Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Mención en Computación

Creación de una IA optimizada para la asignatura de Fundamentos de Programación

---

Alumno: Pablo de Arriba Mendizábal

Tutor: Alejandro Ortega Arranz

Dedicado a la gente que me ha marcado durante estos años, tanto los que están como los que se fueron.

Agradecimientos

A mi tutor, Alejandro, por su enorme implicación en el trabajo, estando siempre dispuesto a ayudarme con todo y en cualquier momento.

A los profesores del grado, los cuales me han formado con los conocimientos necesarios para realizar este trabajo y comenzar mi carrera profesional.

A mis amigos, que siempre han estado y van a estar, aunque estén a muchos kilómetros.

Por último, a mi familia, lo más importante que tengo y los que más me tienen que aguantar.

Resumen

En los últimos años, los agentes conversacionales apoyados por inteligencia artificial (*chatbots*), como ChatGPT o Microsoft Copilot, han adquirido una gran relevancia, especialmente en el ámbito educativo, por su capacidad para interactuar de manera natural, resolver dudas de los alumnos y apoyar la generación de contenido por parte de los profesores. Este Trabajo de Fin de Grado presenta una aplicación web que integra un *chatbot* dirigido a la asignatura de Fundamentos de Programación, orientado tanto a estudiantes como a profesores.

Fundamentos de Programación es una asignatura del primer curso del Grado en Ingeniería Informática de la Universidad de Valladolid. Debido a que para muchos alumnos es la primera vez que se enfrentan a la programación, es una asignatura con un alto grado de fracaso académico. Además, muchos alumnos confunden que el programa funcione, con que esté bien estructurado desde el punto de vista de los requisitos que los profesores piden en la asignatura.

De esta manera, el sistema desarrollado permite plantear consultas, acceder a materiales, gestionar archivos y valorar respuestas de forma autónoma. El *chatbot* utiliza un modelo de lenguaje local combinado con un enfoque de Recuperación Aumentada por Generación (RAG). Esta técnica permite la contextualización de las respuestas en base a documentación proporcionada previamente por el profesor, pudiendo así mejorar las respuestas del *chatbot* hacia los objetivos de la asignatura.

La aplicación está integrada con Moodle, que se encarga de la autenticación y adapta la interfaz según el rol del usuario (estudiante o profesor), permitiendo así que el *chatbot* sea ofrecido como un recurso más de la asignatura junto con el resto de contenidos. Además, el sistema puede recopilar archivos directamente desde Moodle, enriqueciendo así el contexto que utiliza el *chatbot*.

Para garantizar el correcto funcionamiento del sistema, se han realizado pruebas para ajustar los parámetros que influyen en la recuperación de la información, identificando los valores óptimos para la calidad de las respuestas. También se han llevado a cabo evaluaciones con usuarios finales, tanto profesores como alumnos, a través de cuestionarios estandarizados y preguntas abiertas. Los resultados muestran una valoración positiva tanto en la facilidad de uso como en la utilidad de la herramienta para el aprendizaje y la enseñanza.

Este proyecto demuestra cómo la inteligencia artificial y las tecnologías de procesamiento de lenguaje natural pueden aplicarse en entornos educativos, adaptándose a las necesidades reales de profesores y estudiantes.

Abstract

In recent years, conversational agents powered by artificial intelligence (chatbots) such as ChatGPT or Microsoft Copilot have gained significant importance, especially in the field of education, due to their ability to interact naturally, answer students' questions, and support content creation for teachers. This Bachelor's Thesis presents a web application that integrates a chatbot specifically designed for the subject "Fundamentals of Programming", aimed at both students and instructors.

Fundamentals of Programming is a first-year course in the Computer Engineering degree of the University of Valladolid. For many students, it is their first contact with programming, which results in a high failure rate. Moreover, students typically confuse having a program that works with having one that is well structured according to the requirements set by the instructors.

The developed system allows users to submit queries, access course materials, manage files, and evaluate answers autonomously. The chatbot uses a local language model combined with a Retrieval-Augmented Generation (RAG) approach, which enables responses to be contextualized based on documentation previously provided by instructors, thus improving alignment with the objectives of the course.

The application is integrated with Moodle, which handles user authentication and adapts the interface depending on the user's role (student or instructor), making the chatbot available as an additional resource alongside the rest of the course content. Furthermore, the system can automatically retrieve files from Moodle, enriching the context available to the chatbot.

To ensure optimal system performance, tests were conducted to fine-tune the parameters influencing information retrieval, identifying the most suitable values for answer quality. Additionally, evaluations were carried out with end users—both instructors and students—using standardized questionnaires and open-ended questions. The results show positive feedback regarding both the usability and usefulness of the tool for teaching and learning.

This project demonstrates how artificial intelligence and natural language processing technologies can be applied in educational environments, adapting to the real needs of both instructors and students.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

En la actualidad, la inteligencia artificial (IA) se ha convertido en una herramienta muy presente en el día a día de muchas personas. En particular, los grandes modelos de lenguaje, o *Large Language Models* (LLM), han liderado este enorme crecimiento de los últimos años. En estos modelos se basan los agentes conversacionales o *chatbots*, herramientas diseñadas para simular conversaciones con usuarios a través de texto o voz. Su objetivo principal es responder preguntas, proporcionar asistencia y automatizar tareas, reduciendo la necesidad de intervención humana en ciertos procesos.

Esta tecnología ha supuesto una revolución en todos los ámbitos, especialmente desde el lanzamiento de uno de los *chatbots* de propósito general más populares: ChatGPT de OpenAI [20]. En la misma línea, existen infinidad de herramientas basadas en LLM, como pueden ser Copilot de Microsoft [17], Gemini de Google [9], DeepSeek-V3 de DeepSeek [6] o Llama de Meta AI [16].

Uno de los campos más influenciados por estos avances es el de la educación, donde tanto a alumnos como a profesores se les presenta un nuevo paradigma al que deben adaptarse. Los LLMs han demostrado una gran capacidad para generar contenido relevante y de utilidad en diversas disciplinas. Sin embargo, su uso en entornos educativos específicos requiere una contextualización que tenga en cuenta el temario, los materiales y la metodología de cada asignatura, para no perjudicar a los alumnos de cara a los objetivos de la misma.

Para abordar el problema de la contextualización de los *chatbots* en entornos educativos, han surgido diversas soluciones, cada una con sus propias ventajas e inconvenientes. El desempeño de cada una dependerá de la naturaleza del problema y de las necesidades del entorno en el que se implemente. En este contexto, las técnicas más utilizadas son ***Retrieval-Augmented Generation*** (RAG) y ***Fine-tuning***. Mediante estas técnicas se proporciona información adicional al modelo, de forma que las respuestas que recibe el usuario son más específicas y contextualizadas en relación a la información dada.

Más adelante (Sección 2.3), veremos una comparativa de ambos métodos, su aplicación en diferentes situaciones y se justificará la elección de uno de ellos para el desarrollo de este proyecto.

1.2. Motivación

Este Trabajo de Fin de Grado se centra en el desarrollo y optimización de una IA basada en un modelo LLM para la asignatura de Fundamentos de Programación del Grado en Ingeniería Informática de la Universidad de Valladolid.

La asignatura hace especial hincapié en las buenas prácticas de programación utilizando el lenguaje de programación Java y el paradigma de programación estructurada. Sin embargo, cuando se pregunta a estas aplicaciones (ChatGPT, Copilot, DeepSeek, etc.) sobre determinados ejercicios de programación, sus respuestas no siempre están contextualizadas al paradigma utilizado o no aplican dichas prácticas de programación. Por esta razón, es necesario dotar a estas aplicaciones de un contexto específico dependiente de los objetivos y contenidos de la asignatura para la que van a ser utilizadas.

De esta forma, se busca realizar o adaptar un sistema de inteligencia artificial que sea capaz de responder a las preguntas de los alumnos, indicarles los errores que tienen en sus programas (si los tienen) y plantearles alternativas correctas. No se busca que el sistema dé una solución a un problema (para esto, pueden seguir utilizando herramientas como ChatGPT o Copilot), sino que la solución esté contextualizada.

La mayor motivación de este trabajo reside en lograr un producto que pueda ser empleado en próximos años en esta asignatura, adaptándola a los nuevos avances y facilitando el aprendizaje por parte del alumno, así como la docencia por parte del profesor.

1.3. Objetivos

Como se ha mencionado anteriormente, este trabajo tiene como objetivo general desarrollar un agente conversacional basado en LLM para asistir a los estudiantes en la programación con Java estructurado, proporcionando respuestas contextualizadas y alineadas con los contenidos de la asignatura.

Adicionalmente, el desarrollo del proyecto abarca objetivos más específicos que llevan al objetivo principal:

- Obtener y estructurar materiales de la asignatura de Fundamentos de Programación, como apuntes o problemas resueltos, para utilizarlos como fuente de conocimiento.
- Implementar una arquitectura RAG, que permita al modelo recuperar información relevante antes de generar la respuesta.

- Estudiar los parámetros de configuración RAG y sus mejores valores para la obtención de respuestas contextualizadas a la documentación de la asignatura.
- Evaluar el rendimiento del modelo en términos de precisión, eficiencia y utilidad para la asignatura.
- Explorar futuras líneas de investigación y la viabilidad para implementarlo en otras asignaturas del grado.

1.4. Estructura de la memoria

La memoria se organiza en los siguientes capítulos, cada uno dedicado a una fase fundamental del proyecto:

Capítulo 2. Estado del arte: Presenta un análisis del conocimiento existente sobre *chatbots*, inteligencia artificial generativa, grandes modelos de lenguaje (LLM) y técnicas como RAG, además de un repaso de su uso en el ámbito educativo.

Capítulo 3. Planificación: Describe la metodología de trabajo utilizada, la gestión de riesgos, la planificación temporal y la estimación de recursos para el desarrollo del proyecto.

Capítulo 4. Análisis: Incluye el análisis de requisitos (funcionales, de información y no funcionales), los actores del sistema, los casos de uso y el modelo de dominio. También se incluye la realización en análisis de los casos de uso mediante diagramas de actividad.

Capítulo 5. Diseño: Aquí se presentan las decisiones de diseño y la arquitectura general del sistema (cliente-servidor), la persistencia de datos mediante bases de datos vectoriales y relacionales, y el diagrama de despliegue de la aplicación.

Capítulo 6. Implementación: Explica el proceso de desarrollo, detallando las fases y aspectos clave de la construcción de la aplicación, como las tecnologías más importantes o decisiones de código que se han tomado.

Capítulo 7. Pruebas: Recoge las pruebas realizadas para evaluar el sistema, incluyendo la evaluación de parámetros, la comparación entre el sistema con y sin RAG y las pruebas con usuarios finales a través de cuestionarios de usabilidad, carga de trabajo y experiencia de usuario.

Capítulo 8. Conclusiones: Resume los principales resultados obtenidos y plantea posibles líneas de trabajo futuras.

Capítulo 2

Estado del arte

La época actual se caracteriza por el gran crecimiento y adopción de la inteligencia artificial generativa y los grandes modelos de lenguaje (LLMs) en la sociedad actual. Estas tecnologías han ganado popularidad entre todos los grupos de edades, que las utilizan con frecuencia en su día a día, a menudo incluso de manera inconsciente. Por ejemplo, al hacer una consulta en Google actualmente, el primer resultado que se muestra es una explicación generada por IA sobre el tema consultado.

Debido a esto, los LLMs han acaparado gran parte de la atención en la literatura en los últimos años, principalmente por la necesidad de conocer a fondo esta tecnología y seguir avanzando para lograr productos y soluciones de mayor calidad. A continuación, se van a profundizar en los fundamentos teóricos en los que se basa esta tecnología y a repasar los últimos avances que se han propuesto en el ámbito de la IA generativa y los LLMs, así como nuevas técnicas que se emplean en la mejora y personalización de los modelos, centrándonos principalmente en el ámbito educativo.

2.1. IA generativa en la educación

La inteligencia artificial es una rama de la informática dedicada a desarrollar sistemas y programas con habilidades propias de los seres humanos, como aprender o planificar acciones, imitando sus capacidades [27]. Dentro de esta área, se encuentra la **inteligencia artificial generativa** (GenAI), que se caracteriza por su capacidad para producir contenido original como música, videos, texto, audio o imágenes. Este tipo de modelos aprende a partir de los patrones y estructuras presentes en los datos con los que ha sido entrenado, y a partir de ellos genera nueva información que tiene características similares.

Esta tecnología ha supuesto una enorme revolución en el campo de la educación, donde se pretende enfocar el presente trabajo, lo que ha provocado la aparición de numerosos estudios que profundizan en el impacto que ha tenido tanto en las aulas como en la investigación. En

J. Á. Ariza *et al.* [15], los autores llevaron a cabo una revisión de 146 estudios sobre el uso de GenAI en educación en ingeniería e informática. El estudio muestra que ChatGPT es la herramienta más utilizada y que la mayoría de propuestas de uso se centran en el aprendizaje y apoyo en la programación. Además, se observa diferencia entre el uso que dan los alumnos y los profesores, donde se observa un mayor uso en los primeros que en los segundos. Este estudio concluye que, si bien GenAI puede ser realmente útil en el proceso de enseñanza y aprendizaje, su uso debe estar controlado, asegurando que tanto alumnos como profesores tengan una buena formación para hacer un uso adecuado de estas herramientas.

Por otro lado, para llegar a una conclusión acerca de si el uso de GenAI es adecuado en la educación, debemos basarnos en los fundamentos teóricos de la educación. Autores como Yi Wu [29] mencionan el uso de esta tecnología dentro de algunas teorías educativas, como el constructivismo, la teoría sociocultural, la teoría de la carga cognitiva o la teoría del aprendizaje social, entre otras. A partir de esto, se afirma que herramientas como ChatGPT tienen el potencial de mejorar la educación al ofrecer retroalimentación inmediata, personalización del aprendizaje y apoyo continuo, haciendo que el estudiante sea más autónomo en su aprendizaje.

A pesar de que los beneficios sean numerosos, también hay que señalar los desafíos que estas herramientas implican. Entre ellos, se destaca la pérdida de habilidades como el pensamiento crítico, la reducción de la interacción social y la sobrecarga de información. Estos aspectos podrían afectar al aprendizaje si no se manejan adecuadamente [29]. Por ello, en lugar de prohibir el uso de estas herramientas, lo ideal sería integrarlas en la educación con criterio, de forma que se pueda aprovechar su potencial sin sacrificar las bases de la educación.

Un estudio de Study.com [26] encuestó a más de 100 profesores y 1.000 estudiantes para conocer el impacto de ChatGPT en la educación. El 82 % de los profesores universitarios conocen ChatGPT, y el 72 % está preocupado por su posible uso para hacer trampas, aunque un 66 % apoya que los alumnos tengan acceso a la herramienta. Entre los estudiantes, más del 90 % han usado ChatGPT para hacer deberes, exámenes en casa o redactar trabajos, pero el 72 % de los universitarios cree que debería prohibirse en su campus. El estudio muestra que ChatGPT es muy popular y puede ser útil, pero profesores y estudiantes coinciden en que hace falta controlar su uso para evitar problemas como el plagio.

2.2. LLM

Los *Large Language Models* (LLM), o grandes modelos de lenguaje, son herramientas de inteligencia artificial diseñadas para generar texto con apariencia humana. Están basados en redes neuronales profundas, específicamente en arquitecturas tipo *transformer*, que permiten procesar grandes volúmenes de datos en paralelo y de forma muy eficiente. A diferencia de los modelos de lenguaje tradicionales que utilizan técnicas estadísticas simples para predecir la siguiente palabra en una secuencia, los LLM utilizan capas de atención que capturan relaciones complejas entre palabras y frases a lo largo de grandes contextos de texto [11].

Estos modelos suelen ser entrenados con enormes cantidades de texto provenientes de internet, libros, artículos y otras fuentes, con el objetivo de aprender patrones de lenguaje,

estructuras gramaticales y asociaciones semánticas. Gracias a este entrenamiento, son capaces de realizar tareas como redactar textos, responder preguntas, traducir idiomas, generar código o incluso resumir contenido técnico. La clave de su rendimiento está en su capacidad para generalizar el conocimiento aprendido y generar respuestas relevantes a partir de una entrada de texto, aunque también tienen limitaciones, como errores contextuales, sesgos debidos al entrenamiento o información errónea por falta de conocimiento.

2.3. Generación Aumentada por Recuperación (RAG)

A pesar de los avances que han surgido en los últimos años, los LLM presentan todavía algunas limitaciones, especialmente cuando se busca obtener información precisa, actualizada y adaptada a contextos específicos. Entre estos retos destacan la aparición de respuestas incorrectas o inventadas (*alucinaciones*), la generación de información genérica o desactualizada y la dificultad para verificar la procedencia y la fiabilidad de las respuestas ofrecidas por el modelo [3, 12, 2, 25].

Para solucionar esto, en los últimos años se ha popularizado la **Generación Aumentada por Recuperación** (*Retrieval-Augmented Generation*, RAG), una técnica que mejora las capacidades de los LLM al permitirles acceder a una base de conocimiento externa, normalmente de un contexto determinado (por ejemplo, una asignatura). En lugar de confiar únicamente en la información aprendida durante el entrenamiento del LLM, RAG añade una fase previa de recuperación de información relevante, de modo que el modelo puede generar respuestas que se apoyan en información relevante y ajustada al contexto de la consulta [3].

En la Figura 2.1 se puede observar el aumento de la popularidad del término *'Retrieval-Augmented Generation'* en los últimos años. Un valor de 100 indica la popularidad máxima de un término, mientras que 50 y 0 indican que un término es la mitad de popular en relación con el valor máximo o que no había suficientes datos del término, respectivamente.

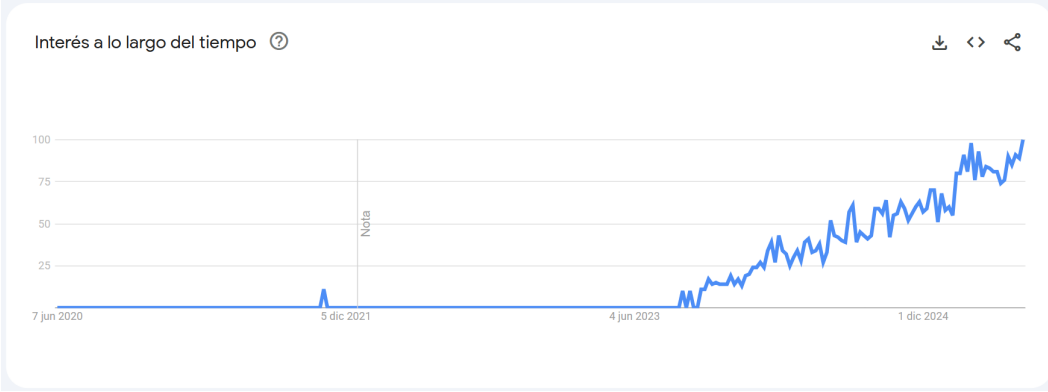

Figura 2.1: Popularidad del término *'Retrieval-Augmented Generation'* en el tiempo (Fuente: Google Trends [28])

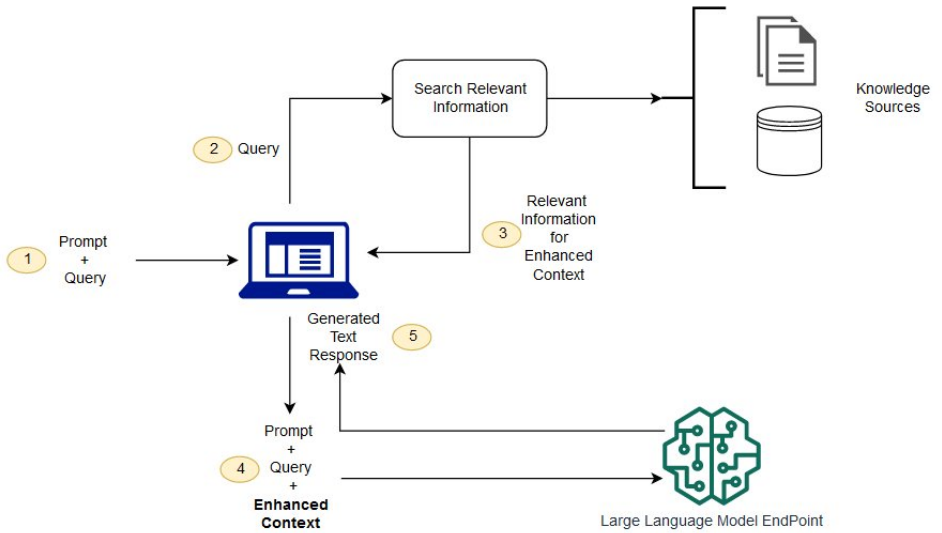

Figura 2.2: Funcionamiento del flujo RAG (Fuente: AWS [3])

El proceso de RAG suele dividirse en dos etapas: la *recuperación* y la *generación*. En la primera etapa, el sistema utiliza la pregunta del usuario para buscar y recuperar fragmentos de información relevantes desde una base de datos vectorial (como ChromaDB, FAISS, etc.). Esta recuperación utiliza la búsqueda semántica, usando modelos de *embeddings* (representaciones numéricas de textos en un espacio vectorial, donde textos con significados similares quedan cerca entre sí) para capturar el significado del texto de forma más precisa que con la coincidencia de palabras clave. En la segunda etapa, los fragmentos recuperados se combinan con la pregunta del usuario, creando lo que se conoce como *prompt*. Después, se introduce en el LLM, que genera una respuesta contextualizada y basada en la información que ha recuperado [2, 3]. En la Figura 2.2 se muestra el funcionamiento completo del flujo RAG, desde la consulta del usuario hasta la respuesta generada.

