamiento de los chatbots. Finalmente se describe los procesos que abarca este proyecto, mostrando el entorno de la Sede Electrónica.

El Capítulo 3 se presenta el estado del arte, donde se realiza una revisión de diferentes soluciones enfocadas en varios sectores empresariales, con el fin de apreciar la compenetración de los asistentes en el panorama económico-social. Por otro lado, se analizan todas las aplicaciones capaces de desarrollar por un asistente, explicando los puntos a favor y en contra de cada uno, para luego tomar la decisión más acertada a este trabajo. En este apartado se incluye una tabla comparativa que resume las principales características de cada propuesta analizada.

El Capítulo 4 presenta la metodología y análisis del proyecto. Primero se explican diferentes diagramas que han sido esenciales para estructurar el asistente. Luego, se incide en el funcionamiento interno de Rasa, desarrollando el contenido de cada fichero. Finalmente se tratará el tema del despliegue de Rasa y cómo se realiza la conexión entre esta herramienta, Telegram y un servidor web.

El Capítulo 5 describe los experimentos realizados para evaluar la efectividad de la programación del asistente propuesta en el capítulo 4 y se presentan los resultados obtenidos.

El Capítulo 6 muestra las principales conclusiones de este trabajo y las líneas futuras de investigación.

Los Capítulos 7 y 8 son las traducciones al inglés de la Introducción y de las Conclusiones.

Capítulo 2

Contexto de la Investigación

En esta sección se explica el contexto en el que se desenvuelve el Trabajo Fin de Grado. En el punto 2.1, se resume la historia de los chatbots y cómo comenzaron a hacerse un hueco en esta sociedad. En la sección 2.2 se sitúa en qué contexto social se encuentra la humanidad y cómo esta integrada la digitalización en el día a día de las personas. En el punto 2.3, se explica en qué consiste la inteligencia artificial y se puntualiza el término NLP. En el siguiente apartado 2.4 se detalla cómo funciona un chatbot y los elementos esenciales que lo forman. En el siguiente punto 2.5 se explicará la combinación de librerías que se utilizan para llevar a cabo el funcionamiento del asistente. Finalmente, en el apartado 2.6 se explica el punto de partida del chatbot, la página web de la Sede Electrónica y el apartado donde se encuentra el asistente.

2.1. Historia de los Asistentes Conversacionales

Sobre el año 1950, Alan Turing publicó el paper "Computer Machinery and Intelligence", en el que se explicó el Test de Turing [TH50], donde verificaba si una máquina puede llegar a tener un comportamiento inteligente similar al del ser humano. El test consistía en mantener una conversación entre una persona y otra entidad (persona o máquina) a través del chat. Entonces la persona tenía que determinar si hablaba con una máquina o un humano.

Posteriormente, en 1960, el primer programa que simulaba una conversación fue ELIZA [Wei66], creado por el profesor Joseph Weizenbaum. ELIZA reconoce palabras clave o frases para reproducir una respuesta. El objetivo de este chatbot era simular a un psicoterapeuta escuchando los problemas de su interlocutor y escribiéndole las respuestas a sus preguntas; de esta manera conseguía ayudar a la persona y así continuará hablando sobre sus preocupaciones. Si al final no tenía respuestas disponibles, optaba por otras técnicas, como la generación de preguntas utilizando como base la frase que le decía su paciente.

Once años después, en 1971, Kenneth Colby creó PARRY [PCFR16]; un asistente que utilizaba una estructura parecida a ELIZA con el fin de simular a un paciente con paranoia. Pero había una diferencia entre estos dos chatbots, PARRY si era consciente de la conversación y también de su estado de ánimo. Por tanto, las respuestas no venían influenciadas solo por la entradas sino también por las creencias,deseos e intenciones de PARRY. Por ejemplo, tenía la capacidad de ignorar, cambiar de tema de conversación e

introducir microrrelatos en las conversaciones.

En 1995, se creó un chatbot esencial en la historia llamado *Artificial LinguisticInternet Computer Entity* (A.L.I.C.E) por Richard Wallace [AA15]. Dicho asistente utilizaba patrones de reconocimiento avanzados, *Artificial Intelligence Markup Language* (AIML), desarrollado también por Wallace. Debido a estas cualidades, ganó 3 veces el premio Loebner y más concursos relacionados con la evaluación del procesamiento del lenguaje natural.

Aunque estos chatbots estuvieran perfectamente implementados, era muy complicado imitar a la perfección el comportamiento humano ya que cometían faltas de ortografía y carecían de sentido en algunas expresiones. Pero en 1997, Robby Garner desarrollo Albert One [Der10], un asistente popular por su comportamiento humano . Albert se basa en el marco de FRED, una serie de simuladores de conversación creados en1 1978, y estudios de conversaciones humanas derivados de instalaciones de Internet y prototipos desarrollados en el estudio de Robby. Dicho asistente figura en el Libro Guinness de los Récords Mundiales de 2001 como el programa informático más humano del mundo.

El último chatbot remarcable ha sido Mitsuku [ST20], ganador del premio Loebner cuatro veces. Ha sido desarrollado por Steve Worswick, utilizando como base los ficheros AIML de A.L.I.C.E. La base de datos del asistente tiene muchos puntos diferentes, en el que cada uno de ellos tienen varias propiedades que lo describen.

Y finalmente, llegando a la actualidad, se encuentran asistentes conversacionales como Siri o Alexa, chatbots con el fin de ayudar a los ciudadanos en su día con cualquier acción de su vida cotidiana.

2.2. Momento Actual

En marzo de 2020 se proclamó estado de alarma es España debido a la pandemia del COVID-19. Este mandato obligó a permanecer en casa durante 6 meses, sin poder salir a no ser por casos excepcionales. Por esta razón, la sociedad recurrió de manera más frecuente a los servicios online y con ello comenzaron a colapsarse varios sistemas por dicha demanda.

Los servicios online ante este panorama reaccionaron, creando un mayor soporte para sus clientes. Algunos de ellos decidieron implantar chatbots para resolver preguntas simples y así no colapsar los sistemas o líneas telefónicas. Además de poder responder preguntas, también se han utilizado los chatbots a la hora de aconsejar sobre temas del coronavirus o por ejemplo, pedir cita médica a través de una conversación sencilla.

Un ejemplo es el asistente virtual de Northwell Health [Die21] que ayuda a reducir las faltas para las citas de colonoscopias, un procedimiento esencial en el diagnóstico del cáncer colorrectal.

Este problema es grave ya que el 40 % de la población desfavorecida no cumplen con dicho paso [dV21]. Está solución basada en inteligencia artificial se está desarrollando en el *Long Island Jewish* (LIJ) Medical Center y Southside Hospital. El chatbot tiene el objetivo de aclarar las preocupaciones de los pacientes mediante respuestas informativas por correo electrónico o mensajes de texto.

En otros casos, los chatbots de asistencia médica pueden conectar a los pacientes con los médicos, para poder dar un diagnóstico o tratamiento. Por tanto, el propósito futuro

es de que los chatbots desempeñen el papel de primer contacto en el sector de la atención primaria. En caso de que el chatbot no pueda responder a las preguntas, transferirá los problemas planteados a un médico especializado.

A su vez también los chatbots se están explotando en otros sectores, como por ejemplo en el de la educación. Las principales funciones que pueden desempeñar son el compartir información y resolver preguntas, pudiéndolo así utilizar tanto profesores, alumnos y padres.

Un ejemplo es el asistente CourseQ [Her21], donde los alumnos, pueden buscar información y resolver dudas de manera inmediata gracias a un servicio que está disponible las 24 horas del día. También se pueden utilizar incluso para reservar una tutoría con algún profesor. Por el lado de los maestros, pueden seguir la evolución de sus estudiantes y además utilizar el chatbot como refuerzo académico. Y respecto a los padres, se pone a su disponibilidad la capacidad de mantener el contacto con el centro de estudios, evitando así las llamadas o los correos electrónicos.

A su vez los chatbots académicos se han utilizado para dar conferencias básicas, como se está realizado en la escuela pública Summit (Estados Unidos) [Com19]. Este proyecto está llevado a cabo por Bill, Melissa Gates Foundation y Facebook. El objetivo final de este proyecto es de que los estudiantes dispongan de asesores virtuales y que se adapten al ritmo de aprendizaje de cada alumno.

Y por último, el sector bancario también está utilizando los asistentes virtuales para agilizar sus trámites o responder preguntas frecuentes, guiando a los usuarios para realizar transacciones como transferencias de dinero o verificaciones de saldo en sus propias cuentas. Otra de las funcionalidades que dispone es dar respuestas personalizadas a los clientes gracias a la información que dispone de ellos. Otra característica que disponen los asistentes bancarios, es la capacidad de anticipar las acciones de los clientes y brindar ofertas y servicios particulares, aumentando así la fidelidad de sus usuarios.

Pero no solo los clientes son los beneficiados del uso de los chatbots, los empleados también reducen gastos operativos, minimizan los errores humanos y ahorran tiempo. Según un estudio de Juniper Research [Gro21], con el uso de los chatbots los bancos de todo el mundo se podrán ahorrar hasta 7.300 millones de dólares para 2023, lo que les dará la capacidad de competir con otras sucursales e implementar innovaciones tecnológicas de manera satisfactoria.

Para finalizar, se ha explicado como los chatbots están integrados en la sociedad mediante tres grandes sectores: la medicina, la enseñanza y la economía [Tea21]. Esto quiere decir a su vez, que los asistentes no remplazarán a los agentes humanos, sino que dan una solución automatizada de procesos simples y sin molestar a los trabajadores, agilizando así a las empresas.

2.3. Inteligencia Artificial y Procesamiento del Lenguaje Natural

2.3.1. Inteligencia Artificial

La *Inteligencia Artificial* (IA) tiene el objetivo de simular la inteligencia humana en máquinas implementadas para pensar como las personas o imitar sus acciones

[PÁPGMAV21]. A medida que la tecnología ha ido avanzando, la investigación en el sector de IA también ha ido evolucionando dando a conocer nuevos términos como *Machine Learning* (ML) y *Deep Learning* (DL). La relación de estos tres términos se ve representada en la Figura 2.1.

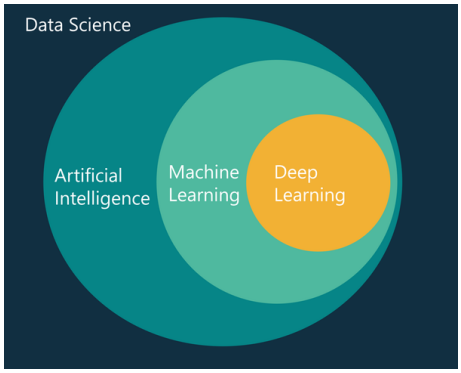


Figura 2.1: Inteligencia Artifical, Machine Learning y Deep Learning

Al principio los métodos que dominaban la IA eran la lógica formal y los sistemas expertos. Sin embargo, con el desarrollo de la solución de problemas específicos, y también nuevos vínculos entre la IA y otros campos, surge el ML. En este caso, el ML brinda a las máquinas la capacidad de aprender y mejorar automáticamente, basándose en ensayos anteriores. Estos sistemas transforman los datos en conocimiento o *insights* con el fin de perfeccionar la elección de decisiones en prácticamente cualquier ámbito. El ML está conectada con la estadística computacional, que se centra en realizar predicciones. Gracias a los avances más actuales en ML, se ha podido utilizar estas técnicas para mejorar el desempeño en diferentes zonas de conocimiento, como por ejemplo en el campo de NLP.

2.3.2. Procesamiento del Lenguaje Natural

El NLP [KKKS17] es una rama de la IA y la lingüística enfocada en la comprensión por parte de los computadores, de los enunciados o palabras escritas en lenguaje natural, con el fin de obtener conocimientos a partir de información en formato de texto. En la Figura 2.2 se puede ver la importancia que acoge el NLP en el mundo de la IA.

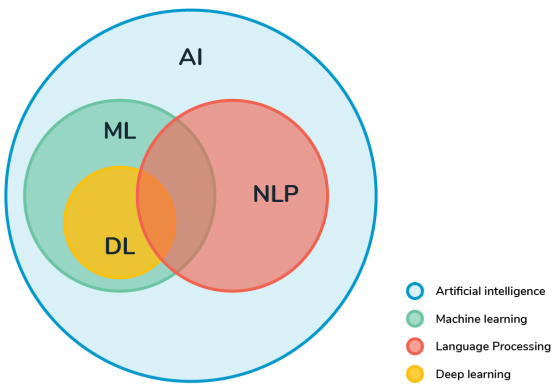


Figura 2.2: Relación entre IA y NLP

Su principal funcionalidad es recopilar información sobre cómo los humanos comprenden y usan el lenguaje, de manera que se fomente el desarrollo de las herramientas y técnicas para que los sistemas informáticas puedan entender e interactuar el lenguaje humano [SSA+18].

Al principio de la historia del NLP, se aplicaba una colección de reglas elaboradas de manera manual para que las computadoras pudieran emular el lenguaje natural. Posteriormente se mejoraron los resultados gracias al uso perfeccionado de los métodos estadísticos y probabilísticos. Actualmente, gracias al DL se están realizando notables avances con la aparición de modelos pre-entrenados de Google [DCLT18].

La estructura de NLP se divide en dos subgrupos: Comprensión del Lenguaje Natural y Generación del Lenguaje Natural, que realiza tareas de comprender y generar el texto. Esta composición se puede ver representada en la Figura 2.3.

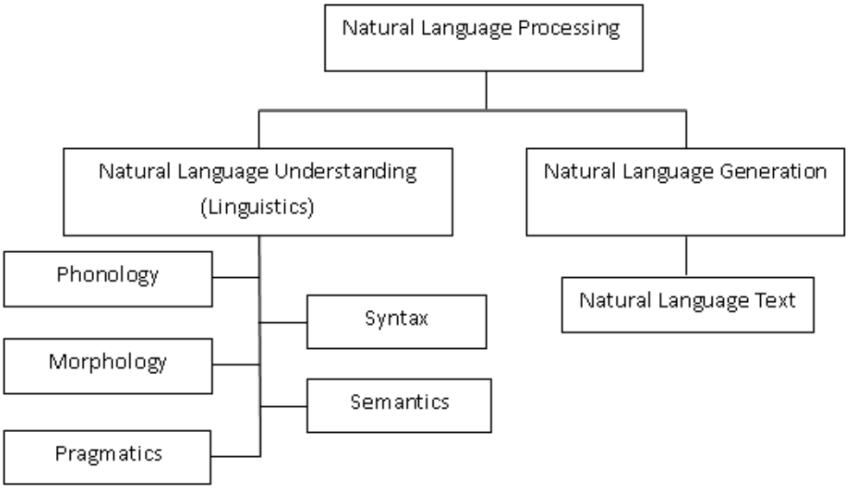


Figura 2.3: Estructura de NLP

La lingüística es la ciencia del lenguaje que analiza la fonología, la morfología, la semántica, la pragmática y la sintaxis. El lenguaje es un conjunto de oraciones y secuencias de palabras construidas gracias a reglas gramaticales.

El lenguaje se divide en dos grupos, el lenguaje natural y el formal. El lenguaje natural es aquel que nace de un grupo de hablantes al mantener una comunicación verbal. Esta relacionado con cada civilización y va cambiando según el uso que dan sus hablantes. Por el contrario, los lenguajes formales han sido diseñados para una área de aplicación en concreto, normalmente enfocados en el sector de las ciencias y se definen por ser precisos y libres de cualquier ambigüedad.

Ejemplos de lenguas formales son:

- Lenguaje matemático y lógico
- Lenguajes de programación (C, Java, Fortran, ...)
- Lenguaje de la música

La área de NLP está enlazada con diversas teorías y técnicas que tratan de solucionar problemas de comunicación entre la sociedad y las máquinas. Un problema principal que

se puede presentar a nivel sintáctico es la ambigüedad, pero se puede resolver con un tratamiento exhaustivo para conocer todos los puntos clase de la frase.

Los elementos básicos de cualquier técnica de NLP son [KKKS17]:

- **Análisis fonológico**: consiste en el estudio de la ordenación sistemática del sonido.
- **Análisis morfológico o "léxico"**: trata de analizar las palabras que integran sentencias para obtener sus raíces, unidades léxicas compuestas y rasgos flexivos con la finalidad de dividir el texto en *tokens*, propios del lenguaje original en el que aparece. Otro fin que tiene es el etiquetado morfológico. El etiquetado permite clasificar los elementos léxicos identificados según su categoría gramatical en etiquetas morfológicas. Ejemplos de tipos de etiquetas pueden ser de género, número o tiempo.
- **Análisis sintáctico**: tiene el fin de analizar de forma sintáctica si una secuencia de *tokens* se ajusta a una estructura gramatical.
- **Análisis semántico**: tiene el objetivo de determinar el significado de las oraciones.
- **Análisis del discurso**: analiza los diferentes discursos que puede haber en una oración, buscándolos un significado al relacionar las conexiones entre las sentencias que los componen.
- **Análisis pragmático**: realiza un estudio del contexto que rodea al texto analizado y en cómo éste afecta en su significado.

Gracias a estos tipos de análisis, las tareas de NLP dividen el lenguaje en secciones esenciales o *tokens* con el fin de entender las conexiones entre dichos *tokens* y explorar cómo es su funcionamiento para crear un significado juntos. Estas tareas de análisis requieren de recursos [KKKS17] como:

- **Categorización del contenido**: consiste en una sinopsis del contenido del documento basado en la lingüística.
- **Descubrimiento y modelado de temas o del inglés *Topic Modeling***: analiza la relación que tienen las palabras y frases con el fin de clasificar automáticamente el grupo de palabras que caracteriza mejor a un documento.
- **Extracción contextual**: extrae de manera automática datos estructurados de fuentes basadas en textos.
- **Clasificación de texto**: tiene la finalidad de extraer los patrones existentes entre las instancias compuestas por variables independientes y su relación con el objetivo. Una vez que el modelo de clasificación se ha analizado, se adjudican las etiquetas a las instancias basándose en los patrones detectados durante el entrenamiento.
- **Análisis de sentimiento**: detección de pensamientos subjetivas y sentimientos en grandes volúmenes de texto.
- **Conversión de habla a texto y de texto a habla o del inglés *Speech Recognition***
- **Resumen de documentos**

- **Traducción automática**
- **Reconocimiento de Entidades Nombradas**
- **Chatbots**
- **Interfaces en lenguaje natural**
- **Chatbots**
- **Reconocimiento óptico de caracteres, del inglés Optical Character Recognition**

Por último, las técnicas de preprocesamiento de NLP están enfocadas en el limpiar los documentos originales para poder aplicar las diferentes tareas que ocupan la minería de texto. En el contexto del NLP [ara20], se utilizan términos que es necesario entenderlos:

- **Corpus**: se trata del grupo de documentos sobre el que se aplica el análisis.
- **Token**: conjunto de palabras o fragmentos de caracteres que representan al texto. Los tokens se pueden agrupar de varias maneras:
  - Bolsa de Palabras: cuenta el número de veces que se repite cada palabra. Por ejemplo, las conjugaciones de verbos o similares se agrupan en lemas. A continuación, se comparan con la base de datos del bot y se buscan similitudes.
  - Expresiones regulares: funciona mediante la igualdad entre una frase y una plantilla predefinida en la base de datos del bot.
  - Reconocimiento de Entidades: diferencia las palabras claves entre nombres de personas, lugares, grupos, localizaciones, fechas etc. El objetivo que tiene este método es dar una respuesta más precisa según las entidades que extraiga.
  - Análisis sintáctico y gramático: las palabras de la frase se separan y vectorizan, y mediante algoritmos de machine learning, se pueden obtener predicciones sobre el significado de la frase.
- **Documento**: representación escrita de un pensamiento, concepto o diálogo.
- **Vocabulario**: tokens obtenidos de "tokenizar" el *corpus* completo.

