

UNIVERSIDAD DE SANTIAGO DE COMPOSTELA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

AIWantTheJob

Autor:

Jorge Alcalde Vesteiro

Tutores:

María Jesús Taboada Iglesias

Walter Vinci

Grado en Ingeniería Informática

Julio 2024

Trabajo de Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de la
Universidad de Santiago de Compostela para la obtención del grado en Ingeniería
Informática



Dña. María Jesús Taboada Iglesias, Profesora del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela, y **D. Walter Vinci**, profesional experto en IA asociado a HP SCDS.

INFORMAN:

Que la presente memoria titulada *AIWantTheJob*, presentada por **D. Jorge Alcalde Vesteiro** para superar los créditos correspondientes al Trabajo de Fin de Grado de la Titulación del Grado en Ingeniería Informática, se realizó bajo nuestra tutoría en el Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela.

Y para que así conste para los efectos oportunos, emiten el presente informe en Santiago de Compostela a 3 de Julio de 2024.

Tutora

Cotutor

Alumno

María Jesús Taboada
Iglesias

Walter Vinci

Jorge Alcalde Vesteiro

Agradecimientos

Me gustaría agradecer tanto a María Jesús como a Walter la ayuda que me han brindado durante todo el trabajo, por haberme ayudado y aguantado durante la realización de este trabajo. Gracias por estar pendiente del correo para resolver mis despistes.

Agradecer también a mis padres, que fueron los que me ayudaron a sobrellevar las últimas semanas de estrés extremo con este trabajo, especialmente a mi padre, que estaba sufriendo aún más que yo.

Resumen

En los últimos diez años, la popularidad, la capacidad y la inversión en técnicas de Inteligencia Artificial basadas en aprendizaje profundo han aumentado considerablemente. Los grandes modelos de lenguaje (LLM, *Large Language Model*) son actualmente las herramientas más empleadas para generación de texto en un gran abanico de tareas. El tamaño de estos modelos y del corpus con el que se entrena los han convertido en grandes bases de conocimiento. Una de las líneas de trabajo más prometedoras se centra en el uso de estos modelos como herramienta de ayuda al estudio. Un ejemplo de esta tendencia podría ser Google workspaces for education.

Las entrevistas de trabajo son una parte integral de nuestras vidas y sorprendentemente no hay muchas herramientas interactivas para ayudar en su preparación. Este trabajo de fin de grado titulado ***AIWantTheJob*** surge para explorar el uso de LLMs como herramientas de ayuda al aprendizaje con el fin de cubrir esta carencia detectada.

AIWantTheJob tiene como objetivo desarrollar un simulador de entrevistas de trabajo para preparar candidatos para un puesto específico. Utilizando técnicas de *Deep Learning* y NLP, el sistema genera escenarios de entrevistas realistas y personalizadas que brindan evaluaciones en tiempo real para mejorar las habilidades de los entrevistados. Para mejorar los resultados del LLM encargado este trabajo plantea un RAG (*Retrieval-Augmented Generation*) que genera y proporciona acceso a la base de conocimiento específica del puesto de trabajo para el que se prepara el usuario. A mayores, se emplea una herramienta paralela llamada Nemo Guardrails, que sirve para, entre otras muchas cosas, definir flujos de conversaciones entre la persona y el LLM. El sistema se ha implementado como una app que se puede desplegar en entornos virtualizados de manera rápida y que requiere recursos informáticos de bajo coste (teniendo en cuenta los requisitos estándar de aplicaciones basadas en IA). Además, la aplicación está dotada de técnicas de despliegue realista para poder simular un entorno de producción real. Más allá del caso de negocio del entrevistador, el sistema creado permite que cualquier desarrollador pueda implementar *chatbots* de otros tópicos con acciones específicas realizando una inversión temporal mínima y una gran facilidad.

El trabajo de fin de grado proporciona como resultado un *chatbot* que realiza adecuadamente las tres tareas propuestas (preguntar, evaluar y simular respuestas) adhiriéndose bien al contexto y alucinando relativamente poco. Además, podemos concluir que los resultados se pueden mejorar empleando un modelo más inteligente, que requeriría de más recursos computacionales que los disponibles durante el proyecto.

Índice

Capítulo 1: Introducción	1
Objetivos	2
Alcance del proyecto	3
Estructura de la aplicación	4
Limitaciones y Supuestos.....	5
Criterios de aceptación.....	5
Plan de proyecto	6
Metodología.....	6
Gestión de la configuración, del código y de la documentación	6
Gestión del tiempo.....	6
Estructura de la memoria	6
Estructura de los entregables	7
Capítulo 2: Requisitos y Riesgos.....	8
Actores.....	8
Requisitos funcionales.....	8
Módulo de web - app y base de datos	9
Módulo de la API.....	10
Módulo de IA	11
Módulo del sistema de RAG	12
Módulo de despliegue	13
Requisitos de información.....	14
Requisitos no funcionales	15
Requisitos de Sistema - Entorno de desarrollo	15
Trazabilidad	15
Plan de riesgos.....	16
Identificación y Clasificación de riesgos	16
Análisis estratégico frente a los riesgos.....	18
Historial de riesgos	19
Capítulo 3: Diseño	20
Diseño de componentes del sistema	20
Frontend	21
Backend.....	21
Bases de datos y ficheros externos.....	22
Diagrama de clases.....	22
Explicación detallada de los componentes más importantes.....	23
Frontend	23
Utils	25
API.....	25
Proxy y Caché.....	26
LLM.....	26
Gestor de comportamiento (Nemo).....	27
RAG y VectorStore	28
BD	29
Configuraciones	29

Implementación de los casos de uso	30
CU – 001	30
CU – 002	30
CU – 003	31
CU – 004	31
CU – 005	32
CU – 006	32
CU – 007	33
CU – 007.1.....	33
CU – 007.2.....	34
CU – 007.3.....	35
CU – 008	35
CU – 009	36
CU – 010	36
CU – 011	36
CU – 012	36
CU – 013	36
CU – 014	36
CU – 015	36
CU – 016	37
Gestión de errores	37
Tecnologías	37
Backend.....	37
Frontend	38
Bases de datos	38
RAG	38
HuggingFace	38
Langchain	39
Nemo Guardrails	39
Deployment.....	40
Otro caso de negocio.....	40
Capítulo 4: Pruebas	42
Pruebas Unitarias	42
Pruebas de Integración.....	43
Pruebas de extremo a extremo	44
Pruebas de usabilidad.....	44
Pruebas de despliegue	45
Aceptación	46
Capítulo 5: Conclusión y Ampliaciones	47
Conclusiones	47
Ampliaciones.....	47
Casos de negocio complejos	47
Más generalización.....	48
Mejorar el Frontend	48
Arquitectura de microservicios y paso de señales.....	48
Otras posibles mejoras	48
Anexo A: Guías	49

Guía del Usuario	49
Guía del Desarrollador	63
Bibliografía	79
Anexo B.....	82
Plan de proyecto	82
Plan de Riesgos.....	88
Matriz de Trazabilidad de requisitos Completa	92
Plan de Pruebas	93
Gestión Cronograma	132

Índice de figuras

Figura 1: BenchMarks de la familia llama de modelos open-source de llama.....	2
Figura 2: Diagrama de componentes resumido	4
Figura 3: Diagrama de comportamiento RAG y Nemo resumido	4
Figura 4: Casos de uso módulo GUI	9
Figura 5: Casos de Uso Módulo de la API	10
Figura 6: Casos de uso Módulo de IA	11
Figura 7: Casos de Uso Módulo RAG	12
Figura 8: Casos de Uso Módulo Despliegue	13
Figura 9: Diagrama de componentes	20
Figura 10: Diagrama de comportamiento	21
Figura 11: Diagrama de clases	22
Figura 12: Login	23
Figura 13: Creación Cuenta.....	23
Figura 14: página de tutorial	24
Figura 15: botones de tutorial y logout	24
Figura 16: Entrevistador y las tres opciones de chat	24
Figura 17: RAG cliente	25
Figura 18: Gráfico de puntuaciones	25
Figura 19: cuarta acción (Resumir)	28
Figura 20: fórmula del cálculo de semejanza por distancia de coseno	29
Figura 21: Diagrama Relacional	29
Figura 22: Diagrama de Secuencia del Login	30
Figura 23: Diagrama de Secuencia del Registro	31
Figura 24: Actualizar la base de conocimiento	32
Figura 25: Mandar mensaje al chatbot por defecto	33
Figura 26: Petición de una pregunta.....	34
Figura 27: Evaluación de una respuesta.....	34
Figura 28: Petición de ejemplo de respuesta	35
Figura 29: Ejemplo de gestión errores	37
Figura 30: Entorno virtualizado	40
Figura 31: Ejecución de los tests 1	43
Figura 32: Ejecución de los tests 2.....	43
Figura 33: Ejemplo de preguntas del formulario	44
Figura 34: Ejemplo de respuesta de la encuesta de usabilidad	45
Figura 35: Imagen de coste temporal frente a la calidad apreciada	45
Figura 36: Script Conda funcionando.....	45

Índice de tablas

Tabla 1: Objetivos desglosados	3
Tabla 2: Limitaciones	5
Tabla 3: Supuestos	5
Tabla 4: Criterios de aceptación	6
Tabla 5: Contenido del proyecto	7
Tabla 6: A-1 Usuario	8
Tabla 7: A-2 Administrador / Desarrollador.....	8
Tabla 8: RF-1 Módulo de GUI	9
Tabla 9: Casos de uso del módulo de GUI	10
Tabla 10: RF-2 Módulo de la API.....	10
Tabla 11: Casos de uso del Módulo de la API.....	11
Tabla 12: RF-3 Módulo de IA.....	11
Tabla 13: Casos de uso del módulo de IA.....	12
Tabla 14: RF-4 Módulo de RAG	12
Tabla 15: Casos de uso del módulo de RAG	13
Tabla 16: RF-5 Módulo de despliegue	13
Tabla 17: Casos de uso del módulo de despliegue	13
Tabla 18: RI-1 Información de usuarios	14
Tabla 19: RI-N Tabla de evaluación por usuario	14
Tabla 20: RI-2 General Context	14
Tabla 21: RI-3 Client Context	14
Tabla 22: RI-4 Similarity context.....	14
Tabla 23: RRI-1 Tablas Volátiles	15
Tabla 24: RnF-1 tiempo de respuesta	15
Tabla 25. Matriz de trazabilidad (versión reducida)	15
Tabla 26: Riesgos detectados	16
Tabla 27: Acciones frente a riesgos.....	18
Tabla 28: Logs de riesgos.....	19
Tabla 29: Ejemplo de tabla para los tests unitarios	43

Índice de códigos de programación

Ejemplo de código 1: Templates usadas para las acciones del entrevistador.....	27
Ejemplo de código 2: Configuraciones disponibles	28
Ejemplo de código 3: Ejemplo de Langchain y HuggingFace	39
Ejemplo de código 4: Ejemplo Flujo	39
Ejemplo de código 5: Flujo de conversación en otro caso de negocio.....	41
Ejemplo de código 6: Funciones de testing	43

Capítulo 1: Introducción

La realización de este proyecto es un acuerdo formal entre la Universidad de Santiago de Compostela¹ y HP SCDS² en la forma de un trabajo de fin de grado. Este es un proyecto de desarrollo de tipo B por lo que esta memoria incluye información del diseño y del proceso de desarrollo formal.

El proyecto titulado ***AIWantTheJob*** presenta una solución sobre cómo prepararse para preguntas de ciertos tópicos de entrevista dentro del marco empresarial, teniendo acceso a preguntas, evaluaciones y respuestas en tiempo real. En el marco tecnológico, ***AIWantTheJob*** es el caso de negocio del sistema que está detrás, el cual una plataforma con la que crear *chatbots* centrados en un tópico de manera rápida y sencilla que, si bien no era la intención del proyecto, sí fue un resultado que se ha obtenido del mismo. En este proyecto se integran los asistentes virtuales con una interfaz de usuario fácil de usar para todo tipo de usuarios. Los dos pilares que soportan tanto el sistema general como, sobre todo, nuestro caso de negocio (el entrevistador) son las siguientes técnicas: por un lado, se emplea RAG (Retrieval-Augmented Generation)³ para aportar al *bot* la parte de conocimiento arbitrario pero factual para los tópicos de las entrevistas; por otro, se emplea un gestor de conversaciones para que la conversación con el asistente fluya según lo deseado.

La popularidad de la IA estos últimos años ha hecho que salgan decenas de *framework* y librerías para trabajar. En nuestro caso, para el *frontend* se plantea el uso de Streamlit, siendo esta una manera sencilla de desarrollar *web-apps* centradas en la ciencia de datos. Para el *backend*, el pilar de nuestro trabajo es el modelo de lenguaje que usemos. Como carecemos de presupuesto monetario, la opción principal es el empleo de modelos de lenguaje *open-source*. Estos son normalmente más pequeños, pero mucho mejores de lo que cabría pensar.⁴ La idea del proyecto es emplear el LLM de Meta Llama⁵ como modelo, pero se puede cambiar sencillamente debido a como está diseñado el proyecto.

El diseño del proyecto está pensado para ser altamente modular y adaptable (posiblemente) a una arquitectura de servicios. De los módulos presentados, los más importantes son el módulo de RAG y el gestor de conversaciones y comportamientos.

El RAG (Retrieval-Augmented Generation) es la técnica que empleamos para mantener a nuestro entrevistador dentro del tópico de la entrevista. La técnica de RAG (Naive en este caso) es un proceso estandarizado a través del cual el usuario puede vectorizar conocimiento (usando documentos) para después añadirlo como contexto a preguntas y/o respuestas semánticamente parecidas. De esta manera, si en nuestra base de conocimiento el examinador tiene prácticas de ciencia de datos, las preguntas que se le van a realizar al entrevistado son de ciencia de datos. De esta manera aseguramos que el entrevistador sea de tópicos arbitrarios.

El gestor de conversaciones se construye con [Nemo Guardrails](#)⁶. A través de este módulo podemos controlar al *bot* de una manera muy dinámica. Por un lado, podemos hacer que la instrucción del modelo cambie según lo que tengamos que hacer (así controlamos cuando el entrevistador pregunta, cuando contesta y cuando evalúa). Por otro lado,

¹ [Inicio | Universidade de Santiago de Compostela \(usc.gal\)](#)

² [Observatorio HP 2022-2023 - HP SCDS](#)

³ <https://arxiv.org/abs/2005.11401>

⁴ <https://arxiv.org/abs/2311.11045>

⁵ <https://arxiv.org/abs/2307.09288>

⁶ <https://arxiv.org/abs/2310.10501>

empleando su manera dinámica de enfocar las cosas, se nos permite que el propio *bot* sea el que tome la decisión de que estrategia emplear.

A continuación se presentan los benchmarks de la familia de los modelos de llama que compiten con modelos propietarios de pago.

Category	Benchmark	Llama 3 8B	Llama2 7B	Llama2 13B	Llama 3 70B	Llama2 70B
General	MMLU (5-shot)	66.6	45.7	53.8	79.5	69.7
	AGIEval English (3-5 shot)	45.9	28.8	38.7	63.0	54.8
	CommonSenseQA (7-shot)	72.6	57.6	67.6	83.8	78.7
	Winogrande (5-shot)	76.1	73.3	75.4	83.1	81.8
	BIG-Bench Hard (3-shot, CoT)	61.1	38.1	47.0	81.3	65.7
	ARC-Challenge (25-shot)	78.6	53.7	67.6	93.0	85.3
Knowledge reasoning	TriviaQA-Wiki (5-shot)	78.5	72.1	79.6	89.7	87.5
Reading comprehension	SQuAD (1-shot)	76.4	72.2	72.1	85.6	82.6
	QuAC (1-shot, F1)	44.4	39.6	44.9	51.1	49.4
	BoolQ (0-shot)	75.7	65.5	66.9	79.0	73.1
	DROP (3-shot, F1)	58.4	37.9	49.8	79.7	70.2

Figura 1: BenchMarks de la familia llama de modelos open-source de llama⁷

Objetivos

Este proyecto tiene como objetivo desarrollar un simulador de entrevistas de trabajo general impulsado por inteligencia artificial para preparar candidatos para un puesto específico. Utilizando técnicas de Deep Learning y NLP, el sistema genera escenarios de entrevista realistas y personalizados que brindan retroalimentación instantánea para mejorar las habilidades de los entrevistados.

1. Diseñar un sistema de simulación de entrevistas de trabajo basado en un modelo generativo. La llamada al modelo empleará técnicas de RAG Retriever-Active Gathering con documentos relevantes a la tarea para mejorar las prestaciones ya existentes del LLM.
2. Desarrollar un sistema real (demo) de una combinación Aplicación Web + API para poder realizar pruebas de comportamiento y emular una versión prototípica del producto.
3. Desarrollar un sistema que evalúe las capacidades de los entrevistados y las valore e implementar un método de seguimiento del progreso de las actuaciones de los entrevistados.

Estos objetivos son lo que se van a evaluar y que se tienen que completar dentro de este proyecto. Por cuestiones de formalización, dentro del plan de proyecto de [anexo](#) estos objetivos se han subdividido de la siguiente manera (notar que simplemente son divisiones y que los objetivos a alcanzar son los mismos):

ID objetivo	Descripción
O - 001	Implementar entrevistador LLM de tipo Q/A
O - 002	Emplear RAG junto al modelo

⁷ [meta-llama/Meta-Llama-3-8B · Hugging Face](#)

O - 003	Implementar una aplicación completa
O - 004	Desarrollar un prototipo real
O - 005	Implementar un <i>deployment</i> rápido
O - 006	Simular un entorno de producción realista
O - 007	Implementar un sistema de evaluación
O - 008	Abstracción y customización

Tabla 1: Objetivos desglosados

Alcance del proyecto

El alcance de este proyecto es una aplicación *completa* donde podemos identificar por lo menos cuatro componentes diferentes: *web*, *api*, modelo de IA y una base de datos. A través de este sistema se implementará como caso de uso el *chatbot* como entrevistador y/o preparador de preguntas.

A nivel de negocio, la estructura sobre la cual está construido el sistema es la parte más interesante. Si bien para el caso de uso en concreto se han añadido elementos extras que no siempre son necesarios, en el caso de querer crear *chatbots* personalizados de temas arbitrarios, el sistema propuesto facilita al desarrollador lograr esto de una manera fácil e intuitiva.

A nivel de TFG, la solución presentada es un entrevistador. Para emular una situación de entrevista empleamos un gestor de comportamientos que establece el comportamiento del entrevistador al *bot* (preguntar, evaluar, responder, ...). Para emular el conocimiento particular que se pide, el entrevistador tiene acceso a bases de conocimiento particulares y arbitrarias a través del sistema RAG Naive ⁸asociado. Esto significa que para que el bot se adhiera a tópicos de entrevista permitimos que tenga acceso a documentos de los cuales extraer la información.

A nivel de usuario, el sistema presentado (acotado al caso de uso de un entrevistador) permite realizar las siguientes acciones:

1. Registrarse e iniciar sesión para tener acceso a tu historial de evaluaciones
2. Subir un documento propio con el que el chatbot puede tener conocimiento particular de lo que el usuario deseé
3. Generar preguntas de entrevista de temas concretos
4. Generar respuestas de ejemplo a las preguntas que te hace el evaluador
5. Recibir una evaluación de la respuesta que el usuario ha dado en base al contexto
6. Ver el historial de evaluaciones de las respuestas

Como desarrollador, la customización de los *chatbots* permite realizar los siguientes cambios:

1. Cambiar la base de conocimiento del *bot* (en el lado del servidor como en el del cliente)
2. Modificar el modelo que ejecuta el sistema en el *backend* (teniendo a disposición cualquier modelo que tenga formato de HuggingFace)
3. Modificar el flujo de conversación (explicado más en detalle en [Nemo Guardrails](#)) de tal manera que se puedan asignar diferentes tipos de “personalidades”⁹ (a modo de

⁸ <https://arxiv.org/abs/2312.10997>

⁹ <https://arxiv.org/abs/2401.10065>

sysprompts) o “tareas” (a modo de *callbacks* o *acciones*) al *bot* en caso de que detecte comportamientos particulares.

Estructura de la aplicación

La idea es separar el papel del usuario (con acceso preferiblemente solo la *url* de la *web app*) del papel del administrador (que puede controlar el *deployment* de la aplicación y que tiene acceso a todo).

Este despliegue modular (estructura de servicios) es muy estándar en el mercado porque permite separar los componentes entre sí, teniendo un acoplamiento relativamente bajo y, como se verá en un [futuro](#), se podría extender a una estructura de microservicios completa.

Lo primero que definimos es como sería un diagrama de componentes del sistema.

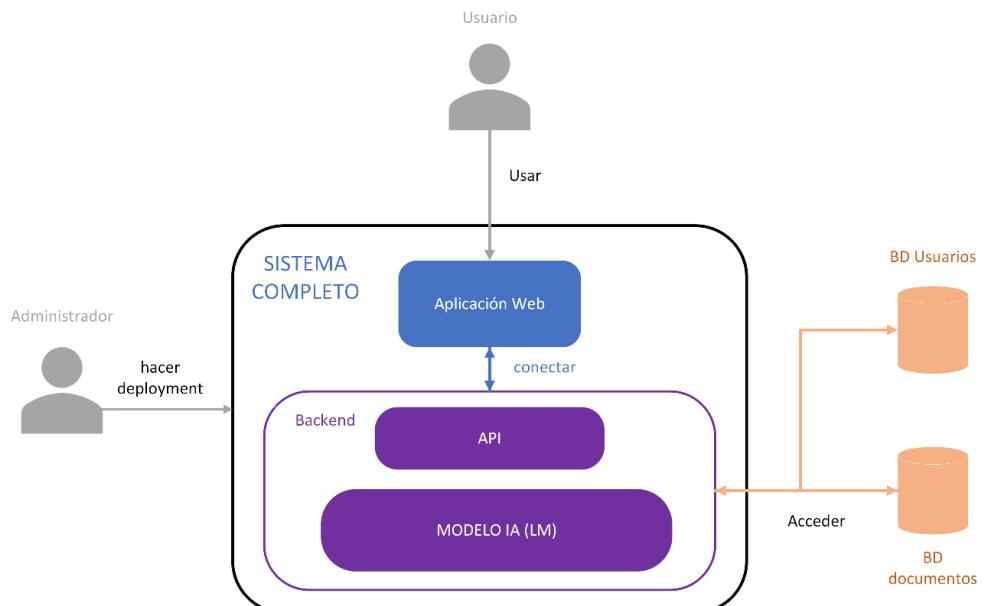


Figura 2: Diagrama de componentes resumido

Mientras que la figura previa muestra la estructuración de los componentes, las interacciones base definidas (IA – LLM – Gestor de comportamientos) previamente funcionaría de la siguiente manera:

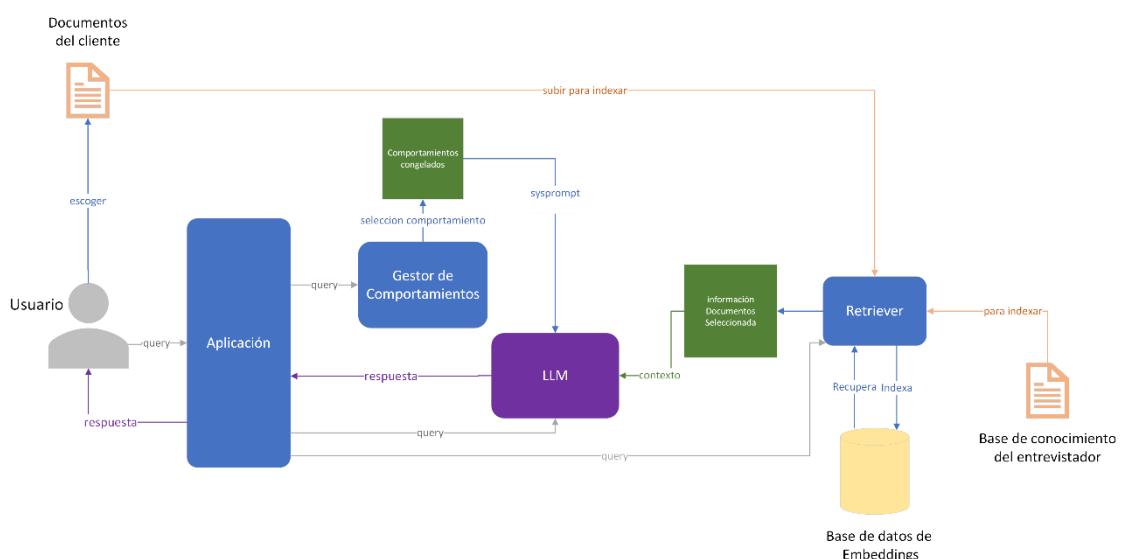


Figura 3: Diagrama de comportamiento RAG y Nemo resumido

Podemos observar el comportamiento de los puntos clave del RAG y del Gestor de comportamientos del LLM (Nemo). El LLM dicta su actuación (modo del examinador) en base a lo que dictamine el gestor de comportamientos en base a la *query*. Por otro lado, los documentos de conocimiento se han guardado como *embeddings* y, a través de la *query*, dictaminamos con el RAG la información (contexto) del LLM que extraemos de esos documentos. El LLM, con su comportamiento definido y con su documento, realiza entonces su tarea de responder de la manera indicada.

Limitaciones y Supuestos

Para acompañar al alcance se presentan a continuación las limitaciones y los supuestos.

La limitación principal del proyecto es tanto el número de horas a dedicar (401 horas) como la fecha de entrega el 3 de julio. Además, en este caso, otra gran limitación son los recursos informáticos limitados, ya que los modelos de lenguaje son muy costosos computacionalmente.

Por otro lado, los supuestos del proyecto son que las personas que vayan a emplear cumplen con las necesidades de hardware y de software para la ejecución principalmente.

Ambos apartados están formalizados en el plan de proyecto asociado, un resumen de la formalización sería el siguiente:

ID limitación	Limitación
L - 001	Fecha de entrega del proyecto
L - 002	Uso de librerías solo de investigación
L - 003	Límite de tamaño de los modelos para probar
L - 004	Límite de la velocidad de los modelos

Tabla 2: Limitaciones

ID Supuesto	Descripción
S - 001	La persona que hace el <i>deployment</i> lo hace en linux
S - 002	La persona que hace el <i>deployment</i> usa equipamiento que cumple con los requisitos de sistema mínimos

Tabla 3: Supuestos

Criterios de aceptación

El criterio principal de aceptación de este proyecto es que se cumplan los objetivos definidos en el apartado de [objetivos](#). A mayores, en el plan de proyecto se formalizan los siguientes criterios de aceptación (resumidos):

ID criterio	Descripción
A - 001	Porcentaje de tests pasados
A - 002	Evaluación empírica del sistema
A - 003	Aprobación de los responsables

A - 004	Requisitos mínimos
---------	--------------------

Tabla 4: Criterios de aceptación

Plan de proyecto

Se adjunta en el apartado [Plan de Proyecto](#) el plan de proyecto completo.

Metodología

La metodología de desarrollo escogida fue una variante de la metodología *agile scrum*. En este caso, como no se tiene *scrummaster*, empleábamos las historias de usuario de *Scrum* como si fueran tareas en un tablero de *kanban*. Eso sí, las historias se completaron por *sprints*.

De esta manera, nos encontramos con una manera ágil de realizar incrementos a la aplicación a base de prototipos funcionales.

La otra variación frente a *scrum* es que se realizó una documentación un poco más extendida a lo que se esperaría de una metodología ágil (plan de riesgos, plan de pruebas, ...). Esto se debe por saber que se iba a tener que construir esta memoria con esas inclusiones desde el principio del proyecto, por lo que se construyó teniendo este entregable como hito.

Gestión de la configuración, del código y de la documentación

Para gestionar la configuración y el código se empleó la herramienta corporativa de la empresa asociada al TFG: HP SCDS. La herramienta es GitLab, que es un gestor de repositorios muy popular, junto con GitHub.

Para gestionar la configuración se empleó tanto Google Drive como OneDrive, el primero por comodidad y el segundo por ser la herramienta oficial de la Universidad.

El proyecto final se puede encontrar en mi GitHub: https://github.com/Jorge-A-V/TFG_AIWantTheJob.git.

Gestión del tiempo

Al emplearse metodologías ágiles, se empleó el *backlog* de *scrum* para manejar el tiempo. Cada *sprint* de dos semanas iba realizando tareas y se iban descontando del total previsto. Al construirse de manera incremental, la organización de los *sprints* era por características que se querían añadir al prototipo.

Estructura de la memoria

La estructura de la memoria, tal y como se indica en el índice, es de la siguiente manera:

1. Capítulo 1: En este apartado se realiza una breve introducción, un breve resumen de cómo es el estado del arte de las tecnologías empleadas, se describe una visión general del plan de proyecto y de algunos documentos de gestión agregados.
2. Capítulo 2: En este apartado se definen claramente los requisitos funcionales y casos de uso, requisitos no funcionales, de información y de sistema del proyecto. Además, en este apartado se hace un resumen del plan de riesgos realizado para el proyecto.
3. Capítulo 3: En este apartado se describe todo lo relacionado con el diseño del sistema. Esto incluye una visión general de las tecnologías del sistema y el porqué de estas metodologías. Se hacen descripciones claras del sistema desde alto nivel a nivel de componente y se presentan diagramas de secuencia de cómo se espera que funcione. Por

último, se añade un apartado en el que se describe el procedimiento para usar el sistema para un caso diferente al nuestro entrevistador.

4. Capítulo 4: En este apartado se hace resumen del plan de pruebas que va anexado al documento mostrando como se pasan las pruebas de unidad, de despliegue y de integración.
5. Capítulo 5: En este apartado se describen las conclusiones del sistema y, además, se describe una posible lista de ampliaciones para el mismo.
6. Otros: Además, se anexan las guías de uso, la bibliografía y alguno de los planes que no se pueden documentar completos en la memoria por tamaño.

Estructura de los entregables

Este documento es la memoria principal del TFG. Esto significa que es una condensación de todo lo realizado. Sin embargo, no es el único componente que se entrega junto con el proyecto. En la siguiente tabla se describen los entregables de este proyecto.

Tabla 5: Contenido del proyecto

Nombre	Herramienta	Contenido
TFG_Alcalde_Vesteiro_Jorge.pdf	Pdf reader	Memoria del proyecto. Overview de todo lo realizado.
https://github.com/Jorge-A-V/TFG_AIWantTheJob.git	Git	Repositorio (código fuente).

Los anexos de esta memoria son los siguientes:

- Plan de proyecto
- Overview del Cronograma
- Plan de riesgos
- Logs de riesgos
- Plan de pruebas
- Manual de usuario
- Manual de desarrollo

Capítulo 2: Requisitos y Riesgos

En este apartado se describen los requisitos que se tuvieron en cuenta a la hora de desarrollar el sistema. Notar que, al seguirse una metodología ágil e incremental, los requisitos presentados aquí son la versión final deseada y que durante el transcurso del proyecto se fue construyendo hacia a ellos en base a prototipos / incrementos.

Los requisitos fueron escogidos en base al conocimiento de los participantes (tutor y yo), basándonos en lo que una aplicación de este calibre requiere siguiendo ejemplos reales y simplificándolos a un alcance adecuado.

Además, se presenta una visión general de los riesgos identificados, tratados y sufridos a través de un plan de riesgos resumido.

Actores

Lo primero que se hace dentro del sistema diseñado es dictaminar quiénes son los actores implicados y que permisos y papeles van a tener. En este caso, nos encontramos con dos actores diferentes, pero no mutuamente excluyentes entre sí (es decir una persona puede ser ambos al mismo tiempo).

A - 001	Usuario
Descripción	Se llama usuario a la persona que interactúa con el sistema. Se supone que simplemente se beneficia de lo ofrecido desde la <i>web app</i> y que no va a tocar nada en el sistema.
Permisos	<ul style="list-style-type: none"> • No puede modificar el sistema. • Puede subir documentos en caliente para modificar la base de conocimiento del <i>bot</i>.

Tabla 6: A-1 Usuario

A - 002	Administrador/Desarrollador
Descripción	Caracterizamos al administrador como la persona que realiza el <i>despliegue</i> de la aplicación; por ende, puede tocar el sistema antes de implementarlo.
Permisos	<ul style="list-style-type: none"> • Tiene permisos para modificar cualquier parte del sistema (aunque sólo se recomienda tocar algunas). • Determina la base de conocimiento principal del <i>bot</i> (si así lo considera necesario)

Tabla 7: A-2 Administrador / Desarrollador

Requisitos funcionales

Los requisitos funcionales se describen de la siguiente manera: “Quiero que el sistema haga [requisito]”. Sin embargo, podemos también identificarlos como historias de usuario como las que se emplean en scrum: “Como [persona] quiero [realizar acción] para/por [conseguir objetivo]”.

En este caso, definimos los requisitos funcionales por agrupaciones modulares que se encuentran en el repositorio del entregable final.

Módulo de web - app y base de datos

RF - 001	El sistema permite al usuario tener una “cuenta” dentro del mismo
Descripción	Para poder guardar los históricos de puntuaciones y asociarlos a usuarios se guardan en la base de datos los datos de la cuenta. Los usuarios pueden usar la aplicación web para iniciar sesión, registrarse y recuperar sus historiales
Historia	Como usuario quiero validarme dentro de la aplicación para poder registrarme, iniciar sesión y acceder a mis historiales
Actores	A - 001 Usuario
Casos de Uso	<ul style="list-style-type: none"> ● CU - 001: Realización del <i>login</i> en la aplicación ● CU - 002: Realización del registro de usuario en la aplicación ● CU - 003: Recuperación del historial de puntuaciones ● CU - 004: Inserción de un nuevo dato en el historial

Tabla 8: RF-1 Módulo de GUI

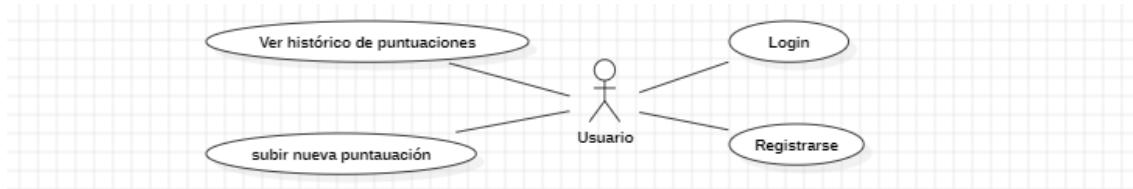


Figura 4: Casos de uso módulo GUI

CU - 001	Realización del <i>login</i> en la aplicación
Historia	Como usuario quiero iniciar sesión en la aplicación para acceder a mi cuenta
Descripción	De manera consistente con los estándares de aplicación, un usuario registrado es capaz de acceder a su cuenta a través de un <i>login</i>
CU - 002	Realización del registrado de un usuario en la aplicación
Historia	Como usuario quiero ser capaz de registrarme en la aplicación para crear una cuenta
Descripción	De manera consistente con los estándares de aplicación un usuario registrado es capaz de acceder a su cuenta a través de un <i>login</i>
CU - 003	Recuperación del historial de puntuaciones
Historia	Como usuario quiero poder ver mi historial de progreso
Descripción	Se mantienen los datos de progreso. El usuario es capaz de verlos y recuperar su progreso actual en cualquier momento.
CU - 004	Inserción de un nuevo dato en el historial de puntuaciones

Historia	Como usuario quiero ser capaz de actualizar mi historial de puntuaciones para poder ver mi progreso
Descripción	Se insertan datos nuevos según los resultados de uso de la aplicación en el historial.

Tabla 9: Casos de uso del módulo de GUI

Módulo de la API

RF - 002	El sistema permite al usuario realizar operaciones a través de la API
Descripción	Se habilitan ciertos <i>endpoints</i> de la API para que el usuario pueda realizar las operaciones descritas en los CU.
Historia	Como usuario, quiero ser capaz de subir un documento, chequear la salud del servidor y hacer consultas con el <i>chatbot</i>
Actores	A - 001 Usuario
Casos de Uso	<ul style="list-style-type: none"> ● CU - 005: Subir un documento como base de conocimiento ● CU - 006: Chequeo de Salud ● CU -007: Consulta a través del <i>chatbot</i>

Tabla 10: RF-2 Módulo de la API



Figura 5: Casos de Uso Módulo de la API

CU - 005	Subir un documento como base de conocimiento
Historia	Como usuario quiero ser capaz de actualizar la base de conocimiento subiendo un documento.
Descripción	El usuario es capaz de subir sus propios documentos en formato pdf para que el <i>bot</i> los use para basar su operatividad en ellos
CU - 006	Chequeo de salud
Historia	Como usuario quiero ser capaz de chequear la salud de la API para ver si me puedo conectar
Descripción	La API expone un <i>endpoint/heath</i> para chequear la salud
CU - 007	Consulta a través de <i>chatbot</i>
Historia	Como usuario quiero ser capaz de usar el <i>chatbot</i> ofrecido para sacarle el partido a la aplicación.

