



ugr | Universidad
de Granada

TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

Agente conversacional multiedad

Agente conversacional adaptable

Autor

Mario Carmona Segovia

Directora

Rocío Celeste Romero Zaliz



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, julio de 2022

Agente conversacional multiedad: Agente conversacional adaptable

Mario Carmona Segovia

Palabras clave: Agentes conversacionales, Aprendizaje automático, Transformers, BlenderBot, SCRUM, Arquitectura MVC, Heroku, Responsive, FastAPI.

Resumen

Este proyecto tiene como objetivo crear de forma automatizada un agente conversacional para “La noche europea de los investigadores” (ERN). Deberá poder contestar a las preguntas que le realicen los asistentes, adaptándose a sus edades y conocimientos de forma interactiva.

Para el desarrollo de este proyecto se ha construido un sistema con el que poder hacer uso del chatbot obtenido del entrenamiento. Además, se ha elaborado toda una estructura de programas para obtener información, entrenar a los modelos con esa información y finalmente desplegar estos modelos en el sistema mencionado anteriormente.

Para la adaptación del chatbot a distintos perfiles de usuario se ha elaborado un módulo que es capaz de deducir la edad de un usuario de forma explícita o de manera implícita a través de imágenes.

Multi-age conversational agent: Teaching-focused conversational agent

Mario Carmona Segovia

Keywords: Conversational Agents, Machine Learning, Transformers, BlenderBot, SCRUM, MVC Architecture, Heroku, Responsive, FastAPI.

Abstract

This project aims to create an automated conversational agent for “The European Researchers’ Night” (ERN). It should be able to answer the questions asked by the attendees, adapting to their ages and knowledge in an interactive way.

For the development of this project, a system has been built to make use of the chatbot obtained from the training. In addition, a whole structure of programs has been developed to obtain information, train the models with this information and finally deploy these models in the aforementioned system.

In order to adapt the chatbot to different user profiles, a module has been developed that is able to deduce the age of a user explicitly or implicitly through images.

Yo, **Mario Carmona Segovia**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 45922466E, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Mario Carmona Segovia

Granada a 7 de julio de 2022.

D. **Rocío Celeste Romero Zaliz**, Profesora del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Agente conversacional multiedad*, *Agente conversacional adaptable*, ha sido realizado bajo su supervisión por **Mario Carmona Segovia**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 7 de julio de 2022 .

La directora:

Rocío Celeste Romero Zaliz

Agradecimientos

En primer lugar, me gustaría dar las gracias a la Universidad de Granada y a toda su comunidad estudiantil que ha hecho del tránsito de la carrera toda una experiencia que me ha convertido en la persona que soy a día de hoy. Por supuesto, me gustaría hacer especial énfasis en todos aquellos amigos de la carrera, Dani, Sergio, Óscar, Paco, Víctor, Pedro y un sin fin más; con lo que he compartido los momentos más felices y también los más tristes, y sin olvidar todos los momentos graciosos que hemos vivido, ya que cuatro años dan para muchas experiencias. No me gustaría dejar por alto a mi querida tutora Rocío, la cual me ha dado mucha confianza a lo largo de todo el proyecto y me ha apoyado en todos aquellos momentos donde el proyecto se ha puesto cuesta arriba.

Y finalmente, está claro que un apoyo fundamental ha sido toda mi familia, la cual no solamente me ha apoyado emocionalmente, sino que si ha hecho falta me ha ayudado con ideas para avanzar en el proyecto.

Índice general

| | |
|--|-----------|
| 1. Introducción | 21 |
| 1.1. Motivación y objetivos | 22 |
| 1.2. Descripción del proyecto | 22 |
| 1.3. Estructura de la memoria | 23 |
| 2. Estado del Arte | 25 |
| 2.1. Análisis del Estado del Arte | 25 |
| 2.2. Herramientas para el desarrollo del chatbot | 34 |
| 2.3. Comparativa de las herramientas | 36 |
| 2.4. Elección de la herramienta | 36 |
| 3. Análisis | 39 |
| 3.1. Metodología de desarrollo | 39 |
| 3.1.1. Tipos de metodologías | 39 |
| 3.1.2. Comparativa de las metodologías | 44 |
| 3.2. Aplicación de Scrum al proyecto | 44 |
| 3.3. Historias de usuario | 44 |
| 3.3.1. Tarjetas de historias de usuario | 45 |
| 3.3.2. Listado de las historias de usuario | 47 |
| 3.4. Requisitos adicionales | 47 |
| 3.5. Análisis de riesgos | 48 |
| 3.6. Riesgos materializados | 48 |
| 4. Planificación | 51 |
| 4.1. Planificación de los sprints | 51 |
| 4.2. Planificación temporal | 53 |
| 5. Gestión | 55 |
| 5.1. Gestión del código y de la documentación | 55 |
| 5.2. Gestión de recursos | 56 |
| 5.2.1. Recursos humanos | 56 |
| 5.2.2. Recursos materiales para el desarrollo | 56 |
| 5.2.3. Recursos software | 57 |
| 5.2.4. Recursos de comunicación y documentación | 60 |

| | | |
|-----------|--|-----------|
| 5.3. | Gestión de costes | 60 |
| 5.3.1. | Costes de recursos humanos | 60 |
| 5.3.2. | Costes de recursos materiales | 61 |
| 5.3.3. | Costes de recursos software | 63 |
| 5.3.4. | Costes de recursos de comunicación y documentación . | 63 |
| 5.3.5. | Costes adicionales | 63 |
| 5.4. | Presupuesto total | 64 |
| 6. | Diseño | 67 |
| 6.1. | Arquitectura Modelo-Vista-Controlador | 67 |
| 6.2. | Diseño de la Vista | 68 |
| 6.3. | Diseño del Controlador | 72 |
| 6.4. | Diseño del Modelo | 74 |
| 7. | Implementación | 79 |
| 7.1. | Implementación de la Vista | 79 |
| 7.1.1. | Herramientas para el desarrollo | 79 |
| 7.1.2. | Interfaz responsive | 79 |
| 7.1.3. | Modo oscuro | 80 |
| 7.1.4. | Menú lateral | 80 |
| 7.1.5. | Ventanas emergentes | 80 |
| 7.1.6. | Interfaces web | 81 |
| 7.1.7. | Interfaces de Telegram | 81 |
| 7.2. | Implementación del Controlador | 81 |
| 7.2.1. | Herramientas para el desarrollo | 81 |
| 7.2.2. | Implementación del servidor | 82 |
| 7.2.3. | Despliegue del servidor | 83 |
| 7.2.4. | Gestión del sitio web | 84 |
| 7.2.5. | Secciones adicionales del servidor | 85 |
| 7.2.6. | Procesado de las imágenes | 87 |
| 7.2.7. | Gestión de la conversación | 88 |
| 7.2.8. | Guardado de la conversación en el log | 88 |
| 7.3. | Implementación del Modelo | 89 |
| 7.3.1. | Herramientas para el desarrollo | 89 |
| 7.3.2. | Implementación del servidor | 90 |
| 7.3.3. | Despliegue del servidor | 90 |
| 7.3.4. | Generación de los conjuntos de datos | 91 |
| 7.3.5. | Entrenamiento de modelos | 91 |
| 7.3.6. | Generación de respuestas | 93 |
| 7.3.7. | Deducción de edades | 94 |

| | |
|---|------------|
| 8. Pruebas | 97 |
| 8.1. Pruebas sobre la interfaz web | 97 |
| 8.2. Pruebas sobre la deducción de edad | 98 |
| 8.3. Pruebas sobre la generación de respuestas | 101 |
| 9. Conclusiones y posibles ampliaciones | 105 |
| 9.1. Posibles ampliaciones | 107 |
| A. Características de las herramientas analizadas | 109 |
| A.1. AIML | 109 |
| A.2. IBM Watson | 113 |
| A.3. Google Dialogflow | 116 |
| A.4. Rasa Stack | 119 |
| A.5. GPT-3 | 121 |
| A.6. BlenderBot | 124 |
| B. Detalles de la implementación | 127 |
| B.1. Definición de las interfaces responsive | 127 |
| B.2. Funcionamiento del interruptor de cambio de tema | 128 |
| B.3. Implementación del despliegue del menú lateral | 130 |
| B.4. Inclusión de las ventanas emergentes en el sistema | 130 |
| B.5. Inclusión de la interfaz web de Dialogflow en el sistema | 134 |
| B.6. Inclusión de la interfaz de Telegram de Dialogflow en el sistema | 136 |
| B.7. Razones adicionales de la elección del framework para el servidor | 138 |
| B.8. Despliegue del servidor en Heroku | 139 |
| B.9. Proceso de generación de plantillas renderizadas | 141 |
| B.10. Paso de la información a los script de Javascript | 142 |
| B.11. Solución a la inactividad de las apps de Heroku | 142 |
| B.12. Proceso para la gestión de la conversación desde el Controlador | 143 |
| B.13. Establecimiento de la conexión entre el servidor y el log | 144 |
| B.14. Cabeceras añadidas a las conexiones con el servidor del Modelo | 145 |
| B.15. Proceso de despliegue del servidor del Modelo mediante Ngrok | 145 |
| B.16. Preprocesado de los conjuntos de datos | 146 |
| B.17. Extracción de información de Quora | 147 |
| C. Manual de usuario | 149 |
| C.1. Instalación | 149 |
| C.1.1. Instalación del servidor de la Vista y el Controlador . | 149 |
| C.1.2. Instalación del servidor del Modelo | 149 |
| C.2. Inicio del sistema | 150 |
| C.3. Generación de los conjuntos de datos de entrenamiento | 150 |
| C.4. Entrenamiento de los modelos | 151 |
| Glosario | 151 |

Índice de figuras

| | |
|---|-----|
| 2.1. Estimación de la inversión en IA en el campo de las telecomunicaciones | 26 |
| 2.2. Historial de las publicaciones sobre temas relacionados con chatbots | 27 |
| 2.3. Estructura de los chatbots actuales | 28 |
| 2.4. Estructura de los chatbots basados en intenciones | 30 |
| 3.1. Etapas de la metodología clásica | 40 |
| 3.2. Etapas de la metodología ágil | 42 |
| 3.3. Sprint de la metodología Scrum | 43 |
| 5.1. Planes de pago sobre las apps de Heroku | 58 |
| 5.2. Planes de pago sobre los dynos de Heroku | 58 |
| 5.3. Planes de pago sobre las bases de datos de Heroku | 59 |
| 6.1. Ciclo de vida de MVC | 69 |
| 6.2. Sitemap de la Vista | 70 |
| 6.3. Iconos usados en el sitio web | 71 |
| 6.4. Wireframes de la Vista | 73 |
| 6.5. Diagrama de actividad del acceso a la interfaz del chatbot | 75 |
| 6.6. Diagrama de actividad de la gestión de la conversación | 76 |
| 6.7. Diagrama de actividad de la gestión del procesado de la imagen | 77 |
| 7.1. Ejemplo de definición de sección del servidor del Controlador | 83 |
| 8.1. Imágenes de las caras usadas para las pruebas | 100 |
| A.1. Cuentas disponibles en Pandorabots | 112 |
| A.2. Editor de IBM Watson | 115 |
| B.1. Interfaz responsive usando el portátil | 128 |
| B.2. Interfaz responsive usando la tablet en orientación vertical | 129 |
| B.3. Estados del menú lateral | 131 |
| B.4. Botones para el acceso a los canales de comunicación con el chatbot | 132 |

| | |
|--|-----|
| B.5. Ventanas emergentes de la Vista | 133 |
| B.6. Apartado Integrations de un agente de Dialogflow | 134 |
| B.7. Sección Text based del apartado Integrations | 135 |
| B.8. Ventana emergente de Web Demo de Dialogflow | 135 |
| B.9. Ventana donde se muestra el código del iframe | 136 |
| B.10. Código del iframe del chatbot para adultos | 136 |
| B.11. Aspecto de la interfaz web del chatbot para adultos desde una tablet en orientación vertical | 137 |
| B.12. Ventana emergente de Telegram de Dialogflow | 137 |
| B.13. Chat con el bot de Telegram BotFather | 138 |
| B.14. Aspecto de la interfaz de Telegram del chatbot para adultos . | 138 |
| B.15. Número de respuestas por segundo para peticiones (Datos del 02/08/2021) | 139 |
| B.16. Métodos de despliegue de apps en Heroku | 139 |
| B.17. Ejemplo de definición de una clave en una plantilla HTML . | 142 |

Índice de cuadros

| | |
|---|-----|
| 2.1. Comparativa de pros y contras de las herramientas de desarrollo | 37 |
| 2.2. Comparativa de características de las herramientas de desarrollo | 37 |
| 3.1. Comparativa de las metodologías de desarrollo | 44 |
| 3.2. Historia de Usuario - HU.1 | 45 |
| 3.3. Historia de Usuario - HU.2 | 45 |
| 3.4. Historia de Usuario - HU.3 | 45 |
| 3.5. Historia de Usuario - HU.4 | 46 |
| 3.6. Historia de Usuario - HU.5 | 46 |
| 3.7. Historia de Usuario - HU.6 | 46 |
| 3.8. Listado de historias de usuario junto con su prioridad | 47 |
| 3.9. Tabla de riesgos del proyecto | 48 |
| 3.10. Matriz probabilidad-impacto de riesgos | 49 |
| 4.1. Diagrama de Gantt sobre la planificación temporal | 53 |
| 5.1. Costes asociados a los recursos humanos | 62 |
| 5.2. Costes asociados a los recursos materiales | 63 |
| 5.3. Costes asociados a los recursos adicionales | 64 |
| 5.4. Presupuesto total del proyecto | 65 |
| 8.1. Puntuación de accesibilidad de la interfaz web | 98 |
| 8.2. Pruebas de traducción con la API de DeepL | 101 |
| 8.3. Primera conversación de las pruebas del módulo de generación de respuestas | 102 |
| 8.4. Segunda conversación de las pruebas del módulo de generación de respuestas | 103 |
| B.1. Restricciones de tamaño de pantalla para los dispositivos . . . | 127 |

Capítulo 1

Introducción

La motivación de los seres humanos por difundir los conocimientos existentes a la mayor cantidad de personas posible ha existido desde que el ser humano es ser humano. Por supuesto, en sus inicios esta difusión se producía a una menor escala, principalmente a una escala local, de forma oral y a un ritmo muy lento. Además, eran conocimientos básicos, útiles sobre todo para la realización de las tareas diarias.

Conforme ha ido evolucionando el ser humano, esta difusión de conocimiento ha evolucionado junto a él. Por ejemplo, con hitos como la invención de la escritura, la difusión pudo aumentar su ritmo, tener un rango de alcance un poco mayor, y sobre todo posibilitó difundir conocimientos más complejos.

Aunque la evolución de la difusión de conocimiento se ha ido produciendo paulatinamente, el ritmo al que lo ha hecho no ha sido algo destacable, ya que siempre han existido dos grandes impedimentos. Por un lado, la falta de educación en la mayor parte de la población, lo que provoca la ausencia de motivación por aprender. Y, por otro lado, la dificultad de difusión, puesto que a pesar de haber tenido algunos avances gracias a hitos en la historia de la humanidad, tales como la invención de la escritura, mencionado anteriormente, o el aumento del tráfico mundial.

La falta de educación en la población se ha ido disipando en las últimas décadas, al hacer más fácil el acceso a los conocimientos.

El hito que si supuso un boom en la difusión de conocimiento fue la creación de la World Wide Web. Este hito no solamente supuso un avance enorme en el aumento del tráfico de información mundial, sino también hizo aún más fácil el acceso de cualquier persona al conocimiento, sobre todo cuando se extendió el uso de ordenadores personales.

Después del último hito se ha llegado a tal punto de difusión, que el aumento de la difusión ha pasado a un segundo plano y ahora los esfuerzos se están centrándo en hacer que la difusión sea más natural para aquellas personas que consultan la información.

Una de las posibles ramas para difundir conocimiento de forma natural es mediante la conversación con un bot, o también llamados chatbots.

1.1. Motivación y objetivos

El potencial que tienen los chatbots como herramienta para la difusión de conocimientos de modo natural ha motivado la creación de este proyecto.

Además de lo anteriormente dicho, también ha motivado el desarrollo de este proyecto, la posible aplicación del mismo para los eventos de "La Noche Europea de l@s Investigador@s", en los cuales se realizan distintas actividades culturales. Una de estas actividades es el fomento sobre el mundo de la informática, acercando a todo tipo de personas a las posibilidades que tiene la informática de ser usada en las tareas que ellos llevan a cabo o que pretenden iniciar.

El principal objetivo de este proyecto es facilitar la creación de chatbots orientados a la difusión de información sobre una cierta temática y que además disponen de una cierta personalización a la hora de conversar dependiendo de ciertas características de la persona con la que se conversa.

1.2. Descripción del proyecto

El nombre que se le ha dado al chatbot es Sara. El nombre ha tenido como origen el nombre de la ciudad Siracusa en siciliano, Sarausa. El motivo de elegir este nombre es mi interés por la historia antigua. Además, idealmente el nombre de un chatbot debe ser corto y fácil de recordar, y el nombre de Sara lo cumple perfectamente.

El proyecto consiste en el desarrollo de una aplicación cliente-servidor. La aplicación es desplegada en la nube. Mi papel en este proyecto será parecida a la de un programador Full Stack.

En cuanto a la parte del cliente, es adaptable a cualquier tipo de front-end o interfaz siguiendo un diseño responsive. Esta parte en todo momento busca ser lo más cómoda y accesible posible para el usuario.

Pero la parte del servidor o back-end es la más importante de esta aplicación, ya que contiene toda la lógica del chatbot. La parte del servidor es una API REST, la cual permite la adaptabilidad de la interfaz, separando cliente y servidor.

El proyecto en su totalidad ha sido alojado en un repositorio de Github, cuyo nombre es SARA_Chatbot. Este repositorio se ha mantenido en privado durante el desarrollo del proyecto, pero tras su finalización se pondrá en público para su libre uso y consulta. Durante el desarrollo del proyecto se ha seguido una metodología de desarrollo ágil, la cual permite ir generando prototipos de la aplicación durante el desarrollo del proyecto, los cuales se pueden evaluar para tener en cuenta los fallos para futuros prototipos.

1.3. Estructura de la memoria

- **Capítulo 1. Introducción:** Breve introducción al proyecto, indicando además los motivos por los que se ha desarrollado y una breve descripción técnica del proyecto.
- **Capítulo 2. Estado del arte:** Revisión del estado actual de la tecnología de desarrollo de los chatbots.
- **Capítulo 3. Análisis:** Análisis de la metodología de desarrollo a seguir, las historias de usuario y los riesgos que se pueden o no materializar durante el proyecto.
- **Capítulo 4. Planificación:** Planificación de las tareas a realizar en cada sprint y planificación temporal de la realización de los sprints.
- **Capítulo 5. Gestión:** Explicación detallada de la gestión del código, de los distintos recursos utilizados durante el proyecto, y de los costes del mismo.
- **Capítulo 6. Diseño:** Diseño de las distintas partes en la que se divide el sistema.
- **Capítulo 7. Implementación:** Exposición de la implementación seguida en cada una de las partes del sistema.
- **Capítulo 8. Pruebas:** Realización de las pruebas en cada una de las partes principales del sistema y la posterior exposición de los resultados.
- **Capítulo 9. Conclusión y posibles ampliaciones:** Exposición de las conclusiones extraídas del desarrollo del proyecto y sugerencia de posibles ampliaciones para la mejora del sistema.

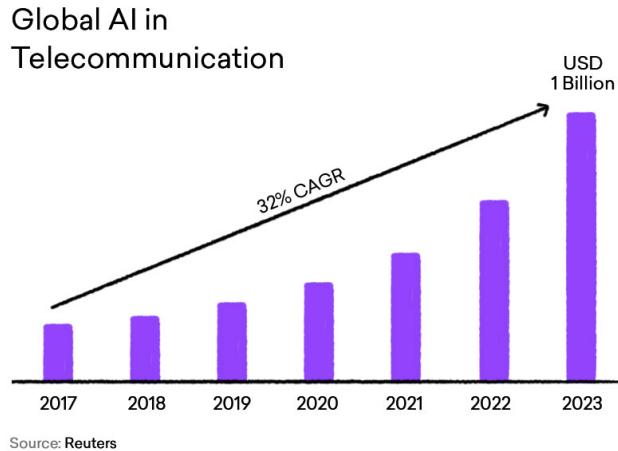
Capítulo 2

Estado del Arte

2.1. Análisis del Estado del Arte

Los chatbots son agentes conversacionales que pueden interactuar, a través de voz o texto, con los usuarios a través de lenguajes naturales (1). El auge de esta tecnología en los últimos años es debido a que puede ser utilizada como mano de obra a un coste menor en una amplia gama de campos. Tal y como se indica en (2) los chatbots no están diseñados solamente para imitar y entretenir a los humanos, sino que también puede diseñarse para actividades como la atención al cliente, la asistencia personal o la pedagogía entre otras. Y no hay mejor forma de asegurar el potencial de los chatbots que mostrando el aumento de inversión en inteligencia artificial en el campo de las telecomunicaciones (Figura 2.1) o mostrando el aumento de publicaciones sobre chatbots (Figura 2.2); y viendo qué empresas están apostando por esta tecnología, como pueden ser cinco grandes empresas tecnológicas como son Google, Apple, Facebook, Microsoft y Amazon. Aunque basándonos en este párrafo de la sensación de que todas las empresas deberían disponer de chatbots, esta tecnología necesita de cierta planificación y comprensión de la tecnología, pues requiere de una gran inversión inicial, dado su gran requerimiento de información para funcionar de forma correcta.

Aunque es una tecnología en auge y con mucho potencia en este momento, hay aspectos de ella que todavía no están lo suficientemente desarrollados como para decir que se trata de una tecnología del todo madura. Estas carencias están sobre todo enfocadas en tres puntos. En primer lugar, nos encontramos con el problema que aparece nada más empezar, y es la carencia de un esquema de clasificación de chatbots, dado que no se ha definido una clasificación global. La clasificación más extendida es la clasificación según el mecanismo de recuperación y de generación de respuestas, donde destacan los chatbots que hacen uso de bases de conocimientos estáticas, es decir, chatbots basados en reglas; y los chatbots que hacen uso de mecanismos de aprendizaje e inferencia adaptativos, es decir, los chatbots basados en



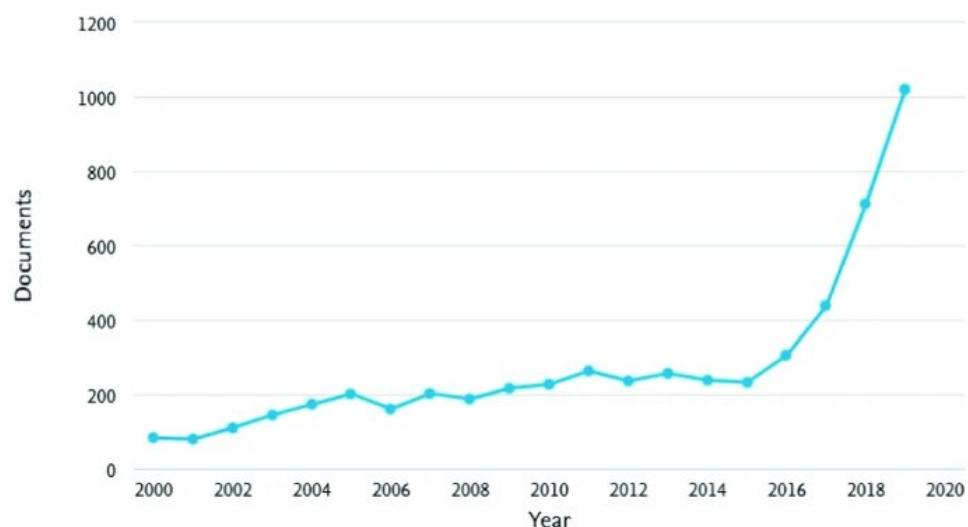
Fuente: <https://es.aivo.co/blog/chatbots-in-telecom>

Figura 2.1: Estimación de la inversión en IA en el campo de las telecomunicaciones

inteligencia artificial, los cuales son el objetivo de la mayoría de los desarrollos actuales. En segundo punto, las revisiones están desactualizadas en el ámbito de la generación de respuestas dado el rápido avance de las técnicas computacionales. En tercer punto, también las revisiones no están muy centradas en la aplicación empresarial de los chatbots ni en su usabilidad, sino que más bien están centradas en el aspecto técnico, lo cual complica su desarrollo por parte de trabajadores ajenos al mundo de la informática y/o de la inteligencia artificial.

La estructura de los chatbots ha ido evolucionando y extendiéndose. Originalmente, la estructura de los chatbots era la descrita por Abdul-Kader y Woods (1), donde se podía reconocer tres componentes: interfaz, un clasificador y un graphmaster. Esta estructura está muy desactualizada a día de hoy, ya que en la estructura original la interfaz solo tenía previstas entradas en forma de texto, pero hoy en día las entradas son de tipo multimedia, por lo que es necesario actualizar la interfaz a una interfaz multimedia, y añadir un nuevo componente a la estructura, encargado de manejar esas entradas multimedia y convertirlas en entradas de texto que se puedan manejar para generar respuestas; además dado el auge de los chatbots basados en generación, a día de hoy el componente del graphmaster no tiene sentido, dado que las respuestas se generan utilizando modelos pre entrenados realizando inferencias basándose en las entradas, por lo que tener un manejador para el almacenamiento de conocimiento deja de tener sentido. Por lo tanto, en la

Documents by year



Fuente:

https://link.springer.com/chapter/10.1007/978-3-030-49186-4_31

Figura 2.2: Historial de las publicaciones sobre temas relacionados con chatbots

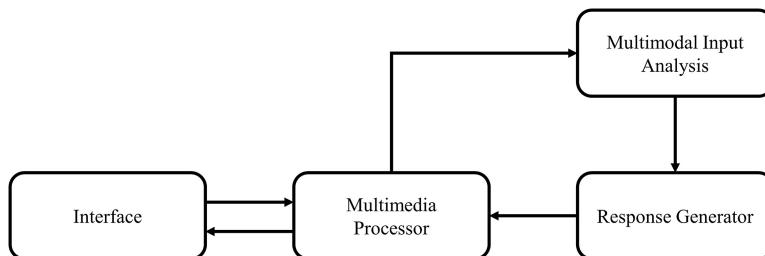
estructura el graphmaster debe ser actualizado por un modelo generador de respuestas. Aplicando todas estas actualizaciones sobre la estructura original de Abdul-Kader y Woods, obtenemos la estructura actual de los chatbots, que se muestra en la Figura 2.3. Finalmente, tenemos cuatro componentes: la interfaz, un procesador multimedia, un análisis de entrada multimodal y un generador de respuestas.

Un componente que se muestra en la Figura 2.3 y que no se ha mencionado explícitamente cuál es su labor es el análisis de entrada multimodal. Este análisis efectúa un pretratamiento de los datos de entrada, hoy en día se intenta mejorar este componente con técnicas avanzadas de comprensión del lenguaje natural (NLU).

En el caso de los chatbots basados en la recuperación, el generador de respuestas desempeña el mismo papel que el graphmaster, ya que almacena y recupera las respuestas; mientras que en el caso de los chatbots basados en la generación, genera una salida a partir de la entrada que ha sido transferida desde la unidad de pretratamiento (1).

Centrándonos en el componente del generador de respuestas, a día de hoy se utilizan distintas técnicas para implementar este componente. Las técnicas más destacadas son las siguientes:

- **Template:** Esta técnica se compone de patrones y plantillas formando



Fuente: A critical review of state-of-the-art chatbot designs and applications (1)

Figura 2.3: Estructura de los chatbots actuales

pares con ellos. Para generar las respuestas se busca el patrón que más concuerde con la entrada recibida, y una vez se haya elegido un patrón se devuelve la plantilla asociada al mismo. El lenguaje que más destaca en esta técnica es AIML¹. La desventaja de tener que crear los archivos AIML se puede suplir mediante la extracción automática de conocimientos, reduciendo el trabajo manual. Los chatbots basados en plantillas son sencillos de producir, por lo tanto, son una opción popular para chatbots sencillos a pequeña escala.

- **Corpus:** Son una buena técnica como alternativa a los templates cuando se están construyendo chatbots a gran escala, debido a que con los templates al aumentar de escalas aumenta el número de patrones y trae consigo búsquedas más lentas lo que ralentiza la generación de respuestas. Existen dos tipos de chatbots basados en Corpus, un tipo de chatbots basados en corpus que no selecciona cuidadosamente el conocimiento y otro que si lo hace. El tipo que selecciona cuidadosamente el conocimiento se puede implementar mediante bases de datos o mediante modelos de ontología, los modelos de ontología tienen una gestión más flexible del conocimiento que las bases de datos, la generación en este tipo de chatbots se realiza también con búsqueda pero esta vez en las estructuras mencionadas anteriormente. El tipo que no selecciona cuidadosamente el conocimiento transforma las palabras en vectores que representan los significados semánticos de las palabras en un espacio multidimensional, la generación en este tipo de chatbots se realiza mediante el cálculo de la distancia entre los pares de entrada del usuario y de consulta-respuesta, se empareja la consulta con la menor distancia y se selecciona su correspondiente respuesta como salida. Debido a la gran capacidad de gestión del conocimiento, varias aplicaciones que requieren un conocimiento formal del dominio utilizan

¹<http://www.aiml.foundation>

chatbots basados en corpus (1).

- **Intent:** Los chatbots basados en intenciones se emplean ampliamente para los sistemas orientados a tareas que presentan diálogos de varios turnos (1). Una intención representa un mapeo entre lo que dice el usuario y la acción que debería ejecutar el chatbot. El emparejamiento de las entradas con las distintas intenciones se hace de una forma distinta a la vista anteriormente, dado que ahora debemos emparejar los elementos semántico en vez de la frase completa. Puesto que la entrada es una frase, en el caso de esta técnica es necesario añadir adicionalmente un componente de análisis de entrada multimodal (SLU), dentro de él se efectúa un análisis semántico de las entradas, para este análisis se suelen usar técnicas avanzadas de comprensión del lenguaje natural. Una vez tenemos los elementos semánticos de la entrada se procede a emparejar la entrada con una intención o dicho de otra manera, se procede a efectuar una clasificación para saber a qué intención pertenece la entrada, esta clasificación se suele hacer mediante técnicas de aprendizaje automático. Otra característica de este tipo de chatbots es la existencia de estados, debido a que son chatbots orientados a tareas; los cuales determinan el momento de la conversación en el que nos encontramos, que intenciones han sido ejecutadas, y cuáles pueden ser elegidas para ejecutar posteriormente. El manejo de los estados se compone de dos tareas, por un lado, la obtención del estado actual del diálogo, para lo cual se utilizan técnicas de DST; y, por otro lado, el procesamiento del estado del diálogo para determinar a qué estado se va a pasar el diálogo, para esta tarea se utilizan técnicas de DPO. Las dos técnicas mencionadas son necesarias para establecer la secuencia de los diálogos. Las bases de conocimiento de los generadores de respuestas en los chatbots basados en intenciones están compuestas principalmente de frases de entrenamiento, intenciones y respuestas. Para generar las respuestas se analiza semánticamente las entradas en el módulo SLU, posteriormente se emplea el módulo DST para estimar el estado de los diálogos actuales y finalmente se emplea el módulo DPO para determinar las acciones y respuestas.

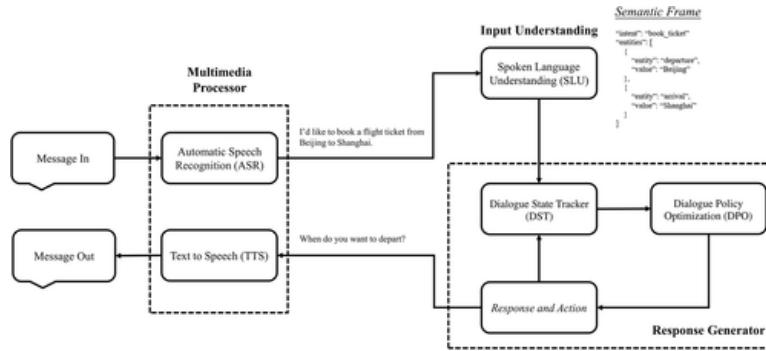
Para un mejor entendimiento de esta compleja estructura se dispone de la Figura 2.4.

Muchas de las grandes plataformas actualmente están orientadas a la creación de chatbots de este tipo, como puede ser la plataforma de Google con Dialogflow² (originalmente llamado Api.ai) o la plataforma de código abierto de Rasa.³.

- **Red neuronal recurrente (RNN):** Los chatbots creados con esta

²<https://dialogflow.cloud.google.com>

³<https://rasa.com/>



Fuente: A critical review of state-of-the-art chatbot designs and applications (1)

Figura 2.4: Estructura de los chatbots basados en intenciones

técnica dejarán de estar basados en la recuperación, como pasaba en las tres anteriores técnicas, si no que estarán basados en la generación, pero esto que quiere decir, en definitiva quiere decir que la generación de las respuestas no se realizará mediante búsquedas en alguna estructura y sustitución mediante pares consulta-respuesta. La generación de las respuestas se efectúa mediante inferencias con modelos de redes neuronales recurrentes. Estos chatbots, a diferencia de los chatbots basados en intenciones, no están orientados a las tareas y esto deriva del problema que llevan las redes neuronales, y no es otro que las redes convierten al chatbot en una caja negra, esto quiere decir que la respuesta del chatbot ante cierta entrada es incierta incluso para su creador, este inconveniente se convierte en una ventaja si orientamos a los chatbots al chat, dado que esta incertidumbre en la respuesta da una gran sensación de naturalidad y creatividad durante la conversación, por lo que son apropiados para actividades relacionadas con el entretenimiento y la salud mental entre otras. Estos chatbots se han visto muy potenciados en los últimos años debido al rápido desarrollo del aprendizaje profundo. Uno de los mayores hitos en el aprendizaje profundo ha sido la creación de GPT-3⁴ creado por la empresa OpenAI, el cual es un nuevo modelo de inteligencia artificial que permite generar lenguaje escrito. Como se ha mencionado anteriormente, ya no se realiza recuperación para la generación de respuestas, por lo tanto, las bases de conocimiento de los chatbots basados en RNN estarán compuestas de los conjuntos de datos de entrenamiento. El resultado de los chatbots basados en la generación depende de la especificación del modelo, los conjuntos de datos de entrenamiento, el proceso de

⁴<https://openai.com/blog/openai-api>

entrenamiento y la entrada del usuario (1).

- **Aprendizaje por refuerzo (RL):** Este tipo de chatbots suelen estar basados en la recuperación y utilizan plenamente el contexto de entrada para la búsqueda de respuestas óptimas (1). Dado que se basan en la recuperación, en sus bases de conocimientos hay pares de consulta-respuesta predefinidos. El método RL se basa principalmente en el proceso de decisión de Markov. Al igual que pasaba en los chatbots basados en templates, en los chatbots basados en RL se usan representaciones vectoriales, en concreto el estado del chatbot es una representación vectorial de un turno de conversación. Este vector puede contener la respuesta derivada de la entrada o adicionalmente puede contener el acto de diálogo, el sentimiento del usuario y el enunciado genérico del usuario. El contexto de la comunicación forma una secuencia de estados. Aunque es una técnica basada en la recuperación, esta recuperación es distinta a la descrita en las anteriores técnicas, en el caso de esta técnica se emplea una política para encontrar la respuesta adecuada, esta política debe ser bien entrenada para lo que son necesarios diálogos predefinidos. Este tipo de chatbots se pueden utilizar para secciones de "Preguntas frecuentes".
- **Enfoques híbridos:** El intento de combinar varias técnicas para suplir las desventajas de ambas con las ventajas de cada técnica es algo que se ha intentado en una multitud de ámbitos, y este no iba a ser menos. En (1) se hace referencia a una serie de propuestas de diferentes investigadores. Una de ellas puede ser la realizada por Li Wu y Wu Wu, los cuales propusieron un marco de correspondencia secuencial basado en una RNN. El marco considera las relaciones entre los enunciados anteriores y las respuestas candidatas y selecciona las respuestas óptimas para el contexto.

Si nos fijamos en la técnica utilizada para crear el chatbot podemos obtener una buena clasificación de los mismos.

Una vez estudiada la estructura actual de los chatbots. A continuación voy a exponer algunas de las posibles aplicaciones que pueden tener los distintos chatbots.

