Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Construyendo un bot usando Prompt Engineering. Un caso de estudio

Building a chatbot using Prompt Engineering. A Case Study

Gabriel Jonay Vera Estévez

La Laguna, 24 de mayo de 2024

Dr. **Casiano Rodríguez León**, profesor Catedrático de Universidad del área de Lenguajes y Sistemas Informáticos, adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

I N F O R M A (N)

Que la presente memoria titulada:

”*Construyendo un bot usando Prompt Engineering. Un caso de estudio*”

ha sido realizada bajo su dirección por Dr. **Casiano Rodríguez León**.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 24 de mayo de 2024

Agradecimientos

A mi tutor Casiano Rodríguez León por el trato respetuoso y el conocimiento que me ha proporcionado.

A mis padres y a mi hermana por brindarme los recursos y ánimos para llegar hasta aquí.

A mis amigos y compañeros de carrera por esas tardes de estudio y risas sin fin.

Y a mi pareja, el pilar fundamental de mi vida, que me ha acompañado tanto en los malos como en los buenos momentos, brindándome ayuda cuando más la he necesitado.

A todos ellos muchas gracias.

Licencia

Resumen

En la actualidad, la integración de la inteligencia artificial (IA) en diversos ámbitos ha revolucionado la interacción entre las personas y la tecnología. Un punto clave en este avance es el desarrollo de chatbots avanzados, diseñados para proporcionar asistencia personalizada en tiempo real. Este proyecto se centra en la creación de un chatbot avanzado que interpreta documentos académicos, específicamente las Guías Docentes y el reglamento de evaluación y calificación de la Universidad de La Laguna. Se emplean conceptos de IA generativa y prompt engineering para lograr respuestas más humanas y precisas. Este chatbot tiene como objetivo simplificar la búsqueda de información sobre asignaturas y procedimientos de evaluación, brindando explicaciones claras y concisas. Al mejorar la accesibilidad y la eficiencia en la obtención de información, se espera mejorar la experiencia educativa y fomentar una comunicación más eficiente entre estudiantes, profesores y personal administrativo en el entorno universitario. La implementación de esta herramienta ofrece beneficios tangibles en términos de accesibilidad, eficiencia y claridad en el ámbito académico.

Palabras clave: Inteligencia artificial (IA), Chatbot, Guías Docentes, Universidad de La Laguna, Interpretación de documentos, IA generativa, Prompt engineering, Experiencia educativa, Comunicación eficiente, Procedimientos de evaluación, Gestión de errores, Interfaz Gráfica

Abstract

Today, the integration of artificial intelligence (AI) in various fields has revolutionised the interaction between people and technology. A key point in this progress is the development of advanced chatbots, designed to provide personalised assistance in real time. This project focuses on the creation of an advanced chatbot that interprets academic documents, specifically the Teaching Guides and the evaluation and grading regulations of the University of La Laguna. Concepts of generative AI and prompt engineering are used to achieve more human and accurate responses. This chatbot aims to simplify the search for information about subjects and assessment procedures by providing clear and concise explanations. By improving accessibility and efficiency in obtaining information, it is expected to enhance the educational experience and foster more efficient communication between students, teachers and administrative staff in the university environment. The implementation of this tool offers tangible benefits in terms of accessibility, efficiency and clarity in the academic environment.

Keywords: Artificial Intelligence (AI), Chatbot, Teaching Guides, University of La Laguna, Document Interpretation, Generative AI, Prompt engineering, Educational experience, Efficient communication, Evaluation procedures, Error management, Graphical interface

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y Plan de Trabajo

1.1. Introducción

Uno de los temas de interés en la actualidad es la integración de la inteligencia artificial en multitud de ámbitos con la intención de mejorar la interacción entre las personas y la tecnología. Debido a esto, el desarrollo de *chatbots* se ha convertido en un punto clave para facilitar la comunicación y proporcionar asistencia personalizada en tiempo real.

En este TFG se plantea la creación de un *chatbot* avanzado con el objetivo de interpretar los distintos documentos donde se recoge la información sobre las diversas asignaturas de la Universidad de La Laguna (Guías Docentes) (1) y el reglamento de evaluación y calificación (2), y brindar información precisa sobre esta a los usuarios.