Esta arquitectura aporta varios beneficios significativos:

- **Actualización dinámica del conocimiento:** Permite utilizar información reciente, ya que la base de datos puede actualizarse fácilmente, sin necesidad de reentrenar el modelo.
- **Reducción de las alucinaciones:** Al basar las respuestas en fragmentos de la base de conocimiento, se minimiza la tendencia del modelo a generar información incorrecta, inventada o engañosa.
- **Confianza:** La posibilidad de citar fuentes o referencias en las respuestas generadas aumenta la confianza del usuario en el sistema y le permite verificar la procedencia de la información.

- **Personalización:** Los desarrolladores pueden definir qué fuentes se consultan, restringir el acceso a ciertos documentos y adaptar el comportamiento del sistema a las necesidades del cliente, dentro de ciertos límites.
- **Escalabilidad y eficiencia:** RAG resulta mucho más eficiente y rentable que el *fine-tuning*, ya que no requiere modificar los parámetros del modelo base (reentrenar el modelo). Esto hace que pueda implementarse en situaciones de pocos recursos computacionales. [3, 12].

En la literatura se encuentran diversas investigaciones que comparan el enfoque RAG con otros métodos de adaptación de los LLM, como el *fine-tuning* o la ingeniería avanzada de *prompts* (*prompt engineering*). El *fine-tuning* implica reentrenar el modelo con datos adicionales y etiquetados, lo cual es costoso y puede conllevar riesgos como la sobre-especialización, mientras que RAG tiene mayor flexibilidad sin afectar a la generalización del modelo. Estudios recientes indican que RAG es adecuado en contextos donde la información es específica de un contexto o requiere una actualización frecuente, como ocurre en la educación [12, 25].

En el contexto educativo, los sistemas RAG se están utilizando cada vez más para personalizar el aprendizaje y facilitar el acceso a los recursos de las asignaturas. Por ejemplo, Alario-Hoyos *et al.* [2] implementaron un *chatbot* académico basado en RAG para apoyar a estudiantes de una asignatura de programación, utilizando como base de conocimiento materiales seleccionados por el profesorado, ejercicios resueltos y preguntas frecuentes. Se realizaron más de mil interacciones reales, demostrando que la mayoría de los estudiantes valoraron positivamente la precisión, claridad y utilidad de las respuestas. También se destaca la importancia de poder actualizar fácilmente el corpus de documentos, de forma que el *chatbot* evolucione a medida que se avanza en la asignatura.

Algunas de las grandes tecnológicas han apostado por el enfoque RAG en sus servicios. Por ejemplo, Amazon Web Services destaca la eficiencia de RAG para crear *chatbots* empresariales conectados a fuentes de información, como redes sociales, sitios de noticias o bases de datos científicas, evitando los costes y riesgos del reentrenamiento [3].

Sin embargo, es importante señalar algunas limitaciones de RAG identificadas en la literatura. Aunque mejora significativamente la precisión de las respuestas, su efectividad depende en gran medida de la calidad de los documentos empleados, así como de la capacidad del sistema para encontrar los fragmentos más relevantes para cada pregunta. [25]. Estas limitaciones hacen que su implementación no sea trivial y requiera de un estudio previo de la documentación aportada para la contextualización, intentando ajustar de la mejor manera posible tanto la selección como la estructura de los materiales.

Además, es fundamental ajustar correctamente los parámetros del sistema de recuperación, como el tamaño de los fragmentos (*chunks*), el umbral de similitud o el número de documentos recuperados en cada consulta. Este ajuste suele requerir la realización de numerosas pruebas y evaluaciones con preguntas reales, analizando el impacto de cada parámetro en la precisión y relevancia de las respuestas. Con estas pruebas y ajustes se puede contribuir a que el sistema RAG ofrezca resultados adecuados para los objetivos de este trabajo.

Capítulo 3

Planificación

3.1. Metodología de trabajo

Durante la planificación inicial del proyecto, se valoraron diversas metodologías de trabajo que podrían emplearse. Algunas de las opciones fueron:

- **SCRUM**, una metodología ágil que se basa en *sprints*, con entregas periódicas funcionales y reuniones diarias [24]. Sin embargo, esta metodología suele componerse por grupos multidisciplinares compuestos por 3+ personas, requiriendo reuniones periódicas con el cliente.
- **SDRM (System Development Research Method)**, una metodología centrada en proyectos de desarrollo de sistemas de información. Sin embargo, esta metodología está orientada principalmente a la investigación [14].
- **ASAP (Agile Student Academic Projects)** [18], una metodología desarrollada en la UVa que adapta prácticas ágiles habituales en el sector profesional para alcanzar los objetivos de aprendizaje propios del TFG. Sin embargo, el trabajo por *sprints* no encaja con mi temporalidad y disponibilidad en este proyecto, debido a otros compromisos de trabajo.

Tras debatirlo con el tutor, decidimos que ninguna de ellas se ajustaba a nuestro plan de trabajo. El desarrollo del proyecto se ha planteado en base a reuniones periódicas entre el estudiante y el tutor, en las cuales se va a evaluar el estado del trabajo, se resolverán dudas y se darán indicaciones sobre cómo continuar. No se han planteado iteraciones, sino que el trabajo será lineal, intentando completar cada fase antes de continuar a la siguiente. Además, se mantendrán reuniones puntuales con los profesores de la asignatura para la recolección de requisitos, como por ejemplo recopilar las preguntas que suelen hacer los estudiantes a ChatGPT o reunir los documentos necesarios para contextualizar el *chatbot*.

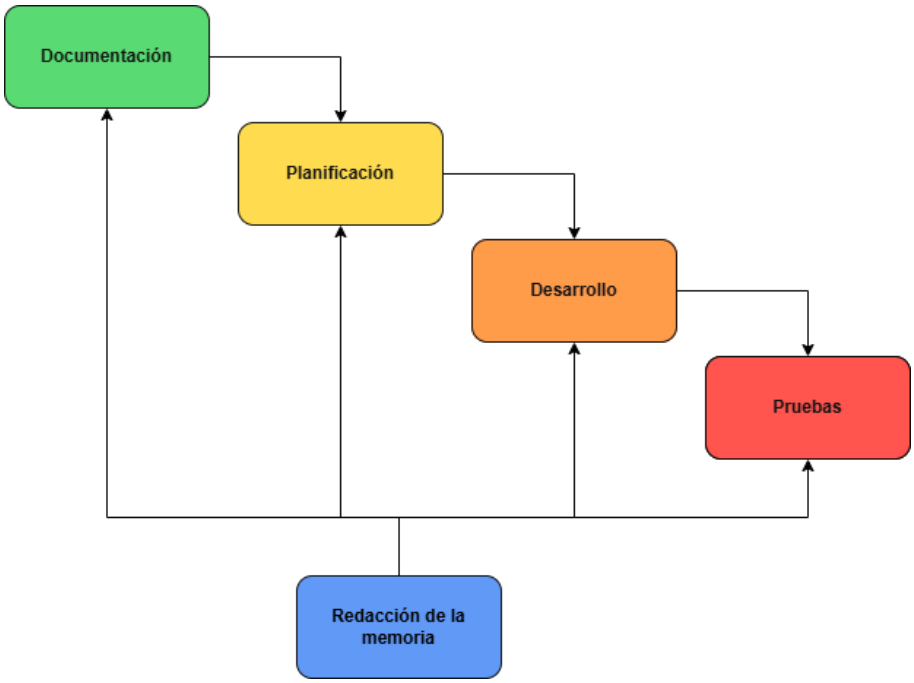

Figura 3.1: Diagrama metodología en cascada

La idea es mantenerse en contacto directo con el usuario final, aunque no de forma sistemática o periódica.

Por esta razón, se optó por utilizar una metodología en cascada [23], que se caracteriza por ser secuencial y estructurada. El proyecto se divide en etapas: documentación, planificación, desarrollo, pruebas y redacción de la memoria, y cada etapa debe completarse antes de pasar a la siguiente. En la Figura 3.1 se muestra el flujo de trabajo que sigue esta metodología.

En nuestro caso, se ha variado ligeramente este enfoque, ya que el proceso de redacción de la memoria se realizará en paralelo al resto de procesos. A pesar de ello, esta metodología se ajusta bastante bien a nuestra forma de trabajo.

En la siguiente sección, se detalla la duración y las tareas correspondientes a cada una de estas fases de la metodología en cascada.

3.2. Plan de trabajo

El desarrollo del Trabajo de Fin de Grado se ha organizado para ajustarse a las 300 horas recomendadas por la universidad. El proyecto comenzó el 27 de febrero de 2025 y está previsto que finalice el 9 de junio del mismo año.

Id	Nombre de tarea	Duración	Comienzo	Fin
1	Documentación inicial	10 días	jue 27/02/25	mié 12/03/25
2	Planificación	5 días	mar 11/03/25	lun 17/03/25
3	Diseño del sistema	21 días	lun 17/03/25	lun 14/04/25
4	Implementación chatbot	21 días	lun 14/04/25	lun 12/05/25
5	Interfaz gráfica	6 días	lun 12/05/25	lun 19/05/25
6	Pruebas y validación	12 días	lun 19/05/25	mar 03/06/25
7	Redacción del documento	71 días	mié 12/03/25	mié 18/06/25

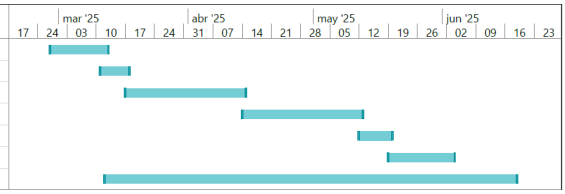

Figura 3.2: Diagrama de Gantt de las tareas del proyecto

Sin embargo, durante el desarrollo pueden aparecer nuevos requisitos o ideas para mejorar la aplicación que, por cuestión de tiempo, no sea posible implementar dentro del plazo del TFG. Todas esas mejoras o características adicionales que no lleguen a estar listas para la entrega se propondrán como posibles líneas de trabajo futuro. Aún así, el cómputo total de horas ha excedido las 300 planificadas inicialmente, ya que a medida que ha evolucionado el trabajo, han ido surgiendo ideas que se han considerado interesantes para implementar.

La Figura 3.2 muestra el diagrama de Gantt que incluye las principales tareas que se llevan a cabo en este proyecto, desglosando la fase de desarrollo en las distintas tareas que la componen.

El desarrollo del proyecto se estructuró en las siguientes etapas:

1. **Documentación (40h):** Revisión bibliográfica y estudio de las tecnologías que se van a utilizar, así como recopilación de materiales relevantes para la asignatura.
2. **Planificación (20h):** Organización de las tareas, identificación de riesgos y elaboración del presupuesto y la hoja de ruta del proyecto.
3. **Desarrollo (120h):** En esta etapa se incluye el análisis, el diseño y la implementación del sistema. Se definen los requisitos, se diseña la arquitectura y la base de datos, y se desarrolla tanto el *backend* como la interfaz gráfica.
4. **Pruebas y validación (40h):** Realización de pruebas funcionales y de usabilidad, corrección de errores y ajustes necesarios para garantizar el correcto funcionamiento de la aplicación.
5. **Redacción de la memoria (80h):** Documentación técnica y elaboración de la memoria del TFG. Esta fase se ha realizado en paralelo al resto de etapas, reflejando los avances y decisiones tomadas durante el desarrollo del proyecto.

3.3. Riesgos

La identificación de riesgos es esencial en la planificación de cualquier proyecto de desarrollo, especialmente cuando se trabaja con varias tecnologías distintas, como en este caso. Los riesgos que pueden suceder son innumerables, por lo que anticiparse a ellos permite actuar de forma más eficaz y reduce la probabilidad de sufrirlos.

Además, analizar los riesgos facilita la toma de decisiones. Por ejemplo, si se detecta que el rendimiento de un modelo no es bueno, se puede solucionar más rápidamente si ya hay modelos alternativos propuestos. Esto hace que se aproveche el tiempo de forma más efectiva ante posibles errores que puedan surgir.

Los riesgos identificados para este proyecto se muestran en las Tablas 3.1 a 3.8:

Riesgo 1	
Riesgo	Dificultades en la integración entre tecnologías
Descripción	Posibles problemas con la integración entre las tecnologías que se emplean en el desarrollo.
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Realizar pruebas iniciales para comprobar que no hay problemas de integración.
Plan de contingencia	Explorar soluciones alternativas para la interfaz si la integración presenta problemas, como Streamlit.

Tabla 3.1: Riesgo 1: Dificultades en la integración entre tecnologías

Riesgo 2	
Riesgo	Problemas con rendimiento del LLM utilizado
Descripción	El modelo puede no responder adecuadamente en términos de velocidad o precisión.
Probabilidad	Media
Impacto	Alto
Plan de mitigación	Seguimiento del modelo durante el desarrollo y realización de pruebas para observar posibles caídas de rendimiento.
Plan de contingencia	Evaluar modelos alternativos que ofrezcan mejor rendimiento.

Tabla 3.2: Riesgo 2: Problemas con rendimiento del LLM utilizado

Riesgo 3	
Riesgo	Errores en la recuperación de contexto con RAG
Descripción	La técnica de recuperación de contexto podría fallar en la precisión o relevancia de la información recuperada.
Probabilidad	Media
Impacto	Alto
Plan de mitigación	Realizar evaluaciones periódicas y ajustar umbral de recuperación.
Plan de contingencia	Cambiar el modelo de *embedding* o la fuente de datos empleada.

Tabla 3.3: Riesgo 3: Errores en la recuperación de contexto con RAG

Riesgo 4	
Riesgo	Problemas de conexión o configuración con las bases de datos
Descripción	Posibles dificultades al conectar y configurar las bases de datos correctamente.
Probabilidad	Baja
Impacto	Medio
Plan de mitigación	Comprobar la conexión frecuentemente.
Plan de contingencia	Considerar bases de datos alternativas o utilizar versiones en la nube que simplifiquen el proceso.

Tabla 3.4: Riesgo 4: Problemas de conexión o configuración con las bases de datos

Riesgo 5	
Riesgo	Pérdida de datos del usuario o chats
Descripción	Riesgo de pérdida accidental de información.
Probabilidad	Baja
Impacto	Medio. No se almacena información sensible sobre el usuario.
Plan de mitigación	Implementar *backups* periódicos.
Plan de contingencia	Establecer un sistema de recuperación de datos de forma inmediata a partir de copias de seguridad ya creadas.

Tabla 3.5: Riesgo 5: Pérdida de datos del usuario o chats

Riesgo 6	
Nombre del riesgo	Problemas en la integración del *chatbot* en Moodle
Descripción	Posibles incompatibilidades o problemas de configuración al integrar el *chatbot* en Moodle.
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Realizar pruebas durante todo el desarrollo y el despliegue para comprobar su correcto funcionamiento.
Plan de contingencia	Utilizar métodos alternativos de acceso al *chatbot* sin hacer uso de Moodle.

Tabla 3.6: Riesgo 6: Problemas en la integración del *chatbot* en Moodle

Riesgo 7	
Nombre del riesgo	Problemas de autenticación o permisos al usar la API de Moodle

Descripción	Posibles dificultades para gestionar los permisos de acceso mediante la API de Moodle debido a actualizaciones o cambios en su funcionamiento.
Probabilidad	Media
Impacto	Bajo
Plan de mitigación	Hacer comprobaciones periódicas del acceso a los documentos mediante la API.
Plan de contingencia	Implementar formas alternativas para acceder a los documentos, como cargas manuales desde la interfaz del profesor.

Tabla 3.7: Riesgo 7: Problemas de autenticación o permisos al usar la API de Moodle

Riesgo 8	
Nombre del riesgo	Conocimientos insuficientes sobre las tecnologías utilizadas
Descripción	Falta de formación que puede provocar retrasos y fallos en el funcionamiento del sistema.
Probabilidad	Media
Impacto	Medio
Plan de mitigación	Realizar una buena investigación y preparación inicial sobre las herramientas y tecnologías necesarias.
Plan de contingencia	Ante posibles dudas o bloqueos, buscar información o cursos sobre la herramienta o acudir al tutor.

Tabla 3.8: Riesgo 8: Conocimientos insuficientes sobre las tecnologías utilizadas

Riesgo 9	
Nombre del riesgo	Identificación de gran número de requisitos funcionales de la aplicación
Descripción	Durante la fase de análisis pueden surgir numerosos requisitos funcionales, lo que podría dificultar el desarrollo completo de la aplicación en el tiempo disponible.
Probabilidad	Alta
Impacto	Medio
Plan de mitigación	Realizar una evaluación de los requisitos funcionales desde el inicio, junto con el tutor, identificando aquellos que sean imprescindibles para el funcionamiento básico.
Plan de contingencia	Si no es posible implementar todos los requisitos, centrarse únicamente en los prioritarios y dejar el resto como posibles mejoras futuras.

Tabla 3.9: Riesgo 9: Identificación de gran número de requisitos funcionales de la aplicación

Riesgo 10	
Nombre del riesgo	Fallo de los equipos
Descripción	El fallo o pérdida de acceso a los equipos de desarrollo podría provocar la pérdida de avances importantes en el proyecto.
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	Realizar copias de seguridad periódicas del proyecto, utilizando herramientas de control de versiones como Git.
Plan de contingencia	En caso de fallo de los equipos o pérdida de datos, restaurar el proyecto desde las copias de seguridad o el repositorio remoto.

Tabla 3.10: Riesgo 10: Fallo de los equipos

Riesgo 11	
Nombre del riesgo	Falta de documentación de las tecnologías utilizadas o problemas en su desarrollo
Descripción	Puede darse el caso de que alguna de las tecnologías empleadas (librerías, *frameworks*, etc.) no disponga de suficiente documentación o genere problemas inesperados durante el desarrollo, lo que puede dificultar la integración o la resolución de problemas.
Probabilidad	Media
Impacto	Medio
Plan de mitigación	Analizar y elegir tecnologías bien documentadas, si es posible. Consultar foros, documentación oficial y ejemplos de código antes de decidir su uso.
Plan de contingencia	Si la falta de documentación o los problemas técnicos impiden avanzar, valorar el cambio a otras tecnologías mejor documentadas.

Tabla 3.11: Riesgo 11: Falta de documentación de las tecnologías utilizadas o problemas en su desarrollo

3.4. Presupuesto

Material

Los materiales empleados en este proyecto no han sido adquiridos específicamente para el mismo. Por ello, para calcular el coste estimado del material utilizado, es necesario tener en cuenta la amortización del mismo.

El equipo utilizado es un ordenador portátil HP Pavilion con procesador Ryzen 7, 8GB de RAM y tarjeta gráfica GTX 1650Ti. Su precio original es de 999€ y ha sido utilizado durante 5 años de carrera. La vida útil de un portátil es de unos 5 años y la duración de este proyecto es de en torno a 4 meses, por lo tanto, el gasto de amortización incurrido a lo largo del desarrollo del proyecto es de 66.67€.

$$Coste\,equipo = \frac{999}{5} \times \frac{4}{12} = 66{,}67\,€$$

Producto	**Observaciones**	**Coste (EUR)**
Ordenador portátil Ryzen 7 8GB RAM	Uso principal para desarrollo y pruebas. Amortización por uso.	66,67€
Ollama	*Framework* para ejecutar modelos LLM de forma local, sin coste	0,00
Gradio	Librería para crear interfaces web rápidas, de código abierto	0,00
Flask	*Framework* para gestión de la API, gratuito	0,00
LangChain	Utilizado para facilitar la integración con LLMs y RAG	0,00
PostgreSQL	Sistema gestor de base de datos utilizado para almacenar mensajes, chats y documentos	0,00
Moodle	Plataforma educativa donde se integrará el *chatbot* mediante LTI. Proporcionado por el grupo de investigación GSIC-EMIC.	0,00
Microsoft Project	Herramienta utilizada para la planificación temporal del proyecto, licencia gratuita	0,00
Visual Studio Code	Editor de código principal utilizado para el desarrollo	0,00
Astah Professional	Software para la realización del análisis y diseño, licencia proporcionada por la UVa	0,00
TOTAL		**66,67€**

Tabla 3.12: Estimación de material necesario y coste asociado

De cara a un futuro despliegue real de la aplicación, será necesario tener en cuenta otros costes, como el alojamiento en un servidor y la adquisición de *hardware* específico, por ejemplo, una tarjeta gráfica como la NVIDIA T1000[1]. La elección del servidor y del hardware dependerá del número de usuarios previstos y de los requisitos de rendimiento, por lo que estos aspectos deberán evaluarse más adelante, dependiendo de la demanda.

