

UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA INFORMÁTICA EN
TECNOLOGÍAS DE LA INFORMACIÓN



DESARROLLO DE UNA APLICACIÓN DE
INTELIGENCIA ARTIFICIAL
CORPORATIVA PARA
DESARROLLADORES

TRABAJO FIN DE GRADO

Septiembre - 2024

AUTOR: Francisco Jiménez Sánchez
DIRECTOR: Jesús Javier Rodríguez Sala

RESUMEN

En este proyecto se pretende abordar la realización de una aplicación que sirva como asistente de consulta de lenguajes de programación estandarización para proyectos del Banco Santander.

Puesto que existe una gran hermeticidad y las conexiones realizadas en remoto son a través de máquinas virtuales con sistemas operativos conectados únicamente a una intranet, no es posible usar asistentes como ChatGPT, Copilot, etc.

La empresa tiene terminantemente prohibido el uso de estos asistentes, ya que toda la información confidencial que se introduzca puede usarse para el propio aprendizaje de éstas, quedando partes de código expuestos al público.

Es por ello que se requiere realizar un asistente para poder usarse de forma hermética e interna, de modo que también pueda ser entrenado usando los estándares de programación de la compañía y la opinión de los propios usuarios internos.

La aplicación en su totalidad consistirá de una página web que irá anexada dentro de la caja de herramientas de la compañía, microservicios backend alojados en los repositorios internos y el propio asistente que se ejecutará en el servidor donde la potencia de cómputo es crucial.

AGRADECIMIENTOS

Quiero agradecer a toda mi familia el apoyo recibido que me ha servido para poder continuar con dicho proyecto en los momentos difíciles.

A mis amigos, los cuales han sido de gran ayuda en esos pequeños momentos libres entre trabajo, cursos y proyecto de carrera.

A mis compañeros de trabajo, los cuales me han dado consejos y directrices para cumplir con el estándar solicitado.

Mención especial a mi tutor de proyecto, quien me ha apoyado en la medida de lo posible todo el trabajo realizado, por su paciencia y su comprensión, aportando su esfuerzo para que pudiera completar dicho proyecto.



ÍNDICE GENERAL

1. Introducción	8
1.1. LAS INTELIGENCIAS ARTIFICIALES GENERATIVAS	8
1.1.2. IAs Generativas para la creación de GPTs	10
1.1.3. Proceso de entrenamiento de un GPT	11
1.1.4. Ajuste fino de una IA	12
1.1.5. Conclusión	13
1.2. EMPRESA/ENTORNO DE APLICACIÓN	13
1.3. JUSTIFICACIÓN DEL PROYECTO	14
1.4. OBJETIVOS	15
1.4.1. Objetivo Principal	15
1.4.2. Objetivo Secundario	16
1.4.3. Objetivo Personal	16
1.5. LÍMITES DEL PROYECTO	17
2. Antecedentes	18
2.1. INTELIGENCIAS ARTIFICIALES GENERATIVAS	18
2.1.1. Historia de la IA Generativa	18
2.1.2. Funcionamiento de la IA Generativa	19
2.1.3. Redes Generativas Antagónicas	20
2.1.4. Modelos de Transformadores	21
2.1.4.1. Arquitectura Codificador-Decodificador de modelos transformer	21
2.1.4.2. Desventajas de la arquitectura transformer	23
2.2. MODELOS GRANDES DE LENGUAJE	23
2.2.1. Definición de Modelo de Lenguaje Grande (LLM)	23
2.2.2. LLMs y Procesamiento de Lenguaje Natural	24
2.2.3. Componentes de un Modelo de Lenguaje Grande	25
2.2.3.1. Capa de Incrustación	25
2.2.3.2. Capa Feedforward Neural Network (FNN)	27
2.2.3.3. Capa Recurrente	27
2.2.4. Funcionamiento de LLM	28
2.2.5. Usos y aplicaciones de los LLMs	29
2.2.6. Inconvenientes de los LLMs	30
2.3. HERRAMIENTAS DE CÓDIGO ABIERTO	32
2.3.1. Mistral AI	32
2.3.2. Llama AI	34
3. Hipótesis de trabajo	37
3.1. MODELOS LLM DE MISTRAL	37
3.1.1. Mistral-7B-Instruct-v0.2	37
3.1.2. Mistral-7B-Instruct-v0.2-GPTQ	38

3.4. POSTGRESQL Y DBeaver	41
3.5. JAVA, SPRINGBOOT Y MAVEN	43
3.6. OTRAS HERRAMIENTAS	45
3.6.1. El asistente Pinokio	45
3.6.2. Postman	46
3.6.3. IntelliJ IDEA - Community Edition	47
3.6.4. RabbitMQ	49
4. Metodología y resultados	52
4.1. Planificación del proyecto	52
4.1.1. Ciclo de vida - Espiral	53
4.1.2. Diagrama de Gantt	55
4.2. Captura de requisitos	55
4.3. Diseño	56
4.4. Implementación	61
4.5. Pruebas	77
5. Conclusiones y trabajo futuro	87
5.1. CONCLUSIONES	87
5.2. POSIBLES DESARROLLOS FUTUROS	88
6. Bibliografía	89
7. Anexos	97
7.A.1. Anexo I Text Generation WebUI	97
7.A.1.1. INSTALACIÓN	97
7.A.1.2. CONFIGURACIÓN	98
7.A.1.3. MANUAL DE USO	102
7.A.1.4. AJUSTE FINO	103
7.A.2. Anexo II Pinokio	105
7.A.2.1. INSTALACIÓN	105
7.A.3. Anexo III Postman	110
7.A.3.1. INSTALACIÓN	110
7.A.3.2. CONFIGURACIÓN DE UN API Y USO	111

ÍNDICE DE FIGURAS

Figura 1.1 Ejemplo de imagen generada con la IA “DALL E3 mediante Copilot	9
Figura 2.1 Proceso Generador - Discriminador GAN	20
Figura 2.2 Diagrama Codificador - Decodificador	22
Figura 2.3 Representación Vectorial de parámetros	22
Figura 2.4 Búsqueda de similitudes por medio de vectores	26
Figura 2.5 Dependencia entre neuronas entrada - salida	28
Figura 2.6 Ejemplo JSON Ajuste Fino	29
Figura 2.7 Ejemplo uso Cleverbot	30
Figura 2.8 Comparación entre modelos Mistral - Llama	33
Figura 2.9 Eficiencia modelo Mistral	33
Figura 2.10 Calidad comprensiva Llama vs ChatGPT	35
Figura 3.1 Interfaz Text Generation WebUI	39
Figura 3.2 Formato de entrada para datos de entrenamiento	40
Figura 3.3 Ejemplo de array JSON con datos de entrenamiento	41
Figura 3.4 Logotipo PostgreSQL	41
Figura 3.5 Logotipo DBeaver	42
Figura 3.6 Interfaz DBeaver	42
Figura 3.7 Logotipo Java	43
Figura 3.8 Logotipo SpringBoot	43
Figura 3.9 Logotipo Maven	44
Figura 3.10 Logotipo Pinokio	45
Figura 3.11 Logotipo Postman	46
Figura 3.12 Interfaz Postman	47
Figura 3.13 Logotipo IntelliJ IDEA	48
Figura 3.14 Interfaz IntelliJ IDEA	48
Figura 3.15 Logotipo RabbitMQ	49
Figura 3.16 Proceso Producer/Consumer RabbitMQ	49
Figura 3.17 Interfaz RabbitMQ	50
Figura 3.18 Comunicación productor/consumidor con RabbitMQ	50
Figura 3.19 Publicación de mensaje en canal y cola de mensajes	51
Figura 4.1 Ciclo de vida Espiral	53
Figura 4.2 Diagrama de Gantt	55
Figura 4.3 Diagrama de casos de uso	56
Figura 4.4 Módulos de la aplicación	57
Figura 4.5 Diseño pantalla de usuario (interfaz web)	57
Figura 4.6 Diagrama de secuencia Desarrollador	58
Figura 4.7 Diagrama de secuencia Technical Lead	59
Figura 4.8 Diagrama de estados	59

Figura 4.9 Diagrama de actividad Desarrollador	59
Figura 4.10 Diagrama de actividad administrador	60
Figura 4.11 Gráfico de flujo completo	60
Figura 4.12 Página web Santander IA	62
Figura 4.13 JavaScript consulta códigos de proyecto	63
Figura 4.14 Configuración API códigos	63
Figura 4.15 Consulta códigos base de datos	63
Figura 4.16 Api JavaScript formulario	64
Figura 4.17 Estructura de cuerpo de mensaje	64
Figura 4.18 Flujo parcial de proceso en micro interfaz web	65
Figura 4.19 Configuración y llamada API a micro interfaz GPT	65
Figura 4.20 Flujo completo de proceso en micro interfaz web	65
Figura 4.21 Adaptaciones de respuesta para la web	66
Figura 4.22 Flujo proceso servicio interfaz GPT	66
Figura 4.23 Proceso de llamada de recuperación de datos de proyecto	67
Figura 4.24 Consulta a base de datos de proyectos	67
Figura 4.25 Flujo preparatoria de pregunta	67
Figura 4.26 Estructura de objeto interno de array	68
Figura 4.27 Proceso de añadido de pregunta al listado	68
Figura 4.28 Estructura principal de cuerpo de mensaje	68
Figura 4.29 Selección de precisión	69
Figura 4.30 Estructura de cuerpo de mensaje	69
Figura 4.31 Ejemplo JSON	70
Figura 4.32 Llamada a servicio IA	70
Figura 4.33 Envío y respuesta desde web al micro interfaz	71
Figura 4.34 Procesamiento de respuesta antes de enviar a la web	72
Figura 4.35 Ejemplo conversión estructura ChatRequest a String	72
Figura 4.36 Ejemplo conversión de String a estructura Question	73
Figura 4.37 Estrategia de valor generado	73
Figura 4.38 Estrategia de valor autogenerado	73
Figura 4.39 Consumo de recurso hardware por parte de la IA	74
Figura 4.40 Consola con información Text Generation WebUI integrada en Pinokio	74
Figura 4.41 Error al intentar entrenar un modelo	74
Figura 4.42 Carga de un modelo	75
Figura 4.43 Selección de formato y archivo json	75
Figura 4.44 Archivo json	75
Figura 4.45 Configuración parámetros ajuste fino I	76
Figura 4.46 Configuración parámetros ajuste fino II	76
Figura 4.47 Pregunta 1 Postman sin ajuste fino	78
Figura 4.48 Pregunta 1 Resultado sin ajuste fino	78
Figura 4.49 Pregunta 1 Postman con ajuste fino	79
Figura 4.50 Pregunta 1 Resultado con ajuste fino	79

Figura 4.51 Pregunta 2 Postman sin ajuste fino	80
Figura 4.52 Pregunta 2 Resultado sin ajuste fino	80
Figura 4.53 Pregunta 2 Postman con ajuste fino	81
Figura 4.54 Pregunta 2 Resultado con ajuste fino	81
Figura 4.55 Pregunta 3 Postman sin ajuste fino	82
Figura 4.56 Pregunta 3 Resultado sin ajuste fino, resultado en navegador	83
Figura 4.57 Pregunta 3 Postman con ajuste fino	84
Figura 4.58 Pregunta 3 Resultado con ajuste fino	84
Figura AI.1 Buscador Pinokio	97
Figura AI.2 Text Generation WebUI	98
Figura AI.3 Consola comandos Text Generation WebUI	98
Figura AI.4 Menú Text Generation WebUI	99
Figura AI.5 Buscador Hugging Face	99
Figura AI.6 Descarga de modelos	99
Figura AI.7 Descarga archivos de Modelo por consola	99
Figura AI.8 Listado Modelos descargados	100
Figura AI.9 Listado de cargadores de modelos	100
Figura AI.10 Configuración hardware	101
Figura AI.11 Apis externas	101
Figura AI.12 Configuración API en Postman	102
Figura AI.13 Respuesta API en Postman	103
Figura AI.14 Panel de entrenamiento	103
Figura AI.15 Cargando LorAs	104
Figura AII.1 Web inicial Pinokio	105
Figura AII.2 Buscador aplicaciones Pinokio	106
Figura AII.3 Ventana instalación Hallo en Pinokio	106
Figura AII.4 Proceso de instalación Hallo en Pinokio	107
Figura AIII.1 Web de la aplicación Postman	108
Figura AIII.2 Descarga de la aplicación Postman	109
Figura AIII.3 Workspace de la aplicación Postman	110
Figura AIII.4 Nueva API Http en Postman	110
Figura AIII.5 Selección método Http en Postman	111
Figura AIII.6 Url y cuerpo mensaje Http en Postman	111
Figura AIII.7 Figura AIII.7: Respuesta Api Http en Postman	112

Capítulo 1

Introducción

1.1.- LAS INTELIGENCIAS ARTIFICIALES GENERATIVAS

Hoy en día, es imposible hablar de los últimos avances tecnológicos sin hacer mención a la inteligencia artificial (IA), sin embargo, ésta no es más que el primer paso detrás de un mundo de posibilidades a todos los niveles tecnológicos y dentro de dichos avances destaca la Inteligencia Artificial Generativa[1].

Imaginemos que deseamos crear un documento de texto donde queremos explicar cuáles son los fundamentos básicos de la programación. Es posible que, si dicho documento contiene definiciones muy técnicas, una gran cantidad de lectores no serán capaces de comprender dichos contenidos. Sin embargo, es posible que al usar palabras genéricas y entendibles para todo el mundo, no se aborde toda la información que se quiere transmitir. Es aquí donde entran en acción las IAs Generativas.

Una IA Generativa es capaz de crear contenido, ya sea texto, imágenes o incluso audios, en los que intenta recrear aquello que se le solicita. Sería tan fácil como pedirle que explique cuáles son los fundamentos básicos de la programación y, una vez generado dicho contenido, solicitarle que lo explique en términos sencillos, que use ejemplos cotidianos y que sea creativa en las explicaciones más tediosas para hacer más amena la lectura.

Para lograr este tipo de respuestas, las IAs actuales se basan en modelos denominados LLMs (“*Large Language Models*” o modelos de lenguaje amplios) para la creación de sistemas donde aprende a "entender" qué es lo que solicita el usuario y cómo abordarlo para dar la respuesta más coherente posible.

La idea de un modelo de lenguaje amplio, o LLM por sus siglas en inglés, parte de una colosal cantidad de información que se utiliza para entrenar a estas IAs y conseguir que “aprendan” por sí mismas. Normalmente un LLM consta de miles de millones de parámetros que deben ser ajustados para su funcionamiento.

Con una de estas IAs ya entrenada, un ejemplo de uso podría ser solicitarle que genere una fotografía de una biblioteca, pero con un estilo futurista, donde se mezclen elementos clásicos con otros de ciencia ficción. La IA debería ser capaz de comprender cada término y combinar todos los conceptos para generar la imagen solicitada.



Figura 1.1.- Ejemplo de imagen generada con la IA “DALL E3 mediante Copilot”

Al final, la habilidad básica de la IA Generativa es la de ser capaz de recrear, por medio de un ajuste más refinado denominado “*Fine Tune*” (ver más adelante), un contenido creado por personas de modo que pueda producirlo por sí misma. Una vez entrenada, será capaz de crear cualquier contenido de forma autónoma.

1.1.2.- IAs Generativas para la creación de GPTs

Un GPT, o por sus siglas Generative Pre-trained Transformer[12][13] (Transformador Generativo Pre-entrenado), es un modelo que se basa en una arquitectura denominada “Transformer”, que consiste en una red neuronal profunda que “transforma” una secuencia de datos de entrada en una secuencia de salida. La meta es lograr como respuesta un contenido lo más coherente y relevante posible con la entrada proporcionada. Uno de los puntos fuertes donde más ha destacado la arquitectura transformer es, sin duda, en el campo del procesamiento del lenguaje natural "NLP"[14][15].

Para crear un modelo GPT se requiere realizar un entrenamiento, el cual consiste en el uso de gran cantidad de datos que el modelo organiza en función de los patrones que va detectando, ajustando sus parámetros según dichos patrones. Por ejemplo, los modelos tipo chat (p.e.: ChatGPT) deben predecir la palabra que sigue a una secuencia dada por el usuario. Dicho método de entrenamiento se conoce como "*entrenamiento autorregresivo*", y en él se logra desarrollar una profunda comprensión del contexto. Algunos aspectos básicos de los modelos de tipo GPT son:

Arquitectura de Transformer

Se trata de una red neuronal donde se utiliza un sistema codificador-decodificador a la hora de realizar procesamiento de lenguaje natural [2][3]. Una novedad de la arquitectura transformer con respecto a otras redes neuronales es la posibilidad de introducir parámetros de entrada con una longitud variable.

En el proceso de codificación, el modelo desglosa los datos de entrada y los analiza por separado, de este modo se logra clasificarlos para realizar un posterior reconocimiento de cada entidad. En el proceso de decodificación se recopila toda la información consultada por medio de las entidades y se genera la respuesta final solicitada por el usuario.

Aprendizaje autorregresivo

La idea detrás de un aprendizaje autorregresivo [4][5] es la capacidad de adaptar una respuesta a un problema, de modo que ésta se ha visto afectada debido a perturbaciones dentro de los parámetros de entrada. El motivo por el que se produce dicha regresión se

debe a que las respuestas generadas en consultas a lo largo del tiempo pasan también a formar parte de los parámetros de entrada en futuras consultas. Esto provoca que, para una misma petición, se obtengan distintas respuestas dependiendo del momento en el que se ha realizado dicha petición.

Este aprendizaje puede realizarse ya sea por un gran número de iteraciones con el usuario final, donde el modelo se adapta a las características de dicho usuario, o simplemente con el fin de seguir una coherencia en la conversación llevada a cabo.

1.1.3.- Proceso de entrenamiento de un GPT

El proceso de entrenamiento de un GPT [6] es una tarea ardua que requiere de una serie de pasos, sin los cuales no se alcanzará un nivel de eficiencia aceptable, dichos pasos son:

1. **Recopilación de información:** En este paso se reúne una gran cantidad de datos a través de fuentes diversas, como pueden ser sitios webs, artículos, etc., en general cualquier información que consideremos válida para el modelo que se quiere desarrollar.
2. **Preprocesamiento de la información:** Una vez completada la recopilación de datos, se procede a realizar un preprocesamiento sobre los mismos. En este punto se busca simplificar y disponer la información, normalmente textual, en sus unidades más pequeñas, denominadas tokens. Además se eliminá la información no válida como caracteres o formatos que puedan obstaculizar el futuro preentrenamiento.
3. **Elección de la arquitectura a utilizar por el GPT:** En este paso se seleccionará la arquitectura basada en transformadores que constituye los cimientos del GPT que se quiere construir.
4. **Primera toma de contacto con pre-entrenos:** Con los datos preprocesados y la arquitectura escogida, se empieza realizando un primer entrenamiento sin supervisión. Dicho entrenamiento consiste en, a partir de los datos preprocesados de entrada, el modelo tiene que predecir las palabras o tokens que suceden a cada serie de entrada, sin romper el contexto de la idea principal. De este modo se logra que el GPT consiga habilidades lingüísticas, gramaticales y una cierta “*comprensión*” del lenguaje a nivel genérico.
5. **Afinamiento “Fine-Tune” (Ajuste Fino):** Tras el preentrenamiento, se pasa a realizar un afinamiento del modelo, también denominado Fine-Tune. En términos sencillos, dicho ajuste consiste en dar al modelo un conjunto de preguntas y junto a ellas las respuestas a cada una de ellas, de este modo se consigue que la respuesta

se adapte a las especificaciones deseadas. Por ejemplo, si se quiere un modelo para una empresa agroalimentaria, se espera que en respuesta a una pregunta sobre tipos de aceite y sus propiedades responda refiriéndose al aceite de oliva, girasol, etc, y no al aceite de motor usado en automoción. De esta etapa de afinamiento se hablará con más detalle en el punto 1.1.4 “Ajuste fino de una IA”.

6. **Optimización:** La optimización del modelo consiste en aplicar diversas técnicas que ayuden a mejorar el rendimiento del modelo. Una muy sencilla puede consistir en iterar la fase de afinamiento con nuevos datos y/o ajustando algunos de los parámetros del modelo para tratar de obtener un mejor desempeño del mismo.
7. **Despliegue del modelo:** Una vez completados todos los pasos anteriores, ya se puede realizar el despliegue del GPT para poder usarlo en aplicaciones de todo tipo. En este punto, el modelo debe ser capaz de responder a preguntas que se le hagan, e incluso realizar tareas relativamente complejas, dependiendo siempre del afinamiento que se le haya querido dar. Hay que tener en cuenta que, para que un modelo funcione de forma óptima, debe contar con un gran nivel de recursos informáticos, así como un gran banco de datos preprocesados a los que acceder.

1.1.4.- Ajuste fino de una IA

El ajuste fino de una IA [7] es uno de los pasos mencionados anteriormente a la hora de realizar el proceso de creación de un GPT. Con dicho ajuste, se logra que nuestro modelo pueda especializarse en un campo con una gran eficiencia y precisión. Hay que tener en cuenta que se trata de un proceso supervisado, por lo que se usarán datos de entrada previamente clasificados (o etiquetados), estos datos podrán haber sido etiquetados de antemano por el propio desarrollador (según su criterio), o de forma automático por algún otro modelo de clasificación.

Para la realización del ajuste se debe preparar un conjunto de datos específicos consistentes en una serie de conversaciones que se usarán a modo de demostración, acompañadas de las respuestas esperadas que el modelo debería proporcionar. El motivo por el que se escogen unos datos concretos se debe a la detección previa de errores en la respuesta usando las preguntas de estas conversaciones, de este modo, el modelo irá “*aprendiendo*” las respuestas correctas para las conversaciones utilizadas con las cuales deberá guiarse en futuras interacciones.

La idea principal detrás del ajuste fino (o fine-tuning) es poder beneficiarse del uso de un modelo GPT previamente entrenado, como por ejemplo ChatGPT, que ya ha adquirido “conocimientos” genéricos extensos gracias a su preentrenamiento con un gran banco de datos recopilados de Internet durante años, este modelo se puede adaptar con información

específica para la realización de una tarea más concreta, de esta forma no hay que entrenar el modelo desde cero, sino que el punto de partida es más avanzado. Debemos añadir que esto último no tiene porqué ser necesario, ya que también es posible indicar al modelo qué dirección debe tomar a la hora de ofrecernos una información u otra.

1.1.5.- Conclusión

Todas las herramientas GPTs que existen actualmente disponibles no hubiesen sido posibles sin las IAs generativas que hay detrás de su programación. Una IA generativa, tras la realización de una solicitud, es capaz de responder de forma inteligente mediante el uso de texto, audios, vídeos o cualquier otro tipo de formato (dependiendo de su entrenamiento). La fortaleza de una inteligencia artificial generativa se debe al aprendizaje de patrones y otras características gracias a un gran conjunto de datos, los cuales usa para aprender cómo utilizar dicho conocimiento de forma autónoma en cada una de sus respuestas.

Sin embargo, no sólo los modelos GPTs hacen uso de la inteligencia artificial generativa. Existen las llamadas RGA (GAN en inglés) o Redes Generativas Adversarias [16], las cuales se basan en el uso de dos tipos de redes neuronales que, al trabajar unidas, consiguen reproducir información lo más realista posible y difícil de distinguir de la información real.

La inteligencia artificial generativa ofrece un sistema que, por medio de patrones y otras características, proporciona respuestas realistas y con una gran concordancia respecto a la pregunta formulada inicialmente.

1.2.- EMPRESA/ENTORNO DE APLICACIÓN

El presente Proyecto Fin de Grado quiere buscar una solución a las limitaciones que manifiestan diferentes GPTs (Generative Pre-trained Transformers) a la hora de realizar consultas y peticiones de un dominio específico sobre una inteligencia artificial entrenada con datos genéricos extraídos de Internet.

La empresa para la cual se pretende desarrollar el trabajo es el Banco Santander[8]. Dicha empresa opera alrededor del mundo entero, con una gran visibilidad en Norteamérica, México, países de Latinoamérica como Brasil, Chile, Perú entre otros, países de Europa como Reino Unido, Portugal, y por supuesto, España, donde opera como una de las principales entidades financieras del país.

Pese a que su nacimiento se originó en la ciudad de Santander, actualmente, su sede operativa se encuentra en la Ciudad Financiera de la localidad de Boadilla del Monte, ubicada en la comunidad autónoma de Madrid, donde se encuentra todo su centro tecnológico y donde yo trabajo.

El proyecto que se quiere desarrollar será utilizado por el equipo que forma parte del desarrollo de programas para los microservicios que contienen toda la lógica de negocio de la entidad. Dichos desarrolladores no sólo se encuentran en Madrid, ya que para los distintos proyectos dentro de la entidad existe una gran cantidad de consultoras y colaboradores repartidos por toda España, Reino Unido y Latinoamérica.

Debido a las exigencias de seguridad que conlleva trabajar para una entidad financiera, todas las conexiones se realizan a través de máquinas virtuales con conexión a través de VPN (red privada virtual) y triple verificación. Por tanto, ningún empleado del equipo de desarrollo trabaja sobre su máquina física real, sino que lo hace a través de simuladores VDI (infraestructura de escritorios virtuales) totalmente hermetizados con la máquina real donde se ejecuta (nada puede entrar y salir de ella).

El proyecto a realizar irá integrado dentro del gestor de aplicaciones de la empresa, donde cada máquina virtual tendrá la posibilidad de instalar dicha aplicación en su entorno para poder hacer uso de la misma.

También podrá hacerse uso del modelo en los repositorios internos donde desplegar los distintos microservicios de la aplicación y, además de poder realizar consultas para resolver dudas de implementación y desarrollo, tener la oportunidad de poder mejorar el modelo, acorde a las nuevas necesidades que puedan surgir, o las carencias que se vayan detectando.

1.3.- JUSTIFICACIÓN DEL PROYECTO

A lo largo de mi carrera trabajando para Banco Santander en la Ciudad Financiera de Boadilla he compartido cada día conocimientos con mis otros compañeros de trabajo, buscando siempre programar con la mayor limpieza, calidad y eficiencia usando las herramientas disponibles en todo momento.

Antes de la llegada de los GPTs[9], cualquier consulta externa se realizaba en páginas de Internet especializadas en programación, siendo StackOverflow[10] una de las más representativas para este tipo de búsquedas. El problema ocurre cuando queremos conseguir una mayor eficiencia en las búsquedas sin sacrificar la calidad del código, lo cual no es siempre fácil de conseguir. Los manuales de programación disponibles online

presentan un problema recurrente, y es que no dan una respuesta satisfactoria a ciertos problemas genéricos que se suelen repetir con cierta frecuencia, algunos ejemplos de estos errores comunes son:

- El intento de conexión con la base de datos produce un error de dependencias.
- Los Beans están incorrectamente configurados al ejecutar código.
- No se puede lanzar Debug por un problema derivado del Socket.
- etc...

Lamentablemente, en muchas ocasiones, al realizar una de estas búsquedas para tratar de solventar algún problema de la forma más eficaz y eficiente posible, se tiene que la solución encontrada no se acerca a lo que realmente se necesita y requiere de muchas pruebas y retoques antes de tener una versión del código correcta y funcional que pueda ser utilizada y considerarse definitiva.

Con la llegada y difusión de las inteligencias artificiales generativas capaces de escribir código[11], se ha logrado mejorar de forma drástica la calidad de los resultados de las consultas realizadas en busca de una solución ante ciertos problemas de programación, consiguiendo mejor código funcional, con menos errores y en menos tiempo. Se ha logrado que, dando indicaciones muy específicas, un GPT retorne una información que se acerca en gran medida al resultado esperado. Sin embargo, es habitual que estas respuestas no cumplan al 100% con las características específicas del proyecto que se está realizando. Es posible que las respuestas ofrecidas por un GPT sirvan a corto plazo para proyectos dispares, en distintas empresas. El problema ahora está en poder generar código para proyectos más grandes, críticos, donde hay que cumplir con un cierto número de restricciones, y siguiendo una metodología concreta, con un GPT no entrenado específicamente para ello se obtendrán resultados que no serán completamente válidos para un proyecto concreto de estas características. Una posible solución podría consistir en proporcionar al modelo todo el contexto del problema antes de hacer una pregunta específica, pero resultaría una tarea muy tediosa, que habría que repetir cada vez, al arrancar una nueva sesión de preguntas; esta es una solución que se descarta por considerarse poco usable.

1.4.- OBJETIVOS

1.4.1.- Objetivo Principal

Como ya he mencionado, actualmente me dedico a la programación de proyectos del Banco Santander, donde se aplica un riguroso orden en la gestión y el uso de la

información, y un estándar de calidad a nivel de código a todos los niveles en todos los servicios y microservicios. También se siguen unas pautas estandarizadas para desarrolladores y diseñadores del producto y, como ya se ha indicado, se busca la mayor eficiencia a nivel de microservicio, así como limpieza y claridad a nivel de código.

Dada la minuciosa importancia que se da a todos y cada uno de estos puntos, resulta de gran interés tener a mano una documentación estándar para la empresa, como una documentación específica para cada proyecto, donde se deba seguir unos estándares y prácticas de programación mínimas requeridas por el equipo de desarrollo. Sin embargo, una documentación tan extensa puede resultar un problema para aquellas nuevas incorporaciones que necesitan amoldarse lo antes posible a un proyecto. Es aquí donde entraría en juego el uso de un GPT bien afinado.

Existen varios problemas al usar IAs Generativas que se van a abordar con la realización del presente Trabajo Final de Grado:

- Falta de calidad y eficiencia en el código para nuestro proyecto a la hora de realizar consultas específicas para el mismo.
- Garantizar la seguridad de los sistemas a pesar de estar proporcionando detalles de los mismos (contexto) al hacer preguntas al modelo de lenguaje. Este problema se debe al hecho de que para conseguir una respuesta adaptada a la estandarización del proyecto, se deben realizar consultas dando como ejemplo trozos de código o detalles del proyecto. Por seguridad debemos evitar esto a toda costa.

Por tanto, el objetivo principal de este TFG es tratar de integrar en nuestra operativa de desarrollo una IA generativa que, por medio de un ajuste fino, logre aprender de todos los desarrolladores que actualmente trabajan en un proyecto, siendo entrenada por ellos, a la vez que ofrece respuestas coherentes y cada vez mejores para las nuevas consultas de estos desarrolladores de dicho proyecto.

Se debe recalcar que, una vez la IA comprenda la metodología usada por los integrantes del equipo, no se debería necesitar dar información demasiado crítica al GPT ya entrenado, mejorando la seguridad y a su vez la eficiencia, dando resultados esperados sin ser tan específicos en lo que se está trabajando.

1.4.2.- Objetivo Secundario

Como objetivo secundario, se aplicarán diferentes GPTs de entre los disponibles actualmente en el mercado dentro de nuestra aplicación, utilizando sus APIs, de modo que, al realizar una petición, nuestra aplicación solicite una respuesta a dichas APIs para recibir

tantas respuestas como GPTs tengamos incorporados. Esto permitirá comparar los resultados de dichos GPTs, estudiando cuál ofrece una mejor información en cada búsqueda, ya sea de carácter general o consultando algo específico en el área de la programación, siempre orientada a mejorar la calidad del proyecto en curso.

1.4.3.- Objetivo Personal

Son muchos los desarrolladores que han pasado por mi proyecto y se han visto abrumados por la cantidad de documentación que como mínimo deben consultar para poder comenzar a trabajar en los microservicios. Por tanto, mi objetivo personal trata de poder implementar una especie de “*herramienta oficial*” para desarrolladores, de tal forma que pueda ser descargada e instalada en todas las VDI de aquellos que la soliciten y así tener acceso a un GPT personalizado y mejor adaptado para el proyecto en curso.

También se quiere mantener un repositorio donde cada desarrollador pueda subir su versión mejorada dependiendo de cada proyecto y de sus necesidades, dependiendo del desarrollador que utilizará dicho recurso.


1.5.- LÍMITES DEL PROYECTO

Puesto que Banco Santander es una entidad financiera muy importante, es de esperar que todas las aplicaciones deban pasar un estándar de calidad antes de pasar a usarse por desarrolladores. Esto es imprescindible, ya que puede producirse una fuga de datos en una de las consultas si no estuviese bien diseñado y entrenado el GPT que se pretende utilizar. Por tanto, es posible que dicha aplicación no pueda integrarse dentro del Store interno de Santander, aunque se espera que la aplicación esté desplegada a futuro en uno de los microservicios de la empresa, claro está, con posterioridad a la entrega del presente trabajo.

Además, en su versión para la entrega y evaluación académica, no se va a emplear información sobre los proyectos reales de la entidad financiera, por lo que, para mostrar la funcionalidad del modelo se utilizará un proyecto personal, una aplicación propia que consistirá en un conjunto de microservicios, donde uno de ellos será el que recibirá todas las consultas de los usuarios a través de peticiones REST, ya sea para consultas o para el entrenamiento por medio de nuevos datos. El resto de microservicios se encargará de la comunicación con los distintos GPTs implementados, así como un registro de las peticiones realizadas en una base de datos. Las comunicaciones para los registros se realizan por colas para asegurar la integridad de los datos.

Capítulo 2

Antecedentes y estado de la cuestión



2.1.- INTELIGENCIAS ARTIFICIALES GENERATIVAS

2.1.1 Historia de la IA Generativa

En los últimos años se ha visto un importante auge a nivel usuario de la IA Generativa, especialmente tras el lanzamiento y/o revelación por parte de OpenAI, allá por el año 2022, de una herramienta denominada ChatGPT, que tenía la capacidad de dar una mayor facilidad en la elaboración de las tareas diarias, presentando un gran potencial [17][18].