Descripción	El sistema ofrece un <i>chatbot</i> a través del cual el usuario puede hacer consultas a un entrevistador (caso de negocio).
CU - 007.1	Extensión del caso de uso: El usuario puede pedir al <i>bot</i> una pregunta en base a un tópico
CU - 007.2	Extensión del caso de uso: El usuario puede pedir al <i>bot</i> una evaluación a una respuesta que el <i>bot</i> le hizo al usuario
CU - 007.3	Extensión del caso de uso: El usuario puede pedir al <i>bot</i> una simulación de respuesta

Tabla 11: Casos de uso del Módulo de la API

Módulo de IA

RF - 003	El sistema permite el uso y la customización del modelo en <i>backend</i>
Descripción	Se ofrece al desarrollador un <i>backend</i> de IA muy versátil que es fácilmente personalizable
Historia	Como desarrollador y usuario, quiero ser capaz de customizar el modelo de IA que se usa en el <i>backend</i> para que se acople a mi caso de negocio
Actores	A - 001 Usuario A - 002 Administrador
Casos de Uso	<ul style="list-style-type: none"> ● CU - 008: Escoger el modelo de Lenguaje ● CU - 009: Escoger el flujo que va a emplear el modelo ● CU - 010: Escoger el <i>sys prompting</i> y los <i>templates</i> que quiero que use el modelo

Tabla 12: RF-3 Módulo de IA

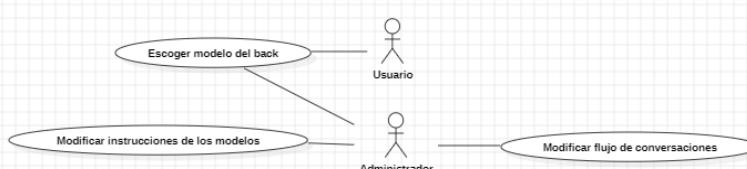


Figura 6: Casos de uso Módulo de IA

CU - 008	Escoger el modelo de lenguaje
Historia	Como usuario y/o administrador quiero escoger el modelo de lenguaje que se usa en el sistema para darle flexibilidad.
Descripción	Se ofrece una manera genérica de instanciar modelos de lenguaje de tal manera que el sistema funcione con cualquier IA con formato de HuggingFace.
CU - 009	Escoger el flujo que va a emplear el modelo
Historia	Como administrador quiero modificar el flujo de conversación del <i>bot</i> para que se acople a mi caso de negocio

Descripción	Se ofrece un <i>template</i> de COLANG para poder modificar el comportamiento predeterminado del <i>bot</i> .
CU - 010	Escoger el <i>sys prompting</i> y los <i>templates</i> del modelo
Historia	Como administrador quiero modificar las instrucciones de los <i>chatbots</i> para que tengan otro comportamiento.
Descripción	Se ofrecen <i>Prompt Templates</i> modificables para usar como <i>sysprompts</i>

Tabla 13: Casos de uso del módulo de IA

Módulo del sistema de RAG

RF - 004	El sistema permite el uso de <i>Retrieval-Augmented Generation</i>
Descripción	Se habilita el uso de una base de datos vectorial para que el <i>chabot</i> tenga acceso a varios contextos a la hora de realizar las preguntas
Historia	Como desarrollador, quiero poder emplear varios contextos para que el modelo use en las respuestas
Actores	A - 002 Administrador
Casos de Uso	<ul style="list-style-type: none"> ● CU - 011: Establecer el contexto general del lado del servidor ● CU - 012: Habilitar/Deshabilitar el contexto por similitud ● CU - 013: Cargar/Descargar contextos en caliente ● CU - 014: Guardar contextos similares para reutilización

Tabla 14: RF-4 Módulo de RAG

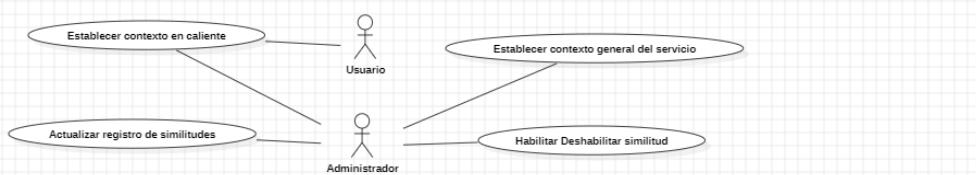


Figura 7: Casos de Uso Módulo RAG

CU - 011	Establecer el contexto general del lado del servidor
Historia	Como administrador quiero escoger el contexto general del sistema para que se acople a mi caso de negocio.
Descripción	Se habilita la posibilidad de escoger la base de conocimiento principal.
CU - 012	Habilitar/Deshabilitar el contexto por similitud
Historia	Como administrador quiero administrar el <i>bias</i> que introduce la base de datos de ejemplos.
Descripción	El <i>similarity context</i> , al ofrecer ejemplos, introduce <i>bias</i> en el sistema y, por ende, se puede habilitar y/o deshabilitar.
CU - 013	Cargar y Descargar el contexto del cliente en caliente

Historia	Como administrador quiero que el cliente pueda subir y usar su propio contexto.
Descripción	Se habilita la posibilidad de cargar documentos en caliente.
CU - 014	Guardar contextos similares para reutilización
Historia	Como administrador quiero que se guarde cada vez que el bot genera una pregunta para un contexto para poder usarla de ejemplo.
Descripción	Se habilita una tabla de <i>similarity</i> donde se guardan los ejemplos pasados.

Tabla 15: Casos de uso del módulo de RAG

Módulo de despliegue

RF - 005	El sistema permite un despliegue sencillo
Descripción	Se habilitan varios métodos de despliegue para el desarrollador
Historia	Como desarrollador, quiero poder desplegar la versión del producto de manera rápida y fácil
Actores	A - 002 Administrador
Casos de Uso	<ul style="list-style-type: none"> ● CU - 015: Despliegue en entorno virtual ● CU - 016: Despliegue a fuerza bruta

Tabla 16: RF-5 Módulo de despliegue

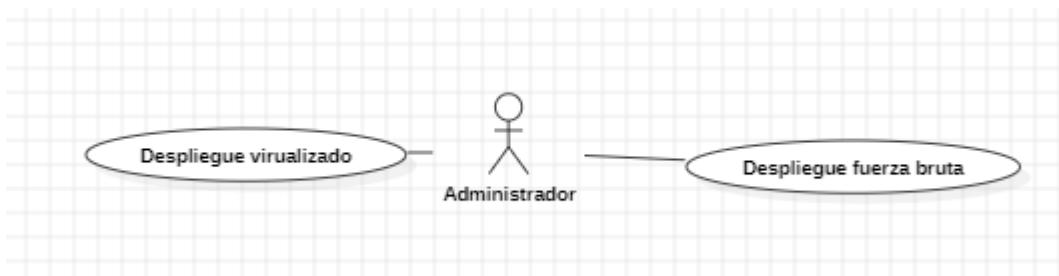


Figura 8: Casos de Uso Módulo Despliegue

CU - 015	Despliegue en entorno virtual
Historia	Como administrador quiero poder despegar el sistema en mi máquina en un entorno virtualizado por sencillez.
Descripción	Se habilitan scripts para virtualizar el entorno en la máquina del administrador.
CU - 016	Despliegue a fuerza bruta
Historia	Como administrador quiero poder desplegar el sistema de manera manual para poder hacer pruebas.
Descripción	Se facilita esta tarea con la existencia de algunos scripts básicos pero que completan tareas como la descarga de requisitos.

Tabla 17: Casos de uso del módulo de despliegue

Requisitos de información

Dentro del sistema, tenemos dos bases de datos a las cuales el sistema va a acceder. Para la base de datos de usuarios (tablas de la base de datos), los requisitos de información son los siguientes:

Tabla 18: RI-1 Información de usuarios

RI - 001	Información sobre los usuarios
Descripción	Almacena la información correspondiente a los usuarios que están registrados dentro del sistema
Campos	<ul style="list-style-type: none"> ● username: nombre de usuario en la tabla ● password: contraseña del usuario en la tabla ● Identifier: identificador de la tabla de evaluación asociada ● Salt: sal para la contraseña

Tabla 19: RI-N Tabla de evaluación por usuario

RI - 00N	Tabla de evaluación para el usuario N
Descripción	Almacena la información de las evaluaciones del usuario N
Campos	<ul style="list-style-type: none"> ● valor (N): valores de evaluación del usuario en la tabla

Para la base de datos vectorial (índices de las colecciones de la base de datos), necesitamos los siguientes requisitos de información:

Tabla 20: RI-2 General Context

RI - 002	General Context
Descripción	Almacena el contexto general que tiene el <i>chatbot</i> para una instancia en particular
Campos	<ul style="list-style-type: none"> ● id: id del documento ● Embeddings: contenido en formato vectorial para el LM ● Metadata: datos asociados al documento

Tabla 21: RI-3 Client Context

RI - 003	Client Context
Descripción	Almacena el contexto particular que el usuario quiere usar para una instancia en particular
Campos	<ul style="list-style-type: none"> ● id: id del documento ● Embeddings: contenido en formato vectorial para el LM ● Metadata: datos asociados al documento

Tabla 22: RI-4 Similarity context

RI - 004	Similarity Context
Descripción	Almacena asociaciones contexto-pregunta para que se pasen ejemplos de preguntas pasadas similares para contextos similares al <i>bot</i> como ejemplo

Campos	<ul style="list-style-type: none"> ● id: id del documento ● Embeddings: contenido en formato vectorial para el LM ● Metadata: datos asociados al documento
--------	---

Para esta base de datos, tenemos además las siguientes restricciones de información:

Tabla 23: RRI-1 Tablas Volátiles

RRI - 001	Tablas volátiles
Descripción	El contenido de estas tablas se espera que sea borrado entre ejecución y ejecución (al menos ese es el comportamiento esperado) para no empezar a acumular documentos (mezcla de contextos)
Se aplica a	<ul style="list-style-type: none"> ● RI - 002: <i>General Context</i> ● RI - 003: <i>Client Context</i>

Requisitos no funcionales

Tabla 24: RnF-1 tiempo de respuesta

RnF - 001	Tiempo de respuesta rápido
Descripción	El sistema tiene que ser rápido en ejecutarse. Si bien esto puede variar dependiendo del entorno y de la pregunta, se considera rápido en este contexto si tarda menos de 60 segundos en responder
Métrica	Se evalúa el tiempo desde que la respuesta llega al entorno (Nemo) hasta que se devuelve la respuesta (en segundos)

Requisitos de Sistema - Entorno de desarrollo

Las características del *hardware* donde se realizó tanto el desarrollo como las pruebas son las siguientes:

- Ubuntu 20:04 (yammy)
- 16 GB RAM
- 6 GB VRAM (RTX 3060 laptop)
- Intel i7-11370H @ 3.30GHz

De dichas características, el único requisito mínimo real son los 6GB de VRAM. Para el resto, aun con capacidades menores, el sistema debería funcionar. Tal y como se explicará en apartados posteriores, el punto de mayor calidad relativa sería teniendo 16GB de RAM y entre 12 y 16 GB de VRAM. De esta manera, el usuario tiene la capacidad de ejecutar modelos pequeños completos (repartidos entre ambas) o cuantizados a 8 bits en la GPU (siendo la cuantización a 8 bits muchísimo mejor a la cuantización 4 bits que se usa para los requisitos mínimos).

Trazabilidad

Para realizar una trazabilidad de los requisitos se ofrece una matriz de trazabilidad resumida. La completa está en los anexos.

Tabla 25. Matriz de trazabilidad (versión reducida)

Código y Requisito	Objetivos	Dependencias
RnF	1	O-004, O-005, O-003
		RF-007, RF-015, RF-016, RF-017

RF	1	O-003, O-004, O-006	RI-1, RI-0
RF	2	O-003, O-004, O-006	RI-1, RI-0
RF	3	O-003, O-004, O-006, O-007	RI-1, RI-0
RF	4	O-003, O-004, O-006, O-007	RI-1, RI-0
RF	5	O-002, O-006	RF-13
RF	6	O-003, O-006	N/A
RF	7	O-001, O-002, O-003, O-004, O-006, O-007	RI-2, RI-3, RI-4, RF-11, RF-12, RF-13, RF-14, RF-8, RF-9, RF-10
RF	8	O-001, O-006, O-008	N/A
RF	9	O-001, O-006, O-008	RF-8
RF	10	O-001, O-006, O-008	RF-8
RF	11	O-002, O-006	RI-2
RF	12	O-002, O-006	RI-4
RF	13	O-002, O-006	RI-3
RF	14	O-002, O-006	RI-4
RF	15	O-003, O-004, O-005, O-006, O-008	N/A
RF	16	O-003, O-005	N/A
RI	1	O-003, O-004	N/A
RI	0	O-003, O-004, O-007	N/A
RI	2	O-002	N/A
RI	3	O-002	N/A
RI	4	O-002	N/A

Plan de riesgos

Tras el establecimiento del proyecto lo primero que se hizo fue un plan de riesgos para evitar problemas durante el desarrollo. Por falta de espacio en este apartado solamente se va a describir una versión general de dicho plan de riesgos. Para ver el plan completo ver el anexo [plan de riesgos](#).

Identificación y Clasificación de riesgos

Tabla 26: Riesgos detectados

ID	Nombre	Descripción
RSG.1	Falta de tiempo	La falta de tiempo o la mala organización llevan a problemas temporales que retrasan o imposibilitan la entrega de la demo
RSG.2	IA muy grande en memoria	El modelo de Inteligencia artificial escogido es muy grande en memoria de la GPU y es imposible de usar (en GPU)
RSG.3	Compatibilidad de entornos	El empleo de una cantidad super diversa de librerías dentro del sistema puede causar problemas de compatibilidad directa o indirecta (a través de las versiones de librerías auxiliares en segundo plano)

RSG.4	Mal <i>fine tuning</i>	El <i>dataset</i> o el formato usado para realizar el <i>fine tuning</i> no es correcto y el modelo no se comporta como se espera de él
RSG.5	<i>Jailbreak</i> de la IA	La realización de consultas externas a las características del modelo causa que dicho modelo se salga de los raíles de comportamiento establecidos
RSG.6	Mucho coste temporal de peticiones	El empleo de un sistema <i>web-app + API</i> produce que el tiempo en obtener las respuestas de las peticiones sea mucho mayor del esperado
RSG.7	Cambio de scope	Un ajuste en el <i>scope</i> del sistema produce que se tengan que realizar muchos cambios dentro del propio sistema que se está desarrollando
RSG.8	<i>Crasheo</i> de peticiones de la IA	Debido a la cantidad de peticiones, se puede provocar que el comportamiento del modelo no sea el esperado o que el modelo <i>crashee</i> por falta de memoria
RSG.9	Mal <i>performance</i> de la IA	El modelo integrado dentro del sistema definido no actúa como se espera o produce unos resultados no acordes con la calidad esperada
RSG.10	Testeo Limitado	Debido a la naturaleza de la aplicación, se realiza un testeo limitado de las aptitudes del modelo y del alcance real de la aplicación desarrollada
RSG.11	Estabilidad de la Demo	La demo desarrollada, pese a estar pensada para ejecutarse como una aplicación completa <i>web + API</i> se va a ejecutar en local. Esto puede causar problemas de estabilidad en base a los recursos.
RSG.12	Compatibilidad CORS	Debido a probar el modelo en local se produce una incompatibilidad CORS al tratar con la misma dirección de <i>request-response</i> (<i>localhost</i>)
RSG.13	<i>Feedback</i> de usuario Limitado	Debido a alguna razón (retrasos, mala planificación, mala exportación, etc....), no se puede enviar la aplicación a otros usuarios para que la prueben a ver qué les parece
RSG.14	Problemas con la actuación del <i>retrieval</i> de RAG	El empleo de las técnicas de RAG dentro del sistema no produce las salidas del sistema y afecta negativamente en la actuación del modelo
RSG.15	Entrada de RAG limitada	La idea del RAG se establece como muy rígida (un formato de documento en concreto) y causa comportamiento inesperado o incorrecto con documentos con un formato diferente

Análisis estratégico frente a los riesgos

Las acciones de prevención son las acciones que se realizan a la hora de detectar los riesgos para intentar prevenirlas. Las acciones de corrección son las que se realizan a la hora de que se produzca un riesgo para evitar sus efectos en el sistema.

Tabla 27: Acciones frente a riesgos

ID	Acciones de prevención	Acciones de corrección
RSG.1	Empleo de una herramienta de planificación de proyectos (<i>taiga - scrum</i>) para mantener un avance constante de los objetivos	Reducir el scope del proyecto y ajustar los objetivos a una meta más aceptable
RSG.2	Empleo de modelos de lenguaje de tamaño suficientemente pequeño como para poder correrlos en la GPU del portátil: 7B o menos	Cambiar el modelo de IA empleado por un modelo más pequeño, a cambio de un peor performance
RSG.3	Intentar minimizar el número de dependencias siempre que sea posible y mantener las versiones según lo que sugieren las librerías	Arreglar las dependencias entre librerías
RSG.4	Seleccionar un <i>dataset</i> adecuado para la tarea. Emplear métodos y herramientas de <i>fine tuning</i> adecuadas. Probar con varios <i>Epochs</i> .	Buscar un <i>dataset</i> mejor, cambiar la estrategia de adiestramiento, usar herramientas externas para el <i>fine tuning</i>
RSG.5	Emplear <i>safeguards</i> para el modelo a entrenar. Se emplea Nemo-guardrails tanto para evitar el <i>jailbreak</i> como para limitar el <i>input</i> simplemente a los formatos pedidos	Ampliar el número de <i>guardrails</i> que tiene el sistema
RSG.6	N/A	N/A
RSG.7	Se establece claramente el scope del sistema a diseñar, construyendo de manera poco restrictiva en caso de tener que añadir más piezas	Ajustar el trabajo al cambio de scope
RSG.8	Se emplean limitadores de peticiones desde el <i>frontend</i> . Se espera emplear limitaciones de procesado de peticiones en la API.	Aplicar métodos mucho más estrictos de control de peticiones en el lado del servidor
RSG.9	Se realiza la combinación de raíles + rag + ejemplos en el <i>prompt</i> para limitar el posible error	Probar a cambiar el <i>prompt</i> y los ejemplos, intentar probar con un posible modelo de IA más grande
RSG.10	Se desarrolla un plan de testeo lo antes posible que se va desarrollando y aplicando de manera incremental	Realizar un plan de testeo o unas pruebas más exhaustivas

	durante el tiempo de vida del proyecto	
RSG.11	Se emplea un entorno controlado y se miran los <i>logs</i> en caso de que haya un error causado por ellos	Minimizar las posibles variables del entorno con relación a la ejecución. Monitorizar de manera más exhaustiva el sistema
RSG.12	Se emplea FLASKCORS en la API para evitar problemas con el CORS	N/A
RSG.13	Se planea realizar un procedimiento de <i>feedback</i> de usuario en cuanto se tenga una demo lo suficientemente aceptable	Establecer el diálogo con una serie de interesados de confianza para recibir un <i>feedback</i> del sistema y tomar acciones en base al <i>feedback</i>
RSG.14	Se testea el RAG antes de usarse	Modificar el formato del RAG dentro del sistema, modificar el tipo de uso que se le da a la estrategia de RAG, cambiar el método de RAG utilizado
RSG.15	Se planea usar diversos tipos de documentos para el RAG durante el proyecto para limitar los errores	Modificar el formato del RAG dentro del sistema, modificar el tipo de uso que se le da a la estrategia de RAG, cambiar el método de RAG utilizado

Historial de riesgos

En el caso de este proyecto, sucedió el caso de que se dieron varios de los riesgos y se tuvieron que aplicar las acciones correctivas:

Tabla 28: Logs de riesgos

ID	Fecha	Acción
RSG.12	10/02/2024	Se instala una librería para solucionar los posibles errores CORS (<i>FlaskCORS</i>)
RSG.7	06/03/2024	Se descarta el uso de <i>fine tuning</i> por la falta de existencia de un <i>corpus</i> lo suficientemente decente y se decanta por el uso de una IA de propósito general LM
RSG.2	20/04/2024	Llama3 salió con 1B más de parámetros de los esperados por lo que tenemos que continuar haciendo la demo con Llama2

Capítulo 3: Diseño

En este apartado vamos a ver el diseño de cada uno de los componentes de la aplicación. Vamos a ofrecer a un nivel de detalle elevado descripciones de los elementos del sistema, las razones de porque son así y sus funcionalidades. De esta manera podremos entender las relaciones intrínsecas dentro del sistema.

Diseño de componentes del sistema

En el apartado de [estructura](#) en la introducción se presentó un diseño de cómo era la visualización de la aplicación a muy alto nivel para que un posible cliente entendiese el producto a desarrollar. Una descripción a más bajo nivel de dicho diseño sería la siguiente:

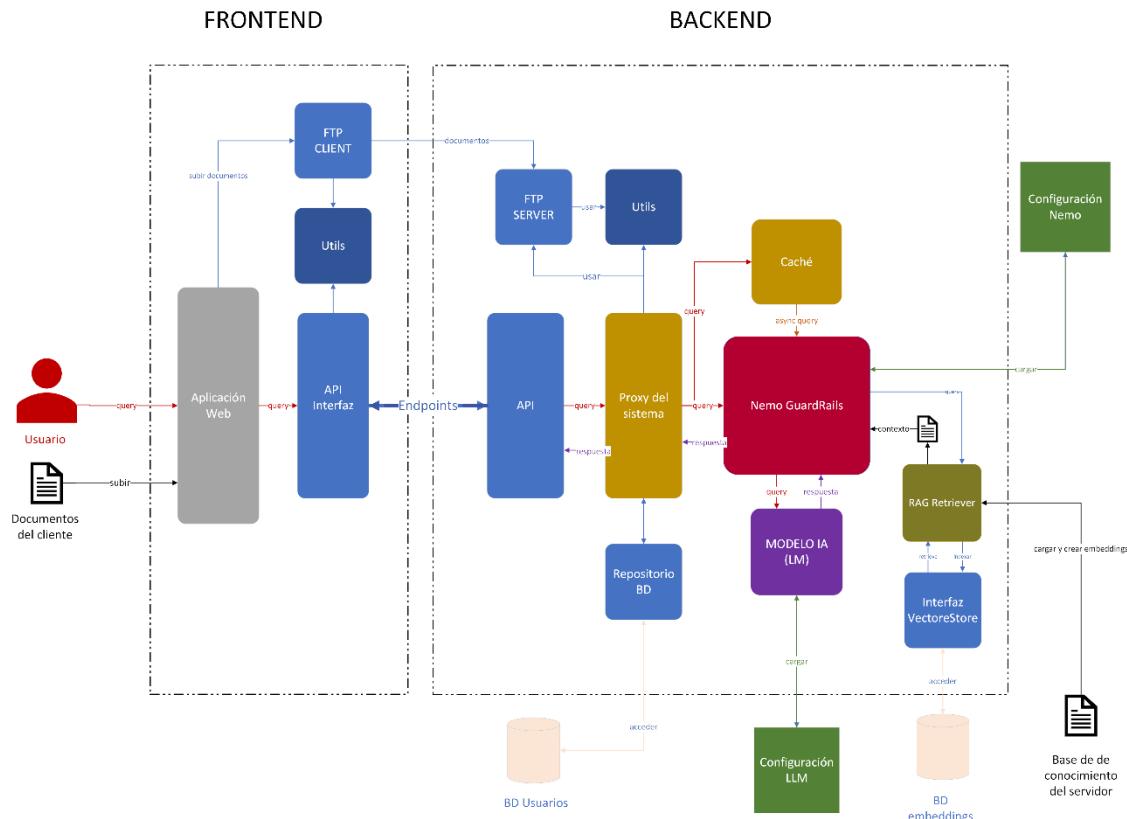


Figura 9: Diagrama de componentes

En este diagrama podemos ver los componentes del sistema. Los componentes del sistema se explican en detalle en siguientes apartados.

De manera análoga al diagrama anterior, entramos más en detalle en el diagrama de comportamiento presentado en la introducción. Presentamos un diagrama de bajo nivel también del comportamiento del sistema donde podemos ver AG.

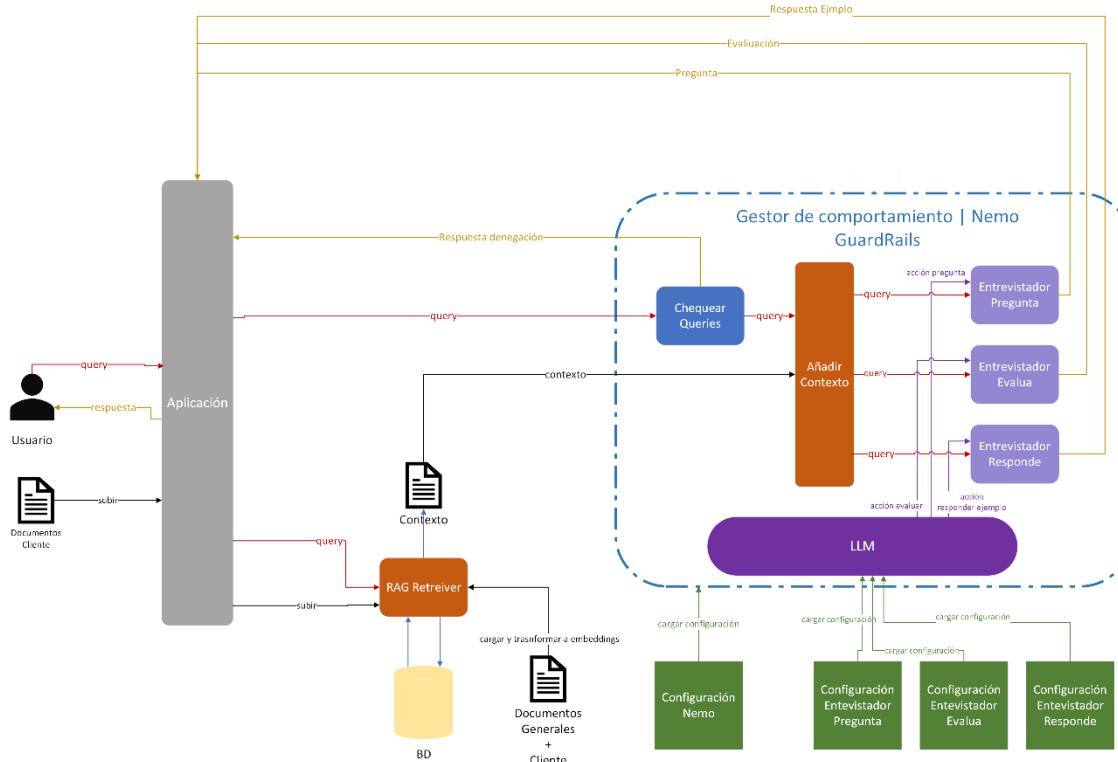


Figura 10: Diagrama de comportamiento

En el diagrama anterior, podemos observar cómo funciona el comportamiento por defecto de nuestro entrevistador. En base al contexto (la *query*) recuperamos información adecuada y ejecutamos una de las tres modalidades del entrevistador (LLM).

Dividimos los componentes de la aplicación en tres grandes módulos: *frontend*, *backend* y bases de datos / ficheros externos.

Frontend

En esta capa encontramos la capa de interacción con el usuario. Esto se hace a través de una interfaz gráfica creada con Streamlit¹⁰. A su vez, esta capa presenta varios componentes. Por un lado, tenemos la interfaz gráfica en sí, que presenta un *login/registro*, un tutorial de uso y el *chatbot* en sí. Por otro lado, tenemos los módulos de apoyo, que sirven para delegar y abstraer lógica; estos serían un cliente ftp para subir archivos, una interfaz de la API para el cliente y un lector de ficheros.

Backend

Aquí es donde se encuentra la lógica del trabajo real de la aplicación puesto que todas las operaciones se hacen en el servidor. A su vez, también podemos dividir el *backend* en varios componentes que se explicarán en más detalle en un futuro:

1. Una parte integral de la aplicación es la API, necesaria para poder realizar las consultas.
2. El modelo de lenguaje que se usa para las operaciones se define en GenericLLM y permite instanciar cualquier modelo de HuggingFace.
3. El gestor de comportamiento, que es la parte del backend que lee las normas escritas y establece que comportamientos del entrevistador se tienen que ejecutar para cada *input* del usuario. Además, a través del gestor es donde se encapsulan tanto el LLM como el RAG.

¹⁰ <https://streamlit.io/>

4. El RAG. Esto es una técnica que depende de guardar los *embeddings* de los documentos para poder asociarlos como contexto en tiempo de ejecución. En este caso, tenemos una VectorStore que sirve para almacenar los *embeddings* de los documentos para que un retriever que tenemos instanciado pueda devolver dichos documentos.
5. Por último, tenemos también ayudantes, como serían un *parser* para los argumentos, un servidor FTP o una caché para ir precargando respuestas en algunas situaciones.

Bases de datos y ficheros externos

En esta capa tenemos primero la base de datos que almacena los datos de los usuarios y sus puntuaciones. También tenemos la base de datos vectorial que almacena los *embeddings* de los documentos. Y, además, las configuraciones estáticas del modelo de lenguaje y del gestor de comportamientos que se pueden cambiar en frío.

Diagrama de clases

Podemos abstraer el previo sistema a un diagrama de clases con el que poder observar las relaciones entre módulos de manera más directa. Cabe destacar que:

1. El diagrama de clases, pese a ser bastante detallado, no incluye a todo detalle los contenidos de cada uno de los componentes puesto que eso añadiría mucha verbosidad.
2. Algunos elementos del diagrama de clases tienen el nombre algo cambiado por la incompatibilidad entre la herramienta de dibujado y los nombres (formato).

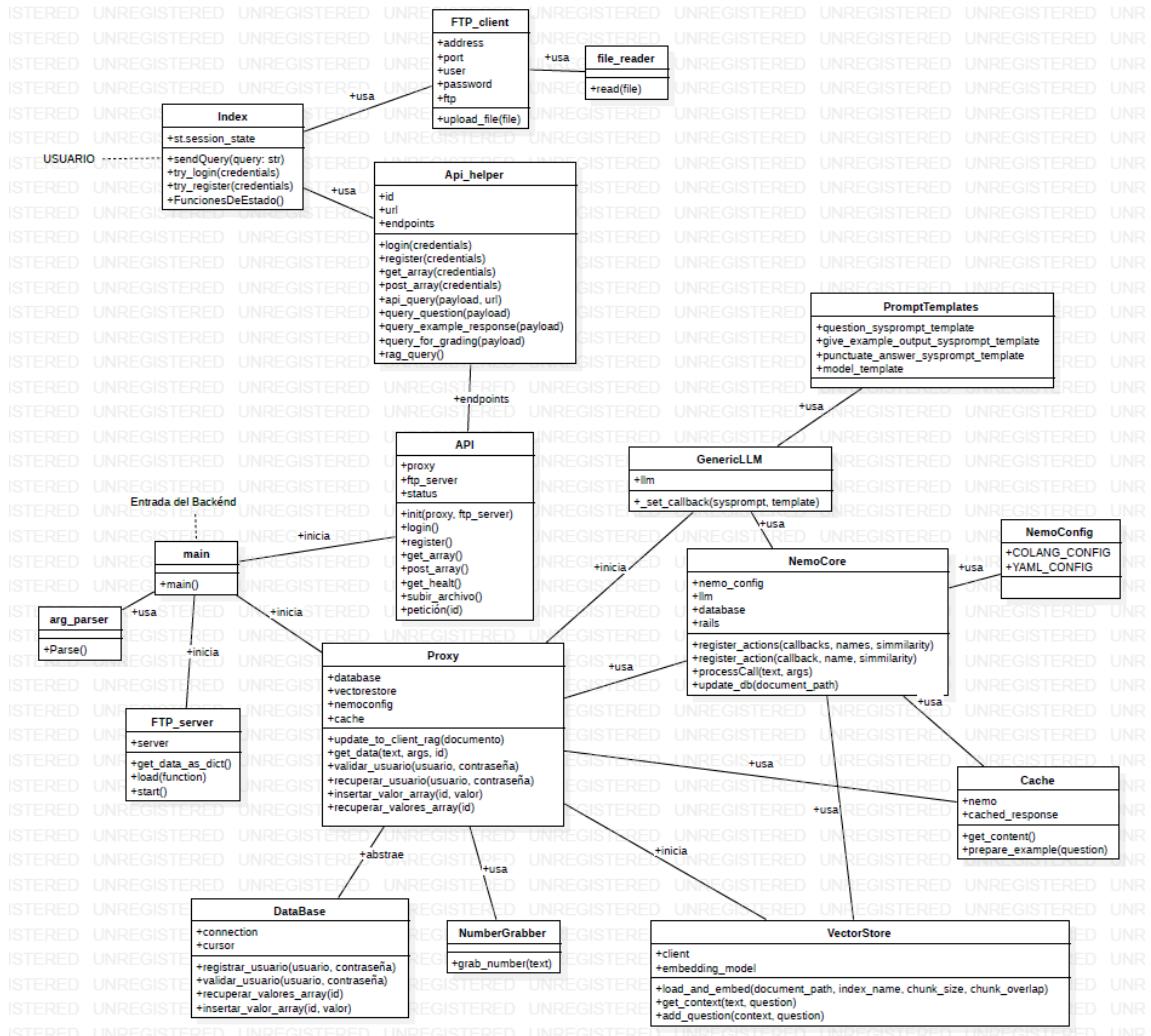


Figura 11: Diagrama de clases

Explicación detallada de los componentes más importantes

Frontend

La parte con la que va a interactuar y que tiene una importancia enorme en la usabilidad de la aplicación es la interfaz gráfica. La tecnología empleada para desarrollar está se base en Tecnologías: *Streamlit*¹¹ y *Streamlit-chat*¹².

Hemos definido una aplicación donde el punto importante es que el usuario sienta que está siendo entrevistado pero que, al mismo tiempo, su interfaz sea sencilla y usable. Para el diseño de la interfaz se tuvo la limitación del *framework* a utilizar, que es bastante restrictivo con lo que se puede y no se puede hacer. Se ha de destacar, además, que el *chatbot* implementado por defecto habla en inglés (puesto que es el idioma ofrecido comúnmente por los modelos *open-source* de gama alta) por lo que la interacción con la aplicación tiene componentes en inglés.

La primera ventana que vemos al entrar a la *url* de la aplicación es la pestaña de *login*. Es una pestaña con un diseño minimalista pero que se corresponde con el estándar de los *logins* existentes; en caso de error notifica dicho error.

Figura 12: Login

Si pulsamos el botón de registrarnos, cambiamos a la pestaña de registro. Esta pestaña, de manera análoga al *login*, tiene un diseño típico y no requiere de una explicación extensa.

Figura 13: Creación Cuenta

¹¹ <https://streamlit.io/>

¹² <https://pypi.org/project/streamlit-chat/>

Destacar que el objetivo de la cuenta es, más que la privacidad, tener un registro de los datos de la persona; por ende, no hay criterios de exigencia en cuanto a usuario y contraseña más que no coincida con uno ya existente.

Al registrarnos en la aplicación, lo primero que podemos ver es una guía de uso, que se corresponde con la guía de uso del usuario que está definida para la aplicación. En esta guía podemos ver las opciones de operaciones que podemos hacer, que significa cada botón y las limitaciones del sistema. Si se quiere leer completa, revisar la [guía de usuario](#). El usuario tiene habilitado un botón de continuar con el que puede redirigirse a la página principal de la aplicación.



Tutorial

We present the tutorial for the use of this app. First as you have seen we have presented you with both

Figura 14: página de tutorial

La página principal de la aplicación está organizada de la siguiente manera: Lo primero que vemos son dos botones (uno para volver al tutorial y otro para cerrar sesión). Seguido de estos están el título de la aplicación y de un enlace al repositorio de *GiltHub*.



Figura 15: botones de tutorial y logout

Posteriormente, encontramos el *chat* que se va a usar para hablar con el entrevistador. Si se quiere interactuar con el *bot*, se sigue el procedimiento estándar de escribir el contenido del mensaje y pulsar un botón correspondiente a alguna de las tres acciones que puede realizar:

- Pedir una pregunta de entrevista en base a un tópico
- Responder la pregunta como si el fuera el entrevistado
- Ser evaluado, es decir, pedir que el *bot* que evalue la respuesta que va a dar el usuario

Ofrecemos un botón distinto para cada una de las acciones para facilitar el entendimiento. Se puede escribir en el *textbox* que está encima de dichos botones. El proceso de uso es primero se pide una pregunta, y después o se responde o se pide un ejemplo.

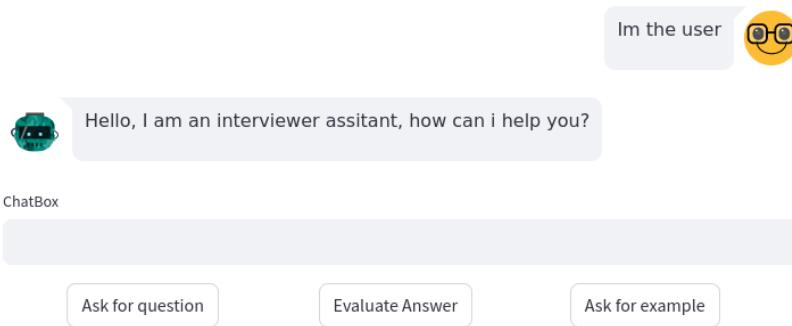


Figura 16: Entrevistador y las tres opciones de chat

Siguiendo hacia abajo en la interfaz encontramos el *widget* con el que podemos subir el documento que queramos para el conocimiento.