[1]https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/productspage/quadro/quadro-desktop/proviz-print-nvidia-T1000-datasheet-us-nvidia-1670054-r4-web.pdf

Personal

En cuanto al personal, únicamente será necesario un desarrollador *fullstack* junior (como es mi caso) para la realización del proyecto. El total de horas de trabajo será 300, lo correspondiente a un proyecto de 12 ECTS. El sueldo medio de un desarrollador *fullstack* junior en España es de 12,98€/hora brutos [8], lo que hace un coste total de 3.894€.

Rol	**Observaciones**	**Coste (EUR)**
Desarrollador *Fullstack* Junior	Encargado de todo el desarrollo *backend* (Python) y *frontend* (Gradio). Integración de LLM y RAG. Gestión del proyecto.	3.894,00€
TOTAL		3.894,00€

Tabla 3.13: Estimación de personal necesario y coste asociado

3.4.1. Espacio de trabajo

El proyecto se desarrollará principalmente en dos lugares: desde casa y desde la biblioteca. El coste estimado de luz y conexión a internet en casa se estima en unos 30€ mensuales, considerando únicamente los costes asociados al proyecto. Por tanto, como el proyecto tiene una duración de unos 3 meses y medio, el coste total estimado del espacio de trabajo asciende a 105€.

3.4.2. Presupuesto final

El presupuesto final, incluyendo materiales, personal y espacio de trabajo, asciende a **4.065,67€** (Tabla 3.14).

Concepto	**Observaciones**	**Coste (EUR)**
Material	Ordenador portátil y herramientas utilizadas para el desarrollo y pruebas.	66,67€
Personal	Trabajo de desarrollo, análisis, diseño, implementación y pruebas del sistema.	3.894,00€
Espacio de trabajo	Acceso a instalaciones, electricidad y recursos físicos necesarios durante el desarrollo.	105,00€
TOTAL		**4.065,67€**

Tabla 3.14: Resumen de costes del proyecto

3.5. Tecnologías y software utilizado

Para la gestión y desarrollo de este proyecto se ha hecho uso de diversas tecnologías y herramientas de software, cada una con un propósito concreto. A continuación, se resumen las principales herramientas y su función dentro del proyecto:

- **Python:** Lenguaje principal para implementar toda la lógica del *chatbot* y del servidor LTI, por su versatilidad, sencillez y amplia comunidad. Permite integrar múltiples librerías de forma sencilla.
- **Ollama:** Gestor local de modelos de lenguaje (LLM). Permite la descarga y despliegue local de modelos como Llama y muestra respuestas en *streaming*, manteniendo los datos en local y sin depender de la nube [30].
- **LangChain:** *Framework* de código abierto para crear aplicaciones basadas en LLMs, facilitando la personalización de *prompts*, el flujo RAG y la integración con bases vectoriales y módulos de *embeddings* [4].
- **HuggingFace:** Plataforma y ecosistema de librerías especializadas en procesamiento de lenguaje natural. En este proyecto se utiliza el modelo de *embeddings* `all-MiniLM-L6-v2`, proporcionado por HuggingFace, para convertir textos en vectores semánticos que luego se almacenan y consultan en ChromaDB.
- **LTI (Learning Tools Interoperability):** Estándar que permite la integración del *chatbot* en plataformas educativas como Moodle, gestionando la autenticación de usuarios y la incrustación de la interfaz mediante `iframe` [19].
- **PostgreSQL:** Sistema de gestión de base de datos relacional empleado para almacenar documentos, mensajes, usuarios, metadatos y, mediante extensiones como `pgvector`, almacenar *embeddings*.
- **Flask:** *Framework* web de Python utilizado para implementar la API LTI que conecta el sistema con Moodle y redirige a la interfaz según el rol del usuario.
- **Gradio:** *Framework* que facilita la creación de interfaces de *chatbot* y paneles de gestión de documentos, con componentes interactivos y subida de archivos.
- **Astah Professional:** Herramienta de modelado UML empleada para el diseño de diagramas de casos de uso, de actividades, de despliegue y de dominio que documentan la arquitectura y el comportamiento del sistema.
- **Visual Studio Code:** Editor de código fuente utilizado para el desarrollo y la organización del proyecto, por su compatibilidad con Python y su integración con múltiples herramientas.
- **LaTeX:** Herramienta empleada para la redacción de la memoria, por su potencia en la gestión de referencias, bibliografía y organización de secciones y elementos gráficos.
- **Microsoft Project:** Herramienta utilizada para la planificación temporal del proyecto y la elaboración del diagrama de Gantt con las principales tareas a realizar.

Capítulo 4

Análisis

Este capítulo se centra en la fase de análisis correspondiente a cualquier proyecto de desarrollo. En él se hará un análisis del funcionamiento deseado de nuestra aplicación, incluyendo historias de usuario que justifican su utilidad, requisitos del cliente, casos de uso, modelo de dominio y diagramas de secuencia.

4.1. Análisis de requisitos

En este apartado se definen los requisitos que debe cumplir nuestro sistema, los cuales se determinan a partir de las necesidades del cliente y el criterio del desarrollador. Como clientes actuarán los profesores de la asignatura de Fundamentos de Programación, que definirán los requisitos necesarios, así como los alumnos de dicha asignatura, de los cuales se ha recogido información sobre el uso habitual que hacen de herramientas similares a este *chatbot* en el curso 2024/2025 (ChatGPT, Copilot, etc.).

A continuación, se proporciona una descripción detallada de las funcionalidades, restricciones, datos y normas que debe cumplir la aplicación para satisfacer las necesidades de los usuarios y alcanzar los objetivos del proyecto.

4.1.1. Requisitos funcionales

Los requisitos funcionales definen lo que el sistema debe hacer en cuanto a comportamiento. Especifican las funciones, acciones o servicios que el sistema ofrece a sus usuarios y cómo debe responder ante interacciones.

En este caso, los requisitos funcionales se han derivado de una primera interacción con los profesores de la asignatura, de reuniones con el tutor y de decisiones propias del estudiante,

con el objetivo de que la aplicación sea lo más efectiva posible y favorezca su adopción tanto por parte de profesores como de alumnos.

Código	**Requisito Funcional**
RF01	El sistema debe permitir a los usuarios enviar preguntas al *chatbot*.
RF02	El sistema debe recuperar información relevante de los documentos y generar respuestas usando RAG.
RF03	El sistema debe permitir valorar las respuestas como útiles o no útiles.
RF04	El sistema debe permitir al profesor subir archivos PDF manualmente.
RF05	El sistema debe dividir automáticamente los documentos en fragmentos (*chunks*) y almacenarlos como *embeddings*.
RF06	El sistema debe permitir al profesor actualizar los documentos desde Moodle.
RF07	El sistema debe permitir al profesor eliminar archivos del sistema.
RF08	El sistema debe permitir modificar el umbral de recuperación semántica.
RF09	El sistema debe permitir iniciar nuevos chats.
RF10	El sistema debe almacenar todas las preguntas y respuestas en una base de datos relacional.
RF11	El sistema debe seleccionar la interfaz según el rol del usuario recibido desde Moodle (profesor o estudiante).

Tabla 4.1: Requisitos funcionales del sistema

4.1.2. Requisitos de información

Los requisitos de información describen los datos que el sistema debe gestionar, almacenar o procesar. Incluyen tanto datos persistentes (como documentos, mensajes o valoraciones) como configuraciones del sistema (umbral de recuperación).

Código	**Requisito de Información**
RI01	El sistema debe almacenar cada documento con nombre y fecha de subida.
RI02	El sistema debe registrar los chats iniciados y su fecha de creación.
RI03	El sistema debe registrar todos los mensajes enviados por el usuario y las respuestas generadas por el *chatbot*.
RI04	El sistema debe almacenar la valoración de las respuestas (*like*/*dislike*).
RI05	El sistema debe guardar la configuración actual del umbral de recuperación.

Tabla 4.2: Requisitos de información del sistema

4.1.3. Requisitos no funcionales

Los requisitos no funcionales son restricciones técnicas que debe cumplir el sistema, incluyendo aspectos como el rendimiento, la usabilidad, la seguridad o las tecnologías usadas.

Código	**Requisito No Funcional**
RNF01	El sistema debe integrarse con Moodle mediante LTI.
RNF02	El tiempo de respuesta del *chatbot* no debe superar los 10 segundos en promedio.
RNF03	El sistema debe permitir subir documentos en formato PDF.
RNF04	El sistema debe estar desarrollado en Python y usar Gradio como interfaz.
RNF05	Los *embeddings* deben almacenarse en una base vectorial local (ChromaDB).
RNF06	El sistema debe ser usable desde navegadores web modernos (Chrome, Firefox...).
RNF07	El sistema debe soportar a todos los usuarios matriculados en la asignatura sin perder rendimiento.
RNF08	Solo se deben indexar documentos nuevos (sin nombre duplicado).

Tabla 4.3: Requisitos no funcionales del sistema

4.1.4. Reglas de negocio

Las reglas de negocio son restricciones respecto al comportamiento del sistema, excluyendo aspectos técnicos. Son normas que deben cumplirse para que el sistema sea coherente con sus objetivos de funcionamiento.

Código	**Regla de Negocio**
RB01	Solo los usuarios con rol de profesor pueden subir, eliminar o actualizar archivos.
RB02	Si no se recupera contexto relevante, se debe generar una respuesta genérica.
RB03	Cada nuevo chat debe iniciar con un historial vacío.
RB04	La valoración de una respuesta se debe realizar una única vez por respuesta.

Tabla 4.4: Reglas de negocio del sistema

4.2. Casos de Uso

4.2.1. Actores principales

Los actores principales que interactúan con el sistema son tres:

- **Alumno.** Puede utilizar las funciones del *chatbot*, pero tiene acceso restringido al panel de configuración, en el que se encuentra la documentación RAG y otras opciones de configuración.
- **Profesor.** Tiene acceso a todas las funcionalidades del sistema. Puede interactuar con el *chatbot*, gestionar los documentos del flujo RAG y modificar los parámetros necesarios.
- **Moodle.** Sistema externo en el que se integrará la aplicación y del que se extraerá la documentación para la contextualización del modelo del *chatbot*.

Para los casos de uso realizables tanto por alumnos como por profesores, el actor se denominará `‘‘Usuario’’`, haciendo referencia a cualquier usuario que utilice el sistema. En ningún caso un alumno tendrá acceso a funcionalidades que no sean accesibles por el profesor.

4.2.2. Descripción de los casos de uso

A continuación, se describen de forma detallada los casos de uso identificados para el sistema. En cada caso de uso se muestra la secuencia principal de interacción entre los actores y el sistema, así como las posibles secuencias alternativas, precondiciones y postcondiciones. Las Tablas 4.5 hasta 4.11 contienen esta información para cada uno de los casos de uso.

Nombre	CU01 - Preguntar al *chatbot*
Actores	Profesor, Alumno
Descripción	El usuario introduce una pregunta y el sistema devuelve una respuesta generada con RAG.
Precondición	Usuario autenticado en Moodle
Secuencia principal	1. El usuario accede a la interfaz de chat. 2. El sistema muestra la interfaz de chat. 3. El usuario escribe una pregunta. 4. El sistema recupera documentos relevantes y genera la respuesta. 5. El sistema muestra la respuesta al usuario.
Secuencias alternativas	4a. Si no se encuentran documentos, se genera una respuesta genérica y finaliza el caso de uso.
Postcondición	Respuesta mostrada en la interfaz.

Tabla 4.5: Descripción CU01 - Preguntar al *chatbot*

Nombre	CU02 - Valorar respuesta
Actores	Profesor, Alumno
Descripción	El estudiante puede valorar la respuesta del *chatbot* como útil o no útil.
Precondición	Haber recibido una respuesta válida

Secuencia principal	1. El sistema muestra la respuesta al usuario. 2. El usuario pulsa *"like"* o *"dislike"*. 3. El sistema registra la valoración.
Secuencias alternativas	2a. Si no se valora, no se registra interacción y finaliza el caso de uso.
Postcondición	Valoración guardada.

Tabla 4.6: Descripción CU02 - Valorar respuesta

Nombre	CU03 - Subir archivo manualmente
Actores	Profesor
Descripción	El profesor puede subir documentos desde su equipo para incluirlos en el corpus del sistema RAG.
Precondición	Usuario autenticado como profesor
Secuencia principal	1. El profesor accede a la pestaña de gestión de archivos. 2. El sistema muestra la interfaz de subida de archivos. 3. El profesor selecciona uno o más archivos locales. 4. El sistema procesa e indexa los archivos.
Secuencias alternativas	3a. Tipo de archivo no soportado, se muestra el error y se omite ese archivo.
Postcondición	Archivos disponibles para recuperación.

Tabla 4.7: Descripción CU03 - Subir archivo manualmente

Nombre	CU04 - Actualizar archivos de Moodle
Actores	Moodle
Descripción	Permite actualizar los archivos importados automáticamente desde Moodle.
Precondición	Integración con Moodle configurada
Secuencia principal	1. El profesor accede a la pestaña de gestión de archivos. 2. El sistema muestra el botón de actualizar archivos de Moodle. 3. El profesor pulsa el botón de actualizar archivos. 4. El sistema conecta con Moodle y descarga los documentos del curso. 5. El sistema procesa e indexa los documentos automáticamente.

Secuencias alternativas	4a. Error al conectar con Moodle, se vuelve al paso 2. 5a. Si hay documentos no admitidos, se omiten y finaliza el caso de uso.
Postcondición	Documentos incorporados correctamente.

Tabla 4.8: Descripción CU04 - Actualizar archivos de Moodle

Nombre	CU05 - Modificar umbral de recuperación
Actores	Profesor
Descripción	El profesor puede ajustar el valor que determina la similitud mínima requerida para recuperar un chunk de información.
Precondición	Usuario autenticado como profesor
Secuencia principal	1. El profesor accede a la pestaña de gestión de archivos. 2. El sistema muestra el *slider* que permite modificar el parámetro. 3. El profesor ingresa un nuevo valor de umbral. 4. El sistema guarda el nuevo valor y lo aplica.
Secuencias alternativas	No hay
Postcondición	Nuevo umbral configurado.

Tabla 4.9: Descripción CU05 - Modificar umbral de recuperación

Nombre	CU06 - Eliminar archivo
Actores	Profesor
Descripción	Permite eliminar documentos del sistema RAG.
Precondición	Archivos subidos previamente
Secuencia principal	1. El profesor accede a la pestaña de gestión de archivos. 2. El sistema muestra la lista de archivos subidos. 3. El profesor selecciona un archivo para eliminar. 4. El sistema borra el archivo de la base de datos.
Secuencias alternativas	No hay
Postcondición	Archivo eliminado del sistema.

Tabla 4.10: Descripción CU06 - Eliminar archivos del corpus

Nombre	CU07 - Crear nuevo chat
Actores	Profesor, Alumno

Descripción	Permite iniciar una nueva conversación desde cero, creando un nuevo chat en la base de datos.
Precondición	Usuario autenticado y en la interfaz del *chatbot*
Secuencia principal	1. El usuario pulsa el botón "Nuevo Chat". 2. El sistema resetea el historial de mensajes mostrado en la interfaz. 3. El sistema crea un nuevo chat en la base de datos y muestra el nuevo chat vacío.
Secuencias alternativas	No hay.
Postcondición	Nuevo chat iniciado y preparado para nuevas interacciones.

Tabla 4.11: Descripción CU07 - Crear nuevo chat

4.2.3. Diagrama de casos de uso

La Figura 4.1 muestra el diagrama de casos de uso del sistema, en el que se representan los actores principales y su interacción con las distintas funcionalidades. Este diagrama muestra de forma clara qué acciones puede realizar cada tipo de usuario dentro de la aplicación.

El caso de uso 'Valorar respuesta' solo es accesible tras haber realizado 'Preguntar al *chatbot*'. Sin embargo, este caso de uso no se realiza automáticamente tras hacer una pregunta al *chatbot*, sino que el actor debe realizarlo.

Por otro lado, se identifica a 'Moodle' como un actor externo, cuya única función es llevar a cabo la actualización de los archivos existentes en la plataforma. Aunque es el profesor quien inicia la acción mediante la pulsación del botón correspondiente, la ejecución principal de la tarea recae en Moodle. Por este motivo, y tras consultar con profesores, se ha decidido que el actor responsable del caso de uso 'Actualizar archivos Moodle' sea Moodle.

En el diagrama se han duplicado los casos de uso `Crear nuevo chat` y `Preguntar al chatbot` para mostrar que pueden ser realizados tanto por el Profesor como por el Alumno. En UML, el hecho de que varios actores estén conectados a un mismo caso de uso puede interpretarse de forma ambigua, ya que en algunos contextos esto puede implicar que ambos actores deben intervenir simultáneamente. Por este motivo, se ha decidido duplicar estos casos de uso, dejando clara su disponibilidad para ambos roles de forma independiente.

4.3. Modelo de dominio

El modelo de dominio describe las entidades principales que forman el sistema y las relaciones entre ellas. Mediante clases conceptuales, se estructura el flujo y gestión de los datos que se emplean en nuestro sistema. La Figura 4.2 muestra el diagrama del modelo de dominio.

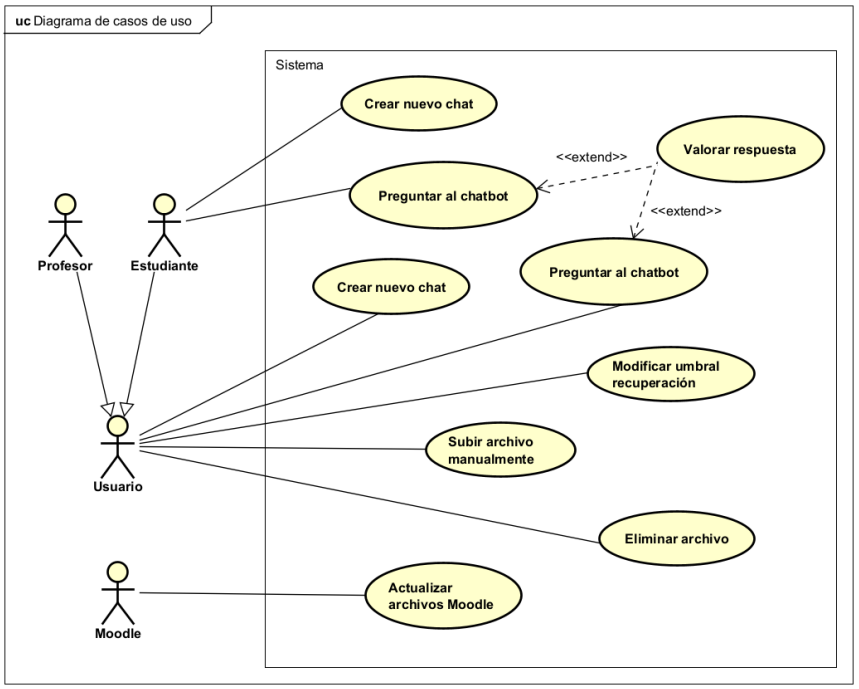

Figura 4.1: Diagrama de casos de uso

A continuación, se detallan las entidades que lo conforman:

- Clase **Chat**. Representa los chats creados por los usuarios. Cada uno tendrá un identificador y la fecha de creación. Un chat puede contener varios mensajes.
- Clase **Mensaje**. Se almacenan todos los mensajes de cada chat, incluyendo preguntas del usuario y respuestas del bot. Cada mensaje pertenece a un solo chat.
- Clase **Documento**. Los documentos se identifican con el nombre y el momento de subida. Se almacenan en el sistema para recuperar contexto con RAG, pero no están asociados a un chat específico ni a quién los subió. Un mensaje "puede utilizar" un documento para mostrar la información correspondiente.

Para recopilar datos sobre el uso y funcionamiento del sistema, únicamente nos interesan los chats y sus mensajes. Es por eso que los chats no van asociados al usuario que los creó, de forma que se mantiene la privacidad de las conversaciones de los usuarios.

Por otro lado, la autenticación de los usuarios se realiza a través de Moodle, y es quien identifica el rol de cada usuario para mostrar la interfaz correspondiente. Por esta razón, la información de autenticación de los usuarios no se almacena. Únicamente se utiliza el rol de cada usuario en la clase `Chat` para conocer el tipo de usuario que crea cada chat.