Cronológicamente hablando, se pueden establecer tres etapas o épocas bien diferenciadas en lo que respecta al desarrollo de la inteligencia artificial. La primera etapa se sitúa entre los años 1950 y 1970. En los años 50 se desarrollaron los fundamentos de la IA, momento

en el cual se empezó a explorar la posibilidad de crear una máquina que pudiera simular, copiar y/o actuar con inteligencia humana. Cuando se produjo el desarrollo de la IA y la creación de los algoritmos [19][20].

Al hablar de los inicios de la inteligencia artificial hay que hablar de Alan Turing, considerado uno de los precursores de la IA y creador del juego de la imitación, más conocido actualmente como el test de Turing, el cuál mide la capacidad que posee una máquina para tener una conversación haciéndose pasar por un ser humano [21]. Fue el primero en plantear la cuestión sobre si las máquinas podrían o no pensar, de ahí su artículo escrito en 1950 llamado “*Computing machinery and intelligence*” en el cual se preguntaba ¿pueden las máquinas pensar? [22]

Otro de los precursores en el campo de la inteligencia artificial fue el científico estadounidense John McCarthy, fue precisamente el primero en usar el término “inteligencia artificial”, definiéndola como “*la ciencia y la ingeniería de hacer máquinas inteligentes*” [23]. Estas ideas fueron tratadas durante la conferencia de Dartmouth de 1956, donde también asistieron, entre otros, Nathaniel Rochester, Claude Shannon y Marvin Minsky; este último, junto a McCarthy, crearon el primer laboratorio dedicado a la inteligencia artificial en el MIT.

La siguiente etapa de la inteligencia artificial transcurre entre los años 1970 y los años 1990, a pesar de que los primeros años del surgimiento de las ideas sobre inteligencia artificial fueron de gran esplendor, durante los años 70 y 80 se vivió una fase de estancamiento, las expectativas no se vieron cumplidas ya que la creación de máquinas inteligentes era una labor ardua y altamente compleja. Esto sirvió a su vez para buscar nuevos enfoques y aquellos expertos se dedicaron principalmente al desarrollo de los denominados sistemas de conocimiento y de algoritmos de tipo simbólicos.

La última etapa de la IA iría desde los años 90 hasta nuestros días, esta época se caracteriza por un alto crecimiento y una fuerte innovación en este campo, todo ello fruto a la aparición del aprendizaje automático o ML (por sus iniciales en inglés “*machine learning*”), pero sobre todo a la aparición del aprendizaje profundo o Deep learning, que consiste en la capacidad que poseen los algoritmos para imitar del cerebro humano, mediante redes neuronales artificiales de múltiples capas [24]

2.1.2 Funcionamiento de la IA Generativa

La posibilidad de que una máquina pueda generar ideas a partir de cierta información recopilada, ha supuesto un gran avance debido a las múltiples perspectivas de desarrollo que manifiesta este tipo de creación autómatas, y esa es la idea detrás del concepto de IA generativa.

Las tecnologías de IA al ser tan complejas y basarse en redes neuronales, ya que éstas pretenden en cierta medida imitar la inteligencia humana, pueden prepararse para aprender distintas materias, ya sean idiomas, física, química, programación u otras disciplinas con mayor complejidad. Todo lo aprendido forma parte de un conjunto de bases de datos que pueden ir reutilizando para poder solventar nuevas cuestiones que se les presente. A partir de unos datos iniciales empleados durante el entrenamiento, se generan nuevos datos y estos a su vez sirven para un entrenamiento adicional que puede ayudar a la resolución de futuros nuevos problemas. [25]

Actualmente, las IAs generativas usan distintas disciplinas para su aprendizaje, una de las más conocidas es el Deep Learning, el cual mantiene la esencia del machine learning pero adaptada a técnicas de aprendizaje de inteligencia artificial actuales. [26]

Los sistemas neuronales en los que se basan las IAs generativas, están estructurados en múltiples capas imitando en alguna medida al modo en que las neuronas biológicas se organizan en el cerebro humano para procesar la información. Existen dos tipos de modelos principales para abordar esta cuestión, las redes generativas antagónicas o GANs, y los transformadores o transformers.

2.1.3 Redes Generativas Antagónicas

Las redes generativas antagónicas (GAN) están compuestas por dos tipos de redes que se encuentran en una continua confrontación, ya que una de ellas se dedica a la producción de nuevos contenidos, mientras que la otra trata de determinar la legitimidad de dichos contenidos. [27]

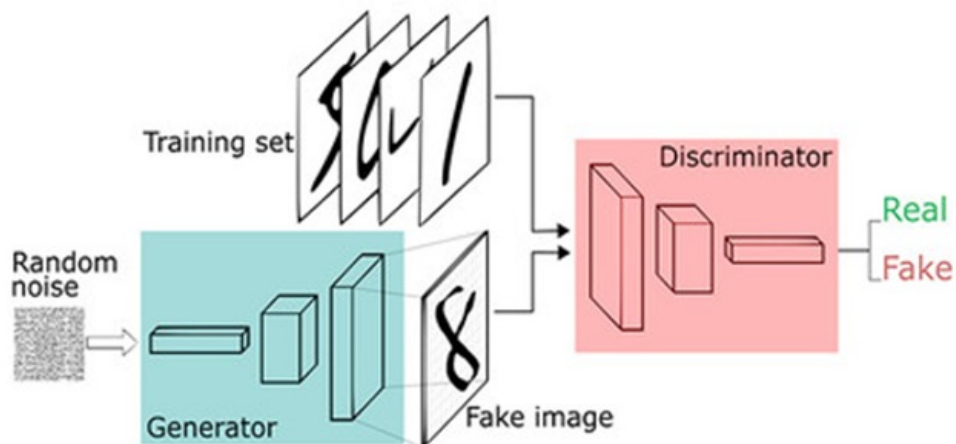


Figura 2.1.- Proceso Generador - Discriminador GAN

Como se observa en la figura 2.1, por un lado está la información utilizada en el entrenamiento (“*training set*”), y por otro la información que produce una de las redes (“*generator*”), ambos conjuntos de datos pasan a través de la red que verifica la información (“*discriminator*”) y la clasifica según si es verdadera o falsa. No obstante, el uso de las GAN tiene ciertos inconvenientes entre los que destacan los siguientes [28]:

- Colapso modal.- Los datos producidos por el generador deben presentar una cierta variabilidad, pero en un colapso modal, el generador alcanza un modo en el que consigue “engañar” al discriminador con pocas salidas y repite este proceso, reduciendo la complejidad de las muestras, y con ello produciendo un pobre entrenamiento para el modelo final.
- Convergencia.- Resulta complejo que, tanto el generador como el discriminante converjan a un punto donde ambos tengan un rendimiento óptimo, a veces el generador y el discriminador continúan oscilando sin alcanzar un equilibrio.
- Calidad.- Existen problemas para poder saber cuándo las imágenes creadas por el generador son de alta calidad.
- Métrica.- El adecuado procedimiento entre el generador y el discriminador no siempre termina resultando en una adecuada calidad o en una apropiada diversidad de estos resultados.

2.1.4 Modelos de Transformadores

Los modelos transformadores son un tipo de red neuronal que son capaces de aprender tanto el contexto como el significado de la información que reciben como parámetros rastreando las posibles relaciones en referencias secuenciales, como si de las palabras de una oración se trataran. Esto lo hacen mediante unas técnicas algorítmicas que tienen como fin localizar dependencia entre datos y que tipo de influencia tienen. [29][30]

La primera reseña que se tienen de los modelos de transformadores data del año 2017, son una de las últimas clases de redes neuronales y con más potencia, siendo la base del nuevo desarrollo en machine learning, obteniendo el apodo de IA de transformer.

2.1.4.1 Arquitectura Codificador-Decodificador de modelos transformer

La arquitectura Encode-Decoder es el motor que hace posible tareas como la traducción automática entre idiomas donde, a raíz de una entrada (p.e.: texto inglés) se genera una salida (p.e.: texto español). Esta tarea puede extrapolarse a muchas otras tareas, como la respuesta a peticiones por medio de la información de un Modelo de Lenguaje Grande cargado [43]. También es posible utilizar esta arquitectura en modo de entrenamiento, usando el campo TARGET para indicar cómo debe de ser la salida final esperada.

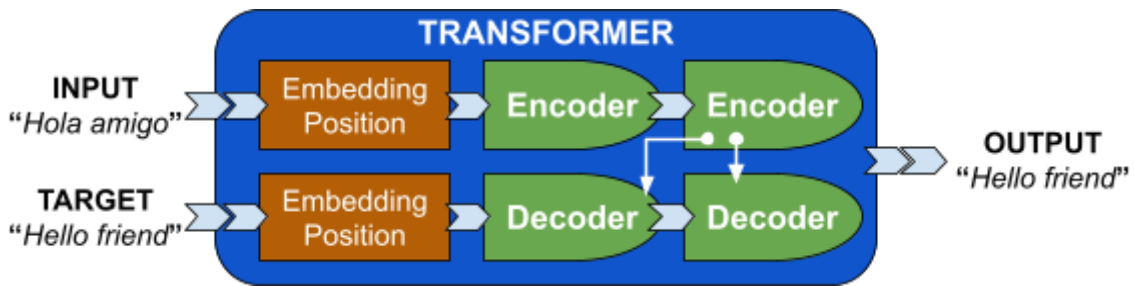


Figura 2.2.- Diagrama Codificador - Decodificador

Como se observa en la figura 2.2, y como indica el propio nombre de este transformador, hay dos componentes fundamentales en esta arquitectura, el codificador (“*encoder*”) y el decodificador (“*decoder*”):

- **Codificador.**- El trabajo del codificador consiste en convertir los parámetros de entrada en una representación vectorial de dicha entrada, clasificando fragmentos de éstos conforme se analiza. Dicho codificador no se encarga de generar ningún tipo de salida, únicamente tokeniza la entrada para poder centrarse en las partes más importantes de ésta y así comprender su contexto.
- **Decodificador.**- Una vez la información ha pasado y ejecutado dentro del codificador, pasa a generar los parámetros de salida por medio de la decodificación. Por medio de los vectores fijos que se han creado con los parámetros de entrada, se buscan similitudes en el modelo conforme se van recorriendo los vectores, de modo que se obtiene una salida que se va generando conforme se avanza por dichos vectores, siempre buscando asociaciones dentro de la matriz del modelo.

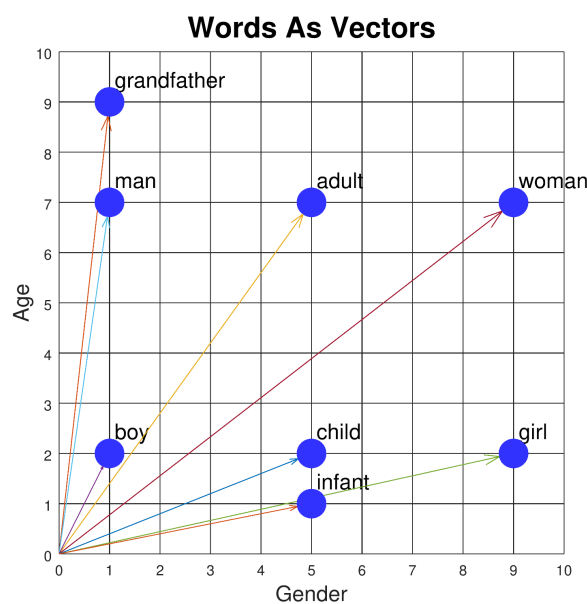


Figura 2.3.- Representación Vectorial de parámetros

2.1.4.2 Desventajas de la arquitectura transformer

Pese a que la arquitectura transformer basada en un modelo de codificación-decodificación es muy potente, esto no lo exime de limitaciones de cómputo, sobre todo a nivel de hardware. El alto costo computacional aumenta de forma exponencial conforme se aumenta la longitud de la secuencia. Esto hace que en sistemas con pocos recursos no se pueda usar parámetros de entrada extensos si se requieren unos tiempos de respuesta aceptables.

También es posible que, dependiendo de los parámetros de entrada, exista dificultad a la hora de interpretar y depurar toda la secuencia de entrada. Además, si se desea realizar ajustes en el modelo, es posible que se produzca un sobreajuste si se enfoca a datos específicos, lo que provocaría resultados erróneos o no esperados en otras casuísticas que no tengan nada que ver con lo solicitado.

Pese a todo ello, la arquitectura transformers ha permitido un gran avance en la clasificación de textos, así como traducción y generación de idiomas, es por ello que se ha convertido en el estándar para muchos modelos de Procesamiento de Lenguaje Natural.

2.2.- MODELOS GRANDES DE LENGUAJE

2.2.1 Definición de Modelo de Lenguaje Grande (LLM)

Los Modelo de Lenguaje Grandes o LLMs (por sus iniciales en inglés “*Large Language Models*”) están basados en redes neuronales profundas (con múltiples capas), simulando el funcionamiento de las neuronas del cerebro. Son un tipo de algoritmo que, mediante el uso de aprendizaje profundo, logra realizar una gran cantidad de funciones basadas en el procesamiento de lenguaje natural. Dichos modelos están entrenados con una enorme cantidad de datos que, unidos a un modelo de transformadores, logran generar texto o cualquier otro contenido mediante predicciones, reconocimiento y traducción de datos.[31]

Existen multitud de aplicaciones donde es posible emplear los LLMs para realizar importantes tareas en el ámbito científico, como puede ser el estudio de proteínas o generar un código completo de un programa informático. Sin embargo, para poder llegar hasta este punto, se debe realizar un pre-entrenamiento de dicho modelo y realizar un ajuste posterior para adaptarlo a las necesidades propias del ámbito de aplicación.

Un LLM bien entrenado podrá ayudar a la resolución de problemas dentro de aquellos campos para los que ha sido creado, ya sea en el sector de la medicina, informática o simple entretenimiento. En todos ellos cumple la utilidad para una gran variedad de aplicaciones de procesamiento de lenguaje natural, aprendiendo mediante la recopilación de parámetros a modo de conocimientos para dicho modelo.

2.2.2 LLMs y Procesamiento de Lenguaje Natural

Para que un LLM funcione es importante lograr una correcta comunicación entre la IA y las personas que pretenden interactuar con ella. Esto no sería posible sin el Procesamiento de Lenguaje Natural [32][33] o PLN (también llamado a veces NLP por su acrónimo en inglés “*Natural Language Processing*”).

La sociedad en general está acostumbrada a las aplicaciones en las que un PC u otro dispositivo es capaz de comprender cuando se le indican ciertas órdenes o instrucciones por voz o por escrito, y con frecuencia no existe conciencia de la IA que hay detrás para que dicho lenguaje humano sea correctamente tratado e interpretado. Por tanto, se entiende por Procesamiento de Lenguaje Natural (PLN) a la rama informática donde un sistema informático es capaz de comprender a los seres humanos, bien por escrito, o bien de forma hablada.

Hay de entender que el lenguaje humano está lleno de dificultades a la hora de ser descrito formalmente, ya que se debe diferenciar entre expresiones metafóricas, sarcasmos, ironías y todo tipo de variaciones idiomáticas que incluso pueden resultar confusas para un ser humano, por lo que un desarrollador de PLN debe enseñar a sus aplicaciones reconocer e interpretar correctamente todas estas situaciones para que sean de utilidad. Algunas de las tareas principales de un sistema PLN son las siguientes:

- **Reconocimiento de voz.**- El reconocimiento de voz es la habilidad que tiene un sistema para transformar los datos de voz en datos de texto para poder ser procesados. Es muy difícil de implementar ya que no todas las voces son iguales ni tampoco se habla a la misma velocidad, sin contar los acentos y las incorrecciones gramaticales.
- **Etiquetado gramatical.**- Trata de la habilidad del sistema en reconocer la lógica de una palabra concreta en lo que respecta a su uso y al contexto en el que aparece dicha palabra, por ejemplo, como reconocer si la palabra “rojo” es un sustantivo o un adjetivo dentro de la frase.
- **Análisis de significado de palabra.**- Cuando se identifica una palabra que pueda tener varios significados se intenta, por medio del análisis de la frase, qué

significado concuerda más con dicha palabra. Un ejemplo podría ser la palabra “bajo”, que puede entenderse como un instrumento musical, una planta baja o un verbo.

- **Reconocimiento de entidades.**- Es muy usado cuando se utilizan iniciales a modo de diminutivos. Un ejemplo es ESP para España o CHL para Chile.
- **Referencias entre palabras.**- Cuando se tiene una frase donde se usan pronombres y nombres propios, el sistema debe tener la habilidad de reconocer si dichos pronombres se refieren a un nombre propio concreto que aparezca en la frase e incluso diferenciar entre distintos actores por medio de estos pronombres.
- **Extracción de cualidades subjetivas.**- El sarcasmo, la ironía o el uso de otras emociones en la escritura es muy común en el habla humana, pero más complicado para una IA. Sin embargo, ésta debe ser capaz de detectarla.
- **Generar lenguaje.**- Por último, el sistema debe de ser capaz de generar una salida una vez haya procesado la información que ha recibido como parámetro de entrada. Es de vital importancia, ya que esto permitiría la comunicación entre humano y máquina.

2.2.3 Componentes de un Modelo de Lenguaje Grande

Como anteriormente se indica, los LLMs se componen de varias capas de redes neuronales, las cuales se unen para realizar distintas tareas como el procesamiento de los datos de entrada con los que posteriormente se generará un determinado contenido a modo de respuesta.

2.2.3.1 Capa de Incrustación

La función de esta capa consiste en transformar palabras, frases o cualquier dato en numeraciones que relacionen su significado. La idea de este método es crear un espacio multidimensional, de modo que los datos que tengan más relación entre ellos se encuentren más cercanos, lo que permite al modelo tener acceso a toda la información relacionada de una forma sencilla y rápida.[34]

Existen muchos tipos de incrustación, los más comunes consisten en el tratamiento de palabras y frases, pero también pueden usarse imágenes, donde se buscan patrones para buscar coincidencias en el contenido visual y buscar formas similares.

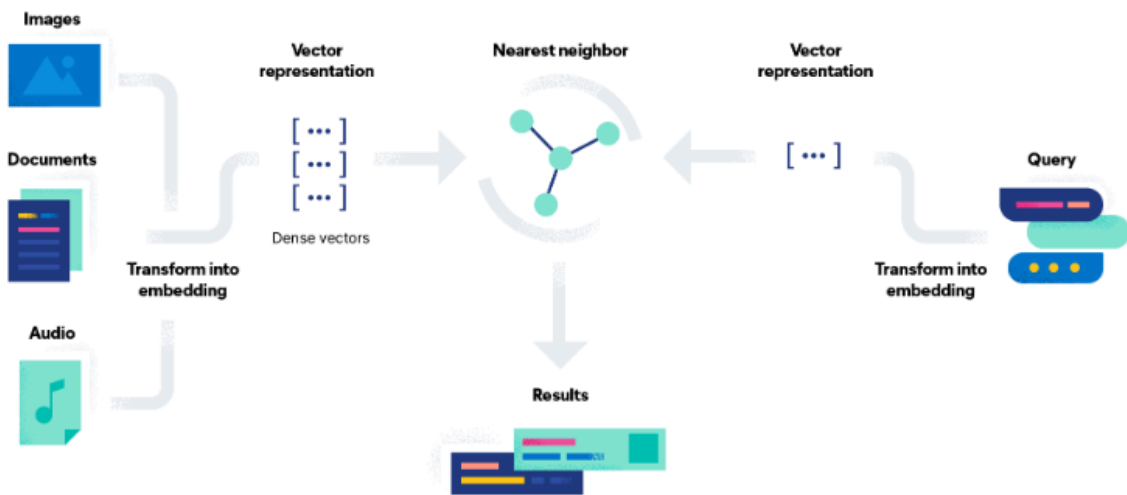


Figura 2.4.- Búsqueda de similitudes por medio de vectores

Los tipos de incrustaciones más comunes que se pueden aplicar son:

- **Incrustaciones de palabras:** Es la más sencilla, cada palabra se representa como un vector, de modo que se pueda captar relaciones semánticas con otros vectores y la información que pueda estar relacionada con éstas.
- **Incrustaciones de oraciones:** Mantiene la misma esencia que la incrustación de palabra, pero usando una oración completa.
- **Incrustaciones de documentos:** En este caso, los vectores están formados por todo aquello que contenga dicho documento, lo que resulta en una complejidad extra con respecto a la incrustación de oraciones.
- **Incrustaciones de imágenes:** Consiste en extraer patrones representativos de una imagen y transformarlos en vectores que se puedan comparar con similitudes en patrones extraídos de otras imágenes para buscar similitudes entre dichas imágenes.
- **Incrustaciones de usuarios:** Dicho tipo de incrustación es ampliamente utilizado en estrategias de Marketing, ya que vectoriza todos los usuarios de un sistema concreto usando sus atributos tanto físicos como los relacionados con su comportamiento dentro de dicho sistema.
- **Incrustaciones de producto:** Al igual que la incrustación de usuarios, la incrustación de productos se realiza captando todos los atributos de dicho producto, así como cualquier información que pueda estar relacionada y la transforma en vectores. Es importante a la hora de realizar comparaciones y recomendaciones de estos productos al usuario.

Para convertir cualquier información en vectores numéricos se utilizan ciertas técnicas de Machine Learning, comenzando por recopilar un gran conjunto de datos para el que se desea crear incrustaciones. Estos datos se deben limpiar eliminando cualquier ruido que pueda desvirtuar la información, normalizando el texto que lo contiene, estandarizando el tamaño de las imágenes, etc.[34]

Llegados a este punto y conociendo bien el objetivo final, se seleccionará un modelo de red neuronal, de modo que realice pequeños ajustes en todos los parámetros dando como resultado el aprendizaje de patrones y relaciones de todos los datos.

Las incrustaciones se irán generando conforme el modelo aprende, obteniéndose un vector único representando los datos. En este punto ya se puede comprobar la calidad y eficiencia de las incrustaciones generadas en aquellas tareas específicas para las que se han creado.

2.2.3.2 Capa Feedforward Neural Network (FNN)

En el caso de la capa FNN, se cuenta con varias subcapas para transformar las incrustaciones de la capa de incrustación. La principal finalidad es tener una mayor comprensión de los datos de entrada introducidos por el usuario [35]. Las subcapas de una arquitectura de red neuronal Feedforward son las siguientes:

- **Capa de entrada.**- En ella el usuario introduce los parámetros de entrada, donde existe una red neuronal de tamaño equivalente a los datos de entrada.
- **Capa oculta.**- En dicha capa es donde se aloja el motor que realiza toda la computación de la red neuronal. Con ella, se adapta los parámetros de salida de las capas de entrada mediante una ponderación de los datos antes de pasar la información a la siguiente capa.
- **Capa de salida.**- Como bien dice su nombre, se construye una salida con respecto a los datos de entrada dados por las capas anteriores.

Es importante entender que cada neurona de una capa está interconectada con todas las neuronas de su siguiente capa y viceversa. Estas conexiones tendrán mayor o menor relevancia dependiendo de los pesos que tengan asignados.

2.2.3.3 Capa Recurrente

En esta capa se analiza cada parámetro de entrada de forma secuencial buscando la comprensión que las relaciona entre ellas. Cada neurona recibe como entrada dos

parámetros, el correspondiente de los datos de entrada propiamente dichos, y la suma ponderada de las salidas de las neuronas anteriores, las cuales han analizado sus parámetros del mismo modo.[36]

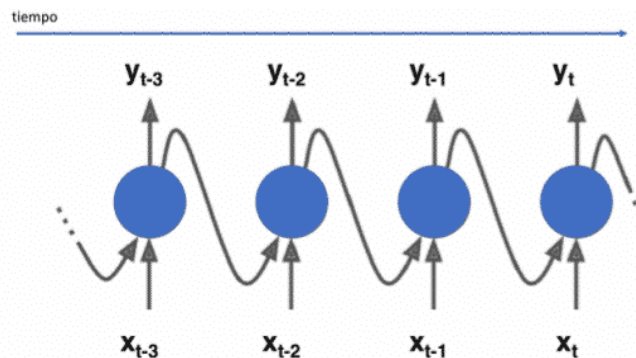


Figura 2.5.- Dependencia entre neuronas entrada - salida

Una vez terminada la secuencia, se obtiene como respuesta una interpretación de los datos fiable para su uso en el modelo.

2.2.4 Funcionamiento de LLM

Como se ha visto, un LLM recibe parámetros de entrada, los codifica y posteriormente realiza una decodificación produciendo una salida por medio de predicciones, pero para lograrlo, se debe realizar un entreno para que pueda realizar funciones genéricas y hacerle un ajuste a posteriori si se quiere que se enfoque en tareas específicas.[31]

- **Capacitación.-** Un LLM debe obtener un enorme banco de datos con el cual realizar un pre-entreno. Dichos datos pueden estar compuestos por miles de millones de palabras, lo cual puede influir en el rendimiento del modelo (cuantos más datos mejor, pero estos deben ser correctos para un pre-entreno satisfactorio). El pre-entreno se realizará de forma no supervisada, de modo que el algoritmo aprenderá a relacionar unas palabras con otras, así como los significados de cada una para aprender diferencias elementales (como la diferencia entre el aceite de cocina y el aceite de motor).
- **Ajuste.-** El ajuste del modelo es indispensable para que éste pueda realizar tareas enfocándose en actividades específicas. Una forma eficaz de hacerlo es mediante el ajuste de solicitudes. Para realizar un entrenamiento por medio de ajuste de solicitudes, se debe cargar un alto número de ejemplos de tipo entrada/salida (petición/respuesta), de modo que para una petición similar, se logrará que la respuesta esté adaptada a los requisitos y definiciones indicados en la respuesta. Dichos ajustes tienen muchísima potencia, ya que, además del entrenamiento

específico, se puede enseñar al modelo estados de aprobación y desaprobación respecto a comentarios de halago u otros despectivos para que tenga un comportamiento más adecuado.

```
{
  "instruction": "Clasifica entre animales, plantas y minerales",
  "input": "Secuoya, Aluminio, Elefante",
  "output": "Animales: Elefante\nPlantas: Secuoya\nMinerales: Aluminio"
}
```

Figura 2.6.- Ejemplo JSON Ajuste Fino

2.2.5 Usos y aplicaciones de los LLMs

Desde que comenzó a utilizarse, son muchos los usos que se le han dado a los LLMs [31]. A continuación se destacan los más importantes:

- **Recopilación y búsqueda de información.**- Sin ir más lejos, dos de los buscadores más famosos de los últimos años que son Google y Bing, dependen de un LLM Grande para responder la información adecuada respecto a las búsquedas realizadas por los usuarios.
- **Análisis de sentimientos.**- Aunque el concepto puede confundir, es importante para el sector empresarial poder clasificar datos por sentimientos. Esto es debido a que los datos que se manejan pueden tener un índice de aprobación o desaprobación para el usuario medio, por lo que es importante conocer esta información para generar una respuesta lo más adecuada posible al usuario.

Ejemplo:

Aprobación: Persona de color. Persona de religión musulmana.

Desaprobación - Persona negra. Persona mora.

- **Generador de texto.**- Este uso es el más conocido por el usuario medio. Ya que se trata de los LLMs detrás de las inteligencias artificiales generativas. Es aquí donde entran herramientas como ChatGPT o Copilot, las cuales generan texto de salida en base al texto de entrada. Se han hecho muy populares estos últimos años ya que pueden generar cualquier tipo de texto con gran precisión, ya que están entrenados por medio de gran cantidad de información recopilada en Internet.
- **Generador de código.**- Aunque las herramientas mencionadas en el punto anterior pueden realizar funciones de generación de código, existen modelos específicos para realizar consulta de código a nivel de programación, de modo que se puede

crear cualquier programa 100% usable con sólo solicitar unos parámetros y unas características concretas.

- **Uso en chats de IA.-** Los LLM permiten interactuar con el usuario por medio de una conversación fluida, en la cual se puede obtener una información a modo de servicio. Esto se consigue interpretando la conversación a modo de parámetros de entrada para posteriormente interpretar y ofrecer respuestas a éstas.

Gracias al uso que se le da a los LLMs con los fines mencionados anteriormente, una gran cantidad de campos se han visto beneficiados hasta el punto de llegar a facilitar investigaciones, desarrollos y servicios. Se enumera alguno de ellos:

- **Sector tecnológico.-** Donde se puede usar como asistente para programadores en grandes empresas o en motores de búsquedas.
- **Ciencia y medicina.-** Donde es importante a la hora de desarrollar nuevos fármacos y realizar investigaciones para la búsqueda de curas de enfermedades. Es muy importante el uso de técnicas de Machine Learning para detectar a tiempo un diagnóstico.
- **Servicio al cliente.-** Donde muchas empresas usan un chat con IA para intentar resolver un problema antes de poner en contacto al usuario con un operador humano.
- **Marketing.-** A la hora de realizar campañas, es importante tener a mano una gran cantidad de datos recopilados de los que extraer información mediante IA y tomar ciertas estrategias.
- **Sector bancario.-** El cual es uno de los más importantes, ya que estudia por medio de patrones de conducta de clientes si una transacción puede ser o fraudulenta.

El uso de estos LLMs está en pleno auge y el rendimiento de éstos están mejorando continuamente ya que se están constantemente agregando nuevas mejoras. Además, tienen la habilidad de aprender de manera continua sin introducir parámetros adicionales por medio de la técnica few-shot prompting.[44]

2.2.6 Inconvenientes de los LLMs

Los LLMs son extremadamente útiles en el sector tecnológico, sin embargo, no están exentos de limitaciones que suponen un gran desafío para quienes desean hacer uso de ellos en distintos ámbitos [31]:

- **Ilusión de datos o alucinaciones.**- Es posible que en algunos casos, durante la conversación se produzca un desvío del tema debido a que la IA comienza a responder erróneamente. Esto se debe a que se usan predicciones a la hora de elaborar la respuesta que, pese a ser sintácticamente correcta, no tenga sentido debido al mal aprendizaje a la hora de clasificar información. Es un problema muy conocido en IAs como cleverbot.



Figura 2.7.- Ejemplo uso Cleverbot

- **Problemas de seguridad.**- Normalmente, los LLMs aprenden de las consultas e iteraciones que se producen al usarlos. Si se solicita que resuelva un problema muy concreto, especificando todos y cada uno de los puntos de aquel problema que se desea resolver seguramente nos lo solucione, pero usará todos los parámetros de entrada que se le indicó para ayudar a otros usuarios. Con el tiempo y con consultas concretas, es posible que estas IAs acaben ofreciendo dichos datos, potencialmente privados, como respuesta a otro usuario.
- **Sesgo de datos.**- Para realizar consultas al modelo y que éste responda de forma coherente y acertada, se debe tener una gran cantidad de volumen de datos con la que poder entrenar el modelo. Esto conlleva también tener una diversidad de datos, con los que tener una base sólida a la hora de ofrecer una respuesta.
- **Consentimiento externo.**- Al entrenar un modelo con billones de datos, puede que gran parte de ellos no se hayan obtenido bajo consentimiento. Es un gran problema para aquellos modelos que aprendan de datos de Internet en tiempo real, ya que estos pueden tener derechos de autor que la IA ignora por completo. Al generar una respuesta por medio de esta gran cantidad de datos, no es posible descubrir la procedencia de aquellos que han sido utilizados para su construcción y por tanto, se

vuelve irrastreable, lo que conlleva problemas de infracciones por derecho de autor, sin contar aquellos datos personales que podrían estar expuestos.

- **Escalado de datos.**- Cuanto más datos hay en Internet, más difícil es tener un modelo que se entrene abarcando tal cantidad de información.

2.3.- HERRAMIENTAS DE CÓDIGO ABIERTO

Debido al auge que se vive con la llegada de la IA a la sociedad actual son muchas las herramientas orientadas a la creación de modelos que están emergiendo de la mano de empresas privadas para sacar el máximo partido de dicha tecnología y estar a la vanguardia. Sin embargo, todos pueden acceder de forma gratuita a dichos modelos gracias a herramientas de código abierto que permiten construir una IA propia. Dos de estas herramientas son Mistral AI o Llama.

2.3.1 Mistral AI

Mistral AI es una empresa que ha emergido recientemente con la intención de crear modelos de inteligencia artificial que ofrecen un gran rendimiento a cambio de un bajo coste de recursos, por lo que se brinda la eficiencia en todos y cada uno de sus desarrollos. Esto permite a cualquier usuario con recursos relativamente escasos poder crear modelos personalizados para sus necesidades particulares sin tener que depender de empresas como Google u OpenAI, las cuales son las principales competidoras de Mistral.[37]

Mistral ya ha ofrecido de forma gratuita varios Modelos, entre los que destaca el primero de ellos: Mistral 7B. Dicho modelo de 7 billones de parámetros, pese a ser el menos potente, es capaz de competir con otros modelos como GPT 3.5 o Llama, el cual posee el mismo rendimiento en su versión 34B de 34 billones de parámetros, lo cual se revisará más adelante.

Tal rendimiento es un logro para la startup Mistral, ya que se consiguió en tiempo récord y con una financiación relativamente escasa. Esto hizo que no se detuvieran en ese punto y sacasen su segundo modelo, el Mistral 8x7B con un rendimiento muy superior.