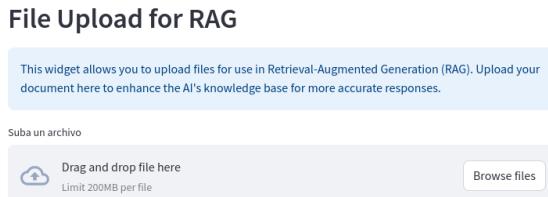


Figura 17: RAG cliente

Por último, permitimos al usuario visualizar su progreso en tiempo real a través de una gráfica de puntuaciones señalizada como tal.

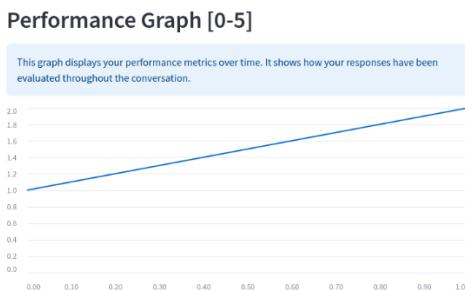


Figura 18: Gráfico de puntuaciones

Utils

Tanto para el *backend* como para el *frontend* se habilitan una serie de clases categorizadas como *helpers* o *utils*. Son módulos relativamente sencillos que sirven para ayudar a otros módulos.

- *API_helper*: módulo que sirve para abstraer la comunicación entre la interfaz y la API; habilita las funciones para cada una de las consultas que se pueden hacer.
- *FTP_client*: cliente ftp que se instancia en el *frontend* para poder subir los archivos que el cliente quiere subir al servidor.
- *File_reader*: ayudante del *FTP_client*, sirve para poder leer los ficheros de manera correcta.
- *FTP_server*: servidor FTP que se instancia en el *backend* para poder recibir los archivos; habilita un usuario solamente con permisos para subir el archivo y realiza, cuando se carga, la función que se haya definido en el *proxy* (en este caso, por defecto carga el documento en el *client_context* de la base de *embeddings* y borra el fichero del lado del servidor).
- *Parser*: sirve para leer los argumentos que se le pasan al *main* del *backend* para la instanciación.
- *Main*: fichero que inicia cada uno de los componentes necesarios para que funcione el *backend*.
- *NumberGrabber*: clase que sirve para seleccionar la nota de la evaluación cuando el entrevistador evalúa.

API

El api es el módulo que habilita la comunicación entre el *frontend* y los servicios ofrecidos. Para ello habilitamos un servidor Flask¹³estándar en el que ofrecemos los siguientes *endpoints*:

¹³ <https://flask.palletsprojects.com/en/3.0.x/>

1. /heatlh: sirve para acceder a la salud de la API.
2. /login: sirve para realizar el *login* de la aplicación (credenciales en argumentos).
3. /register: sirve para realizar el registro en la aplicación (credenciales en los argumentos).
4. /subirarchivo: sirve para acceder a las credenciales del servidor ftp para poder realizar la subida de datos.
5. /petición/<id>: sirve para realizar una comunicación con el *bot* para el usuario id. Dentro de las credenciales de la petición ya se encuentra la pregunta. Será el gestor de comportamientos el que decida cómo se comporta LLM ante esa pregunta.

Proxy y Caché

Dentro de los módulos de la aplicación se emplea el *proxy* como la manera para realizar el acceso a otros módulos. Es decir, centralizamos el acceso tanto al modelo como a la interfaz de la base de datos a través de este *proxy* que actúa como una interfaz general. Además, el *proxy* dictamina el uso de extras como la *caché*.

Cuando el entrevistador hace una pregunta, el usuario tiene dos opciones: o contestarla él para ser evaluado o que el propio *bot* responda a la pregunta. Bajo esta premisa, establecemos una *caché* que va precargando la respuesta del *bot* ya que en caso de que el usuario quiera el ejemplo de la respuesta ya la tiene cargada o a medio cargar y la comunicación sería casi instantánea (mientras el usuario escribe su respuesta, el entrevistador tiene tiempo de sobra para terminar de cargar su ejemplo y así el usuario no va a notar prácticamente ningún *delay*).

LLM

Ofrecemos un módulo de LLM genérico que sirve para eso, para instanciar un modelo genérico con formato de HuggingFace¹⁴¹⁵ ya sea en local o desde el repositorio de HuggingFace oficial.

Las limitaciones de este módulo son las siguientes:

1. Si el modelo es propietario, como lo es el llama2 que se emplea por defecto, requiere de un token para poder hacer el *login* en la API de HuggingFace.
2. El modelo se puede “cuantizar” de manera que se baja su coste de recursos de memoria a cambio de perder precisión¹⁶. Este proceso solo se puede realizar si se ejecuta en la tarjeta gráfica, por lo que si no hay GPU entonces la “cuantización” se descarta automáticamente.
3. Se emplea el uso de un elemento dentro de HuggingFace que se llama *chat-templates*¹⁷. Este es un parámetro que tienen los modelos de tipo instrucción por defecto y que automatiza la plantilla de la pregunta. En caso de que el modelo no tenga esta plantilla se tiene que poner la plantilla manualmente en el módulo de *prompt-templates* (la plantilla está en internet y se tarda un minuto en modificar).

La manera en la que realizamos la generalización en este módulo es empleando el uso de cadenas de *langchain*¹⁸ para poder crear callbacks con sysprompts diferentes. Es decir, podemos crear varias “personalidades” para el mismo modelo y de esta manera generar los comportamientos deseados (en nuestro caso el entrevistador). Más detalles en el apartado de [Tecnologías:Langchain](#).

¹⁴ <https://huggingface.co/>

¹⁵ <https://arxiv.org/abs/1910.03771>

¹⁶ <https://arxiv.org/abs/2402.18158>

¹⁷ https://huggingface.co/docs/transformers/main/en/chat_templating

¹⁸ <https://www.langchain.com/>

Gestor de comportamiento (Nemo)

Tal y como se ha indicado previamente se emplea un gestor de comportamiento para gestionar las diversas configuraciones que instanciamos para el entrevistador. Nemo se explica algo más en detalle en [Tecnologías:Nemo](#). Empleamos varias de las opciones que se nos ofrece dentro de esta tecnología:

1. Lo primero que hacemos es registrar las acciones¹⁹ de nuestro entrevistado unidas con el RAG para que, cuando se realicen las consultas, se tenga acceso a los documentos.
2. También registramos dentro de la configuración los flujos de las conversaciones. Ofrecemos dos opciones. La primera sirve para cumplir a rajatabla los objetivos del proyecto: que el entrevistador funcione. Para ello habilitamos variables dentro de la conversación con las que vamos cambiando entre modos del entrevistador. Es emplear un “if” un poco más dinámico y que se instancia desde la configuración y no desde el código. Por otro lado, ofrecemos una configuración más Naive (inocente) en la que el propio *bot* escoge como responder antes las acciones. Esta opción tiene el problema de que está sujeta a la capacidad semántica del gestor y en este caso al emplear reducción de precisión (por el ordenador) funciona de manera mediocre.

Es dentro de esta configuración donde situamos los tres modos del entrevistador. Para ello los registramos con los siguientes *prompts* de sistema:

```
class PromptTemplates:
    [
        question_sysprompt_template = "You are an experienced job interviewer. Your task is to generate a single, relevant interview question based on the given topic or job position. The question should be thought-provoking and designed to assess the candidate's skills, experience, or fit for the role. Provide only the question with the minimal human small talk commentary or explanation."
        give_example_output_sysprompt_template = "You are a job candidate in an interview. Your task is to provide a concise, professional answer to the given interview question. Your response should demonstrate relevant skills, experience, or qualities that would make you a strong candidate for the position. Answer as a human would, with a balance of confidence and humility. Provide only the answer without any additional commentary."
        punctuate_answer_sysprompt_template = """You are an expert hiring manager evaluating candidates' responses to interview questions. Your task is to assess the given answer based on its relevance, accuracy, and effectiveness. Use the following scale for evaluation:
        0 - Refuses to answer
        1 - Poor: Answer is off-topic or inadequate
        2 - Fair: Answer is somewhat relevant but lacks depth or clarity
        3 - Good: Answer is relevant and demonstrates basic understanding
        4 - Very Good: Answer is well-articulated and shows strong relevant skills or experience
        5 - Excellent: Answer is exceptional, demonstrating deep understanding and outstanding fit for the role

        Provide your evaluation in the following format:
        Score: [0-5]
        Brief explanation: [Your concise assessment of the answer's strengths and weaknesses with humanlike writing]
    ]
```

Ejemplo de código 1: Templates usadas para las acciones del entrevistador

Ahora lo que hacemos es crear una configuración de tipo COLANG de Nemo, donde se inscriben las acciones a realizar y además se definen los flujos de conversación. Ofrecemos dos tipos de conversación.

1. Por un lado, una empleando “ifs” por lo que está mucho más regida por las normas siendo menos dinámica. Estos “ifs” siguen siendo mejores que uno dentro del propio código porque se pueden cambiar desde la configuración
2. Por otro lado, ofrecemos una configuración dinámica donde el propio *bot* escoge qué modo emplear.

¹⁹ https://docs.nvidia.com/nemo/guardrails/colang_2/language_reference/python-actions.html

```

HARDCODED_COLANG_CONFIG = """
define user express ill intent
    "I hate you" [...]

define bot express cannot respond
    "I am sorry but that is outside of my
capabilities"

define flow
    user express ill intent
    bot express cannot respond

# Hardcoded Question flow
define flow
    user ...
    if $answer
        $answer = execute
    user_answer(inputs=$last_user_message)
        bot $answer
    if $example
        $answer = execute
    bot_answer(inputs=$last_user_message)
        bot $answer
    if $question
        $answer = execute
    bot_response(inputs=$last_user_message)
        bot $answer
"""

NAIVE_COLANG_CONFIG = """
define user express ill intent
"""

```

Ejemplo de código 2: Configuraciones disponibles

Hay que destacar el uso de una cuarta acción intermedia para el resumen de los temas cuando se hace una pregunta.

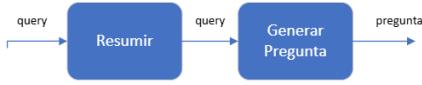


Figura 19: cuarta acción (Resumir)

RAG y VectorStore

La técnica de RAG²⁰ se explicó anteriormente de manera sencilla. El proceso de realización para el RAG Naive que se implementa es el siguiente:

1. Se recolecta la base de conocimiento que queremos emplear (documentos) y se dividen en segmentos (normalmente los segmentos tienen acoplamiento entre sí).
2. Se escoge un modelo de IA que va a ser el *embedder* y el *retriever*. Este modelo coge los segmentos que hemos fragmentado y los traduce a *embeddings* (vectores de representaciones de tokens).
3. Esos *embeddings* se guardan en la base de datos, normalmente acompañados de metadatos, como puede ser el número de página o el contenido sin transformar en vector.
4. A la hora de querer recuperar un contexto guardado en la base de datos vectorial, el *retriever* parte del texto que se tenga en caliente, y también lo traduce a *embeddings*.
5. Después de traducirlo, hace un proceso que se llama búsqueda semántica, donde se emplea una fórmula para calcular la semejanza (*similarity*²¹). De esta manera se calcula la semejanza entre el texto actual y todos los contextos guardados y se devuelve el más parecido (si se supera un *threshold* de parecido).

²⁰ <https://arxiv.org/abs/2005.11401>

²¹ <https://arxiv.org/abs/2304.01330>

6. Este contexto parecido es el que se agrega al *query* del LLM para agregar factualidad a su respuesta.

Existen varias maneras de realizar el cálculo de la semejanza. En este caso se emplea el método de la distancia por coseno²² en la que se multiplican los vectores y se dividen entre sus distancias para devolver un valor entre [-1,1]. De estos valores, -1 simboliza contrariedad, 0 simboliza indiferencia y 1 semejanza.

$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \cdot \sqrt{\sum_{i=1}^n B_i^2}},$$

Figura 20: fórmula del cálculo de semejanza por distancia de coseno

Lo que se hace en el módulo es aplicar esta técnica con la diferencia de que se hace con tres contextos al mismo tiempo. Por un lado, se tiene el contexto del servidor que es la base de conocimiento que se tiene en frío. Por otro lado, el contexto del cliente que es el posible documento que suba el cliente. Por último, es el contexto por semejanza, donde si se tiene un contexto actual muy parecido a uno previo se anexan ejemplos de preguntas anteriores.

En este caso tenemos una base de datos de *embeddings* vectorial²³ que guarda el contexto por similitud (el tercero) pero en la práctica los otros dos contextos son volátiles, es decir, cada vez que cambian se borran. Esto es porque el cliente puede subir diferentes documentos en caliente, y en caso del servidor se cargan todos los documentos al empezar.

BD

La base de datos sirve para guardar los datos de los usuarios y sus puntuaciones. Se emplea SQLite por ser la manera más sencilla de aplicar una base de datos. Tiene su propio módulo interfaz y realiza las operaciones que se han designado previamente de *login*, registro, recuperación del historial y modificación del historial.

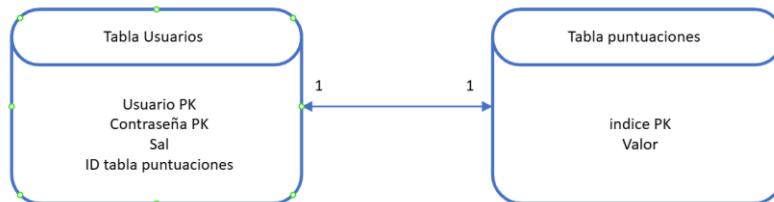


Figura 21: Diagrama Relacional

Configuraciones

Dentro del sistema tenemos dos tipos de configuraciones que se van a emplear:

1. Por un lado, tenemos la configuración del LLM, donde podemos modificar el modelo, la plantilla y el nivel de precisión.
2. Por otro lado, tenemos las configuraciones del NEMO, donde podemos modificar tanto las acciones que se van a realizar como el flujo de las conversaciones que se van a emplear.

Ambas fueron escritas en detalle en el apartado de [Nemo Guardrails anterior](#).

²² <https://arxiv.org/abs/2403.05440>

²³ <https://arxiv.org/abs/2310.11703>

Implementación de los casos de uso

Como se ha explicado anteriormente, los casos de uso descritos pueden ser descritos como historias de usuario. A continuación, presentamos las maneras en las que se ha implementado cada uno de los casos de uso. He de destacar que los casos de uso clasificados como complejos se presentan junto con un diagrama de secuencia para simplificar su explicación.

CU – 001

La historia de usuario a implementar en este caso de uso es: “como usuario quiero iniciar sesión en la aplicación para acceder a mi cuenta”.

Para realizar este caso de uso se emplea el uso de los módulos de la interfaz gráfica, de la comunicación con la API y de la base de datos como componentes principales. Los pasos para realizar este caso de uso son los siguientes:

1. Introducir las credenciales en la interfaz gráfica (*login*) y pulsar botón de acción.
2. Se emplea el módulo de Ayuda de la API para hacer la petición al *endpoint* correspondiente de la API (/login).
3. La API recibe la petición y emplea el *Proxy* para hacer el rutado hacia la interfaz de la base de datos.
4. Las credenciales llegan a la base de datos. En esta se realiza una función de validación para recuperar el id de usuario si existe.
5. Si existe se recupera además las puntuaciones acumuladas del usuario y si no existe se devuelve un mensaje de error.
6. Desde el proxy se construye la respuesta: id (un hash) y las puntuaciones (array de valores) y la API se las devuelve al *frontend*.

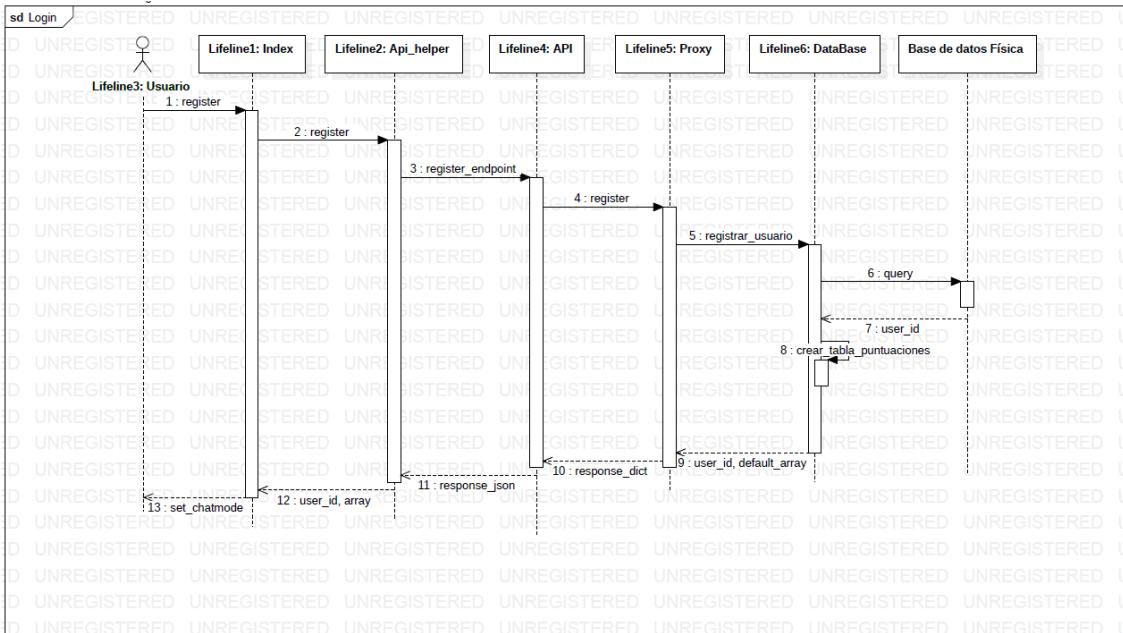


Figura 22: Diagrama de Secuencia del Login

CU – 002

La historia de usuario a implementar en este caso de uso es: “como usuario quiero ser capaz de registrarme en la aplicación para crear una cuenta nueva”.

Análogo al *login* en cuanto al uso de módulos, principalmente interfaz gráfica, API y base de datos.

1. Introducir las credenciales en la interfaz gráfica (*register*) y pulsar botón de acción.
2. Se emplea el módulo de Ayuda de la API para hacer la petición al *endpoint* correspondiente de la API (/register).
3. La API recibe la petición y emplea el *Proxy* para hacer el rutado hacia la interfaz de la base de datos.
4. Las credenciales llegan a la base de datos. En esta se intenta insertar el usuario.
5. Si no existe se crea y si existe devuelve un error.
6. Con el id recién creado iniciamos la tabla de datos de las puntuaciones de ese nuevo usuario (si se ha realizado el registro con éxito).
7. Desde el proxy se construye la respuesta: id (un *hash*) y las puntuaciones (array de valores) y la API se las devuelve al *frontend*.

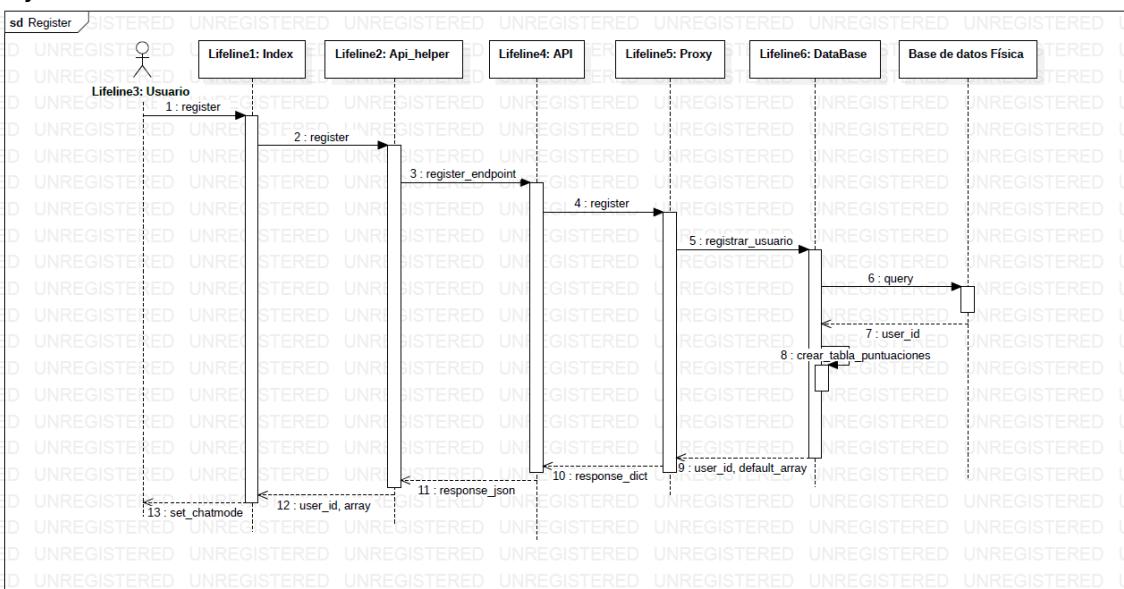


Figura 23: Diagrama de Secuencia del Registro

CU – 003

La historia de usuario a implementar en este caso de uso es: “como usuario quiero poder ver mi historial de progreso”.

De manera intrínseca, emplea principalmente los módulos de la GUI, de la API y de la base de datos.

1. Realizar el CU – 001 o CU – 002 para entrar en tu cuenta.
2. Desplazarnos hacia abajo en la interfaz del *chatbot* y observar la gráfica de progreso.

En la práctica, podemos observar cómo se cumple esto a través de la [Imagen de login anterior](#).

CU – 004

La historia de usuario a implementar en este caso de uso es: “como usuario quiero ser capaz de actualizar mi historial de puntuaciones para poder ver mi progreso”.

Empleamos de nuevo, los módulos de la interfaz, la API y la base de datos principalmente.

1. Realizar el CU – 007 con el entrevistador en modo evaluación.
2. El sistema extrae la puntuación automáticamente, la agrega en la base de datos y actualiza el vector de puntuaciones.
3. Como usuario lo vemos reflejado en la gráfica de puntuaciones en tiempo real.

En la práctica, podemos observar cómo se cumple esto a través de la [imagen del registro anterior](#).

CU – 005

La historia de usuario a implementar en este caso de uso es: “como usuario quiero ser capaz de actualizar la base de conocimiento subiendo un documento”.

Este caso de uso hace uso del *widget* de subir archivo de la interfaz, de la comunicación de la API, de los servicios FTP y del módulo RAG para hacer el indexamiento del documento.

1. El usuario selecciona el documento que quiere subir al LLM como base de conocimiento en el *widget* de la interfaz gráfica.
2. Se realiza la petición de la API al *endpoint* (/subirarchivo).
3. Conseguimos las credenciales para acceder al servidor FTP.
4. Subimos el documento al servidor FTP.
5. El *Proxy* redirecciona el documento al módulo de RAG.
6. El *Retriever* instanciado indexa el documento y guarda los *embeddings* en la colección *client-context*.
7. El sistema borra el archivo temporal que se había subido al *backend*.

En la práctica, podemos observar cómo se cumple esto a través de la realización del caso de uso CU – 007 o sus extensiones, ya que podemos apreciar el contexto subido en tiempo real (lo que tarde en actualizarse la base de conocimiento).

Notar también, que el punto de entrada es la [imagen de subir RAG](#) descrita anteriormente.

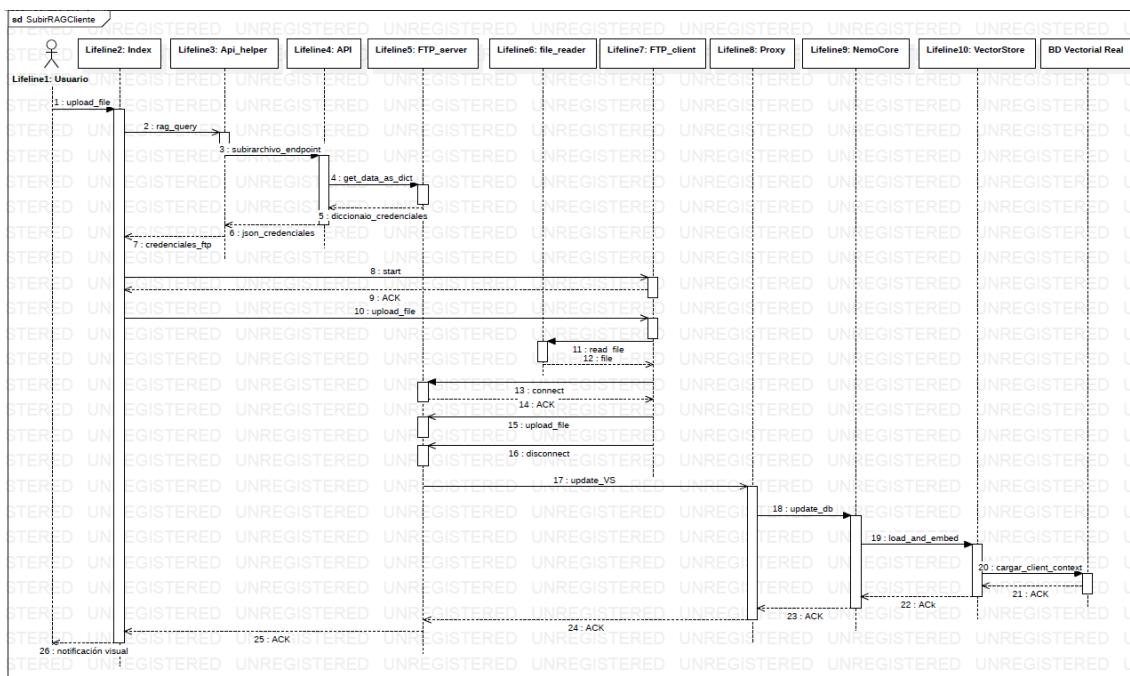


Figura 24: Actualizar la base de conocimiento

CU – 006

La historia de usuario a implementar en este caso de uso es: “como usuario quiero ser capaz de chequear la salud de la API para ver si me puedo conectar a ésta”.

El caso de uso emplea el módulo de la API y la función de chequeo definida en el módulo de ayuda. Se consulta cada X tiempo o se puede consultar de manera manual. Se ve limitada por la interacción sin arreglar con los hilos del *framework* de *streamlit*.

1. El administrador habilita una función que realizar en base al estado de salud detectado desde el *helper* de la API. Por defecto, *loggea* el estado de salud.
2. El usuario puede acceder al *endpoint* de salud para ver el estado de la API.
En la práctica, podemos observar los logs de salud en tiempo real (cada 2 segundos).

CU – 007

La historia de usuario a implementar en este caso de uso es: “como usuario quiero ser capaz de usar el *chatbot* ofrecido para sacarle partido a la aplicación”.

Este caso de uso y sus derivados hacen uso de la mayoría de los módulos que comprenden el sistema. Entre ellos destacamos la GUI, la API, el gestor de comportamientos, el LLM accionado internamente, el RAG para el contexto y extras como el actualizador de puntuaciones o la caché para las extensiones. El caso de uso genérico realiza los siguientes pasos:

1. El usuario escribe el mensaje en el *textbox* de la GUI y acciona el botón de enviar. La [imagen del chat](#) se expuso anteriormente.
2. El ayudante manda el mensaje al *endpoint* de la API /petición.
3. La API emplea el *proxy* que a su vez redirecciona al gestor de comportamientos.
4. Se selecciona un comportamiento.
5. Se actualiza el mensaje con el contexto que el *Retriever* del RAG haya considerado oportuno.
6. La variante del entrevistador (u otro *chatbot*) empleada realiza su tarea.
7. Devolvemos el resultado al *proxy* que puede realizar tareas extra sobre el mensaje o basado en él.
8. Devolvemos el mensaje construido a la GUI.

Este caso de uso se realizaría en caso de habilitar el modo genérico en el *chatbot* siguiendo la guía de desarrollador.

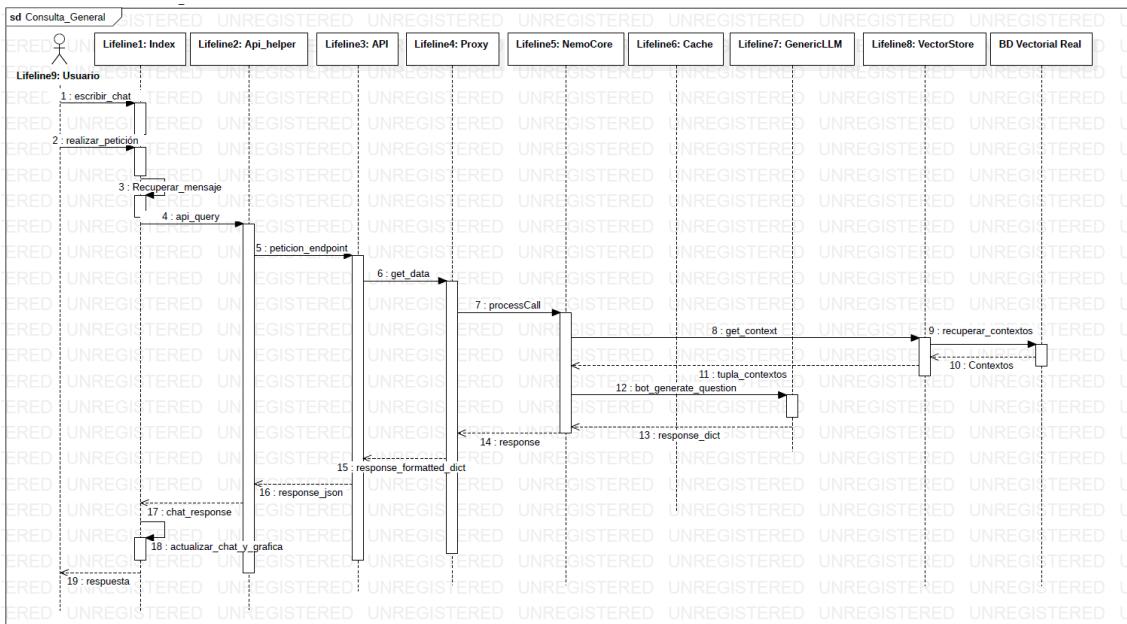


Figura 25: Mandar mensaje al chatbot por defecto

CU – 007.1

La historia de usuario a implementar en este caso de uso es: “como usuario quiero pedir al *chatbot* una pregunta para poder responderla”. Es una extensión de CU – 007, por lo que emplea los mismos módulos y además emplea la caché.

1. Se introduce el tema del que se quiere recibir una pregunta y se pulsa el botón correspondiente. La [imagen del chat](#) se expuso anteriormente.
2. Se recibe la petición en el *endpoint* /petición.
3. El proxy emplea al gestor de comportamientos para responder.
4. Se añade el contexto a la petición y se usa al entrevistador en modo pregunta para generar la pregunta.
5. Se coge la pregunta, se guarda en el *similarity-context* de la base de datos de *embeddings* y se deja precargando la respuesta en la caché.
6. Se devuelve la pregunta en la interfaz.

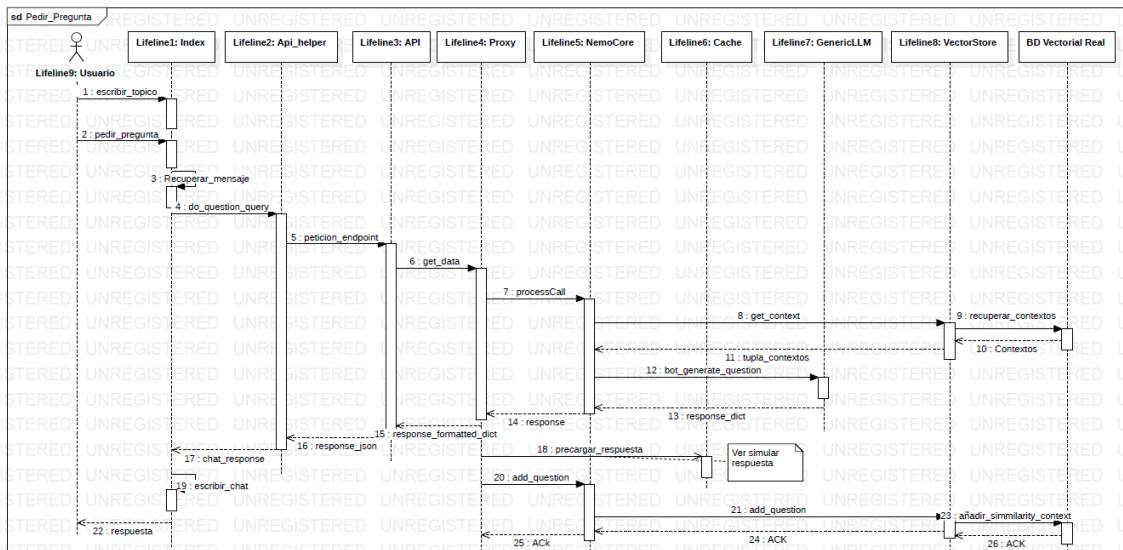


Figura 26: Petición de una pregunta

CU – 007.2

La historia de usuario a implementar en este caso de uso es: “como usuario quiero responder al *chatbot* la pregunta y que me evalúe la respuesta”. Es una extensión de CU – 007, por lo que emplea los mismos módulos y además emplea y gestiona los módulos de gestión de la evaluación.

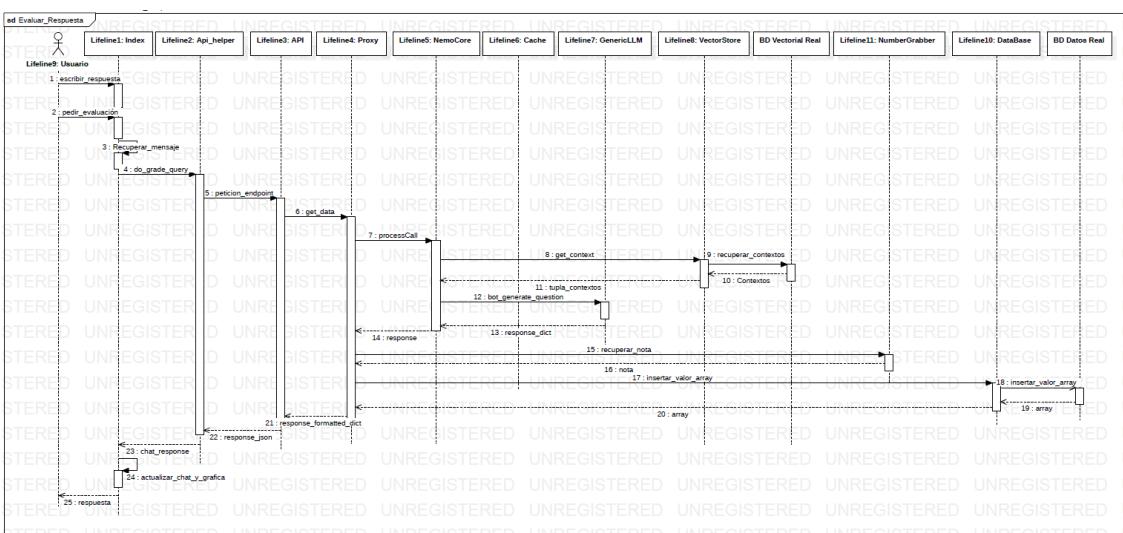


Figura 27: Evaluación de una respuesta

1. Se introduce la respuesta y se pulsa el botón de evaluación. La [imagen del chat](#) se expuso anteriormente.

2. El proxy emplea el comportamiento del evaluador del entrevistador.
3. Se añade el contexto a la pregunta y el evaluador genera su evaluación.
4. Se extrae la puntuación y se guarda en el historial.
5. Se devuelve la evaluación generada y el historial de evaluaciones actualizadas. El historial se puede ver en la [imagen de puntuaciones anterior](#).

CU – 007.3

La historia de usuario a implementar en este caso de uso es: “como usuario quiero pedir al *bot* un ejemplo de respuesta a la pregunta que me acaba de hacer”. Es una extensión de CU – 007, por lo que emplea los mismos módulos y además emplea la caché.

1. En segundo plano la respuesta se está precargando en la caché desde el caso de uso CU – 007.1. Esta emplea el comportamiento adecuado, el RAG y el contexto para realizar la respuesta a la pregunta (básicamente análogo al caso de uso general CU – 007 en segundo plano). La [imagen del chat](#) se expuso anteriormente.
2. El usuario pulsa el botón de pedir ejemplo de respuesta.
3. Esto llega al proxy que espera a la caché o coge la respuesta que ya estaba precargada.
4. Se devuelve la respuesta al usuario.

