



Universidad Politécnica
de Madrid

**Escuela Técnica Superior de
Ingenieros Informáticos**



Grado en Grado en Ingeniería Informática

Trabajo Fin de Grado

**Desarrollo de un Chatbot para Público
Infantil para Clasificar Sonidos del Cielo.
Versión 2**

Autor: Marcos Pino Gamazo
Tutor(a): Raquel Cedazo León

Madrid, «Febrero 2021»

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 Grado en Ingeniería Informática

Título: Desarrollo de un Chatbot para Público Infantil para Clasificar Sonidos del Cielo. Versión 2

«Febrero 2021»

Autor: Marcos Pino Gamazo
Tutor: Raquel Cedazo León
Ingeniería Eléctrica, Electrónica Automática y Física Aplicada
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Este Trabajo Final de Grado tiene como misión el diseño y desarrollo de una aplicación cuya principal funcionalidad es, la clasificación de meteoros mediante sonidos generados por ordenador a partir de detecciones llevadas a cabo mediante radiofrecuencia.

El proyecto Sonidos del Cielo tiene como objetivo hacer accesible la ciencia a personas con discapacidad visual y para ello, se pretende desarrollar una herramienta mediante la cual se pueda seguir una conversación en *lenguaje natural*, estas herramientas se conocen por el término en inglés *chatbot*. Será necesario que la herramienta sea capaz de reconocer la voz y a su vez transmitir mediante voz la información generada por la aplicación. Esto va a facilitar el acceso a la herramienta no sólo a personas con problemas de visión sino también a niños que aún no sepan leer o escribir.

El diseño completo del proyecto requiere del diseño y desarrollo estructural tanto de una bases de datos en las que se almacenen los elementos necesarios para el proyecto (sonidos de meteoros, curvas de luz, espectrograma, parámetros de detección, etc) como de una API Rest que sirva como controlador y ofrezca servicios entre la base de datos y el chatbot y futuras herramientas que puedan requerirla.

Además, se ha desarrollado una versión preliminar del chatbot para realizar las pruebas necesarias sobre la API Rest y así consolidar las tecnologías y servicios a utilizar en el proyecto Sonidos del Cielo.

Abstract

The objective of this Final Degree Project is the design and development an application whose principal functionality is the classification of meteors using computer-generated sounds from meteor echoes detected by radio frequency.

The Sonidos del Cielo project aims to make science accessible to people with visual disabilities and to achieve this, it is intended to develop a tool through which you can have a conversation in natural language, these tools are known as chatbots for the union of the words chat and robot.

It will be necessary for the tool to be able to recognize the voice and at the same time transmit information generated by the chatbot by voice, which will facilitate access to the tool not only for people with visual disabilities but also for children who still can't read or write.

The complete design of the project requires the design and structural development of a database in which all the necessary elements for the project are stored (meteor sounds, light curves, spectrograms, detection parameters, etc) and a REST API that will be used as a controller and will offer services between the database and the chatbot and future tools that may require it.

In addition, a preliminary version of the chatbot has been developed to carry out the necessary test on the REST API and thus consolidate the technologies and services to be used in the Sonidos del Cielo project.

Agradecimientos

Este trabajo no hubiese sido posible sin la ayuda de mi tutora, Raquel y de todos mis compañeros de equipo, que me han brindado su ayuda y conocimientos siempre que los he necesitado.

A mi familia y mi pareja, por acompañarme en los momentos buenos y ayudar a levantarme en los malos.

A mis abuelas, sin vosotras no hubiese llegado donde estoy ahora mismo, os estaré eternamente agradecido.

Por último, a mi abuelo, me hubiese encantado que pudieses acompañarme en este momento, pero estoy seguro de que estás orgulloso de ver todo lo que he conseguido alcanzar.

Tabla de contenidos

1. Introducción	1
1.1. Motivación del proyecto	1
1.2. Contexto del proyecto	2
1.3. Objetivos	3
1.4. Estructura del documento	4
2. Estado del Arte	5
2.1. Asistentes Virtuales	5
2.1.1. Historia de los Chatbots	5
2.1.2. Frameworks de Desarrollo de Chatbots	7
2.1.2.1. DialogFlow	7
2.1.2.2. Microsoft Bot Framework	8
2.1.2.3. IBM Watson	8
2.1.2.4. Amazon Lex	8
2.1.2.5. Rasa	8
2.1.3. Librerías	8
2.1.3.1. Chatterbot	9
2.1.3.2. Natural Language ToolKit	9
2.1.3.3. ChatbotAI	9
2.1.3.4. Tensorflow	9
2.1.4. Procesamiento de Lenguaje Natural	9
2.1.5. Ventajas e Inconvenientes de Utilizar Chatbots en un Entorno Profesional	10
2.1.5.1. Ventajas	10
2.1.5.2. Inconvenientes	10
2.1.6. Principales Aplicaciones de los Chatbots	11
2.1.6.1. Información	11
2.1.6.2. Salud	12
2.1.6.3. Viajes	12
2.1.6.4. Banca	13
2.1.6.5. Entretenimiento	14
2.1.6.6. Restauración	14
2.2. Bases de Datos	15
2.2.1. Historia de las bases de datos	15
2.2.2. Tipos de Bases de Datos	16
2.2.2.1. Bases de Datos Jerárquicas	16
2.2.2.2. Bases de Datos en Red	16
2.2.2.3. Bases de Datos Relacionales	17

2.2.2.4. Modelo de Base de Datos Orientado a Objetos	17
2.2.2.5. Modelo de Base de Datos Orientado a Documentos	18
2.3. API REST	19
2.3.1. Historia de las API REST	19
2.3.2. Características principales	20
2.3.3. Ventajas de desarrollo con API REST	21
3. Desarrollo	23
3.1. Arquitectura del Proyecto	23
3.2. Arquitectura General del Trabajo	25
3.3. Herramientas utilizadas	25
3.3.1. Base de Datos	25
3.3.1.1. Funcionamiento de MongoDB	26
3.3.2. Lenguaje de Programación Python	28
3.3.3. Arquitectura aplicación RestFul	28
3.3.3.1. Funcionamiento de Node.js	29
3.3.3.2. Ventajas de utilizar Node.js	29
3.3.3.3. Librerías utilizadas	29
3.3.4. Visual Studio Code	30
3.3.5. Git y Github	31
3.3.6. Swagger	33
3.3.7. Postman	35
3.3.8. Docker	37
3.3.8.1. ¿Qué es Docker?	37
3.3.8.2. Elementos básicos	37
3.3.8.3. Funcionamiento	37
3.3.8.4. Principales características	39
3.3.8.5. Tipos de despliegues	39
3.3.9. Rasa	40
3.3.9.1. Rasa NLU	40
3.3.9.2. Rasa Core	41
3.3.9.3. Rasa NLG	41
3.3.9.4. Procesado de un mensaje	41
3.3.9.5. Componentes de Rasa	42
3.3.9.5.1. Intents	43
3.3.9.5.2. Utterances	43
3.3.9.5.3. Slots	44
3.3.9.5.4. Dominio	44
3.3.9.5.5. Actions	45
3.3.9.5.6. Stories	46
3.3.9.5.7. Rules	46
3.3.9.5.8. Conectores	47
3.3.10 HTML	48
3.3.11 CSS3	48
4. Diseño	51
4.1. Diseño de las herramientas	51
4.1.1. Diseño de la base de datos	51
4.1.2. Estructura de la aplicación RESTful	53
4.1.3. Estructura de la aplicación Rasa	55

TABLA DE CONTENIDOS

4.2. Despliegue de las herramientas	56
4.2.1. Despliegue del asistente	56
4.2.2. Despliegue de la base de datos y API	58
5. Resultados	61
5.1. Manual de uso de la base de datos	61
5.1.1. Inserción en colección	61
5.1.2. Obtención de un documento	62
5.1.3. Modificación de un documento	62
5.1.4. Eliminación de un documento	63
5.2. Manual de uso de la API	63
5.2.1. Utilización del método GET	64
5.2.2. Utilización del método POST	64
5.2.3. Utilización del método Patch	66
5.2.4. Utilización del método Delete	66
5.3. Manual de uso del Chatbot	67
6. Conclusiones	71
Bibliografía	71

Índice de figuras

1.	Clasificación en el portal Zooniverse	2
2.	Empresas impulsoras del proyecto	3
3.	Funcionamiento del radar GRAVES	3
4.	Interfaz de Siri	6
5.	Telegram BotFather	7
6.	Interfaz de dialogflow	7
7.	Interfaz World Health Organization	11
8.	Interfaz Woebot	12
9.	Interfaz gráfica chatbot KLM	13
10.	Interfaz gráfica Eno	14
11.	Chatbot Burguer King	15
12.	Modelo jerárquico	16
13.	Modelo de red	16
14.	Modelo relacional	17
15.	Modelo orientado a objetos	17
16.	Modelo orientado a documentos	18
17.	Crecimiento API Web desde 2005	20
18.	Arquitectura del proyecto	24
19.	Arquitectura general	25
20.	Logo de MongoDB	25
21.	Operación con terminal de mongo	26
22.	Interfaz de Robo3T	27
23.	Logo de Python	28
24.	Logo de Node.js	29
25.	Visual Studio Code	31
26.	Logo de Git	31
27.	Inicialización de repositorio en Git	32
28.	Swagger Editor	33
29.	Opciones del servidor de Swagger	34
30.	Definición Swagger	34
31.	Interfaz web Swagger	35
32.	Generador de código en Postman	36
33.	Logo de Docker	37
34.	Interfaz gráfica de Docker	38
35.	Inicio de un contenedor docker	38
36.	Proceso de clasificación de un mensaje	42
37.	Definición del intent greet	43

38. Definición de un utterance	43
39. Definición de un slot	44
40. Definición de un dominio	44
41. Definición de un action	45
42. Definición de un story	46
43. Request de un mensaje a través del canal REST	47
44. Response de un mensaje a través del canal REST	48
45. Estructura de un fichero HTML5	48
46. Estructura de un fichero HTML con referencia al fichero CSS	49
47. Colecciones de la base de datos	52
48. Estructura de la aplicación RESTful	53
49. Estructura del asistente virtual	55
50. Creación del entorno virtual	56
51. Entrenamiento del asistente	57
52. Fichero dockerfile	58
53. Fichero docker-compose	59
54. Ejecución del docker-compose	60
55. Inserción en el terminal de MongoDB	62
56. Obtención de documento en el terminal de MongoDB	62
57. Modificación de documento en el terminal de MongoDB	63
58. Eliminación de un documento en el terminal de MongoDB	63
59. Obtención de un recurso por medio de Swagger	64
60. Inserción de un recurso por medio de Swagger	65
61. Inserción a través del script en Python	65
62. Eliminación de un recurso por medio de Swagger	66
63. Comienzo de una conversación con el chatbot	67
64. Muestra de opciones del chatbot	68
65. Muestra de sonido y primera pregunta	68

Capítulo 1

Introducción

1.1. Motivación del proyecto

El concepto del proyecto Sonidos del Cielo está sustentado sobre las bases de la ciencia ciudadana. Este campo tiene como finalidad la de acercar la ciencia a personas que se encuentran fuera de un enclave profesional como puede ser el estudio en un laboratorio o trabajo de campo. La motivación del proyecto es además hacer que la ciencia ciudadana sea más accesible a grupos sociales que en la actualidad no lo son, como pueden ser las personas con algún tipo de discapacidad visual o niños pequeños que aun no dominen la lectura y la escritura.

Es por eso que se ha pensado en el desarrollo de un asistente automatizado a través del cual se pueda interactuar ya sea de manera escrita o a través de la voz y que mediante la utilización de inteligencia artificial, sea capaz de seguir perfectamente una conversación cómo si se tratase de una persona y ayudar a los usuarios a realizar la clasificación de cuerpos celestes.

Este tipo de herramientas se conocen con el anglicismo **chatbot** que proviene de la unión de las palabras conversar *chat* con un robot *bot*

Según una encuesta realizada por el Instituto Nacional de Estadística (INE), en torno al 90 % de los de niños entre 10 y 12 años utilizan ordenadores y navegan por internet de manera habitual [1]. Hoy en día está muy extendido el uso de asistentes virtuales para la simplificación de tareas.

Nuestro objetivo es utilizar estas herramientas para hacer a los niños partícipes del proyecto y hacerlo de una manera entretenida, mediante técnicas de *gamificación*¹ que nos ayuden a clasificar los distintos tipos de meteoros mediante su sonido. Esto hace que tengamos que pensar en que no todos los usuarios son iguales, la interfaz de la aplicación no puede ser igual para un niño de 5 años que apenas sabe leer y escribir que para un adulto.

Una vez los usuarios hayan clasificado un meteoro un número determinado de veces,

¹Gamificación: uso de técnicas, elementos y dinámicas propias de los juegos para potenciar la motivación y mejorar el aprendizaje.

científicos analizarán y comprobarán la validez de las clasificaciones.

1.2. Contexto del proyecto

En el instante en que un meteorito entra en las zonas más densas de la atmósfera se produce una fricción que provoca una elevación exponencial de la temperatura que produce sublimación y ablación. Esto crea un trazo de electrones libres, el cual es posible detectar mediante una estación de radio.

El proyecto Sonidos del Cielo dispone de un anteproyecto en Zooniverse, que es una plataforma online de ciencia ciudadana. A través de él, cualquier usuario con acceso a internet puede clasificar una detección de meteorito a través de su sonido o de la curva de luz generada de la detección, tal y como se puede apreciar en la figura 1.

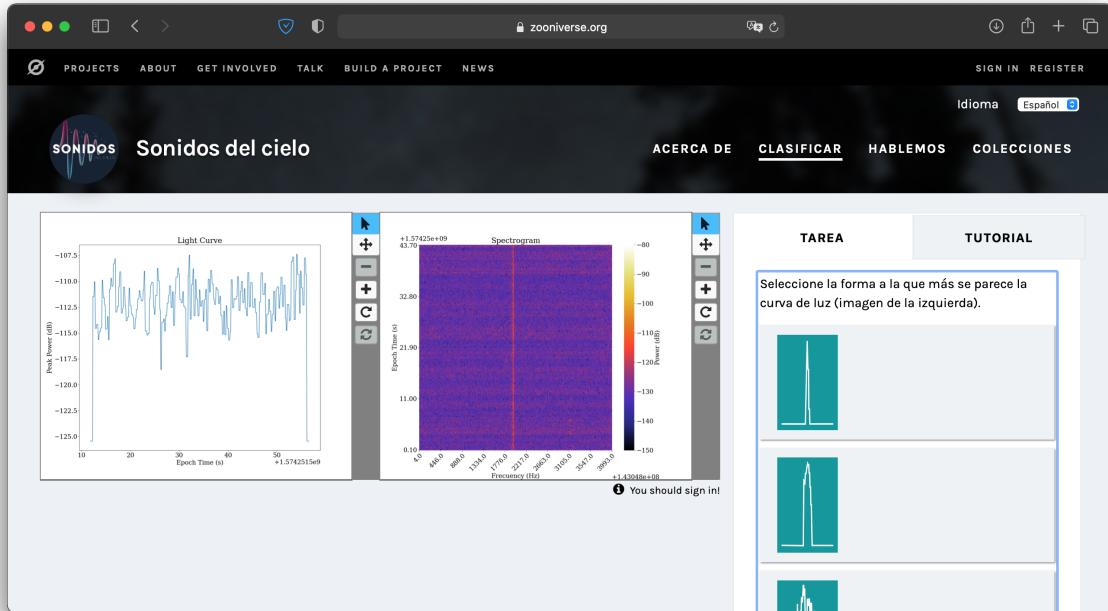


Figura 1: Clasificación en el portal Zooniverse

La idea del proyecto es realizar una herramienta paralela a este servicio a través de la cual los ciudadanos puedan contribuir con la clasificación de cuerpos celestes, y que sea utilizable tanto por adultos como por niños y gente con algún tipo de discapacidad visual.

El proyecto es responsabilidad del equipo de investigadores y estudiantes del Citizen Science Lab de la Universidad Politécnica de Madrid en colaboración con el Instituto Astrofísico de Canarias, el Grupo Docente de Astronomía Kepler y la Agrupación Astronómica Madrid Sur, representadas en la figura 2 y tiene como objetivo la detección y catalogación de estos meteoroides.

Introducción



Figura 2: Empresas impulsoras del proyecto

En la actualidad los meteoros son conocidos por el público general debido a las lluvias de estrellas, las más conocidas son las Perseídas que se producen de manera anual entre los meses de julio y agosto; y las Gemínidas que provienen de un asteroide conocido como Faetón y se pueden observar en el mes de diciembre. La ambición de este proyecto es despertar el interés en los ciudadanos y realizar una tarea de divulgación desde un punto de vista científico. Para poder detectar los meteoros se cuenta actualmente con una infraestructura de dos estaciones de radiodetección situadas en Fuenlabrada (Madrid) y Fregenal de la Sierra (Badajoz) construidas por miembros aficionados de la Agrupación Astronómica de Madrid Sur y del Grupo Docente de Astronomía Kepler [2], estas dos estaciones monitorizan la señal del radar GRAVES [3] ubicado en Dijón (Francia) que emite a 143.050 MHz mediante varias antenas de tipo Yagi, un receptor SDR [4], se puede ver el funcionamiento del radar en la figura3. A través del programa de código abierto Echoes [5] es posible generar una curva de luz, un espectrograma de esta detección y posteriormente generar a partir de ellas el sonido del meteoroide.

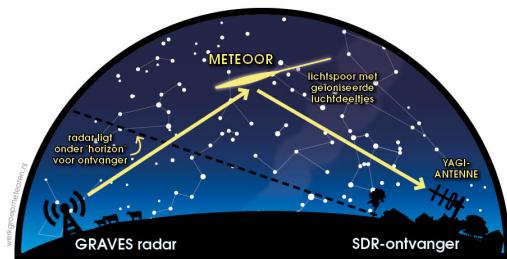


Figura 3: Funcionamiento del radar GRAVES

1.3. Objetivos

Puesto que el proyecto ha recibido financiación, ha sido necesario realizar la contratación de un desarrollador que se une al equipo a colaborar con el desarrollo de las distintas herramientas. Esto ha provocado que se hayan tenido que reajustar las tareas a realizar por los miembros del equipo y los requisitos del Trabajo de Fin de Grado, puesto que era necesario desarrollar las herramientas que fuesen la base del proyecto. A continuación se listan dichos objetivos:

- Diseño y desarrollo de la arquitectura de la base de datos del proyecto.
- Diseño y desarrollo de la API del proyecto.
- Integración de la API con la base de datos del proyecto.
- Creación de scripts para la inserción automática de nuevas detecciones de me-

teoros.

- Creación de una versión preliminar del chatbot para probar las funcionalidades de la API.

1.4. Estructura del documento

- Capítulo 2: Este capítulo es el estado del arte sobre las tecnologías en las que está basado el trabajo de fin de grado.
- Capítulo 3: En este capítulo se detallan las herramientas utilizadas y el desarrollo del proyecto.
- Capítulo 4: En este capítulo se especifican los resultados del trabajo y las posibles mejoras a implementar.
- Bibliografía: publicaciones utilizadas en el estudio y desarrollo del trabajo.
- Anexos (opcional)

Capítulo 2

Estado del Arte

En este capítulo se expresará cómo ha sido la evolución histórica de las principales herramientas a desarrollar durante el proyecto así como las diferentes alternativas más importantes dentro de cada campo.

2.1. Asistentes Virtuales

2.1.1. Historia de los Chatbots

Un chatbot es una aplicación de software capaz de mantener una conversación con un usuario dando una serie de respuestas automáticas, establecidas con anterioridad a diferentes entradas que pueda dar el usuario. Existen distintas teorías sobre el origen de los chatbots.

La primera de ellas, defiende que en la década de 1950, el matemático inglés Alan Turing investigó si una máquina sería capaz de imitar las respuestas de un humano mediante el análisis de una conversación de texto entre un humano y una máquina.

La segunda, más extendida, sitúa el origen en el año 1966 en el Instituto Tecnológico de Massachusetts (MIT), allí el profesor de informática Joseph Weizenbaum desarrolló en el laboratorio de inteligencia artificial el programa *Eliza*, el concepto es que actuara como si se tratase de un terapeuta. Funcionaba de la siguiente manera: examinaba palabras clave que tenía el enunciado del emisor para poder responder con una serie de oraciones que tenía previamente registradas.

En 1972, surgió el chatbot *Parry*, que simulaba ser una persona con esquizofrenia paranoides. A diferencia de Eliza disponía de una estrategia de comunicación cimentada en premisas y "respuestas emocionales" en base a las interacciones con los usuarios. [6]

Como detalle curioso, Eliza y Parry fueron puestos a conversar entre sí mediante la red ARPANET¹. Posteriormente fue desarrollado *Jabberwocky* por el programador inglés Rollo Carpenter, capaz de mantener una conversación mediante la voz. Aunque fue terminado en 1981 no fue hasta 1997 cuando fue publicado online. A partir del año 2006 han surgido una gran cantidad de chatbots entre los que cabe destacar:

¹ARPANET: red de ordenadores creada por el Departamento de Defensa de los Estados Unidos para conectar varias instituciones académicas y estatales.

IBM Watson, nombrado así por el primer director ejecutivo de IBM. En un principio fue desarrollado para responder preguntas y respuestas ideadas por humanos para el concurso Jeopardy. El concurso es el típico de preguntas y respuestas solo que las preguntas son formuladas mediante juegos de palabras y giros lingüísticos. Se presentó al concurso en 2011 por primera vez y fue capaz de ganar a dos especialistas. A partir de ese instante, ha pasado por varias adaptaciones utilizando procesamiento de lenguaje natural y machine learning² para procesar una gran cantidad de datos. En 2013, IBM anunció que Watson podía ser utilizado para la toma de decisiones en el tratamiento del cáncer de pulmón.

Tal vez el chatbot más conocido mundialmente sea el asistente virtual desarrollado por Apple, Siri. Utiliza consultas dadas mediante comandos de voz para ayudar al usuario de diversas formas, realizar tareas, recordatorios, búsquedas y modificar configuraciones del sistema tal y como se observa en la figura 4.

De este mismo estilo tenemos los chatbot Alexa creado por Amazon, Cortana desarrollado por Microsoft y Google Now desarrollado por Google.