- **Mistral 7B.**- Mistral 7B fue el primer modelo creado por Mistral AI. Se creó con algo más de 7 billones de parámetros y está optimizado para ser un asistente en el ámbito de la programación. Si se requiriese que realizar una comparación directamente con Llama, estaría al nivel de Llama 1 34B y codeLlama 7B.[38]

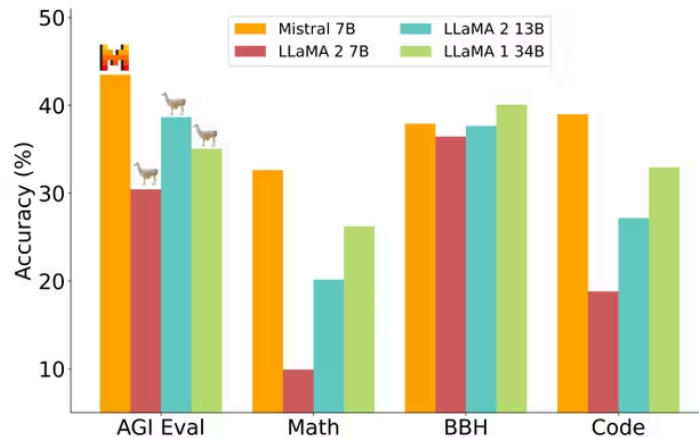


Figura 2.8.- Comparación entre modelos Mistral - Llama

- Mistral 8x7B.**- Es una evolución del Mistral 7B, el cual posee una eficiencia superior al Llama 2 70B e incluso al GPT3.5 en la mayoría de pruebas que se han realizado. Dicho modelo es multilingaje y si se ajusta adecuadamente es uno de los mejores modelos generando código para programadores. La eficiencia de Mistral 8x7B se consigue gracias a la acotación de la información dependiendo del tipo de tarea que deba realizar, adaptándose continuamente para enfocarse sólo en aquellos sectores del modelo que verdaderamente son útiles para su ejecución.[38]

	LLaMA 2 70B	GPT - 3.5	Mistral 8x7B
MMLU (MCQ in 57 subjects)	69.9%	70.0%	70.6%
HellaSwag (10-shot)	87.1%	85.5%	86.7%
ARC Challenge (25-shot)	85.1%	85.2%	85.8%
WinoGrande (5-shot)	83.2%	81.6%	81.2%
MBPP (pass@1)	49.8%	52.2%	60.7%
GSM-8K (5-shot)	53.6%	57.1%	58.4%
MT Bench (for Instruct Models)	6.86	8.32	8.30

Figura 2.9.- Eficiencia modelo Mistral

- Otros modelos de Mistral.**- Además de estos modelos de referencia. Mistral AI ha creado varios modelos que sólo están disponible bajo el API de Mistral u otros; Small, Medium y Large.
 - Mistral Small.- Es una versión del Modelo Mistral 8x7B el cual se optimizó para la plataforma Mistral, pero también para algunas otras como AWS o Azure de forma exclusiva.

- Mistral Medium.- Tiene un rendimiento entre GPT 3.5 y GPT4, es multi idioma y ofrece un mejor rendimiento que Mistral Small.
- Mistral Large.- Es la joya de la corona de Mistral, ofrece un rendimiento casi similar a GPT4, teniendo como idioma nativos el inglés, español, francés, alemán e italiano. Es capaz de seguir varias instrucciones con gran precisión y comprender gramaticalmente documentación de gran tamaño.

2.3.2 Llama AI

Llama AI es un conjunto de modelos creados por Meta (Facebook) con el fin de responder a aquellos modelos que han tenido gran impacto en estos meses como puede ser GPT. Su principal finalidad no es la de ser un ChatBot público, sino que se creó como un conjunto de modelos de código abierto para todos aquellos que soliciten su acceso [39]. Sin embargo, Llama en sí se trata de un modelo básico de propósito general el cual requiere muchos recursos a nivel de hardware para poder hacerlo funcionar localmente, además de requerir ajustes adicionales mediante entrenamientos.

Actualmente se cuentan con versiones de Llama de 7, 13, 33 y 65 billones de parámetros, siendo cada una más potente a mayor número de parámetros pero también consumiendo cada vez mayor cantidad de recursos a cambio. Posteriormente se crearon variantes para cubrir tareas específicas o para mejorar la eficiencia de los modelos básicos. Sus nombres son Alpaca, Vicuna, Koala y GPT4-x-Alpaca.

- **Alpaca**.- La idea principal a la hora de crear Alpaca es la de darle un comportamiento mucho más similar a ChatGPT, logrando conversaciones mucho más fluidas y coherentes. Para ello se realizaron pequeños ajustes en las prioridades de los parámetros de Llama 7B y 13B, naciendo de este modo los modelos Alpaca 7B y 13B respectivamente. El ajuste fino en Alpaca se realizó mediante peticiones a GPT3 y tomando la respuesta de este último como ejemplos para dicho ajuste, por lo que se acaba teniendo un rendimiento en las respuestas similar con un modelo 26 veces más pequeño.
- **Vicuna**.- Este modelo se ha entrenado mediante conversaciones compartidas de un gran número de usuarios en la web ShareGPT.com, logrando de este modo conversaciones que imitan el comportamiento de ChatGPT de forma más óptima que Alpaca. Para evaluar el propio rendimiento de Vicuna se utilizó GPT4, al cual se le solicitó preguntas difíciles a las cuales respondería Vicuna. Las respuestas de este último volverían a pasarse a GPT4 para evaluar su rendimiento.

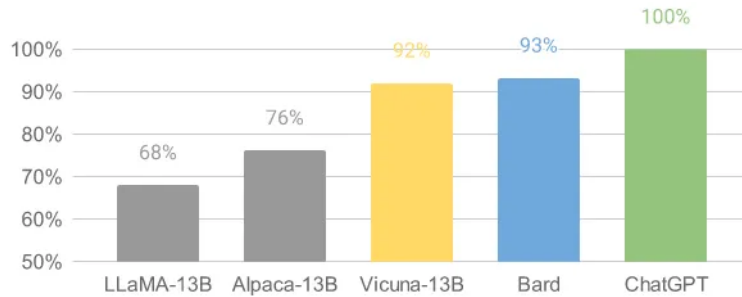


Figura 2.10.- Calidad comprensiva Llama vs ChatGPT

- **Koala.**- Al igual que Alpaca, Koala es una evolución del modelo Llama 7B y 13B, pero este ha sido ajustado con diálogos publicados por un equipo de la Universidad de Berkeley. De esta evolución nacieron los modelos Koala-all y Koala-Distill, de los cuales el segundo utiliza un conjunto de datos mucho más ajustado que el primero, sin embargo, éste ofrece un mayor rendimiento, lo que significa que la calidad de datos utilizados es superior.
- **GPT4-x-Alpaca.**- Dicho modelo se basa en Llama 13B y ha sido entrenado por medio de charlas con GPT4, las cuales son posteriormente evaluadas a fin de alcanzar el mejor rendimiento posible frente a ChatGPT4. Actualmente es el que ofrece mejor rendimiento.[40]

Hasta aquí, se han descrito los modelos basados en la primera versión de Llama. Llama 2 es una evolución de los modelos de Llama 1 originales, lanzándose varios modelos tanto básicos como de tipo Chat, todos orientados a la rama de investigación o para uso comercial de forma gratuita [41][42]. La principal finalidad de dicha evolución es poder lograr una mejora de rendimiento en los modelos más pequeños, manteniéndose en 7, 13 y 70 billones de parámetros. De este modo, pequeñas empresas y comunidades de investigación pueden implementarlos sin realizar una gran inversión en infraestructura y con un tiempo de computación adecuado. Las principales diferencias con los modelos Llama 1 son:

- **Mayor número de tokens.**- Lo que quiere decir que el modelo podrá tener mucha más capacidad de recordar a la hora de mantener una conversación, permitiendo una mayor complejidad, fluidez y coherencia en el habla.[41]
- **Accesibilidad.**- Llama 2 está disponible para cualquier que lo desee, al contrario que Llama 1 que sólo está disponible para investigación.
- **Sólida capacitación.**- Llama 2 está entrenada con una mayor base de conocimiento y usando retroalimentación humana para reforzar dicho aprendizaje, logrando una mayor comprensión contextual.

Al igual que Llama 1, esta segunda versión, Llama 2, también presenta una variedad de modelos básicos denominados Alpaca, Vicuna, Orca, WizardLM y la específica CodeLlama.[41]

- **Alpaca.**-Solo está únicamente disponible en 7B ya que se basa en dicho modelo básico. Dicho modelo está perfeccionado para lograr un resultado similar a GPT-3.5 con mucho menos recursos.
- **Vicuna.**- Para esta versión se usó el modelo básico 13B entrenado con conversaciones de ShareGPT.com, consiguiendo un rendimiento superior a Alpaca por la mitad de recursos.
- **Orca.**- Entrenado por Microsoft, de modo que usa el modelo de Llama 2 básico más grande para poder generar ejemplos con los que poder entrenar al modelo más pequeño que éste.
- **WizardLM.**- Utiliza la técnica Evol-Instruct para generar una gran cantidad de datos para su entrenamiento y lograr un rendimiento superior a GPT-3.5 en más de la mitad de las evaluaciones.
- **CodeLlama.**- Dicha Variante ha sido optimizada para la generación de código de programación. Es extremadamente potente ya que cuenta con conocimiento de Python, C, Java, PHP, entre otros y cuenta con una longitud de hasta 100 mil tokens. Está disponible en modelos de 7, 13 y 34 billones de parámetros. Además, cuenta con dos variantes, las cuales están ajustadas para código Python y para el seguimiento de instrucciones.

Capítulo 3

Hipótesis de trabajo

3.1.- MODELOS LLM DE MISTRAL

3.1.1.- Mistral-7B-Instruct-v0.2

Se trata de un Modelo de Lenguaje Grande, siendo una versión mejorada de Mistral-7B-v0.1. Es un modelo de tipo Transformers, consta de grandes bloques de información que se codifica-decodifica para el procesamiento de sus datos [49]. Los cambios más visibles en comparación con Mistral-7B-v01 son:

- La posibilidad de usar 32k en contexto (rango a la hora de procesar un fragmento de texto) frente a un contexto de 8k en v0.1.
- RoPE- θ de 1E6. Esta técnica se utiliza para entender el orden de las palabras de una oración a la vez que se va procesando, por lo que en este caso, el modelo es capaz

de rotar un millón de veces por todas las palabras introducidas para no perder su contexto.

- "No Sliding-Window Attention", característica que permite calcular todas las palabras que se van a generar en un solo paso, evitando redundancias y mejorando la eficiencia de forma significativa.

Dicho modelo básico de Mistral-7B instruct se usará para realizar un ajuste fino explicado en el **Anexo I.4** y realizar comparativas una vez se haya entrenado.

3.1.2.- Mistral-7B-Instruct-v0.2-GPTQ

Se trata del mismo modelo que Mistral-7B-Instruct-v0.2 pero cuantificado, lo cual quiere decir que ha sido convertido a un formato específico para una eficiente implantación en CPUs y GPUs específicas para IA. Además, con esto conseguimos reducir la complejidad del modelo a cambio de no sacrificar apenas la eficiencia de éste, lo que quiere decir que puede ejecutarse en máquinas modestas con un rendimiento superior al modelo base.[50]

La cuantificación consiste en la reducción del número de bits que se utilizan para representar parámetros dentro del modelo. Por tanto, si en el modelo base se usasen 32 bits para esto, al modelo cuantificado le bastaría 8 bits para tener un rendimiento similar a cambio de mucha más eficiencia, reduciendo significativamente la cantidad de operaciones que necesita para cumplir con su objetivo.

Un hardware específico para IA pueden ser las CPUs de última generación de AMD e INTEL, con una gran cantidad de núcleos, aunque la parte más importante está en la GPU, donde los núcleos tensor de las GPUs de NVIDIA aumentan drásticamente el rendimiento.

3.2.- LA INTERFAZ TEXT GENERATION WEBUI

Text Generation WebUI es una aplicación Interfaz de tipo PLAYGROUND que brinda la posibilidad de ejecutar Modelos de Lenguaje Grande de forma sencilla en cualquier PC [51], sus principales características son:

- Cuenta con tres interfaces para interactuar con el modelo: Interfaz de dos columnas, blog de notas y chat.
- Permite cargar múltiples tipos de modelos, entre los que se encuentran los tipo Transformers, llama.cpp, ExLlamaV2, AutoGPTQ y AutoAWQ entre otros.

- Cuenta con una gran cantidad de extensiones, una de las cuales permite la conexión con el Modelo cargado a través de ApiREST, lo que es muy útil si se quiere realizar una conexión desde una aplicación externa.
- Entrenamiento por medio de la generación de LoRA, el cual se cargará como una extensión al modelo del cual se generó tras el entreno.
- Múltiples opciones a la hora de cargar un modelo de tipo transformers, adaptando los parámetros al hardware donde se desea ejecutar.

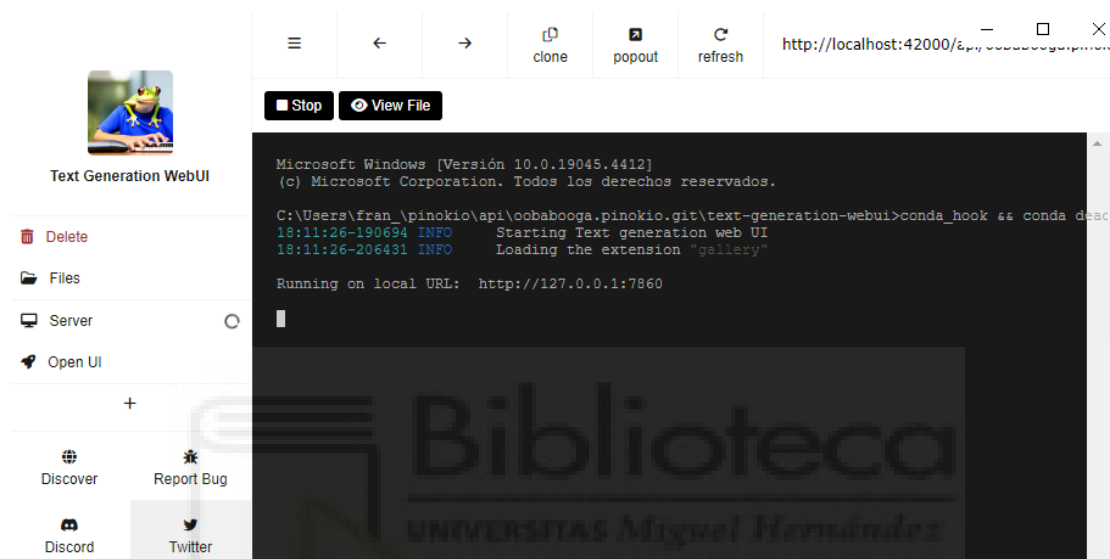


Figura 3.1.- Interfaz Text Generation WebUI

Text Generation WebUI permitirá ejecutar los modelos indicados en el punto 3.1 y poder interactuar con ellos desde una aplicación externa (ver **Anexo I**).

3.3.- DATOS DE ENTRENAMIENTO

En la mayoría de ocasiones, es probable que el modelo orientado a la programación que se esté usando para ofrecer un servicio no sea 100% eficaz, ya que las tecnologías van cambiando con el tiempo y es posible que el banco de datos de dicho modelo esté obsoleto o no abarque todas las características de programación necesarias.

Para suplir este problema, normalmente bastaría con utilizar un modelo IA actualizado al momento presente, sin embargo, existen casos donde una empresa exija seguir unas buenas prácticas de programación o unos determinados protocolos, que el modelo IA no ofrecerá a no ser que se especifique en la petición que se le solicita a dicha IA (prompt). Es aquí donde entran en juego nuevos datos de entrenamiento para el ajuste fino.

Los datos de entrenamiento[71] son cadenas de texto relacionadas que siguen una coherencia sobre un tema específico, como por ejemplo un párrafo donde se especifica cómo crear una interfaz para una clase “servicio” dentro de spring boot.

Una vez introducidas las nuevas cadenas de texto a modo de entrenamiento, el modelo cambiará sus parámetros, y por tanto, su forma de redactar una respuesta, tomando en cuenta dichas cadenas de texto nuevas. Por ejemplo, si a un modelo se le pregunta “*cómo sería la casa ideal*”, este puede responder algo como:

"La casa ideal tiene cocina, baño, salón, habitaciones y una terraza"

Es posible entrenar el modelo con cadenas de texto donde se indique “*la casa azul es ideal*”, de modo que al volver a realizar la misma pregunta, dado que el modelo ha detectado el sustantivo “*casa*”, el adjetivo “*ideal*” y el verbo “*es*”, hay muchas posibilidades de que añada a su respuesta el adjetivo “*azul*”. De este modo realizaríamos un ajuste fino por medio de datos de entrenamiento sin formato.

Otra opción consiste en introducir cadenas de texto a modo de petición y respuesta[72]. Esto se consigue escribiendo de forma literal un conjunto de preguntas acompañadas con sus respuestas, de modo que se le enseñe al modelo cuál sería la respuesta ideal a un posible conjunto de preguntas. Volviendo al ejemplo de la casa, indicando al modelo una cadena de texto con una petición “*describeme las partes de una casa ideal*”, junto con la respuesta “*la casa ideal tiene un baño con ducha, lavabo y bidé, una habitación con una cama de matrimonio, una cocina con una encimera amplia para cocinar, un salón con chimenea y un jardín donde los niños puedan jugar*”. De modo que, a la hora de preguntar por una casa ideal, la respuesta se asemejará un poco más a la introducida en los datos de entrenamiento. Estos datos se pueden introducir de forma libre o siguiendo algún tipo de formato, en el caso de codeLlama-Alpaca dicho formato consiste en un array de objetos compuestos por una o dos entradas (instruction, input) y una salida (output), como se muestra en la figura 3.2:

```
{
  "instruction,output": "Below is an instruction that describes a task. \n\n
    ### Instruction:\n%instruction%\n\n
    ### Response:\n%output%",
  "instruction,input,output": "Describes a task, paired with an input that
  provides further context. Write a response that appropriately completes the
  request.\n\n
    ### Instruction:\n%instruction%\n\n
    ### Input:\n%input%\n\n
    ### Response:\n%output%"
}
```

Figura 3.2.- Formato de entrada para datos de entrenamiento

Atendiendo a este formato se puede construir un objeto JSON que cumpla dicha especificación y que contenga un array con la información de entrenamiento deseada (ver figura 3.3).

```
[
  {
    "instruction": "¿Puedo programar código Java en Springboot?",
    "output": "Sí, puedes programar código Java en Springboot,
              además de Kotlin y Groovy."
  },
  {
    "instruction": "¿Cuál es la última versión de Springboot?",
    "output": "Hasta mi última actualización, la última versión cerrada de
              Springboot es 3.3.0"
  }
]
```

Figura 3.3.- Ejemplo de array JSON con datos de entrenamiento

Llegados a este punto, se debe usar un conjunto de datos lo suficientemente amplio para que se logre notar un cambio en la respuesta de las peticiones realizadas al modelo. Para ello, se usarán modelos ya entrenados como Mistral 8x7B, Codellama 70B o incluso ChatGPT. Tanto Mistral como Codellama son modelos muy lentos, los cuales tienen una tasa de respuesta de 2 y 5 min por pregunta respectivamente, debido a la gran cantidad de información que se debe procesar, por lo que no son viables para este proyecto pero sí para extraer datos de entreno.

3.4.- POSTGRESQL Y DBeaver

PostgreSQL es un sistema de base de datos relacional de código abierto muy potente, con el cual se pretende ampliar el lenguaje SQL combinando funciones de almacenamiento y escalando aquellas cargas de trabajo que son complicadas.

PostgreSQL cuenta con una sólida reputación debido a su confiabilidad e integridad de datos, ofreciendo flexibilidad añadiendo extensiones y soluciones innovadoras de alto rendimiento gracias a su comunidad de código abierto.[52]



Figura 3.4.- Logotipo PostgreSQL

PostgreSQL posee una gran cantidad de funciones para ayudar tanto a desarrolladores a crear sus aplicaciones como a administradores que desean proteger la integridad de sus

datos mediante la creación de entornos tolerantes a fallos. Todo esto es posible sin importar el tamaño de los datos que se manejen.

Además de todas las ventajas que ofrece, PostgreSQL cumple con el estándar SQL, de modo que posee todas las características tradicionales de una base de datos tradicional, lo que conduce a buenas decisiones arquitectónicas. La mayoría de características del estándar SQL Server son compatibles con PostgreSQL, aunque existen diferencias en funciones y sintaxis ligeramente distintas en algunos casos. Algunas diferencias reseñables entre SQL Server y PostgreSQL son[53]:

- Ambos son sistemas de administración de bases de datos relacionales, pero PostgreSQL relaciona objetos.
- PostgreSQL es de código abierto mientras que SQL Server pertenece a Microsoft.
- Ambos pueden ejecutarse en Microsoft o Linux, pero PostgreSQL puede ejecutarse en cualquier sistema operativo que se proponga.
- PostgreSQL no usa T-SQL, sin embargo, permite otras funcionalidades gracias a que es de código abierto.

Con respecto a DBeaver, es una herramienta de código abierto y totalmente gratuita orientada a administradores y desarrolladores de bases de datos. El objetivo principal de dicha herramienta es mejorar la usabilidad en la gestión de bases de datos [54].



Figura 3.5.- Logotipo DBeaver

La ventaja principal es que se trata de una herramienta de código abierto, multiplataforma y con multitud de extensiones y utilidades para la gestión de cualquier base de datos que disponga de un controlador de tipo JDBC, así como cualquier fuente externa tenga o no dicho controlador.

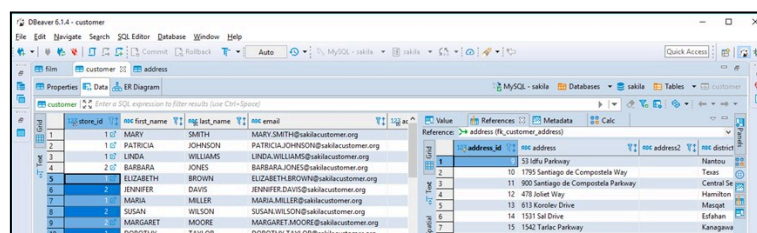


Figura 3.6.- Interfaz DBeaver

3.5.- JAVA, SPRINGBOOT Y MAVEN

Java es una plataforma conocida por el lenguaje de programación con el mismo nombre, el cual ha evolucionado hasta estar presente en la actualidad en una gran mayoría de desarrollos digitales, incluyendo todos los desarrollos previstos a futuro [55].



Figura 3.7.- Logotipo Java

Actualmente, todo desarrollador que desee programar en Java deberá incorporar a su máquina un paquete llamado JDK (Java Development Kit), el cual indica la versión de Java que se estará utilizando en el desarrollo. JDK es un software para desarrolladores que incluye herramientas Java como el compilador, depurador, entre otros, todo para desarrolladores Java. [56]

JDK permite desarrollar aplicaciones que puedan ejecutarse en cualquier máquina virtual Java o usarse en cualquier sistema sin tener que volver a compilar el código.

Sobre SpringBoot, se trata de un maquetador que permite crear aplicaciones basadas en Java sin esfuerzo, gracias a la plataforma spring y la biblioteca de terceros.[57]



Figura 3.8.- Logotipo SpringBoot

Entre las características más importantes de SpringBoot están:

- La posibilidad de incrustar Tomcat, Jetty o Undertow sin tener que recurrir a implementaciones con archivos WAR.
- Realizar una compilación e iniciar la aplicación de forma simple y sencilla gracias a dependencias preconfiguradas.
- Realizar configuraciones de todas las bibliotecas Spring e incluso bibliotecas de terceros que lo permitan.
- Proporcionar métricas, configuraciones externas y otras funciones listas para producción.

Dentro del framework de SpringBoot se encuentra la herramienta de gestión de proyectos Java llamada Maven, la cual facilitará la creación y el desarrollo del proyecto que se desea construir. Otra herramienta para gestión de proyectos es Gradle, sin embargo, se decide usar Maven por su fácil uso además de configurarse más intuitivamente.



Figura 3.9.- Logotipo Maven

Maven permite crear y gestionar proyectos desarrollados en Java facilitando todo el proceso y proporcionando un sistema de construcción uniforme y de calidad que fomenta las buenas prácticas de desarrollo. Gracias a Maven, la información de todas las dependencias utilizadas en el proyecto se incluyen dentro del objeto pom.xml (POM, Project Object Model) de SpringBoot, donde también se refleja la versión utilizada. Cada dependencia proporciona código ya definido que facilita la implementación de aquello para lo que ha sido creado, como por ejemplo, toda la configuración para poder conectar con bases de datos de tipo JDBC o la configuración en torno al desarrollo de APIs para la interacción con la aplicación por medio de llamadas de tipo REST.

En el presente proyecto se utilizan las siguientes dependencias gestionadas por Maven:

- SpringBoot Starter Web.- Se usa para aplicaciones web utilizando un modelo vista controlador (MVC) y usando Tomcat como contenedor integrado.[58]
- Jackson Databind.- Utilizado para el mapeo de datos, permitiendo además transformar objetos JSON en clases y viceversa.[59]
- SpringBoot Starter Webflux.- Es un framework introducido en Spring 5.0, que se utiliza en conjunto junto a SpringBoot Starter Web para la creación del Api controlador.[60]
- SpringFox Swagger2.- Herramienta para generar la documentación API en aplicaciones Spring. Esto facilita la accesibilidad tanto a desarrolladores como a usuarios finales.[61]
- Lombok.- Biblioteca de Java para simplificar todo tipo de anotaciones dentro del desarrollo, automatizando la generación de código repetitivo. Se pueden generar métodos tipo get, set, builder, toString, entre otros, de forma automática. con ello se reduce la posibilidad de errores y la simplificación del código.[62]

- SpringBoot Starter AMQP.- Permite desarrollar soluciones de mensajería basadas en amqp, de modo que se puedan enviar y recibir mensajes por medio de plantillas. Además, cuenta con un gestor de recursos proporcionando soporte POJO, útil para implementar un contenedor de escucha de mensajes.[63]
- Spring Data JPA.- Librería para implementar repositorios basados en JPA (Java Persistence API), facilitando la creación de aplicaciones que requieren acceso a base de datos. JPA trae implementadas todas las capas de acceso a datos reduciendo significativamente el esfuerzo que se hubiese requerido en otro caso.[64]
- H2 database.- Se trata de una base de datos integrada muy útil para desarrollos de aplicaciones con acceso a base de datos. Es de tipo cliente/servidor y toda la información se almacena directamente en memoria (datos no persistentes). Esto la hace perfecta para pruebas sin alterar la base de datos principal.[65]

3.6.- OTRAS HERRAMIENTAS

3.6.1.- El asistente Pinokio

Pinokio es un navegador que permite realizar la instalación, ejecución y automatización de todo tipo de inteligencia artificial en una computadora personal. Esto es posible gracias a la automatización de cualquier línea de comando dentro del script Pinokio por medio de una interfaz fácil de usar para cualquier usuario.[66][67]



Figura 3.10.- Logotipo Pinokio

Pinokio puede utilizarse para automatizar y administrar aplicaciones y modelos de IA, crear flujos de trabajo con el fin de organizar toda aplicación de IA que haya instaladas y ejecutar cualquier comando en ellas (entre otras posibilidades). Algunas características de Pinokio son:

- Ejecución local: Todas las aplicaciones se instalan y ejecutan localmente evitando servidores ajenos.

- Privacidad: Ninguna de las aplicaciones IA ejecutadas por Pinokio envía información a Internet. Todo se ejecuta en la computadora local ajena a Internet garantizando la privacidad de las peticiones.
- Uso de interfaz: La interfaz que ofrece Pinokio es muy intuitiva, de modo que no se requiere uso de terminal en ningún momento.
- Autonomía: Con Pinokio no se necesita recurrir a otros programas para su instalación (lo único que necesita Pinokio es Pinokio).
- Multiplataforma: Pinokio está disponible para Windows, Linux y Mac.
- Optimización: Debido a la naturaleza de Pinokio y la posibilidad de brindar la instalación cualquier aplicación de IA en una computadora personal, Pinokio cuenta con muchas funciones de optimización para consumir el mínimo número de recursos posible ya sea a nivel de almacenamiento o memoria.
- Portátil: Una vez instalado, todo se almacena en una única ruta del sistema que es totalmente portable. Únicamente se debe copiar dicha carpeta a otra computadora y ya puede ser ejecutada.

3.6.2.- Postman

Postman es una herramienta de desarrollo de interfaces de programación de aplicaciones (APIs) para realizar pruebas sobre aquellas que se estén creando. Es capaz de realizar solicitudes HTTP de tipo GET, POST, PUT, PATCH entre otras, introducirlas dentro de entornos de proyecto y que se auto configuren adaptándose a las necesidades del mismo. Además, podemos convertir dichas APIs en código para lenguajes de programación como JavaScript o Python.



Figura 3.11.- Logotipo Postman

Hoy en día es difícil desarrollar código para un sistema backend con controladores APIs sin usar este tipo de herramientas para comprobar su correcto funcionamiento, y Postman es de las más completas que existen actualmente.

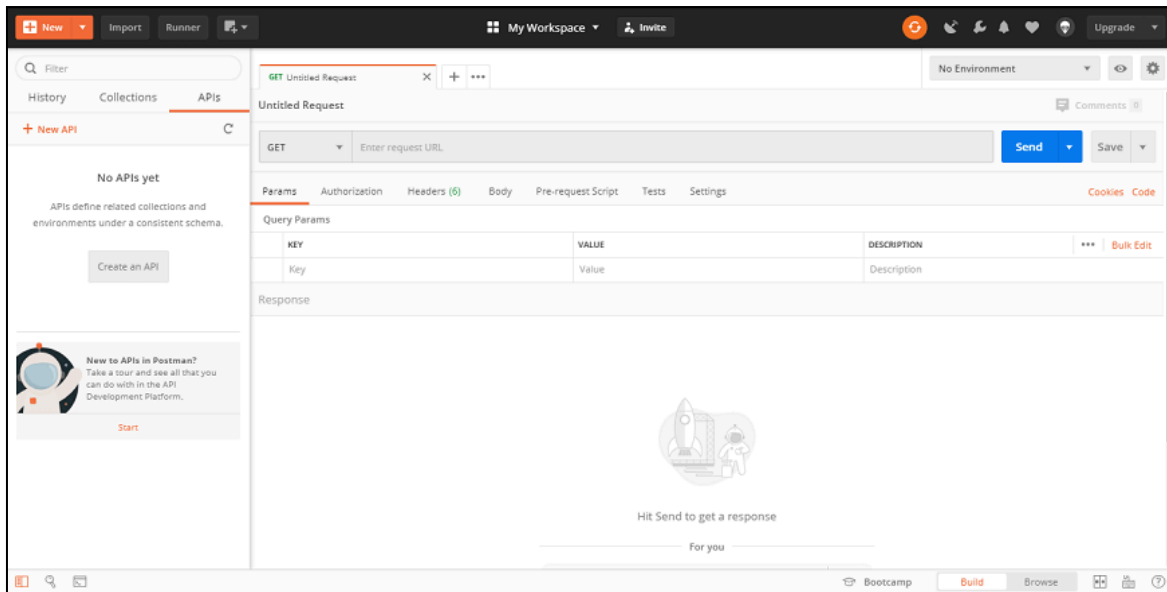


Figura 3.12.- Interfaz Postman

Algunas características destacables de Postman son:

- Herramienta versátil: Realiza solicitudes HTTP de todo tipo, permite configurarlas a voluntad.
- Cuerpo de mensaje flexible: Normalmente en las aplicaciones backend se usa el formato JSON, sin embargo, Postman permite montar un cuerpo de mensaje en un gran número de formatos, incluidos datos sin procesar (raw) o código binario.
- Sistema de autenticación simple: Donde Postman ofrece varios métodos de autenticación por claves, tokens, Oauth, etc. Importante cuando se desarrollan entornos de un alto nivel de seguridad.
- Organización: Todas las APIs de un proyecto se pueden organizar por entornos, cada uno con su propia autenticación y raíz. Se podría usar la misma API para un entorno de desarrollo, preproducción y producción, con distintas URLs, distintas autenticaciones y distintos parámetros con tan solo unos clics.
- Documentación: Postman cuenta con una amplia documentación, así como constantes actualizaciones para adaptarse a las últimas tecnologías del mercado.

3.6.3.- IntelliJ IDEA - Community Edition

IntelliJ IDEA es un entorno de desarrollo integrado (IDE) para el desarrollo de código tanto en Java como en Kotlin. Su interfaz está diseñada de modo que se pueda maximizar

la productividad en los desarrollos de proyectos complejos. Pese a que tiene muchos puntos fuertes, destaca la posibilidad de proporcionar finalización inteligente de código en tareas que pueden ser repetitivas, análisis activo de código estático y refactorizaciones, de modo que el desarrollador pueda enfocarse en la creación del software.[69]



Figura 3.13.- Logotipo IntelliJ IDEA

IntelliJ IDEA es un entorno multiplataforma, lo que permite la instalación tanto en Windows y Mac como en Linux. Además de Java y Kotlin, también es posible desarrollar aplicaciones de forma nativa en lenguajes como Scala y Groovy. Adicionalmente, es posible usar otros lenguajes de programación por medio de plugins, algunos lenguajes soportados son: Python, Ruby, PHP, SQL, Go, JavaScript, TypeScript, Thymeleaf, JSON, Markdown, HTML, XML, XPatch, Velocity y hojas de estilo como CSS, Less y Sass.

La interfaz proporcionada por IntelliJ IDEA se centra en ofrecer al desarrollador un entorno centrado en la edición. En ella se busca ofrecer de forma automática todas las herramientas necesarias durante la programación, de modo que el editor no se salga en ningún momento del contexto sin ser interrumpido.