El principal fin es obtener un documento y realizar sobre el un análisis y transformaciones básicas con el fin de obtener cualidades útiles para hacer alguna tarea analítica posterior más importante.

2.4. Funcionamiento de un Chatbot

Como se describe anteriormente los asistentes conversacionales basan su funcionamiento en NLP, y siguen los siguientes pasos:

1. Se recibe el mensaje del usuario.
2. Se obtienen las entidades o los intentos.

3. Se buscan respuestas candidatas en la base de datos según las cualidades recogidas en los intentos, las entidades y el contexto.
4. Se crea una lista con las respuestas escogidas.
5. Se selecciona la respuesta a base del contexto del mensaje y la lista de resultados.
6. Finalmente se obtiene una respuesta.

Esta información se puede ver esquematizada en la Figura 2.4.

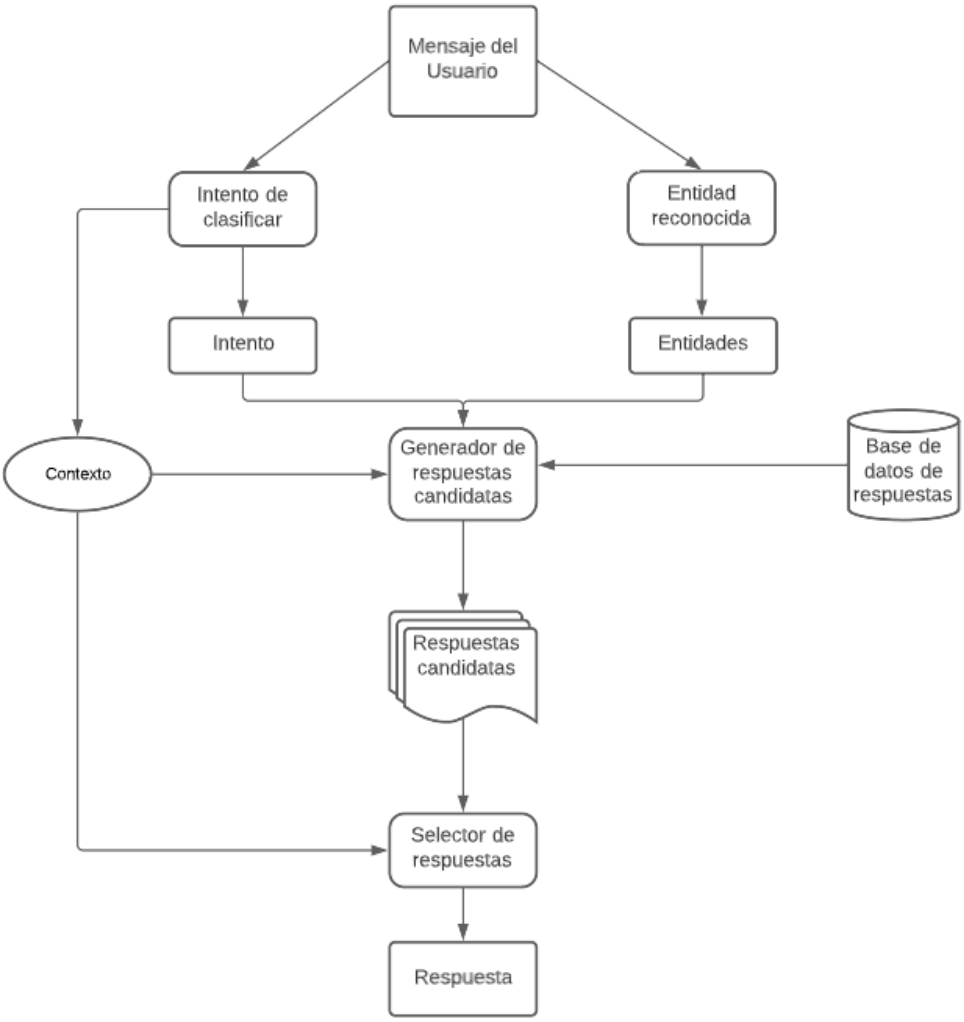


Figura 2.4: Estructura del Chatbot [RAMI17]

2.5. Frameworks y Librerías Utilizadas

En este apartado se va a explicar los frameworks y librerías que se han utilizado para programar el chatbot, además del framework de Rasa Stack.

2.5.1. Herramientas Utilizadas

- **Telegram - The BotFather** [Ló20] ofrece un soporte de open-source, asegurando solidez y sencillez a la hora de crear y usar bots. Si se quiere crear un bot en Telegram, el primer paso es iniciar un chat con BotFather, un bot específico para crear asistentes. Situándose en la conversación, preguntará al usuario sobre las cualidades que se aplican al bot que se está creando (imagen del perfil, nombre, descripción,...), además se proporcionará al usuario información imprescindible como el Token o el enlace directo al bot. Esta clave se trata de una cadena de caracteres que permite su control mediante peticiones HTTP a la API.
- **Github** [Git22] brinda la posibilidad de crear una página web, ya que dispone de la herramienta de *pages*, con la que deja subir un repositorio a una url. Funcionalidad sencilla que con un simple index.html ya se puede testear el chatbot en internet.
- **BotFront** [Joa21] es open source que se ha utilizado para dar formato a la vista web del Chatbot, gracias al código dejan en su página web de Github. Simplemente con copiar el script en el index.html y personalizar algunos datos, ya se puede conectar el proyecto con una página web.
- **Spacy** [CC20] es una librería que se encuentra integrada internamente en Rasa, ya que se utiliza para analizar el lenguaje natural del usuario. A diferencia con otras librerías, proporciona resultados más eficaces y rápidos, con un 90 %. Con ella, se realiza una tokenización de los mensajes, ofrece de vectores pre-entrenados y soporte multidioma.
- **TensorFlow** [AC21] es open source que utiliza Rasa de manera interna también para realizar procesos de aprendizaje automático. A su vez opera sobre arrays de datos que disponen de más de una dimensión, gracias a las redes neuronales llamadas *tensores*. Al entrenar el chatbot se hace uso de esta librería.A su vez también realiza una tokenización sobre el lenguaje natural del usuario.
- **Rasa_sdk** [CC20] forma parte del conjunto de librerías que hace posible la programación de *actions.py* con las clases Action. Estas clases implementan funciones que filtran los mensajes del usuario y devuelven una respuesta concreta mediante las funciones de *name()* y *run()*. Para obtener un resultado hace uso de elementos como el Tracker, el Dispatcher y *domain.yml*
- **Fuzzywuzzy** [GLPC21] es otra biblioteca que se utiliza para comparar dos cadenas de caracteres, como por ejemplo palabras u oraciones. El resultado devuelto es un número que equivale a una proporción entre el 0 al 100 de la semejanza entre estos dos elementos. Si el valor que torna está más próximo al 0, entonces su equivalencia es mínima, pero en cambio si es más cercano al 100, su igualdad es mayor.
- **JSON** [Ali16] es una librería que se utiliza en este proyecto con el fin de obtener datos de ficheros con formato *.json* para su posterior análisis. Lo beneficioso de esta librería es cómo estructura los datos y lo fácil que resulta acceder a los elementos.

A continuación en la Figura 2.5 se muestra la relación entre cada de una de las herramientas explicadas:

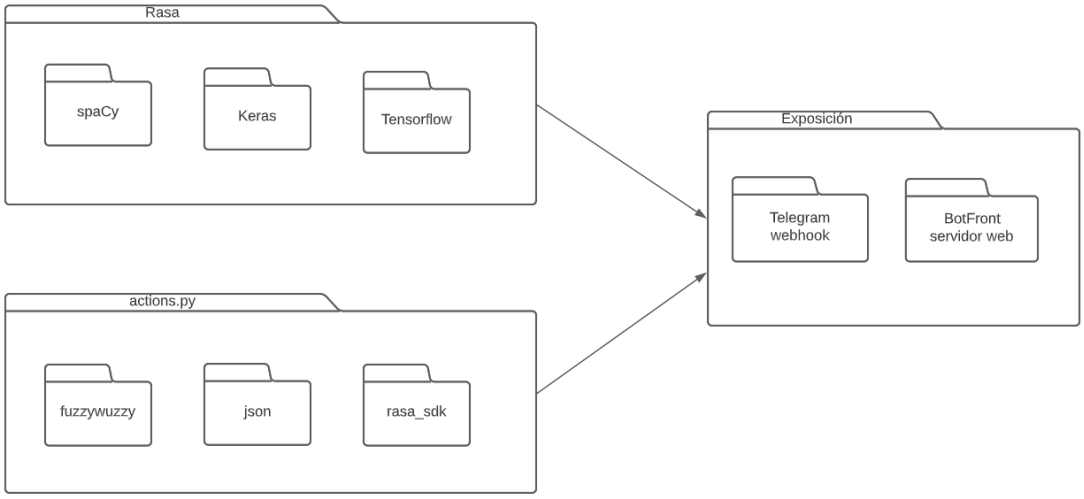


Figura 2.5: Esquema de paquetes

2.6. Sede Electrónica

La Sede Electrónica es la página web donde se va encontrar aplicado el chatbot [Nac22], específicamente en los apartados de extranjería, pasaporte y DNI. La Sede Electrónica pertenece al conjunto de servicios nacionales de España, y pone a la disposición de los ciudadanos varios procesos para gestionar trámites legales.

En este caso los apartados son los siguientes:

- Procesos selectivos
- DNI
- Pasaporte
- Antecendentes policiales
- Extranjería
- Seguridad privada
- Registros electrónicos
- Documentación para viajar
- Quejas y sugerencias

Luego cada sección tiene subapartados y enlaces que redirigen a los formularios o las webs donde se pueden realizar estos trámites. Los trámites actualmente se gestionan de manera presencial, pero como objetivo futuro tienen planeado dar el suficiente soporte web para realizar la mayoría de manera online. Un ejemplo de la apariencia que tiene la página inicial de la Sede Electrónica se ve en la Figura 2.6.

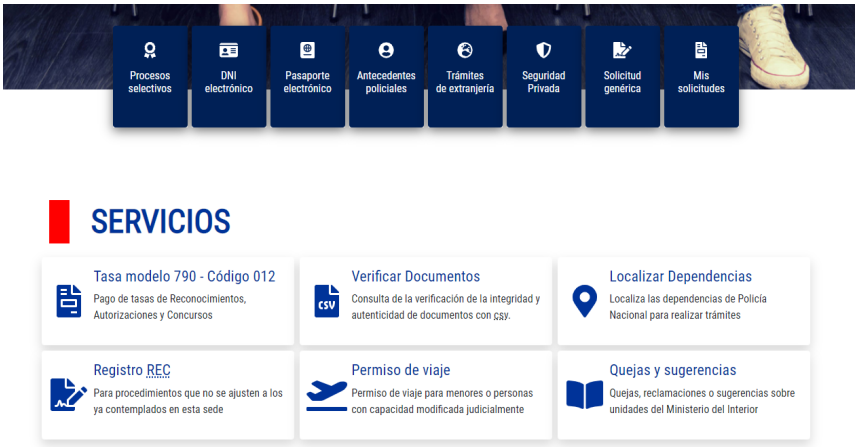


Figura 2.6: Página inicial

2.6.1. Apartado Extranjería

Este proyecto está centrado justo en esta sección, para poder responder preguntas sencillas y teóricas que puedan surgir al usuario a la hora de querer solicitar algún trámite.

Los procedimientos disponibles en el apartado de Extranjería se pueden ver en la Figura 2.7. Al seleccionar un apartado de los que se han mostrado anteriormente, te redirige a una explicación sobre el trámite. También muestra enlaces de interés para el usuario como donde consultar las tasas o la oficina de extranjería o comisaría de policía más cercana.

De todas estás páginas se ha extraído la información para crear las respuestas a las preguntas que puede recibir el chatbot, por ejemplo "¿Qué papeles necesito para solicitar la cédula?" o "¿Cuánto cuesta solicitar la tarjeta de residencia?". En la Figura 2.8 se ve un ejemplo de todos los datos que se pueden encontrar por cada trámite.

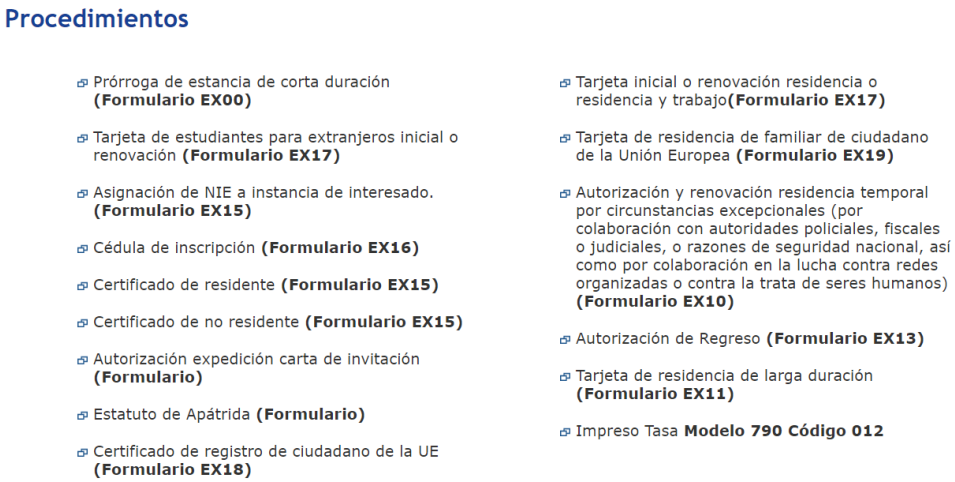


Figura 2.7: Procedimientos de Extranjería

Figura 2.8: Ejemplo de Trámite

2.6.2. Renovación y Solicitud de Pasaporte o DNI

Para acceder a estos trámites se debe de acceder primero a los procesos de DNI o Pasaporte electrónico. Se puede localizar en la página una sección con la siguiente información mostrada en la Figura 2.9. Cuando se hace click en "Cita Previa para DNI o Pasaporte", se redirige a una página donde el usuario debe rellenar los campos obligatorios con sus datos personales, elegir la fecha y el centro donde desea pedir cita.

De estos trámites también se responde a que documentación es necesaria o el precio para realizar la renovación o solicitud del DNI o pasaporte. En la Figura 2.10 se muestra la página de inicio para solicitar o renovar el DNI y el pasaporte, de donde se ha extraído la información para redactar el procedimiento de estos trámites.

Figura 2.9: Acceso a los trámites

Figura 2.10: Petición de cita

Capítulo 3

Estado del Arte

Hoy en día el uso de asistentes virtuales está en auge y cada vez hay más empresas interesadas en incorporar estos mecanismos para atender a sus clientes. Por ello, hay que conocer este software y saber que tipo de chatbot es el más apropiado para el negocio. En la sección 3.1 se muestran diferentes chatbots divididos en tres sectores que ya están siendo utilizados de manera cotidiana. En el punto 3.2 se muestran las diferentes aplicaciones que permiten el desarrollo de un chatbot. El el último punto 3.3, después de haber explicado las herramientas de diseño de chatbots, se describen otros proyectos desarrollados con Rasa Stack, aplicación que posteriormente se explicará su funcionamiento.

3.1. Contexto Digital

Como se ha citado anteriormente, debido a la crisis y posterior confinamiento por el COVID-19, la digitalización es importante para el día a día de las empresas, ya que aportan al usuario la capacidad de resolver sus dudas de forma remota. Por ello proponen ventajas tecnológicas, como los chatbots, para agilizar sus comercios y trámites.

3.1.1. Chatbots en el Sector de la Medicina

El primer asistente a tratar es OneRemission [kee21a], chatbot dirigido a ayudar a los pacientes que luchan contra el cáncer. Las tecnologías que utiliza son Node.js, MongoDB, Wit.ai, BotPress y WebSocket. El resultado es un soporte que ofrece planes de alimentación, ejercicios y prácticas seleccionadas por profesionales de la medicina integrativa. De esta manera, pueden preguntar dudas simples y prescindir por tanto de la supervisión de un médico. Sin embargo, si se llegará a necesitar la ayuda de un especialista, OneRemission ofrece sesiones con oncólogos en línea las 24 horas del día durante toda la semana.

El siguiente es Youper [kee21b], asistente dirigido a mejorar la salud psicológica de los pacientes. Está desarrollado con 13 tecnologías y servicios, incluidas HTML5, Google Analytics, jQuery, SPF, Google Apps for Business y IPv6 [cru22]. El resultado es una aplicación que ofrece terapia online por chat, sesiones virtuales con psicólogos especializados y planes de medicación en caso de necesitar.

Otro chatbot a destacar es Babylon Health [Bab21], idea que surgió con el fin de detectar la enfermedad que podían padecer los usuarios según sus síntomas. El desarrollo

consistió en crear un conjunto de herramientas, diseñado en torno al cerebro de un médico, utilizando IA. El fruto de este proyecto es un asistente conversacional, en el que los pacientes informan, por voz o mensaje, del estado en el que se encuentran. En paralelo, los médicos estudian los casos para recetar medicamentos o derivar a los pacientes a un especialista.

El posterior asistente virtual, Florence [oT17], tiene la función de simular el comportamiento de una enfermera personal. El desarrollo se basa en software desarrollado con algoritmos de aprendizaje automático, incluido el NLP; además dispone de servidores en las plataformas de Facebook Messenger y Skype. El asistente final es capaz de recordar a los pacientes que tomen sus medicamentos y realizar un esquema de los días y horas a las que deben de tomarlos.