- **Atención al cliente para el comercio electrónico:** Puede que sea la aplicación más usada. Dado el gran crecimiento del sector, se ha producido un elevado aumento de la carga de trabajo. Como se ha indicado anteriormente, por ejemplo, en el caso de los chatbots basados en intenciones, los chatbots son capaces de reducir la carga de trabajo y consecuentemente también reduce los costes humanos. La calidad del servicio proporcionado por estos chatbots está en continua mejora debido a la dificultad de manejar peticiones muy complejas por parte del

usuario y otros problemas como el amplio dominio necesario para contestar de forma correcta durante toda la conversación y no solamente para contestar, sino también para realizar un servicio agradable para el usuario.

- **Asistencia personal:** Esta aplicación está muy extendida y es utilizada asiduamente por multitud de personas. Los chatbots de asistencia personal pueden emplearse para aumentar la eficiencia del trabajo o para gestionar la vida cotidiana del usuario. Un posible ejemplo podría ser Siri⁵, que es un chatbot incorporado en todos los dispositivos iPhone.
- **Asistencia sanitaria:** Esta clase de chatbots actúan como asistentes médicos para el diagnóstico de enfermedades. La comprensión de los síntomas del paciente es primordial en el proceso de detección de la enfermedad, y una de las principales funciones de estos chatbots es recoger suficiente información de los pacientes (1). Aunque en los aspectos médicos la última palabra la tiene un profesional humano, los chatbots puede realizar diagnósticos previos rápidamente dada su inmenso conocimiento al que se puede acceder en un instante, cosa que no sucede con los humanos, lo que ralentiza el diagnóstico. Esta enorme base de conocimiento es muy importante en la medicina debido al vasto dominio de este campo y el cual está en continuo crecimiento y mejora.
- **Pedagogía:** Estos chatbots están diseñados para ayudar al aprendizaje. Dado que la función principal de los chatbots es comunicarse con los usuarios, el aprendizaje de idiomas ha sido una de las principales vías perseguidas (1), ya que el aprendizaje de idiomas está a la orden del día de todo el mundo. Estos chatbots son útiles para enseñar a usuarios mediante conocimientos de expertos, sin que sea necesario una ingente cantidad de expertos para enseñar, puesto que esta cantidad de expertos en ciertos campos y lugares no está disponible.

En todas estas aplicaciones existe el problema de la seguridad de los datos necesarios por el chatbot, este aspecto está muy presente en los últimos años debido a los múltiples casos publicados sobre falta de seguridad con los datos de los usuarios. De cara al futuro se pretende que aumente la seguridad de estos datos. Una solución de reciente desarrollo para la problemática de la privacidad de los datos, es el aprendizaje federado. Tal y como indica Karen Hao (3): ".Aprendizaje federado: la nueva arma de IA para asegurar la privacidad". Este tipo de aprendizaje influye en una de las partes críticas de la privacidad de los datos, y es el traspaso de los datos entre dos ubicaciones. El aprendizaje federado evita ese traspaso de los datos, entrenando pequeños

⁵<https://www.apple.com/es/siri>

modelos en cada ubicación donde se encuentren los datos, y posteriormente traspasando estos pequeños modelos y finalmente combinando todos estos modelos en un modelo maestro único. Un ejemplo claro de aplicación de estos algoritmos lo da Karen Hao (3), y es su uso en el ámbito de la sanidad, dado que para elaborar modelo de IA en la salud es necesario traspasar la información almacenada en cada hospital para poder elaborar ese modelo; pero con el aprendizaje federado se podría evitar ese traspaso crítico de información.

Llegados a este punto en el que ya hemos analizado el estado actual de los chatbots, únicamente nos queda analizar cuál es la dirección de las futuras investigaciones en este sector. Estas investigaciones estarán centradas en paliar las deficiencias y en mejorar aún más las capacidades actuales.

Uno de los principales problemas de los chatbots es la ingente cantidad de datos que necesitan para empezar a funcionar. Actualmente, este conocimiento es almacenado en estructuras de distinto tipo, sin tener un esquema unificado. Esta variedad de bases de conocimiento distintas dificulta la creación de nuevos chatbots al no poder reutilizar datos utilizados en chatbots anteriores. Una posible futura mejora sería la estandarización de la gestión del conocimiento para poder realizar esta transferencia de datos entre chatbots de forma eficiente. Una solución a este problema, la cual es ampliamente empleada, es el empleo de modelos pre entrenados. Estos modelos son simplemente modelos que han sido entrenados con un cierto tipo de información, los cuales son modelos simples que podemos usar como apoyo para la creación de modelos más complejos. Esta manera de elaborar los modelos sigue la filosofía del Transfer learning, la cual consiste en transferir conocimientos adquiridos con la resolución de ciertos problemas en la resolución de otros problemas distintos a los usados en un principio.

Un aspecto analizado anteriormente es la generación de las respuestas, este aspecto aunque ha sufrido un gran desarrollo en los últimos años, todavía no ha llegado a alcanzar su máximo nivel, esto se puede comprobar simplemente con ver, tal y como indica (1), que todavía nadie ha ganado el premio de oro del Premio Loebner, el concurso de pruebas Turing más famoso en el ámbito de los chatbots. Las futuras investigaciones están orientadas a la mejora de las técnicas de aprendizaje profundo. Estas técnicas son entre otras las redes generativas adversativas (GAN), este tipo de redes está muy extendido en el mundo de la visión por computador, pero todavía no se ha llegado a conseguir una integración de esta técnica en el mundo de los chatbots.

Un aspecto de los chatbots que todavía tiene mucho potencial por explotar es sin duda la interfaz. Hasta ahora solamente hemos indicado que se están utilizando interfaces multimodales. Pero no se está sacando todo el partido a este tipo de interfaces, dado que actualmente únicamente se realiza una conversión de todo tipo de entradas a una entrada en formato de texto. Pero las entradas multimodales contienen información adicional a la que se

extrae hoy en día, como puede ser el lenguaje no verbal durante una conversación por voz. Este lenguaje no verbal proporciona mucha información que puede ayudar a la generación de respuestas y a tener un mejor contexto de la conversación. Para que toda esta información no se desperdicie se debe avanzar en la investigación para la extracción de esta información. Otro aspecto de la interfaz donde es necesaria la investigación es la devolución de la respuesta a través de la interfaz, ya que hasta ahora estábamos hablando de como se debía mejorar la entrada de información en la interfaz. La devolución de esta información se puede efectuar de muchas maneras, escrita, por voz, etc. Lo verdaderamente importante en la devolución de la respuesta es que el usuario perciba esa respuesta con facilidad, con una buena estética que atraiga la atención del mismo. Una de las líneas de futuras investigaciones son los avatares 3D, es decir, la respuesta es devuelta por voz, pero lleva añadida la gesticulación de un avatar 3D.

Una dirección en la que se pueden realizar futuras investigaciones son las aplicaciones del chatbot. Dado el auge del internet de las cosas (IoT) hoy en día hay todo un campo donde integrar los chatbots para mejorar el servicio a los usuarios, como pueden ser los coches inteligentes, los hogares inteligentes, y muchos más.

Por último, tal y como se ha indicado anteriormente, uno de los grandes problemas actualmente de los chatbots es la falta de seguridad con los datos, este es un aspecto crucial en todas las futuras investigaciones.

2.2. Herramientas para el desarrollo del chatbot

Hay dos caminos para crear un chatbot, por un lado, usando cualquier lenguaje de programación como Java, Python, C++, etc; y, por otro lado, usando plataformas para el desarrollo de chatbots. En este apartado voy a analizar distintas herramientas para el desarrollo de mi chatbot.

Un aspecto a tener en cuenta a la hora de elegir una plataforma para desarrollar un chatbot es que existen dos elementos en esta creación.

- **Bot Frameworks:** son las plataformas encargadas de la creación y el alojamiento de los chatbots.
- **Bot Platforms:** son los entornos y aplicaciones donde van a ser desplegados los chatbots para su uso por parte de los usuarios.

A la hora de elegir la plataforma donde desarrollar el chatbot habrá que tener en cuenta que tanto el Bot Framework como el Bot Platform elegidos se adecuan a los requisitos del chatbot a crear.

Una posible clasificación de las plataformas puede ser basándonos en el modo de creación de los chatbots. Dentro de esta clasificación encontramos las siguientes plataformas:

- **Plataformas visuales**
- **Plataformas conversacionales**
- **Plataformas programables**

Plataformas visuales

En estas plataformas no es necesario tener conocimientos técnicos para producir un chatbot. Por esta razón son las plataformas ideales para personas que no tienen conocimientos en programación o inteligencia artificial. Algunos ejemplos de este tipo de plataformas pueden ser: Chatfuel y Octane AI. Esta simpleza en la creación del chatbot afecta en la posible complejidad que pueda tener el chatbot, por lo tanto, no son la mejor opción para chatbots con cierta complejidad, pero si para chatbots muy simples.

Plataformas conversacionales

En estas plataformas es posible la creación de chatbots algo más complejos que en las plataformas visuales. Estos chatbots son capaces de mantener una conversación con un usuario, pero que a diferencia de los anteriores chatbots, esta conversación no tiene un objetivo específico. Un posible ejemplo de este tipo de plataformas puede ser AIML, aunque AIML no es en sí una plataforma sino más bien un lenguaje de programación, sí se convierte en una plataforma si se combina el lenguaje con alguna plataforma donde se permita al menos el alojamiento del chatbot. Este tipo de plataformas ya no está orientado a usuarios sin conocimientos técnicos, sino más bien a usuarios que tengan cierto nivel técnico y que busquen crear chatbots con cierta complejidad, pero que no tengan mucha escala.

Plataformas programables

En estas plataformas se pueden generar desde los chatbots más simples hasta los más complejos. Para poder aumentar la complejidad se hace uso de técnicas de inteligencia artificial junto a las técnicas que ya se utilizaban en las anteriores plataformas, y aumenta la posibilidad de interactuar con servicios externos a la plataforma como pueden ser bases de datos y muchos más servicios. Algunos ejemplos de este tipo de plataformas pueden ser: Google Dialogflow, Microsoft Bot Framework e IBM Watson.

En cuanto a la implementación de chatbots mediante lenguajes de programación, es una herramienta únicamente disponible para personas con los suficientes conocimientos técnicos, no es una opción para aquellas personas que no tengan conocimientos técnicos en el ámbito de la informática.

En la actualidad se dispone de un número elevado de herramientas capaces de producir chatbots de diferentes complejidades. De entre este número elevado de herramientas he seleccionado unas cuantas para analizarlas en detalle como posibles herramientas con las que generar mi chatbot. En esta lista de herramientas no se encuentra ninguna plataforma visual dado que no posibilitan la complejidad necesaria para generar mi chatbot.

Las herramientas analizadas son las siguientes:

- AIML
- IBM Watson
- Google Dialogflow
- Rasa Stack
- GPT-3
- BlenderBot

Para consultar la información extraída de cada una de las herramientas, la cual ha ayudado en la elección de una de ellas para el desarrollo del chatbot, debes consultar el Apéndice A.

2.3. Comparativa de las herramientas

Después de revisar las distintas herramientas, es hora de compararlas para facilitar la elección de una ellas para el desarrollo del proyecto. En primer lugar, se hará una comparación de pros y contras de todas las herramientas (Tabla 2.1), y seguidamente se mostrará una tabla (Tabla 2.2) con una serie de características, indicando cuáles están presentes en las distintas herramientas.

2.4. Elección de la herramienta

Dado que ninguna de las plataformas se adecua perfectamente en su totalidad a las características que busco en la creación del chatbot, he decidido seguir un enfoque híbrido, ya que como se indica en la sección 2.1 proporciona muchas ventajas el combinar varias herramientas.

Como mi idea es montar un chatbot de código abierto para reducir lo máximo posible los gastos derivados de su creación, pero sin perder potencia, la mejor herramienta que se adecua a esta definición es la herramienta BlenderBot. Aunque esta herramienta trae consigo un gran inconveniente, su dificultad de despliegue. Este inconveniente es crítico para el proyecto, dado que se quiere facilitar el acceso al chatbot para que pueda usar en eventos

| | PROS | CONTRAS |
|------------|---|--|
| AIML | <ul style="list-style-type: none"> - Multilenguaje - El chatbot no es una caja negra - Lenguaje de programación simple - Posibilidad de conexión con API's propias - Muchas funcionalidades de forma gratuita | <ul style="list-style-type: none"> - Alto gasto de tiempo en la creación del agente conversacional - Necesidad de una cuenta de pago usando servicios web - No es una plataforma en la nube - Conversaciones poco naturales y poco flexibles |
| IBM Watson | <ul style="list-style-type: none"> - Posibilidad de crear chatbots muy complejos - Gran cantidad de conexiones con API's de mucha calidad - Editor de arrastrar y soltar - Facilidad de desplegar el chatbot en muchos canales importantes - Es una plataforma en la nube | <ul style="list-style-type: none"> - Necesidad de una cuenta de pago para crear chatbots complejos - Enfocado para el mundo empresarial - El chatbot es una caja negra |
| Dialogflow | <ul style="list-style-type: none"> - Posibilidad de crear chatbots muy complejos sin necesidad de pago - Posibilidad de conexión con un servidor web de forma gratuita - Facilidad de desplegar el chatbot en muchos canales importantes - Creación de chatbots usando solamente frases y entidades - Es una plataforma en la nube | <ul style="list-style-type: none"> - Para hacer agentes muy complejos es necesario gastar tiempo en la creación del servidor - Se necesita tener cierto conocimiento de la plataforma para sacar todo su potencial - Calidad regular en la documentación - Parte del chatbot es una caja negra |
| Rasa Stack | <ul style="list-style-type: none"> - Documentación de calidad - El chatbot no es una caja negra - Facilidad de desplegar el chatbot en muchos canales importantes - Posibilidad de crear chatbots muy complejos | <ul style="list-style-type: none"> - Necesidad de alto nivel técnico - Plataforma en desarrollo - Necesidad de mucho tiempo para la creación del chatbot - No es una plataforma en la nube |
| GPT-3 | <ul style="list-style-type: none"> - Posibilidad de crear chatbots muy complejos - Chatbots con una conversación muy natural - Buena documentación | <ul style="list-style-type: none"> - Necesidad de conocimientos técnicos - Necesidad de pago para hacer uso de los servicios - Necesidad de una solución para mantener el contexto - Requiere de un conjunto de datos para poder adaptar el chatbot a su dominio - Requiere de una plataforma aparte para el despliegue |
| BlenderBot | <ul style="list-style-type: none"> - Posibilidad de crear chatbots muy complejos - Chatbots con una conversación muy natural - Uso gratuito - Buena documentación | <ul style="list-style-type: none"> - Necesidad de conocimientos técnicos - Necesidad de una solución para mantener el contexto - Requiere de un conjunto de datos para poder adaptar el chatbot a su dominio - Requiere de una plataforma aparte para el despliegue |

Cuadro 2.1: Comparativa de pros y contras de las herramientas de desarrollo

| Nombre | Código abierto | Multilenguaje | Necesidad Conocimientos técnicos | Cloud Service | Elaboración de Chatbots complejos con facilidad | Conversación con naturalidad | Documentación de gran calidad | Facilidad para despliegue del chatbot |
|------------|----------------|---------------|----------------------------------|---------------|---|------------------------------|-------------------------------|---------------------------------------|
| AIML | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ |
| IBM Watson | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Dialogflow | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ |
| Rasa Stack | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ |
| GPT-3 | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| BlenderBot | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ |

Cuadro 2.2: Comparativa de características de las herramientas de desarrollo

culturales. Por esta razón voy a utilizar la herramienta BlenderBot combinada con la herramienta Dialogflow. Con la herramienta Dialogflow nos favorecemos de su facilidad de despliegue, sin afectar al coste de la creación del chatbot, dado que tiene versión gratuita.

Para conectar ambas herramientas se va a hacer uso de la sección Webhook de Dialogflow. Por ello será necesario crear un servidor propio donde se alojará la herramienta BlenderBot.

Capítulo 3

Análisis

3.1. Metodología de desarrollo

Las metodologías de desarrollo de software surgieron hace décadas como consecuencia de la acumulación de proyectos fallidos o de mala calidad, en los que los resultados no satisfacían al cliente. Esto era debido a que todo el desarrollo seguía el enfoque de los programadores del proyecto, los cuales no tenían en la mayoría de las ocasiones la misma visión que el cliente para el desarrollo del proyecto.

Hoy en día la elección de una metodología de desarrollo de software es fundamental para el desarrollo de un proyecto de calidad. El uso de una metodología, sin importar de qué tipo sea, nos asegura que nuestro proyecto se adecuará a las peticiones del cliente y será un buen producto final.

La implantación de estas metodologías en la mayoría de los proyectos actuales ha aumentado el porcentaje de proyectos que terminan en éxito, y además en los plazos de tiempo establecidos.

3.1.1. Tipos de metodologías

Desde que empezó la preocupación por el desarrollo de los proyectos, distintas personas han investigado y desarrollado nuevas metodologías. Una posible clasificación de todas estas metodologías puede ser según la época de su creación, teniendo metodologías clásicas y metodologías ágiles.

Metodología Clásica

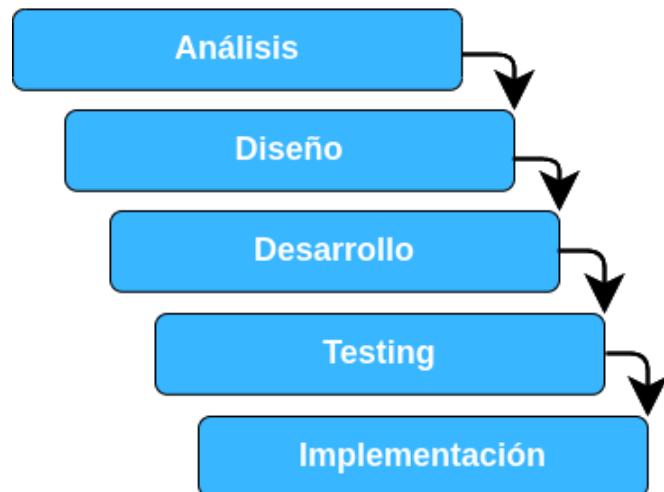
Las metodologías clásicas fueron las primeras metodologías que surgieron como solución al caos que había en la planificación de proyectos. La metodología clásica se caracteriza por una estructura secuencial, es decir, no se vuelve a la etapa anterior una vez se ha completado una etapa de la planificación; además el proceso es rígido y no cambia, es decir, una vez se

acuerda como va a ser la planificación, a lo cual se le dedica grandes cantidades de tiempo, no se modifica nada de la planificación, ni siquiera los requisitos y, por lo tanto, la comunicación con el cliente se limita al comienzo del proyecto durante la elaboración de la planificación. Aunque, viendo su nombre, parezca que se trata de una metodología arcaica y en desuso, no es así bajo ningún concepto. Esta metodología se sigue puliendo y mejorando actualmente, aunque uno de sus problemas es su enfoque "mayorista", es decir, la metodología está enfocada a los macroproyectos. Un macroproyecto es un proyecto con una gran inversión, con un equipo de trabajo de varias decenas de personas, con los roles bien definidos; y con un amplio plazo de tiempo. Aunque un proyecto no encaje con la definición de macroproyecto, esta metodología se puede adaptar al proyecto en cuestión.

La metodología clásica o metodología "en cascada", recibiendo este nombre por la ausencia de vuelta a atrás en las etapas, su ciclo de etapas se puede visualizar en la Figura 3.1.

Visual Paradigm Online Free Edition

Metodología en cascada



Text
Visual Paradigm Online Free Edition

Basado en: <https://www.yunbitsoftware.com/blog/2016/05/20/desarrollo-de-software-metodologias-waterfall-agile>

Figura 3.1: Etapas de la metodología clásica

Dado que la metodología clásica únicamente consta de un ciclo para realizar las etapas, será imprescindible una buena planificación de alcance, tiempo y coste del proyecto y de cada una de las etapas. Esta planificación inicial es tan importante porque no se obtendrán resultados del proyecto hasta que se encuentre cercano a su cierre.

Entre los problemas de la metodología clásica podemos encontrar los siguientes:

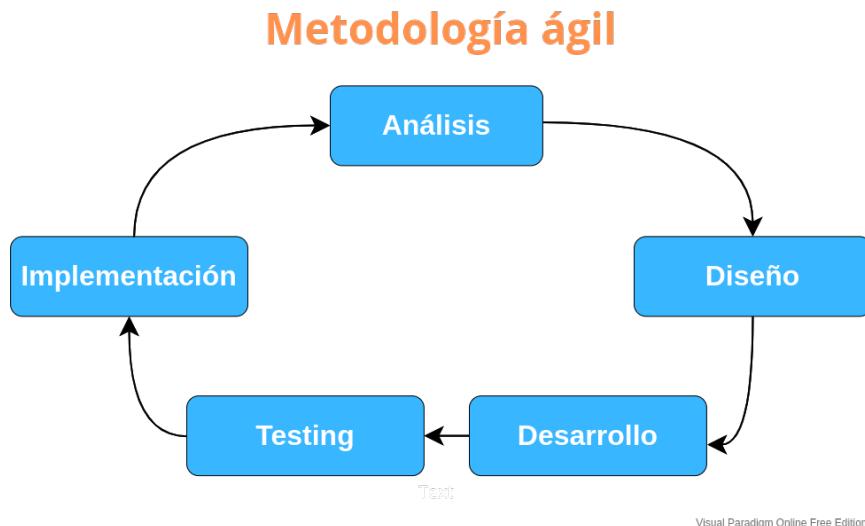
- Los proyectos de software reales difícilmente se adecuan a este modelo de proceso.
- Dificultad para expresar por parte del cliente todos los requisitos al principio del proyecto.
- Poca comunicación con cliente/usuario, hasta las etapas finales no hay un producto final que se pueda evaluar.

Metodología Ágil

Esta metodología surge como solución a los problemas de la metodología clásica, sobre todo para la planificación de proyectos cortos y cambiantes. La metodología ágil se caracteriza por su flexibilidad, durante el desarrollo del proyecto la planificación sufre continuos cambios debido a la constante comunicación con el cliente; una estructura incremental, con esta metodología no se obtiene un único proyecto final, si no más bien se van generando prototipos que se van analizando y mejorando en una retroalimentación constante, aumentando la colaboración; y un flujo de proceso iterativo, a diferencia de la metodología clásica donde cada etapa una vez finalizada no se vuelve a ella teniendo un flujo en cascada, en esta metodología el flujo de las etapas es más parecido a un círculo convertido en un bucle sin fin. Algo muy importante a tener en cuenta es que a pesar de que los sprints, un sprint es la realización de un ciclo completo; duren poco tiempo, esto no permite la ausencia de documentación, al igual que se hace en la metodología clásica, se debe documentar un mínimo a pesar de la brevedad de los sprints. Esta metodología se adapta mejor a los proyectos del ámbito de las TIC, es decir, en el desarrollo de software, donde es fundamental la flexibilidad del proyecto y el carácter incremental de la metodología. Este modo de trabajar no se adecua a proyecto que puedan estar relacionados con sectores alejados de las TIC, como puede ser el sector de la construcción, por ejemplo en la construcción de un puente, dado que no se pueden crear prototipos de una construcción, ya que no serían seguros, y además son proyectos poco cambiantes en los requisitos del proyecto durante su desarrollo.

El ciclo de etapas de la metodología ágil se puede visualizar en la Figura 3.2.

Visual Paradigm Online Free Edition



Basado en: <https://www.yunbitsoftware.com/blog/2016/05/20/desarrollo-de-software-metodologias-waterfall-agile>

Figura 3.2: Etapas de la metodología ágil

Dentro de la categoría de metodología ágil podemos encontrar las siguientes metodologías:

- Scrum
- Programación extrema (XP)
- Mobile-D

En mi caso me voy a centrar en la metodología Scrum, muy popular hoy en día en la planificación de proyectos TIC. Tal y como se indica en el artículo (4), Scrum no corresponde a ningún acrónimo, su nombre proviene del rugby, es una formación requerida para la recuperación rápida del juego ante una infracción menor. Esta metodología se caracteriza por el empleo de un conjunto de reglas y la definición de roles. La existencia de estos roles favorece una colaboración eficaz del equipo de trabajo. Los roles que se definen son: El Scrum master, el dueño del producto o Product owner y el equipo de desarrollo o team. El scrum master es la persona que lidera el equipo asegurándose que el equipo cumpla las reglas y procesos de la metodología. El dueño del producto es el representante de los accionistas y clientes que usan el software. El equipo de desarrollo es el grupo de profesionales encargados

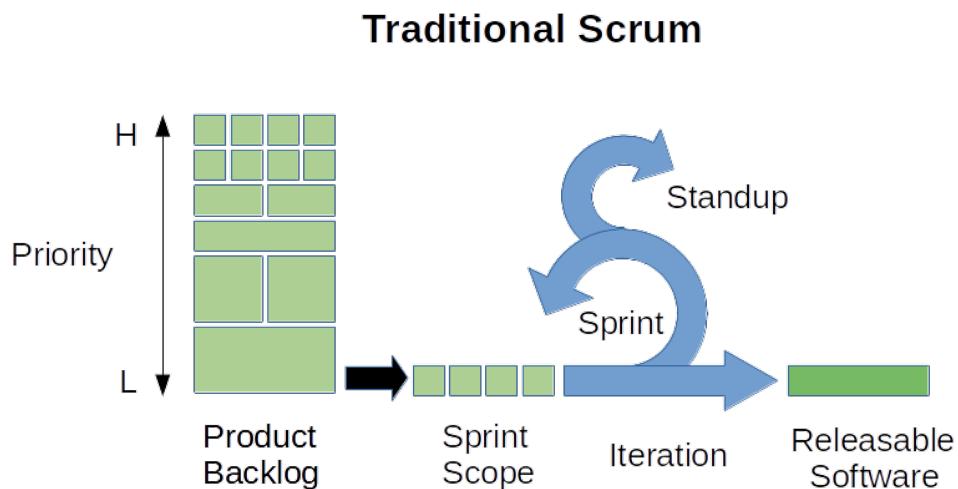
de convertir la lista de requerimientos o también llamado Product Backlog en funcionalidades del software (4).

La característica incremental de las metodologías ágiles se materializa en la metodología Scrum como un elemento llamado Sprint (Figura 3.3). Cada sprint consistirá en un recorrido completo de todas las etapas del proyecto, dando como resultado una versión funcional del producto.

Los elementos de un Sprint son los siguientes:

- Reunión de planificación del sprint
- Daily Scrum o reunión diaria
- Trabajo de desarrollo
- Revisión
- Retrospectiva del sprint

Durante la reunión de planificación del sprint se seleccionan que tareas se van a desarrollar en ese sprint. Las tareas son extraídas del Product Backlog, que no es más que una lista de tareas ordenadas por prioridad. Esta lista será actualizada y modificada durante el desarrollo del proyecto.



Basado en: <https://openwebinars.net/blog/que-es-un-sprint-scrum>

Figura 3.3: Sprint de la metodología Scrum

| Metodología Clásica | Metodología Ágil |
|---|--|
| <ul style="list-style-type: none"> - Estructura secuencial o en cascada - Proceso rígido - Enfocada a macroproyectos | <ul style="list-style-type: none"> - Estructura incremental e iterativa - Proceso flexible - Enfocada a proyectos TIC |

Cuadro 3.1: Comparativa de las metodologías de desarrollo

3.1.2. Comparativa de las metodologías

3.2. Aplicación de Scrum al proyecto

Tomando como partida la metodología Scrum original, se ha realizado una adaptación a mi proyecto. La primera adaptación es la asignación de roles, donde una misma persona efectúa los roles de Scrum master y el equipo de desarrollo, a la vez, este equipo de desarrollo lo formará una única persona; y por último, el papel de Product owner lo realizará el tutor del proyecto. Otra consecuencia, derivada de la formación del equipo por una única persona, es que no se van a hacer reuniones diarias, dado que no hay que proceder a una supervisión del trabajo del equipo en conjunto. Aunque no se hagan reuniones diarias, se ha optado por efectuar autorreflexiones semanalmente para llevar un control del avance durante el sprint. Los sprints tendrán una duración variable dependiendo de la cantidad y el peso de las tareas efectuadas en cada sprint. Además de las propias autorreflexiones, se harán reuniones con el tutor cada dos semanas, aunque el contacto con el tutor es constante durante todo el sprint vía correo electrónico.

3.3. Historias de usuario

Dado que hemos elegido una metodología SCRUM las características y funcionalidades del sistema no van a estar definidas en función de una lista de requisitos, tal y como se hace en las metodologías tradicionales, indicando los requisitos funcionales, no funcionales y de información; sino que se van a definir por historias de usuario.

Las historias de usuario definen estos requisitos de una forma informal y desde el punto de vista de un rol. La historia de usuario estará formada por una serie de frases, pero la cantidad de frases será escasa, sin generalizar mucho en la funcionalidad descrita en la historia de usuario.

Las frases de las historias de usuario sigue el siguiente patrón: Como <rolgt;, quiero <eventgt; para <funcionalidadgt;.

Además de la frase que describe la funcionalidad, una historia de usuario consta de una frase que indica cuando se considera completada la historia de usuario y de las tareas a realizar.

Una vez definida la estructura de las historias de usuario procedemos a

describir cada una de ellas.

3.3.1. Tarjetas de historias de usuario

| | |
|-----------------------|---|
| Identificación | HU.1 - Deducción de la edad |
| Descripción | Como usuario quiero que se deduzca mi edad para personalizar las respuestas del chatbot. |
| Aceptación | El sistema debe deducir la edad en base a información proporcionada por el usuario o en caso de no poder deducir esta información deberá preguntar explícitamente al usuario. |
| Tareas | <ul style="list-style-type: none"> • Añadir módulo de deducción al sistema antes de comenzar la conversación temática. • Comprobar que se ha deducido la información antes de comenzar la conversación temática. • Realizar la pregunta explícitamente si se detecta que no se conoce la edad. |

Cuadro 3.2: Historia de Usuario - HU.1

| | |
|-----------------------|--|
| Identificación | HU.2 - Interfaz sencilla |
| Descripción | Como usuario quiero que la interfaz sea sencilla para facilitar su uso por cualquier persona. |
| Aceptación | La interfaz debe tener una estructura de conversación sencilla, de pregunta-respuesta. |
| Tareas | <ul style="list-style-type: none"> • Integrar en el sistema una interfaz de conversación estructuralmente sencilla. |

Cuadro 3.3: Historia de Usuario - HU.2

| | |
|-----------------------|--|
| Identificación | HU.3 - Interfaz armónica |
| Descripción | Como usuario quiero que la interfaz siga la armonía del sistema para que el uso de la misma sea agradable para el usuario. |
| Aceptación | La interfaz debe mantener la apariencia que tiene el resto del sistema. |
| Tareas | <ul style="list-style-type: none"> • Integrar la apariencia del sistema en la interfaz. |

Cuadro 3.4: Historia de Usuario - HU.3

| | |
|-----------------------|--|
| Identificación | HU.4 - Interfaz responsive |
| Descripción | Como usuario quiero que la interfaz se adapte al dispositivo que esté utilizando en cada momento para aumentar el volumen de dispositivos con los que podré utilizar el sistema. |
| Aceptación | La interfaz debe adaptarse al tamaño de la pantalla del dispositivo que esté utilizando el usuario. |
| Tareas | <ul style="list-style-type: none"> • Elaborar una interfaz responsive. |

Cuadro 3.5: Historia de Usuario - HU.4

| | |
|-----------------------|--|
| Identificación | HU.5 - Conversación a tiempo real |
| Descripción | Como usuario quiero que la respuesta del sistema se genere en un corto período de tiempo para que su uso se haga más natural y entretenido. |
| Aceptación | El sistema debe generar las respuestas y proporcionárselas al usuario en un corto período de tiempo que podría ser como máximo 5 segundos. |
| Tareas | <ul style="list-style-type: none"> • Implementar un sistema eficiente en cuanto al tiempo necesario para la generación de las respuestas. |

Cuadro 3.6: Historia de Usuario - HU.5

| | |
|-----------------------|--|
| Identificación | HU.6 - Conversación coherente |
| Descripción | Como usuario quiero que las respuestas del sistema tenga una coherencia con el resto de la conversación realizada con anterioridad a la pregunta para que su uso sea más agradable e interesante para el usuario. |
| Aceptación | Las respuestas del sistema deben tener cierta coherencia con la conversación realizada hasta ese momento, haciendo uso de la información proporcionada por el usuario para generar la respuesta. |
| Tareas | <ul style="list-style-type: none"> • Implementar un mecanismo que incluya la información proporcionada por el usuario en la generación de las respuestas. • Implementar un mecanismo que genere distintas respuestas según la información deducida del usuario por el módulo de deducción. |

Cuadro 3.7: Historia de Usuario - HU.6

3.3.2. Listado de las historias de usuario

Una vez se han definido todas las historias de usuario debemos clasificar las historias de usuario por algún criterio para decidir cuáles tienen un mayor impacto en el sistema y decidir cuáles abarcar en un primer momento a la hora de ser implementadas las funcionalidades descritas en las historias. Este criterio será el nivel de prioridad. El nivel de prioridad toma valores desde 1 hasta 3, donde 1 es el mínimo nivel de prioridad y 3 el máximo nivel de prioridad. Este listado se puede ver en la Tabla 3.8.

| Identificador | Descripción | Prioridad |
|------------------|--|-----------|
| HU.1 (Tabla 3.2) | Como usuario quiero que se deduzca mi edad para personalizar las respuestas del chatbot. | 3 |
| HU.2 (Tabla 3.3) | Como usuario quiero que la interfaz sea sencilla para facilitar su uso por cualquier persona. | 2 |
| HU.3 (Tabla 3.4) | Como usuario quiero que la interfaz siga la armonía del sistema para el uso de la misma sea agradable para el usuario. | 1 |
| HU.4 (Tabla 3.5) | Como usuario quiero que la interfaz se adapte al dispositivo que esté utilizando en cada momento para aumentar el volumen de dispositivos con los que podré utilizar el sistema. | 2 |
| HU.5 (Tabla 3.6) | Como usuario quiero que la respuesta del sistema se genere en un corto período de tiempo para que su uso se haga más natural y entretenido. | 3 |
| HU.6 (Tabla 3.7) | Como usuario quiero que las respuestas del sistema tengan una coherencia con el resto de la conversación realizada con anterioridad a la pregunta para que su uso sea más agradable e interesante para el usuario. | 3 |

Cuadro 3.8: Listado de historias de usuario junto con su prioridad

3.4. Requisitos adicionales

- **Robustez:** El chatbot deberá responder siempre sin importar que mensaje envíe el usuario.
- **Documentación:** Se incluirá información a modo de manual de usuario para facilitar la utilización del sistema y el ajuste de sus parámetros.
- **Extensibilidad:** El chatbot debe estar preparado para ser fácilmente escalado y tener la suficiente potencia para cumplir sus requisitos.
- **Costo:** El costo final de implementación del sistema debe ser bajo, para ello el sistema debe usar la menor cantidad de recursos de pago sin perder la potencia necesaria.

- **Seguridad:** Se deben mantener privados todos los datos proporcionados por el usuario durante el uso del sistema.

3.5. Análisis de riesgos

Una problemática de la que ningún proyecto se puede escapar es la aparición de continuos riesgos durante el desarrollo del mismo, que pueden ocasionar problemas en el desarrollo del proyecto. Pero esto no quiere decir que estos riesgos se tenga que materializar siempre. Por esta razón existe el análisis de riesgos, para detectarlos y poner medidas para minimizar la probabilidad de su aparición.

Este análisis de riesgos consiste en detectar los riesgos que puedan aparecer, analizar las posibles causas que pueden originar este riesgo, y por último, elaborar un plan de actuación para evitar la aparición del riesgo asociado a ese plan de actuación.