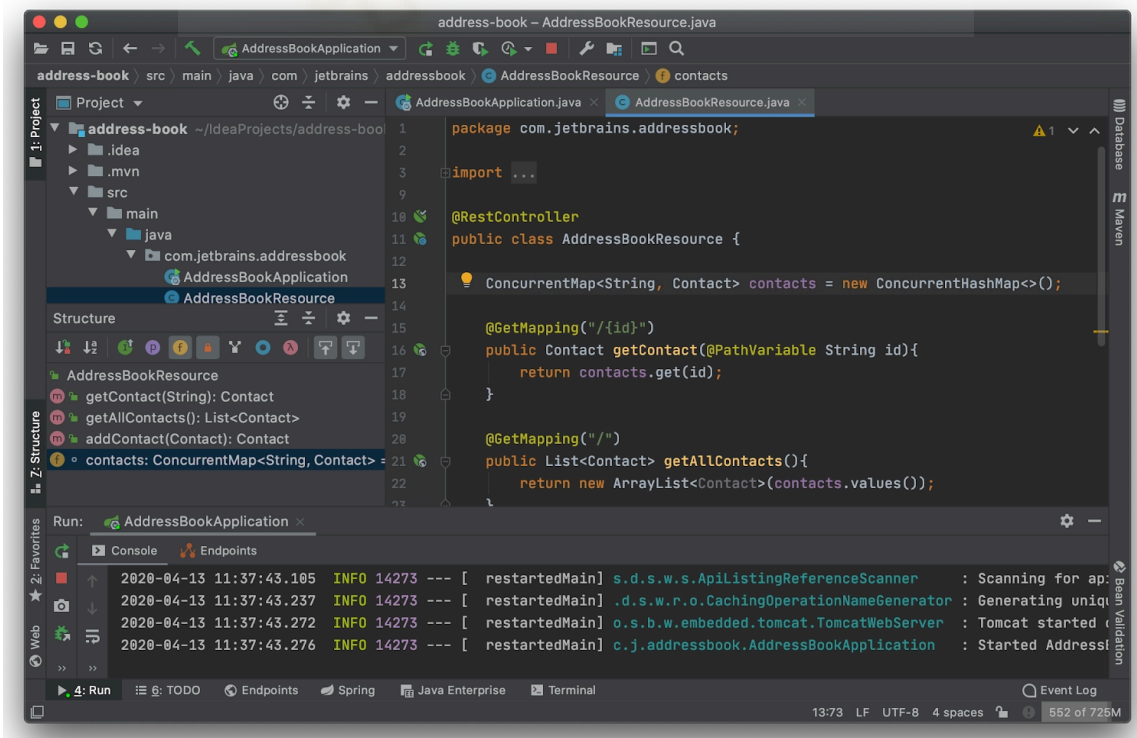


Figura 3.14.- Interfaz IntelliJ IDEA

Igualmente, la plataforma permite configurar toda la apariencia para adaptarla al desarrollador, ya se trate de colores del código o el IDE en sí o incluso de la estructura misma de la interfaz, además de ajustar el comportamiento de cualquier aviso o ventanas de mensajes que pueda distraer.

3.6.4.-RabbitMQ

RabbitMQ es un software de colas y envío de mensajes. Hace de intermediario de mensajes entre aplicaciones, administrando las colas por las que viajarán los mensajes. Gracias a este software, se pueden conectar aplicaciones definiendo colas en éstas y, por medio de RabbitMQ, transferir mensajes.[70]



Figura 3.15.- Logotipo RabbitMQ

Las colas están preparadas para poder transmitir cualquier tipo de información, desde mensajes de texto hasta instrucciones dadas a otro servicio alojado en un servidor distante. El software proporcionado por RabbitMQ se encarga de almacenar el mensaje de la cola hasta que un receptor se conecte para consumir y retirar de la cola dicha información. Hecho esto, la aplicación receptora procesa el mensaje recibido.



Figura 3.16.- Proceso Producer/Consumer RabbitMQ

Como intermediario de mensajes, RabbitMQ cumple con la función de reducir cargas y tiempos de entrega entre servicios gracias a que esa función se delega al propio Rabbit, preparado para este cometido y evitando efectuar este tipo de tarea en los servicios.

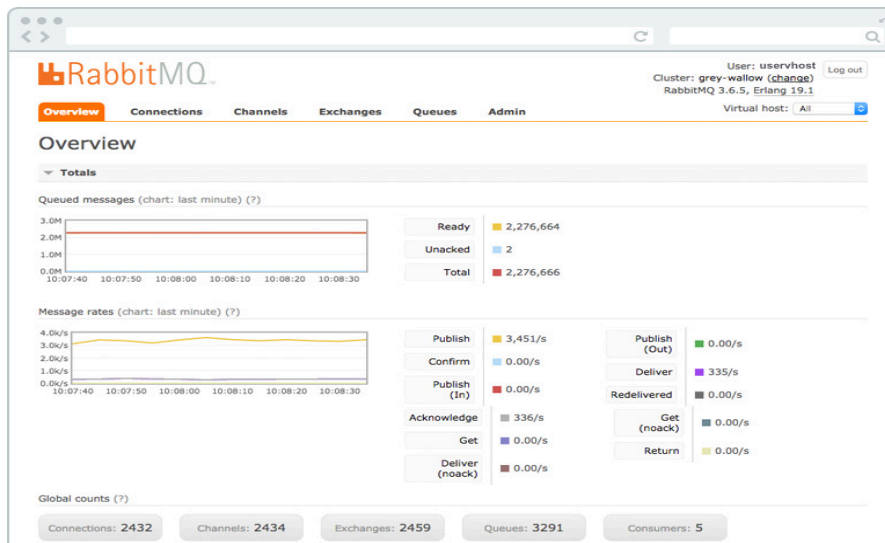


Figura 3.17.- Interfaz RabbitMQ

Gracias a RabbitMQ, se pueden usar colas para permitir a los servidores responder a solicitudes rápidamente, consumiendo muy pocos recursos, evitando retrasos en la respuesta. Ya que los mensajes de colas están destinados a todos aquellos servidores que estén consumiendo de dicha cola, es una buena práctica para distribuir la carga de trabajo entre servicios o incluso de un mismo servicio donde se ha habilitado más de un Pod para cumplir con la demanda de uso.

Para conocer más a fondo los fundamentos de RabbitMQ se deben tomar en cuenta los conceptos que se listan a continuación:

- **Productor:** Se trata de la aplicación que envía el mensaje a la cola.
- **Consumidor:** Se encargará de estar a la escucha de la cola que se indica en su configuración para recibir y consumir el mensaje.



Figura 3.18.- Comunicación productor/consumidor con RabbitMQ

- **Cola:** Es un buffer de almacenamiento de mensajes. Si se publica un mensaje en dicha cola quedará almacenado hasta que alguna aplicación a la escucha lo consuma.
- **Mensaje:** Se trata de la información que se envía desde el productor (aplicación que produce el mensaje) hasta el consumidor (aplicación que recibe el mensaje y lo procesa).

- Canal: Cuando se realiza una publicación o se consume, se genera una conexión virtual la cual hace de canal para permitir el paso del mensaje enviado.

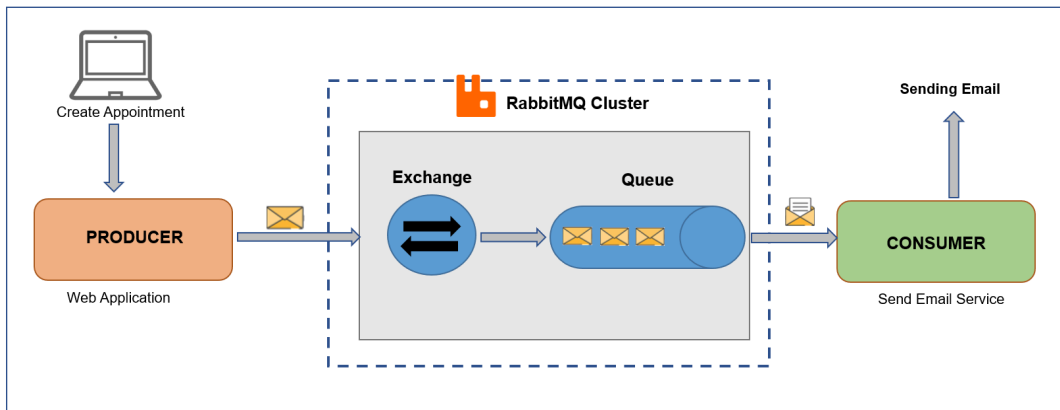


Figura 3.19.- Publicación de mensaje en canal y cola de mensajes



Capítulo 4

Metodología y

resultados

4.1.- Planificación del proyecto

Para la realización de este proyecto, se ha llevado a cabo un estudio utilizando un ciclo de vida en espiral, donde se han implementado funcionalidades, se han realizado sus pruebas y, en las siguientes iteraciones, se han ido añadiendo nuevas funcionalidades hasta llegar al producto final.

Tanto el ciclo de vida como la planificación temporal del proyecto, con el correspondiente diagrama de Gantt, se muestran en los siguientes subapartados.

4.1.1.- Ciclo de vida - Espiral

Para el ciclo de vida en espiral que se ha utilizado en este proyecto, se ha realizado un total de 3 iteraciones bien definidas, en las que se ha realizado un planteamiento del problema, se ha realizado un análisis de los posibles riesgos a tener en cuenta y, tras una etapa de desarrollo y pruebas, se ha evaluado el producto y discutido las siguientes mejoras.

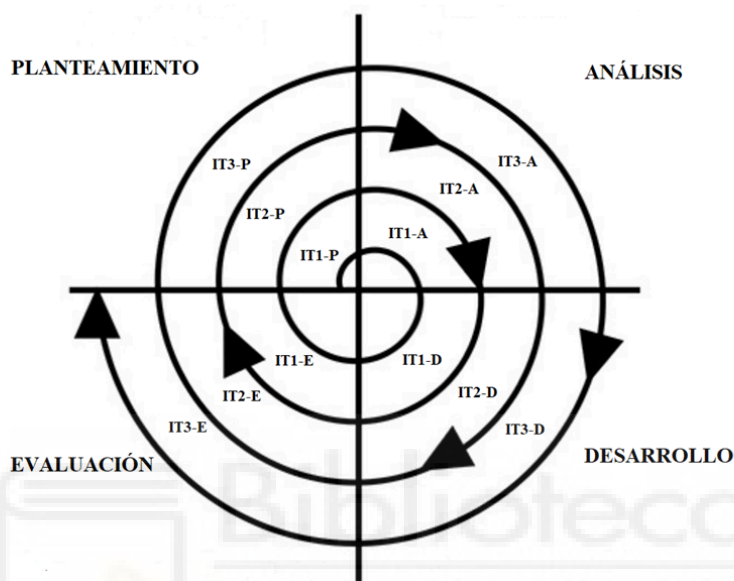


Figura 4.1.- Ciclo de vida Espiral

Primera iteración

- **Planteamiento:** Se define el objetivo principal, implementar una aplicación backend con la cual el desarrollador puede realizar consultas a través del cliente Postman a un GPT, de modo que se desarrolle la estructura principal de la aplicación. La conexión con un GPT será, temporalmente, un simulador estático (o Mock) que recibirá respuestas aleatorias simuladas para poder avanzar con el desarrollo.
- **Análisis de riesgos:** Para incrementar la seguridad, se divide la aplicación en 3 microservicios, de modo que únicamente uno de ellos esté expuesto a desarrolladores comunes y, en el futuro, a un portal web. Un segundo microservicio será el encargado de la conexión con el GPT, este se podrá duplicar en PODs (contenedores) dependiendo de la demanda, y un tercer microservicio que se encarga exclusivamente de la conexión con la Base de Datos.
- **Desarrollo y prueba:** Se desarrolla la aplicación utilizando el maquetador de SpringBoot, acorde a la mayoría de los desarrollos de Santander. Esto permitirá reutilizar configuraciones en servidores de Red Hat Openshift o AWS, dependiendo

del entorno donde se requiera el despliegue. Las pruebas se realizan íntegramente en local, de modo que se busca la mayor eficiencia posible.

- Evaluación: En dicha evaluación, se tiene en cuenta el flujo del código dependiendo del rol dentro de la aplicación.

Segunda Iteración

- Planteamiento: Se reemplaza el Mock por un servicio local con un Modelo IA orientado a desarrollo de código en ejecución al cual se le podrá realizar peticiones del mismo modo que se realizaba al Mock. Se añade una base de datos que contiene la información de los proyectos que se encuentran actualmente en desarrollo, de modo que el GPT pueda consultar y adaptarse a ellos.
- Análisis de riesgos: Se añade la auditoría de posibles errores en el flujo para registrar en la Base de Datos cualquier posible error que se produzca en el sistema. Se revisa el tiempo de respuesta por parte del GPT, ya que se busca un equilibrio entre su eficiencia y la calidad de sus respuestas.
- Desarrollo y pruebas: Se habilita el servicio de IA en local cargando varios modelos hasta dar con un equilibrio entre rendimiento y calidad. También se añade una auditoría de errores.
- Evaluación: Se comprueba el rendimiento del GPT, así como el flujo y el proceso de auditoría.

Tercera Iteración

- Planteamiento: Se requiere un mejor rendimiento en el GPT, así como en su ajuste fino. También se requiere añadir auditoría para todo el proceso, no sólo para errores de la aplicación. Se solicita una interfaz web para interactuar con la aplicación (en lugar del usar Postman).
- Análisis de riesgos: Se realizan pruebas con varios modelos IA, se decide por la versión 0.2 de Mistral, la cual se asemeja a las respuestas de ChatGPT con un rendimiento y calidad aceptable. Se estudia la posibilidad de publicar toda la auditoría detallada en Base de Datos por medio de colas RabbitMQ. Se plantea también una mejora en las excepciones para dar información detallada sobre cada error.
- Desarrollo y pruebas: Se implementan las nuevas colas para la auditoría. Se modifica la información de cada excepción para los errores. Cambios del modelo

IA y adaptación de la respuesta para diferenciar entre código y chat en la interfaz web. Se implementa una pequeña interfaz utilizando PHP, HTML, CSS y JavaScript.

- **Evaluación:** Se realizan pruebas de rendimiento directamente desde la interfaz web. Se comprueba que el código de respuesta se adapta al código del proyecto seleccionado.

4.1.2.- Diagrama de Gantt

A continuación, se muestra la información correspondiente a los tiempos requeridos para completar cada tarea especificada en el ciclo de vida (cada día de trabajo equivale a unas 2 horas de trabajo aproximadamente):

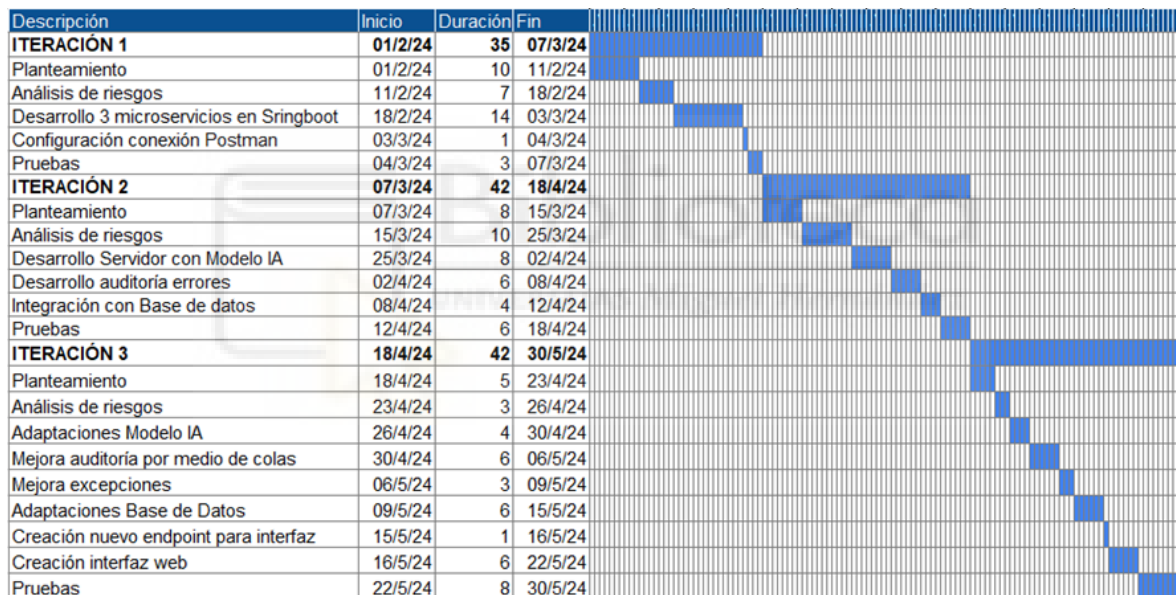


Figura 4.2.- Diagrama de Gantt

4.2.- Captura de requisitos

En este proyecto, la funcionalidad que se pretende obtener para los dos roles existentes es bastante limitada y lineal, ya que no se busca que un desarrollador pueda realizar más actividades que el mero hecho de realizar consultas a la IA. Por tanto, se definen dos roles:

- **Developer:** La única actividad que se desea que realice un desarrollador es la de realizar consultas a la IA, de modo que su actividad estará limitada a la selección de uno de los proyectos que existen actualmente en el sistema y lanzar una pregunta a

la IA para recibir la respuesta. Dicha respuesta debe ser más que suficiente para que el desarrollador pueda continuar con su labor en la empresa.

- Technical Lead o Admin: Este usuario es un “Developer” que además, contará con la información necesaria para poder registrar nuevos proyectos en el sistema a través de un API Rest. Dicha información se usará a través de la página web para la realización de consultas.

La figura 4.3 muestra el diagrama de casos de uso de ambos roles.

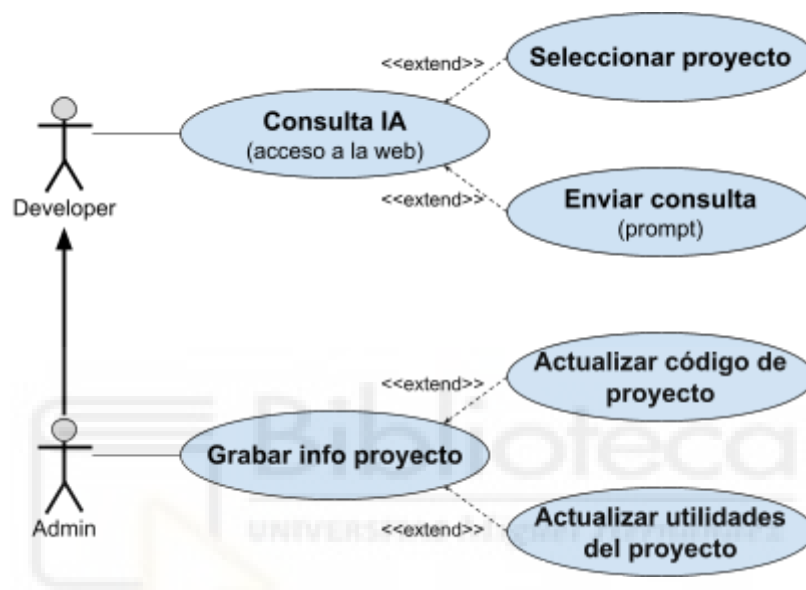


Figura 4.3.- Diagrama de casos de uso

4.3.- Diseño

La aplicación estará distribuida en dos VPNs de la empresa:

- VPN Externa.- Está orientada a desarrolladores comunes, en ella se alojan los repositorios de código en desarrollo y da acceso a herramientas útiles para los desarrolladores.
- VPN Interna.- Está orientada a desarrolladores de más alto rango, líderes de proyecto y arquitectos de software (administradores).

Para el proyecto presente, no será necesario realizar ningún tipo de login, ya que es una aplicación exclusivamente para desarrolladores de Santander y, por tanto, sólo debe ser accesible a través de la VPN externa de la entidad.

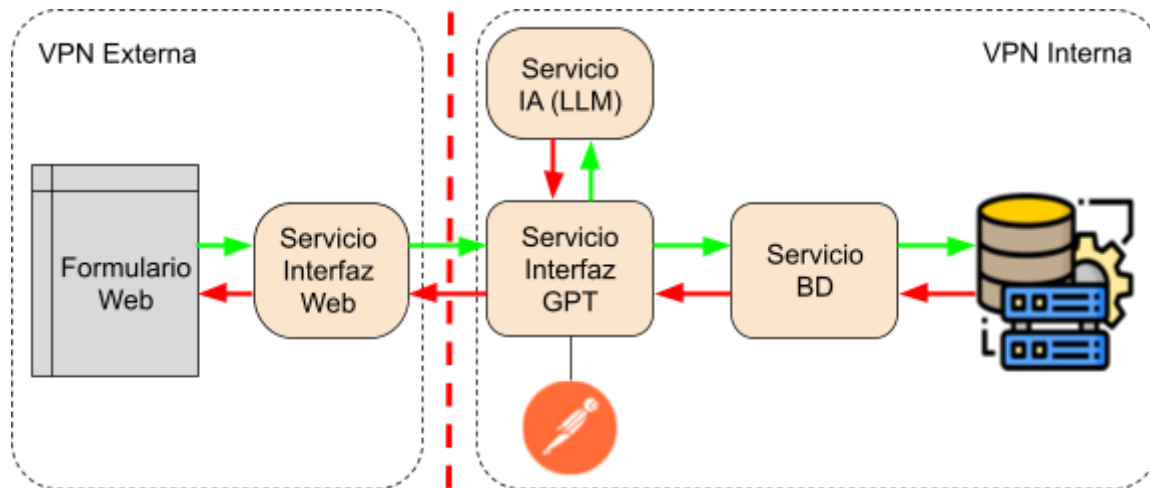


Figura 4.4.- Módulos de la aplicación

La VPN externa contendrá una interfaz web (formulario) para que los desarrolladores puedan lanzar peticiones a la IA. Por seguridad se incluye un servicio de interfaz web que oculta detalles de conexión con la VPN interna.

Figura 4.5.- Diseño pantalla de usuario (interfaz web)

La web será sencilla (ver figura 4.5), habrá un selector de proyecto, en el cual el desarrollador elegirá el proyecto en el que esté trabajando, una caja de texto para la pregunta del usuario (prompt), otra para la respuesta del sistema, y un selector para la precisión del modelo. Cuando se envía el formulario, toda la información la recibe el servicio de interfaz web que almacena un registro de logs para auditorías posteriores, lanza la petición al servicio de interfaz con la IA GPT, y muestra el resultado en la web al recibir la respuesta.

Cuando el servicio de interfaz GPT, ya en la VPN interna, recibe la petición, consulta en la base de datos toda la información relativa al proyecto seleccionado por el usuario, prepara la pregunta para el servidor de IA (donde se aloja el LLM) y se la lanza. Por razones de

seguridad, todas las preguntas construidas por este servicio, así como las respuestas que recibe desde el módulo de IA, se almacenan también en un registro de logs para posibles auditorías futuras.

Las conexiones con la BD se realizan a través de un tercer microservicio que se encarga de recibir las peticiones de consulta de los otros microservicios, así como recoger todos los registros de logs para auditorías a través de colas RabbitMQ (ver en la figura 4.6 el diagrama de secuencia de este proceso).

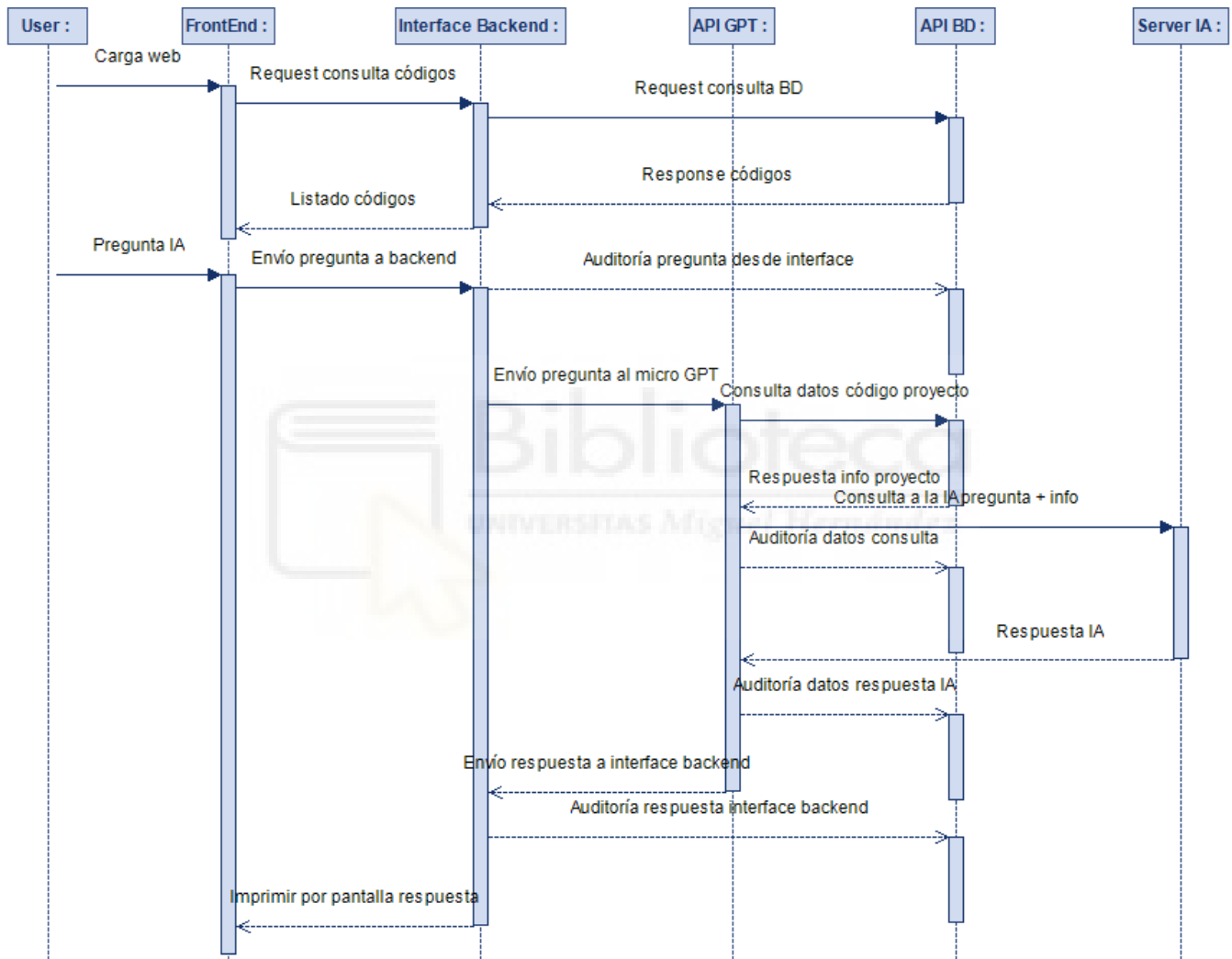


Figura 4.6.- Diagrama de secuencia Desarrollador

Los administradores podrán acceder directamente al microservicio interfaz GPT conectándose por la VPN interna, realizando peticiones HTTP con el cliente Postman, se encargan de almacenar en la BD la información de los proyectos, y reciben una respuesta favorable en el caso de que dicha información se haya publicado correctamente en la cola. El motivo de usar colas es para evitar la congestión del microservicio de base de datos, pudiendo grabarse dicha información en el momento que esté disponible (figura 4.7).

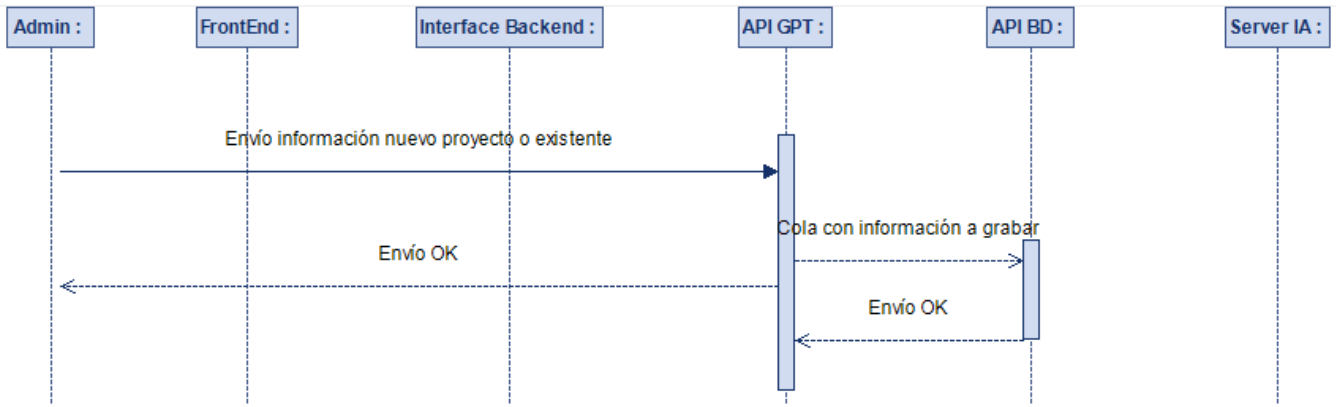


Figura 4.7.- Diagrama de secuencia Technical Lead

Los estados por los que pasa el sistema cuando se utiliza son pocos, ya que, al no existir grandes interacciones por parte del desarrollador, el sistema se limita a los estados que se muestran en la figura 4.8:

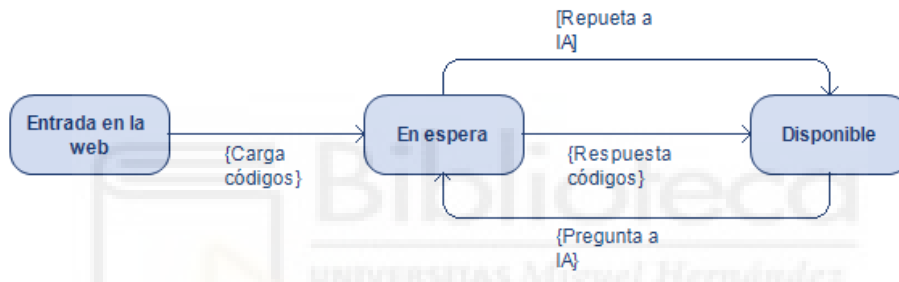


Figura 4.8.- Diagrama de estados

A continuación se muestra el diagrama de actividad tanto para desarrolladores como para el administrador. En él se muestra el recorrido que realiza el sistema cuando al hacer una consulta a la IA, mostrando además en qué momento pueden producirse fallos.

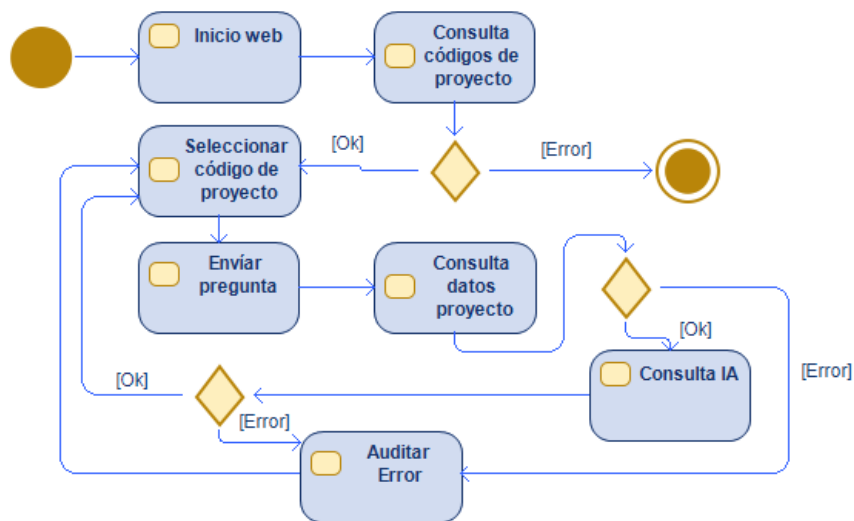


Figura 4.9.- Diagrama de actividad Desarrollador

En el caso de que se produjese un error en el inicio de la web, no se podrá seguir usando ya que es imprescindible el identificador del proyecto para poder realizar algún tipo de petición. Para el resto de situaciones siempre se alcanzará el estado donde se selecciona el código de proyecto.

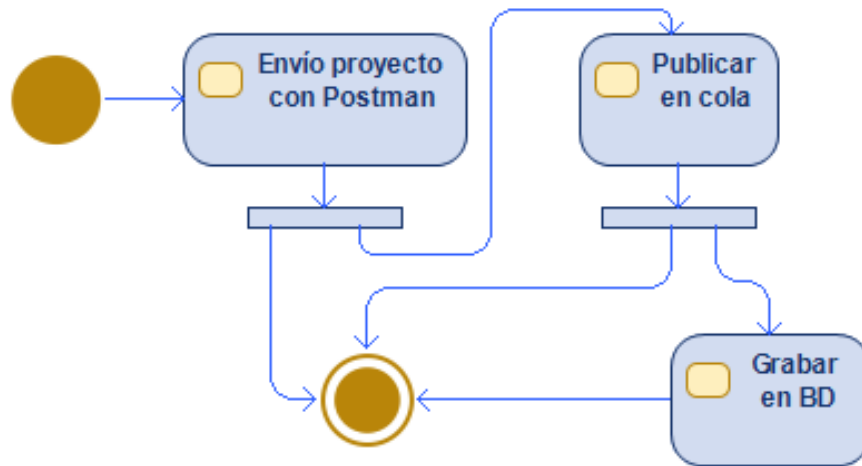


Figura 4.10.- Diagrama de actividad administrador

Para el proceso de la grabación en base de datos simplemente se realizará una petición con Postman, respondiéndolo un “ok” o un “error” en caso de que no se haya podido publicar la información en la cola. De un modo u otro la actividad terminará pese a continuar la publicación y la grabación en cola y base de datos respectivamente.

Descripción gráfica del flujo

Para conocer mejor el flujo, se aporta un esquema gráfico con las direcciones que toma la información durante todo el proceso (figura 4.11).