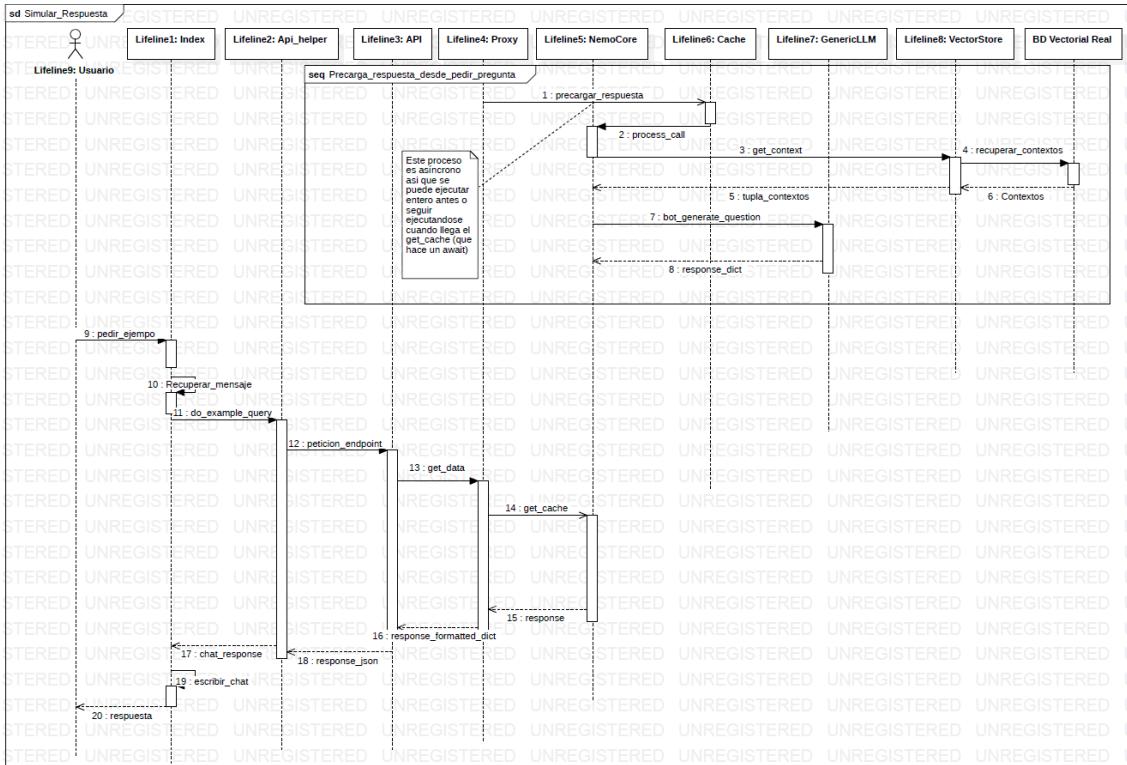


Figura 28: Petición de ejemplo de respuesta

CU – 008

La historia de usuario a implementar en este caso de uso es: “como usuario y/o administrador quiero escoger el modelo de lenguaje que se usa en el sistema para darle flexibilidad”.

Empleamos el módulo del LLM genérico o alguna de las opciones donde se permite cambiar el modelo en los scripts de ejecución tal y como se explica en la [guía de usuario para cargar un modelo diferente](#).

Se puede observar en las respuestas, ya que cada modelo tiene sus propios matices a la hora de responder.

CU – 009

La historia de usuario a implementar en este caso de uso es: “como administrador quiero modificar el flujo de conversación del *bot* para que se acople a mi caso de negocio”.

Se puede emplear la configuración de conversación o las acciones del gestor de comportamientos para modificar estos. Seguir la [guía de desarrollador](#).

CU – 010

La historia de usuario a implementar en este caso de uso es: “como administrador quiero modificar las instrucciones de los *chatbots* para que tengan otro comportamiento”.

Se puede emplear la configuración de los *prompt-templates* o la configuración del gestor de comportamientos para realizar esta tarea. Seguir la [guía de desarrollador](#).

CU – 011

La historia de usuario a implementar en este caso de uso es: “como administrador quiero escoger el contexto general del sistema para que se acople a mi caso de negocio”.

Los documentos situados en *backend/archivos/base* son los que comprenden la base de conocimiento base del entrevistador. Cambiar esta base de conocimiento cambia mucho el tópico de las preguntas que va a realizar el entrevistador.

CU – 012

La historia de usuario a implementar en este caso de uso es: “como administrador quiero administrar el *bias* que introduce la base de datos de ejemplos”.

Empleamos el módulo del RAG o del Gestor de comportamientos para poder habilitar/deshabilitar la capacidad de poder recibir el contexto de preguntas similares.

Se puede observar en caso de generar muchas preguntas con contextos similares, ya que si está habilitado el *similarity context* las respuestas tienen dejes muy parecidos.

CU – 013

La historia de usuario a implementar en este caso de uso es: “como administrador quiero que el cliente pueda subir y usar su propio contexto”.

Como administrador puedes habilitar/deshabilitar la posibilidad de realizar el CU – 005 de subir documentos desde el lado del cliente.

CU – 014

La historia de usuario a implementar en este caso de uso es: “como administrador quiero que se guarde cada vez que el *bot* genera una pregunta para poder usarla como ejemplo”.

Parte del caso de uso CU – 007.1. En este caso, cuando se genera una pregunta se guarda en el *similarity-context* con su contexto para añadirla al contexto de otras preguntas en caso de que sean muy parecidas. Se puede habilitar/deshabilitar según se considere oportuno.

CU – 015

La historia de usuario a implementar en este caso de uso es: “como administrador quiero poder desplegar el sistema en mi máquina en un entorno virtualizado por sencillez”.

Se ofrecen tres *scripts* para instalar Conda, crear un entorno virtual con todas las dependencias y hacer el *run* de la aplicación en ese entorno virtual. Seguir la [guía de desarrollador](#) o de [usuario](#).

CU – 016

La historia de usuario a implementar en este caso de uso es: “como administrador quiero poder desplegar el sistema de manera manual para poder realizar pruebas”.

Se presentan *scripts* para la facilitación del *deployment* de manera manual del usuario. Seguir la [guía de desarrollador](#) o de [usuario..](#)

Gestión de errores

Durante el proceso de ejecución de estos casos de uso se pueden producir errores. Estos errores han sido revisados en el plan de pruebas del proyecto. Los errores son tratados correctamente para el usuario. Por ejemplo:

1. En caso de que la base de datos falle, a la interfaz llega un ID nulo, por lo que sabe que ha habido un error y muestra el mensaje.
2. En caso de que el modelo de lenguaje falle, el propio *chat* avisará por mensaje que ha habido un error.
3. En caso de que realicemos el flujo de conversación de manera desordenada para nuestro caso de negocio (generar pregunta siempre primero) el *chat* nos responderá con “falta una pregunta” pero podremos seguir trabajando sin problema.
4. En caso de que el *chat* falle por un fallo de memoria en la gráfica (por ejemplo) la aplicación no colapsará simplemente responderá el *chat* que ha habido un error. Con recursos computacionales suficientes, ahí es donde una arquitectura de microservicios entraría en juego para lanzar otro *chat* de ayuda.
5. API tiene una función de chequeo de salud, en caso de que haya algún comportamiento extraño poder comprobar si el error está ahí.
6. Tokens o nombres de repositorio inválidos sí que producen un error porque el programa necesita de ellos para funcionar.

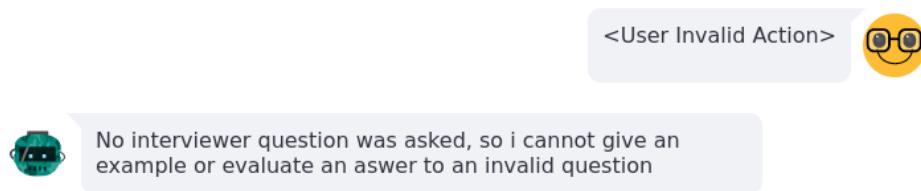


Figura 29: Ejemplo de gestión errores

Tecnologías

A lo largo del documento, se han ido mencionando tecnologías que se han implementado en la aplicación presentada. En este apartado se va a hacer un resumen técnico de dichas tecnologías.

Backend

Para el desarrollo del *backend* y de la mayor parte de la aplicación se ha usado python 3.11 o superiores²⁴. La decisión del uso de lenguaje es obvia, puesto que la mayoría de frameworks de desarrollo de IA están implementados en este desarrollo. Para la API de tipo

²⁴ [Python Release Python 3.11.0 | Python.org](https://www.python.org/)

RESTful se ha usado Flask²⁵ por su simplicidad y por la facilidad de traducción en caso de querer mudarse a Fast API²⁶.

Frontend

Para el desarrollo del *frontend* se ha empleado tecnología basada en Python especializada para hacer aplicaciones de *data science*, streamlit²⁷. La idea de emplear este tipo de herramienta es aplicar un *framework lightweight* que facilita el desarrollo de aplicaciones Web a través de un *wrapper* en Python. Empleando este tipo de *frameworks* conseguimos una regularización en los módulos del programa empleando el mismo lenguaje base para el desarrollo.

En este caso, sobre todo, destacar el uso de *streamlit-chat*, uno de los componentes de streamlit que se pueden usar para las conversaciones de tipo *chat* y que fue empleado en este módulo. A través de este componente podemos delegar toda la construcción del propio *chat* y centrarnos en la personalización del resto de la aplicación web.

Bases de datos

La base de datos SQL se construyó empleando SQLite²⁸ al ser la herramienta más básica para guardar información de este tipo.

RAG

En el apartado de módulos se ha explicado la funcionalidad de las técnicas de RAG. Para poder aplicar estas técnicas necesitamos lo que se llama un VectorStore o una base de datos vectorial. En esta base de datos se guardan los documentos en formato de *embeddings* para poder usarlos en un futuro. En este caso, la herramienta seleccionada es ChromaDB²⁹ que junto con PineCone³⁰ son una de las opciones más escogidas. Además, estas herramientas se pueden integrar con las demás herramientas de IA que vienen a continuación.

HuggingFace

HuggingFace es un entorno muy completo en el mundo de la IA *open-source*. Por un lado, tiene un repositorio *online* donde se tiene acceso a cientos de modelos y *datasets* de uso normalmente libre. Por otro lado, también ofrece una librería de adopción simple con la que el usuario puede realizar tareas varias, desde descargar a un *dataset*, a entrenar y evaluar un modelo usado.

En el entorno de este proyecto, empleamos HuggingFace como repositorio de los modelos *open-source* que acepta nuestro sistema. Esto quiere decir que los modelos que se pueden cargar deben tener el mismo formato de *safe-tensors* que tiene HuggingFace, no que necesariamente tengan que ser modelos de HuggingFace. Aun así, se recomienda descargar los modelos directamente del repositorio oficial.

²⁵ [Welcome to Flask — Flask Documentation \(3.0.x\) \(palletsprojects.com\)](https://flask.palletsprojects.com/en/3.0.x/)

²⁶ [FastAPI \(tiangolo.com\)](https://fastapi.tiangolo.com/)

²⁷ [Streamlit • A faster way to build and share data apps](https://streamlit.io/)

²⁸ [sqlite3 — DB-API 2.0 interface for SQLite databases — Python 3.12.4 documentation](https://sqlite.org/3_12_4.html)

²⁹ [Chroma Docs \(trychroma.com\)](https://trychroma.com/)

³⁰ [Learn | Pinecone](https://pinecone.io/)

Langchain

Si bien HuggingFace da acceso a que el usuario sea capaz de realizar las tareas básicas con los modelos, Langchain es un *framework* paralelo que otorga al desarrollador la posibilidad de simplificar las funciones que implican interactuar con los modelos a través de lo que se llaman *chains*. A través de estas *chains*, el desarrollador es capaz de invocar de maneras determinadas modelos instanciados de HuggingFace de manera recursiva. Además, Langchain ofrece muchos otros servicios (como por ejemplo un RAG simple) aunque en este caso el RAG se hace de manera pseudo-manual.

Un ejemplo de uso de ambas tecnologías juntas sería el siguiente (recuperado y acotado del repositorio oficial de este trabajo):

```
#Instanciamos el modelo con la API de huggingface
model =
transformers.AutoModelForCausalLM.from_pretrained( ...
)
[...]

#Creamos una pipeline de texto con langchain
text_pipeline = pipeline("text-generation",
model=model, ...)

#Hacemos un wrapper para marcar que es un pipeline
#de huggingace
llm = HuggingFacePipeline(pipeline=text_pipeline)
#o
llm =
HuggingFacePipelineCompatible(pipeline=text_pipeline
)

#Cogemos el template del modelo
template = "<s>[INST] <>{sys_p}<>{text}
[/INST]</>

#Le metemos el sysprompt que queramos
template = template.replace("{sys_p}", sysprompt)

# Creamos la prompttemplate indicando que se va a
#sustituir
prompt = PromptTemplate(
    input_variables=[ "text"],
    template=template,
)
#Chain de LangChain
chain = LLMChain(llm=self.llm, prompt=prompt)

#Usamos la chain
await chain.invoke(inputs)[ "text"]"
```

Ejemplo de código 3: Ejemplo de Langchain y HuggingFace

Nemo Guardrails

Uno de los pilares del sistema es el gestor de comportamientos de los *bots* que usamos para usar los diferentes modos del entrevistador. Esta parte del sistema, además, es la que ayuda a personificar los *chatbots* a crear. La herramienta escogida para realizar esto es Nemo GuardRails³¹³².

Nemo es un *framework open-source* creado por NVIDIA que busca facilitar a los desarrolladores el control de configuraciones y flujos en conversaciones con los *chatbots*. El *framework* ofrece muchas características, pudiendo cubrir el funcionamiento de Langchain en caso de que así se desee.

En este caso, no realizamos una integración completa, pero usamos la característica principal del *framework*, que es definir flujos de conversación en ficheros estáticos, para determinar cómo interactúan los elementos.

```
define user express question
    "Topic: ..."

define flow
    user express question
    $resumen = execute summarize(inputs=$last_user_message)
    $answer = execute bot_response(inputs=$resumen)
    bot $answer
```

Ejemplo de código 4: Ejemplo Flujo

La herramienta por dentro funciona de manera similar al RAG, en el sentido que usa la semejanza entre los *embeddings* de ejemplos y el texto que entra para decidir qué camino emplear. En este caso, como se ha mostrado en el apartado de módulos, ofrecemos tanto una

³¹ <https://github.com/NVIDIA/NeMo-Guardrails>

³² <https://docs.nvidia.com/nemo-guardrails/index.html>

configuración de comportamientos *hardcoded* como otra dinámica donde el *bot* toma todas las decisiones.

Deployment

La idea del *deployment* es que el producto construido y entregado sea capaz de usarse en producción de manera rápida. Para ello se ofrece al usuario dos opciones para lanzar la aplicación.

La primera es dar al usuario accesos a scripts de requisitos para que el manualmente lance la aplicación como quiera.

La otra idea es la más realista si hablamos del paso a producción. Esta sería usar el entorno de virtualización Miniconda³³ para crear un entorno virtual arbitrario en el que instalar los paquetes y dependencias de la aplicación de manera rápida con Poetry³⁴ para que el usuario solo tenga que ejecutar un par de scripts y tenga la aplicación funcionando.

De esta manera lograríamos ejecuciones independientes de otros recursos a través del siguiente esquema

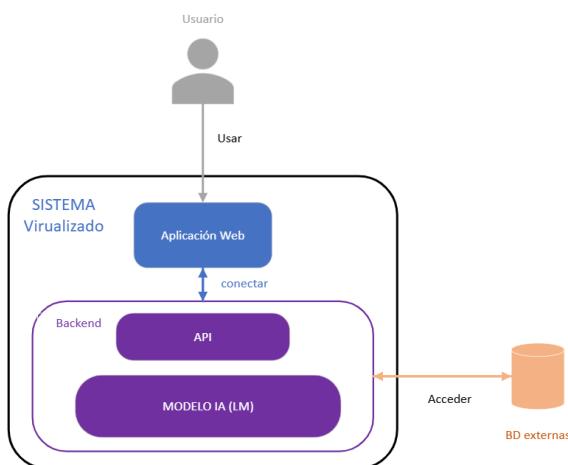


Figura 30: Entorno virtualizado

Otro caso de negocio

Uno de los principales puntos de venta de este sistema es la posibilidad de crear *chatbots* personalizados bajo demanda.

Para probar que esto es posible vamos a realizar, como ejemplo, un *chatbot* para una compañía de seguros y solamente vamos a aplicar el caso de uso para realizar el cálculo del seguro.

Lo primero que hay que hacer es seguir la [guía de desarrollador](#) para deshabilitar las funciones específicas del caso de negocio del TFG y habilitar las funciones genéricas.

Una vez hecho esto, la customización del *bot* se realizaría de la siguiente manera:

1. Lo primero que hacemos es construir documentos PDF donde se almacena la información. Estos se guardan en *backend/archivos/base* y se cargaran en la base de conocimiento al instanciar el *bot*.
2. Vamos a partir de la base de que queremos cambiar el modelo de IA de llama2 a (por ejemplo) phi3 de Microsoft. Realizamos el cambio de modelo editando la línea del script

³³ [Miniconda — Anaconda documentation](#)

³⁴ [Poetry - Python dependency management and packaging made easy \(python-poetry.org\)](#)

correspondiente según la guía de desarrollador (solamente hay que entrar en HuggingFace y copiar el enlace).

3. Ahora vamos a emplear Nemo para construir un flujo muy sencillo y poder implementar el *chatbot*. En este caso definimos que el usuario exprese interés en calcular el seguro.

```
define user express interest in insurance calculation
    "How do I calculate insurance?"
    "Can you help me figure out my insurance costs?"

# Hardcoded response
define bot explain insurance calculation steps
    "To calculate your insurance, follow these steps: ..."

define user express other interest
What insurances do you offer?

flow calculate insurance
    user express interest in insurance calculation
    bot explain insurance calculation steps

flow other response
$answer = execute response($last_user_message)
bot $answer
```

Ejemplo de código 5: Flujo de conversación en otro caso de negocio

Podemos ver en este ejemplo las dos funciones que empleamos nosotros con el *bot*. Por un lado, si hay un mensaje que queremos que el *bot* responda obligatoriamente, podemos hacer que así sea como en este caso con los pasos para calcular el seguro. Si queremos que el *bot* acceda a la base de conocimiento y responda de manera generativa, lo podemos declarar como en “other response” donde lo que hará será responder a la pregunta contrastándolo con la información que tengamos en el documento Base.pdf.

Para ver una versión más completa referir a la [guía de desarrollador](#).

Capítulo 4: Pruebas

En este apartado presentamos una versión reducida de lo que se encuentra en el [plan de pruebas](#) anexo al proyecto. Estas pruebas presentan como objetivo final librarse de fallos eliminables del sistema y poder ofrecer un producto lo más correctamente funcional acercándose así a lo pedido según el objetivo O – 006 (entorno de producción realista).

Las pruebas que se van a realizar van a ser las siguientes:

1. Pruebas Unitarias para ver que el código funciona según lo esperado
2. Pruebas de Integración para ver que se pueden ejecutar los flujos de información estándar con clases interactuando entre sí.
3. Pruebas de extremo a extremo para comprobar que el programa funciona en su completitud
4. Pruebas de usabilidad, en la que varios usuarios validan si la *web app* cumple con los estándares de diseño mínimos
5. Pruebas de despliegue donde se comprobarán los casos de uso de despliegue.

Pruebas Unitarias

Se realiza una comprobación de que los componentes individuales desarrollados como código en el proyecto cumplen con un estándar de calidad. Para se realiza una validación de los elementos del sistema chequeando que cada uno de ellos se comporte dentro de los límites deseados.

Este apartado en su completitud excede el tamaño deseado para una memoria de esta índole; sin embargo, se puede acceder al [plan de pruebas](#) asociado para verlo completo. Se puede ver además en Test/unitary en el repositorio

El software usado para utilizar es *Pytest*³⁵, a través de este podemos realizar las validaciones y verificaciones de manera muy rápida.

Para realizar los tests primero definimos una función que queremos testear con sus salidas y sus entradas y luego implementamos dicha acción en *pytest*. Además, realizamos *mocks*³⁶ (una imitación) de las clases externas que influyen en el comportamiento para ir por el camino que nosotros deseamos. Un ejemplo de tabla y código es el siguiente:

Función	_get_chat_template	
Descripción	Se inicializa el chat template	
Inputs	Expected Output	Test ID
Válido: {un formato messages correcto}	Se asignan el chat template encontrado: Mockeado: "Applied chat template"	test_get_chat_template_with_chat_template
Válido: {No hay chat template}	Autocorrección: Se aplica un chat template por defecto: Mockeado: "Applied chat template"	test_get_chat_template_without_chat_template

³⁵ [pytest: helps you write better programs - pytest documentation](#)

³⁶ [unit testing - What is Mocking? - Stack Overflow](#)

Válido: {Error}	Autocorrección: Se aplica un chat template por defecto: Mockeado: "Applied chat template"	test_get_chat_template_exception_handling
-----------------	--	---

Tabla 29: Ejemplo de tabla para los tests unitarios

```

def test_get_chat_template_with_chat_template(llm):
    llm.tokenizer.chat_template = "Test chat template"
    llm.tokenizer.apply_chat_template.return_value = "Applied chat template"
    messages = [
        {"role": "system", "content": "System message"},
        {"role": "user", "content": "User message"}
    ]
    template = llm._get_chat_template(messages)
    assert template == "Applied chat template"
    llm.tokenizer.apply_chat_template.assert_called_once_with(
        messages,
        tokenize=False,
        add_generation_prompt=True,
    )

def test_get_chat_template_without_chat_template(llm):
    llm.tokenizer.chat_template = None
    [...]

def test_get_chat_template_exception_handling(llm):
    llm.tokenizer.chat_template = "Test chat template"
    llm.tokenizer.apply_chat_template.side_effect = [Exception("Test exception"), "Applied chat template"]
    [...]

```

Ejemplo de código 6: Funciones de testing

Podemos comprobar rápidamente, gracias a la ejecución de `pytest` (existe un script asociado en el repositorio), que todos los tests tienen el estado de pasado. Nótese que los `warnings` marcados son por incompatibilidad entre versiones y no por errores.

```

test_api.py .....
test_api_helper.py .....
test_cache.py ...
test_database.py .....
test_file_reader.py ...
test_ftp_client.py ...
test_ftp_server.py ...
test_generic_llm.py .....
test_main_start.py .
test_nemo_config.py ...
test_nemo_core.py ...
test_number_grabber.py .....
test_parser.py .....
test_prompt_templates.py .....
test_proxy.py .....
test_rag.py .....

```

Figura 31: Ejecución de los tests 1

```

===== 103 passed, 24 warnings in 22.85s =====

```

Figura 32: Ejecución de los tests 2

Pruebas de Integración

Las pruebas de integración sirven para comprobar las funcionalidades definidas por los casos de usos descritos anteriormente. Para ello realizamos flujos de trabajo estándar con las clases interactuando entre sí. Se pueden ver en el apartado de Tests/integration del repositorio.

El proceso de comprobación se puede hacer de una manera estimada a través de Notebooks de Jupyter, en los que se pueden probar las funcionalidades del sistema acopladas, pero sin depender de una ejecución completa. Este proceso, sin embargo, es difícil de contabilizar, puesto que las funcionalidades funcionan o no lo hacen; por ende, que los

notebooks realizados funcionen implican que los casos de uso se pueden hacer y, por ende, que estos tests se pasan.

De nuevo, describir aquí cada uno de los tests realizados queda fuera del alcance por lo que se pueden ver en el documento asociado.

Estos tests se han determinado como pasados.

Pruebas de extremo a extremo

Una prueba de extremo a extremo implica validar los flujos de trabajo de los usuarios. Estas pruebas son empíricas y se realizan validando los flujos descritos en el apartado de [casos de uso de diseño](#) en su completitud. Como realizar estas pruebas implica usar la aplicación y que funcione, cada una de las personas que desplieguen la aplicación pueden validar de manera arbitraria que efectivamente el sistema funciona.

Como la aplicación realiza los objetivos deseados, funciona según lo empleado y aplica todos los requisitos descritos previamente, estas pruebas quedan validadas con el funcionamiento de la aplicación mostrado en el apartado de [diseño](#).

Se realiza tests de pruebas de inferencia del *bot* que se evalúan por los usuarios en el apartado de pruebas de usabilidad.

Pruebas de usabilidad

Una prueba de usabilidad requiere que uno o varios evaluadores interactúen con la aplicación y la valoren en base a su grado de usabilidad³⁷ (la facilidad con la que un usuario puede usar la herramienta para realizar su cometido).

The screenshot shows a survey form with three questions using Likert scales:

- Diseño de la aplicación:** A horizontal scale from 1 (Malo) to 5 (Bueno). The 5th option is selected.
- Simplicidad de la aplicación:** A horizontal scale from 1 (Muy Compleja) to 5 (Muy Simple). The 5th option is selected.
- Otros comentarios (El diseño esta muy limitado por el framework):** A text input field labeled "Texto de respuesta larga".

Figura 33: Ejemplo de preguntas del formulario

El cuestionario para realizar la evaluación se puede observar en su completitud en [el documento asociado](#). A continuación, presentamos la abstracción que sacamos de las pruebas donde podemos ver que, aunque el diseño de la aplicación es sencillo, el usuario promedio designa la aplicación con connotaciones neutras y positivas.

En general la aplicación es fácil de usar y con un diseño bastante sencillo.

³⁷ Usabilidad - Qué es, características, definición y concepto (definicion.de)

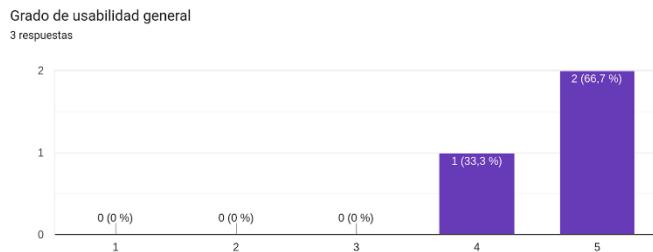


Figura 34: Ejemplo de respuesta de la encuesta de usabilidad

Se hace además una prueba de usabilidad, integridad y calidad de las respuestas de los bots, comparando si el llama2 pequeño (cuantizado a 4bits) con el llama2 entero. Observamos que pese a ser unas 10 veces más lento, las respuestas del modelo grande no son observablemente mejores que las del pequeño a los ojos de los evaluadores

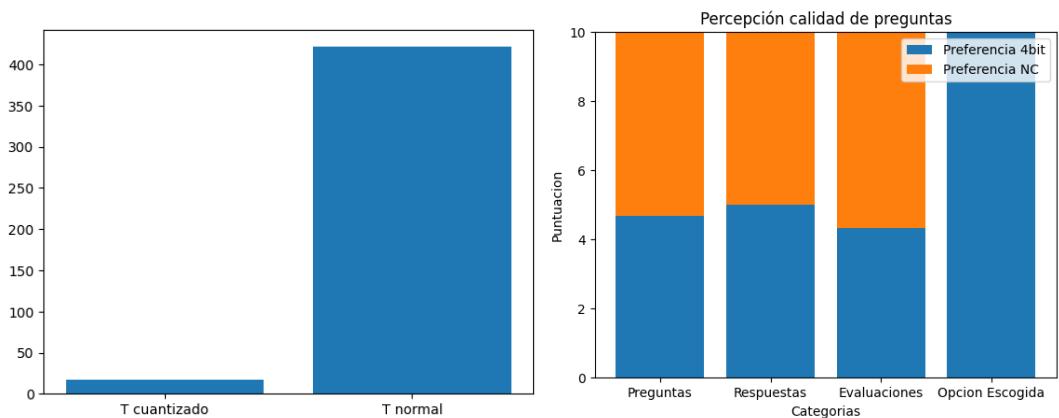


Figura 35: Imagen de coste temporal frente a la calidad apreciada

Pruebas de despliegue

Uno de los objetivos claves del proyecto era el despliegue realista de la aplicación simulando un entorno de producción realista (O – 006). Para hacer esto se ofrece al administrador el módulo de despliegue.

Verificar que el módulo de despliegue funciona requiere verificar que los *scripts* que se le proveen al usuario para realizar los despliegues a producción funcionan, por lo que de nuevo es una tarea de respuesta binaria (o funciona o no lo hace).

En este caso se pasan las pruebas porque se verifica que los *scripts* presentados para los métodos de despliegue funcionan (ejecuta Conda por detrás con el comando *source*).

```
(base) jorge@OJU:~/Escritorio/TFGoficial/usc-aiwantthejob/scripts (Feature/tests)
$ ./run.sh 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
Cargando backend...
Cargando frontend...

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.0.36:8501
```

Figura 36: Script Conda funcionando

Aceptación

Dentro del apartado de criterios de aceptación se especificaba la existencia del criterio A – 001 que exige que el sistema pase las pruebas de manera verificable si son objetivas o que el evaluador determine las pruebas como pasadas para las subjetivas.

Los apartados de pruebas contenidos en este capítulo verifican que dicho criterio de aceptación ha sido pasado con éxito. Además, por la variabilidad de las pruebas, estas también sirven para verificar que los objetivos presentados han sido alcanzados y que los requisitos definidos han sido tenidos en cuenta.

Capítulo 5: Conclusión y Ampliaciones

Conclusiones

El objetivo principal de este proyecto ha sido el desarrollo de un simulador de entrevistas de trabajo virtual. Para ello se ha desarrollado una aplicación *a modo de chatbot* donde el propio *LLM* es el entrevistador. Además, en el núcleo de la aplicación se ha introducido: 1) el uso de técnicas de RAG para que el entrevistador (*LLM*) se ajuste a cualquier tema de entrevista, y 2) un gestor de comportamiento para habilitar las tres funcionalidades básicas del entrevistador (preguntar, evaluar y responder). A mayores, los usuarios tienen acceso a un historial de progreso de las evaluaciones de los entrevistadores. Para emplear a este entrevistador se ha creado un sistema interno que sirve para instanciar *chatbots* para cualquier tema y además establecer cualquier comportamiento que el desarrollador quiera. Se han cumplido todos los subobjetivos planteados:

1. El primer subobjetivo, desarrollar un entrevistador virtual basado en un *LLM* y que emplee técnicas de RAG se ha cumplido con la infraestructura creada en el proyecto.
2. El segundo subobjetivo, la creación de una aplicación realista (demo) con la que se pueda emular un entorno de producción realista, también se ha alcanzado al crear una aplicación web con un *backend* asociado, con un despliegue rápido a través del empleo del entorno virtualizado de Miniconda.
3. El tercer subobjetivo, la cuantificación de las evaluaciones y su progreso, se cumple a través del modo evaluador del entrevistador y el almacenamiento del progreso del usuario y su visualización a través de una gráfica de progreso.

Este proyecto nos permitió explorar el estado del arte de varias tecnologías de última generación de IA y construir con ellas el TFG ***AIWantTheJob***. Con este proyecto, los usuarios tienen acceso bajo demanda a un entrevistador que se puede personalizar para cualquier tema que requieran. Además, los desarrolladores tienen acceso a un *backend* fácilmente personalizable para cambiar el caso de negocio a cualquier tipo de *bot* que se plantee.

Por último, resaltar que la dificultad del desarrollo de esta aplicación ha recaído en el empleo de varias tecnologías novedosas que hay no se habían visto durante el grado, y que por ello han requerido tiempo de estudio y familiarización. Además, la dificultad ha estado en la complejidad del proyecto que integra muchos componentes diferentes.

Ampliaciones

Tal y como se ha visto en este documento, se verifica que el proyecto cumple los objetivos en muy alta medida. Sin embargo, la parte interesante de la conclusión es este punto, ya que se pueden ver los diversos caminos por los que se puede evolucionar la aplicación desarrollada. Destacamos a continuación alguna de las posibles ampliaciones futuras.

Casos de negocio complejos

Si bien en el apartado de diseño se hace alusión a un ejemplo de un [caso de negocio diferente](#) aplicado al sistema creado, la opción de crear un caso de negocio complejo sería el siguiente camino a desarrollar usando de este sistema.

Más generalización

La falta de tiempo y la necesidad de realizar ciertas operaciones (guardar evaluaciones, por ejemplo) causaron que, pese a tener una versión genérica del sistema, la transición entre genérica y no genérica es un poco incómoda ya que hay pasos que requieren de acción manual.

La base de la generalización es muy fácil de emplear y funciona adecuadamente, pero el sistema se puede optimizar de manera general. Lo importante para el proyecto era el caso de negocio más que la generalización.

Mejorar el Frontend

Streamlit es muy cómodo para realizar tareas simples como la propuesta y para el alcance del proyecto era más que suficiente. Sin embargo, es una herramienta muy limitante a la hora de crear entornos personalizados y un desarrollador de *frontend* podría realizar una *Web App* más perfeccionada, en caso de querer ampliar el proyecto.

Arquitectura de microservicios y paso de señales

Si bien en el proyecto se ha empleado virtualización con Conda, la versión más correcta de desplegar esta aplicación si se hace el paso a una arquitectura de microservicios sería a través de un *docker-compose* en el cual se implementa una aplicación de servicios. La estructura del proyecto permite una ampliación a una arquitectura de microservicios completo en caso de que se quiera usar dicha arquitectura.

Como las funciones son mayoritariamente asíncronas, el sistema se podría implementar con métodos de comunicación completamente asíncronos, como el paso de señales, que lo harían mucho más genérico. Aun así, esto requeriría una modificación considerable del código por lo que sería una ampliación considerable.

Otras posibles mejoras

Por último, otras posibles mejoras que se podrían realizar en base al estado del proyecto serían:

1. Emplear más herramientas de las que ofrece Nemo GuardRails, como podría ser su propio *rag* o la conversación *multi-prompt*, para poder crear comportamientos mucho más complejos (agentes).
2. Permitir una clase de LLMs basados en APIs de *chatbots* conocidos (Chatgpt³⁸, Claude³⁹...).
3. Realizar la traducción a tecnologías totalmente diseñadas para producción (cambiar Flask por FastAPI por ejemplo).
4. Modificar el modelo base por un modelo basado en castellano.
5. Perfeccionar el proyecto arreglando posibles inconsistencias de nomenclatura o de arquitectura.

³⁸ <https://chatgpt.com/>

³⁹ <https://claude.ai>

Anexo A: Guías

Para acceder al repositorio: https://github.com/Jorge-A-V/TFG_AIWantTheJob.git. En el siguiente apartado se presentan las guías de uso del proyecto. Notar que las guías son análogas entre sí con excepción del último apartado que es solo para desarrolladores.

Guía del Usuario

USC-AIWANTHEJOB

Este proyecto se corresponde con el TFG **AIWantTheJob** de la USC en combinación con HP SCDS. Es un proyecto que tiene como objetivo realizar el desarrollo de un simulador de entrevistas donde el usuario puede interactuar con un entrevistador basado en IA. Para realizar esta tarea hemos empleado modelos de lenguaje ya existentes (siendo nuestro modelo por defecto llama2). Para lograr la simulación de un entrevistador hemos empleado dos pilares clave:

1. Por un lado, ofrecemos técnicas de RAG para poder dotar al entrevistador de cualquier conocimiento que se quiera. Este conocimiento se puede establecer tanto desde el lado del servidor como del lado del cliente.
2. Por otro lado, empleamos un gestor de comportamientos basado en Nemo GuardRails con el que acotamos el comportamiento de los modos del entrevistador (los cuales se han definido como: entrevistador pregunta, entrevistador evalúa, entrevistador realiza un ejemplo de cómo responde su propia pregunta).

Guía de Uso

Acceso

Si eres un usuario que simplemente quiere acceder a la aplicación *web*, lo más probable es que te pasen una URL con el enlace a la página. Por defecto, este enlace es <http://localhost:8501/> ya que la aplicación suele correr en local.

Como lo más probable es que el propio usuario sea el que está lanzando la aplicación en este caso, se presenta en el apartado de Guía de Despliegue ya que en los métodos de despliegue se hace ya la conexión automáticamente.

Guía de uso | Componentes de la GUI

El punto de acceso de la aplicación es una interfaz de login típica. En esta interfaz podemos conectarnos a nuestra cuenta si ya la tenemos creada o podemos redirigir a la creación de cuenta en caso de que así sea.

AIWantTheJob

[Gitlab](#)

Login

Username

Password

Please enter your credentials to login.

En caso de querer realizar el registro nos situamos de nuevo en una página de registro bastante estandarizada donde podemos crear la cuenta para la que solamente necesitamos usuario y contraseña o podemos volver al login.

AIWantTheJob

[Gitlab](#)

Register

Username

Password

Password_check

Please enter your credentials to register.

Desde el registro antes de acceder al entrevistador, se presenta un tutorial que se corresponde bastante con este apartado para explicar cómo funcionan los componentes en caso de que no sea lo suficientemente claro. Para pasar el tutorial hay un botón claro donde pone continuar para pasar directamente al chat. (En el login se pasa directamente al entrevistador).

AIWantTheJob

[Gitlab](#)

Continue

Tutorial

We present the tutorial for the uso of this app. First as you have seen we have presented you with both

Lo primero que encontramos en la página del chat es el nombre y el enlace del repositorio de GitHub en el que se ha desarrollado. A mayores podemos encontrar dos elementos. Un botón para cerrar sesión y un botón para volver al tutorial descrito en el punto anterior.