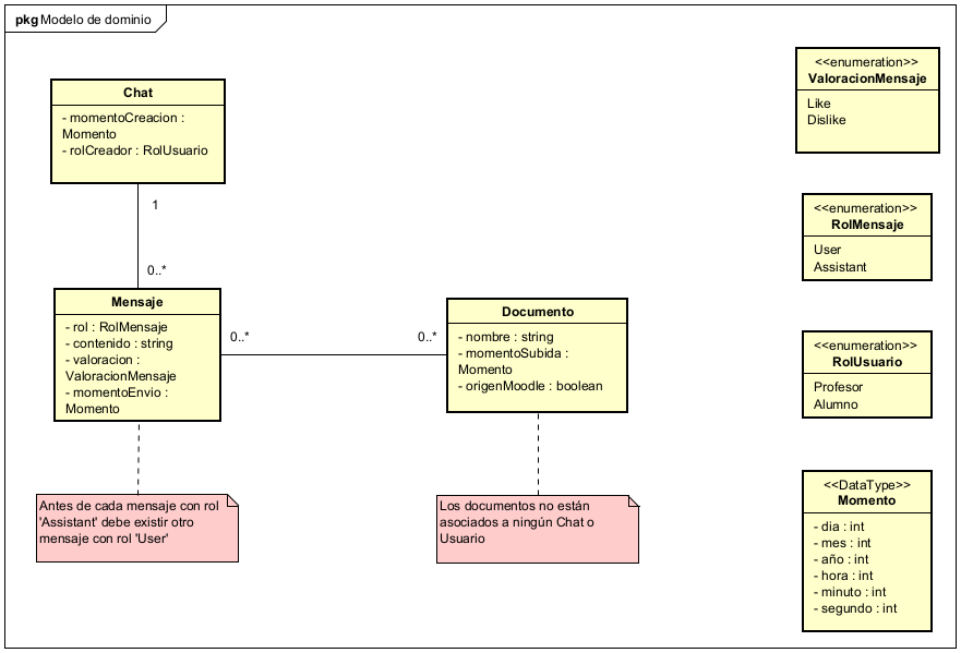

Figura 4.2: Modelo de dominio

En posteriores versiones del *chatbot* se prevé incluir una entidad `Curso`, de forma que se pueda implementar esta herramienta en varios cursos diferentes y distinguirlos entre sí, donde cada uno tendrá sus documentos correspondientes.

4.4. Realización en análisis de los casos de uso

En este apartado, dado que el sistema no está estructurado en clases ni define objetos, no consideramos apropiado el uso de diagramas de secuencia. En su lugar, se emplearán diagramas de actividades, ya que permiten representar de forma más adecuada los flujos de ejecución y de datos que tienen lugar en el sistema.

Los diagramas de actividades son una herramienta del lenguaje UML que permite representar cómo se coordinan distintas actividades para proporcionar un servicio. Su principal utilidad es la de describir el flujo de acciones necesarias para alcanzar un objetivo [21]. En las Figuras 4.3 a 4.9 se muestran dichos diagramas de actividades.

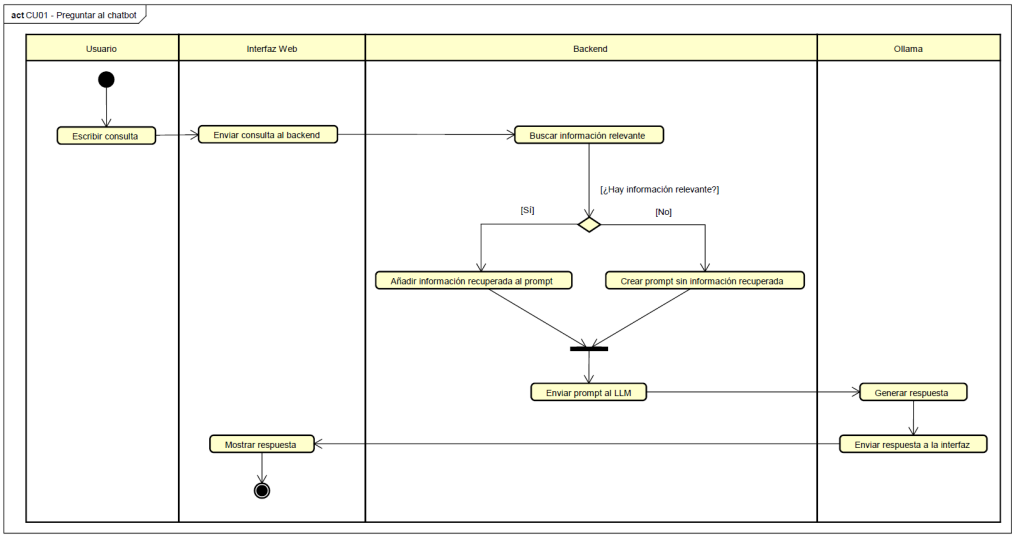

Figura 4.3: Diagrama de actividad CU01 - Preguntar al *chatbot*

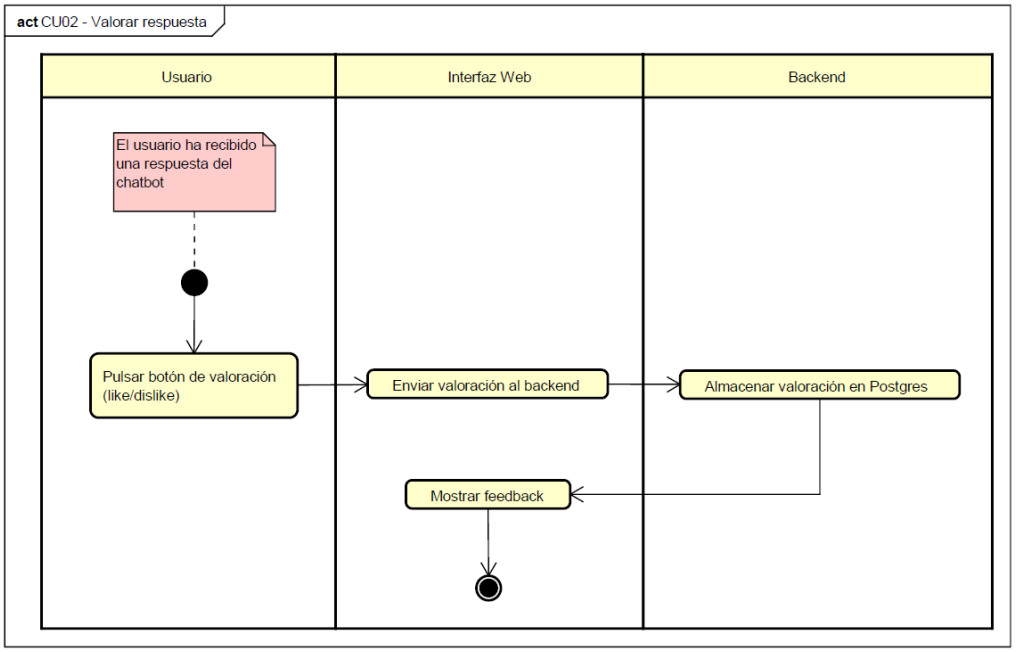

Figura 4.4: Diagrama de actividad CU02 - Valorar respuesta

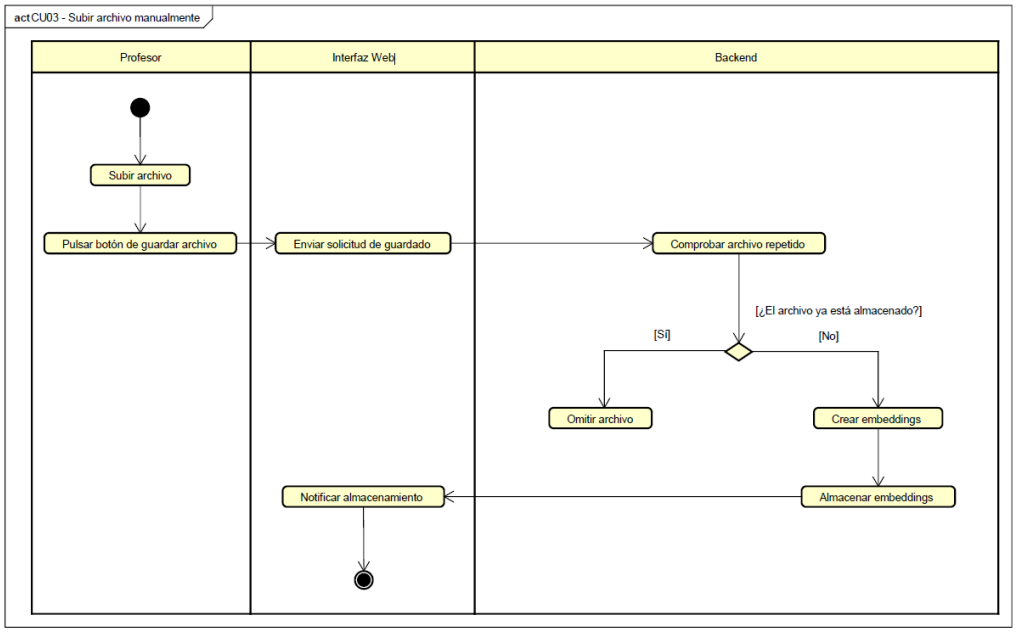

Figura 4.5: Diagrama de actividad CU03 - Subir archivo manualmente

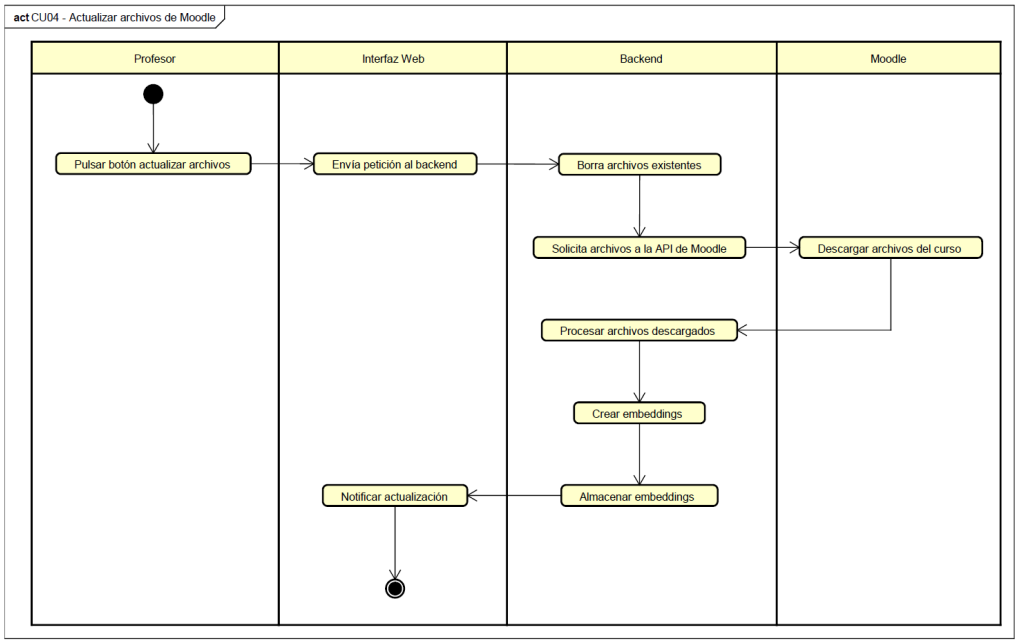

Figura 4.6: Diagrama de actividad CU04 - Actualizar archivos Moodle

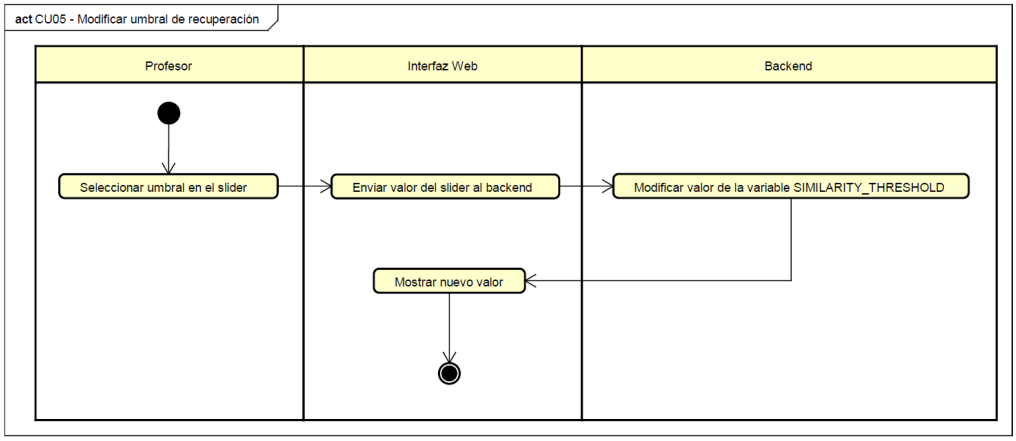

Figura 4.7: Diagrama de actividad CU05 - Modificar umbral de recuperación

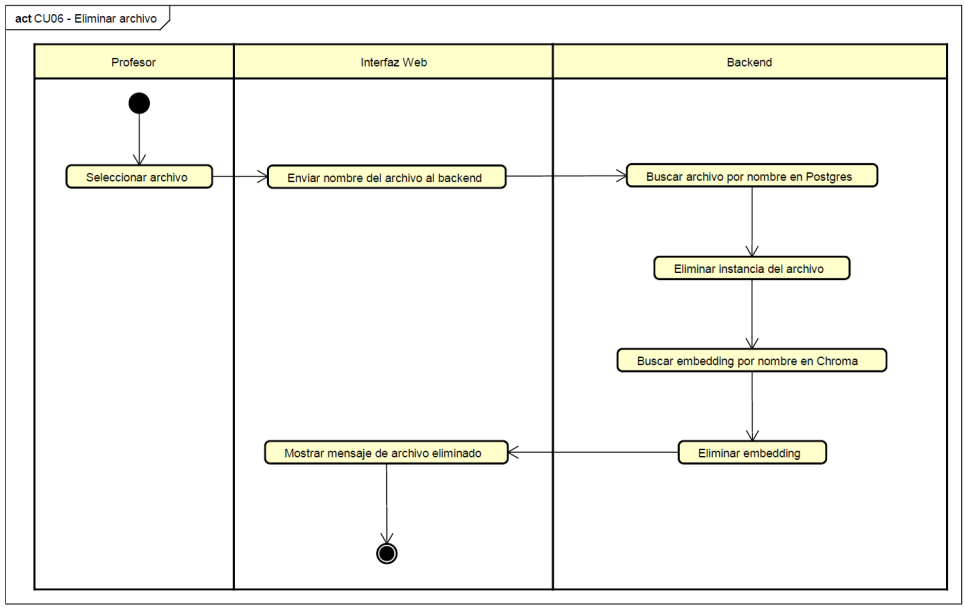

Figura 4.8: Diagrama de actividad CU06 - Eliminar archivo

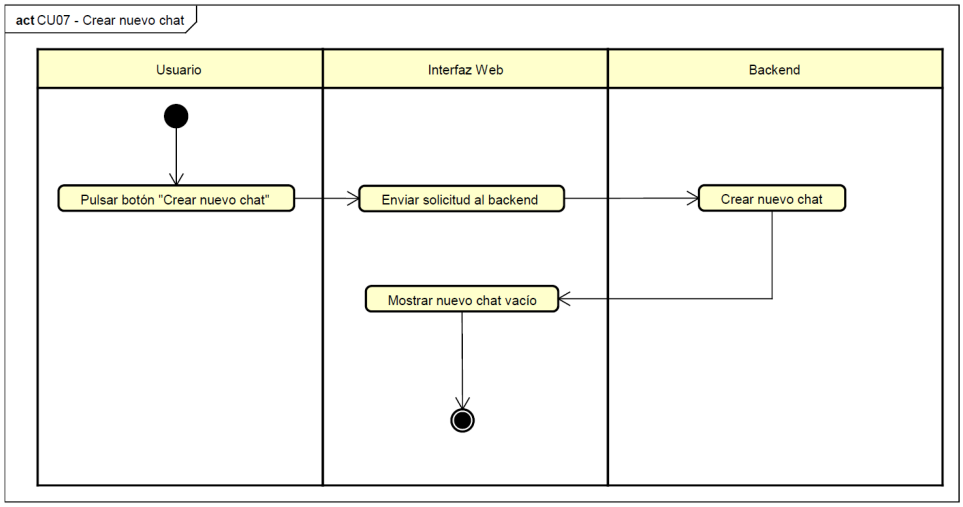

Figura 4.9: Diagrama de actividad CU07 - Crear nuevo chat

Capítulo 5

Diseño

En este capítulo se detalla el proceso de diseño del sistema desarrollado, abordando tanto la arquitectura general como los componentes que lo forman. Se describen las decisiones tomadas en cuanto a diseño, la interacción entre componentes y la organización de los mismos dentro del sistema. Además, se justificará el uso de algunas de las tecnologías que se emplean en el desarrollo.

5.1. Decisiones de diseño

A lo largo del desarrollo del sistema se han tomado una serie de decisiones técnicas y de diseño con el fin de satisfacer los requisitos planteados.

- **Lenguaje de programación backend: Python.** Se ha utilizado Python como lenguaje principal por su versatilidad, su ecosistema de librerías especializadas en inteligencia artificial y procesamiento de lenguaje natural, y su integración sencilla con bases de datos y *frameworks* de interfaz.
- **Interfaz gráfica con Gradio.** Para construir la interfaz web se ha optado por el *framework* Gradio, que permite generar interfaces de *chatbot* funcionales de forma rápida y conectarlas directamente con funciones de Python.
- **Modelo de lenguaje llama3.2 con Ollama.** Se ha elegido ejecutar localmente el modelo `llama3.2` mediante el gestor Ollama por ser un modelo muy ligero y, a su vez, ofrecer buenos resultados para preguntas relacionadas con programación. Esto permite mantener el sistema completamente funcional en local para el desarrollo de este TFG.
- **Arquitectura RAG (*Retrieval-Augmented Generation*).** La recuperación de información se realiza mediante un sistema RAG, que permite combinar la generación de respuestas con la recuperación semántica de documentos, lo que especializa al *chatbot*

en la asignatura que se desee. Esto supone una decisión de diseño ya que, en comparación con la técnica de *fine-tuning*, proporciona mejores resultados para nuestro contexto específico y hay mayor interés de las empresas, como se ha explicado anteriormente en el capítulo 2.

- **Modelo de *embeddings* all-MiniLM-L6-v2.** Para el almacenamiento vectorial de los textos se ha utilizado el modelo `all-MiniLM-L6-v2`, que ofrece un buen equilibrio entre precisión y rendimiento.

- **Almacenamiento vectorial en ChromaDB.** Los documentos procesados se almacenan en una base de datos vectorial ChromaDB, que permite realizar búsquedas por similitud utilizando los *embeddings* generados.

- **Base de datos relacional PostgreSQL.** Toda la información estructurada del sistema (mensajes, chats, documentos, etc.) se guarda en una base de datos PostgreSQL, por su popularidad y el conocimiento previo sobre ella.

- **Conexión a Moodle mediante API Flask** Para integrar el sistema con Moodle se ha desarrollado una API ligera utilizando Flask, compatible con el estándar LTI. Esta API permite recibir peticiones directamente desde Moodle, incluyendo la identificación del usuario y su rol, y redirigir al usuario a la interfaz correspondiente del *chatbot* (estudiante o profesor).

- **Persistencia de sesiones de chat.** Cada conversación del usuario se asocia a una nueva entrada en la tabla de chats, lo que permite identificar los mensajes que pertenecen a cada chat.

- **Separación de roles: profesor y estudiante.** El sistema implementa dos perfiles de usuario distintos. Los profesores disponen de funcionalidades avanzadas como la gestión de documentos y configuración del sistema, mientras que los estudiantes solo pueden interactuar con el *chatbot* y valorar las respuestas.

- **Gestión de documentos desde Moodle.** Se permite importar automáticamente los archivos del curso desde Moodle a través de su API, facilitando la actualización periódica de la base de conocimiento por parte del profesor.

5.2. Arquitectura cliente-servidor de la aplicación

El sistema desarrollado en este trabajo sigue el modelo clásico de arquitectura cliente-servidor, una arquitectura muy utilizada en aplicaciones web. En este modelo, las responsabilidades se dividen entre dos componentes principales: el cliente (ligero), que interactúa directamente con el usuario, y el servidor, que se encarga del procesamiento, la gestión de datos y las operaciones internas del sistema.

Cliente

En el contexto de esta aplicación, el cliente corresponde al navegador web, desde el cual el usuario accede a la interfaz gráfica. A través de esta interfaz, puede formular preguntas al *chatbot*, subir archivos, valorar respuestas o configurar parámetros, dependiendo del rol del usuario. Esta capa se encarga únicamente de la presentación de la información y de capturar las acciones del usuario para enviarlas al servidor.

Servidor

Por otro lado, el servidor gestiona toda la lógica del sistema. Está desarrollado en Python y se encarga de:

- Procesar las preguntas del usuario.
- Crear y almacenar *embeddings* en ChromaDB a partir de los archivos proporcionados.
- Recuperar archivos de la base de datos de vectores ChromaDB.
- Generar respuestas mediante el modelo de lenguaje local (Ollama).
- Almacenar información del sistema en la base de datos (PostgreSQL).
- Gestionar funcionalidades como la valoración de respuestas, la creación de nuevos chats o la actualización de los documentos.

La comunicación entre el cliente y el servidor se realiza mediante peticiones HTTP. El cliente envía datos como preguntas o valoraciones a través del navegador, y el servidor responde procesando la solicitud y devolviendo una respuesta adecuada, como la respuesta del chatbot o una confirmación de la acción realizada.