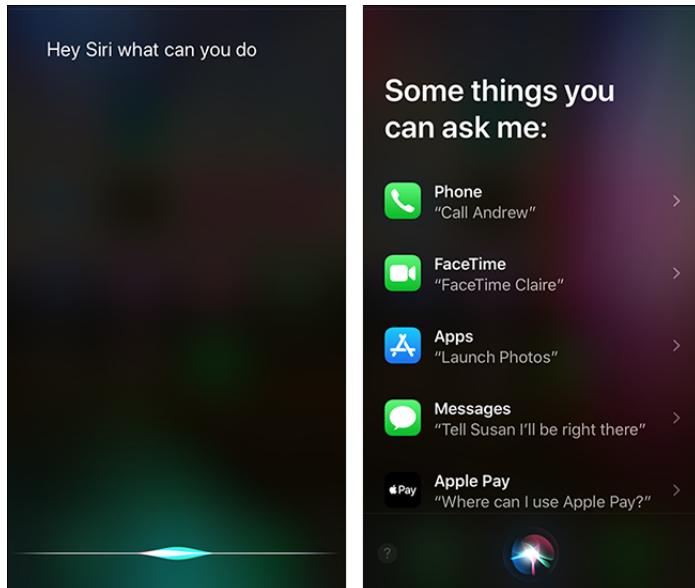


Figura 4: Interfaz de Siri

La utilización de estos asistentes ha crecido exponencialmente debido a la necesidad de dar servicios a los clientes, sin importar la hora ni el lugar ya que están siempre disponibles cuando el usuario lo necesite. Hoy en día se están implementando cada vez más en servicios de mensajería instantánea para la transmisión de noticias, las más destacables son Telegram (figura 5), Slack, Whatsapp, Discord, Skype, Kik y WeChat.

Algunas de estas aplicaciones han desarrollado servicios mediante los cuales puedes crear nuevos bots sin tener que programar una sola línea de código, como puede ser BotFather de Telegram. Así como tiendas con búsquedas y sistemas de valoraciones para aumentar el uso de estas aplicaciones.

²Machine Learning: disciplina dentro de la Inteligencia Artificial que crea sistemas y software capaces de aprender automáticamente

Estado del Arte

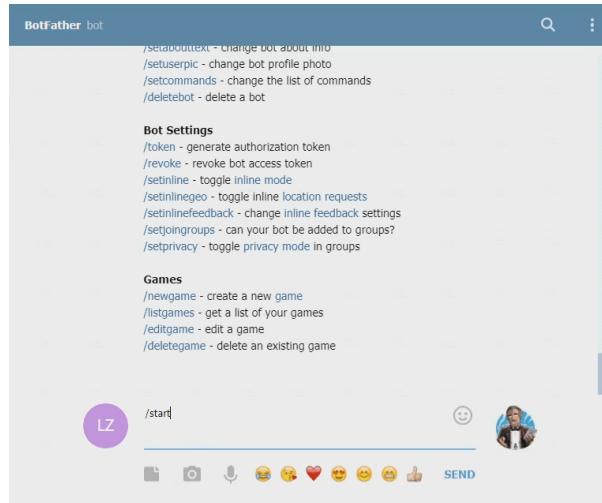


Figura 5: Telegram BotFather

2.1.2. Frameworks de Desarrollo de Chatbots

Un framework es un tipo de estructura con una serie de archivos y pautas que se utiliza para desarrollar proyectos con una estructura y metodología, es decir, algo así como una plantilla que simplifica la creación de una solución.

2.1.2.1. DialogFlow

Framework de desarrollo de chatbots creado en 2010 y mantenido por Google[7]. Es capaz de comprender el lenguaje natural y nos brinda herramientas para la fabricación de diálogos (figura 6) y la recreación de conversaciones. Destaca por la gran cantidad de interfaces de conversación en los que se puede desplegar (Google home, google assistant, wearables, teléfonos, coches). Tiene soporte para más de 14 idiomas y es capaz de resolver abreviaturas y funcionar con faltas de ortografía. Posee una interfaz muy intuitiva y permite crear chatbots en una cantidad pequeña de tiempo.

A screenshot of the Dialogflow interface. On the left, there's a sidebar with options: Developer-Panda, Intentos (selected), Entities, Training [beta], Integrations, Analytics [new], Fulfillment, Prebuilt Agents, Small Talk, and Docs. The main area shows an intent named 'showLibrary'. It has a 'SAVE' button and a '...' button. Below that, there's a 'User says' section with a search bar and a list of user expressions: 'tell me best chat sdk on github', 'search a machine learning library for me', 'any skill kit for alexa', 'what is amazon lambda service', 'i want to integrate messaging into my web app', 'tell me about amazon alexa', 'what are the benefits of amazon alexa', 'database-connection check', 'database', and 'what are the benefits of amazon alexa'. Each expression is preceded by a small green '99' icon.

Figura 6: Interfaz de dialogflow

2.1.2.2. Microsoft Bot Framework

Desarrollado por Microsoft, crea chatbots rápidamente a través de la herramienta Microsoft Bot Builder y los conecta con Azure Bot Service, lo que nos permite una rápida creación del bot ya que nos proporciona diversas plantillas para seleccionar cuando se está creando el bot y nos brinda todas las mejoras de la nube creada por Microsoft. Puede editarse directamente desde la página web usando el editor de Azure o algún IDE de desarrollo como Visual Studio o Visual Studio Code [8]. Posee su propio sistema de procesamiento natural del lenguaje llamado LUIS (*Language Understanding Intelligent Service*).

2.1.2.3. IBM Watson

Creado por IBM, es capaz de comprender y responder a las preguntas de los usuarios mediante lenguaje natural [9]. Watson está compuesto actualmente por un clúster de al menos 750 servidores *IBM Power 750*, con unos 16 TB de memoria RAM, lo que proporciona una potencia de cálculo bruto de unos **80 Petalops**, convirtiéndolo así en uno de los supercomputadores más potentes del mundo.

2.1.2.4. Amazon Lex

Creado y gestionado por Amazon, permite establecer comunicaciones con todos sus productos Eco y con su asistente virtual Alexa. Es una de las mejores herramientas en cuanto a conversión de voz a texto. Con esta herramienta se pueden crear bots con un lenguaje natural sofisticado. A diferencia de los anteriores, tiene una interfaz más intuitiva para principiantes aunque por el contrario, dispone de menos herramientas, no obstante, posee todas las necesarias en un chatbot.

2.1.2.5. Rasa

Por último cabe destacar Rasa, una framework *Open Source* de Machine Learning que nos permite crear conexiones entre las máquinas y el usuario. Posee herramientas para entender al usuario mediante el componente Rasa NLU (Natural Language Understanding), generar el diálogo con Rasa NLG (Natural Language Generation) y un motor (Rasa Core) capaz de definir cuál será la siguiente acción a tomar en función del mensaje transmitido por el usuario.

2.1.3. Librerías

Una librería es uno o varios archivos escritos en algún tipo de lenguaje de programación, que proporcionan varias funcionalidades. Al contrario que un framework, una librería no establece la estructura sobre cómo debe realizarse el desarrollo, sino que da funcionalidades genéricas que han sido programadas con anterioridad y evitan que haya que escribir código de más, aumentando la calidad del código y reduciendo el tiempo de desarrollo. Vamos a analizar las librerías existentes más importantes para el lenguaje de programación Python, debido a que se trata de un lenguaje simple, elegante ordenado, portable y que requiere de pocas líneas de código para generar programas completos.

2.1.3.1. Chatterbot

Se trata de una librería de Machine Learning basada en las conversaciones y diálogos tradicionales. Está diseñada de manera independiente al idioma, por lo que se puede entrenar en cualquier lenguaje, ya que está asegurado su correcto funcionamiento. El Machine Learning implementado en Chatterbot, permite, que mediante la interacción con los humanos se mejore su propio conocimiento. También es posible implementarlo junto a otras librerías para añadir funcionalidad como Text to Speech³

Es también compatible con librerías para aportar más funcionalidades como puede ser la conversión de texto a voz y así poder interactuar con el usuario sin necesidad de escribir.

2.1.3.2. Natural Language ToolKit

NTLK por sus siglas en inglés, se trata de una plataforma para crear chatbots con lenguaje humano. Dispone de funcionalidades muy interesantes desde el punto de vista del reconocimiento del lenguaje como la tokenización, derivación, etiquetado, análisis y razonamiento semántico.

2.1.3.3. ChatbotAI

Nos permite crear un chatbot con muy pocas líneas de código. Genera un controlador del chat y bots con inteligencia artificial que permiten una integración muy sencilla con API Rest. Esta inteligencia artificial nos genera múltiples características como aprender, memorizar, manejar conversaciones según el tema [10].

2.1.3.4. Tensorflow

Es una plataforma de código abierto gestionada por Google. Se trata de una biblioteca de aprendizaje automático con la que es posible construir y entrenar redes neuronales para detectar patrones y correlaciones en el aprendizaje y razonamiento de los humanos. Tiene diversas formas de interacción con el usuario (interfaces web, API Rest, interfaces gráficas de usuario, y líneas de comandos).

2.1.4. Procesamiento de Lenguaje Natural

Una de las partes más importantes de un asistente virtual es el Natural Language Processing (NLP), gracias a esta herramienta, el programa es capaz de entender la forma en la que nos expresamos los humanos. El procesamiento del lenguaje natural es una rama de la inteligencia artificial la cual se encarga de analizar y estudiar las interacciones en lenguaje natural entre los humanos y los ordenadores. Este campo posee varias funciones atrayentes.

- **Comprendión del lenguaje natural:** mediante esta función el programa consigue analizar lo que dice el usuario y comprender el mensaje. Para ello es estrictamente necesario que exista una base de datos del lenguaje en el que queremos

³Text to Speech: por sus siglas en inglés TTS, procedimiento de síntesis mediante el cual se transforma el texto escrito en voz.

que se exprese el chatbot de manera que pueda comprender la gramática, la semántica y el vocabulario del lenguaje que queremos que domine.

- **Recuperación de la información:** esta parte es la encargada de extraer las palabras claves del texto proporcionado.
- **Generación del lenguaje natural:** capacidad de dar respuestas a los mensajes enviados en lenguaje natural. Esta parte del programa se encarga de escoger las posibles respuestas y establecer un orden para que la respuesta proporcionada sea la más indicada y correcta gramaticalmente.
- **Reconocimiento y síntesis del lenguaje hablado:** hace que el chatbot sea capaz de transformar las respuestas que se le entregan mediante el habla y transformarlas a texto para su tratamiento. Una vez la respuesta del chatbot se genere, el módulo se encargará de convertirla a voz.
- **Detección de emociones y sentimientos:** hay ciertos programas que a través del estudio del lenguaje de las entradas que da el usuario son capaces de detectar los sentimientos y emociones.

2.1.5. Ventajas e Inconvenientes de Utilizar Chatbots en un Entorno Profesional

El afloramiento de los asistentes virtuales ha permitido la mejora y la automatización de procesos y servicios dentro del entorno profesional pero al no tener todas las habilidades que posee un humano, su utilización también posee ciertos inconvenientes.

2.1.5.1. Ventajas

- Disponibilidad: un chatbot una vez implementado puede funcionar 24 horas al día 7 días a la semana.
- Reducción de costes: puesto que es capaz de responder las preguntas más frecuentes y realizar las tareas más solicitadas no será necesario tener asistentes humanos, permitiendo una reducción de costes y su posible utilización en otros campos más necesarios.
- Auto-aprendizaje: posee algún tipo de inteligencia artificial. Es capaz de aprender nuevos comportamientos a través de las interacciones con los clientes, así como nuevos comportamientos que no hayan sido programados.
- Eficiencia: Un asistente humano tiene una capacidad limitada de atención en un rango de tiempo mientras que el asistente puede tener tantas instancias como sea necesario.

2.1.5.2. Inconvenientes

- Inteligencia: aunque tenga implementado inteligencia artificial, un asistente no es más inteligente que un humano y hay veces que no entiende el mensaje que el usuario le entrega y puede provocar un atasco del proceso. Esto puede llevar a que si el usuario pierde la paciencia, no se complete el proceso de compra.
- Tiempo: es necesario emplear tiempo en entrenar al asistente.

Estado del Arte

- Toma de decisiones: un chatbot no es capaz de decidir en base a lo que se haya enseñado con anterioridad.
- Coste: es cierto que la implementación de un chatbot permite reducir los costes de mano de obra aunque, si bien es cierto, en un primer momento el coste de implementarlo puede ser mayor, puesto que hay que programarlo de tal manera que se adapte al campo de la empresa, es decir, hay que especializarlo.

2.1.6. Principales Aplicaciones de los Chatbots

2.1.6.1. Información

Durante una crisis sanitaria como la que está viviendo el planeta con el virus SARS-CoV-2 la tendencia más normal es que noticias falsas o bulos se propaguen con una gran velocidad aunque cada día hay más gente que busca información y ayuda a través de recursos en línea.

Para todos ellos, la Organización Mundial de la Salud (OMS) [11] ha desarrollado un chatbot para que las personas se mantengan informadas mediante un chatbot implementado en la aplicación de mensajería instantánea WhatsApp. Ha sido diseñado de tal manera que permita dar información rápida y fiable sobre temas relacionados con el coronavirus. Se trata de un bot que funciona mediante comandos, utilizando emojis o palabras que nos indica el propio bot. Nos proporcionará información sobre varios temas como podemos observar en la figura 7. De momento el bot sólo está disponible en inglés aunque está en estudio el ampliarlo a español, árabe, chino, francés y ruso.

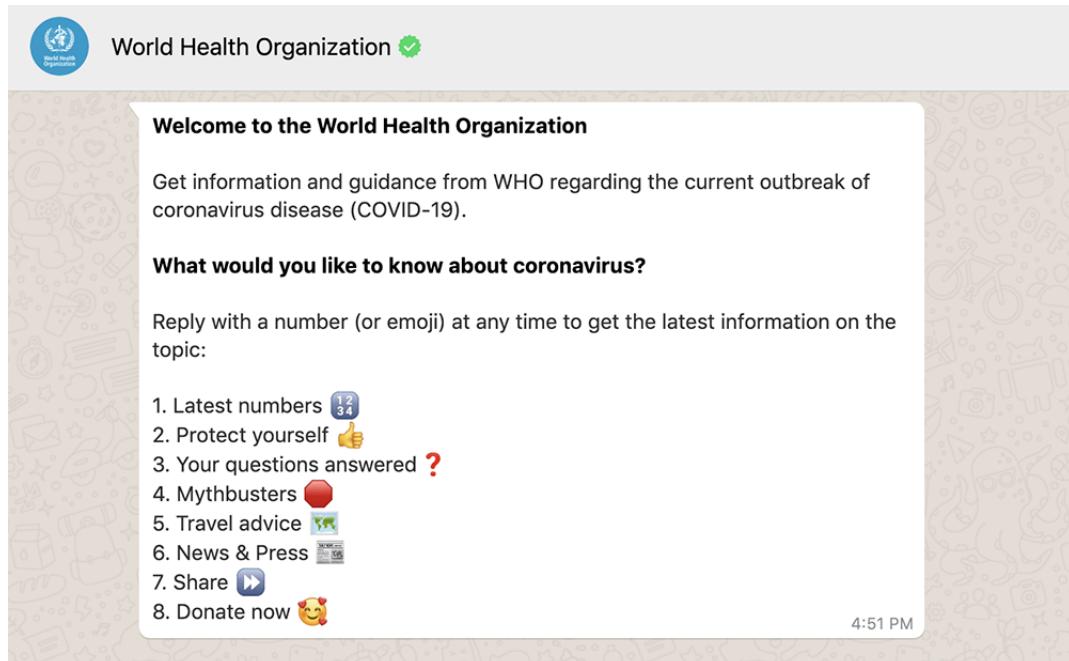


Figura 7: Interfaz World Health Organization

2.1.6.2. Salud

En consecuencia a lo expuesto en la sección anterior, mucha gente en contraindicación de lo que aconsejan los médicos, busca información sobre los síntomas que puedan padecer a través de internet. En esta línea se desarrolló Woebot [12], un chatbot que ayuda a reducir los síntomas de la depresión escuchando de forma activa los mensajes que envía el usuario y reaccionando a ellos con ánimo y con GIFs divertidos a mensajes positivos (figura 8). Combina años de investigación en el campo de la psicología con la inteligencia artificial y el procesamiento del lenguaje natural permitiendo que se evalúe el estado emocional del usuario y guiándolo para recibir la ayuda correcta de los profesionales necesarios en el momento adecuado.

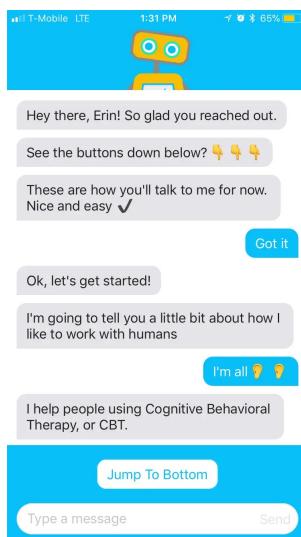


Figura 8: Interfaz Woebot

2.1.6.3. Viajes

Un bot de viaje puede ahorrar una cantidad significativa de tiempo a los usuarios (organización del viaje, lugares de interés a visitar) además mediante el uso de inteligencia artificial han hecho que varias empresas del sector turístico se lancen a la implementación de un chatbot que por un lado aumente su número de ventas y por otro reduzca la cantidad de personas necesarias para contestar a todos los usuarios con dudas.

A día de hoy, uno de los mejores chatbots de la industria es el desarrollado por KLM Royal Dutch Airlines [13]. Este chatbot funciona a través de Facebook Messenger, como se puede ver en la figura 9, pero con la particularidad que utiliza un complemento de casilla de verificación de Facebook Messenger a la hora de realizar el pago, de tal manera que es capaz de proporcionar información sobre la tarjeta de embarque, el check-in y actualizaciones del estado del vuelo a través de la aplicación de mensajería instantánea.

Antes de la implementación del chatbot, KLM tenía que contestar unas 15000 conversaciones a la semana a través de las redes sociales en una docena de idiomas distintos. Para poder tratar todas estas conversaciones se requiere una gran cantidad de personal y aún así existirán horarios en los cuales no se pueda conversar

Estado del Arte

con el usuario. Para solucionarlo comenzaron a estudiar varias formas para prestar una atención excelente en cualquier momento del día, rápida y personalizada. Como resultado, llegaron a la conclusión de implementar un chatbot. En el primer mes de funcionamiento, el volumen de mensajes en Facebook aumentó un 40 por ciento y envió 1.7 millones de mensajes a más de 500000 clientes.

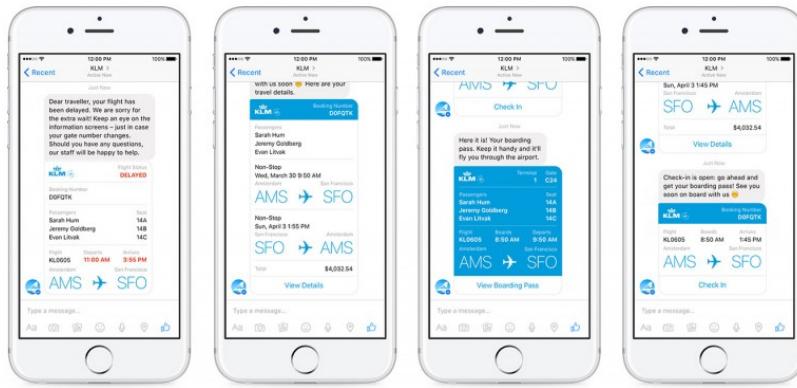


Figura 9: Interfaz gráfica chatbot KLM

2.1.6.4. Banca

En los últimos tiempos los bancos están intentando mejorar las relaciones y la comunicación con sus clientes. Este sector tiene fama de utilizar aplicaciones y sistemas arcaicos pero cada vez son mas las entidades que buscan actualizarse y llegar a más clientes.

El holding bancario Capital One [14] es uno de los más grandes de Estados Unidos y en 2018 lanzó su chatbot *ENO* (figura 10), se trata de un chatbot conversacional, es decir, es capaz de continuar una conversación y no funciona mediante comandos. Los clientes pueden mandar un sms a Eno en cualquier momento a través de la página web, la aplicación móvil del banco para comprobar transacciones recientes, el saldo o incluso para informar sobre algún problema con la tarjeta de crédito. Tiene la capacidad de incluso avisar cuando se cobra de más por una factura.



Figura 10: Interfaz gráfica Eno

En España existen algunas alternativas como por ejemplo, la aplicación imaginBank implementado por La Caixa a través de Facebook Messenger.

2.1.6.5. Entretenimiento

Los chatbots también se pueden emplear para realizar planes y lugares de ocio. Un ejemplo de esto es el asistente Mahoudrid creado por la empresa Mahou en colaboración con Ontwice basándose en la guía de ocio de nombre homónimo.

Este asistente es capaz de recomendar locales, bares, tapas y planes. Es un bot conversacional implementado a través de Facebook Messenger mediante inteligencia artificial y te recomienda el plan perfecto en base a lo que deseas tomar y tu estado de ánimo.

2.1.6.6. Restauración

En la actualidad existen múltiples chatbots que ayudan al usuario a consultar recetas o consejos de dietética, sin embargo, el verdadero potencial de estos es el de poder recomendar restaurantes cercanos a la ubicación del usuario e incluso realizar un pedido simplemente escribiendo en un chat de mensajería instantánea y que lo tengas en tu casa en el menor tiempo posible sin tener que realizar ninguna llamada telefónica.

Un ejemplo es el chatbot desarrollado por Burguer King [15] e implementado en WhatsApp que se muestra en la figura 11.



Figura 11: Chatbot Burguer King

2.2. Bases de Datos

2.2.1. Historia de las bases de datos

Hay que remontarse un poco atrás en el tiempo, concretamente hasta 1880 cuando Herman Hollerith desarrolló una máquina para procesar los datos más rápido de lo que los humanos podían hacerlo. Desarrolló unas tarjetas perforadas mediante las cuales podía realizar un censo de la población completa de Estados Unidos en solo dos años, algo que en esa época era poco más que una quimera. Aun así, la primera vez que se tiene conocimiento del uso del término bases de datos fue en el año 1963 en un simposio celebrado en California y se refiere a ella como un conjunto de información relacionada y que tiene cierta estructura.

Los primeros modelos que fueron desarrollados fueron bases de datos jerárquicas y en red, pero rápidamente se vio que estaban muy limitadas técnicamente y eran demasiado simples. Posteriormente, IBM revolucionó el sector en la década de los 70 creando el modelo relacional de base de datos, mucho más potente y muy bien adoptada por el mundo laboral. En los 2000 empezaron a aparecer proyectos de código libre que supusieron una competencia en un sector liderado siempre por sistemas privados (IBM y Oracle), entre los más usados destacan MySQL y PostgreSQL. La tendencia a la utilización de sistemas NoSQL contribuyó al detrimento de los sistemas de bases de datos de las grandes empresas. [16]

2.2.2. Tipos de Bases de Datos

2.2.2.1. Bases de Datos Jerárquicas

Se trata del modelo más antiguo, se ha visto superado por el modelo relacional. El lenguaje de marcado XML utiliza este tipo de modelo para almacenar los datos. El sistema más conocido es el IMS/DB creado por IBM.

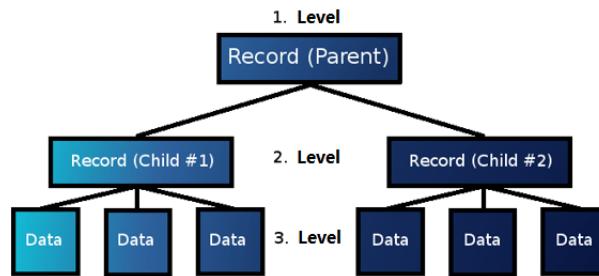


Figura 12: Modelo jerárquico

En este tipo de modelo, cada registro sólo posee un precedente excepto la raíz, formando una estructura arbolada como la que se puede ver en la figura 12. En este tipo de modelo, los hijos sólo pueden tener un padre pero un padre puede tener múltiples hijos. Dada esta relación estricta, los niveles que no tengan una relación directa, no pueden interactuar entre ellos y realizar una conexión entre varios árboles tampoco es sencillo. Por este motivo, estas estructuras son consideradas inflexibles aunque por otro lado son muy fáciles de comprender.