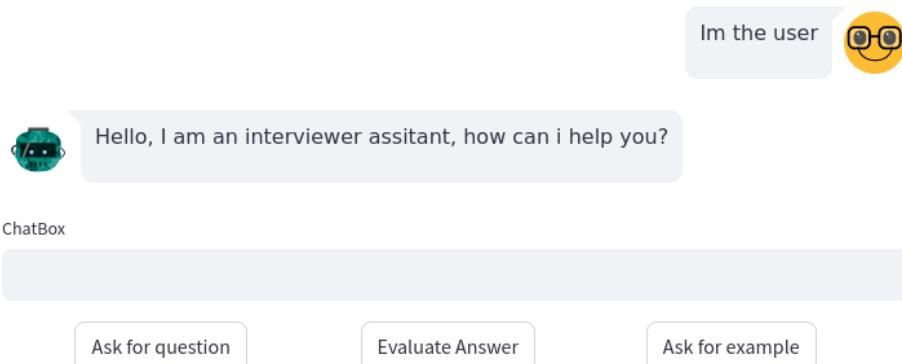
AIWantTheJob

[Gitlab](#)

Tutorial

Logout

Bajando para abajo observamos el chat que se corresponde con el entrevistador. Ofrecemos las tres opciones descritas anteriormente



1. Un botón para generar la pregunta en el que nosotros tenemos que escribir el tema de la pregunta (este pudo ser un extracto de texto de longitud considerable).
2. Un botón para enviar una respuesta y que el entrevistador evalúe dicha respuesta para ver cuál es su calidad (de este punto extraemos las puntuaciones que veremos en un futuro).
3. Un botón con el que podemos pedir que sea el propio entrevistador el que responda la pregunta que ha hecho para ver una posible respuesta. Para este apartado da igual lo que escribamos en el textbox.

Bajando para abajo, vemos un widget con el que podemos subir nosotros conocimiento sobre el que queremos que el entrevistador nos realice preguntas.

File Upload for RAG

This widget allows you to upload files for use in Retrieval-Augmented Generation (RAG). Upload your document here to enhance the AI's knowledge base for more accurate responses.

Suba un archivo



Drag and drop file here

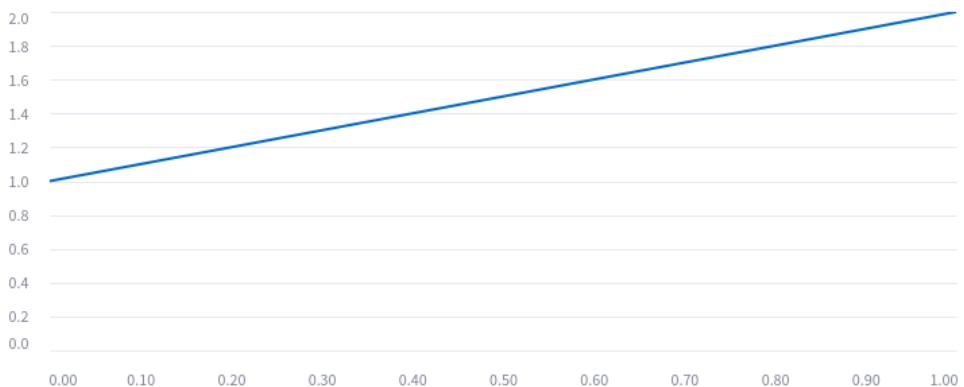
Limit 200MB per file

[Browse files](#)

Y por último podemos ver una gráfica en la que podemos ver el historial de progreso que hemos tenido hasta el momento

Performance Graph [0-5]

This graph displays your performance metrics over time. It shows how your responses have been evaluated throughout the conversation.



Demo

Consideraciones de uso

La aplicación requiere del empleo de un modelo de lenguaje de un tamaño considerable. Los tiempos de respuesta pueden variar entre 5 y 30 segundos aproximadamente.

Mientras está esperando la respuesta se observa un ícono que pone running en la esquina derecha



Guía de Despliegue

Para usar la aplicación se ofrecen 2 métodos diferentes.

Instalación y runeo manual

Para ello primero hay que instalar las dependencias
`pip install --no-cache-dir -r requirements.txt`

Después runearemos el backend

```
cd /backend
python3 main.py "token_de_hugging"
```

Por último runearemos el frontend

```
cd /frontend
streamlit run index.py
```

Entorno virtual

En este modo se instalará Conda y se realizarán las operaciones en un entorno virtual

Nos vamos a scripts

```
cd /scripts
```

Instalamos Conda

```
./conda_install.sh
```

Construimos el entorno

```
./build.sh
```

Lo ejecutamos

```
./run.sh "token_de_hugging"
```

Configuración NAIVE

Si queremos habilitar el comportamiento dinámico del entrevistador hay que realizar los siguientes cambios:

En backend/modelo_ia/NemoConfig.py

```
12:         self.COLANG_CONFIG =
self._set_colang_config("hardcoded_colang.co") # NAIVE COLANG AQUI PARA
CAMBIAR DE CONFIG
```

```
# transformar en
```

```
self.COLANG_CONFIG = self._set_colang_config("naive_colang.co")
```

En backend/modelo_ia/Generic_llm.py

```
# Descomentar
133: self.summarize = self._set_callback("Please summarize the topic of the
following text")
```

En backend/modelo_ia/Nemo_Core.py

```
# Intercambiar los comentarios (comentar la primera descomentar la segunda)
67:
self.register_actions(
```

```

        [self.llm.question_response, self.llm.punctuate_answer,
self.llm.emulate_answer],
        ["bot_response", "user_answer", "bot_answer"],
[True, False, False]
)
"""
# Para el naive
self.register_actions(
    [self.llm.question_response, self.llm.punctuate_answer,
self.llm.emulate_answer, self.llm.summarize],
    ["bot_response", "user_answer", "bot_answer", "summarize"],
[True, False, False, False]
)

```

Guía de desarrollador

Estructura del repositorio

Distribución de archivos del backend

```

| backend/
| --> main.py
| --> setup.py
| --> README.md
| --> api
| --- --> api.py
|
| --> archivos/
|
| --> database/
| --- --> database.py
| --- --> data/
| --- --- --> sqlite databases
|
| --> helpers/
| --- --> ftpserver.py
| --- --> number_grabber.py
| --- --> parser.py
|
| --> modelo_ia/
| --- --> engine_proxy.py
| --- --> generic_llm.py
| --- --> nemo_config.py
| --- --> nemo_core.py
| --- --> prompt_template.py
| --- --> pseudo_cache.py
| --- --> vector_store.py

```

main.py

Fichero principal que es el que ejecuta el programa.

```
python3 main.py "token_hugging"
```

setup.py

Declara los requisitos y los paquetes de los modulos, en caso de querer instalarse simplemente esta parte del programa.

```
cd backend
pip install .
```

README.md

Este fichero

archivos

Contiene el archivo que se va a usar como contexto general para el modelo (server-side). Es donde se suben los archivos del client side, también de contexto.

Modelo IA

engine_proxy.py

Intermediario absoluto del backend. Sirve para conectar cada uno de los componentes entre si
`proxy = Proxy(...)`

```
# invocación al modelo interno
proxy.get_data(texto, argumentos, id)

# acceso (y operaciones) sobre la base de datos
proxy.registrar_usuario(nombre, contraseña)
    .validar_usuario(nombre, contraseña)
    .insertar_valor_array(user_id, valor)
    .recuperar_valores_array(user_id)
```

generic_llm.py

Define de manera genérica un modelo de lenguaje (LLM) de huggingface. Permite al usuario instanciar cualquier tipo de modelo.

Si el modelo es de tipo instruct y tiene `chat_template` habilitado es super simple, simplemente pones el nombre y ya está. Si el modelo NO tiene el `chat_template`, tienes que modificar el fichero en `callback -> template` al template de hugging que aparece en la página descriptiva del modelo:

```
def _set_callback(self, sysprompt,
                  template="copiar aquí template"
)
```

```
Funciones de interés
modelo = LLM(...)

#definir un callback
callback = LLM._set_callback("sysprompt")

#usar el callback
respuesta = callback(pregunta)
```

nemo_config.py

Define la estructura/configuración del engine del Nemo. Tocar la *COLANG CONFIG* si se quiere modificar el flujo de la conversación:

```
config = NemoConfig("nombre modelo")
```

nemo_core.py

Nucleo del sistema de Nemo empleado. Básicamente se usa NemoGuardRails para guiar el flujo de conversación del agente (chatbot). En este archivo se define además como interactúa el LLM con el contexto (RAG).

```
Nemo = NemoCore(...)

# metodo de uso
respuesta = Nemo.processCall(texto, argumento)

# actualización de La base de conocimientos del client-side
Nemo.update_db(path_al_document) # se sube con ftp
```

A la hora de registrar los callbacks dentro del Nemo (*NemoGuardRails.register_action()*), primero le hacemos un setup de todos los contextos que tengamos definidos dentro de la base de datos vectorial

prompt_template.py

Contiene strings absolutos que se corresponden a los sysprompts a emplear dentro del sistema. Se pueden añadir todas las que uno quiera

```
PromptTemplates.nombre_template
```

pseudo_cache.py

Clase pensada para agilizar algunas consultas (ya que uno de los problemas de ejecutar esto en un portatil es la falta de velocidad).

La idea es pre-cargar alguna respuesta mientras el usuario hace tareas en el front para emular así una pre-carga de la cache. En este caso se pre-supone que si el usuario quiere una respuesta,

ahorraremos algunos segundos mientras que si quiere una evaluación, la pre-carga (fallida) no causará delays porque acabará antes de que el usuario escriba su propia respuesta a evaluar.

```
cache = Cache(...)
```

```
# preparar la cache
cache.prepare_example("pregunta")

# acceder al contenido (espera a que se cargue si no ha terminado)
cache.get_context()
```

vector_store.py

Clase base de datos vectorial basada en ChromaDB (uno de los nombres más estándar de la industria.) Chroma usa una base sqlite por detrás.

La idea es que la base de datos tenga 3 colecciones. Una del contexto del cliente, una del contexto del Servidor y una de preguntas.

Cliente y servidor son teóricamente volátiles mientras que la de preguntas es permanente.

```
vs = VectorStore(...)

# subir un archivo a una colección
vs.load_and_embed("archivo", "nombre de la colección", ...)

# acceder a todos los contextos
vs.get_context("texto", ...)

# añadir una pregunta a la colección permanente
vs.add_question("contexto de la pregunta", "pregunta")
```

Si se quiere que cliente y servidor sean persistentes, borrar la siguiente linea:

```
#VectorStore -> Load_and_embed
try:
    # Borramos la colección si existe (no debería ser permanente)
    self.client.delete_collection(index_name)
except Exception:
    pass
```

api

En la carpeta de la API encontramos el desarrollo hecho con Flask de como el cliente se comunica con la propia api

api.py

Define las siguientes rutas:

```
# chequeo de salud de la api
@app.route("/health")

# devuelve los datos del servidor ftp para poder subir archivos a través de
```

```

este
@app.route("/subirarchivo")

# Realiza una consulta al modelo
@app.route("/peticion/<id>")

# Operaciones de la base de datos
@app.route("/login")
@app.route("/register")
@app.route("/array_post")
@app.route("/array_get")

```

Database

Empleamos una base de datos sqlite3 para poder guardar de manera arbitraria los datos de los usuarios

database.py

```

db = DataBase()

# registro de un usuario
db.registrar_usuario(self, nombre: str, password: str) -> id (str)

# validación de un usuario
db.validar_usuario(self, nombre: str, password: str) -> id (str)

# registro de un valor en el array de puntuaciones de un usuario
db.insertar_valor_array(self, identifier: str, value: int) -> List[Any] # 
(array de datos)

# recuperación del array de puntuaciones de un usuario
db.recuperar_valores_array(self, identifier: str) -> List[Any] # array de
datos

```

data/{sqlite databases}

Encontramos aquí las definiciones de la base de datos de clientes y la base de datos que usa chroma por detrás.

Helpers

Encontramos aquí funciones auxiliares que se usan en algún que otro archivo para facilitar la comprensión. Equivalente al fichero "utils" común.

ftpserver.py

Define el ftpserver que se tiene montado en el server-side para poder recibir los archivos que se suban desde el cliente.

```

ftp = FTPserverEu()

# acceder a los datos del servidor
ftp.get_data_as_dic()

# cargar el handler
ftp.load(target="funcion a ejecutar on_load")

# iniciar el ftp_server en un hilo y devolver el hilo
hilo = ftp.start()

```

parser.py

Usa argparse para devolver un diccionario desempaquetable como argumentos para Proxy

```

def parser() ... # returns dict
python3 archivo "token" -n "nombre modelo" -c "cuantización"

```

number_grabber.py

Coge el primer número de un texto para establecerlo como nota de un valor.
Funciona en conjunto con el sysprompt de evaluación definido

```
NumberGrabber.grab_number(texto) -> numero
```

Distribución de contenidos del frontend

Frontend define la aplicación con la que va a interactuar el usuario en tiempo de ejecución. Es una aplicación basada en estados que interactua con la api a través de una interfaz definida.

```

| /frontend
| --> index.py
| --> setup.py
| --> README.md
| --> /helpers
|     --> api_functions.py
|     --> file_reader.py
|     --> ftp_client.py

```

index.py

Define la página web principal con la que va a interactuar el usuario. La página se compromete de tres subsecciones:

Login y Register son las funciones de inicio de sesión y/o registro típicas de una aplicación.
El chatbot viene a continuación.

setup.py

Define el módulo y los requisitos a instalar para esta parte del código

Para instalar manualmente (no recomendado)

```
cd frontend/
pip install .
```

README.md

Este fichero

/helpers/

api_functions.py

Define una interfaz para poder interactuar con la API a través del frontend.

Funciones de interés:

```
interfaz_api = API_helper(...)

# Funciones para pregunta/resuesta/ejemplo
interfaz_api.query_question(json_payload)
    .query_example_response(json_payload)
    .query_for_grading(json_payload)

# Pedir los detalles del ftpserver
interfaz_api.rag_query()

# Funcion de chequeo de salud
interfaz_api.start_health_checker(target=funcion_a_ejecutar)

# Funciones de la bd
interfaz_api.login(user, password)
    .register(user, password)
    .get_array(id)
    .post_array(id, valor)
```

file_reader.py

Sirve para hacer el procesado rápido del pdf a subir desde el lector de ficheros del frontend

Funciones de interés:

```
# Lectura de fichero
FileReader.read(fichero) -> BytesIO
```

ftpclient.py

Sirve para realizar las conexiones con el ftp server (somos el client-side)

Funciones de interés

```

cliente = FTPclient(...)

#Subida de archivo
cliente.upload_file(fichero)

```

Distribución de contenidos Scripts

Esta carpeta contiene los varios scripts a ejecutar para hacer el *build* de la aplicación y que runee perfectamente

```

| scripts/
| --> build.sh
| --> run.sh
| --> stop.sh
| --> conda_install.sh
| --> no_env_run.sh

```

conda_install.sh

Instala el entorno de Miniconda en el portátil para poder usar el método nº2 (entornos virtuales) para la ejecución de la aplicación.

```

./conda_install.sh
exec bash #resetear La terminal

```

build.sh

Sirve para construir el entorno de Conda correcto una vez este Conda instalado.

```
./build.sh
```

run.sh

Hace un run de la aplicación dentro del entorno de Conda instalado para tener acceso a las librerías.

```
./run.sh "token_de_hugging"
```

stop.sh

Busca los procesos por nombre en el árbol de trabajo y los elimina manualmente. (Es posible que streamlit y sus hilos interactúen de manera rara con esto y se tenga que cerrar a mayores la pestaña del navegador).

```
./stop.sh
```

no_env_run.sh

Hace un run de la aplicación directamente sobre los recursos del ordenador en ese momento (es decir no activa ningún tipo de virtualización). Es la manera más sencilla de hacer las cosas.

Requiere de instalar los requisitos:

```
pip install -r requirements.txt
```

Es el fichero que se usa dentro para ejecutar el archivo fuera de Conda

```
./no_env_run.sh "token de hugging"
```

Guía del Desarrollador

USC-AIWANTHEJOB

Este proyecto se corresponde con el TFG **AIWantTheJob** de la USC en combinación con HP SCDS. Es un proyecto que tiene como objetivo realizar el desarrollo de un simulador de entrevistas donde el usuario puede interactuar con un entrevistador basado en IA. Para realizar esta tarea hemos empleado modelos de lenguaje ya existentes (siendo nuestro modelo por defecto llama2). Para lograr la simulación de un entrevistador hemos empleado dos pilares clave:

3. Por un lado, ofrecemos técnicas de RAG para poder dotar al entrevistador de cualquier conocimiento que se quiera. Este conocimiento se puede establecer tanto desde el lado del servidor como del lado del cliente.
4. Por otro lado, empleamos un gestor de comportamientos basado en Nemo GuardRails con el que acotamos el comportamiento de los modos del entrevistador (los cuales se han definido como: entrevistador pregunta, entrevistador evalúa, entrevistador realiza un ejemplo de cómo responde su propia pregunta).

Guía de Uso

Acceso

Si eres un usuario que simplemente quiere acceder a la aplicación web, lo más probable es que te pasen una URL con el enlace a la página. Por defecto, este enlace es <http://localhost:8501/> ya que la aplicación suele correr en local.

Como lo más probable es que el propio usuario sea el que está lanzando la aplicación en este caso, se presenta en el apartado de Guía de Despliegue ya que en los métodos de despliegue se hace ya la conexión automáticamente.

Guía de uso | Componentes de la GUI

El punto de acceso de la aplicación es una interfaz de login típica. En esta interfaz podemos conectarnos a nuestra cuenta si ya la tenemos creada o podemos redirigir a la creación de cuenta en caso de que así sea.

AIWantTheJob

[Gitlab](#)

Login

Username

Password

[or register](#)

Please enter your credentials to login.

En caso de querer realizar el registro nos situamos de nuevo en una página de registro bastante estandarizada donde podemos crear la cuenta para la que solamente necesitamos usuario y contraseña o podemos volver al login.

AIWantTheJob

[Gitlab](#)

Register

Username

Password

Password_check

Register

or login

Please enter your credentials to register.

Desde el registro antes de acceder al entrevistador, se presenta un tutorial que se corresponde bastante con este apartado para explicar cómo funcionan los componentes en caso de que no sea lo suficientemente claro. Para pasar el tutorial hay un botón claro donde pone continuar para pasar directamente al chat. (En el login se pasa directamente al entrevistador).

AIWantTheJob

[Gitlab](#)

Continue

Tutorial

We present the tutorial for the uso of this app. First as you have seen we have presented you with both

Lo primero que encontramos en la página del chat es el nombre y el enlace del repositorio de GitHub en el que se ha desarrollado. A mayores podemos encontrar dos elementos. Un botón para cerrar sesión y un botón para volver al tutorial descrito en el punto anterior.

AIWantTheJob

[Gitlab](#)

[Tutorial](#)

[Logout](#)

Bajando para abajo observamos el chat que se corresponde con el entrevistador. Ofrecemos las tres opciones descritas anteriormente

The screenshot shows a chat interface. At the top right, there is a button labeled "Im the user" with a yellow circular icon containing a smiling face with glasses. Below this, a message from the AI says: "Hello, I am an interviewer assitant, how can i help you?". Below the message is a placeholder text area labeled "ChatBox". At the bottom, there are three buttons: "Ask for question", "Evaluate Answer", and "Ask for example".

4. Un botón para generar la pregunta en el que nosotros tenemos que escribir el tema de la pregunta (este puede ser un extracto de texto de longitud considerable).
5. Un botón para enviar una respuesta y que el entrevistador evalúe dicha respuesta para ver cuál es su calidad (de este punto extraemos las puntuaciones que veremos en un futuro).
6. Un botón con el que podemos pedir que sea el propio entrevistador el que responda la pregunta que ha hecho para ver una posible respuesta. Para este apartado da igual lo que escribamos en el textbox.

Bajando para abajo, vemos un widget con el que podemos subir nosotros conocimiento sobre el que queremos que el entrevistador nos realice preguntas.

File Upload for RAG

This widget allows you to upload files for use in Retrieval-Augmented Generation (RAG). Upload your document here to enhance the AI's knowledge base for more accurate responses.

Suba un archivo



Drag and drop file here

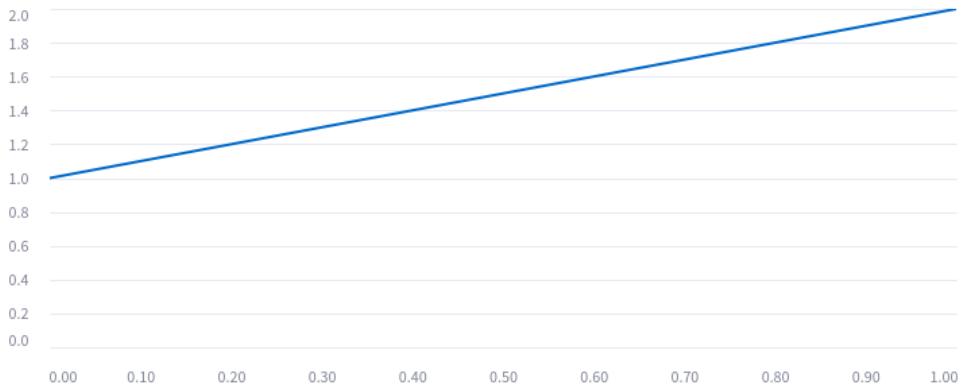
Limit 200MB per file

[Browse files](#)

Y por último podemos ver una gráfica en la que podemos ver el historial de progreso que hemos tenido hasta el momento

Performance Graph [0-5]

This graph displays your performance metrics over time. It shows how your responses have been evaluated throughout the conversation.



Demo

Consideraciones de uso

La aplicación requiere del empleo de un modelo de lenguaje de un tamaño considerable. Los tiempos de respuesta pueden variar entre 5 y 30 segundos aproximadamente.

Mientras está esperando la respuesta se observa un icono que pone running en la esquina derecha



Guía de Despliegue

Para usar la aplicación se ofrecen 2 métodos diferentes.

Instalación y runeo manual

Para ello primero hay que instalar las dependencias

```
pip install --no-cache-dir -r requirements.txt
```

Después runearemos el backend

```
cd /backend
python3 main.py "token_de_hugging"
```

Por último runearemos el frontend

```
cd /frontend
streamlit run index.py
```

Entorno virtual

En este modo se instalará Conda y se realizarán las operaciones en un entorno virtual

Nos vamos a scripts

```
cd /scripts
```

Instalamos Conda

```
./conda_install.sh
```

Construimos el entorno

```
./build.sh
```

Lo ejecutamos

```
./run.sh "token_de_hugging"
```

Configuración NAIVE

Si queremos habilitar el comportamiento dinámico del entrevistador hay que realizar los siguientes cambios:

En backend/modelo_ia/NemoConfig.py

```
12:         self.COLANG_CONFIG =
self._set_colang_config("hardcoded_colang.co") # NAIVE COLANG AQUI PARA
CAMBIAR DE CONFIG
```

transformar en

```
self.COLANG_CONFIG = self._set_colang_config("naive_colang.co")
```

En backend/modelo_ia/Generic_llm.py

```
# Descomentar
133: self.summarize = self._set_callback("Please summarize the topic of the
following text")
```

En backend/modelo_ia/Nemo_Core.py

```
# Intercambiar los comentarios (comentar la primera descomentar la segunda)
```

```
67:
```

```
self.register_actions(
    [self.LLM.question_response, self.LLM.punctuate_answer,
self.LLM.emulate_answer],
    ["bot_response", "user_answer", "bot_answer"],
    [True, False, False]
)
"""
```

```
# Para el naive
```

```
self.register_actions(
    [self.LLM.question_response, self.LLM.punctuate_answer,
self.LLM.emulate_answer, self.LLM.summarize],
    ["bot_response", "user_answer", "bot_answer", "summarize"],
    [True, False, False, False]
)
```

Guía de desarrollador

Estructura del repositorio

Distribución de archivos del backend

```

| backend/
| --> main.py
| --> setup.py
| --> README.md
| --> api
| --- --> api.py

--> archivos/

--> database/
--- --> database.py
--- --> data/
--- --- --> sqlite databases

--> helpers/
--- --> ftpserver.py
--- --> number_grabber.py
--- --> parser.py

--> modelo_ia/
--- --> engine_proxy.py
--- --> generic_llm.py
--- --> nemo_config.py
--- --> nemo_core.py
--- --> prompt_template.py
--- --> pseudo_cache.py
--- --> vector_store.py

```

main.py

Fichero principal que es el que ejecuta el programa.

`python3 main.py "token_hugging"`

setup.py

Declara los requisitos y los paquetes de los modulos, en caso de querer instalarse simplemente esta parte del programa.

`cd backend`
`pip install .`

README.md

Este fichero

archivos

Contiene el archivo que se va a usar como contexto general para el modelo (server-side). Es donde se suben los archivos del client side, también de contexto.

Modelo IA**engine_proxy.py**

Intermediario absoluto del backend. Sirve para conectar cada uno de los componentes entre si
`proxy = Proxy(...)`

```
# invocación al modelo interno
proxy.get_data(texto, argumentos, id)

# acceso (y operaciones) sobre la base de datos
proxy.registrar_usuario(nombre, contraseña)
    .validar_usuario(nombre, contraseña)
    .insertar_valor_array(user_id, valor)
    .recuperar_valores_array(user_id)
```

generic_llm.py

Define de manera genérica un modelo de lenguaje (LLM) de huggingface. Permite al usuario instanciar cualquier tipo de modelo.

Si el modelo es de tipo instruct y tiene `chat_template` habilitado es super simple, simplemente pones el nombre y ya está. Si el modelo NO tiene el `chat_template`, tienes que modificar el fichero en `callback -> template` al template de hugging que aparece en la página descriptiva del modelo:

```
def __set_callback(self, sysprompt,
                  template="copiar aquí template"
)
```

Funciones de interés

```
modelo = LLM(...)
```

```
#definir un callback
callback = LLM.__set_callback("sysprompt")
```

```
#usar el callback
```

```
respuesta = callback(pregunta)
```

nemo_config.py

Define la estructura/configuración del engine del Nemo. Tocar la *COLANG CONFIG* si se quiere modificar el flujo de la conversación:

```
config = NemoConfig("nombre modelo")
```

nemo_core.py

Nucleo del sistema de Nemo empleado. Básicamente se usa NemoGuardRails para guiar el flujo de conversación del agente (chatbot). En este archivo se define además como interactúa el LLM con el contexto (RAG).

```
Nemo = NemoCore(...)

# metodo de uso
respuesta = Nemo.processCall(texto, argumento)

# actualización de la base de conocimientos del client-side
Nemo.update_db(path_al_documento) # se sube con ftp
```

A la hora de registrar los callbacks dentro del Nemo (*NemoGuardRails.register_action()*), primero le hacemos un setup de todos los contextos que tengamos definidos dentro de la base de datos vectorial

prompt_template.py

Contiene strings absolutos que se corresponden a los sysprompts a emplear dentro del sistema. Se pueden añadir todas las que uno quiera

```
PromptTemplates.nombre_template
```

pseudo_cache.py

Clase pensada para agilizar algunas consultas (ya que uno de los problemas de ejecutar esto en un portatil es la falta de velocidad).

La idea es pre-cargar alguna respuesta mientras el usuario hace tareas en el front para emular así una pre-carga de la cache. En este caso se pre-supone que si el usuario quiere una respuesta, ahorraremos algunos segundos mientras que si quiere una evaluación, la pre-carga (fallida) no causará delays porque acabará antes de que el usuario escriba su propia respuesta a evaluar.

```
cache = Cache(...)
```

```
# preparar la cache
```

```
cache.prepare_example("pregunta")
```

```
# acceder al contenido (espera a que se cargue si no ha terminado)
cache.get_context()
```

vector_store.py

Clase base de datos vectorial basada en ChromaDB (uno de los nombres más estándar de la industria.) Chroma usa una base sqlite por detrás.

La idea es que la base de datos tenga 3 colecciones. Una del contexto del cliente, una del contexto del Servidor y una de preguntas.

Cliente y servidor son teóricamente volátiles mientras que la de preguntas es permanente.

```
vs = VectorStore(...)

# subir un archivo a una colección
vs.load_and_embed("archivo", "nombre de la colección", ...)

# acceder a todos los contextos
vs.get_context("texto", ...)

# añadir una pregunta a la colección permanente
vs.add_question("contexto de la pregunta", "pregunta")
```

Si se quiere que cliente y servidor sean persistentes, borrar la siguiente linea:

```
#VectorStore -> Load_and_embed
try:
    # Borramos la colección si existe (no debería ser permanente)
    self.client.delete_collection(index_name)
except Exception:
    pass
```

api

En la carpeta de la API encontramos el desarrollo hecho con Flask de como el cliente se comunica con la propia api

api.py

Define las siguientes rutas:

```
# chequeo de salud de la api
@app.route("/health")

# devuelve los datos del servidor ftp para poder subir archivos a través de este
@app.route("/subirarchivo")

# Realiza una consulta al modelo
@app.route("/peticion/<id>")

# Operaciones de la base de datos
@app.route("/login")
@app.route("/register")
@app.route("/array_post")
```

```
@app.route("/array_get")
```

Database

Empleamos una base de datos sqlite3 para poder guardar de manera arbitraria los datos de los usuarios

database.py

```
db = DataBase()

# registro de un usuario
db.registrar_usuario(self, nombre: str, password: str) -> id (str)

# validación de un usuario
db.validar_usuario(self, nombre: str, password: str) -> id (str)

# registro de un valor en el array de puntuaciones de un usuario
db.insertar_valor_array(self, identifier: str, value: int) -> List[Any] # 
(iarray de datos)

# recuperación del array de puntuaciones de un usuario
db.recuperar_valores_array(self, identifier: str) -> List[Any] # array de
datos
```

data/{sqlite databases}

Encontramos aquí las definiciones de la base de datos de clientes y la base de datos que usa chroma por detrás.

Helpers

Encontramos aquí funciones auxiliares que se usan en algún que otro archivo para facilitar la comprensión. Equivalente al fichero "utils" común.

ftpserver.py

Define el ftpserver que se tiene montado en el server-side para poder recibir los archivos que se suban desde el cliente.

```
ftp = FTPserverEu()

# acceder a Los datos del servidor
ftp.get_data_as_dic()

# cargar el handler
ftp.load(target="funcion a ejecutar on_load")

# iniciar el ftp_server en un hilo y devolver el hilo
hilo = ftp.start()
```

parser.py

Usa argparse para devolver un diccionario desempaquetable como argumentos para *Proxy*

```
def parser() ... # returns dict
python3 archivo "token" -n "nombre modelo" -c "cuantización"
```

number_grabber.py

Coge el primer número de un texto para establecerlo como nota de un valor.
Funciona en conjunto con el sysprompt de evaluación definido

NumberGrabber.grab_number(texto) -> numero

Distribución de contenidos del frontend

Frontend define la aplicación con la que va a interactuar el usuario en tiempo de ejecución. Es una aplicación basada en estados que interactua con la api a través de una interfaz definida.

```
| /frontend
| --> index.py
| --> setup.py
| --> README.md
| --> /helpers
|     --> api_functions.py
|     --> file_reader.py
|     --> ftp_client.py
```

index.py

Define la página web principal con la que va a interactuar el usuario. La página se compromete de tres subsecciones:

Login y Register son las funciones de inicio de sesión y/o registro típicas de una aplicación.
El chatbot viene a continuación.

setup.py

Define el módulo y los requisitos a instalar para esta parte del código
Para instalar manualmente (no recomendado)

```
cd frontend/
pip install .
```

README.md

Este fichero

/helpers/**api_functions.py**

Define una interfaz para poder interactuar con la API a través del frontend.

Funciones de interés:

```
interfaz_api = API_helper(...)

# Funciones para pregunta/resuesta/ ejemplo
interfaz_api.query_question(json_payload)
    .query_example_response(json_payload)
    .query_for_grading(json_payload)

# Pedir los detalles del ftpserver
interfaz_api.rag_query()

# Funcion de chequeo de salud
interfaz_api.start_health_checker(target=funcion_a_ejecutar)

# Funciones de la bd
interfaz_api.login(user, password)
    .register(user, password)
    .get_array(id)
    .post_array(id, valor)
```

file_reader.py

Sirve para hacer el procesado rápido del pdf a subir desde el lector de ficheros del frontend

Funciones de interés:

```
# Lectura de fichero
FileReader.read(fichero) -> BytesIO
```

ftpclient.py

Sirve para realizar las conexiones con el ftp server (somos el client-side)

Funciones de interés

```
cliente = FTPclient(...)
```

```
#Subida de archivo
cliente.upload_file(fichero)
```

Distribución de contenidos Scripts

Esta carpeta contiene los varios scripts a ejecutar para hacer el *build* de la aplicación y que runee perfectamente

```

| scripts/
| --> build.sh
| --> run.sh
| --> stop.sh
| --> conda_install.sh
| --> no_env_run.sh

```

conda_install.sh

Instala el entorno de Miniconda en el portátil para poder usar el método nº2 (entornos virtuales) para la ejecución de la aplicación.

```

./conda_install.sh
exec bash #resetear La terminal

```

build.sh

Sirve para construir el entorno de Conda correcto una vez este Conda instalado.

```
./build.sh
```

run.sh

Hace un run de la aplicación dentro del entorno de Conda instalado para tener acceso a las librerías.

```
./run.sh "token_de_hugging"
```

stop.sh

Busca los procesos por nombre en el árbol de trabajo y los elimina manualmente. (Es posible que streamlit y sus hilos interactúen de manera rara con esto y se tenga que cerrar a mayores la pestaña del navegador).

```
./stop.sh
```

no_env_run.sh

Hace un run de la aplicación directamente sobre los recursos del ordenador en ese momento (es decir no activa ningún tipo de virtualización). Es la manera más sencilla de hacer las cosas.

Requiere de instalar los requisitos:

```
pip install -r requirements.txt
```

Es el fichero que se usa dentro para ejecutar el archivo fuera de Conda

```
./no_env_run.sh "token de hugging"
```

Creación de un chatbot Diferente

A continuación, presentamos la manera de realizar la creación de un *chatbot* nuevo con el sistema. Vamos a crear un *chatbot* para ayudar a la creación de seguros.

Cambiar funciones

Lo primero que tenemos que hacer es editar en:

frontend/index.py

COMENTAMOS ESTO

462: comentario = """

Esto es lo específico

"""

```
def send_query(petition: str) -> None:  
    [...]
```

495

DESCOMENTAMOS ESTO

510:

comentario = """

Esto es lo genérico

```
def send_query() -> None:  
    "Realiza una petición genérica"
```

```
        st.session_state.past.append(user_input)  
        output =  
        asyncio.run(st.session_state.api_helper.general_query(prepare_query(user_in  
ut)))  
        st.session_state["input"] = ""  
        st.session_state["generated"].append(output)
```

```
_, col1, _ = st.columns([1,2,1])  
with col1:  
    crear_boton_estandar("Envía Pregunta", key="b_pregunta",  
on_click=send_query, disabled=st.session_state.get("disabled", True))
```

```
"""

```

También editamos el endpoint de peticiones:

backend/api/api.py

```
#COMENTAMOS
48:
async def peticion(id): # <- específica
    [...]
    print(f"datos procesados {datos}")
    return jsonify(datos), 200 #devolvemos Los datos en json y status OK

# DESCOMENTAMOS
76:
async def peticion(id):
    datos = {
        "id": id,
    }
    pregunta = request.args.get("pregunta")
    if pregunta:
        datos["pregunta"] = pregunta
        response = await system_proxy.get_data(pregunta, args=None, id=id)
        datos.update(response)
    else:
        datos["respuesta"] = "pregunta vacia"
    return jsonify(datos), 200 #devolvemos Los datos en json y status OK
```

Cambios en el modelo

Podemos realizar cambios en el modelo a través de los argumentos de los scripts o de GenericLLM

Para los comportamientos

Nos vamos a la configuración de tipo COLANG del Nemo y realizamos los cambios.

Aquí tenemos que pensar que el bot debe tener un ejemplo de lo que tiene que buscar (una frase con la que comparar la intención) para poder realizar sus cálculos.

