



Universidad Politécnica de Madrid
Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

**Desarrollo de un ChatBot para Público
Infantil para Clasificar Sonidos del Cielo**

Autor: Jhosef Anderson Cardich Palma

Tutora: Raquel Cedazo León

Madrid, junio 2020

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Título: Desarrollo de un ChatBot para Público Infantil para Clasificar Sonidos del Cielo

Mes Año: junio de 2020

Autor: Jhosef Anderson Cardich Palma

Tutor: Raque Cedazo León

ETSI Informáticos

Universidad Politécnica de Madrid

Resumen

El proyecto tiene como objetivo el diseño y desarrollo de una arquitectura para la implementación de una aplicación, que tiene la principal funcionalidad de la clasificación de sonidos del cielo mediante un juego orientado al público infantil.

Entre las tareas del proyecto, se encuentra la definición y desarrollo de un asistente inteligente, el cual tendrá la característica de poder comunicarse con el usuario de la manera más natural posible. Otra tarea importante es la integración de dicho asistente con terceras aplicaciones. Se presenta también la descripción de tecnologías estándar, *frameworks*, patrones de diseño, adecuados para poder implementar dicho asistente virtual.

Para el diseño de la aplicación se ha empleado el patrón de diseño Modelo Vista Controlador (MVC), consiguiendo así una estructura robusta y clara de todos los componentes del proyecto.

En cuanto al desarrollo del asistente inteligente se ha seguido una implementación marcada por el *framework* de código abierto Rasa, del cual se han utilizado todas sus características para plasmar todos los requisitos recogidos para las funcionalidades de la parte conversacional del proyecto.

También se ha diseñado la lógica del juego para la cual se ha usado el lenguaje de programación Python. Esta se encarga de los cálculos, procesamientos necesarios y el acceso a los datos (sonidos, clasificaciones) que las funcionalidades requieran.

Otra de las tareas descritas es el desarrollo de una aplicación frontal para dotar de una interfaz en la cual el usuario pueda interactuar de manera amigable con el asistente. En el componente de la interfaz de proyecto se ha realizado una aplicación web basada en el *framework* web Django. Esta aplicación tiene la capacidad de capturar los mensajes del usuario para enviárselos al asistente, y así mismo transmitir los mensajes del asistente al usuario.

En cuanto al funcionamiento del juego, se tienen tres opciones principales: Aprendizaje, entrenamiento y clasificación. En el aprendizaje, el usuario podrá ser capaz de aprender los cinco tipos de sonidos de meteoros que existen. En la opción de entrenamiento se presentan los cinco tipos de sonidos y se le pide al usuario que intente clasificarlos. La última opción es la de clasificación, se presentan sonidos de cualquier tipo para que el usuario intente clasificarlos; cuando el usuario termine de clasificar un sonido, se guardara la clasificación y recibirá información sobre el tipo de clasificación que han ido dándole otros usuarios a dicho sonido.

Al final de ese documento se presentan otras características interesantes con una visión de desarrollo a futuro, como la implementación de un sintetizador de voz para el habla del asistente, y la captura de la voz del usuario, con el objetivo de facilitar la accesibilidad a las personas con discapacidades visuales etc.

Abstract

The project aims to design an architecture and develop an application for the classification of sky sounds using a game aimed at children.

Among the tasks of the project is the definition and development of an intelligent assistant, which, endowed with the ability to communicate with the user in the most natural way possible, in addition to the vision of ensuring integration with third-party applications. Research is also presented on what standard technologies, frameworks, and design patterns can be used to implement said virtual assistant.

For the design of the application, the Design Model Controller View (MVC) has been followed, thus achieving a clear structure of all the components of the system.

For part of the smart assistant module, an implementation marked by the Rasa open source framework has been followed, from which all its features have been used to implement all the requirements of the conversational part (smart assistant) of the project.

The game logic has also been designed and implemented using the Python programming language. It is responsible for the calculations, necessary processing, and access to the data (sounds, classifications) that the user requires.

Another of the tasks described is the development of a front-end application, to provide an interface with which the user can interact in a friendly way with the assistant. For the project interface, a web application based on the Django web framework has been made. This application can capture user messages to send to the assistant and to transmit the messages from the assistant to the user as well.

Regarding the operation of the game, there are three main options: Learning, training, and classification. In learning, the user will be able to learn the five types of meteor sounds that exist. In the training option, the five types of sounds are presented, and the user is asked to try to classify them. The last option is the classification, sounds of any type are presented for the user to try to classify them. When the user finishes classifying a sound, they will receive information about the type of classification that other users have been giving them.

At the end of this document, other interesting characteristics are presented with a vision for future development.

Tabla de Contenidos

I	Resumen	
II	Abstract	
III	Tabla de Contenidos	
IV	Índice de Tablas	
V	Índice de Ilustraciones	
1	Introducción	1
1.1	Motivación del proyecto.....	1
1.2	Contexto	2
1.3	Objetivos.....	3
2	Estado del Arte.....	5
2.1	Asistentes Virtuales	5
2.2	¿Qué son los Asistentes Virtuales?	6
2.3	Arquitectura Genérica de un Asistente Virtual	9
3	Desarrollo.....	10
3.1	Requisitos	10
3.2	Planteamiento de la Arquitectura	13
3.2.1	MVC - Diseño de Componentes del Controlador/Modelo.....	14
3.2.2	MVC - Esquema de Diseño para el Frontal-Vista del Proyecto	15
3.2.3	MVC – Descripción de la Aplicación Acceso para el acceso de Datos	16
3.2.4	Requerimientos y herramientas	18
3.3	Implementación de los Módulos del Controlador del Proyecto.....	19
3.3.1	Módulo Controlador - Definición del Asistente.....	19
3.3.2	Rasa Framework	19
3.3.2.1	Rasa NLU.....	19
3.3.2.2	Rasa Core	20
3.3.2.3	Rasa NLG	20
3.3.2.4	Procesamiento de un mensaje en Rasa.....	20
3.3.2.5	Componentes Elementales de Rasa	21
3.3.2.5.1	Intents.....	21
3.3.2.5.2	Utterances.....	22
3.3.2.5.3	Actions.....	22
3.3.2.5.4	Stories.....	23

3.3.2.5.5 Slots.....	24
3.3.2.5.6 Dominio	24
3.3.2.5.7 Implementación de Rasa para el Asistente – Storie	24
3.3.3 Módulo Modelo/Controlador – Servidor de Acciones del Juego	26
3.3.3.1 Implementaciones de funcionalidades.....	26
3.3.4 Integración entre el Módulo de Acciones y el Asistente.....	29
3.3.4.1 Función Aprendizaje y Función de Entrenamiento.....	29
3.3.4.2 Función Clasificación.....	29
3.3.4.3 Pruebas de Integración.....	30
3.3.4.3.1 Pruebas	32
• Saludo y reconocimiento:	32
• Función de Aprendizaje:	32
• Prueba de Entrenamiento 1	33
• Prueba de Entrenamiento 2	33
• Prueba de Clasificación.....	34
• Prueba de Clasificación 2:.....	37
• Prueba Información de Ayuda en Clasificación.....	38
• Prueba Información de Ayuda en Clasificación.....	38
• Prueba Información de Ayuda en Clasificación.....	39
• Prueba Información de Ayuda General del Sistema	39
3.3.5 Módulo Vista - Aplicación Frontal Web del Proyecto	40
3.3.5.1 Frontal Web del Proyecto – Django.....	40
3.3.5.2 Diseño Visual del Frontal.....	42
3.3.6 Interacción entre el módulo de Conversacional y el Frontal.....	44
3.3.6.1 Características de los mensajes entre módulos	44
3.3.6.2 Pruebas de integración	44
• Saludo y reconocimiento:	44
• Opciones de la Aplicación:.....	45
• Fase de Aprendizaje	46
• Prueba de Entrenamiento:	47
• Prueba de Clasificación:.....	48
• Prueba de Ayuda en la Clasificación:.....	50
• Prueba otros mensajes:.....	51
3.3.7 Tecnologías para una implementación de Voz en el frontal.....	52
3.3.7.1 Voz del Asistente - Google Cloud Text to Speech	52
3.3.7.2 Voz del Usuario a Texto - Google Speech to Text.....	52

Funcionamiento del Sistema	53
3.3.8 Comandos y Funcionalidades.....	53
3.3.8.1 Bienvenida	53
3.3.8.2 Aprendizaje sobre los Sonidos del Cielo	53
3.3.8.3 Selección de un Sonido en Concreto para su aprendizaje	53
3.3.8.4 Entrenamiento	54
3.3.8.5 Clasificación	54
3.3.8.6 Ayuda en la Clasificación.....	54
3.3.8.7 Ayuda General	54
3.3.9 Organización de los Directorios del Proyecto	54
4 Conclusiones y Propuestas de Ampliación	56
4.1 Conclusiones	56
4.2 Propuestas de Mejora.....	57
5 Bibliografía.....	59
6 Anexos	61
6.1 Información de Configuraciones del Proyecto	61
6.1.1 Repositorio del Proyecto	61
6.1.2 Configuración de los Servidores del proyecto.....	61
6.1.2.1 Endpoints del servidor de acciones y del Asistente en Rasa.....	61
6.1.2.2 El Servidor Django	62

Índice de Tablas

Tabla 1. Requisitos Generales del Proyecto.	10
Tabla 2. Asociación entre actions y clases asociadas para la funcionalidad de Aprendizaje	29
Tabla 3. Asociación entre actions y clases asociadas para la funcionalidad de clasificación.	30
Tabla 4. Correspondencia entre los nombres de sonidos del cielo y su clasificación en el sistema.....	48

Índice de Ilustraciones

Ilustración 1. Lluvia de Estrellas. Fotografía de Casey Horner.	2
Ilustración 2. Captura de ondas reflejadas sobre un meteoro.	3
Ilustración 3. Google Assistant en un teléfono, esperando comandos del usuario	5
Ilustración 4. Asistente KLM dentro de Facebook Messenger.	6
Ilustración 5. Siri de Apple agendando un evento en el calendario.	7
Ilustración 6. Asistente virtual de Renfe.	7
Ilustración 7. Altavoz Google Home con Google Assistant.	8
Ilustración 8. Amazon Echo con Alexa.	8
Ilustración 9. Diagrama de elementos que componen un Asistente Virtual.	9
Ilustración 10. Componentes del Proyecto en los Componentes del Patrón MVC.	13
Ilustración 11. Componentes del Controlador. Incluyen elementos del Modelo.	14
Ilustración 12. Elementos dentro del Componente Vista del Patrón MVC.	15
Ilustración 13. Elementos que forman parte del Modelo del Patrón MVC.	16
Ilustración 14. Modelo Vista Controlador del Proyecto.	17
Ilustración 15. Diagrama del flujo de procesamiento de un mensaje.	21
Ilustración 16. Definición del intent “greet” dentro del archivo nlu.md.	22
Ilustración 17. Definición de la utterance “utter_happy” y todas sus posibles respuestas, dentro del archivo domain.yml.	22
Ilustración 18. Definición del action “action_classifying” dentro del dominio.	23
Ilustración 19. Uso del action “action_classifying” dentro de la Storie “clasificación”	23
Ilustración 20. Definición de algunas entidades y slots dentro de dominio.	24
Ilustración 21. Definición de la storie para empezar el dialogo con el asistente, en el archivo stories.md	25
Ilustración 22. Intents posibles, asociados a la storie anterior para empezar el dialogo con el asistente, en el archivo nlu.md.	25
Ilustración 23. Generación de un mensaje JSON que contiene la URI del sonido.	26
Ilustración 24. Archivo de clasificación asociado al archivo de audio “sound8.wav”.	27
Ilustración 25. Archivo de clasificación asociado al archivo de audio “sound1.wav”.	28
Ilustración 26. Progreso de entrenamiento del asistente con los datos de entrenamiento	30
Ilustración 27. Mensaje de confirmación de la generación del modelo.	31
Ilustración 28. Rasa Shell donde se puede interactuar con el asistente.	31
Ilustración 29. Rasa Actions corriendo en el servidor.	31
Ilustración 30. Saludo y captura del nombre del usuario por medio del shell de Rasa.	32
Ilustración 31. Formato de datos de los mensajes en la comunicación para el aprendizaje.	32
Ilustración 32. Respuesta del asistente cuando se identifica la funcionalidad “Aprender”.	33
Ilustración 33. Reconocimiento de la función “Entrenamiento”	33
Ilustración 34. Acierto en la prueba de entrenamiento.	34

Ilustración 35. Listado de generación de archivos de clasificación de los sonidos disponibles.	34
Ilustración 36. Archivos de clasificación de sonidos en el directorio de los sonidos.....	35
Ilustración 37. Datos en un mensaje de clasificación.	36
Ilustración 38. Datos de clasificación del sonido 8.....	36
Ilustración 39. Mensaje después de una clasificación.	37
Ilustración 40. Actualización de los datos de clasificación del sonido 8.	37
Ilustración 41. Elección de otro sonido para su clasificación.	37
Ilustración 42. Respuesta del asistente para una clasificación inválida.	38
Ilustración 43. La funcionalidad de ayuda muestra información para guiar al usuario... Ilustración 44. Función de reproducción de un sonido en concreto.	38
Ilustración 45. Entrenamiento como ayuda para la clasificación.	39
Ilustración 46. Ayuda general del sistema.	39
Ilustración 47. Esquema general del funcionamiento de un proyecto en Django.....	40
Ilustración 48. Diseño visual del concepto Chat para el asistente del proyecto.....	42
Ilustración 49. Aspecto general de la apariencia visual en el frontal.	43
Ilustración 50. Aspecto final del frontal.	43
Ilustración 51. Formato de mensaje para la funcionalidad de bienvenida.	44
Ilustración 52. Aspecto del frontal para la bienvenida al usuario.	45
Ilustración 53. Información para las opciones del sistema.	45
Ilustración 54. Información mostrada en los mensajes del frontal....	46
Ilustración 55. En Postman, información para clasificar un sonido.	46
Ilustración 56. Reproductor de sonido para el recurso anterior.....	47
Ilustración 57. Datos para el entrenamiento.	47
Ilustración 58. Respuesta correcta en un entrenamiento.	48
Ilustración 59. Clasificación errónea del usuario.	48
Ilustración 60. Solicitud de otro sonido para su clasificación.	49
Ilustración 61. Mensaje de clasificación inválida.	49
Ilustración 62. Ayuda en la clasificación.	50
Ilustración 63. Otras opciones de ayuda en la clasificación.	50
Ilustración 64. Elección de un sonido de ayuda en la clasificación.....	51
Ilustración 65. Mensaje de ánimos al usuario.	51
Ilustración 66. Logo del Sintetizador de voz de Google.	52
Ilustración 67. Organización de los directorios del proyecto.	55
Ilustración 68. Endpoint del Servidor de Acciones.	61
Ilustración 69. Configuración del cliente definido en chat.js.	62
Ilustración 70. Negociación de la Operación POST.	62

1 Introducción

1.1 Motivación del proyecto

Desde el campo de la inteligencia artificial, la interacción entre las personas y los ordenadores lleva ya un tiempo implantándose de manera asidua, evolucionando junto con las necesidades de los consumidores tecnológicos, o también por la automatización de trabajos en diferentes tipos de sectores. Estos consumidores ya no solo son personas con conocimientos técnicos acerca de tecnologías, nos podemos encontrar con muchos tipos de perfiles debido a que la tecnología está revolucionando la manera en la que los humanos realizamos ciertas tareas que llevan ya implantadas desde los inicios de las civilizaciones, como es el traspaso de conocimientos para el aprendizaje, es decir, la educación. El acceso a la información para la educación, o la manera en la que la enseñanza ha ido cambiando, se ha ido adaptando para hacer frente a los problemas que pueden presentar los estudiantes, ya sean por motivos de acceso debido a distancias geográficas o de desplazamiento, accesibilidad para personas discapacitadas, etc. Solamente hay que pensar como suele ser la enseñanza desde siempre: presencial, mediante un profesor, en un aula, con alumnos escuchándole y resolviendo las dudas que puedan surgir.

Pero imaginemos ¿Y si no pudiéramos ir clases por algún motivo?, ¿Y si no pudiéramos ser capaces de utilizar los medios físicos de aprendizaje para estudiar? O ¿Y si tuviéramos al profesor en cualquier momento y en cualquier lugar? También debido a la situación personal de cada estudiante, un reto bastante importante es la adaptación de la educación para las personas que presentan discapacidades físicas, como pueden ser las personas ciegas. La visión tecnológica que tenemos de la sociedad con la implantación de asistentes virtuales presenta un panorama futurista, estos están presentes en muchas de las tareas cotidianas que antes se hacían de manera rudimentaria, en otros casos son capaces de poder realizar tareas de manera independiente, sin ningún tipo de intervención humana. Ahora también estos toman parte en la educación. Estos asistentes son cada día más humanos, algunos pueden dotarse de capacidades, como poder interactuar mediante voz con el usuario haciendo así la experiencia más cercana y fluida. Estas herramientas tecnológicas hacen que el aprendizaje haya evolucionado a otro nivel, ya que incluso hacen uso de toda la información procesable de internet. Aprovechando estas ventajas se desea que este proyecto pueda acercar la ciencia al público en general, para que puedan permitirse interactuar con una fuente de conocimiento sobre los sonidos del cielo, de manera entretenida y bajo demanda.

1.2 Contexto

La experiencia de poder entender los elementos del espacio exterior siempre ha sido algo emocionante para los seres humanos. Por ejemplo, los meteoros son cuerpos rocosos que viajan por el espacio y han sido siempre objeto de investigación por la toda la información que pueden brindar a los científicos para desarrollar sus investigaciones y entender así el espacio infinito que nos rodea. La astronomía y su comunidad tienen que entender las características de los meteoros, saber clasificarlos. Uno de los fenómenos astronómicos más espectaculares en los que podemos disfrutar los meteoros de manera visual, son las lluvias de estrellas.



Ilustración 1. Lluvia de Estrellas. Fotografía de Casey Horner.

Pero ¿podríamos ser capaces de poder escuchar esa lluvia de meteoros? La tecnología puede ayudar en esto, aprovechados de las características de este fenómeno. Pero para ello hay que entender un poco mejor que es lo que pasa cuando observamos el rastro de una estrella fugaz en el cielo. Cuando un meteoro entra en la atmósfera, este lo hace con gran velocidad y teniendo una gran fricción con la atmósfera, el rastro brillante que vemos en el cielo es debido a la ionización del aire alrededor del meteoro [1], así podemos ser capaces de ver el recorrido que hacen hasta que terminan de desintegrarse o incluso llegar a la superficie terrestre (meteoritos) [2]. Los meteoros están conformados también por metal y otros elementos, gracias a ello y con ayuda de ondas de radio reflejadas en los meteoros,

se puede capturar con un receptor los ecos de esas ondas reflejadas y transformarlas en sonido [3].

De esa manera ya podemos escuchar los meteoros. Pero ¿con qué tipos de sonidos podemos identificarlos? Se puede decir que existen cinco tipos de sonidos meteoros: Meteoros Underdense, Meteoros de tipo M, Overdense largo, Overdense medio y Overdense corto.

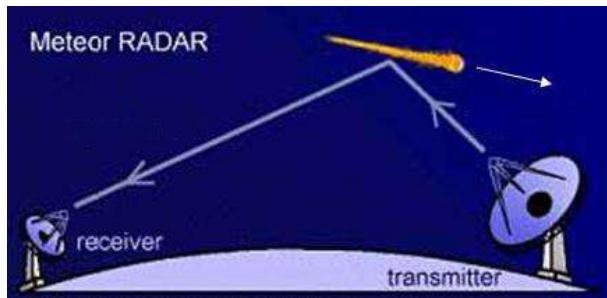


Ilustración 2. Captura de ondas reflejadas sobre un meteoro.

Es interesante y despierta curiosidad poder clasificar los sonidos del cielo y este es el objetivo de este proyecto de ciencia, con el trasfondo de que el público general forme parte de este experimento, involucrando así no solo a las personas con conocimientos técnicos sobre astrología, sino todo aquellos que estén interesados, incluyendo también a aquellas personas que presenten alguna discapacidad visual, para convertirlos también en protagonistas de este experimento de clasificación, brindándoles la experiencia de poder escuchar los sonidos del cielo. El desarrollo de esta aplicación forma parte del proyecto “[Contadores de Estrellas](#)”, impulsado por la Universidad Politécnica de Madrid y en colaboración con el Instituto Astrofísico de Canarias, con el principal objetivo de atraer al público en general y especialmente a los más jóvenes, ya que este gran grupo de usuarios son los que más usan la tecnología, y por ello podría ser un gran oportunidad de crear una gran comunidad para este proyecto

Esta aplicación será una herramienta educativa para los usuarios, los cuales podrán aprender de manera entretenida, ya que el asistente que guiara al público durante la clasificación aporta una experiencia de aprendizaje amigable y natural. Además la información que se puede recuperar de la aplicación, como son los datos de posibles clasificaciones para un sonido, servirán de provecho para cualquier otra aplicación en el estudio de los meteoros.

1.3 Objetivos

El principal objetivo de este proyecto es poder desarrollar la implementación de la inteligencia del asistente que servirá a la aplicación, siguiendo estos unos estándares dirigidos por un *framework* y enmarcando dicho asistente como un componente más dentro del modelo MVC que seguirá el proyecto. Se desea que las tecnologías usadas para la lógica del juego y el desarrollo del asistente sean compatibles con implementación de una vista (interfaz web) y además con el modelo (acceso a los datos, sonidos, con los que trabajará el proyecto). Los principales objetivos son:

- Diseño de la Arquitectura del Proyecto.
- Búsqueda de Tecnologías que satisfagan los requerimientos del Proyecto.
- Estudio de la compatibilidad entre las diferentes implementaciones.
- Adaptación del comportamiento del asistente a la tecnología a emplear.
- Implementación del Asistente Inteligente.
- Implementación de la Lógica del Juego.
- Implementación de una aplicación Frontal basada en Tecnología Web.
- Descripción de tecnologías para una implementación de diálogo por voz con el asistente.