Otro asistente que hay que citar es Healthily [Hea22], aplicación que tiene el labor de informar sobre temas de medicina basados en fuentes de alta fiabilidad. El resultado final es una aplicación disponible en iOS, Android, Facebook Messenger, Slack, KIK, Telegram y versión web que permite al paciente decidir de manera más eficiente sobre temas de su salud. También funciona como intermediara para dar información sobre servicios médicos en línea.

3.1.2. Chatbots en el Sector de la Educación

El primer chatbot que se va describir es CourseQ [GBFAMC18], el cual dispone de funcionalidades como recordar fechas de entrega o mostrar los horarios de los estudiantes y profesores. Su desarrollo se realizó gracias a los avances de en el campo inteligencia artificial, concretamente en las ramas del ML, Big Data (recopilación y análisis de datos) y NLP. El aspecto final es el de una aplicación capaz de desempeñar las funcionalidades anteriormente mencionadas y con la capacidad de ser integrada en los *Learning Management System* (LMS) o *Virtual Learning Environment* (VLE), por ejemplo Moodle.

El siguiente en describir es el asistente Ani [GBFAMC18], que tiene el objetivo de realizar tutorías particulares para facilitar el compromiso y la responsabilidad de los alumnos. El desarrollo es similar a los anteriores ya que está programado a partir de algoritmos de aprendizaje automático con el fin de obtener resultados personalizados para casa usuario, llegando incluso a poder sustituir acciones de profesores. Finalmente, se consigue un asistente que aporta elementos que motivan y evalúan a los estudiantes, además de incluir un curso para aprender inglés.

Otro chatbot a destacar es Botter [GBFAMC18], que a diferencia de los demás bots, es un robot físico que ayuda a los alumnos en la Universidad Oberta, situada en Cataluña. El funcionamiento de Botter se basa en tecnología cognitiva para poder aprender a partir del comportamiento demostrado por el alumnos. El fruto final es un asistente físico con interacciones a base de señales de luz y mensajes sonoros (como frases de motivación) o movimientos, con el objetivo de servir de ayuda a los estudiantes a organizar sus tareas.

El posterior asistente virtual es el bot desarrollado en la Universidad CEU Cardenal Herrera [CIO17].El objetivo es conseguir acompañar al alumno en su curso académico y responder sus preguntas de manera instantánea y sin interrupciones. Su implementación se basa en la herramienta de Microsoft Azure, y actualmente soluciona dudas administrativas, pero se espera que en el futuro pueda determinar la conducta del alumno con el fin de aconsejarle durante su periodo académico.

Por último, los robots Pepper y Nao [Rob22] están centrados en la educación universitaria. Se pueden programar por completo con una interfaz gráfica o directamente con el *Software Development Kit* (SDK) y herramientas de codificación como Python y C++. Entre sus funciones se encuentran dar y dinamizar las clases y evaluar actividades.

3.1.3. Chatbots en el Sector Bancario y Seguros

Comenzando por el asistente Eno [One22] fundado por la empresa Capital One, es un chatbot bancario y puramente "conversacional", lo que significa que depende casi por completo de IA para generar sus respuestas. Esto significa que Eno analiza cada mensaje que recibe para encontrar la mejor respuesta. El aspecto final es un asistente que responde preguntas sobre el saldo, movimientos recientes, fechas de vencimiento y límites de la tarjeta.

El segundo asistente se trata de ABIE [NEW18], que tiene el objetivo de guiar a los agentes a través del trámite de venta, permitiéndoles acceder a información y a documentos. En relación con su implementación, cuenta con algoritmos de Inteligencia Artificial y el soporte de la empresa Allstate. Respecto a su aspecto final, es un asistente conversacional que ofrece datos y productos de manera personalizada según el cliente. Además maneja más de 25.000 consultas por mes de 12.000 agentes [dit18], incrementando a un ritmo del 10 %. Esto es debido a que los agentes se están concienciando de que gracias a este asistente obtienen un mejor resultado de manera más veloz.

El siguiente chatbot es BotOferta [dit18], que facilita datos sobre un programa de descuentos en ocio y compras en España. Para ello, los usuarios pueden realizar preguntas por mensaje directo o por voz y obtendrán respuestas sobre los descuentos en determinados ámbitos o lugares. El lenguaje que utiliza Botoferta es cercano y ameno, con el fin del que el cliente se sienta cómodo en la conversación. Por otro lado, este servicio ya está disponible desde el año pasado en aplicaciones como Facebook Messenger, y se puede acceder a él mediante la propia web o por la App de imaginBank.

El cuarto asistente trata seguros de viaje y está creado por la empresa ARAG [ARA22]. Consiste en un asistente de seguros de viaje que muestra información sobre los productos de la compañía con el fin de ayudar a los clientes con sus viajes. Esta herramienta permite que las empresas puedan hablar, a través de la mensajería instantánea de Facebook, con los consumidores sin necesidad de que haya una persona detrás de la cuenta. El vocabulario que emplea divertido, sencillo y ameno, para que el usuario se sienta más confiado al planificar su desplazamiento.

El último asistente es Blue [BBV22], el nuevo asistente virtual del banco BBVA. El objetivo es ayudar a los clientes y consultar información sobre su cuenta. La implementación se basa en algoritmos de NLP y de ML, o aprendizaje automático. De esta manera,el usuario interactúa con este chatbot no solo vía texto escrito, sino también mediante el uso de la voz. El resultante asistente se encuentra en la parte superior derecha de la ventana de navegación de la aplicación de BBVA.

3.2. Frameworks de Desarrollo de Chatbots

Los frameworks son un tipo de herramientas con una series de archivos y pautas que se utilizan para implementar proyectos con una estructura y metodología con el fin de

simplificar la creación de una solución. A continuación se presentarán varios frameworks de desarrollo de chatbots, los cuales aportan ya una estructura, un vocabulario y un entrenamiento necesarios para implementar el asistente.

- **AIML** [RAMI17] define cómo un robot debe responder a una pregunta con patrones y plantillas de elementos. Un patrón representa al usuario que pregunta, y la plantilla define la respuesta del bot. Utiliza lenguaje basado en scripts, con el qué divide las interacciones entre categorías, plantillas y patrones .

- **DialogFlow** [Jan17] funciona como una extensión de Google, que a la vez hace de soporte. Al igual que los demás frameworks es capaz de comprender el lenguaje natural y proporciona al programador ya una estructura intuitiva para el desarrollo del chatbot. Destaca por la gran diversidad de interfaces de conversación en los que se puede desplegar (Google home, Google Assistant, wereables, teléfonos, coches). Tiene soporte para más de 14 idiomas y es capaz de resolver abreviaturas y funcionar con faltas de ortografía.

- **IBM Watson** [PA+19] fue lanzada en el año 2010. Dicha plataforma está construida en base a una red neural utilizando para su entrenamiento un billón de palabras de la Wikipedia. Además cuenta con la opción de integrar varias extensiones de IBM para realizar la aplicación. Gracias a esto, se puede desplegar en una página web, en una aplicación móvil, en el teléfono, en canales de mensajería y en herramientas de servicio al cliente. Proporciona un SDK para los desarrolladores que se puede utilizar en Java, Python e iOS. Es un herramienta muy utilizada por empresas y desarrolladores.

- **Rasa** [CC20] es un asistente proporcionado por una empresa de software centrada en Berlín. Dicha herramienta tiene el gran factor de que es código abierto y en Python. Su framework está instalado localmente, eso significa que aunque el software deje de mantenerse, el chatbot seguirá funcionando. El framework de Rasa consta de dos partes, Rasa NLU y RASA Core. Rasa NLU se encarga del lenguaje natural, para ello se nutre de los idiomas disponibles en la librería spaCy con el objetivo de procesar el mensaje y posteriormente transformarlo en datos con una estructura concreta. El segundo componente, Rasa Core, se encarga de la gestión del diálogo y decide qué acciones tomar en cada momento haciendo uso de un modelo de ML creado con Tensorflow y Keras.

- **Flow XO** [GS+19] realiza chatbots online y además no es necesario un conocimiento previo de programación. Su funcionamiento se basa en crear estructuras por cajas o acciones en una ventana, para que el chatbot siga un flow. Todo flow comienza con un trigger(disparador) y posteriormente tiene una secuencia de acciones que permiten la conversación entre el chatbot y el usuario. El disparador se lanza cuando se detectan palabras o frases claves por parte del usuario, y las acciones tienen la función de responder preguntas del usuario.

- **Wit.ai** [HD16] es una herramienta online que interpreta las interacciones que le llegan y las clasifica en diferentes entidades previamente definidas. Las entidades son palabras claves que detecta Wit con el objetivo de devolver al usuario la respuesta correspondiente a la entidad reconocida. Pero quizás no reconozca ninguna entidad y tenga que fiarse entonces de la intención de mensaje (*intent*) para poder decidir cómo actuar. Es una herramienta muy utilizada y sencilla, que funciona bajo la supervisión

de Facebook. Esto significa que para realizar un chatbot con Wit.ai, primero se debe de contar con una cuenta de Facebook.

3.3. Asistentes Conversacionales Realizados con Rasa Stack

- **Rasa framework: Análisis e implementación de un chatbot** [CC20] tenía principal objetivo de analizar el framework de Rasa Stack. Para realizar este estudio se decidió crear un asistente con el fin de que respondiera preguntas sencillas sobre un evento de empresa.

  Por tanto, los usuarios pueden realizar las siguientes acciones:

  - Consultar el lugar del evento, fecha y hora.
  - Notificar la ubicación del evento gracias a la herramienta Google Maps.
  - Preguntar sobre las actividades que se van a hacer en cada sala durante el evento.
  - Filtrar las exposiciones dependiendo del título, fecha o ponente.
  - Examinar el tiempo que hará en el evento.

  A la hora de desarrollar el proyecto, también se estudió todas las herramientas con las que se podían realizar asistentes y finalmente,se decidió de utilizar la herramienta de Rasa Stack. Por otro lado, también utilizó los servidores de Telegram y Google Assitant para disponer del asistente de manera online.

- **Desarrollo de un Chatbot para Público Infantil para Clasificar Sonidos del Cielo** [PG+20] tiene un alto nivel de complejidad ya que cumple todos los siguientes objetivos:

  - Diseño y desarrollo de la base de datos del proyecto.
  - Diseño y desarrollo de la API del trabajo.
  - Fusión de la API y la base de datos
  - Programar los scripts destinados a la inserción automática de nuevas detecciones de meteoritos.
  - Integración de la API en una versión preliminar del chatbot con el fin de probar todas las funcionalidades.

  Para poder realizar este asistente primero se realizó un estudio de las herramientas que desarrollan asistentes conversacionales. Además se hizo un análisis de más asistentes creados en diferentes sectores empresariales. La herramienta que se escogió para desarrollar fue la base de datos MongoDB, ya que es veloz y era perfecta para aprovechar los recursos del sistema por su balance entre la funcionalidad y rendimiento. Por otro lado, utiliza la API RESTful como intermediario entre la base de datos y el cliente. El servidor se programa en Node.js, con el fin de poder integrar aplicaciones escalables. A diferencia con el proyecto anterior, este diseña desde cero un archivo HTML y CSS para ejecutar de manera web el asistente. El resultado del diseño se muestra en la Figura 3.1.

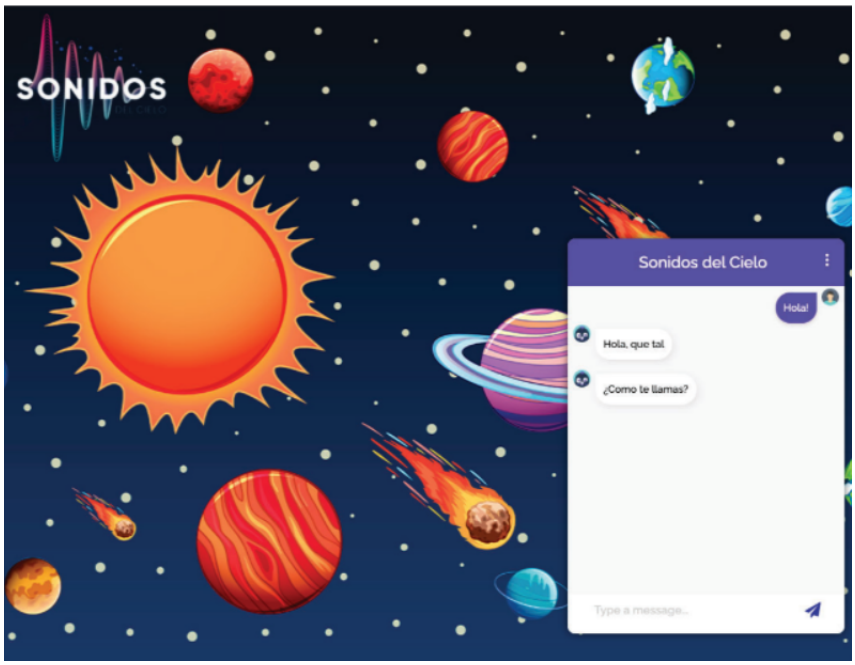


Figura 3.1: Resultado visual del asistente

- **Diseño e implementación de un Chatbot para el software de IDBOD** [PG+20] tiene el fin de desarrollar una prueba de concepto de un chatbot para el software IDbox de la empresa CIC Consulting Informático. Este asistente debe de responder a preguntas sencillas sobre atención al cliente. Por tanto el alcance del chatbot será el siguiente:

  - Entendimiento de saludos, despedidas, preguntas sobre la representación gráfica o una señal específica.
  - Requiere que sea capaz de tratar los fallos ortográficos de usuario.
  - También debe de responder a todas los mensajes del usuario, con el mensaje correspondiente o de feedback. Si no entiende el mensaje del usuario, sugiere la reformulación de la frase mostrando posibles peticiones que se pueden hacer.

  El desarrollo del asistente se basa en programar sobre un archivo de Rasa Stack. Además, se utiliza la librería de pandas para hacer un filtro de las dos bases de datos, archivos en formato **.csv** y de Matplotlib para poder responder con imágenes, generando un servidor local asociado a una imagen. A diferencia de los demás trabajos, se termina de enlazar el asistente con el servidor web de Rasa X.

Capítulo 4

Metodología y análisis del proyecto

En este capítulo se quiere mostrar el proceso que se ha seguido para desarrollar este proyecto, con el fin de conocer todos los elementos esenciales que lo componen. Primero en la sección 4.1 se concreta la metodología que se ha utilizado para entrenar al asistente, mostrando diferentes diagramas.Posteriormente en el apartado 4.2, se realiza una clasificación de las ventajas y desventajas de las herramientas actuales y se introduce a la herramienta que se ha utilizado para desarrollar este proyecto. En la sección 4.3, se incide en el funcionamiento interno de Rasa, haciendo un análisis de cómo se extrae las palabras claves de las frases y justificando el contenido de cada fichero del proyecto. A continuación, en la sección 4.4 se tratará el tema del despliegue de la herramienta Rasa. Y finalmente en el punto 4.5, se especifica como se realiza las conexiones entre Rasa, Telegram y la página web.

4.1. Arquitectura General del Trabajo

Para que el funcionamiento de un chatbot sea más eficiente se necesita entrenar al asistente. Por tanto es esencial mantener actualizada la base de datos e ir corrigiendo los posibles errores del asistente siguiendo los diagramas a continuación representados en la Figura 4.1.

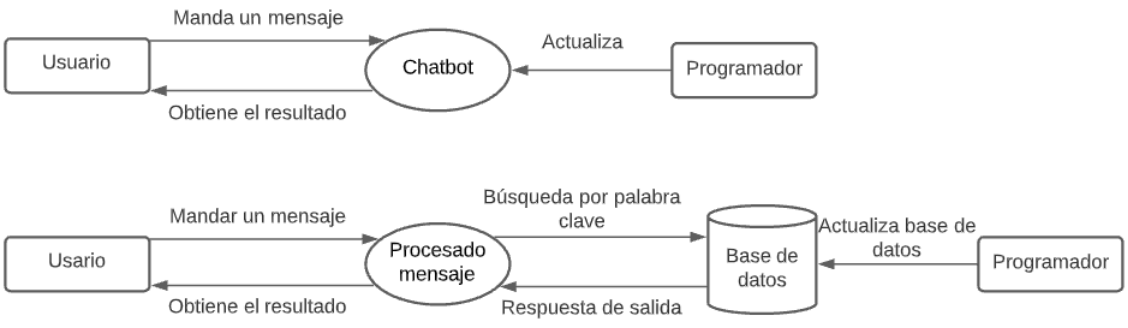


Figura 4.1: Data Flow [LBP+18]

El primer diagrama sigue los siguientes pasos:

1. El usuario manda un mensaje al asistente.
2. El programador actualiza el código del asistente según el mensaje que ha recibido
3. El asistente manda la respuesta al usuario.

El segundo diagrama sigue el siguiente patrón:

1. El usuario manda un mensaje al asistente.
2. Se realiza un procesado de mensaje, en el que se extraen las palabras claves de mensaje, para luego realizar una búsqueda selectiva en la base de datos.
3. Si esas palabras claves no están registradas, el programador actualiza la base de datos.
4. Una vez actualizada la base de datos con la respuesta correspondiente a las palabras claves recibidas, envía la respuesta de salida al usuario.

A su vez se realiza un filtro, representado en la Figura 4.2, para saber si la entrada que procesa el asistente es correcta o si figura en la base de datos.

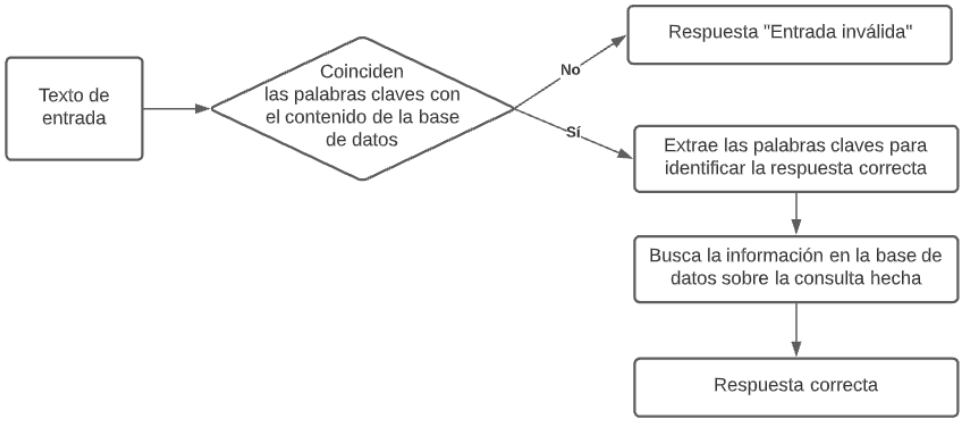


Figura 4.2: Comprobación de entrada

A la hora de analizar el tipo de conversación que van a seguir los usuarios con el asistente, se realizó un diagrama, que se encuentra en la Figura 4.3, que representa el flujo de acciones que se va llevar a cabo por parte de los dos roles. El color verde representa los hechos del chatbot y el color rojo los del usuario.