Aquí hay un ejemplo de una respuesta en la que nosotros establecemos la respuesta exacta y donde nosotros establecemos una respuesta generativa. (Estas dos opciones serán lo más típico a usar, para cosas más complejas referir a la documentación de Nemo).

```
define user express interest in insurance calculation
"How do I calculate insurance?"
"Can you help me figure out my insurance costs?"

# Hardcoded response
define bot explain insurance calculation steps
"To calculate your insurance, follow these steps: ... "

define user express other interest
```

What insurances do you offer?

```
# Hardcoded response
flow calculate insurance
    user express interest in insurance calculation
    bot explain insurance calculation steps

# El Bot usa el LLM y el RAG de la aseguradora para responder
flow other response
    $answer = execute response($last_user_message)
    bot $answer
```

Podemos ver en este ejemplo las dos funciones que empleamos nosotros con el bot. Por un lado, si hay un mensaje que queremos que el bot responda si o sí, podemos hacer que así sea como en este caso con los pasos para calcular el seguro. Si queremos que el bot acceda a la base de conocimiento y responda de manera generativa, lo podemos declarar como en “other response” donde lo que hará será responder a la pregunta contrastándolo con la información que tengamos en la base de conocimiento

Base de conocimiento

Todos los documentos pdf que pongamos en backend/archivos/base/serán incluídos en la base de conocimiento

Conclusión

A través de estos pasos podemos realizar la customización de un bot de manera muy sencilla

Bibliografía

- Anaconda. (2018). *Miniconda — Anaconda documentation*. Docs.anaconda.com. Recuperado el 2 de julio de 2024 de <https://docs.anaconda.com/miniconda/>.
- Anthropic. (2023). *Claude*. Claude.ai. Recuperado el 2 de julio de 2024 de <https://claude.ai>
- Chroma. (n.d.).  Home | Chroma. [Www.trychroma.com](http://www.trychroma.com). Recuperado el 2 de julio de 2024 de <https://docs.trychroma.com/>
- FastAPI. (n.d.). *FastAPI*. Fastapi.tiangolo.com. Recuperado el 2 de julio de 2024 de <https://fastapi.tiangolo.com/>
- Flask. (2010). *Welcome to Flask — Flask Documentation (3.0.x)*. Flask.palletsprojects.com. Recuperado el 2 de julio de 2024 de <https://flask.palletsprojects.com/en/3.0.x/>
- Gahman, N., & Elangovan, V. (2023, April 3). *A Comparison of Document Similarity Algorithms*. ArXiv.org. <https://doi.org/10.48550/arXiv.2304.01330>
- Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., & Wang, H. (2023, December 18). *Retrieval-Augmented Generation for Large Language Models: A Survey*. ArXiv.org. <https://doi.org/10.48550/arXiv.2312.10997>
- Google. (2024). *Herramientas de aprendizaje y soluciones educativas*. Google for Education. Recuperado el 2 de julio de 2024 de https://edu.google.com/intl/ALL_es/workspace-for-education/editions/overview/
- Han, Y., Liu, C., & Wang, P. (2023, October 18). *A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge*. ArXiv.org. <https://doi.org/10.48550/arXiv.2310.11703>
- HP SCDS. (2022). *Observatorio HP 2022-2023*. HP SCDS. Recuperado el 2 de julio de 2024 de <https://hpscdis.com/innovacion/observatorio-tecnologico/observatorio-hp-22-23/>
- Hugging Face. (2024). *Hugging Face – On a mission to solve NLP, one commit at a time*. Recuperado el 2 de julio de 2024 de Huggingface.co. <https://huggingface.co/>
- huggingface. (n.d.). *Templates for Chat Models*. Huggingface.co. Retrieved July 2, 2024, from https://huggingface.co/docs/transformers/main/en/chat_templates
- langchain. (n.d.). *LangChain*. [Www.langchain.com](http://www.langchain.com). Recuperado el 2 de julio de 2024 de <https://www.langchain.com/>
- Learn | Pinecone. (n.d.). [Www.pinecone.io](http://www.pinecone.io). Retrieved July 2, 2024, from Recuperado el 2 de julio de 2024 de <https://www.pinecone.io/learn/>
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2021, April 12). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. ArXiv.org. <https://doi.org/10.48550/arXiv.2005.11401>
- Li, S., Ning, X., Wang, L., Liu, T., Shi, X., Yan, S., Dai, G., Yang, H., & Wang, Y. (2024, February 28). *Evaluating Quantized Large Language Models*. ArXiv.org. <https://doi.org/10.48550/arXiv.2402.18158>

- meta-llama. (2024, April 18). *meta-llama/Meta-Llama-3-8B · Hugging Face*. Huggingface.co. Recuperado el 2 de julio de 2024 de <https://huggingface.co/meta-llama/Meta-Llama-3-8B>
- Mitra, A., Corro, D., Mahajan, S., Codas, A., Simoes, C., Agrawal, S., Chen, X., Razdaibiedina, A., Jones, E., Aggarwal, K., Palangi, H., Zheng, G., Rosset, C., Khanpour, H., & Awadallah, A. (2023). Orca 2: Teaching Small Language Models How to Reason. ArXiv (Cornell University). <https://doi.org/10.48550/arxiv.2311.11045>
- NVIDIA. (2023a). *NVIDIA NeMo Guardrails*. NVIDIA Docs. Recuperado el 2 de julio de 2024 de <https://docs.nvidia.com/nemo-guardrails/index.html>
- NVIDIA. (2023b). *Python Actions — NVIDIA NeMo Guardrails latest documentation*. Docs.nvidia.com. Recuperado el 2 de julio de 2024 de https://docs.nvidia.com/nemo/guardrails/colang_2/language_reference/python-actions.html
- NVIDIA. (2023c, September 22). *NeMo Guardrails*. GitHub. Recuperado el 2 de julio de 2024 de <https://github.com/NVIDIA/NeMo-Guardrails>
- OpenAI. (2024). *ChatGPT*. ChatGPT. Recuperado el 2 de julio de 2024 de <https://chatgpt.com/>
- Pérez Porto, J., & Merino, M. (2023, June 6). *Usabilidad - Definicion.de*. Definición.de. Recuperado el 2 de julio de 2024 de <https://definicion.de/usabilidad/#:~:text=El%20concepto%20proviene%20del%20ing%C3%A9niero%20y%20hace>
- Puerto, H., Tutek, M., Aditya, S., Zhu, X., & Gurevych, I. (2024, February 25). *Code Prompting Elicits Conditional Reasoning Abilities in Text+Code LLMs*. ArXiv.org. <https://doi.org/10.48550/arXiv.2401.10065>
- pytest. (n.d.). *pytest: helps you write better programs — pytest documentation*. Docs.pytest.org. Recuperado el 2 de julio de 2024 de <https://docs.pytest.org/en/8.2.x/>
- Python Software Foundation. (n.d.). *sqlite3 — DB-API 2.0 interface for SQLite databases — Python 3.8.2 documentation*. Docs.python.org. Recuperado el 2 de julio de 2024 de <https://docs.python.org/3/library/sqlite3.html>
- Rebedea, T., Dinu, R., Sreedhar, M., Parisien, C., & Cohen, J. (2023, October 16). *NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails*. ArXiv.org. <https://doi.org/10.48550/arXiv.2310.10501>
- Steck, H., Ekanadham, C., & Kallus, N. (2024). Is Cosine-Similarity of Embeddings Really About Similarity? ArXiv (Cornell University). <https://doi.org/10.1145/3589335.3651526>
- Streamlit • The fastest way to build and share data apps.* (n.d.). Streamlit.io. Recuperado el 2 de julio de 2024 de <https://streamlit.io/>
- Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., & Fuller, B. (2023, July 19). *Llama 2: Open Foundation and Fine-Tuned Chat Models*. ArXiv.org. <https://doi.org/10.48550/arXiv.2307.09288>
- unit testing - What is Mocking?* (2010, May). Stack Overflow. Recuperado el 2 de julio de 2024 de <https://stackoverflow.com/questions/2665812/what-is-mocking>

- USC. (n.d.). *Inicio | Universidade de Santiago de Compostela*. [Www.usc.gal](http://www.usc.gal). Retrieved July 2, 2024, from <https://www.usc.gal/es>
- Vercel. (2018). *Poetry - Python dependency management and packaging made easy*. Python-Poetry.org. Recuperado el 2 de julio de 2024 de <https://python-poetry.org/>
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., & Brew, J. (2020). HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv:1910.03771 [Cs]*, 5. <https://arxiv.org/abs/1910.03771>

Anexo B

Plan de proyecto

Descripción

Este proyecto tiene como objetivo desarrollar un simulador de entrevistas de trabajo general impulsado por inteligencia artificial para preparar candidatos para un puesto específico. Utilizando técnicas de Deep Learning y NLP, el sistema genera escenarios de entrevista realistas y personalizados que brindan retroalimentación instantánea para mejorar las habilidades de los entrevistados.

1. *Diseñar un sistema de simulación de entrevistas de trabajo basado en un modelo generativo. La llamada al modelo empleará técnicas de RAG Retriever-Active Gathering con documentos relevantes a la tarea para mejorar las prestaciones ya existentes del LLM.*
2. *Desarrollar un sistema real (demo) de una combinación Aplicación Web + API para poder realizar pruebas de comportamiento y emular una versión prototípica del producto.*
3. *Desarrollar un sistema que evalúe las capacidades de los entrevistados y las valore e implementar un método de seguimiento del progreso de las actuaciones de los entrevistados.*

Estos objetivos son lo que se van a evaluar y que se tienen que completar dentro de este proyecto. Por cuestiones de formalización, dentro del plan del proyecto, estos objetivos se han subdividido de la siguiente manera (notar que simplemente son divisiones y que los objetivos a alcanzar son los mismos):

Objetivos

A partir de la descripción planteada y, teniendo en cuenta las decisiones tomadas en las primeras reuniones donde se acotaba el plan de proyecto y los criterios de diseño, podemos destacar los siguientes objetivos:

O - 001	Implementar un LM de tipo Q/A
Descripción	Implementar un modelo generativo para poder generar e interactuar con él, simulando entrevistas de trabajo y/o realizando preguntas y respuestas en base a un tópico concreto.
Criterio Éxito	El caso de negocio permite realizar estas tres acciones de manera clara.
O - 002	Dar acceso a RAG al modelo
Descripción	Acompañar al modelo con técnicas de RAG (<i>retrieval and gathering</i>) para poder asignar al modelo contexto y conocimiento de temas determinados “on-demand”.
Criterio Éxito	Los documentos introducidos causan efectos evidentes en el contexto del bot.

O - 003	Implementar una aplicación completa
Descripción	Crear una aplicación <i>full-stack</i> que incluya tanto la <i>web</i> como una API integrada
Criterio Éxito	Existe una separación clara que dice que el proyecto cumple con esta distribución
O - 004	Desarrollar un prototipo real
Descripción	Desarrollar un sistema real (demo) de la aplicación descrita que se pueda usar en situaciones reales
Criterio Éxito	Existen pruebas realistas del caso de negocio.
O - 005	Implementar un <i>deployment</i> rápido
Descripción	Desarrollar una manera de realizar un <i>deployment</i> rápido para el prototipo a través de diferentes entornos de virtualización.
Criterio Éxito	Existen métodos de despliegue rápidos que requieren de menos de 3 decisiones del encargado del <i>deployment</i> .
O - 006	Simular un entorno de producción realista
Descripción	Facilitar tanto al usuario final como a la persona que vaya a realizar el <i>deployment</i> el proceso para simular una producción real.
Criterio Éxito	Existen métodos de <i>deployment</i> que simulan un entorno de producción real (lo que recibiría un cliente)
O - 007	Implementar un sistema de evaluación
Descripción	Habilitar dentro del sistema una cuantificación del progreso dentro de las entrevistas y una manera de visualización para la persona que lo está usando
Criterio Éxito	Existe una manera de mantener el tracking de progreso y el usuario tiene acceso a su historial.
O - 008	Abstracción y customización
Descripción	Abstraer lo máximo posible del sistema, convirtiéndolo así en una herramienta altamente customizable que para el caso de uso se usa como entrevistador pero que tiene como alcance tareas de <i>chatbot</i> especializado genéricas y aplicables a cualquier entorno
Criterio Éxito	Existen maneras de modificar el caso de negocio sin tener que cambiar grandes cantidades de código. Acciones como comentar/quitar comentario sí que se pueden realizar siempre y cuando sean claras.

Alcance del proyecto

El alcance de este proyecto es una aplicación *completa* donde podemos identificar por lo menos cuatro componentes diferentes: *web*, *api*, modelo de IA y una base de datos. A través de este sistema se implementará como caso de uso el *chatbot* como entrevistador y/o preparador de preguntas.

A nivel de negocio, la estructura sobre la cual está construido el sistema es la parte más interesante. Si bien para el caso de uso en concreto se han añadido elementos extras que no siempre son necesarios, en el caso de querer crear *chatbots* personalizados de temas arbitrarios, el sistema propuesto facilita al desarrollador lograr esto de una manera fácil e intuitiva.

A nivel de TFG, la solución presentada es un entrevistador. Para emular una situación de entrevista empleamos un gestor de comportamientos que establece el comportamiento del entrevistador al *bot* (preguntar, evaluar, responder, ...). Para emular el conocimiento particular que se pide, el entrevistador tiene acceso a bases de conocimiento particulares y arbitrarias a través del sistema RAG Naive asociado. Esto significa que, para que el *bot* se adhiera a tópicos de entrevista, permitimos que tenga acceso a documentos de los cuales ha de extraer la información.

A nivel de usuario, el sistema presentado (acotado al caso de uso de un entrevistador) permite realizar las siguientes acciones:

1. Registrarse e iniciar sesión para tener acceso a tu historial de evaluaciones
2. Subir un documento propio con el que el *chatbot* puede tener conocimiento particular de lo que el usuario desee
3. Generar preguntas de entrevista de temas concretos
4. Generar respuestas de ejemplo a las preguntas que te hace el evaluador
5. Recibir una evaluación de la respuesta que el usuario ha dado en base al contexto
6. Ver el historial de evaluaciones de las respuestas

A nivel de desarrollador, la customización de los *chatbots* permite realizar las siguientes variaciones:

1. Cambiar la base de conocimiento del *bot* (en el lado del servidor como en el del cliente)
2. Modificar el modelo que ejecuta el sistema en el *backend* (teniendo a disposición cualquier modelo que tenga formato de HuggingFace)
3. Modificar el flujo de conversación (explicado más en detalle en [Nemo Guardrails](#)) de tal manera que se puedan asignar diferentes tipos de “personalidades (a modo de *sysprompts*) o “tareas” (a modo de *callbacks* o *acciones*) al *bot* en caso de que detecte comportamientos particulares.

Estructura de la aplicación

La idea es separar el papel del usuario (con acceso preferiblemente solo la *url* de la *web app*) del papel del administrador (que puede controlar el *deployment* de la aplicación y que tiene acceso a todo).

Este despliegue modular (estructura de servicios) es muy estándar en el mercado porque permite separar los componentes entre sí, teniendo un acoplamiento relativamente bajo y, como se verá en un [futuro](#), se podría extender a una estructura de microservicios completa.

Lo primero que definimos es como sería un diagrama de componentes del sistema.

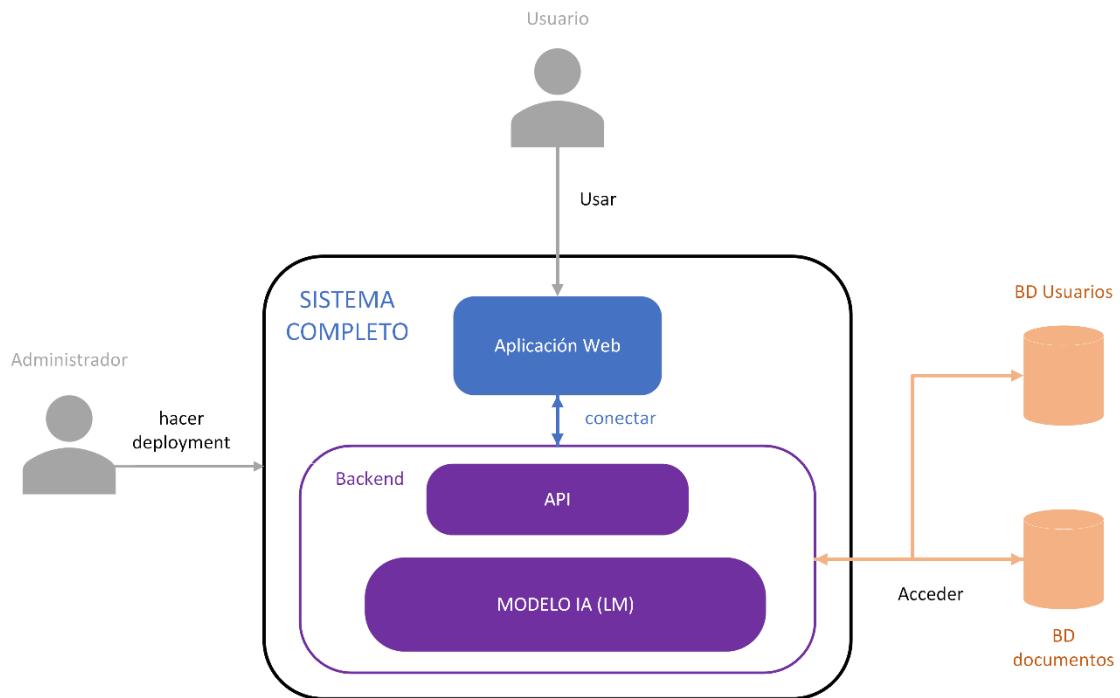


Figura 37: Diagrama de componentes resumido

Mientras que la figura previa muestra la estructuración de los componentes, las interacciones base definidas (IA – LLM – Gestor de comportamientos) previamente funcionaría de la siguiente manera:

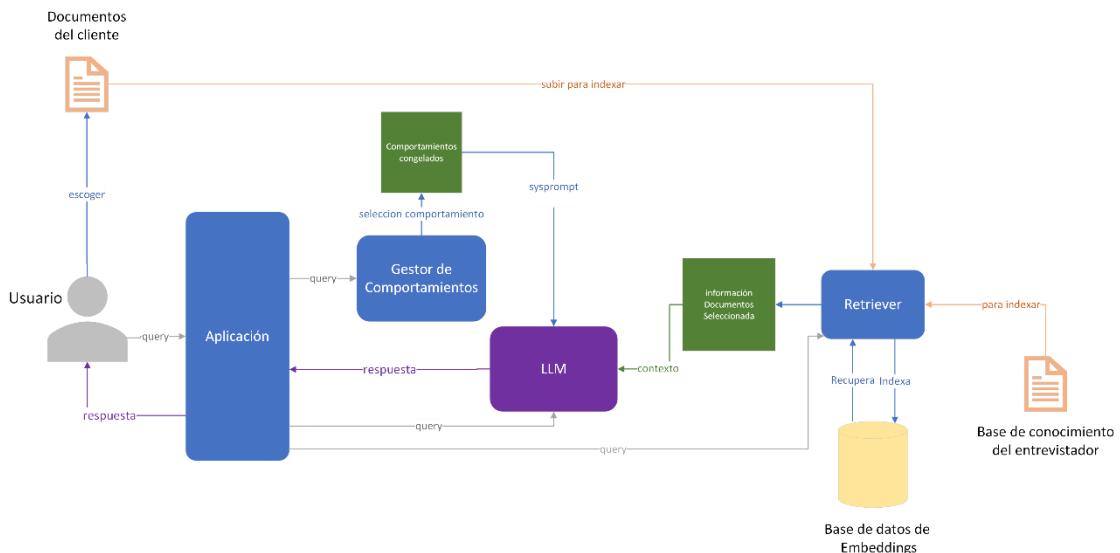


Figura 38: Diagrama de comportamiento RAG y Nemo resumido

Podemos observar el comportamiento de los puntos clave del RAG y del Gestor de comportamientos del LLM (Nemo). El LLM dicta su actuación (modo del examinador) en base a lo que dictamine el gestor de comportamientos en base a la *query*. Por otro lado, los documentos de conocimiento se han guardado como *embeddings* y, a través de la *query*, dictaminamos con el RAG la información (contexto) del LLM que extraemos de esos documentos. El LLM, con su comportamiento definido y con su documento, realiza entonces su tarea de responder de la manera indicada.

Criterios de aceptación

Definimos los siguientes criterios de aceptación del producto a desarrollar:

A - 001	Porcentaje de tests pasados
Descripción	La tasa de acierto de los errores en el plan de pruebas supera el 90%. En estos tests solamente se incluyen aquellos que puedan ser objetivamente medidos como pasados y no pasados.
A - 002	Evaluación empírica del sistema
Descripción	Los tests subjetivos/empíricos de performance del sistema tienen una recepción mayormente positiva. Esto incluye interactuar con el sistema de manera arbitraria para ver si en el día a día se comporta como debería
A - 003	Aprobación de los responsables
Descripción	Las tres personas implicadas en el proyecto dan su aprobación personal al despliegue de la aplicación.
A - 004	Requisitos mínimos
Descripción	Los requisitos mínimos (objetivos, requisitos funcionales, requisitos no funcionales) se cumplen al menos en un mínimo grado.

Limitaciones

L - 001	Fecha de entrega del proyecto
Descripción	El límite de entrega de todos los documentos asociados, el código fuente y la memoria es el 3 de julio de 2024
L - 002	Uso de librerías solo de investigación
Descripción	La naturaleza de desarrollo del proyecto a modo de demo implica usar tecnologías que en un paso a producción habría que cambiar. Ejemplos de esto son Flask (que por políticas habría que mudar a FastAPI para el despliegue comercial) o el modelo de trabajo (hay modelos que son pseudo-privados, con los que puedes investigar o trabajar con ellos para uso recreativo, pero no puedes usar de manera comercial).
L - 003	Límite de tamaño de los modelos para probar
Descripción	El desarrollo del proyecto en el portátil y su entorno limitado dificulta el uso de los modelos óptimos, por lo que la actuación para requisitos computacionales mínimos presentada presenta algo de diferencia frente a la actuación para requisitos recomendados.

L - 004	Límite de la velocidad de los modelos
Descripción	El entorno de desarrollo hace que la percepción del requisito no funcional del tiempo sea relativa. Dependiendo del modo de ejecución, se puede sacrificar precisión por mejoras de tiempo y viceversa, además que diferentes componentes de hardware afectan de manera diferente a la velocidad.

Supuestos

S - 001	La persona que hace el <i>deployment</i> lo hace en linux
Descripción	El <i>deployment</i> de la aplicación se va a probar únicamente en entornos linux, por lo que no necesariamente funcionará con windows.
Impacto	No se asegura que la interacción de los <i>Conda</i> y de CUDA funcione perfectamente con los permisos de windows. Otro sistema operativo podría causar que el sistema no funcione.
S - 002	La persona que hace el despliegue usa equipamiento que cumple con los requisitos de sistema mínimos
Descripción	Para usar los <i>bots</i> se necesitan los 6 GB de VRAM mencionados previamente si se quiere cumplir el requisito de la velocidad de respuesta mínima.
Impacto	Si no se tienen los requisitos mínimos el sistema no va a funcionar con los modelos recomendados. Si bien existen modelos más pequeños, con esos tamaños se degradan mucho las capacidades.

Metodología

La metodología de desarrollo escogida fue una variante de la metodología *agile scrum*. En este caso, como no se tiene *scrummaster*, empleábamos las historias de usuario de Scrum como si fueran tareas en un tablero de *kanban*. Eso sí, las historias se completaron por *sprints*.

De esta manera, nos encontramos con una manera ágil de realizar incrementos a la aplicación a base de prototipos funcionales.

La otra variación frente a *scrum* es que se realizó una documentación un poco más extendida a lo que se esperaría de una metodología ágil (plan de riesgos, plan de pruebas, ...). Esto se debe por saber que se iba a tener que construir esta memoria con esas inclusiones desde el principio del proyecto, por lo que se construyó teniendo este entregable como hito.

Gestión de la configuración, del código y de la documentación

Para gestionar la configuración y el código se empleó la herramienta corporativa de la empresa asociada al TFG: HP SCDS. La herramienta es GitLab, que es un gestor de repositorios muy popular, junto con GitHub.

Para gestionar la configuración se empleó tanto Google Drive como OneDrive, el primero por comodidad y el segundo por ser la herramienta oficial de la Universidad.

El proyecto final se puede encontrar en mi GitHub: https://github.com/Jorge-A-V/TFG_AIWantTheJob.git.

Gestión del tiempo

Al emplearse metodologías ágiles, se empleó el *backlog* de scrum para manejar el tiempo. Cada *sprint* de dos semanas iba realizando tareas y se iban descontando del total previsto. Al construirse de manera incremental, la organización de los *sprints* era por características que se querían añadir al prototipo.

Plan de Riesgos

Lo primero que hacemos es identificar como vamos a evaluar los riesgos:

- **Impacto:** Coste, efecto o duración que tendría una ocurrencia en el proyecto a desarrollar
- **Probabilidad:** representa la expectativa de ocurrencia real del riesgo,
- **Nivel de Exposición al Riesgo:** es el producto **Impacto * Probabilidad**, de manera que será este parámetro (agregado de los anteriores) el que gobierne la gestión de los riesgos del proyecto.

Valoración del Impacto	
Repercusión en Plazo / Esfuerzo / Coste	Impacto
> 25%	Alto
Entre 20% y 25%	Medio Alto
Entre 15% y 20%	Medio
Entre 10% y 15%	Medio Bajo
< 10%	Bajo

Valoración de la Probabilidad	
Ocurrencia del Riesgo	Probabilidad
>= 90% (casi segura)	Alta
Entre 60% y 90% (muy probable)	Media Alta
Entre 40% y 60% (probable)	Media
Entre 10% y 40% (poco probable)	Media Baja
<= 10% (improbable)	Baja

		Nivel de Exposición al Riesgo				
		Probabilidad				
		Alta	Medio Alta	Media	Medio Baja	Baja
Impacto	Alto	Alto	Alto	M. Alto	M. Alto	Medio
	Medio Alto	Alto	M. Alto	M. Alto	Medio	M. Bajo
	Medio	M. Alto	M. Alto	Medio	M. Bajo	M. Bajo
	Medio Bajo	M. Alto	Medio	M. Bajo	M. Bajo	Bajo
	Bajo	Medio	M. Bajo	M. Bajo	Bajo	Bajo

Realizamos una cuantización de la tabla donde bajo es 1 y alto es 5 y permutamos:

Matriz de exposición (cuatizada)						
Impacto	Alto	5	10	15	20	25
	Medio Alto	4	8	12	16	20
	Medio	3	6	9	12	15
	Medio Bajo	2	4	6	8	10
	Bajo	1	2	3	4	5
		Baja	Medio Baja	Media	Medio Alta	Alta
Probabilidad						

Una vez hecho eso podemos transformar los valores en porcentajes:

Matriz de exposición (cuatizada y normalizada)						
Impacto	Alto	20,00%	40,00%	60,00%	80,00%	100,00%
	Medio Alto	16,00%	32,00%	48,00%	64,00%	80,00%
	Medio	12,00%	24,00%	36,00%	48,00%	60,00%
	Medio Bajo	8,00%	16,00%	24,00%	32,00%	40,00%
	Bajo	4,00%	8,00%	12,00%	16,00%	20,00%
		Baja	Medio Baja	Media	Medio Alta	Alta
Probabilidad						

Identificación de Riesgos y Técnicas

Para ver la tabla completa ir al apartado de documentos del repositorio.

Código	Nombre	Descripción		Probabilidad	Impacto	Exposición	Prod. Prob.	Prod. Imp.	Prod.Exp -	Fecha detección	TIPO DE ESTRATEGIA	DESCRIPCIÓN DE LA ACCIÓN	Responsable	Indicador (criterio)	Acción (Plan de Contingencia)	Responsable	Coste y Duración (horas)
RSG.1	Falta de tiempo	La falta de tiempo o la mala organización llevan a problemas temporales que retrasan la ejecución y la entrega de la demo		Medio Baja	Alto	Medio Alta	10%	30%	40.00%	27/3/2024	Mitigar	- Emplear una herramienta de planificación de proyectos (p.ej.: scrum) para mantener un avance constante de los objetivos	Desarrollador - Jorge	Alta cantidad de puntos restantes para cumplir con el SCrum + falta de poco tiempo para finalizar	Reducir el tiempo del proyecto para los objetivos a una meta más aceptable	Desarrollador - Jorge	15h
RSG.2	I/A muy grande en memoria	El modelo de Inteligencia artificial escogido es muy grande en memoria de la GPU y es importante usar "en gpu"		Media	Alto	Medio Alta	50%	30%	60.00%	10/2/2024	Mitigar	- Empleo de modelos de lenguaje de tamaño controlado y optimización como para poder correrlos en la gpu del portátil: 7B+ menos	Desarrollador - Jorge	Cashflows controlados y mejoras en las notificaciones de "fechas límite" en la gpu + "memoria" mejorada al cargarlo en la gpu	Cambiar el modelo de IA en ejecución en la GPU por uno más pequeño o a cambio de un peor performance	Desarrollador - Jorge	4h
RSG.3	Compatibilidad de entornos	El empleo de una cantidad superdiversa de librerías diferentes entre sí puede causar problemas de compatibilidad indirecta (a través de las versiones de las librerías o la ausencia de algunas dependencias).		Medio Baja	Medio	Medio Baja	10%	15%	24.00%	10/2/2024	Mitigar	- Intentar minimizar el número de dependencias siempre que sea posible y mantener las actualizaciones según lo establecido en las librerías.	Desarrollador - Jorge	Errores de ejecución relacionados con las versiones y dependencias de las librerías. "X dependency not found"	Alinear las dependencias entre las librerías	Desarrollador - Jorge	5h
RSG.4	Mal finetuning	El dataset o el formato usado para realizar el finetuning no se adapta bien a la IA y su comportamiento comienza a despegarse		Medio Baja	Alto	Medio Alta	10%	30%	40.00%	25/2/2024	Mitigar	- Seleccionar un dataset adecuado para la tarea. Emplear métodos y herramientas de finetuning adecuados. Probar diferentes tipos de datasets.	Desarrollador - Jorge	Outputs del modelo no se corresponden con las expectativas de "out of the box".	Buscar un dataset mejor, cambiar la estrategia de finetuning y probar otras herramientas externas para el finetuning	Desarrollador - Jorge	15h
RSG.5	Jailbreak de la I/A	La realización de consultas externas a sistemas de información que no son parte del modelo o el modelo se salga de los ralles de comportamiento		Media	Medio Bajo	Medio Baja	50%	10%	24.00%	25/2/2024	Mitigar	- Empiezar a refreguardar para el modelo a empelar. Se emplea Nemo-quadraphant para evitar el jailbreak como punto de partida para limitar las peticiones pedidas	Desarrollador - Jorge	Outputs del modelo a preguntas a las que no sabe responder	Ampliar el número de raliquotes que tiene el sistema	Desarrollador - Jorge	10h
RSG.6	Mucho coste temporal de peticiones	El empleo de un sistema tipo API genera un gran coste temporal de las peticiones que se suele multiplicar por el número de veces que no se ha separado o que el modelo creó crachas por falta de memoria		Medio Baja	Bajo	Baja	30%	5%	8.00%	10/2/2024	Aceptar	- N/A	N/A	N/A	N/A	N/A	N/A
RSG.7	Cambio de scope	Un ajuste en el scope del proyecto que se tienen que hacer implica cambios dentro del propio sistema que están siendo desarrollado		Medio Baja	Alto	Medio Alta	30%	40%	40.00%	25/2/2024	Mitigar	- Se establece claramente el scope del sistema a diseño, construyendole de manera poco restrictiva en caso de tener que añadir más piezas	Desarrollador - Jorge	Cambiar radical en la terminología y el modelo planteado	Ajustar el trabajo al cambio de scope	Desarrollador - Jorge	15h
RSG.8	Crashses de peticiones de la I/A	Debido a la cantidad de peticiones se puede provocar que el modelo se caiga o muere si no se ha separado o que el modelo creó crachas por falta de memoria		Media	Medio Alto	Medio Alta	40%	20%	48.00%	10/2/2024	Mitigar	- Se emplean limitaciones de peticiones desde el front-end. Si se emplean limitaciones de peticiones de peticiones en la API.	Desarrollador - Jorge	Errores de memoria o falta de espacio (p.ej.: memoria de peticiones) o respuesta a request simple pero con errores otros clientes	Aplicar métodos mucho más estrictos de control de peticiones y uso del servidor	Desarrollador - Jorge	3h
RSG.9	Mal performance de la I/A	El modelo integrado dentro del sistema desarrollado actualmente se separa o produce una respuesta que no cumple con la calidad esperada		Medio Baja	Medio Bajo	Medio Baja	35%	10%	16.00%	25/2/2024	Mitigar	- Se realiza la combinación de rales + rag = ejemplos en el prompt para limitar el posible error	Desarrollador - Jorge	Mala respuesta de la IA con respecto a las respuestas de los demás	Probar a eliminar el rag y/o ejemplos en el prompt para corregir un posible error	Desarrollador - Jorge	10h
RSG.10	Testeo Limitado	Debido a la naturaleza de la IA, el desarrollo de pruebas es limitado de las aptitudes del modelo y del alcance real de la aplicación.		Media	Medio Baja	Medio Baja	50%	10%	24.00%	29/3/2024	Mitigar	- Se desarrolla un plan de testeo lo antes posible que se va desarrollando y mejorando durante el tiempo de vida del proyecto	Desarrollador - Jorge	Fallas de dimensiones de datos y de respuesta de pruebas o en el espacio de prueba del resultado	Realizar un plan de testeo o unas pruebas más exhaustivas	Desarrollador - Jorge	10h
RSG.11	Estabilidad de la Demo	La demo desarrollada, pasa a estar en producción como una aplicación completa web + API se va a ejecutar en un entorno controlado y sin problemas de estabilidad en base a los resultados		Medio Baja	Medio Bajo	Medio Baja	20%	10%	16.00%	5/2/2024	Mitigar	- Se emplea un entorno controlado y se miran los logs en caso de que haya un error causado por éllos	Desarrollador - Jorge	Errores relacionados con el uso de los dos sistemas en el mismo ordenador.	Minimizar las posibles variables del entorno en relación a la ejecución. Hacer que el sistema sea más exhaustivo el sistema	Desarrollador - Jorge	5h
RSG.12	Compatibilidad CORS	Debido a que el modelo en local se produce una incompatibilidad CORS al tratar de enviar peticiones de request-response		Medio Alta	Bajo	Medio Baja	70%	5%	16.00%	10/2/2024	Evitar	- Se emplea FLASCOMS en la API para evitar problemas con el CORS	Desarrollador - Jorge	Errores CORS a la hora de realizar las peticiones	N/A	N/A	N/A
RSG.13	Feedback de usuario Limitado	Usando a cliente razon (retrofits, mala planificación, mala ejecución, etc.) para enviar el feedback a otros usuarios para que el modelo se adapte		Medio Baja	Medio Bajo	Medio Baja	30%	10%	16.00%	29/3/2024	Mitigar	- Se planea realizar un procedimiento de feedback de usuario en cuanto se tenga una demo lo suficientemente aceptable	Desarrollador - Jorge	Falta de logs de feedback de usuarios	Establecer el dialogo con una serie de intercambios de ideas y feedback del sistema, tomar acciones en base al feedback	Desarrollador - Jorge	1-2 días
RSG.14	Problemas con la actuación del retinal de RAG	El empleo del RAG dentro del sistema no maneja bien las salidas, sistemas y efectos de la actuación del modelo		Medio Baja	Medio Alto	Media	30%	20%	32.00%	29/3/2024	Mitigar	- Se testeaa el RAG antes de usarse	Desarrollador - Jorge	Errores de precisión con la salida de los mensajes	Modificiar el formato del RAG dentro del sistema, modificar el uso de que se le da a la estrategia de RAG, cambiar el formato de RAG utilizado	Desarrollador - Jorge	10h
RSG.15	Entrada de RAG limitada	La idea del RAG se establece como una entrada de documento (en concreto) y su comportamiento es inadecuado cuando se trabaja con documentos con un formato		Medio Baja	Medio Alto	Media	30%	20%	32.00%	29/3/2024	Mitigar	- Se planea usar diversos tipos de documentos para el RAG durante el proyecto para limitar los errores	Desarrollador - Jorge	Errores de precisión con la salida de los mensajes	Modificiar el formato del RAG dentro del sistema, modificar el uso de que se le da a la estrategia de RAG, cambiar el formato de RAG utilizado	Desarrollador - Jorge	10h

Aplicación de técnicas sobre los riesgos

Para ver la tabla completa ir al apartado de documentos del repositorio. Variamos los porcentajes tras aplicar las acciones de prevención.