2 Estado del Arte

2.1 Asistentes Virtuales

Antes que nada, nos tenemos que detener a conocer una de las tecnologías más interesantes del proyecto, los asistentes virtuales. Existen muchos tipos de implementaciones de este software que están presentes hoy en nuestro día a día, incluso aunque nos los usemos, por ejemplo, en electrodomésticos, coches, páginas web, etc. La mayoría de estos están implementados en muchos dispositivos, los teléfonos móviles por ejemplo cuentan con asistentes, los cuales ejecutan tareas sencillas como hacer llamadas, mandar mensajes, etc., pero la evolución de estos ha hecho que estos se vuelvan cada vez más personales, más inteligentes. Pero ¿Qué es un asistente virtual?

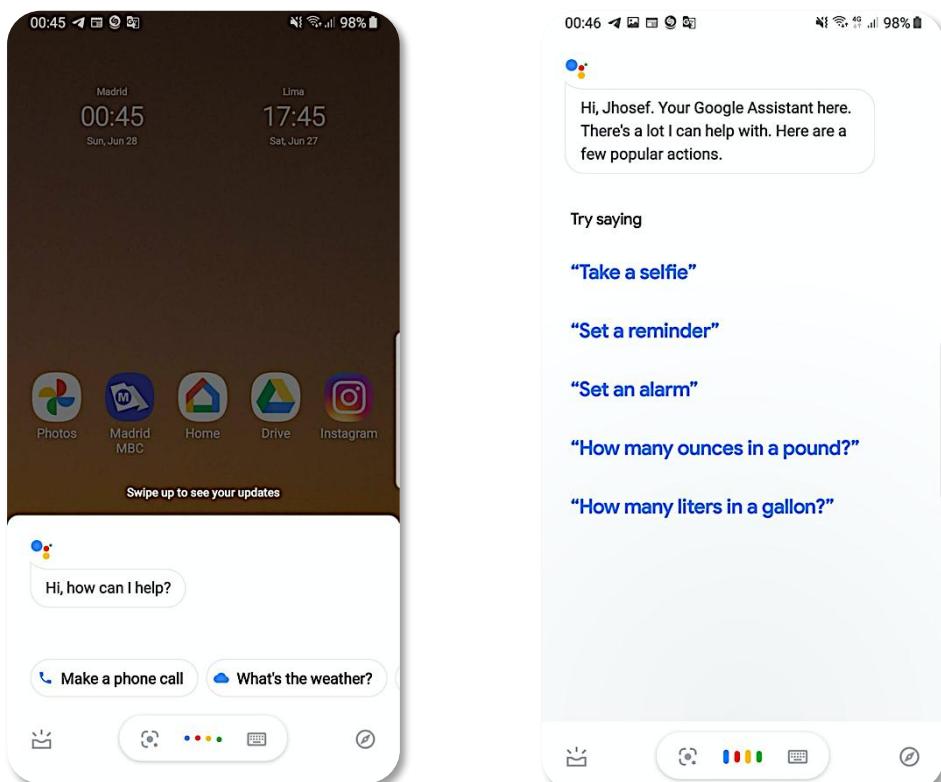


Ilustración 3. Google Assistant en un teléfono, esperando comandos del usuario

2.2 ¿Qué son los Asistentes Virtuales?

Los asistentes virtuales tienen un origen en la década de los 60's, en donde se usaban para experimentos en los que se trataba de engañar a una persona, en una conversación con ChatBot haciéndole creer que la persona hablaba con un humano [4]. ELIZA fue una implementación considerada uno de los primeros ChatBots que era capaz de reconocer palabras claves y responder con frases predefinidas para mantener una conversación [5]. Como casi toda la mayoría de las tecnologías desarrolladas presentes hoy en día, los ChatBots evolucionaron con el paso del tiempo para convertirse en asistentes, para poder cubrir necesidades de consulta de información, para guiar a través de pasos al usuario de alguna aplicación, para la realización de alguna tarea, etc. Estas tareas en principio son bastante sencillas, sin embargo, los asistentes virtuales aportaban algo más: la disponibilidad de ejecutar esas tareas en cualquier momento, es decir, ofrecer soporte al usuario de manera ininterrumpida, y el potencial de interactuar de manera natural .

Su implementación en las empresas que tienen presencia online ha aumentado con mucha rapidez. Por ejemplo, en tiempos actuales, la pandemia del COVID-19 ha obligado a todas las empresas y organizaciones a emplear medidas para hacer frente a la reorganización en el modelo en el que desarrollaban sus tareas, ahora el trabajo a distancia se ha vuelto una moda necesaria. Por ejemplo algunas organizaciones de salud italianas han implementado ChatBots para poder hacer frente a las consultas de los pacientes [6].

Así entonces el empleo de asistentes virtuales se ha visto impulsado por la necesidad de ofrecer los servicios al cliente, brindando una experiencia distinta, sin importar la ubicación o el momento, ya que pueden ser usados desde cualquier lugar y cuando el usuario lo requiera.

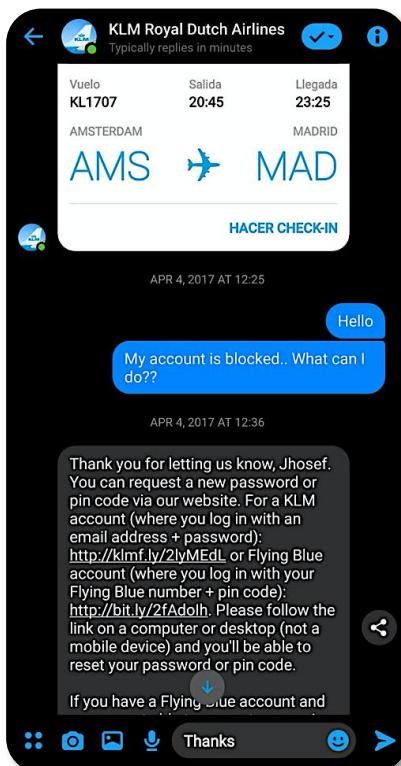


Ilustración 4. Asistente KLM dentro de Facebook Messenger.

Los asistentes virtuales funcionan mediante la comunicación con el usuario. En la idea más básica, estos tienen que ser capaces de poder entender o reconocer ciertos comandos o palabras clave para identificar el servicio o tarea que el usuario quiere realizar.

La comunicación entre el usuario y el asistente puede tener diferentes medios y formas. Se tienen asistentes con los que podemos comunicarnos de manera escrita, enviándole nuestras órdenes mediante una entrada de texto, el asistente también responderá de la misma manera. Por ejemplo, el asistente de KLM, es capaz de brindarnos soporte en cuanto a los viajes en avión, trámites con el billete del vuelo y demás gestiones de reserva. Se encuentra disponible en la aplicación de mensajería WhatsApp, pero también está



Ilustración 5. Siri de Apple agendando un evento en el calendario.

integrada dentro de la aplicación *Messenger* de Facebook. La interacción que ofrece es bastante menos robótica, como si *chateáramos* con cualquier otra persona usando este servicio de mensajería social.

Siri de Apple es otro asistente, con el que podemos tener una conversación mediante voz, el sistema es capaz de reconocer todas las tareas y de mantener conversaciones cortas gracias al tecnólogo de síntesis de voz que usa [7], de esa manera puede comprender que es lo que queremos decir y responder por voz. Con respecto a esto, hay una característica presente en los asistentes más modernos. Muchos de los asistentes se centran en ser más humanos. Por ejemplo, tenemos los asistentes conversacionales, los cuales usan sistemas de diálogos bastante amplios, permitiéndoles expresarse de la forma más humana posible,



Ilustración 6. Asistente virtual de Renfe.

haciendo uso de alguna interfaz gráfica con apariencia humana, incluso mostrando gestos, voz .

De todo lo comentado hay que tener en cuenta la inteligencia que está por detrás de estos asistentes, ya que son capaces de reconocer una infinidad de comandos, incluso muchos de ellos que no se han predefinido en el inicio de su creación. Estos también aprenden de los datos de las conversaciones de todos los usuarios que los utilizan, extrayendo patrones de las acciones que solicitan, de la manera en la que se expresan los usuarios, de sus gustos, brindándoles este aprendizaje a los asistentes, experiencia más amplia para reconocer lo que el usuario quiere y como lo quiere [8].

Hoy en día los asistentes son más personales, esto quiere decir que nos entienden cada día mejor, pues ahora aprenden de los usuarios constantemente. Esto es gracias a la implementación de la inteligencia artificial en este tipo de software, que es capaz de reconocer acciones, predecir comportamientos esperados ante un comando del usuario, basándose en la experiencia previa de infinidad de casos [9].

Existen asistentes que ahora forman parte del hogar, aportando automatización en las tareas en la administración de los servicios básicos, todo esto se refiere a otro campo comercial de la tecnología llamada domótica [10]. Estas tecnologías se encargan de la seguridad, control de energía y administración de otros dispositivos presentes en el hogar. La mayoría de estos dispositivos con asistentes, dedicados a estos trabajos se presentan en formato altavoz, y son capaces de entender comandos por voz y responder de la misma manera [11], como son los casos de Amazon Echo, el cual tiene integrado el asistente de voz Alexa y Google Nest Mini, el cual usa la tecnología Google Assistant, también presente en muchos otros productos de la marca [12].

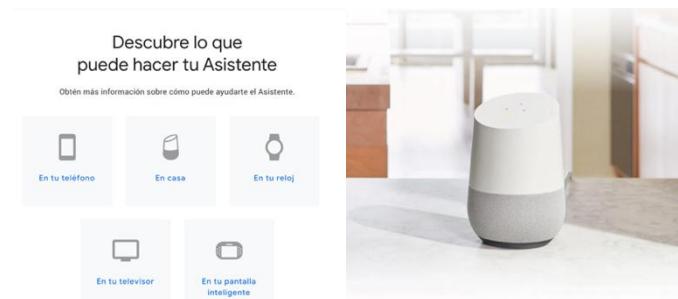


Ilustración 7. Altavoz Google Home con Google Assistant.

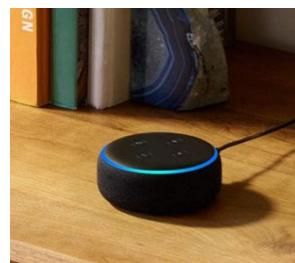


Ilustración 8. Amazon Echo con Alexa.

2.3 Arquitectura Genérica de un Asistente Virtual

Los asistentes de voz presentan en su arquitectura una serie de elementos los cuales son importantes conocer, que es de interés en este proyecto por ser la creación de un asistente un requisito fundamental. La arquitectura que presentan se puede extraer de una conversación, en este caso por voz, y de manera general los elementos que se identifican se organizan secuencialmente como se muestra en el diagrama:



Ilustración 9. Diagrama de elementos que componen un Asistente Virtual.

Podemos describir una conversación en el diagrama presentado: El usuario es el que normalmente inicia el diálogo, entonces el sistema conversacional se activa y espera más información del usuario. Cuando el usuario empieza a hablar, lo primero que suele suceder, es la captura de la voz del usuario para poder analizar el lenguaje. Cuando la tecnología apropiada para este trabajo analiza el lenguaje, devuelve dicho análisis en información procesable que es analizado por el asistente con la finalidad de reconocer la acción o comando que el usuario desea realizar, en función de esto, el asistente tendrá que decidir qué acción tomar a continuación y además modificar el flujo del diálogo para tomar un camino de la conversación. La respuesta devuelta al usuario suele ser seleccionada función de las experiencias del usuario o por el reconocimiento de alguna palabra clave, que el asistente entiende como un comando. Dentro del análisis de reconocimiento puede intervenir tecnología de aprendizaje, para poder guardar esta interacción como una experiencia más. La respuesta se genera, y luego este es pasado a un sintetizador de voz, el cual generar la respuesta hablada [13] [14].

3 Desarrollo

3.1 Requisitos

Los requerimientos de la aplicación van ligados directamente con las funcionalidades presentes dentro de la experiencia de uso que ofrecerá el juego, el componente que se encarga de la parte conversacional y el frontal de la aplicación. Por lo tanto, mencionamos de manera general, los principales requisitos a nivel de usuario y también a nivel de software que se han identificado, asociando a una tarea para la realización del proyecto.

Tabla 1. Requisitos Generales del Proyecto.

Requisito	Tareas	Componente Asociado
El usuario interactúa por el navegador web.	<ul style="list-style-type: none">• Implementación de navegación en un frontal web.• Diseño de navegación y apariencia de la experiencia visual de usuario.	<ul style="list-style-type: none">• Frontal.
El usuario es capaz de introducir texto para los comandos.	<ul style="list-style-type: none">• Captura de datos en el frontal web.	<ul style="list-style-type: none">• Frontal
El usuario interactúa con el asistente en forma de dialogo.	<ul style="list-style-type: none">• Diseño de flujos conversacionales.• Diseño de la forma en que los mensajes son presentados.	<ul style="list-style-type: none">• Frontal
El juego ofrece la opción de aprendizaje	<ul style="list-style-type: none">• Desarrollo de la lógica del Juego.• Funciones de procesamiento para el acceso a los sonidos.• Definición del flujo conversacional en asistente.	<ul style="list-style-type: none">• Frontal• Lógica de procesamiento del Juego.• Asistente conversacional.

El juego ofrece la opción de entrenamiento	<ul style="list-style-type: none"> • Desarrollo de la lógica del Juego. • Definición del flujo conversacional en el asistente. • Diseño de mensajes a intercambiar entre el frontal y mensajes • Diseño implementación de interacción entre el frontal y el asistente. • Diseño entre el asistente y la lógica del juego, para el acceso a los sonidos. 	<ul style="list-style-type: none"> • Frontal • Lógica del procesamiento del Juego. • Asistente conversacional.
El juego ofrece la opción de clasificación.	<ul style="list-style-type: none"> • Desarrollo de la lógica del Juego. • Funciones de acceso a los sonidos. • Almacenamiento de resultados. • Tratamiento de archivos de clasificación. • Definición de <i>path</i> conversacional en asistente. 	<ul style="list-style-type: none"> • Frontal • Lógica de procesamiento del Juego. • Asistente conversacional
El sistema ofrece la opción de ayuda	<ul style="list-style-type: none"> • Definición de mensajes de ayuda en los mensajes del asistente. 	<ul style="list-style-type: none"> • Asistente conversacional.
El sistema tendrá que reproducir sonidos para el usuario.	<ul style="list-style-type: none"> • Implementación de un reproductor de sonidos en el frontal. 	<ul style="list-style-type: none"> • Frontal.
El sistema tiene que ser capaz de almacenar la información de las clasificaciones.	<ul style="list-style-type: none"> • Tratamiento de respuestas en el flujo conversacional. • Almacenamiento de los datos de clasificación por la lógica del juego. • Diseño de los tipos de Archivos donde se guardan las clasificaciones. • Definiciones de funciones de escritura, lectura de archivos de texto. 	<ul style="list-style-type: none"> • Lógica de procesamiento del juego. • Asistente conversacional.

El sistema tiene que mostrar información de clasificación de los sonidos.	<ul style="list-style-type: none"> Cálculos sobre las clasificaciones existentes para un sonido. 	<ul style="list-style-type: none"> Lógica de procesamiento del juego.
El sistema tiene que ser capaz de poder desarrollar un diálogo lo más natural posible.	<ul style="list-style-type: none"> Implementación de un lenguaje natural. Definir las posibles variaciones para una conversación. 	<ul style="list-style-type: none"> Asistente conversacional
El aspecto del sistema tiene que ser amigable.	<ul style="list-style-type: none"> Diseño gráfico orientado al público infantil. 	<ul style="list-style-type: none"> Frontal
El frontal tiene que ser capaz de solicitar sonidos al asistente.	<ul style="list-style-type: none"> Diseño de mensajes para la comunicación de sonidos en JSON. Acceso mediante funciones usando JQuery. 	<ul style="list-style-type: none"> Frontal Lógica de procesamiento del Juego.

3.2 Planteamiento de la Arquitectura

Una vez se han identificado los requisitos de la aplicación, surge la necesidad de tener una organización que permita una visualización global del proyecto, en donde se ubiquen los diferentes componentes a implementarse y la interacción entre estos, con la finalidad de guiarnos en los diferentes contextos del desarrollo de cada componente.

Debido a la complejidad del proyecto, se necesita de una base sólida de diseño, en la cual basarse para concretar las ideas y requerimientos. Para conseguir esto, se ha recurrido al patrón de diseño de *software* Modelo-Vista-Controlador (**MVC**), el cual va a permitir definir los componentes necesarios del sistema. Este modelo va a organizar cada parte de la aplicación, haciendo que sea más claro, en qué lugar dentro de la arquitectura se tienen que situar las diferentes funcionalidades a desarrollarse. A continuación, se resume la correspondencia entre las funcionalidades de la aplicación y los tres componentes que conforman un proyecto que sigue este patrón.

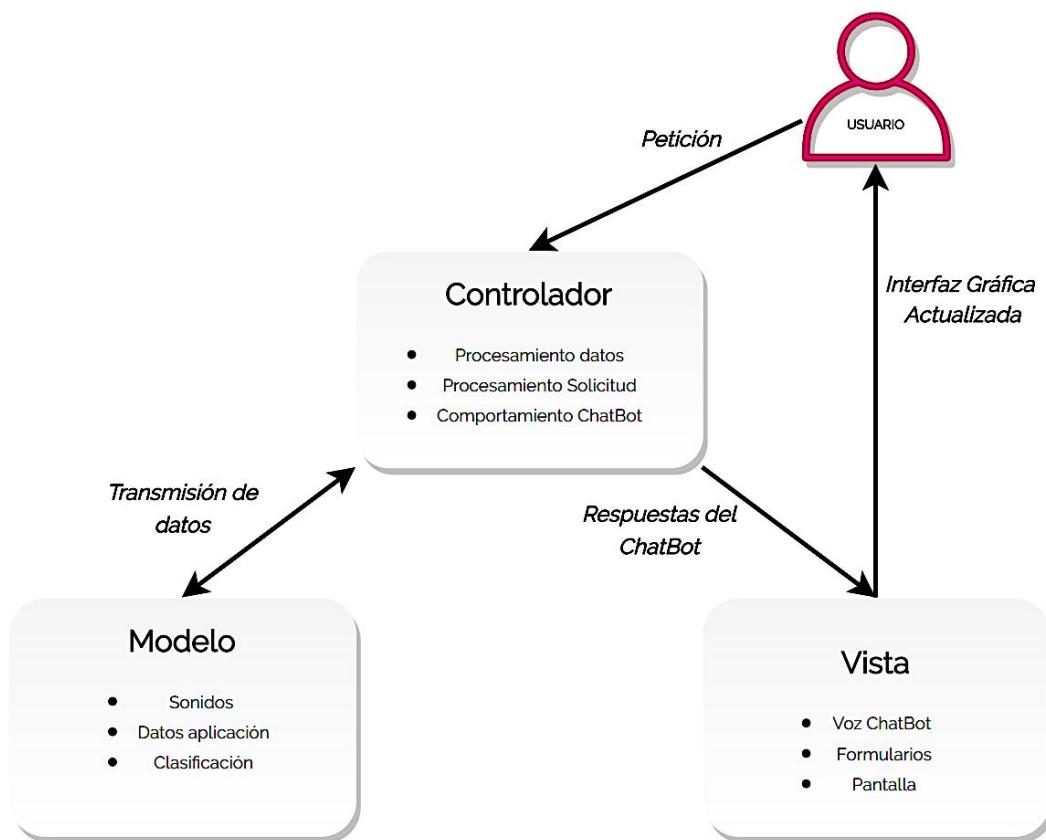


Ilustración 10. Componentes del Proyecto en los Componentes del Patrón MVC.

El Modelo-Vista-Controlador es un patrón de arquitectura que se caracteriza por la facilidad con la que se puede separar el desarrollo de los componentes de un proyecto, además de ofrecer escalabilidad para la expansión de un sistema o modificación de alguno de sus componentes sin afectar al resto.

3.2.1 MVC - Diseño de Componentes del Controlador/Modelo

El controlador de la aplicación se encarga de comunicar la vista y el modelo. En este componente del patrón se definen las funcionalidades que realizan los cálculos sobre los datos obtenidos del modelo para enviárselos a la vista, para que así puedan ser presentados al usuario. En otras palabras, aquí se encuentra la lógica de la clasificación de los sonidos, el cálculo de los resultados del usuario después de una sesión de juego, se definen las acciones que están disponibles en nuestra aplicación a las cuales el usuario podrá llamar mediante la vista.

También se encuentra la inteligencia del asistente, los manejadores que podrán contextualizar las solicitudes, los cuales definen el comportamiento del asistente ante los comandos del usuario. Se encuentran también definidas todas las funcionalidades que cubrirán todos los posibles diálogos que pueden ocurrir en una interacción entre el usuario y el asistente. Esta componente será responsable de enviar los sonidos (para la clasificación) a la vista, para que el usuario los escuche y los clasifique. La administración de los sonidos y otros datos procesados como las clasificaciones y el almacenamiento de estos, estarán a cargo de la lógica de procesamiento del juego que es parte del modelo también. La interfaz gráfica de la vista y el controlador son los objetivos de desarrollo, por ser los más componentes elementales del proyecto.