2.2.2.2. Bases de Datos en Red

Se desarrolló de manera concurrente al relacional pero con el tiempo se vería superado. No revelan relaciones padre-hijo estrictas sino que cada registro puede tener múltiples precedentes, lo que genera una estructura en red y de ahí su nombre.

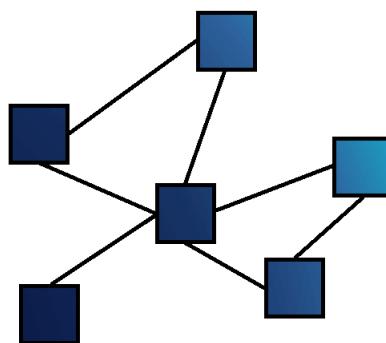


Figura 13: Modelo de red

Al registro central se tiene acceso desde los otros cinco y desde él, puede accederse a otros cinco registros. En este modelo pueden definirse dependencias, de forma que si accedes al registro superior necesitas pasar por el central para poder alcanzar el registro de la derecha, como puede verse en la figura 13. Hoy en día este modelo

Estado del Arte

de bases de datos se suele utilizar en algunos supercomputadores. Algunos de los modelos más conocidos son el UDS de Siemens y el DMS de Sperry Univac.

2.2.2.3. Bases de Datos Relacionales

Es el modelo más utilizado en la actualidad y es considerado como el estándar de la industria. Normalmente utiliza el lenguaje SQL y se trata de un modelo basado en tablas. Para definir las relaciones se utiliza álgebra relacional y gracias a esta se puede hallar la información de las diferentes relaciones como se puede ver en la figura 14.

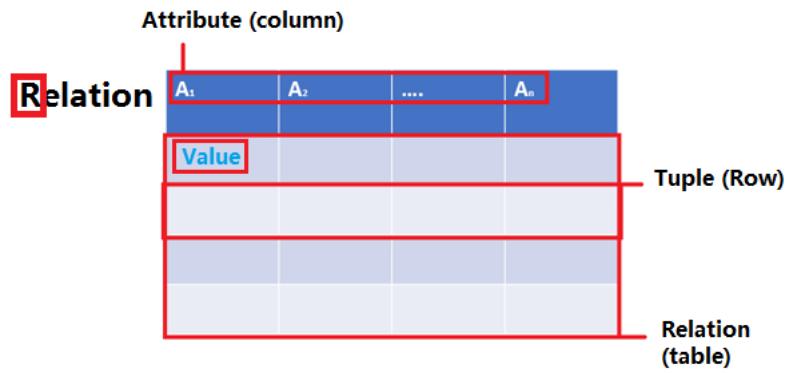


Figura 14: Modelo relacional

El modelo relacional funciona mediante tablas independientes que determinan la organización de los datos y sus conexiones. Para poder identificar sin problema un registro es necesario establecer una **clave primaria**, que habitualmente se trata del primer atributo y no se puede modificar.

2.2.2.4. Modelo de Base de Datos Orientado a Objetos

Surgen a finales de 1980 y a día de hoy han tenido una aplicación muy minoritaria. Este tipo de bases de datos suelen utilizarse en plataformas de lenguajes Java y .NET. La más conocida es db4o y destaca por su bajo consumo de memoria. Utilizan un lenguaje OQL, muy parecido a SQL.

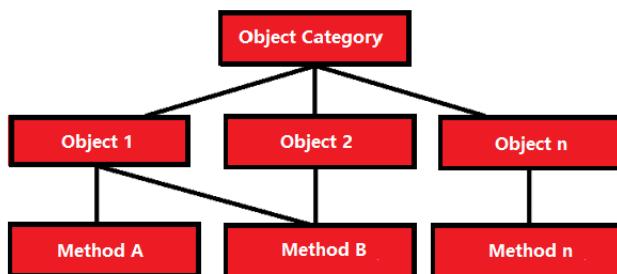


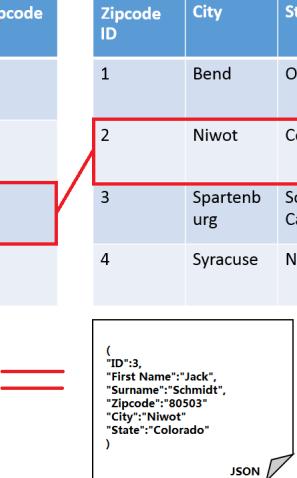
Figura 15: Modelo orientado a objetos

En este modelo los datos se almacenan en objetos además de las funciones y los atributos que los definen (figura 15). Los objetos pueden ser complejos y estar formados

por múltiples tipos de datos; se identifican por un identificador de objeto único y se agrupan en clases al igual que en lenguaje de programación orientado a objetos, creando una jerarquía de clases.

2.2.2.5. Modelo de Base de Datos Orientado a Documentos

En este modelo, la unidad mínima de almacenamiento de datos es el documento pero no se debe confundir con los documentos que se generan mediante un procesador de texto. En estos documentos los datos se almacenan como **pares claves-valor** tal y como se muestra en la figura 16. Como no tienen una definición, los documentos que forman este tipo de bases de datos son muy diferentes entre sí. Cada documento es una unidad cerrada entre sí y establecer relaciones entre documentos no es una tarea sencilla, aunque por otro lado, en este tipo de bases de datos no es necesario establecer relaciones.



Key	First Name	Surname	Zipcode ID	Zipcode ID	City	State	Zipcode
1	Walker	McClain	1	1	Bend	Oregon	97701
2	Blain	Muller	2	2	Niwot	Colorado	80503
3	Jack	Schmidt	2	3	Spartenburg	South Carolina	29301
4	Greg	Cohn	4	4	Syracuse	New York	13201


```
{
  "ID":3,
  "First Name":"Jack",
  "Surname":"Schmidt",
  "Zipcode":80503
  "City":"Niwot"
  "State":Colorado
}
```

JSON

Figura 16: Modelo orientado a documentos

La idea fundamental de este tipo de modelos es que los datos que tienen relación se guarden en un mismo documento, esto provoca que el número de consultas a la base de datos sea mucho menor que en una base de datos relacional. Son muy útiles para aplicaciones web debido a que pueden almacenarse formularios HTML. Con el avance de la web 2.0 estas bases de datos aumentaron exponencialmente su popularidad y son utilizadas hoy en día por plataformas como Google, Facebook, Twitter, Amazon, etc.

2.3. API REST

2.3.1. Historia de las API REST

En la década de los 90, la gran mayoría de desarrolladores trabajaban con el protocolo *SOAP* (abreviatura en inglés de Simple Object Access Protocol). Este protocolo consiste en el envío de datos mediante documentos escritos en lenguaje XML de forma manual, con una llamada RCP en el elemento *body* y además era necesario especificar el endpoint. Esto lo convertía en una herramienta compleja de utilizar, construir y depurar. En ese momento no existía ninguna directriz o estándar sobre como diseñar y construir una API, sólo se buscaba la flexibilidad en el uso de la API.

En el año 2000, Roy Fielding desarrolló en su tesis doctoral la manera mediante la cual dos servidores cualesquiera del mundo pudiesen establecer una comunicación entre sí. Para ello creó unos principios básicos, propiedades y restricciones que hoy en día se conocen como *REST* (es la abreviatura de Representational State Transfer).

La empresa Salesforce fue la primera empresa en vender una API como una porción de un paquete de software en el año 2000, pero fueron muy pocos los desarrolladores que consiguieron aprovechar la API XML que venía además con un manual de uso de 400 páginas.

Ebay viendo lo que Salesforce trataba de crear, decide construir una API accesible y de sencilla utilización y brinda acceso a sus socios además de equiparla con una documentación online exhaustiva. De esta manera, Ebay consiguió expandir su negocio fuera de su web Ebay.com a cualquier página capaz de acceder a su API. Debido a que la implementación seguía el estándar RESTful, muchos sitios web se apresuraron a aprovechar la oportunidad de expansión y de esta forma ampliar la oferta de productos a sus clientes. Amazon siguió rápidamente los pasos de Ebay, lo que propició que las plataformas web empezaran a cuantificar el valor de su código y no sólo el valor del producto que se pretendía vender.

En 2004, Flickr lanza su propia API REST coincidiendo con el auge de las redes sociales, convirtiéndose rápidamente en una plataforma líder en la utilización de imágenes y permitiendo que se pudiese compartir e incrustar imágenes de forma sencilla en sitios web y feeds de redes sociales. Facebook se lanzó ese mismo año y Twitter en 2006. En un comienzo, ambas plataformas tenían sus APIs de forma interna lo que provocó que muchos desarrolladores utilizaron técnicas de scraping⁴ y crearan sus propias API. Al ser un cúmulo de métodos sin seguir un mismo diseño, se les conocía como API Frankenstein. Esto provocó que ambos sitios lanzaran sus propias API oficiales en 2006.

En 2006, Amazon Web Services (AWS) comenzó a lanzar la nube. Esto permitía a los desarrolladores acceder a una cantidad enorme de datos a través de la API REST AWS. La implementación de APIs se ha multiplicado por más de 50 desde el año 2006 tal y como se puede ver en la figura 17.

⁴Scraping: serie de técnicas empleadas para la obtención de datos en sitios web

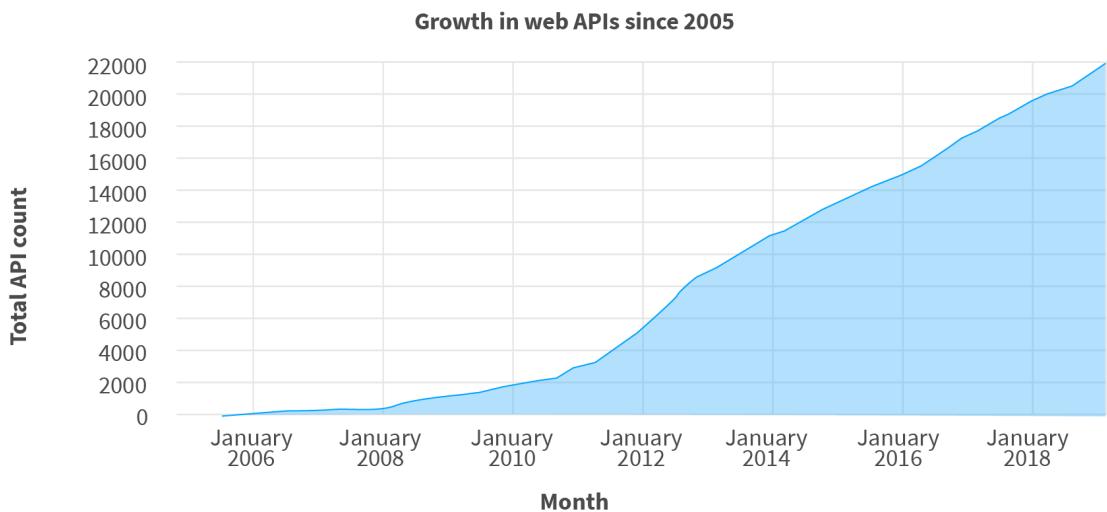


Figura 17: Crecimiento API Web desde 2005

Las APIs son las columnas de cualquier modelo de negocio y muchas veces se convierten directamente en el producto a vender por empresas de desarrollo de software. Gracias a la evolución de las API ya no es necesario tener equipos grandes de trabajo. Cualquier desarrollador capaz de obtener una clave e interprete la documentación puede integrar funcionalidades de ese servicio en su software.

2.3.2. Características principales

Una API REST siempre utiliza los métodos implementados dentro del protocolo HTTP. Son conocidos como verbos HTTP aunque también existe algún sustantivo. Cada uno de ellos tiene una llamada diferente aunque existen ciertas características compartidas.

Los métodos de petición existentes son:

- GET: solicita la representación del recurso especificado. Sólo recupera datos.
- HEAD: tiene el mismo comportamiento que la petición GET aunque sin recuperar el cuerpo de la respuesta.
- POST: con este método se envía información a un recurso específico.
- PUT: reemplaza todas las representaciones del recurso de destino con la información introducida en la petición.
- DELETE: elimina un recurso específico.
- CONNECT: establece una conexión hasta el servidor identificado en el recurso.
- OPTIONS: en él se describen las opciones de comunicación para el recurso de destino.
- TRACE: realiza una prueba de bucle de loop-back a lo largo de la ruta.
- PATCH: se usa para realizar modificaciones parciales a un recurso.

Dentro de estas funciones las más utilizadas son POST, PUT, GET y DELETE. Cualquier API tiene que tener una serie de características comunes que se detallan a continuación:

- Sin estado: cada petición HTTP contiene por sí sola la información necesaria para ejecutarse, lo que permite que ni el servidor ni el cliente tengan que recordar ningún estado previo.
- Objetos manipulados con URI: una URI es un identificador único de cada recurso. Esto hace que sea más sencillo acceder a la información para modificarla o eliminarla.
- Interfaz uniforme: esto significa que siempre se utilizan los *verbos HTTP* y siempre se utilizan las URIs de los recursos. Como resultado siempre se obtendrá una respuesta HTTP con un código de estado y un body.
- Sistema de capas: tiene una estructura jerárquica entre todos sus componentes y cada capa tiene una funcionalidad propia dentro del sistema REST.
- Uso de hipermedios: los hipermedios permiten al usuario navegar entre objetos mediante enlaces HTML. En un API es la capacidad de una interfaz de dar al cliente y al servidor los enlaces necesarios para realizar acciones sobre datos.

2.3.3. Ventajas de desarrollo con API REST

La implementación de un API REST tiene múltiples ventajas, la más destacable de ellas es que se trata de un medio independiente a lenguajes de programación o plataformas de desarrollo, debido a que puede adaptarse a cualquier medio en el que se esté desarrollando, lo que proporciona una gran libertad. El único requisito indispensable es que se realicen las peticiones y las respuestas en XML o JSON.

Al tratarse de un protocolo que separa totalmente al cliente y al servidor nos brinda una manera sencilla de exportar los proyectos a otros entornos y proporciona una gran escalabilidad. Además la separación nos permite tener el frontend y el backend desplegado en distintos servidores, dando a los desarrollos una gran flexibilidad a la hora de trabajar.

Capítulo 3

Desarrollo

Para el desarrollo de este Trabajo de Fin de Grado es necesario utilizar múltiples tecnologías y herramientas. En este capítulo se explicará brevemente el estado actual de los chatbots, las distintas herramientas para desarrollarlos, así como los lenguajes de programación utilizados, qué es el procesamiento natural del lenguaje.

Lo primero que hay que tener en cuenta es que se han desarrollado tres servicios desde cero. Por un lado se ha creado una base de datos no relacional, luego una Api Rest para manejar la base de datos y por último un chatbot que será el que consuma la API y a su vez la base de datos.

3.1. Arquitectura del Proyecto

Este Trabajo de Fin de Grado está incluido dentro de una estructura que está formada por varios proyectos que se explicarán a continuación, así como de las tecnologías seleccionadas para desarrollar el proyecto. En el ordenador instalado en la estación de Fuenlabrada está colocado el programa *Echoes Watcher* [17]. Este programa lee el archivo de configuración generado por Echoes para obtener datos de interés (peak, directorio, nombre estación, etc) y según se van generando ficheros de detecciones, el programa mediante un programa de monitorización, lee esos ficheros y en caso de encontrar alguna detección, se envían los datos a través de un Broker MQTT [18] al equipo ubicado en el Centro de Supercomputación y Visualización de Madrid (Cesvima).

Todos los días se procede a hacer un backup de los datos sin filtrar a otro computador del proyecto ubicado en la Escuela Técnica Superior de Ingeniería y Diseño Industrial (ESTIDI).

En el Cesvima se encuentra ubicado el programa StreamGenerator [6]. Este programa está suscrito al broker y puede recibir datos de configuración de la estación y datos de las detecciones. Cuando recibe datos de configuración de las detecciones genera un fichero de liquidsoap [19] si no existe o si ha cambiado la configuración de la estación. Este fichero liquidsoap genera una lista de reproducción que está ubicada en el Instituto Astrofísico de Canarias y se trata de un bucle infinito de ruido (basado en el archivo de configuración) y a medida que le llegan detecciones de una estación se altera dicha lista de reproducción para añadir los sonidos generados con la detección que recibe del broker, y una vez que el sonido es reproducido, se borra.

3.1. Arquitectura del Proyecto

Por cada estación existente hay una lista de reproducción, lo que significa que existe un proceso liquidsoap ejecutándose por cada estación.

Todos los datos existentes en la estación de Fuenlabrada pasan a través de un Pipeline gestionado mediante **Apache Airflow**. Este programa es un orquestador de servicios, es decir, es utilizado para automatizar tareas sistemáticas dividiendo estas mismas en subtareas. Los casos de uso más típicos son la automatización de carga de datos, acciones periódicas de mantenimiento y tareas de administración. Para ello permite el uso de herramientas como *cron* y ejecutarlos bajo demanda. Mediante esta herramienta se envían los datos a un servidor existente en el Departamento de Arquitectura y Tecnología de Sistemas Informáticos. En este servidor existe una base de datos relacional en MySQL donde se almacenan los datos en bruto del programa Echoes.

Para entender mejor toda la estructura del proyecto, se incluye el siguiente diagrama explicativo 18.

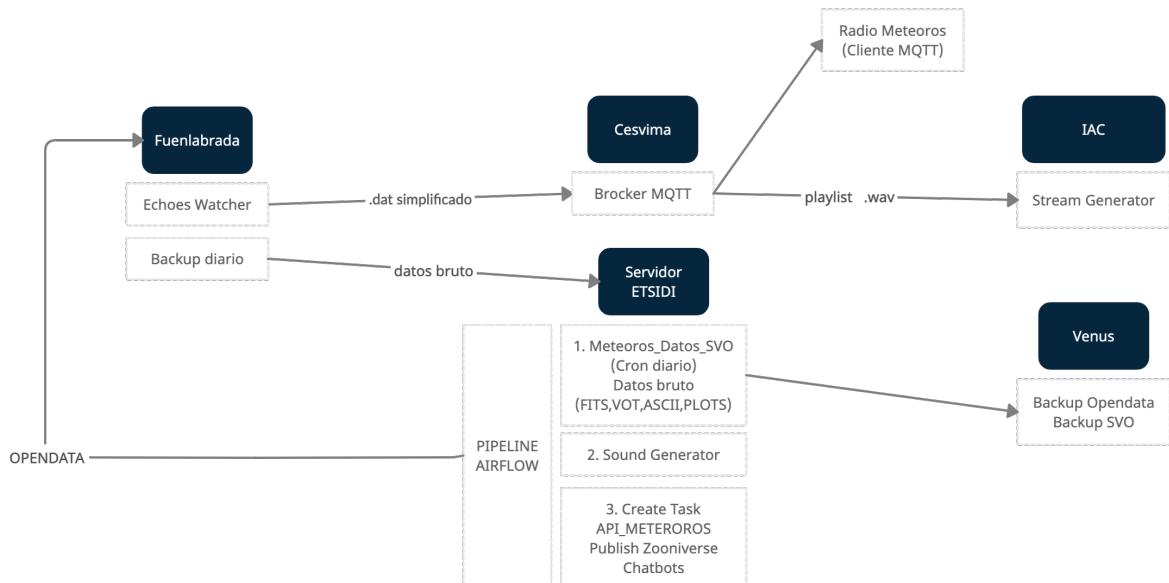


Figura 18: Arquitectura del proyecto

3.2. Arquitectura General del Trabajo

A continuación, se mostrará la arquitectura en la que se rige este Trabajo de Fin de Grado.

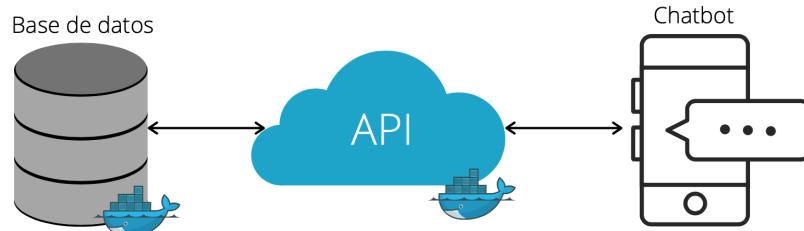


Figura 19: Arquitectura general

Como se puede observar en la figura 19, la API RestFul es el software encargado de gestionar e interactuar con la base de datos, sirviendo los datos a los usuarios. Para que el proyecto sea lo más escalable posible se ha decidido separar los servicios de forma que estén débilmente acopladas.

3.3. Herramientas utilizadas

3.3.1. Base de Datos

Para el proyecto Sonidos del Cielo hemos elegido MongoDB (figura 20). En gran parte debido a que es una herramienta perfecta para entornos con bajos recursos de computación. No es necesario pagar ningún tipo de licencia, posee una gran comunidad y muy buena documentación, además de muy escalable. La ventaja principal de MongoDB es que los documentos no tienen que seguir una estructura definida, es decir, se puede trabajar con documentos independientes, modificar el contenido de uno de forma individual y no afectar a la estructura.



Figura 20: Logo de MongoDB

La característica principal de MongoDB es la velocidad, dado que está escrito en C++ tiene la capacidad de aprovechar los recursos del sistema de una manera eficiente, es decir, tiene un gran balance entre funcionalidad y rendimiento gracias a su sistema de consulta de contenidos. Las características principales de la plataforma son:

- Consultas ad-hoc: se pueden realizar todo tipo de consultas. Podemos buscar por campos como si de una base de datos relacional se tratase pero además podemos buscar por rango, mediante expresiones regulares.
- Indexación: el concepto es muy parecido al utilizado en las bases de datos relacionales, sin embargo, en MongoDB se puede indexar cualquier campo documentado e incluir múltiples índices secundarios.

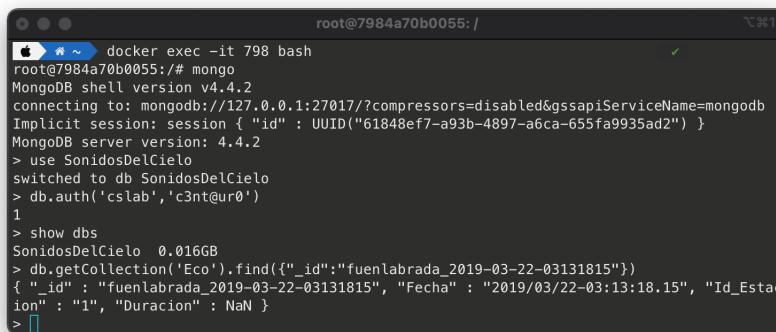
- Balanceo de carga: MongoDB posee la capacidad de ejecutarse de manera simultánea en múltiples servidores, dando un servicio de balanceo de carga o de replicación de datos, de manera que es posible mantener el funcionamiento del sistema en caso de un fallo de hardware.
- Almacenamiento de archivos: puede ser utilizado también como almacenamiento de archivos, esta función, conocida como GridFS está incluida en la distribución oficial y permite manipular archivos y su contenido.
- Ejecución de JavaScript en el lado del servidor: es posible realizar consultas utilizando código escrito en JavaScript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutadas.