La recuadro de acción, se refiere a todos lo tipos de preguntas que puede realizar el usuario hacia el chatbot y obtener una respuesta acertada por parte de él. Según se acordó con la policía, el asistente debía de basar sus respuestas en información oficial de la sede electrónica, por ello a las preguntas que responde son sobre precios, documentación necesaria y guías sobre los trámites que se realizan en la web. A continuación se muestra un diagrama en la Figura 4.4 que representa el tipo de dudas que puede hacer el cliente y las diferentes observaciones que recibe.

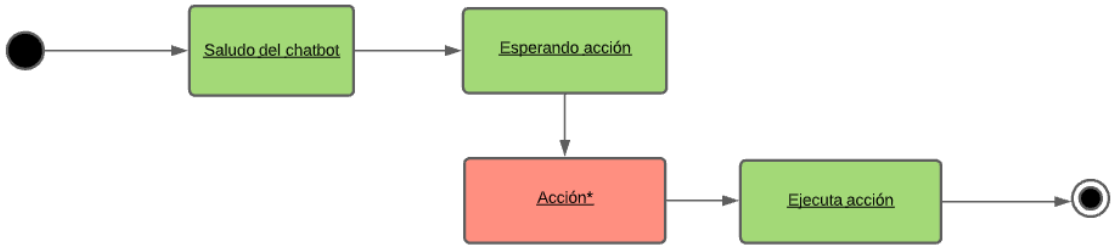


Figura 4.3: Diagrama de estados-Pregunta Usuario

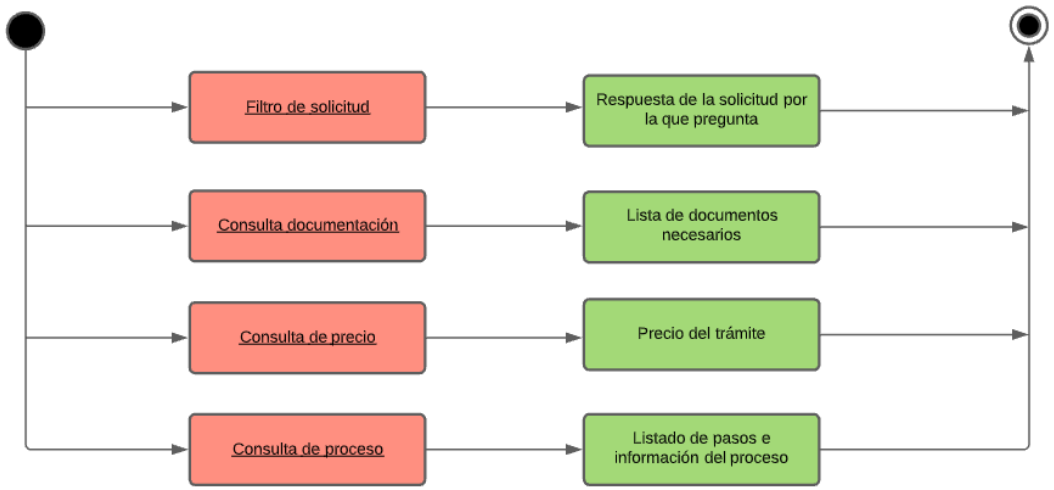


Figura 4.4: Tipo de preguntas y respuestas

Pero el usuario no es el único que puede interactuar con el chatbot, también se encuentra el perfil de administrador, llevado a cabo por un policía que se encargará del soporte y actualización del asistente. El esquema de casos de uso se puede ver en la Figura 4.5.

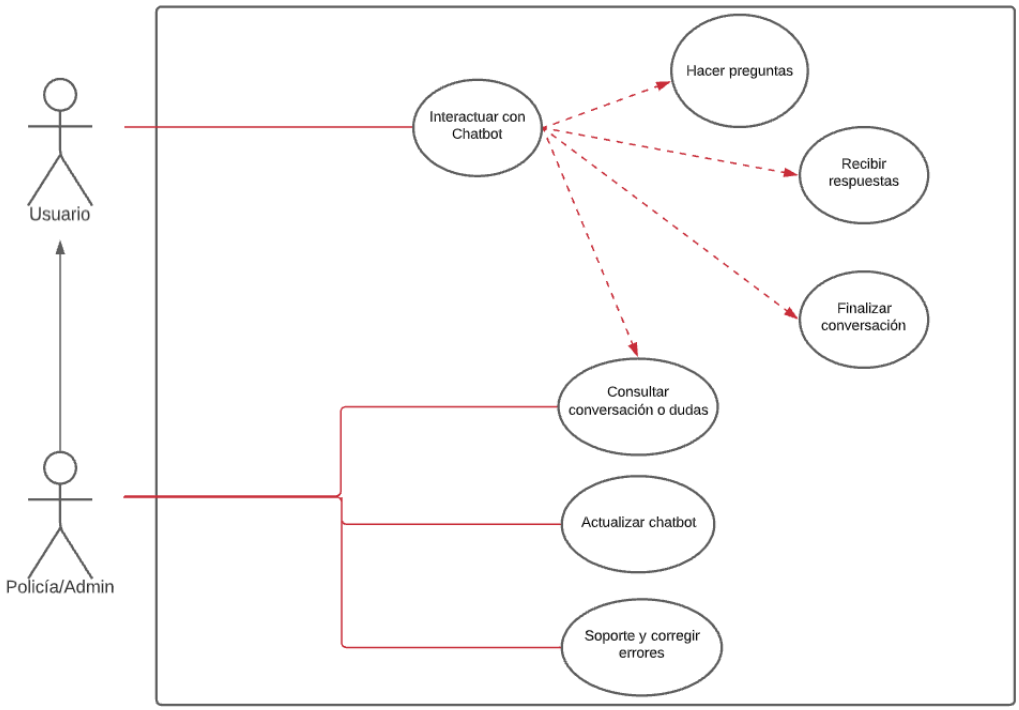


Figura 4.5: Casos de uso

4.2. Elección de Herramienta

Frente a todas la herramientas descritas anteriormente, se ha elegido la opción de desarrollar el chatbot con Rasa [VLRL19]. Es el framework que presenta mejores características, ya que tiene un gran soporte, documentación y además se puede diseñar chatbots más complejos. Las cualidades que se van a tener en cuenta son:

- Facilidad de aprendizaje: Este concepto se refiere al grado de éxito obtenido durante el aprendizaje en el transcurso del tiempo.
- Soporte de la comunidad: Se valora si el framework tiene una comunidad detrás dispuesta a ayudar.
- Documentación existente: Se valora la cantidad y calidad de información que existe tanto oficial como creada por la comunidad.
- Madurez tecnológica: Cuantifica 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 personalizar: Característica que permite a la tecnología adaptarse al proyecto de cada cliente.
- Actualización de software: Analiza si los desarrolladores introducen cambios y añaden funciones nuevas a la tecnología.

A continuación, se puede ver en la Tabla 4.1 la puntación (1-10) de diferentes cualidades esenciales que debe de tener todo framework.

	AIML	Dialogflow	IBM Watson	Rasa	Flow XO	Wit.ai
Aprendizaje(20 %)	6	7	7	6	7	6
Comunidad(20 %)	4	4	3	9	6.5	5
Documentación(20 %)	5	6.5	6	9	7	6.5
Madurez(15 %)	5	8	9	5	5	6
Personalizar(15 %)	7	6.5	7	5	7	5
Actualización(10 %)	7	7	8	8	6	5
TOTAL	5.5	6.38	7	7.4	6.5	5.65

Tabla 4.1: Tabla comparativa entre herramientas

Después de analizar esta tabla, el framework más completo es Rasa Stack. Por ello será la herramienta que se va a utilizar en este proyecto.

4.3. Funcionamiento de Rasa Stack

Como se ha especificado antes Rasa está dividida en dos principales secciones: la parte de NLU y del Core. Por tanto, a la hora de programar y formar el *dataset* es importante tener en mente el flujo que sigue dicha herramienta.

Las Figura 4.6 y 4.7 muestran el funcionamiento que toma Rasa Stack [PG21].

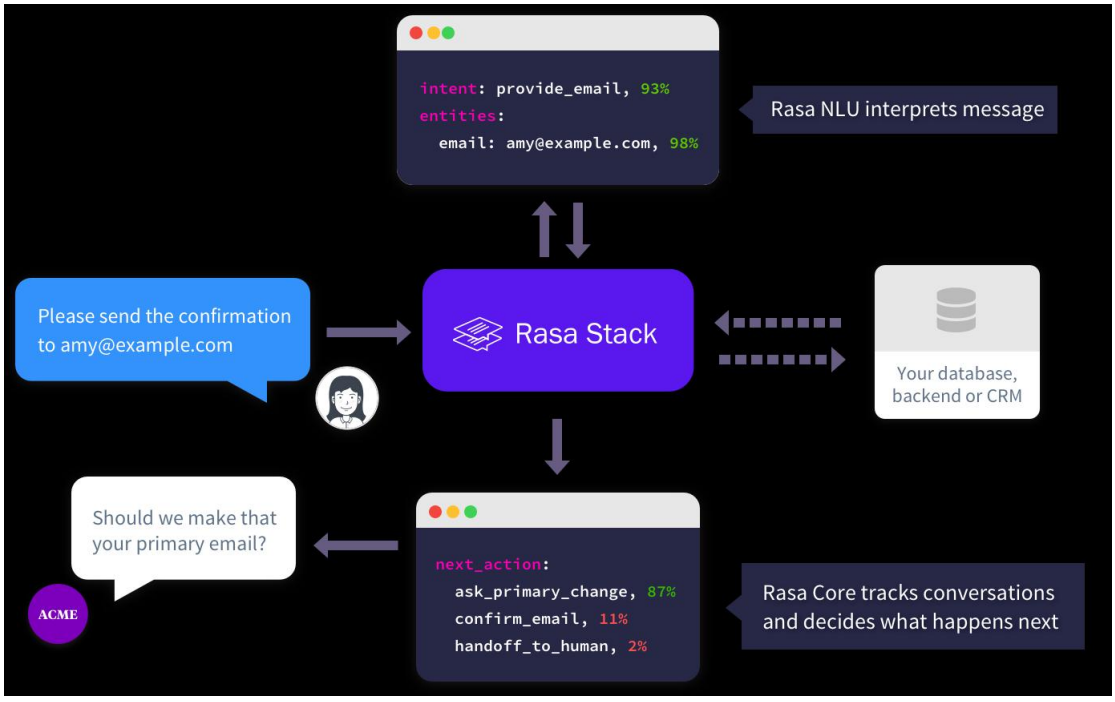


Figura 4.6: Clasificación de un mensaje

1. El trigger que da comienzo al flow se trata del envío de un mensaje a Rasa Stack, que hace de Broker u Orquestador y tiene la funcionalidad de organizar las conexiones entre las diferentes capas.

2. Posteriormente, la sección que entra en acción es la del NLU, descifrando el *intent* del usuario y tokenizando el mensaje, consiguiendo así las *entities* y los *slots* asociados. Cuando se realiza este paso, también se calcula un grado de confianza, que es un porcentaje basado en lo que entiende Rasa del mensaje. Si dicho cálculo es menor al 40 %, se muestra un mensaje de que no se ha entendido al usuario.

3. En paralelo, el componente Core se encarga de decidir que acción tomar según el punto de la conversación en que se encuentra el asistente con el usuario. Esta determinación de respuesta la toma gracias a las historias que se han establecido en su entrenamiento y al porcentaje calculado con la coincidencia con dichas historias definidas en stories.yml.

4. Finalmente se devuelve una respuesta al usuaria por el mismo canal que el de entrada.

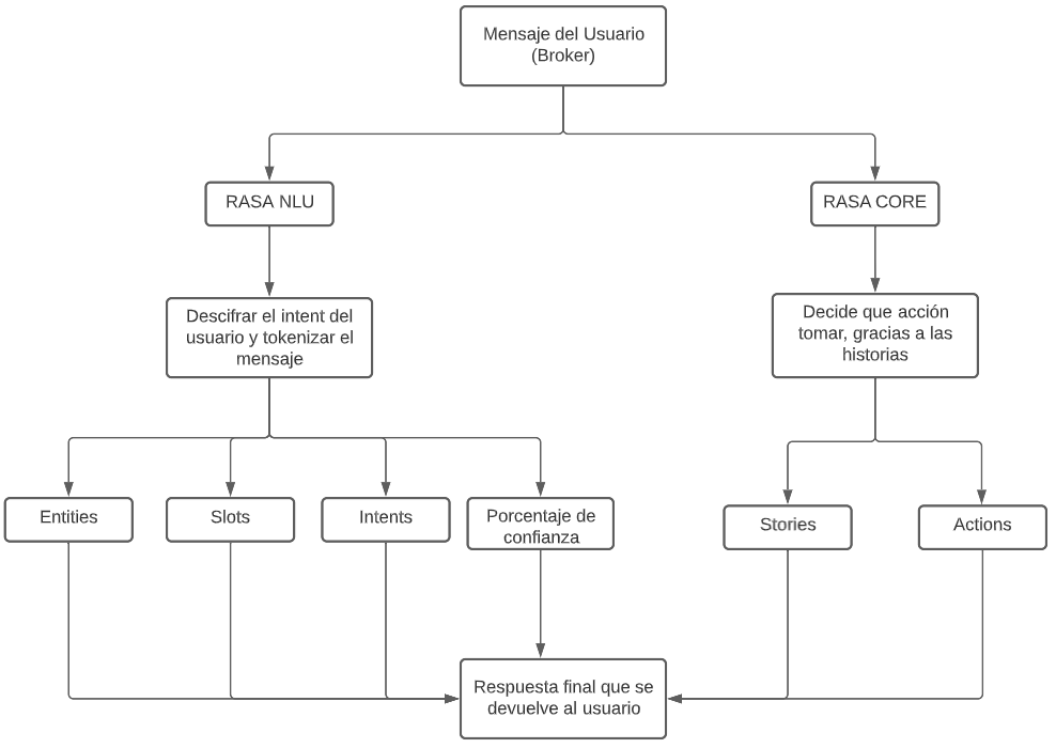


Figura 4.7: Esquema de pasos - Rasa Stack

4.3.1. Componentes de Rasa

Las entidades o *entities* son palabras claves imprescindibles para filtrar la intención o el *intent* de la pregunta que haga el usuario y saber que respuesta o *utterance* da el asistente. Además se pueden guardar los valores de las entidades en *slots* con el fin de acceder a ellos en las diversas funciones o *actions*. Todos estos componentes formarán parte de las estructuras *stories*, donde se crea un flujo de *intents* y *utterance*.

Si se quiere saber más sobre estos elementos se recomienda leer la información recopilada en el Anexo A para entender mejor el contenido de cada fichero del proyecto. A continuación se detallan su programación.

- Nlu.yml

  Aquí residen todas los *intents* con los que se ha entrenado el asistente. Cada *intent* tiene está formado por los mensajes más frecuentes definidos dentro de la Sede Electrónica. Los *intents* forman parte de Rasa NLU y es la manera que permite organizar y clasificar los mensajes enviados por el usuario.

  Los mensajes están divididos por la acción por la que pregunta el usuario:

  - Proceso
  - Precio
  - Documentación necesaria

  Y también por el tipo de trámite, ya que se ha abarcado todos los procesos del apartado de extranjería y renovación/solicitud del pasaporte y DNI.

  Como se puede apreciar en el Algoritmo 4.1 hay palabras seleccionadas entre corchetes, esto quiere decir que el valor que escriba el usuario será guardado y posteriormente analizado en la sección de actions.py. Cuando se estudie este componente se explicará mejor el tratamiento por el que pasa este dato (nombre_tramite).

- Domain.yml

  Sección dedicada a las respuestas posibles que da el asistente, llamadas *utterances*. Como los *intents*, también pertenece a Rasa NLU y se utiliza para el entrenamiento del chatbot.

  Su clasificación está basada por la pregunta a la que responde, y además en este fichero se tiene en cuenta los *intents* definidos en nlu.yml, las acciones que están en actions.py y el tipo de entidades que se guardan. Esta especificación se debe a que también son elementos importantes a la hora de responder al usuario, ya que las acciones también son las encargadas de responder al cliente.

  El comienzo del fichero empieza con la lista de todos los *intents* como se puede ver en el Algoritmo 4.2.

  Posteriormente hay que especificar las entidades como en el Algoritmo 4.3. En este proyecto solo se ha utilizado una entidad tipo texto que guarda el trámite por el que pregunta el usuario.

  A continuación, se escriben todas las respuestas que da el asistente, excepto las que provengan de las acciones. Todas las *utterances* deben de ser nombradas con un nombre único y seguidas por el mensaje que da el chatbot. El Algoritmo 4.4 es un ejemplo de la estructura que hay seguir para declarar las *utterances*.

  Y finalmente se nombran las acciones que se encuentran en el fichero de actions.py como se muestra en el Algoritmo 4.5.

Algoritmo 4.1: Contenido de nlu.yml

```
- intent: tarjetas_solas
  examples: |
    - Tarjeta de [estudiante](nombre_tramite)
    - Tarjeta residencia [familiar de ciudadano UE](nombre_tramite)
    - Tarjeta residencia [larga duración](nombre_tramite)
    - Tarjeta de residencia y [trabajo](nombre_tramite)
    - Tarjeta [familiar](nombre_tramite)

- intent: tramites-solos
  examples: |
    - Asignar [NIE](nombre_tramite)
    - Autorización y renovación la residencia [temporal](nombre_tramite)
    - Autorizacion de [regreso](nombre_tramite)
    - Registrar [ciudadano de UE](nombre_tramite)
```

Algoritmo 4.2: Lista de *intents*

```
intents:
  - start
  - despedir
  - confirmar
  - negar
  - preguntas-infoSede
  - preguntas-gestiones
  - preguntas-certificados
  - preguntas-firmaElec
  - preguntas-extranjeria
  - preguntas-reagrupacion
  - dudas-NIE-TIE
  - ...
```

Algoritmo 4.3: Especificación de la entidad

```
entities:
    - nombre_tramite

slots:
    nombre_tramite:
        type: text
        initial_value: NULL
        auto_fill: true
        influence_conversation: true
```

Algoritmo 4.4: Ejemplos de *utterances*

```
  utter_extranjeria:
  - text: "Para poder solicitar la tarjeta familiar de la UE es
  necesario que el familiar comunitario,
  pueda probar que tiene medios
  suficientes para poder sostenerle economicamente."

  utter_reagrupacion:
  - text: "Para hacer una reagrupacion
  no es necesario tener partida de nacimiento espanola"
```

Algoritmo 4.5: Lista de acciones

```
1          actions:
2            - action_precio
3            - action_proceso
4            - action_documentacion
5            - action_filtro
```

- TramitesJSON.json

  Documento con formato JSON, que funciona como *dataset* de información que se quería separar de domain.yml, ya que está formado por un array, llamado *arrayTramites* que contiene todos los procesos que se pueden realizar tanto en extranjería, e información importante a la hora de renovar/solicitar el pasaporte o DNI. Esta información se puede ver en el Algoritmo 4.6.