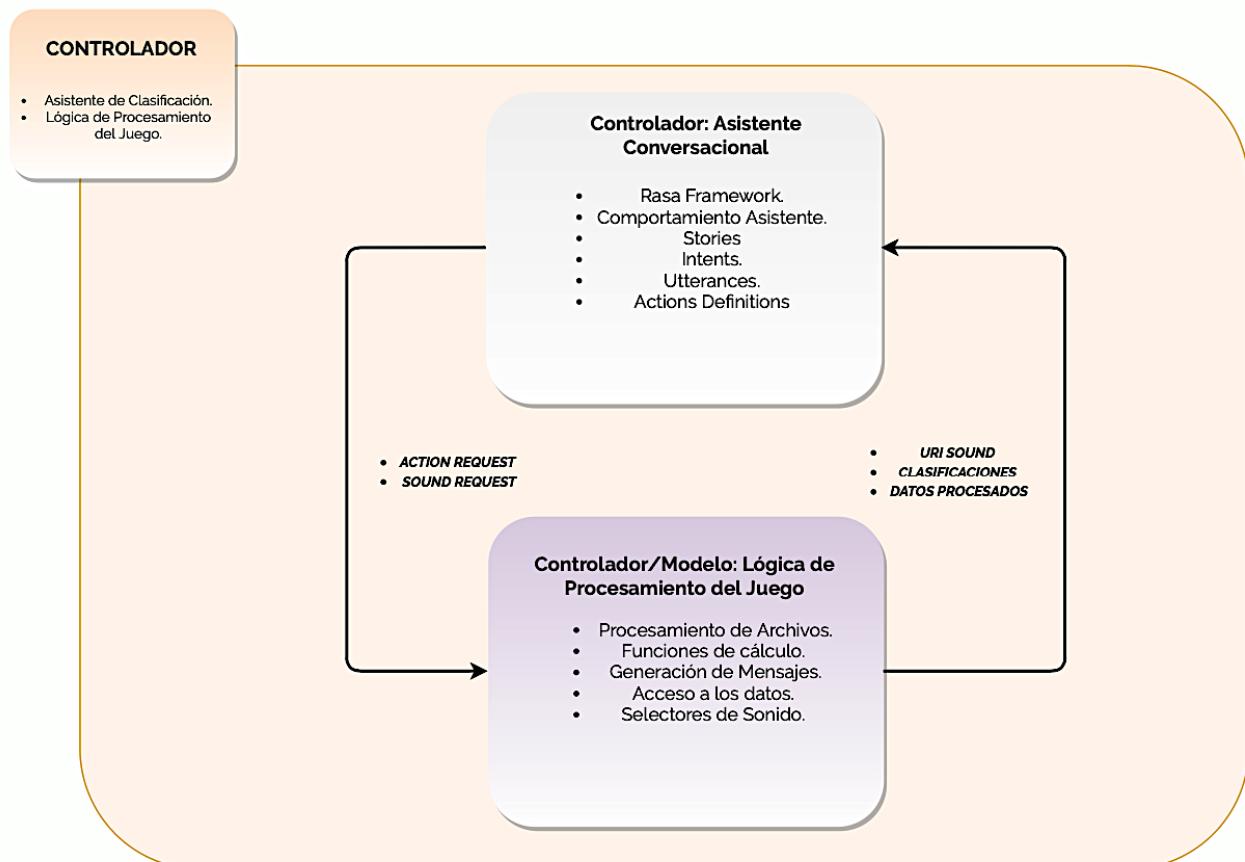


Ilustración 11. Componentes del Controlador. Incluyen elementos del Modelo.

3.2.2 MVC - Esquema de Diseño para el Frontal-Vista del Proyecto

El proyecto necesita de una interfaz con la cual el usuario tendrá que interactuar para comunicarse con el asistente. En este componente se agrupan todas las funcionalidades relacionadas con el desarrollo del *front-end*, tales como la programación de las pantallas que se mostrarán al usuario, el diseño gráfico, la ubicación de los botones, formularios y los resultados o respuestas que el asistente pueda generar por la solicitud de un usuario. Una parte muy importante de nuestra aplicación es la captura de los mensajes por parte del sistema. En el diseño de este componente del proyecto tendrían que ir ubicados la interfaz de entrada de texto o la implementación de la voz de nuestro asistente. Todo el desarrollo de las funcionalidades relacionadas con la captura de los comandos por voz del usuario y las respuestas por voz del asistente estarán en esta categoría del componente Vista que seguirá el proyecto. En cuanto al desarrollo de este componente, se realizará una aplicación web basada en el *framework* de Django, el cual nos va a permitir entre otras cosas, crear una aplicación en la que podremos realizar llamadas a REST para la solicitud de datos del asistente virtual. En esta versión del proyecto, contará con una entrada de texto, de manera que el usuario podrá tener un diálogo con el asistente usando el teclado. Para el diseño gráfico del frontal, se usará algunos componentes de Bootstrap, para dotar de calidad la maquetación del diseño. Aparte se requieren ciertas funcionalidades para dotar de movimiento la actualización cuando el usuario escriba un mensaje nuevo o reciba uno del asistente. Para conseguir esa funcionalidad usaremos JavaScript. También se emplearán los lenguajes HTML y CSS para modificar los elementos presentes en el frontal.

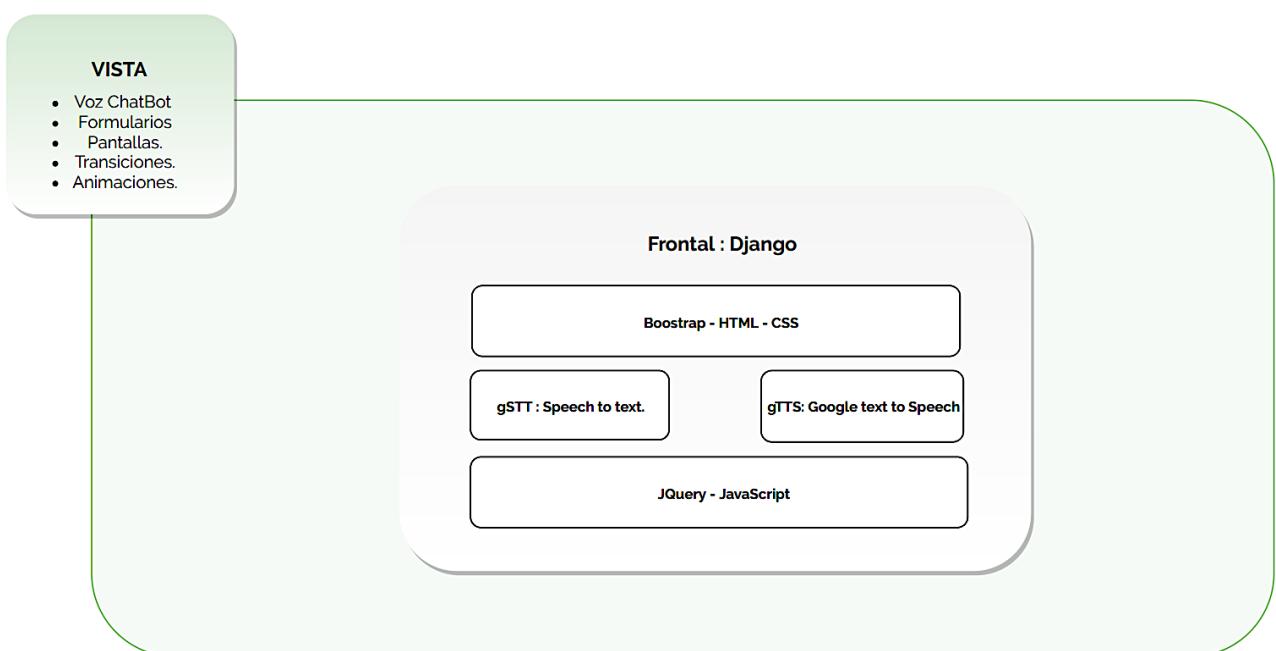


Ilustración 12. Elementos dentro del Componente Vista del Patrón MVC.

3.2.3 MVC – Descripción de la Aplicación Acceso para el acceso de Datos

En el modelo del proyecto se situarán aquellas funcionalidades que interactúen con los datos con los que trabajara el asistente. En este módulo, se manipulan y gestionan los accesos a los sonidos del cielo, que serán clasificados dentro del juego, los accesos a la base de datos donde residen los sonidos o cualquier otra información, también la actualización, búsqueda, ordenación, y mantenimiento de los sonidos, los registros de clasificación obtenidos después de una sesión de clasificación, etc. La definición del formato de los archivos que contendrán los datos de las clasificaciones también se contempla aquí.

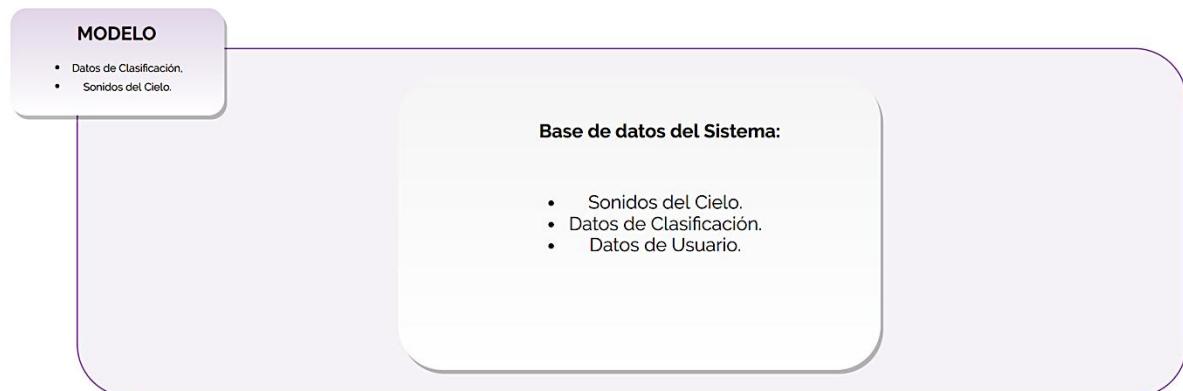


Ilustración 13. Elementos que forman parte del Modelo del Patrón MVC.

Para tener una perspectiva general del proyecto, se muestra en la siguiente imagen una vista en conjunto de los todos los componentes ya descritos que forman parte del sistema.

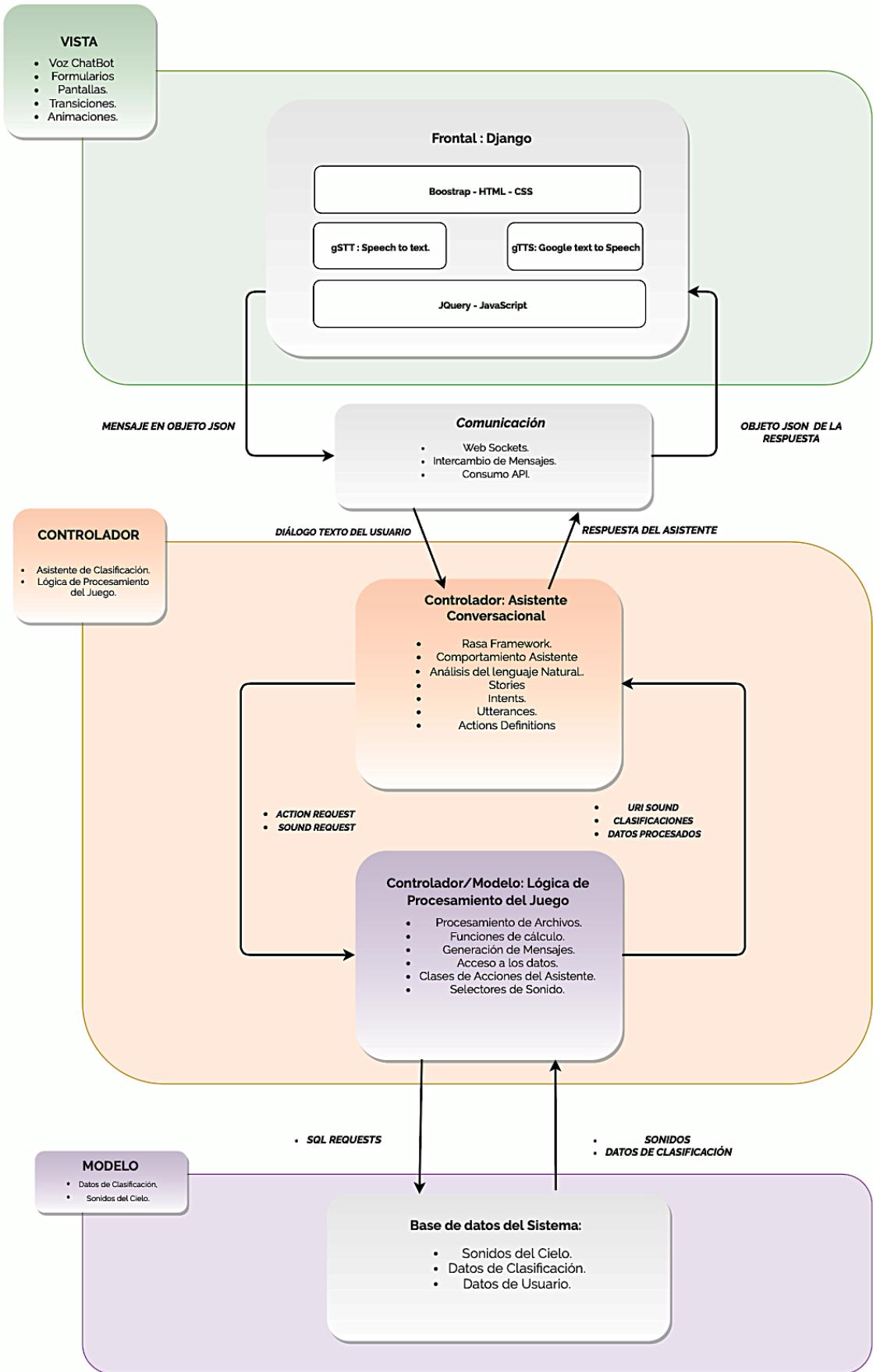


Ilustración 14. Modelo Vista Controlador del Proyecto.

3.2.4 Requerimientos y herramientas

Analizados todos los requisitos se van a pasar a mencionar todas las herramientas usadas en el desarrollo de este proyecto.

- **Lenguaje de programación Python:**
Para el desarrollo del asistente se ha empleado dicho lenguaje ya que la parte de las funcionalidades de procesamiento del juego se ha empleado Python, además así se mantiene el uso de este lenguaje en otros componentes ya que en el frontal se usará Python también.
- **Rasa:**
Este *framework* es una de las principales que usaremos para la implementación del asistente conversacional.
- **Django:**
Es el framework para desarrollo de aplicaciones Web, este se utilizará para marcar el patrón de diseño del frontal del proyecto.
- **PyCharm:**
Es uno de los mejores entornos de desarrollo avanzado para Python.
- **WebStorm:**
Entorno de desarrollo Web. Este se empleará para el desarrollo de la capa frontal de proyecto.
- **JavaScript:**
Ciertos elementos del frontal requieren acciones un poco más complejas para poder actualizar las vistas. Se emplea **JQuery** para poder hacer solicitudes cuando el usuario desea enviar un mensaje al asistente.
- **JSON:**
Se usará en el formato de los mensajes que se envíen entre el frontal, la parte del controlador y el modelo.
- **Bootstrap:**
Se empleará para la maquetación de la aplicación ya que ofrece gran adaptabilidad.
- **Postman:**
Para las pruebas de envío y recepción de mensajes REST.
- **Github:** Sistema de control de versiones donde residirá todo el código implementado.
- **Suite Adobe:**
Se usará para crear una capa visual amigable para el público infantil que será usada en el frontal.

3.3 Implementación de los Módulos del Controlador del Proyecto

3.3.1 Módulo Controlador - Definición del Asistente

Para el desarrollo de los requerimientos para la implementación del asistente se ha realizado investigación para conseguir herramientas, *frameworks* que pudieran definir el desarrollo del asistente. Entre las características que se buscan, se necesita que el *framework* tenga componentes bastante bien definidos, los cuales puedan ser usados para plasmar las necesidades del asistente conversacional del proyecto. También, otra característica es que fuese de código abierto y más importante, que este tuviera las posibilidades o facilidades para la integración con aplicaciones frontales y aplicaciones o API's para el consumo de datos. En principio se optó por ChatterBot, pero al presentar Rasa una buena amplitud de herramientas bastante potentes e interesantes se decidió usar este último.

3.3.2 Rasa Framework

Dentro de la arquitectura planteada, se observa uno de los módulos más importantes, en el cual se define el comportamiento, las características y funcionalidades del asistente. En el módulo controlador (MVC) se implementan y definen, todas las aplicaciones o funcionalidades que se encargarán de gestionar los mensajes que se reciben: peticiones en una conversación con el asistente, gestión de los datos capturados del usuario, reconocimiento del contexto de los mensajes para que el asistente decida cuál es la siguiente acción para realizar, procesamiento para atender las solicitudes del cliente, entrega de datos requeridos (en nuestro caso sonidos y clasificaciones), etc.

Para el desarrollo de este requisito del proyecto se ha necesitado contar con un conjunto de pautas de diseño que permitan construir el módulo conversacional, es decir la definición de este. Para la realización se ha decidido emplear Rasa debido a las interesantes características de su arquitectura. Es necesario entender cómo funciona este *framework* pues en este se fundamenta la interacción de nuestra aplicación con el usuario, es el corazón del asistente.

Rasa es un *framework* de código abierto para desarrollar asistentes que emplean inteligencia artificial. En su *framework* se cuenta con herramientas para la definición datos de entrenamiento de la inteligencia del asistente y también ciertas características ventajosas para su integración que pasaremos a describir a continuación:

3.3.2.1 Rasa NLU

El componente de comprensión natural de lenguaje, en inglés “*Natural Language Understanding*” (NLU), sirve para el reconocimiento de mensajes del usuario, con el objetivo de encontrar una correspondencia con algún *intent*. Los *intents* son todas las posibles frases que el usuario puede llegar a decir/escribir, agrupadas según un tipo de

finalidad. Por ejemplo, si alguien quiere saludar podría decir “*Hola!*”, “*¿Cómo te va?*” o “*¿Qué tal todo?*”, todas estas frases se pueden juntar en el concepto *Saludo*, y así cuando un usuario diga alguna de estas frases, entonces el asistente entenderá que el usuario le está saludando. La NLU de Rasa también tiene la responsabilidad de extraer información del *input* del usuario para hacerle corresponder con alguna entidad (información clave) con la que rellenara algunos *slots* con información que puede ser necesaria para algún procesamiento. Mas adelante se especifican con detalle todos los elementos de Rasa NLU en el contexto de nuestro proyecto. Rasa NLU se define como una tubería por donde pasan los mensajes, a la cual se pueden añadir más “piezas” de procesamiento, para hacer la tubería más compleja. Una de estas piezas es *Spacy*, un componente de lenguajes, que sirve como datos de entrenamiento para el aprendizaje, disponible en varios idiomas. [15]

3.3.2.2 Rasa Core

Es un motor de aprendizaje que usa *machine learning*, el cual define cual será la siguiente acción que tomar en función del mensaje del usuario. Los datos de entrada para este motor para el entrenamiento del asistente se definen en las *stories*, las cuales se detallan más adelante.

3.3.2.3 Rasa NLG

El componente de generación de lenguaje natural, en inglés “*Natural Language Generation (NLG)*”, es el componente de Rasa en el que se definen todos los mensajes y acciones que el asistente es capaz de usar para responder ante la solicitud del usuario, estas se definen en las *utterances*.

3.3.2.4 Procesamiento de un mensaje en Rasa

Se presenta de manera general el procesamiento de un mensaje que llega al sistema y que elementos intervienen para devolver una respuesta al usuario. [16]

Los pasos son:

- El mensaje se recibe y se pasa a un intérprete, que lo convierte en un diccionario que incluye el texto original, el *intent* y las *entities* que se pueden encontrar en el mensaje del usuario. Esta parte es manejada por NLU.
- El *tracker* es el objeto que realiza un seguimiento del estado de la conversación. Recibe la información de que ha entrado un nuevo mensaje. Este elemento es importante ya que cualquier aplicación *back-end* que se quisiera integrar, usara este objeto para poder acceder al estado del asistente, para obtener valores de *slots* que contienen información de las *entities*.

- La *policy* recibe el estado actual del rastreador y elige qué acción tomar a continuación.
- El *tracker* registra la acción elegida y se envía una respuesta al usuario.

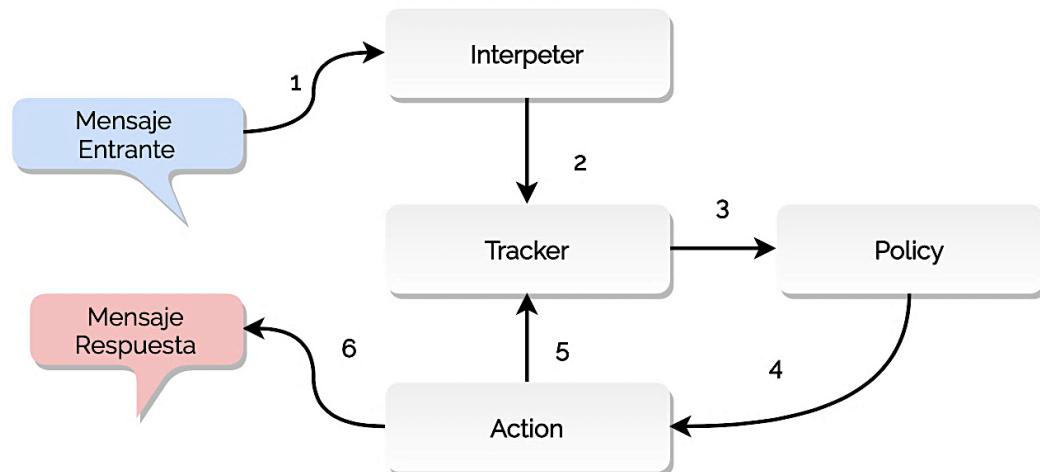


Ilustración 15. Diagrama del flujo de procesamiento de un mensaje.

La configuración de las *policies* permite definir, por ejemplo, cuanta cantidad de la conversación tiene que tomar en cuenta el asistente para poder tomar una decisión, entre otros parámetros disponibles en Rasa docs.

3.3.2.5 Componentes Elementales de Rasa

Se describen con más detalle los componentes básicos y elementales del *framework* Rasa para entender cómo funcionan y cuál es su utilidad, con la finalidad de justificar su uso para la implementación de los requisitos del asistente.

3.3.2.5.1 Intents

Son los mensajes posibles o las conversaciones más comunes, definidos en el universo (contexto de nuestro asistente) que el asistente puede esperar que ocurran con el usuario. Los *intents* forman parte del componente Rasa NLU, que, según la documentación Rasa, la define como una herramienta lenguaje *open source* para la clasificación de *intents*, para la recuperación de respuestas del usuario, la extracción de entidades que se puede necesitar para definir una conversación o pedir datos del usuario [17]. En la siguiente imagen se muestra la definición del *intent* “greet”, en la que se reúnen algunas de las posibles frases o palabras que el usuario podría decir cuando quieras expresar un saludo.

```
## intent:greet
- que tal?
- Hola Skyboy !
- Hey, estas ahi ?
- hola
- Buenas
- hey
```

Ilustración 16. Definición del intent “greet” dentro del archivo nlu.md.