3.3.1.1. Funcionamiento de MongoDB

MongoDB está orientado a documentos. Esto significa que en lugar de guardar los datos en registros se guardan en documentos BSON, que no es otra cosa que un fichero JSON binario. Esto es una de las grandes diferencias respecto a las bases de datos relacionales. MongoDb se compone de colecciones que a su vez están formado por documentos.

Si hiciésemos una comparación con los elementos de una base de datos relacional, se trataría de una tabla. Dentro de cada colección existen múltiples documentos que están compuestos de pares Clave-Valor. La principal diferencia respecto a una base de datos relacional es que no es necesario que los documentos de una misma colección tengan el mismo formato ni los mismos campos.

Una vez creada la base de datos y establecidos los usuarios para restringir el acceso a la misma, podemos realizar operaciones de consulta, inserción, eliminación y actualización de documentos, colecciones y bases de datos a través de la consola incluida dentro del paquete de instalación de MongoDB tal y como se indica en la figura 21.



```

root@7984a70b0055:/ docker exec -it 798 bash
root@7984a70b0055:/# mongo
MongoDB shell version v4.4.2
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("61848ef7-a93b-4897-a6ca-655fa9935ad2") }
MongoDB server version: 4.4.2
> use SonidosDelCielo
switched to db SonidosDelCielo
> db.auth('cslab','c3nt@ur0')
1
> show dbs
SonidosDelCielo 0.016GB
> db.getCollection('Eco').find({"_id": "fuenlabrada_2019-03-22-03131815"})
{ "_id" : "fuenlabrada_2019-03-22-03131815", "Fecha" : "2019/03/22-03:13:18.15", "Id_Estacion" : "1", "Duracion" : NaN }
>

```

Figura 21: Operación con terminal de mongo

También existen herramientas gráficas para modificar cualquier parámetro de una base de datos en Mongo como puede ser MongoDB Compass o Robo 3T. Estas herramientas proporcionan una vía más sencilla y visual para trabajar con las bases de datos además de no tener que aprender o buscar dentro de la documentación los comandos necesarios para manejar una interfaz de línea de comandos propietaria.

Desarrollo

Aunque también nos permite ejecutar comandos como si se tratase del terminal como se puede observar en la figura 22.

The screenshot shows the Robo 3T application window titled "Robo 3T - 1.4". On the left, there is a sidebar with a tree view of database connections and collections. Under "New Connection (1)", there is a single connection named "SonidosDelCielo" which contains five collections: "Curva Luz", "Eco", "Espectrograma", "Estacion", and "Usuario". The "Functions" and "Users" sections are also present. The main pane displays the results of a MongoDB query: "db.getCollection('Espectrograma').find({})". The results are shown in a table with columns "Key", "Value", and "Type". The table has 14 rows, each representing a document from the "Espectrograma" collection. The first row is expanded to show its internal structure, which includes fields like "_id", "Id_Estacion", "Votable", "Imagen", and "Csv". The "Value" column for the first row shows the document's content, and the "Type" column indicates the type of each field (Object, String, or Array).

Key	Type
(1) fuenlabrada_2019-03-22-00321615	Object
_id	String
Id_Estacion	String
Votable	Object
Imagen	Object
Csv	Object
(2) fuenlabrada_2019-03-22-00412715	Object
(3) fuenlabrada_2019-03-22-01090915	Object
(4) fuenlabrada_2019-03-22-01422015	Object
(5) fuenlabrada_2019-03-22-02021115	Object
_id	String
Id_Estacion	String
Votable	Object
Imagen	Object
Csv	Object
(6) fuenlabrada_2019-03-22-03131815	Object
(7) fuenlabrada_2019-03-22-03171215	Object
(8) fuenlabrada_2019-03-22-04043515	Object
(9) fuenlabrada_2019-03-22-04093015	Object
(10) fuenlabrada_2019-03-22-04144215	Object
(11) fuenlabrada_2019-03-22-04335415	Object
(12) fuenlabrada_2019-03-22-04520915	Object
(13) fuenlabrada_2019-03-22-05154015	Object
(14) fuenlabrada_2019-03-22-05193615	Object

Figura 22: Interfaz de Robo3T

3.3.2. Lenguaje de Programación Python

Tanto para la carga de los documentos en la base de datos como para el desarrollo del chatbot ha sido necesario el uso del lenguaje de programación Python (figura 23). Python es un lenguaje de programación creado a finales de los años 80 por Guido van Rossum en el Centro para las Matemáticas y la Informática de Países Bajos. Este lenguaje posee una gran flexibilidad que produce código ordenado, limpio y fácilmente legible. Sus principales características pueden resumirse en las siguientes[20]:



Figura 23: Logo de Python

- Gratis: No es necesario pagar licencia para utilizarlo
- Interpretado: se ejecuta sin tener que pasar por un compilador y los errores son detectados en tiempo de ejecución.
- Multiplataforma: está disponible para Windows, Mac y Linux.
- Tipado dinámico: las variables declaradas se comprueban durante el tiempo de ejecución.
- Multiparadigma: soporta programación funcional, programación imperativa, declarativo, modular, dirigido a eventos, programación asíncrona, lógico, estructurado, con restricciones y programación orientada a objetos.

Es un lenguaje que tiene una curva de aprendizaje muy rápida, además cuenta con una gran comunidad, lo que hace que existan librerías y módulos especializados desarrollados como *pandas* (tratamiento de datos) *librosa* (para procesamiento de audio), *sklearn* (machine-learning) o frameworks como *Django* (para desarrollo web) o en nuestro caso, **RASA** para desarrollar un chatbot.

3.3.3. Arquitectura aplicación RestFul

La API RESTful no es más que el software que hace de intermediario para enviar información de la base de datos al cliente o para insertar datos desde el cliente a la base de datos [21]. El cliente es aquel que consume la información, en nuestro caso, el chatbot. El servidor se ha desarrollado en Node.js (figura 24). Fue diseñado para la construcción de aplicaciones escalables, siendo un punto muy a favor para nuestra elección.



Figura 24: Logo de Node.js

3.3.3.1. Funcionamiento de Node.js

Node.js es un entorno de ejecución de JavaScript [22] orientado a eventos asíncronos. Comparándolo con servicios web clásicos donde cada nueva conexión genera un nuevo subprocesso (ocupando memoria RAM del sistema) y normalmente ocupando toda la memoria RAM disponible. Node.js trabaja en un subprocesso únicamente, utilizando el modelo entrada y salida sin bloqueo de la salida, esto permite soportar una cantidad enorme de conexiones simultáneas mantenidas dentro del bucle de eventos. El servidor consta de un subprocesso que trata un evento tras otro. Cuando se produce una nueva solicitud se genera un evento. El servidor comienza a procesarlo y cuando se produce una operación de bloqueo de la entrada/salida, no espera a que se termine sino que se crea una *función de callback*. El servidor comienza instantáneamente a procesar otro evento y cuando termine la operación de entrada/salida, seguirá trabajando en la solicitud ejecutando la función de callback tan pronto como sea posible.

Gracias a este mecanismo, el servidor no necesita crear subprocessos lo que significa que no tiene apenas sobrecarga. Además, puesto que queremos tratar las peticiones de los usuarios, la mejor manera de hacerlo es de forma asíncrona así no es necesario esperar a que se traten las peticiones realizadas con anterioridad.

3.3.3.2. Ventajas de utilizar Node.js

Node.js es la plataforma de software más utilizada en la actualidad por encima de otros entornos de ejecución y lenguajes de programación como pueden ser C o PHP a la vez que tiene un tiempo de ejecución mucho mayor. No es un lenguaje de programación complejo pero requiere una mayor comprensión y cantidad de líneas de código que PHP. Cuando hablamos de Node.js hay algo que es necesario destacar, el conocido como Node Package Manager (NPM) que viene por defecto con la instalación de Node.js.

3.3.3.3. Librerías utilizadas

A continuación se especificarán los paquetes utilizados para el desarrollo de la API RestFul.

- **Async:** es un módulo que proporciona funciones sencillas para el tratamiento de llamadas asíncronas.
- **Body-parser:** es un middleware encargado de extraer todo el cuerpo de una request entrante y la muestra en el *req.body*

- Cors: este paquete se encarga de controlar el mecanismo que utilizan las cabeceras HTTP para permitir que un *user agent* tenga permiso para acceder a ciertos recursos desde un servidor en un dominio distinto al que pertenece.
- Dotenv: Sirve para cargar variables de entorno desde un fichero .env.
- Express: es un framework de desarrollo web inspirado en Sinatra y por consiguiente, el estandar de la gran mayoría de aplicaciones de Node.js. Es la librería que utilizaremos para desplegar el servidor de la API.
- Mongoose: es una herramienta encargada del tratamiento de bases de datos MongoDB de manera asíncrona.
- Nodemon: este paquete simplifica la tarea de volver a iniciar la aplicación cada vez que se realice un cambio en el código. Es un demonio que se ejecuta en segundo plano y es el encargado de comprobar los cambios e iniciar el servidor.
- Swagger-Jsdoc: genera definiciones OpenAPI a partir de comentarios JSDoc.
- Swagger-UI-Express: crea la interfaz de usuario de Swagger en base a las definiciones creadas con Swagger-JSDoc.

3.3.4. Visual Studio Code

Visual Studio Code es un editor gratuito y de código abierto desarrollado por Microsoft y que se distribuye mediante licencia MIT.¹ Se trata de un software multiplataforma (Windows, macOS y Linux). Está creado en NodeJS y Electron [23] y con Blink como motor gráfico.

Se trata de un editor hakeable, es decir, permite modificar el comportamiento del editor completamente (tema del editor, atajos de teclado, preferencias, etc), tiene soporte para Git integrado, autocompletado de código, terminal integrado y debugger.

Además posee una biblioteca de extensiones (figura 25) mediante la cual puedes añadir herramientas que ayuden en los proyectos o visualizar el código de forma más eficiente. También posee un live server a través del cual puedes visualizar código web en tu navegador predeterminado inmediatamente según guardas el código sin ser necesario refrescar.

A continuación se explicarán brevemente las extensiones utilizadas:

- **Docker**: esta extensión simplifica la forma en la que construir, controlar y desplegar aplicaciones *dockerizadas*, además proporciona herramientas dentro de un contenedor para realizar debugging de aplicaciones desarrolladas en Node.js, Python y .NET Core.
- **YAML**: proporciona compatibilidad completa con el lenguaje YAML con compatibilidad para la sintaxis de Kubernetes.
- **MarkdownLint**: incluye una biblioteca de reglas para entender los estándares y la coherencia de los archivos Markdown.
- **TODO Tree**: esta extensión busca rápidamente las etiquetas TODO y FIXME y las muestra en un árbol.

¹Licencia MIT: licencia creada en el Instituto Tecnológico de Massachusetts, se trata de una licencia de software libre que impone muy pocas restricciones en su reutilización

Desarrollo

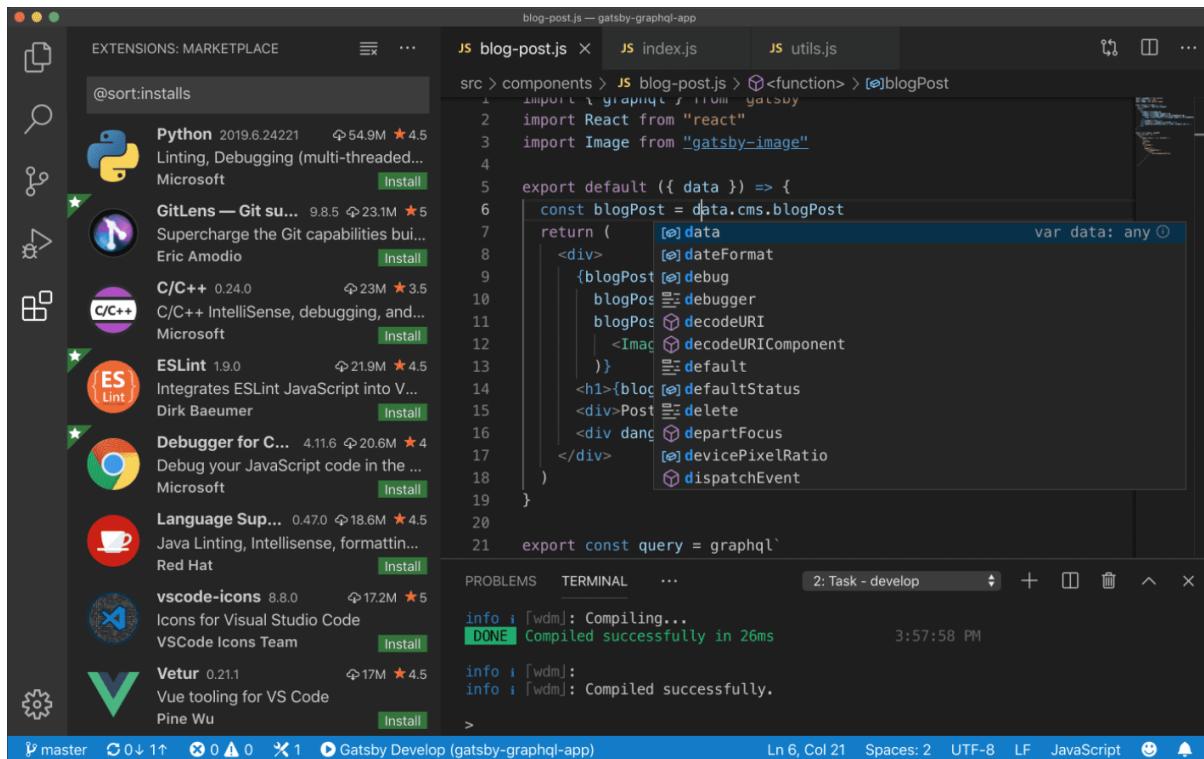


Figura 25: Visual Studio Code

3.3.5. Git y Github

En el desarrollo de cualquier proyecto de software surge la necesidad de gestionar los cambios que se producen entre versiones diferentes de un mismo código fuente. Habitualmente este proceso se inicia utilizando sufijos en los ficheros que se modifican (v1, antiguo, nuevo) pero este proceso no es ni eficiente ni práctico, mucho menos si trabajamos en equipo, dando lugar a la eliminación de archivos válidos realizados por otros componentes del equipo, incapacidad de conocer qué cambios se han realizado, etc.

En todo proyecto, más tarde o más temprano aparece la necesidad de trabajar en distintas ramas, en el entorno empresarial suelen ser desarrollo y producción. En la rama de desarrollo se procede a introducir las nuevas modificaciones y en la rama de desarrollo las versiones estables del programa.

Para ayudarnos con esta tarea aparecen soluciones como Git (figura 26), CVS que realizan tareas fundamentales como pueden ser:



Figura 26: Logo de Git

- Comparación de ficheros de tal manera que rápidamente encontraremos las diferencias entre los dos.

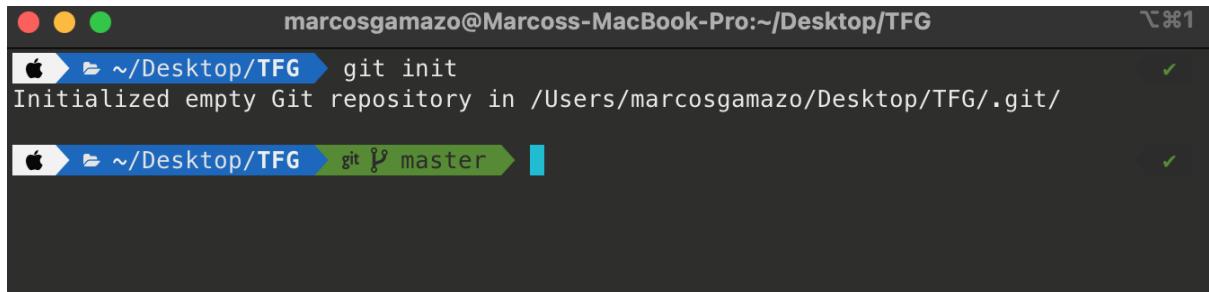
- Restauración de versiones anteriores.
- Fusionar versiones.
- Trabajar con varias ramas de un proyecto.
- Facilitar la colaboración.

En el caso del proyecto Sonidos del Cielo la herramienta de control de versiones utilizada ha sido **Git**.

Git surgió de la mano de Linus Torvalds [24], es un sistema de control de versiones de código abierto, multiplataforma, lo que nos permite crear repositorios en cualquier sistema operativo. Existe gran variedad de herramientas gráficas para manejar Git pero lo más utilizado es la línea de comandos dado que, una vez aprendidos los comandos básicos, nos permite realizar las operaciones con mayor velocidad.

Github es un servicio para alojar repositorios creado en San Francisco en el año 2008. Está pensado para compartir código fuente en cualquier lenguaje de programación y dejarlo a disposición de cualquiera que quiera reutilizarlo siempre y cuando haya permiso para ello.

Cualquier otro usuario que no sea el creador del proyecto puede realizar un *fork*, un fork no es otra cosa sino que la creación de una copia de otro proyecto en otro repositorio. También hay repositorios privados en los que sólo el creador del repositorio y las personas a las que él de acceso podrán participar (figura 27).



```
marcosgamazo@Marcoss-MacBook-Pro:~/Desktop/TFG
git init
Initialized empty Git repository in /Users/marcosgamazo/Desktop/TFG/.git/
git branch master
```

Figura 27: Inicialización de repositorio en Git

Github cuenta con diversas herramientas de las que las más destacables son:

- Gollum: permite la creación de wikis basadas en git.
- Sistema de revisión de código: para añadir comentarios y discutir sobre las modificaciones realizadas.
- Visor de ramas: donde podremos revisar los avances del repositorio.
- Herramienta de seguimiento de problemas: nos permite abrir tickets de incidencias o mejoras a realizar.

Desarrollo

3.3.6. Swagger

Una de las partes más importantes en cualquier proyecto es la documentación del código. Es muy habitual en nuestro campo que haya que modificar, mantener o simplemente utilizar código desarrollado por otra persona y se convierte en una tarea ardua que obliga a invertir demasiado tiempo en comprenderlo.

Para evitar estas situaciones es habitual adaptar herramientas que permitan comentar el código fuente del software que desarrollemos de tal manera que sea legible por cualquier persona. En el proyecto Sonidos del Cielo se ha decidido seleccionar la herramienta **Swagger**[25] para documentar la API RestFul.

Swagger se compone de una serie de herramientas, reglas y especificaciones mediante las cuales podremos documentar y probar una API de una manera sencilla. La principal baza para utilizar Swagger es que es realmente sencilla de implementar y cualquier persona es capaz de entenderlo, tenga conocimientos de programación o no. Esto es posible gracias a **Swagger UI**, es una de las características más importantes de la plataforma. Esta herramienta nos permite organizar las operaciones CRUD de la API basándose en los ficheros YAML o JSON y creando un entorno interactivo.

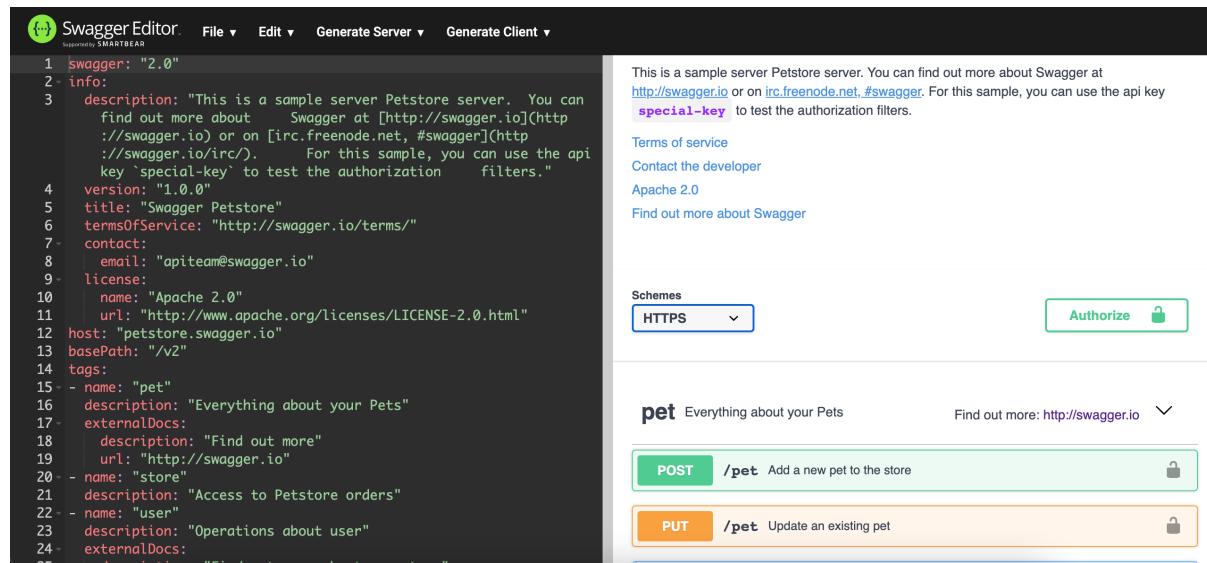


Figura 28: Swagger Editor

Para esta tarea podemos utilizar la versión del editor en línea, *Swagger editor* que proporciona la plataforma como se muestra en la figura 28. Esta herramienta nos indicará los errores que se hayan cometido y además brindará sugerencias y alternativas para mejorar la documentación.

Nuestra implementación pasa por la utilización de Swagger y Swagger UI como varios paquetes instalables dentro de Node.js [26] a través del instalador npm. Dentro de la definición del servidor se incluyen las opciones del servidor como se muestra en la figura 29 entre las que cabe destacar la versión del API Rest, la versión OpenAPI o la información del proyecto.