  De este *dataset* extraen la información las acciones de la anterior figura, dependiendo de la pregunta del usuario. Cada trámite tiene los siguientes elementos:

  - Tramite: apartado que simplemente nombra el elemento y sirve de guía para el programador.
  - Alias: elemento que contiene características que representan al trámite y por el que se le pueden diferenciar de los demás.
  - Precio: se especifica el dinero que cuesta tramitar dicho proceso o te guía al enlace donde poder encontrar el precio.
  - Documentación: se concreta la documentación necesaria para realizar el trámite o también puede conducir al usuario a un link donde también se muestre esta misma información.
  - Proceso: guía para solicitar cualquier expediente dirigida al cliente. En algunos elementos este apartado está complementado con un enlace, que especifica mejor los pasos que debe de seguir el usuario según sus condiciones.

- procesosJSON.json

  Documento con formato JSON, que funciona como *dataset* con el tipo de solicitudes de información que se puede hacer sobre los trámites. El contenido de este fichero se muestra en el Algoritmo 4.7.

  Este *dataset* se utiliza para comparar el último mensaje del usuario con los datos *procesosJSON.json*, de esta manera se comprueba qué tipo de información desea el receptor.

- Actions.py

  Fichero dedicado a la programación de funciones encargadas de extraer la información guardada en las entidades y buscar una respuesta gracias a este dato.

  Por tanto este documento está formado por clases de *actions* procedentes de la librería rasa_sdk y tienen el formato que se muestra en el Algoritmo 4.8.

  En la función *name* se devuelve el nombre de la acción por el que será reconocida la clase en los demás ficheros como en stories.yml o domain.yml. Por otra parte,la función *run* tiene la lógica de la clase, donde se envía el mensaje desde el *dispatcher* gracias a la función de *utter_message("texto")*. Dos ejemplos de *actions* se ven en los Algoritmos 4.9 y 4.10.

Algoritmo 4.6: Fichero *tramitesJSON.json*

```
        "arrayTramites": [
        {
          "tramite": "Prorroga de estancia de corta duracion",

          "alias": "la prorroga de estancia de corta duracion",

          "precio": "El precio puede variar dependiendo si se dispone de
          visado o no.Con visado -> 31,22 euros, Sin visado -> 17,49 euros
          (se incrementara en 1,06 euros mas por cada dia que se prorrogue
          la estancia)",

          "documentacion": "-Impreso de solicitud\n\n
          -Pasaporte completo o titulo de viaje\n\n
          -Documentaci n acreditativa de disponer de medios economicos
          suficientes\n\n
          -Documentacion acreditativa de disponer de seguro m dico
          de viaje.\n\n
          -Documentacion que garantice el retorno al pais de procedencia.\n\n
          -Documentacion acreditativa de las razones excepcionales por las que
          se solicita la prorroga",

          "proceso": "Se realiza dentro de la seccion de Extranjeria y luego
          seleccionas el apartado de Pr rroga de estancia corta de duracion.
          En este apartado podras descargarte el Formulario EX00,
          que tendras que entregar relleno en la Oficina de Extranjeria,
          Jefatura Superior o Comisar a de Policia. Por tanto, dicho tramite
          se procesa de manera presencial."
        },
```

Algoritmo 4.7: Fichero *procesosJSON.json*

```
        {
            "arrayProceso": [" Como  lo solicito?", " Como  pedirlo?",
            " Que  pasos debo de seguir?", " Donde  lo hago?", "Pasos",
            "Intrucciones", "Apartado", "Seccion", "Guia de como hacerlo",
            "Procedimiento", "Proceso", "Guia", "Guide", "Proccess", "Steps"],
            "arrayDocumentacion":["Documentos necesarios",
            "Documentacion necesaria", "Documentacion", "Papeles necesito",
            "Pappers", "Pappers that i need", "Necessary documentation"],
            "arrayPrecio":[ "Precio", "Price", "Money", "Dinero", "Valor",
            "Coste", "Euros", "Precio del tramite"]
        }
```

Algoritmo 4.8: Ejemplo actions

```
    class ActionHello(Action):

      def name(self) -> Text:
          return "action_name"

      async def run(
          self, dispatcher: CollectingDispatcher, tracker: Tracker,
          domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        dispatcher.utter_message(text:"Hello World!")

      return []
```

Algoritmo 4.9: Clase Action Proceso

```
class ActionProceso(Action):

#Donde se adjudica el nombre de la función
def name(self) -> Text:
    return "action_proceso"

def run(self, dispatcher: CollectingDispatcher, tracker: Tracker,
domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
    #Se obtiene el valor de la entidad "nombre tramite"
    nombre_tramite = tracker.get_slot("nombre_tramite")

    #Se obtiene el ultimo mensaje del usuario
    mensaje = tracker.latest_message['text']

    #Se abre el dataset y se obtienen los datos del array
    with open('procesosJSON.json', 'r', encoding='utf8') as file:
            dataProceso = json.load(file)

    maxiProceso = -1

    #Buscamos el apartado que tenga un mayor porcentaje de similitud entre
    los elementos del array y nombre_tramite
    for i in range(15):
        probabilidad = fuzz.token_set_ratio(mensaje,
                    dataProceso['arrayProceso'][i])
        if probabilidad > maxiProceso:
            maxiProceso = probabilidad


    if(nombre_tramite !=NULL and maxiProceso > 50):
        #Se abre el dataset y se obtienen los datos del array
        with open('tramitesJSON.json', 'r', encoding='utf8') as file:
            data = json.load(file)

        maxi = -1
        pos = 0

        #Buscamos el apartado que tenga un mayor porcentaje de similitud
        entre el alias y nombre_tramite
        for i in range(15):
            Token_Set_Ratio = fuzz.token_set_ratio(nombre_tramite,
                                    data['arrayTramites'][i]['alias'])
            if Token_Set_Ratio > maxi:
                maxi = Token_Set_Ratio
                pos = i


        #Se devuelve el valor si el porcentaje es mayor que 50
        if maxi > 50:
            respuesta = data['arrayTramites'][pos]['proceso']
            dispatcher.utter_message(respuesta)

        else:
            dispatcher.utter_message("Lo siento, me podrias repetir el
            tramite del que quieres informarte? Recuerda que tienen que ser
            tramites de la Sede Electronica")

        return [SlotSet("nombre_tramite", NULL)]

    else:
        dispatcher.utter_message("Lo siento, me podrias repetir el
        tramite del que quieres informarte? Recuerda que tienen que ser
        tramites de la Sede Electronica")
        return []
```

Algoritmo 4.10: Clase Action Filtro

```
class ActionFiltro(Action):

#Donde se adjudica el nombre de la función
def name(self) -> Text:
    return "action_proceso"

def run(self, dispatcher: CollectingDispatcher, tracker: Tracker,
domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
    #Se obtiene el valor de la entidad "nombre tramite"
    nombre_tramite = tracker.get_slot("nombre_tramite")

    #Se obtiene el ultimo mensaje del usuario
    mensaje = tracker.latest_message['text']

   #Se calcula el porcentaje de semejanza entre el ultimo mensaje y el slot
    probabilidad = fuzz.token_set_ratio(mensaje, nombre_tramite)

    if(nombre_tramite !=NULL and probabilidad > 50):
        #Se abre el dataset y se obtienen los datos del array
        with open('tramitesJSON.json', 'r', encoding='utf8') as file:
            data = json.load(file)

        maxi = -1

        #Buscamos el apartado que tenga un mayor porcentaje de similitud
        entre el alias y nombre_tramite
        for i in range(15):
            Token_Set_Ratio = fuzz.token_set_ratio(nombre_tramite,
                                    data['arrayTramites'][i]['alias'])
            if Token_Set_Ratio > maxi:
                maxi = Token_Set_Ratio


        #Se devuelve el valor si el porcentaje es mayor que 50
        if maxi > 50:
            dispatcher.utter_message
            ("Y que me quieres consultar sobre este tramite: \n" +
                                    "- Documentaci n necesaria\n" +
                                    "- Precio del tr mite\n" +
                                    "-  Como  pedirlo?\n")

        else:
            dispatcher.utter_message("Lo siento,  me  podrias repetir el
            tramite del que quieres informarte? Recuerda que tienen que ser
            tramites de la Sede Electronica")
    else:
        dispatcher.utter_message("Lo siento,  me  podrias repetir el
        tramite del que quieres informarte? Recuerda que tienen que ser
        tramites de la Sede Electronica")

    return []
```

La función run de las *actions* que filtran el proceso, documentación y precio del trámite se componen de los siguientes pasos:

1. Extrae el dato de la entidad *"nombre_tramite"* y lo guarda en una variable llamada *nombre_tramite*.
2. Extrae el texto del último mensaje y lo guarda en el parámetro mensaje
3. Se abre el archivo *procesosJSON* con permisos de lectura y se guarda el contenido en la variable *dataProceso*, para luego poder acceder a los datos.
4. Después se crea una variable para guardar el máximo porcentaje de similitud.

5. Se recorre el array y se guarda el porcentaje de semejanza entre la variable mensaje y el elemento del array que se está analizando. Luego ese valor se compara con la variable de *maxiProceso* y si es mayor, se actualiza.
6. Se comprueba si el valor de semejanza es mayor de 50 y si *nombre_tramite* no es nulo. Si no cumple estas condiciones, se devuelve un mensaje informando de que no se ha encontrado respuesta a la pregunta del usuario.
7. Si cumple las condiciones anteriormente explicadas, se abre el archivo *tramitesJSON* con permisos de lectura y se guarda el contenido en la variable *data*, para luego poder acceder a los datos.
8. Posteriormente se declaran dos variables que van a guardar la posición del trámite filtrado y el porcentaje máximo de semejanza entre la variable *nombre_tramite* y la sección de alias.
9. Se recorre el array, que en este caso tiene 15 elementos, y se guarda el porcentaje de semejanza entre la variable *nombre_variable* y la sección de alias en *Token_Set_Ratio*. Luego este valor se compara con la variable *maxi* y si es mayor, se actualiza y además se guarda la posición.
10. Se comprueba si el porcentaje encontrado es mayor de 50, ya que sino supera ese umbral, no es fiable la respuesta que pueda dar. Si es mayor, primero obtiene el dato de proceso y luego lo devuelve con *dispatcher.utter_message(respuesta)*. En caso de que fuera menor o igual a 50, se devuelve un mensaje informando de que no se ha encontrado respuesta a la pregunta del usuario.
11. Finalmente en el return se modifica el valor del slot a nulo, para posteriores búsquedas.

Luego en la función run de la *action* que filtra el trámite por el que se pregunta, tiene unas ligeras modificaciones comparadas con el resto.

1. Extrae el dato de la entidad *"nombre_tramite"* y lo guarda en una variable llamada *nombre_tramite*.
2. Extrae el texto del último mensaje y lo guarda en el parámetro mensaje
3. Calcula la semejanza que hay entre la variable *nombre_tramite* y el último mensaje.
4. Luego se comprueba si el *nombre_tramite* no es nulo y además de si el porcentaje de semejanza es mayor que 50. Si no cumple estas condiciones, se devuelve un mensaje informando de que no se ha encontrado respuesta a la pregunta del usuario. Esta comprobación se hace ya que el sistema algunas veces fallaba al captar el trámite por el que se preguntaba.
5. Si cumple las condiciones anteriormente explicadas, se abre el archivo *tramitesJSON* con permisos de lectura y se guarda el contenido en la variable *data*, para luego poder acceder a los datos.
6. Posteriormente se declaran la variable que guarda el porcentaje máximo de semejanza entre la variable *nombre_tramite* y la sección de alias.
7. Se recorre el array, que en este caso tiene 15 elementos, y se guarda el porcentaje de semejanza entre la variable *nombre_variable* y la sección de alias en *Token_Set_Ratio*. Luego este valor se compara con la variable *maxi* y si es mayor, se actualiza.

8. Finalmente se comprueba si el porcentaje encontrado es mayor de 50, ya que sino supera ese umbral, no es fiable la respuesta que pueda dar. Si es mayor, primero obtiene el dato de proceso y luego lo devuelve con *dispatcher.utter_message(respuesta)*. En caso de que fuera menor o igual a 50, se devuelve un mensaje informando de que no se ha encontrado respuesta a la pregunta del usuario.

En resumen, las cuatro clases que protagonizan este proyecto son las siguientes con sus correspondientes salidas:

- ActionProceso → respuesta=data['arrayTramites'][pos]['proceso']
- ActionDocumentacion → respuesta=data['arrayTramites'][pos]['documentacion']
- ActionPrecio → respuesta=data['arrayTramites'][pos]['precio']
- ActionFiltro → respuesta = mensaje con la información por la que puedes preguntar del trámite.

- Stories.yml

  Apartado dedicado a especificar las diferentes versiones de conversaciones que se puede mantener con el chatbot; de esta manera se establece un patrón que debe de seguir el asistente al recibir un mensaje y así saber que responder al usuario. Estas guías reciben el nombre de *stories* y están formadas por la secuencia de *intents* y *actions*.

  En el Algortimo 4.11 se ve un ejemplo de la estructura de una story esencial como la de preguntar_tramite. Los pasos que la componen son los siguientes:

  1. Recibe la pregunta del usuario sobre el proceso que se debe de seguir para tramitar un expediente.
  2. Se llama a *action_proceso*, que filtrará la entidad de *nombre_tramite* y devolverá la respuesta idónea.
  3. Finalmente el chatbot pregunta si ha resuelto la duda.

Algoritmo 4.11: Fichero stories.yml

```
- story: preguntar-expulsar
  steps:
  - intent: expulsion-extranjero
  - action: utter_expulsar
  - action: utter_ayuda

- story: preguntar-sancionador
  steps:
  - intent: sancionador
  - action: utter_sancionador
  - action: utter_ayuda

- story: precio-pasaporte
  steps:
    - intent: precio-pasaporte
    - action: utter_precioPasaporte
    - action: utter_ayuda
```

En este fichero hay tres *stories* parecidas a la anterior y que responden a los diferentes apartados de precio, documentación y proceso como se pueden ver en los Algoritmos

4.12, 4.13 y 4.14. Están separadas con el fin de conseguir sencillez y rapidez a la hora de programar y encontrar posibles errores.

Algoritmo 4.12: Story preguntar-precio

```
- story: preguntar-precio
  steps:
    - or:
        - intent: precio-general
        - intent: precio-tarjetas
        - intent: precio-tramites
        - intent: precio-dni
    - action: action_precio
    - action: utter_ayuda
```

Algoritmo 4.13: Story preguntar-documentacion

```
- story: preguntar-documentacion
  steps:
    - or:
        - intent: documentacion-general
        - intent: documentacion-tarjetas
        - intent: documentacion-tramites
        - intent: documentacion-SPasaporte
        - intent: documentacion-RPasaporte
        - intent: documentacion-dni
    - action: action_documentacion
    - action: utter_ayuda
```

Algoritmo 4.14: Story preguntar-documentacion

```
- story: preguntar-tramite
steps:
  - or:
    - intent: proceso-general
    - intent: proceso-tarjeta
    - intent: proceso-tramites
    - intent: procedimiento-RPasaporte
    - intent: prodecimiento-SPasaporte
    - intent: proceso-dni
  - action: action_proceso
```

- Creedentials.yml

  Documento donde se concreta los diferentes conectores que puede tener el asistente. Un conector establece la conexión entre el asistente y los canales de entrada y salida de datos. Rasa Stack predefine los siguientes al crear un proyecto nuevo:

  - Telegram
  - Facebook
  - Slack
  - Socketio
  - Mattermost

Para este proyecto se han utilizado dos conectores, el de Telegram y Socketio como se ve en los Algoritmos 4.15 y 4.16.

Algoritmo 4.15: Conector Telegram

```
1     telegram:
2       access_token:
3       verify: "SedeChatbot"
4       webhook_url:
```

Algoritmo 4.16: Conexión con Socketio

```
1     socketio:
2       user_message_evt: user_uttered
3       bot_message_evt: bot_uttered
4       session_persistence: true
```

Las conexiones en un principio aparecen como comentadas, pero según se vaya creando dichas conexiones se puede ir activando los conectores. Los valores de cada conector se explicarán en el apartado de conexión con Rasa Stack.

- Config.yml

  En este archivo se define como procesar los mensajes mediante una sucesión de componentes que se ejecutan secuencialmente. La configuración que se requiera para el proyecto se debe de especificar en el apartado del pipeline. A continuación, se explican los componentes esenciales que hacen posible un entrenamiento efectivo de los datos.

  En este caso se ha decidido usar la configuración recomendada por Rasa Stack [ith21], que utiliza WhitespaceTokenizer. Esta herramienta genera un vector de tokens extraídos del mensaje, eliminando así los espacios en blanco y signos de puntuación u otros símbolos.

  También utiliza CountsVectorsFeaturizer, herramienta con la función de crear una bolsa de palabras, ignorando así su orden; de esta manera es más fácil clasificar las intenciones y escoger decisiones. Su fundamento se basa en el algoritmo de *scikit-learn CountVectorizer*, que cuenta el número de veces que se repite un token como se ve en la Figura 4.8. A su vez permite el procesamiento de datos de texto antes de generar el vector.

  Por otro lado, otro componente que utiliza Rasa Stack es RegexFeaturizer, que crea un vector de características de cada mensaje proveniente del usuario, para luego extraer las entidades y clasificar la intención. En el entrenamiento del asistente se genera una lista de expresiones regulares definidas con anterioridad en el archivo nlu.yml.

  De manera continua, utiliza LexicalSyntacticFeaturizer, que crea características léxicas y sintácticas para que se pueda extraer entidades del mensaje del usuario. En la Figura 4.9 se aprecia como que se extrae cada *token* y se especifican sus características según la configuración acordada.