3.3.2.5.2 Utterances

Las *utterances* son mensajes predefinidos que usa el asistente para poder responder al usuario. Estos mensajes son definidos en el *Domain* del asistente; tienen que incluir todas las posibles respuestas que pueden darse cualquier conversación. Las *utterances* forman parte del dominio, que, a su vez, forman parte de la componente NLU, que sirve como datos para el entrenamiento de la inteligencia del asistente virtual.

```
utter_happy:
- text: "Perfecto!"
- text: "Me alegro mucho que estes bien!"
- text: "Que guay ! empiezemos con energia!"
```

Ilustración 17. Definición de la utterance “utter_happy” y todas sus posibles respuestas, dentro del archivo domain.yml.

3.3.2.5.3 Actions

Las *actions* son los nombres de las acciones que serán usadas por el asistente para poder responder al usuario. En detalle, a diferencia de las *utterances*, un *action* es código que se puede ejecutar: para realizar alguna llamada a una base de datos, procesar cálculos, etc. El código asociado a una *action* no se encuentra definido en el asistente, sino que son los nombres de los métodos con los cuales el asistente llamará algún componente *back-end* o aplicación externa para poder ejecutar el código. Todas las *actions* tienen que estar definidos en el *Domain* del asistente, ya que forman parte del universo que tiene que conocer y las cosas que puede realizar en cuanto a integración con otras aplicaciones. A diferencia que las *utterances* las *actions* son mensajes que se envían al usuario, que contienen información que ha sido procesada por otra o aplicación. Para el desarrollo de las *actions* de este asistente se ha usado el lenguaje de desarrollo Python y se usado la utilidad de integración que tiene Rasa dentro de su *framework*, más adelante se detalla la

definición de estos. Se muestra a continuación el empleo de una *action* en concreto, en el dominio y como parte de una *story*.

```
actions:  
  - action_hola_mundo  
  - action_classifying
```

Ilustración 18. Definición del action “action_classifying” dentro del dominio.

```
## 2A - clasificacion  
* bot_clasificar  
  - utter_clasificar  
  - action_classifying
```

Ilustración 19. Uso del action “action_classifying” dentro de la Storie “clasificación”.

3.3.2.5.4 Stories

En Rasa, las *stories* definen el comportamiento del asistente ante ciertos comandos enviados a los usuarios. Con ellos podemos definir los diferentes caminos que puede tomar un diálogo. En una *story* se definen los *intents* y *utterances* que pueden formar parte de una conversación, con estos elementos podemos recrear las diferentes escenarios para los cuales el asistente tiene que estar preparado, siendo así capaz de poder reconocer los comandos del usuario (que se correspondan con las *intents* definidas en la capa NLU) y respondiendo a ellas mediante las acciones las cuales pueden ser simples respuestas (frases) o el desencadenamiento de llamadas al servidor-módulo de acciones en el que están definidos las funciones de clasificación. Dentro de la arquitectura de Rasa, las *stories* forman parte del CORE, que es el componente de Rasa que reúne todas las características para manejar los contextos y administrar el aprendizaje automático mediante *machine learning*. Las *stories*, son la información que usara el CORE para aprender. Gracias a este enfoque de Rasa, el asistente podrá entender lo que dice el usuario y además aprender de esa entrada, lo que hará, con el paso del tiempo, que el asistente pueda manejarse en situaciones que nosotros no hallamos definido manualmente. Es decir, mediante experiencias, el asistente podrá definir reglas nuevas, así entonces entendemos que es una gran ventaja ya que no tendremos que pensar que solo los creadores del asistente podrán crear todas las reglas posibles, sino que un asistente pensado en Rasa también lo hará de manera automática. A continuación, se describen algunas de las *stories* definidas para el funcionamiento para el manejo de las conversaciones entre el asistente y el usuario.

3.3.2.5.5 Slots

Los *slots* son un componente importante de Rasa, en estos se almacena la información clave, que se ha extraído del dialogo del usuario, por ejemplo, puede ser un nombre o una respuesta a alguna pregunta que el asistente le hace al usuario. En otras palabras, sirven de memoria de almacenamiento de datos a los cuales puede acceder una tercera aplicación para saber el estado del asistente. Estos están ligados a las *entities*, las cuales están ligadas a piezas de información que puede proveer el usuario, estas son almacenadas en los *slots*.

```
entities:  
  - name  
  - respuesta  
  
slots:  
  name:  
    type: text  
    initial_value: "amig@"  
  
  sonido_actual:  
    type: text  
    initial_value: "0"  
  
  respuesta:  
    type: text  
    initial_value: "0"
```

Ilustración 20. Definición de algunas entidades y slots dentro de dominio.

3.3.2.5.6 Dominio

El dominio es en donde se encuentran todos los componentes que definen el comportamiento del asistente, en este se encuentran las *entities*, *utterance*, *intents*, *slots* y *actions*. Cada vez que se realice algún cambio en el comportamiento del asistente, hay que asegurarse de que este se vea reflejado en este componente. Su implementación se encuentra en el archivo “*domain.yml*”.

3.3.2.5.7 Implementación de Rasa para el Asistente – Storie

Se muestra una de las definiciones de una *storie* de Rasa, que se han definido para la implementación del comportamiento del asistente cuando se inicia el diálogo con el usuario, el cual implica la captura del nombre del usuario y la presentación de las opciones posibles que puede realizar el usuario en el sistema.

```

## 1A camino para empezar el dialogo del con el asistente
* greet
  - utter_greet
* mood_great
  - utter_happy
  - utter_question_a

## 1B - Preguntar el nombre
* mi_nombre
  - utter_if_you_want
  - utter_question_b

```

Ilustración 21. Definición de la storie para empezar el dialogo con el asistente, en el archivo stories.md

```

## intent:greet
)- que tal?
- Hola Skyboy !
- Hey, estas ahi ?
- hola
- Buenas
- hey
- hello
- hi
- good morning
- good evening
)- hey there

## intent:mi_nombre
)- hola mundo
- mi nombre es [jhoosef]{"entity":"name","value":"jhoosef"}
- yo me llamo [jhoosef]{"entity":"name","value":"jhoosef"}
- me llamo [jhoosef]{"entity":"name","value":"jhoosef"}
- me llamo [Juan] {"entity":"name","value":"Juan"}
- me llamo [Raquel] {"entity":"name","value":"Raquel"}
- me llamo [Paco] {"entity":"name","value":"paco"}
- me llamo [Isabel] {"entity":"name","value":"isabel"}

```

Ilustración 22. Intents posibles, asociados a la storie anterior para empezar el dialogo con el asistente, en el archivo nlu.md.

3.3.3 Módulo Modelo/Controlador – Servidor de Acciones del Juego

Como se ha mostrado ya en el planteamiento de la arquitectura, el asistente tiene que ser capaz de poder ofrecer ciertas funcionalidades que requieren de procesamiento, cálculos y el acceso a datos con los que trabajará el clasificador. En nuestro caso, tenemos que ofrecer todas las posibles acciones con las que cuenta la aplicación para la clasificación de los sonidos. Hay que recordar que esta parte es excluyente de la implementación del comportamiento del asistente, ya que es independiente, con servicios que están en otro servidor el cual funcionará para que el asistente pueda hacer uso de estos cuando en la conversación con el usuario se identifique alguna solicitud de acción (p. ej. “*dame un sonido*”), desencadenando así llamadas estos servicios cuando sea necesario.

En este apartado se explican estos servicios. Cada uno de estos, o la conjunción de algunos, cumplen con las funcionalidades requeridas para definir los caminos que puede tomar la clasificación o el entrenamiento y poder brindar procesamiento, etc. Aunque crear un servidor de acciones en el lenguaje que se considere oportuno, Rasa ofrece un SDK de Python para el desarrollo de un servidor de acciones, el cual se usara en este proyecto. La definición de estas acciones se encuentra dentro del archivo “*actions.py*”, en donde están todas las clases Python las cuales se corresponden con las acciones disponibles para el asistente [18].

3.3.3.1 Implementaciones de funcionalidades

Entre las siguientes cinco primeras clases, se encargan de seleccionar un sonido en concreto para el entrenamiento y para el aprendizaje de los tipos de sonidos. Como sabemos, podemos encontrar 5 tipos de sonidos. Estas envían un sonido de los disponibles y sus datos de clasificación al asistente para que este se la devuelva al frontal en formato de URI, empaquetado dentro de un mensaje de tipo JSON, el cual va identificado con la clave “soundUri”.

```
dispatcher.utter_message(json_message= {"soundUri":'sounds/sonidos_entrenamiento'})
```

Ilustración 23. Generación de un mensaje JSON que contiene la URI del sonido.

También una funcionalidad importante es la modificación del estado del asistente, ya que estas clases son capaces de modificar *slots* de información. Retornará mediante el objeto *tracker* el valor de uno de los Slots llamado ‘sonido_actual’, de esa manera el asistente sabrá sobre de que tipo de sonido se está conversando con el usuario. También se describen las clases para la comprobación de respuestas del usuario, listado de sonidos en el sistema disponibles para su clasificación, etc.:

- **ActionUnderdenseSelector** : En esta acción se seleccionará los sonidos de *tipo 1*, los cuales corresponden con los meteoritos que se clasifican como underdense.
- **ActionMSelector**: En esta acción se seleccionará los sonidos de *tipo 2*, los cuales corresponden con los meteoritos que se clasifican como tipo M.
- **ActionLongOverdenseSelector**: En esta acción se seleccionará los sonidos del tipo 3 los cuales corresponden con los meteoritos que se clasifican como *largos overdenses*.
- **ActionMediumOverdenseSelector**: En esta acción se seleccionará los sonidos del tipo 4 los cuales corresponden con los meteoritos que se clasifican como *overdense medio*.
- **ActionShortOverdenseSelector**: En esta acción se seleccionará los sonidos del tipo 5 los cuales corresponden con los meteoritos que se clasifican como *overdense corto*.
- **ActionSoundCheck**: Cuando el usuario responde ante la pregunta sobre la clasificación de sonidos en el entrenamiento, esta acción se encarga de verificar si la respuesta es correcta o no, en ambos casos envía un mensaje al usuario, con una breve descripción del resultado de la clasificación.
- **SoundListing**: Esta clase sirve para poder listar los sonidos de un directorio. Devuelve una lista con todos los sonidos (y sus localizaciones) y crea un archivo de texto en el *path* especificado, con la lista de todos los sonidos existentes y su clasificación media. Además, a cada archivo de audio se le asocia un archivo de texto. En la primera línea de dicho archivo se encuentra el número de veces que ha sido clasificado ese sonido y en las siguientes líneas las clasificaciones con el autor de la clasificación en cada línea.

1	3	
2	1 Raquel	
3	2 Jhosef	
4	5 Jhosef	
5		

Ilustración 24. Archivo de clasificación asociado al archivo de audio “sound8.wav”.

Si este archivo de clasificación no existe, entonces lo crea e inicializa a cero el número de veces que se ha clasificado ese sonido, poniendo dicha información en la primera línea en dicho archivo de clasificación. En caso de que el archivo ya

existiera, lo único que se hace es realizar la lectura de la primera línea de dicho archivo para obtener el número de veces que ha sido clasificado el sonido asociado. Además, crea un objeto de tipo lista, en el que se devuelve las

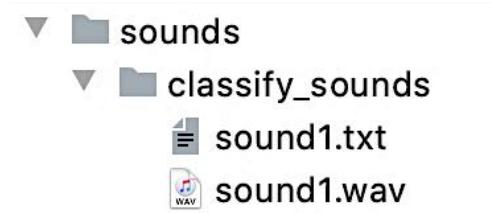


Ilustración 25. Archivo de clasificación asociado al archivo de audio “sound1.wav”.

localizaciones de los sonidos, el número de clasificaciones de cada sonido extraído de cada archivo de clasificación, la ubicación de cada archivo de clasificación y la ubicación del archivo de texto donde está la lista de sonidos disponibles para clasificar. El tipo de archivos de texto que se crean para la clasificación se ubicará siempre en el mismo directorio donde se encuentra el sonido.

- **ActionSaveClasification:** Esta acción se ejecuta para guardar la respuesta de clasificación, después de que el usuario haya dado una respuesta, esta se almacena en el “*slot_respuesta*”. Para ello accederá mediante el *tracker* al estado de asistente y extraerá la respuesta de dicho *slot*. Dicha información se almacenará, en el archivo de clasificaciones asociado a ese sonido, se aumentará en uno el número de clasificaciones del sonido (la primera línea) y junto con la clasificación se pondrá el nombre del usuario que clasificó dicho sonido.
- **ActionHelloWorld:** Esta clase se ejecuta cuando la acción “*action_hola_mundo*” es reconocida por el asistente. Este se encarga de almacenar el nombre del usuario y de dar la bienvenida al usuario al sistema.
- **ActionClassifying:** Esta clase se ejecuta cuando el asistente reconoce en la conversación con el usuario la *action* “*action_classifying*”. La funcionalidad de esta es escoger un sonido para ser clasificado y enviárselo al asistente para que este pueda enviárselo a su vez al frontal de la aplicación, para que el usuario lo pueda escuchar. Para enviarle el sonido al usuario, este se encapsula en un objeto JSON para que el frontal pueda presentarlo en un reproductor. Para hacer la selección se sigue la política en la que siempre se devuelve el sonido que menos veces se ha clasificado, de esta manera nos aseguramos de que ningún sonido se quede sin clasificar y así tener en cuenta los nuevos sonidos que lleguen. En caso de que todos los sonidos tengan el mismo número de clasificaciones, entonces se escoge cualquiera de manera aleatoria. Para el funcionamiento de esta clase, se hace un llamado a la clase *soundListing* la cual devolverá todos los sonidos y sus datos de clasificación como se ha especificado antes, para ser usados en esta clase.

3.3.4 Integración entre el Módulo de Acciones y el Asistente

Es necesario entender como interactúan el módulo del servidor de acciones y el módulo conversacional que define al asistente, ya que la experiencia que tendrá el usuario al realizar un comando depende de llamadas del asistente hacia las funcionalidades implementadas en las acciones. En este apartado se describe la relación entre las funcionalidades implementadas en el módulo de acciones y el módulo del asistente.

3.3.4.1 Función Aprendizaje y Función de Entrenamiento

Las funcionalidades de aprendizaje y entrenamiento están dentro del controlador del MVC (módulo asistente y módulo de acciones en conjunto), son las que se encargan de poner a disposición del asistente, los tipos de sonidos con sus descripciones, que le servirán al usuario para poder aprender que tipos de sonidos existen o entrenar.

Recordemos implementaciones de las acciones asociadas se encuentran dentro del archivo “*actions.py*”, donde se define el servidor de acciones. En estas funcionalidades se hacen la selección de sonidos, entre otros procesamientos. Para estas funcionalidades se usan las siguientes acciones y sus correspondientes elementos definidos en el módulo del asistente.

Tabla 2. Asociación entre actions y clases asociadas para la funcionalidad de Aprendizaje

Action definida en el asistente	Acciones en el Modelo
action_underdense_sample	ActionUnderdenseSelector
action_m_sample	ActionMSelector
action_long_over_sample	ActionLongOverdenseSelector
action_med_over_sample	ActionMediumOverdenseSelector
action_short_over_sample	ActionShortOverdenseSelector

Para la función de entrenamiento, se usan todas las anteriores, además de la clase *ActionSoundCheck* asociado al action “action_check_sound”.

3.3.4.2 Función Clasificación

En la funcionalidad de clasificación, lo que se desea es poner a disposición del usuario sonidos para que este los pueda clasificar. Cuando un usuario realiza una solicitud de clasificación, el asistente intentará contactar con el servidor de acciones para ejecutar las clases destinadas para este fin. Cuando el asistente haya ejecutado con éxito la llamada a dichas funcionalidades, entonces estas le responderán al usuario mediante el objeto *tracker* del *framework* de Rasa, enviándole el sonido al frontal en formato de recurso URI

empaquetado en un mensaje JSON. Cuando el usuario haya respondido con su clasificación para un sonido dado, este se guardará en un *slot* en el módulo del asistente llamado “*respuesta*”, entonces el asistente podrá ejecutar otra acción que se encargará de extraer, mediante el *tracker* la clasificación del usuario del *slot* “*respuesta*”. Dicho *slot* está definido en el archivo “*domain.yml*”, en la definición del asistente. Cuando la acción haya extraído la información de dicho *slot* entonces este guardará esta información en el archivo de clasificación asociado al sonido actual que se está clasificando, insertando dicha información en una línea, al final. También se guardará en la misma línea junto con la clasificación, el nombre de la persona que realizó dicha clasificación. Las clases de las acciones y los *actions* usados para esta funcionalidad son los siguientes:

Tabla 3. Asociación entre actions y clases asociadas para la funcionalidad de clasificación.

Action definida en el asistente	Acciones en el Servidor de Acciones
<code>action_classifying</code>	<code>ActionClassifying</code>
<code>action_save_clasificacion</code>	<code>ActionSaveClasificacion</code>
<code>action_classifying</code>	<code>SoundListing</code>

3.3.4.3 Pruebas de Integración

En este apartado se muestran las pruebas de integración entre el módulo del asistente y el servidor de acciones para conformar entre los dos el módulo Controlador, perteneciente al Modelo Vista Controlador.

Antes que nada, tendremos que situarnos dentro de la carpeta “*mvc_control_bot_juego*” y ejecutar el siguiente comando: “*rasa train*”, el cual servirá para poder entrenar el modelo de nuestro asistente con el dominio y todos los datos definidos anteriormente (*intents*, *actions*, *utterances*, etc.). La terminal tendrá el siguiente aspecto:

```
jhos@MacBook-Pro ~/Desktop/GIT-LOCAL/UPM-ChatBot-SoundsOfMeteors/mvc_control_bot_juego (master) $ rasa train
2020-06-26 20:29:49 INFO    rasa.model - Data (stories) for Core model section changed.
2020-06-26 20:29:49 INFO    rasa.model - Data (messages) for NLU model section changed.
Training Core model...
Processed Story Blocks: 100%|██████████| 0/0, # trackers=1
Processed Story Blocks: 100%|██████████| 0/0, # trackers=1
```

Ilustración 26. Progreso de entrenamiento del asistente con los datos de entrenamiento

Después de haber entrenado nuestro modelo, nos tiene que aparecer el siguiente mensaje de confirmación, mostrándonos la ubicación del archivo donde se encuentra el modelo de nuestro asistente que ya se puede usar:

```
NLU model training completed.  
Your Rasa model is trained and  
located at '/home/fernando/Proyectos/ChatBot/ChatBot/venv/lib/python3.7/site-packages/rasa/core/assets/20200626-203207.tar.gz'.
```

Ilustración 27. Mensaje de confirmación de la generación del modelo.

Una vez entrenado ahora solo queda ejecutar el *shell* de Rasa para poder interactuar con el asistente. Para eso ejecutamos el comando “*rasa shell*” y si no ha habido problemas la terminal presentara el siguiente aspecto:

```
) $ rasa shell  
2020-06-26 23:18:33 INFO      root - Connecting to channel 'cmdline' which was specified by the  
'--connector' argument. Any other channels will be ignored. To connect to all given channels,  
omit the '--connector' argument.  
2020-06-26 23:18:33 INFO      root - Starting Rasa server on http://localhost:5005  
Bot loaded. Type a message and press enter (use '/stop' to exit):  
Your input -> [REDACTED]
```

Ilustración 28. Rasa Shell donde se puede interactuar con el asistente.

Cuando el asistente este esperando a los comandos del usuario, el siguiente paso es activar el servidor de acciones que Rasa provee, en el que se han desarrollado todas las clases necesarias para implementar las acciones de procesamiento. Para ello se escribirá en otra terminal: “*rasa run actions*”, y cuando el servidor este activo se mostrará el siguiente aspecto en la terminal, listándose todos los servicios disponibles el módulo de acciones:

```
./Desktop/GIT-LOCAL/UPM-ChatBot-SoundsOfMeteors/mvc_control_bot_juego (master) $ rasa run actions  
INFO      rasa_sdk.endpoint - Starting action endpoint server...  
  
Join our community. https://www.pygame.org/contribute.html  
INFO      rasa_sdk.executor - Registered function for 'action_hola_mundo'.  
INFO      rasa_sdk.executor - Registered function for 'action_underdense_sample'.  
INFO      rasa_sdk.executor - Registered function for 'action_m_sample'.  
INFO      rasa_sdk.executor - Registered function for 'action_long_over_sample'.  
INFO      rasa_sdk.executor - Registered function for 'action_med_over_sample'.  
INFO      rasa_sdk.executor - Registered function for 'action_short_over_sample'.  
INFO      rasa_sdk.executor - Registered function for 'action_classifying'.  
INFO      rasa_sdk.executor - Registered function for 'action_check_sound'.  
INFO      rasa_sdk.executor - Registered function for 'action_learning'.  
INFO      rasa_sdk.executor - Registered function for 'action_training'.
```

Ilustración 29. Rasa Actions corriendo en el servidor.

3.3.4.3.1 Pruebas

- Saludo y reconocimiento: del usuario por parte del módulo de acciones: El usuario saluda al asistente, esta captura su nombre y lo usa para poder seguir el diálogo

```
Your input -> Hola
Buenas, como va todo ?
Your input -> Bien
Perfecto!
¿Cuál es tu nombre?
Your input -> Jhosef
Podemos aprender, entenar o clasificar sonidos del cielo.
¿Qué deseas hacer Jhosef ?
Your input ->
```

Ilustración 30. Saludo y captura del nombre del usuario por medio del shell de Rasa.

- Función de Aprendizaje: Para esta prueba se ha arrancado el servidor y se ha conectado Postman para realizar una operación POST con el comando de aprender. Como se puede ver, el mensaje está estructurado en formato JSON.