Para la creación de este *chatbot* se usarán conceptos relacionados con la inteligencia artificial generativa y *prompt engineering* (práctica de diseñar entradas para herramientas de inteligencia artificial que produzcan mejores resultados) (3), gracias a esta combinación, es nuestro objetivo que las respuestas que se obtengan tengan un carácter más humano y cuenten con una buena precisión.

En el caso de estudiantes y profesores, la complejidad de la información académica a menudo puede generar confusiones. La existencia de un *chatbot* capaz de interpretar y explicar de manera clara y concisa los detalles de las asignaturas, así como los procedimientos de evaluación, puede simplificar en gran medida la búsqueda de información. Esto no solo ahorra tiempo, sino que también contribuye a una comprensión más profunda de los procesos académicos, mejorando así la experiencia educativa.

Además, la naturaleza accesible y de respuesta rápida del *chatbot* podría fomentar una comunicación más eficiente entre estudiantes, profesores y personal administrativo. Al aliviar la carga de interpretar documentos extensos, el *chatbot* se convierte en una herramienta valiosa para la comunidad universitaria, facilitando la toma de decisiones informadas y el cumplimiento de los requisitos académicos.

Se destaca que la implementación de esta herramienta en el entorno universitario ofrece beneficios tangibles en términos de accesibilidad, eficiencia y claridad en la obtención de información.

1.2. Actividades a realizar

1.2.1. Tarea 1: Estado del arte, decisiones tecnológicas y formación.

En esta etapa, se llevará a cabo una revisión exhaustiva del estado actual de las herramientas similares disponibles en el mercado. Esto incluirá la investigación de plataformas de agentes conversacionales, sistemas de inteligencia artificial aplicados al procesamiento del lenguaje natural y tecnologías relacionadas.

También se realiza un análisis de desafíos comunes y casos de éxito con el fin de extraer buenas prácticas.

Además, se identificarán y evaluarán las tecnologías más adecuadas para el desarrollo del agente conversacional, considerando factores como la facilidad de implementación, el rendimiento, la escalabilidad y la disponibilidad de recursos de aprendizaje.

1.2.2. Tarea 2: Diseño y prototipado.

En esta fase, se procederá al diseño detallado de la arquitectura del agente conversacional y sus funcionalidades principales. Se llevará a cabo un proceso iterativo de creación de prototipos, donde se desarrollarán versiones preliminares del agente y se realizarán pruebas de concepto. Se utilizarán técnicas de *prompt engineering* para refinar y ajustar las respuestas del agente, asegurando una interacción natural y efectiva con los usuarios.

1.2.3. Tarea 3: Realización y despliegue.

Una vez completado el diseño y prototipado, se procederá a la implementación del agente conversacional en un entorno de desarrollo controlado. Se llevará a cabo la integración de las tecnologías seleccionadas y se realizarán pruebas exhaustivas para garantizar el correcto funcionamiento del agente.

Cuando el agente esté listo, se procederá al despliegue en un entorno de producción controlado, donde se continuará monitorizando y ajustando su rendimiento según sea necesario.

1.2.4. Tarea 4: Pruebas.

En esta etapa, se llevarán a cabo pruebas exhaustivas del agente conversacional utilizando a usuarios reales. Estos utilizarán la plataforma creada y rellenarán una encuesta referente a la experiencia de usuario en cuanto a usabilidad y precisión.

Posteriormente se analizarán los resultados de las pruebas y se realizarán ajustes según los comentarios y sugerencias recibidos. El objetivo es garantizar que el agente sea capaz de manejar una amplia variedad de consultas y proporcionar respuestas precisas y útiles en tiempo real.

1.2.5. Tarea 5: Documentación y elaboración de la memoria.

Finalmente, se llevara a cabo la redacción de la memoria del Trabajo Fin de Grado, la realización del vídeo de presentación del mismo, así como el resumen para su presentación oral.