5.3. Persistencia de los datos

El sistema aplicación almacena información persistente de dos formas distintas: **ChromaDB** y **PostgreSQL**. Esto permite cubrir tanto la recuperación semántica de documentos como la gestión de datos estructurados del sistema.

5.3.1. ChromaDB: base de datos de vectores

ChromaDB se utiliza para almacenar *embeddings* vectoriales generados a partir del contenido de los documentos cargados tanto por el usuario como desde Moodle. Estos vectores son

generados mediante el modelo de *embeddings* `all-MiniLM-L6-v2`, y sirven para representar el significado semántico del texto de forma numérica.

Una vez cargados los documentos, su contenido se divide en fragmentos (*chunks*), lo que se conoce como *tokenización*. Cada fragmento se transforma en un vector que se almacena en ChromaDB y, cuando el usuario realiza una pregunta, se realiza una búsqueda por similitud en ChromaDB para recuperar los fragmentos más relevantes, lo que permite mejorar las respuestas generadas por el LLM en cuanto a precisión y contextualización.

5.3.2. PostgreSQL: almacenamiento estructurado

PostgreSQL se emplea como sistema gestor de bases de datos para almacenar información estructurada. En este sistema se almacenan:

- Los **mensajes** intercambiados entre el usuario y el *chatbot*, incluyendo el contenido, el rol (usuario o asistente) y la valoración.
- Los **chats** creados por el usuario, que agrupan los mensajes relacionados.
- Los **metadatos de documentos**, como el nombre de archivo y la fecha de subida.

5.4. Diagrama de despliegue

Como se ha mencionado anteriormente, el sistema sigue una arquitectura de tipo cliente-servidor, donde la lógica principal del sistema reside en el *backend*, desarrollado en Python, y la interfaz de usuario se muestra a través del navegador web. En el diagrama de despliegue (Figura 5.1) se han representado los distintos elementos que conforman el sistema, así como sus relaciones.

Por un lado, se ha modelado el nodo **Cliente Web**, dentro del cual se encuentra el dispositivo **Navegador**. En su interior se ha incluido el artefacto **Interfaz Gradio**, que representa la interfaz HTML/JS que el usuario ve y utiliza para comunicarse con el *chatbot*. Esta interfaz se renderiza en el navegador del usuario, pero no es un *frontend* tradicional como los desarrollados con React o Angular, sino que se ejecuta desde el *backend* en Python, generando dinámicamente el contenido de la interfaz. Por tanto, el navegador solo actúa como punto de entrada, pero la interfaz está completamente controlada desde el servidor.

En el nodo **Servidor *Backend*** se representan cuatro entornos de ejecución distintos:

- **Python Runtime**, donde se despliegan tres componentes:
  - **API Flask**, responsable de recibir las peticiones LTI desde Moodle.
  - **Gradio**, que genera y sirve la interfaz al usuario.

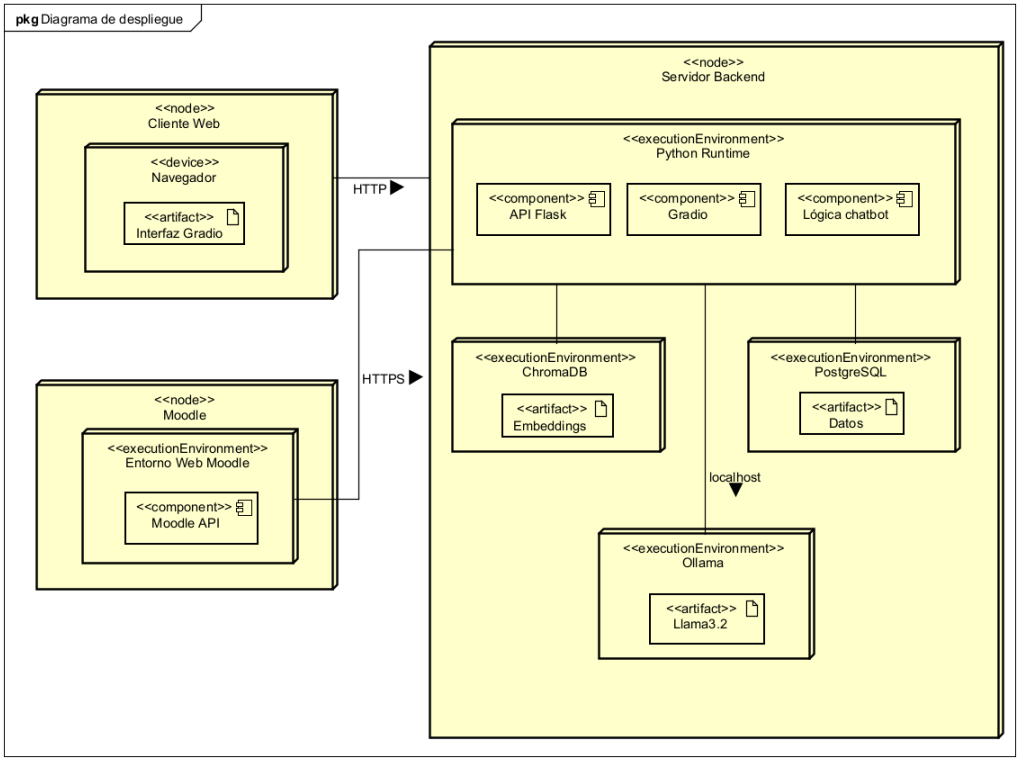

Figura 5.1: Diagrama de despliegue del sistema

    - **Lógica del *chatbot***, que procesa preguntas, documentos, valoraciones y gestiona todos los procesos.

- **Ollama**, encargado de ejecutar el modelo de lenguaje `LLaMA 3.2`, representado como un artefacto. Ollama recibe *prompts* desde la lógica del *chatbot* y devuelve las respuestas generadas.
- **ChromaDB**, donde se almacenan los ***embeddings*** generados a partir del contenido de los documentos. Esta base de datos se utiliza para recuperar contexto relevante.
- **PostgreSQL**, que contiene los datos estructurados del sistema, como los mensajes intercambiados en el chat, los metadatos de los documentos, el *feedback* del usuario o información sobre los chats.

También se ha incluido el nodo **Moodle**, que tiene dos funciones:

- Autenticación mediante el protocolo LTI
- Acceso a los documentos del curso a través de su API REST

La interacción con Moodle se representa con una única línea de comunicación entre los entornos de ejecución Python Runtime y Entorno Web Moodle, aunque en realidad son dos flujos distintos. El primero es una petición HTTP POST enviada por Moodle a la API Flask para autenticar al usuario e identificar su rol (estudiante o profesor). El segundo es una petición HTTPS GET realizada por el backend para descargar los archivos del curso y añadirlos al sistema. En el diagrama se muestra una única comunicación para simplificar el diagrama y evitar confusión con demasiadas líneas.

En cuanto a la posible relación entre Flask y Gradio, estos se ejecutan de forma independiente. Flask actúa como intermediario, recibiendo la autenticación desde Moodle y redirigiendo al usuario a la interfaz adecuada mediante un `iframe`, mientras que Gradio sirve directamente la interfaz de usuario. Por ello, en el diagrama no se ha representado una dependencia directa entre ambos.

En el Servidor *Backend*, los diferentes entornos de ejecución se comunican entre sí. El componente Lógica *Chatbot* utiliza Ollama para generar las respuestas del modelo, consulta ChromaDB para recuperar información relevante y emplea PostgreSQL para guardar los mensajes, valoraciones, chats y otros datos importantes. Estas conexiones se reflejan en el diagrama de despliegue como líneas de comunicación entre los entornos de ejecución que forman parte del *backend*.

Capítulo 6

Implementación

En este capítulo se detalla el proceso de construcción de la aplicación, que se ha basado en las especificaciones definidas en el análisis y el diseño. Se explican todas las fases que ha habido en la implementación, las decisiones tomadas a nivel de código y arquitectura, y los problemas que han ido surgiendo. La idea es mostrar cómo se ha llegado hasta el sistema final y qué herramientas se han utilizado para ello.

6.1. Implementación de las tecnologías

El sistema utiliza diferentes tecnologías de código abierto para implementar el *chatbot* con RAG. Aunque en apartados anteriores ya se ha explicado la selección de estas herramientas, en esta sección se describe de manera más práctica cómo se realiza el lanzamiento de cada componente, qué significa cada parte del código y algunas dificultades encontradas durante la integración.

- **Gradio**: Se utiliza para crear la interfaz web del chatbot. El lanzamiento de la interfaz se realiza en Python agrupando los distintos bloques (chat y gestor de archivos) en una única interfaz con pestañas. Por ejemplo:

```
with gr.Blocks() as chatbot_interface:
gr.Markdown("...")
chatbot = gr.Chatbot(...)
...
interface_profesor = gr.TabbedInterface([chatbot_interface, file_manager
    ], ["Chatbot", "Gestor de Archivos"])
interface_profesor.launch(...)
```

Código 6.1: Inicialización de la interfaz con Gradio

En este fragmento, `chatbot_interface` corresponde a la pestaña del chat, mientras que `file_manager` contiene la interfaz de gestión de RAG. El método `.launch()` despliega toda la interfaz en el navegador para el usuario.

- **Flask (API)**: Esta API gestiona la autenticación de usuarios desde Moodle mediante LTI. La integración LTI consiste en una llamada POST enviada por Moodle al servidor, incluyendo varios parámetros entre los que nos interesa el rol del usuario (por ejemplo, "*Instructor*" o "*Student*"). En el código, el sistema simplemente distingue el rol recibido y muestra la interfaz correspondiente:

```
@app.route("/", methods=["GET", "POST"])
def launch_lti():
user_role = request.form.get("roles", "")
if 'Instructor' in user_role:
return render_template_string(...iframe profesor...)
else:
return render_template_string(...iframe alumno...)
```

Código 6.2: Redirección según rol en Flask

Esto permite adaptar las funcionalidades mostradas en función del tipo de usuario autenticado desde Moodle.

- **moodle_api**: Es un módulo de un repositorio público (`https://github.com/mrcinv/moodle_api.py`) que permite realizar llamadas a la API REST de Moodle de forma sencilla. Se utiliza para obtener automáticamente los documentos del curso de Moodle y procesarlos desde el *backend*:

```
course_content = moodle_api.call('core_course_get_contents', courseid=
    MOODLE_COURSE_ID)
```

Código 6.3: Llamada a la API de Moodle

La integración fue sencilla, aunque hubo que adaptar los parámetros de autenticación (como el id del curso o la API *key*) para evitar errores al acceder a los contenidos del curso.

- **LangChain**: Esta librería se utiliza para enlazar el modelo de *embeddings* con la base vectorial ChromaDB.

```
embedding_model = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
vectorstore = Chroma(persist_directory=CHROMA_PATH, embedding_function=
    embedding_model)
```

Código 6.4: Inicialización de embeddings y vectorstore

Aquí la principal dificultad fue ajustar correctamente el tamaño de los fragmentos y el umbral de similitud para que las respuestas fueran lo más relevantes posible. En el capítulo 7 se mostrarán las pruebas realizadas para la configuración de dichos parámetros.

- **Ollama**: Permite ejecutar el modelo LLaMA 3.2 localmente para generar respuestas. El parámetro `stream=True` sirve para que los *tokens* se devuelvan a medida que se generan, sin esperar a que se generen todos, acelerando la respuesta en la interfaz web:

```
stream = ollama.chat(model="llama3.2", messages=messages, stream=True)
```

Código 6.5: Llamada al modelo LLaMA 3.2 con Ollama

- **PostgreSQL**: La base de datos almacena de forma persistente los datos estructurados del sistema: chats, mensajes y documentos. Estos datos y sus relaciones están definidos según el modelo de dominio presentado en el capítulo de análisis 4.2.
- **NLTK**: Es una librería de Python orientada al procesamiento de lenguaje natural. Aquí se emplea principalmente para procesar y limpiar las consultas de los usuarios, facilitando así la recuperación semántica más relevante desde la base de conocimiento:

```
def extraer_palabras_clave_nltk(texto):
texto = texto.lower().translate(str.maketrans('', '', string.punctuation)
    )
tokens = word_tokenize(texto, language='spanish')
stop_words = set(stopwords.words('spanish'))
return " ".join([palabra for palabra in tokens if palabra not in
    stop_words and palabra.isalpha()])
```

Código 6.6: Preprocesamiento de la consulta

El mayor reto aquí fue ajustar el preprocesamiento para evitar que se perdiera información importante de la consulta, manteniendo el equilibrio entre limpieza y contexto.

6.2. Estructura de archivos del proyecto

El sistema está estructurado de forma modular para facilitar su organización y entendimiento. A continuación, se describen los archivos y carpetas principales que componen el proyecto:

- **server.py**: Aplicación desarrollada con Flask que actúa como punto de entrada. Recibe las peticiones LTI desde Moodle, identifica el rol del usuario y redirige mediante `iframe` a la interfaz correspondiente.
- **interfaz_profesor.py**: Contiene la lógica de la interfaz gráfica para el rol de profesor. Dependiendo del rol del usuario, se mostrará la interfaz correspondiente a este archivo (profesor) o la del archivo `interfaz_alumno.py`.
- **interfaz_alumno.py**: Implementa la interfaz de usuario para alumnos. Esta versión presenta únicamente el *chatbot*, sin acceso a herramientas de gestión del RAG.
- **moodle_api.py**: Módulo obtenido desde el repositorio mencionado anteriormente, que facilita el acceso a la API REST de Moodle, permitiendo obtener los documentos del curso.

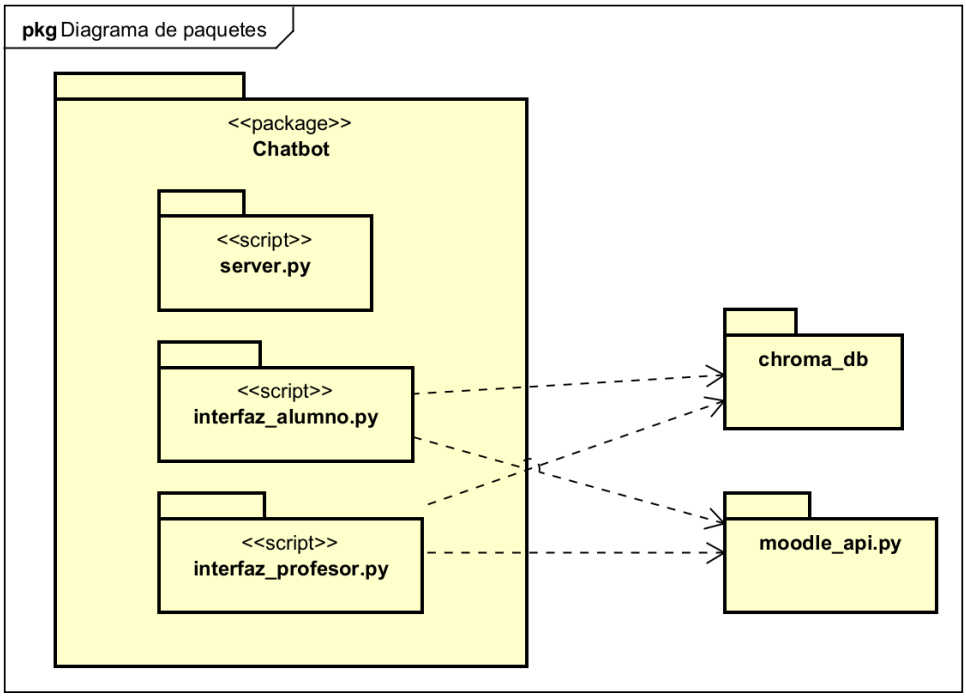

Figura 6.1: Diagrama de paquetes del proyecto.

- **chroma_db/**: Carpeta donde se almacena la base de datos vectorial generada con ChromaDB. Contiene los *embeddings* de los fragmentos de texto extraídos de los documentos PDF.

La Figura 6.1 muestra la organización de los distintos archivos y paquetes dentro del proyecto.

6.3. Flujo de funcionamiento (RAG)

El proceso RAG implementado consta de las siguientes fases:

1. Recepción de la consulta desde la interfaz Gradio.
2. Preprocesamiento y generación del *embedding* de la consulta.
3. Búsqueda semántica de fragmentos relevantes en ChromaDB.
4. Construcción del *prompt* para el modelo LLM.
5. Generación de la respuesta usando Ollama.

6. Almacenamiento de la interacción.

En el fragmento de código 6.7 se muestra el código utilizado para recuperar los `k` *chunks* más relevantes.

```
def retrieve_context(query, threshold):
    vectorstore = Chroma(...)
    docs_with_scores = vectorstore.similarity_search_with_relevance_scores(
    query_procesada, k=1)
    relevant_docs = [doc for doc, score in docs_with_scores if score >=
    threshold]
    ...
```

Código 6.7: Recuperación de contexto con embeddings

6.4. Integración con Moodle

Para sincronizar los documentos del RAG con los del curso de Moodle, se utiliza la función `actualizar_moodle`, que se encarga de conectarse con Moodle, obtener los contenidos del curso, descargar los archivos PDF, procesarlos y añadirlos a las bases de datos. De esta forma, se pueden mantener actualizados los documentos que se emplean en el flujo RAG, sin necesidad de que el profesor los suba manualmente.

Moodle tiene su propia API REST, que permite acceder a diferentes funciones. En este caso, se utiliza la función `core_course_get_contents`, que permite recuperar los contenidos de un curso a partir de su `courseid`. Este proceso se ejecuta automáticamente cuando el profesor pulsa el botón "Actualizar documentos desde Moodle" en la interfaz. Antes de añadir los nuevos archivos, se eliminan los antiguos para que solo se mantengan los documentos más recientes del curso.

La documentación oficial de la API se puede consultar en: `https://docs.moodle.org/dev/Web_service_API_functions#Web_service_functions`

```
def actualizar_moodle():
    ...
    course_content = moodle_api.call('core_course_get_contents', courseid=
    MOODLE_COURSE_ID)
    ...
    for section in course_content:
        for module in section.get('modules', []):
            if module['modname'] == 'folder':
                ...
```

Código 6.8: Actualización de archivos desde Moodle

6.5. Persistencia y gestión de datos

Los datos estructurados se almacenan en PostgreSQL, mientras que los datos vectoriales se guardan en ChromaDB. Cada documento PDF se divide en fragmentos, se vectoriza y se indexa:

```
chunks = text_splitter.split_text(text)
documents = [Document(page_content=chunk, metadata={"documento": filename})
    for chunk in chunks]
vectorstore.add_documents(documents)
vectorstore.persist()
```

Código 6.9: División y vectorización de documentos

6.5.1. Interfaz gráfica

La interfaz gráfica del sistema se ha desarrollado utilizando el *framework* **Gradio**, una herramienta de alto nivel utilizada para construir interfaces de usuario en aplicaciones de inteligencia artificial.

Gradio ofrece una interfaz sencilla para *chatbots* y para adaptarla a los requerimientos del proyecto, se ha personalizado incorporando dos pestañas mediante el componente `gr.TabbedInterface`. La primera pestaña está dedicada al ***chatbot*** y la segunda al **gestor de archivos**, accesible exclusivamente para usuarios con rol de profesor.

En la pestaña del ***chatbot***, se incluye un componente (`gr.MultimodalTextbox`) que permite al usuario escribir preguntas. Además, las respuestas generadas pueden ser valoradas por el usuario con un sistema de "*like/dislike*" para su posterior análisis.

En la pestaña del **gestor de archivos**, los profesores pueden:

- Subir archivos de forma manual.
- Consultar la lista de documentos cargados.
- Eliminar archivos.
- Actualizar el repositorio de documentos sincronizándolo con los materiales disponibles en Moodle.
- Ajustar el umbral de similitud usado para la recuperación semántica mediante un *slider*.

Toda la interfaz está contenida en una única aplicación lanzada con Gradio, y se visualiza directamente desde Moodle mediante una integración por `iframe`, que se adapta dinámicamente según el rol detectado por la API Flask.

```
interface_profesor = gr.TabbedInterface(
    [chatbot_interface, file_manager],
    ["Chatbot", "Gestor de Archivos"]
)
interface_profesor.launch(server_name="0.0.0.0", server_port=7860)
```

Código 6.10: Creación de la interfaz con pestañas usando Gradio

Esta distribución permite separar las funcionalidades para estudiantes y profesores.