The screenshot shows a POST request in Postman. The URL is `http://localhost:5005/webhooks/rest/webhook`. The request body is a JSON object:

```
1 {
2   "sender": "Rasa",
3   "message": "aprender"
4 }
```

The response body shows a list of messages in JSON format:1 [
2 {
3 "recipient_id": "Rasa",
4 "text": "Vamos a aprender los sonidos!"
5 },
6 {
7 "recipient_id": "Rasa",
8 "text": "Te voy a ir presentando varios sonidos"
9 },
10 {
11 "recipient_id": "Rasa",
12 "text": "El sonido que escucharas a continuacion"
13 },
14 {
15 "recipient_id": "Rasa",
16 "text": "es un meteoro underdense o tipo 1"
17 },
18 {
19 "recipient_id": "Rasa",
20 "custom": {
21 "soundUri": "sounds/sonidos_entrenamiento/meteor1_underdense.wav"
22 }
23 },
24]

Ilustración 31. Formato de datos de los mensajes en la comunicación para el aprendizaje.

La conectividad queda asegurada con la prueba anterior, sin embargo, podemos realizar la misma prueba en el *Rasa shell*. Esto es útil en caso de que no se pueda instanciar la aplicación externa que quiera consumir los servicios de nuestro asistente y se quiera probar una funcionalidad.

```
Podemos aprender, entenar o clasificar sonidos del cielo.  
¿Qué te gustaria hacer Raquel?  
Your input -> Aprender  
Vamos a aprender los sonidos!  
Te voy a ir presentando varios sonidos  
El sonido que escucharas a continuacion  
es un meteoro underdense o tipo 1  
Custom json:  
{  
    "soundUri": "sounds/sonidos_entrenamiento/meteor1_underdense.wav"  
}  
¿ Desea seguir ?  
Your input -> █
```

Ilustración 32. Respuesta del asistente cuando se identifica la funcionalidad "Aprender".

- Prueba de Entrenamiento 1: Se le pide entrenar, el asistente hace una llamada al modelo de datos para pedirle un sonido. El asistente espera una respuesta del usuario. El usuario contesta mal.

```
Your input -> Hola, quiero entrenar  
Perfecto! Vamos a entrenar  
Te voy a ir presentando varios sonidos  
y me tendras que decir de que tipos son  
Escucha el siguiente audio:  
Custom json:  
{  
    "soundUri": "sounds/sonidos_entrenamiento/meteor1_underdense.wav"  
}  
amig@, de que tipo es el sonido?  
Your input -> tipo 2  
oh! el sonido no es un m, la respuesta es: underdense
```

Ilustración 33. Reconocimiento de la función "Entrenamiento".

En este caso, el sistema le ofrece la respuesta para poder corregir al usuario, con información del tipo de sonido.

- Prueba de Entrenamiento 2: El usuario acierta en la clasificación del tipo de sonido.

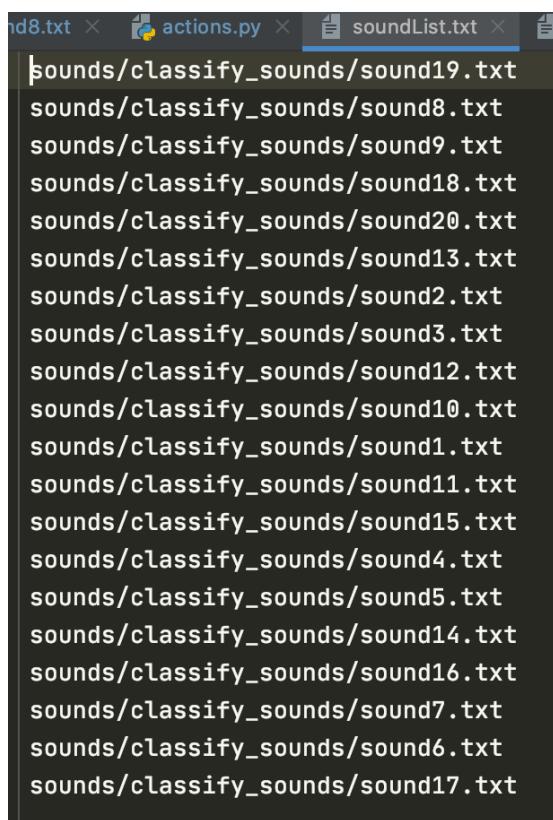
```

Muy bien! Has acertado el sonido es un meteoro m
Escucha el siguiente audio:
Custom json:
{
    "soundUri": "sounds/sonidos_entrenamiento/meteor3_long_overdense.wav"
}
Juan, dime de que tipo crees que es el sonido?
Your input -> tipo 3
Muy bien! Has acertado el sonido es un meteoro tipo overdense largo

```

Ilustración 34. Acierto en la prueba de entrenamiento.

- Prueba de Clasificación: Cuando el usuario recibe un sonido, después de clasificarlo el sistema le indica que se ha guardado su respuesta. Además, recibe el promedio del tipo de clasificaciones que ha recibido ese sonido. Se describe todo el proceso:
 - El sistema lista todos los sonidos disponibles en local para ofrecerlos en el nivel de clasificación.



The screenshot shows a terminal window with three tabs at the top: 'sound8.txt', 'actions.py', and 'soundList.txt'. The 'soundList.txt' tab is active, displaying a list of 20 text files located in the 'sounds/classify_sounds' directory. The files are named sequentially from 'sound19.txt' down to 'sound1.txt'. The terminal window has a dark background with light-colored text.

```

sound8.txt × actions.py × soundList.txt ×
[sounds/classify_sounds/sound19.txt
sounds/classify_sounds/sound8.txt
sounds/classify_sounds/sound9.txt
sounds/classify_sounds/sound18.txt
sounds/classify_sounds/sound20.txt
sounds/classify_sounds/sound13.txt
sounds/classify_sounds/sound2.txt
sounds/classify_sounds/sound3.txt
sounds/classify_sounds/sound12.txt
sounds/classify_sounds/sound10.txt
sounds/classify_sounds/sound1.txt
sounds/classify_sounds/sound11.txt
sounds/classify_sounds/sound15.txt
sounds/classify_sounds/sound4.txt
sounds/classify_sounds/sound5.txt
sounds/classify_sounds/sound14.txt
sounds/classify_sounds/sound16.txt
sounds/classify_sounds/sound7.txt
sounds/classify_sounds/sound6.txt
sounds/classify_sounds/sound17.txt]

```

Ilustración 35. Listado de generación de archivos de clasificación de los sonidos disponibles.

- El sistema asocia un archivo de clasificación a cada sonido existente: Si este archivo no existe, entonces lo crea, inicializando a cero el número de veces que ha sido clasificado. Cuando el archivo existe, se carga de ese archivo, la información con respecto al número de veces que ha sido clasificado.



Ilustración 36. Archivos de clasificación de sonidos en el directorio de los sonidos.

- Después de haber listado todos los sonidos con sus respectivos archivos de clasificación, se escoge un sonido a clasificar según las siguientes políticas:
 - Tiene prioridad el sonido que tenga cero clasificaciones.
 - Si todos tienen al menos una clasificación, entonces se escoge cualquiera aleatoriamente.
- El sistema escoge sonido para el usuario. Se observa el recurso que devuelve el sistema donde se encuentra el sonido (URI). El sistema espera por una respuesta del usuario

```

Your input -> Clasificar
Muy bien! vamos a clasificar
Estoy escogiendo un sonido para ti...
Dime, ¿De que tipo crees que es el siguiente sonido ?
Custom json:
{
    "soundUri": "sounds/classify_sounds/sound8.wav"
}
Pedro de que clase crees que es 1, 2,3,4 o 5?
Your input -> 

```

Ilustración 37. Datos en un mensaje de clasificación.

- Este es el estado de la información del archivo de clasificaciones que tiene asociado el sonido ‘sound8’:

1	7
2	1 Raquel
3	2 Jhosef
4	5 Jhosef
5	4 Jhosef
6	2 Juan
7	3 Juan
8	4 Juan
9	

Ilustración 38. Datos de clasificación del sonido 8.

- El usuario introduce una respuesta y esta se almacena, además recibe el promedio de clasificación para ese sonido en concreto.

```

Pedro de que clase crees que es 1, 2,3,4 o 5?
Your input -> tipo 3
Vale ! has dicho que es tipo 3, estoy guardando tu clasificación para el sonido sound8.
Pedro las clasificaciones anteriores dicen que este sonido podría ser de tipo 3.
Si quieres clasificar el siguiente sonido di 'dame otro sonido'

```

Ilustración 39. Mensaje después de una clasificación.

- El archivo de clasificaciones asociado al sonido ‘sound8’ se actualiza con la clasificación de Pedro.

sound8.txt		actions.
1	8	
2	1 Raquel	
3	2 Jhosef	
4	5 Jhosef	
5	4 Jhosef	
6	2 Juan	
7	3 Juan	
8	4 Juan	
9	3 Pedro	
10		

Ilustración 40. Actualización de los datos de clasificación del sonido 8.

- Prueba de Clasificación 2: Cuando el usuario introduce un rango de valores que no existe para la clasificación, el sistema no guarda la respuesta. El sonido queda pendiente para otra sesión de clasificaciones

```

Your input -> dame otro sonido
Muy bien! vamos a clasificar
Estoy escogiendo un sonido para ti...
Dime, ¿De que tipo crees que es el siguiente sonido ?
Custom json:
{
    "soundUri": "sounds/classify_sounds/sound12.wav"
}
Pedro de que clase crees que es 1, 2,3,4 o 5?
Your input ->

```

Ilustración 41. Elección de otro sonido para su clasificación.

```
Pedro de que clase crees que es 1, 2,3,4 o 5?
Your input -> clase 7

7 no es una clasificacion valida, tu respuesta no ha sido guardada, el sonido quedará pendiente de clasificación :).
Si quieres clasificar el siguiente sonido di 'dame otro sonido'
```

Ilustración 42. Respuesta del asistente para una clasificación inválida.

- Prueba Información de Ayuda en Clasificación: El usuario pide ayuda cuando no sabe clasificar un sonido.

```
Muy bien! vamos a clasificar
Estoy escogiendo un sonido para ti...
Dime, ¿De que tipo crees que es el siguiente sonido ?
Custom json:
{
  "soundUri": "sounds/classify_sounds/sound9.wav"
}
Raquel de que clase crees que es 1, 2,3,4 o 5?
Your input -> ayuda para clasificar
Raquel, para clasificar tienes que escuchar el sonido y despues decir de que clase de meteoro se trata
Si crees que es un meteoro tipo underdense solo di es de 'clase 1', si es de tipo M entonces di 'es de clase 2', para overdense largo 'clase 3', para overdense medio 'clase 4' y para overdense corto di 'clase 5'.
Aun te cuesta reconocer los sonidos?
Your input ->
```

Ilustración 43. La funcionalidad de ayuda muestra información para guiar al usuario.

- Prueba Información de Ayuda en Clasificación: El usuario pide ayuda cuando no sabe clasificar un sonido. Opta por recibir ayuda reproduciendo un sonido parecido al que tiene que clasificar

```
Aun te cuesta reconocer los sonidos?
Your input -> Si
Si quieres puedo reproducir el sonido de cualquier meteoro. Para un meteoro tipo M di ': Como suena un meteoro tipo M?'. De la misma forma para cualquier tipo. O si deseas podemos entrenar, para ello solo di 'quiero entrenar'
Your input -> como suena un meteoro tipo m
El sonido que escucharas a continuacion
Custom json:
{
  "soundUri": "sounds/sonidos_entrenamiento/meteor2_M.wav"
}
Es un meteoro tipo M o tipo 2
Ahora podemos aprender, entenar o clasificar sonidos del cielo.
¿Qué te apeteceria hacer Raquel?
```

Ilustración 44. Función de reproducción de un sonido en concreto.

- Prueba Información de Ayuda en Clasificación: El usuario pide ayuda cuando no sabe clasificar un sonido. Opta por el entrenamiento para reforzar su conocimiento.

```
Your input -> si
Si puedes reproducir el sonido de cualquier meteorito. Para un meteorito tipo M di ': Como suena un meteoro tipo M?'. De la misma forma para cualquier tipo. O si deseas podemos entrenar, para ello solo di 'quiero entrenar'
Your input -> entrenar
Perfecto! Vamos a entrenar
Te voy a ir presentando varios sonidos
y me tendras que decir de que tipos son
Escucha el siguiente audio:
Custom json:
{
  "soundUri": "sounds/sonidos_entrenamiento/meteor1_underdense.wav"
}
Raquel de que clase crees que es 1, 2,3,4 o 5?
Your input ->
```

Ilustración 45. Entrenamiento como ayuda para la clasificación.

- Prueba Información de Ayuda General del Sistema: El usuario puede pedir ayuda en cada momento para poder conocer las opciones.

```
Your input -> que opciones tengo
Raquel podemos hacer las siguientes cosas:
Si deseas aprender, solo di 'quiero aprender', de la misma forma para entrenar o clasificar.
¿Qué te apetecería hacer Raquel?
Your input -> ■
```

Ilustración 46. Ayuda general del sistema.

3.3.5 Módulo Vista - Aplicación Frontal Web del Proyecto

3.3.5.1 Frontal Web del Proyecto – Django

Django es una de las principales tecnologías que se utilizan en el desarrollo de lo que puede ser considerado la parte de procesamiento del frontal del proyecto. Es uno de los frameworks web *open-source* basado en Python el cual contiene módulos que nos aportaran agilidad en la implementación de aspectos elementales de proyectos web dinámicos, como pueden ser: generación de las vistas, creación de plantillas, funcionalidades más avanzadas como autenticación de usuarios, etc. También emplea el patrón de diseño MVC.

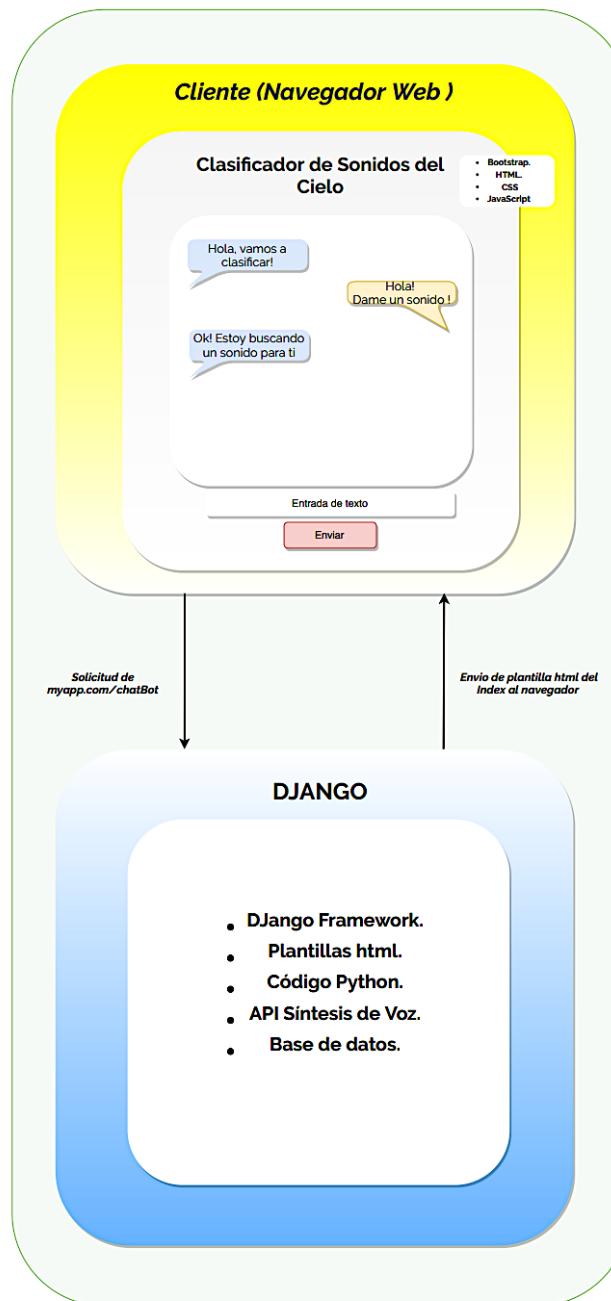


Ilustración 47. Esquema general del funcionamiento de un proyecto en Django.

En el diagrama anterior se muestra la interacción entre el navegador y el servidor Djago. La vista que el usuario puede observar es actualizada por medio de plantillas HTML que son enviadas desde el servidor de Django. En el caso de este proyecto tenemos una vista principal que es donde será ubicado el *chat* del asistente.

En el proyecto Django tiene las siguientes tareas:

- Generar la vista para el frontal, implementada en archivo “*page-chat.html*”.
- Realizar las operaciones POST para consumir el *endpoint* de Rasa en el que tenemos definido el asistente. Se ha creado un cliente usando JavaScript, y sus configuraciones se encuentran en el archivo “*chat.js*”.
- Se encarga de presentar los sonidos que escoge el asistente. El asistente tiene una lista de sonidos disponibles para clasificar al igual que Django. Cuando el asistente escoge un sonido, este le manda la ubicación del sonido, un path, así Django accede a su directorio de sonidos con ese *path*, y lo presenta en el reproductor para que el usuro lo pueda escuchar.
- Para una implementación a futuro, los servicios de síntesis de voz y procesamiento de voz del usuario tiene que estar definidas en el módulo del proyecto.

La decisión de usar Django viene fundamentada en por las siguientes motivaciones:

- Uno de los requerimientos más importantes es que el frontal sea desarrollado sobre tecnología web. Esto es conveniente por la variedad de sistemas que existen, evitando así el problema de la incompatibilidad que se occasionaría si la aplicación se desplegara con una tecnología en concreto (Android, iOS, etc.).
- Seguir un patrón de diseño bien definido, el cual permita cuando sea necesario, reorganizar, modificar o ampliar las funcionalidades de los elementos como las vistas, ya que el patrón MVC en el que se fundamenta dota de independencia a los principales elementos que la componen.
- También el desarrollo del componente frontal del proyecto tiene que incluir capacidades de procesar datos, hacer llamadas REST. Es necesario ya que, por la naturaleza del proyecto, el frontal necesita de información que será brindada por el controlador/modelo la cual será accesible mediante recursos URI (sonidos) que actualizaran la vista.
- La tecnología empleada en el frontal tiene que permitir la integración con terceras aplicaciones, con la finalidad de poder consumir datos de diferentes fuentes de sonidos del cielo, de donde se puedan extraer los sonidos que se quieran clasificar.
- También como parte del diseño de la arquitectura con respecto al frontal, se tiene que tomar en cuenta que a futuro el asistente tendrá voz, por lo que se debe incluir todas las tecnologías necesarias para enmarcarlas dentro del diseño de la arquitectura. Esto es, contemplar también que el frontal sea capaz de integrarse con otras tecnologías de síntesis de voz para dotar de una experiencia conversacional más humana al asistente.

3.3.5.2 Diseño Visual del Frontal

La interfaz de la aplicación es un elemento importante ya que es la capa con la que el usuario final va a interactuar para clasificar los sonidos del cielo.

El diseño del frontal esta realizado para que sea accesible desde pantallas medianamente grandes como puede ser una *tablet*, y aunque nuestra aplicación está pensada para ese tipo de dispositivos, como también para ordenadores, hay que tener en cuenta que el tráfico de acceso a las utilidades de internet se produce en su mayoría desde dispositivos móviles con diferentes tipos de pantallas, sistemas, y la mayoría de estos por medio de navegadores web.

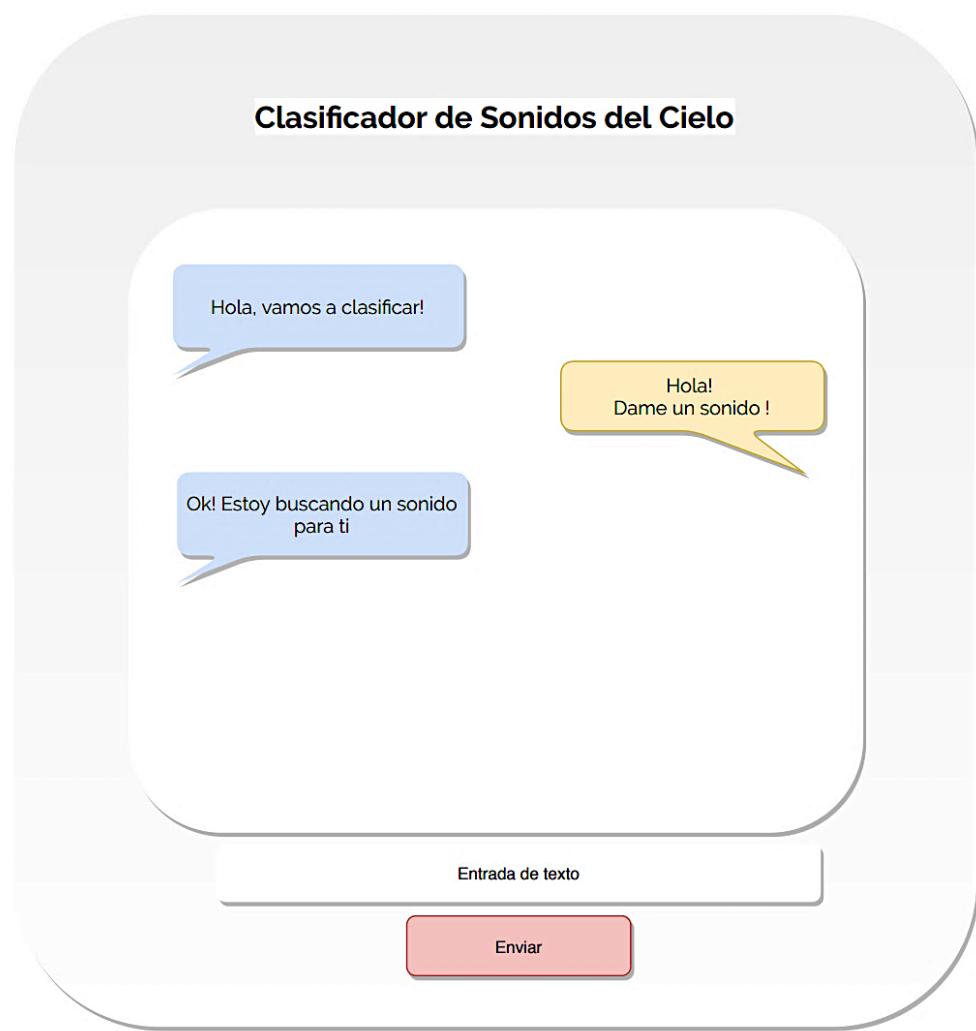


Ilustración 48. Diseño visual del concepto Chat para el asistente del proyecto.