1.3. Plan de Trabajo

Teniendo en cuenta las tareas descritas anteriormente, se propone la siguiente planificación (Tabla 1.1) para la ejecución del proyecto, en las quince semanas de un cuatrimestre, así como las fechas de las actividades de la asignatura:

Tarea/Actividad	**Fechas**
Tarea 1: Estado del arte, decisiones tecnológicas y formación	(diciembre de 2023)
Tarea 2: Diseño y prototipado	(diciembre de 2023)
Tarea 3: Realización y despliegue	(marzo de 2024)
Tarea 4: Pruebas	(abril de 2024)
Tarea 5: Documentación y elaboración de la memoria	(mayo de 2024)

Tabla 1.1: Plan de trabajo.

Capítulo 2

Estado del arte y decisiones tecnológicas

En este capítulo se abordaran cuestiones tales como: ¿Que técnicas de *prompt engineering* se conocen?, ¿Qué tecnologías son las más utilizadas en el desarrollo de este tipo de proyectos?, ¿Cuáles son los puntos a favor de cada tecnología mencionada?

2.1. Antecedentes y estado del arte

La integración de *chatbots* ha experimentado un crecimiento significativo en los últimos años, impulsado por avances en el procesamiento del lenguaje natural y la inteligencia artificial. Los *chatbots* se utilizan en una variedad de sectores, desde el comercio electrónico hasta la atención al cliente, brindando respuestas instantáneas a usuarios y clientes. Sin embargo, el estado actual de muchos *chatbots* todavía presenta desafíos en términos de comprensión contextual y respuestas personalizadas. A pesar de los avances, existe una oportunidad para desarrollar un *chatbot* más sofisticado que pueda abordar estas limitaciones y proporcionar una experiencia más enriquecedora. En este contexto, este proyecto busca explorar y mejorar las capacidades de los *chatbots*, centrándose en la interpretación de documentos de texto haciendo uso de técnicas de *prompt design* y *prompt engineering*.

En el ámbito de los *chatbots* de propósito general, se han desarrollado modelos que pueden realizar una variedad de tareas, desde responder preguntas generales hasta ofrecer información sobre temas diversos. Estos *chatbots* a menudo se entrenan en grandes conjuntos de datos para comprender una amplia gama de consultas y proporcionar respuestas coherentes y relevantes. Ejemplos notables incluyen GPT-3.5, que ha demostrado habilidades sorprendentes en la generación de texto diverso y contextual.

Sin embargo, en el ámbito de *chatbots* de propósito específico, se han implementado soluciones más especializadas para cumplir funciones particulares. Un ejemplo es el *chatbot* de atención al cliente que puede ayudar a los usuarios con consultas relacionadas con productos o servicios específicos. Estos *chatbots* a menudo se integran con sistemas empresariales y bases de datos para ofrecer respuestas precisas y personalizadas.

En cuanto a las técnicas de *prompt design* y *prompt engineering*, existen varias estrategias para mejorar la interacción y el rendimiento de los *chatbots* (4):

- ***Zero Shot***: Permite que el modelo responda a consultas para las cuales no ha sido explícitamente entrenado. Esto es útil para situaciones en las que se necesita flexibilidad en las respuestas.
- ***Few Shots***: Permite que el modelo se adapte a nuevas tareas con una pequeña cantidad de ejemplos de entrenamiento. Esto es beneficioso para la adaptación rápida a situaciones específicas.
- ***Chain of Thoughts***: Proporciona coherencia en las respuestas al seguir una cadena lógica de pensamientos. Esto mejora la cohesión en las interacciones y ayuda a evitar respuestas contradictorias.
- ***Reason-and-Act***: Habilita la capacidad del *chatbot* para razonar sobre la información proporcionada y tomar acciones basadas en ese razonamiento. Esto puede ser esencial para ofrecer respuestas más útiles y relevantes.

2.1.1. Plataformas a tener en cuenta

- **ChatPDF**. ChatPDF(5) es una plataforma que permite establecer una conversación con un documento PDF. Puedes realizar preguntas y demás consultas sobre la información aportada.
- **AskYourPDF**. AskYourPDF(6) es una plataforma que, a groso modo, ofrece las mismas funcionalidades que ChatPDF pero a través de una interfaz diferente.

Tras analizar estas plataformas podemos extraer tres ideas principales que todo *chatbot* de documentos debería seguir.