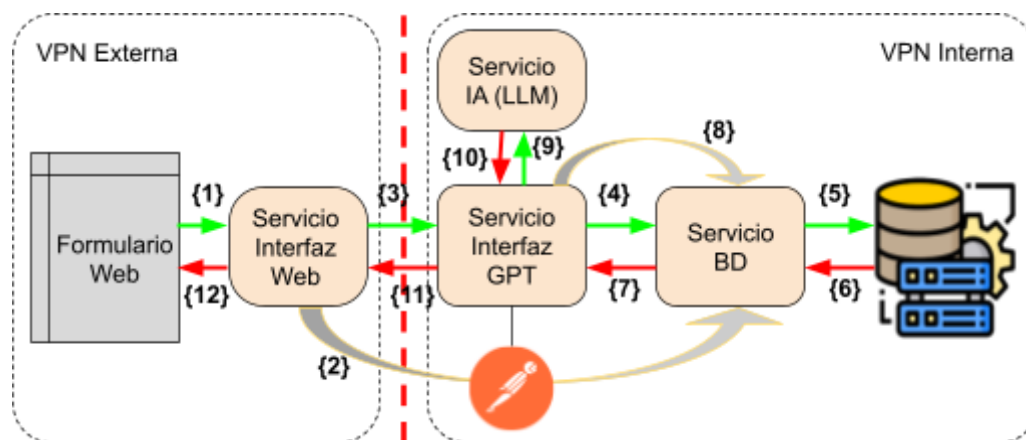


Figura 4.11.- Gráfico de flujo completo

Para el flujo de una consulta por parte de un desarrollador, se realiza una petición desde la web y se envía la información al primer microservicio (interfaz web) como indica la flecha {1} de envío. En el microservicio interfaz web registra el log correspondiente {2} y solicita a través de otra petición la pregunta al microservicio Interfaz GPT {3}. Éste se encuentra en una VPN interna, sólo accesible para administradores.

Llegados a este punto, el microservicio interfaz GPT realiza una petición al microservicio de base de datos para recuperar la información que haya guardada del proyecto que viene ligada a la pregunta {4} y el micro BD hace una consulta JPA a la base de datos {5}, esperando la respuesta de ésta {6}. Una vez tenga esa información en su poder, se envía los datos a la interfaz GPT {7}, y este registra toda la información recopilada junto a la pregunta {8}.

Es en este momento en el que se realiza la petición al servicio de IA (LLM) {9}, manteniéndose a la espera de que este responda {10}. Recogida la respuesta se vuelve a registrar el log correspondiente {8} y se responde al micro Interfaz web {11}. El microservicio interfaz web recoge la información, la procesa para que pueda ser legible para la web y se la envía a esta {12}, no sin antes registrar por última vez dicha información procesada {2}.

El flujo del administrador comienza directamente conectado a la VPN interna usando Postman, de modo que podrá realizar una petición REST directamente al microservicio interfaz GPT. Sólo es necesario recibir una respuesta afirmativa por parte del microservicio, ya que este se limitará a publicar en local la información recogida, de modo que el microservicio BD lo recogerá en el momento que esté disponible y lo guardará con una inserción JPA.

4.4.- Implementación

Para que un desarrollador pueda interactuar con la aplicación se ha implementado un formulario web básico con lo necesario para poder realizar consultas a la IA que se ha entrenado. Dicha página web ofrece 3 parámetros de entrada, un botón de enviar y un panel de respuesta.

- Para el primer parámetro de entrada se selecciona el código de proyecto, el cual se habrá cargado junto a la web a través de una petición REST a nuestra aplicación.
- El segundo parámetro es un panel por el cual se recibirá la pregunta que se desea realizar a la IA.

- El tercer parámetro es el ajuste de precisión, donde se habilitan 3 selectores para decidir si se desea tener una respuesta más precisa, más creativa o equilibrada. Esto se consigue modificando ligeramente los ajustes de temperatura, top_p y semillas explicado anteriormente.
- El hecho de pulsar enter dentro de la casilla de entrada principal no hace que el formulario se envíe, ya que se permite la posibilidad de realizar retorno de carro en la pregunta, en cambio, se implementa un botón para enviar toda la información empaquetada a la aplicación y que ésta actúe en consecuencia antes de lanzar la pregunta a la IA.
- Para la casilla de respuesta, se habilita un amplio espacio donde se formatea el código de respuesta. En él se diferencia el chat de respuesta del código de ejemplo usando un fondo grisáceo y un estilo de letra formateado para mejor comprensión del código mostrado.

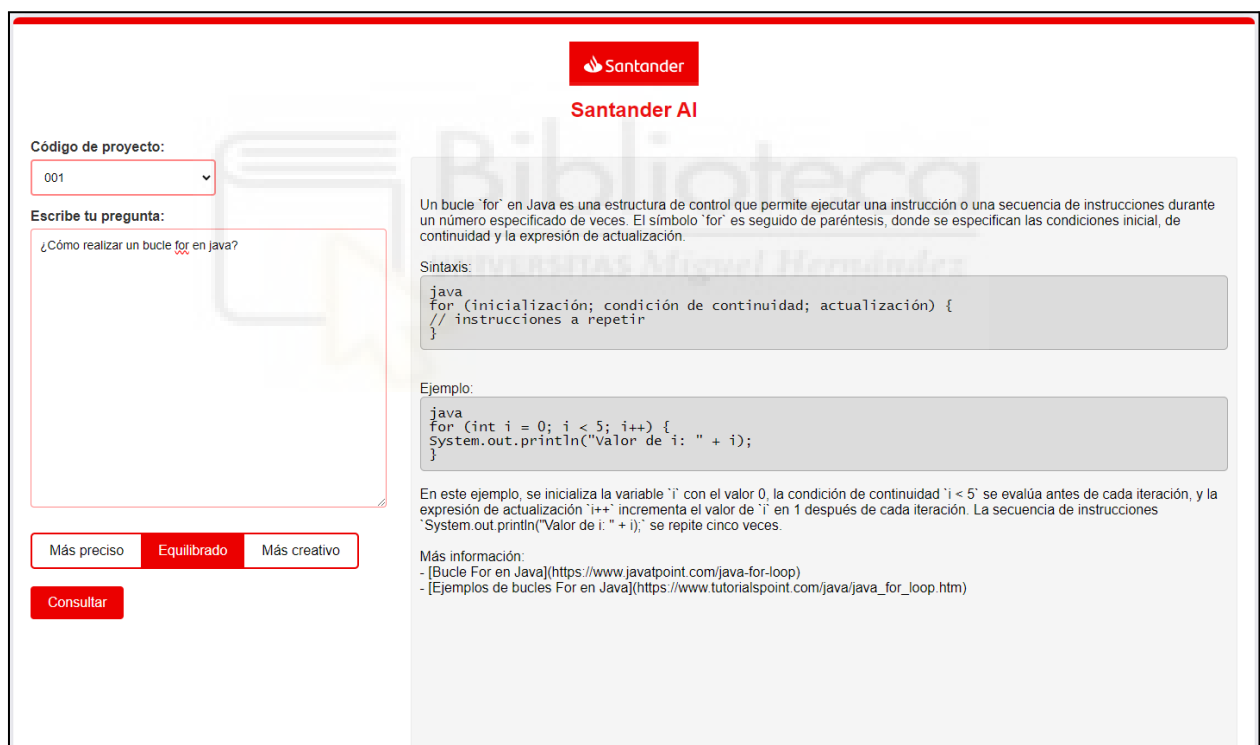


Figura 4.12.- Página web Santander IA

El desplegable “Código de proyecto” se rellena automáticamente al cargar la página, para recolección de códigos se realiza una llamada por medio de JavaScript al microservicio de interfaz web (figura 4.13).

```

fetch('http://localhost:8080/api/v2/codes', {
  method: 'GET',
  headers: {
    'Content-Type': 'application/json'
  }
})
.then(response => response.json())
.then(data => {
  const options = data.map(item => ({ id: item.id}));
  populateSelectOptions(options);
})
.catch(error => {
  console.error('Error fetching data:', error);
  projectMessageSelect.innerHTML = '<option value="" disabled selected>Error cargando opciones</option>';
});

```

Figura 4.13.- JavaScript consulta códigos de proyecto

Desde el mismo microservicio de interfaz web, se realiza una petición directamente al microservicio de base de datos ubicado en otra localización para poder obtener los códigos (figura 4.14).

```

@Override 1 usage
public String getCodes() throws JsonProcessingException {
  HttpHeaders headers = new HttpHeaders();
  headers.setContentType(MediaType.APPLICATION_JSON);
  HttpEntity<String> requestEntity = new HttpEntity<>(body: "", headers);
  return restTemplate.exchange(uri: "http://localhost:8082/api/v1/getCodes", HttpMethod.GET
    , requestEntity, String.class).getBody();
}

```

Figura 4.14.- Configuración API códigos

La recolección de códigos se realiza en el lado del microservicio de base de datos por medio de una petición JPA donde se busca toda la información guardada en tablas para posteriormente eliminar duplicados (figura 4.15).

```

@Override 1 usage
public String getCodes() throws JsonProcessingException {
  List<ProjectData> projectData = projectDataRepository.findAll();
  Set<String> seenCodes = new HashSet<>();
  List<Code> codeList = projectData.stream() Stream<ProjectData>
    .map(fineT -> new Code(fineT.getCode())) Stream<Code>
    .filter(code -> seenCodes.add(code.getId()))
    .collect(Collectors.toList());
  return objectMapper.writeValueAsString(codeList);
}

```

Figura 4.15.- Consulta códigos base de datos

Llegados a este punto, se selecciona el proyecto deseado y se escribe la pregunta. Al pulsar sobre el botón de enviar se lanzará la petición con el cuerpo de mensaje correspondiente (figura 4.16).

```
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $input = json_decode(file_get_contents('php://input'), true);
    $userMessage = $input['message'];
    $userProjectCode = $input['projectCode'];
    $radioMessage = $input['radioMes'];
    $url = 'http://localhost:8080/api/v2/interfaz';

    $data = [
        "projectCode" => $userProjectCode,
        "request" => $userMessage,
        "radio" => "$radioMessage"
    ];

    $options = [
        'http' => [
            'header' => "Content-Type: application/json\r\n",
            'method' => 'POST',
            'content' => json_encode($data),
        ],
    ];
};
```

Figura 4.16.- Api JavaScript formulario

El microservicio de interfaz web recibirá la información en el mismo formato que ha sido enviado (figura 4.17).

```
@Getter 20 usages
@Setter
@RequiredArgsConstructor
public class Question {
    private String projectCode;
    private String request;
    private String radio;
}
```

Figura 4.17.- Estructura de cuerpo de mensaje

Gracias a las dependencias lombok no hay de qué preocuparse a la hora de crear clases y funciones para recoger y grabar datos.

En el paso del servicio se formatea el cuerpo de mensaje a estilo JSON y se envía la solicitud a la función “responseTemplate”, el cual se encargará de realizar la comunicación con el servicio de interfaz GPT (figura 4.18).

```

@Override 2 usages
public String process(Question question, String typeGpt) throws JsonProcessingException {
    try {

        auditProducer.sendMessage(auditService.getAuditRequest(question));
        String jsonBody = objectMapper.writeValueAsString(question);
        long startTime = System.nanoTime();
        ResponseEntity<String> response =
            responseTemplate(jsonBody, (typeGpt.equals(Constants.CGPT) ? url : urlMistral));
    }
}

```

Figura 4.18.- Flujo parcial de proceso en micro interfaz web

Dicha función (definida en la figura 4.19) contiene los parámetros adecuados para poder realizar correctamente la solicitud

```

private ResponseEntity<String> responseTemplate(String jsonBody, String uri){ 1 usage
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    HttpEntity<String> requestEntity = new HttpEntity<>(jsonBody, headers);
    return restTemplate.exchange(uri, HttpMethod.POST, requestEntity, String.class);
}

```

Figura 4.19.- Configuración y llamada API a micro interfaz GPT

Una vez haya respondido el microservicio de interfaz GPT, se continúa con la ejecución del código del servicio, de modo que pasará por un proceso de formateo para adaptar la respuesta a la web en la función “replaceAlternating” (figura 4.20).

```

@Override 2 usages
public String process(Question question, String typeGpt) throws JsonProcessingException {
    try {

        auditProducer.sendMessage(auditService.getAuditRequest(question));
        String jsonBody = objectMapper.writeValueAsString(question);
        long startTime = System.nanoTime();
        ResponseEntity<String> response =
            responseTemplate(jsonBody, (typeGpt.equals(Constants.CGPT) ? url : urlMistral));
        long endTime = System.nanoTime();
        long elapsedTime = endTime - startTime;
        double elapsedTimeInSeconds = elapsedTime / 1_000_000_000.0;
        System.out.printf("Tiempo de ejecución: %.2f segundos%n", elapsedTimeInSeconds);
        String formatResponse = replaceAlternating(response.getBody()).replace(target: "\\n", replacement: "<br>");
        auditProducer.sendMessage(auditService.getAuditResponse(question, formatResponse));
        return formatResponse;
    } catch (RestClientException | JsonProcessingException e){
        auditErrorProducer.sendMessage(auditService.getAuditError(question, e.getClass().getName(), e.getMessage()));
        throw new RuntimeException(e);
    }
}

```

Figura 4.20.- Flujo completo de proceso en micro interfaz web

Ya que Mistral identifica el código de ejemplo con tres comillas simples, se debe reemplazar por aquellas indicaciones HTML adecuadas, además de impedir que el navegador pueda realizar cualquier tipo de traducción de este código (figura 4.21).

```

public static String replaceAlternating(String input) { 1 usage
    StringBuilder sb = new StringBuilder();
    String replacement1 = "<div class='codigo'><div translate='no' class='notranslate'>";
    String replacement2 = "</div></div>";
    boolean toggle = true;
    int i = 0;
    while (i < input.length()) {
        if (i + 2 < input.length() && input.charAt(i) == ' '
            && input.charAt(i + 1) == ' '
            && input.charAt(i + 2) == ' ') {
            if (toggle) {
                sb.append(replacement1);
            } else {
                sb.append(replacement2);
            }
            toggle = !toggle;
            i += 3;
        } else {
            sb.append(input.charAt(i));
            i++;
        }
    }
    return sb.toString();
}

```

Figura 4.21.- Adaptaciones de respuesta para la web

A continuación, se procede a explicar el paso intermedio donde se solicita al microservicio interfaz GPT la comunicación con el servicio IA que contiene el LLM.

Tal como se muestra en la figura 4.22, se recoge la pregunta recibida por parte del microservicio interfaz web en formato JSON para transformarla a la misma clase Question (vista en la figura 4.17).

```

@Override 1 usage
public String mistralProcess(String pregunta) throws JsonProcessingException {
    Question question = new Question();
    try {
        question = objectMapper.readValue(pregunta, Question.class);
        ProjectDataResponse projectDataResponse = projectDataService.responseCode(new RequestCode(question.getProjectCode()));
        question.setRequest(requestTemplateChatMistral(question, projectDataResponse));
        auditProducer.sendMessage(auditService.getAuditRequest(question));
        String response = responseTemplateChatMistral(question);
        auditProducer.sendMessage(auditService.getAuditResponse(question, response));
        return response;
    } catch (RestClientException | JsonProcessingException e) {
        auditErrorProducer.sendMessage(auditService.getAuditError(
            question.getProjectCode(), question.getRequest(), e.getClass().getName(), e.getMessage()));
        throw new RuntimeException();
    }
}

```

Figura 4.22.- Flujo proceso servicio interfaz GPT

Seguidamente, se pasa como parámetro de entrada el código de proceso a la clase “projectDataService”, el cual se encarga de llamar al microservicio de base de datos (figura 4.23).

```
@Override 2 usages
public ProjectDataResponse responseCode(RequestCode requestCode) throws JsonProcessingException {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    HttpEntity<String> requestEntity = new HttpEntity<>(objectMapper.writeValueAsString(requestCode), headers);
    return objectMapper.readValue(restTemplate.exchange(urlFineTune, HttpMethod.POST, requestEntity, String.class).getBody(), ProjectDataResponse.class);
}
```

Figura 4.23.- Proceso de llamada de recuperación de datos de proyecto

En el lado del microservicio de base de datos se recoge la solicitud y se realiza una búsqueda de todas las filas filtradas por código y se genera una lista con toda la información antes de mandarla de vuelta (figura 4.24)

```
@Override 1 usage
public ProjectDataResponse getProcess(String code){
    List<ProjectData> projectData = projectDataRepository.findAllByCode(code);
    return new ProjectDataResponse(projectData);
}
```

Figura 4.24.- Consulta a base de datos de proyectos

Recuperada dicha información, se genera la nueva pregunta llamando a la función "requestTemplateChatMistral", indicando la pregunta en sí y los valores recuperados para procesarlos (figura 4.25).

```
private String requestTemplateChatMistral(Question question, ProjectDataResponse projectDataResponse) 1 usage
throws JsonProcessingException, RestClientException {
    MistralRadio mistralRadio = mistralRadio(question);
    List<MistralRole> mistralRoles = new ArrayList<>();
    projectDataResponse.getFineTuneEntities().stream().forEach(fineTuneEntity -> {
        MistralRole mistralRole = MistralRole.builder().role(fineTuneEntity.getRole()).content(fineTuneEntity.getDescription()).build();
        mistralRoles.add(mistralRole);
    });
    MistralRole role = MistralRole.builder().role("user").content(question.getRequest()).build();
    mistralRoles.add(role);
    ChatMistralRequest mistralRequest = ChatMistralRequest.builder()
        .messages(mistralRoles)
        .mode("chat")
        .character("assistant")
        .max_tokens(mistralRadio.getMax_tokens())
        .temperature(mistralRadio.getTemperature())
        .top_p(mistralRadio.getTop_p())
        .seed(mistralRadio.getSeed())
        .build();
    return objectMapper.writeValueAsString(mistralRequest);
}
```

Figura 4.25.- Flujo preparatoria de pregunta

La estrategia consiste en generar un array con cada fila recogida de la base de datos y marcar cada una de ellas como información para el asistente (figura 4.26).

```

@Getter 6 usages
@Setter
@AllArgsConstructor
@Builder
public class MistralRole {
    private String role;
    private String content;
}

```

Figura 4.26.- Estructura de objeto interno de array

De este modo, se obtiene en el mensaje principal un listado de “rol” más el contenido recogido, tras lo cual se añade, en la última posición del array, la pregunta del usuario (figura 4.27).

```

MistralRole role = MistralRole.builder().role("user").content(question.getRequest()).build();
mistralRoles.add(role);

```

Figura 4.27.- Proceso de añadido de pregunta al listado

El siguiente paso es añadir dicho listado a la clase que se pasará al servicio IA donde se indicará el listado creado anteriormente, además de los valores como el modo de comunicación por chat, el rol de la IA, el cual será el de asistente, y los valores de token, temperatura, “top_p” y semillas (figura 4.28).

```

ChatMistralRequest mistralRequest = ChatMistralRequest.builder()
    .messages(mistralRoles)
    .mode("chat")
    .character("assistant")
    .max_tokens(mistralRadio.getMax_tokens())
    .temperature(mistralRadio.getTemperature())
    .top_p(mistralRadio.getTop_p())
    .seed(mistralRadio.getSeed())
    .build();
return objectMapper.writeValueAsString(mistralRequest);

```

Figura 4.28.- Estructura principal de cuerpo de mensaje

Para la recolección de estos últimos datos, se llamó dentro de la función “requestTemplateChatMistral” de la figura 4.25 a la función interna “mistralRadio” (figura 4.29).

```

private MistralRadio mistralRadio(Question question){ 1 usage
    String temperature;
    String top_p;
    String seed;
    if(question.getRadio().equals("1")){ //Preciso
        temperature = "0.2";
        top_p = "0.9";
        seed = "500";
    }else if (question.getRadio().equals("2")){//Equilibrado
        temperature = "0.7";
        top_p = "0.9";
        seed = "500";
    }else { //Creativo
        Random random = new Random();
        temperature = "1";
        top_p = "0.95";
        seed = String.valueOf( i: random.nextInt( bound: 401) + 100);
    }

    return MistralRadio.builder()
        .max_tokens("2000")
        .temperature(temperature)
        .top_p(top_p)
        .seed(seed)
        .build();
}

```

Figura 4.29.- Selección de precisión

En dicha función se recoge el valor “temperature” dentro de la pregunta, el cual corresponde a uno de los valores que se indicó en el formulario (figura 4.12) para obtener una respuesta más precisa (temperature = 0.2), equilibrada (temperature = 0.7) o creativa (temperature = 1). De este modo, adaptamos la configuración que ha sido minuciosamente escogida para este tipo de modelo IA de Mistral. Llegados a este punto, ya se ha formado el cuerpo de mensaje para poder enviar la pregunta al servidor IA (figura 4.30).

```

@Getter 2 usages
@Setter
@AllArgsConstructor
@Builder
public class ChatMistralRequest {
    private List<MistralRole> messages;
    private String mode;
    private String character;
    private String max_tokens;
    private String temperature;
    private String top_p;
    private String seed;
}

```

Figura 4.30.- Estructura de cuerpo de mensaje

Un ejemplo de este objeto formado en formato JSON sería el siguiente:

```
{
  "messages": [
    {
      "role": "user",
      "content": "¿como puedo hacer un bucle for?"
    },
    {
      "role": "assistant",
      "content": "Usos: SpringBoot, SpringBatch"
    },
    {
      "role": "assistant",
      "content": "Lenguaje: Java 11"
    }
  ],
  "mode": "chat",
  "max_tokens": 2000,
  "temperature": 1,
  "top_p": 0.9,
  "seed": 500
}
```

Figura 4.31.- Ejemplo JSON

Es en este momento en el que se procede a enviar la petición al servidor IA usando como contenedor del objeto formado a la propia clase “Question”, donde se introdujo tal como se muestra en la figura 4.32, donde “Question.getRequest” será lo que se envíe

```
private String responseTemplateChatMistral(Question question) throws JsonProcessingException, RestClientException {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    HttpEntity<String> requestEntity = new HttpEntity<>(question.getRequest(), headers);
    return restTemplate.exchange(urlChatMistral, HttpMethod.POST, requestEntity, String.class).getBody();
}
```

Figura 4.32.- Llamada a servicio IA

Una vez recibida la respuesta, se pasará a objeto JSON tal y como se muestra la figura 4.22 y se enviará al servicio interfaz web.

Problemas y complejidades encontrados

Para el desarrollo del presente proyecto, han sido varias las dificultades o pegos que ha sido necesario resolver o, en su caso, optar por una solución alternativa para abordar alguno de los problemas encontrados por el camino.

Durante la fase de desarrollo de la web, se pretendía realizar una solicitud a la aplicación de modo que, una vez respondida, ésta pudiera ser decodificada para que se pudiese diferenciar entre chat de respuesta y código de respuesta. Para ello, se requería encapsular el código de respuesta en etiquetas HTML de tipo <div> de modo que se le pudiera dar estilo desde un archivo .css, sin embargo, el resultado era que la propia etiqueta aparecía como mensaje de respuesta al desarrollador y no era lo que se buscaba.

Después de varios intentos fallidos se planteó la posibilidad de realizar la decodificación directamente en la aplicación y que la web únicamente tuviese que mostrar lo que recibía.

La IA de Mistral encapsula el código de respuestas en tres comillas simples tanto en la apertura como en el cierre. Eso permite realizar un reemplazo de dichas comillas por las etiquetas de apertura y cierre con su propio identificador. De este modo llegaría a la web con el formato correcto.

Cabe destacar que dicho reemplazo se hace en la fase final del backend, lo que quiere decir que está integrado en el proceso de respuesta del microservicio que hace de interfaz para la web.

```
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $input = json_decode(file_get_contents('php://input'), true);
    $userMessage = $input['message'];
    $userProjectCode = $input['projectCode'];
    $radioMessage = $input['radioMes'];
    $url = 'http://localhost:8080/api/v2/interfaz';

    $data = [
        "projectCode" => $userProjectCode,
        "request" => $userMessage,
        "radio" => "$radioMessage"
    ];

    $options = [
        'http' => [
            'header' => "Content-Type: application/json\r\n",
            'method' => 'POST',
            'content' => json_encode($data),
        ],
    ];

    $context = stream_context_create($options);
    $result = file_get_contents($url, false, $context);

    $response = json_decode($result, true);
    $reply = $response['choices'][0]['text'];

    echo json_encode(['reply' => $reply]);
}
?>
```

Figura 4.33.- Envío y respuesta desde web al micro interfaz


```

public static String replaceAlternating(String input) { 1 usage
    StringBuilder sb = new StringBuilder();
    String replacement1 = "<div class='codigo'>";
    String replacement2 = "</div>";
    boolean toggle = true;
    int i = 0;

    while (i < input.length()) {
        if (i + 2 < input.length() && input.charAt(i) == ' '
            && input.charAt(i + 1) == ' '
            && input.charAt(i + 2) == ' ') {
            if (toggle) {
                sb.append(replacement1);
            } else {
                sb.append(replacement2);
            }
            toggle = !toggle;
            i += 3;
        } else {
            sb.append(input.charAt(i));
            i++;
        }
    }
    return sb.toString();
}

```

Figura 4.34.- Procesamiento de respuesta antes de enviar a la web

Otra complejidad encontrada fue a la hora de realizar llamadas de tipo REST entre microservicios, así como la publicación y consumición de colas vía RabbitMQ. El problema se localiza en el cuerpo del mensaje a utilizar, donde normalmente se utiliza una estructura dentro de una clase para realizar el envío, sin embargo, las APIs fallaban y los mensajes no se publicaban en cola. Para solventar dicho problema, se optó por la conversión de dichas estructuras a una cadena de caracteres con formato de tipo JSON que se transmite a través de una variable de tipo String.

Tanto para el lado del controlador que recibe la petición como los consumidores de colas realizarán de nuevo la conversión a las clases correspondientes. Para ello, se ha usado la librería Jackson Databind que contiene las herramientas para realizar dicha conversión en ambos sentidos.

```

HttpEntity<String> requestEntity = new HttpEntity<>(objectMapper.writeValueAsString(chatRequest), headers);
return restTemplate.exchange(url, HttpMethod.POST, requestEntity, String.class).getBody();

```

Figura 4.35.- Ejemplo conversión estructura ChatRequest a String

```

@Override
public String process(String request){
    try {
        Question question = objectMapper.readValue(request, Question.class);
    }
}

```

Figura 4.36.- Ejemplo conversión de String a estructura Question

Un problema menor, pero que llevó tiempo resolver, fue en la migración de la base de datos de prueba a la base de datos externa de PostgreSQL. Al instalar las dependencias de dicha BD, así como el cargador JDBC, no se conseguía realizar adecuadamente la conexión, alegando problemas de compatibilidad.

Tras mucho investigar, se llegó a la conclusión de que el problema era de cómo estaba definido el valor autoincremental en las tablas.

```

@Id 2 usages
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

```

Figura 4.37.- Estrategia de valor generado

Esto impedía que el propio microservicio levantase con un error en el cargador que no dejaba nada en claro dónde estaba el problema. La solución fue tan simple como cambiar el valor del parámetro “strategy” a “generationType.AUTO”.

```

@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;

```

Figura 4.38.- Estrategia de valor autogenerado

El mayor problema que se ha abordado en dicho proyecto, y que todo el resultado de la aplicación dependía de ello, es el de la selección de un modelo adecuado para usarlo como base y dar vida a la IA que hay detrás, así como el entrenamiento de dicho modelo.

Para el desarrollo del proyecto se ha utilizado un portátil de última generación (procesador Intel I7-13700H, tarjeta gráfica Nvidia RTX 4050, 32 Gbytes de RAM) que, aunque potente, acaba dándose por vencido frente a modelos que ofrecen una calidad de respuesta deseable, como pueden ser los Codellama 70B. El intento de ejecución de uno de estos modelos siempre termina por consumir todos los recursos de memoria, usando archivos de paginación alojados en el disco duro y dando una experiencia de usuario pésima.

Procesos		Ejecutar nueva tarea		Finalizar tarea	
Nombre	Estado	12% CPU	97% Memoria	100% Disco	0% Red
Python		3,6%	21.650,8 MB	163,4 MB/s	0 Mbps

Figura 4.39.- Consumo de recurso hardware por parte de la IA

Es por ello que se ha realizado una búsqueda exhaustiva de modelos más livianos con 7 mil millones de parámetros o 7B. Sin embargo, la respuesta de dichos modelos dejaba mucho que desear, hasta la aparición de Mistral v0.2, modelo finalmente elegido para el presente proyecto.

La segunda parte del problema es el entrenamiento del modelo. Para la versión v0.2 de Mistral existe un modelo básico, así como otras variantes del mismo modelo en su versión cuantificada, que funcionan adecuadamente, sin embargo la opción cuantificada implica una optimización que permite obtener respuestas en menos de 5 segundos, lo que lo convierte en el modelo ideal. El problema es que esta propiedad de cuantificación no permite entrenar dicho modelo, hecho que se desconocía hasta que se revisó la información de la consola de pinokio.

```

22:18:09-719164 INFO Loading "TheBloke_Mistral-7B-Instruct-v1.2-GPTQ"
22:18:10-246166 INFO LOADER: "Exllamav2_HF"
22:18:10-246166 INFO TRUNCATION LENGTH: 32768
22:18:10-246166 INFO INSTRUCTION TEMPLATE: "Custom (obtained from model metadata)"
22:18:10-246166 INFO Loaded the model in 8.53 seconds.
22:18:38-365110 WARNING LoRA training has only currently been validated for LLaMA, OPT, GPT-J, and GPT-NeoX models. (Found model type: Exllamav2HF)
Traceback (most recent call last):
  File "C:\Users\fran\pinokio\api\oobabooga.pinokio.git\text-generation-webui\installer_files\env\Lib\site-packages\gradio\queueing.py", line 566, in process_events
    response = await route_utils.call_process_api(
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\fran\pinokio\api\oobabooga.pinokio.git\text-generation-webui\installer_files\env\Lib\site-packages\gradio\route_utils.py", line 261, in call_process_api
    output = await app.get_blocks().process_api(
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\fran\pinokio\api\oobabooga.pinokio.git\text-generation-webui\installer_files\env\Lib\site-packages\gradio\blocks.py", line 1786, in process_api
    result = await self.call_function(
               ^^^^^^^^^^^^^^^^^^^^^

```

Figura 4.40.- Consola con información Text Generation WebUI integrada en Pinokio

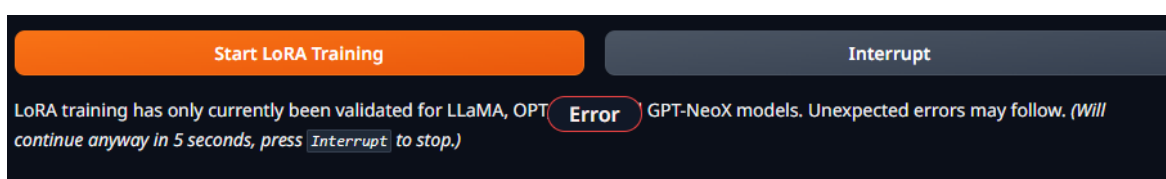


Figura 4.41.- Error al intentar entrenar un modelo

Por tanto se optó por usar el modelo base para entrenamiento y el modelo cuantificado para demostraciones in situ.

Para entrenar el modelo de Mistral, lo primero es cargar dicho modelo base (figura 4.42) para luego pasar a la pestaña de entrenamiento y seleccionamos el formato que tendrá el archivo JSON así como el archivo en sí (figura 4.43).