3.3. Herramientas utilizadas

```
const swaggerOptions = {
  definition: {
    openapi: '3.0.0',
    info: {
      title: 'Sonidos del Cielo API',
      version: '1.0.0',
      description: "You can find out more about the project at \n[http://www.sonidosdelcielo.org/](http://www.sonidosdelcielo.org/)\n",
      contact: {
        name: 'Sonidos del Cielo',
        url: 'http://sonidosdelcielo.org',
      },
      layout: "OperationsLayout",
      servers: [
        {url:'http://localhost:3000/'},
      ]
    },
    apis:['index.js','./routes/*.js']
  };
const swaggerDocs = swaggerJSDoc(swaggerOptions);

app.use('/api/docs', swaggerUI.serve, swaggerUI.setup(swaggerDocs));
```

Figura 29: Opciones del servidor de Swagger

Una vez establecidos los parámetros básicos del servidor y haber definido los componentes que forman la API Rest, es momento de añadir el código YAML [27] para crear la documentación de los distintos métodos. En ese código (figura 30) se indica el tipo de formato que se produce, **JSON** en nuestro caso. Los códigos de respuesta que proporciona la API.

```
13  /**
14  *
15  * @swagger
16  *
17  * /ecos/:
18  *   get:
19  *     tags: ['Ecos']
20  *     description: Devuelve un eco aleatorio
21  *     produces:
22  *       - application/json
23  *     responses:
24  *       '200':
25  *         description: OK
26  *         example:
27  *           $ref: '#/components/schemas/Eco'
28  *       '400':
29  *         description: Error
30  */
31
```

Figura 30: Definición Swagger

Una vez estén comentados todos los métodos de nuestra API en los distintos *end-points* que se hayan definido se mostrará una interfaz gráfica una vez accedamos a la url indicada en las opciones que se observan en la figura 29 dando como resultado la interfaz que se observa en la figura 31

Desarrollo

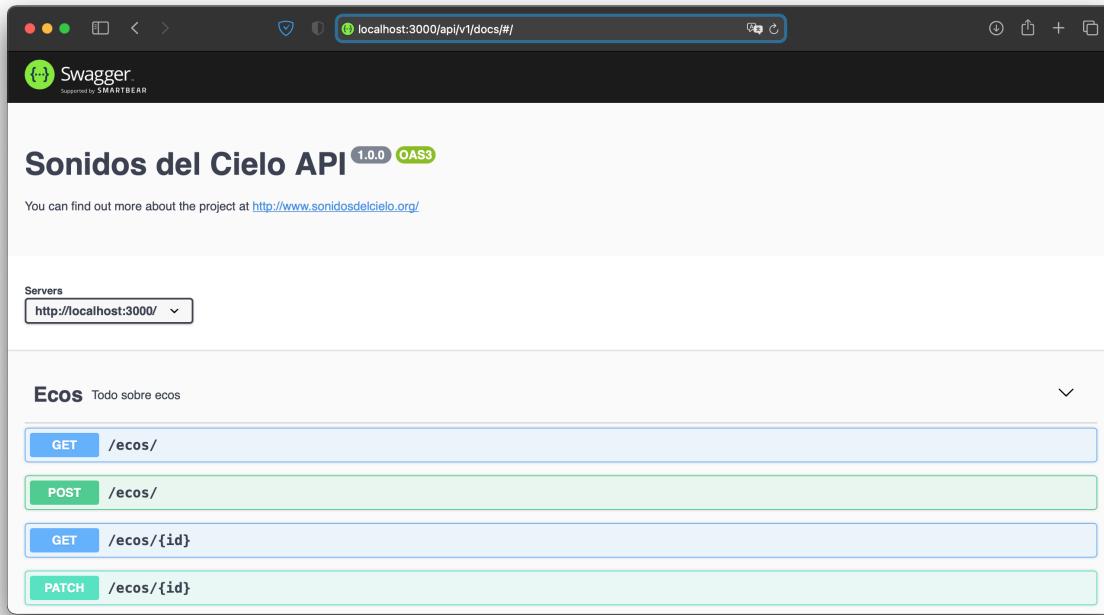


Figura 31: Interfaz web Swagger

3.3.7. Postman

Durante el desarrollo del servicio de la API Rest es necesario comprobar si las operaciones se están realizando correctamente y si se arrojan los resultados esperados. Para comprobarlo existen herramientas diversas que van desde una sentencia *curl* desde cualquier terminal de un sistema operativo a la utilización de herramientas gráficas que nos proporcionen información extra sobre las explotación de los métodos creados y sus resultados.

En esta última dirección, se hallan varias herramientas como pueden ser Advanced REST Client, Insomnia Rest Client, soapUI, Postman, etc. Para este trabajo se ha seleccionado **Postman** por el hecho de conocer la herramienta y su funcionamiento con anterioridad.

En el momento de su creación, Postman fue diseñado como una extensión del navegador Google Chrome aunque en la actualidad ya existen aplicaciones nativas para Windows, Linux y macOS. Agrupa una variedad de utilidades y herramientas que ayudan en el diseño y desarrollo de un API Rest. Algunas de las más destacables son:

- Creación de peticiones a APIs
- Desarrollo de test de validación de la API
- Entornos de trabajo

El interés principal de la utilización de esta herramienta es para realizar peticiones a la API y tener la capacidad de generar colecciones de peticiones que nos permitan testear de una forma sencilla y rápida. Todas las llamadas que se realicen en la API se

3.3. Herramientas utilizadas

guardan en colecciones. Todas las llamadas guardadas se pueden exportar a varios lenguajes de programación como se ve en la siguiente figura 32.

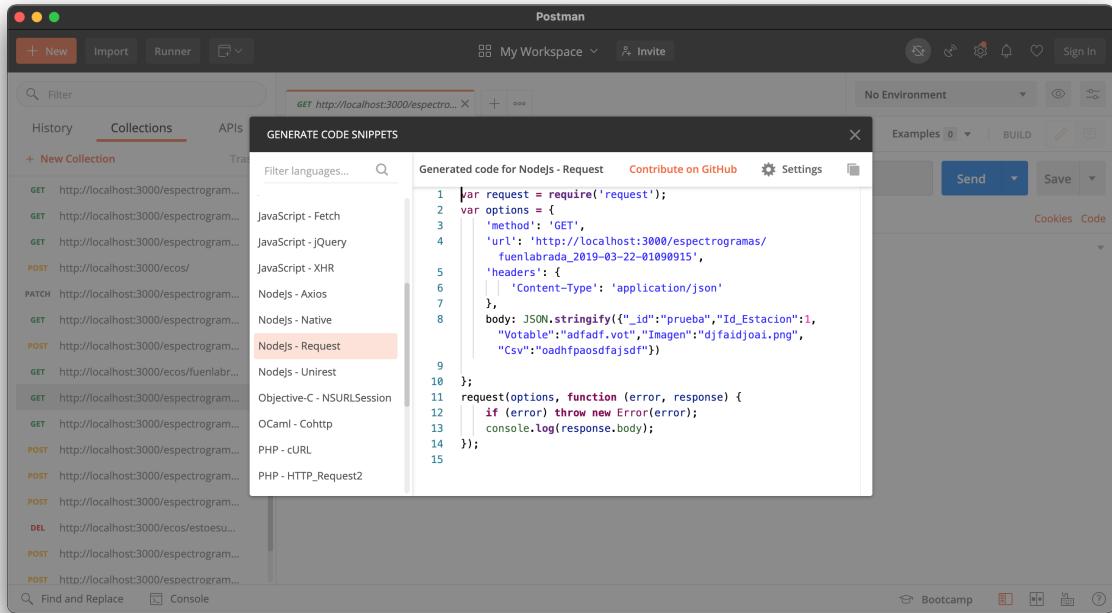


Figura 32: Generador de código en Postman

Otra característica interesante del programa es la capacidad de organizarse en entornos de trabajo. Gracias a esta característica, si es necesario realizar test en más de un API Rest, además de poder definir variables y clasificarlas según los entornos de trabajo. También tiene una herramienta mediante la cual se puede generar automáticamente documentación de la API y exportar toda la configuración a un fichero JSON.

3.3.8. Docker

En cualquier proyecto de desarrollo de software que tenga un tamaño considerable siempre surge el mismo problema, el software que se ha desarrollado en una máquina tiene un rendimiento y funcionamiento diferentes al que tenía en la máquina del desarrollo. Para ello se ha decidido utilizar docker (figura 33)



Figura 33: Logo de Docker

3.3.8.1. ¿Qué es Docker?

Se trata de una herramienta de virtualización pero a diferencia de las máquinas virtuales tradicionales es posible levantar el servicio de una forma mucho más rápida y en sistemas operativos mucho más eficientes. La idea principal es la de crear contenedores portables y ligeros de aplicaciones de software con la posibilidad de ejecutarse en cualquier dispositivo que tenga Docker instalado sin importar que sistema operativo esté funcionando [28].

3.3.8.2. Elementos básicos

Docker se compone de tres elementos básicos

- **Contenedor:** contiene todas las herramientas necesarias para el correcto funcionamiento de una aplicación sin necesidad de repositorios externos. Cada contenedor es una plataforma con sus propias directivas de seguridad y aislada del resto de la máquina.
- **Imagen Docker:** la imagen de docker se puede ver como un sistema operativo mínimo con aplicaciones instaladas (Por ejemplo un sistema operativo RedHat con MySQL instalado). A partir de una base se pueden añadir más elementos que sean necesarios en otro equipo donde se vaya a utilizar la imagen. Docker además proporciona herramientas sencillas mediante las cuales se mantienen las imágenes actualizadas.
- **Docker Hub:** es el lugar donde se almacenan las imágenes creadas por otros usuarios de tal forma que cualquiera pueda utilizarlas. Existen repositorios públicos en los que todo el mundo puede utilizar las imágenes y repositorios privados en los cuales es necesario comprar las imágenes que se vayan a usar. Con estos registros es muy sencillo desarrollar nuevas imágenes utilizandolos como plantilla en la que añadir nuevos componentes.

3.3.8.3. Funcionamiento

Docker incluye una interfaz gráfica como se muestra en la figura 34 para gestionar las imágenes y contenedores que existen en el equipo además de realizar los cambios necesarios para crear nuestra propia imagen.

3.3. Herramientas utilizadas

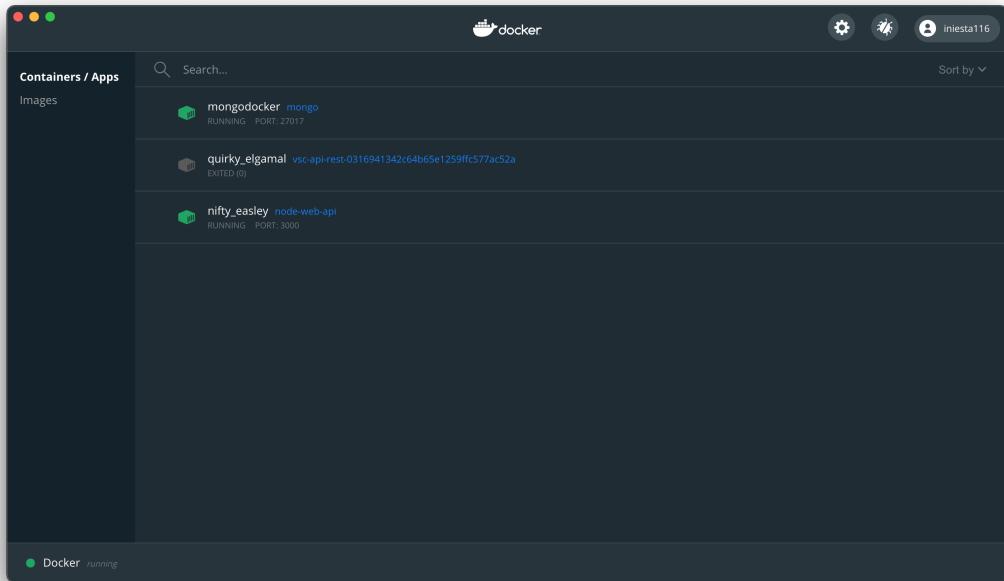


Figura 34: Interfaz gráfica de Docker

Partiendo del ejemplo de una aplicación web, ésta necesita diferentes tipos de software para poder ser desarrollada y desplegada (spring, MySQL, Tomcat, Maven...).

Docker permite introducir dentro de un contenedor todas las herramientas que necesite la aplicación además de la propia aplicación. De esta forma, puedo transportar el contenedor a cualquier tipo de máquina (servidor, ordenador portátil, Raspberry PI, etc) que tenga instalado Docker y directamente ejecutar la aplicación sin necesidad de realizar instalaciones complementarias, ejecutar ninguna herramienta más o comprobar compatibilidades.

Cada vez que se vaya a ejecutar el contenedor serán necesarias tanto la imagen base como las nuevas 'capas' que se han añadido. Docker automáticamente se encarga de acoplar la base, la imagen y las distintas capas para levantar el entorno deseado sobre el que poder trabajar.

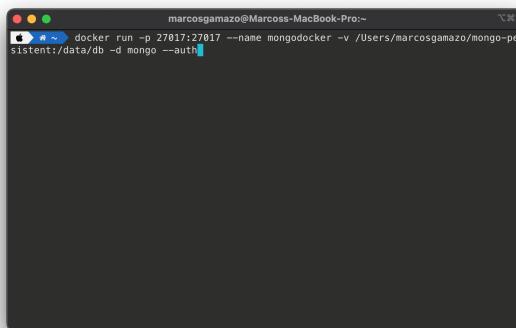


Figura 35: Inicio de un contenedor docker

Esto se puede hacer de manera muy sencilla mediante un terminal [29] y un comando como se ve en la figura 35 en la que se realiza un despliegue de una imagen con MongoDB sobre la que se ha desarrollado la base de datos del proyecto.

3.3.8.4. Principales características

- Autogestión de los contenedores.
- Posibilidad de desplegar múltiples contenedores en un mismo equipo.
- Fiabilidad.
- Contenedores ligeros que simplifican la labor de almacenamiento, transporte y despliegue.
- Capacidad de ejecutar una gran variedad de aplicaciones.
- Levantar los servicios es una operación muy rápida.
- La aplicación Docker es capaz de gestionar los recursos de la máquina para asignarlos a los distintos contenedores.
- Compartir las imágenes en Docker Hub.

3.3.8.5. Tipos de despliegues

Existen varias formas de desplegar un contenedor en Docker, la más sencilla es la que se muestra en la figura 35 en la que se puede crear un contenedor mediante un solo comando.

Otra forma de hacerlo es mediante un *Dockerfile* como el que se muestra a continuación:

```
1 FROM ubuntu:latest
2 MAINTAINER john doe
3
4 RUN apt-get update
5 RUN apt-get install -y python python-pip wget
6 RUN pip install Flask
7
8 ADD hello.py /home/hello.py
9
10 WORKDIR /home
```

Su funcionamiento es muy sencillo, se crea un fichero de texto plano en el que se indica la imagen base a utilizar y posteriormente mediante mandatos de terminal se instalan las distintas herramientas a incluir en la imagen personalizada. Una vez generada la imagen con el comando docker build se puede desplegar el contenedor mediante la interfaz gráfica o mediante la terminal.

Docker compose es una herramienta pensada para definir y correr aplicaciones con múltiples contenedores, se pueden definir los distintos servicios que van a formar la aplicación dentro del fichero *docker-compose.yml* de forma que pueden correr juntas en un entorno aislado.

```
1 version: "3"
2 services:
3   web:
4     build: .
5     ports:
6       - '5000:5000'
7     volumes:
8       - .:/code
9       - logvolume01:/var/log
10    links:
11      - redis
12  redis:
13    image: redis
14    volumes:
15      logvolume01: {}
```

Una vez creado el fichero, el contenedor se puede levantar simplemente con el comando `docker-compose up`.

3.3.9. Rasa

Para la realización del chatbot y dado que era la parte primordial del proyecto fue necesario estudiar las diferentes alternativas que existen en el mercado. Por un lado, tenemos las alternativas privadas (Amazon Lex, DialogFlow, Microsoft Bot Framework). Si bien es cierto que poseen todas las herramientas necesarias para desarrollarlo de una manera rápida y sencilla, en el momento que la aplicación creciese se volvería insostenible dado que la financiación es limitada y habría que desarrollar una nueva solución desde cero.

Es por eso que pasamos a analizar las alternativas Open Source y a poder ser que tuviese todas o gran parte de las herramientas que tenían las alternativas privadas además de poder acoplarla con Frontends para la visualización vía web y la capacidad de poder realizar llamadas a APIs externas. Teniendo en cuenta todos estos requisitos, se llegó a la conclusión de que la mejor herramienta a utilizar era **Rasa** ya que incluía la mayoría de herramientas que se necesitaban, es una herramienta de código abierto y además cuenta con una comunidad muy grande que en caso de que exista algún tipo de error o fallo puede ser de gran utilidad.

Dentro del framework tenemos herramientas mediante las cuales podemos definir las clases y los datos de entrenamiento del asistente y otros componentes que nos ayudarán en el diseño y el desarrollo del chatbot para nuestro proyecto.

3.3.9.1. Rasa NLU

El nombre proviene de sus siglas en inglés "*Natural Language Understanding*", se trata del componente mediante el cual nuestro asistente debe entender los mensajes que le está transmitiendo el usuario. A partir de los mensajes se busca una serie de parámetros conocidos como *intents* y en caso de existir, las *entities*.

Los *intents* son todas las sentencias que introduce el usuario separadas según la finalidad. Un usuario es capaz de expresar una misma acción mediante distintas palabras *utterances*. Por ejemplo, un usuario puede realizar un saludo de varias formas diferentes "*Buenas*", "*Hola*", "*¿Que tal todo?*" o "*hey*" y el asistente detectará de manera automática que se trata de un saludo.

Desarrollo

Dentro de un mensaje también es necesario analizar otro componente como son las entidades, que es la información clave, ya que a través de estas es posible personalizar la respuesta del asistente. Un ejemplo podría ser si al asistente le decimos "*Qué tiempo va a hacer en {nombre_ciudad}*", donde nombre_ciudad es una entidad que se podría utilizar para buscar el dato en una API externa.

En RASA, tanto los intents como las entities están definidas y almacenadas dentro de una carpeta que genera el propio framework conocida como *data*. En el interior de esa carpeta existe un fichero en lenguaje *Markdown* llamado nlu.md. Se entrará en la definición de los datos y los documentos de una forma más detallada más adelante.

Rasa NLU es un pipe por el cuál se introducen los mensajes. A esta herramienta se le pueden acoplar más librerías para mejorar el procesamiento de la entrada de mensajes, una de ellas puede ser *Spacy*, que se trata de un componente mediante el cual se añaden elementos de aprendizaje en varios idiomas para el entrenamiento del asistente [30].

3.3.9.2. Rasa Core

Es una herramienta del asistente que posee gestión de diálogos basada en el aprendizaje mediante *machine learning* que es capaz de predecir la siguiente acción basada en la entrada del asistente, los datos de entrenamiento y el historial de conversaciones. Los datos de entrada están definidos en los diferentes *stories* detallados más adelante.

3.3.9.3. Rasa NLG

El nombre procede de las siglas en inglés "*Natural Language Generation*" y se trata del componente en el cuál están definidos todas las acciones y mensajes que el asistente puede utilizar para responder a mensajes proporcionados por el usuario y están definidas en las *utterances*.

3.3.9.4. Procesado de un mensaje

En el diagrama mostrado en la figura 36 se puede ver la forma en la que Rasa NLU procesa un mensaje y cómo da la respuesta el Rasa Core.

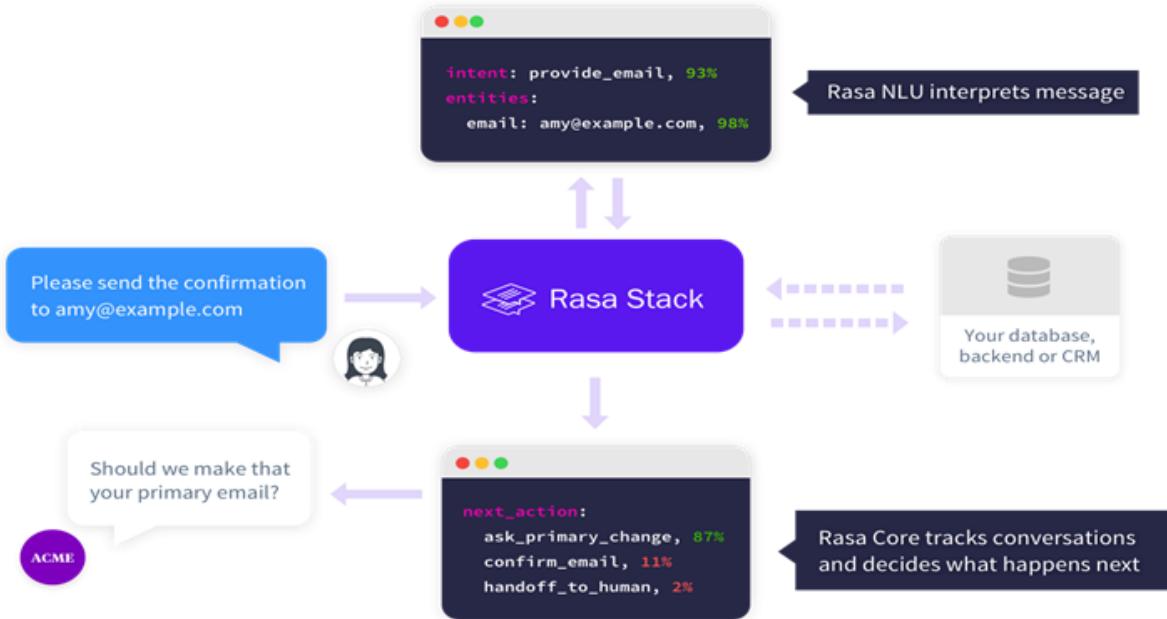


Figura 36: Proceso de clasificación de un mensaje

1. En primer lugar el usuario envía un mensaje que lo recibe el Rasa Stack que es el encargado de hacer la función de orquestador (se encarga de gestionar las comunicaciones entre distintas capas del framework).
2. El componente NLU del asistente se encarga de catalogar los intents que existen en el mensaje y extrae las características del mensaje (entities y slots). Esta parte muestra un porcentaje (grado de confianza de la predicción) que en caso de ser inferior al 40% se mostrará un mensaje mostrando que el asistente no ha entendido correctamente al usuario. Estos mensajes se pueden personalizar dentro de la plantilla *utter_default* del fichero domain.yml.
3. Además Rasa Core tiene que analizar qué acción tomar con el mensaje recibido, en función de las historias definidas durante el entrenamiento del chatbot. Se generan múltiples opciones aunque se selecciona la que tiene un mayor % de coincidencia. Aquí es dónde se ejecutan las acciones, por ejemplo una consulta a una API o una base de datos externa.
4. Para terminar, el chatbot retorna la respuesta al usuario por el mismo canal de entrada.

3.3.9.5. Componentes de Rasa

En esta sección se detallan las herramientas básicas del framework para entender la utilidad y el funcionamiento para saber si es necesario o no su utilización en nuestro proyecto.