6.6. Dificultades y soluciones aplicadas

- **Selección del modelo de *embeddings***: Se probaron varios modelos como `all-mpnet-base-v2` o `BAAI/bge-small-en-v1.5`. Tras numerosas consultas realizadas al *chatbot* con cada modelo, el que mejor recuperaba la información relevante fue `all-MiniLM-L6-v2`, que es el de propósito más general.
- **Selección del LLM**: Se valoraron varias opciones disponibles, tanto modelos cerrados como GPT de OpenAI como modelos de código abierto. Como buscábamos ejecutar la aplicación localmente, de decidió emplear un modelo de código abierto y ligero, de forma que pueda correr sin fallos de rendimiento. Finalmente, se optó por el modelo `llama3.2`, que ofrece un buen equilibrio entre rendimiento, uso de memoria y calidad de las respuestas, con buenos resultados en programación.
- **Preprocesamiento de consultas**: Se observó que las consultas demasiado largas o con frases genéricas generaban fallos en la recuperación de documentos. Para mejorar esto, se implementó un sistema de simplificación que elimina palabras comunes. Este proceso tiene dos partes: primero se utiliza `NLTK` para convertir el texto a minúsculas, eliminar signos de puntuación y filtrar las *stopwords* en español. Luego, se aplica un filtro adicional con una lista de palabras típicas de enunciados, como "haz", "escribe" o "función", que no aportan valor semántico a la búsqueda. El siguiente código muestra la función utilizada:

```
def extraer_palabras_clave(texto):
    # Convertir a min sculas y eliminar puntuacion
    texto = texto.lower().translate(str.maketrans('', '', string.
    punctuation))
    # Tokenizar
    tokens = word_tokenize(texto, language='spanish')
    # Cargar stopwords en espa ol
    stop_words = set(stopwords.words('spanish'))
    # Filtrar tokens
    palabras_clave = [palabra for palabra in tokens if palabra not in
    stop_words and palabra.isalpha()]
    return " ".join(palabras_clave)

PALABRAS_OMITIR = {
    "escribe", "haz", "metodo", "programa"...
}

def extraer_palabras_clave_personalizado(texto):
    tokens = texto.lower().split()
```

```
18      tokens_limpios = [t.strip(".,;:()\"'") for t in tokens if t not in
        PALABRAS_OMITIR]
19      return " ".join(tokens_limpios)
```

Código 6.11: Simplificación de consultas

Este preprocesado permitió mejorar la precisión en la recuperación de fragmentos relevantes, evitando coincidencias con textos que contenían palabras demasiado generales.

- **Sincronización Moodle**: En esta fase se dudó entre almacenar los documentos en disco o en memoria. Finalmente se decidió almacenarlos únicamente en memoria para hacer el proceso más ligero y no ocupar espacio en disco.
- **Gestión de archivos**: Al subir documentos, se comprueba si ya existen en la base de datos para no almacenar duplicados. Cuando se eliminan, deber ser eliminados tanto en PostgreSQL como en ChromaDB, ya que una inconsistencia entre las dos bases de datos provocaría duplicación o falta de documentos.

Capítulo 7

Pruebas

Este capítulo recoge el conjunto de pruebas realizadas para evaluar la calidad y el comportamiento del sistema desarrollado. Se han llevado a cabo tres tipos de pruebas: pruebas de ajuste de parámetros, pruebas comparativas entre respuestas generadas con y sin RAG, y pruebas de usabilidad, carga de trabajo y experiencia con usuarios reales mediante cuestionarios validados.

7.1. Pruebas de evaluación de parámetros

En esta sección se evalúa el impacto de distintos parámetros clave en la calidad y precisión de las respuestas generadas por el *chatbot*:

- Umbral de recuperación semántica.
- Tamaño de los fragmentos en los que se divide el texto (*chunks*).
- Número de documentos recuperados por consulta (*top-k*).

Para ello, se ha creado un conjunto de preguntas de prueba y se han analizado las respuestas generadas por el sistema al variar cada uno de estos parámetros. Las preguntas se han seleccionado de las preguntas que hicieron los alumnos de Fundamentos de Programación del curso 24-25, y que se recogieron a través de una iteración con ellos al inicio del trabajo. Gran parte de las preguntas proporcionadas van en la dirección de resolver ejercicios u obtener pistas sobre cómo empezar a resolver un problema. Otros alumnos indican que hacen uso de estas herramientas para verificar si el código creado por ellos mismos es correcto.

Por otro lado, los archivos con los que se hace RAG contienen ejercicios resueltos con su enunciado, por lo que algunas de las preguntas que se harán son enunciados de dichos ejercicios. De esta forma se podrá comprobar si la recuperación de los documentos funciona

correctamente. Con esta información, se han definido las siguientes consultas para realizar las evaluaciones:

1. ¿Cómo recorro un *array*?
2. Escribe un método que devuelva el valor medio de todos los elementos contenidos en una matriz.
3. Crea funciones para imprimir líneas en blanco.
4. Haz un programa que devuelva la secuencia de Fibonacci de un número dado usando recursividad.

La calidad de las respuestas ha sido evaluada a criterio del desarrollador (exalumno de dicha asignatura), en función de su adaptación al contexto, su claridad y su precisión. Debido a la falta de tiempo, no ha sido posible realizar una evaluación en profundidad de la calidad de las respuestas por parte de los profesores, pero antes de la integración de la herramienta en la asignatura, los profesores deberán realizar pruebas para determinar, bajo su criterio, qué parámetros son adecuados.

Para la elección de parámetros, hay que tener en cuenta la estructura de los documentos que actúan como contexto en RAG. En nuestro caso, cada documento contiene 1 único ejercicio que consta de su enunciado y su solución. Debido a esto, la idea es generar un *chunk* por documento, de forma que se recuperen ejercicios enteros y no queden incompletos.

Teniendo en cuenta lo anterior, será necesario que el tamaño de los *chunks* sea suficiente para abarcar los documentos completos. Como el modelo llama3.2 tiene una ventana de contexto grande (128k *tokens*) [1], se ha decidido fijar un **tamaño de *chunk* de 8000 caracteres** ($\approx$ 2k *tokens*). Así, cada *chunk* contendrá un documento completo y habrá espacio suficiente en la ventana de contexto para incluir el *prompt* y la respuesta.

Durante la realización de las pruebas, observé que una consulta más simple mejoraba la precisión de la recuperación de archivos. Por ello, se ha implementado un preprocesado de consultas, de forma que se extraen las palabras más relevantes, evitando frases como "escribe una función" o "crea un programa", que podían ocasionar falsas similitudes, y eliminando palabras sin significado semántico como artículos o preposiciones (Ver sección 6.6).

Como ejemplo del preprocesamiento realizado, si un usuario introduce la consulta ***"Haz un programa que calcule la media de los valores de una matriz"***, el sistema aplicará una serie de transformaciones para eliminar palabras genéricas. Tras el procesamiento con `NLTK` y el filtro personalizado, la frase queda reducida a ***"media valores matriz"***, manteniendo su valor semántico y mejorando la recuperación.

7.1.1. Variación del umbral de recuperación

En la Tabla 7.1 se muestran los resultados de esta evaluación. La calidad de la respuesta se mide en una escala de Baja, Media o Alta.

Umbral	Consulta	Documentos recuperados	Calidad de respuesta	Observaciones
0.1	1	Sí (irrelevantes)	Baja	Recupera aunque no haya ejercicios adecuados
	2	Sí (irrelevantes)	Baja	Recupera contexto incorrecto que empeora la respuesta
	3	Sí (correcto)	Alta	Recupera el archivo correcto
	4	Sí (varios)	Alta	Recupera más contexto de lo necesario, pero incluye el archivo correcto
0.25	1	No	Alta	Mejora respuesta al no usar contexto erróneo
	2	No	Alta	Mejora respuesta al no usar contexto erróneo
	3	Sí (correcto)	Alta	Igual que con 0.1
	4	Sí (varios)	Alta	Igual que con 0.1
0.4	1	No	Alta	Igual que 0.25
	2	No	Alta	Igual que 0.25
	3	No	Baja	No recupera archivo, respuesta más débil
	4	Sí (varios)	Alta	Igual que 0.1 y 0.25

Tabla 7.1: Resumen de resultados por umbral y consulta en el *chatbot*

La Tabla 7.1 muestra los resultados obtenidos al variar el valor del umbral de recuperación, utilizando las cuatro consultas ya mencionadas. Se observa que un umbral demasiado bajo (por ejemplo, 0.1) tiende a recuperar archivos irrelevantes, lo que perjudica la calidad de las respuestas. En cambio, con un umbral intermedio (0.25), el sistema evita ese problema y mantiene una alta calidad, incluso cuando no se recupera ningún archivo. Esto demuestra que, en muchos casos, es preferible no recuperar contexto a recuperar documentos que no están relacionados con la consulta.

Por otro lado, cuando en la columna de *Documentos recuperados* aparece *Sí (correcto)*, significa que se ha recuperado exactamente el documento que contiene la solución adecuada al ejercicio planteado. En los casos marcados como *Sí (varios)*, se han recuperado varios documentos, pero entre ellos se encuentra el archivo correcto. Esta situación, aunque no es tan precisa como recuperar un único documento, sigue ofreciendo resultados muy buenos y es preferible a no recuperar nada o a obtener documentos erróneos. En dicha tabla, se destacan en verde los casos en los que la información recuperada ayuda a la mejora de la respuesta.

Con estos resultados, se puede concluir que un **umbral de 0.25** parece ofrecer el mejor equilibrio entre recuperar información útil y evitar errores con documentos no relevantes.

7.1.2. Variación del número de archivos recuperados

Para esta prueba se va a emplear el umbral óptimo obtenido en el apartado anterior (0.25), y se va a variar el número de archivos recuperados (k) en cada consulta para determinar si la calidad de las respuestas puede variar con este parámetro.

Como se ha visto en la prueba anterior, con un umbral de 0.25 solo se recuperaba información con las consultas 3 y 4, por lo que serán las que se empleen para esta prueba. Las consultas 1 y 2 arrojan exactamente los mismos resultados.

Se han utilizado valores de 1, 2 y 3 archivos recuperados, y las respuestas han sido muy similares para todos los valores. Por lo general, cuando se recuperan varios archivos entre los cuales se encuentra el archivo correcto, la respuesta suele ir bastante enfocada hacia dicho documento, por lo que no parece un problema el recuperar más archivos de los necesarios.

Top-k	**Consulta**	**Documentos recuperados**	**Calidad de respuesta**	**Observaciones**
1	3	Sí (correcto)	Alta	Se recupera únicamente el ejercicio esperado, la respuesta es precisa
	4	Sí (correcto)	Alta	Se recupera solo el ejercicio sobre Fibonacci, respuesta directa y adecuada
2	3	Sí (varios)	Alta	Aunque se recuperan más archivos, la respuesta sigue centrada en el correcto
	4	Sí (varios)	Alta	Recupera varios ejercicios, pero el *chatbot* selecciona correctamente el relevante
3	3	Sí (varios)	Alta	Contexto adicional no interfiere, la respuesta se mantiene centrada en el ejercicio clave
	4	Sí (varios)	Alta	A pesar del mayor número de documentos, la respuesta es coherente y adecuada

Tabla 7.2: Resultados al variar el número de documentos recuperados (k) con umbral fijo en 0.25

Los resultados muestran que el valor de k no influye de forma notable en la calidad de las respuestas, al menos en las consultas utilizadas en estas pruebas. Para sacar conclusiones más firmes, sería necesario probar con más consultas y diferentes combinaciones de parámetros. Sin embargo, por limitaciones de tiempo, esto no se ha podido llevar a cabo en este trabajo, por lo que esta evaluación más exhaustiva se plantea como una posible línea de trabajo futuro.

A partir de los resultados obtenidos en las pruebas anteriores, se concluye que la mejor combinación de parámetros para este sistema es utilizar un **umbral de recuperación de 0.25** y permitir recuperar hasta **3 documentos por consulta** (*top-k = 3*). Esta configuración permite obtener respuestas precisas y coherentes, evitando el uso de contexto irrelevante. Además, se ha fijado un **tamaño de *chunk* de 8000 caracteres**. De esta forma, cada fragmento almacenado en la base de datos vectorial contiene un documento completo, lo cual mejora la recuperación semántica y facilita que el modelo genere respuestas más contextualizadas a partir de un solo fragmento.

7.2. Comparación RAG vs no RAG

Para analizar el valor que aporta el sistema RAG, se han utilizado las cuatro preguntas formuladas anteriormente. Cada pregunta se envió al *chatbot* tanto con RAG desactivado como activado, utilizando los parámetros obtenidos en la evaluación anterior, y se compararon las respuestas obtenidas.

En este apartado se van a evaluar dos aspectos:

- Calidad, precisión y contextualización de las respuestas en el marco de la asignatura.
- Eficiencia del modelo con RAG respecto al modelo sin RAG introduciendo un contexto largo manualmente.

7.2.1. Calidad, precisión y contextualización

En esta sección se comparan las respuestas obtenidas para las consultas definidas anteriormente, evaluando tres aspectos clave: la calidad general de la respuesta, su precisión técnica y su nivel de contextualización respecto a la asignatura.

En las cuatro consultas analizadas, se observa que el uso del sistema con RAG aporta una mejora significativa. En primer lugar, permite generar las respuestas en el lenguaje Java, evitando la aparición de otros lenguajes como JavaScript, que son frecuentes en modelos genéricos. Además, las respuestas generadas con RAG tienden a ser más simples y adecuadas al nivel de los estudiantes que están empezando a programar. Por el contrario, el sistema sin RAG genera con frecuencia código avanzado que incluye librerías o estructuras no abordadas en la asignatura, lo que puede dificultar su comprensión.

El valor del sistema con RAG va más allá de seleccionar el lenguaje adecuado o simplificar el código. Para ilustrarlo, se ha escogido una consulta concreta que pone de manifiesto una de las principales ventajas de contextualizar las respuestas:

Consulta: Haz un programa que define una clase Tiempo y permite sumar dos tiempos introducidos por el usuario.

Ante esta consulta, el sistema sin RAG proporciona una solución válida desde el punto de vista técnico, pero basada en programación orientada a objetos. Esta aproximación no es adecuada, ya que la asignatura no incluye ese paradigma. La respuesta incluye conceptos como clases, métodos personalizados y decoradores como `@Override`, todos ellos fuera del alcance del temario.

Este ejemplo muestra cómo el uso de RAG permite generar respuestas adaptadas al contenido real de la asignatura. Al recuperar un ejercicio resuelto que aborda la misma problemática desde un enfoque estructurado, el sistema es capaz de producir una solución que encaja mejor con lo que se espera de los alumnos. En cambio, un chatbot sin recuperación de contexto puede ofrecer resultados correctos en lo técnico pero desalineados con el enfoque docente, lo que puede generar confusión, dificultar el aprendizaje y afectar negativamente al rendimiento académico del estudiante.

7.2.2. Eficiencia del sistema

Con el objetivo de evaluar la influencia de la arquitectura RAG sobre la eficiencia del sistema, se ha realizado una comparativa entre el funcionamiento del *chatbot* utilizando RAG y sin RAG. Estas pruebas permiten ver las diferencias en tiempos de respuesta para el usuario final.

Para el experimento, se han utilizado las consultas mencionadas anteriormente. Las pruebas se han llevado a cabo de la siguiente manera:

- **Sistema con RAG:** El usuario introduce una pregunta corta o poco detallada. El sistema, de forma automática, recupera los fragmentos de contexto más relevantes y construye el *prompt* para el modelo de lenguaje, que genera la respuesta final.
- **Sistema sin RAG:** El usuario debe escribir un contexto extenso junto con su consulta, para que el modelo disponga de la información necesaria para generar una respuesta adecuada.

Los resultados de las pruebas muestran que, si se mide únicamente el tiempo de procesamiento del sistema, el sistema sin RAG es ligeramente superior en cuanto a velocidad de generación de la respuesta. Esto se debe a que, al no realizarse búsquedas en la base de datos de vectores, el flujo se simplifica y se elimina el paso de recuperación de contexto.

Sin embargo, esta mejora de tiempo se contrarresta con el tiempo adicional que el usuario debe invertir en redactar manualmente el contexto necesario en cada consulta. En la práctica, el sistema con RAG resulta más eficiente desde el punto de vista del usuario, ya que automatiza la recuperación de la información relevante.

Estos resultados muestran que la integración de RAG no solo mejora la precisión y contextualización de las respuestas, sino que también optimiza el proceso, reduciendo la carga de trabajo para el usuario y el tiempo para obtener una solución adecuada.

7.3. Pruebas con usuarios finales

Finalmente, se realizaron pruebas con usuarios finales (estudiantes y profesores), que incluyeron:

- Cuestionario de usabilidad **System Usability Scale (SUS)**.
- Cuestionario de lealtad **Net Promoter Score (NPS)**.
- Cuestionario de carga de trabajo **NASA-RTLX**.
- Cuestionario sobre la utilidad percibida.
- Preguntas abiertas.

En estas pruebas han participado un total de 6 participantes, 3 de ellos profesores relacionados con la asignatura y con la programación Java, y 3 alumnos que han cursado la asignatura de Fundamentos de Programación. Cada uno completó una serie de tareas dependiendo de su rol y, posteriormente, respondió a los cuestionarios.

Tareas

Las tareas a realizar por cada uno de los usuarios corresponden a los casos de uso definidos en la fase de análisis:

- **Alumno**
  1. Preguntar al *chatbot*
  2. Valorar respuesta
  3. Crear nuevo chat
- **Profesor**
  1. Preguntar al *chatbot*
  2. Valorar respuesta
  3. Crear nuevo chat
  4. Modificar umbral de recuperación
  5. Subir archivo manualmente
  6. Eliminar archivo
  7. Actualizar archivos desde Moodle

7.3.1. System Usability Scale (SUS)

El *System Usability Scale (SUS)* es un cuestionario validado que permite medir la usabilidad de un sistema de forma rápida. Fue desarrollado por John Brooke en 1986 [5] y consta de 10 enunciados que el usuario debe valorar en una escala de *Likert* [7] de 1 (totalmente en desacuerdo) a 5 (totalmente de acuerdo). A partir de las respuestas se obtiene una puntuación entre 0 y 100, donde valores superiores a 68 suelen considerarse satisfactorios.

A continuación, se presenta el cuestionario SUS con los 10 enunciados que los participantes deben valorar:

1. Creo que me gustaría usar este sistema con frecuencia.
2. Encontré el sistema innecesariamente complejo.
3. Pensé que el sistema era fácil de usar.
4. Creo que necesitaría la ayuda de una persona con conocimientos técnicos para poder utilizar este sistema.
5. Las diversas funciones del sistema están bien integradas.
6. Encontré mucha inconsistencia en el sistema.
7. Creo que la mayoría de las personas aprenderían a usar este sistema muy rápidamente.
8. Encontré el sistema muy complicado de usar.
9. Me sentí muy seguro usando este sistema.
10. Necesité aprender muchas cosas antes de poder comenzar a utilizar el sistema.

En la Tabla 7.3 se muestra la puntuación SUS obtenida por cada uno de los participantes, junto a su rol, y en la Tabla 7.4 la media obtenida en función del rol y la media global.

Usuario	**Rol**	**Puntuación SUS**
Usuario 1	Profesor	90.0
Usuario 2	Profesor	70.0
Usuario 3	Profesor	82.5
Usuario 4	Alumno	92.5
Usuario 5	Alumno	92.5
Usuario 6	Alumno	92.5

Tabla 7.3: Puntuación SUS de cada usuario

Grupo	**Media SUS**
Profesores	80.83
Alumnos	92.5
Global	86.67

Tabla 7.4: Media de puntuaciones SUS por rol y global

Las puntuaciones medias obtenidas en el cuestionario SUS son bastante elevadas: 80.83 para los profesores, 92.5 para los alumnos y 86.67 como media global. Una puntuación superior a 68 se considera una usabilidad adecuada en el cuestionario SUS, por lo que estos resultados indican que tanto profesores como alumnos han encontrado el sistema cómodo y fácil de usar. Concretamente, la puntuación obtenida corresponde al nivel A+, el nivel más alto de satisfacción [13]. La puntuación de los profesores es inferior al tener que realizar más tareas y más complejas que los alumnos. Aun así, se alcanza el nivel más alto en cuanto a usabilidad en el sistema.

7.3.2. Net Promoter Score (NPS)

El ***Net Promoter Score (NPS)*** es una métrica ampliamente utilizada para evaluar la lealtad de los usuarios hacia un sistema y el potencial de adopción de una nueva herramienta [10]. Se basa en una única pregunta directa:

> *"¿Qué probabilidad hay de que recomiendes este sistema a un compañero?"*

Los participantes deben responder utilizando una escala de 0 a 10, donde:

- **0–6**: Detractores (usuarios insatisfechos o críticos).
- **7–8**: Pasivos (usuarios satisfechos pero no entusiastas).
- **9–10**: Promotores (usuarios entusiastas que recomendarían activamente el sistema).

La puntuación final del NPS se calcula como:

$$NPS = \%Promotores - \%Detractores$$

El resultado puede oscilar entre –100 y +100, donde un valor positivo indica una percepción favorable del sistema. Un NPS superior a 0 se considera aceptable, superior a 30 es bueno y superior a 50 es excelente. Esta métrica permite estimar de forma sencilla el potencial de adopción del sistema evaluado.

Usuario	**Respuesta NPS**
Usuario 1	10
Usuario 2	9
Usuario 3	10
Usuario 4	9
Usuario 5	10
Usuario 6	9

Tabla 7.5: Respuestas individuales del cuestionario NPS

En este caso, todas las respuestas recogidas fueron 9 o 10, lo que significa que todos los usuarios se consideran promotores según la metodología estándar del NPS. Por tanto, el valor NPS final obtenido es de 100, el máximo posible en la escala. Este resultado indica un nivel de satisfacción y recomendación inmejorable, ya que todos los participantes, tanto alumnos como profesores, recomendarían el sistema a otros compañeros.