A continuación se sigue con *Dual Intent Entity Transformer* (DIET), que se utiliza para la clasificación de los *intents* y *entities* extraídos. En la Figura 4.10 se ve un ejemplo con el que se puede ver mejor el nivel de detalle que da este componente.

En paralelo para la búsqueda de sinónimos se utiliza EntitySynonymMapper, que asigna *tokens* detectados a los datos de entrenamiento que sean similares.

El último componente a recalcar es ResponseSelector [CI22], que crea un esquema de predicción con las respuestas candidatas que puede dar el asistente ante un mensaje de un usuario. Gracias a esta estructura, el gestor de diálogos puede escoger que *action* o *utter* realizará el chatbot. En la Figura 4.11 se muestra un ejemplo de los datos que se recopilan para la posterior selección.

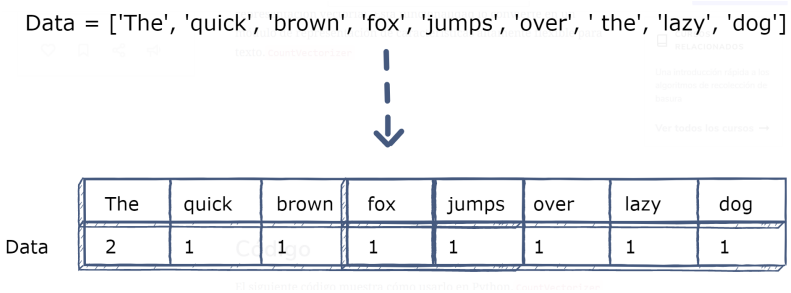


Figura 4.8: Ejemplo de uso con CountVectorizer

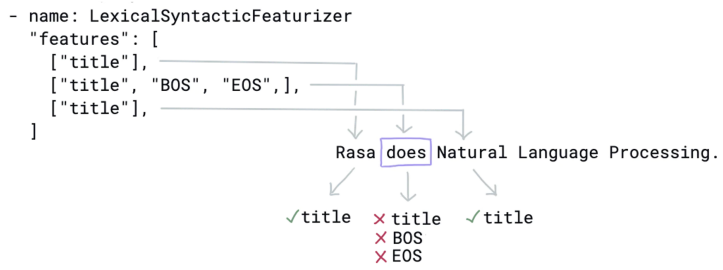


Figura 4.9: Ejemplo de uso con LexicalSyntacticFeaturizer

Figura 4.10: Ejemplo de uso con DIETClassifier

```
{
    "response_selector": {
      "default": {
        "response": {
          "id": 1388783286124361986,
          "confidence": 0.7,
          "intent_response_key": "chitchat/ask_weather",
          "responses": [
            {
              "text": "It's sunny in Berlin today",
              "image": "https://i.imgur.com/nGF1K8f.jpg"
            },
            {
              "text": "I think it's about to rain."
            }
          ],
          "utter_action": "utter_chitchat/ask_weather"
        },
        "ranking": [
          {
            "id": 1388783286124361986,
            "confidence": 0.7,
            "intent_response_key": "chitchat/ask_weather"
          },
          {
            "id": 1388783286124361986,
            "confidence": 0.3,
            "intent_response_key": "chitchat/ask_name"
          }
        ]
      }
    }
}
```


Figura 4.11: Ejemplo de ResponseSelector

Finalmente el resultado del pipeline de este proyecto se muestra en el Algoritmo 4.17.

Algoritmo 4.17: Configuración por defecto del pipeline

```
language: es  # your two-letter language code

pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: "char_wb"
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 100
```

- Endpoints.yml

  Es este fichero es donde se puede modificar los puertos en donde se lanzan los diferentes componentes de Rasa. Hay que activar el action_endpoint si se utilizan actions en el proyecto y la url deberá apuntar al puerto 5055, como se muestra en el Algoritmo 4.18. Esto es debido a que las acciones del archivo actions.py se deben de

copilar como punto final en el puerto 5055 de nuestro ordenador para que funcione de manera correcta.

Algoritmo 4.18: Configuración de endpoints.yml

```
1     action_endpoint:
2
3     url: "http://localhost:[number-host]/webhook"
```

4.4. Despliegue de Herramienta

El desarrollo del proyecto se realizó en un ordenador con sistema operativo Windows 11 y se utilizó la versión de python 3.9 para programar el fichero de actions.py y entrenar el asistente.

Lo primero que se debe de hacer es crear un entorno virtual como en la Figura 4.12 para no comprometer la estabilidad del sistema operativo al descargar la herramienta de Rasa Stack.

```
PS C:\Users\User> python -m venv --system-site-packages ./venv
PS C:\Users\User> ./venv/Scripts/activate
```

Figura 4.12: Creación de entorno virtual

Cuando ya se tiene todas estas características cumplidas, seguiremos los pasos descritos a continuación para instalar y crear de manera correcto un proyecto de Rasa:

1. Para la instalación se escribirá este comando por terminal **pip install rasa**
2. Una vez instalada la herramienta de Rasa, se crea una carpeta donde alojar el proyecto con el comando **rasa init –no-prompt**. Con la etiqueta de **-no-prompt** se evita la interrupción de la creación del proyecto por preguntas innecesarias. La carpeta creada tendrá el formato de la Figura 4.13.
3. A continuación, se debe de personalizar los ficheros para que se ajuste a las características del chatbot deseado.
4. Una vez se haya modificado los ficheros, se debe de pasar al entrenamiento del chatbot con el comando **rasa train**. Este paso creará un modelo en la carpeta de models, como se muestra en la Figura 4.14, que será analizado por el asistente a la hora de mantener una conversación con el usuario.
5. A continuación para copilar nuestro proyecto en terminal y probar su eficiencia sin tener que depender de ningún conector externo se utiliza el comando **rasa shell**. También se debe de tener en cuenta la ejecución de las *actions* con el comando **rasa run actions** en otra terminal a parte. Cuando se lancé el asistente, se tomará el último modelo creado. Un ejemplo de la ejecución se muestra en la Figura 4.15.

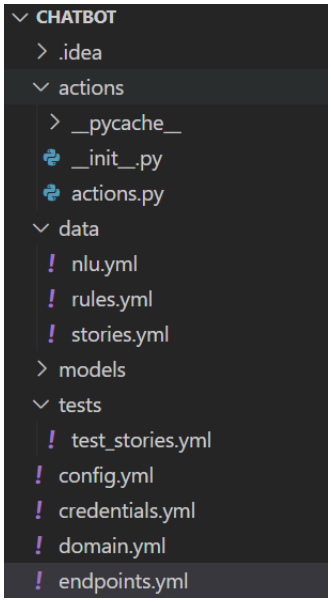


Figura 4.13: Carpeta de trabajo

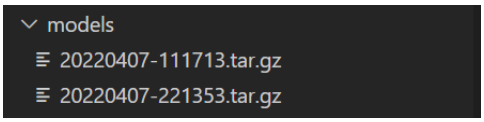


Figura 4.14: Ejemplos de modelos creados

```
PS C:\Users\User\Documents\5 CARRERA\TFG\Chatbot> rasa shell
2022-04-19 18:33:48 INFO     rasa.model  - Loading model models\20220407-221353.tar.gz...
2022-04-19 18:34:04 INFO     root  - Connecting to channel 'cmdline' which was specified by the '--connec
tor' argument. Any other channels will be ignored. To connect to all given channels, omit the '--connecto
r' argument.
2022-04-19 18:34:04 INFO     root  - Starting Rasa server on http://localhost:5005
2022-04-19 18:34:04 INFO     rasa.model  - Loading model models\20220407-221353.tar.gz...
2022-04-19 18:34:40 WARNING  rasa.shared.utils.common  - The UnexpecTED Intent Policy is currently experi
mental and might change or be removed in the future 🔬 Please share your feedback on it in the forum (htt
ps://forum.rasa.com) to help us make this feature ready for production.
2022-04-19 18:34:48 INFO     root  - Rasa server is up and running.
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input ->
```

Figura 4.15: Ejecutar el asistente por terminal

4.5. Conexión con Telegram y Servicio Web

Antes de explicar el procedimiento que se ha seguido para realizar las conexiones, hay que tener en cuenta que el servidor de Rasa ejecuta el proyecto de manera local, mientras que Telegram y la página web se ejecutan en la nube. Por tanto se tienen que enlazar las dos ejecuciones mediante el software de *ngrok*. Es una herramienta que hay que instalarla como zip y posteriormente descomprimirla para su uso. De esta manera, será posible acceder a la terminal *ngrok.exe*, desde donde se lanzará el comando **ngrok http 5005**, puerto donde se ejecuta localmente el proyecto. Es importante mantener abierto este túnel mientras se este utilizando el asistente vía Telegram o web, ya que si se cierra ngrok dejará de funcionar la conexión.

En el Algoritmo 4.19 se muestra como proporciona una url operativa durante 8 horas que simula un túnel al mismo puerto de nuestro localhost. El enlace que se debe de escoger es el https, ya que se debe de realizar una conexión segura, y por ejemplo Telegram no acepta enlaces http.

Algoritmo 4.19: Ejecución de ngrok

```
1   ngrok by @inconshreveable
2
3   Session Status                online
4   Account                       Rebeca (Plan: Free)
5   Version                       2.3.40
6   Region                        United States (us)
7   Web Interface
8   Forwarding
9   Forwarding
10  Connections                   ttl     opn     rt1     rt5     p50     p90
11                                0       0       0.00    0.00    0.00    0.00
```

4.5.1. Telegram

Como se ha explicado anteriormente, Telegram pone a la disposición del usuario la herramienta de The BotFather. Como se ve en la Figura 4.16, esta funcionalidad permite crear un bot de manera sencilla en la que BotFather irá haciendo preguntas al usuario de cómo quiere personalizar su asistente.

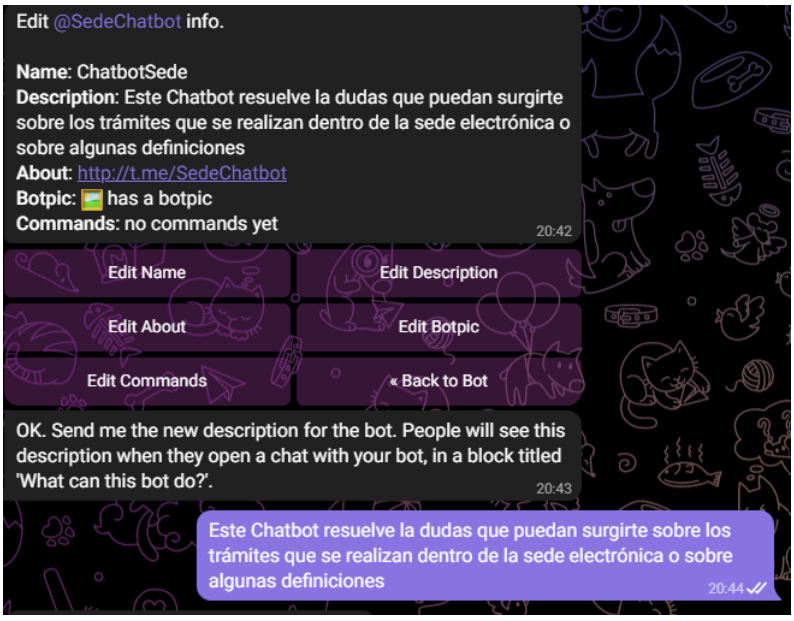


Figura 4.16: Personalizar bot

Los pasos para realizar la conexión con Telegram son los siguientes:

1. Crear y personalizar el asistente en Telegram con BotFather.
2. Una vez creado se debe de guardar el API TOKEN y el nombre del asistente en el proyecto de Rasa.

3. En el Algoritmo 4.20 se especifica la configuración del archivo creedentials.yml. Se tiene que descomentar el apartado de Telegram y se escriben los datos guardados entre comillas.

   - Acces_token → API TOKEN
   - Verify → Nombre del Chabtot
   - webhook_url → Mnlace proporcionado por ngrok, añadiendo luego el sufijo de /webhooks/telegram/webhook.

Algoritmo 4.20: Conexión Telegram

```
telegram:
  access_token:
  verify: "SedeChatbot"
  webhook_url:
```

4. Abrir dos terminales localizadas en la carpeta del proyecto y ejecutar **rasa run** en una y **rasa run actions** en la restante.

Realizados ya estos pasos se puede pasar a hablar con el asistente vía Telegram. En la Figura 4.17 se ve el formato de una conversación por Telegram.

(a) Saludo inicial (b) Información del Pasaporte

Figura 4.17: Resultados en Telegram

4.5.2. Servidor Web

Para esta opción se necesita de ngrok, Github y BotFront, herramientas que se utilizan a manera de prueba, ya que también se puede utilizar una página de web. Además de que habrá que realizar pequeñas modificaciones en el código para su correcto funcionamiento.

Los pasos a seguir son los siguientes:

1. Creamos un repositorio en Github, con el nombre de usuario y el sufijo **.github.io.**

2. Añadimos a ese repositorio nuestro proyecto y además un documento index.html. El resultado del repositorio sería el que se muestra en la Figura 4.18.

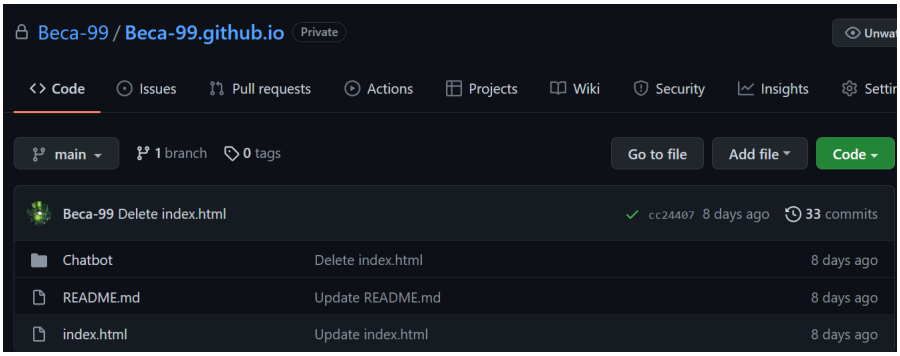


Figura 4.18: Repositorio de Github

3. Ahora es el momento de completar el archivo index.html con el diseño que se desee para la página web. En este caso se ha utilizado BotFront, un open source en Github sencillo de utilizar y que funciona con el Algoritmo 4.21.

Algoritmo 4.21: BotFront Script

```
<script>!(function () {
let e = document.createElement("script"),
  t = document.head || document.getElementsByTagName("head")[0];
(e.src =
  "https://cdn.jsdelivr.net/npm/rasa-webchat@1.x.x/lib/index.js"),
  // Replace 1.x.x with the version that you want
  (e.async = !0),
  (e.onload = () => {
    window.WebChat.default(
      {
        customData: { language: "en" },
        socketUrl: "https://bf-botfront.development.agents.botfront.cloud",
        // add other props here
      },
      null
    );
  }),
  t.insertBefore(e, t.firstChild);
})();
</script>
```

El único cambio que habría que realizar sería modificar el dato de **socketUrl** y añadir el enlace de **ngrok**.

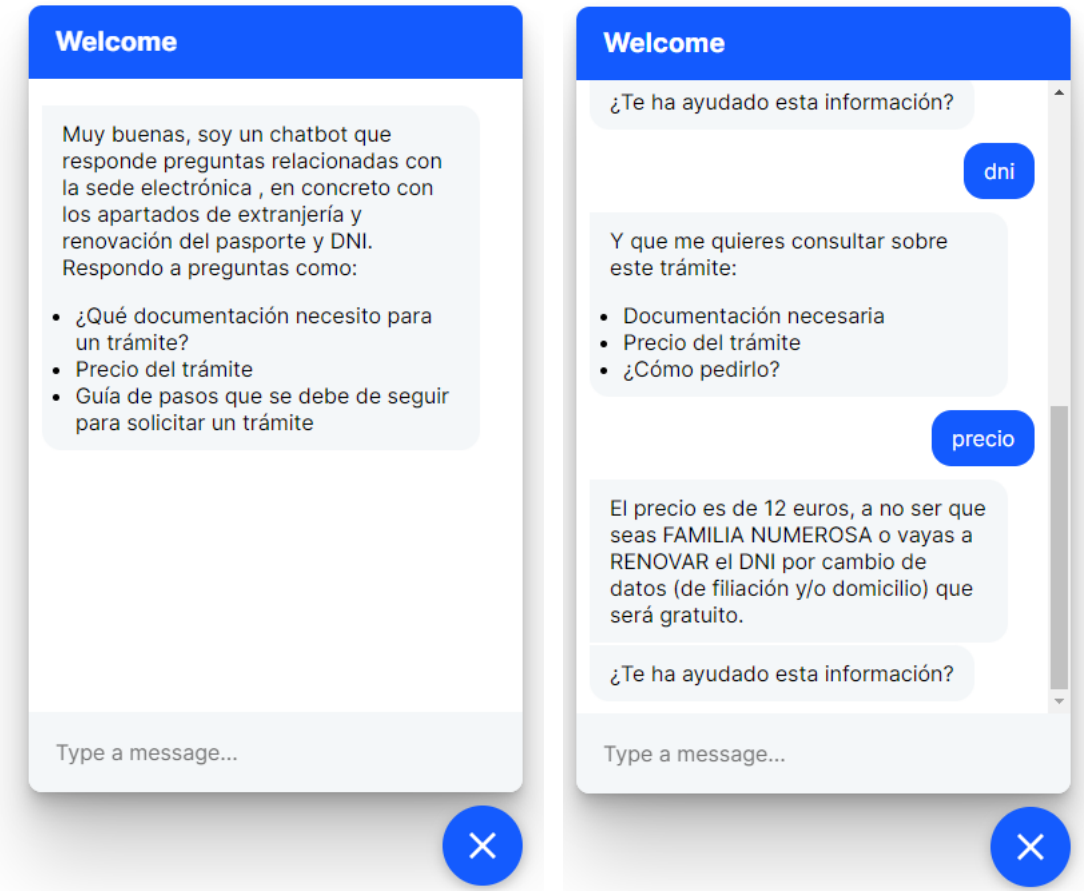
4. Pasando al proyecto, hay que descomentar la opción de *socketio* en el archivo de creedentials.yml y escribir las características del Algoritmo 4.22.

Algoritmo 4.22: Conexión con Socketio

```
socketio:
  user_message_evt: user_uttered
  bot_message_evt: bot_uttered
  session_persistence: true
```

5. A la hora de ejecutar el asistente, también habrá que abrir dos terminales situadas en la carpeta del proyecto, y en una lanzar el comando de **rasa run actions** y en otra rasa **run -m models –enable-api –cors "*"**.