3.3.9.5.1. Intents

Son los mensajes más frecuentes definidos dentro del entorno del chatbot que se espera que sucedan con el usuario. Los intents están encuadrados dentro del componente Rasa NLU tal y trata sobre cómo hay que clasificar los mensajes enviados por el usuario. En la figura 37 se observa cómo se realiza la definición de un intent, en este caso el **intent greet**.

```
- intent: greet
  examples: |
    - hey
    - hello
    - hi
    - hello there
    - good morning
    - good evening
    - moin
    - hey there
    - let's go
    - hey dude
    - yo
    - goodmorning
    - goodevening
    - good afternoon
    - hola
    - Buenas
    - holi
    - que hay?
    - Como estas?
    - Hay alguien ahi?
```

Figura 37: Definición del intent greet

3.3.9.5.2. Utterances

Los utterances son las respuestas automáticas definidas que puede dar el asistente para responder los mensajes introducidos por el usuario. Se trata de una parte fundamental del componente Rasa NLU y se utilizan como una base para el entrenamiento del asistente. Están definidos dentro del fichero domain.yml como se ve en la figura 38 del proyecto y en él, hay que incluir todas y cada una de las posibles respuestas que el chatbot puede utilizar en cualquier momento de una conversación.

```
utter_greet:
  - text: "Hey! How are you?"
  - text: "Hola, Como estas?"
  - text: "Que tal estas?"
  - text: "Hola, que quieres hacer?"
```

Figura 38: Definición de un utterance

3.3.9.5.3. Slots

Se trata de uno de los componentes más importantes dentro del framework Rasa. En ellos se guarda toda la información destacable extraída de las conversaciones como puede ser el nombre de una ciudad o alguna respuesta que el usuario entrega al asistente tal y como se muestra en la figura 39. Se trata del componente que hace de memoria del programa y la cual es accesible desde fuera del chatbot de manera que se puede tener constancia del estado a tiempo real del asistente. Están asociados a los entities y se encuentran definidos dentro del fichero domain.yml.

```
slots:
  name:
    type: text
    initial_value: "friend"

  sound:
    type: text
    initial_value: "0"
```

Figura 39: Definición de un slot

3.3.9.5.4. Dominio

El dominio es el fichero en el cuál están definidos todos los componentes que conforman el universo del chatbot al momento de producirse una conversación con el usuario. En nuestro caso una vez presentado el mensaje inicial y recibido el nombre que proporciona el usuario, se presentan las distintos flujos que puede tomar la conversación. En la siguiente figura 40 se presenta un ejemplo de definición de un fichero domain.yml

```
session_config:
  session_expiration_time: 10
  carry_over_slots_to_new_session: false

intents:
  - greet
  - opciones
  - tutorial
  - sonidos_del_cielo
  - mi_nombre

entities:
  - nombre
  - respuesta1
  - respuesta2
  - respuesta3
  - eco

responses:
  utter_preguntar_nombre:
    - text: "¿Cuál es tu nombre?"
    - text: "¿Como te llamas?"

slots:
  nombre:
    type: text
    initial_value: "amigo"

actions:
  - action_dar_sonido
  - action_pregunta_uno
```

Figura 40: Definición de un dominio

Desarrollo

3.3.9.5.5. Actions

Las acciones en Rasa son funciones programadas en Python dentro del fichero *actions.py* que se utilizan de forma que el asistente tenga un comportamiento programado ante una acción. Al realizarse en Python permite definir múltiples funcionalidades y es en este punto donde se debe desarrollar cualquier comportamiento personalizado dentro del bot, además de retornar mensajes estáticos.

Una acción se define de la siguiente manera:

1. Se define una clase de tipo *Action*
2. Dentro de esa clase, es necesario definir una función que retorne el nombre del action que se ha creado. Esto se realiza de la siguiente manera:

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

3. Definimos la función *run*, es aquí donde irá la lógica de la función.

```
1 def run(self, dispatcher: CollectingDispatcher,  
2         tracker: Tracker,  
3         domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:  
4  
5     dispatcher.utter_message(text="Hello World!")  
6  
7     return []
```

4. Se manda el mensaje desde el *dispatcher* mediante la función *utter_message("mensaje")*. Este mensaje será el que envíe el chatbot al usuario
5. Por último se colocan las variables que sea necesario mantener

Se puede observar la definición completa de un *action* en la figura 41.

```
#####
# Devuelve un sonido para analizar
#####

class action_dar_sonido(Action):
    def name(self):
        return 'action_dar_sonido'

    def run(self, dispatcher, tracker, domain):
        #Nos da el path absoluto de un sonido aleatorio del directorio sounds
        file = random.choice(os.listdir("sounds"))
        path = os.path.join("/Users/marcosgamazo/PycharmProjects/chatbot-TFG/sounds",file)
        dispatcher.utter_message(path)
        return [SlotSet("eco",os.path.splitext(file)[0])]
```

Figura 41: Definición de un action

3.3.9.5.6. Stories

Las stories de Rasa son un elemento mediante el cual se especifica las diferentes variaciones disponibles dentro del flujo de conversación, es importante definir todas las distintas ramas que puede tomar la conversación para que el chatbot sepa en todo momento que camino escoger. Esto se realiza mediante la secuencia de intents y actions (ya sean custom actions o utterances). La definición de las stories se realiza en YAML como se muestra en la figura 42.

```
1  version: "2.0"
2
3  #####
4  ##### FLUJOS DE DIALOGO #####
5  #####
6 stories:
7
8 - story: hasta dar las opciones disponibles
9   steps:
10    - intent: greet
11    - action: utter_greet
12    - action: utter_preguntar_nombre
13    - intent: mi_nombre
14    - action: utter_opciones
15
16 - story: en caso de seleccionar el tutorial
17   steps:
18    - intent: tutorial
19    - action: utter_tutorial
20    - action: utter_opciones
21
22 - story: en caso de que el usuario quiera clasificar
23   steps:
24    - intent: clasificar
25    - action: action_dar_sonido
26    - action: action_pregunta_uno
27
```

Figura 42: Definición de un story

3.3.9.5.7. Rules

En Rasa las reglas son pequeñas porciones del flujo de diálogo que siempre tienen un camino concreto, es decir, son interacciones en las que siempre se recibe la misma respuesta o un el mismo conjunto de respuestas. Se utilizan principalmente para responder preguntas frecuentes. Se definen de la siguiente forma:

```
1  version: "2.0"
2
3 rules:
4
5 - rule: da las opciones disponibles
6   steps:
7    - intent: opciones
8    - action: utter_opciones
```

Desarrollo

Esta regla lo que nos permite es que siempre que el usuario introduzca una cadena y el chatbot detecte que está solicitando las opciones, nos muestre un menú con todas las opciones disponibles a realizar en el chatbot.

3.3.9.5.8. Conectores

Los conectores son elementos que facilitan la conexión entre el asistente con los canales de entrada y salida de información. Posee gran variedad de conectores pre-establecidos aunque también se pueden desarrollar conectores personalizados para otros servicios.

Estos son los conectores predefinidos:

- Web
- Facebook Messenger
- Slack
- Telegram
- Twilio
- Google Hangouts Chat
- Microsoft Bot Framework
- Cisco Webex Teams
- RocketChat

En nuestro caso hemos utilizado un canal de tipo REST. Una vez que se haya activado el canal y esté el servidor levantado, podremos interactuar con el chatbot enviando mensajes a través de un método POST en la dirección `http://<host>:<port>/webhooks/rest/webhook` mediante mensajes con formato request como se muestra en la figura 43.

```
{
  "sender": "test_user", // sender ID of the user
  "message": "Hi there!",
}
```

Figura 43: Request de un mensaje a través del canal REST

La respuesta que proporciona Rasa es un body que contiene un JSON con las diferentes respuestas que brinda el asistente, tal y como se puede ver en la figura 44.

```
[{"text": "Hey Rasa!"}, {"image": "http://example.com/image.jpg"}]
```

Figura 44: Response de un mensaje a través del canal REST

3.3.10. HTML

Es necesario recalcar las principales mejoras de HTML respecto a su versión predecesora HTML 4.01 [31].

- Manejo de los gráficos vectoriales, audio y video desde HTML mediante la utilización de etiquetas. Se incorporan las etiquetas `<audio>` para controlar ficheros de sonido, `<video>` para el uso de archivos de video, `<canvas>` para crear dibujos y `<svg>` para el tratamiento de gráficos vectoriales.
- Creación de nuevas etiquetas como para la simplificación de la sintaxis.
- Mejora de los formularios adaptándose a nuevas entradas como `email`, `number`, `time`, etc.
- Control de APIs mediante Javascript

En la figura siguiente 45 se enseña un ejemplo de estructura de una página web desarrollada en HTML5. La estructura se compone principalmente de una cabecera o head, definida mediante la etiqueta `<head>` y siendo necesario especificar la codificación de caracteres a usar a través de la etiqueta `<meta>`. Indicamos el título de la página mediante la etiqueta `<title>`. A continuación estableceremos la etiqueta `<body>`, que es el cuerpo de la página web. En este ejemplo aparece una etiqueta `<p>` que define un párrafo.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>EJEMPLO</title>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```

Figura 45: Estructura de un fichero HTML5

3.3.11. CSS3

CSS es un lenguaje de hojas de estilo en cascada, en otras palabras, es el encargado de definir la apariencia que deben adquirir los documentos HTML. Engloba todo lo relativo a fuente, colores, líneas, altura, márgenes. Estas operaciones se pueden realizar mediante etiquetas HTML pero es mucho más tedioso y complicado.

Desarrollo

En un primer momento, CSS se utilizaba tal y como se ha expuesto en el párrafo anterior, aunque en la actualidad es capaz de generar animaciones, transformaciones y otros efectos de mayor complejidad. En la siguiente figura 46 se muestra un extracto del fichero CSS utilizado en el proyecto.

```
/* ===== css related to chats ===== */
.widget {
    display: none;
    width: 350px;
    right: 15px;
    height: 500px;
    bottom: 5%;
    position: fixed;
    background: #f7f7f7;
    border-radius: 10px 10px 10px 10px;
    box-shadow: 0 0px 1px 0 rgba(0, 0, 0, 0.16), 0 0px 10px 0 #00000096;
}
```

Figura 46: Estructura de un fichero HTML con referencia al fichero CSS

Capítulo 4

Diseño

4.1. Diseño de las herramientas

4.1.1. Diseño de la base de datos

Las diferentes colecciones de la base de datos se han extraído a partir de ficheros existentes dentro de un servidor del Spanish Virual Observatory (SVO) dentro del Departamento de Arquitectura y Tecnología de Sistemas Informáticos de la Escuela Técnica Superior de Ingenieros Informáticos de la Universidad Politécnica de Madrid.

A partir de estos ficheros CSV se tomó la decisión de crear las siguientes colecciones.

- **Estación:** En esta colección tenemos la información necesaria sobre las estaciones de radiodetección disponibles. En el momento de la realización del trabajo sólo existe una estación trabajando, aunque existe previsión de que en un futuro se adhieran varias más al proyecto.

- _id: el identificador de la estación.
- Localización: la ubicación donde se encuentra la estación.
- web: página web de la estación.

- **Eco:** esta colección contiene los datos de las detecciones.

- _id: el identificador de la estación.
- Fecha: la fecha en la que se produce la detección.
- Id_Estacion: la estación donde se detecta el meteoroide.
- Duración: la duración que tiene el eco.

- **Espectrograma:** espectrograma que se genera a partir de los datos del eco detectado.

- _id: el identificador del espectrograma.
- Id_Estacion: la estación de detección..
- Votable: ubicación del fichero votable del espectrograma en el servidor.
- Imagen: ubicación de la imagen del espectrograma en el servidor.

4.1. Diseño de las herramientas

- Csv: ubicación del fichero csv del espectrograma en el servidor.
- **Curva Luz:** curva de luz que se genera a partir de los datos eco detectado.
 - _id: el identificador de la curva de luz.
 - Id_Estacion: la estación de detección.
 - Votable: ubicación del fichero votable de la curva de luz en el servidor.
 - Imagen: ubicación de la imagen de la curva de luz en el servidor.
 - Csv: ubicación del fichero csv de la curva de luz en el servidor.
- **Sonido:** sonido generado a través de la curva de luz
 - _id: el identificador del sonido.
 - Ruta: localización del fichero de sonido dentro del servidor.
- **Clasificación:** clasificación de un eco, puede provenir del asistente virtual o de la plataforma zooniverse.
 - _id: el identificador de la clasificación.
 - idUsuario: identificador del usuario que realiza la tarea de clasificación.
 - Respuesta1: primera respuesta dada por el usuario.
 - Respuesta2: segunda respuesta dada por el usuario.
 - Respuesta3: tercera respuesta dada por el usuario.

Las tablas las obtenemos desde ficheros que se encuentran en un servidor del Spanish Virtual Observatory (SVO) dentro del Departamento de Arquitectura y Tecnología de Sistemas Informáticos de la Escuela Técnica Superior de Ingenieros Informáticos de la Universidad Politécnica de Madrid. A partir de este fichero se crean las tablas que se aprecian en la figura 47

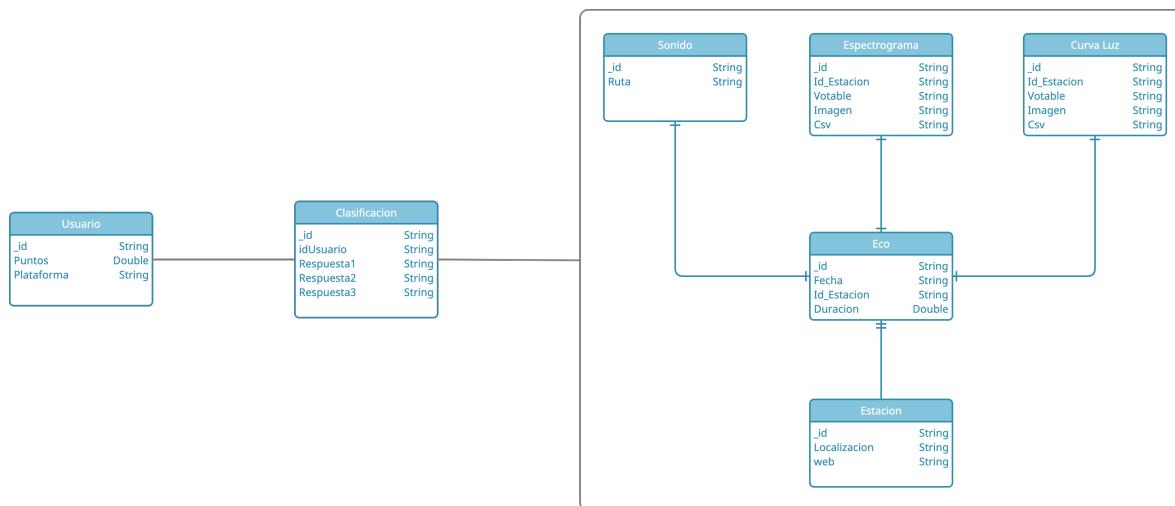


Figura 47: Colecciones de la base de datos

Diseño

Aunque hemos explicado anteriormente que no es estrictamente necesario que existan relaciones entre las tablas, en este proyecto sí que es necesario, puesto que varias tablas proceden del mismo meteorito.

4.1.2. Estructura de la aplicación RESTful

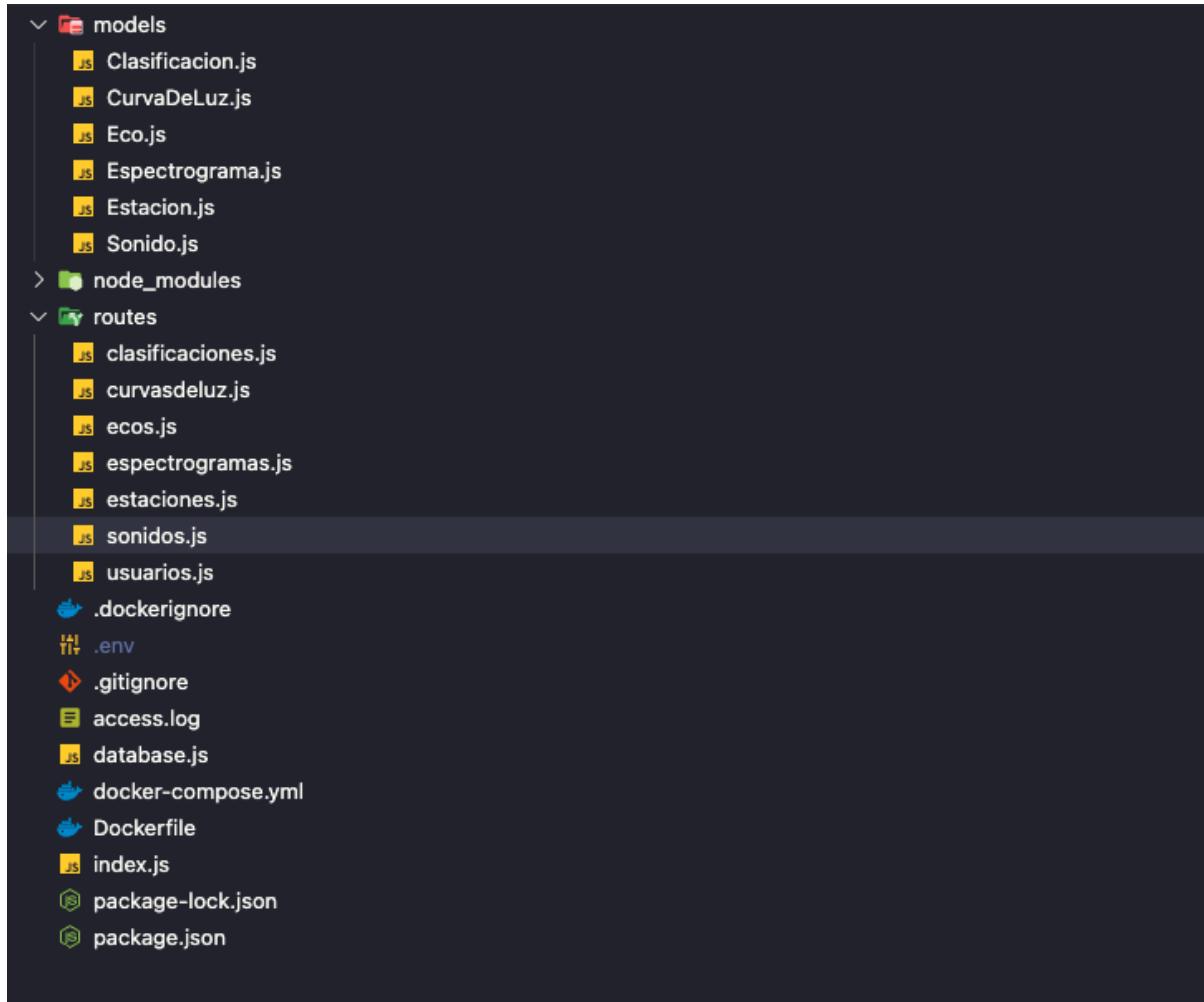


Figura 48: Estructura de la aplicación RESTful

En esta figura 48 se observa la estructura que adopta el desarrollo de la API RESTful. A continuación se va a explicar qué es fichero y qué contiene cada carpeta existente en el proyecto.

En la raíz del proyecto tenemos diversos ficheros:

- .dockerignore: en este fichero se especifican los ficheros a excluir dentro de un contenedor de docker.
- .env: se trata de un fichero en el cual se establecen ciertas variables de entorno. En nuestro caso contiene la cadena de conexión a la base de datos. Se utiliza este fichero para evitar posibles filtraciones de la contraseña en caso de que el código sea compartido en Github o cualquier otra plataforma de colaboración.

- .gitignore: fichero con un funcionamiento similar al dockerignore pero destinado a Git y Github.
- database.js: se establece la comunicación entre la API REST y la base de datos a través de la cadena de conexión existente dentro del fichero .env.
- docker-compose.yml: este fichero se utiliza para el despliegue automático de varios contenedores de manera simultánea, estableciendo entre sí una especie de red virtual"que permite la conexión entre los distintos contenedores.
- Dockerfile: este fichero se utiliza para la generación de una imagen del proyecto que luego se utiliza en el despliegue de un contenedor o dentro de un docker-compose.
- index.js: es el fichero principal de la aplicación. En él están definidas la ruta principal de la API, las rutas de los demás controladores, las variables principales de Swagger y la ruta en la que se despliega el servicio.
- package.json: en este fichero están definidas las opciones de los módulos de node.js utilizados en el desarrollo y las principales opciones de nuestra aplicación (nombre, versión, descripción y fichero *main*).

En primer lugar se observa la carpeta models. Esta carpeta contiene todos los modelos necesarios de las colecciones existentes en la base de datos. Es necesario definir modelos puesto que el módulo de Node.js que estamos utilizando en el desarrollo de la API nos exige su utilización para un correcto funcionamiento.

La carpeta node_modules contiene todos los módulos necesarios para nuestra implementación.

A continuación tenemos la carpeta routes, en ella están definidas todas las rutas que tiene accesible la API y todos las llamadas disponibles (GET,POST,PATCH y DELETE).

4.1.3. Estructura de la aplicación Rasa

En la siguiente figura 49 se muestra cuál es la estructura de la aplicación desarrollada en Rasa.



Figura 49: Estructura del asistente virtual

Dentro de todos los ficheros y carpetas que se crean al iniciar un proyecto de Rasa se van a explicar todos los elementos necesarios para el correcto desarrollo.

- **Actions:** dentro de esta carpeta el único fichero que necesitamos es el *action.py*. En él se encuentran todas las acciones personalizadas que requiere el proyecto.
- **data:** dentro de esta carpeta se encuentran los ficheros más importantes del módulo NLU (Natural Language Understanding). Estos ficheros son:
 - nlu.yml: en este fichero se encuentran los intents definidos por el usuario.
 - rules.yml: se definen cada una de las posibles reglas a utilizar por el asistente.
 - stories.yml: en él se desarrollan los diferentes caminos posibles dentro del flujo de diálogo.
- **models:** dentro de esta carpeta se guardan los modelos autogenerados cada vez que entrenamos el chatbot.
- **sounds:** en el interior de esta carpeta se encuentra una muestra de los sonidos disponibles dentro de la base de datos.
- **test:** aquí están los ficheros autogenerados con posibles conversaciones que puede seguir el chatbot.
- **venv:** se trata del entorno virtual en el que se ejecuta el chatbot.
- **.gitignore:** fichero destinado a Git y Github en el cual se especifican los ficheros y carpetas dentro del documento que no se subirán al repositorio.
- **config.yml:** dentro de este fichero de configuración se definen las distintas políticas y distintos componentes que el modelo utilizará para realizar predicciones

basadas en las entradas que proporcione el usuario.

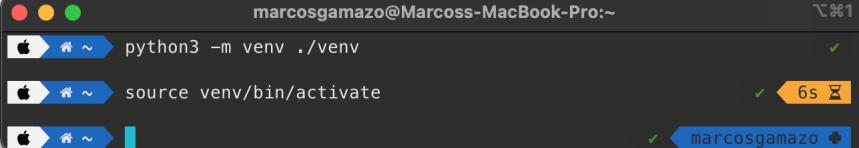
- **credentials.yml**: en este fichero se incluyen los diferentes credenciales para conectar Rasa con múltiples servicios.
- **domain.yml**: se trata de uno de los ficheros más importantes del chatbot. Define el universo en el que opera el asistente. Dentro de él se incluyen todos los intents, entities, slots, responses, forms y acciones que el chatbot debe conocer. En él también se incluyen parámetros de configuración de las sesiones del chatbot.
- **endpoints.yml**: se especifican los distintos endpoints a los que se puede conectar el asistente. En nuestro caso sólo se conecta con el servidor en el que se despliegan las acciones.