En base a lo mencionado, se ha realizado un diseño en el que la presentación de los mensajes tiene el aspecto de una conversación por medio de un *chat* de texto con el asistente, emulando así una aplicación de mensajería de texto en donde el asistente y el usuario son los protagonistas del *chat*.

Bootstrap es un conjunto de componentes que son usados para la implementación de páginas web adaptativas. En este proyecto se usan para la definición de formularios, botones y contenedores. También se ha empleado elementos en JavaScript, para actualizar la vista cuando llegue alguna información nueva a la vista, como puede ser un mensaje o sonido.



Ilustración 49. Aspecto general de la apariencia visual en el frontal.

Una de las características del frontal con la que el usuario podrá entender el concepto al usar la aplicación es el aspecto de la interfaz. Se desea que sea amigable, que el aspecto evoque un ambiente científico, astronómico y fácil de captar a primera vista, ya que la aplicación está destinada a que sea un medio de aprendizaje entretenido, sin dejar el contexto del tema que es la clasificación de los sonidos del cielo.

Todo lo anterior ha sido desarrollado teniendo en cuenta los códigos visuales de los productos diseñados para un público infantil, como, por ejemplo: la tipografía, la paleta de colores, el estilo de ilustración y la composición de los elementos mencionados.

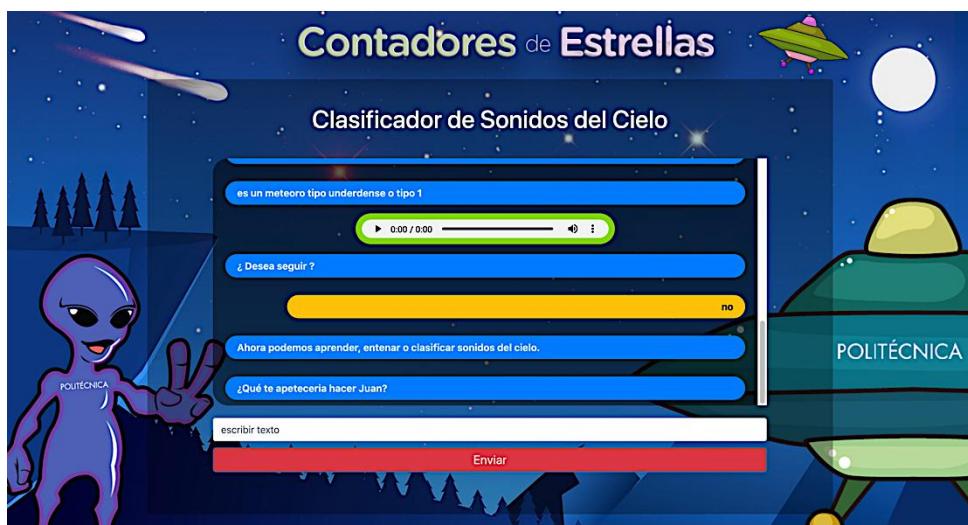


Ilustración 50. Aspecto final del frontal.

3.3.6 Interacción entre el módulo de Conversacional y el Frontal

3.3.6.1 Características de los mensajes entre módulos

Después de haber hecho las configuraciones necesarias para arrancar el servidor del asistente, el servidor de acciones y el servidor Django (Anexo) para el frontal, se mostrarán en los siguientes apartados las pruebas de integración del frontal con los demás componentes del proyecto, analizando algunos ejemplos de intercambios de mensajes para conocer cómo se comunican y la apariencia del comportamiento del frontal ante un comando del usuario.

3.3.6.2 Pruebas de integración

- Saludo y reconocimiento:

El usuario saluda al asistente y este le responde. Se muestra la captura de la aplicación Postman donde se envía el mensaje de saludo y el asistente responde. Se observa que se realiza una operación POST, en la que se envía el mensaje encapsulado en un objeto JSON. En la respuesta se observa el mensaje del asistente devolviéndole el saludo al usuario.

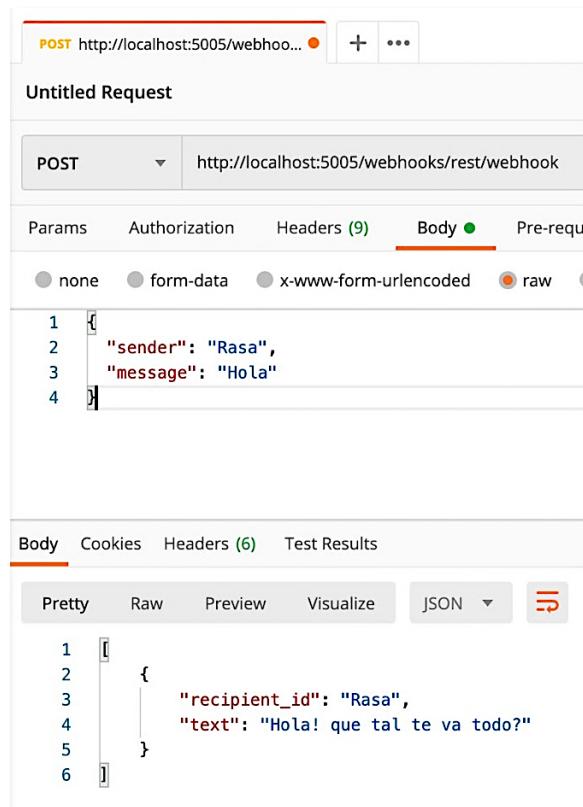


Ilustración 51. Formato de mensaje para la funcionalidad de bienvenida.

Se muestra la misma prueba, pero ahora desde el frontal del proyecto. Se observa que el sistema reconoce el nombre del usuario.



Ilustración 52. Aspecto del frontal para la bienvenida al usuario.

- Opciones de la Aplicación:

Se puede pedir ayuda en cualquier momento para mostrar las opciones del sistema. En las siguientes pruebas se contemplan los mensajes que envía el asistente en caso de reconocer que el usuario necesita ayuda para las opciones. En primer lugar, se observan los mensajes en formato JSON y también la misma prueba en el frontal.

The screenshot shows a Postman request with the following details:

- Method: GET
- URL: /options
- Body: JSON
- Headers: Content-Type: application/json
- Body (JSON):

```

1  [
2    {
3      "sender": "Rasa",
4      "message": "muestrame las opciones"
5    }
6  ]

```

The response tab shows:

- Status: 200 OK
- Time: 48 ms
- Size: 46
- Body (Pretty):

```

1  [
2    {
3      "recipient_id": "Rasa",
4      "text": "Raquel podemos hacer las siguientes cosas:"
5    },
6    {
7      "recipient_id": "Rasa",
8      "text": "Si deseas aprender, solo di 'quiero aprender', de la misma forma para entrenar o clasificar."
9    },
10   {
11     "recipient_id": "Rasa",
12     "text": "¿Qué te gustaría hacer Raquel?"
13   }
14 ]

```

Ilustración 53. Información para las opciones del sistema.

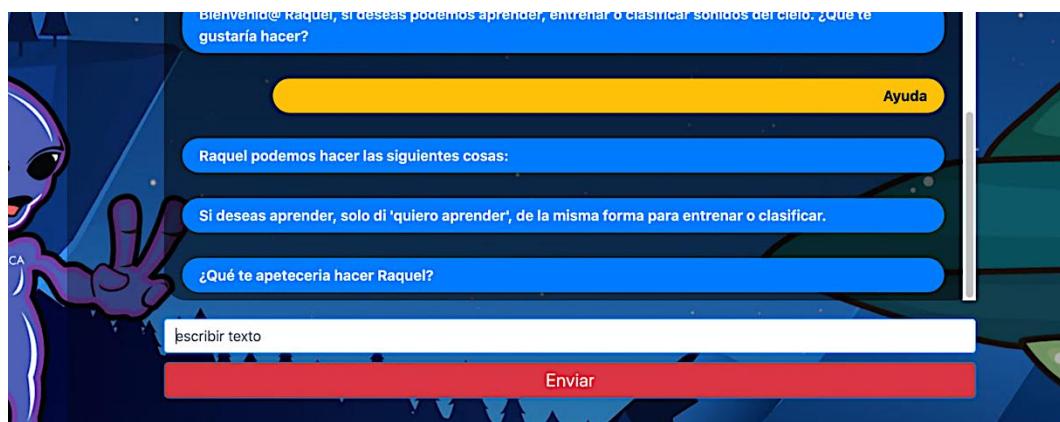


Ilustración 54. Información mostrada en los mensajes del frontal.

- Fase de Aprendizaje

La aplicación ofrece la opción de aprender los sonidos del cielo. En esta prueba se pone en funcionamiento el reconocimiento del asistente para la ejecución de la acción de sección de sonidos de para el aprendizaje, la cual está definida dentro del módulo de acciones. Se observa en la prueba cuando el asistente envía el recurso del sonido al frontal y este lo muestra en un reproductor.

```

[{"sender": "Rasa", "message": "quiero aprender"}, {"recipient_id": "Rasa", "text": "Vamos a aprender los sonidos!"}, {"recipient_id": "Rasa", "text": "Te voy a ir presentando varios sonidos"}, {"recipient_id": "Rasa", "text": "El sonido que escucharás a continuación"}, {"recipient_id": "Rasa", "text": "Es un meteorito tipo underdense o tipo 1"}, {"recipient_id": "Rasa", "custom": {"soundUri": "sounds/sonidos_entrenamiento/meteor1_underdense.wav"}}, {"recipient_id": "Rasa", "text": "amig@ seguimos con el siguiente ?"}]

```

Ilustración 55. En Postman, información para clasificar un sonido.



Ilustración 56. Reproductor de sonido para el recurso anterior.

- Prueba de Entrenamiento:

El usuario puede realizar un entrenamiento para afinar su clasificación para los sonidos. En esta opción el usuario pide entrenar, entonces el sistema le reproduce un sonido, para después preguntarle sobre el tipo de sonido que ha escuchado. Los sonidos usados en esta fase de entrenamiento son sonidos que se corresponde con los 5 tipos de sonidos con los que se pueden clasificar a los meteoritos en esta aplicación. Después de dar su respuesta el usuario recibe información sobre el sonido que acaba de escuchar.

```

POST http://localhost:5005/webhooks/rest/webhook
Send

1  "sender": "Rasa",
2  "message": "quiero entrenar"
3
4 }

Body Cookies Headers (6) Test Results
Pretty Raw Preview Visualize JSON Status: 200 OK Time: 79 ms Size: 632 B Save
1 [
2   {
3     "recipient_id": "Rasa",
4     "text": "Perfecto! Vamos a entrenar"
5   },
6   {
7     "recipient_id": "Rasa",
8     "text": "Te voy a ir presentando varios sonidos"
9   },
10  {
11    "recipient_id": "Rasa",
12    "text": "y me tendrás que decir de qué tipos son"
13  },
14  {
15    "recipient_id": "Rasa",
16    "text": "Escucha el siguiente audio:"
17  },
18  {
19    "recipient_id": "Rasa",
20    "custom": {
21      "soundUrl": "sounds/sonidos_entrenamiento/meteor1_underdense.wav"
22    }
23  },
24  {
25    "recipient_id": "Rasa",
26    "text": "amig@ de qué clase crees que es 1, 2, 3, 4 o 5?"
27  }
28 ]

```

Ilustración 57. Datos para el entrenamiento.

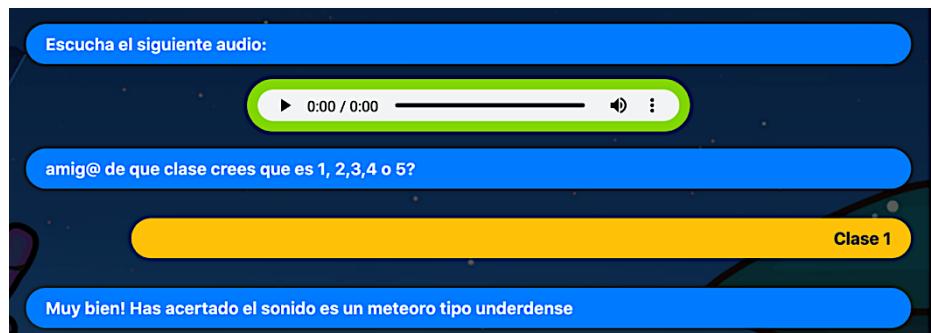


Ilustración 58. Respuesta correcta en un entrenamiento.

En la fase de entrenamiento, cuando el usuario no acierta con un sonido, también recibe información del tipo de sonido que acaba de escuchar.

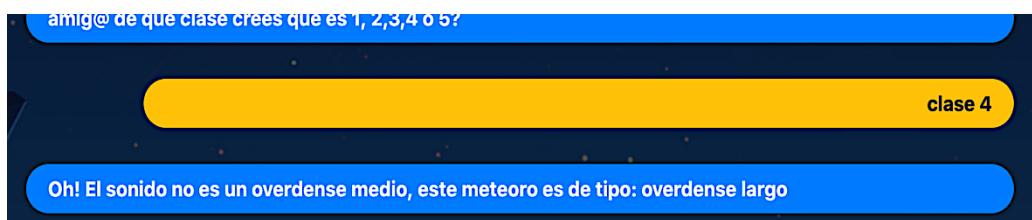


Ilustración 59. Clasificación errónea del usuario.

- Prueba de Clasificación:

El usuario accede a la fase de clasificación de sonidos del cielo en donde el sistema le irá presentando sonidos de manera secuencial, los cuales él, después de escucharlos tendrá que darles una clasificación. La clasificación que puede recibir un sonido se corresponde con la siguiente tabla:

Tabla 4. Correspondencia entre los nombres de sonidos del cielo y su clasificación en el sistema.

Definición	Clasificación
Meteoro tipo Underdense	Clase 1
Meteoro tipo M	Clase 2
Meteoro tipo Overdense Largo	Clase 3
Meteoro tipo Overdense Medio	Clase 4
Meteoro tipo Overdense Corto	Clase 5

En la prueba se solicita la clasificación. Cuando la aplicación reconoce el comando, realiza una solicitud de sonido al módulo de acciones. Se presenta el sonido y el usuario le da una clasificación, dicha clasificación se guarda. Y se muestra información sobre la clasificación actual del sonido, basada en anteriores clasificaciones.

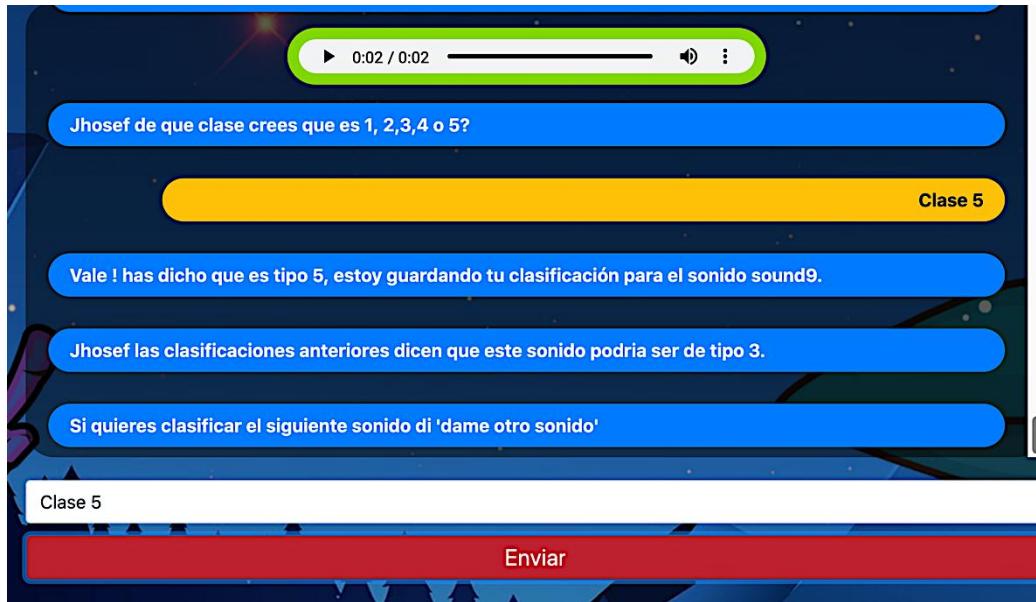


Ilustración 60. Solicitud de otro sonido para su clasificación.

En el caso de que se introduzca una clasificación invalida, el sistema responde de la siguiente manera:

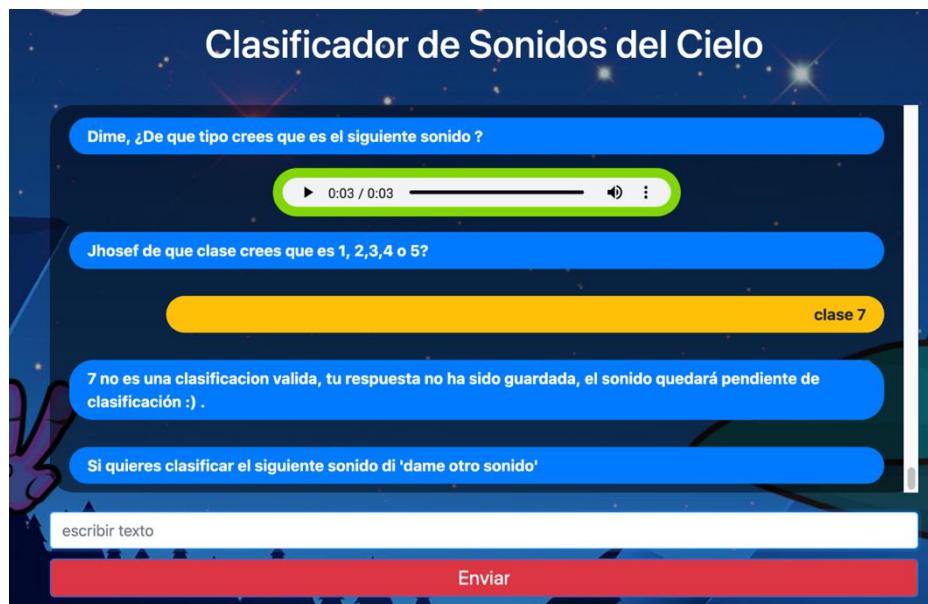


Ilustración 61. Mensaje de clasificación inválida.

- Prueba de Ayuda en la Clasificación:

En esta prueba, el usuario encuentra dificultades para poder clasificar el sonido presentado, y cuando el sistema detecta que el usuario no sabe cómo clasificar, se le ofrece ayuda.

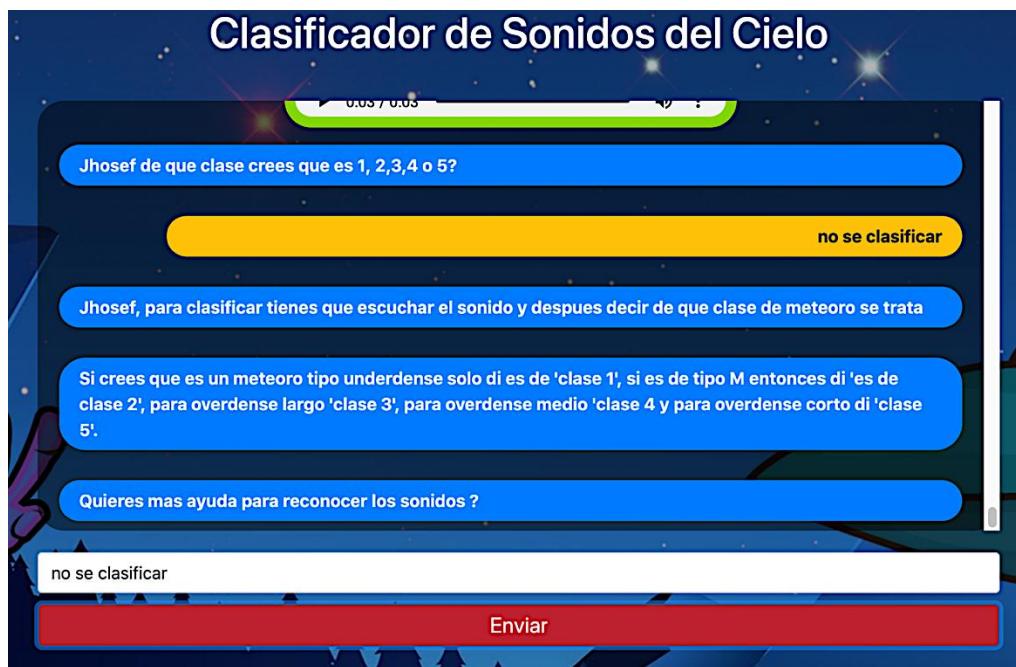


Ilustración 62. Ayuda en la clasificación.

En el caso de que el usuario siga teniendo dificultades en la clasificación se presentan más ayudas. Se le ofrece la posibilidad de repetir un sonido o repetir el entrenamiento.

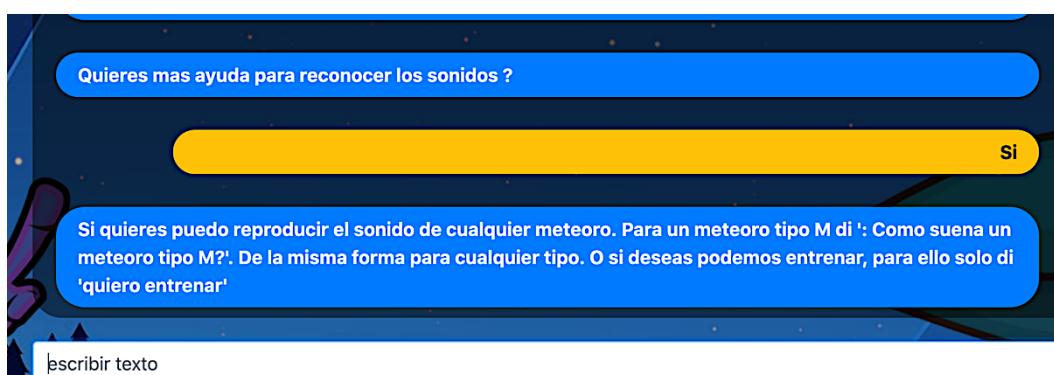


Ilustración 63. Otras opciones de ayuda en la clasificación.