6. La Figura 4.19 muestra el resultado de la página→**https://nombre-usuario.github.io.**



(a) Saludo inicial (b) Precio del DNI

Figura 4.19: Resultado en web

Capítulo 5

Experimentos y Resultados

Para realizar pruebas sobre este proyecto se han elaborado pruebas de estrés y test sobre el asistente, además de las correspondientes revisiones durante su desarrollo. El primer apartado 5.1 se describe el proceso de realizar pruebas de estrés con las extensión de RESTful Stress. La segunda parte de este capitulo 5.2, se describen tres test que se han programado en python para comprobar el funcionamiento de las acciones y además se presentan las cifras de confianza de los *intents*. En la última sección 5.3, se mostrarán resultados de otros proyectos relacionados con Rasa Stack y se hará una breve comparación con este proyecto.

5.1. Pruebas con la Herramienta RESTful Stress

La herramienta RESTful Stress es una extensión de Google Chrome que permite analizar el asistente para saber datos de tiempo de respuesta según el número de usuarios conectados a la vez. El funcionamiento de la extensión se basa en llamadas HTTP(GET,POST,PUT y DELETE) al endpoint de la API.

Los pasos para realizar las pruebas de estrés son las siguientes:

1. Primero se debe descargar la extensión. Simplemente con buscarla en el buscador Google e instalarla, ya se podrá ejecutar.
2. Se escribe la dirección del **target**, dónde se realizará una petición POST. La url que habrá que indicar será la siguiente: http://localhost:5005/webhooks/rest/webhook.
3. Se rellena el recuadro de **request body** con el mensaje de prueba. El formato más indicado para enviar el mensaje es en JSON y especificando los campos de **sender** y **message**.
4. También se pueden modificar los campos de **options**. En este caso, el valor de *iterations* es 1500, el *delay* es de 100 ms y el *timeout* de 30000.
5. Para realizar la prueba, se debe de lanzar el siguiente comando por terminal en la carpeta del proyecto: **run -m models –enable-api –cors “*” –debug** y en paralelo se pincha en el botón de play de la aplicacion RESTful.

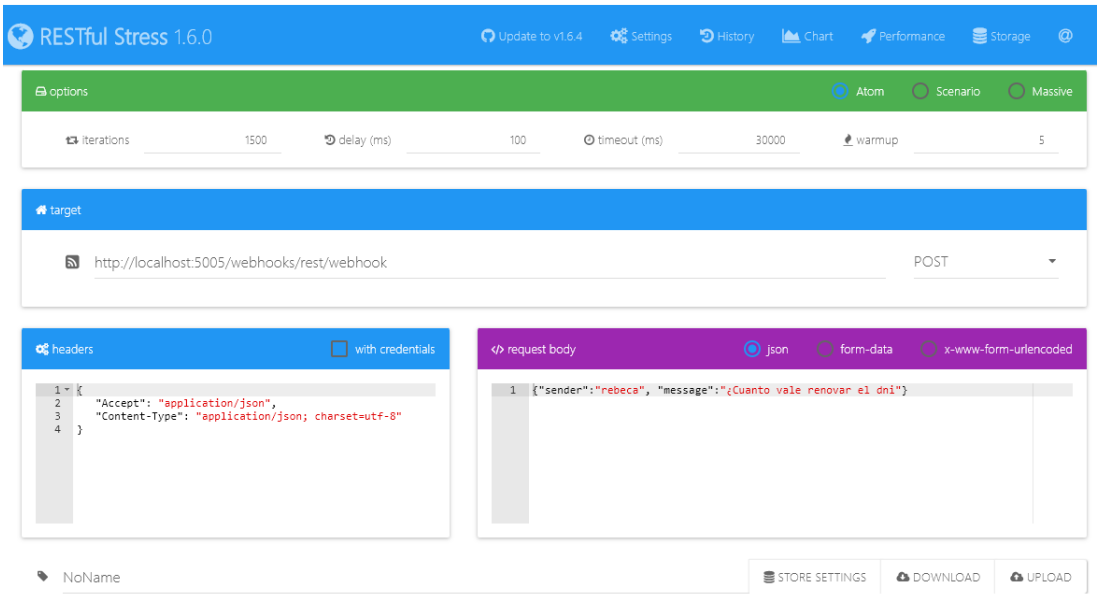


Figura 5.1: RESTful Stress - Settings

En la Figura 5.1 se muestra que el campo del message es *¿Cuánto vale renovar el dni?*, para saber el tiempo que tarda en filtrar el *dataset* y encontrar la respuesta correcta.

Se han realizado al final 235 iteraciones, con los resultados siguientes:

- Duración media: 2778ms
- Correctos 235
- Duración máxima: 4603 ms
- Errores: 0
- Duración mínima: 2176 ms
- Warmups: 5
- Duración total: 652939
- Usuarios: 3 de manera concurrente

Estos datos se pueden revisar en las dos pestañas de **Performance** y **Chart**, mostradas en las Figuras 5.2 y 5.3.

5.2. Test Unitarios sobre Actions y Porcentajes de Confianza

Para probar el funcionamiento de las tres acciones descritas en el archivo de **actions.py**, se han realizado tests unitarios con la librería de *unittest* de Python.

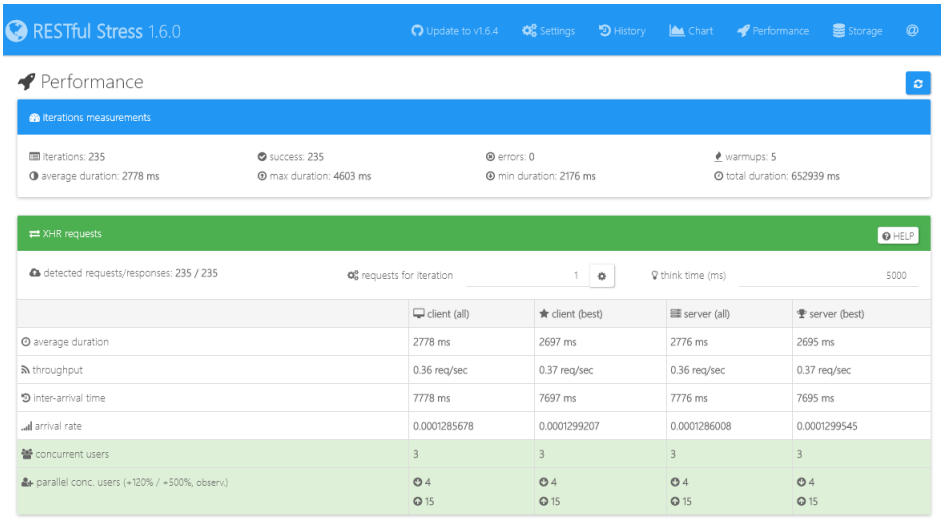


Figura 5.2: RESTful Stress - Performance

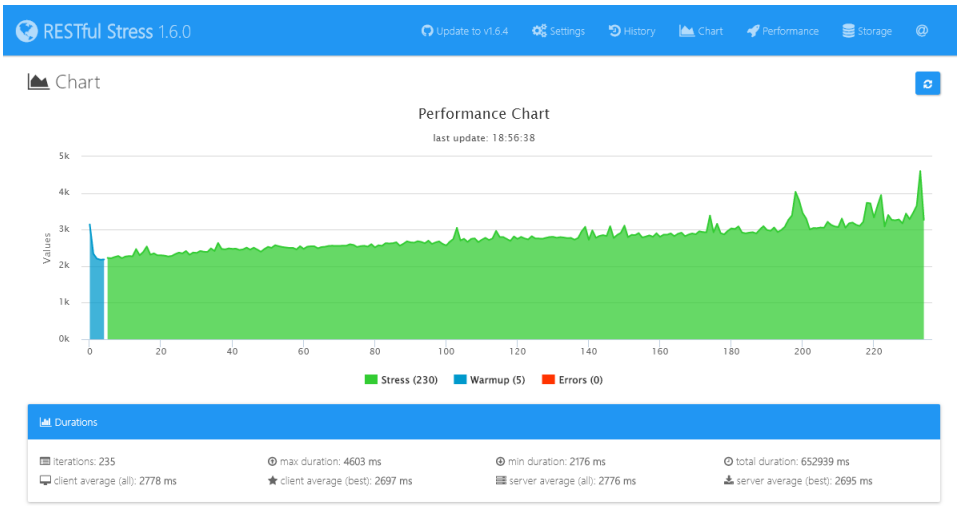


Figura 5.3: RESTful Stress - Chart

Los tests tienen un formato como el que muestra el Algoritmo 5.1 y los tipos que hay son:

- **test_filtro_precio** comprueba si se ha escogido el precio correcto con un un trámite ejemplo, que en este caso es la solicitud de la cédula de inscripción
- **test_filtro_documentation** revisa si la documentación filtrada es la correcta para el trámite de asignar el NIE.
- **test_filtro_proceso** verifica si se ha obtenido el proceso correcto para el trámite de solicitar la tarjeta de estudiante.

Para ejecutar estos tests [CC20], se debe de abrir una terminal localizada en el proyecto y llamar al método de *main* de la librería *unittest* con el siguiente comando **python test_unitarios.py -v**, como se muestra en la Figura 5.4.

Algoritmo 5.1: Ejemplo del test_filtro_precio

```
1     class TestActions(unittest.TestCase):
2
3     def test_filtro_precio(self):
4         precio = "El precio es 3,27 euros"
5         tramite = "cedula de incripcion"
6
7
8         maxi = -1
9         pos = 0
10
11        #Buscamos el apartado que tenga un mayor porcentaje de similitud
12        entre el alias y nombre_tramite
13        for i in range(15):
14            Token_Set_Ratio = fuzz.token_set_ratio(tramite, data
15                ['arrayTramites'][i]['alias'])
16            if Token_Set_Ratio > maxi:
17                maxi = Token_Set_Ratio
18                pos = i
19
20        #Se devuelve el valor si el porcentaje es mayor que 50
21        if maxi > 50:
22            respuesta = data['arrayTramites'][pos]['precio']
23
24        else:
25            respuesta = "ninguna"
26
27        self.assertEqual(respuesta,precio, 'Error al encontrar el precio')
```

```
PS C:\Users\User\Documents\5 CARRERA\TFG\Chatbot> python test_unitarios.py -v
test_filtro_documentation (__main__.TestActions) ... ok
test_filtro_precio (__main__.TestActions) ... ok
test_filtro_proceso (__main__.TestActions) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.003s

OK
```

Figura 5.4: Ejecución de los tests unitarios

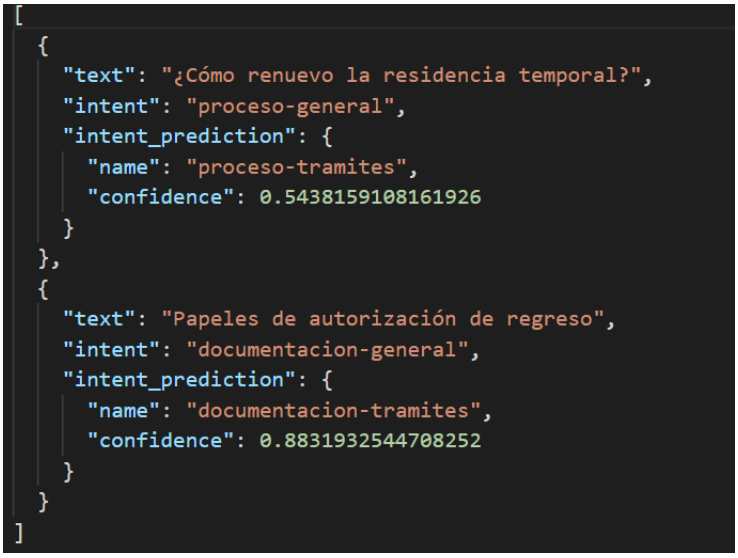
Por otra parte, para saber sobre los porcentajes de confianza de cada *intent* [PG+20], se ha utilizado los ficheros de validación que se encuentran en la carpeta de **results**.

En un principio cuando se crea el proyecto, esta carpeta no existe. Por tanto, para evaluar el modelo para la clasificación de la intención y extracción de entidades se debe de escribir **rasa test** por la terminal situada en la carpeta del proyecto.

Este comando creará la carpeta de **results**, con varios archivos de datos analizando el asistente. Los archivos que se quieren destacar para analizar los resultados son:

- intent_errors.json
- intent_successes.json

Como indica su nombre, en el primero se encuentran los errores posibles que pueden aparecer en el asistente, mientras que en el segundo archivo se obtienen los *intents* correctos. En la Figura 5.5 se muestra el contenido del fichero de errores con sus correspondientes predicciones.



```
[
  {
    "text": "¿Cómo renuevo la residencia temporal?",
    "intent": "proceso-general",
    "intent_prediction": {
      "name": "proceso-tramites",
      "confidence": 0.5438159108161926
    }
  },
  {
    "text": "Papeles de autorización de regreso",
    "intent": "documentacion-general",
    "intent_prediction": {
      "name": "documentacion-tramites",
      "confidence": 0.8831932544708252
    }
  }
]
```


Figura 5.5: Archivo intent_errors.json

Y en **intent_successes.json** se han analizado los porcentajes de confianza de los diez primeros *intents*, con sus respectivas frases de ejemplos. En la Tabla 5.1 se han recopilado los resultado obtenidos.

Intención	Confianza media
start	0.96
despedir	0.90
confirmar	0.92
negar	0.84
preguntas-infoSede	0.90
preguntas-gestiones	0.78
preguntas-certificados	0.88
preguntas-firmaElec	0.76
preguntas-extranjeria	0.76
pregunta-reagrupacion	0.94

Tabla 5.1: Resultados del archivo intent_successes.json

Se puede apreciar que los valores son altos debido al gran número de ejemplos que se han añadido a cada intención.

5.3. Comparación de Resultado Obtenidos

En el trabajo de *Rasa framwork: Ánalisis e implementación de un chatbot* [CC20] se han realizado pruebas de estrés con la misma herramienta que se ha visto anteriormente,

RESTful. Los resultados obtenidos son un tiempo medio de respuesta de 2861ms. Este resultado es levemente mayor que el resultado obtenido en este proyecto (2778 ms). En cambio el mejor tiempo por parte del servidor es más bajo, con un 2614ms que el presentado, con 2695ms. Respecto al número de usuarios que pueden utilizar el chatbot de manera concurrente es igual en los dos trabajos.También se realizaron test unitarios, pero al ser un mayor número de funciones la diferencia de tiempos es más destacable. El tiempo de ejecución es de 1,297s mientras que en este trabajo es de 0,003s.

Por otra parte, en el proyecto de *Diseño e implementación de un Chatbot para el software de IDBOD* [PG+20], los resultados que analiza son los niveles de confianza que obtiene al mantener algunas conversaciones con el asistente. Realiza dos tablas con entradas de texto fuera del alcance del bot y con los niveles de confianza según el *intent* que se esté utilizando. Las conclusiones que extrae de los resultados son que el alcance del bot es limitando, ya que no tienes muchas funciones implementadas y tiene expresiones muy restringidas.

La tabla 5.2 muestra los resultado de testear 10 intenciones.

Intención	Confianza media
saludo	0.77
buen_humor	0.61
agradecer	0.45
despedida	0.45
grafica	0.77
grafica_serie	0.89
grafica_hist	0.76
informacion	0.84
descripcion	0.86
dar_senal	0.82

Tabla 5.2: Resultados de confianza media

Los resultados son similares a los obtenidos en este trabajo, la mayoría presenta un porcentaje mayor 0.70 a excepción de los *intents* **buen_humor** y **agradecer**.

También se quiere destacar el proyecto de Desarrollo de un Chatbot para P ublico Infantil para Clasificar Sonidos del Cielo [PG21], que aunque no se realizaron pruebas de estrés ni se analizaron los porcentaje de confianza, si que hay que destacar su alta complejidad. Por ello constará de tiempos de ejecución más altos solamente por contar con herramientas como MongoDB o acceder a la API de RESTful.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

El objetivo de este trabajo era programar un asistente que ayudará a los usuarios sobre las dudas que tuvieran dentro de la Sede Electrónica. En un primer momento, no se tenían definidos los apartados a los que daría alcance, pero al final el asistente responde ante todos los trámites de extranjería, renovación y solicitud de pasaporte o DNI.

Se han investigado varias herramientas capaces para desarrollar asistentes conversacionales, concluyendo que la tecnología que se utilizaría sería Rasa Stack por ser una herramienta completa y sencilla. Por otro lado es una herramienta que permite programar todas la funcionalidades que dispone un asistente, mientras que otras aplicaciones eran más limitadas y disponían de poco soporte.

A nivel personal, ha sido todo un reto desarrollar un asistente conversacional con una aplicación que desconocía. Pero por otra parte, ha sido interesante conocer este tipo de herramientas y sobre todo ir aprendiendo a lo largo de estos seis meses como utilizarla. A su vez me he podido sumergir en algunos trámites que se realizan dentro de la Sede Electrónica, haciéndome crecer como informática al realizar un proyecto solicitado por la policía. También me ha gustado informarme e iniciar mis conocimientos dentro de la Inteligencia Artificial, que era un tema que todavía no había podido ver en la carrera.

Sin duda es una herramienta que en un futuro tendrá mas protagonismo, ya que el porvenir está protagonizado por la IA, y qué mejor manera que ayudando a la humanidad. Me alegro de haber formado parte de este proyecto y sobre todo de solucionar algunas dudas que puedan surgirles a los usuarios dentro de la sede.

6.2. Trabajo Futuro

Gracias a los requisitos que caracterizan a Rasa, se ha podido realizar un asistente de manera exitosa y apto para futuros trabajos que se quieran realizar sobre él. Pero igualmente se puede perfeccionar e incluir posible mejoras como:

- Añadir más *intents* o *entities* con el de que responda a más preguntas sobre otros trámites de la Sede Electrónica.
- Desarrollar la parte gráfica del asistente.

- Enlazar el asistente a otras aplicaciones como WhatsApp.
- Automatizar el despliegue de la herramienta.

Capítulo 7

Introduction

7.1. Motivation

The procedures which are carried out over the National Police Electronic Headquarters have suffered a series of changes and evolutions due to the COVID-19 pandemic. Therefore, the citizens are, more than ever, in need of assistance regarding some of the points available on this webpage.

The police grew aware of this increase in the number of questions posed by users and hence decided that it would be useful to develop a chatbot which helped said users through any problems they may encounter during the procedures. Therefore, the motivation of this Dissertation lies in the need for these questions to be answered via an intelligent conversational assistant.