4.2. Despliegue de las herramientas

4.2.1. Despliegue del asistente

Es obligatorio especificar en qué sistema operativo se va a realizar el despliegue de la herramienta. En caso de que se realice sobre Windows, sería necesario instalar primeramente el lenguaje Python, ya que todo el framework de Rasa está basado en este lenguaje. Si se realiza sobre un sistema Linux o un sistema UNIX no es necesario instalarlo ya que en la mayoría de distribuciones viene instalado por defecto.

Una vez tengamos instalado el lenguaje Python, se crea un entorno virtual como se muestra en la figura 50. Esta herramienta nos permite instalar todas las dependencias necesarias para el framework sin comprometer la estabilidad del sistema operativo.



A screenshot of a macOS terminal window titled "marcosgamazo@Marcoss-MacBook-Pro:~". It shows two commands being run: "python3 -m venv ./venv" and "source venv/bin/activate". The terminal interface includes a status bar at the bottom with the user's name and a battery icon.

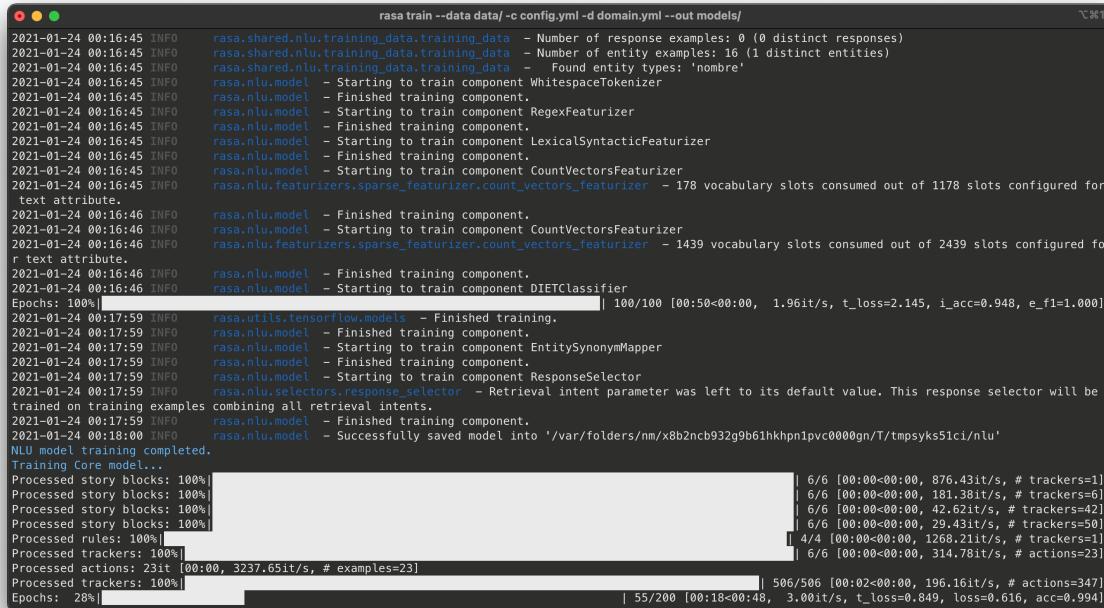
Figura 50: Creación del entorno virtual

Una vez creado y activado el entorno virtual procedemos a instalar Rasa mediante el comando *pip3 rasa install*. Utilizamos pip3 para especificar que se instalará el framework utilizando Python 3. Cuando termine de instalarse el framework es el momento de crear la carpeta que va a contener el proyecto y acceder a ella mediante un terminal. Una vez realizado este proceso, simplemente tendremos que introducir el mandato *rasa init -no-prompt*. Este mandato nos crea el bot de ejemplo de Rasa con el que se puede interactuar nada más crearlo.

Una vez tengamos la estructura creada, es hora de modificar los ficheros para que nuestro asistente se adapte a las necesidades del proyecto. Una vez modificados los

Diseño

distintos archivos necesarios, se debe entrenar el asistente para que cree nuestro propio modelo a utilizar para que sea capaz de comprender los mensajes que pueda introducir el usuario y que ejecute las acciones requeridas en cada momento. Para realizar el entrenamiento introduciremos en un terminal el mandato **rasa train -data data/ -c config.yml -d domain.yml -out models/**, en él se especifican los distintos archivos que tiene que utilizar para entrenar el NLU y la carpeta en la que se almacenará el modelo generado a partir del entrenamiento, como se muestra en la figura 51.



A screenshot of a terminal window titled "rasa train --data data/ -c config.yml -d domain.yml --out models/". The window displays a log of training progress from January 24, 2021, at 00:16:45. The log shows the initialization of various NLU components like whitespace tokenizer, regex featurizer, lexical syntactic featurizer, and count vectors featurizer. It tracks the consumption of vocabulary slots and the training of the DIET classifier. The log concludes with the successful saving of the trained NLU model into a specified directory and the completion of the training process.

```
2021-01-24 00:16:45 INFO rasa.shared.nlu.training_data.training_data - Number of response examples: 0 (0 distinct responses)
2021-01-24 00:16:45 INFO rasa.shared.nlu.training_data.training_data - Number of entity examples: 16 (1 distinct entities)
2021-01-24 00:16:45 INFO rasa.shared.nlu.training_data.training_data - Found entity types: 'nombre'
2021-01-24 00:16:45 INFO rasa.nlu.model - Starting to train component WhitespaceTokenizer
2021-01-24 00:16:45 INFO rasa.nlu.model - Finished training component.
2021-01-24 00:16:45 INFO rasa.nlu.model - Starting to train component RegexFeaturizer
2021-01-24 00:16:45 INFO rasa.nlu.model - Finished training component.
2021-01-24 00:16:45 INFO rasa.nlu.model - Starting to train component LexicalSyntacticFeaturizer
2021-01-24 00:16:45 INFO rasa.nlu.model - Finished training component.
2021-01-24 00:16:45 INFO rasa.nlu.model - Starting to train component CountVectorsFeaturizer
2021-01-24 00:16:45 INFO rasa.nlu.featurizers.sparse_featurizer.count_vectors_featurizer - 178 vocabulary slots consumed out of 1178 slots configured for text attribute.
2021-01-24 00:16:46 INFO rasa.nlu.model - Finished training component.
2021-01-24 00:16:46 INFO rasa.nlu.model - Starting to train component CountVectorsFeaturizer
2021-01-24 00:16:46 INFO rasa.nlu.featurizers.sparse_featurizer.count_vectors_featurizer - 1439 vocabulary slots consumed out of 2439 slots configured for text attribute.
2021-01-24 00:16:46 INFO rasa.nlu.model - Finished training component.
2021-01-24 00:16:46 INFO rasa.nlu.model - Starting to train component DIETClassifier
Epochs: 100% | 100/100 [00:50<00:00, 1.96it/s, t_loss=2.145, i_acc=0.948, e_f1=1.000]
2021-01-24 00:17:59 INFO rasa.utils.tensorflow.models - Finished training.
2021-01-24 00:17:59 INFO rasa.nlu.model - Finished training component.
2021-01-24 00:17:59 INFO rasa.nlu.model - Starting to train component EntitySynonymMapper
2021-01-24 00:17:59 INFO rasa.nlu.model - Finished training component.
2021-01-24 00:17:59 INFO rasa.nlu.model - Starting to train component ResponseSelector
2021-01-24 00:17:59 INFO rasa.nlu.selectors.response_selector - Retrieval intent parameter was left to its default value. This response selector will be trained on training examples combining all retrieval intents.
2021-01-24 00:17:59 INFO rasa.nlu.model - Finished training component.
2021-01-24 00:18:00 INFO rasa.nlu.model - Successfully saved model into '/var/folders/nm/x8b2ncb932g9b61hkhpn1pvc0000gn/T/tmpsyks51ci/nlu'
NLU model training completed.
Training Core model...
Processed story blocks: 100% | 6/6 [00:00<00:00, 876.43it/s, # trackers=1]
Processed story blocks: 100% | 6/6 [00:00<00:00, 181.38it/s, # trackers=6]
Processed story blocks: 100% | 6/6 [00:00<00:00, 42.62it/s, # trackers=42]
Processed story blocks: 100% | 6/6 [00:00<00:00, 29.43it/s, # trackers=50]
Processed rules: 100% | 4/4 [00:00<00:00, 1268.21it/s, # trackers=1]
Processed trackers: 100% | 6/6 [00:00<00:00, 314.78it/s, # actions=23]
Processed actions: 23it [00:00, 3237.65it/s, # examples=23] | 506/506 [00:02<00:00, 196.16it/s, # actions=347]
Processed trackers: 100% | 55/200 [00:18<00:48, 3.00it/s, t_loss=0.849, loss=0.616, acc=0.994]
Epochs: 28% | 55/200 [00:18<00:48, 3.00it/s, t_loss=0.849, loss=0.616, acc=0.994]
```

Figura 51: Entrenamiento del asistente

Una vez realizado el entrenamiento es necesario poner en funcionamiento el servidor principal de Rasa, para ello introducimos en un terminal el comando **rasa run -m models -enable-api -cors "*"**, de esta manera le estamos diciendo a rasa qué modelo tiene que utilizar, también indicamos que se tiene que activar la API de Rasa para enviar y recibir mensajes y además activamos CORS para poder acceder a los servicios de Rasa desde un dominio diferente al de ejecución.

También es necesario activar el servidor que ejecuta las distintas acciones a realizar por el asistente, para ello se introduce el mandato **rasa run actions -actions actions -vv**.

4.2.2. Despliegue de la base de datos y API

La base de datos es un elemento fundamental para el funcionamiento de este proyecto pero a la vez de varios procesos paralelos en los que trabaja el equipo. Es por ello que se ha decidido realizar un despliegue mediante Docker, para reducir al mínimo las posibles incompatibilidades. De la mano del despliegue de la base de datos está el de la API RESTful, en nuestro caso se ha decidido realizar el despliegue de ambas herramientas de forma simultánea y en varios contenedores, aunque con la posibilidad de conectarse entre ellos.

Existen dos requisitos para el despliegue de las herramientas. Por un lado, es necesaria la creación de una carpeta o de un volumen para dotar de persistencia a la base de datos, es decir, en caso de que por cualquier motivo se parase el contenedor de MongoDB, no perderíamos los datos. Y además, tener instalado *Docker* dentro del sistema en el que se vaya a realizar el despliegue.

Una vez creada la carpeta, es necesario movernos hasta la carpeta donde esté ubicado el proyecto de la API RESTful y crear un fichero Dockerfile como el que se muestra en la figura 52 mediante el cuál vamos a especificar las distintas opciones que va a tener la imagen que creemos a partir de nuestro API creado.

```
FROM node:latest
WORKDIR /app
COPY package.json /app
RUN npm install
COPY . /app
EXPOSE 3000
ENTRYPOINT ["node", "index.js"]
CMD ["npm", "start"]
```

Figura 52: Fichero dockerfile

En este fichero indicamos la imagen que hay que utilizar como base (node:latest). Para nuestra imagen, también se especifican los distintos paquetes de node.js que hay que instalar y en qué carpeta es necesario ubicarlos. Mediante el mandato **RUN npm install** se produce la instalación de dichos paquetes y con **EXPOSE 3000** indicamos el puerto en el cuál se va a poder acceder al servicio. Con el **ENTRYPOINT** decimos qué fichero va a ser nuestro "main". Por último, indicamos que se ejecute el comando **npm install** para poner en marcha el servicio.

Una vez tengamos creada la imagen de nuestro API REST, crearemos el fichero docker-compose [32] mediante el cuál realizaremos el despliegue conjunto de los dos servicios, tal y como se muestra en la siguiente figura 53.

```
services:  
  web:  
    build: .  
    ports:  
      - "3000:3000"  
    depends_on:  
      - mongo  
    restart: always  
  mongo:  
    image: mongo  
    ports:  
      - "27017:27017"  
    volumes:  
      - /Users/marcosgamazo/mongo-persistent:/data/db  
    command: [--auth]
```

Figura 53: Fichero docker-compose

A continuación se va a proceder a la explicación de cada línea del fichero:

- services: los distintos servicios que van a desplegarse mediante el docker-compose.
- web: servicio de la API.
- "build .": con el "." le indicamos al fichero que tiene que construir el contenedor en base a la imagen que se crea en esa carpeta.
- ports: puertos a exponer, a la izquierda colocamos el puerto privado y a la derecha el público.
- depends_on: con esto indicamos que es obligatorio que exista un contenedor previo de mongo antes de la creación del contenedor de la API.
- restart: en caso de que se produzca cualquier tipo de error con el contenedor en funcionamiento, docker correrá el contenedor de nuevo automáticamente.
- mongo: servicio de la base de datos.
- image: imagen a utilizar para el contenedor de la base de datos.
- volumes: carpeta en la que se ubica la persistencia de la base de datos.
- command: comando a ejecutar una vez creado el contenedor de la base de datos, en nuestro caso, permite que se activen las autenticaciones en las diferentes bases de datos creadas dentro de MongoDB.

Una vez tengamos creado el fichero docker-compose, podemos crear ambos contenedores e iniciarlos mediante el mandato **docker-compose up -d**. Con la opción '-d' indicamos que se active el modo *detached* y se ejecuten en segundo plano, como se muestra en la figura 54

4.2. Despliegue de las herramientas

```
 MacBook-Pro:~/Doc/API-Rest p master :1 !1 docker-compose up -d
Building web
Step 1/8 : FROM node:latest
--> d6740064592f
Step 2/8 : WORKDIR /app
--> Running in dcfc0bb5e2b79
Removing intermediate container dcfc0bb5e2b79
--> c67928b3d193
Step 3/8 : COPY package.json /app
--> 296a97988c8a
Step 4/8 : RUN npm install
--> Running in 22f27dfb8954
added 260 packages, and audited 261 packages in 32s

15 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 7.3.0 -> 7.4.3
npm notice Changelog: <https://github.com/npm/cli/releases/tag/v7.4.3>
npm notice Run `npm install -g npm@7.4.3` to update!
npm notice
Removing intermediate container 22f27dfb8954
--> c70d8c59512f
Step 5/8 : COPY . /app
--> 06004647c37f
Step 6/8 : EXPOSE 3000
--> Running in dba5fe8ce283
Removing intermediate container dba5fe8ce283
--> d0f882b04261
Step 7/8 : ENTRYPOINT ["node", "index.js"]
--> Running in 9da8e1422d17
Removing intermediate container 9da8e1422d17
--> d88eeef6c4a69
Step 8/8 : CMD ["npm", "start"]
--> Running in 39ebaee7aec6
Removing intermediate container 39ebaee7aec6
--> aac0e36041d9

Successfully built aac0e36041d9
Successfully tagged api-rest_web:latest
WARNING: Image for service web was built because it did not already exist. To rebuild this image you must use `docker-compose build` or `docker-compose up --build`.
Starting api-rest_mongo_1 ... done
Creating api-rest_web_1 ... done
```

Figura 54: Ejecución del docker-compose

Capítulo 5

Resultados

5.1. Manual de uso de la base de datos

A continuación, se enseñarán los pasos necesarios para una correcta utilización de la base de datos. Esta sección del documento está indicada para todo aquel que participe en el desarrollo del proyecto.

Para acceder a la consola de control de MongoDB es necesario localizar el identificador del contenedor de docker de Mongo dentro de la máquina donde se haya realizado el despliegue. Esto es posible mediante el comando **docker ps -a**, que listará todos los contenedores levantados en la máquina. Sólo es necesario recordar los tres primeros caracteres del contenedor. Una vez tengamos el identificador, introducimos el siguiente comando en un terminal, **docker exec -t 'identificador' bash**. Con el último parámetro indicamos que queremos abrir un terminal dentro del contenedor y como indicador podemos introducir los tres caracteres mencionados anteriormente. Esto provocará que cambie el prompt de la consola al prompt del contenedor. Posteriormente introducimos el comando **mongo** que abrirá el cliente de línea de comandos de mongo.

Primeramente tenemos que acceder a la base de datos con la que queremos interactuar, esto se consigue mediante el comando **use SonidosDelCielo**.

Después introducimos los credenciales necesarios de la siguiente forma:
db.auth('usuario','contraseña'). Esta operación retornará 1 en caso de ser correcta o 0 en cualquier otro caso.

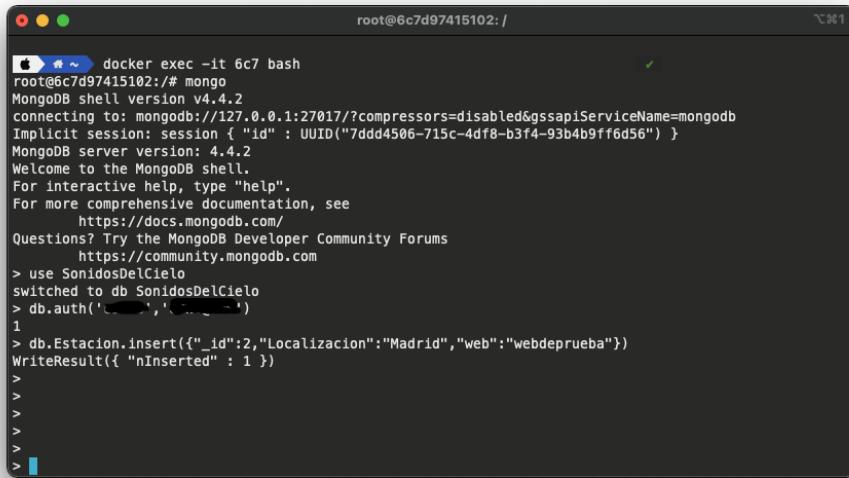
Una vez estemos logueados dentro de la base de datos, podremos realizar varios tipos de operaciones.

5.1.1. Inserción en colección

Cuando ya estemos dentro del terminal de mongo, tenemos que introducir los siguientes comandos:

- **db.collection.insert(documento)**: es necesario sustituir el parámetro <collection>por cualquier colección que esté declarada previamente, en nuestro caso (Eco, Estacion, Usuario, Espectrograma, Curva Luz, Sonido y Clasificacion) y como documento se introduce un fichero en formato JSON.

En caso de producirse una inserción correcta, retornará un JSON con el par clave-valor {"nInserted", <número>}, siendo número la cantidad de documentos insertados correctamente. En la siguiente imagen 55 se puede ver el proceso completo de inserción correcta.



```
root@6c7d97415102:/# mongo
MongoDB shell version v4.4.2
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("7ddd4506-715c-4df8-b3f4-93b4b9ff6d56") }
MongoDB server version: 4.4.2
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
    https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
    https://community.mongodb.com
> use SonidosDelCielo
switched to db SonidosDelCielo
> db.auth('...', '...')
1
> db.Estacion.insert({"_id":2,"Localizacion":"Madrid","web":"webdeprueba"})
WriteResult({ "nInserted" : 1 })
>
>
>
>
> |
```

Figura 55: Inserción en el terminal de MongoDB

5.1.2. Obtención de un documento

Para la obtención del documento, una vez estemos ubicados y logueados dentro de la base de datos indicada, solamente será necesario introducir el siguiente comando: **db.getCollection('Colección').find(JSON)**.

Dentro del JSON podemos colocar la cantidad de filtros que consideremos necesarios. En este caso hemos introducido el filtro de Localizacion dentro de la colección Estacion, como podemos ver en la imagen 56. Si hay algún documento que satisfaga esos filtros, se retornará un JSON con los diferentes documentos, en caso de no existir ninguno, el retorno estará vacío.

```
> db.getCollection('Estacion').find({"Localizacion":"Madrid"})
{ "_id" : 2, "Localizacion" : "Madrid", "web" : "webdeprueba" }
> |
```

Figura 56: Obtención de documento en el terminal de MongoDB

5.1.3. Modificación de un documento

Para realizar la actualización de un documento en MongoDB es muy sencilla, simplemente tenemos que introducir el comando **db.collection.updateOne(filter, update)** con los filtros que queramos y como update un documento JSON con los diferentes parámetros a añadir o modificar, ya que en este tipo de bases de datos no es necesa-

Resultados

rio que todos los documentos tengan la misma estructura, podemos ver un ejemplo de actualización en la figura 57.

```
> db.Estacion.updateOne({"Localizacion":"Madrid"}, {$set:{ "Localizacion": "Alcorcon" }})  
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Figura 57: Modificación de documento en el terminal de MongoDB

MongoDB también nos permite actualizar varios documentos que cumplan con los filtros introducidos a través del comando **db.collection.updateMany(filter, update)**

5.1.4. Eliminación de un documento

En caso de querer eliminar todos los documentos existentes dentro de la colección, introduciremos el comando **db.Coleccion.deleteMany()** y como parámetro un JSON vacío. Si sólo queremos eliminar un documento debemos introducir el mandato **db.Coleccion.deleteOne(JSON)** y dentro del JSON pasaremos los filtros adecuados para la eliminación del documento, como se muestra en la figura 58.