7.3.3. NASA-RTLX

Para evaluar la carga de trabajo percibida durante el uso del sistema, se ha utilizado el cuestionario **NASA-RTLX (*Raw Task Load Index*)**, que es una versión simplificada del método NASA-TLX empleado para medir el esfuerzo que requieren ciertas tareas. Los resultados del RTLX no parecen verse afectados a pesar de su simplificación [22]. En este caso, el cuestionario solo se ha hecho al grupo de profesores, ya que nos interesa conocer la carga de trabajo que supone la configuración del *chatbot*, al ser un factor importante para la adopción de la herramienta.

En este cuestionario se miden 6 factores, cada uno puntuado en una escala de 0 a 100, donde valores más altos indican mayor carga de trabajo, excepto en el caso del rendimiento, que suele interpretarse de manera inversa.

Los seis factores evaluados son:

- **Carga mental**: esfuerzo mental requerido.
- **Carga física**: esfuerzo físico requerido.
- **Carga temporal**: presión de tiempo.
- **Rendimiento percibido**: percepción del grado de éxito en la tarea.
- **Esfuerzo general**: nivel total de esfuerzo necesario.
- **Frustración**: nivel de estrés, irritación o incomodidad durante las tareas.

En este trabajo se ha utilizado la versión simplificada del cuestionario porque es más rápido y fácil de responder, ya que no es necesario ponderar los factores como en el método original. En este proyecto, al tratarse de tareas muy simples, es suficiente para obtener una valoración útil por parte de los usuarios.

Cada factor se ha valorado en una escala de 0 a 20 y, para unificar las puntuaciones con la escala estándar (0 a 100), cada respuesta se ha multiplicado por 5. Para el cálculo final, se ha invertido la puntuación del factor rendimiento (restando su valor a 100), ya que en este caso una mayor puntuación indica menor carga. La media de los seis valores resultantes corresponde al índice NASA-RTLX global para cada usuario. Los resultados obtenidos se muestran en la Tabla 7.6.

Profesor	Índice NASA-RTLX
Profesor 1	15.83
Profesor 2	24.17
Profesor 3	27.50

Tabla 7.6: Índice NASA-RTLX obtenido por los profesores

Las puntuaciones obtenidas en el índice NASA-RTLX para los tres profesores son bastante bajas, todas por debajo de 30 sobre 100. Esto indica que, en general, los profesores han percibido una carga de trabajo baja al utilizar el sistema, lo que indica que la aplicación resulta cómoda y no supone un esfuerzo excesivo para los profesores en las tareas evaluadas, aunque hay posibilidad de mejora, pues la puntuación se sitúa en el segundo cuartil.

7.3.4. Cuestionario de utilidad percibida

Además de los cuestionarios estandarizados, se diseñó un cuestionario propio para medir la utilidad percibida del sistema por parte de los usuarios. Este cuestionario utiliza una escala de 1 a 5, donde 1 indica que no estás de acuerdo con el enunciado y 5 que estás totalmente de acuerdo.

Los enunciados que conforman este cuestionario son:

1. *Las respuestas dadas por la aplicación me parecen adecuadas a mis preguntas.*
2. *Las respuestas dadas por la aplicación me parecen lo suficientemente rápidas.*
3. *Las explicaciones ofrecidas por la aplicación son comprensibles.*
4. *Creo que la información proporcionada por el chatbot puede ser de utilidad para estudiantes de programación.*
5. *Me ha parecido más eficaz que buscar en Internet.*

Para analizar los resultados, se calcula la media de las puntuaciones obtenidas. Este valor da una visión general sobre cómo ha sido valorada la utilidad del sistema y puede servir para comparar con futuras pruebas que se hagan en versiones posteriores del sistema.

Las puntuaciones individuales recogidas en el cuestionario se muestran en la Tabla 7.7.

Tabla 7.7: Resultados del cuestionario de utilidad percibida

Usuario	**Puntuación de utilidad**
Usuario 1	4.8
Usuario 2	4.0
Usuario 3	3.6
Usuario 4	4.2
Usuario 5	3.4
Usuario 6	3.8

La media de las puntuaciones es **3.97** sobre 5 y la desviación típica es aproximadamente **0.45**, lo que refleja una percepción positiva respecto a la utilidad del sistema. Estos resultados muestran que la mayoría de los usuarios han encontrado la herramienta útil para el contexto en el que va a ser utilizada.

7.3.5. Valoraciones y comentarios de los usuarios

Para completar la evaluación, se realizaron preguntas abiertas en las que los usuarios pueden mostrar sus opiniones y sugerencias sobre el sistema, lo que ayuda a identificar posibles mejoras no encontradas en los cuestionarios y en las pruebas realizadas por el desarrollador.

Las principales aportaciones de los usuarios son:

- Incorporar *tooltips* o mensajes de ayuda para explicar el funcionamiento de los diferentes elementos de configuración, entre los cuáles está el umbral de recuperación de documentos, con el cual varios profesores indicaron que no entendían bien su significado.
- Mejorar la visibilidad y el diseño del botón de *like/dislike* de las respuestas del modelo, ya que algunos usuarios, al realizar la tarea de valorar una respuesta, no fueron capaces de encontrarlo rápidamente.
- Indicar más claramente que es necesario pulsar el botón de guardar tras la subida de un archivo para almacenarlo. Algunos profesores pensaron que simplemente subiendo el archivo ya estaba almacenado, pero para eso es necesario pulsar el botón.
- Hacer el menú desplegable más visible para seleccionar los archivos a eliminar, pues uno de los profesores sugirió esta mejora.
- Permitir la creación de carpetas para clasificar y organizar los documentos, como recomendación de otro de los profesores.
- Ofrecer la funcionalidad de subir archivos a los alumnos para hacer consultas (no para incluirlos en el flujo RAG).
- Es preferible que el *chatbot* diga que no sabe responder antes que responder de forma incorrecta. Varios profesores indicaron esta preferencia.

Las sugerencias destacan algunos aspectos mejorables, centrados principalmente en la interfaz. También hay que destacar comentarios sobre las respuestas del *chatbot*, donde algunos profesores prefieren que no se obtenga una respuesta antes que recibir una respuesta errónea para la asignatura.

Debido a restricciones de tiempo en el desarrollo del proyecto, algunas de estas sugerencias no han podido implementarse en la versión actual, pero han sido propuestas para ser incluidas en futuras actualizaciones del sistema.

Capítulo 8

Conclusiones

En este capítulo se resumen las principales conclusiones alcanzadas tras el desarrollo del sistema, así como las limitaciones encontradas durante su implementación y las posibles líneas de trabajo que podrían abordarse en el futuro para ampliar y perfeccionar el *chatbot*.

8.1. Resumen de conclusiones

Tras la realización de este Trabajo de Fin de Grado se han obtenido los siguientes resultados y conclusiones:

I. La combinación de un modelo LLM local con un sistema de Recuperación Aumentada por Generación (RAG) permite generar respuestas más contextualizadas, fiables y ajustadas a los contenidos de una asignatura específica.

II. La aplicación cumple con los requisitos definidos en la fase de análisis y ha sido validada mediante pruebas con usuarios reales, obteniendo una buena puntuación tanto en usabilidad como en utilidad y potencial de adopción.

III. La integración con Moodle mediante LTI supone una solución muy interesante para la autenticación de los usuarios, evitando tener que manejar datos de inicio de sesión y haciendo la herramienta fácilmente accesible a través del curso de Moodle de la asignatura.

IV. Se ha logrado recopilar, estructurar e indexar material relevante de la asignatura (apuntes, ejercicios resueltos, etc.), que sirve como base de conocimiento para el sistema RAG y permite personalizar las respuestas del modelo.

V. Se ha llevado a cabo un estudio de los principales parámetros que influyen en el funcionamiento del sistema RAG, como el umbral de similitud y el número de documentos recuperados, identificando las combinaciones que ofrecen mejores resultados para esta aplicación.

VI. Este trabajo es una base sólida para su posible extensión a otras asignaturas del grado, abriendo así una línea de trabajo futura para adaptarlo a diferentes contextos educativos.

8.2. Limitaciones del estudio

A pesar de los buenos resultados obtenidos, el sistema presenta una serie de limitaciones por el entorno, el alcance del proyecto y los recursos disponibles.

Una de las principales limitaciones ha sido el **hardware disponible**. Al ejecutarse el sistema en local, con recursos muy limitados, no ha sido posible utilizar modelos de lenguaje más potentes que podrían haber ofrecido respuestas más completas o precisas. Esto también limita el uso simultáneo por múltiples usuarios, ya que el procesamiento se realiza todo en la misma máquina.

Otro aspecto relevante ha sido la **limitación del corpus de documentos**. Se ha trabajado con materiales reales de la asignatura de Fundamentos de Programación, pero el número de archivos era reducido y algunos no tenían la estructura adecuada, teniendo poco texto, lo que dificulta su recuperación por parte del sistema RAG. Una base de documentos más grande y mejor organizada permitiría una recuperación de contexto más precisa.

También hay que destacar las **restricciones de tiempo** del proyecto. Como en todos los proyectos, el tiempo es limitado y muchas de las ideas que había pensadas, como hacer una interfaz más personalizable o realizar más pruebas con usuarios, no se han podido llevar a cabo por cuestiones de tiempo propias del desarrollo de un TFG.

Además, el sistema actual no tiene una gestión de sesiones ni de usuarios. Aunque los chats se almacenan en la base de datos, no se asocian de forma explícita a un usuario concreto, lo que limita funcionalidades como el historial personalizado o el seguimiento de uso. Este punto habría que adaptarlo, como trabajo futuro, para que la aplicación cumpla con la GDPR.

Por último, aunque el sistema ha sido probado en condiciones reales, las pruebas se han realizado con un número reducido de participantes. Lo ideal habría sido hacer pruebas con alumnos de la asignatura de Fundamentos de Programación, pero al ser una asignatura del primer cuatrimestre, esto ha resultado imposible.

8.3. Líneas de trabajo futuras

A partir de los comentarios recogidos durante las pruebas y de las ideas surgidas durante el desarrollo, se han ideado varias posibles líneas de trabajo que permitirían seguir mejorando el sistema y ampliarlo a otros contextos.

Una de las mejoras más demandadas por los usuarios ha sido en temas de usabilidad de la interfaz. La idea sería prescindir del *framework* Gradio, que supone una solución útil y

rápida, y crear una interfaz personalizada (por ejemplo, con React) que permita incluir más funcionalidades y adaptarla más a las necesidades y sugerencias de los usuarios.

Otra línea clara de mejora es la ampliación del corpus de documentos. Cuantos más materiales de calidad haya (apuntes, ejercicios resueltos, enunciados, etc.), mejor funcionará el sistema RAG. Además, habría que establecer una estructura uniforme para todos los tipos de documentos, por ejemplo, que todos los ejercicios tengan la misma estructura de enunciado-solución, que los apuntes estén organizados en secciones, etc.

A nivel más técnico, sería interesante experimentar con otros modelos de *embeddings* más específicos para programación o aplicar *re-ranking* para mejorar la recuperación de documentos. El *re-ranking* consiste en hacer una lista de los documentos más relevantes y pasársela al LLM para que decida cuál de esos documentos es el más importante.

Finalmente, la idea sería adaptar el sistema para funcionar como herramienta general para otras asignaturas. En cada asignatura se incluirían los documentos existentes en el curso de Moodle de dicha asignatura, para lo cual habría que incluir una entidad 'Curso' en la base de datos que permita distinguir entre distintos cursos.

Todas estas mejoras harían que el sistema fuera mucho más completo y útil tanto para profesores como para alumnos, haciendo que la experiencia sea más satisfactoria y facilite tanto el aprendizaje como la docencia.

Bibliografía

[1] Meta AI. Llama 3.2: Revolutionizing edge AI and vision with open, customizable models. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/, 2024. Accessed: 2025-5-23.

[2] Carlos Alario-Hoyos, Rebiha Kemcha, Carlos Delgado Kloos, Patricia Callejo, Iria Estévez-Ayres, David Santín-Cristóbal, Francisco Cruz-Argudo, and José Luis López-Sánchez. Tailoring Your Code Companion: Leveraging LLMs and RAG to Develop a Chatbot to Support Students in a Programming Course. https://ieeexplore.ieee.org/abstract/document/10834365, 2024. Accessed: 2025-2-28.

[3] AWS. ¿Qué es RAG? https://aws.amazon.com/es/what-is/retrieval-augmented-generation/. Accessed: 2025-3-5.

[4] Amazon AWS. ¿Qué es LangChain? https://aws.amazon.com/es/what-is/langchain/. Accessed: 2025-4-24.

[5] John Brooke. SUS: A quick and dirty usability scale. https://www.researchgate.net/publication/228593520_SUS_A_quick_and_dirty_usability_scale, 1995. Accessed: 2025-5-23.

[6] DeepSeek. Deepseek-v3. https://www.deepseek.com/. Accessed: 2025-3-11.

[7] Guadalupe Gante, Wadi Elim Gonzalez, Jaime Bautista-Ortega, Escobar Castillo, and Alberto Fernández. Escala de Likert: Una alternativa para elaborar e interpretar un instrumento de percepción social. https://www.researchgate.net/publication/361533522_Escala_de_Likert_Una_alternativa_para_elaborar_e_interpretar_un_instrumento_de_percepcion_social, 2020. Accessed: 2025-5-23.

[8] Glassdoor. Sueldos para el puesto de Full Stack Developer en España. https://www.glassdoor.es/Sueldos/full-stack-developer-sueldo-SRCH_KO0,20.htm. Accessed: 2025-4-24.

[9] Google. Gemini. https://gemini.google.com//. Accessed: 2025-3-11.

[10] S. G. Hart. NASA-task load index (NASA-TLX); 20 years later. https://hbr.org/2003/12/the-one-number-you-need-to-grow, 2003.

[11] Thomas Pyka George Prenosil Kuangyu Shi Axel Rominger Ali Afshar-Oromieh Ian L.Alberts, Lorenzo Mercolli. Large language models (LLM) and ChatGPT: what will the impact on nuclear medicine be? https://link.springer.com/article/10.1007/s00259-023-06172-w, 2023. Accessed: 2025-4-29.

[12] IBM. RAG en comparación con fine-tuning. https://www.ibm.com/es-es/think/topics/rag-vs-fine-tuning. Accessed: 2025-3-5.

[13] Ph.D Jeff Sauro PhD James R., Lewis. Item Benchmarks for the System Usability Scale. https://uxpajournal.org/item-benchmarks-system-usability-scale-sus/.

[14] Minder Chen Jay F Nunamaker Jr and Titus DM Purdin. Systems development in information systems research. journal of management information systems. https://doi.org/10.1080/07421222.1990.11517898, 1990.

[15] Jonathan Álvarez Ariza, Milena Benítez Restrepo, Carola Hernández Hernández. Generative AI in Engineering and Computing Education: A Scoping Review of Empirical Studies and Educational Practices. https://www.researchgate.net/publication/388974860_Generative_AI_in_Engineering_and_Computing_Education_A_Scoping_Review_of_Empirical_Studies_and_Educational_Practices, 2025. Accessed: 2025-3-5.

[16] MetaAI. Llama. https://www.llama.com/. Accessed: 2025-3-11.

[17] Microsoft. Copilot. https://copilot.microsoft.com/. Accessed: 2025-3-11.

[18] Miguel A. Martínez-Prieto, Jorge Silvestre, Anibal Bregon, Patricia Baz, Clara Gándara-González, Paula Mielgo, Irene Peñas. Una metodología basada en prácticas ágiles para la realización de Trabajos Fin de Grado. Accessed: 2025-3-5.

[19] Moodle. What is LTI and how it can improve your learning ecosystem. https://moodle.com/news/what-is-lti-and-how-it-can-improve-your-learning-ecosystem/.

[20] OpenAI. ChatGPT. https://chatgpt.com/. Accessed: 2025-3-5.

[21] Visual Paradigm. What is Activity Diagram? https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/.

[22] Frederick F Reichheld. The one number you need to grow., 2006.

[23] Soobia Saeed, NZ Jhanjhi, Mehmood Naqvi, and Mamoona Humayun. Analysis of software development methodologies, 2019.

[24] Sutherland J. Schwaber, K. Scrum software development process. proceedings of the 10th annual conference on object-oriented programming systems, languages, and applications, 1995.

[25] Martin Maciol Fabian Ridder Marius Schmitz Jennifer Spanagel Jens Wienkamp Christopher Burgahn Sebastian Kahl, Felix Löffler and Malte Schilling. Enhancing AI Tutoring in Robotics Education: Evaluating the Effect of Retrieval-Augmented Generation and Fine-Tuning on Large Language Models . https://www.uni-muenster.de/imperia/md/content/angewandteinformatik/aktivitaeten/publikationen/enhancing_ai_tutoring_in_robotics_education_-_2024.pdf, 2024. Accessed: 2025-3-5.

[26] Study.com. Productive teaching tool or innovative cheating? https://study.com/resources/perceptions-of-chatgpt-in-schools.

[27] Telefónica. ¿qué es la inteligencia artificial generativa? https://www.telefonica.com/es/sala-comunicacion/blog/inteligencia-artificial-generativa/. Accessed: 2025-4-29.

[28] Google Trends. Retrieval augmented generation - trend. https://trends.google.es/trends/explore?date=today%205-y&q=Retrieval-augmented%20generation&hl=es.

[29] Yi Wu. Integrating Generative AI in Education: How ChatGPT Brings Challenges for Future Learning and Teaching. www.pioneerpublisher.com/jare, 2023. Accessed: 2025-4-29.

[30] Xataka. Qué es Ollama y cómo usarlo para instalar en tu ordenador modelos de inteligencia artificial. https://www.xataka.com/basics/que-ollama-como-usarlo-para-instalar-tu-ordenador-modelos-inteligencia-artificial-como-llama-deepseek#:~:text=Ollama%20es%20un%20programa%20que,una%20IA%20que%20quieras%20utilizar. Accessed: 2025-4-24.

Apéndice A

Manual de despliegue

Este apartado describe cómo desplegar el sistema en un entorno local, incluyendo la ejecución de los componentes, la integración con Moodle y las restricciones de acceso al código.