Tras realizar el análisis de riesgos se obtendrá una lista de riesgos, esta lista de riesgos se materializa como una tabla (Tabla 3.9) que sigue el esquema Riesgo-Causa-Actuación. Además, se clasifican los distintos riesgos en una matriz de probabilidad-impacto (Tabla 3.10).

| Identificador | Riesgo | Causa | Plan de actuación |
|---------------|---|--|--|
| R.1 | Falta de información para generar el conjunto de datos con el que entrenar el chatbot | Uso del chatbot en temas muy especializados y poco discutidos en Internet | Intentar usar el chatbot con temas frecuentes en foros de Internet |
| R.2 | Pérdida del proyecto | Fallos en los equipos que contienen la información del proyecto o pérdidas de los mismos | Utilizar herramientas de backup para tener siempre una copia de seguridad |
| R.3 | Pérdida del listado de cambios realizados en el proyecto | Realizar el proyecto sin guardar versiones del mismo | Utilizar herramientas de control de versiones como Github |
| R.4 | Complejidad tecnológica | Desconocimiento de la tecnología base del proyecto o inmadurez de la tecnología utilizada | Realizar un análisis previo de las tecnologías a utilizar en el proyecto para analizar su viabilidad de uso en el proyecto |
| R.5 | Solicitud de cambios continuos por parte del cliente | Mala comunicación con el cliente o falta de entendimiento entre ambas partes | Realizar reuniones frecuentes para el análisis del estado del proyecto y mantener un flujo de feedback continuo con el cliente |
| R.6 | Falta de conocimientos por parte del cliente sobre la tecnología en que se basa el proyecto | Uso de tecnologías muy técnicas y especializadas, o de reciente aparición | Analizar los conocimientos del cliente al inicio del proyecto y tratar de explicar de una forma sencilla las bases de las tecnologías utilizadas |
| R.7 | Los objetivos del proyecto no son realistas | Sobreestimación del alcance del proyecto | Realizar una planificación coherente con el tiempo y recursos disponibles |
| R.8 | Generación de las respuestas en un tiempo excesivo | Utilización de tecnologías lentas y poco eficientes | Realizar optimizaciones en la generación de respuestas |
| R.9 | Problemas puntuales con las herramientas de comunicación | Caídas del servicio de correo o de videollamada | Buscar herramientas alternativas a las herramientas que están teniendo problemas puntuales |
| R.10 | Quejas sobre la interfaz del sistema | Implementación poco intuitiva de la interfaz | Evaluación de los prototipos de interfaz por parte del cliente |
| R.11 | Fugas de información sobre los usuarios | Mala gestión en la seguridad de los datos gestionados por el sistema | Intentar conservar la mínima cantidad de información en el sistema, y aquella información que se necesite conservar guardarla con la mayor seguridad posible |
| R.12 | Aparición de tareas imprevistas | Planeación errónea basada en un mal análisis del proyecto | Replanificar las tareas y actualizar tanto el backlog como los sprints a realizar |
| R.13 | Los modelos no llegan a cumplir con las expectativas | Dificultad para entrenar los modelos en la tarea que se tiene pensada en un principio | Dedicar un mayor tiempo a la búsqueda de información y a la labor de entrenamiento de los modelos |
| R.14 | Finalización de los períodos de prueba gratuitos de ciertas herramientas | Extensión del periodo de tiempo de uso de las herramientas | Realizar una buena planeación del inicio de los períodos de prueba acorde al ritmo de trabajo del proyecto |
| R.15 | Deducción errónea de la edad del usuario | Mala calidad de la imagen tomada por el usuario o uso de un modelo poco preciso para la clasificación de personas mediante la edad | Mostrar al usuario la imagen que va a enviar al proceso de deducción, y buscar modelos probados que tengan una fiabilidad y precisión aceptables |

Cuadro 3.9: Tabla de riesgos del proyecto

3.6. Riesgos materializados

Durante el tránscurso del proyecto no se han materializado riesgos en gran cantidad, aunque en la etapa final si se han materializado algunos de ellos.

| Probabilidad → Impacto ↓ | 0,1 | 0,3 | 0,5 | 0,7 | 0,9 |
|-----------------------------|------|---------------------|-------------|-------------|-----|
| Muy Bajo | | R.9 | | | |
| Bajo | | R.6 | | | |
| Medio | R.5 | R.3 | R.10 | | |
| Alto | R.12 | R.4 R.14 R.15 | R.7 | R.1 R.13 | |
| Muy Alto | R.2 | | R.8 R.11 | | |

Cuadro 3.10: Matriz probabilidad-impacto de riesgos

Los riesgos materializados se han centrado sobre todo en los modelos que generan las respuestas desde distintos puntos. Uno de los primeros riesgos que se manifestó fue el riesgo de generar las respuestas en un tiempo excesivo (R.8). A pesar de su materialización, aplicando el plan de actuación previsto se pudo solucionar rápidamente. Este plan de actuación consiste en emplear módulos de optimización para mejorar los tiempos de inferencia de estos módulos, en concreto, el módulo de optimización utilizado es DeepSpeed, del cual se hablará en posteriores apartados del proyecto.

Finalmente, se materializó el riesgo por la falta de información para generar los conjuntos de datos necesarios para entrenar a los modelos (R.1). Al igual que pasó con el anterior riesgo, se puso en marcha el plan de actuación, el cual consistía en recopilar información de foros para obtener nuestra propia información con la que entrenar a los modelos. La implementación de este plan de actuación se puede ver en el Apartado 7.3.4. Con este plan de actuación se pudo obtener un conjunto de datos suficientemente grande como para entrenar de forma adecuada nuestros modelos.

En definitiva, todos los riesgos que se pudieron materializar durante el proyecto pudieron ser solventados gracias a los planes de actuación que se desarrollaron durante la fase de análisis. Y el proyecto pudo llegar a cumplir todos sus objetivos previstos.

Capítulo 4

Planificación

En este capítulo se procederá a la planificación del proyecto, dado que hemos adoptado una metodología SCRUM para el desarrollo del proyecto, deberemos agrupar las distintas tareas analizadas en el anterior capítulo (Tabla 3.8) en sprints.

La primera reunión con la tutora se produjo el 9 de febrero de 2022 y el inicio del periodo de solicitud de evaluación para la convocatoria ordinaria es el 30 de junio de 2022, y dado que se ha empezado a trabajar en el proyecto nada más realizar la primera reunión; se dispone, por lo tanto, de aproximadamente 5 meses para el desarrollo completo del proyecto.

4.1. Planificación de los sprints

En relación con el tiempo total para el desarrollo del proyecto, se ha planificado que serán necesarios alrededor de 9 sprints para la realización total del proyecto. Estos sprints tendrán una duración de entre 1 y 3 semanas, dependiendo del número y peso de tareas asignadas al sprint.

A continuación se va a definir el contenido de cada sprint.

Durante el **Sprint 1** se investigará sobre el estado del arte de los chatbots, para tener una visión global de las posibilidades que existen a día de hoy; se analizará qué metodología de desarrollo es la más conveniente para el proyecto; y finalmente se llevará a cabo un análisis del proyecto, para extraer las tareas que serán necesarias para su completo desarrollo y los posibles riesgos que puedan acontecer durante la elaboración de las mismas. Se estima que la duración de este sprint será de 2 semanas.

Durante el **Sprint 2** se investigarán las posibles herramientas con las que implementar el chatbot, se realizará un análisis comparativo de las herramientas, y finalmente se elegirá una de ellas, basándonos en sus posibilidades y los requisitos que se han estimado para el proyecto en la fase de análisis (Tabla 3.8). Se estima que la duración de este sprint será de 2 semanas, al igual que el anterior.

Durante el **Sprint 3** se investigarán las plataformas donde se podrán alojar los distintos elementos de la estructura del proyecto. Se estima que la duración de este sprint será de 1 semana.

Durante el **Sprint 4** se elaborará una versión inicial del proyecto, esta versión inicial consistirá en un sistema que puede responder a las preguntas que se le van efectuando al chatbot. Se estima que la duración de este sprint será de 2 semanas.

Durante el **Sprint 5** se elaborarán herramientas para el correcto formateo de los datos de entrenamiento, y se crearán modelos para cada tramo de edad, basándose en un ajuste de un modelo base. Se estima que la duración de este sprint será de 2 semanas.

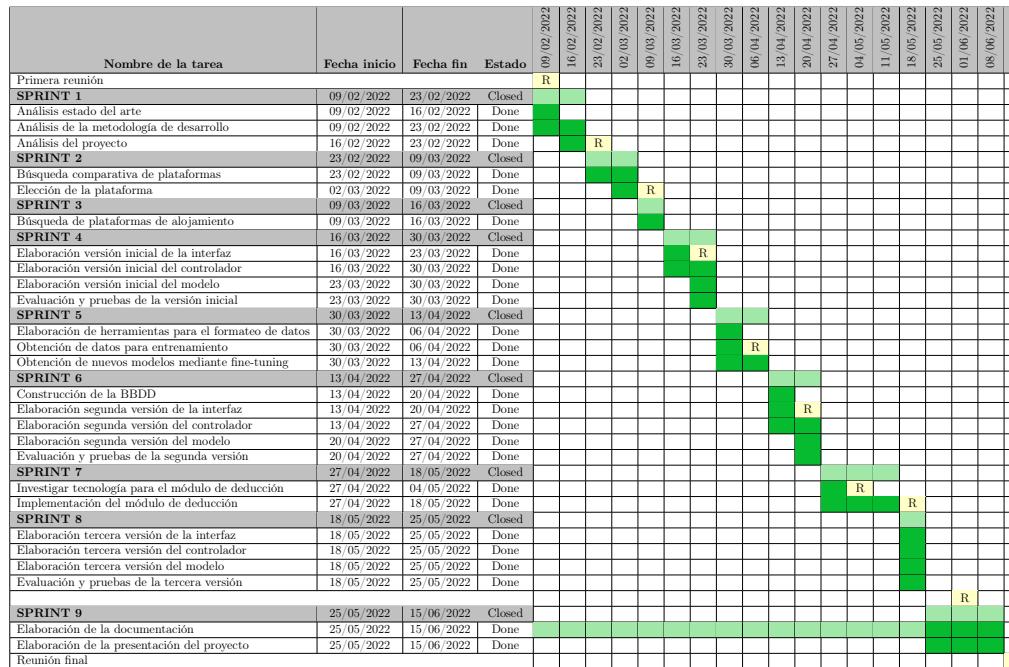
Durante el **Sprint 6** se generará una BBDD donde almacenar los datos de entrenamiento y el contexto de las conversaciones realizadas con el chatbot, y se elaborará una nueva versión del proyecto donde se probarán los modelos obtenidos en el anterior sprint y la nueva BBDD. Se estima que la duración de este sprint será de 2 semanas.

Durante el **Sprint 7** se investigará la tecnología con la que hacer el módulo de deducción del chatbot; y se implementará el módulo, donde se intentará deducir la edad del usuario con el que está hablando el chatbot. Se estima que la duración de este sprint será de 3 semanas.

Durante el **Sprint 8** se elaborará una nueva versión del proyecto donde se probará el módulo de deducción, elaborado en el anterior sprint. Se estima que la duración de este sprint será de 1 semanas.

Durante el **Sprint 9** se elaborará la documentación del proyecto. Aunque esta tarea se estará realizando de forma continua durante todo el desarrollo del proyecto, en este último sprint se intensificará la dedicación a la tarea; y se elaborará la presentación del proyecto, así como todos los recursos necesarios para su correcta exposición. Se estima que la duración de este sprint será de 3 semanas.

4.2. Planificación temporal



Cuadro 4.1: Diagrama de Gantt sobre la planificación temporal

Capítulo 5

Gestión

En este capítulo se detalla la gestión realizada de todos los elementos involucrados en el desarrollo de un proyecto.

5.1. Gestión del código y de la documentación

Por supuesto, una parte fundamental de un proyecto TIC es el código y la documentación. Estos dos elementos son la base de todo el proyecto, por lo que habrá que gestionarlos de forma eficiente y correcta.

Dado que son cruciales, será imprescindible asegurar su seguridad ante pérdidas, ya que son elementos sobre los que se invierte mucho tiempo. Siguiendo el plan de actuación contra el riesgo de perder información del proyecto, se efectuarán backup y controles de versiones de ambos elementos.

En primer lugar, para la gestión del código, en vez de alojar simplemente este código de manera local, lo que suele hacer en la mayoría de los proyectos actuales es crear un repositorio privado, llamado SARA_Chatbot, en la plataforma GitHub¹. Una vez tenemos un repositorio privado donde tener un sistema de control de versiones a la vez que sirve de backup, es necesario sincronizar este repositorio que está en la nube, con un repositorio local para poder trabajar con el código. A través de esta sincronización se irán actualizando ambos repositorios conforme se vaya modificando el código y el programador vaya enviando commits. Un commit no es más que una petición para guardar los cambios en el sistema de control de versiones del repositorio privado en GitHub. Para la edición de todo el código se hará uso del editor de código Visual Studio Code.

Y en segundo lugar, para la gestión de la documentación, de igual modo que se hace con el código, se alojará en GitHub, aunque se gestionará de una forma distinta a la realizada con el código. De igual modo, se creará un repositorio privado, llamado Memoria_TFG; y un repositorio local. La diferencia con la gestión del código llega a la hora de la edición de la

¹<https://github.com>

documentación. Para la edición de la documentación se hará empleo de la plataforma online Overleaf², una de las plataformas más empleadas para la edición de documentos escritos en LaTeX. La gestión de la documentación se podría hacer sin necesidad de tener un repositorio local, dado que Overleaf tiene una herramienta para sincronizar los proyectos alojados en Overleaf con repositorios de GitHub. Pero como para efectuar esta sincronización con Overleaf es necesaria una cuenta de pago, y una de las prioridades de este proyecto es minimizar el uso de recursos de pago, para simular esta sincronización se ha creado un repositorio local y además se ha creado un script para la descarga del proyecto completo de la plataforma Overleaf y su posterior movimiento al repositorio local. Además, opcionalmente se puede usar el script para realizar el movimiento del proyecto descargado y un posterior commit al repositorio en Github.

5.2. Gestión de recursos

5.2.1. Recursos humanos

En la gestión de los recursos humanos se gestiona todo lo relacionado con las personas que trabajan en el proyecto. Dado que este proyecto se efectúa en el marco de un Trabajo de Fin de Grado, la lista de personas involucradas en el proyecto es muy reducida. En concreto, el personal del proyecto es el siguiente:

- **Mario Carmona Segovia**, alumno en el grado de Ingeniería Informática en la Escuela Técnica Superior de Ingenierías Informática y Telecomunicación de la Universidad de Granada.
- **Dña. Rocío Celeste Romero Zaliz**, profesora del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada, en calidad de tutora del proyecto.

Los roles dentro del proyecto de cada uno de los integrantes del mismo se detalló en el apartado 3.2. En concreto, el tutor del proyecto era el Product owner, y yo como alumno realizaba el resto de roles del proyecto.

5.2.2. Recursos materiales para el desarrollo

Los recursos materiales utilizados para el desarrollo del proyecto han sido muy pocos. En concreto, se han usado los siguientes recursos:

- **Portátil Personal**: Lenovo Legion Y520 con un procesador Intel Core i7-7700HQ, una arquitectura de 64 bits, y una memoria RAM de 16GB. Con este recurso se elabora toda la programación y la documentación del proyecto.

²<https://es.overleaf.com>

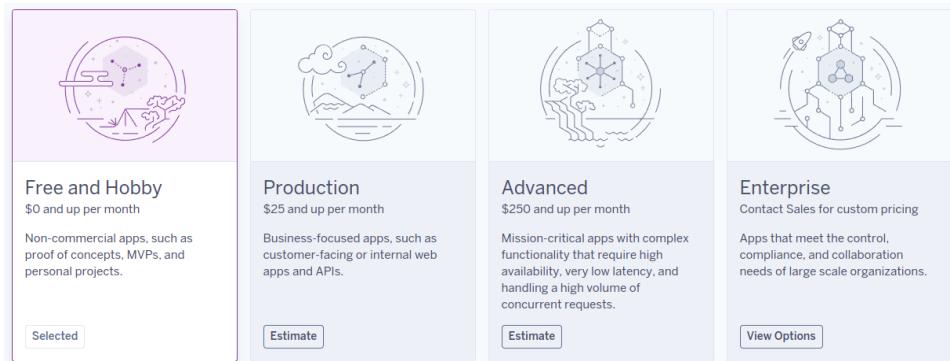
- **Servidores GPU del Instituto DaSCI:** Estos servidores están compuestos por una serie de nodos de cómputo unidos a un nodo cabecera, sobre el cual se hacen las peticiones. Estas peticiones son gestionadas por un gestor de colas, en concreto el gestor es SLURM. Los nodos de cómputo se dividen en varias colas, con la cuenta que se me concedió al poco de iniciar el proyecto, únicamente puedo acceder a los nodos de cómputo situados en la cola *dios*. Dentro de los servidores a cada usuario se le asigna un espacio de almacenamiento donde se puede guardar de forma permanente el código a ejecutar en los servidores.

5.2.3. Recursos software

Los recursos software engloban a todas aquellas herramientas software utilizadas para el desarrollo del proyecto. Siguiendo con la filosofía de intentar reducir al máximo los costes, se han usado en su mayoría herramientas de software libre o gratuitas. Las herramientas utilizadas son las siguientes:

- **Sistema Operativo:** El S.O. empleado es la distribución de Linux más utilizada en el ámbito personal a nivel mundial, Ubuntu. En concreto, se ha usado la versión Ubuntu 20.04.4 LTS. Este sistema operativo es de software libre.
- **GitHub:** Es la plataforma donde se alojará todo el código y la documentación del proyecto. Esta herramienta es una de las plataformas de software libre de alojamiento de código más empleadas a nivel mundial. Además, es usada como gestor de versiones para el proyecto.
- **Git:** Es un software de control de versiones. Es el software empleado por GitHub para el control de versiones de su plataforma.
- **Visual Studio Code:** Es un editor de código fuente. Con este editor se desarrolla todo el código del proyecto. Este editor es de software libre.
- **Heroku:** Es una plataforma que proporciona servicios de computación en la nube. Aunque no es en su totalidad un software libre, para este proyecto he hecho uso de la versión gratuita de la plataforma. El inconveniente de esta versión gratuita es que no se puede utilizar el sistema en producción, solamente para hacer pruebas. Para tener una idea de los distintos planes que existen en Heroku, en la Figura 5.1 se muestran todos los planes de pago, y adicionalmente en la Figura 5.2 se muestran todos los niveles de dynos que existen en Heroku. Un dyno es un contenedor, ya que Heroku divide su potencia de cómputo en unidades, y esas unidades son los dynos. Adicionalmente, es interesante también poder visualizar los planes que existen para añadir una

base de datos a la app alojada en Heroku, estos planes para la base de datos se pueden ver en la Figura 5.3.



Fuente: <https://www.heroku.com/pricing>

Figura 5.1: Planes de pago sobre las apps de Heroku

| Free | Standard | Performance | Private |
|--|--|--|---|
| <p>Try Heroku with no commitment. See full specs →</p> <p>550-1,000 dyno hours per month</p> <ul style="list-style-type: none"> Deploy with Git and Docker Custom domains Container orchestration Automatic OS patching <p>Add to estimate</p> | <p>Run business apps in production. See full specs →</p> <p>\$25-\$50 per dyno per month Prorated to the second</p> <ul style="list-style-type: none"> Includes all Hobby features Simple horizontal scalability App metrics and threshold alerts Preboot and zero-downtime deploys Unlimited background workers <p>Standard 1X \$25 Choose for lightweight apps and APIs that can boot with 512MB RAM. Add to estimate</p> <p>Standard 2X \$50 Choose for greater web concurrency or compute-intense background workers needing more CPU or 1GB RAM. Add to estimate</p> | <p>Run high traffic, low latency apps. See full specs →</p> <p>\$250-\$500 per dyno per month Prorated to the second</p> <ul style="list-style-type: none"> Includes all Standard features Predictable performance for your highest traffic applications Dedicated resources Autoscaling Can mix with Standard dynos <p>Performance M \$250 Choose for optimizing concurrency over Standard 2X. Comes with 2.5GB RAM. Add to estimate</p> <p>Performance L \$500 Choose for extremely high concurrency and parallelism for max throughput for your most latency-sensitive, highest traffic apps. Comes with 14GB RAM. Add to estimate</p> | <p>Run apps needing network isolation, dedicated resources, and greater control. See full specs →</p> <ul style="list-style-type: none"> Full network isolation Available in six global regions Dedicated runtime environment Private network and data services S, M, and L dyno types available <p>Contact Sales for custom pricing</p> |
| <p>Small side projects and concepts. See full specs →</p> <p>\$7 per dyno per month Prorated to the second</p> <ul style="list-style-type: none"> Includes all Free features Free SSL Automated certificate management Never sleeps <p>Add to estimate</p> | | | <p>High compliance apps. See full specs →</p> <ul style="list-style-type: none"> Dedicated environment for high compliance apps Ability to sign BAAs for HIPAA compliance PCI compliance Keystroke logging Space level log drains Strict TLS enforcement <p>Contact Sales for custom pricing</p> |

Fuente: <https://www.heroku.com/pricing>

Figura 5.2: Planes de pago sobre los dynos de Heroku

| Hobby Dev | Standard | Premium | Private |
|---|---|---|---|
| <p>Try Heroku Postgres at no risk. See full specs →</p> <p>\$0 per month For student, personal, and hobbyist use.</p> <p>Add to estimate</p> | <p>Database plans for commercial apps in production. See full specs →</p> <p>\$50 and up per month Prorated to the second</p> <ul style="list-style-type: none"> • Max downtime 1 hour per month • Roll back up to 7 days • Database log metrics • Encryption-at-rest • Fork and follow support <p>Choose Standard for most general purpose web apps.</p> <p>Add to estimate</p> | <p>High availability and heavy workloads. See full specs →</p> <p>\$200 and up per month Prorated to the second</p> <ul style="list-style-type: none"> • Max downtime 15 minutes per month • Roll back up to 7 days • Database log metrics • Encryption-at-rest • Fork and follow support • The highest level of availability and data protection • HA with automatic failover <p>Choose Premium for apps with workloads that require extreme availability.</p> <p>Add to estimate</p> | <p>Databases for network-isolated apps. See full specs →</p> <p>For apps that need network-isolated, private data services to store sensitive data.</p> <p>Contact Sales for custom pricing</p> |
| <p>Hobby Basic</p> <p>Small side projects and proof-of-concepts. See full specs →</p> <p>\$9 per month Prorated to the second</p> <p>For expanded proof-of-concepts, dev and test, and hobbyist use.</p> <p>Add to estimate</p> | | | <p>Shield</p> <p>High compliance. See full specs →</p> <p>For apps that need to store sensitive, regulated data to meet compliance requirements, such as HIPAA and PCI.</p> <p>Contact Sales for custom pricing</p> |

Fuente: <https://www.heroku.com/pricing>

Figura 5.3: Planes de pago sobre las bases de datos de Heroku

- **BlenderBot:** Es una herramienta de software libre para la creación del chatbot.
- **Dialogflow:** Es una herramienta para el despliegue del chatbot. Se hará uso de la versión gratuita de la herramienta.
- **SweetAlert2:** SweetAlert es un plugin de jQuery y con el cual podemos dar un aspecto profesional a los mensajes que lancemos a los usuarios acorde a las tendencias actuales. Además, tenemos la posibilidad de configurar el plugin de muchas formas diferentes.
- **Docker:** Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.
- **Uvicorn:** Uvicorn es una implementación de servidor web ASGI para Python.
- **FastAPI:** FastAPI es un marco de trabajo moderno, rápido (de alto rendimiento), para la construcción de API's con Python 3.6+ basado en sugerencias de tipo estándar de Python.
- **Ngrok:** Ngrok es un servicio que nos permite crear nuestro servidor local en un subdominio para poder visualizarlo fuera de la LAN, a través de internet.

5.2.4. Recursos de comunicación y documentación

Todo buen proyecto debe ser resultado de una buena comunicación entre su personal. Además, se debe elaborar una documentación de calidad para conseguir que el proyecto sea accesible por personas ajenas al desarrollo del proyecto. Para conseguir estos objetivos se ha hecho uso de varias herramientas, entre las cuales se encuentran las siguientes:

- **Google Meet:** Es un servicio de videotelefonía desarrollado por Google. Este servicio se ha utilizado para realizar las distintas reuniones con el tutor.
- **Correo UGR:** Es un servicio de correo electrónico de la UGR. Este servicio se ha utilizado para realizar conversaciones con el tutor sobre temas puntuales o de menor peso.
- **Overleaf:** Es una plataforma de edición de documentos escritos en LaTeX. Esta plataforma se ha usado para la elaboración de la documentación del proyecto.
- **Texmaker:** Es un editor empleado para desarrollar documentos con LaTeX.
- **Visual Paradigm:** Es una herramienta de modelado de diagramas de desarrollo software. En principio es una herramienta de pago, pero la UGR dispone de licencias de uso educativo para los estudiantes.
- **Microsoft Powerpoint:** Es un programa de presentación desarrollado por Microsoft. Con este programa se ha elaborado la presentación del proyecto.

5.3. Gestión de costes

Todo proyecto debe tener su gestión de costes para evaluar la viabilidad económica del proyecto. En la gestión de costes se tiene en cuenta el coste derivado de cada uno de los recursos citados en el apartado 5.2.

5.3.1. Costes de recursos humanos

Los costes de recursos humanos vienen derivados del trabajo realizado por el equipo de desarrollo, como en este proyecto este equipo está formado por una única persona, el alumno, el coste de recursos humanos será el equivalente al trabajo realizado por un ingeniero informático durante el desarrollo del proyecto, es decir, se simula que el alumno se trata de un ingeniero informático que trabaja de forma autónoma.

Para estimar este coste de recursos humanos es necesario estimar el coste de ese trabajo y para ello es necesario recabar información.

En primer lugar, es necesario calcular el tiempo que ha estado trabajando el ingeniero. Dado que el proyecto tiene una duración de 5 meses.

$$\text{Días totales} = 5 \text{ meses} * 30 \text{ días/mes} = 150 \text{ días}$$

El tiempo total aproximado será de 150 días. A este tiempo total habrá que descontar los días no laborables. Los días no laborables en estos 5 meses serían aproximadamente:

$$\text{Días no laborables} = 5 \text{ meses} * 8 \text{ días/mes} = 40 \text{ días}$$

Por lo tanto, los días laborables gastados por el trabajador son:

$$\text{Días laborables} = \text{Días totales} - \text{Días no laborables} = 110 \text{ días}$$

Además, dentro de cada día laborable hay una jornada laboral. Normalmente, esta jornada es de 8 horas, pero como el desarrollo del proyecto se ha hecho en conjunto con la realización de otras asignaturas, no se ha dispuesto de todo el tiempo, por lo tanto, se estimará que aproximadamente se ha trabajado unas 5 horas al día. En consecuencia, el número total de horas trabajadas es el siguiente:

$$\text{Horas laborables} = \text{Días laborables} * 5 \text{ horas/día} = 550 \text{ horas}$$

Este número de horas excede en gran medida el número de horas correspondientes a los ECTS de un TFG. El número mínimo de horas que se debe emplear en el desarrollo de un TFG deben ser 300 horas.

Una vez tenemos el tiempo trabajado por el ingeniero, es necesario obtener qué salario tiene este trabajador a la hora. Para obtener esta información me he basado en el estudio de la página talent.com³. En este estudio se analiza el sueldo medio de un ingeniero informático en España, el salario medio es de 2.208 € al mes. Por lo tanto, el salario medio por hora será de 13,59 €.

Con la información recabada ya podemos estimar el coste del trabajo realizado por el ingeniero. El coste total se puede ver en la Tabla 5.1.

5.3.2. Costes de recursos materiales

Los costes derivados de los recursos materiales se calculan de una forma distinta a la realizada en los costes de recursos humanos, por la simple razón de que los recursos materiales se van devaluando con el paso del tiempo tras su adquisición. Por esta razón deberemos calcular el coste de los distintos recursos materiales con la depreciación aplicada.

Para calcular el coste actual del recurso se debe obtener previamente la siguiente información:

³<https://acortar.link/MiqarT>

| Recurso | Precio / Hora | Nº Horas | Importe |
|------------------|---------------|----------|-----------|
| Trabajo autónomo | 13,59 €/ hora | 550 | 7474,50 € |
| Total: | | | 7474,50 € |

Cuadro 5.1: Costes asociados a los recursos humanos

- Coste de adquisición del producto
- Coste residual del producto, es decir, el valor del producto al final de su vida útil
- Coeficiente de amortiguación lineal

Nos vamos a centrar en el portátil personal, ya que a pesar de que los servidores GPU del Instituto DaSCI tienen un coste de uso, al ser estudiantes de la UGR el coste para nuestro proyecto ha sido nulo y de esta forma lo hemos indicado en el cálculo de los costes; y en el caso de usar servidores en la nube ajenos a la UGR, el coste de estos servidores suele tener un coste fijo cada mes a través de una cuota mensual o en ocasiones el coste de estos servidores se calcula según el tiempo de uso que se haya hecho de estos servidores.

El coste de adquisición del producto es de 1100 €. Como coste residual fijamos la cantidad de 100 €. Y por último queda fijar el coeficiente de amortiguación lineal, para fijar este valor he recabado información sobre el coeficiente de amortiguación lineal de los equipos electrónicos. Esta información la encontré en un artículo de la página web Iberley⁴, donde se indica que el coeficiente de amortiguación lineal máximo para estos equipos electrónicos es del 20 %.

Para calcular el coste actual del producto se utiliza la siguiente fórmula:

$$\begin{aligned}
 C_{ac} &= (C_{ad} - C_{re}) * C_{amor} \\
 C_{ac} &: \text{Coste actual} \\
 C_{ad} &: \text{Coste adquisicion} \\
 C_{re} &: \text{Coste residual} \\
 C_{amor} &: \text{Coeficiente amortiguacion}
 \end{aligned}$$

Siguiendo esta fórmula, el coste actual del portátil es de 200 €. Pero este coste indica el coste anual del recurso, por lo que habrá que calcular el coste del recurso teniendo en cuenta el tiempo de empleo. El tiempo de uso es el mismo que el tiempo trabajado por el alumno, es decir, 5 meses trabajando los 5 días laborables de la semana a razón de 5 horas por día. Basándonos

⁴<https://www.iberley.es/temas/tablas-amortizacion-i-sociedades-30681>

en los cálculos del anterior apartado, el total de horas laborables es de 550 horas. Por lo que el coste final del portátil es el siguiente:

$$\text{Coste final} = 200 * \frac{550}{1200} = 91,66 \text{ euros}$$

Al igual que se hace con los costes asociados a los recursos humanos, en este caso se puede ver un resumen de los costes asociados a los recursos materiales en la Tabla 5.2.

| Recurso | Precio / Hora | Nº Horas | Importe |
|-------------------|---------------|----------|---------|
| Portátil personal | 0,16 €/ hora | 550 | 91,66 € |
| Total: | | | 91,66 € |

Cuadro 5.2: Costes asociados a los recursos materiales

5.3.3. Costes de recursos software

En este apartado se estima el coste de todos los recursos software listados en el apartado 5.2.3. Tal y como se indica en este apartado, todos los recursos software utilizados son de software libre, y en concreto con Heroku, aunque no es un software gratuito, sí que dispone de una versión gratuita que tiene ciertas limitaciones, pero que nos sirve para el desarrollo del proyecto.

5.3.4. Costes de recursos de comunicación y documentación

En este apartado se estima el coste de todos los recursos para la comunicación con la tutora y la elaboración de la documentación. Al igual que pasa con el apartado anterior, tal y como se indica en el apartado 5.2.4, todos los recursos empleados son de software libre, aunque hace falta puntualizar que la herramienta Visual Paradigm es de pago, pero como la UGR dispone de licencias gratuitas para los alumnos, se ha aprovechado esta ventaja para la elaboración de los diagramas del diseño del proyecto.

5.3.5. Costes adicionales

Dentro de los costes adicionales se tiene el coste derivado de todos aquellos recursos que no hayan sido listados en ninguno de los anteriores apartados. Los principales recursos de este apartado son el Internet y la factura de la luz. La tarifa de Internet contratada tiene un coste de 30 €/mes, por lo tanto, el coste total por el Internet asciende a 150 €. Y en cuanto a la factura de la luz, es un coste más difícil de calcular, dado que hay muchas variables que intervienen en su valor, pero una posible estimación sobre su

valor podría ser 25 €. El coste total derivado de estos recursos adicionales se puede ver en la Tabla 5.3.

| Recurso | Importe |
|-------------------|-----------------|
| Internet | 150,00 € |
| Factura de la luz | 25,00 € |
| Total: | 175,00 € |

Cuadro 5.3: Costes asociados a los recursos adicionales

5.4. Presupuesto total

Para tener una visión global de todos los costes analizados se ha elaborado una tabla donde se desglosa toda la información de cada apartado y finalmente se calcula el coste acumulado de todo el proyecto. Estos datos se pueden ver en la Tabla 5.4.

| Recurso | Importe |
|--------------------------------------|------------------|
| Costes de recursos humanos | 7474,50 € |
| Trabajo autónomo | 7474,50 € |
| Costes de recursos materiales | 91,66 € |
| Portátil personal | 91,66 € |
| Servidores GPU de la UGR | 0,00 € |
| Costes de recursos software | 0,00 € |
| Sistema Operativo | 0,00 € |
| GitHub | 0,00 € |
| Git | 0,00 € |
| Visual Studio Code | 0,00 € |
| Heroku | 0,00 € |
| BlenderBot | 0,00 € |
| Dialogflow | 0,00 € |
| SweetAlert2 | 0,00 € |
| Docker | 0,00 € |
| Uvicorn | 0,00 € |
| FastAPI | 0,00 € |

| | |
|---|-----------------|
| Ngrok | 0,00 € |
| Costes de recursos de comunicación y documentación | 0,00 € |
| Google Meet | 0,00 € |
| Correo UGR | 0,00 € |
| Overleaf | 0,00 € |
| Texmaker | 0,00 € |
| Visual Paradigm | 0,00 € |
| Microsoft Powerpoint | 0,00 € |
| Costes adicionales | 175,00 € |
| Internet | 150,00 € |
| Factura de la luz | 25,00 € |
| <hr/> | |
| Total: 7741,16 € | |

Cuadro 5.4: Presupuesto total del proyecto

Capítulo 6

Diseño

6.1. Arquitectura Modelo-Vista-Controlador

Para facilitar la escalabilidad del sistema se va a seguir el patrón de diseño de arquitectura Modelo-Vista-Controlador (MVC). Esta arquitectura permite un bajo acoplamiento entre los elementos del sistema. Tal y como indica el nombre de la arquitectura, sus componentes principales son tres:

- Modelo
- Vista
- Controlador

Teniendo cada uno de los componentes una funcionalidad clara y en gran medida independiente del resto de componentes.

Las principales ventajas de esta arquitectura son las siguientes:

- Permite la programación paralela e independiente de cada uno de los componentes de la arquitectura.
- Posibilita la independencia en el funcionamiento de los componentes.
- Facilita la gestión de errores al ser un sistema modular, dado que los errores estarán más aislados y se podrán abordar de forma más eficiente, y por supuesto también porque se pueden sustituir esos módulos con errores por otros en buen estado.
- Simplifica la escalabilidad del sistema al poder conectar nuevos módulos de cada uno de los componentes de la arquitectura, sin tener que modificar el resto de componentes.
- Permite una importante separación entre los datos y su representación visual.

Pero todas estas ventajas tienen un coste que hay que asumir, y este coste es principalmente un aumento de complejidad en todos los ámbitos. Por un lado, provoca un aumento de la cantidad de código necesaria, este aumento de código implica tener un mayor número de archivos que mantener; y por último dificulta la curva de aprendizaje del sistema.

A continuación se mostrará una breve definición de cada uno de estos componentes.

Vista

La Vista es el objeto que maneja la presentación visual de los datos representados por el Modelo, es decir, es la capa sobre la que se muestran los resultados esperados y con la que realiza la interacción el usuario.

Controlador

El Controlador es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo. Centra toda la interacción entre la Vista y el Modelo, es decir, es la capa encargada de centralizar todas las conexiones entre el resto de capas, manejando las entradas, transfiriendo las al modelo de forma correcta y devolviendo la respuesta del Modelo a la Vista. Además, es el encargado de modificar el Modelo en caso de que sea necesario.

Modelo

El Modelo es el objeto que representa la información del programa. Maneja los datos y controla todas sus transformaciones, es decir, contiene y maneja toda la información del sistema, salvo el conocimiento específico del Controlador o de la Vista.

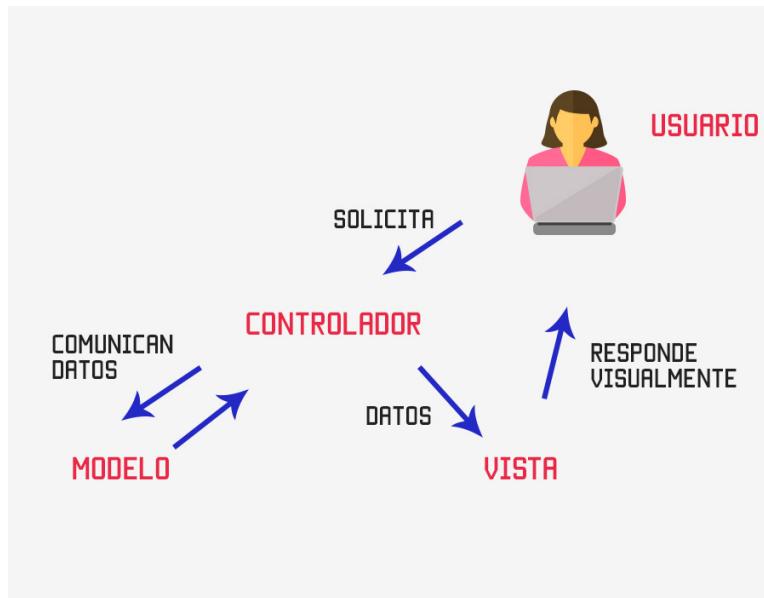
Ciclo de vida de MVC

Las conexiones entre las distintas partes están muy bien definidas y acotadas para tener un bajo acoplamiento.