Información de Riesgos									
Código	Nombre	Descripción	Nueva Probabilidad	Impacto	Exposición	Nueva Prod. Prob.	Prod. Imp	Nueva Prod.Exp.	Estrategia Empleada
RSG.1	Falta de tiempo	La falta de tiempo o la mala organización llevan a problemas temporales que retrasan o imposibilitan la entrega de la demo	Baja	Alto	Media	5%	30%	20,00%	Mitigar →
RSG.2	IA muy grande en memoria	El modelo de Inteligencia artificial escogido es muy grande en memoria de la GPU y es imposible de usar (en gpu)	Medio Baja	Alto	Medio Alta	10%	30%	40,00%	Mitigar →
RSG.3	Compatibilidad de entornos	El empleo de una cantidad super diversa de librerías dentro del sistema puede causar problemas de compatibilidad directa o indirecta (a través de las versiones de librerías auxiliares en segundo plano)	Baja	Medio	Medio Baja	5%	15%	12,00%	Mitigar →
RSG.4	Mal finetunning	El dataset o el formato usado para realizar el finetunning no es correcto y el modelo no se comporta como se espera de él	Baja	Alto	Media	5%	30%	20,00%	Mitigar →
RSG.5	Jailbreak de la IA	La realización de consultas externas a las características del modelo causa que dicho modelo se salga de los ralles de comportamiento establecidos.	Medio Baja	Medio Bajo	Medio Baja	15%	10%	16,00%	Mitigar →

RSG.6	Mucho coste temporal de peticiones	El empleo de un sistema app web + API produce que las respuestas de las peticiones sea mucho mayor del esperado	Medio Baja	Bajo	Baja	30%	5%	8,00%	Aceptar
RSG.7	Cambio de scope	Un ajuste en el scope del sistema produce que se tengan que realizar muchos cambios dentro del propio sistema que se está desarrollando	Medio Baja	Alto	Medio Alta	20%	40%	40,00%	Mitigar
RSG.8	Crasheo de peticiones de la IA	Debido a la cantidad de peticiones se puede provocar que el comportamiento del modelo no sea el esperado o que el modelo crashee por falta de memoria	Medio Baja	Medio Alto	Media	20%	20%	32,00%	Mitigar
RSG.9	Mal performance de la IA	El modelo integrado dentro del sistema definido no actúa como se espera o produce unos resultados no acordes con la calidad esperada	Medio Baja	Medio Bajo	Medio Baja	25%	10%	16,00%	Mitigar
RSG.10	Testeo Limitado	Debido a la naturaleza de la aplicación se realiza un testeo limitado de las aptitudes del modelo y del alcance real de la aplicación desarrollada	Medio Baja	Medio Bajo	Medio Baja	25%	10%	16,00%	Mitigar
RSG.11	Estabilidad de la Demo	La demo desarrollada, pese a estar pensada para ejecutarse como una aplicación completa web + API se va a ejecutar en local. Esto puede causar problemas de estabilidad en base a los recursos.	Medio Baja	Medio Bajo	Medio Baja	15%	10%	16,00%	Mitigar
RSG.12	Compatibilidad CORS	Debido a probar el modelo en local se produce una incompatibilidad CORS al tratar con la misma dirección de request-response (localhost)	Baja	Bajo	Baja	0%	5%	4,00%	Evitar
RSG.13	Feedback de usuario Limitado	Debido a cierta razón (retrasos, mala planificación, mala exportación, etc...) no se puede enviar la aplicación a otros usuarios para que prueben a ver qué les parece	Medio Baja	Medio Bajo	Medio Baja	20%	10%	16,00%	Mitigar
RSG.14	Problemas con la actuación del retrival de RAG	El empleo de las técnicas de RAG dentro del sistema no produce las salidas del sistema y afecta negativamente en la actuación del modelo	Medio Baja	Medio Alto	Media	20%	20%	32,00%	Mitigar
RSG.15	Entrada de RAG limitada	La idea del RAG se establece como muy rígida (un formato de documento en concreto) y causa comportamiento inesperado e incorrecto con documentos con un formato	Medio Baja	Medio Alto	Media	10%	20%	32,00%	Mitigar

Visión general de la prevención de riesgos

Riesgos e Incidencias - Métricas

Métricas	
Número de riesgos de exposición Alta	0
Número de riesgos de exposición Alto Alta	5
Número de riesgos de exposición Media	2
Número de riesgos de exposición Medio Baja	7
Número de riesgos de exposición Baja	1
Número de incidencias	15

Métricas Modificadas	
Número de riesgos de exposición Alta	0
Número de riesgos de exposición Alto Alta	2
Número de riesgos de exposición Media	5
Número de riesgos de exposición Medio Baja	6
Número de riesgos de exposición Baja	2
Número de incidencias	15

Historial de riesgos

Logs de Riesgos				
Identificador	Fecha identificación	Acción	Encargado	Fecha de cierre
RSG. 12	10/2/2024	Se instala una librería para solucionar los posibles errores CORS (FlaskCORS)	Jorge - Desarrollador	12/2/2024
RSG. 7	06/03/2024	Se descarta el uso de <i>finetunning</i> por la falta de existencia de un corpus lo suficientemente decente y se decanta por el uso de una IA de propósito general LM	Jorge - Desarrollador	06/03/2024
RSG 2	20/04/2024	Llama3 salió con 1B más de parámetros de los esperados, por lo que tenemos que continuar haciendo la demo con Llama2	Jorge - Desarrollador	21/04/2024

Matriz de Trazabilidad de requisitos Completa

Aquí se muestra la versión completa de la matriz de trazabilidad de requisitos y objetivos.

Código Requisito	Nombre	Objetivos	Dependencias
RnF	1 Tiempo de Respuesta rápido	O-004, O-005, O-003	RF-007, RF-015, RF-016, RF-017
RF	MÓDULE DE WEB-APP Y BASE DE DATOS		
RF	1 Realización del login en la aplicación	O-003, O-004, O-006	RI-1, RI-0
RF	2 Realización del registro de un usuario en la aplicación	O-003, O-004, O-006	RI-1, RI-0
RF	3 Recuperación del historial de puntuaciones	O-003, O-004, O-006, O-007	RI-1, RI-0
RF	4 Inserción de un nuevo dato en el historial de puntuaciones	O-003, O-004, O-006, O-007	RI-1, RI-0
RF	MÓDULO DE LA API		
RF	5 Subir un documento como base de conocimiento	O-002, O-006	RF-13
RF	6 Chequeo de Salud	O-003, O-006	
RF	7 Consulta a través de un chatbot	O-001 ,O-002, O-003, O-004, O-006, O-007	RI-2, RI-3, RI-4, RF-11, RF-12, RF-13, RF-14, RF-8, RF-9, RF-10
RF	MÓDULO DE IA		
RF	8 Escoger el modelo de lenguaje	O-001, O-006, O-008	
RF	9 Escoger el flujo que va a emplear el modelo	O-001, O-006, O-008	RF-8
RF	10 Escoger el sys-prompting y los templates que usa el modelo	O-001, O-006, O-008	RF-8
RF	MÓDULO DEL SISTEMA DE RAG		
RF	11 Establecer el contexto general del lado del servidor	O-002, O-006	RI-2
RF	12 Habilitar/DesHabilitar el contexto por similitud	O-002, O-006	RI-4
RF	13 Cargar/Descargar contextos en caliente	O-002, O-006	RI-3
RF	14 Guardar contextos similares para reutilización	O-002, O-006	RI-4
RF	MÓDULO DE DESPLIEGUE		
RF	15 Despliegue en entorno virtual	O-003, O-004, O-005, O-006, O-008	
RF	16 Despliegue a fuerza bruta	O-003, O-005	
RI	1 Información sobre los usuarios	O-003, O-004	
RI	0 Tabla de evaluación X usuario	O-003, O-004, O-007	
RI	2 General Context	O-002	
RI	3 Client Context	O-002	
RI	4 Simmilarity Context	O-002	

Objetivos	Nombre
O 1	Implementar un LM de tipo Q/A
O 2	Dar acceso RAG al modelo
O 3	Implementar una aplicación completa
O 4	Desarrollar un prototipo real
O 5	Implementar un deployment rápido
O 6	Simular un entorno de producción realista
O 7	Implementar un sistema de evaluación
O 8	Abstracción y customización

Plan de Pruebas

Introducción

En este documento se presenta la documentación y la formalización de las pruebas realizadas para el Trabajo de fin de grado (TFG) de tipo B titulado “***AIWantTheJob***”.

En este documento se realizará una explicación de qué pruebas se han realizado y se formalizarán los resultados. El plan de pruebas abarca diferentes niveles de evaluación:

1. Pruebas unitarias a nivel de componente
2. Pruebas de integración entre módulos
3. Pruebas de usabilidad del diseño del *frontend*
4. Pruebas de ejecución *end to end*
5. Pruebas de despliegue

Los resultados del testeo y de la aplicación se realizarán en base a los criterios definidos en el apartado de *criterios de aceptación de las pruebas*.

Características Que Probar

Dada la naturaleza del proyecto (TFG) el alcance de las pruebas a realizar es amplio. Esto implica que se van a probar todos los componentes que forman parte del sistema a nivel unitario y a nivel de integración.

Notas

1. Modularidad y dependencias: el nivel de modularidad del código es alto por lo que hay cierta dependencia entre módulos. Para solventar esa dependencia vamos a emplear Mocks que emulan el proceso.
2. Evaluación de IA: el empleo de técnicas de Inteligencia artificial implica que los resultados también se deberán evaluar en base a criterios semánticos. Eso significa que en las pruebas unitarias se emplea un modelo pequeño que devuelva texto cuando tenga que funcionar y en las pruebas de integración ya se empleará el modelo correcto puesto que en estas pruebas la evaluación semántica se hace manualmente.

Enfoque de las pruebas

Se adoptará un enfoque exhaustivo explorando todos los aspectos del sistema diseñado, empezando a nivel de componente y llegando a nivel de sistema completo.

Cabe destacar que en las pruebas unitarias para este proyecto el acoplamiento entre caja blanca y caja negra es alto por lo que se empezará con las pruebas de caja negra y luego se realizará las pruebas de caja blanca de los procesos que no se hayan validado con la caja negra.

Sistema

Para realizar las pruebas se va a emplear *pytest* para las unitarias, *notebooks* para las de integración y cuestionarios y un formulario de Google con correcciones por otros usuarios para las de usabilidad.

Para ejecutar sobre todo las pruebas de integración se mantienen los requisitos mínimos de sistema mencionados en la memoria principal del sistema, 6 GB de VRAM.

Objetivos

El objetivo principal de este plan de pruebas es verificar la robustez del código proporcionado y verificar que se cumple el criterio de aceptación A - 001.

Los resultados de las pruebas se recopilaron en este documento y se hará un *overview* de los mismos en la memoria a la que este plan va asociado.

Cualquier cambio que estas pruebas generen en el código será realizado según la gestión de configuración definida en la memoria.

Criterios de aceptación de las pruebas

Tal y como se menciona en la memoria y según el criterio de aceptación, las pruebas se pasarán sin se obtiene más del 90% de paso. Esto se aplica principalmente a las pruebas unitarias donde este porcentaje se puede contabilizar. Para el resto de las pruebas, queda a disposición de los integrantes del proyecto decidir si la semántica y la aplicación en si funcionan según lo definido en los estándares.

Criterios de suspensión

En caso de que se detecte que más de un 30% de las pruebas fallan se habilita la opción a el encargado de las pruebas el poder parar la fase de testeo y volver a revisar el código puesto que dichas pruebas ya no se van a pasar.

Organización

Estas pruebas están organizadas de la siguiente manera

1. Pruebas Unitarias
2. Pruebas de Integración
3. Pruebas de Usabilidad
4. Pruebas *end-to-end*
5. Pruebas de despliegue

Entorno

Las pruebas se han realizado con las siguientes características

- Ubuntu 20.04
- Framework de PyTest (7.2.0 o superiores)
- Librerías de Jupyter Notebook

Puede ser que las pruebas necesiten acceso a la red si no se tiene descargados previamente los modelos a emplear sobre todo para las pruebas de integración.

Entregables

Asociadas a la memoria del TFG ***AIWantTheJob*** está este plan de pruebas para simbolizar la realización de las pruebas. Dentro del repositorio del proyecto se encuentra un apartado de Tests donde se pueden verificar todas las pruebas.

Pruebas Unitarias

Como ya se ha mencionado anteriormente, las pruebas unitarias sirven para ver la robustez a nivel de componente. Para realizar estas pruebas vamos a dividir los apartados por módulo, y dentro de cada módulo especificar las pruebas de Caja Negra y las de Caja Blanca.

Esto se debe a que, para nuestro sistema modular, las pruebas de caja negra cubren en su mayoría las pruebas de caja blanca.

En este caso, la mayoría de las funciones tienen pocas opciones a probar y los caminos dependen de las opciones así que se validan juntas. Para ver las pruebas y ejecutarlas referir al apartado Tests/unit del repositorio anexado

Módulo: API

Función	get_health	
Descripción	Verificar el estado de salud de la API	
Inputs	Expected Output	Test ID
Válido: N/A	-Código de estado HTTP 200 -JSON response con campo 'status' que contiene 'available' o 'loading'	test_get_health_status
No Válido: N/A	N/A	N/A

Función	subirarchivo	
Descripción	Subir un archivo al servidor FTP y obtener datos.	
Inputs	Expected Output	Test ID
Válido: N/A	-Código de estado HTTP 200 -JSON response con los datos retornados por el servidor FTP	test_subirarchivo
No Válido: N/A	N/A	N/A

Función	peticion	
Descripción	Hacer una consulta al bot	
Inputs	Expected Output	Test ID
Válido: (args de la request)	-Código de estado HTTP 200 -JSON response con campos 'respuesta' y 'array'	test_peticion
No Válido: N/A	N/A	N/A

Función	login	

Descripción	Autenticar a un usuario	
Inputs	Expected Output	Test ID
Válido: user-password correctos	-Código de estado HTTP 200 -JSON response con campo 'id' conteniendo el ID del usuario	test_login
No Válido: user-password incorrectos	-Código de estado HTTP 200 -JSON response con campo 'id' conteniendo None	test_login_fallo

Función	register	
Descripción	Registrar un nuevo usuario	
Inputs	Expected Output	Test ID
Válido: user-password-password correctos	-Código de estado HTTP 200 -JSON response con campo 'id' conteniendo el ID del usuario	test_register
No Válido: user ya existe	-Código de estado HTTP 200 -JSON response con campo 'id' conteniendo None	test_register_fallo

Función	array_post	
Descripción	Insertar un valor en el historial de puntuaciones de un usuario	
Inputs	Expected Output	Test ID
Válido: (argumentos de petición)	-Código de estado HTTP 200 -JSON response con campo 'array' conteniendo el array actualizado	test_array_post
No Válido: N/A	N/A	N/A

Función	array_get	
Descripción	Obtener el historial de valores de un usuario	
Inputs	Expected Output	Test ID
Válido: (argumentos de petición)	-Código de estado HTTP 200	test_array_get

	-JSON response con campo 'array' contenido el array actualizado	
No Válido: N/A	N/A	N/A

Función	<u>__init__</u>	
Descripción	Iniciar el API Server con sus valores de inicio	
Inputs	Expected Output	Test ID
Válido: valores de inicio	-Flask instance	test_init
No Válido: N/A	N/A	N/A

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_api.py ..... [100%]
=====
===== 10 passed, 15 warnings in 5.84s =====
```

Módulo: API Helper

Función	<u>__init__</u>	
Descripción	Iniciar el API helper con sus valores de inicio	
Inputs	Expected Output	Test ID
Válido: valores de inicio	-Tener las variables internas: id = 1 _ip = "127.0.0.1" _port = "5000" API_URL = "http://127.0.0.1:5000/peticion/" HEALTH_URL = "http://127.0.0.1:5000/health" RAG_URL = "http://127.0.0.1:5000/subirarchivo"	test_init
No Válido: N/A	N/A	N/A

Función	set_id
Descripción	Establecer el ID del usuario en el API Client

Inputs	Expected Output	Test ID
Válido: "id_nuevo"	-Tener las variables internas: id = "id nuevo"	test_set_id
No Válido: N/A	N/A	N/A

Función	api_query	
Descripción	Realizar una consulta a la API	
Inputs	Expected Output	Test ID
Válido: diccionario payload y url	- Diccionario Resultado (Mockeado a {test:data} por ejemplo)	test_api_query
No Válido: N/A (Esta hardcodeado)	N/A	N/A

Función	query_question	
Descripción	Realizar una consulta de tipo pregunta a la API	
Inputs	Expected Output	Test ID
Válido: diccionario payload	- String Resultado (Mockeado a "test answer" por ejemplo)	test_query_question
No Válido: N/A (Esta hardcodeado)	N/A	N/A

Función	query_example_response	
Descripción	Realizar una consulta de tipo ejemplo a la API	
Inputs	Expected Output	Test ID
Válido: diccionario payload	- String Resultado (Mockeado a "test answer" por ejemplo)	test_query_example
No Válido: N/A (Esta hardcodeado)	N/A	N/A

Función	query_for_grading	

Descripción	Realizar una consulta de tipo evaluación a la AP	
Inputs	Expected Output	Test ID
Válido: diccionario payload	<ul style="list-style-type: none"> - String Resultado (Mockeado a "test answer" por ejemplo) - Array de valores (Mockeado) 	test_query_evaluation
No Válido: N/A (Esta hardcodeado)	N/A	N/A

Función	rag_query	
Descripción	Pedir los credenciales para poder subir los documentos a la API	
Inputs	Expected Output	Test ID
Válido: {Vacio}	- Retorna una tupla con los valores de 'address', 'port', 'user', y 'password' de la respuesta de la API	test_rag
No Válido: N/A (Esta hardcodeado)	N/A	N/A

Función	login	
Descripción	Realizar una validación de usuario con el login	
Inputs	Expected Output	Test ID
Válido: credenciales correctos	- Retorna el valor ID	test_login_bien
No Válido: credenciales incorrectos	- Devuelve None	test_login_fallo

Función	register	
Descripción	Realizar un registro de usuario con el register	
Inputs	Expected Output	Test ID
Válido: credenciales correctos	- Retorna el valor ID	test_register_bien
No Válido: credenciales incorrectos	- Devuelve None	test_register_fallo

Función	get_array	
Descripción	Realizar una petición para recuperar el array	
Inputs	Expected Output	Test ID
Válido: id existente	- Retorna el array de puntuaciones del usuario	test_get_array_bien
No Válido: id no existe	- Devuelve [0, (0,0),]	test_get_array_mal

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_api_helper.py ..... [ 100% ]
=====
===== 13 passed in 0.07s =====
```

Módulo: Caché

Función	get_content	
Descripción	Obtener el contenido cacheado y limpiar el caché	
Inputs	Expected Output	Test ID
Válido: se ha cacheado o se está cacheando la respuesta	- Retorna el diccionario respuesta	test_get_content
No Válido: no hay respuesta cacheada	- Devuelve None	test_get_content_none

Función	prepare example	
Descripción	Obtener el contenido cacheado y limpiar el caché	
Inputs	Expected Output	Test ID
Válido: pregunta	- Retorna el diccionario respuesta	test_prepare_example
No Válido: N/A está hardcodeado	N/A	N/A

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_cache.py ...
=====
[100%]
===== 3 passed, 11 warnings in 4.44s =====
```

Módulo: Database

Función	<code>__init__</code> intrínseco con un fixture	
Descripción	Iniciar la BD	
Inputs	Expected Output	Test ID
Válido: {argumentos por defecto}	- Se inicia	fixture
No Válido: N/A está hardcodeado	N/A	N/A

Función	<code>_generate_identifier</code>	
Descripción	Generar un identificador único para un usuario	
Inputs	Expected Output	Test ID
Válido: {credenciales usuario y contraseña}	- un hash que empieza y termina por “	<code>test_generate_identifier</code>
No Válido: N/A está hardcodeado	N/A	N/A

Función	<code>_hash_password</code>	
Descripción	Generar el hash de la contraseña	
Inputs	Expected Output	Test ID
Válido: {contraseña}	- hash de la contraseña y de la sal	<code>test__hash_password</code>
No Válido: N/A está hardcodeado	N/A	N/A

Función	registrar_usuario
Descripción	Registra el usuario en la BD

Inputs	Expected Output	Test ID
Válido: {usuario y contraseña}	- ID del usuario	test_registrar_usuario_success
No Válido: {usuario y contraseña existentes}	- string: "Name probably existing or internal error"	test_registrar_usuario_existing_user
No Válido: {Error}	- string: "Name probably existing or internal error"	test_registrar_usuario_error

Función	validar_usuario	
Descripción	Realiza el login de usuario validando las credenciales en la BD	
Inputs	Expected Output	Test ID
Válido: {usuario y contraseña}	- ID del usuario	test_validar_usuario_success
No Válido: {usuario y contraseña existentes}	- string: "Error on validation"	test_validar_usuario_not_found
No Válido: {Error}	- string: "Error on validation"	test_validar_usuario_fail

Función	insertar_valor_array	
Descripción	inserta un valor nuevo en el historial	
Inputs	Expected Output	Test ID
Válido: {id correcto, valor existente}	- Array de puntuaciones del usuario	test_insertar_valor_array_success
No Válido: {id incorrecto o valor inexistente}	- [(0, 0.0)]	test_insertar_valor_array_error

Función	recuperar_valores_array	
Descripción	inserta un valor nuevo en el historial	
Inputs	Expected Output	Test ID
Válido: {id correcto}	- Array de puntuaciones del usuario	test_recuperar_valores_array_success

No Válido: {id incorrecto}	- [(0, 0.0)]	test_recuperar_valores_array_error
----------------------------	--------------	------------------------------------

Función	close	
Descripción	inserta un valor nuevo en el historial	
Inputs	Expected Output	Test ID
Válido: {Vacio}	- Se cierra la BD	test_close
No Válido: N/A	N/A	N/A

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_database.py ..... [100%]
=====
===== 14 passed in 0.05s =====
```

Módulo: File Reader

Función	read	
Descripción	Convertir los bytes de un fichero en BytesIO	
Inputs	Expected Output	Test ID
Válido: bytes de un fichero	- Retorna el objeto BytesIO	test_read
No Válido: no bytes o bytes incorrectos	- Devuelve un IOError	test_read_error

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_file_reader.py .. [100%]
=====
===== 2 passed in 0.01s =====
```

Módulo: FTPclient

Función	__init__	
Descripción	Inicializar el cliente FTP con los parámetros correctos	
Inputs	Expected Output	Test ID

Válido: {por defecto}	Se inician los parámetros - address: "localhost" - port: "9999" - user: "user" - password: "password"	test_init
No Válido: {Error interno}	Error de FTplib	test_init_connection_error

Función	start	
Descripción	Inicializar el cliente FTP (a runnear)	
Inputs	Expected Output	Test ID
Válido: {por defecto}	Se realizan los dos métodos internos	test_start
No Válido: N/A	N/A	N/A

Función	upload_file	
Descripción	Subir un fichero	
Inputs	Expected Output	Test ID
Válido: {por defecto}	Se sube el fichero correcto (se llaman a las funciones indicadas)	test_upload_file
No Válido: {Error interno}	Error de FTplib	test_upload_file_error

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_ftp_client.py ..... [100%]
=====
===== 5 passed in 0.02s =====
```

Módulo: FTPserverEu

Función	__init__	
Descripción	Inicializar el servidor FTP con los parámetros correctos	
Inputs	Expected Output	Test ID
Válido: {por defecto}	Se inician los parámetros - address: "localhost"	test_init

	- port: "9999" - user: "user" - password: "password"	
No Válido: N/A	N/A	N/A

Función	get_data_as_dic	
Descripción	Inicializar el cliente FTP con los parámetros correctos	
Inputs	Expected Output	Test ID
Válido: {por defecto}	Devuelve un diccionario: - address: "localhost" - port: "9999" - user: "user" - password: "password"	test_get_data_as_dic
No Válido: N/A	N/A	N/A

Función	load	
Descripción	Cargar el servidor con un target	
Inputs	Expected Output	Test ID
Válido: {por defecto}	Se llama a la función indicada	test_load
No Válido: {Error interno}	FTPServer Error	test_load_error

Función	start	
Descripción	Inicializar el servidor FTP (a runnear)	
Inputs	Expected Output	Test ID
Válido: {por defecto}	Se crea el hilo	test_start
No Válido: N/A	N/A	N/A

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_ftp_server.py ..... [ 100% ]
===== 5 passed, 4 warnings in 0.02s =====
```

Módulo: LLM (Generic LLM)

Función	__init__	
Descripción	Inicializar el LLM con los parámetros correctos	
Inputs	Expected Output	Test ID
Válido: {por defecto}	Se llama a la función de creación (AutoModelFromPretrained)	test_llm_initialization
Válido: {cuantizacion 4bit manual}	Se llama a la función de creación (AutoModelFromPretrained)	test_llm_start_4bit
Válido: {cuantizacion 8bit manual}	Se llama a la función de creación (AutoModelFromPretrained)	test_llm_start_8bit
Válido: {No cuantizacion manual}	Se llama a la función de creación (AutoModelFromPretrained)	test_llm_start_no_cuant
No Válido: {Error}	Error	N/A

Función	_set_default_callbacks	
Descripción	Configurar los callbacks por defecto	
Inputs	Expected Output	Test ID
Válido: {por defecto}	Se asignan los 3 callbacks y son Callables	test_llm_set_default_callbacks
No Válido: {N/A}	N/A	N/A

Función	_set_callback	
Descripción	Se crea un callback	
Inputs	Expected Output	Test ID
Válido: {un sysprompt}	Se asignan el callback	test_llm_set_callback
No Válido: {N/A}	N/A	N/A

Función	_get_chat_template	
Descripción	Se inicializa el chat template	
Inputs	Expected Output	Test ID
Válido: {un formato messages correcto}	Se asignan el chat template encontrado: Mockeado: "Applied chat template"	test_get_chat_template_with_chat_template
Válido: {No hay chat template}	Autocorrección: Se aplica un chat template por defecto: Mockeado: "Applied chat template"	test_get_chat_template_without_chat_template
Válido: {Error}	Autocorrección: Se aplica un chat template por defecto: Mockeado: "Applied chat template"	test_get_chat_template_exception_handling

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_generic_llm.py ..... [100%]
=====
===== 9 passed, 11 warnings in 5.75s =====
```

Módulo: main

Función	main	
Descripción	Inicializa todo el backend	
Inputs	Expected Output	Test ID
Válido: {por defecto}	Se llaman a todas las funciones internas	test_main_script
No Válido: N/A	N/A	N/A

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_main_start.py . [100%]
=====
===== 1 passed, 15 warnings in 4.51s =====
```

Módulo: Nemo Config

Función	__init__	

Descripción	Inicializar e NemoConfig (valida los 2 métodos internos)	
Inputs	Expected Output	Test ID
Válido: {por defecto}	Se realizan los dos métodos internos	test_init
No Válido: N/A	N/A	N/A

Función	_set_colang_config (error)	
Descripción	Instanciar la configuración Colang	
Inputs	Expected Output	Test ID
Válido: validado test_init	N/A	N/A
No Válido: {fichero_incorrecto}	FileNotFoundException	test_set_colang_config_error

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_nemo_config.py .. [100%]
=====
===== 2 passed in 0.01s =====
```

Módulo: Nemo Core

Función	__init__	
Descripción	Inicializar e Nemo Core	
Inputs	Expected Output	Test ID
Válido: {por defecto}	Variables internas esperadas: - nemo_core.db == mock_vector_store - nemo_core.nemo_config == mock_nemo_config - nemo_core.LLM == mock_llm - nemo_core.rails is not None	test_init
No Válido: N/A	N/A	N/A

Función	register_action
Descripción	Registra una acción en el sistema de rieles (rails)

Inputs	Expected Output	Test ID
Válido: {mock_callback, test_action}	nemo_core.rails.register_action es llamado una vez	test_register_action
No Válido: N/A	N/A	N/A

Función	register_actions	
Descripción	Registra múltiples acciones en el sistema.	
Inputs	Expected Output	Test ID
Válido: {mock_callback, test_action, array de are_questions}	nemo_core.rails.register_action es llamado tres veces	test_register_actions
No Válido: N/A	N/A	N/A

Función	_set_context	
Descripción	Establece el contexto para el procesamiento.	
Inputs	Expected Output	Test ID
Válido: {mock_context}	El resultado contiene "global_context", "client_context", y "other_context"	test_set_context
No Válido: N/A	N/A	N/A

Función	_get_complete_args	
Descripción	Obtiene los argumentos completos para diferentes tipos de contenido. Inputs válidos: - "question": Para obtener argumentos de pregunta. - "answer": Para obtener argumentos de respuesta. - "example": Para obtener argumentos de ejemplo.	
Inputs	Expected Output	Test ID
Válido: {"question"}	MOCK: {"content": {"question": True, "answer": False, "example": False}}	test_get_complete_args
Válido: {"answer"}	MOCK: {"content": {"question": False, "answer": True, "example": False}}	test_get_complete_args

Válido: {"example"}	MOCK: {"content": {"question": False, "answer": False, "example": True}}	test_get_complete _args
---------------------	--	----------------------------

Función	processCall	
Descripción	Procesa una llamada al sistema	
Inputs	Expected Output	Test ID
Válido: {mock_context}	MOCK: {"content": "Generated response"}	test_processCall
No Válido: N/A	N/A	N/A

Función	update_db	
Descripción	Actualiza la base de datos con un nuevo documento	
Inputs	Expected Output	Test ID
Válido: {mock_documenttt}	- nemo_core.db.load_and_embed es llamado con "test_document.txt" - os.remove es llamado con "test_document.txt"	test_update_db
No Válido: N/A	N/A	N/A

Función	_get_context	
Descripción	Obtiene el contexto para un texto dado	
Inputs	Expected Output	Test ID
Válido: {mock_texttt}	MOCK: ("global", "client", "other")	test_get_context
No Válido: N/A	N/A	N/A

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_nemo_core.py ..... [100%]
=====
===== 8 passed, 11 warnings in 4.90s =====
```

Módulo: NumberGrabber

Función	grab_number
---------	-------------

Descripción	Extrae el primer número entero de una cadena de texto		
Inputs	Expected Output	Test ID	
Válido: {"The answer is 42"}	42	test_grab_un_numero	
Válido: {"There are 3 apples and 5 oranges"}	3	test_grab_solo_primer_numero	
Válido: {"There are no numbers here"}	0	test_grab_sin_numeros	
Válido {"The temperature is -5 degrees"}	0	test_grab_number_negativo	
Válido {"The price is 10.99 dollars"}	0	test_grab_number_decimal	
Válido {"("")"}	0	test_grab_number_string_vacio	

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_number_grabber.py ..... [100%]
=====
===== 6 passed in 0.01s =====
```

Módulo: Parser

Función	parser	
Descripción	Analiza los argumentos de línea de comando.	
Inputs	Expected Output	Test ID
Válido: ['script_name', 'test_token']	{'token': 'test_token', 'model_name': 'meta-llama/Llama-2-7b-chat-hf', 'cuantization': '4bit'}	test_parser_por_efecto
Válido: ['script_name', 'test_token', '-n', 'custom_model', '-c', '8bit']	{'token': 'test_token', 'model_name': 'custom_model', 'cuantization': '8bit'}	test_parser_valores_custom
No Válido: ["script name"]	Lanza SystemExit	test_parser_sin_token

No Válido: ['script_name', 'test_token', '-- invalid_arg', 'value']	Lanza SystemExit	test_parser_argumento_equivocado
--	------------------	----------------------------------

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_parser.py .... [100%]
=====
===== 4 passed in 0.01s =====
```

Módulo: PromptTemplates

Función	Verificación Atributos	
Descripción	Verifica la existencia y tipo de los atributos de la clase PromptTemplates.	
Inputs	Expected Output	Test ID
Válido: {Vacío}	<ul style="list-style-type: none"> - PromptTemplates tiene el atributo question_sysprompt_template - question_sysprompt_template es una cadena de texto 	test_entrevistador_modo_pregunta
Válido: {Vacío}	<ul style="list-style-type: none"> - PromptTemplates tiene el atributo give_example_output_sysprompt_template - give_example_output_sysprompt_template es una cadena de texto 	test_entrevistador_modo_ejemplo
Válido: {Vacío}	<ul style="list-style-type: none"> - PromptTemplates tiene el atributo punctuate_answer_sysprompt_template - punctuate_answer_sysprompt_template es una cadena de texto 	test_entrevistador_modo_evaluacion
Válido: {Vacío}	<ul style="list-style-type: none"> - PromptTemplates tiene el atributo phi1template - phi1template es una cadena de texto 	test_template_de_phi

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_prompt_templates.py .... [100%]
=====
===== 4 passed in 0.01s =====
```

Módulo: Proxy

Función	__init__	
Descripción	Inicializa la clase Proxy.	
Inputs	Expected Output	Test ID
Válido: {por defecto}	Variables internas esperadas: - proxy.database is not None - proxy.llm is not None - proxy.nemo_system is not None - proxy.cache is not None	test_init
No Válido: N/A	N/A	N/A

Función	get_data	
Descripción	Inicializa la clase Proxy.	
Inputs	Expected Output	Test ID
Válido: {MOCK: "test", "question", "user1"}	MOCK: {"respuesta": "Generated question"}	test_get_data_question
Válido: {MOCK: "test", "example", "user1"}	MOCK: {"respuesta": "Example response"}	test_get_data_example
Válido: {MOCK: "test", "answer", "user1"}	MOCK: {"respuesta": "Generated answer with score 8", "array": [8]}	test_get_data_answer

Función	registrar_usuario (mismo test para las dos clases)	
Descripción	Registra un nuevo usuario en la base de datos	
Inputs	Expected Output	Test ID
Válido: {Usuario no existe}	MOCK: {"id": "user123"}	test_registrar_usuario

Función	validar_usuario (el mismo test para las dos clases)	
Descripción	Valida las credenciales de un usuario.	