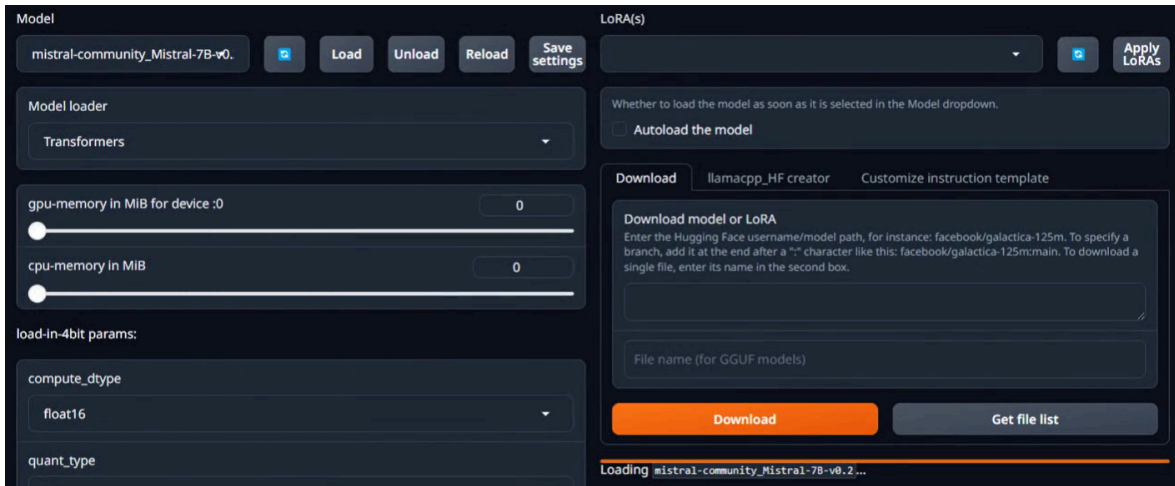


Figura 4.42.- Carga de un modelo

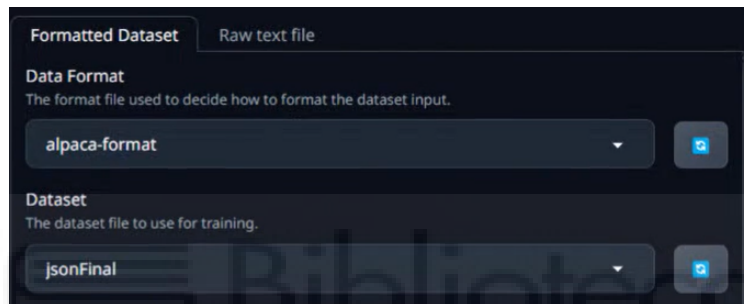


Figura 4.43.- Selección de formato y archivo json

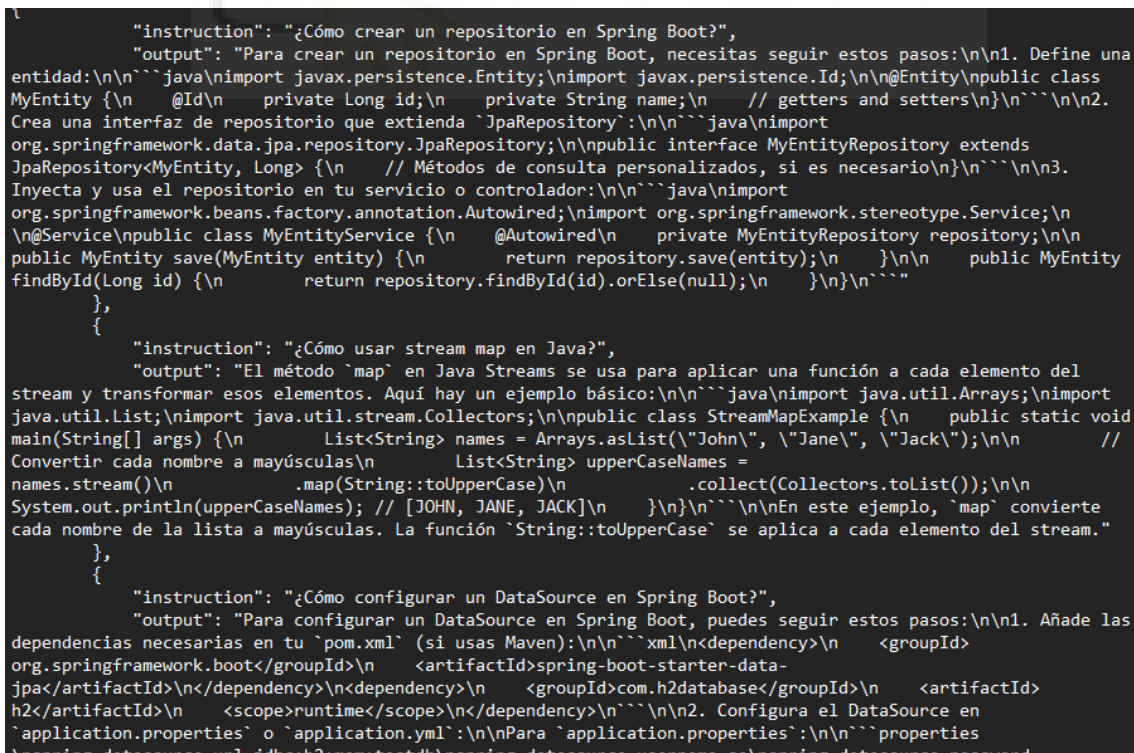


Figura 4.44.- Archivo json

Sobre la configuración de los parámetros de entrenamiento, en primera instancia se dejarán tal y como están por defecto exceptuando el “Lora Rank”, cuyo valor se sube a más de 200 para el ajuste fino tal y como se muestra en la figura 4.45.



Figura 4.45.- Configuración parámetros ajuste fino I

Una vez iniciado el entrenamiento, se debe esperar un tiempo prudencial dependiendo del tamaño del archivo y la configuración de los parámetros hasta que acabe, se vuelve a la pestaña del modelo y seleccionaremos el Lora que se ha generado y se aplicará al modelo que está en ejecución para que tenga en cuenta los nuevos parámetros (figura 4.46).

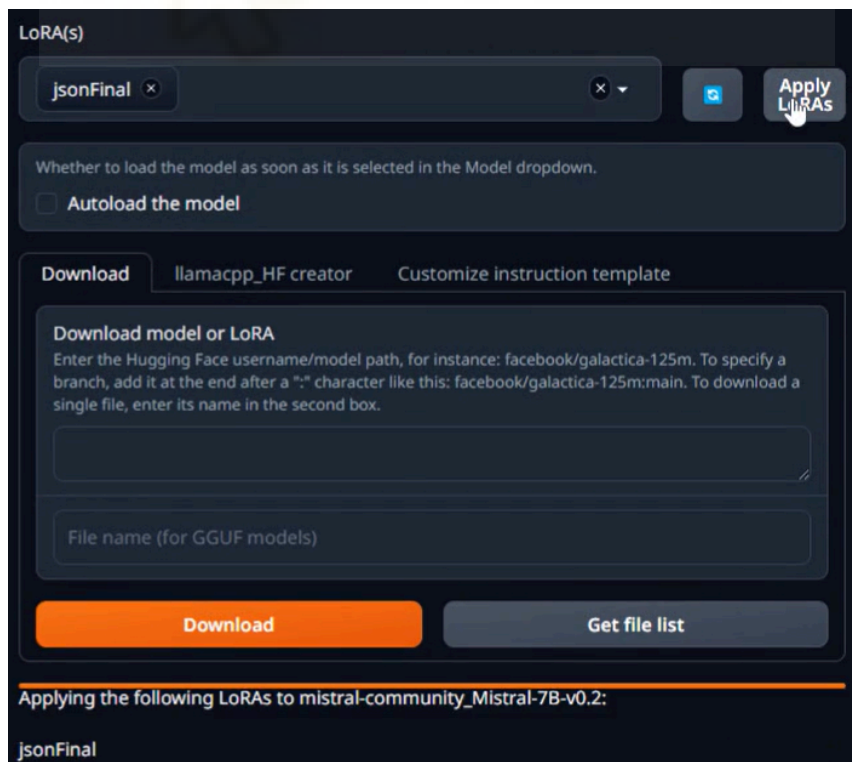


Figura 4.46.- Configuración parámetros ajuste fino II

Para ilustrar como se ha hecho el entrenamiento se ha realizado una grabación donde se muestran los detalles para poder hacer el ajuste fino, este vídeo se puede consultar en el siguiente enlace.

https://drive.google.com/file/d/1UpE0CYbQPTcQTSzoV06foiR35BR-Wfy-/view?usp=drive_link



4.5.- Pruebas

En este apartado se va a ilustrar con tres ejemplos como realizar pruebas de funcionamiento del modelo, con y sin ajuste fino:

1. ¿Podrías decirme cómo crear la interfaz de un controlador?
2. ¿Cómo recorrer una lista?
3. ¿Cómo crear una clase consumidor RabbitMQ?

En todos los casos las pruebas se han realizado utilizando el cliente Postman en vez del formulario web, ya que la respuesta tarda demasiado tiempo en obtenerse y en ese caso el navegador da un error de “*timeout*”. La respuesta generada por el modelo LLM incluye código HTML, para su correcta visualización dicho código se ha migrado a archivo .html que ya pueden renderizar correctamente en cualquier navegador. A continuación se muestran varias figuras que ilustran el resultado de estas pruebas.

Pregunta 1 sin ajuste fino

¿Podrías decirme cómo crear la interfaz de un controlador?

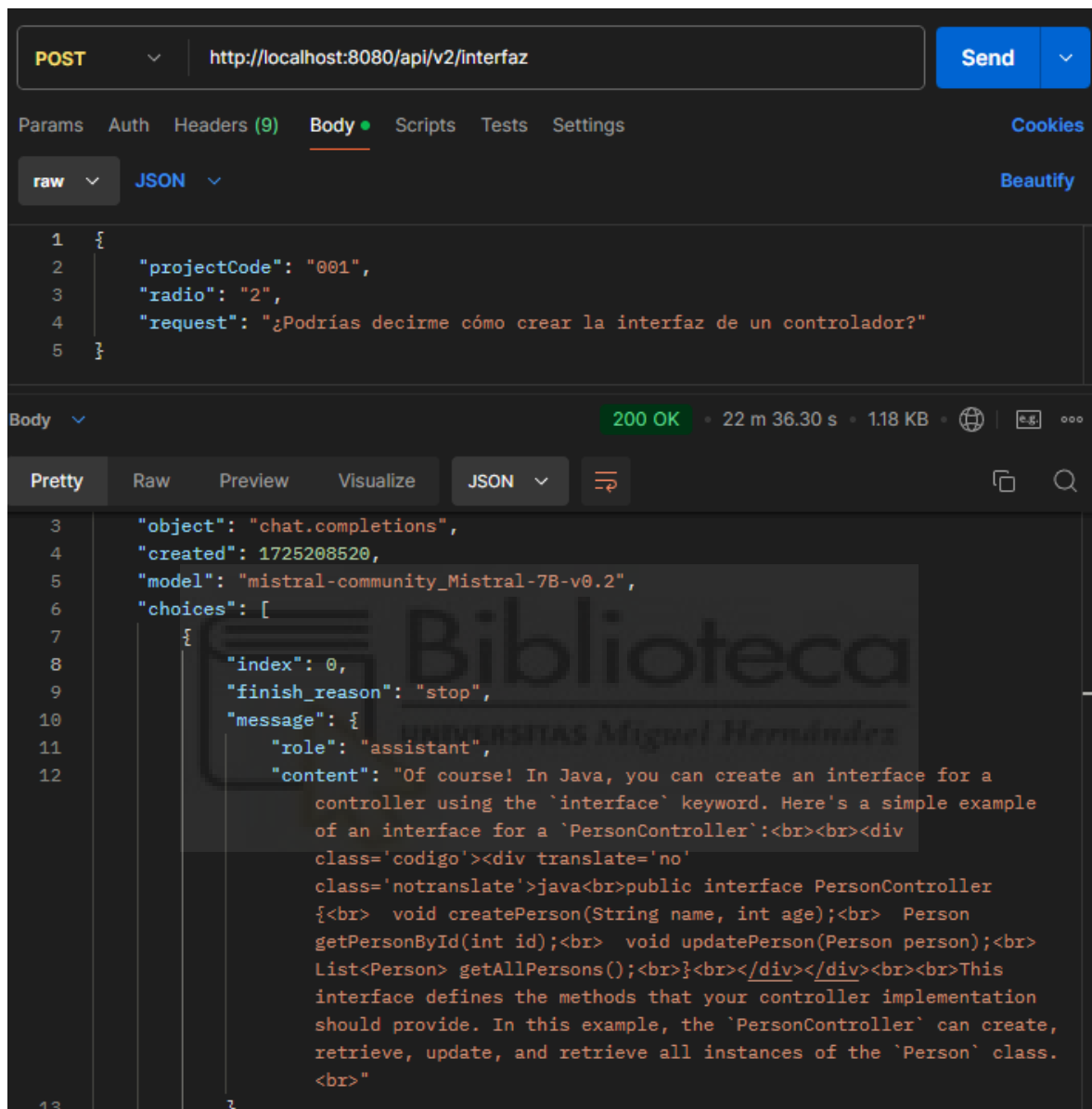


Figura 4.47.- Pregunta 1 sin ajuste fino, Postman

Of course! In Java, you can create an interface for a controller using the `interface` keyword. Here's a simple example of an interface for a `PersonController`:

```
java
public interface PersonController {
void createPerson(String name, int age);
Person getPersonById(int id);
void updatePerson(Person person);
List getAllPersons();
}
```

This interface defines the methods that your controller implementation should provide. In this example, the `PersonController` can create, retrieve, update, and retrieve all instances of the `Person` class.

Figura 4.48.- Pregunta 1 sin ajuste fino, resultado en navegador

Pregunta 1 con ajuste fino

¿Podrías decirme cómo crear la interfaz de un controlador?

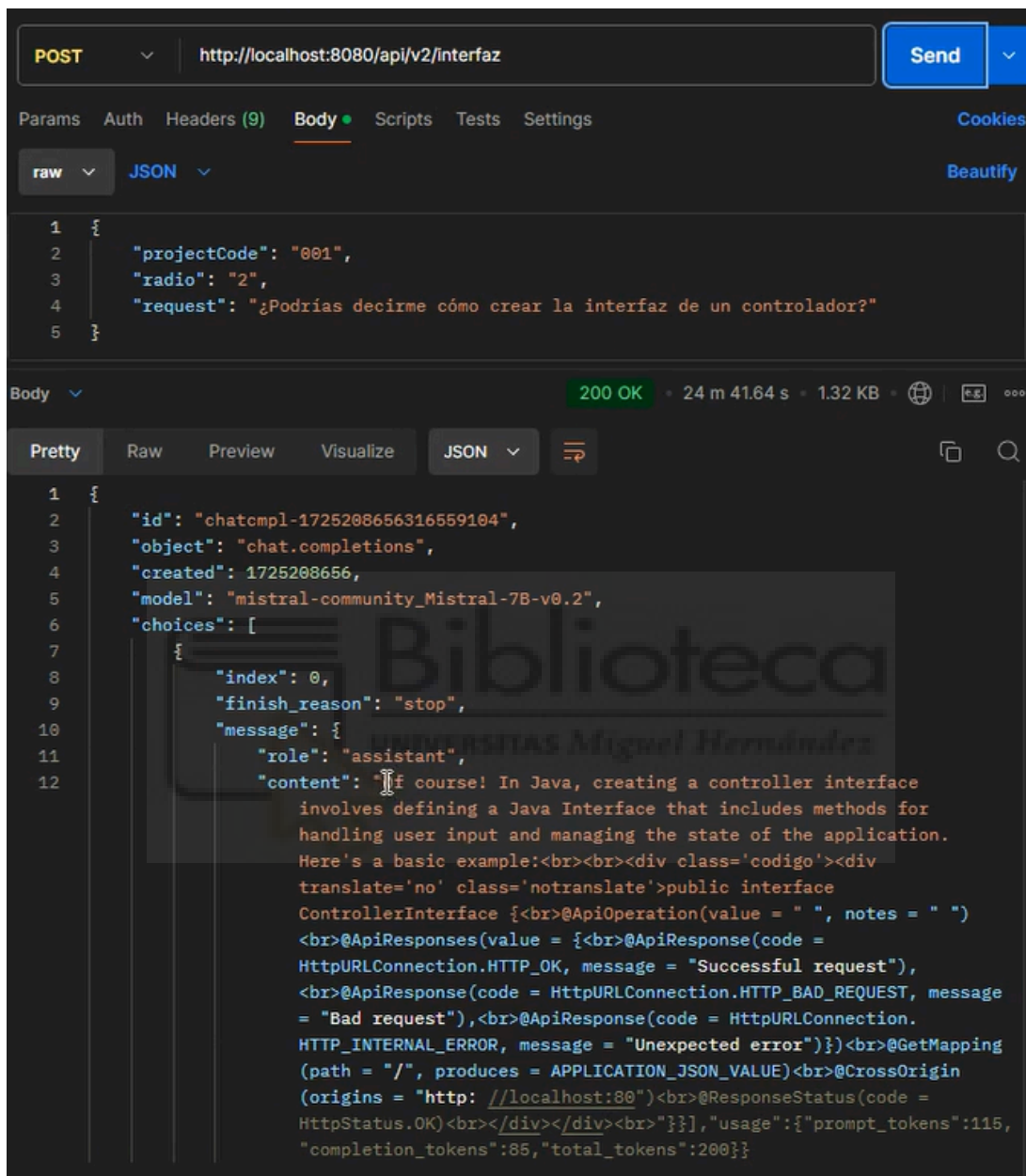


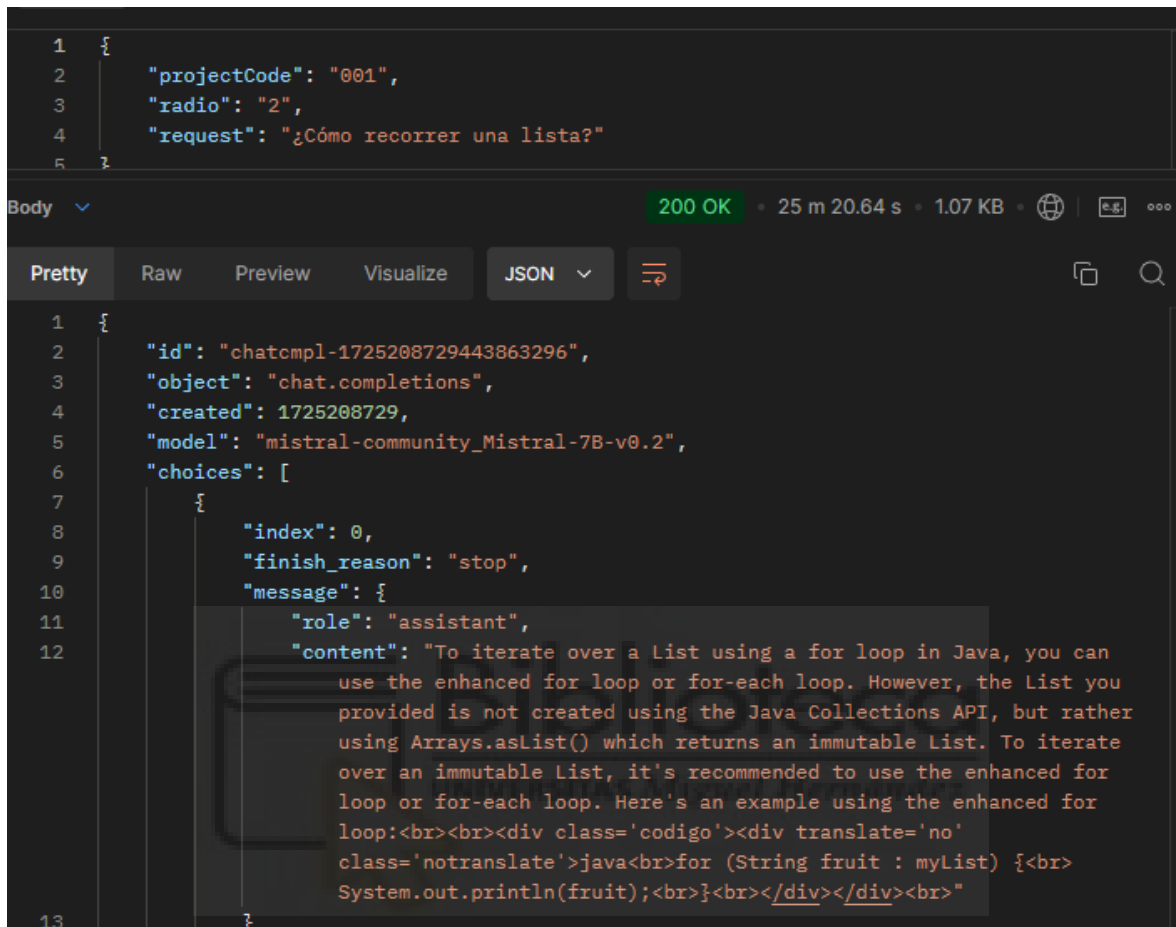
Figura 4.49.- Pregunta 1 con ajuste fino, Postman

"Of course! In Java, creating a controller interface involves defining a Java Interface that includes methods for handling user input and managing the state of the application. Here's a basic example:

```
public interface ControllerInterface {
    @ApiOperation(value = " ", notes = " ")
    @ApiResponses(value = {
        @ApiResponse(code = HttpURLConnection.HTTP_OK, message = "Successful request"),
        @ApiResponse(code = HttpURLConnection.HTTP_BAD_REQUEST, message = "Bad request"),
        @ApiResponse(code = HttpURLConnection.HTTP_INTERNAL_ERROR, message = "Unexpected error")})
    @GetMapping(path = "/", produces = APPLICATION_JSON_VALUE)
    @CrossOrigin(origins = "http://localhost:80")
    @ResponseStatus(code = HttpStatus.OK)
```

Figura 4.50.- Pregunta 1 con ajuste fino, resultado en navegador

Pregunta 2: sin ajuste fino ¿Cómo recorrer una lista?



```
1 {
2   "projectCode": "001",
3   "radio": "2",
4   "request": "¿Cómo recorrer una lista?"
5 }
```

Body 200 OK · 25 m 20.64 s · 1.07 KB

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "chatcmpl-1725208729443863296",
3   "object": "chat.completions",
4   "created": 1725208729,
5   "model": "mistral-community_Mistral-7B-v0.2",
6   "choices": [
7     {
8       "index": 0,
9       "finish_reason": "stop",
10      "message": {
11        "role": "assistant",
12        "content": "To iterate over a List using a for loop in Java, you can use the enhanced for loop or for-each loop. However, the List you provided is not created using the Java Collections API, but rather using Arrays.asList() which returns an immutable List. To iterate over an immutable List, it's recommended to use the enhanced for loop or for-each loop. Here's an example using the enhanced for loop:<br><br><div class='codigo'><div translate='no' class='notranslate'>java<br>for (String fruit : myList) {<br>System.out.println(fruit);<br>}</div></div><br>"
13      }
14    }
15  ]
16 }
```

Figura 4.51.- Pregunta 2 sin ajuste fino, Postman

To iterate over a List using a for loop in Java, you can use the enhanced for loop or for-each loop. However, the List you provided is not created using the Java Collections API, but rather using Arrays.asList() which returns an immutable List. To iterate over an immutable List, it's recommended to use the enhanced for loop or for-each loop. Here's an example using the enhanced for loop:

```
java
for (String fruit : myList) {
System.out.println(fruit);
}
```

Figura 4.52.- Pregunta 2 sin ajuste fino, resultado en navegador

Pregunta 2: con ajuste fino ¿Cómo recorrer una lista?

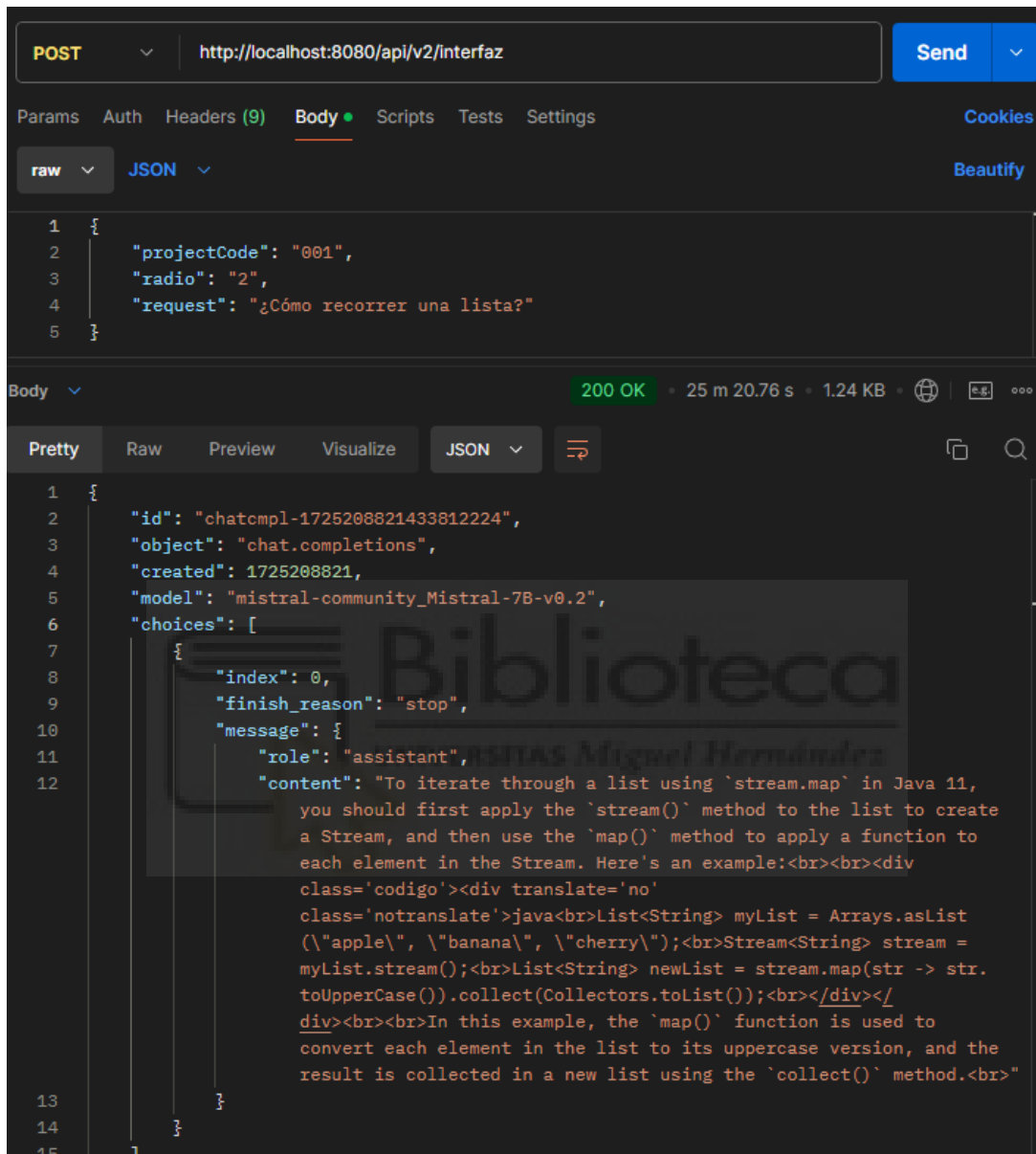


Figura 4.53.- Pregunta 2 con ajuste fino, Postman

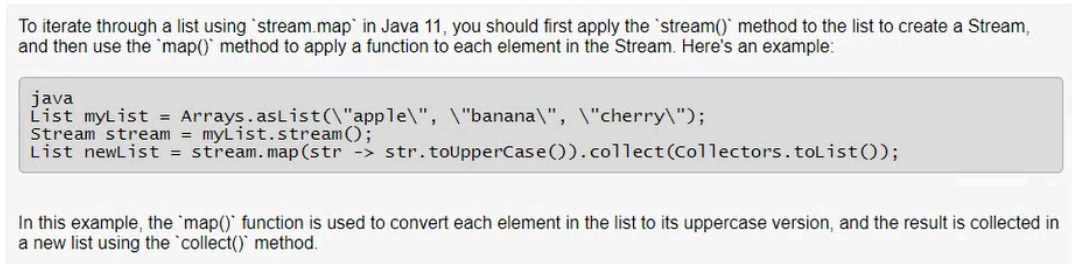
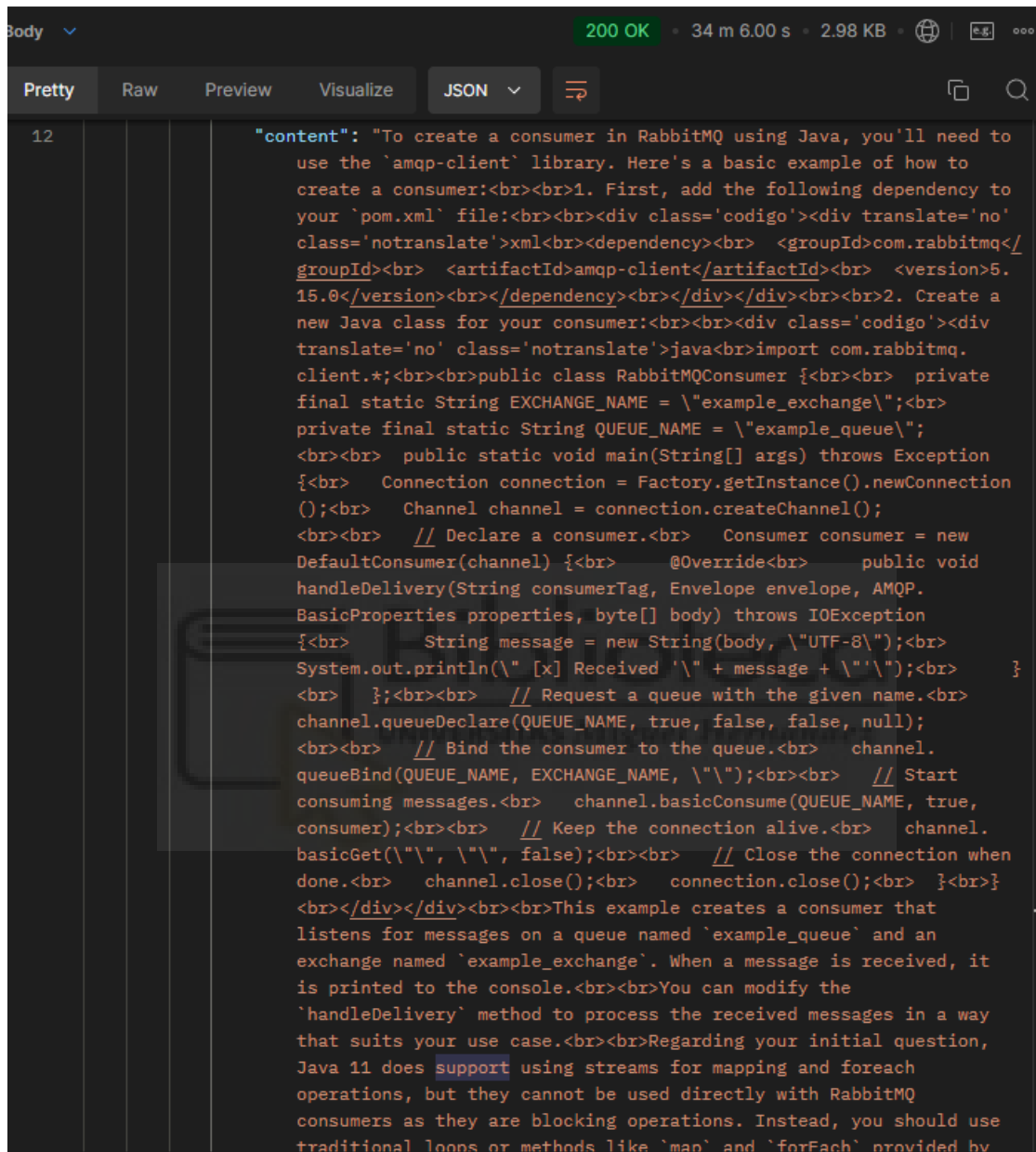


Figura 4.54.- Pregunta 2 con ajuste fino, resultado en navegador

Pregunta 3 sin ajuste fino

¿Cómo crear una clase consumidor RabbitMQ?



```
Body 200 OK 34 m 6.00 s 2.98 KB
Pretty Raw Preview Visualize JSON
12 "content": "To create a consumer in RabbitMQ using Java, you'll need to use the `amqp-client` library. Here's a basic example of how to create a consumer:<br><br>1. First, add the following dependency to your `pom.xml` file:<br><br><div class='codigo'><div translate='no' class='notranslate'>xml<br><dependency><br> <groupId>com.rabbitmq</groupId><br> <artifactId>amqp-client</artifactId><br> <version>5.15.0</version><br></dependency><br></div></div><br><br>2. Create a new Java class for your consumer:<br><br><div class='codigo'><div translate='no' class='notranslate'>java<br>import com.rabbitmq.client.*;<br><br>public class RabbitMQConsumer {<br><br> private final static String EXCHANGE_NAME = \"example_exchange\";<br> private final static String QUEUE_NAME = \"example_queue\";<br><br> public static void main(String[] args) throws Exception {<br>     Connection connection = Factory.getInstance().newConnection();<br>     Channel channel = connection.createChannel();<br><br>     // Declare a consumer.<br>     Consumer consumer = new DefaultConsumer(channel) {<br>         @Override<br>         public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {<br>             String message = new String(body, \"UTF-8\");<br>             System.out.println(\" [x] Received '\" + message + '\"\");<br>         }<br>     };<br><br>     // Request a queue with the given name.<br>     channel.queueDeclare(QUEUE_NAME, true, false, false, null);<br><br>     // Bind the consumer to the queue.<br>     channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, \"\");<br><br>     // Start consuming messages.<br>     channel.basicConsume(QUEUE_NAME, true, consumer);<br><br>     // Keep the connection alive.<br>     channel.basicGet(\"\", \"\", false);<br><br>     // Close the connection when done.<br>     channel.close();<br>     connection.close();<br> }<br></div></div><br><br>This example creates a consumer that listens for messages on a queue named `example_queue` and an exchange named `example_exchange`. When a message is received, it is printed to the console.<br><br>You can modify the `handleDelivery` method to process the received messages in a way that suits your use case.<br><br>Regarding your initial question, Java 11 does support using streams for mapping and foreach operations, but they cannot be used directly with RabbitMQ consumers as they are blocking operations. Instead, you should use traditional loops or methods like `map` and `forEach` provided by
```

Figura 4.55.- Pregunta 3 sin ajuste fino, Postman

To create a consumer in RabbitMQ using Java, you'll need to use the `amqp-client` library. Here's a basic example of how to create a consumer:

1. First, add the following dependency to your `pom.xml` file:

```
xml
com.rabbitmq
amqp-client
5.15.0
```

2. Create a new Java class for your consumer:

```
java
import com.rabbitmq.client.*;

public class RabbitMQConsumer {

    private final static String EXCHANGE_NAME = "example_exchange";
    private final static String QUEUE_NAME = "example_queue";

    public static void main(String[] args) throws Exception {
        Connection connection = Factory.getInstance().newConnection();
        Channel channel = connection.createChannel();

        // Declare a consumer.
        Consumer consumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
            properties, byte[] body) throws IOException {
                String message = new String(body, "UTF-8");
                System.out.println("[x] Received '" + message + "'");
            }
        };

        // Request a queue with the given name.
        channel.queueDeclare(QUEUE_NAME, true, false, false, null);

        // Bind the consumer to the queue.
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "");

        // Start consuming messages.
        channel.basicConsume(QUEUE_NAME, true, consumer);

        // Keep the connection alive.
        channel.basicGet("", "", false);

        // Close the connection when done.
        channel.close();
        connection.close();
    }
}
```

This example creates a consumer that listens for messages on a queue named `example_queue` and an exchange named `example_exchange`. When a message is received, it is printed to the console.

You can modify the `handleDelivery` method to process the received messages in a way that suits your use case.

Regarding your initial question, Java 11 does support using streams for mapping and foreach operations, but they cannot be used directly with RabbitMQ consumers as they are blocking operations. Instead, you should use traditional loops or methods like

Figura 4.56.- Pregunta 3 sin ajuste fino, resultado en navegador

Pregunta 3 con ajuste fino

¿Cómo crear una clase consumidor RabbitMQ?