El Controlador recibe las entradas del usuario, y transmite las peticiones al Modelo, el cual realiza las tareas de la petición y devuelve el resultado al Controlador; y este a su vez devuelve la respuesta a la Vista para que el usuario pueda observar los resultados de su entrada (Figura 6.1).

6.2. Diseño de la Vista

La Vista de nuestro sistema estará compuesta por el conjunto de páginas web que forman el sitio web donde se permitirá el acceso al chatbot, y por las distintas interfaces de uso del chatbot.



(Fuente: <https://codigofacilito.com/articulos/mvc-model-view-controller-explicado>)

Figura 6.1: Ciclo de vida de MVC

Dado que un objetivo del proyecto es conseguir tener una interfaz sencilla (Tabla 3.3). Un paso para llegar a este objetivo es el adecuado diseño del movimiento que se puede realizar dentro del sitio web. Una manera sencilla de visualizar este diseño es a través de los sitemaps, que no es más que un esquema visual donde se representan las distintas secciones del sitio web mediante figuras y las conexiones entre estas secciones como flechas que unen las distintas figuras. El sitemap de nuestro sitio web se puede observar en la Figura 6.2.

Como se puede ver en nuestro sitemap (Figura 6.2), el diseño hecho es sencillo, ya que solamente existe una actividad en todo el sitio web, que es la actividad del chatbot. Además, dentro de la actividad mencionada solamente hay dos opciones. Estas opciones deciden el canal por el que se efectuará la conversación con el chatbot. Aunque como se puede apreciar en el sitemap, pasada la elección del canal parece que el diseño se complica, pero en realidad no es así, ya que esta parte del diseño simplemente quiere mostrar como existen dos vías para el acceso a las interfaces con las que hacer uso del chatbot. Estas dos vías se diferencian por un paso, en una de las vías se realiza la captura de imágenes para la deducción de la edad del usuario, mientras que por la otra vía se rechaza esta captura de imágenes y el usuario indica explícitamente su edad, pasando directamente a las interfaces sin tener que

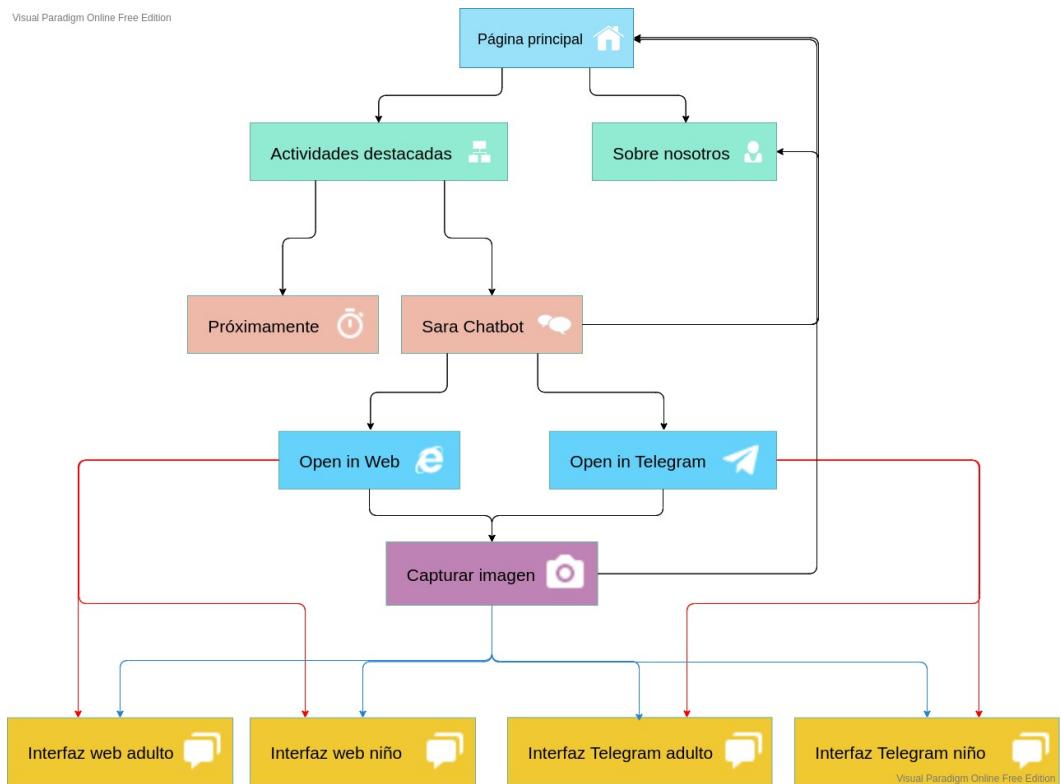


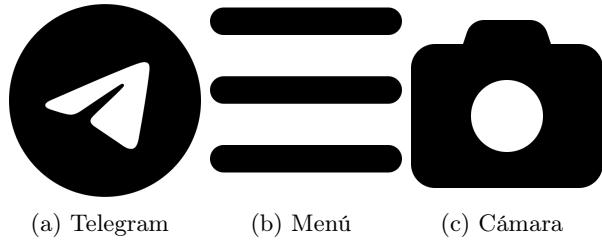
Figura 6.2: Sitemap de la Vista

pasar por la sección de captura de imágenes. Otra parte a destacar del diseño es la sección de las interfaces, porque como se puede comprobar hay varias secciones de interfaz, incluso varias con el mismo nombre. Esto es debido a que habrá un tipo de interfaz por cada canal disponible para el chatbot, y además dentro de las interfaces de cada canal habrá un tipo de interfaz por cada categoría de edad que exista en el sistema. Por último, cabe destacar que en general en todas las secciones habrá links en la barra de navegación que permitan la vuelta a secciones anteriores para facilitar un movimiento rápido y sencillo por el sitio web.

Otra ayuda para conseguir una interfaz sencilla es el uso de iconos. Estos iconos facilitan la rápida adaptación del usuario al sitio web. Estos iconos permiten la abstracción de cierta información, que de otro debería ser indicada de forma implícita. Para obtener estos iconos se ha hecho uso de páginas web que disponen de conjuntos de iconos gratuitos, recalco lo de gratuitos porque normalmente dentro de estas páginas web existen iconos que son de pago, los cuales pueden tener mayor calidad o mejor diseño. Las principales páginas que se han utilizado son Font Awesome¹ y Boxicons². Algunos de

¹<https://fontawesome.com/icons>

²<https://boxicons.com/>



(a) Telegram (b) Menú (c) Cámara

Figura 6.3: Iconos usados en el sitio web

los iconos empleados son los mostrados en la Figura 6.3.

Todos los iconos de la Figura 6.3 abstraen algún concepto de la página web. Por ejemplo, el ícono de Telegram se coloca en un botón para deducir rápidamente que ese botón realiza alguna acción relacionada con la aplicación Telegram.

Otra parte del diseño de la vista que influye en la sencillez del sitio web es el diseño que se haga de cada una de las secciones de la página web. Dado que con el sitemap se consigue un diseño demasiado abstracto, también es útil elaborar los bocetos web o wireframes. Estos bocetos consisten en una representación visual de un sencillo esquema de la página web. En este caso se han realizado los bocetos de las siguientes secciones:

- **Página principal:** Está compuesta por una breve descripción del evento que aloja la actividad del chatbot de nuestro sistema, y la zona de actividades, donde se recomienda una serie de actividades (Figura 6.4a).
- **Actividad del chatbot:** En esta sección, al igual que pasa en la página principal, se visualiza una breve descripción; pero en este caso esta descripción es sobre la actividad del chatbot. Y además, se muestran los distintos accesos al chatbot a través de los distintos canales disponibles (Figura 6.4b).
- **Captura de imágenes:** Dentro de esta sección volvemos a tener dos secciones, ya que para la gestión de las imágenes se debe tomar la imagen y posteriormente enviar la imagen para la deducción de la edad. En primer lugar, en la sección de captura de la imagen se muestra una ventana que irá mostrando el vídeo captado por la cámara y también se muestra un botón para realizar la captura. Y en segundo punto, en la sección de envío de la imagen se muestra una ventana que muestra la imagen tomada y también se muestra un botón para efectuar el envío (Figura 6.4c).
- **Interfaz web:** Esta sección es simple, ya que sigue el diseño típico de

un chat. Teniendo una entrada de texto y un visor de la conversación, donde aparecen los mensajes encapsulados orientados a la derecha o a la izquierda dependiendo del emisor del mensaje (Figura 6.4d).

- **Interfaz de Telegram:** Esta sección es similar a la sección de la interfaz web, pero adaptado al diseño que tiene la aplicación de Telegram. Esta interfaz no deja de ser un chat de Telegram (Figura 6.4e).

Todas las secciones descritas tienen en común las barras de navegación, tanto la que se encuentra en la parte superior de la página web como la lateral. Estas barras de navegación estarán compuestas por el logo del evento, una serie de links para el rápido movimiento a través del sitio web, un interruptor para el cambio de tema de colores, y un botón para desplegar menú lateral si es necesario según el dispositivo que se esté utilizando.

Llegados a este punto tenemos un claro diseño de los elementos que componen el sitio web; aunque, por otra parte, otro objetivo que influye en el diseño de la vista es la obtención de una interfaz armónica (Tabla 3.4), es decir, una interfaz que tiene colores llamativos para el usuario, pero que no llega a cansarlo rápidamente. Para la elección de colores me he basado en un principio en los colores empleados en la página oficial del evento de La Noche Europea de l@s Investigador@s³. Los colores que he elegido son más claros de los elegidos para la página oficial. Estos colores van desde el azul al blanco. Y en concreto para la tipografía, el negro y el dorado en ciertas ocasiones. Pero los colores anteriormente descritos forman parte del modo claro de la página web. Como se ha comentado en anteriores párrafos de este apartado, cada página web dispone de un interruptor para el cambio al modo oscuro. La utilización de dos modos de colores, uno claro y otro oscuro; está muy extendida actualmente dada la gran acogida del modo oscuro en muchas aplicaciones con gran volumen de usuarios. En la elección para los colores del modo oscuro se debe tener en cuenta que el texto debe seguir siendo legible, para ello se deben elegir colores que tenga mucho contraste entre ellos. Los colores utilizados para nuestro modo oscuro van desde el gris al negro, y en algunas ocasiones el naranja. Y en concreto, para la tipografía, el blanco, el gris en ciertos textos secundarios, y el dorado en ciertas ocasiones.

6.3. Diseño del Controlador

El Controlador deberá ser la unión entre la Vista y el Modelo. Por esta razón, deberá tener entradas que gestionen las peticiones creadas por la Vista, las procesen y las reenvíen al Modelo; y entradas que gestionen las peticiones creadas por el Modelo, como respuesta a las peticiones llegadas desde la Vista, las procesen y las reenvíen a la Vista.

³<https://lanochedelosinvestigadores.fundaciondescubre.es>

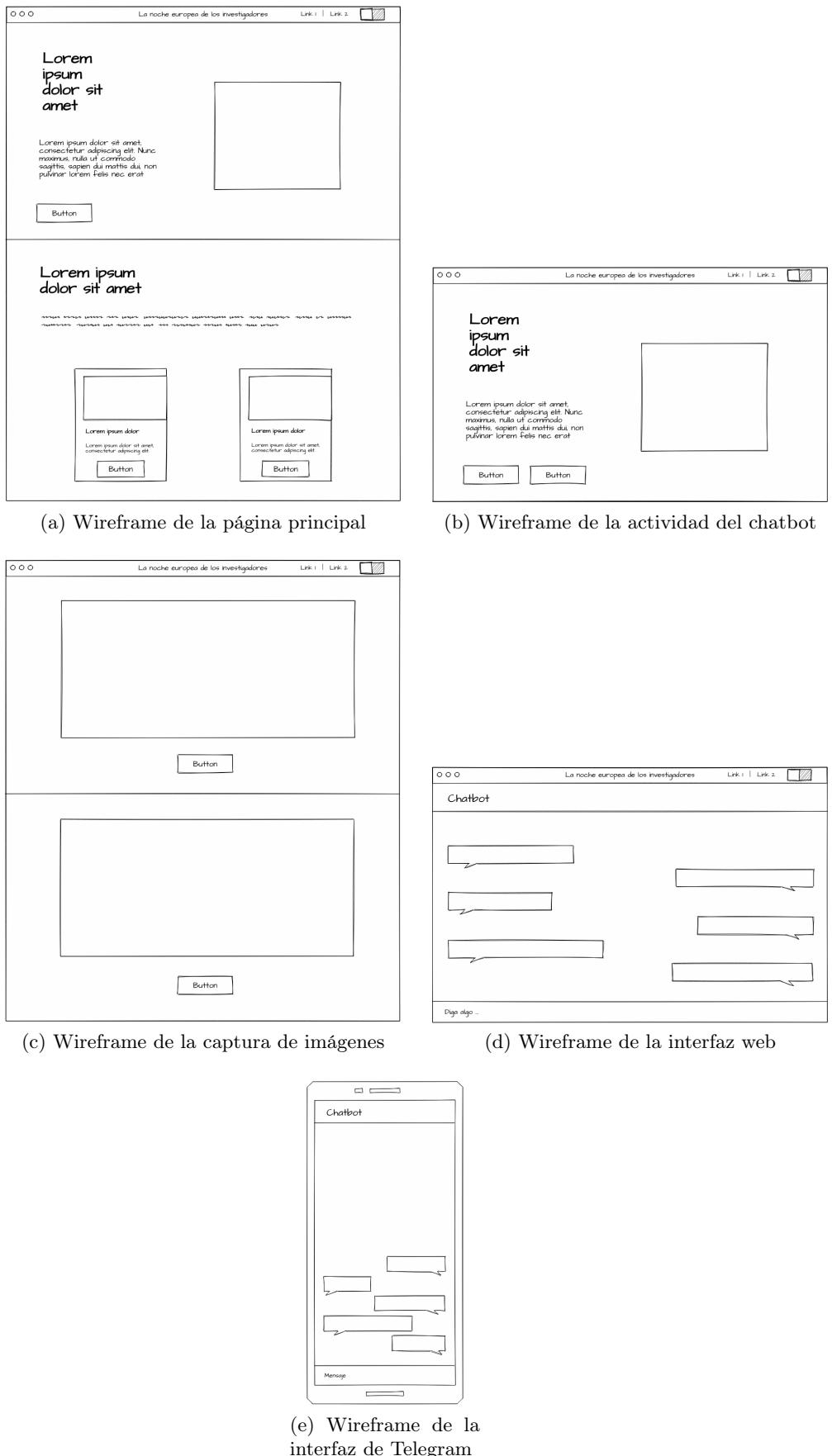


Figura 6.4: Wireframes de la Vista

Este Controlador será un servidor que tendrá al menos las entradas necesarias para gestionar las distintas secciones del sitio web, cuyo diseño se expuso en el apartado de diseño de la Vista (Apartado 6.2). Estas secciones son las siguientes:

- Página principal
- Actividad de chatbot
- Captura de imágenes
- Interfaz web para adultos
- Interfaz web para niños

Las únicas secciones del sitio web que no tendrán una sección en el servidor son las interfaces de Telegram, ya que son gestionadas por la propia aplicación de Telegram. Parte de estas secciones no requerirán una interacción con el Modelo; mientras que otras si lo requerirán, como puede ser la sección de captura de imágenes.

Para tener una visión más abstracta de todo el proceso seguido a través del sitio web hasta llegar a la interfaz del chatbot, se ha elaborado un diagrama de actividad, el cual se puede ver en la Figura 6.5.

En la Figura 6.5 se pueden ver todas los puntos de decisión que tiene el usuario a lo largo de su interacción con el sitio web, y cuáles son los posibles caminos existentes para llegar a alguna de las interfaces del chatbot.

Aparte de todas las secciones mencionadas, el servidor contará con otras secciones que serán definidas en la implementación, ya que serán necesarias por las condiciones impuestas por las distintas herramientas que se usen a la hora de implementar el Controlador.

A parte de indicar el diseño de las secciones del sitio web, es interesante tener una primera imagen de como deben quedar dos de las secciones más importantes del sistema, las cuales son llevadas a cabo por el Controlador. La primera es la gestión de la conversación, la cual se puede ver en la Figura 6.6. Y la segunda es la gestión del procesado de la imagen, la cual se puede ver en la Figura 6.7.

6.4. Diseño del Modelo

El diseño del Modelo de nuestro sistema se divide en varias partes debido a que este Modelo suministra distintos datos al Controlador dependiendo de la parte del Modelo que se utilice.

Algo a destacar es que para implementar todas las funcionalidades del Modelo del sistema se han empleado modelos basados en Transformers basados en Transformers implementados en PyTorch.

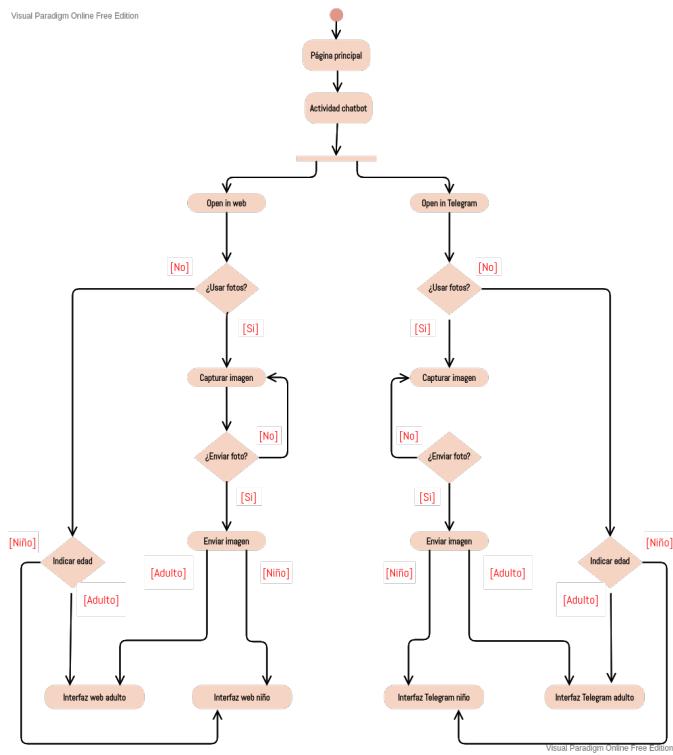


Figura 6.5: Diagrama de actividad del acceso a la interfaz del chatbot

Módulo de generación de respuestas

Esta parte del Modelo es la encargada de proporcionar las respuestas a las entradas proporcionadas por el Controlador. Siguiendo la filosofía del Transfer Learning, la cual fue explicada en el análisis del estado del arte (Apartado 2.1), para la generación de respuestas se va a hacer uso de modelos pre entrenados del modelo BlenderBot. De esta forma facilitamos la creación de un chatbot orientado a la funcionalidad que buscamos.

Módulo de deducción de edad

De igual forma que se pretende hacer con el modelo generador de respuestas, para el módulo de deducción de edad se hará empleo de modelos pre entrenados para el reconocimiento de imágenes.

Log de las conversaciones

Los datos a guardar en este log serán las conversaciones terminadas con los distintos usuarios del sistema. El diseño para modelar estas conversaciones en el log estará compuesto por una única tabla. Los elementos de esta tabla serán los siguientes:

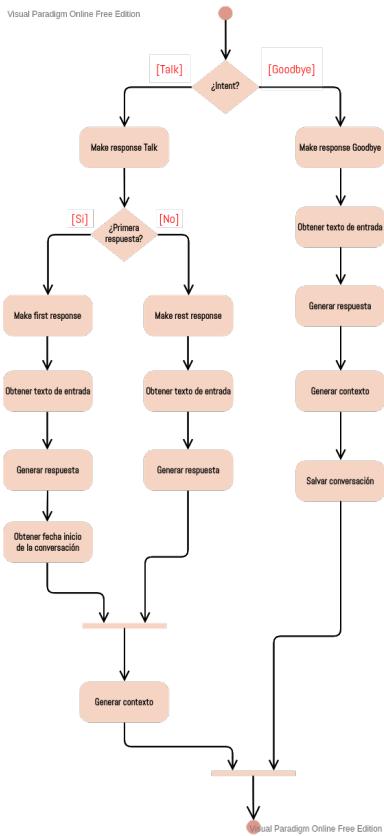


Figura 6.6: Diagrama de actividad de la gestión de la conversación

- Contenido de la conversación
- Fecha de inicio de la conversación
- Fecha de finalización de la conversación
- Duración de la conversación
- Edad del usuario con el que se realizó la conversación
- Identificador de la conversación
- Verificación de la conversación por parte del administrado del sistema

Cabe destacar que el identificador de la conversación será la clave principal de la tabla, la duración de la conversación será un atributo calculado a partir de la fecha de inicio y de finalización, y por último, el contenido de la conversación estará compuesto por la sucesión de entradas y de respuestas que ha recibido el sistema, indicando tanto las entradas como las respuestas en español y en inglés.

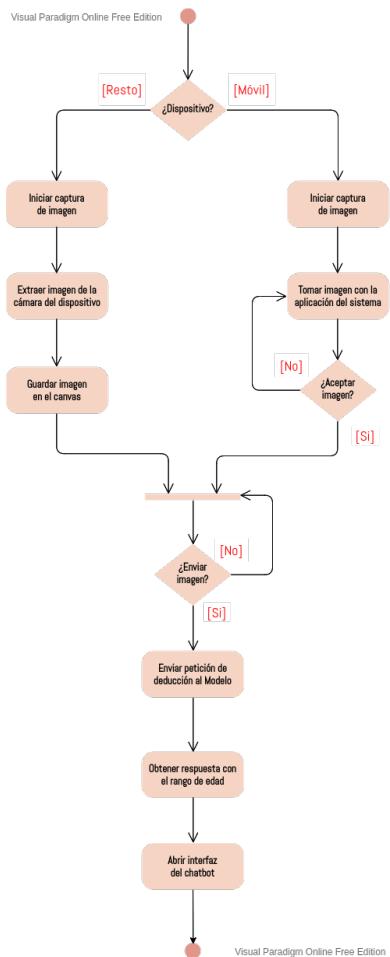


Figura 6.7: Diagrama de actividad de la gestión del procesado de la imagen

Entrenamiento de modelos

A pesar de que utilicemos modelos pre entrenados, los cuales puedan funcionar desde el primer momento, debemos acercar el dominio de los modelos a la funcionalidad que queremos que tengan. Para realizar este cambio de dominio se debe realizar un ajuste (fine-tuning) de los modelos pre entrenados. Se hace un ajuste y no un entrenamiento desde cero por la simple razón de que estamos usando modelo pre entrenados y queremos aprovecharlos del entrenamiento que llevan implícito estos modelos. El ajuste de los modelos requerirá de unos conjuntos de datos. Estos conjuntos de datos serán los que modifiquen el dominio del modelo, por lo tanto, los conjuntos de datos deberán representar el nuevo dominio que queremos que tengan los modelos. El ajuste, a diferencia de un entrenamiento desde cero, consiste en la ejecución de un entrenamiento de unas pocas épocas de duración. Se efectúa este bre-

ve entrenamiento porque queremos adaptar ligeramente el dominio, ya que un movimiento muy brusco provocaría la pérdida del trabajo implícito que traen los modelos pre entrenados. Si necesitamos adaptar mucho el dominio, sería más recomendable buscar otro modelo pre entrenado o entrenar uno propio desde cero.

Generación de los conjuntos de datos

Por supuesto estos conjuntos de datos no vienen elaborados por defecto, debemos generar nuestros propios conjuntos de datos a base de extraer información de distintas fuentes. Deberemos generar los conjuntos de datos con un formato adecuado para su uso en el ajuste de los modelos. Esta parte es la más difícil de llevar a cabo entre todas las partes necesarias para elaborar un chatbot propio, dada la complejidad de encontrar información sobre algunos temas. Todo dependerá de la funcionalidad que queramos que tenga el chatbot, y de la facilidad que encontraremos a la hora de extraer información útil.

Capítulo 7

Implementación

7.1. Implementación de la Vista

7.1.1. Herramientas para el desarrollo

Para la programación del sitio web se ha hecho uso de los lenguajes de programación HTML, CSS, y Javascript. En concreto, se ha utilizado HTML5 y CSS3.

Adicionalmente, se han empleado recursos varios, como pueden ser imágenes extraídas de Internet y un plugin para la creación de ventanas emergentes, SweetAlert2¹. En concreto, este plugin lo forman dos archivos extraídos del repositorio de la aplicación en JSDelivr². Como se puede ver en esta página, la versión empleada de SweetAlert2 es 11.4.17. Los dos archivos se guardan una carpeta concreta llamada plugin que se encuentra junto a la carpeta de imágenes y la carpeta de scripts de Javascript.

7.1.2. Interfaz responsive

Una interfaz responsive es un tipo de interfaz que se adapta al dispositivo donde se esté visualizando el sitio web. Para crear estas interfaces responsive se crean distintos estilos CSS dependiendo del dispositivo, en concreto dependiendo del tamaño de su pantalla. Esta filosofía de creación de sitios web permite generar varios sitios web, definiendo una única vez los elementos que forman el sitio web y definiendo varios estilos que modifiquen los atributos de esos elementos, permitiendo el acceso a un mayor volumen de usuarios al no limitar los dispositivos con los que se puede acceder al sistema. Para obtener más detalles sobre las interfaces responsive de nuestro sistema puedes ver el Apartado B.1.

¹<https://sweetalert2.github.io>

²<https://www.jsdelivr.com/package/npm/sweetalert2>

7.1.3. Modo oscuro

Hasta hace pocos años los sitios web únicamente tenían implementados temas claros, es decir, la tonalidad de la página web se torna a colores más claros y el texto a colores más oscuros. Últimamente, se han empezado a usar masivamente los temas oscuros, que son totalmente lo contrario a los temas claros. Definir ambos temas es sencillo, pero hace falta un mecanismo para pasar de un tema a otro de forma intuitiva y sencilla para el usuario.

El mecanismo usado para cambiar de tema es un interruptor. Este interruptor tiene dos estados, uno para cada tema. Visualmente, el estado del interruptor se muestra distinto en modo inactivo () que en modo activo ().

Este interruptor se encontrará en la barra de navegación de cada una de las secciones de nuestro sitio web. Para obtener más información sobre el funcionamiento y la implementación del interruptor puedes consultar el Apartado B.2.

7.1.4. Menú lateral

Esta funcionalidad forma parte de las interfaces responsive, ya que en algunos dispositivos el ancho de la pantalla no es lo suficientemente amplio como para mostrar toda la información de la barra de navegación de las secciones del sitio web.

Para poder hacer uso de este menú lateral se dispondrá de un botón ubicado en la barra de navegación que desplegará un menú lateral que contendrá la misma información que contendría la barra de navegación si esta se pudiese mostrar en su totalidad con el ancho disponible. La implementación de este mecanismo de despliegue del menú lateral se explica de forma detallada en el Apartado B.3.

7.1.5. Ventanas emergentes

Una herramienta utilizada en el sistema para facilitar la toma de decisiones del usuario son las ventanas emergentes. Con estas ventanas emergentes conseguimos que el usuario preste atención, ya que se tratarán de puntos de inflexión importantes para el funcionamiento del sistema.

El lenguaje Javascript tiene por defecto órdenes para crear ventanas emergentes, pero estas ventanas tienen un estilo muy básico. Para tener ventanas emergentes con un estilo interesante para el usuario se ha utilizado el plugin SweetAlert2. En su página oficial ³ se muestran varios ejemplos de usos de este plugin. Para la creación de mis ventanas me he basado en varios de estos ejemplos.

³<https://sweetalert2.github.io>

Para conocer más detalles de la implementación de estas ventanas y en que decisiones se han aplicado las mismas puedes ver el Apartado B.4.

7.1.6. Interfaces web

La interfaz web utilizada ha sido la proporcionada por las integraciones de Dialogflow. Ya que implementar una interfaz web propia llevaría cierto tiempo que está mejor invertido si se aprovecha en otras secciones del sistema, puesto que la interfaz que proporciona Dialogflow es adecuada para nuestro sistema. La posible ventaja que hemos perdido por no hacer nuestra propia interfaz es la pérdida de control en nuestro sistema, pues ahora hay un elemento que no podemos controlar su funcionamiento entorno y ni podemos modificarlo internamente. Para conocer como se ha integrado esta interfaz en nuestro sistema puedes consultar esta información en el Apartado B.5.

El aspecto final de nuestra interfaz web se puede ver en la Figura B.11.

7.1.7. Interfaces de Telegram

Para dar mayor acceso a nuestro chatbot se han elaborado interfaces para Telegram. De modo parecido a como se hace con la interfaz web, se ha hecho uso de las integraciones de Dialogflow, pero en este caso con Telegram. Para obtener información de que como es el proceso de inclusión de este tipo de interfaz, el cual es en parte parecido al realizado con las interfaces web, se puede consultar en el Apartado B.6.

El aspecto final de nuestra interfaz de Telegram será el que se muestra en la Figura B.14.

7.2. Implementación del Controlador

7.2.1. Herramientas para el desarrollo

Las herramientas usadas en el Controlador son muy diversas debido a la multitud de funcionalidades de este módulo, a su conexión con el resto de módulos y a su despliegue de forma pública.

En la implementación del Controlador se tendrá como lenguaje central al lenguaje de programación Python. La versión de Python utilizada será la versión 3.9.10.

Pero este lenguaje Python se utilizará en conjunto con otros lenguajes y herramientas dependiendo de la región del servidor que se esté implementando.

Para la parte de implementación de las conexiones con la Vista se empleará el lenguaje Javascript para elaborar las peticiones HTTPS que permiten la conexión con el Controlador. Y en pequeña medida se hará uso del lenguaje HTML5 para el paso de información útil.

Para el despliegue del servidor se hará empleo de un conjunto de herramientas como son Docker, Unicorn, y de la plataforma Heroku ⁴. Este despliegue se desarrollará más en detalle en el Apartado 7.2.3.

Para la implementación de la estructura del servidor se hace empleo del lenguaje Python, aunque más en concreto se hace uso de un framework basado en Python como es FastAPI ⁵.

Para el resto de secciones, como puede ser la gestión de la conversación, se hará empleo únicamente del lenguaje Python.

7.2.2. Implementación del servidor

Dado que vamos a implementar el servidor con el lenguaje Python, disponemos de múltiples posibilidades para la implementación del mismo. Desde la implementación utilizando únicamente Python, hasta utilizar frameworks modernos basados en Python, como puede ser FastAPI.

En mi caso, he optado por la utilización del framework FastAPI, debido a que este tipo de framework te permite desarrollar API's rápidas y con un gran rendimiento sin demasiado poner demasiado esfuerzo en su desarrollo. Además, te dan un gran abanico de posibilidades para añadir de forma simple funcionalidades a nuestro servidor. Para obtener las demás razones que me han llevado a elegir este framework en concreto debes consultar el Apartado B.7.

Una vez hemos elegido el framework con el que implementar el servidor, nos disponemos a explicar la forma en la que se realiza esta implementación con este framework en concreto. Muchos de los frameworks basados en Python siguen la metodología de FastAPI para la elaboración de servidores.

Lo primero que se debe hacer es la creación del servidor, para ello se hace uso de la función *FastAPI*. Con esta función obtenemos un objeto con el que podremos definir las distintas secciones del servidor.

La definición de cada una de las secciones se componen de dos pasos. En primer lugar, se define el tipo de peticiones que recibe la sección, GET o POST; la ruta interna de la sección, la cual debe contener como mínimo la ruta raíz '/'; y por último el tipo de respuesta, que en el caso de nuestro servidor la respuesta puede ser código HTML o texto plano. En segundo punto, se define la función que se ejecutará al entrar a esa sección del servidor. Esta función deberá tener unos parámetros y unas salidas que estén en sintonía con el tipo de petición que se haya elegido en el primer paso.

Un ejemplo de definición de una sección del servidor se puede ver en la Figura 7.1.

⁴<https://heroku.com>

⁵<https://fastapi.tiangolo.com>

```
@app.get("/wakeup", response_class=PlainTextResponse)
def wakeup():
    """! Función asociada a la sección de reactivación del servidor.

    @return Mensaje indicando el estado activo del servidor.

    """
    return "Server ON"
```

Figura 7.1: Ejemplo de definición de sección del servidor del Controlador

7.2.3. Despliegue del servidor

Como se ha indicado en la exposición de las herramientas del Controlador (Apartado 7.2.1). Para el despliegue se hace uso principalmente de tres herramientas.

La primera herramienta a utilizar es Uvicorn. Uvicorn es un servidor ASGI basado en uvloop y http tools. Una hemos creado nuestro servidor con FastAPI, empleamos el mismo objeto con el que definíamos las secciones del servidor para ejecutar el servidor. Ayudándonos de la librería de Python de Uvicorn, utilizamos la función *run* para ejecutar nuestro servidor FastAPI, pasándole como argumentos el objeto que define a la aplicación de FastAPI, el puerto en el que el servidor escuchará y la dirección de host. El puerto y la dirección host se pueden obtener tanto del archivo de configuración del servidor como de las variables de configuración de nuestra app en Heroku. Con esta información podremos ejecutar la aplicación de FastAPI mediante el módulo Uvicorn.

Pero hasta este punto únicamente tendríamos un servidor ejecutándose de forma local, situación que no es la conveniente para nuestro sistema, dado que uno de sus objetivos es que pueda ser empleado por cualquier persona y en cualquier momento. Es por ello que en este punto entra en acción el uso de una plataforma como es Heroku, herramienta analizada en el Apartado 5.2.3. Y de la mano de Heroku viene la utilización de la herramienta Docker.

Heroku es una plataforma de tipo PaaS que permite a los desarrolladores implementar, ejecutar y trabajar con aplicaciones que se estén ejecutando en su totalidad en la nube. De esta manera, al ejecutar nuestro servidor en esta plataforma, dispondremos de un punto de acceso público a nuestro sitio web y a todas las funcionalidades que proporciona.

Una vez sabemos de qué se trata la plataforma Heroku, debemos concretar como se traspasa nuestro trabajo realizado de forma local en nuestra máquina a un espacio de trabajo en la nube.

El despliegue a la plataforma de Heroku se realiza mediante la creación de un contenedor del servidor y su posterior subida a una de las apps que tengamos en nuestra cuenta de Heroku. Para obtener información más detallada sobre el proceso de despliegue en la plataforma Heroku puedes consultar el Apartado B.8.

7.2.4. Gestión del sitio web

Una de las ventajas de usar FastAPI para la implementación del servidor es la posibilidad de utilizar cualquier motor para las plantillas HTML. Para el caso de nuestro servidor he optado por Jinja2, que es el motor que se recomienda en la documentación de FastAPI y el cual es frecuentemente empleado en otros frameworks. El único requisito para el uso de este gestor de plantillas es la instalación de la librería de Python *jinja2*.

Para un fácil empleo de este gestor aparecen ejemplos de uso del mismo en la documentación de FastAPI⁶.

Para hacer uso del gestor en nuestro servidor es necesario seguir los siguientes pasos:

- Importar `Jinja2Template` de la librería `fastapi.templating` .
- Montaje de los archivos de la carpeta static, carpeta cuyo contenido es todo aquél distinto de las plantillas HTML.
- Creación del objeto `templates` con el cual se podrán manejar las plantillas
- Declaración de un parámetro de tipo `Request` en cada una de las funciones asociadas a una ruta que devuelve una plantilla HTML. Además, cada una de estas funciones debe devolver una plantilla.
- Generación de una plantilla renderizada mediante el objeto `templates` obtenido en el tercer paso.

Para conocer como se realiza el proceso de generación de las plantillas renderizadas puedes consultar el Apartado B.9.

Con lo comentado hasta este momento podríamos tener un servidor web totalmente funcional en cuanto a lo estético, pero para poder realizar algunas de las funcionalidades del sitio web es necesario hacer uso de las posibilidades que nos da el gestor Jinja2 con el paso de valores a través del diccionario que se comenta en el Apartado B.9. A través de este diccionario se pasará la información que requieran las plantillas y el cual no se pueda saber de forma estática, sino dinámica. Algunos ejemplos de este tipo de información pueden ser el estado del modo oscuro, es decir, si está activado o no; o la URL del servidor GPU, la cual puede cambiar durante la ejecución del Controlador si se inicia varias veces el servidor del Modelo durante la ejecución del servidor del Controlador.