A chatbot - or conversational bot - is a computer programme which interacts with people through a text or voice chat, providing users with quick answers and solutions to common and repetitive tasks and questions, susceptible to being automated. These programmes have existed since the beginning of programming. However, thanks to the advances and developments in the field of Artificial Intelligence (AI) these last few years, as well as the new natural language recognition and analysis techniques, interest has been raised once again.

Within this context, two fields of AI can be put to use: Natural Language Understanding (NLU), and Natural Language Processing (NLP). These two components provide us with concrete answers to the questions posed by the clients, being comprehensive to the human speech as it is produced, while also understanding the significance and intentionality behind a phrase. Currently, there are several applications which make use of these fields in order to design a conversational assistant. The example explored and seen in this paper is Rasa, a simple yet complete framework, with every necessary functionality.

But first, before starting programming, we need to extract all the possible doubts found in forums as well as the information placed in the Electronic Headquarters themselves, so that it can be later analysed and compiled into a dataset, used to train the chatbot.

7.2. Purpose of the Investigation

The aim of this project is to design and implement a chatbot based on beforehand posed questions - both those asked in forums and those provided by the National Police. The main purpose of this element is to improve and speed up those procedures belonging to the Electronic Headquarters, while, at the same time, helping the average citizen by answering their questions as they visit the webpage. Therefore, the key to this is finding out how to quicken the interactions between people and services, thus improving the general experience of the client. Simultaneously, this project aims to provide the National Police with a tool to improve the process of compromise with the clients while also taking advantage of the chatbot's operative efficiency by reducing the cost of a customer support service.

At the same time, this work intends to classify the analysed questions and answers in order to create a corpus of expressions with which the chatbot could be later trained. Moreover, it is the intention of this paper to understand how conversational assistants work, analyse their structure, and discover the main difficulties that computer engineers have to face when developing said software.

7.3. Work Plan

The development of this work has been carried out in three phases:

1. **Investigation:**

   During the first three months, a process of adaptation to the context and the environment, as well as that of acquisition of the necessary knowledge was undergone, in order to be able to start the later development of the programme. At the beginning of the project, a general meeting was held in which the starting point of the work was explained, what the objectives to be achieved were and what the necessary knowledge was going to be. After that, several subsequent meetings were called, in order to monitor the progress of the investigation and solve any potential doubts that may arise. Another reason why these meetings were arranged was to explain the basics of the applications with which the conversational bots can be programmed. The meetings also explained how to use a number of tools that would facilitate the research. One of these tools was Google Scholar. This tool would be used to search for scientific articles about previous research in the same field of study. Finally, once several conclusions about the project's requirements were obtained, priority was given to the development process.

2. **Development:**

   Once the necessary knowledge to be able to start creating a solid version of the project was acquired, it was decided to begin with the codification of the proposal. The investigation stage, however, was not over, since it was essential to know about advanced concepts of the programming language Python and be able to handle the Rasa Stack tools. During this stage, training datasets were also built, based on information extracted from the papers read during the previous phase.

3. **Experimentation:**

Finally, the prototypes of the initial idea were made, thus beginning the experimentation process. In this stage, tests were carried out on web servers. These tests consisted of checking whether the bot responded correctly to all the tasks involving the project. Additionally, stress tests were carried out, with the tool RESTful. Finally, a comparison was made, taking the results obtained from both the experimentation and the investigation stages.

7.4. Dissertation Structure

The rest of the paper is organized into 6 chapters and 1 annexe with the following structure:

Chapter 2 explains the history of conversational assistants, as well as describing the current relationship between them and AI. Additionally, concepts that are fundamental for the comprehension of the operation of chatbots are clarified. Finally, the processes covered by this project are described, showing the environment of the Electronic Headquarters of the National Police.

Chapter 3 presents the state of the art, reviewing different solutions focused on various business sectors, in order to appreciate the interpenetration of the assistants in the economic and social panorama. On the other hand, all the applications capable of developing an assistant are analysed, explaining the points for and against each one, and then making the optimal decision for this work. This section includes a comparative table that summarizes the main characteristics of each proposal analysed.

Chapter 4 presents the methodology and analysis of the project. First, different diagrams that have been essential to structure the bot are explained. Then, the internal operation of Rasa is revised, delving into the content of each file. Finally, the deployment of Rasa will be discussed, as well as the connection between the tool, Telegram, and a web server.

Chapter 5 describes the experiments that were carried out to evaluate the effectiveness of the proposed bot's programming. Results are later presented.

Chapter 6 shows the main conclusions of this work and future lines of research.

Chapters 7 and 8 are the English translations of the Introduction and the Conclusions.

Capítulo 8

Conclusions and Future Work

8.1. Conclusions

The aim of this work was to program an assistant that would help users solve the doubts they had within the National Police Electronic Headquarters. Initially, the sections to be covered were not defined. Nevertheless, the bot responds to every task concerning immigration matters, and renewal or application for a passport or ID card.

Several tools capable of developing conversational assistants have been researched, concluding that the technology to be used would be Rasa Stack, as a simple yet complete tool. In addition to this, it is a tool that allows you to program all the functionalities that an assistant has, while other applications were more limited and had little support.

Personally, developing a conversational assistant with an application that was unknown to me has been quite a challenge. But on the other hand, it has been an interesting process to get to know this kind of application and, above all, to learn throughout these six months how to use it. Additionally, I have been able to immerse myself in some of the procedures that are carried out within the environment of the National Police Electronic Headquarters. I feel this has made me grow as a computer engineer, carrying out a project requested by the National Police themselves. I also liked to begin my work within the field of *Artificial Intelligence* (AI); one that I had not yet been able to study during my degree.

This is, undoubtedly, a tool that will have more importance and prominence in the future, since AI will be paramount in future technology, and what better way than to help humanity. I am glad to have been part of this project and, above all, to help solve some of the doubts that might arise for users within the Electronic Headquarters.

8.2. Future Work

Thanks to the requirements that characterise Rasa, it has been possible to successfully develop an assistant, suitable for future works that shall be carried out on it. However, it can be improved in the following ways:

- Adding more intents or entities, in order to answer more questions about other procedures available at the Electronic Headquarters.
- Develop the graphic part of the assistant

- Link the bot to other applications, i.e., WhatsApp.
- Automate tool deployment.

Apéndice A

Información sobre Rasa Stack

A.1. Componentes de Rasa Stack

A.1.1. Entidades, Intents y Slots

Las entidades o *entities* son críticas en el caso de que se quiera dar una respuesta personalizada al usuario. Por ejemplo, si tenemos un *intent* asociado a consultar las reuniones que tiene cierta persona en un día, uno de los ejemplos podría ser "¿Qué reuniones tiene Marcos {*nombre_persona*} el día 23 de mayo{*fecha*}?", donde *nombre_persona* y *fecha* es una *entity* que servirá para buscar las reuniones que tiene esa persona en ese día concreto.

Por otro lado, los *intents* son las intenciones que tiene el usuario cuando interactúa con el chatbot. El usuario puede utilizar diferentes expresiones y palabras para declarar el mismo *intent*. Por ejemplo, una persona puede saludar de varias maneras, diciendo hola, hey, cómo estás, etc.

Algoritmo A.1: Intents

```
nlu:
- intent: start
  examples: |
    - Hola
    - Buenos días
    - Buenas
    - Muy buenas
    - start
    - /start

- intent: despedir
  examples: |
    - Hasta luego
    - Nos vemos
    - Adiós
    - Chao
    - Ten buen día
```

Si se quisiera almacenar datos de una conversación, se tienen que utilizar los *slots*. Son elementos parecidos a las entidades pero tienen la capacidad de guardar palabras claves de los mensajes del usuario. Por ejemplo, si se crea un chatbot con el que se puede pedir

cita médica; la fecha, la hora y el nombre del cliente serían *slots*, porque son datos que interesan guardar.

Algoritmo A.2: Slots

```
slots:
  nombre_tramite:
    type: text
```

A.1.2. Sinónimos y Tablas de Búsqueda

También existen la opción de utilizar sinónimos y tablas de búsqueda a la hora de analizar un *intent* del usuario.

Los sinónimos se aplican para definir dos entidades que tienen el mismo significado Estos elementos suelen ser usados para tratar abreviaturas o cuando se usará para la entrada de una Action.

Las tablas de búsqueda son muy útiles para reducir el número de ejemplos que aportamos al asistente para entrenar. Permiten definir palabras que pertenezcan al mismo grupo aunque tengan significados diferentes. Se consideran grupo cuando se obtiene la misma respuesta por parte del chatbot.

A.1.3. Historias para el Entrenamiento del NLU

Las historias de Rasa [GS+19] sirven para entrenar los modelos de gestión de diálogo del proyecto. Representan conversaciones entre un usuario y un asistente; además tienen una estructura donde las entradas del usuario se expresan como *intents* mientras que los mensajes del asistente se expresan como nombres de acción o respuestas correspondientes (*utterances*).

Los intentos del usuario no tienen ninguna marca delante, simplemente se identifican con la palabra *intent* y las respuestas del bot se nombran con la palabra *action*.

Algoritmo A.3: Stories

```
stories:

- story: despues-ok
  steps:
  - intent: confirmar
  - action: utter_volver
  - intent: negar
  - action: utter_despedir

- story: despues-ok2
  steps:
  - intent: confirmar
  - action: utter_volver
  - intent: confirmar
  - action: utter_cuentame
```

A.2. Estructura de Rasa Stack

Cuando se crea un proyecto de Rasa desde la terminal, tiene un diseño inicial formado por los siguientes ficheros:

- nlu.yml → apartado donde se clasifican los intentos con las frases que tiene para entrenar el chatbot
- rules.yml → sección donde se especifica el patrón que debe de seguir el chatbot para dar una respuesta en un específico grupo de intentos.
- stories.yml → parte donde se escriben todos los flujos de las historias, para que el chatbot pueda seguir estos patrones.
- test_stories.yml → su función es servir de guía para realizar pruebas sobre el chatbot. Muestra como el patrón que debes de seguir y lo que debe de ir respondiendo el chatbot.
- domain.yml → sección donde se encuentran la respuestas del chatbot que da a ante las interacciones del usuario.
- creedentials.yml → parte dónde se especifica el token y los datos necesarios para realizar una conexión con un enlace web externo.
- actions.py → el objetivo de esta sección es la creación de funciones que permitan automatizar las respuestas por parte del asistente y además sean respuestas más personalizadas ante el mensaje del usuario.

Bibliografía

[AA15] Bayan AbuShawar and Eric Atwell. Alice chatbot: Trials and outputs. *Computación y Sistemas*, 19(4):625–632, 2015.

[AC21] Luis Estuardo Azurdia Cárcamo. *Definir una guía teórica y práctica para poder implementar asistentes PhD thesis, Universidad de San Carlos de Guatemala, 2021.*

[Ali16] *Abder-Rahman Ali. How to work with json data using python. https://acortar.link/FU2lCW, 2016.*

[ara20] *aravindpai. What is tokenization in nlp? here's all you need to know. t.ly/hRJG, 2020.*

[ARA22] *ARAG. Arag, pionera en el lanzamiento de un chatbot de seguros de viaje en españa. https://www.arag.es/conocenos/sala-de-prensa/00165/, 2022.*

[Bab21] *Babylon. Babylon app. https://www.babylonhealth.com/en-us/what-we-offer/chatbot, 2021.*

[BBV22] *BBVA. ¿cómo es blue, el nuevo asistente virtual de bbva? https://www.bbva.es/finanzas-vistazo/ef/banca-digital/como-es-blue-el-nuevo-asistente-virtual-de-bbva.html, 2022.*

[CC20] *Álvaro Castillo Cabero. Rasa framework-análisis e implementación de un chatbot. B.S. thesis, Universitat Politècnica de Catalunya, 2020.*

[CI22] *Github CI. Components rasa docs. https://rasa.com/docs/rasa/components/#language-models, 2022.*

[CIO17] *CIO. La universidad ceu implanta un bot que mejora la relación con sus alumnos. https://www.ciospain.es/educacion/la-universidad-ceu-implanta-un-bot-que-mejora-la-relacion-con-sus-alumnos, 2017.*

[Com19] *Chat Compose. Chatbots para la educación: Aplicaciones y beneficios. https://www.chatcompose.com/chatbots-educacion.html, 2019.*

[cru22] *crunchbase. Youper. https://www.crunchbase.com/organization/youper/technology, 2022.*

[DCLT18] *Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding.* arXiv preprint arXiv:1810.04805*, 2018.*

[Der10] *OV Derygina. Chatterbots.* Scientific and Technical Information Processing*, 37(2):143–147, 2010.*

[Die21] *Erin Dietsche. Northwell health teams up with conversa for ai patient engagement chatbot. https://medcitynews.com/2018/10/northwell-health-conversa/?rf=1, 2021.*

[dit18] *ditrendia. Chatbots: estadísticas y ejemplos de uso en banca y seguros.* https://mktefa.ditrendia.es/blog/chatbots-banca-y-seguros, *2018.*

[dV21] *Gil del Valle. Top 5 de los mejores chatbots de asistencia médica.* https://acortar.link/MZFAM3, *2020-2021.*

[GBFAMC18] *Guillem Garcia Brustenga, Marc Fuertes Alpiste, and Núria Molas Castells. Briefing paper: los chatbots en educación. 2018.*

[Git22] *Github. Websites for you and your projects.* https://pages.github.com/, *2022.*

[GLPC21] *Maikol Javier Gilces León and Christian Fernando Pin Cevallos. Duplicidad de código en stack overflow en español. B.S. thesis, Universidad de Guayaquil. Facultad de Ciencias Matemáticas y Físicas . . . , 2021.*

[Gro21] *IT Digital Media Group. La banca ahorrará hasta 7.300 millones de dólares a través de los chatbots.* https://www.ittrends.es/negocios/2019/02/la-banca-ahorrara-hasta-7300-millones-de-dolares-a-traves-de-los-chatbots, *2021.*

[GS+19] *José María Gutiérrez Siliceo et al. Desarrollo de chatbots con entornos de código abierto. 2019.*

[HD16] *Pol Hernández Delgado. Programación de un [ro] bot para una consulta interactiva de la información de un proyecto. 2016.*

[Hea22] *Healthily. Healthily.* https://www.livehealthily.com/, *2017-2022.*

[Her21] *Noelia Hernández. Los chatbots: un nuevo recurso para el aula.* https://acortar.link/pzbr9q, *2020-2021.*

[ith21] *ipglobal tech hub. ¿cómo entiende un bot? nlu en rasa.* https://www.ipglobal.es/como-entiende-un-bot-nlu-en-rasa/, *2021.*

[Jan17] *Srini Janarthanam.* Hands-on chatbots and conversational UI development: build chatbots and voice user interfaces with Chatfuel, Dialogflow, Microsoft Bot Framework, Twilio, and Alexa Skills. *Packt Publishing Ltd, 2017.*

[Joa21] *Matthieu Joannon. Botfront.* https://github.com/botfront/rasa-webchat, *2021.*

[kee21a] *keen.ethics. Oneremission - making the lives of cancer survivors easier.* https://keenethics.com/project-one-remission, *2021.*

[kee21b] *keen.ethics. Youper.* https://www.youper.ai/, *2021.*

[KKKS17] *Diksha Khurana, Aditya Koli, Kiran Khatter, and Sukhdev Singh. Natural language processing: State of the art, current trends and challenges.* arXiv preprint arXiv:1708.05148, *2017.*

[LBP+18] *Tarun Lalwani, Shashank Bhalotia, Ashish Pal, Vasundhara Rathod, and Shreya Bisen. Implementation of a chatbot system using ai and nlp.* International Journal of Innovative Research in Computer Science & Technology (IJIRCST) Volume-6, Issue-3, *2018.*

[Ló20] *José María López. Crea tu propio bot de telegram sin saber programar.* https://blogthinkbig.com/crear-bot-de-telegram-botfather/, *2020.*

[Nac22] *Policia Nacional. Sede electrónica del cuerpo nacional de policía.* https://sede.policia.gob.es/portalCiudadano/, *2022.*

[NEW18] *Allstate NEWSROOM. Just ask abie: Allstate business insurance shares an innovative tool to help small business owners consumers with top-of-mind questions.* t.ly/rSAx, *2018.*

[One22] *Capital One. Meet eno. https://www.capitalone.com/digital/eno/, 2022.*

[oT17] *Massachusetts Institute of Technology. Babylon app. https://www.babylonhealth.com/en-us/what-we-offer/chatbot, 2017.*

[PA+19] *Sof Pizarro Arbelo et al. Bot conversacional detector de sentimientos. 2019.*

[PÁPGMAV21] *Daniel Povedano Álvarez, Javier Portela García-Miguel, and Esteban Alejandro Armas Vega. Estudio de la percepción pública de la vacuna contra la covid-19 mediante técnicas de pln y de aprendizaje automático. 2021.*

[PCFR16] *Maria João Pereira, Luísa Coheur, Pedro Fialho, and Ricardo Ribeiro. Chatbots' greetings to human-computer communication.* arXiv preprint arXiv:1609.06479, *2016.*

[PG+20] *Verónica Pinilla Gómez et al. Diseño e implementación de un chatbot para el software de idbox. 2020.*

[PG21] *Marcos Pino Gamazo. Desarrollo de un chatbot para público infantil para clasificar sonidos del cielo. versión 2. 2021.*

[RAMI17] *AM Rahman, Abdullah Al Mamun, and Alma Islam. Programming challenges of chatbot: Current and future prospective. In* 2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC), *pages 75–78. IEEE, 2017.*

[Rob22] *SoftBanck Robotics. Pepper y nao, robots para la enseñanza. https://www.softbankrobotics.com/emea/es/pepper-and-nao-robots-education, 2022.*

[SSA+18] *Yasir Ali Solangi, Zulfiqar Ali Solangi, Samreen Aarain, Amna Abro, Ghulam Ali Mallah, and Asadullah Shah. Review on natural language processing (nlp) and its toolkits for opinion mining and sentiment analysis. In* 2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS), *pages 1–4. IEEE, 2018.*

[ST20] *Siddhant Singh and Hardeo K Thakur. Survey of various ai chatbots based on technology used. In* 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO), *pages 1074–1079. IEEE, 2020.*

[Tea21] *Inbenta Team. Chatbots bancarios: un imprescindible para la atención a cliente. https://acortar.link/uMsOWe, 2021.*

[TH50] *Alan M Turing and J Haugeland. Computing machinery and intelligence.* The Turing Test: Verbal Behavior as the Hallmark of Intelligence, *pages 29–56, 1950.*

[VLRL19] *Rubén Darío Villamizar Laguado and Daniel Fernando Rico León. Chatbot para la búsqueda y comparación de productos tecnológicos a través de la plataforma messenger. 2019.*

[Wei66] *Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine.* Communications of the ACM, *9(1):36–45, 1966.*