```
1  {
2  "projectCode": "001",
3  "radio": "2",
4  "request": "¿cómo crear una clase consumidor rabbitmq?"
5  }
```

Body 200 OK • 37 m 26.04 s • 1.19 KB

Pretty Raw Preview Visualize JSON

```
1  {
2  "id": "chatcmpl-1725208958785302016",
3  "object": "chat.completions",
4  "created": 1725208958,
5  "model": "mistral-community_Mistral-7B-v0.2",
6  "choices": [
7  {
8  "index": 0,
9  "finish_reason": "stop",
10 "message": {
11 "role": "assistant",
12 "content": "Here's a basic example of how to create a
13 consumer:<br><br><div class='codigo'><div translate='no'
class='notranslate'><br>@Component<br>public class RabbitMQConsumer
<br>@Autowired<br>private ObjectMapper objectMapper;
<br>@RabbitListener(queues = "example_queue")<br>public void
consumer(String msg) {<br>try {<br>ExampleObject request =
objectMapper.readValue(msg, ExampleObject.class);<br>} catch
(JsonProcessingException e) {<br>e.printStackTrace();<br>}<br>}<br>
<br></div></div><br><br>This example creates a consumer that
listens for messages on a queue named `example_queue` and an
exchange named `example_exchange`. When a message is received, it
is printed to the console."
13 }
```

Figura 4.57.- Pregunta 3 Postman con ajuste fino

Here's a basic example of how to create a consumer:

```
@Component
public class RabbitMQConsumer {
    @Autowired
    private ObjectMapper objectMapper;
    @RabbitListener(queues = "example_queue")
    public void consumer(String msg) {
        try {
            ExampleObject request = objectMapper.readValue(msg, ExampleObject.class);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
    }
}
```

This example creates a consumer that listens for messages on a queue named `example_queue` and an exchange named `example_exchange`. When a message is received, it is printed to the console.

Figura 4.58.- Pregunta 3 Resultado con ajuste fino

En este último ejemplo se puede observar que la respuesta es mucho más simple y directa, en lugar de introducir tanta información que pueda tender a confusión.

En conclusión, el ajuste fino introducido junto a la modificación de algunos parámetros a tener en cuenta, permite que se realicen cambios en la respuesta de cada pregunta. Sin embargo, se requieren miles y miles de ejemplos para poder realizar un ajuste fino adecuado.

Las seis pruebas anteriores fueron grabadas y los vídeos correspondientes están disponibles en la nube para su visualización. Debe tenerse en cuenta que, debido a las limitaciones hardware del equipo donde está desplegado el proyecto, la ejecución de estas pruebas, en su versión “con ajuste fino” tiene una duración excesiva para mostrarla en tiempo real. Para las opciones “sin ajuste fino”, en los vídeos disponibles, también se ha utilizado la opción sin cuantificar (muy lenta). De haber usado el modelo cuantificado, optimizado para computadoras de hardware limitado, los tiempos de ejecución hubieran sido mucho menores. A continuación se indican los enlaces para acceder a estos 6 vídeos:

Pregunta 1: ¿Podrías decirme cómo crear la interfaz de un controlador?

Sin Ajuste Fino



https://drive.google.com/file/d/1G4Slns5xZcQ0qJyAsii0guBTdAf2Wo6k/view?usp=drive_link

Con Ajuste Fino



https://drive.google.com/file/d/1jlFMKL7ntr3OUppzUV-R5xmsPqT11_5Q/view?usp=drive_link

Pregunta 2: ¿Cómo recorrer una lista?

Sin Ajuste Fino



https://drive.google.com/file/d/10kUydqTYU3BwsIPww5xb_jiFy8hwT-Lq/view?usp=drive_link

Con Ajuste Fino



https://drive.google.com/file/d/1ARHKhLxOU3ckOuILbWaq2p2WOWGVREDxJ/view?usp=drive_link

Pregunta 3: ¿Cómo crear una clase consumidor RabbitMQ?

Sin Ajuste Fino



https://drive.google.com/file/d/16OGaLKnmLhEdU9wnuSgYUV_2IIAlbgyC/view?usp=drive_link

Con Ajuste Fino



https://drive.google.com/file/d/13X2B_844wNGBffCffd2d0BpmqG2o_59f/view?usp=drive_link



Capítulo 5

Conclusiones y trabajo futuro

5.1.- CONCLUSIONES

El proyecto ha sido desarrollado completamente usando Java 11, el cual es el mínimo requerido para los estándares de calidad para la empresa a la que está destinada.

Los objetivos que se han logrado alcanzar son los siguientes:

- Creación de una aplicación dividida en 3 microservicios, de modo que exista acceso limitado dependiendo del rol de uso dentro de la organización.
- Creación de una pequeña interfaz web para poder interactuar con la aplicación en cuestión.
- Uso de PostgreSQL como base de datos tal y como se exige en los estándares de calidad de la empresa.

- Ejecución de un Modelo IA destinado a responder preguntas de programación en distintos lenguajes.
- Realización de un pequeño ajuste fino apto para dar sentido a su realización, adaptándose a los estándares de calidad de la empresa.

Todo ello se ha realizado localmente en un dispositivo con un hardware limitado para su ejecución. Se espera un rendimiento muy superior al observado una vez esté alojado en los servicios de la organización.

5.2.- POSIBLES DESARROLLOS FUTUROS

La principal actualización que se debe de tener en el futuro es la constante actualización del ajuste fino que se realice en el sistema, así como implementarlo en distintos modelos de última generación que se desarrollen.

Se espera también en un futuro la posibilidad de realizar cambios en base de datos por parte de los Technical Lead para que puedan realizar modificaciones e incluso borrados de aquellos proyectos que no se requieran. Por lo pronto, únicamente deben tener acceso al servicio de base de datos los administradores, project leaders y otros usuarios de rango superior.

El sistema de registros para una posible auditoría se deberá anexar a los microservicios destinados a ello en lugar de guardar la información directamente en base de datos, de modo que cumpla con el estándar de calidad de la empresa.

Se espera también recopilar información por parte de los usuarios, de modo que puedan mostrar el feedback que han tenido con las preguntas realizadas. Dicho feedback estará registrado en base de datos, indicando la pregunta, la puntuación dada, las observaciones por parte del usuario y la respuesta que se le ha dado. De este modo podrá ser revisada por los administradores.

Debido a la limitación por hardware, no es posible arrancar un modelo multilenguaje, por lo que debería integrarse un traductor adecuado. Sin embargo, el código prepara la respuesta del sistema de modo que sólo traduce el chat, obviando por completo la parte del código.



Bibliografía

- [1] ¿Qué es la IA generativa?
<https://cloud.google.com/use-cases/generative-ai?hl=es>
Google / 01-2024

 - [2] Cómo funcionan los transformadores
<https://pro.arcgis.com/es/pro-app/latest/tool-reference/geoai/how-transformers-work.htm>
ArcGIS / 12-2023

 - [3] ¿Qué es Transformers?
<https://gamco.es/glosario/transformers/#:~:text=La%20arquitectura%20Transformador%20se%20basa,y%20el%20reconocimiento%20de%20voz.>
Gamco / 01-2024
-

- [4] ¿Qué son los modelos autorregresivos?
<https://aws.amazon.com/es/what-is/autoregressive-models/>
Amazon Web Services / 12-2023
- [5] ¿Qué es Modelo autorregresivo?
<https://gamco.es/glosario/modelo-autorregresivo/#:~:text=Un%20modelo%20autorregresivo%20>
Gamco / 01-2024
- [6] ¿Cómo puedo formar mi propio modelo GPT?
<https://botpress.com/es/blog/how-can-i-train-my-own-gpt-model>
botpress (2023) / 01-2024
- [7] Fine-tuning Preparing-your-dataset
<https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset>
Openai (2024)/ 01-2024
- [8] Título del libro, artículo o web
<https://santander.com/es/sobre-nosotros/nuestra-historia>
Banco Santander (2022) / 12-2023
- [9] A Brief History of Generative AI
<https://www.linkedin.com/pulse/brief-history-generative-ai-jhon-jairo-murillo-giraldo/>
Jhon Jairo Murillo Giraldo (2023) / 12-2023
- [10] Stack Overflow Web
<https://stackoverflow.com/>
Stack Overflow/ 01-2024
- [11] GPT-4 App Development Guide: Transform App Development with AI
<https://docs.kanaries.net/articles/gpt-4-coding>
Rajiv Chandra (2023) / 01-2024
- [12] Introducing GPTs
<https://openai.com/blog/introducing-gpts>
Openai (2023) / 01-2024
- [13] A Brief History of The Generative Pre-trained Transformer (GPT)
<https://blog.wordbot.io/ai-artificial-intelligence/a-brief-history-of-the-generative-pre-trained-transformer-gpt-language-models/>
Adam VanBuskirk (2023) / 01-2024

- [14] ¿Qué es el procesamiento del lenguaje natural?
<https://www.ibm.com/es-es/topics/natural-language-processing>
IBM / 01-2024
- [15] El procesamiento del lenguaje natural, tecnología en transición
https://cvc.cervantes.es/obref/congresos/sevilla/tecnologias/ponenc_carbonell.htm
Jaime Carbonell (2023) / 01-2024
- [16] Qué son las redes adversarias generativas
<https://techaffinity.com/blog/redes-adversarias-generativas/>
Rodrigo Artiga (2023) / 02-2024
- [17] El auge de la IA generativa en el sector tecnológico: ¿dónde está su techo?
<https://www.computing.es/opinion/el-auge-de-la-ia-generativa/>
Rafael Conde del Pozo (2023) / 03-2024
- [18] ¿Qué es la IA generativa? Todo lo que necesita saber
<https://www.computerweekly.com/es/definicion/Que-es-la-IA-generativa-Todo-lo-que-necesita-saber>
George Lawton (2023) / 03-2024
- [19] Evolución de la Inteligencia Artificial
<https://inlocrobotics.com/es/evolucion-de-la-inteligencia-artificial/>
Inlocrobotics (2021) / 03-2024
- [20] La Historia de la Inteligencia Artificial: Desde sus Orígenes hasta el Presente
<https://medium.com/@natisr/historia-de-la-inteligencia-artificial-63277f78fe2c>
Natalia Rodríguez (2023) / 03-2024
- [21] Alan Turing, el padre de la inteligencia artificial
<https://www.cultura.gob.ar/alan-turing-el-padre-de-la-inteligencia-artificial-9162/>
cultura.gob (2020) / 03-2024
- [22] Breve historia de la inteligencia artificial: origen, datos y auge de la IA
<https://www.elfernativa.com/historia-inteligencia-artificial/>
elfernativa.com (2023) / 03-2024
- [23] Inteligencia artificial según John McCarthy: definición y aportaciones
<https://iccsi.com.ar/que-es-la-inteligencia-artificial-segun-john-mccarthy/>
iccsi.com (2023) / 03-2024

- [24] Deep Learning o Aprendizaje profundo : ¿qué es?
<https://datascientest.com/es/deep-learning-definicion>
datascientest.com (2022) / 03-2024
- [25] ¿Qué es la IA generativa?
<https://aws.amazon.com/es/what-is/generative-ai/>
Amazon (2023) / 03-2024
- [26] Inteligencia artificial generativa: qué es, cómo funciona y cuáles son sus riesgos
<https://adevinta.es/stories/articles/inteligencia-artificial-generativa-que-es-como-funciona>
adevinta.es (2023) / 03-2024
- [27] Cómo funciona la inteligencia artificial generativa: una mirada profunda
<https://juliomayol.com/como-funciona-la-inteligencia-artificial-generativa-una-mirada-profunda/>
Julio Mayol (2024) / 03-2024
- [28] Todo lo que necesitas saber sobre las GAN: Redes Generativas Antagónicas
<https://puentesdigitales.com/2019/04/05/todo-lo-que-necesitas-saber-sobre-las-gan-redes-generativas-antagonicas/>
Javier Buhigas (2019) / 03-2024
- [29] ¿Cómo funcionan los Transformers?
<https://www.aprendemachinelearning.com/como-funcionan-los-transformers-espanol-nlp-gpt-bert/>
aprendemachinelearning.com (2022) / 03-2024
- [30] ¿Qué es un Modelo Transformer?
<https://la.blogs.nvidia.com/2022/04/19/que-es-un-modelo-transformer/>
Rick Merritt (2022) / 03-2024
- [31] ¿Qué es un modelo de lenguaje grande (LLM)?
<https://www.elastic.co/es-es/what-is/large-language-models>
elastic.co (2023) / 03-2024
- [32] ¿Qué es el procesamiento del lenguaje natural (NLP)?
<https://www.ibm.com/es-es/topics/natural-language-processing>
ibm.com (2023) / 03-2024

- [33] Procesamiento del lenguaje natural ¿qué es?
<https://www.iic.uam.es/inteligencia/que-es-procesamiento-del-lenguaje-natural/>
Antonio Moreno (2024) / 03-2024
- [34] ¿Qué son las incrustaciones de vectores?
<https://www.elastic.co/es/what-is/vector-embedding>
elastic.co (2023) / 03-2024
- [35] Feed Forward Neural Network
<https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network>
deepai.org (2022) / 03-2024
- [36] Redes Neuronales Recurrentes
<https://torres.ai/redes-neuronales-recurrentes/>
torres.ai (2019) / 03-2024
- [37] What is Mistral (AI)?
<https://www.androidpolice.com/mistral-ai-guide/>
Zach Gray, Steven Wikelman and Ben Khalesi (2024) / 03-2024
- [38] Mistral AI
https://golden.com/wiki/Mistral_AI-X93AJ6D
golden.com (2023) / 03-2024
- [39] A brief history of LLaMA models
<https://agi-sphere.com/llama-models/>
Andrew Sagio (2023) / 03-2024
- [40] AI Showdown: GPT-4-x-Alpaca vs. Vicuna, GPT-4 as the judge
https://www.reddit.com/r/LocalLLaMA/comments/12lksqo/ai_showdown_gpt4xalpaca_vs_vicuna_gpt4_as_the/
reddit.com (2023) / 03-2024
- [41] What is Llama 2?
<https://www.ibm.com/topics/llama-2>
Dave Bergmann (2023) / 03-2024
- [42] Meta and Microsoft Introduce the Next Generation of Llama
<https://about.fb.com/news/2023/07/llama-2/>
Meta (2023) / 03-2024

- [43] A Comprehensive Overview of Transformer-Based Models: Encoders, Decoders, and More
<https://medium.com/@minh.hoque/a-comprehensive-overview-of-transformer-based-models-encoders-decoders-and-more-e9bc0644a4e5>
Minhajul Hoque (2023) / 03-2024
- [44] Prompt Engineering Guide - Prompting Techniques
<https://www.promptingguide.ai/techniques>
promptingguide.ai (2024) / 04-2024
- [45] Pinokio Computer
<https://pinokio.computer/>
Pinokio Computer (2024) / 04-2024
- [46] Pinokio: The Ultimate AI Browser
<https://www.linkedin.com/pulse/pinokio-ultimate-ai-browser-joe-stallings/>
Joe Stallings (2023) / 04-2024
- [47] Tutorial - text-generation-webui
https://www.jetson-ai-lab.com/tutorial_text-generation.html
jetson-ai-lab (2024) / 04-2024
- [48] Mistral AI
<https://mistral.ai/>
mistral.ai (2024) / 04-2024
- [49] Mistral-7B-Instruct-v0.2
<https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2>
The Mistral AI Team (2024) / 05-2024
- [50] Mistral-7B-Instruct-v0.2-GPTQ
<https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.2-GPTQ>
The Mistral AI Team (2024) / 05-2024
- [51] Text Generation WebUI
<https://github.com/oobabooga/text-generation-webui>
Oobabooga (2024) / 05-2024
- [52] PostgreSQL: The World's Most Advanced Open Source Relational Database
<https://www.postgresql.org/>
Postgresql (2024) / 05-2024

- [53] Diferencias entre PostgreSQL y SQL Server
<https://cloud.google.com/learn/postgresql-vs-sql>
Google Cloud (2024) / 05-2024
- [54] DBeaver Community - Free Universal Database Tool
<https://dbeaver.io/>
Dbeaver.io (2024) / 05-2024
- [55] What is Java
https://www.java.com/es/download/help/whatis_java.html
java.com (2024) / 05-2024
- [56] Java Development Kit
<https://www.ibm.com/docs/es/i/7.3?topic=platform-java-development-kit>
IBM (2021) / 05-2024
- [57] Spring Boot
<https://spring.io/projects/spring-boot>
Spring.io (2024) / 05-2024
- [58] What's New in Maven
<https://mavenrepository.com/>
MvnRepository (2024) / 05-2024
- [59] Package - jackson Databind
<https://developer.adobe.com/experience-manager/reference-materials/6-5/javadoc/com/fasterxml/jackson/databind/package-summary.html>
Adobe.com (2024) / 05-2024
- [60] The “Spring WebFlux Framework”
<https://docs.spring.io/spring-boot/reference/web/reactive.html>
Spring.io docs (2024) / 05-2024
- [61] Introduction to SpringFox: An Easy Way to Document Your Spring APIs
<https://medium.com/thefreshwrites/introduction-to-springfox-an-easy-way-to-document-your-spring-apis-267533b485c3>
Samuel Catalano (2023) / 05-2024
- [62] Project Lombok
<https://mavenrepository.com/artifact/org.projectlombok/lombok>
MvnRepository (2024) / 05-2024

- [63] Spring AMQP
<https://spring.io/projects/spring-amqp>
Spring.io (2024) / 05-2024
- [64] Spring Data JPA
<https://spring.io/projects/spring-data-jpa>
Spring.io (2024) / 05-2024
- [65] Spring Boot – H2 Database
<https://www.geeksforgeeks.org/spring-boot-h2-database/>
Geeksforgeeks.org (2023) / 05-2024
- [66] Pinokio - AI Browser
<https://www.linkedin.com/pulse/all-ai-tools-free-your-local-computer-privately-pinokio-jab%C5%82onka-2h8af/>
Jacek Jabłonka (2024) / 05-2024
- [67] Pinokio introduction
<https://program.pinokio.computer/>
pinokio.computer (2024) / 05-2024
- [68] Introduction to Postman for API Development
<https://www.geeksforgeeks.org/introduction-postman-api-development/>
geeksforgeeks.org (2024) / 05-2024
- [69] IntelliJ IDEA overview
<https://www.jetbrains.com/help/idea/discover-intellij-idea.html>
jetbrains.com (2024) / 05-2024
- [70] RabbitMQ for beginners - What is RabbitMQ?
<https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html>
cloudamqp.com (2019) / 05-2024
- [71] ¿Qué son los datos de entrenamiento en el aprendizaje automático?
<https://es.shaip.com/blog/the-only-guide-on-ai-training-data-you-will-need-in/>
shaip.com (2024) / 06-2024
- [72] Alpaca Instructions for Large Language Model
<https://www.kaggle.com/datasets/mnavaid/alpaca-instructions-for-large-lanuage-model>
kaggle.com - Muhammad Navaid (2023) / 06-2024

Anexo I

Text Generation WebUI

AI.1.- INSTALACIÓN

Una de las aplicaciones dentro de Pinokio para la generación de texto a través de Modelos IA es “Text Generation Web UI”. Se puede encontrar en el buscador de Pinokio para ser instalada con un par de clics [47]. Al instalarlo aparecerá dentro de la bandeja principal de aplicaciones instaladas de Pinokio en modo de listado.

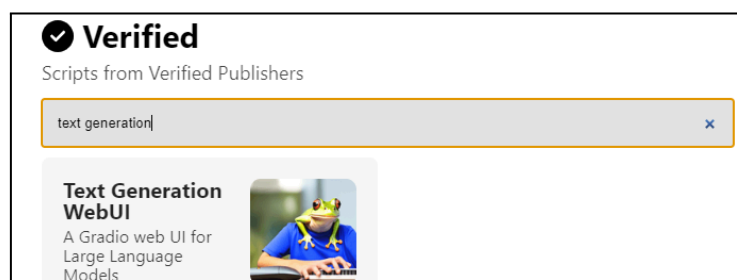


Figura AI.1: Buscador Pinokio

Haciendo Clic en él, se accede a una sencilla interfaz para poder borrar la aplicación, consultar sus archivos o directamente ejecutar la aplicación.

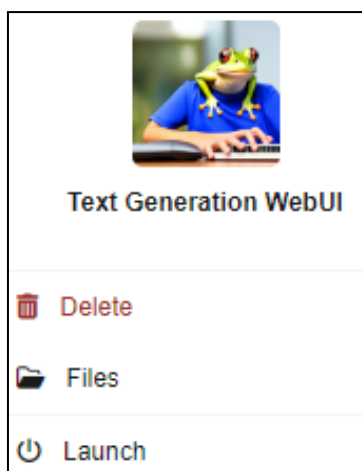


Figura AI.2: Text Generation WebUI

Pulsando en Launch (figura 3.2), se comenzará a ejecutar la aplicación, donde aparecerá un enlace local para poder acceder a la aplicación a través del navegador web. Es aconsejable realizarlo de este modo, ya que se puede navegar por la aplicación sin perder de vista la consola de comandos, donde aparecerá cualquier aviso o error que tenga dicha app.

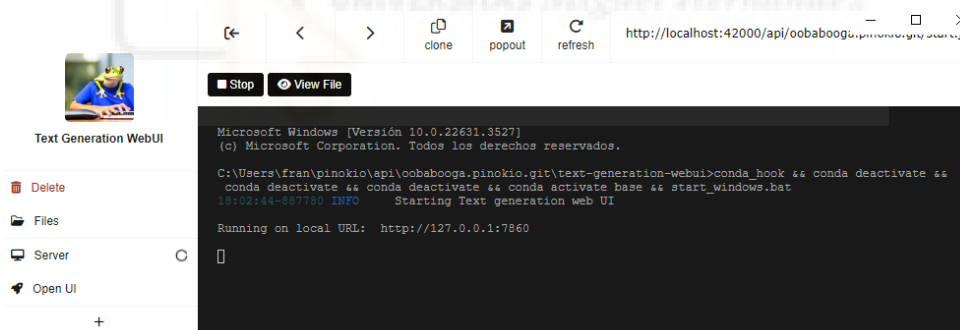


Figura AI.3: Consola comandos Text Generation WebUI

En el caso de no desear usar navegador, se puede intercalar ambas interfaces pulsando en los botones "Server" y "Open UI".

AI.2.- CONFIGURACIÓN

Una vez dentro de la App, se puede acceder a las distintas pestañas donde se puede chatear con el modelo que se haya elegido, descargar modelos y cargarlos en el sistema, entrenarlos y habilitar herramientas como el acceso por api, entre otros.

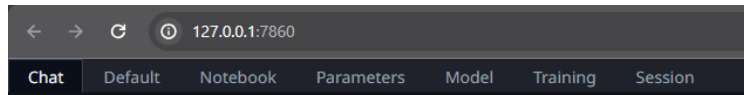


Figura AI.4: Menú Text Generation WebUI

La pestaña Model es la más interesante, ya que es donde se descarga el modelo a utilizar y se configura con un cargador. Para descargar un modelo se accede a la web <https://huggingface.co/> y se buscará el modelo que se desea usar. En nuestro caso usaremos `codellama/CodeLlama-7b-hf` a modo de ejemplo por su sencillez y se copiará el nombre con el icono habilitado para ello.

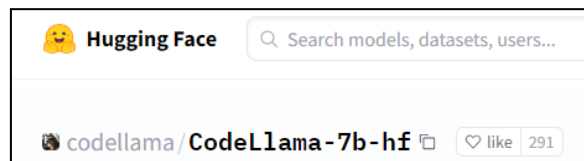


Figura AI.5: Buscador Hugging Face

Posteriormente, en la pestaña “*model*” de la app, se indicará el nombre copiado del modelo a usar y se pulsará sobre el botón download.

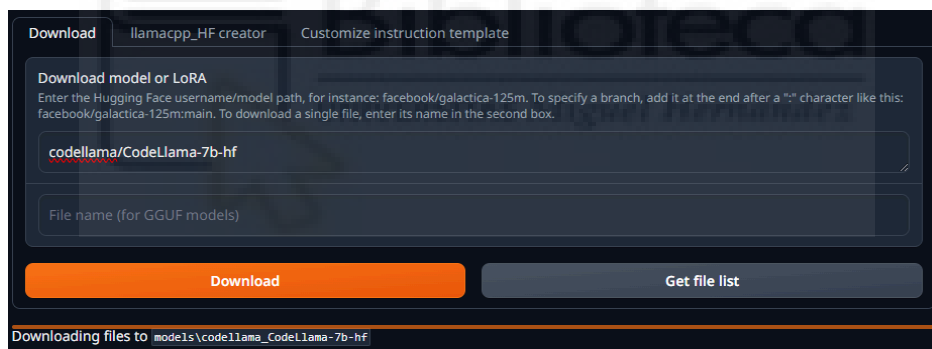


Figura AI.6: Descarga de modelos

Mientras se descarga el modelo, se puede observar en la consola de comandos el proceso de descarga hasta que esté completado.

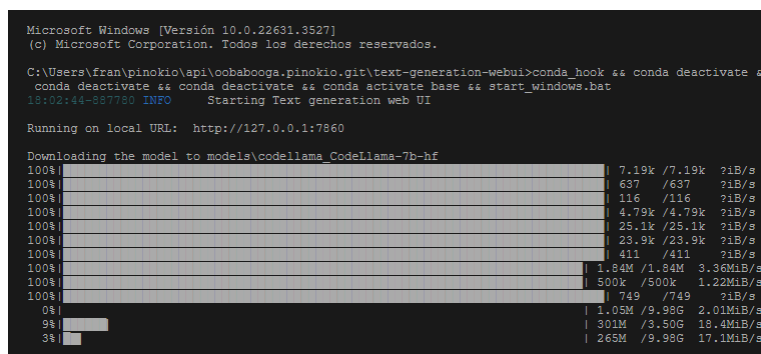


Figura AI.7: Descarga archivos de Modelo por consola

Una vez descargado, se puede encontrar dentro del listado de modelos descargados para poder seleccionarlo.

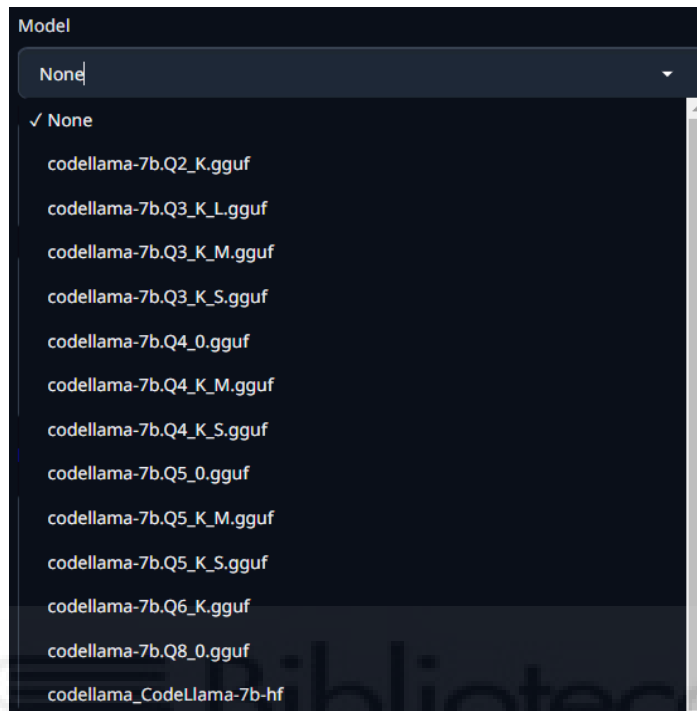


Figura AI.8: Listado Modelos descargados

El siguiente paso será seleccionar un cargador para la ejecución del modelo elegido. Como por ejemplo Transformer.

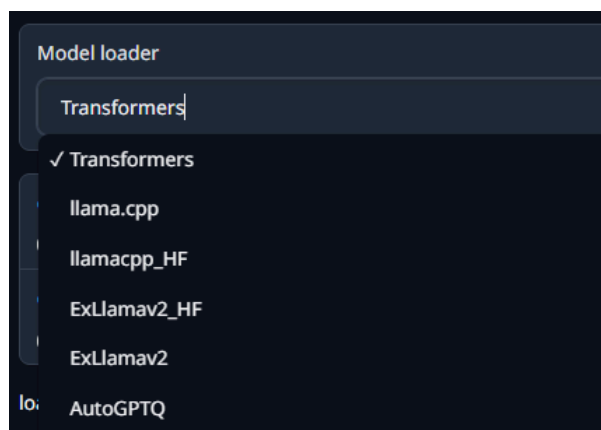


Figura AI.9: Listado de cargadores de modelos

Cada cargador debe configurarse para adaptarse al Hardware donde se desea ejecutar dicho modelo. Dependiendo de esta configuración se obtendrá un modelo más o menos eficiente, además de variar la respuesta a las peticiones realizadas.

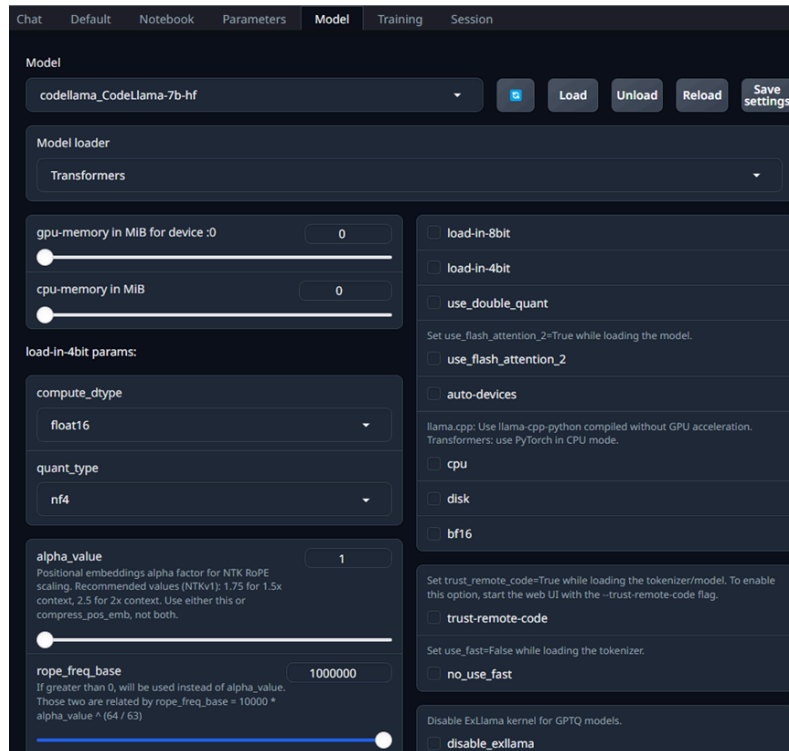


Figura AI.10: Configuración hardware

En la pestaña Session se contemplan todas las herramientas externas que podrían usarse como una expansión del servicio. En el caso actual únicamente se necesitará habilitar la herramienta API, ya que se desea realizar una comunicación a través de peticiones de tipo REST.

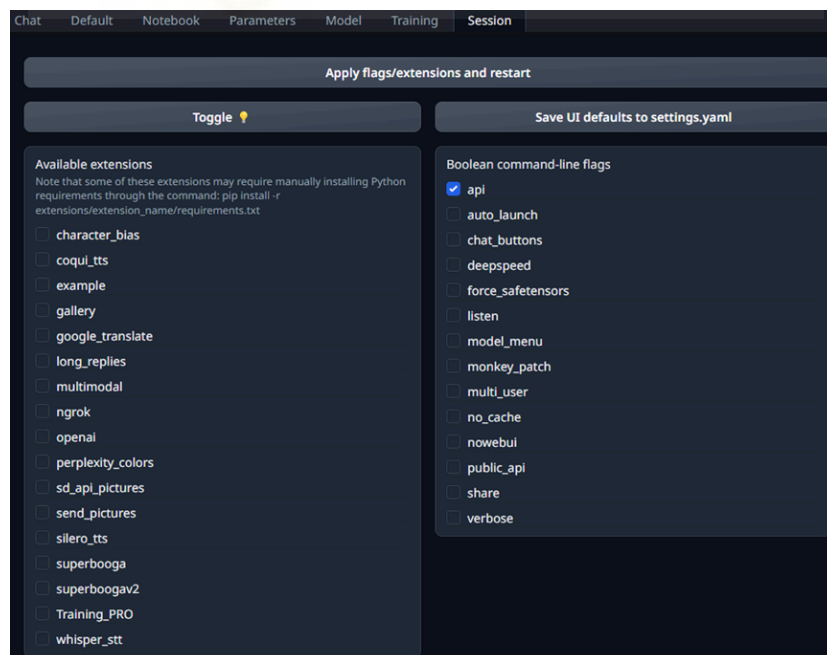


Figura AI.11: Apis externas

AI.3.- MANUAL DE USO

Una vez realizados los pasos anteriores. Se podrá realizar una petición por medio de la aplicación Postman donde indicaremos la siguiente url y estructura en el cuerpo del mensaje:

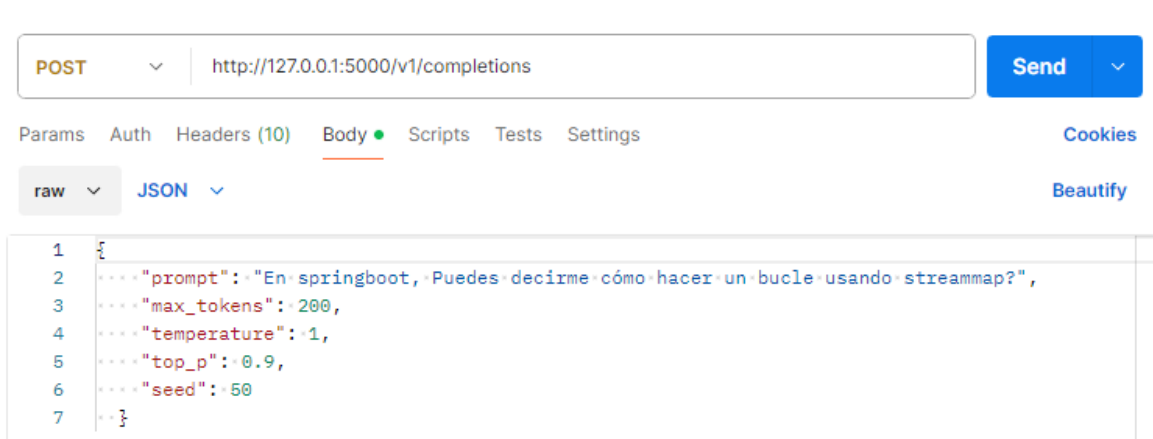


Figura AI.12: Configuración API en Postman

- Prompt: en esta variable se indica la pregunta que se quiere realizar a la IA.
- Max_tokens: Se le indica cuántos tokens debe de tener en cuenta a la hora de responder a la petición, de modo que un token puede corresponder a una palabra, un espacio o un signo de puntuación. Cuanto más tokens más información se recibirá en la respuesta pero puede repercutir en una mayor carga de trabajo.
- Temperature: La temperatura es una variable que se toma en cuenta durante el proceso de muestreo, de modo que, si ésta se acerca a 0 la respuesta será más predecible ya que usa los tokens más probables. En cambio, si supera la cifra 1 es posible que la respuesta sea más aleatoria y creativa por parte de la IA, logrando respuestas mucho más variadas.
- Top_p: También controla la aleatoriedad en la respuesta, pero en lugar de ajustar la probabilidad de cada token, lo que hace es limitar el conjunto de tokens considerados a aquellos que, al sumar sus probabilidades, sea menor o igual que P. Si top_p tiene valor 0.9, sólo se considerarán los tokens que tienen una probabilidad de un 90% o inferior, de modo que conseguimos una pequeña diversidad y creatividad.
- Seed: Las semillas son importantes ya que se utiliza para el control de la aleatoriedad, ya que repercute directamente en el generador de números aleatorios dentro de la IA. De este modo, a misma temperatura, top_p y tokens, al cambiar el núm de semillas podemos obtener valores totalmente diferentes.

Una vez se tienen claros los conceptos, ya está el API lista para realizar una petición:

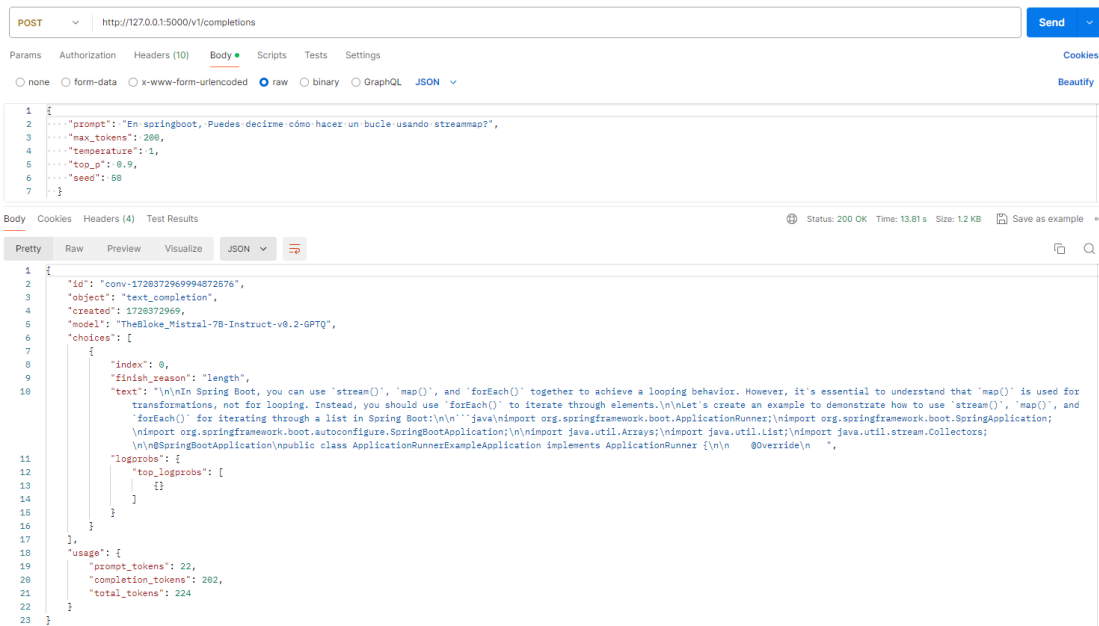


Figura AI.13: Respuesta API en Postman

Se observa que, además de la respuesta que se busca, también obtenemos datos de interés para el desarrollador, permitiendo ajustar los parámetros de entrada para mejorar la respuesta al menor coste posible.

AI.4.- AJUSTE FINO

En la pestaña de training es donde se realizará el entrenamiento del modelo que esté en ejecución en estos momentos. Debemos seleccionar el formato en el que se encuentre las preguntas/respuestas y pasarlas en un segundo archivo antes de comenzar a entrenar.

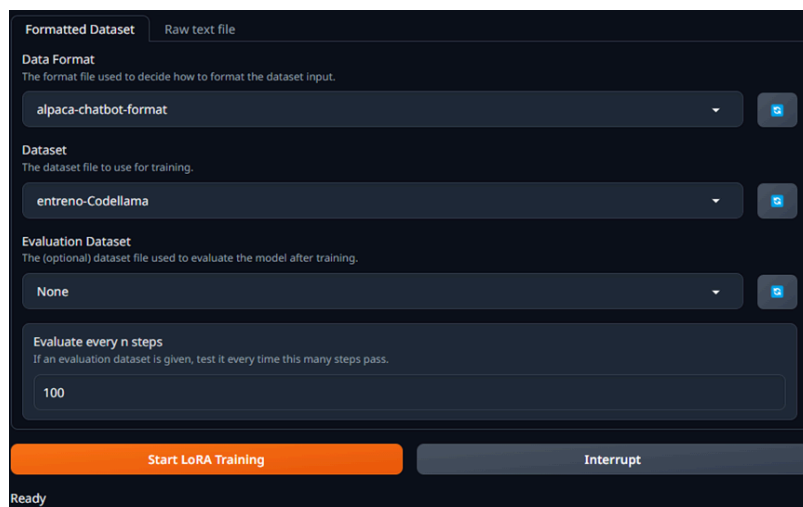


Figura AI.14: Panel de entrenamiento

Esto generará un paquete o LoRA, el cual será cargado junto al modelo, afectando en la respuesta de dicho modelo y logrando un mejor o peor resultado dependiendo de la rigurosidad de los datos que se han usado para éste.

Para cargar el/los Loras correspondientes se debe seleccionar la pestaña Model y en la sección derecha seleccionar uno o más Loras generados y aplicarlos.

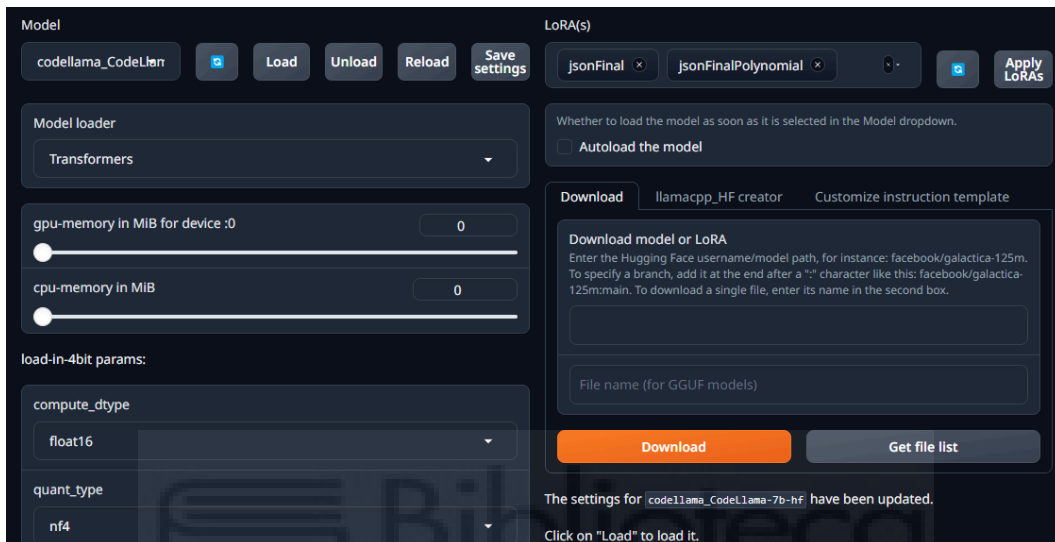


Figura AI.15: Cargando LorAs

Anexo II

Pinokio



AII.1.- INSTALACIÓN

Para la instalación de Pinokio en un PC personal debemos entrar a la web <https://pinokio.computer/> y aparecerá la opción de descarga en la primera pantalla.

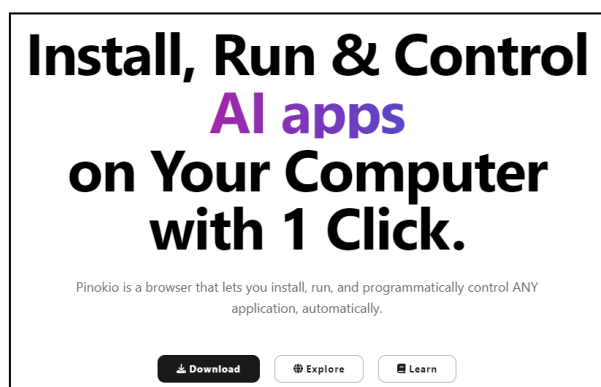


Figura AII.1: Web inicial Pinokio

Una vez descargado, la instalación no tiene gran misterio ya que no tarda más de 30 segundos en realizar dicha instalación.

Una vez instalado, al abrir la aplicación aparecerá la ventana principal con un buscador donde se puede acceder a todas las aplicaciones disponibles.

Para la prueba de cómo instalar cualquier aplicación en pinokio se instalará hallo haciendo clic encima de su sección.

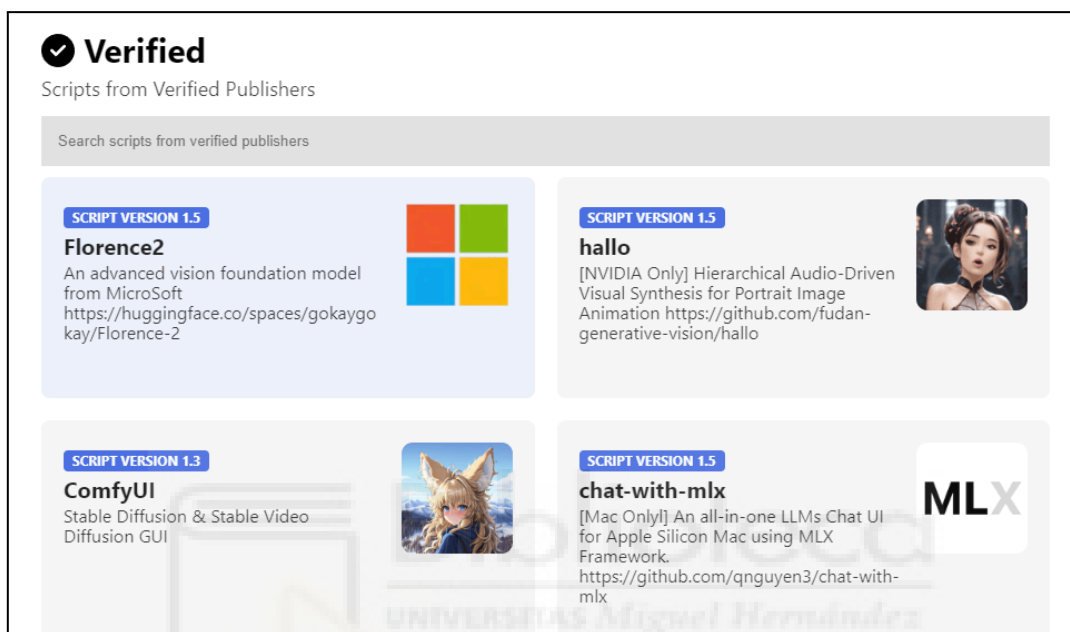


Figura AII.2: Buscador aplicaciones Pinokio

A continuación pulsaremos sobre el botón de download tal como muestra la siguiente imagen.

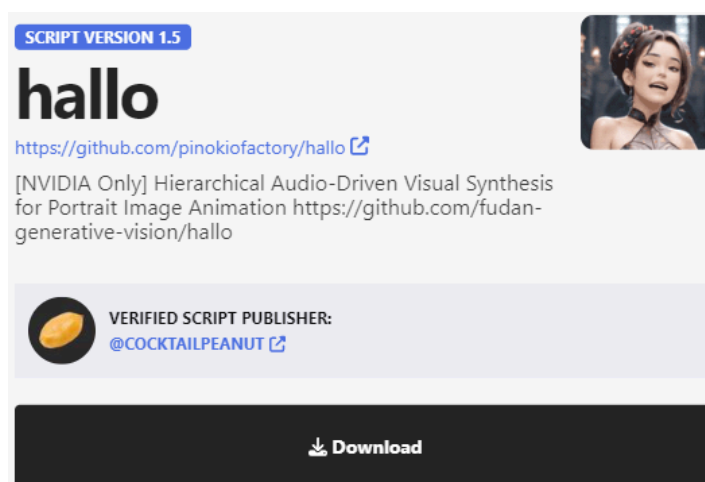


Figura AII.3: Ventana instalación Hallo en Pinokio

Ahora sólo quedaría esperar a que pinokio descargue todos los paquetes necesarios y él mismo realizará la instalación.

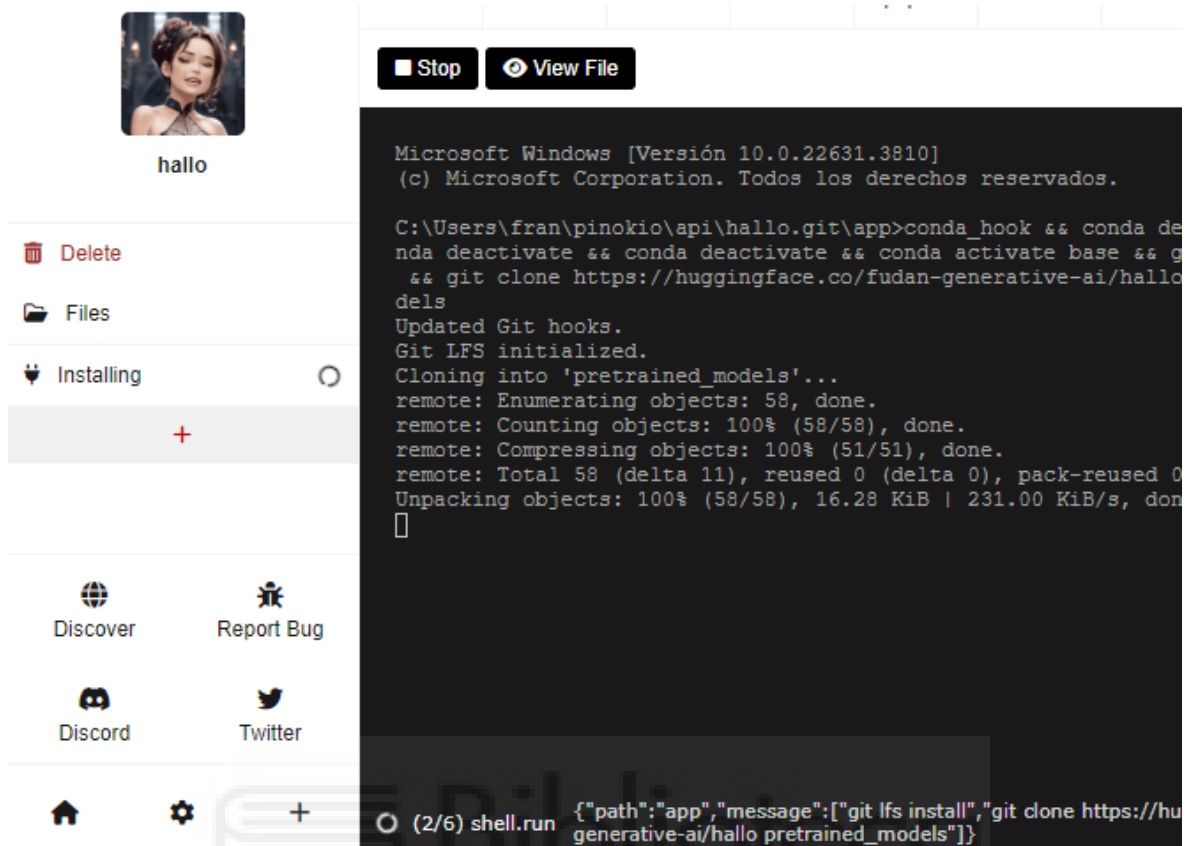


Figura AII.4: Proceso de instalación Hallo en Pinokio

Anexo III

Postman

AIII.1.- INSTALACIÓN

Para la instalación de Postman, se debe acceder a la web <https://www.postman.com/> y buscar el icono de windows debajo a la izquierda tal y como se muestra en la siguiente imagen. Dicho icono corresponde a la sección de instalación de Postman para Windows.

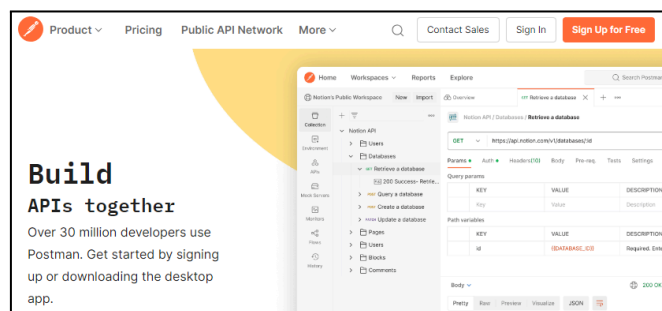


Figura AIII.1: Web de la aplicación Postman

Una vez pulsado, se accede a la sección de descarga donde se puede descargar la aplicación Postman para Windows 64 bits.

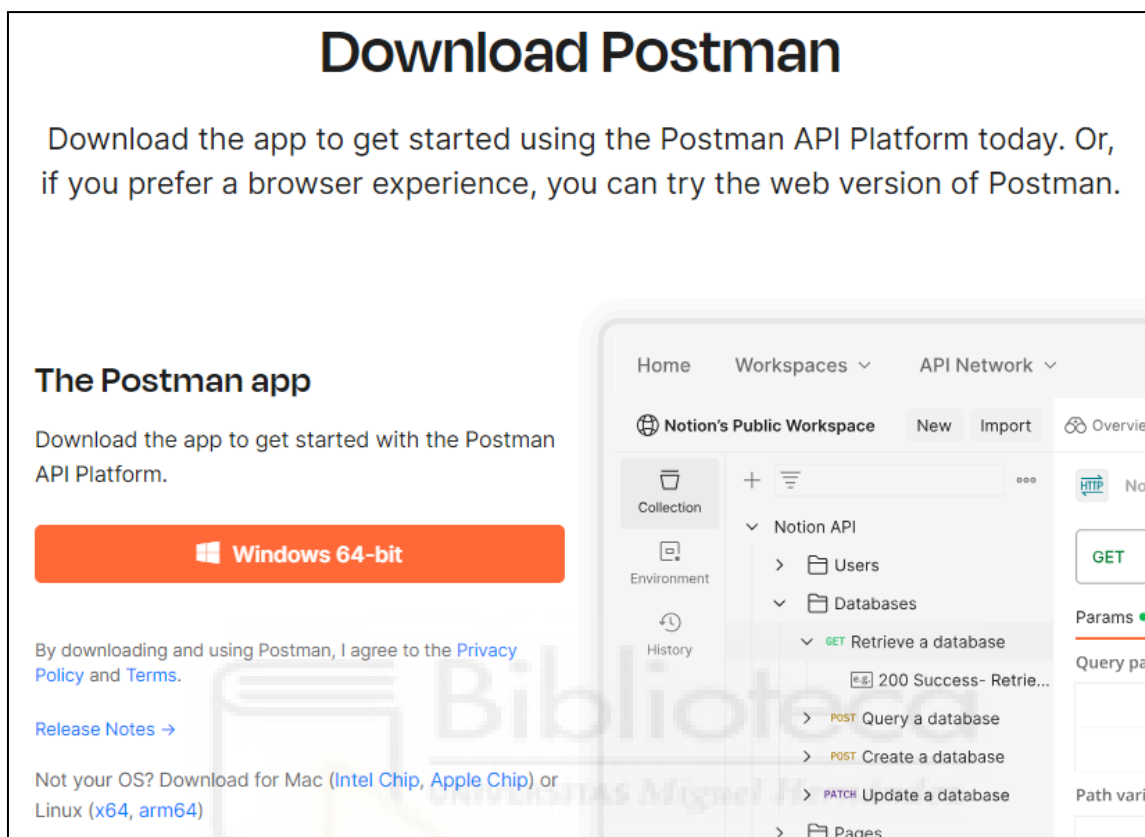


Figura AIII.2: Descarga de la aplicación Postman

Al igual que pinokio, no tiene gran misterio la instalación, la cual tarda en torno a 1 min una vez se ejecute el instalador descargado.

AIII.2.- CONFIGURACIÓN DE UN API Y USO

Una vez abierta la aplicación Postman. Aparecerá la pestaña My Workspace, donde se muestra un conjunto de carpetas que se podrán ir creando para tener las distintas Apis ordenadas.

En nuestro caso se ha creado la carpeta TFG con subcarpetas de cada microservicio en uso, entre ellas una de ejemplo donde se recogen los códigos de proyectos recogidos en base de datos. Para ello se realizará una llamada al microservicio ApiBD que es quién tiene las conexiones a la Base de Datos.

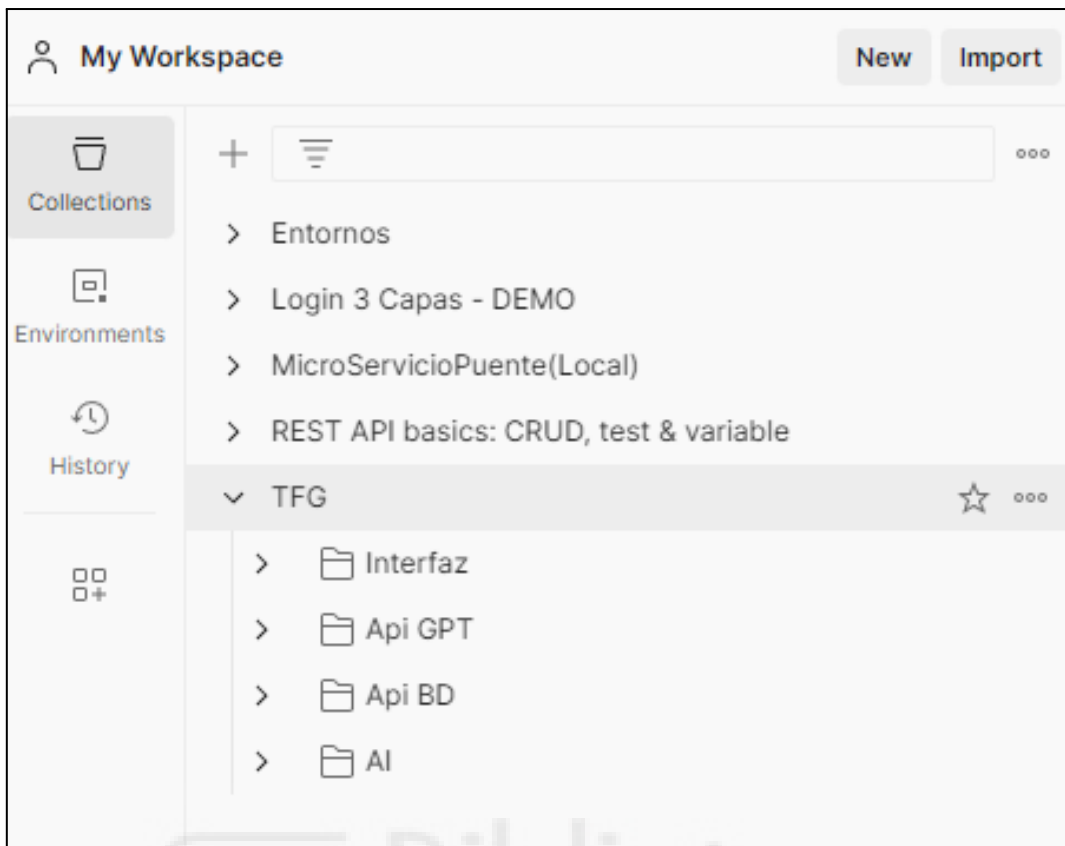


Figura AIII.3: Workspace de la aplicación Postman

Para ello, se pulsará en el botón New y se pulsará sobre la opción HTTP, ya que se solicita realizar una conexión por medio de un API Rest.

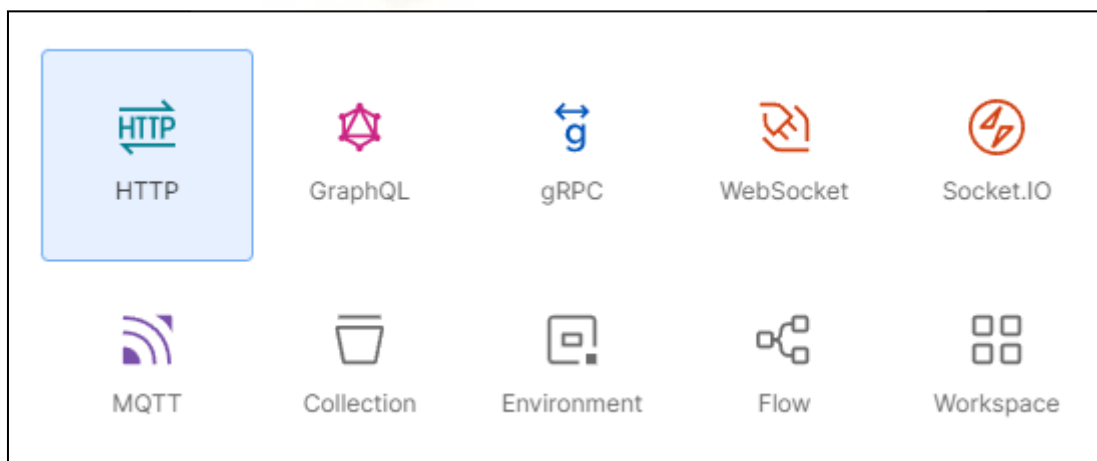


Figura AIII.4: Nueva API Http en Postman

A continuación, se selecciona el método con el que se configuró dicha Api en el microservicio, el cual es de tipo GET.

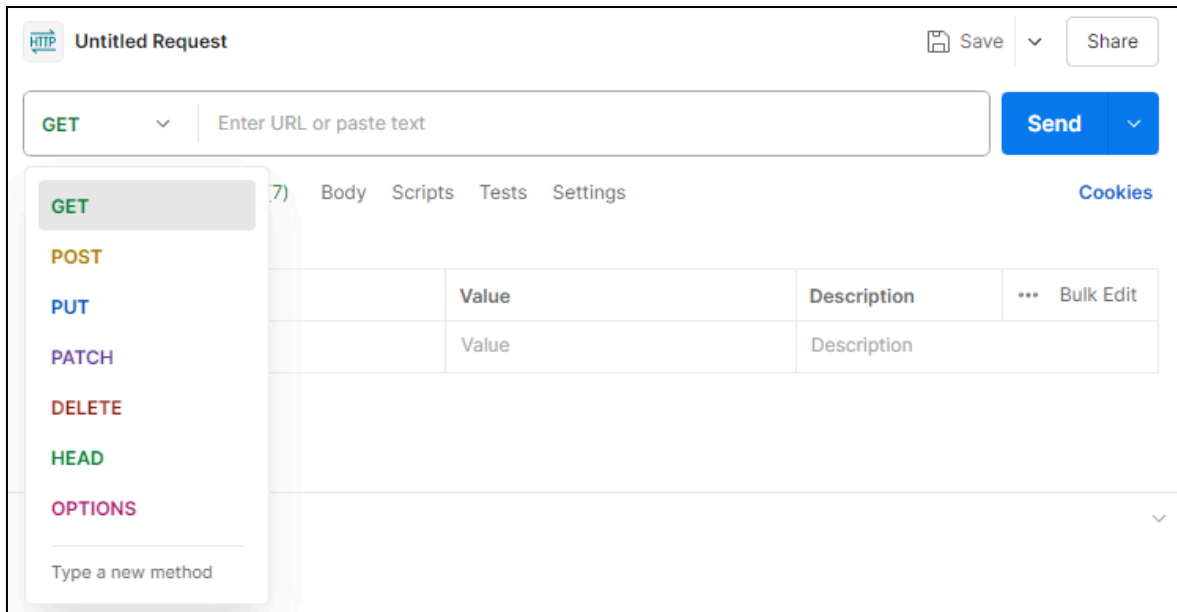


Figura AIII.5: Selección método Http en Postman

Además, indicaremos la URL correspondiente a dicha Api. En este caso no hará falta un cuerpo de mensaje ya que no se trata de una llamada de tipo POST.

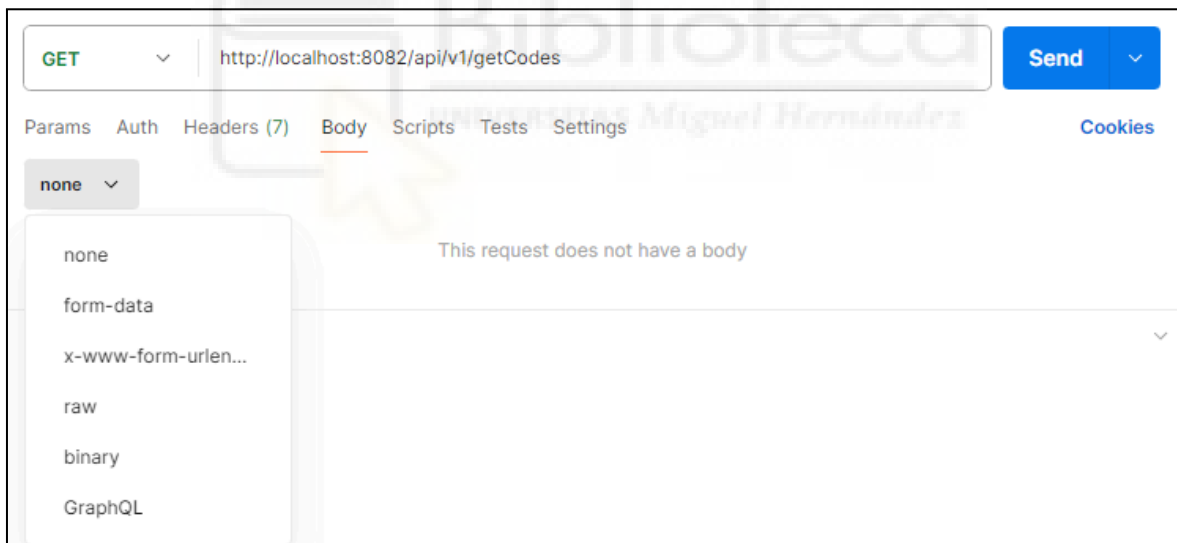


Figura AIII.6: Url y cuerpo mensaje Http en Postman

Una vez realizada dicha configuración, se lanzará una llamada y se comprobará la respuesta, la cual consiste en un JsonArray de objetos Json con la estructura clave:valor.

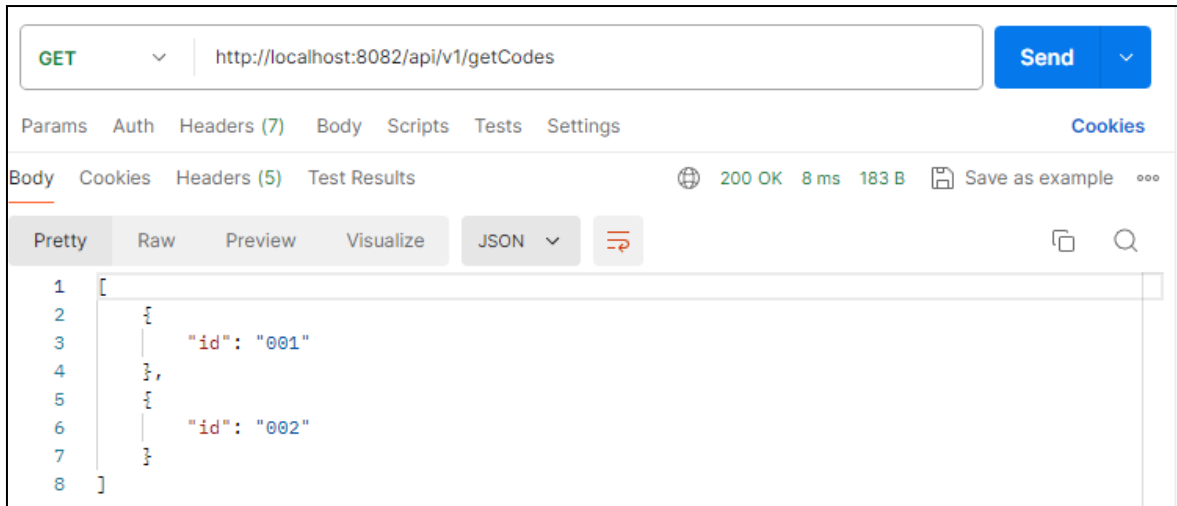


Figura AIII.7: Respuesta Api Http en Postman