Aunque para acceder a esta información desde los scripts de Javascript se deberá hacer de una manera menos directa a como puede parecer. Para conocer en más profundidad el proceso seguido para solucionar la dificultad de acceso a esta información desde los scripts, puedes consultar el Apartado B.10.

⁶<https://fastapi.tiangolo.com/advanced/templates>

7.2.5. Secciones adicionales del servidor

En este apartado se abordan las secciones del servidor del Controlador que no están orientadas al sitio web, sino que han surgido como solución a algún problema que haya surgido durante el desarrollo del servidor. En concreto, estamos hablando de dos secciones. Por un lado, una sección para la reactivación del servidor cada cierto tiempo; y, por otro lado, una sección para la actualización de la URL pública del servidor del Modelo.

Sección para la reactivación del servidor

Por supuesto, la versión gratuita de Heroku tiene sus inconvenientes, principalmente enfocados en la actividad del servidor. El primer inconveniente se trata de una limitación en el uso de los recursos que nos proporciona la plataforma. Como se indicó en el apartado de gestión del proyecto, la plataforma Heroku dispone de unas unidades llamadas dynos, que son las encargadas de ejecutar nuestras apps. Con la versión gratuita, que es la que estamos utilizando, disponemos de una cuota de 550 horas gratuitas de utilización de los dynos. Esta cuota se renueva mensualmente. Esta información se puede comprobar en la sección de la documentación de Heroku sobre el empleo de los dynos gratuitos⁷. Además, esas 550 horas son horas de utilización con un único dyno, en el caso de usar dos dynos para nuestra app, dispondríamos de 275 horas gratuitas en total, es decir, la mitad. Si comparamos esta cantidad de horas, 550 horas, con la cantidad de horas de las que dispone como máximo un mes, las cuales son 744 horas, vemos como el número de horas que nos proporciona la versión gratuita se queda corta para poder proporcionar un servicio continuo del servidor. En el caso de que nuestra app supere la cuota de uso de nuestra cuenta, todos los dynos que estén ejecutando nuestras apps entrarán de forma forzosa en estado inactivo, dejando sin servicio a todas nuestras hasta que se vuelve a renovar la cuota de uso.

Para poder superar esta diferencia de horas sin tener llegar al apagado forzoso de los dynos, debemos hallar una solución para aumentar nuestra cuota de uso hasta llegar a una cuota que nos permita dar un servicio continuo. La plataforma Heroku nos da la posibilidad de aumentar esta cantidad de horas simplemente introduciendo los datos de la tarjeta de crédito en nuestra cuenta, sin tener que realizar ningún pago. La plataforma nos dará 1000 horas por el simple hecho de introducir estos datos. De esta manera superamos las 744 horas que puede tener como máximo un mes del año.

Un tip que es interesante para llevar el control del consumo de nuestra cuota de utilización de los dynos es hacer uso de Heroku CLI mediante el comando *heroku ps -a namegt;*, con el cual obtendremos el porcentaje de la cuota que ya ha sido usado y el número de horas restante de la cuota. Aunque alternativamente se puede controlar este consumo de la cuota desde

⁷<https://devcenter.heroku.com/articles/free-dyno-hours#usage>

el apartado de pagos de nuestra cuenta, por si el administrador del sistema le es más agradable tener una interfaz gráfica para realizar este control.

Pero si analizamos el primer inconveniente, vemos como no tiene influencia en la implementación del servidor, y así es, aunque este no es el único inconveniente derivado del uso de la cuenta gratuita. El otro inconveniente de este tipo de cuentas es que los dynos que están ejecutando nuestro sitio web entran en estado inactivo al trascurrir cierto tiempo sin actividad en el servidor, de esta forma la plataforma no está gastando recursos de forma innecesaria. Esta desconexión de los dynos al no detectar actividad en la app se explican en la sección de la documentación de Heroku sobre la inactividad de los dynos⁸. Este paso a estado inactivo es un problema, ya que cuando se vuelve a tener actividad en el servidor se deberá volver a iniciar el servidor, lo que conllevará una espera que no será agradable para el usuario. A diferencia del primer inconveniente, Heroku no nos proporciona una solución a este segundo inconveniente, por lo que habrá que buscar una solución provisional para este inconveniente. Para más detalles sobre la solución a esta problemática, ver el Apartado B.11.

Después de aplicar estas dos soluciones tendremos un servidor disponible en cualquier momento.

Sección para la actualización de la URL pública del servidor del Modelo

La gestión de la actualización de la URL pública del servidor es de igual forma importante para asegurar un servicio continuo del sitio web. Ya que puede darse el caso de que el servidor del Modelo se reinicie por cualquier motivo y cambie la URL pública del servidor. Esto sucede porque la URL pública del servidor se genera nuevamente cada vez que se inicia el servidor y cada vez que se genera es distinta. Esto sucede por la forma en que se ha implementado, lo cual se explicará en la sección de implementación del Modelo (Apartado 7.3).

Para gestionar este cambio dinámico de la URL, se ha creado una sección en el servidor del Controlador, donde se actualiza la variable que contiene la URL pública del servidor del Modelo. Esta sección se llama *setURL*. Esta sección recibirá peticiones POST del servidor del Modelo, cuando se necesite actualizar la URL. Se necesitará actualizar la URL en los siguientes casos:

- Inicio del servidor del Modelo
- Reinicio del servidor del Modelo
- Inicio o reinicio del servidor del Controlador cuando el servidor del Modelo ya está activo.

⁸<https://devcenter.heroku.com/articles/free-dyno-hours#dyno-sleeping>

Con el último caso descrito se deberá ejecutar una sección del servidor del Modelo para realizar la actualización. Este proceso se describirá en la sección de implementación del Modelo (Apartado 7.3).

La lógica de la sección generada es simple. Únicamente se recibe una petición POST que contiene la URL pública del servidor del Modelo, se extrae esa URL de la petición y finalmente se asigna el valor de la URL a la variable que contendrá el valor de la misma para poder ser usado por el resto de secciones del servidor.

7.2.6. Procesado de las imágenes

El procesado de las imágenes se realiza de forma distinta dependiendo del dispositivo que se esté utilizando. Aunque todo el procesado de las imágenes realizado por el Controlador se ha implementado en scripts de Javascript.

Si se usa un móvil, esta sección únicamente se compone de una parte, dado que el mismo botón de captura de la imagen lanza el envío de la imagen. Al pulsar sobre el botón de captura se lanzará la aplicación del sistema, la cual se encarga de capturar una imagen y preguntar al usuario si quiere utilizar esa imagen para la deducción de la edad. Una vez hemos aceptado cuál será nuestra imagen, el usuario será preguntado por el sistema por el permiso para el envío de la imagen. Si el usuario acepta el envío, se procederá a realizar el mismo. El envío consistirá en la lectura de la imagen en formato base64, seguido del envío de la imagen dentro de una petición POST a la ruta de deducción de edad del servidor del Modelo. Como respuesta a esta petición obtendremos el rango de edad que se ha deducido. Dependiendo de este rango de edad se abrirá una interfaz u otra, pero también habrá que tener en cuenta el canal que ha elegido el usuario. El canal elegido, la URL del servidor del Modelo, y la dirección a cada una de las interfaces disponibles se pasará al script a través del paso de valores con el módulo Jinja2, tal y como se indica en el Apartado 7.2.4.

En el caso de usar otro dispositivo distinto del móvil, esta sección estará compuesta de dos partes. En la primera parte se realiza el mismo proceso que se realiza en la aplicación del sistema en el caso del móvil, aunque en este caso en vez de ver la imagen en la aplicación de captura de imágenes del sistema, se verá un elemento que mostrará el vídeo captado por la cámara del dispositivo y otro elemento que será un canvas que almacenará la imagen que se captura del vídeo mencionado, ya que un vídeo no deja de ser una visualización continua de muchas imágenes. Una vez hemos capturado la imagen y la tenemos en el canvas, tendremos un botón para iniciar el envío de la imagen. Este botón lanza la misma ventana que en el otro caso para pedir el permiso de envío de la imagen. El envío de la imagen se efectúa de forma similar a como se hace en el otro caso, cambiando únicamente la forma en que se obtiene la imagen, ya que en vez de leer la imagen de un archivo, se debe extraer la imagen almacenada en el canvas y pasarl a formato base64.

La información externa que se necesita para realizar el envío se obtiene de igual forma con el paso de valores con el módulo Jinja2.

7.2.7. Gestión de la conversación

La manera de gestionar la conversación por parte del Controlador está muy condicionada por el uso de las interfaces que proporciona Dialogflow. Dado que, como se ha mencionado en la implementación de la Vista, todo lo que sucede dentro de la interfaz no se puede controlar, ni siquiera los mensajes. Por ello se debe hacer uso de una funcionalidad que tiene Dialogflow, que es el Webhook. En esta sección de Dialogflow se deberá introducir una URL que servirá como punto de conexión entre Dialogflow y el servidor que funcione como webhook del chatbot. La URL que se colocará será aquella que nos lleve a la sección de webhook del servidor del Controlador. Habrá una sección de webhook por cada rango de edad, ya que habrá que gestionar por separado las interfaces de adulto y de niño. En cada uno de los Intents del chatbot de Dialogflow se activará la llamada al webhook, de esta forma cuando se active uno de los Intent se realizará el envío de una petición POST al webhook. Esta petición contendrá información de la conversación, como puede ser el Intent que se ha activado, el texto de entrada que ha activado al Intent o el contexto del chatbot.

Para conocer en detalle como se realiza la gestión de las peticiones recibida desde Dialogflow puedes consultar el Apartado B.12.

7.2.8. Guardado de la conversación en el log

Para facilitar la implementación del log de nuestro sistema, se ha hecho uso de las múltiples funcionalidades que proporciona Heroku a través de sus Add-ons. En concreto, existe el Add-ons llamado Heroku Postgres, el cual crea una base de datos Postgres que es accesible desde la app de Heroku, y que hará la función de log del sistema. Este Add-ons se puede añadir de forma gratuita a nuestra app. Al instalar este Add-ons se añade de manera automática la URL del log a las variables de configuración de nuestra app. Este último punto es importante, ya que será necesaria esta URL para realizar las conexiones con el log desde nuestro código del servidor del Controlador.

Para conocer como se realiza la conexión entre el servidor y el log a nivel de código puedes consultar el Apartado B.13.

Como se indicó en el diseño del Controlador, la duración de la conversación era un atributo calculado. Para obtener su valor se ha creado un disparador que se activa cada vez que se realiza una inserción en la tabla de las conversaciones. Este disparador activa una función que recibe la fecha de inicio y la fecha de fin de la conversación y calcula el tramo de tiempo que ha durado la conversación.

7.3. Implementación del Modelo

7.3.1. Herramientas para el desarrollo

Las herramientas usadas en el Modelo, al igual que pasa en el Controlador, son muy diversas. Aunque hay ciertas herramientas que coinciden en ambas partes del sistema.

Toda la implementación del Modelo se realizará con el lenguaje de programación Python. La versión de Python utilizada será la versión 3.9.10.

Las herramientas comunes se encuentran en la sección de implementación y despliegue del servidor del Modelo. En cuanto a la implementación de la estructura del servidor, se hace empleo del framework FastAPI. Para el despliegue, a diferencia de lo que pasa con el Controlador, se deja de utilizar la herramienta Docker. Aunque se podría usar la herramienta Docker en caso de que fuese útil para una mejora de la calidad del sistema. La herramienta que si se emplea para el despliegue del servidor es la herramienta Uvicorn. Otra herramienta que no se emplea para el despliegue es la plataforma Heroku. No se hace uso de esta plataforma por la simple razón de que no dispone de apps con servicio de GPU. Por ello, para realizar el despliegue del servidor del Modelo se ha empleado la herramienta Ngrok, para realizar el despliegue de forma local en el servidor GPU del Instituto DaSCI y poder generar un punto de acceso público al servidor del Modelo.

Una parte fundamental del Modelo es la generación de respuestas, ya que es básica para la elaboración del chatbot. Como se ha comentado en el diseño del Modelo, esta generación de respuestas se va a basar en modelo pre entrenados del chatbot BlenderBot. Es en este punto cuando interviene el uso de la plataforma Huggingface⁹. Esta plataforma sirve como una base de modelos que son subidos por la comunidad. El modelo pre entrenado utilizado para toda la generación de respuestas es el modelo blenderbot-400M-distill de la empresa Facebook.

Si nos fijamos en el lenguaje empleado por el modelo de generación de respuestas, vemos que trabaja con texto en inglés, por lo tanto, será necesario traducir el texto de español a inglés y viceversa. Para nuestro sistema hemos empleado la API de DeepL, la cual nos permite tener un traductor varias veces mejor que el resto de traductores disponibles actualmente.

Otra sección del Modelo que ha hecho uso de la plataforma Huggingface, es la deducción de la edad a través de imágenes. Para reducir el tiempo de desarrollo del sistema sin dejar tener un sistema fiable, he optado por usar un modelo pre entrenado de clasificación de imágenes, el cual es capaz de clasificar una imagen de una persona en un rango de edad concreto. El modelo utilizado es vit-age-classifier del desarrollador nateraw.

⁹<https://huggingface.co/>

7.3.2. Implementación del servidor

Dado que en la sección de herramientas se ha indicado que la implementación de la estructura del servidor se realizaría con el framework FastAPI, la implementación llevada a cabo para el servidor del Modelo no diferirá mucho de la implementación del servidor del Controlador, la cual se puede ver en el Apartado 7.2.2. Una de las diferencias será el número y la funcionalidad de las secciones del servidor, y la otra diferencia estará relacionada con el intercambio de información entre los servidores. Esta última diferencia viene originada por la propia estructura de nuestro sistema, ya que debemos implementar dos servidores con dominio distinto que se comunican entre ellos. Este intercambio de información tiene especial importancia en las comunicaciones entre la interfaz y el servidor del Modelo. Dado que la interfaz del sistema se encuentra en el dominio del servidor del Controlador y el servidor del Modelo se encuentra en un dominio distinto, deberemos añadir ciertas cabeceras en las conexiones entre ambas partes del sistema.

Para obtener una información detallada de las cabeceras añadidas a las conexiones con el servidor del Modelo, puedes consultar el Apartado B.14.

7.3.3. Despliegue del servidor

Para el despliegue del servidor se hace uso de dos herramientas, una de ellas ya empleada para el despliegue del servidor del Controlador.

La herramienta que ya ha sido utilizada es Uvicorn. Esta herramienta, como ya se explicó en el despliegue del Controlador, es un servidor ASGI el cual nos permite poner en funcionamiento la estructura de nuestro servidor, la cual se ha creado con FastAPI. Para poner en funcionamiento el servidor será necesario definir el puerto y la dirección host que empleará el módulo Uvicorn.

Llegados a este punto tenemos un servidor ejecutando de forma local, por lo tanto, no hay forma de acceder al servidor desde el exterior del sistema. A diferencia de lo aplicado con el servidor del Controlador, no se hará uso de la plataforma Heroku, dado que necesitamos tener disponibles GPU's para la utilización de los modelos basados en Transformers. Por ello debemos mantener el servidor ejecutando de forma local, pero no en nuestra máquina, sino en el servidor GPU del Instituto DaSCI. Pero nos apoyaremos en la herramienta Ngrok para posibilitar un acceso público al servidor del Modelo. Ngrok es un servicio que permite el acceso a un servidor a través de internet, creando el servidor local en un subdominio.

Dado que para la implementación se hace empleo del lenguaje Python, no se usará la herramienta Ngrok en sí, sino una librería de Python llamada pyngrok¹⁰.

¹⁰<https://github.com/alexdlaird/pyngrok>

Una vez que sabemos con qué vamos a posibilitar el acceso al servidor, falta detallar la manera en que se hará. El proceso seguido para desplegar el servidor mediante la herramienta Ngrok se explica detalladamente en el Apartado B.15.

7.3.4. Generación de los conjuntos de datos

Por supuesto que los modelos que generan las respuestas del chatbot son una parte principal del sistema, pero no podemos disfrutar de ellos, si no disponemos de buenos conjuntos de datos con los que entrenar a los modelos.

Un conjunto de datos para entrenar un modelo conversacional consiste simplemente en una tabla con dos columnas, por un lado, las preguntas o entradas; y, por otro lado, las respuestas o salidas.

En un principio no disponíamos de información suficiente para crear un chatbot. Por esta razón optamos por extraer la información de la página Quora¹¹. Esta fase es la más costosa tanto en tiempo como en dinero a la hora de crear un chatbot. La página Quora es una red social de preguntas y respuestas, donde existen conversaciones sobre distintos temas. Actualmente, existen ciertos módulos para la extracción de información especializados en esta página, pero en mi caso he optado por generar mi propio programa de extracción de información para obtener los conjuntos de datos con el formato deseado. Este programa extraerá la información de la página y generará una tabla con todos los datos extraídos. Para obtener más detalles de esta extracción de información se puede ver el Apartado B.17.

Disponiendo de este programa ya podremos extraer un conjunto de datos inicial con información sobre los temas que nos interesen.

El conjunto de datos que obtendremos no tendrá el formato adecuado para ser usado por los modelos, por ello será necesario realizar un pre procesado al conjunto de datos. Además, como disponemos de dos modelos, uno para adultos y otro para niños, y únicamente tenemos un conjunto de datos. A la vez que ejecutamos el pre procesado deberemos tratar los textos del conjunto de datos de tal forma que se adapten al rango de edad al que van destinados. Por lo tanto, tras el pre procesado del conjunto de datos inicial obtendremos como resultados dos conjuntos de datos, uno para adultos y otro para niños. Ambos con el formato adecuado para ser utilizados por los modelos.

Para conocer cuál es el proceso seguido para realizar el pre procesado puedes consultar el Apartado B.16.

7.3.5. Entrenamiento de modelos

El siguiente paso lógico, tras haber generado los conjuntos de datos, es el entrenamiento de los modelos pre entrenados mediante los conjuntos de datos

¹¹<https://www.quora.com/>

obtenidos. Como ya se ha indicado en anteriores apartados, el entrenamiento de nuestros modelos consistirá en un ajuste de los mismos, es decir, en un entrenamiento de unas pocas épocas. Para realizar el ajuste se elaborará un programa que realizará este proceso. Durante la ejecución del programa se irá mostrando por pantalla información detallada sobre el estado del proceso. Esta información va desde la época en que nos encontramos hasta las métricas obtenidas en ese punto del entrenamiento.

Por supuesto, lo primero que debemos hacer si queremos entrenar unos modelos es efectuar la carga de los mismos. Un modelo se compone de tres partes principales, su configuración, su tokenizer, que es el módulo encargado de pasar de texto a lista de tokens y viceversa; y el propio modelo.

Y en segundo lugar, debemos cargar otras de las partes fundamentales para el entrenamiento, los conjuntos de datos. Se deben cargar los conjuntos de datos de entrenamiento y validación del rango de edad que se quiera aplicar al modelo cargado previamente. Estos conjuntos de datos pueden ser cargados en su totalidad o parcialmente. Pero tras la carga no termina el trabajo con los conjuntos de datos, dado que los modelos trabajan con tokens no con cadenas de caracteres. Entonces debemos aplicar un pre procesado a los conjuntos de datos cargados, para tener conjuntos de datos cuyo contenido sean tokens.

Una vez tenemos todos los elementos para proceder al entrenamiento, debemos crear el objeto que realiza el entrenamiento, el cual hará uso de todos los elementos anteriormente descritos. Este entrenador será el encargado de ir entrenando el modelo cargado basándose en las métricas que vaya obteniendo. El cálculo de las métricas se efectuará en cada época, al igual que el guardado de modelos. Finalmente, se guardará el mejor modelo encontrado durante el proceso de ajuste, al igual que las métricas obtenidas con ese modelo.

Este programa podrá también evaluar el modelo cargado, guardando las métricas obtenidas.

Todas las métricas obtenidas durante todo el programa se guardarán en un archivo para su posible análisis futuro.

La métrica utilizada para guiar todo el proceso de entrenamiento ha sido la métrica BLEU. BLEU (Bilingual Evaluation Understudy) es un algoritmo para evaluar la calidad de un texto traducido por una máquina de una lengua natural a otra. Se considera que la calidad es la correspondencia entre el resultado de una máquina y el de un humano. Cuanto más se acerque una traducción automática a una traducción humana profesional, mejor será. El BLEU fue una de las primeras métricas en afirmar una alta correlación con los juicios humanos sobre la calidad, y sigue siendo una de las métricas automatizadas más populares y económicas.

Las puntuaciones se calculan para segmentos individuales traducidos, comparándolos con un conjunto de traducciones de referencia de buena calidad. A continuación, esas puntuaciones se promedian en todo el corpus para

obtener una estimación de la calidad global de la traducción. No se tienen en cuenta ni la inteligibilidad ni la corrección gramatical.

7.3.6. Generación de respuestas

La generación de las respuestas se realiza en dos secciones distintas del servidor del Modelo, una por cada rango de edad que tiene disponible el chatbot. Estas secciones serán las encargadas de recibir las peticiones del Controlador, generar la respuesta y devolver esa respuesta como respuesta a la petición del Controlador. Por supuesto, antes de iniciar el servidor deberemos cargar los modelos de los distintos rangos de edad. Esta carga se hará mediante la función *from_pretrained*, para asegurarnos de que únicamente un proceso pueda cargar el modelo y el vocabulario al mismo tiempo. Para esta sección se hará uso de dos herramientas adicionales. La primera herramienta son los pipelines, en nuestro caso los pipelines conversacionales. Estos pipelines facilitan la generación de respuestas al poder crear instancias de las conversaciones, las cuales guardan un registro de las entradas y sus correspondientes respuestas más recientes. La segunda herramienta, y puede que la más importante de este módulo, es el módulo deepspeed. Este módulo es un optimizador de los modelos basados en Transformers. Este módulo permite que la generación de respuestas se haga de forma casi instantánea, permitiendo una respuesta del chatbot en un tiempo muy breve. Para realizar esta optimización se pasan todos los modelos utilizados en los pipelines por este módulo, mediante la función *init_inference*. Una funcionalidad que no se ha usado de este módulo, pero la cual se podría llegar a usar, es el uso conjunto de varias GPU's para la generación de respuestas. Para nuestro proyecto únicamente se hace uso de una GPU.

La petición del Controlador no contiene únicamente el texto de entrada, sino también el identificador de la conversación y un indicador para saber si se trata de la última respuesta de la conversación.

La generación de respuestas consistirá en los siguientes pasos:

- Traducir al inglés el texto recibido en la petición del Controlador
- Obtener la conversación actual mediante su identificador
- Añadir el texto traducido como entrada a la conversación obtenida
- Generación de la respuesta mediante el pipeline
- Actualización de la conversación añadiendo la respuesta generada
- Traducir al español la respuesta
- Elaborar la respuesta a la petición del Controlador

Por último, cabe destacar que la generación llevada a cabo por el pipeline se basa en una serie de parámetros, los cuales serán definidos en la configuración del proceso de ajuste. Principalmente, la generación se basa en dos parámetros, *temperature* y *top_p*. Ambos parámetros modifican la flexibilidad del chatbot a la hora de generar texto.

Un punto a destacar, que no se puede pasar por alto es la gestión del historial de las conversaciones, ya que para que la generación de respuestas tenga sentido se debe tener un historial de la conversación. Este historial se va generando a través de los resultados que van devolviendo los pipelines durante la generación de las respuestas. Pero para poder identificar cada conversación con un usuario se debe mantener un control de las mismas, puesto que las peticiones al servidor del Modelo no van en un principio enlazadas con ninguna conversación. Este control se realiza mediante un diccionario, cuya clave es el identificador de la conversación. Este diccionario se irá actualizando conforme vayan progresando las conversaciones activas en el sistema. El identificador de la conversación se irá transmitiendo junto a las peticiones que efectúe el Controlador, y finalmente en la última respuesta se eliminará la entrada asociada a la conversación indicada en la petición.

7.3.7. Deducción de edades

A lo largo de la implementación del Modelo se ha visto como se hace diferenciación según el rango de edad del usuario, pero todavía no hemos visto como llegamos a saber cuál es el rango de edad del usuario.

Antes de comenzar con la explicación sobre como se realiza la deducción, cabe destacar dos aspectos importantes en esta deducción. En primer lugar, el límite de edad entre el rango de edad de niño y de adulto se encuentra a los 12 años. Para nuestro sistema un niño tendrá como máximo 12 años inclusive. Una vez sabemos como se dividen los rangos de edad en nuestro sistema, es el momento de indicar el segundo aspecto. Este segundo aspecto son los distintos rangos de edad que el modelo pre entrenado utilizado, es capaz de clasificar. Los rangos de edad del modelo son los siguientes:

- 0 - 2 años
- 3 - 9 años
- 10 - 19 años
- 20 - 29 años
- 30 - 39 años
- 40 - 49 años
- 50 - 59 años

- 60 - 69 años
- Más de 70 años

Como se puede apreciar, el límite de edad entre los dos rangos de edad del sistema se encuentra incluido en el tercer rango de edad de clasificación del modelo. La solución a este problema se irá indicando conforme se explique la implementación de esta sección.

Una vez explicados estos dos aspectos comenzamos con la explicación de la implementación de la sección de deducción. Si el usuario dio permisos al sistema para usar imágenes, el Modelo recibirá peticiones desde el Controlador, las cuales contendrán imágenes del usuario. Estas imágenes no serán guardadas en ningún momento, para asegurar la privacidad de los usuarios; además, las peticiones con las cuales se envían las imágenes son totalmente seguras al ser del tipo HTTPS.

Como se ha indicado en el diseño del Modelo, para la deducción de la edad se hace uso de un modelo basado en Transformers, cuya funcionalidad es la clasificación de personas basándose en su edad.

En primer lugar, se deberá cambiar el formato de la imagen recibida en la petición para poder pasarla como entrada al modelo. Una vez tenemos la imagen en el formato adecuado, realizaremos la clasificación con el modelo. La salida del modelo será una lista de probabilidades. Esta lista tendrá tantos elementos como rangos de edad puede llegar a clasificar el modelo. Una vez tenemos esta lista, nos quedaremos con la mayor probabilidad. Para solucionar el problema de tener el límite en medio de un rango de edad se utilizará una heurística para decidir el rango de edad del usuario. Si la mayor probabilidad se encuentra en el primer o segundo rango, se indicará que es un niño. Si la mayor probabilidad se encuentra en el cuarto rango de edad o posteriores, se indicará que es un adulto. Y para el rango de edad restante, el tercer rango de edad, la decisión se basará en un cálculo. Se calculará la suma de las probabilidades desde el inicio de la lista hasta el tercer rango inclusive, y otra suma desde el tercer rango hasta el final de la lista inclusive. Si la primera suma es mayor o igual a la segunda se indicará que es un niño, en caso contrario se indicará que es un adulto. Con el uso de esta heurística deberíamos poder solucionar ese problema, pero no lo sabremos hasta que no hagamos pruebas con el sistema. Una vez se sabe el rango de edad a devolver, se realiza la devolución de la respuesta a la petición del Controlador.

Capítulo 8

Pruebas

En este punto deberíamos tener un sistema totalmente funcional, pero nos falta comprobar la calidad del mismo. La calidad de nuestro sistema vendrá determinada por la calidad de cada uno de sus elementos. En este apartado se realizarán pruebas sobre las principales partes funcionales del sistema. Estas pruebas irán variando según la funcionalidad de la parte examinada.

8.1. Pruebas sobre la interfaz web

La interfaz web es el conjunto de elementos que se muestran al usuario a través del navegador. Las pruebas aplicadas sobre esta parte del sistema son simples, consisten en un recorrido por todos los posibles caminos existentes en la interfaz web. Durante este recorrido por la interfaz web se verá si hay zonas poco intuitivas, si hay fallos en los redireccionamientos, si hay caminos que no desembocan en alguna de las interfaces del chatbot, entre otros fallos a revisar en la interfaz.

Para obtener unos resultados más objetivos, las pruebas han sido ejecutadas por personas externas al desarrollo del proyecto. A cada uno de los usuarios que ha realizado la prueba, la cual consistía en un uso libre del sistema durante 10 minutos, al finalizar el término establecido se le realizaba una pregunta sobre la puntuación que le daba al sistema del 0 al 10.

La puntuación media obtenida para nuestro sistema ha sido de un 7 sobre 10. Los comentarios de los usuarios destacan que la interfaz web es bastante funcional, pero que la parte que le hacía perder nota era su estética, que podría ser mejorable añadiendo diseños más actuales.

Para tener más datos objetivos con los que evaluar la calidad de la interfaz web, se ha efectuado un test de accesibilidad a la interfaz en su totalidad. Hoy en día hay cierta tendencia a mejorar la accesibilidad para mejorar el empleo de las interfaces web por parte de las personas con alguna discapacidad que les dificulta el uso de las mismas. Para obtener una interfaz con una gran nivel de accesibilidad existen las pautas de accesibilidad al contenido

web (Web Content Accessibility Guidelines - WCAG). Estas pautas son unos documentos que explican cómo hacer el contenido web de nuestro sistema accesible para personas con discapacidad. Actualmente, existen muchos comprobadores online para obtener el nivel de accesibilidad de nuestro sistema. Existen distintos niveles de conformidad para categorizar nuestro sistema, estos niveles son el nivel A, AA y AAA. Los niveles mencionados van desde menos restrictivos a más restrictivos. El nivel AA contiene las restricciones del nivel A y las propias del nivel AA. Y de igual forma pasa con el nivel AAA, que contiene las restricciones de los dos niveles anteriores y las propias del nivel AAA.

El comprobador online que he utilizado para analizar la accesibilidad de nuestra interfaz web ha sido Siteimprove. La puntuación que se ha obtenido de la interfaz web con este comprobador online se puede comprobar en la Tabla 8.1. En esta tabla se puede ver tanto la puntuación media de la interfaz web como la puntuación desglosada de cada sección de la interfaz web. La puntuación de cada página se divide en cuatro puntuaciones distintas. Estas cuatro puntuaciones son la puntuación general de cada página y las tres puntuaciones de los niveles A, AA y AAA.

| | Puntuación general de la página | Puntuación nivel A | Puntuación nivel AA | Puntuación nivel AAA |
|-------------------------------|---------------------------------|--------------------|---------------------|----------------------|
| Página principal | 69.9 / 100 | 84.5 / 100 | 71.3 / 100 | 53.9 / 100 |
| Actividad del chatbot | 77.1 / 100 | 84.5 / 100 | 71.3 / 100 | 75.6 / 100 |
| Página de captura de imágenes | 69.1 / 100 | 84.7 / 100 | 71.3 / 100 | 51.2 / 100 |
| Interfaz del chatbot | 61.2 / 100 | 91.1 / 100 | 57.4 / 100 | 34.9 / 100 |
| Interfaz web | 69.33 / 100 | 86.2 / 100 | 67.83 / 100 | 53.9 / 100 |

Cuadro 8.1: Puntuación de accesibilidad de la interfaz web

Basándonos en las puntuaciones mostradas en la Tabla 8.1, podemos indicar que el nivel de accesibilidad no es muy alto en general en nuestra interfaz web. Sobre todo destaca la baja puntuación en los niveles más restrictivos. Se consideraría que el nivel de accesibilidad es aceptable si se llega a tener una puntuación media de 85 sobre 100, en al menos los dos primeros niveles. Por lo tanto, la accesibilidad de nuestro sistema podría ser un punto a mejorar, sobre todo si el volumen de usuarios que van a usar nuestro sistema está formado por un gran número de usuarios con discapacidad. De todas maneras, el aumento de la accesibilidad nunca será inconveniente para la experiencia de los usuarios con la interfaz, sino todo lo contrario. Aunque, por supuesto, sí que conllevaría un aumento de la carga de trabajo a la hora de implementar la interfaz web.

8.2. Pruebas sobre la deducción de edad

Otra parte fundamental del sistema es la deducción de la edad. Como se indicó en el apartado de implementación del Modelo, se ha hecho uso de

un modelo pre entrenado para realizar esta deducción. Para comprobar la calidad de este modelo, y en consecuencia la calidad del sistema para deducir la edad, se han efectuado una serie de deducciones con imágenes extraídas de una base de imágenes libres llamada Pexels. Las imágenes que se han extraído de esta base de imágenes se pueden ver en la Figura 8.1. Como se puede apreciar, se han extraído imágenes de caras de personas de distintas edades para intentar forzar al modelo y ver su verdadera calidad.

Tal y como se expuso en el apartado de implementación de esta sección de deducción, el modelo pre entrenado que estamos utilizando clasifica las edades dentro de unos ciertos rangos. El rango más comprometido es el rango que va desde los 10 hasta los 19 años, ya que para nuestro sistema una persona deja de ser niño a los 12 años. Por lo tanto, el límite entre niño y adulto para nuestro sistema se encuentra en medio de un rango de edad. La solución a este problema ya se explicó en el mismo apartado de su implementación.

La primera imagen (Figura 8.1a) pertenece claramente a un hombre adulto. El modelo dedujo que se trataba de un adulto; por lo tanto, el modelo realizó un buen trabajo.

La segunda imagen (Figura 8.1b) pertenece a una mujer adulta. Con esta imagen se intenta comprobar si el modelo es capaz de deducir de igual forma con hombres que con mujeres. El modelo dedujo que se trataba de un adulto.

La tercera imagen (Figura 8.1c) pertenece a un niño con una edad en torno a los 7 años. El modelo debería ser capaz de indicar que se trata de un niño sin mucha dificultad. La deducción del modelo fue correcta, indicando que se trataba claramente de un niño.

La cuarta imagen (Figura 8.1d) pertenece a una chica con una edad en torno a los 14 años, por lo tanto, el modelo debería indicar que es adulta para el sistema. Pero el resultado del modelo fue indicar que se trataba de una niña. Las posibles causas de estos resultados podría ser la cercanía de la edad real de la chica al límite entre el rango de edad de niño y de adulto; o los posibles filtros en la cara de la chica, los cuales hacen que la chica parezca más joven.

La quinta imagen (Figura 8.1e) pertenece a un niño con una edad en torno a los 12 años. Tanta proximidad al límite de edad podría afectar a los resultados del modelo. El modelo dedujo que se trataba de un niño. Basándonos en estos resultados, podríamos decir que la anterior deducción pudo fallar en mayor medida por los filtros de la imagen. Además, cabe destacar que no ha afectado al modelo la alta luminosidad de la imagen.

La sexta y última imagen (Figura 8.1f) pertenece a una niña con una edad en torno a los 12 años, al igual que pasaba con la anterior imagen. En este caso, la deducción del modelo fue indicar un rango de edad de adulto, y no por poco porcentaje. Las posibles causas de este resultado fallido podrían ser los rasgos de la niña, la cual aparece tener una edad superior a la real.

Basándonos en los resultados de las pruebas, podemos indicar que el modelo acierta en la mayoría de los casos, acertando la totalidad de los casos



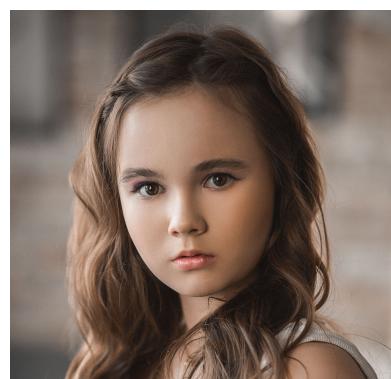
(a)



(b)



(c)



(d)



(e)



(f)

Figura 8.1: Imágenes de las caras usadas para las pruebas

donde está claro el rango de edad. Además, es capaz de deducir correctamente cerca del límite si las condiciones de la imagen son adecuadas para el modelo. Por supuesto, se podría mejorar la calidad entrenando al modelo con las imágenes en las que ha fallado, pero para nuestro sistema la calidad del modelo es más que suficiente.

8.3. Pruebas sobre la generación de respuestas

En esta sección se probarán dos elementos del sistema. En primer lugar, el módulo traductor, ya que el modelo generador de respuestas trabaja con frases en inglés. Por lo tanto, será igual de importante realizar bien la traducción como generar bien la respuesta.