Siguiendo con la conversación anterior, el usuario opta por recibir ayuda reproduciendo un sonido.

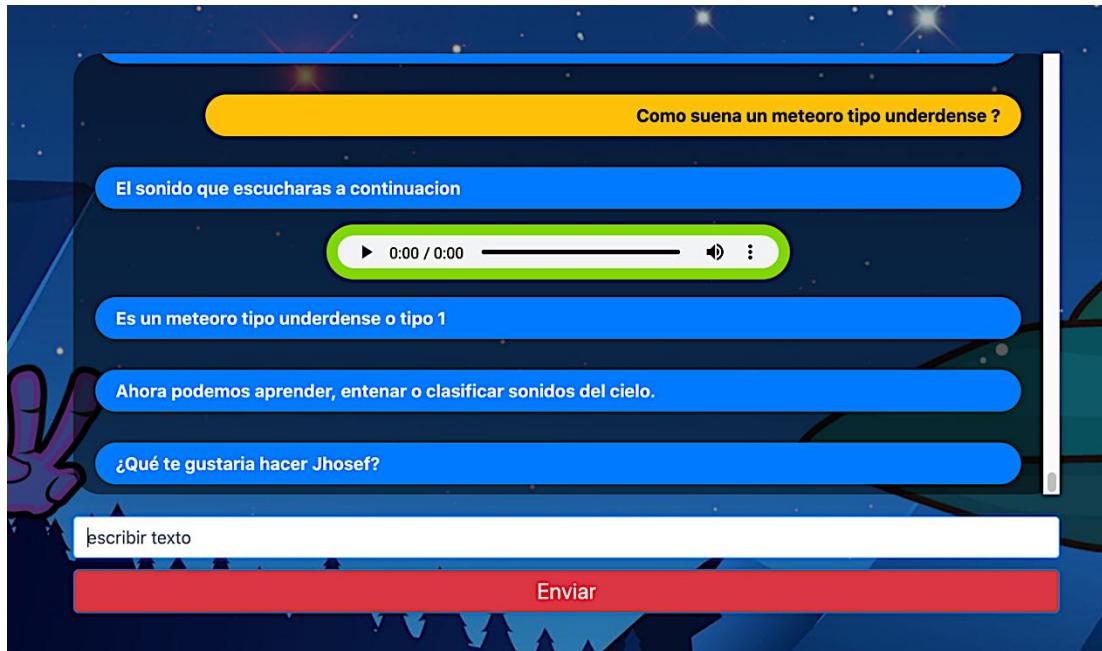


Ilustración 64. Elección de un sonido de ayuda en la clasificación.

- Prueba otros mensajes:

Si el usuario responde negativamente a la pregunta de bienvenida, el asistente le responde de la siguiente manera.

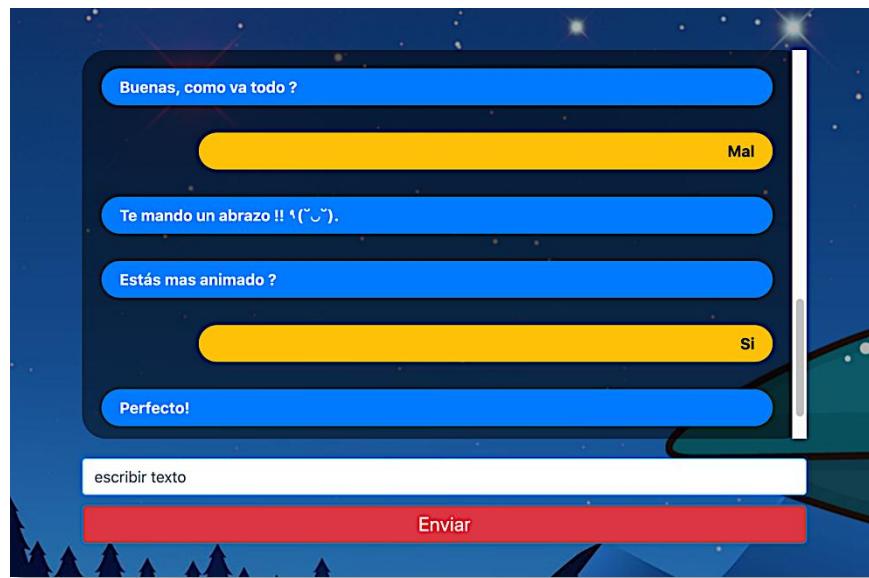


Ilustración 65. Mensaje de ánimos al usuario.

3.3.7 Tecnologías para una implementación de Voz en el frontal

Con una visión para el desarrollo a futuro, se describen tecnologías que podrían ser usadas para dotar de voz a nuestro asistente como se ha dejado señalado en la arquitectura del proyecto. Las funcionalidades como la que captura de la voz del usuario y la que reproduce la voz del asistente, se situarán en el componente vista del patrón MVC que sigue el proyecto.

El sistema cuenta con una definición de mensajes en formato JSON, el cual define de manera sencilla la información que se intercambia entre el frontal, el controlador y el modelo del sistema. Esta información estructurada se puede procesar con mucha facilidad. Al implementar el dialogo por voz en el sistema se debe considerar aquellas tecnologías que sean capaces de procesar este tipo de datos, para generar la voz o capturar los datos del habla del usuario. Se recomiendan usar las siguientes tecnologías.

3.3.7.1 Voz del Asistente - Google Cloud Text to Speech

Es un sintetizador de voz basado en la nube, por lo que si queremos usarlo tendremos crear un cliente que pueda consumir los servicios de la API que permite, según la propia documentación de Google, convertir el texto que se desee en voz humana pudiendo elegir entre 180 tipos de voces y múltiples idiomas, incluido el español [19]. Esta tecnología se recomienda por tener indicaciones para su integración basada en Python, facilitando así la integración en el frontal dentro del *framework* Django que también está basado en Python.



Ilustración 66. Logo del Sintetizador de voz de Google.

3.3.7.2 Voz del Usuario a Texto - Google Speech to Text

En el dialogo por voz con el asistente, la voz del usuario necesita ser capturada y procesada para entender qué es lo que quiere el usuario. *Google Speech to Text* es una tecnología puntera en el reconocimiento del habla, ya que cuenta con una base de entrenamiento que es prácticamente toda la web, por ello es capaz de ser usada en llamadas telefónicas y entender las diferentes entonaciones que la voz del usuario pueda tener. También cuenta con cancelación de ruido, reconoce más de 120 idiomas y además tiene API con servicios que permiten generar texto a partir del audio capturado.

Funcionamiento del Sistema

3.3.8 Comandos y Funcionalidades

Se resumen los comandos con los que el usuario podrá activar las funcionalidades del sistema.

3.3.8.1 Bienvenida

El usuario puede empezar a interactuar con la aplicación con los siguientes comandos:

- que tal?
- Hola Skyboy !
- Hey, estas ahi ?
- hola
- Buenas
- hey
- hello
- hi
- good morning
- good evening
- hey there
- Como estas ?
- Hay alguien ahi?

3.3.8.2 Aprendizaje sobre los Sonidos del Cielo

El usuario puede acceder a la funcionalidad mediante alguno de los siguientes comandos:

- quiero aprender
- que tipos de sonidos existen
- dime todos los sonidos
- deseo aprender

3.3.8.3 Selección de un Sonido en Concreto para su aprendizaje

- recuerdame como suena un underdense
- recuerdame como suena un meteoro tipo underdense
- como suena un meteoro underdense
- como suena un meteoro tipo underdense
- Como suena un meteoro tipo underdense ?
- quiero escuchar un underdense
- quiero escuchar un meteoro tipo underdense
- quiero escuchar un tipo underdense
- reproduce un meteoro underdense
- reproduce un underdense

3.3.8.4 Entrenamiento

- quiero entrenar
- entrenar
- deseo practicar
- entrenamiento
- deseo entrenarme
- quiero practicar

3.3.8.5 Clasificación

- clasificar
- deseo clasificar
- quiero clasificar
- vamos a clasificar

3.3.8.6 Ayuda en la Clasificación

- como se clasifica ?
- como clasificar ?
- dime como se clasifica
- ayuda para clasificar
- me dices como clasificar
- no se clasificar
- no puedo clasificar
- no se clasificar

3.3.8.7 Ayuda General

- necesito ayuda
- dime las opciones
- cuales son las opciones
- repiteme las opciones
- ayuda
- help
- repite las opciones
- me dices las opciones del sistema
- que opciones tengo

3.3.9 Organización de los Directorios del Proyecto

Se muestra un mapa organizativo de los directorios principales del proyecto. En el primer nivel se encuentran: *documentación*, *mvc_control_bot_juego*, el cual contiene toda la implementación del asistente y el módulo de acciones, y *mvc_vista_frontal*, el cual contiene toda la implementación de del frontal del proyecto.

```

1   .
2   |   README.md
3   |   README.pdf
4   |   documentation
5   |       000_conf.jpeg
6   |       001_conf.png
7   |       002_conf.png
8   |       003\ TFG\ Entrega\ Memoria\ Final.DOCX
9   |       003\ TFG\ Entrega\ Memoria\ Final.pdf
10  |       003_conf.png
11  |       005_conf.png
12  |       architecture.png
13  |       cat.gif
14  |       code_doc.html
15  |       demo.png
16  |       filesystem-proy.txt
17  |       presentacion_tfg_JhosefCardichPalma.pptx
18  |       ~$3\ TFG\ Entrega\ Memoria\ Final.DOCX
19  |       filesystem-proy.txt
20  |       models
21  |           mvc_control_bot_juego
22  |               __init__.py
23  |               __pycache__
24  |                   __init__.cpython-37.pyc
25  |                   actions.cpython-37.pyc
26  |               actions.py
27  |               config.yml
28  |               credentials.yml
29  |               data
30  |                   nlu.md
31  |                   stories.md
32  |               domain.yml
33  |               endpoints.yml
34  |               models
35  |                   20200630-155437.tar.gz
36  |                   20200701-174950.tar.gz
37  |               sounds
38  |                   classify_sounds
39  |                   sonidos_asistente
40  |                   sonidos_entrenamiento
41  |               tests
42  |                   conversation_tests.md
43  |               venv
44  |                   bin
45  |                   lib
46  |                   pyvenv.cfg
47  |       mvc_vista_frontal
48  |           mysite
49  |               db.sqlite3
50  |               manage.py
51  |               mysite
52  |               sonidos
53  |                   sonidosdelcielo
54  |               requirements.txt
--
```

Ilustración 67. Organización de los directorios del proyecto.

4 Conclusiones y Propuestas de Ampliación

4.1 Conclusiones

El objetivo de este proyecto ha sido la realización de una aplicación que fuera capaz de ofrecer una experiencia interactiva para la clasificación de los sonidos del cielo. La aplicación ha sido creada con múltiples tecnologías para poder conseguir componentes robustos, independientes y bastante modernos lo cual abre la posibilidad de poder ampliar o mejorar las características de este proyecto. En el frontal se ha usado lenguajes y componentes web HTML, CSS, JavaScript y Bootstrap, todo esto soportado por Django el cual brinda el procesamiento necesario en el frontal.

Toda la información de clasificación de los sonidos se encuentra en los archivos asociados de clasificación asociados a cada sonido. Esta información es importante porque abre la puerta a investigaciones sobre estos datos, pudiendo obtenerse métricas de esta información, la cual puede resultar útil para los científicos interesados en este campo. En cada línea se guarda la clasificación dada por un usuario, separado por un espacio, y a continuación el nombre del usuario que clasificó dicho sonido

Para definir el asistente se ha usado el prometedor *framework* Rasa, el cual tiene múltiples posibilidades de integración y una arquitectura bastante bien definida. Para el procesamiento de los cálculos se ha usado Python, lenguaje el cual tiene múltiples librerías que hacen posible la realización de las tareas de cada acción necesaria para las funcionalidades.

El asistente brinda una experiencia más humana, haciendo así la interacción más dinámica, que, en conjunto con la interfaz, el usuario podrá clasificar los sonidos del cielo *chateando* con el asistente y requiriendo cualquier funcionalidad implementada en cualquier momento.

El comportamiento del asistente puede ser modificado fácilmente, también los mensajes, solo hay que realizar cambios en el archivo “*domain.yml*”.

La realización de este proyecto ha implicado el esfuerzo propio que tiene abordar el desarrollo con herramientas nuevas, como consecuencia se han adquirido conocimientos de tecnologías bastante interesantes, las cuales han generado en el autor, interés por ciertos campos como son la inteligencia artificial, el aprendizaje automático, etc.

Se han completado todos los objetivos previstos del proyecto, dejando cimentadas los conceptos, con recomendaciones para la implementación de la voz del asistente a futuro.

El proyecto ha sido un reto, sin embargo, ha sido abordado de la manera más positiva, con entusiasmo e ilusión, ya que este proyecto define el término de una emocionante etapa en la vida del autor.

4.2 Propuestas de Mejora

Con una perspectiva del futuro para este proyecto se lista una serie de funcionalidades o mejoras, obtenidas durante el desarrollo, las pruebas de usabilidad:

- Síntesis de Voz: Es la principal característica que se puede desarrollar para la aplicación. Esto llevaría a otro nivel la experiencia de usuario para clasificar sonidos del cielo.
- Implementación para la captura de la voz: Esta tecnología haría que el usuario no dependiera del teclado para poder ordenar un comando al sistema, al igual que la característica anterior, haría más interactiva la experiencia de clasificación.
- En cuanto al Asistente: Se ha definido un conjunto de entrenamiento para la definición de su comportamiento, sin embargo, hay características que se pueden mejorar con la interacción de otros componentes compatibles con Rasa, por ejemplo: Spacy que es un módulo (tubería en Rasa) que brinda procesamiento natural del lenguaje, aportando datos en español para el entrenamiento del asistente de voz.
- El asistente al usar inteligencia artificial aprende de las órdenes del usuario, por lo que llega un momento en el que a veces toma decisiones sobre la siguiente acción a tomar independientemente de la definición de las reglas de su comportamiento. Esto puede generar problemas ya que el conjunto de datos y el número de entrenamientos es bastante limitado, por lo que dada una entrada un poco difusa el asistente podría dar una respuesta inesperada. Se necesita más documentación para saber cómo modificar los componentes de Rasa para hacer que el asistente resuelva una situación que desconoce no dando una respuesta cualquiera, sino forzar una acción en concreto.
- La integración con otras aplicaciones puede ser posible. Como esto surge la posibilidad de poder ofrecer el juego de clasificación integrado dentro de otras aplicaciones como pueden ser Telegram, Facebook Messenger, etc. Esto es una posibilidad bastante interesante, ya que haría que la aplicación pudiera llegar a más usuarios rápidamente, porque no tendrían la necesidad de acceder a la aplicación frontal mediante un navegador, sino simplemente usar las aplicaciones de mensajería usuales como las mencionadas para clasificar sonidos del cielo.
- En la parte del frontal se ha empleado Bootstrap, en la cual con ciertas modificaciones se podría hacer la apariencia más *responsive*, es decir, que la aplicación se adapte a diferentes tipos de pantallas mejorando así la usabilidad desde cualquier dispositivo, como pueden ser los teléfonos móviles.

- Otra característica importante, es que se pudiera desarrollar dentro del frontal un cliente que pudiera consumir los datos, sonidos, clasificaciones desde una base de datos. Esta base de datos podríamos pensar que fuera la que se emplea para almacenar los sonidos que se generan a partir de las antenas de radio que capturan los sonidos del cielo. Entonces se podría consumir sonidos, guardar las clasificaciones y otras métricas en la mencionada base de datos, para el estudio de toda la información que genere el clasificador de sonidos del cielo.
- Se podría crear dentro de la aplicación, las estadísticas de clasificación a modo de *ranking* de usuarios de clasificación, formando así una comunidad y animando a los participantes a clasificar los sonidos. Se podría mostrar la información del *ranking* de clasificaciones en el frontal.
- Un sistema que pudiera soportar una comunidad de usuarios, que se pueden dar de alta o iniciar sesión en la aplicación. Las funcionalidades mencionadas se tendrían que implementar en el (modelo del) frontal, con la facilidad que ofrece Django, para desarrollar funcionalidades para que los usuarios se puedan dar de alta, iniciar sesión para acceder a sus clasificaciones y su puntuación, etc.

5 Bibliografía

- [1] AstroCantabria. [En línea]. Available: <https://astrocantabria.org/?q=estrellas-fugaces>. [Último acceso: 1 6 2020].
- [2] N. Science, «Nasa Solas System Exploration,» 25 Enero 2019 . [En línea]. Available: <https://solarsystem.nasa.gov/>. [Último acceso: 5 Junio 2020].
- [3] C. d. Estrellas, «Contadores de Estrellas,» [En línea]. Available: <http://www.contadoresdeestrellas.org/proyecto/>. [Último acceso: 7 Junio 2020].
- [4] B. A. Shawar y E. Atwell, «Chatbots: Are they Really Useful?,,» de *Ldv forum, 2007*, 2007.
- [5] M. Arya, «A brief history of Chatbots,» 11 3 2019. [En línea]. Available: <https://chatbotslife.com>. [Último acceso: 6 6 2020].
- [6] D. Bach, «Cómo utilizan bots las organizaciones internacionales de atención médica para ayudar a combatir el COVID-19,» 8 4 2020. [En línea]. Available: <https://news.microsoft.com/e>. [Último acceso: 7 Junio 2020].
- [7] Apple, «Siri de Apple,» [En línea]. Available: <https://www.apple.com/siri/>. [Último acceso: 5 6 2020].
- [8] A. B. Braden Hancock, P.-E. Mazaré y J. Weston, «Learning from Dialogue after Deployment: Feed Yourself, Chatbot!,» 16 1 2019. [En línea]. Available: <https://arxiv.org/abs/1901.05415>. [Último acceso: 6 Junio 2020].
- [9] O. BBVA y J. Bryson, «La última década y el futuro del impacto de la IA en la sociedad,» 2018. [En línea]. Available: <https://www.bbvaopenmind.com/articulos/la-ultima-decada-y-el-futuro-del-impacto-de-la-ia-en-la-sociedad/>. [Último acceso: 7 Junio 2020].
- [10] Lifegate, «Lifegate,» [En línea]. Available: <https://www.lifegate.com/domotics-home-automation>. [Último acceso: 5 6 2020].
- [11] M. McLaughlin, «What a Virtual Assistant is and How it Works,» 29 Eneero 2020. [En línea]. Available: <https://www.lifewire.com/virtual-assistants-4138533>. [Último acceso: 8 Junio 2020].
- [12] E. Lokøy, «Voice it! Will virtual assistants become a vital part of the future?,» 24 4 2020. [En línea]. Available: <https://www.tietoevry.com/en/blog/>. [Último acceso: 4 6 2020].

- [13] A. Anisimova, «The Basics of Voice Assistants Explained,» 15 Noviembre 2018. [En línea]. Available: <https://idapgroup.com/>. [Último acceso: 8 Junio 2020].
- [14] A. MUTCHLER, «What are Virtual Assistants - VoiceBot AI,» 5 Octubre 2019 . [En línea]. Available: <https://voicebot.ai/2019/10/05/what-are-virtual-assistants/>. [Último acceso: 8 Junio 2020].
- [15] spaCy, «spaCy,» spaCy, [En línea]. Available: <https://spacy.io/>. [Último acceso: 7 Jun 2020].
- [16] Rasa, «Rasa Docs,» [En línea]. Available: <https://rasa.com/docs/rasa/user-guide/architecture/>. [Último acceso: 6 5 2020].
- [17] Rasa, «Rasa,» [En línea]. Available: <https://rasa.com/docs/rasa/nlu/about/>. [Último acceso: 10 Jun 2020].
- [18] Rasa, «Rasa SDK,» Rasa, [En línea]. Available: <https://rasa.com/docs/rasa/api/rasa-sdk/#rasa-sdk>. [Último acceso: 1 Mayo 2020].
- [19] G. Cloud, «Google Cloud,» [En línea]. Available: <https://cloud.google.com/text-to-speech?hl=es-419>. [Último acceso: 20 6 2020].

6 Anexos

6.1 Información de Configuraciones del Proyecto

En este apartado se resumen algunas configuraciones necesarias para la puesta en marcha de la aplicación.

6.1.1 Repositorio del Proyecto

Toda la implementación del proyecto reside en sistema de control de versiones GitHub, accesible desde el siguiente enlace web:

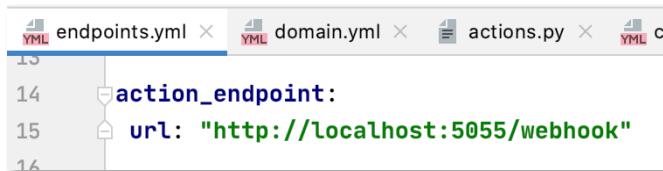
<https://github.com/JhosefCardich/UPM-ChatBot-SoundsOfMeteors>

6.1.2 Configuración de los Servidores del proyecto

El proyecto cuenta con tres servidores en los cuales hay que realizar ciertas modificaciones para poder conectar los *endpoint* con los clientes y poder consumirlos .

6.1.2.1 Endpoints del servidor de acciones y del Asistente en Rasa

Para el consumo del *endpoint* del servidor de acciones se tiene que modificar el archivo “*endpoints.yml*”. Para las pruebas se utilizado la configuración por defecto:



```
YML endpoints.yml × YML domain.yml × actions.py × YML c
13
14     action_endpoint:
15         url: "http://localhost:5055/webhook"
16
```

Ilustración 68. Endpoint del Servidor de Acciones.

Para exponer el *endpoint* del asistente y conectarlo con el frontal hay que tener en cuenta la siguiente dirección que se genera de manera automática al arrancar el servidor Rasa para ejecutar el asistente:

```
rasa run --enable-api --cors "*"
```

```
- Starting Rasa server on http://localhost:5005
```

6.1.2.2 El Servidor Django

- Chat.js

Es el cliente que encargado de las operaciones REST. En este archivo se tiene que modificar la variable “base”, con la URI base donde se encuentra publicado el *endpoint* del asistente. En el momento de las pruebas que se han redactado, se ha usado “localhost”:

```
1  $(document).ready(function () {  
2  
3      $("#boton-chat").click(function(){  
4          base = "http://localhost";  
5          texto = $("#texto-box").val();
```

Ilustración 69. Configuración del cliente definido en chat.js.