Inputs	Expected Output	Test ID
Válido: {Usuario existe}	MOCK: {"id": "user123"}	test_validar_usuario
No Válido: {Usuario no existe}	MOCK: None	test_validar_usuario

Función	insertar_valor_array	
Descripción	Inserta un valor en el array de un usuario.	
Inputs	Expected Output	Test ID
Válido: {MOCK id y valor}	MOCK: {"array": [1, 2, 3]}	test_insertar_valor_array
No Válido: Comprobado en BD (recibe un array por defecto que entra dentro del scope del test anterior)	N/A	N/A

Función	recuperar_valores_array	
Descripción	Inserta un valor en el array de un usuario.	
Inputs	Expected Output	Test ID
Válido: {MOCK id}	MOCK: {"array": [1, 2, 3]}	test_recuperar_valores_array
No Válido: N/A Comprobado en BD (recibe un array por defecto que entra dentro del scope del test anterior)	N/A	N/A

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_proxy.py ..... [100%]
=====
===== 8 passed, 11 warnings in 4.47s =====
```

Módulo: VectoreStore

Función	__init__	
Descripción	Inicializa la clase VectoreStore.	
Inputs	Expected Output	Test ID
Válido: {por defecto}	<ul style="list-style-type: none"> - instancia_mockeada.client == mock_chromadb_client.return_value - instancia_mockeada.embedding_model is not None 	test_init
No Válido: N/A	N/A	N/A

Función	_start_embd_model	
Descripción	Inicia el modelo de embedding	
Inputs	Expected Output	Test ID
Válido: {"sentence-transformers/all-MiniLM-L6-v2"}	El modelo returned no es None	test_start_embd_model
No Válido: N/A	N/A	N/A

Función	load_and_embed	
Descripción	Carga los documentos en la vectorstore	
Inputs	Expected Output	Test ID
Válido: {"document_route"}	<ul style="list-style-type: none"> - mock_loader_instance.load es llamado una vez - mock_splitter_instance.split_documents es llamado una vez con content_mock - mock_embed_documents es llamado una vez con ["split_content"] 	test_load_and_embed
No Válido: N/A	N/A	N/A

Función	get_context	
Descripción	Obtiene el contexto general, del cliente y de similitud.	
Inputs	Expected Output	Test ID
Válido: {"text"}	<ul style="list-style-type: none"> - general_context == "context" - client_context == "context" - similarity_context == "question" - _get_document es llamado dos veces con ("text", "general_context") y ("text", "client_context") - _get_question es llamado una vez con ("text", "similarity_context") 	test_get_context
No Válido: N/A	N/A	N/A

Función	_get_document	
Descripción	Obtiene el contexto general, del cliente y de similitud.	
Inputs	Expected Output	Test ID
Válido: {"text", "index_name"}	MOCK: "document"	test_get_document
No Válido: {Excepción por error}	Excepción	test_get_document_exception

Función	_get_question	
Descripción	Obtiene una pregunta similar de una colección específica	
Inputs	Expected Output	Test ID
Válido: {MOCK: "text", "index_name", "similarity_th"}	MOCK: None o "Question" en base al MOCK similarity_th	test_get_question
No Válido: {Excepción por error}	Excepción	test_get_question_exception

Función	add_question	
Descripción	Añade una pregunta a la colección de similitud	
Inputs	Expected Output	Test ID
Válido: {MOCK: "Context" y "Question"}	<ul style="list-style-type: none"> - mock_get_collection es llamado una vez con "similarity_context" - mock_embed_query es llamado una vez con "%CONTEXT%context%QUESTION%question" - mock_collection.add es llamado una vez con los ids y embeddings correctos 	test_add_question
No Válido: N/A (hardcodeado)	N/A	N/A

Resultados

Los resultados de ejecutar los tests para este módulo son los siguientes:

```
test_rag.py ..... [100%]
=====
===== 9 passed, 9 warnings in 19.15s =====
```

Resultados Generales

Los test unitarios se pasan con un 100% de ratio de paso. Se adjunta a continuación los resultados:

```
test_api.py ..... [ 9%]
test_api_helper.py ..... [ 22%]
test_cache.py ... [ 25%]
test_database.py ..... [ 38%]
test_file_reader.py .. [ 40%]
test_ftp_client.py .... [ 45%]
test_ftp_server.py .... [ 50%]
test_generic_llm.py ..... [ 59%]
test_main_start.py . [ 60%]
test_nemo_config.py ... [ 62%]
test_nemo_core.py ..... [ 69%]
test_number_grabber.py ..... [ 75%]
test_parser.py .... [ 79%]
test_prompt_templates.py .... [ 83%]
test_proxy.py ..... [ 91%]
test_rag.py ..... [100%]
=====
===== 103 passed, 24 warnings in 24.84s =====
```

Pruebas de integración

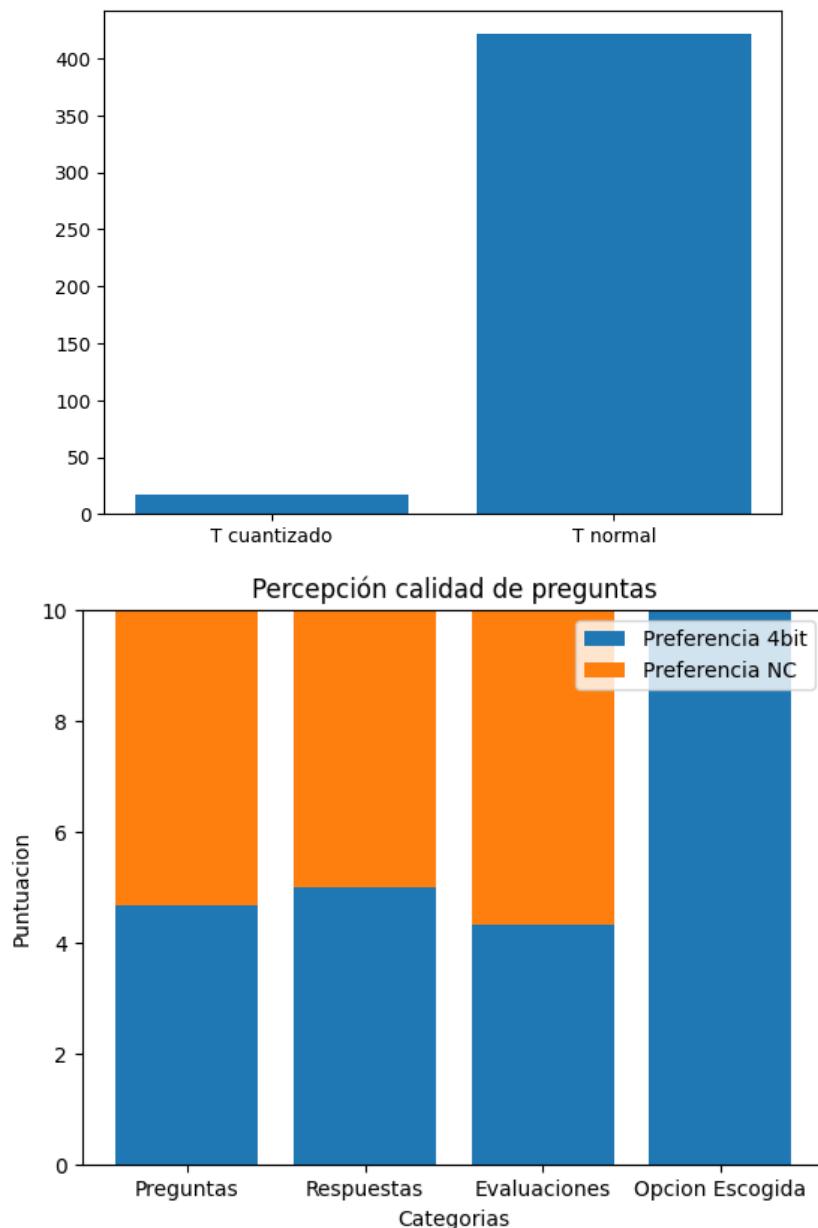
En las pruebas de integración se han realizado notebooks en los que se prueba la integración de los módulos entre sí, probando cambios de flujo estandarizados.

En el apartado de Tests/integration se ofrecen los siguientes tests:

- Tests de comunicación entre la API y la interfaz del *frontend*
- Tests de comunicación entre el servidor y el cliente FTP

- Tests de funcionamiento de la base de datos
- Tests de funcionamiento del LLM genérico
- Tests de funcionamiento del Gestor de comportamientos
- Tests de funcionamiento del Proxy
- Tests de funcionamiento del RAG
- Tests de generación de inferencia con el entrevistador

A mayores, se hace una prueba de generación de muchas respuestas del *bot* del que se ha evaluado tanto el tiempo de respuestas como la calidad (también es parte de usabilidad). Los resultados se pueden ver en las siguientes gráficas. Para ver el Excel de preguntas ir a Tests/Integration en el repositorio.



Pruebas de usabilidad

El primer criterio del que vamos a evaluar la usabilidad es de la GUI. Para ello hemos construido un *mock* en el apartado Tests/usability del repositorio. A partir de ese *mock* construimos un cuestionario (para responder con imágenes y con el *mock*). El enlace:

https://docs.google.com/forms/d/e/1FAIpQLSemyPiMTV0eUEMfbmuZ1VRNAyproW8tO3blx4TaVicVJqw5cA/viewform?usp=sf_link

UsabilidadTFG

Cuestionario de usabilidad de la GUI de AIWantTheJob

Aquí hay imágenes, en la carpeta compartida con este cuestionario está el front para ser ejecutado.

Si vas a ejecutar por favor haz como que te registras para ver el tutorial de primeras (aunque se pueda acceder a este siempre)

Login

AIWantTheJob

[Github](#)

Login

Username

Password

Please enter your credentials to login.

1. Nivel de Usabilidad del Login

Marca solo un ídolo.

1 2 3 4 5

Poco Muy Usable

Registro

AIWantTheJob

[Github](#)

Register

Username

Password

Please enter your credentials to register.

2. Nivel de Usabilidad del Registro

Marca solo un ídolo.

1 2 3 4 5

Poco Muy Usable

2/3 Tutorial

AIWantTheJob

Tutorial

We present the tutorial for the use of this app. First as you have seen we have presented you with both:

Login

Register

Top-of-chat buttons

In the following page you will access the chatbot.

AIWantTheJob

Interviewer Options

On top of the page you have access to a Logout button and a Tutorial button to come back here.

File upload

Grade History

Performance Graph [0-5]

This graph allows you to upload files for use in Behavior-Augmented Interview (BAI). Indeed, senders have to enhance the AI's knowledge base for more accurate responses.

File upload for RAG

This widget allows you to upload files for use in Behavior-Augmented Interview (BAI). Indeed, senders have to enhance the AI's knowledge base for more accurate responses.

File upload

Grade History

Performance Graph [0-5]

This graph shows your performance over time. It allows you to monitor your progress.

3/3 Tutorial

Finally we provide a graph ranging from 0 to 5 that serves as a way of following your progress.

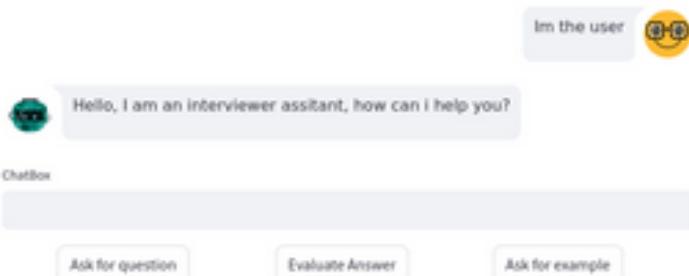
3. El tutorial es útil para entender la aplicación

Marca solo un óvalo.

1 2 3 4 5

No ⚡⚡⚡⚡⚡ Si

Chatbot



4. Entiendo como funciona el entrevistador

Marca solo un óvalo.

1 2 3 4 5

No ⚡⚡⚡⚡⚡ Si

5. Entiendo como funciona el entrevistador después del tutorial

Marca solo un óvalo.

1 2 3 4 5

No ⚡⚡⚡⚡⚡ Si

Botones chat arriba

AIWantTheJob

GitHub

Tutorial

Logout

6. Entiendo para que sirven los botones del chat

Marca solo un óvalo.

1 2 3 4 5

No ⚡⚡⚡⚡⚡ Si

Subir Archivo

File Upload for RAG

This widget allows you to upload files for use in Retrieval-Augmented Generation (RAG). Upload your document here to enhance the AI's knowledge base for more accurate responses.

Sube un archivo

 Drag and drop file here
Limit 200MB per file
Browse files

7. Entiendo como subir un archivo

Marca solo un ítem.

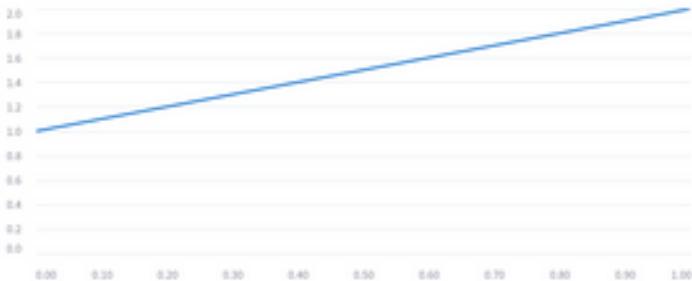
1 2 3 4 5

No      Si

Grafico puntuaciones

Performance Graph [0-5]

This graph displays your performance metrics over time. It shows how your responses have been evaluated throughout the conversation.



8. Entiendo que representa la gráfica

Marca solo un ítem.

1 2 3 4 5

No      Si

9. Grado de usabilidad general

Marca solo un ítem.

1 2 3 4 5

Mala      Buena

10. Diseño de la aplicación

Marca solo un óvalo.

1 2 3 4 5

Malo Bueno

11. Simplicidad de la aplicación

Marca solo un óvalo.

1 2 3 4 5

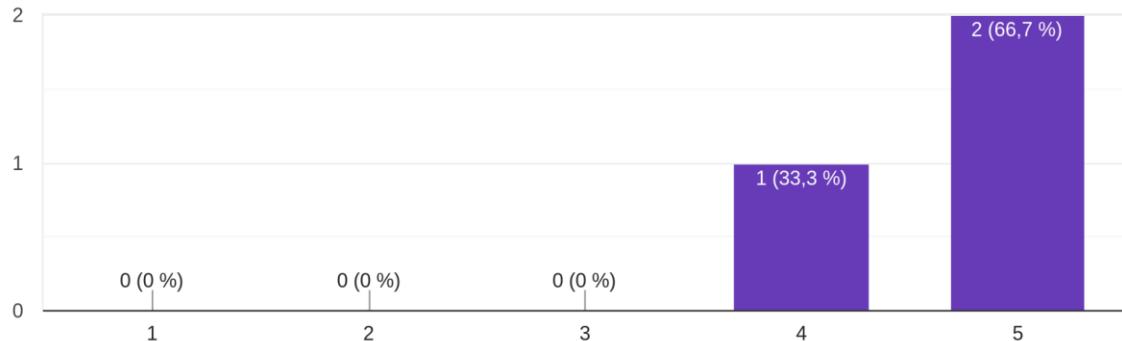
Muy Muy Simple

12. Otros comentarios: (El diseño esta muy limitado por el framework)

Las respuestas a este cuestionario fueron las siguientes:

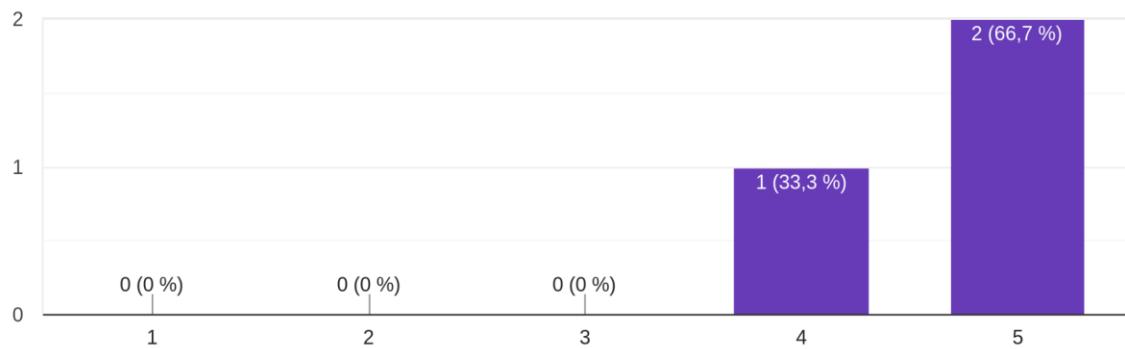
Nivel de Usabilidad del Login

3 respuestas



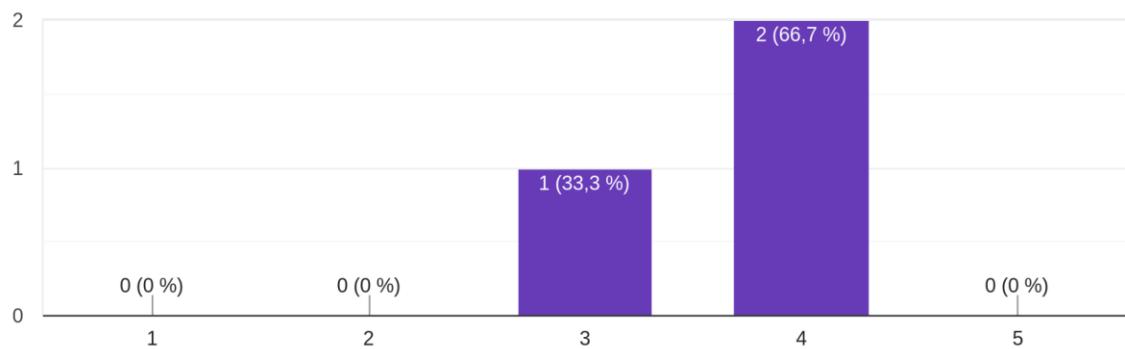
Nivel de Usabilidad del Registro

3 respuestas



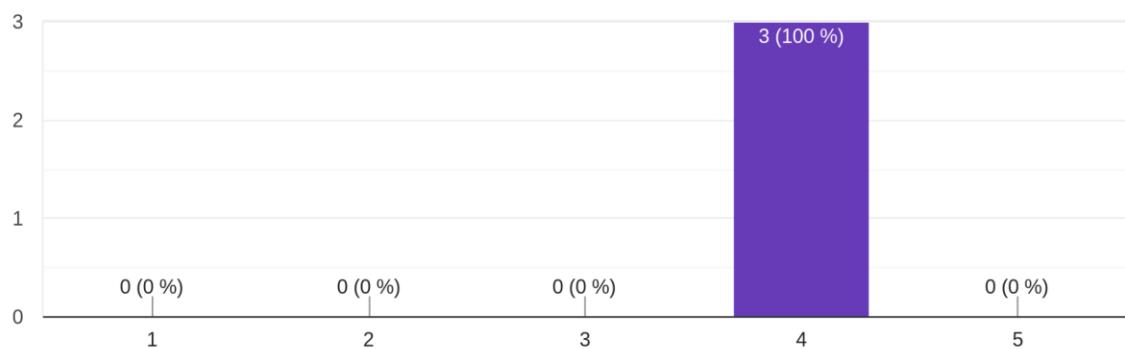
El tutorial es útil para entender la aplicación

3 respuestas



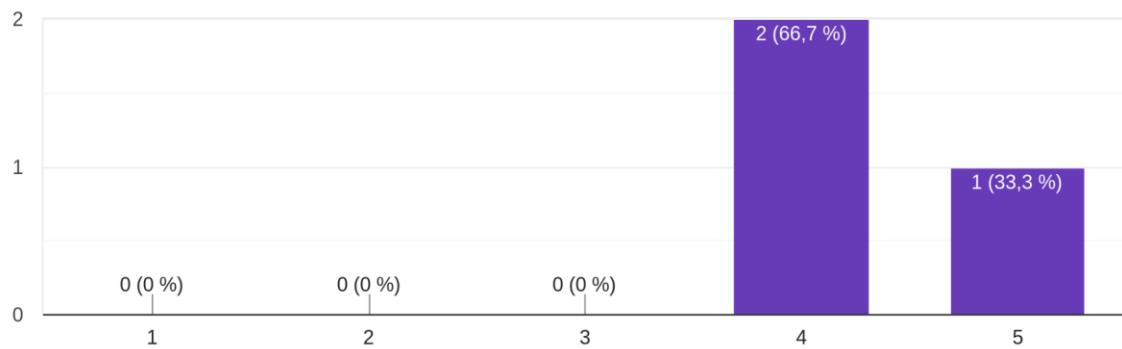
Entiendo como funciona el entrevistador

3 respuestas



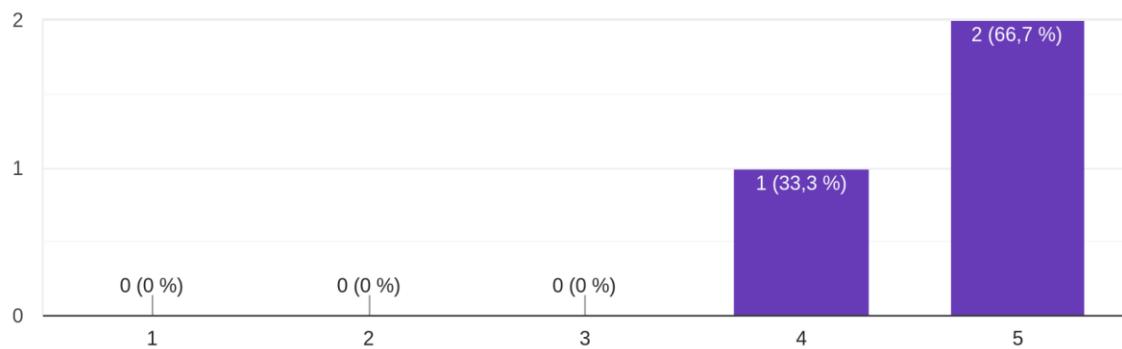
Entiendo como funciona el entrevistador después del tutorial

3 respuestas



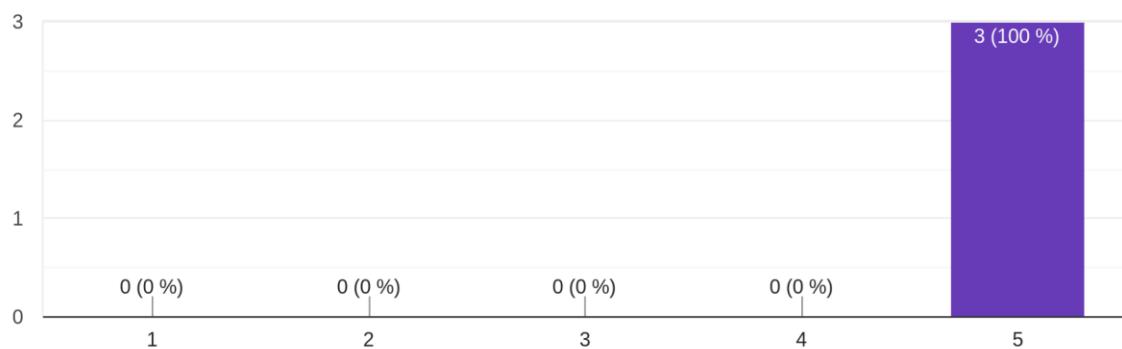
Entiendo para que sirven los botones del chat

3 respuestas



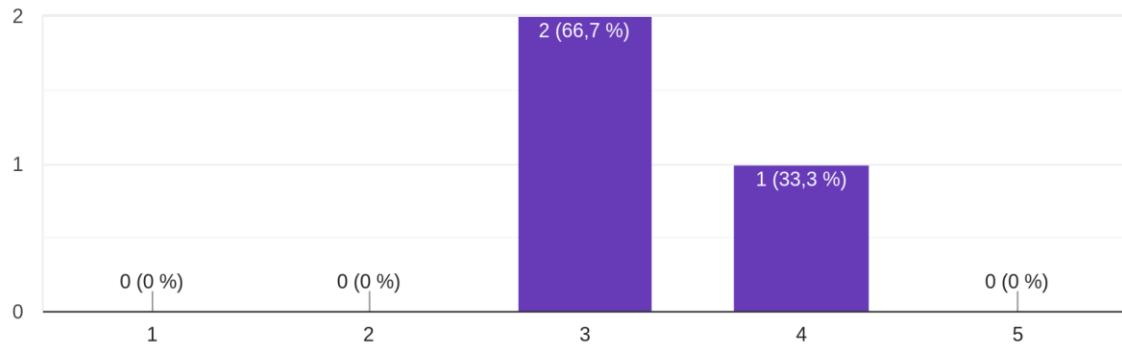
Entiendo como subir un archivo

3 respuestas



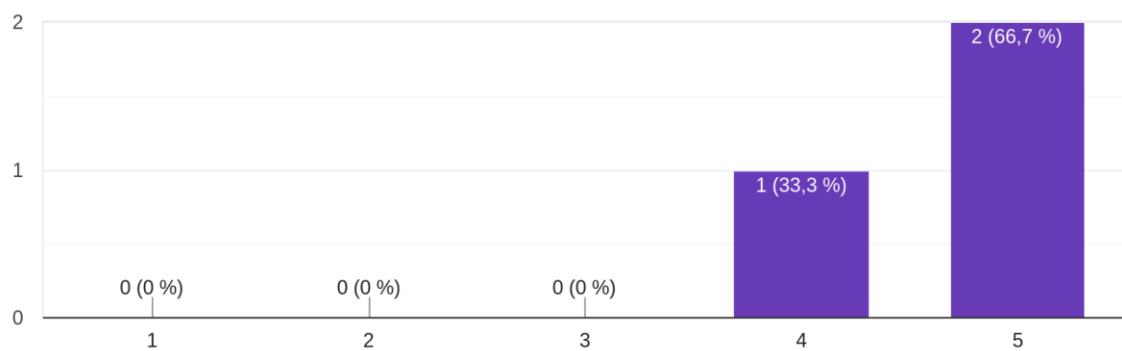
Entiendo que representa la gráfica

3 respuestas



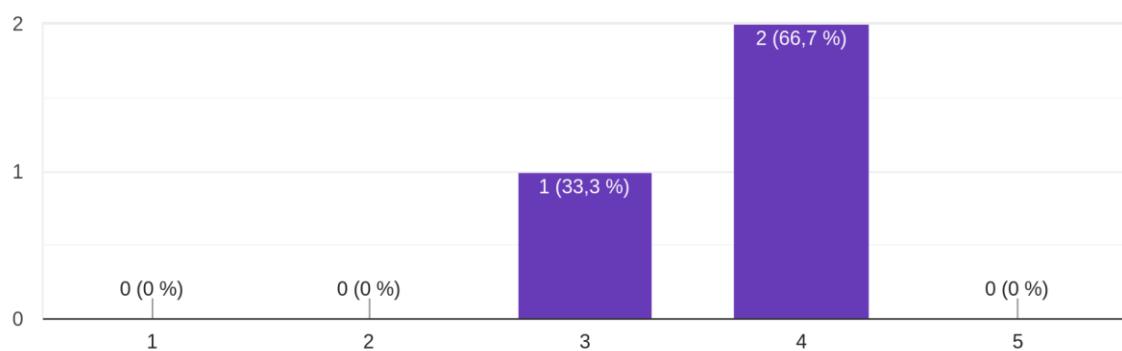
Grado de usabilidad general

3 respuestas



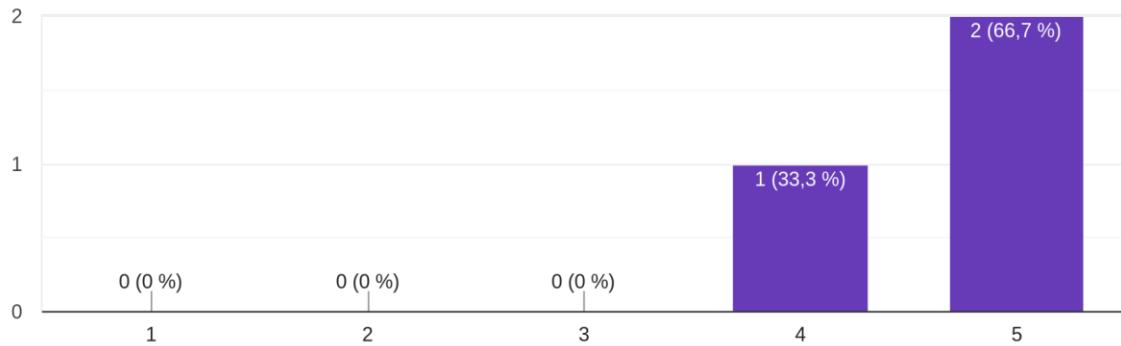
Diseño de la aplicación

3 respuestas



Simplicidad de la aplicación

3 respuestas



Otros comentarios (El diseño esta muy limitado por el framework)

2 respuestas

Diseño simple pero eficaz. Funcionamiento correcto, de uso intuitivo.

Muy simple

Luego realizamos inferencia del modelo con y sin cuantizar (como ya se explicó en el apartado de integración) y realizamos un cuestionario comprobando la calidad de las respuestas para los usuarios. El enlace al cuestionario es el siguiente:
https://docs.google.com/forms/d/e/1FAIpQLSfagpm9ZYqlZPBkdShZT-6ay5jc19Z0bGT4_0uWdKGMuoiA/viewform?usp=sf_link

Este cuestionario fue el siguiente:

Calidad de Respuestas (Usabilidad)

Adjunto a este formulario se ha entregado un excel con las respuestas.

NO es un chatbot Q/A, se supone que emula personalidades entrevistador de trabajo | entrevistado

Hay 3 hojas, comparando 3 tipos de calls diferentes de un modelo de lenguaje. Comparar la calidad de las RESPUESTAS entre 4bit y NoCuant y responder a lo siguiente:

1. Calidad de la generación de preguntas en base a un tópico

Marca solo un óvalo.

1 2 3 4 5 6 7 8 9 10

NoC 4bit es mucho mejor que NoCuant

2. Calidad de la generación de respuestas a una pregunta

Marca solo un óvalo.

1 2 3 4 5 6 7 8 9 10

NoC 4bit es mucho mejor que NoCuant

3. Calidad de la evaluación de pregunta-respuesta

Marca solo un óvalo.

1 2 3 4 5 6 7 8 9 10

NoC 4bit es mucho mejor que NoCuant

4. En general, las respuestas del LLM tienen sentido

Marca solo un óvalo.

1 2 3 4 5

No Si

5. Sabiendo que el coste temporal de ejecutar no cuant es entre 10 y 25 veces más que 4bit, que respuestas (a nivel general) escogerías

Marca solo un óvalo.

4bit

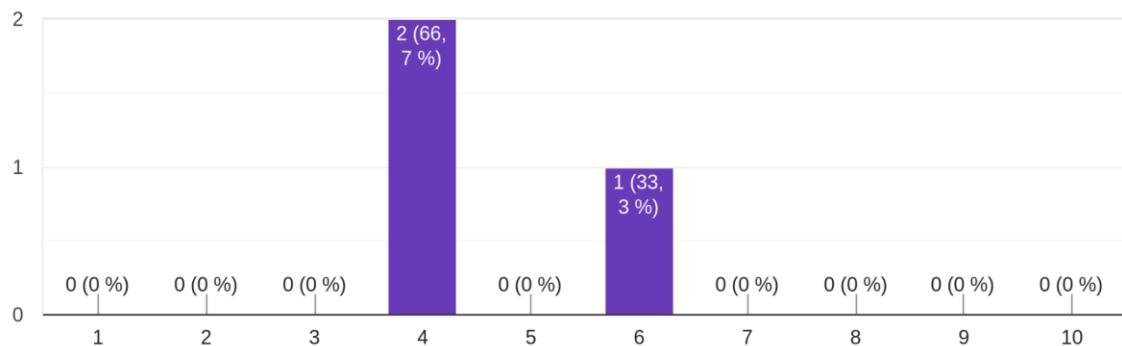
NoCuant

Indiferente

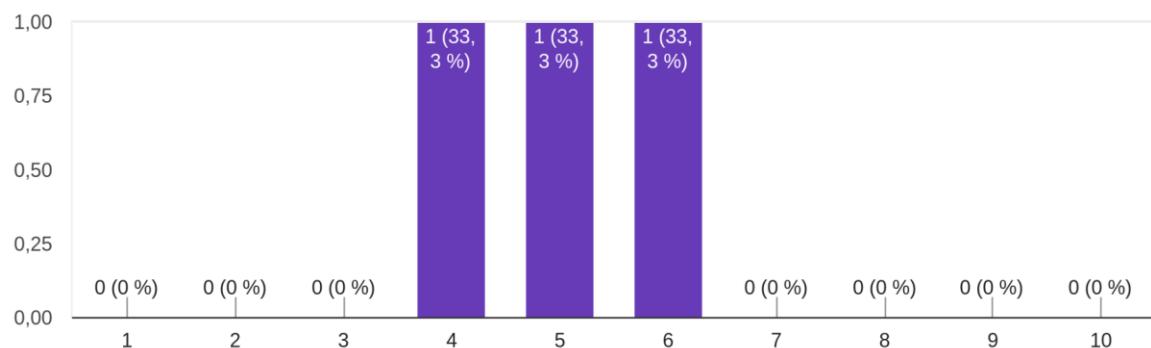
Las respuestas a este cuestionario fueron las siguientes:

Calidad de la generación de preguntas en base a un tópico

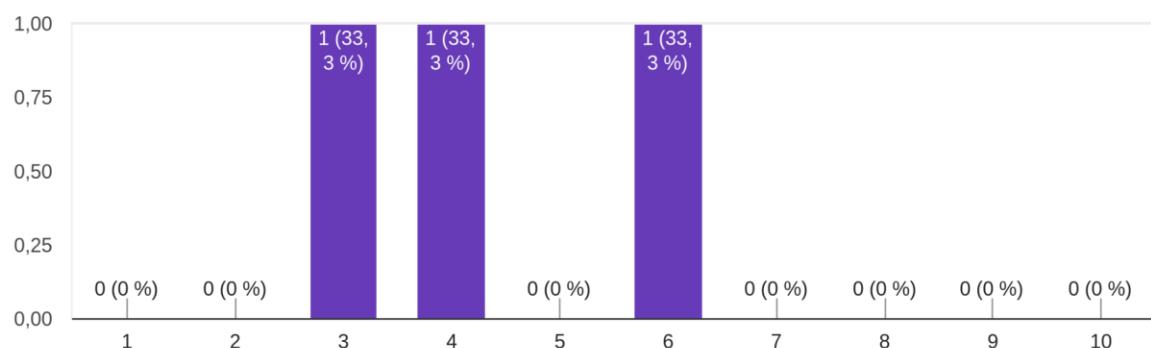
3 respuestas

**Calidad de la generación de respuestas a una pregunta**

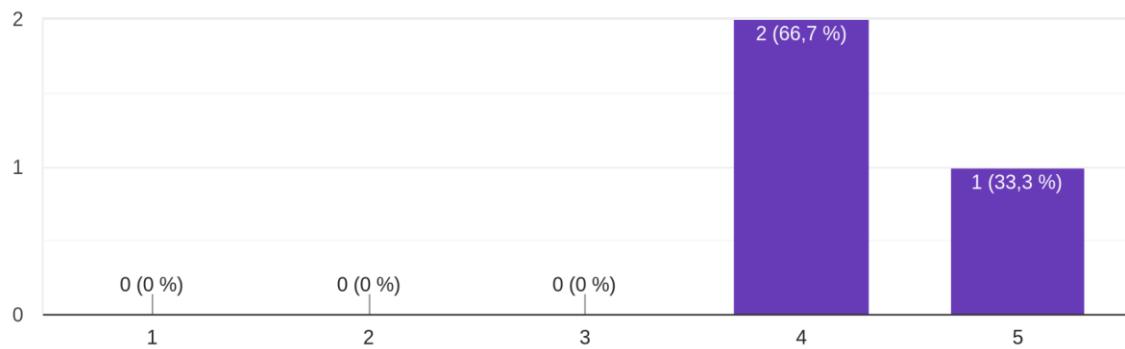
3 respuestas

**Calidad de la evaluación de pregunta-respuesta**

3 respuestas

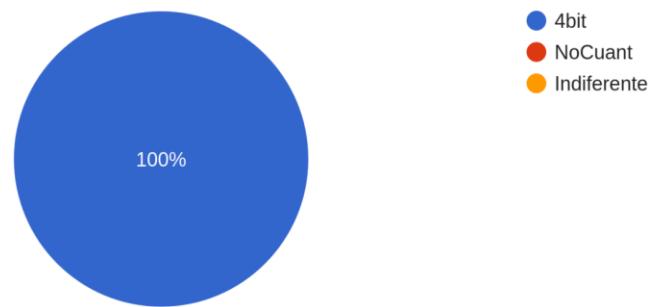


En general, las respuestas del LLM tienen sentido
3 respuestas



Sabiendo que el coste temporal de ejecutar no cuant es entre 10 y 25 veces más que 4bit, que respuestas (a nivel general) escogerías

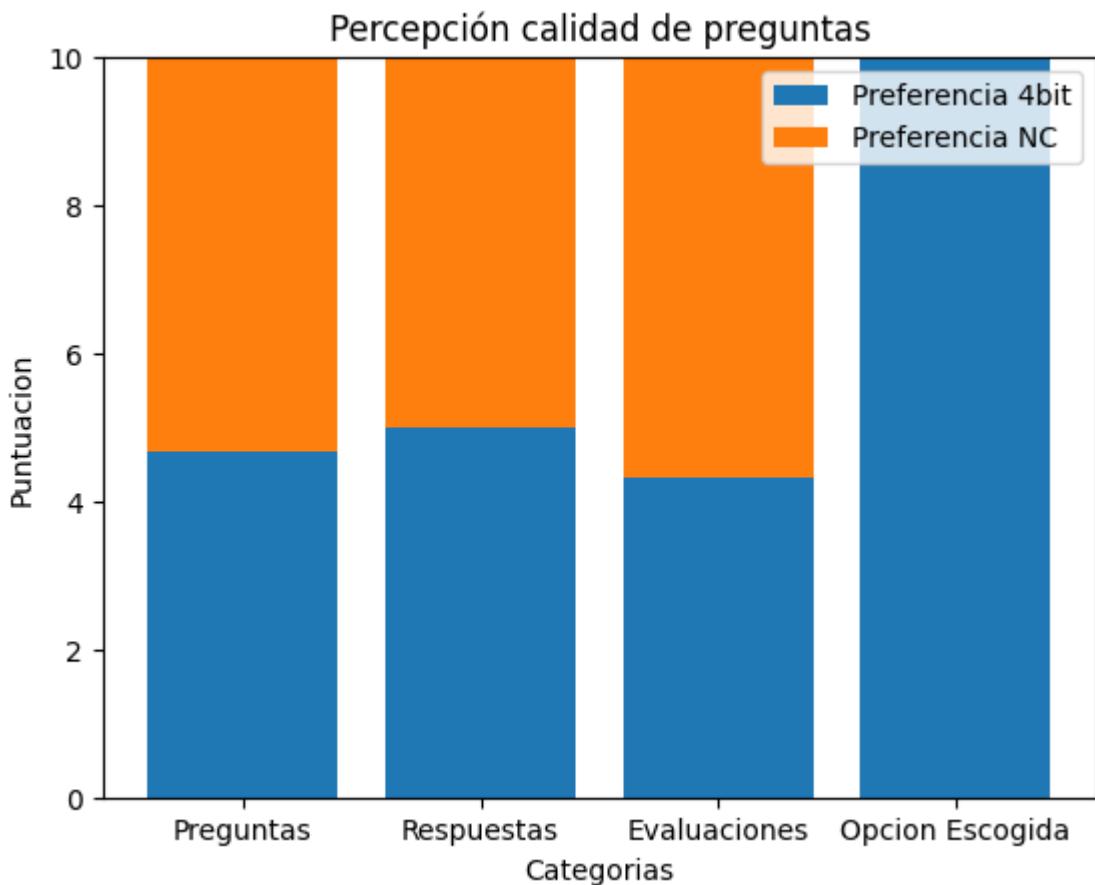
3 respuestas



Conclusiones del apartado

De la parte de diseño se concluye que la GUI es muy usable y sencilla de diseño (esto es tanto algo bueno como algo malo).

Por otro lado, de los resultados de la encuesta se puede resumir la siguiente gráfica, con la que podemos ver que no merece la pena usar un modelo más grande ya que el pequeño funciona muy bien.



Pruebas end-to-end

Las pruebas de extremo a extremo se verifican con la realización de los flujos completos de trabajo del usuario. En este caso, como se presenta una aplicación que implementa todos sus objetivos y requisitos, las pruebas de extremo a extremo se pasan por el funcionamiento de dicha aplicación

Pruebas de despliegue

Para probar el despliegue realizamos dicho despliegue a través de todos los métodos de despliegue planteados:

Despliegue manual

El funcionamiento del despliegue manual se auto verifica porque la aplicación fue desarrollada así y es el método que use como desarrollador por defecto.

Despliegue Conda

Podemos ver como ejecutando el script de *run* (Conda en el *background* gracias al comando *source*) el código funciona. De esta manera ofrecemos el entorno virtualizado. El video de demo en el repositorio está realizado ejecutando el proyecto en Conda.

```
(base) jorge@OJU:~/Escritorio/TFGoficial/usc-aiwantthejob/scripts (Feature/tests)
$ ./run.sh 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
Cargando backend...
Cargando frontend...

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.0.36:8501
```

Conclusión

Hemos pasado todas las pruebas planteadas. De esta manera cumplimos el criterio de aceptación A-001 y se determinan las pruebas realizadas como pasadas.

Gestión Cronograma

La gestión del cronograma se hizo a través de taiga scrum, se adjunta a continuación el gráfico del backlog.

Scrum