Para comprobar la calidad de las traducciones efectuadas por DeepL, voy a efectuar varias traducciones de frases hechas que son originarias de España, las cuales no pueden ser traducidas literalmente para obtener su verdadero significado en inglés. Los resultados de las traducciones se pueden ver en la Tabla 8.2.

| Español | Significado | Inglés |
|-----------------------------------|--|--|
| Estoy hasta el moño | Indica que ya no se puede soportar una situación o a una persona | I'm sick of it |
| Voy a pasar el puente en la playa | Indica que se va a disfrutar de un periodo breve de vacaciones | I'm going to spend the long weekend at the beach |
| Cuesta un ojo de la cara | Indica que algo cuesta mucho dinero | Costs a pretty penny |
| Meter la pata | Indica que se ha fallado o fracasado en una tarea o actividad | Screw up |
| Estar metido en el ajo | Indica que se está al corriente de algo | To be in the thick of it |

Cuadro 8.2: Pruebas de traducción con la API de DeepL

Como se puede apreciar en la Tabla 8.2, las traducciones de DeepL son de gran calidad, por lo que esta buena traducción facilitará la generación de respuestas y en consecuencia la comunicación con el chatbot.

El segundo elemento a probar es el módulo generador de respuestas. Las pruebas a realizar sobre este módulo son clara. En este caso se expondrá una conversación de unas pocas de frases donde se verá como se desenvuelve el chatbot. Para estas pruebas se utilizará el chatbot para adultos, aunque es indiferente cuál de los modelos se elija. Las conversaciones realizadas para la prueba se pueden ver en la Tabla 8.3 y en la Tabla 8.4.

En la primera conversación (Tabla 8.3) se puede ver como el chatbot saluda adecuadamente, y seguidamente le pregunta por su profesión al usuario. Cuando este le indica su profesión empieza a interesarse por ella, preguntando por información sobre la profesión. Además, se puede ver como entiende cuando el usuario está cansado de un tema, sabe detectar cuál es el tema sobre el que estaba hablando y pasa a otro tema totalmente distinto, como son los aires acondicionados. El problema que se refleja en esta conversación

| Emisor | Texto enviado por el emisor |
|---------|---|
| Usuario | Buenos días |
| Chatbot | ¡Buenos días! ¿Cómo está usted? |
| Usuario | Me encuentro genial |
| Chatbot | ¿A qué te dedicas? |
| Usuario | Soy piloto de aviones comerciales |
| Chatbot | ¿Cuánto tiempo lleva siendo piloto comercial? |
| Usuario | En febrero hago tres años de piloto |
| Chatbot | ¿Cómo se inicia en el pilotaje comercial? |
| Usuario | Lo primero es ingresar en la escuela de vuelo |
| Chatbot | ¿Cómo puedo transferir la información de mis pilotos de una plataforma a otra? |
| Usuario | Podemos dejar de hablar de trabajo porque he llegado cansado de trabajar |
| Chatbot | ¿Qué hacen los acondicionadores de aire? |
| Usuario | Tengo entendido que se utilizan para refrescar o calentar una habitación |
| Chatbot | ¿Cómo funcionan los aparatos de aire acondicionado? |
| Usuario | No tengo ni idea. ¿Tú sabes como funcionan los aires acondicionados? |
| Chatbot | El aire acondicionado es un tipo de sistema que sirve para regular la temperatura de un lugar |
| Usuario | ¿Pero cómo funcionan mecánicamente? |
| Chatbot | El aire convierte el calor en impulsos eléctricos en el aire |

Cuadro 8.3: Primera conversación de las pruebas del módulo de generación de respuestas

es la inmovilidad del chatbot a cambiar de un entorno de pregunta-respuesta a un entorno más natural.

En la segunda conversación (Tabla 8.4) se tiene un inicio distinto. En primer lugar, se realiza una presentación donde el chatbot indica su nombre y propone al usuario indicar un tema de conversación. Seguidamente, el usuario no sugiere un tema de conversación, he indica su estado de ánimo y el chatbot es capaz de detectar que no ha indicado un tema y se muestra empático con el usuario interesándose de la causa de su estado de ánimo. El usuario indica la causa de su estado de ánimo con una frase hecha y aun así el chatbot sabe identificar que se trata de una causa mala e intenta consolar al usuario. Por último, el usuario le solicita al chatbot soluciones para mejorar su estado de ánimo y el chatbot entra otra vez en un modo pregunta-respuesta relacionado con el tema de mejorar una situación. El chatbot pregunta en primer punto sobre como se puede mejorar el mundo y después como puede mejorar una persona. Como se puede apreciar en esta fase final de la conversación, volvemos a la misma situación de inmovilidad del chatbot, al igual que pasaba en la primera conversación.

A pesar de estos problemas, el chatbot se maneja perfectamente para los objetivos que tiene el proyecto. Estos problemas son pasajeros porque tienen una fácil solución. La solución consiste en entrenar el modelo para La Noche Europea de l@s Investigadores con un conjunto de entrenamiento específico generado por los investigadores. De esta forma con el desarrollo de mi proyecto se ha creado un chatbot que es fácil de adaptar, y será labor de los investigadores adaptar este chatbot a la funcionalidad que ellos deseen.

| Emisor | Texto enviado por el emisor |
|---------|---|
| Usuario | Hola. ¿Cómo te llamas? |
| Chatbot | Me llamo Sara. ¿De qué te gustaría hablar? |
| Usuario | Decide tú porque yo hoy me encuentro fatal |
| Chatbot | ¿Por qué te sientes tan mal? |
| Usuario | He tenido un día de perros |
| Chatbot | Siento mucho escuchar eso |
| Usuario | ¿Me puedes dar alguna solución para mejorar mi día? |
| Chatbot | No estoy seguro de lo que está hablando |
| Usuario | Dime maneras de mejorar un día malo |
| Chatbot | ¿Hay alguna manera de hacer del mundo un lugar mejor? |
| Usuario | Yo pienso que sí |
| Chatbot | ¿Te imaginas ser una persona mejor? |

Cuadro 8.4: Segunda conversación de las pruebas del módulo de generación de respuestas

Capítulo 9

Conclusiones y posibles ampliaciones

El objetivo del proyecto era implementar un chatbot, con la capacidad de adaptarse fácilmente, y que pudiera ser usado en eventos como La Noche Europea de l@s Investigador@s.

Los conocimientos empleados para el desarrollo del proyecto están basados en los conocimientos adquiridos en un gran abanico de asignaturas del Plan de Estudios del Grado en Ingeniería Informática como son *Fundamentos de Bases de Datos, Fundamentos de Ingeniería del Software, Aprendizaje Automático, Procesadores de Lenguajes, Visión por computador, Programación Técnica y Científica y Estructuras de Datos*.

Basándonos en los resultados obtenidos en las pruebas del Apartado 8, podemos asegurar que se han cumplido las tareas de todas las historias de usuario planteadas en el Apartado 3.3, por lo tanto, podemos considerar que se han cumplido todos los objetivos del proyecto. Además, bien los resultados, podríamos asegurar que no solamente se han completado los objetivos, sino que también se ha obtenido un producto de calidad, cumpliendo de sobra las expectativas del proyecto.

Por ejemplo, la deducción de la edad del usuario se ha resuelto de manera sencilla y rápida mediante un modelo pre entrenado, que como se ha visto en las pruebas, da unos buenos resultados en la mayoría de situaciones. Por su puesto no siempre acierta, ya que en ocasiones el aspecto de una persona no está asociado con su edad real, esto forma parte de la complejidad de extraer información del aspecto humano. Aunque teniendo estos fallos, pienso que es una solución práctica, que evita la complejidad que podría suponer un análisis del lenguaje del usuario, para lo cual sería necesario un tiempo y unos conocimientos difícilmente alcanzables en el desarrollo de este proyecto. Y otro de los grandes logros es la obtención de una interfaz totalmente funcional y con ciertos elementos que la hagan un poco más atractiva como el modo oscuro y la interfaz responsive.

Como ya se indicó en el apartado de gestión, se han invertido un número mayor de horas de las que requiere como mínimo un TFG. Durante estas horas se han adquirido una cantidad ingente de conocimientos, ya que casi todas las tecnologías usadas en el proyecto eran desconocidas, exceptuando la plataforma Github, ciertos conocimientos sobre los contenedores, y la herramienta Overleaf, así como una base en el lenguaje *LaTeX*. Pero, por otro lado, he realizado tareas cuyos conocimientos requeridos no habían sido vistos en todo el grado.

Una de las primeras tareas que me hizo ponerme a investigar sobre el ámbito de las páginas web, fue la elaboración de una interfaz web para nuestro sistema. Durante esta investigación se adquirieron muchos conocimientos en el lenguaje HTML, Javascript y CSS. Y la adquisición de conocimientos no se quedó en lo teórico, sino que se puso en práctica enlazando elementos implementados en los distintos lenguajes y obteniendo un producto final con grandes posibilidades.

Otro punto que me empujó a investigar mucho fue la implementación del servidor del Controlador, ya que tuve que utilizar el framework FastAPI, el cual era totalmente desconocido para mí. Estos conocimientos también me ayudaron con el servidor del Modelo porque se basan en el mismo framework.

Una de las partes más importantes del proyecto y la cual ha llevado un gran gasto de tiempo, aunque también ha permitido adquirir muchos conocimientos que no había podido adquirir durante el desarrollo del grado. Esta parte son los modelos empleados para realizar la deducción de edad o la generación de respuestas. Para estas tareas se han usado modelos pre entrenados, para los cuales hacen falta ciertos conocimientos para manejarlo, ya que aunque su nombre lleve a la confusión de que son modelos fáciles de usar y que no van a conllevar un gasto de tiempo; es todo lo contrario porque se deben tener ciertos conocimientos para manejar estos modelos tan complejos y poder sacar todo su potencial que traen por defecto. En este proyecto se han utilizado modelos basados en Transformers, los cuales no había utilizado nunca porque solamente había usado modelos basados en convoluciones para la asignatura de *Visión por computador*. Además, son modelos implementados con PyTorch y hasta el momento únicamente había trabajado con TensorFlow. Todos estos condicionantes hicieron que tuviera que adentrarme en los conocimientos para el uso de estos modelos. Aunque ambos sean modelos basados en Transformers, cada uno funciona de forma distinta, ya que manejan datos distintos, uno texto y el otro imágenes; realizan tareas distintas, uno genera texto y el otro clasifica imágenes; por lo tanto, es necesario analizar en precisión cada modelo por separado. Además de investigar como funciona la plataforma Huggingface, que es la que contiene los modelos pre entrenados.

En definitiva, este proyecto ha sido todo un reto para mí, ya que se han puesto a prueba todos mis conocimientos adquiridos durante el grado y capacidades para adaptarme, puesto que he desarrollado un sistema desde la

parte más visible como es el frontend y la parte más oculta y lógica del sistema como es el backend. En este proyecto he podido invertir muchas horas en herramientas totalmente nuevas para mí y he podido adquirir un nivel de conocimientos que antes de comenzar el proyecto no tenía ni previstos. Finalmente, he podido conllevar toda la dificultad del proyecto y he conseguido completarlo con grandes resultados.

Por último, la reflexión final es que el sistema implementado es un sistema que cumple con todos sus objetivos, pero que sigue teniendo mucho potencial por explotar si se invierte un poco más de tiempo en él.

9.1. Posibles ampliaciones

Como se ha ido indicando en esta parte final de la memoria, el proyecto tiene claras vías por donde ampliar y mejorar el proyecto. Algunas de las posibles ampliaciones son las siguientes:

- **Búsqueda y utilización de una plataforma de despliegue con GPU:** Si utilizamos una plataforma que tenga las características y las funcionalidades que tiene Heroku, junto con la posibilidad de utilizar una GPU, podríamos realizar una reestructuración del sistema la cual simplificaría esta estructura. La simplificación conllevaría la unión de los servidores del Controlador y del Modelo, y una refactorización completa del código. Esta unión evita el trabajo extra derivado de implementar dos servidores y de tener que mantenerlos en funcionamiento y actualizados por separado, teniendo partes en común. Además, la unión elimina muchos envíos de peticiones entre servidores, ya que el intercambio de información entre el Controlador y el Modelo se efectuarían de forma interna, lo que conlleva mayor rapidez y seguridad.
- **Mejora de la interfaz web:** Basándonos en el feedback de los usuarios que ejecutaron las pruebas de la interfaz web, se podría modernizar la interfaz web empleando herramientas más actuales para implementar la funcionalidad de la interfaz, y que esta tome un aspecto más renovado.
- **Reutilización de los resultados:** Un chatbot nunca deja de mejorarse, ya que normalmente se reutilizan los resultados de conversaciones realizadas con el chatbot para mejorar al mismo. Nuestro sistema ya es capaz de guardar las conversaciones realizadas con el chatbot en un log, pero nos falta un programa que sea capaz de mover esa información y entrenar al chatbot con ella. Además, sería muy interesante que se puede automatizar este proceso, ya que evitaría mucha carga de trabajo para el administrador del sistema.

- **Mejorar el modelo de deducción de edad:** Aunque como se ha visto en las pruebas, el modelo de deducción tiene una calidad más que aceptable para nuestro sistema, siempre se puede mejorar porque se trata de un modelo pre entrenado. Por lo tanto, podríamos entrenar al modelo con muchas más imágenes que podemos obtener de las múltiples bases de imágenes existentes en internet. Claramente, esto conlleva una carga de trabajo grande, ya que se deben obtener las imágenes, etiquetarlas si no lo están, y por último crear un programa para el entrenamiento del modelo pre entrenado.
- **Creación de una interfaz propia para el chatbot:** La creación de una interfaz propia podría ayudar a seguir avanzando en la simplificación del sistema, por el hecho de que ahora mismo se deben generar secciones distintas para cada rango de edad que distingue el sistema. Estas duplicidades vienen derivadas del uso de la interfaz de Dialogflow. Por lo tanto, una gran mejora sería crear nuestra propia interfaz web para eliminar todas estas duplicidades y poder simplificar el sistema. Además, tendríamos un mayor control del sistema. Incluso se podría dejar de hacer uso de Dialogflow, aunque esto conllevaría un mayor trabajo dado que eliminamos las facilidades que nos da Dialogflow a la hora de desplegar el chatbot.
- **Uso de un avatar:** Actualmente se está poniendo de moda producir chatbots con una interfaz visual compuesta por un avatar. Este avatar tiene un aspecto humano para hacer más cómodo el establecimiento de las conversaciones. Con este avatar eliminamos la idea antigua de tener una interfaz textual para poder interactuar con el chatbot.

Apéndice A

Características de las herramientas analizadas

Las características analizadas en cada una de las herramientas son las siguientes:

- Información general
- Uso. Explicación sobre el funcionamiento de la herramienta
- Extensibilidad. Capacidad de conexión con otras herramientas
- Integración. Capacidad para desplegar el chatbot
- Ejemplos

A.1. AIML

Información general

El lenguaje de programación AIML (Artificial Intelligence Markup Language) fue desarrollado por el doctor Richard Wallace y la comunidad de código abierto Alicebot en la segunda mitad de los años 90. AIML es un lenguaje basado en etiquetas, al igual que los lenguajes HTML y XML. En concreto, el lenguaje AIML se basó en gran medida en el lenguaje XML. Este parecido entre lenguajes no es una casualidad, si nos fijamos en el entorno de la época en la que se desarrolló AIML, a finales de la década de 1990 se produjo la explosión de la World Wide Web. Esta explosión trajo consigo al lenguaje HTML, que surgió como un lenguaje simple que sirviese como estándar para la creación de páginas web. El objetivo de HTML era que cualquier persona con pocos conocimientos informáticos fuese capaz de crear una página web. Esta filosofía de estándar de una tecnología y de simplicidad la tomaron como ejemplo los creadores de AIML durante su desarrollo,

intentando que AIML se convirtiese en un estándar en la creación de chatbots y que además fuese accesible al mayor número de usuarios. Además del lenguaje HTML, durante la década de 1990 se creó el lenguaje XML, que también se convirtió en un estándar.

Actualmente, al igual que le ha pasado al lenguaje HTML, el lenguaje AIML se ha ido actualizando con el tiempo, añadiéndole nuevas funcionalidades que permitiesen producir chatbots más complejos, acorde al incremento de requisitos en los chatbots que ha ido surgiendo con el paso de los años. En el momento de la realización de este TFG, la última versión publicada de AIML es la versión 2.1 . Al producir agentes conversacionales con AIML, estos agentes no se convierten en cajas negras, por lo tanto, son más transparentes al programador que los generados con otras plataformas más grandes, ya que con AIML se crean chatbots basados en reglas. AIML se puede escribir en casi cualquier lenguaje natural.

Uso

Para generar un chatbot con AIML es necesario generar una serie de archivos, que contendrán el estado y la configuración del bot.

En el estado del chatbot podemos distinguir el estado del bot y el estado del cliente. El estado del bot se define usando valores globales para las propiedades del bot, cada bot tendrá sus propiedades. El estado del cliente se define empleando variables locales, cada cliente tendrá su estado específico.

La configuración del bot se define en los archivos AIML, en los archivos Learnf, en los Sets y en los Maps. Dentro de los archivos AIML se define la lógica del chatbot a base de añadir reglas, además en estos archivos se pueden realizar conexiones con otros chatbots definidos por otra serie de archivos. Dentro de los archivos Learnf se guardan las categorías aprendidas por el chatbot cuando en un template AIML se activa una etiqueta. Las categorías aprendidas son globales a todos los clientes del chatbot. Dentro de los archivos Set se define un conjunto de cadenas. Dentro de los archivos Map se define un mapeo de cadena a cadena.

Según el estándar de AIML, esta serie de archivos se pueden definir dónde y cómo quiera el creador del chatbot.

Dado que AIML es solamente un lenguaje de programación, es necesario un framework para la creación del agente conversacional, como pueden ser algunos intérpretes y bibliotecas de código abierto (Python, Node JS, Java) o servicios web (Pandorabots).

Si se elige la opción de usar intérpretes y bibliotecas de código abierto, disponemos de las siguientes posibilidades:

- Python → Program-Y ¹

¹<https://github.com/keiffster/program-y>

- Node JS → aimlinterpreter ²
- Java → Program AB ³

Si se elige la opción de usar servicios web como Pandorabots, uno de los más populares actualmente, se podrá alojar el chatbot en la plataforma y se podrá hacer uso de todas sus funcionalidades. Si nos centramos en Pandorabots disponemos de un editor de AIML, un editor de interfaces, control de versiones a través de Github, chatlogs, conversor de texto a voz y viceversa, posibilidad de añadir un personaje 3D como interacción con el chatbot, integración con RESTful APIs y muchos más, según se indica en su página oficial (5).

Muchas de estas funcionalidades depende del tipo de cuenta que se disponga en Pandorabots.

²<https://www.npmjs.com/package/aimlinterpreter>

³<https://code.google.com/archive/p/program-ab>

| | SANDBOX | DEVELOPER | PRO | ENTERPRISE |
|---|------------------------|---------------------------------|----------------------------------|----------------------------|
| Development Sandbox | ✓ | ✓ | ✓ | ✓ |
| Unlimited Sandbox Messages | ✓ | ✓ | ✓ | ✓ |
| API Access | | ✓ | ✓ | ✓ + more |
| Chat Widget and Landing Page | | ✓ | ✓ | ✓ |
| 3rd Party Channels <small>•</small> | | \$9/Channel | ✓ | ✓ + more |
| Channel Messages | | 10.000/month | 100.000/month | Varies |
| Additional Channel Messages | | \$3/1,000 Messages | \$2.50/1,000 Messages | Varies |
| Live Training / Bot Training <small>•</small> | | | ✓ | ✓ + more |
| Rapport Avatar Compatibility | | ✓ | ✓ | ✓ |
| Support | Public Office Hours | Email | Email, Chat, and Phone | SLA and Premium Support |
| Chatbot Libraries | Open Source | Open Source | Open Source | Custom Development |
| | Free | \$19/month 2 week free trial | \$199/month 2 week free trial | |
| | Select | Start Trial | Start Trial | Contact Us |

Fuente: <https://developer.pandorabots.com/home.html>

Figura A.1: Cuentas disponibles en Pandorabots

En la Figura A.1 podemos comprobar como con la cuenta gratuita no disponemos de todas las funcionalidades anteriormente descritas.

Extensibilidad

Dado que hay muchos frameworks para la creación de un chatbot con AIML, dependerá del framework elegido la cantidad de posibilidades de conexiones con APIs o servicios externos al chatbot.

Para que AIML sea flexible y extensible dispone de la posibilidad de integrar sus propias APIs, bases de datos o también se pueden crear etiquetas propias; pero para ello se debe tener cierto nivel con el lenguaje de programación AIML.

Si optamos por servicios web como Pandorabots para poder disponer de conexión con servicios externos al servicio web o con APIs se debe disponer

de una cuenta de pago, no basta con la versión gratuita de Pandorabots.

Integración

Si optamos por servicios web como Pandorabots, al igual que pasa con la extensibilidad, será necesaria una cuenta de pago para poder integrar nuestro chatbot en plataformas de mensajería como Facebook Messenger, Twitter, Telegram y muchos más.

Pero si optamos por frameworks de código abierto, hay una multitud de posibilidades, tantas como el nivel de conocimientos que tenga el creador del chatbot. Un ejemplo puede ser con Node JS, donde podríamos desplegar el chatbot en cualquier página web a través de una interfaz web que enviase peticiones al servidor creado con Node JS que aloja al chatbot.

Ejemplos

Algunos ejemplos de chatbot implementados con AIML son los siguientes:

- **Rosie.** Chatbot base basado en ALICE 2.0 ⁴
- **Cathy.** Chatbot para Discord basado en Python 3 y AIML ⁵

A.2. IBM Watson

Información general

IBM Watson es un portfolio de IBM de aplicaciones, herramientas y soluciones preparadas para la empresa, diseñado para reducir los costes y los obstáculos de inteligencia artificial, además de optimizar los resultados y el uso responsable de la inteligencia artificial según se indica en su página oficial (6). Aunque en un inicio IBM Watson no era lo que es actualmente y tampoco se buscaba llegar a este punto. Originalmente, IBM Watson surgió como el siguiente gran desafío de la empresa IBM, tras haber salido victorioso de otros grandes desafíos como la victoria de Deep Blue contra Garry Kasparov y el desafío de la supercomputadora Blue Gene. El objetivo que perseguía en sus orígenes Watson era conseguir ganar a competidores humanos en el concurso Jeopardy. Este concurso de televisión estadounidense consistía en un concurso de preguntas sobre una multitud de temas, y los concursantes deberán resolver las preguntas de cada prueba realizando preguntas sobre las pistas que va dando el presentador. Las reglas del concurso obligan a que Watson no únicamente sepa conocimientos sobre los temas que se utilizan en el concurso, sino también a saber efectuar preguntas al presentador basándose en sus pistas, y además saber distinguir cuando la pista del presentador

⁴<https://github.com/pandorabots/rosie>

⁵<https://github.com/DevDungeon/Cathy>

no es cierta. Todos estos requisitos convierten el ganar este concurso en un gran desafío, la empresa IBM se embarcó en este proyecto a mediados de la década del 2000. Finalmente, en el año 2011 se llegó a un acuerdo para la realización del programa, donde compitieron dos grandes ex-concursantes del programa como son Ken Jennings y Brad Rutter contra Watson. Tal y como se indica en el artículo (7), Watson ganó el juego con \$77.147, dejando a Rutter y Jennings con \$21.600 y \$24.000 respectivamente.

Tras la victoria de IBM Watson, se empezó a enfocar y rediseñar a Watson hacia distintos sectores como la medicina, la banca o los agentes conversacionales, entre otras. Los agentes conversacionales es el sector que nos interesa investigar.

Dado que IBM Watson está orientado al mundo empresarial, destaca por su confianza, es decir, Watson es transparente en cuanto a las decisiones basadas en inteligencia artificial; también protege la privacidad de los datos y su seguridad. Todas estas características son muy valoradas en cualquier producto empresarial, incluso algunas son imprescindibles.

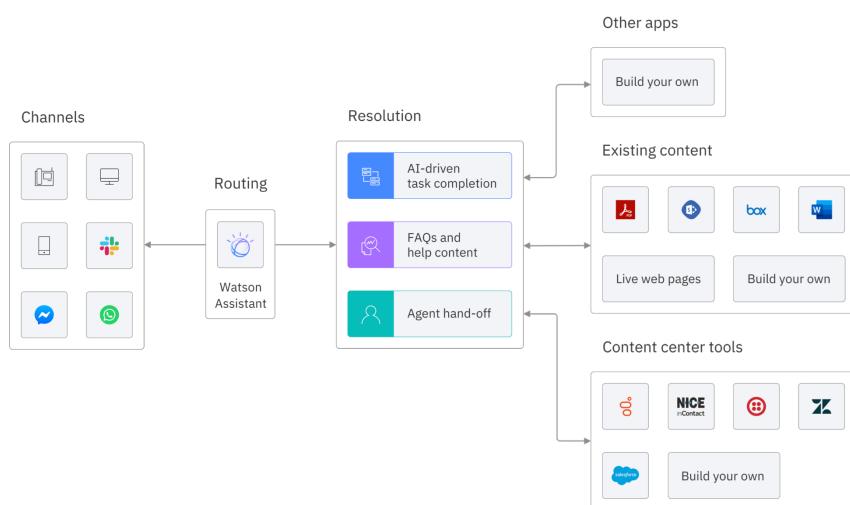
Otra característica de Watson es su procesamiento del lenguaje natural. Al igual que AIML es multilenguaje, pero Watson lo lleva a un aspecto más complejo, añadiendo la capacidad de analizar datos complejos y no estructurados, como pueden ser códigos de programación o incluso formas de expresión específicas de una modalidad de trabajo. Este procesamiento del lenguaje natural no es algo alcanzable por cualquier empresa. La inteligencia artificial de Watson sabe desenvolverse en las situaciones más complejas, como pueden ser situaciones en las que no pueda responder, reenviando la solicitud a un agente humano o hacia algún documento de ayuda; evitando realizar preguntas redundantes facilitando la comunicación y haciéndola más natural; y por último sabe manejar solicitudes ambiguas que son comunes en la comunicación como pueden ser errores ortográficos, cambios de tema, y muchas más situaciones que pueden ser inesperadas para el chatbot en una conversación. Además, IBM Watson intenta mejorar su rendimiento aportando al creador del chatbot información sobre que nuevos temas añadir para mejorar la respuesta a las solicitudes.

Y por último, destaca que Watson trabaja con cualquier servicio en la nube, lo que facilita su integración en las empresas, ya que no será un impedimento el lugar donde residan los datos de la empresa. Y además el trabajar en cloud facilitará el trabajo con los datos.

En concreto, en la página oficial de IBM Watson Assistant (8) se destaca la rentabilidad del producto, de hasta un 337% según el Informe TEI de Forrester (9); su precisión, de hasta un 14,7% superior que las soluciones de la competencia según un reciente estudio publicado sobre machine learning (10); y su fiabilidad, ya que Watson tiene más de 1000 despliegues de clientes en todos los sectores.

Uso

IBM Watson está pensado para ser utilizado por cualquier persona, da igual que no tenga conocimientos en programación. La creación del chatbot se basa en un editor de arrastrar y soltar. Este editor evita la complejidad de la programación de un chatbot y los posibles errores derivados de esa necesidad de escribir el código en su creación o en alguna modificación que necesite el chatbot. Es posible crear un chatbot sin escribir una sola línea de código.



Fuente: <https://www.ibm.com/es-es/products/watson-assistant>

Figura A.2: Editor de IBM Watson

La IA del chatbot es capaz de comprender un tema en cualquier lenguaje natural con unas pocas frases, facilitando la adaptación del chatbot a la funcionalidad que se le quiera dar, adaptándose de forma rápida y precisa al dominio del proyecto.

Extensibilidad

Como se ha indicado en apartados anteriores, IBM Watson es un portfolio de IBM de aplicaciones. En su página oficial (11) se indican las siguientes aplicaciones con las que se puede integrar el chatbot:

- IBM Watson Discovery
- IBM Watson Natural Language Understanding
- IBM Watson Speech to Text

- IBM Watson Text to Speech
- IBM Watson Knowledge Studio
- IBM Watson Language Translator
- IBM Watson Natural Language Classifier

Entre estas aplicaciones se encuentran algunas muy útiles como IBM Watson Discovery para la extracción y búsqueda de información, IBM Watson Speech to Text para transformar la voz a texto escrito gracias a una potente tecnología de machine learning, IBM Watson Language Translator para traducir dinámicamente información, o IBM Watson Knowledge Studio para enseñar a Watson el idioma del dominio del chatbot.

Integración

El chatbot se puede integrar como un chat web, como contestador de llamadas de teléfono o como un chatbot en una aplicación de mensajería como WhatsApp, Facebook Messenger, SMS y muchos más. Todas estas maneras de integrar el chatbot son muy fáciles de realizar.

Ejemplos

Algunos ejemplos de chatbot implementados con Watson son los siguientes:

- **Nanci.** Chatbot desarrollado por la empresa GM Financial ⁶
- **Chatbot desarrollado por la empresa Bradesco** ⁷
- **CARL.** Chatbot desarrollado por la empresa Siemens ⁸
- **Chatbot desarrollado por la empresa Humana** ⁹

A.3. Google Dialogflow

Información general

Dialogflow pertenece a la empresa Google, aunque no fue desarrollada originalmente por ella. Google adquirió API.AI en el año 2016 y le cambió el nombre a Dialogflow. Desde su adquisición, Google ha mejorado Dialogflow

⁶<https://www.gmfinancial.com/en-us/company/newsroom/chatbot.html>

⁷<https://www.ibm.com/watson/stories/bradesco>

⁸<https://www.ibm.com/es-es/products/watson-assistant/client-stories>

⁹<https://www.ibm.com/watson/stories/humana>

gracias a sus técnicas de inteligencia artificial de gran calidad. Esta plataforma es una de las más utilizadas para la creación de chatbots junto con IBM Watson, aunque a diferencia de Watson, Dialogflow está enfocado tanto para el mundo empresarial como para usuarios particulares que están empezando en el mundo de los chatbots. Esta variedad en la complejidad del chatbot es posible gracias a una interfaz que permite crear un chatbot con unos mínimos conocimientos técnicos, únicamente será necesario introducir las frases de las preguntas y de las respuestas para construir el chatbot. Y si se quiere generar un chatbot más complejo, podemos introducirnos en la configuración del chatbot o integrar más funcionalidades que aumenten su complejidad.

Dentro de Dialogflow existen dos tipos de agentes, los Agentes de ES y los agentes de CX. Dentro de los agentes de ES existen dos ediciones, la edición de prueba y la edición Essentials. Con la versión de prueba no se puede acceder a los servicios de Google Cloud, pero las entradas y salidas son gratuitas; mientras que con la versión Essentials si se puede acceder a esos servicios, pero las entradas y salidas del agente empezarán a costar dinero. Entre la versión Essentials y la versión CX la principal diferencia es que en la versión CX la relación entre los intents se define mediante un diagrama de flujo, mientras que en la versión Essential la relación entre los intents se define mediante los contextos. La limitación de los contextos es que no posibilitan la relación de varios intents con otro intent. La versión CX ha sido lanzada hace unos pocos meses, por lo que es algo novedosa y es útil para aquellos que quieran realizar un chatbot algo más complejo o que ya tengan experiencia con la plataforma. La versión Essentials está más enfocada a agentes de complejidad media y la versión CX está más enfocada a agentes de complejidad alta. Aunque la curva de aprendizaje es más suave en la edición CX, la edición CX no dispone actualmente de todas las funcionalidades presentes en la edición Essentials.

Uso

Para empezar a crear un chatbot con Dialogflow será necesaria una cuenta Google. Los apartados claves para la creación de un bot son Intents y Entities.

Dentro del apartado Intents se generan los distintos intents a usar en el chatbot, cada intent es un estado al que accede el chatbot si se cumple el contexto y se detecta una de las frases de entrenamiento del intent. Los contextos sirven para guardar información adquirida que puede resultar útil durante la conversación, como el nombre del usuario que está utilizando el chatbot; además de la información que se quiere conservar, también se puede indicar durante cuento tiempo se estima que es útil esa información, también llamado lifespan. Dentro de cada intent se debe también definir las posibles respuestas que envía el chatbot si se activa el intent. Y por último a cada

intent se le pueden añadir eventos. Por defecto, en el apartado Intents vienen creados los siguientes intents:

- Default Fallback Intent: Intent para cuándo el chatbot no reconoce la pregunta
- Default Welcome Intent: Intent para dar la bienvenida al chatbot

Dentro del apartado Entities se generan las distintas entidades. Una entidad es un conjunto de ejemplos sobre un tema, por ejemplo, la entidad países contiene los siguientes elementos: España, Francia, Italia, etc. Cuando se define una entidad se deben indicar los elementos que la forman, pero adicionalmente se puede indicar los sinónimos de los elementos, este añadido le dará una mayor naturalidad a la conversación al hacer que los textos no sean tan estáticos.

Otro apartado importante para el desarrollo del chatbot es el apartado Training, donde se podrán analizar las conversaciones que se han realizado con el chatbot e indicar si se ha respondido adecuadamente o no, esta acción de aceptación o no hará que el chatbot sea más preciso con sus respuestas. Si se acepta una respuesta, si la pregunta hecha por el usuario no se encuentra entre las preguntas del intent activado, se añade a la lista de preguntas del intent.

Extensibilidad

Dialogflow permite integrar un webhook. Este webhook puede estar alojado tanto en un servidor externo como en Google Cloud. Si se opta por la opción de Google Cloud, para lo que es necesario una cuenta Google de pago, se pueden añadir los archivos que componen el webhook en el apartado Fulfillment dentro de un editor en línea. Si se opta por la opción de un servidor externo, tanto si es local como si es en una plataforma en la nube, se deberá añadir la URL del servidor web en el mismo apartado que se indicó anteriormente. La conexión con el servidor externo se realizará mediante el envío de peticiones tipo POST. Estas peticiones contendrán la información en formato JSON. Para que la información se envíe al servidor, independientemente de la opción que se elija, se debe activar en los intents elegidos la opción del webhook, de esta forma cuando se active el intent se enviará una petición al servidor web.

Una vez se tenga una conexión con un servidor web, dentro de este servidor se podrán añadir más funcionalidad, como BBDD o interpretes de AIML, que permitirán seguir extendiendo el chatbot.

Integración

El chatbot se puede integrar en canales de telefonía, como Twilio; en canales de mensajería, como Telegram, Facebook Messenger o Twitter; en

canales de videollamadas, como Skype; y en páginas web a modo de chat web.

Ejemplos

Algunos ejemplos de chatbot implementados con Dialogflow son los siguientes:

- **SAM. Chatbot desarrollado por la empresa Ubisoft** ¹⁰
- **Chatbot desarrollado por la empresa Dominos Pizza** ¹¹
- **Chatbot desarrollado por la empresa Malaysia Airlines** ¹²

A.4. Rasa Stack

Información general

Rasa Stack es un conjunto de herramientas de aprendizaje automático de código abierto para el desarrollo de chatbots. Rasa Stack está desarrollado por una comunidad de personas pertenecientes a muchos lugares del mundo, de modo que esta plataforma, a diferencia de las dos anteriores, no está respaldada por ninguna de las grandes empresas tecnológicas como pueden ser Google o IBM. Pero esto no quiere decir que no sea una buena plataforma, ya que ha sido probada por grandes empresas como AIRBUS, TOYOTA o Adobe.

En su página oficial (13) se destaca la gran personalización que pueden alcanzar los chatbots en esta plataforma.

Dentro de Rasa Stack se pueden distinguir dos tipos de cuenta, la cuenta para empresas (Rasa Enterprise) y la cuenta gratuita (Rasa Open Source o Rasa X). La diferencia entre Rasa Open Source y Rasa X es que Rasa X no es de código abierto, mientras que Rasa Open Source, como indica su nombre, si es de código abierto. La cuenta para empresas proporciona funcionalidades muy útiles para productos empresariales como pueden ser realizar análisis del chatbot, acceso con roles a la plataforma, disponibilidad de un soporte de calidad, y mucho más.

En Rasa Stack se destaca que los chatbots creados en esta plataforma no se convierten en , sino que el funcionamiento del chatbot es transparente, por lo tanto, se tiene total acceso a la configuración de todo el chatbot, incluido el modelo usado en el entrenamiento del bot.

¹⁰<https://www.hd-tecnologia.com/ubisoft-ia-personal-sam-primer-asistente-gaming>

¹¹Caso de estudio de Dominos Pizza (12)

¹²<https://cloud.google.com/customers/malaysia-airlines>

Uso

Al igual que pasa en Dialogflow, Rasa Stack mantiene un contexto de la conversación, donde se guarda información proporcionada por el usuario, lo que permite una conversación más natural al no hacer preguntas redundantes.