```
> db.Estacion.deleteOne({"Localizacion":"Madrid"})  
{ "acknowledged" : true, "deletedCount" : 1 }  
>
```

Figura 58: Eliminación de un documento en el terminal de MongoDB

5.2. Manual de uso de la API

Con el desarrollo de la API RESTful se incluyó la herramienta Swagger. Esta herramienta permite realizar pruebas sobre los distintos recursos implementados. En nuestro caso se han implementado 4 operaciones sobre cada recurso existente (Eco, Estación, Curva de Luz, Espectrograma, Clasificacion y Usuario) como se ha mostrado en la figura 31. Estas cuatro operaciones son:

1. Get: obtención de un recurso.
2. Put: creación de un nuevo recurso.
3. Patch: modificación sobre un recurso existente. Utilizamos la operación Patch sobre Put debido al funcionamiento de MongoDB. Si utilizamos PUT, MongoDB nos obliga a modificar el id del recurso.
4. Delete: eliminación de un documento.

Vamos a mostrar cómo se realiza cada una de las anteriores operaciones sobre la interfaz de swagger. Además gracias a la utilización de la herramienta, nos brinda el comando equivalente para utilizar desde un terminal.

5.2.1. Utilización del método GET

Cada uno de los recursos se ha dotado de 2 operaciones GET. La primera de ellas y sin la introducción de un identificador nos devuelve un documento JSON con todos los elementos disponibles que existan. En cambio, si introducimos un identificador como parámetro, obtendremos un solo elemento, como se muestra en la siguiente figura 59.

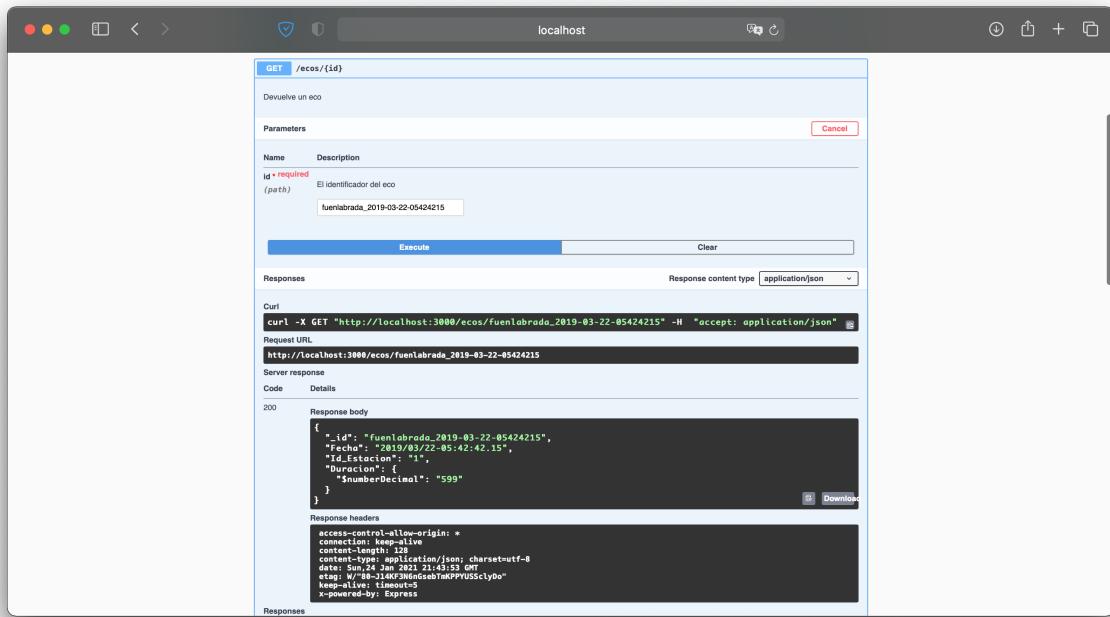


Figura 59: Obtención de un recurso por medio de Swagger

Se observa cómo la herramienta nos proporciona el código que devuelve el response (200 en este caso) y un JSON con el recurso.

5.2.2. Utilización del método POST

Para el uso del método POST se ha implementado una vista de cada uno de los recursos de forma que es visible en todo momento el formato que debe seguir el JSON. En este ejemplo vamos a introducir una nueva estación para lo cual se especifica que el formato a seguir es el siguiente:

```

1 {
2   "_id": "string",
3   "Localizacion": "string",
4   "web": "string"
5 }
```

Para realizar una inserción, simplemente se realiza la modificación del JSON y pulsamos el botón *Execute*. Puesto que MongoDB no es estricto en el formato de los documentos, podemos introducir nuevos campos o incluso reducir los mismo siempre y cuando exista un '_id', tal y como se ve en la figura 60.

Resultados

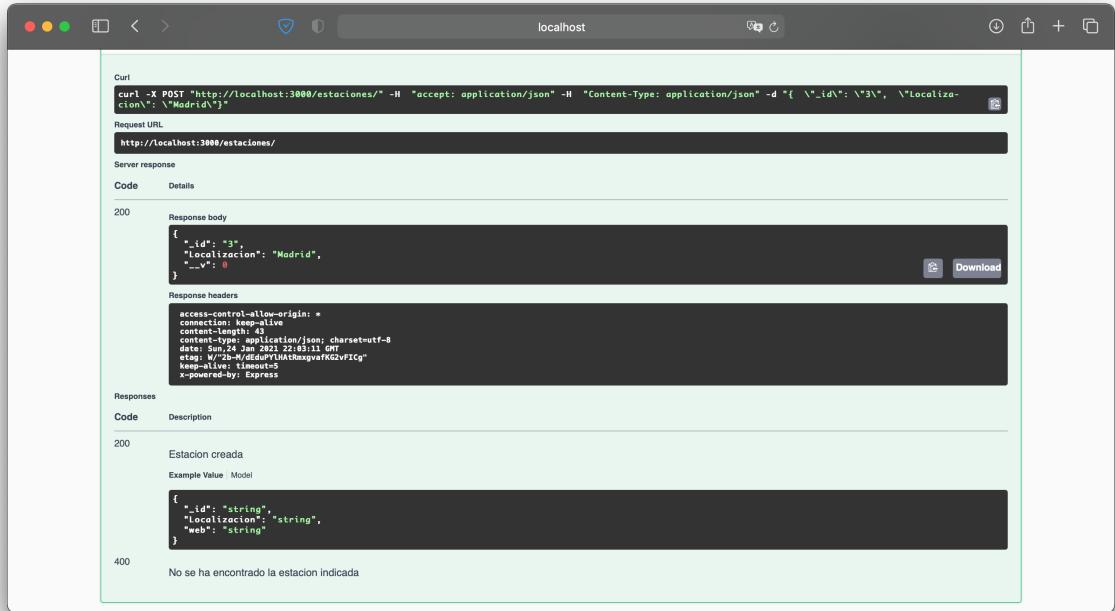


Figura 60: Inserción de un recurso por medio de Swagger

En este caso el código de respuesta es un 200 con el texto "Estacion creada".

Puesto que dentro del proyecto Sonidos del Cielo existen ciertas tareas que se realizan de manera periódica, se ha desarrollado un pequeño programa en lenguaje Python, que sirve para la inserción automática de los elementos que se generan a través del programa Echoes (ecos, espectrogramas, curva de luz) y sonidos generados mediante el programa desarrollado por el equipo del CSLab. El programa se encarga de leer un fichero csv, selecciona las columnas necesarias para cada elemento y tanta peticiones POST como líneas existan en el programa. En la siguiente figura 61 se muestra uno de los métodos utilizados

```
import pandas as pd
import requests
import json

def cargarSonidos():
    url= 'http://localhost:3000/sonidos/'
    df = pd.read_csv('/Users/marcosgamazo/Downloads/muestras_mayo.csv', sep=",", usecols=['ID','lnk_snd'])
    df.columns = ["_id","Ruta"]
    data = df.to_dict("records") #Formato del json a añadir {"_id":"Fuenlabrada....","Ruta":"/home/...."}
    headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
    for key in data:
        #Realizamos el Post a través de la API
        r=requests.post(url,json.dumps(key),headers=headers)
        print(r.status_code)
    return
```

Figura 61: Inserción a través del script en Python

De esta forma y mediante un comando cron, es posible añadir todos los elementos que se detecten diariamente.

5.2.3. Utilización del método Patch

Para utilizar este método es necesario introducir como parámetro el id del recurso a modificar y posteriormente dentro del body las modificaciones pertinentes que se deseen realizar. En caso de introducir otro id, el programa lo eliminará y realizará los cambios sobre el id que se ha pasado como *Path Param*. El proceso de modificación es similar al de inserción, se muestra un JSON como referencia al modelo que debemos introducir y al igual que en el método anterior, podemos añadir, eliminar o actualizar datos de un recurso existente.

5.2.4. Utilización del método Delete

Para la eliminación de un recurso, simplemente lo único que necesitamos es el identificador del recurso que queremos borrar. En caso de que el recurso exista, la API retornará un response con código 200 y el elemento que existía con ese identificador antes de ser eliminado, como se muestra en la figura 62

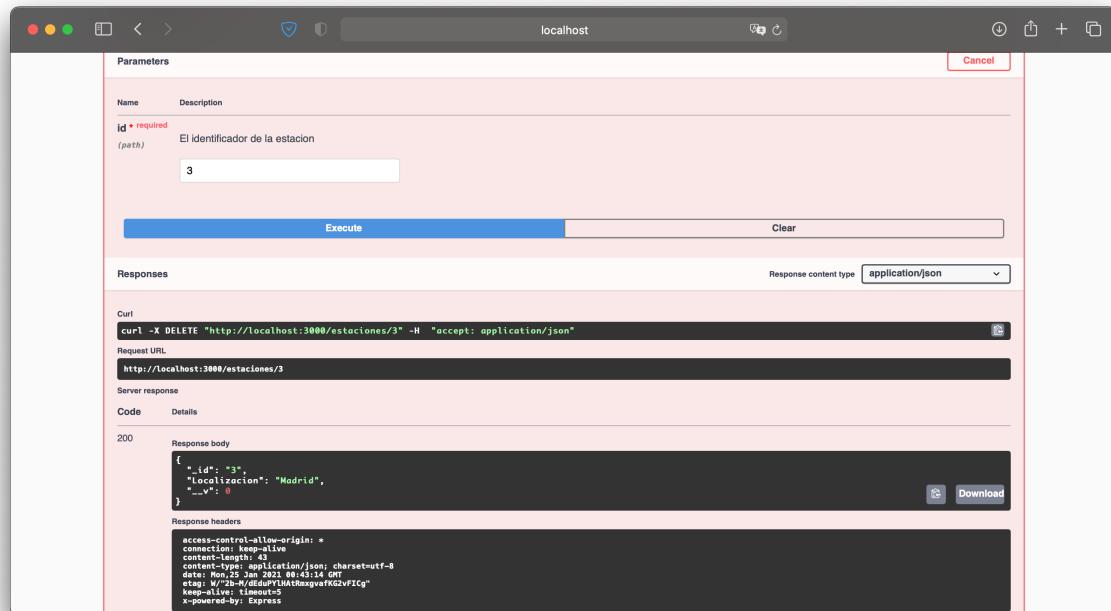


Figura 62: Eliminación de un recurso por medio de Swagger

5.3. Manual de uso del Chatbot

El chatbot se ha desarrollado para corroborar el correcto funcionamiento de la API RESTful. A continuación se explicará paso a paso y con imágenes la utilización del mismo. Una vez se hayan puesto en funcionamiento ambos servidores de Rasa

En primer lugar en la página principal de la web, tenemos un pequeño círculo en la esquina inferior izquierda con un efecto de parpadeo. Una vez pulsemos en el ícono en el que aparece un pájaro, aparecerá un cuadro en el que se puede introducir texto. Para que el chatbot comience a funcionar es necesario que introduzcamos un saludo como se muestra en la imagen 63.

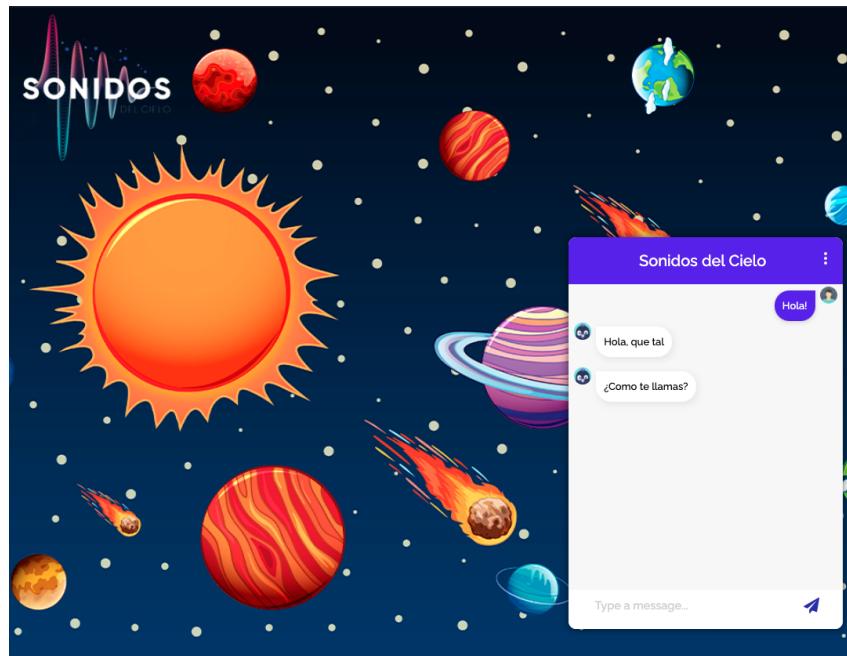


Figura 63: Comienzo de una conversación con el chatbot

Después el chatbot preguntará el nombre al usuario, en caso de proporcionarlo, se utilizará esta información para almacenar la clasificación. Posteriormente el chatbot mostrará las distintas opciones disponibles, puesto que está enfocado a un público infantil. Se mostrarán cuatro botones para que no sea necesario el uso de la escritura para la interacción con el chatbot. Estos cuatro botones contienen las diferentes opciones a realizar en el chatbot tal y como se ve en la figura 64

- Tutorial: Muestra una breve explicación sobre el proceso que tiene que seguir el usuario para clasificar un eco.
- ¿Qué es el proyecto Sonidos del Cielo?: proporciona un enlace para acceder a la página web del proyecto.
- Clasificar: para iniciar la clasificación de un Eco.
- Salir: finaliza el programa.

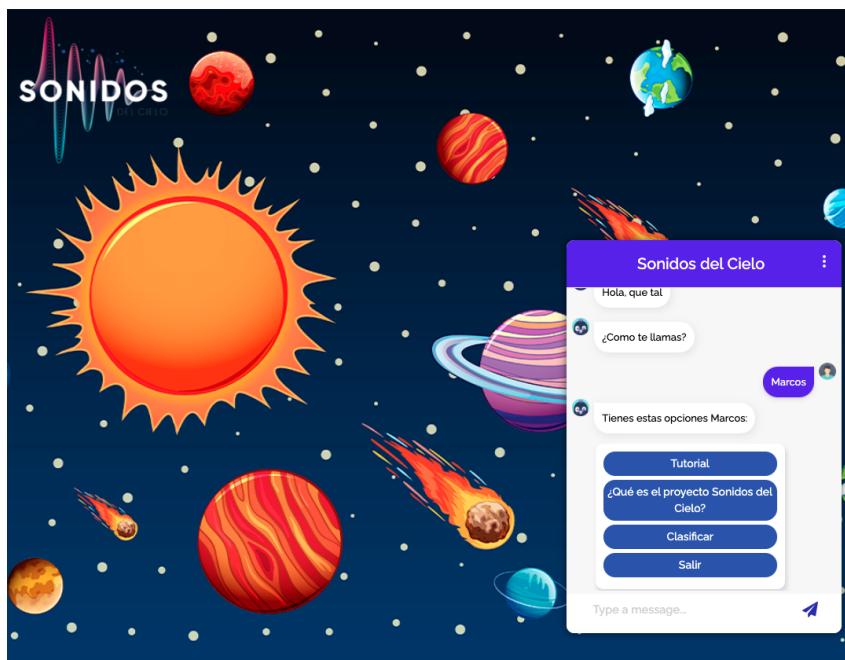


Figura 64: Muestra de opciones del chatbot

Una vez pulsemos en la opción de **Clasificar** se mostrará un reproductor de audio con el sonido del eco y una primera pregunta utilizada para la clasificación.

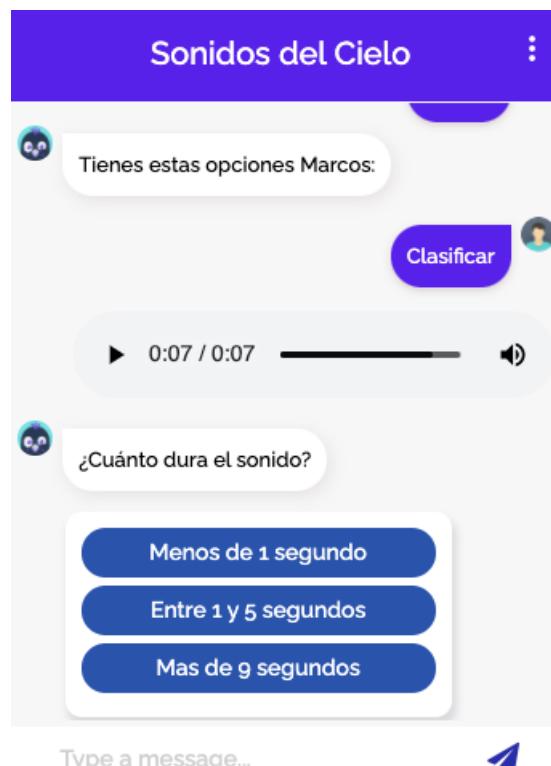


Figura 65: Muestra de sonido y primera pregunta

Resultados

Una vez respondamos a la pregunta, se nos mostrarán dos preguntas más de la misma forma que se observa en la figura 65. Cuando se contesten las tres preguntas, el chatbot internamente llamará a un action personalizado que realizará la petición POST a la API y de esta forma se almacenará la clasificación realizada por el usuario. Una vez acabado, volverá a mostrar las distintas opciones del chatbot (figura 63) para que en caso de que el usuario quiera, pueda clasificar más sonidos.

Capítulo 6

Conclusiones

Con la realización de este proyecto de fin de grado se ha tratado de iniciar un proceso mediante el cual se pueda acercar la ciencia a gente que normalmente no trabaja con ella.

En general, el resultado final de este Trabajo de Fin de Grado es muy positivo, ya que aun tratándose de un proyecto que ha comenzado desde cero y de largo plazo, se ha conseguido desarrollar una base en la que los diferentes miembros del equipo puedan apoyarse para futuros desarrollos. Gracias a este Trabajo de Fin de Grado he podido conocer nuevas herramientas de trabajo, lo que me ha llevado a adquirir nuevos conocimientos tanto en lenguajes de programación que no había utilizado (JavaScript) como en el uso de herramientas nuevas, sobretodo Docker, Node.js, Swagger y MongoDB. También he reforzado conocimientos adquiridos durante el desarrollo de mi grado en Ingerniería Informática como los sistemas orientados a servicios, la ingeniería del software, las bases de datos, además de poder ver cómo se realiza la planificación de un proyecto, y cómo evoluciona el desarrollo y el trabajo en común de un equipo. Una parte fundamental de este trabajo es gracias a Raquel, por darme plena libertad en la organización y por guiarme a encontrar soluciones siempre que lo he necesitado. También recordar a todos mis compañeros de equipo que han salido en mi ayuda siempre que la he necesitado. Por otro lado, la realización de este Trabajo de Fin de Grado sirve como base para futuras herramientas entabladas dentro del proyecto Sonidos del Cielo. Lo que quiero recalcar con esto es que las herramientas desarrolladas ahora mismo sufrirán mejoras, evoluciones, para que adquieran nuevas capacidades y se puedan cumplir nuevos requisitos y metas establecidas en el proyecto. Espero que el desarrollo de este trabajo sirva para acercar la ciencia a todas aquellas personas que habitualmente no tienen trato con ella, personas con discapacidad y especialmente a niños ya que considero de gran importancia despertar en ellos desde jóvenes la pasión y el deseo de trabajar con la ciencia.

Bibliografía

- [1] "Encuesta sobre equipamiento y uso de tecnologías de información y comunicación en los hogares." https://www.ine.es/prensa/tich_2019.pdf, 2019.
- [2] A. de astronomía de Fuenlabrada, "Aula de astronomía fuenlabrada." <https://www.auladeastronomiadefuenlabrada.com/enlaces-de-interes/grupo-kepler/>, 2016.
- [3] W. Kaufmann, "Radio meteor reflection geometrys." <http://www.ars-electromagnetica.de/robs/>, 2019.
- [4] T. Ulversoy, "Software defined radio: Challenges and opportunities," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 4, pp. 531–550, 2010.
- [5] RTL-SDR, "Echoes: An rtl-sdr tool for meteor scatter detection." <https://www rtl-sdr.com/tag/echoes/>, 2018.
- [6] S. L. Perera, "Contadores de estrellas - server side - echoes." <https://github.com/cslab-upm/Echoes-stream-generator>, 2018.
- [7] C. Denis, "Dialogflow: la herramienta de google para la creación de chatbots." <https://www.makingscience.com/blog/dialogflow-la-herramienta-de-google-para-la-creacion-de-chatbots/>, 2019.
- [8] E. F. Comendeiro, "Información sobre desarrollo de bots de microsoft." <https://planetachatbot.com/informacion-para-articulo-de-bots-de-microsoft-943a25eddd5>, 2017.
- [9] R. de Juana, "Ibm watson: casi todo lo que tienes que saber." <https://www.muycomputerpro.com/2019/09/24/ibm-watson-casi-todo-lo-que-tienes-que-saber>, 2019.
- [10] A. F. B. H, "Chatbot ai." <https://github.com/ahmadfaizalbh/Chatbot>, 2020.
- [11] O. P. de la Salud, "La oms lleva la información de la covid-19 a millones a través de whatsapp." https://www.paho.org/hq/index.php?option=com_content&view=article&id=15761:la-oms-lleva-la-informacion-de-la-covid-19-a-millones-a-traves-de-whatsapp?Itemid=1926&lang=es, 2020.
- [12] M. Arthur, "New mental health apps reviewed: Wysa, woebot, joyable, and talkspace." <https://bevoya.com/blog/mental-health-app-wysa-woebot-joyable-talkspacereview>, 2018.

- [13] C. Daly, "Klm: Chatbots are the future of customer support." https://aibusiness.com/document.asp?doc_id=760517, 2018.
- [14] FinTech, "Eno, un chatbot que interactua mediante emoticonos." <https://www.fin-tech.es/2017/03/capital-one-lanza-chatbot-eno.html>, 2017.
- [15] E. Imparcial, "Burger king comienza a recibir pedidos por whatsapp." <https://www.elimparcial.com/tecnologia/Burger-King-comienza-a-recibir-pedidos-por-WhatsApp--20200629-0158.html>, 2020.
- [16] C. L. Sánchez Rodas and D. Monsalve, "Historia y evolución de las bases de datos," 2016.
- [17] S. L. Perera, "Contadores de estrellas - client side - echoes watcher." <https://github.com/cslab-upm/Echoes-watcher>, 2018.
- [18] M. Singh, M. Rajan, V. Shivraj, and P. Balamuralidhar, "Secure mqtt for internet of things (iot)," in *2015 Fifth International Conference on Communication Systems and Network Technologies*, pp. 746–751, IEEE, 2015.
- [19] David, "Liquidsoap: Un paso más para icecast." <https://www.ochobitshacenunbyte.com/2019/08/28/liquidsoap-un-paso-mas-para-icecast/>, 2019.
- [20] G. Rossum, "Python reference manual," 1995.
- [21] J. Ordóñez, "¿qué es una api rest?." <https://www.idento.es/blog/desarrollo-web/que-es-una-api-rest/>, 2019.
- [22] M. Contributors, "Javascript." <https://developer.mozilla.org/es/docs/Web/JavaScript>, 2020.
- [23] A. Medina, "Desarrollar una aplicación de escritorio en javascript: muy fácil con electron." <https://www.alexmedina.net/aplicacion-escritorio-javascript-electron/>, 2020.
- [24] L. Torvalds, J. Hamano, *et al.*, "Git," *Software Freedom Conservancy*, 2005.
- [25] S. Team, "Swagger documentation." <https://swagger.io/docs>.
- [26] D. Souza, "Documenting your express api with swagger." <https://blog.logrocket.com/documenting-your-express-api-with-swagger/>, 2020.
- [27] D. Baltar, "Automatic api documentation in node.js using swagger." <https://medium.com/swlh/automatic-api-documentation-in-node-js-using-swagger-dd1ab3c78284>, 2020.
- [28] R. Hewage, "Step by step guide to dockerize a node.js express application." <https://openwebinars.net/blog/docker-que-es-sus-principales-caracteristicas/>.
- [29] C. Lopez, "Docker básico - primeros pasos." <https://medium.com/@jclopex/docker-básico-primeros-pasos-c57144d18eea>, 2019.
- [30] RASA, "Using custom spacy components in rasa." <https://blog.rasa.com/custom-spacy-components/>, 2020.

BIBLIOGRAFÍA

- [31] S. Luján-Mora, “Html5: de html4 a html5,” *Estándares Web*, 2011.
- [32] Docker, “Networking in compose.” <https://docs.docker.com/compose/networking/>, 2020.