Líneas más abajo, en el mismo archivo, en la negociación, hay que modificar el recurso sobre el que se realiza la operación POST. En Rasa se ha usado “:5005/webhooks/rest/webhook”:

```
14  
15  
16  
17  
18      url: base + ":5005/webhooks/rest/webhook",  
contentType: 'application/json',  
data: JSON.stringify(content),  
type: 'POST',  
success: function (data) {
```

Ilustración 70. Negociación de la Operación POST.

En las siguientes páginas se muestran las demás configuraciones y un resumen del proyecto disponibles también en el archivo “README.md”.

ChatBot para Clasificar Sonidos del Cielo

Sky Sounds Chatbot Classifier

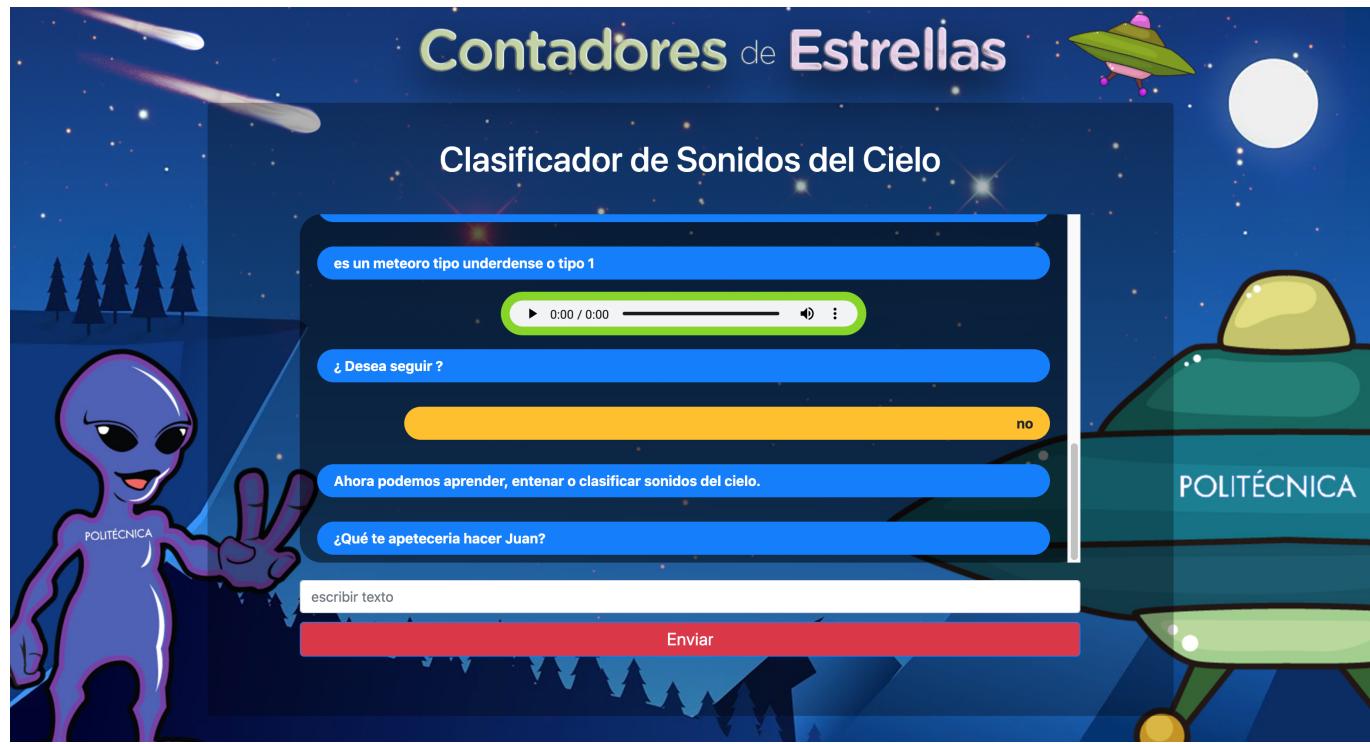


Tabla de contenidos

- Introducción
- About
- Arquitectura
- Instalación
- Características
- Team

Introducción

- El proyecto tiene como objetivo el diseño y desarrollo de una arquitectura para la implementación de una aplicación, que tiene la principal funcionalidad de la clasificación de sonidos del cielo mediante un juego orientado al público infantil.

About

- This chatbot is being developed by [Jhosef A. Cardich Palma](#) as part of his Final Degree Project (TFG) for the [Polytechnic University of Madrid](#) as a part of its Computer Science Degree on the [Higher Technical School of Computer Engineers](#). The **Sky Sounds Chatbot Classifier** is an application that offers an entertaining experience to classify sounds of the sky, bringing the general public closer to

science. This application is part of [Star counter](#), a large project in collaboration with the **Instituto Astrofísico de Canarias**.

-  Este proyecto esta siendo desarrollado por [Jhosef A. Cardich Palma](#) como el trabajo de fin de carrera en la [Universidad Politécnica de Madrid](#) para el grado de Ingeniería Informática en la [Escuela Técnica Superior de Ingenieros Informáticos](#). La aplicación **Chatbot Clasificador de Sonidos del Cielo** tiene como objetivo la realizacion de un bot para el publico infantil, el cual podra clasificar los sonidos del cielo interactuando con el asistente virtual. Esta aplicación forma parte de un proyecto mas grande. [Contadores de Estrellas](#) es un proyecto realizado en colaboración con el [Instituto Astrofísico de Canarias](#).



Arquitectura

La implementación general se ha seguido un patrón **Modelo Vista Controlador (MVC)** , para definir los componentes y sus interacciones.

Instalación

- En este repositorio se encuentra el proyecto, en el que se incluye el entorno virtual con el que se ha trabajado en local (carpeta venv) sin embargo, en esta carpeta, se encuentra otra llamada "lib" (/venv/lib), la cual no se esta sincronizando con el repositorio aqui (pero si usa en local), debido a que esta carpeta tiene dos modulos que pesan demasiado, por ello es recomendable, si deseas hacer funcionar esto:

Configuración Entorno

- Antes que nada, tenemos que instalar [RASA](#)

Instalación RASA

```
$ pip3 install rasa
```

- Una vez clonado el proyecto, hay que hacer una serie de comprobaciones. Se ha usado el entorno Pycharm para el desarrollo del asistente, el siguiente paso hace referencia las opciones de Pycharm. Hay

que asegurarnos que estamos usando la version de Phyton 3.7 y que el entorno venv esta configurado correctamente, también lo haremos cuando hayamos instalado RASA. Podemos encontrar las configuraciones en :

Abrir el proyecto con Pycharm:

```
> Pycharm > preferencias ..
```

Configuración del módulo del juego y del ChatBot

Modulo Back - Lógica del Juego

Servicios RASA

Otra parte escencial del proyecto es la lógica del juego, para ello se ha desarrollado una aplicación en Phyton, integrada en framework de RASA, la cuál define las funcionalidad de acceso a los sonidos, la clasificación, almacenamiento de los datos de clasificación, y otros procesamientos independientes de la parte conversacional que es el bot. Estos servicios serán usados por el bot cuando este reconozca un comando por parte del usuario.

- Para iniciar los servicios solo tenemos que situarnos en el directorio anterior y ejecutar el siguiente comando:

```
$ rasa run actions...
```

Módulo Back - ChatBot

- Para entrenar nuestro modelo nos situamos en el siguiente directorio:

```
"/UPM-ChatBot-SoundsOfMeteors/mvc_control_bot_juego"
```

- Estando ya en el directorio, ejecutamos el siguiente comando para entrenar el modelo:

```
$ rasa train
```

- Una vez que nuestro rasa ha terminado de formar el universo de nuestro bot este se encuentra ya listo para poder usarlo. Entonces podemos conversar con el bot conectando un frontal web o la consola de comandos. La segunda opción es la mas inmediata. Para poder comunicarnos con nuestro bot via consola, ejecutamos el siguiente comando en una terminal. Este inicia el servidor RASA y ademas nos da una terminal de entrada para poder comunicarnos via texto con nuestro bot:

Para iniciar el servidor rasa donde correrá nuestro bot:

```
$ rasa shell
```

- Si el comando se ha realizado con éxito, se mostrara un mensaje como este:

```
jhos@MacBook-Pro ~/Desktop/GIT-LOCAL/UPM-ChatBot-SoundsOfMeteors/mvc_control_bot _juego (master) $ rasa shell
2020-06-25 19:36:31 INFO    root - Connecting to channel 'cmdline' which was specified by the '--connector' argument. Any other channels will be ignored. To connect to all given channels, omit the '--connector' argument.
2020-06-25 19:36:31 INFO    root - Starting Rasa server on http://localhost:5005
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input -> hola
Buenas, como va todo ?
Your input ->
```

Integración con APPS externas

En el caso de la integración con aplicaciones externas lo que tenemos que hacer para exponer los servicios de nuestro tenemos que seguir el siguiente comando . Se pueden realizar pruebas sobre mensajes con el software postman.

Exponer los servicios de nuestro bot

- Nos situamos siempre en el directorio donde hemos entrenado nuestro bot y ejecutamos:

```
$ rasa run
```

Consumir mediante POSTMAN

- En el caso de que el servidor se haya iniciado sin nigen problema, el aspecto de la terminar es la siguiente:

```
jhos@MacBook-Pro ~/Desktop/GIT-LOCAL/UPM-ChatBot-SoundsOfMeteors/mvc_control_bot _juego (master) $ rasa run
2020-06-25 19:48:30 INFO    root - Starting Rasa server on http://localhost:5005
```

Consumir Mediante un Navegador - Cliente Web

Exponer los servicios de nuestro bot con Extra Google Chrome

- Si queremos conectar nuestro asistente desde un navegador, es decir, conectar el frontal con el controlador/modelo del proyecto, hay que usar el siguiente comando que desactivara ciertas características de seguridad de RASA las cuales entran en conflicto con los navegadores. Esto se tiene que hacer al usar el frontal creado para el proyecto. Ejecutar el siguiente comando:

```
rasa run --enable-api --cors "*"
```

Consumir servicios

- Podremos consumir los servicios en la siguiente URI (**POST**):

```
http://localhost:5005/webhooks/rest/webhook
```

Una prueba de operación **POST** en postman, donde se ve el mensaje enviado y la contestación del asistente. Hay que notar un detalle, y es que el la ultima parte del mensaje enviado por el bot ha sido generado por la aplicación del juego definida en las acciones de RASA, las cuales están activas porque las hemos activado previamente (Modulo Back - Lógica del Juego) .

The screenshot shows the Postman interface with the following details:

- Request URL:** `http://localhost:5005/webhooks/rest/webhook`
- Body Content (Raw JSON):**

```
1 {
2   "sender": "Rasa",
3   "message": "clasificar"
4 }
```
- Response Preview (JSON):**

```
1 [
2   {
3     "recipient_id": "Rasa",
4     "text": "Muy bien! vamos a clasificar"
5   },
6   {
7     "recipient_id": "Rasa",
8     "text": "Estoy escogiendo un sonido para ti..."
9   },
10  {
11    "recipient_id": "Rasa",
12    "text": "Dime, ¿De que tipo crees que es el siguiente sonido ?"
13  },
14  {
15    "recipient_id": "Rasa",
16    "custom": {
17      "soundUri": "sounds/classify_sounds/sound9.wav"
18    }
19  }
20 ]
```

En ese momento los servicios estarán disponibles para que nuestro asistente pueda llamarlos si reconoce alguno en la conversación con el usuario.

Configuración del Frontal

Por la arquitectura propuesta, se ha desarrollado una aplicación frontal basada en el framework Web de Django.

- Para poder hacer funcionar el frontal de la aplicación tenemos que situarnos el directorio:

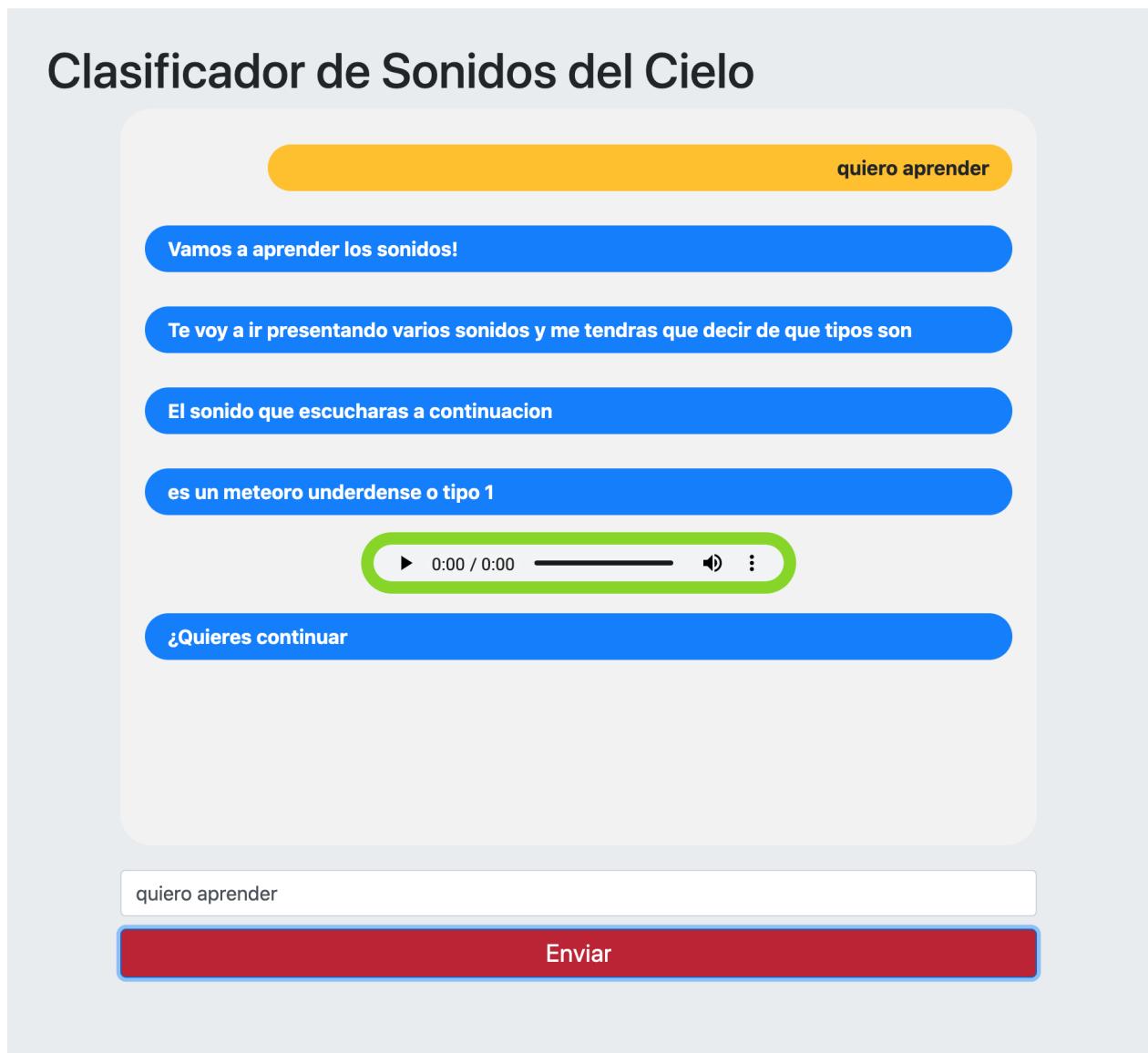
```
/UPM-ChatBot-SoundsOfMeteors/mvc_vista_frontal/mysite
```

- Hay que arrancar el servidor de la siguiente manera:

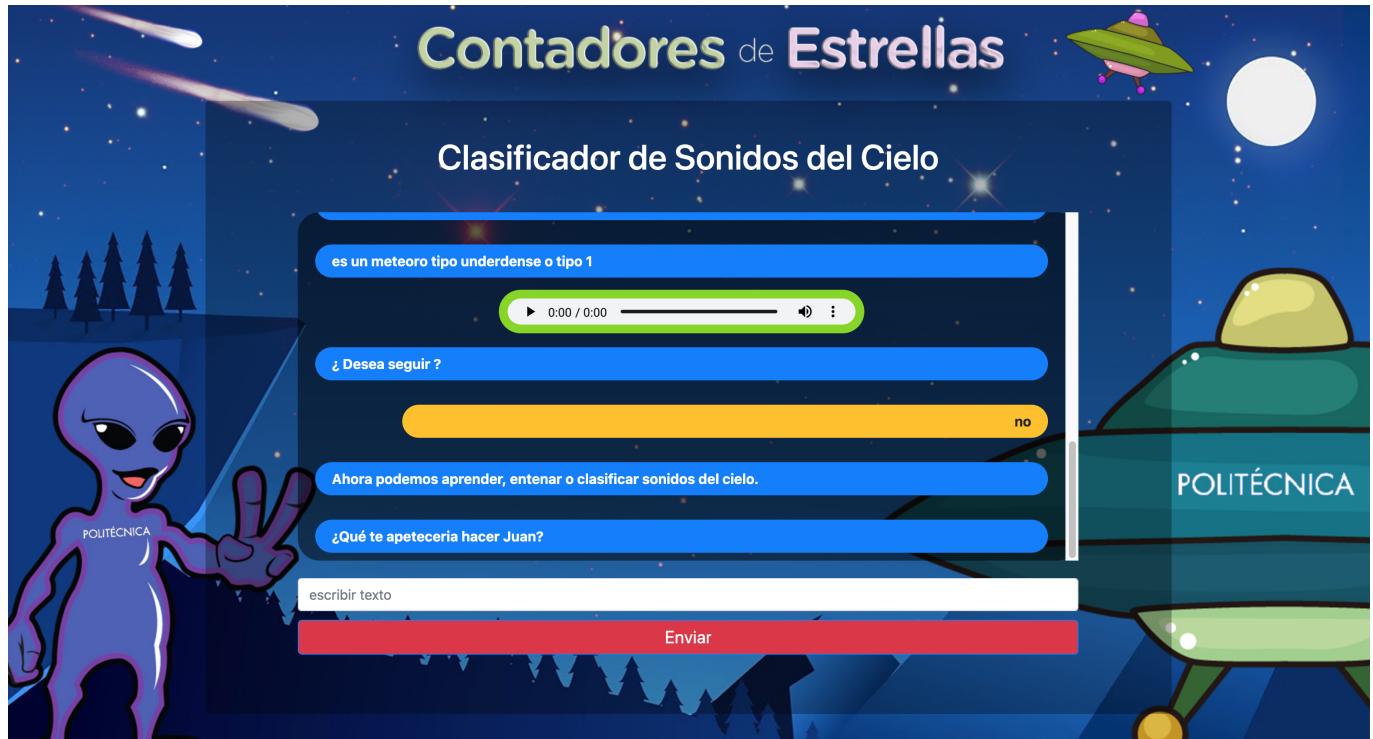
```
$ python3 manage.py runserver 0.0.0.0:8000
```

Vista Previa Frontal

- V1.5



- V2.0



Gracias a esta interfaz el usuario podrá ser capaz de interactuar con el asistente de manera más amigable para clasificar sonidos del cielo.

Características

La aplicación permite la clasificación de sonidos del cielo y cuenta con las siguientes funcionalidades:

Aprendizaje

- La aplicación brinda la posibilidad de poder aprender los sonidos del cielo a través del asistente conversacional.

Para aprender el sistema responde a los siguientes comandos:

- quiero aprender
- que tipos de sonidos existen
- dime todos los sonidos
- deseo aprender

Entrenamiento

- Existe un nivel de entrenamiento para que el usuario pueda aprender los sonidos. En este se presentan sonidos del aprendizaje pero sin información sobre su tipo. El sistema se encargara de evaluar la respuesta del usuario para devolver feedback sobre que tan acertada ha sido su respuesta.

- quiero entrenar
- entrenar
- deseo practicar
- entrenamiento
- deseo entrenarme
- quiero practicar

Clasificación

- Es el nivel mas interesante, aqui el sistema presenta diversos tipos de sonidos que el usuario tendra que clasificar. Cuando el usuario clasifique un sonido, su clasificación se guarda junto con la de otros usuarios. Así cuando un sonido es clasificado, al usuario se le muestra el valor promedio de clasificación que otros usuarios le han dado a ese sonido
- Cuando un usuario escucha un sonido y no puede intuir de que se trata, entonces el usuario podra pedir ayuda. En las opciones de ayuda el usuario puede elegir reproducir cualquiera de los 5 sonidos existentes. Si el usuario aun se ve con dudas, puede volver a entrenar o a aprender.

Para clasificar el sistema se activa con cualquiera de los siguientes comandos

- clasificar
- deseo clasificar
- quiero clasificar
- vamos a clasificar

Ayuda para Clasificar

Si el usuario no se siente seguro o necesita ayuda para reconocer un sonido durante una clasificación, entonces el sistema puede ayudarle mostrandole algun ejemplo o repetir la sesión de entrenamiento.

Para mostrar las opciones de ayuda durante la clasificación el sistema responde a los siguientes comandos:

- como se clasifica ?
- como clasificar ?
- dime como se clasifica
- ayuda para clasificar
- me dices como clasificar
- no se clasificar
- no puedo clasificar
- no se clasificar

Tipos de Sonidos

El usuario podra reproducir un tipo de sonido en cualquier momento para recordar. Para ello se usa cualquier de los siguientes comandos. (Ejemplo para un overdense corto):

Reproducir un Sonido

- recuerdame como suena un overdense corto
- recuerdame como suena un meteoro tipo overdense corto
- como suena un meteoro overdense corto
- como suena un meteoro tipo overdense corto
- Como suena un meteoro tipo overdense corto ?
- quiero escuchar un overdense corto
- reproduce un meteoro overdense corto
- reproduce un tipo overdense corto

Dependencias

Los siguientes paquetes de software son necesarios en el sistema para poder hacer funcional la aplicación:

- [Python 3.7](#)
- [Rasa](#)
- [Django 3.0.7](#)
- Django Rest framework 3.11.0

Team

Contributors/People

| [**Jhosef Anderson Cardich Palma**](#) |