La creación del chatbot se divide en dos módulos, Rasa NLU y Rasa Core. La plataforma proporciona unas técnicas avanzadas de comprensión del lenguaje natural, estas técnicas avanzadas de comprensión del lenguaje natural se pueden entrenar sobre la base de una lista de mensajes, a cada mensaje se le asignará una intención y las entidades que contiene. Una vez está entrenado el módulo técnicas avanzadas de comprensión del lenguaje natural, se procede a configurar el módulo Rasa Core, que es el encargado de confeccionar la respuesta del chatbot a la pregunta identificada por el módulo Rasa NLU. La configuración de Rasa Core consiste en escribir las respuestas de los distintos intents, definir el dominio del chatbot, definir la conexiones con APIs, y definir el modelo de gestión del diálogo, por ejemplo, una CNN, citando además las políticas que se van a usar.

Extensibilidad

La transparencia que tienen los chatbots permiten conectar con él muchas APIs externas que suman más funcionalidades aparte de las que ya proporciona la plataforma. Y al igual que pasa con Dialogflow, se puede conectar a un servidor web y a este servidor conectar las APIs.

Integración

Tal y como se indica en su página oficial (13), un chatbot creado con Rasa Stack se puede integrar en canales como IVR, chat y SMS. Además, se indica que Rasa Stack admite 10 canales de mensajería integrados, como pueden ser Telegram, Facebook Messenger, Slack y algunos más; y que adicionalmente proporciona un punto de conexión con cualquier plataforma de comunicación, como puede ser un servidor web.

Ejemplos

Algunos ejemplos de chatbot implementados con Rasa Stack son los siguientes:

- **Djingoo. Chatbot desarrollado por la empresa Orange**¹³
- **Chatbot desarrollado por la empresa TMobile**¹⁴

¹³<https://rasa.com/customers/orange>

¹⁴<https://rasa.com/customers/t-mobile>

- Chatbot desarrollado por la empresa Helvetia¹⁵

A.5. GPT-3

Información general

GPT-3 (Generative Pre-trained Transformer 3) es un conjunto de modelos de inteligencia artificial desarrollados por la empresa OpenAI en el año 2020. Este modelo es la tercera generación de estos modelos de inteligencia artificial. En concreto, este último modelo tiene 175 billones de parámetros, lo cual es una potencia impresionante que le permite generar texto que parece escrito por humanos y no por máquinas, es incluso capaz de escribir código. GPT-3 se ha visto potenciado por la inversión de Microsoft entre otros. Este modelo ha revolucionado el procesamiento y la generación de lenguaje natural. Esta última generación es de pago, cobrando cada cierta cantidad de tokens analizados, en concreto cada 1000 tokens actualmente, según se indica en su página de precios (14). Un token es un conjunto de palabras o un conjunto de letras, su tamaño depende también del lenguaje que se esté usando. Si se quiere saber más sobre los tokens se puede emplear la herramienta Tokenizer¹⁶, que nos proporciona OpenAI, para calcular el número de tokens en una frase. Algo importante es que únicamente se paga por lo que se emplea, no como en otras páginas que independientemente de lo que se emplee un servicio se paga una cuota fija.

Como se ha indicado anteriormente, GPT-3 es un conjunto de modelos. Según se indica en su página web (15) existen los siguientes modelos:

- Davinci
- Curie
- Babbage
- Ada

La diferencia entre los distintos modelos son su potencia, velocidad y coste por token. El modelo Davinci es el más potente y el más caro, mientras que el modelo Ada es el más rápido y barato.

Las posibles aplicaciones de GPT-3, según su página de ejemplos (16), son las siguientes:

- Generación de lenguaje natural
- Clasificación

¹⁵<https://rasa.com/customers/helvetia>

¹⁶<https://beta.openai.com/tokenizer>

- Generación de código
- Chatbot
- Traducción (tanto código como lenguaje natural)

La aplicación de GPT-3 como un chatbot se explicará más en detalle en el siguiente apartado.

Un problema de GPT-3 es que se trata de una herramienta muy cara, ya que necesita una enorme cantidad de potencia informática para funcionar, por lo tanto, su uso se limita solamente a empresas.

Uso

Dado que GPT-3 es un modelo de aprendizaje, no es una plataforma como las vistas hasta el momento, para poder crear un chatbot será necesario hacer uso de un lenguaje de programación como Python para elaborar el chatbot. En la sección de bibliotecas de OpenAI¹⁷ se pueden encontrar todas las bibliotecas, tanto oficiales como de la comunidad, para poder emplear GPT-3 con distintos lenguajes de programación.

El chatbot consistirá en sucesivas llamadas a la API de OpenAI para obtener las respuestas del chatbot. A continuación se puede observar el código en Python necesario para realizar una llamada a la API:

```

1 import openai
2
3 openai.Completion.create(
4     engine="davinci",
5     prompt="Make a list of astronomical observatories:"
6 )

```

Fuente: <https://openai.com/api>

También se puede hacer empleo del Playground¹⁸ de OpenAI para utilizar las distintas funcionalidades de GPT-3.

GPT-3 ha sido entrenado con millones de datos, pero dispone de una herramienta para enfocar su modelo a los datos que vayamos a usar en nuestro chatbot. Esta herramienta es el ajuste, que consiste en seguir entrenando el modelo con un conjunto de datos durante un cierto número de épocas, a elección del creador del chatbot, permitiendo generar textos relacionados con el dominio del chatbot.

¹⁷<https://beta.openai.com/docs/libraries/libraries>

¹⁸<https://beta.openai.com/playground>

Los datos del conjunto de entrenamiento deberán ser formateados de cierta forma para poder entrenar con ellos. Para formatear los datos, OpenAI dispone de una herramienta que lo posibilita. Al igual que para inferir textos con GPT-3 es necesario pagar cada cierta cantidad de tokens, pasa lo mismo con el ajuste de los modelos. En la documentación (17) se indica el precio de entrenar los distintos modelos.

Una vez se tenga el modelo listo, algo que hay que tener en cuenta durante la creación del chatbot es el mantenimiento de un contexto durante las conversaciones, ya que con GPT-3 cada vez que se llama a la API no se dispone de un contexto, cosa que por ejemplo si se tenía en Dialogflow, únicamente se dispone de la información que se le pase con la llamada.

Una solución a este problema podría ser la creación de una base de datos, por ejemplo con PostgreSQL que dispone de un tipo muy útil para los textos. Con esta base de datos se irá guardando información de las anteriores preguntas, la cual se le pasará al modelo junto con la pregunta a realizar.

Extensibilidad

En la página oficial no se hace referencia a ninguna funcionalidad externa, pero el hecho de que se puedan hacer llamadas a la API con distintos lenguajes de programación, permite que todas las funcionalidades de las que dispongan estos lenguajes se puedan usar junto con GPT-3.

Integración

La integración no es tan sencilla como podría ser en plataformas como Dialogflow, dado que en la creación del chatbot con GPT-3 solamente se implementa el back-end del chatbot, por lo tanto, se deberá implementar, por parte del creador del chatbot, el front-end del mismo. Para implementar el front-end se dispone de muchas posibilidades, como podría ser implementar un chat web o una APP, entre otras.

Ejemplos

Algunos ejemplos de chatbot implementados con GPT-3 son los siguientes:

- **Chatbot para Telegram**¹⁹
- **Chatbot para WhatsApp**²⁰
- **Marcus. Chatbot para guía de viajes**²¹

¹⁹<https://github.com/xwarfare/GPT3-Telegram-Chatbot>

²⁰<https://github.com/theshanergy/whatbot>

²¹<https://github.com/manan-paneri-99/marcus-gpt3-bot>

A.6. BlenderBot

Información general

BlenderBot es un proyecto de código abierto desarrollado por la empresa Facebook. En este trabajo vamos a analizar la versión 1 de este chatbot, ya que recientemente se ha lanzado su segunda versión. Se trata de uno de los chatbots de código abierto con mayor dominio. Y como indica Facebook (18), además, durante una conversación este chatbot es capaz de generar una sensación de humanidad que pocos chatbots pueden igualar. Una característica de este chatbot, el cual fue el primero en poseerla, que está relacionada con su nombre, es su mezcla de habilidades conversacionales, como pueden ser la empatía, el conocimiento y la personalidad en un solo sistema. Este chatbot tiene distintas configuraciones, las cuales se diferencian por su número de parámetros o dicho de otra manera por su potencia.

BlenderBot mejoró la potencia de los chatbots anteriores a él, sin perder la eficiencia necesaria para entablar una conversación de forma fluida. Esta mejora fue posible gracias a las distintas técnicas usadas por Facebook (18), como son el paralelismo de modelos por columna o la división de la red neuronal en partes más pequeñas y manejables.

Otro punto importante en los chatbots es el conjunto de entrenamiento empleado, ya que el chatbot imitará todos los comportamientos de las frases de los conjuntos, tanto buenas como malas, heredando todos los sesgos del conjunto. BlenderBot introdujo una novedosa tarea llamada Blended Skill Talk (BST). Como se indica en el artículo de Facebook (18), estas tareas consiste en las siguientes habilidades asociadas con su respectivo conjunto de entrenamiento:

- Uso atractivo de la personalidad (PersonaChat)
- Uso atractivo del conocimiento (Wizard of Wikipedia)
- Muestra de empatía (Empathetic Dialogues)
- Capacidad para combinar los tres a la perfección (BST)

La combinación de estas habilidades es un desafío difícil porque los sistemas deben poder cambiar entre diferentes tareas cuando sea apropiado, como ajustar el tono si una persona cambia de broma a seria. Nuestro nuevo conjunto de datos BST proporciona una manera de crear sistemas que combinen y muestren estos comportamientos.

Uso

De igual modo que pasa con GPT-3, esta herramienta para generar un chatbot necesitará de un lenguaje de programación, dado que la herramienta

únicamente consiste en un modelo de aprendizaje. El chatbot consistirá en sucesivas inferencias del modelo para obtener las respuestas del chatbot.

Extensibilidad

En este apartado pasa lo mismo que en el anterior apartado. El chatbot dispone de todas las funcionalidades de las que disponen los lenguajes usados para generar el chatbot.

Integración

En la creación del chatbot con BlenderBot solo se implementa el back-end del chatbot, por lo tanto, se deberá implementar, por parte del creador del chatbot, el front-end del mismo. Para implementar el front-end se dispone de muchas posibilidades, como podría ser implementar un chat web o una APP, entre otras.

Apéndice B

Detalles de la implementación

B.1. Definición de las interfaces responsive

Una interfaz responsive es un tipo de interfaz que se adapta al dispositivo donde se esté visualizando el sitio web. Para conseguir esta adaptación se hace uso de una cierta estructura del lenguaje CSS. Esta estructura son los @media¹. Dentro de esta estructura se puede indicar que restricciones queremos que se cumplan sobre la pantalla, como pueden ser el máximo y el mínimo, tanto de ancho como de alto de la pantalla. Con estas restricciones se van generando los estilos propios de cada dispositivo.

Las restricciones que se han utilizado son las mostradas en la Tabla B.1.

| | Altura | | Anchura | |
|--------------------------|--------|-------|---------|--------|
| | Max. | Min. | Max. | Min. |
| Portátil | 1080px | 751px | 1920px | 1025px |
| Tablet Horizontal | 750px | 500px | 1024px | 751px |
| Tablet Vertical | 1024px | 751px | 750px | 500px |
| Móvil | 850px | 500px | 400px | 200px |

Cuadro B.1: Restricciones de tamaño de pantalla para los dispositivos

Para obtener estas medidas de pantalla se han analizado varios dispositivos de cada uno de los tipos de dispositivo utilizados en la Tabla B.1.

Un caso de uso de distintos estilos según el dispositivo, es la visualización de la descripción de la página principal y del mockup. Cuando el dispositivo tiene la suficiente anchura, se muestran ambos elementos en paralelo, como se puede ver en la Figura B.1 (Portátil, Tablet Horizontal); mientras que cuando la anchura no lo permite, se muestran uno detrás del otro como se puede ver en la Figura B.2 (Tablet Vertical, Móvil).

¹https://www.w3schools.com/cssref/css3_pr_mediaquery.asp



Figura B.1: Interfaz responsive usando el portátil

Esta filosofía de creación de sitios web permite crear varios sitios web definiendo una única vez los elementos que forman el sitio web y definiendo varios estilos que modifiquen los atributos de esos elementos.

B.2. Funcionamiento del interruptor de cambio de tema

Suponiendo que hemos definido ambos temas, para generar el interruptor será necesario crear un elemento nuevo en la barra de navegación de las distintas secciones del sitio web. El interruptor será un elemento compuesto por dos espacios, donde en cada espacio habrá un ícono de un sol y una luna respectivamente; y una bola que se encuentra en el interior de estos dos espacios y que se desplaza del extremo izquierdo al extremo derecho y viceversa.

Una vez definidos los componentes de este interruptor es hora de explicar como se ha implementado el movimiento del interruptor para cambiar de tema. Este movimiento se simula a través del cambio del estilo del interruptor. En un principio el interruptor tiene el estilo *switch_dark_mode*, el cual coloca la bola de su interior en el extremo izquierdo, dejando ver el ícono de la luna. De esta forma estamos indicando que al hacer clic en el interruptor se pasará al modo oscuro que está asociado con el ícono de la luna. Dentro del estilo *switch_dark_mode* se definen todos los atributos del espacio que ocupa el interruptor, ya que los atributos de la bola se definen en la parte *::after* del estilo *switch_dark_mode*. Los atributos de la bola se definen en esta sección porque se trata de un pseudo-elemento del interruptor. Hasta ahora tendremos los estilos del interruptor cuando está inactivo y, por lo tanto, el sitio web se encuentra en el tema claro. Falta definir los atributos que cambiarian del estilo del interruptor al activar el interruptor. Para tener

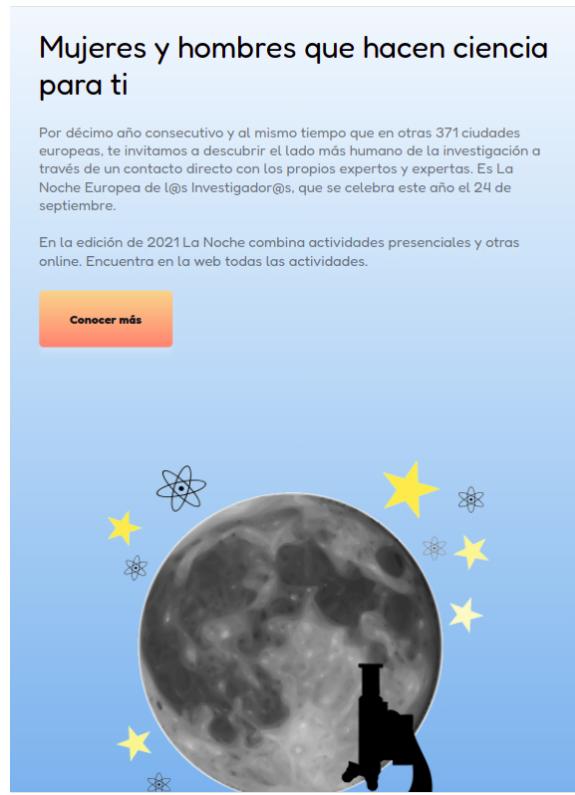


Figura B.2: Interfaz responsive usando la tablet en orientación vertical

de base los atributos del interruptor en modo inactivo y solamente tener que modificar el color y la posición de la bola al activar el interruptor, se añadirá o eliminará el estilo *active* al estilo *switch_dark_mode*. En este nuevo estilo *active* se indica cuál será el color del interruptor cuando se encuentra activo y sitúa la bola en el extremo derecho del interruptor, dejando ver el icono del sol, el cual está asociado con el modo claro y, por lo tanto, indicando que si se vuelve a pulsar sobre el interruptor se desactivará el interruptor y se volverá al modo claro.

Visualmente, el estado del interruptor se muestra distinto en modo inactivo () que en modo activo ().

Por último, falta implementar la parte dinámica de este interruptor, ya que hasta ahora únicamente hemos definido los elementos y los estilos que forman el interruptor, pero es necesario que el cambio de estilos del interruptor se realice mientras se utiliza el sitio web. Para implementar esta funcionalidad se crea un script de Javascript, el cual es incluido en todas las secciones del sitio web para poder hacer uso del interruptor en cada una de ellas. Dentro de este script se obtiene el elemento HTML que contiene al interruptor y posteriormente se define un evento de clic al pulsar sobre el

interruptor. Al definir este evento, asignamos la ejecución de una función al hacer clic sobre el interruptor. La función a ejecutar consiste en dos comandos toggle. Lo que hace un comando toggle en Javascript es añadir el estilo que se le indica, al elemento sobre el que se ejecuta el comando; o eliminar el estilo si el elemento sobre el que se ejecuta el comando ya contiene el estilo. En primer lugar, se añade o elimina el estilo *dark_mode* a todos los elementos de la página web. Y en segundo punto, se añade o elimina el estilo *active* al interruptor.

B.3. Implementación del despliegue del menú lateral

Para la implementación de esta funcionalidad serán necesarios dos elementos. Por un lado, el botón para desplegar el menú; y, por otro lado, el propio menú lateral.

El botón para desplegar el menú consiste en un simple botón de HTML, al cual se le añade un ícono (≡).

Para implementar el despliegue y la ocultación del menú lateral se hace uso de la misma táctica que con el interruptor (Apartado B.2) para el cambio de tema de colores. Esta táctica consiste en utilizar distintos estilos para variar la apariencia de los elementos que deseamos. Siguiendo esta táctica se creará el estilo *sidebar*, que afecta al menú lateral. Este primer estilo mantiene oculto al menú lateral. En segundo lugar, se genera el estilo *deploy*, que también afecta al menú lateral, y el cual hace visible al menú lateral. Con estos dos estilos podemos implementar la funcionalidad del menú lateral.

Para implementar la funcionalidad se hace empleo de un script de Javascript, el cual se coloca en todas las secciones del sitio web, al igual que se hace con el script para el interruptor. Dentro de este script se obtiene tanto el botón de despliegue como el elemento que contiene al menú lateral. Por último, asignaremos un evento de clic al botón obtenido. Al pulsar el botón se ejecutará una función que añadirá o eliminará el estilo *deploy* del menú lateral, haciendo visible o no visible respectivamente al menú lateral.

Un ejemplo de los dos estados que puede tener el menú lateral se pueden ver en la Figura B.3a, donde se puede ver el menú lateral oculto; y en la Figura B.3b, donde se puede ver el menú lateral despegado.

B.4. Inclusión de las ventanas emergentes en el sistema

El lenguaje Javascript tiene por defecto órdenes para crear ventanas emergentes, pero estas ventanas tienen un estilo muy básico. Para tener ventanas emergentes con un estilo interesante para el usuario se ha utilizado



Figura B.3: Estados del menú lateral

el plugin SweetAlert2. En su página oficial ² se muestran varios ejemplos de usos de este plugin. Para la creación de mis ventanas me he basado en varios de estos ejemplos. Para empezar a poder usar este plugin es necesario incluir en el archivo HTML de la página donde se quiera usar, un enlace a los archivos que forman el plugin.

Todas las ventanas disponen de una serie de atributos ³, los cuales pueden ser definidos o no. Entre ellos podemos encontrar el título de la ventana, el color de los botones, y muchos más.

Para lanzar la ventana y que se haga visible, se emplea la función *Swal.fire*. Pero aquí no queda la cosa porque existen ventanas que dan solamente información, y otras que dan información y esperan una interacción del usuario; lo interesante de estas últimas es que se debe gestionar la respuesta del usuario. Esta gestión se realiza en el *then* de la función *Swal.fire*, el cual se activa cuando esta función devuelve una *Promise*.

La información que se suele gestionar es la indicación de que botón ha pulsado el usuario. Para ello, este plugin dispone de funciones que nos facilitan conocer esta información, las cuales se pueden comprobar en su página oficial en el apartado "Handling buttons".

Hasta este momento se puede tener una idea de como funcionan estas ventanas emergentes. Falta explicar como las hemos utilizado en nuestro sistema. En nuestro caso hemos creado tres ventanas emergentes distintas. Todas las ventanas originadas hacen uso del botón de confirmación y de cancelación. Estas ventanas se describen a continuación:

- **Permiso para la toma de imágenes:** Esta ventana se lanza al pulsar sobre alguno de los botones que abre el chatbot en alguno de los canales disponibles. Estos botones se pueden ver en la Figura B.4.

La ventana emergente informa al usuario de que la deducción de la edad se va a producir a través de una imagen, y, por lo tanto, si quiere

²<https://sweetalert2.github.io>

³<https://sweetalert2.github.io/#title>



Figura B.4: Botones para el acceso a los canales de comunicación con el chatbot

o no que se le tomen imágenes. Además, se indica que no se puede revertir la decisión. Esta ventana se puede ver en la Figura B.5a.

En el caso de que el usuario confirme la toma de imágenes, se abrirá la sección de captura de imágenes. En caso contrario, al negar la toma de imágenes por parte del usuario se lanzará una nueva ventana emergente.

Este punto de decisión puede parecer innecesario si leemos la función de la siguiente ventana emergente, donde se decide de forma explícita por parte del usuario su edad. Pero esta opción de deducir la edad a través de las imágenes está pensada como una funcionalidad a futuro, ya que si nuestro sistema evoluciona pasando de un entorno estático a un entorno en movimiento, esta manera de deducir la edad sería ideal para adaptar nuestro sistema a este nuevo entorno. Por esta razón se ha incluido este punto de decisión para poder darle más posibilidades a futuro a nuestro sistema.

- **Indicación explícita de la edad:** Esta ventana emergente es la que se genera en caso de negar la toma de imágenes en la ventana emergente del punto anterior. En esta ventana se preguntará explícitamente la edad del usuario. Y además, aparecerán exactamente dos rangos de edad: Niño o Adulto. Esta ventana se puede ver en la Figura B.5b. Cuando el usuario elige alguno de los dos rangos de edad, se abrirá la interfaz del chatbot a través del canal que haya elegido el usuario cuando pulsó el botón que lanzó la primera ventana emergente.

- **Permiso para el envío de la imagen tomada:** Esta ventana se lanza al pulsar sobre el botón de “Enviar” () situado en la sección de captura de imágenes, el cual aparece una vez se ha capturado la imagen; y también se lanza cuando en el caso del móvil, se acepta enviar la imagen desde la aplicación de captura de imágenes del sistema. Esta última forma depende del dispositivo que se esté utilizando porque depende de la aplicación del sistema.

La ventana emergente informa al usuario de que se va a producir el envío de la imagen tomada por el dispositivo de captura, y, por lo tanto, si quiere o no que se realice el envío de la imagen en cuestión.

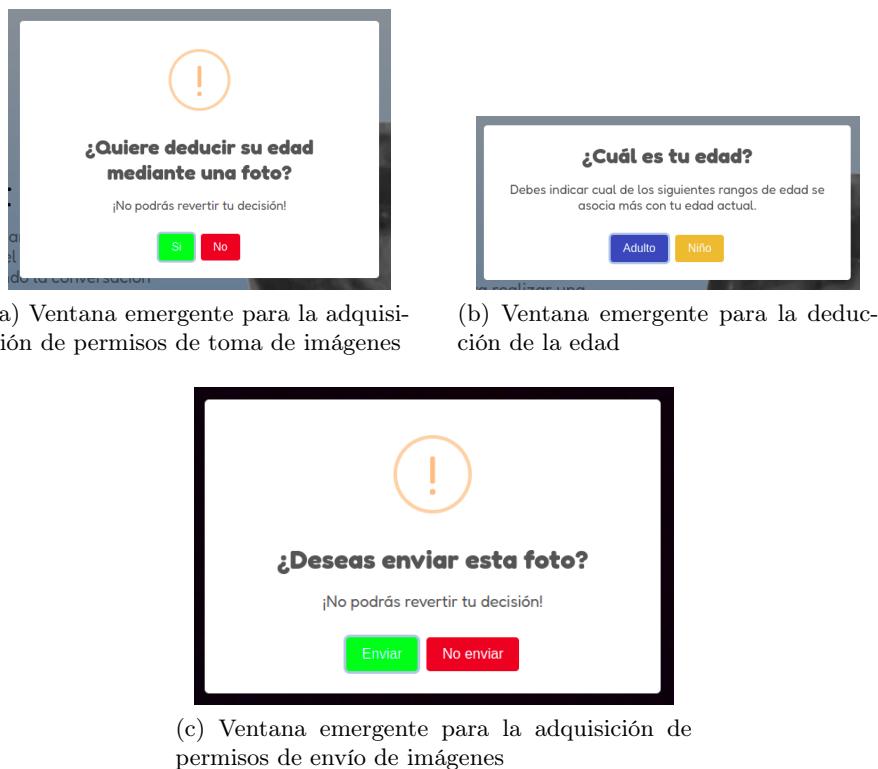


Figura B.5: Ventanas emergentes de la Vista

134 B.5. Inclusión de la interfaz web de Dialogflow en el sistema

Además, se indica que no se puede revertir la decisión, ya que al decidir no enviar la imagen se cierra la ventana emergente. Esta ventana se puede ver en la Figura B.5c.

En el caso de que el usuario confirme el envío de imágenes, se procederá a efectuar el envío de la imagen para su posterior análisis para deducir la edad del usuario. Durante el tiempo que dure el envío de la imagen y su análisis se mostrará una figura de carga para proporcionar feedback al usuario. En caso contrario, al negar el envío de la imagen por parte del usuario, se procederá a cerrar la ventana emergente y a esperar a que el usuario vuelva a tomar una imagen que le parezca adecuada y vuelva a iniciar el proceso de envío de la imagen.

B.5. Inclusión de la interfaz web de Dialogflow en el sistema

Para obtener esta interfaz web debemos acceder al chatbot que hemos creado en Dialogflow. En nuestro caso hemos generado un agente por cada rango de edad. Lo hemos hecho de esta forma para diferenciar con qué agente está charlando el usuario, ya que como hemos indicado anteriormente, al emplear una interfaz que no es nuestra, perdemos parte del control y en consecuencia, si tenemos un agente único, no podremos diferenciar los mensajes de un usuario del rango de Adulto, de los mensajes de un usuario del rango de Niño.

Una vez nos encontramos dentro del agente en Dialogflow, nos dirigimos al apartado de Integrations, el cual se puede ver en la Figura B.6.

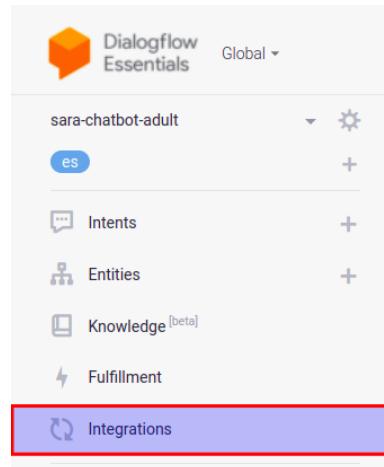


Figura B.6: Apartado Integrations de un agente de Dialogflow

Dentro de este apartado nos dirigimos a la sección Text based donde se

encuentra Web Demo, que es la interfaz web de Dialogflow. Este apartado se puede ver en su totalidad en la Figura B.7.

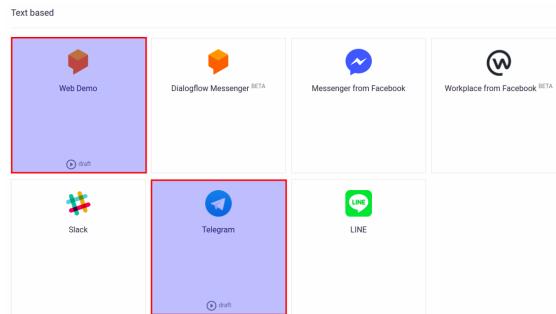


Figura B.7: Sección Text based del apartado Integrations

Al pulsar sobre Web Demo aparece una ventana emergente donde se puede habilitar esta interfaz para el chatbot que se esté utilizando. Esta ventana emergente se puede ver en la Figura B.8. Una vez habilitada se podrá hacer uso de ella mediante un iframe. Este iframe al colocarlo en un archivo HTML, incrustará el código HTML de la interfaz web de Dialogflow.

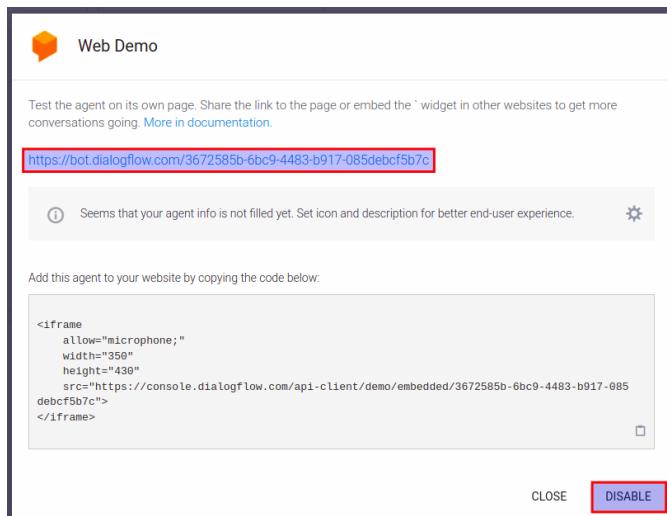


Figura B.8: Ventana emergente de Web Demo de Dialogflow

Para obtener el código del iframe se accede a la URL que aparece marcada en la Figura B.8. Al abrir esta URL nos abrirá una nueva ventana donde se puede ver el código del iframe del chatbot donde nos encontramos. Esta ventana se puede ver en la Figura B.9.

El código del iframe de uno de los chatbots creados en Dialogflow se puede ver en la Figura B.10. Como se puede apreciar se indica el tamaño de ventana que tendrá la interfaz. Para que la interfaz se adapte al tamaño que

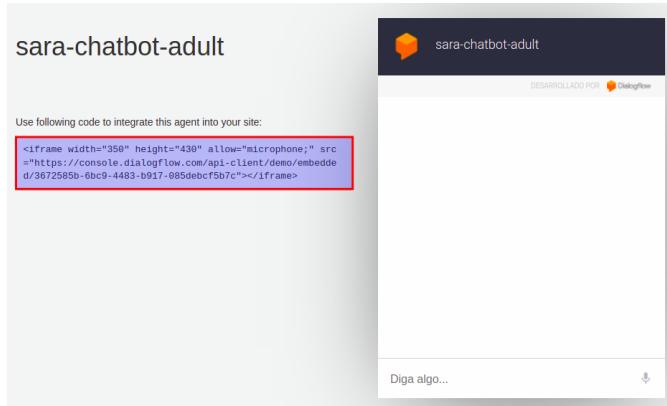


Figura B.9: Ventana donde se muestra el código del iframe

nosotros le indiquemos en nuestro sitio web, se eliminan estos dos atributos del iframe. El siguiente atributo que aparece son los permisos que necesita la interfaz web. Como se puede ver se indica el micrófono, ya que además de poder introducir texto de forma escrita, también se puede introducir de forma oral; pero para ello es necesario que el usuario de permisos para el uso del micrófono por parte de nuestro sitio web. Y por último, el atributo que aparece al final es la dirección al código de la interfaz web, el cual será integrado en nuestro sitio web.

```
<iframe width="350" height="430" allow="microphone;" src="https://console.dialogflow.com/api-client/demo/embedded/3672585b-6bc9-4483-b917-085debcf5b7c"></iframe>
```

Figura B.10: Código del iframe del chatbot para adultos

El aspecto final de nuestra interfaz web se puede ver en la Figura B.11.

B.6. Inclusión de la interfaz de Telegram de Dialogflow en el sistema

Para obtener la interfaz de Telegram primeramente debemos acceder a la sección de Telegram dentro del apartado de Integrations de Dialogflow, que se puede ver en la Figura B.7. Al pulsar en la sección de Telegram, aparecerá una ventana emergente parecida a la de Web Demo, donde se solicita el token del chatbot. Se solicita un token porque todos los bots en Telegram tienen asignado un token. Esta ventana emergente se puede ver en la Figura B.12.

Por lo que llegados a este punto debemos crear un bot en la aplicación de Telegram, para poder hacer uso de él como interfaz para nuestro chatbot. Para generar un chatbot en Telegram se debe acceder a BotFather, que es un

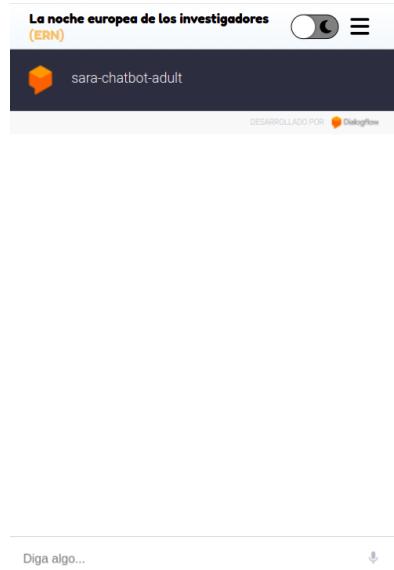


Figura B.11: Aspecto de la interfaz web del chatbot para adultos desde una tablet en orientación vertical

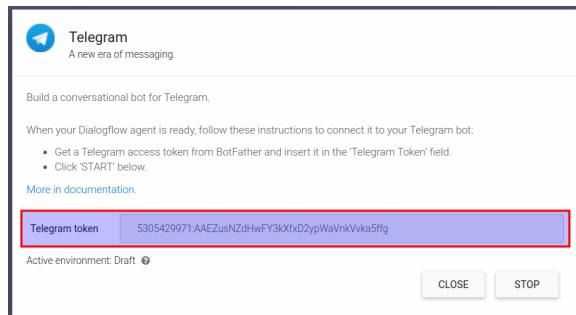


Figura B.12: Ventana emergente de Telegram de Dialogflow

bot cuyo objetivo es la creación del resto de bots que existen en Telegram. El chat con este bot se puede ver en la Figura B.13.

Con el comando /newbot se inicia el proceso de creación del bot. Durante este proceso se nos solicita el nombre del bot; y posteriormente se nos solicita el username del bot, el cual debe terminar con la palabra "bot.º" "Bot". Una vez terminado este proceso aparecerá un mensaje con el token asociado a nuestro nuevo bot. De la misma forma que para la interfaz web se producen dos chatbots en Dialogflow, para la interfaz de Telegram también se deben crear dos bots y obtener los tokens de ambos.

Una vez tenemos este token lo introducimos tal y como aparece en la Figura B.12.

El aspecto final de nuestra interfaz de Telegram será el que se muestra

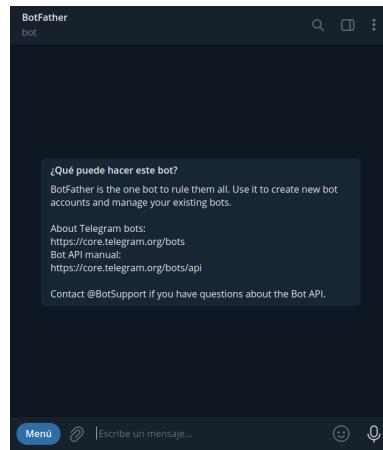


Figura B.13: Chat con el bot de Telegram BotFather

en la Figura B.14.

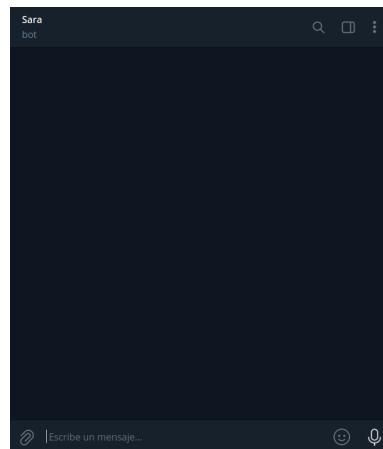


Figura B.14: Aspecto de la interfaz de Telegram del chatbot para adultos

B.7. Razones adicionales de la elección del framework para el servidor

Por supuesto, existe una gran cantidad de frameworks basados en Python para la elaboración de API's, pero como se puede ver en la Figura B.15, donde se compara el rendimiento de distintos frameworks. El framework FastAPI es el que obtiene unos mejores resultados entre los frameworks basado en Python, de ahí su elección para nuestro servidor, ya que queremos que las peticiones al servidor se gestionen lo más rápido posible para agilizar la comunicación con el chatbot y el movimiento por el sitio web.



Fuente: <https://www.techempower.com/benchmarks/#section=data-r20&hw=ph&test=db>

Figura B.15: Número de respuestas por segundo para peticiones (Datos del 02/08/2021)

B.8. Despliegue del servidor en Heroku

El despliegue a la plataforma Heroku se puede efectuar por tres vías. Una vez que creamos nuestra app en la plataforma de Heroku, si accedemos a la misma y nos vamos a la sección Deploy, podremos ver las tres vías que existen. Estas vías se pueden ver en la Figura B.16.



Figura B.16: Métodos de despliegue de apps en Heroku

En mi caso he optado por el tercer método, el cual involucra a los contenedores. Por esta razón se menciona en anteriores párrafos el uso de Docker, ya que se efectuará el despliegue de la app como un contenedor. Las ventajas de los contenedores son que producen una menor sobrecarga, permiten una mayor portabilidad, y son más eficientes, entre muchos otros.