A.1. Dependencias necesarias

Antes de ejecutar la aplicación, es necesario instalar las siguientes bibliotecas de Python. Se recomienda hacerlo en un entorno virtual.

```
pip install gradio flask PyPDF2 psycopg2 requests
pip install nltk chromadb langchain
```

Adicionalmente, para la funcionalidad de procesamiento de texto, es necesario descargar los recursos de `nltk` en la primera ejecución del sistema. Esto se hace automáticamente mediante el siguiente bloque presente en el código:

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
```

A.2. Ejecución de los componentes

El sistema está dividido en tres módulos independientes que deben ejecutarse en paralelo. Cada uno corresponde a una parte funcional del sistema:

- **Interfaz del profesor:**

  Comando:

  ```
  python interfaz_profesor.py
  ```

  Ofrece acceso completo al chatbot y al gestor de archivos. Disponible en el puerto **7860**.

- **Interfaz del alumno:**

  Comando:

  ```
  python interfaz_alumno.py
  ```

  Proporciona acceso exclusivo al chatbot. Disponible en el puerto **7861**.

- **Servidor LTI (Flask):**

  Comando:

  ```
  python server.py
  ```

  Encargado de recibir la autenticación desde Moodle y redirigir al usuario a su interfaz correspondiente. Disponible en el puerto **5000**.

A.3. Integración con Moodle

Para permitir el acceso desde Moodle, se debe añadir la herramienta como un recurso externo dentro del curso:

1. Acceder al curso Moodle correspondiente.
2. Activar el **modo de edición**.
3. Pulsar en **Añadir una actividad o un recurso**.
4. Seleccionar la opción **Herramienta externa**.
5. Rellenar los campos obligatorios:
   - **Nombre:** por ejemplo, "Chatbot Fundamentos de Programación".
   - Pulsar en **Mostrar más...**.
   - En el campo **Tool URL**, introducir:
     `http://127.0.0.1:5000/`

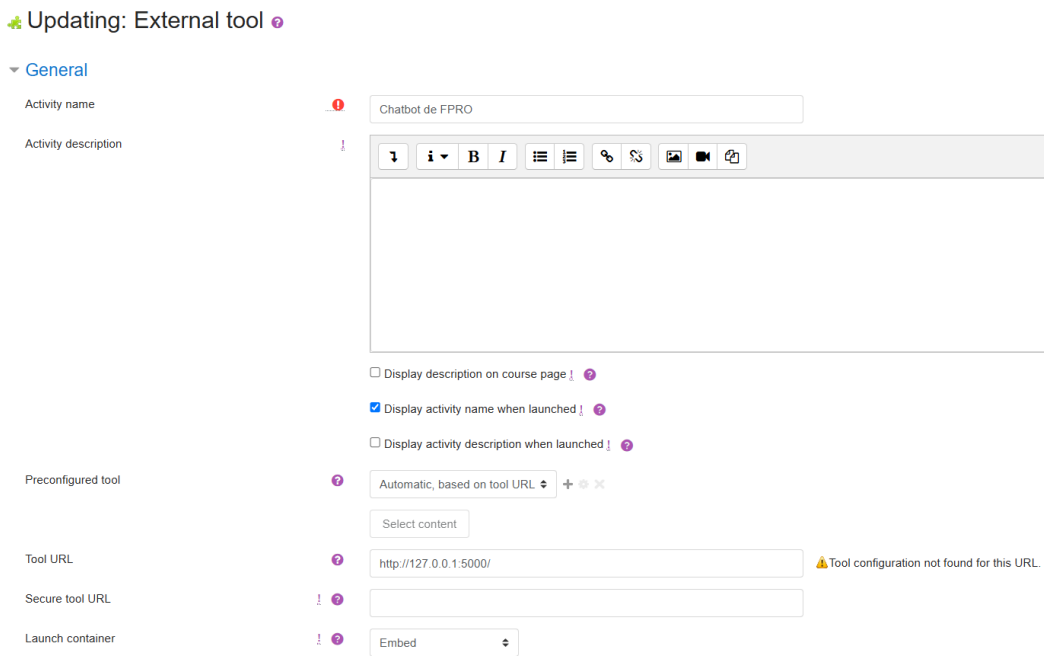

Figura A.1: Configuración de parámetros en Moodle

6. En **Launch container**, seleccionar la opción deseada. Se recomienda **embebido** o **en una pestaña nueva**.
7. Guardar los cambios.

En la Figura A.1 se muestra la configuración de parámetros que debe introducirse.

Una vez completado este proceso, los estudiantes y profesores podrán acceder a la herramienta desde Moodle. El sistema detectará su rol automáticamente y redirigirá a la interfaz correspondiente.

A.4. Accesos

Por motivos de seguridad, el gestor de archivos está restringido únicamente al perfil de **profesor**. Los alumnos pueden interactuar con el chatbot y valorar respuestas, pero no pueden hacer gestiones con documentos, ni para acceder a las rutas del backend. Esta distinción de roles está gestionada desde el servidor mediante la información proporcionada por Moodle en cada inicio de sesión.

Por otro lado, el acceso al código fuente no es público, solo será accesible para los usuarios que se encarguen de desplegar la aplicación en un futuro.

Apéndice B

Manual de uso

Este apartado explica el funcionamiento de la aplicación desde la perspectiva de los dos tipos de usuario: alumno y profesor. Se detalla cómo acceder al sistema, qué funcionalidades ofrece cada perfil y cómo utilizarlas correctamente.

B.1. Acceso al sistema

Para acceder a la aplicación, el usuario debe hacerlo desde el curso correspondiente en Moodle. Al hacer clic en el enlace habilitado, se abrirá la interfaz de la herramienta adaptada a su perfil (estudiante o profesor).

Dependiendo de cómo se haya configurado la herramienta en Moodle, esta aparecerá embebida en el propio Moodle o se abrirá una nueva pestaña. En las Figuras B.1 y B.2 se muestra cómo se ve de forma embebida y en una pestaña nueva, respectivamente.

B.2. Interfaz para estudiantes

Una vez dentro de la aplicación, los estudiantes acceden directamente a la pestaña **Chatbot**, donde pueden interactuar con el modelo contextualizado con RAG.

Chatbot

- El usuario debe escribir su pregunta en el cuadro de texto.
- El *chatbot* responderá con una explicación o un ejemplo de código adaptado al contexto de la asignatura.

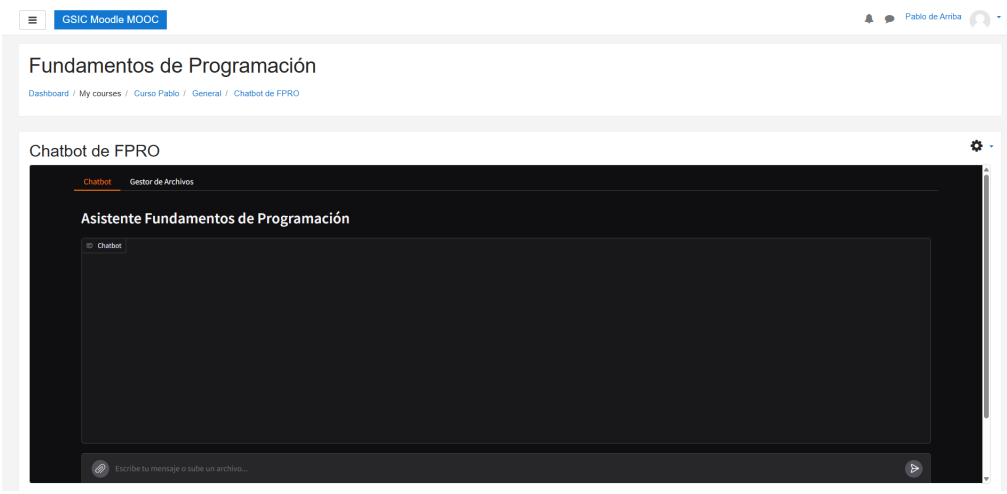

Figura B.1: Vista embebida del *chatbot* en Moodle

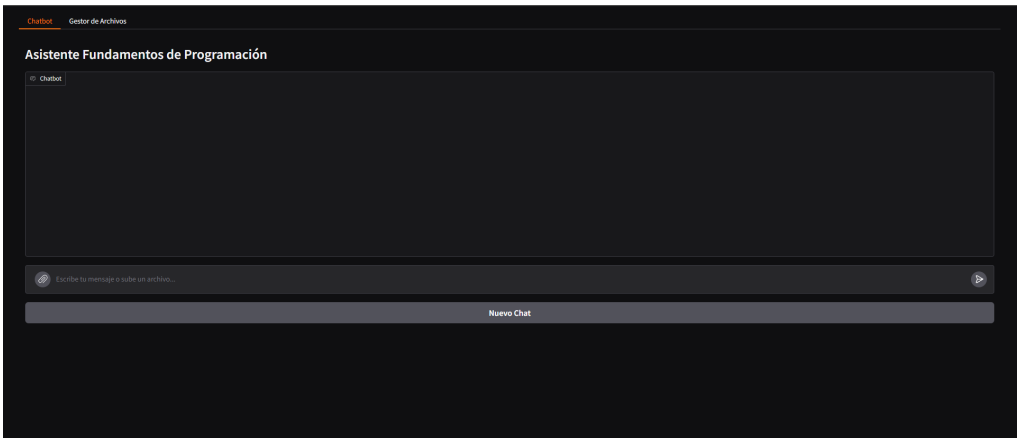

Figura B.2: Vista del *chatbot* en una nueva ventana en Moodle

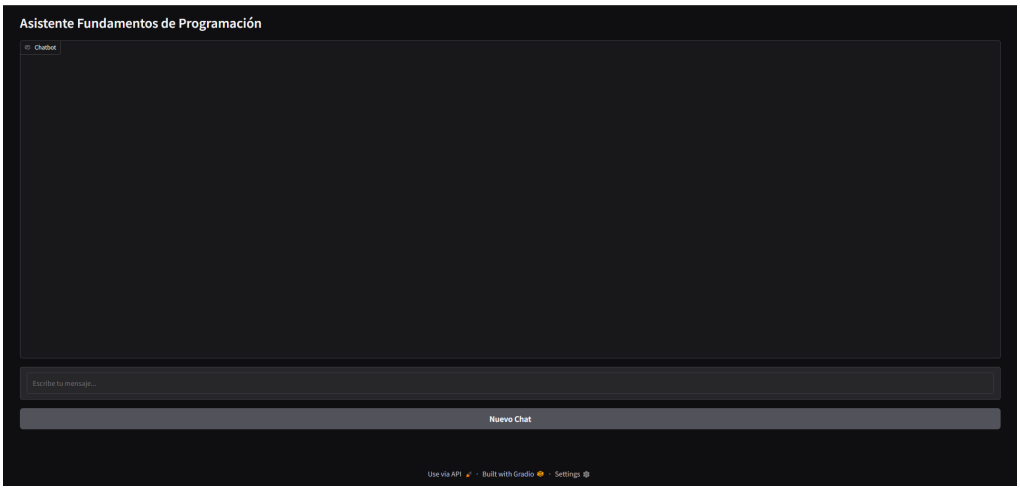

Figura B.3: Interfaz del alumno

- Las respuestas pueden valorarse como útiles o no útiles mediante los botones situados en la parte inferior del mensaje de respuesta.
- Se puede comenzar una conversación nueva en cualquier momento pulsando el botón **Nuevo Chat**.

La Figura B.3 corresponde a la interfaz que verá el alumno, desde la cual solo se tiene acceso al *chatbot* y nunca al gestor de archivos.

B.3. Interfaz para profesores

Los profesores acceden a una versión ampliada de la aplicación, con dos pestañas principales: **Chatbot** y **Gestor de Archivos** (Figura B.4).

Chatbot

El funcionamiento es idéntico al del alumno. Se recomienda utilizarlo para comprobar que las respuestas generadas se ajustan al nivel de la asignatura.

Gestor de Archivos

Esta pestaña permite al profesor gestionar todo lo relacionado con el flujo RAG. Ofrece las siguientes funcionalidades:

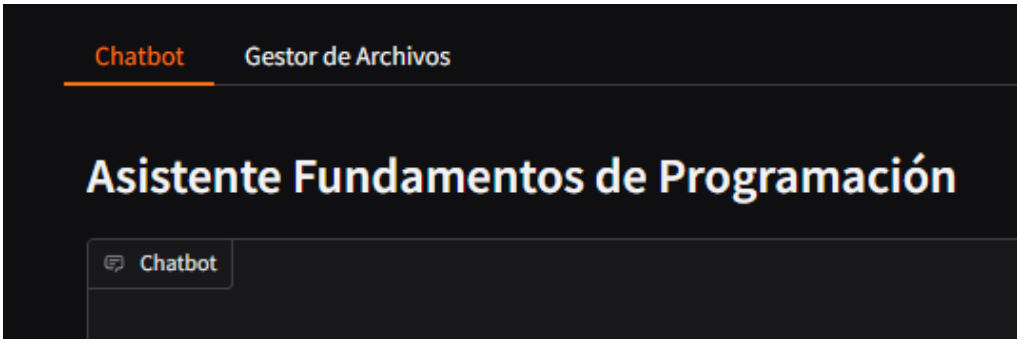

Figura B.4: Pestañas en la interfaz de profesor

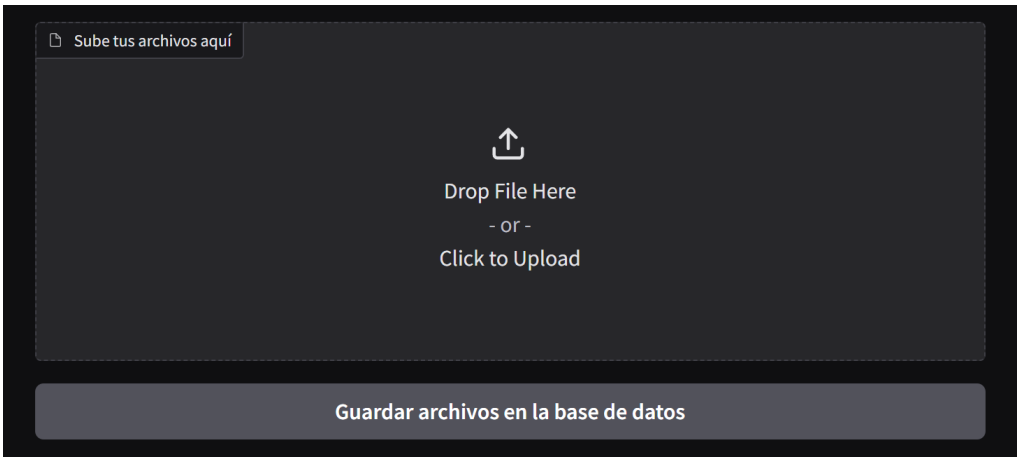

Figura B.5: Componente de la interfaz para subir archivos

- **Subir archivos:** selecciona uno o más documentos PDF y pulsa el botón **Guardar archivos en la base de datos**. El sistema mostrará un listado actualizado.
- **Eliminar archivos:** selecciona un documento del desplegable y pulsa **Eliminar documento** para retirarlo del sistema. Los archivos contenidos en el sistema se muestran en la lista que se ve en la Figura B.6. Es posible buscar documentos en el menú desplegable escribiendo su nombre (Figura B.7).
- **Actualizar desde Moodle:** al pulsar **Actualizar documentos desde Moodle**, se descargan los materiales del curso y se actualiza el corpus automáticamente. Una vez actualizado, se muestra un mensaje informativo. Tanto el botón como el mensaje de informativo se observan en la Figura B.8.
- **Umbral de similitud:** ajusta el control deslizante para modificar el nivel de exigencia en la recuperación de contexto. Este valor corresponde a la similitud requerida entre la consulta del usuario y el documento recuperado. Si el umbral es alto, será necesaria una mayor similitud entre consulta y documento, por lo que es menos probable que se recupere algún archivo relevante (Figura B.9).

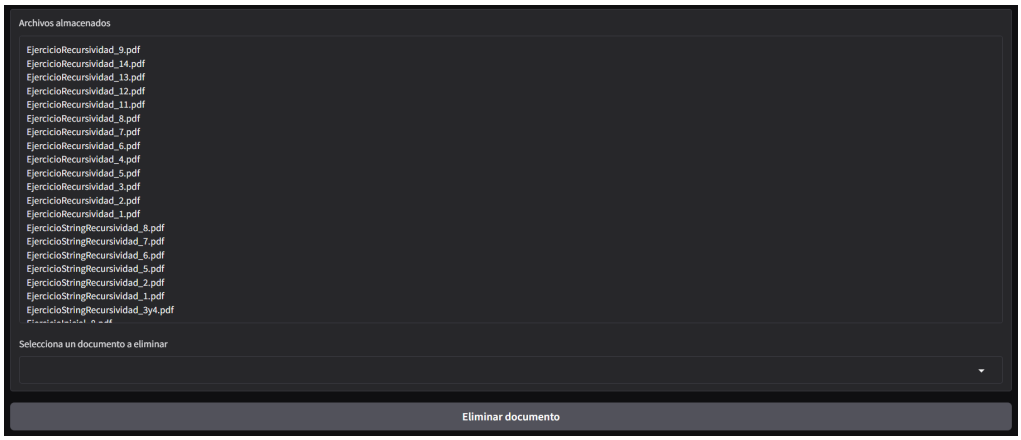

Figura B.6: Lista de archivos contenidos en el sistema

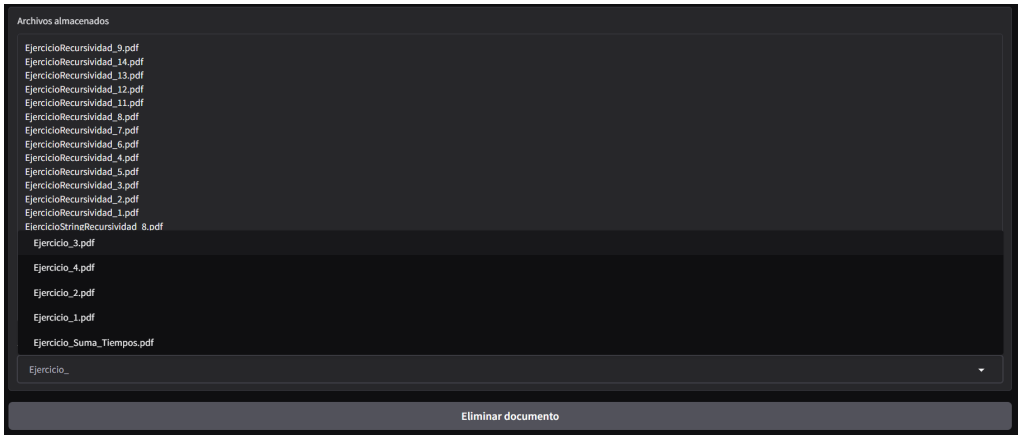

Figura B.7: Menú desplegable para seleccionar archivo a eliminar

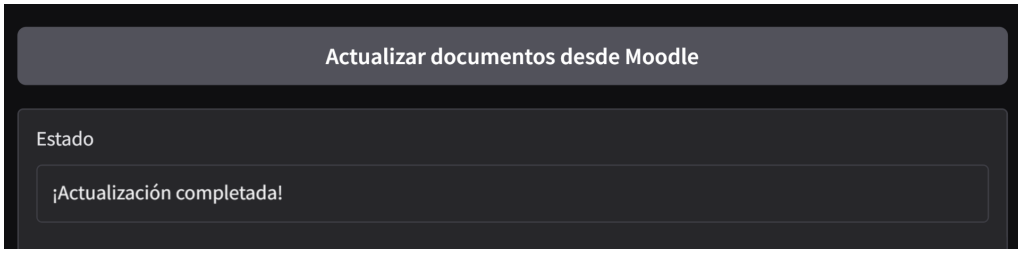

Figura B.8: Botón para actualizar el corpus de archivos con Moodle

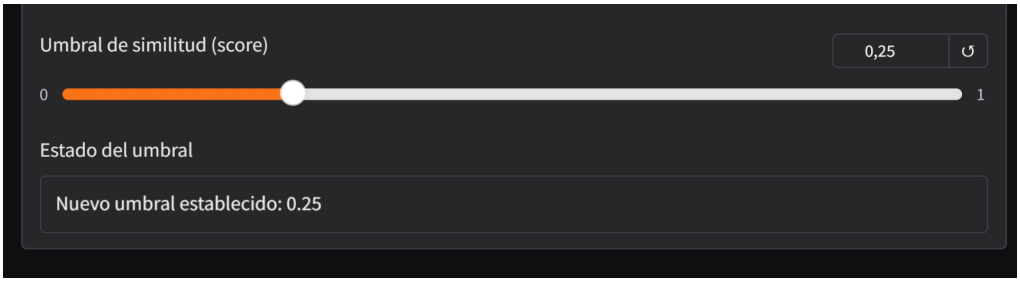

Figura B.9: *Slider* para modificar umbral de recuperación

B.4. Preguntas frecuentes

¿Qué ocurre si no hay información suficiente para una respuesta?
El *chatbot* intentará ofrecer una respuesta general, pero puede indicar que no dispone de contexto suficiente.

¿Qué tipo de archivos se pueden subir?
Solo se admiten documentos en formato PDF.

¿Quién puede subir o eliminar archivos?
Solo los profesores tienen acceso a la pestaña de gestión de archivos.

Apéndice C

Consentimiento informado

Consentimiento informado para participantes en los tests de usabilidad del Trabajo de Fin de Grado:

"Creación de una IA optimizada para la asignatura de Fundamentos de Programación" (Pablo de Arriba Mendizábal, Universidad de Valladolid)

Estimado/a participante:

En primer lugar, queremos agradecerle su participación en esta evaluación. Antes de comenzar, queremos informarle sobre algunos aspectos importantes relacionados con este trabajo y las tareas que debe realizar.

Esta evaluación forma parte del Trabajo de Fin de Grado de Pablo de Arriba Mendizábal, cuyo objetivo es desarrollar una herramienta basada en inteligencia artificial que sirva como asistente conversacional (*chatbot*) para apoyar el aprendizaje en la asignatura de Fundamentos de Programación del Grado en Ingeniería Informática de la Universidad de Valladolid. La finalidad de esta evaluación es recoger datos, opiniones y experiencias de los usuarios en relación al uso de la herramienta, con el objetivo de mejorar el diseño y la funcionalidad de la misma.

Los datos recopilados durante esta evaluación se utilizarán exclusivamente con fines de investigación académica para el presente proyecto. Todos los datos serán tratados de forma confidencial en dispositivos del autor del trabajo y, posteriormente, publicados de forma anonimizada en repositorios de la Universidad de Valladolid.

Usted tiene derecho a corregir o eliminar cualquier dato personal recopilado, contactando

con el investigador principal. Su participación en esta evaluación es completamente voluntaria y puede abandonarla en cualquier momento sin necesidad de justificar su decisión.

La duración estimada de la evaluación es de aproximadamente 15 minutos. Durante este tiempo, se le pedirá lo siguiente:

- Rellenar un cuestionario inicial para conocer su perfil.
- Leer las explicaciones sobre el sistema evaluado.
- Realizar una serie de tareas prácticas utilizando el *chatbot* desarrollado.
- Completar varios cuestionarios para recoger su opinión y experiencia con la herramienta.

Este formulario incluye todos los cuestionarios mencionados anteriormente. Solo debe seguir las indicaciones y responder a las cuestiones que se requieran.

Gracias por su colaboración.

Procedimiento de consentimiento:
El consentimiento informado para la participación en este estudio se otorga marcando la casilla correspondiente (*checkbox*) incluida en el propio cuestionario de Google Forms. De este modo, se declara haber sido informado/a y aceptar voluntariamente participar en la evaluación descrita.