Los pasos a seguir siguiendo este método se explican en el apartado Container Registry⁴ de la sección Deploy de la app. Los únicos requisitos para este método es tener instalado en el sistema tanto Heroku CLI como Docker.

Pero yo he seguido una vía alternativa, dado que estamos usando Github

⁴<https://dashboard.heroku.com/apps/sara-chatbot-tfg/deploy/heroku-container>

como plataforma para el control del código del proyecto, he optado por sacar potencial a la funcionalidad de los Workflows que se nos proporciona en Github. Estos Workflows no son más que procesos automatizados que se pueden configurar con uno o más trabajos. Para la creación de los archivos que componen a estos Workflows se hace empleo del lenguaje YAML.

Basándonos en estos Workflow he configurado un trabajo que realiza el despliegue del Controlador en la plataforma Heroku de forma automática cuando se percibe por parte de Github que ha habido cambios en el código del Controlador.

Además, para una mayor capacidad de configuración se ha hecho uso de los Actions secrets de Github. Esta funcionalidad de Github permite guardar valores en variables que puede ser utilizadas desde los trabajos de los Workflows, y siendo además privados, por lo que se pueden guardar datos sensibles como contraseñas.

Siguiendo con el trabajo que efectúa el despliegue del Controlador, este trabajo se podría haber implementado en su totalidad por mí, pero para ahorrar tiempo y evitar errores que pudiesen surgir he optado por usar un Action creado por otra persona. Los trabajos de los Workflows se componen de una serie de Actions. El Action utilizado pertenece a AkhileshNS y se llama heroku-deploy. Para facilitar el uso de este Action se dispone de ejemplos de uso en su repositorio de Github. Los datos que requiere este Action para realizar el despliegue es la API key de nuestra cuenta de Heroku, el nombre que le hemos asignado a nuestra app a la hora de generarla en Heroku, en nuestro caso es sara-chatbot-tfg, el email empleado a la hora de registrarnos en la plataforma de Heroku, y por último el directorio donde se encuentra el Dockerfile que define al contenedor. Los tres primeros datos estarán guardados en tres variables distintas dentro de los Actions secrets de nuestro repositorio de GitHub, donde se controla el código de nuestro proyecto.

Hasta este punto tenemos una forma de efectuar el despliegue automático, pero necesitamos definir como debe formarse el contenedor. Como se ha indicado en el anterior párrafo, al Action es necesario indicarle donde se encuentra un archivo en concreto, el Dockerfile. Este archivo es vital para la elaboración de un contenedor. Un Dockerfile es un archivo de texto simple que define un conjunto de comandos que se ejecutará para formar el entorno del contenedor. Para generar nuestro contenedor del Controlador hemos usado como base del contenedor una imagen de Python cuya versión es 3.9.10 . Posteriormente, se copia todo el contenido del directorio, incluido el propio directorio, que contiene todo el código del Controlador en la raíz del contenedor. Después de realizar la copia se establece como directorio de trabajo el propio directorio que se ha copia en el contenedor. Seguidamente, procedemos a la instalación de los requisitos del Controlador mediante el comando pip. Y finalmente, establecemos como punto de entrada al contenedor la ejecución del script main.py, el cual ejecutará el inicio del servidor del Controlador.

Después de todos lo dicho ya tendríamos un servidor para el Controlador al que se puede acceder de forma pública y en cualquier momento.

Aunque el trabajo que efectúa el despliegue no es el único que he elaborado relacionado con el Controlador, ya que puesto que estamos usando contenedores para realizar el despliegue, pensé que era buena idea subir el contenedor que se obtiene con el controlador a la plataforma DockerHub⁵. Esta plataforma es el equivalente a GitHub para contenedores, sirve como plataforma para albergar contenedores en repositorios privados o públicos. Esta subida se hace con otro trabajo dentro del mismo Workflow del despliegue del Controlador para aprovechar la automatización lograda. El único requisito para proceder a la subida es tener instalado en la máquina el paquete Docker. Una vez lo tenemos instalado la subida consiste en los siguientes pasos:

- Login en la plataforma de DockerHub con nuestro usuario y contraseña
- Construcción de la imagen del contenedor
- Subida de la imagen al repositorio sara-chatbot-controller

Ahora, al disponer de la imagen del contenedor en un repositorio, se podría cambiar fácilmente la plataforma donde se despliega siempre y cuando se pueda efectuar el despliegue mediante contenedores.

Llegados a este punto tendríamos una estructura de despliegue del Controlador totalmente automatizada.

B.9. Proceso de generación de plantillas renderizadas

Para generar una plantilla renderizada se hace uso del método TemplateResponse del objeto templates. Este método requiere de dos parámetros. El primer parámetro será una cadena con la ruta relativa de la plantilla que se quiere renderizar. Esta ruta relativa tomará como base el directorio que se indicó en la creación del objeto templates, este directorio es el directorio que contiene todas las plantillas del servidor llamado templates. El segundo parámetro será un diccionario donde cada par clave-valor estará compuesto por una cadena que será la clave y por una variable que será el valor. Es importante la cadena que se utilice como clave, ya que con ese nombre se podrá acceder al valor asignado desde el código de las plantillas. Algo relevante a destacar es que al menos este diccionario debe tener un elemento, el cual es el parámetro de tipo Request que se indicó en la lista de pasos.

⁵<https://hub.docker.com>

B.10. Paso de la información a los script de Javascript

Al hacer empleo del diccionario hacemos que esta información únicamente sea accesible desde la plantilla, pero no desde el código Javascript, que es donde realmente es necesaria esta información, ya que es el lugar donde se ejecuta la lógica de las plantillas. Para acceder a la información de una clave se debe escribir en la plantilla de código HTML el nombre de la clave dentro de dos llaves a ambos lados, es decir, si la clave se llama estado para acceder a su valor se debe escribir `{{estado}}` . Esta forma de acceder a la información solamente es posible desde el código HTML, en este punto es donde entra en juego el uso del lenguaje HTML dentro de la implementación del Controlador, la cual se mencionó en el apartado de herramientas de desarrollo del Controlador (Apartado 7.2.1).

Para poder acceder a esta información desde los scripts de Javascript se creará un párrafo de HTML para cada clave que se pase al diccionario, el cual contendrá la información de esa clave. Este párrafo tendrá un estilo que le otorgue la visibilidad de oculto. Y para acceder a la información se obtendrá el elemento HTML desde el script de Javascript y ya tendremos acceso a esa información. Un ejemplo de definición de un párrafo para una clave se puede ver en la Figura B.17.

```
<!-- Estado del switch de cambio a modo oscuro en la anterior ventana -->
<p id="dark_mode" class="oculto">{{ dark_mode }}</p>
```

Figura B.17: Ejemplo de definición de una clave en una plantilla HTML

B.11. Solución a la inactividad de las apps de Heroku

Una solución provisional que se suele usar es la ejecución de comandos ping al servidor de manera periódica. Para la ejecución periódica de estos comandos se hace empleo de la plataforma cron-job. Dentro de esta plataforma se pueden crear trabajos que se ejecutan cada cierto tiempo, tiempo que se puede definir en el trabajo. Estos trabajos ejecutan un comando ping a la dirección URL que se le indique en el trabajo. Para gestionar estos ping al servidor se ha generado la sección wakeup en el servidor del Controlador. A esta sección será a la que se realicen los comandos ping. Esta sección lo único que hace es devolver un texto plano indicando el estado activo del servidor.

B.12. Proceso para la gestión de la conversación desde el Controlador

La gestión de la petición de Dialogflow por parte del Controlador se efectúa de manera similar independientemente del rango de edad, el único cambio es que la respuesta se obtiene efectuando la llamada a secciones distintas del servidor del Modelo. La gestión de la petición comienza obteniendo el Intent que lanzó la petición, ya que según el Intent se realizan algunos pasos distintos. El chatbot tendrá dos Intents, el Intent Talk y el Intent Goodbye. En el Intent Talk se efectúa toda la conversación, excepto la última interacción con el chatbot, que se realiza en el Intent Goodbye.

Si se trata del Intent Talk, el siguiente paso será comprobar si se trata de la primera generación de respuesta o no. Para obtener esta información se revisa el contexto que contiene la petición. Si dentro del contexto está el campo edad quiere decir que no es la primera respuesta, en caso contrario indica que si se trata de la primera respuesta.

Cuando se trata de la primera generación de respuesta, primeramente se verifica si el servidor del Modelo se encuentra activo. Si no se encuentra activo se le indica al usuario mediante una respuesta a través del chatbot. Si se encuentra activo, se extrae la información necesaria del contexto de la petición para efectuar la generación de la respuesta. Esta información es el texto de entrada y la edad, la cual se deduce de forma sencilla al ver a que webhook ha llegado la petición. Una vez se ha generado la respuesta, se elaborará el primer contexto de la conversación. Para la elaboración de este primer contexto será necesario obtener la fecha actual, que marcará el inicio de la conversación. Al contexto se añadirá la información de la fecha de inicio, la edad del usuario, se incluirá la entrada y la salida que se ha generado a partir de esa entrada y finalmente se añadirá el identificador de la conversación, el cual es muy importante para seguir la conversación en el resto de llamadas al webhook. La respuesta a la petición del webhook estará compuesta por el texto que se ha generado como respuesta al texto de entrada, y por el nuevo contexto que se ha generado.

Cuando se trata del resto de generaciones de respuestas, habrá pasos que se eviten sustituyéndolos por otros. En este caso, a la hora de generar la respuesta será necesario, el texto de entrada, al igual que pasaba en el otro caso; la edad, que en este caso se extraerá del contexto de la petición; y el identificador de la conversación, para indicar al Modelo de que se trata de una respuesta que continúa una conversación ya empezada. La generación del contexto evitará el añadir la fecha de inicio y la edad, ya que fueron añadidos en la primera generación. En esta generación de contexto solamente se añadirá el texto de entrada y la respuesta a la misma, y se actualizará el identificador de la conversación. La respuesta a la petición del webhook se elaborará de la misma forma.

144B.13. Establecimiento de la conexión entre el servidor y el log

Una vez hemos detallado el proceso con el Intent Talk. Si se trata del Intent Goodbye, los pasos a seguir son exactamente iguales a los realizados con el Intent Talk cuando no es la primera generación, con la única salvedad de que al final de todos los pasos se guardará la conversación en el log del sistema, para poder utilizar esta información para futuras mejoras del chatbot y también para el análisis de las conversaciones que se han realizado con el chatbot. El contexto no se guarda tal cual en el log, sino que más bien se formatea la información del contexto para guardarla de una manera clara en ella. Cabe destacar que antes de guardar la conversación se debe obtener la fecha actual para usarla como la fecha de finalización de la conversación. El guardado de la conversación en el log se explica en el Apartado 7.2.8.

La generación de la respuesta consistirá en el envío de una petición al servidor del Modelo. La petición se enviará a la sección del servidor del Modelo acorde con la edad que se haya indicado a la función de generación de respuestas. Esta petición contendrá como información el texto de entrada, el identificador de la conversación en caso de conocerse, y un indicador de si es la última respuesta, información que es útil para el Modelo para su propia gestión de las conversaciones. La respuesta se obtendrá de la respuesta a esta petición al servidor del Modelo.

El contexto que se encuentra dentro de la petición al webhook está compuesto de varios elementos, ya que un chatbot en Dialogflow puede tener un contexto compuesto de varios elementos, donde cada elemento tenga cierta información. Pero para nuestro chatbot toda la información se guardará en el mismo elemento. El contexto donde se guardará toda la información será el llamado *talk-followup*.

B.13. Establecimiento de la conexión entre el servidor y el log

Para efectuar la conexión se sigue la documentación de Heroku relacionada con la conexión del log con Python⁶. En primer lugar, será necesario tener instalada la librería *psycopg2-binary*. Dentro del código del servidor se hará import de la librería para poder acceder al log de la app. El primer paso para acceder al log, es la conexión con la misma. La conexión se realiza mediante la función *connect* de la librería. Esta función requiere de la URL del log, la cual se puede obtener mediante las variables de configuración de nuestra app. Esta función devolverá un objeto que representará la conexión con el log. Una vez hemos efectuado la conexión, debemos ver como escribir los datos de la conversación en el log, para ello es necesario obtener el cursor del log. Con este cursor se podrán ejecutar comando SQL sobre el log utilizando los datos de la conversación. Antes de realizar la escritura de los datos se deberán

⁶<https://devcenter.heroku.com/articles/connecting-heroku-postgres>

obtener las conversaciones con un formato claro, para facilitar su análisis en el log. Finalmente, cuando tenemos todos los datos listos, ejecutamos un comando INSERT sobre la tabla que contendrá las conversaciones. Este comando insertará la edad del usuario, la fecha de inicio de la conversación, la fecha de finalización de la misma y el contenido de la conversación, es decir, las distintas entradas y salidas de la conversación.

Cuando ya hemos insertado los datos en la tabla debemos comunicar al log los cambios mediante la función *commit*. De esta forma, los cambios se harán permanentes en el log. Llegados a este punto ya hemos terminado el guardado de la conversación, por lo tanto, ya podemos cerrar tanto el cursor como el log. Ambos cierres se realizan con la función *close* aunque sobre distintos objetos.

B.14. Cabeceras añadidas a las conexiones con el servidor del Modelo

Estas cabeceras son las llamadas cabeceras CORS o cabeceras de Intercambio de recursos de origen cruzado. Dado que para la implementación del servidor se hace uso del framework FastAPI, utilizaremos las herramientas que nos proporciona este framework para solucionar este problema. La herramienta a utilizar en este caso es la adhesión de un middleware de tipo CORS, para ello se debe hacer uso de la función *add_middleware* y de la clase *CORSMiddleware*. Como se indica en la documentación de FastAPI, este tipo de middleware se utiliza en situaciones en las que la interfaz tiene código Javascript que se comunica con un backend y el backend está en un dominio diferente al de la interfaz.

B.15. Proceso de despliegue del servidor del Modelo mediante Ngrok

Por supuesto, antes de nada es necesario importar la librería pyngrok, así como instalar ngrok en el sistema donde se ejecutará el servidor para poder generar la conexión entre el servidor e internet.

En primer lugar, deberemos definir la configuración del túnel que se originará. Esta configuración se guardará en un archivo de tipo YAML. Dentro de este archivo encontraremos la siguiente información:

- Token de autenticación de Ngrok
- Región de la conexión

El token de autenticación se puede extraer de la sección Your Authtoken de nuestra cuenta en la plataforma de Ngrok.

Esta configuración se definirá en el código mediante la clase Pyngrok-Config. Esta clase necesita de dos elementos para poder crear una instancia. Por un lado, la ruta al ejecutable de ngrok que se ha generado al realizar la instalación de ngrok en el sistema; y, por otro lado, la ruta al archivo de configuración anteriormente descrito.

Finalmente, se efectuará la conexión, obteniendo la URL pública del servidor. Para realizar la conexión se hará uso de la configuración definida anteriormente y del puerto utilizado para ejecutar el servidor, el cual es el mismo que se utilizó con el módulo Unicorn. Llegados a este punto habremos efectuado la conexión y dispondremos de un servidor para el Modelo totalmente funcional y accesible.

Pero si nos fijamos la forma de desplegar el servidor del Modelo, causa que tengamos una URL pública distinta cada vez que iniciamos o reiniciamos el servidor, a diferencia de lo que pasa con el servidor del Controlador que tiene una URL pública fija al emplear la plataforma Heroku para su despliegue. Esta condición del servidor del Modelo provoca la creación de una funcionalidad que comparte la URL del servidor del Modelo con el servidor del Controlador. Para ello se ha creado una función que envía la URL pública almacenada en el servidor del Modelo a través de una petición POST con destino a la sección para fijar la URL del servidor del Controlador, la cual fue descrita en la implementación del Controlador. La petición POST contendrá la URL pública del servidor del Modelo.

Esta función se ejecutará junto al código que arranca la ejecución del servidor. Además, se creará una sección en el servidor del Modelo, cuya funcionalidad consista en volver a enviar la URL al servidor del Controlador a pesar de haber iniciado el servidor. La finalidad de esta sección es volver a conectar ambos servidores cuando sea el servidor del Controlador el que se reinicie mientras el servidor del Modelo siga activo.

Un tip que puede ser útil para la gestión de las conexiones con Ngrok, es el uso del comando *killall ngrok*. Este comando elimina todas las conexiones establecidas actualmente mediante Ngrok. Puede llegar a ser útil si el servidor del Modelo finaliza de forma inesperada y no es capaz de finalizar el proceso que ejecuta el servidor adecuadamente. Este problema surge sobre todo cuando se utiliza la versión gratuita de Ngrok, ya que solamente se dispone de una conexión disponible.

B.16. Preprocesado de los conjuntos de datos

El primer paso en el pre procesado es la obtención de los conjuntos de datos para cada rango de edad a partir del conjunto de datos inicial. En primer lugar, se realiza una resumen de todos los textos del conjunto de datos, ya que vamos a implementar un chatbot que tenga respuestas cortas o medianas, pero que no sean excesivamente largas. Este resumen se efec-

tuará con un modelo basado en Transformers extraído de Huggingface. El modelo empleado es pegasus-xsum de la empresa Google. Una vez tenemos los textos resumidos, llega el momento de obtener textos diferentes según el rango de edad. Para el caso de los textos para adultos no se hará ninguna modificación adicional, mientras que para los textos para niños si será necesaria una transformación. Dado que los niños tiene una menor capacidad de comprensión, los textos deberán sufrir un proceso de simplificación. Esta simplificación se llevará a cabo mediante modelos preentrenados usados en los experimentos Multilingual Unsupervised Sentence Simplification (MUSS). El código de estos modelos se encuentra en un repositorio de Github de Meta Research. Dentro de este repositorio existen cuatro modelos, pero no todos utilizan texto en inglés. El modelo que he utilizado para mi proyecto es *muss_en_wikilarge_mined*. Tras realizar la simplificación, tendremos todo lo necesario como para obtener dos conjuntos de datos, cada uno orientado a un rango de edad.

El segundo paso del pre procesado es la unión de cada uno de estos conjuntos de datos orientados al rango de edad con un conjunto de datos que contiene información general que será útil para ambos modelos, como puede ser el nombre del chatbot.

Finalmente, el último paso del pre procesado consistirá en dar el formato adecuado a los dos conjuntos de datos generados para que puedan ser usados para el entrenamiento de nuestros modelos. En primer lugar, se deberán dividir ambos conjuntos de datos en sus correspondientes partes de entrenamiento y de validación. Y a su vez, todos los conjuntos de datos generados de esta división deberán reducir sus columnas, dejando únicamente las columnas de Question y de Answer. Además, estas columnas se renombrarán como source y target respectivamente, para que los conjuntos de datos puedan ser empleados en el proceso de ajuste de los modelos.

B.17. Extracción de información de Quora

Para la extracción de información se va a utilizar el módulo Selenium, el cual tiene implementación en Python. Selenium es un conjunto de utilidades que se emplean para la elaboración de pruebas de aplicaciones web. En nuestro caso usaremos este módulo para recorrer la página de Quora de tal forma que vayamos extrayendo la información que necesitemos de la misma. Selenium puede usar varios navegadores para realizar las pruebas. En el caso de nuestro programa se utilizará el navegador Chrome. Para poder emplear este navegador necesitamos descargar los drivers acordes con nuestra versión de navegador que tengamos instalada en el sistema. En mi caso, la versión de Chrome utilizada es la 101.0.4951.64. Esta información se puede comprobar en la sección de Información de Chrome de la configuración del navegador

⁷. La implementación de este programa extractor de información no ha sido fácil debido a la forma en que está estructura la página, ya que se hace poco uso del campo id en los distintos elementos de la página.

Los parámetros que recibirá el programa son los siguientes:

- Usuario de la cuenta en Quora
- Contraseña de la cuenta en Quora
- Número máximo de preguntas a extraer de cada tema
- Ruta al archivo que contiene las URL's hacia los temas sobre los que extraer información
- Ruta del archivo que guardará toda la información extraída

El formato que tendrán los datos extraídos, los cuales se guardarán en el archivo indicado en la anterior lista, estará compuesto por cuatro columnas. Las columnas serán Topic, que indicará el tema; Subject, que indicará una subsección del tema; Question, que será el texto de la pregunta; y Answer, que será el texto de la respuesta.

⁷<chrome://settings/help>

Apéndice C

Manual de usuario

C.1. Instalación

Para ambas instalaciones se deberá crear un repositorio propio que contenga el código de nuestro repositorio en Github¹.

C.1.1. Instalación del servidor de la Vista y el Controlador

Una vez tenemos el repositorio debemos clonar el repositorio en nuestra máquina. Suponiendo que hemos creado una app en nuestra cuenta de Heroku, deberemos actualizar la información de nuestro Workflow para el servidor de la Vista y el Controlador con la información de nuestra cuenta y nuestra app; como son el email de nuestra cuenta de Heroku, el nombre de la app y la key de la API de nuestra cuenta.

Una vez tenemos actualizada la información del Workflow, al realizar un commit sobre el código de la carpeta app de nuestro nuevo repositorio, se realizará el despliegue de la app en la plataforma Heroku.

C.1.2. Instalación del servidor del Modelo

Esta instalación se realizará en una máquina distinta con GPU, suponiendo que la máquina utilizada para la primera instalación no dispone de GPU. La instalación consistirá en clonar el repositorio en esta máquina con GPU. Adicionalmente se deben descargar los modelos utilizados para todas las funcionalidades del Modelo. Los modelos usados son los siguientes:

- facebook/blenderbot-400M-distill
- mrm8488/t5-base-finetuned-question-generation-ap
- google/pegasus-xsum

¹https://github.com/Mario-Carmona/SARA_Chatbot

- nateraw/vit-age-classifier

Estos modelos deberán ser descargados y movidos a la misma altura que la raíz del repositorio de trabajo.

Finalmente se deberá ejecutar el script de shell llamado *setup.sh*. Los cinco primeros comando de este script deben ser eliminados si no se utilizan los servidores GPU del Instituto DaSCI.

C.2. Inicio del sistema

El servidor que se ha desplegado en Heroku se estará ejecutando indefinidamente. Y en cuanto al servidor del Modelo para iniciar su ejecución se deberá ejecutar el siguiente comando a la altura de la carpeta *server_gpus*:

```
deepspeed --num_gpus 1 main.py --config_file
           configs/config_server.json
```

Tras la ejecución del comando y la finalización de la carga de todos los modelos que utiliza este servidor, el sistema estará disponible en su plenitud.

C.3. Generación de los conjuntos de datos de entrenamiento

En primer lugar, se deberá generar un conjunto de datos inicial mediante la extracción de información de la página Quora. La extracción del conjunto de datos inicial se realiza mediante el siguiente comando:

```
./prueba_extract.py -u "<Correo de Quora>" -p "<Contraseña de
Quora>" -n <Número de ejemplos por Topic>-t <Ruta al archivo con la
lista de Topics>-f <Ruta al archivo de salida>
```

Una vez finalizada la extracción del conjunto de datos inicial, procedemos a generar los conjuntos de datos de entrenamiento. Además del conjunto de datos extraído de Quora se pueden generar conjuntos de datos creados a mano, los cuales se concatenarán al conjunto de datos de Quora durante el preprocesado. Deberemos actualizar el archivo de configuración de generación de conjuntos de datos, *config_genDataset.json*, y posteriormente ejecutar el siguiente comando:

```
python generate_dataset.py configs/config_genDataset.json
```

Tras la ejecución de este comando, obtendremos los conjuntos de datos de entrenamiento en la ruta que se ha indicado en el archivo de configuración.

C.4. Entrenamiento de los modelos

Suponiendo que se han generado los conjuntos de datos de entrenamiento. Primeramente se deberá actualizar el archivo de configuración para el entrenamiento de cierto rango de edad, y posteriormente se ejecuta el siguiente comando:

```
python fine_tuning.py configs/config_finetuning_child.json
```

En el caso del anterior comando, tras su finalización se obtendrá un modelo conversacional entrenado para el rango de edad de niños.

Glosario

Add-ons Un add-on es un programa o módulo adicional que complementa navegadores, programas de correo y otras aplicaciones de software y hardware y añade funciones que el sistema básico no tiene. 88

API Según la página web de IBM (19), una API o interfaz de programación de aplicaciones, es un conjunto de reglas que determinan cómo las aplicaciones o los dispositivos pueden conectarse y comunicarse entre sí. 111, 112, 120, 122, 123, 153

API REST Una API REST es un API que sigue los principios de diseño de REST. El principal objetivo de estos principios es tener una API que posibilite la separación entre cliente y servidor, para ello algunos de sus principios son por ejemplo: tener un interfaz uniforme, es decir, todas las solicitudes hacia un recurso de la API son siempre iguales; o no tener estado, es decir, la API no guarda información, todas la información necesaria para resolver la solicitud debe estar contenida en la misma. 22

back-end El back-end es la capa que se encuentra por debajo del front-end. Es la capa no visible para el usuario. En ella se encuentra el funcionamiento de la aplicación, incluyendo servidores y bases de datos. No existe un lenguaje de programación específico para esta capa. 22, 123, 125, 154

backup Es una copia de seguridad de los datos. Al hacer un backup, se pueden restaurar los datos posteriormente en caso de pérdida. 48, 55

bot Un bot ² es un programa informático que trabaja de forma autónoma. 22

canvas Canvas (lienzo) es un elemento HTML que permite la generación de gráficos y animaciones de forma dinámica por medio de scripts tras elegir el dominio para tu web. 87

²<https://dictionary.cambridge.org/es-LA/dictionary/english/bot>

chatbot Un chatbot³ es un programa informático diseñado para tener una conversación con un humano, especialmente a través de Internet. 22, 23

cloud Cloud significa, literalmente, nube. En términos informáticos nos referimos a un paradigma que permite ofrecer servicios de computación a través de una red, que normalmente es Internet. 114

CNN Una red neuronal convolucional (CNN) es una arquitectura de red para deep learning que aprende directamente de los datos, sin necesidad de extraer características manualmente. 120

DPO El módulo DPO (Dialogue Policy Optimization) es el encargado de aplicar las políticas de optimización sobre el chatbot. 29

DST El módulo DST (Dialogue State Tracker) es el encargado de ir estimando el objetivo del usuario a medida que avanza el diálogo. 29

ECTS ECTS es la sigla correspondiente al European Credit Transfer System (Sistema Europeo de Transferencia de Créditos) y es el sistema adoptado por todas las universidades del Espacio Europeo de Educación Superior (EEES) para garantizar la homogeneidad y la calidad de los estudios que ofrecen. 61

feedback El feedback (retroalimentación) es la acción de ofrecer información a una persona sobre un resultado. 48

framework Un framework es un esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto con objetivos específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de software. 82, 84, 89, 90, 110, 112, 113

front-end El front-end es la capa que se encuentra por encima del back-end. Es la capa visible para el usuario y también es con la que interactúa. Debido a que interactúa con ella debe cumplir unos estándares de usabilidad y estética. Los principales lenguajes de programación con los que se implementa esta capa son: Javascript, CSS y HTML. 22, 123, 125, 153, 154

Full Stack La programación Full Stack abarca tanto la programación del front-end como la del back-end. Los profesionales de esta programación se denominan programadores Full Stack y tienen las habilidades necesarias para administrar un proyecto completo. 22

³<https://dictionary.cambridge.org/es-LA/dictionary/english/chatbot>

Github Github es una plataforma donde multitud de personas y empresas desarrollan software. Esta plataforma permite la comunicación e interacción de la comunidad de desarrolladores de software. Los proyectos se alojan en lo que se denominan repositorios, los cuales pueden ser públicos o privados. Cada repositorio tiene un control de versiones mediante Git. 22

inteligencia artificial La inteligencia artificial (IA) es, en ciencias de la computación, la disciplina que intenta replicar y desarrollar la inteligencia y sus procesos implícitos a través de computadoras. 25, 26, 30, 35, 113, 114, 117, 121

internet de las cosas El Internet de las cosas (IoT) es el proceso que permite conectar elementos físicos cotidianos al Internet. 34

link Un link es un elemento que establece un vínculo con otro recurso. 70, 72

mockup Un mockup, traducido del inglés como bosquejo, es un fotomontaje a través del cual los diseñadores gráficos pueden presentar sus propuestas a los clientes. 127

PaaS PaaS significa Plataforma como servicio. Se trata de un conjunto de servicios basados en la nube que permiten a los usuarios empresariales y desarrolladores crear aplicaciones de forma rápida y rentable. 83

plugin Los plugins son complementos que añaden funcionalidades extra o mejoras a los programas. 79, 80

portfolio Un portfolio es un conjunto de elementos. 113, 115

proceso de decisión de Markov Son las forma idealizada matemáticamente del problema de aprendizaje por refuerzo. Estos procesos describen formalmente el medio ambiente en el cual se desarrolla el aprendizaje por refuerzo. 31

prototipo Un prototipo es un primer modelo que sirve como representación o simulación del producto final y que nos permite verificar el diseño y confirmar que cuenta con las características específicas planteadas. 41

PyTorch PyTorch es una biblioteca de aprendizaje automático de código abierto basada en la biblioteca de Torch. 74

redes generativas adversativas Son una clase de algoritmos de inteligencia artificial que se utilizan en el aprendizaje no supervisado, implementados por un sistema de dos redes neuronales que compiten mutuamente en una especie de juego de suma cero. 33

responsive Según Brett S. Gardner (20), el diseño responsive de las páginas web permite crear una única página web que es capaz de adaptarse a la interfaz y al contenido de la misma para poder ser visible en distintos dispositivos (móviles, tablets y ordenadores). Este diseño permite mejorar la experiencia del usuario. Actualmente el diseño responsive se implementa a través del lenguaje CSS, en concreto con su versión CSS3, la cuál tiene un enfoque centrado en el usuario permitiendo el diseño de aplicaciones web dinámicas. 22

Siracusa Siracusa es una ciudad de Italia, situada en la costa sudeste de la isla de Sicilia, famosa como centro cultural desde la Antigua Grecia. 22

TIC Conjunto de técnicas y equipos informáticos que permiten comunicarse a distancia por vía electrónica. 41, 42, 55

tip Tip es un término inglés que puede traducirse como “consejo” o “sugerencia”. 85

Transformers Se trata de una arquitectura de redes neuronales que en estos momentos se considera estado del arte en modelos secuenciales. 74

técnicas avanzadas de comprensión del lenguaje natural La comprensión de lenguaje natural es una parte del procesamiento de lenguaje natural en inteligencia artificial que trabaja con la comprensión de lectura en dispositivos electrónicos. 27, 29, 120

webhook Los webhooks son herramientas para que las diferentes aplicaciones puedan comunicarse entre sí. 118

World Wide Web La World Wide Web (W3) fue desarrollada por un conjunto de personas, de entre las que destaca Tim Berners-Lee, a finales de la década de los ochenta. El motivo por el que surgió este desarrollo fue el interés por parte del CERN de poder compartir sus investigaciones con el resto del mundo y por tanto mejorar la comunicación en el mundo investigador. 21, 109

épocas Recorrido completo de todos los datos de entrenamiento. 122

Bibliografía

- [1] B. Luo, R. Y. K. Lau, C. Li, and Y.-W. Si, “A critical review of state-of-the-art chatbot designs and applications,” *WIREs Data Mining and Knowledge Discovery*, vol. 12, no. 1, p. e1434, 2022, <https://doi.org/10.1002/widm.1434>; 27. [Online]. Available: <https://doi.org/10.1002/widm.1434>
- [2] E. Adamopoulou and L. Moussiades, “An overview of chatbot technology,” in *Artificial Intelligence Applications and*, I. Maglogiannis, L. Iliadis, and E. Pimenidis, Eds. Cham: Springer International Publishing, Innovations 2020, pp. 373–383, iD: 10.1007/978-3-030-49186-4_31.
- [3] K. Hao, “Aprendizaje federado: la nueva arma de ia para asegurar la privacidad,” -01-01T01:33:00+01:00 1970. [Online]. Available: <http://www.technologyreview.es/s/11017/aprendizaje-federado-la-nueva-arma-de-ia-para-asegurar-la-privacidad>
- [4] H. V. Cevallos and J. D. Cuesta, “Metodologías ágiles frente a las tradicionales en el proceso de desarrollo de software,” *Espirales Revista Multidisciplinaria de Investigación*, pp. 115–120, June 1, 2018.
- [5] “Pandorabots: Home.” [Online]. Available: <https://developer.pandorabots.com/home.html>
- [6] “Ibm watson,” September 29, 2021. [Online]. Available: <https://www.ibm.com/es-es/watson>
- [7] J. Best, “Ibm watson: The inside story of how the jeopardy-winning supercomputer was born, and what it wants to do next,” September 9, 2013. [Online]. Available: <https://acortar.link/rqaLYJ>
- [8] “Ibm watson assistant - agente virtual.” [Online]. Available: <https://www.ibm.com/es-es/products/watson-assistant>
- [9] V. Iles, “El total economic impact™ de ibm watson assistant,” Tech. Rep., Mar 1, 2020. [Online]. Available: <https://drive.google.com/file/d/1FLqzmWyfxHDLPC0-Xj9qLs9HyMHs1krI/view?usp=sharing>

- [10] “Watson assistant improves intent detection accuracy, leads against ai vendors cited in published study,” December 10, 2020. [Online]. Available: <https://acortar.link/UNFjRl>
- [11] “Productos ibm watson,” September 29, 2021. [Online]. Available: <https://www.ibm.com/es-es/watson/products-services>
- [12] “Domino’s-case-study.” [Online]. Available: <https://drive.google.com/file/d/15a5mh-6CuJWoHzvhvwT3xjIoxy05iNNA/view?usp=sharing>
- [13] “Rasa open source conversational ai,” December 1, 2020. [Online]. Available: <https://rasa.com/>
- [14] “Openai. pricing.” [Online]. Available: <https://openai.com/api/pricing/>
- [15] “Modelos de gpt-3.” [Online]. Available: <https://beta.openai.com/docs/engines/gpt-3>
- [16] “Openai. example applications.” [Online]. Available: <https://beta.openai.com/examples/>
- [17] “Openai. precios de los modelos de gpt-3.” [Online]. Available: <https://beta.openai.com/docs/guides/fine-tuning/pricing>
- [18] S. Roller, E. Dinan, N. Goyal, D. Ju, M. Williamson, Y. Liu, J. Xu, M. Ott, K. Shuster, E. M. Smith, Y.-L. Boureau, and J. Weston, “Recipes for building an open-domain chatbot,” 2020. [Online]. Available: <https://arxiv.org/abs/2004.13637>
- [19] “¿qué es una api rest?” -04-06 2021. [Online]. Available: <https://www.ibm.com/mx-es/cloud/learn/rest-apis>
- [20] S. G. Brett, “Responsive web design: Enriching the user experience,” *Sigma. Inside the digital ecosystem*, vol. 11, no. 1, pp. 13–19, 2011. [Online]. Available: <https://www.webdesignblog.gr/wp-content/uploads/2012/03/5.pdf>
- [21] “Aiml foundation.” [Online]. Available: <http://www.aiml.foundation/>
- [22] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret, “The world-wide web,” *Communication of the ACM*, vol. 37, no. 8, 1994. [Online]. Available: <https://dl.acm.org/doi/pdf/10.1145/179606.179671>
- [23] L. Martin, A. Fan, E. de la Clergerie, A. Bordes, and B. . it Sagot, “Muss: Multilingual unsupervised sentence simplification by mining paraphrases,” 2021, journal: arXiv preprint arXiv:2005.00352.
- [24] J. Zhang, Y. Zhao, M. Saleh, and P. J. Liu, “Pegasus: Pre-training with extracted gap-sentences for abstractive summarization,” 2019, 1912.08777.

- [25] Q. Lhoest, A. V. del Moral, Y. Jernite, A. Thakur, P. von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall, J. Davison, M. vSa vsko, G. Chhablani, B. Malik, S. Brandeis, T. L. Scao, V. Sanh, C. Xu, N. Patry, A. McMillan-Major, P. Schmid, S. Gugger, C. 'ement Delangue, T. 'eo Matussi 'ere, L. Debut, S. Bekman, P. Cistac, T. Goehringer, V. Mustar, F. ccois Lagunas, A. Rush, and T. Wolf, "Datasets: A community library for natural language processing," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, nov 2021, pp. 175–184, 2109.02846. [Online]. Available: <https://aclanthology.org/2021.emnlp-demo.21>
- [26] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, oct 2020, pp. 38–45. [Online]. Available: <https://www.aclweb.org/anthology/2020.emnlp-demos.6>