1. **Conversaciones independientes**. Al usuario final le resulta muy cómodo separas sus consultas en conversaciones independientes en función de las consultas que quiera realizar.
2. **Repuestas rápidas**. Las respuestas deben ser obtenidas en un periodo de tiempo corto, este tiempo no debería ser mayor al que el usuario dedicaría en buscar la respuesta por si mismo.
3. **Respuestas precisas**. Las respuestas deben ser precisas y ciertas, la información errónea solo empeoran la experiencia de usuario.

2.2. Decisiones Tecnológicas

A lo largo de este apartado trataremos la elección de tecnologías en el proyecto.

2.2.1. Tecnologías comunes para *chatbots* conversacionales

A continuación, se desarrollan las tecnologías más comunes para el desarrollo de este tipo de proyectos, cabe destacar que no todas las herramientas mencionadas serán utilizadas, pero es importante conocerlas para poder elegir adecuadamente cual se adapta mejor a nuestro caso.

LangChain

LangChain (7) es un marco para el desarrollo de aplicaciones basadas en grandes modelos de lenguaje (*LLM*) (8). Hay que destacar que LangChain no implementa sus propios *LLM* sino que suministra soporte para los creados por OpenAI, Meta, HuggingFace, etc.

LangChain simplifica todas las fases del ciclo de vida de las aplicaciones *LLM*. En cuanto al desarrollo permite construir aplicaciones utilizando los componentes y bloques de construcción de código abierto de LangChain. También permite la utilización de plantillas de terceros para agilizar el desarrollo.

En cuanto a la producción existe LangSmith (Funcionalidad de LangChain), la cual permite inspeccionar, supervisar y evaluar los *prompts*, consiguiendo optimizarlos y obtener mejores resultados.

Para el despliegue existe LangServe (Funcionalidad de LangChain), lo que posibilita desplegar rápidamente la aplicación a través de FastAPI, permitiendo conectar la aplicación con otros módulos.

OpenAI API

La *API* de OpenAI (9) es una herramienta fundamental que permite a los desarrolladores integrar modelos de lenguaje avanzados en sus propias aplicaciones o productos. Entre las principales funcionalidades que ofrece OpenAI se destacan los modelos de generación de textos avanzados y los asistentes inteligentes.

Un componente esencial de la tecnología detrás de los modelos de OpenAI son los *embeddings* (10). Los *embeddings* son representaciones numéricas de palabras y frases que capturan el significado y las relaciones semánticas entre ellas. Al utilizar *embeddings*, los modelos de OpenAI pueden comprender y generar texto de manera más efectiva, identificando patrones y contextos complejos dentro del lenguaje.

La tecnología de los asistentes de OpenAI se basa en modelos de lenguaje avanzados como *GPT* (*Generative Pre-trained Transformer*). Estos modelos han sido entrenados con grandes cantidades de información textual para desarrollar patrones lingüísticos sofisticados, lo que les permite generar texto de alta calidad que en muchas situaciones es indistinguible del producido por un humano.

Los modelos de OpenAI están disponibles a través de su *API*, lo que facilita a los desarrolladores la creación de aplicaciones que ofrecen experiencias de usuario personalizadas y atractivas, ya sea en el ámbito del servicio al cliente, la educación, la atención médica, o cualquier otra área donde la comunicación efectiva sea crucial.

2.2.2. Tecnologías *Frontend* para el desarrollo de la plataforma.

Existen numerosos *frameworks* para realizar interfaces web, a continuación se presentaran de forma breve las principales opciones planteadas.

Next.js

Next.js (11) es un *framework* de React (12) pensado para la creación de aplicaciones *full-stack*. Permite enfocarse en construir una aplicación sin tener que preocuparse por configuraciones tediosas o pesadas. Gracias a este *framework* es fácil crear aplicaciones React interactivas, dinámicas y con un alto rendimiento.

Next.js permite renderizado del lado del cliente y del servidor. Esto es interesante cuando debemos renderizar una página muy pesada o que requiera de mucho esfuerzo computacional, generando tiempos de carga más rápidos.

Otro aspecto a tener en cuenta es la obtención de datos en los componentes del servidor utilizando *async/await*. Este *framework* también ofrece una *API* de fetch mejorada que permite memorizar solicitudes y almacenar datos en caché.

En cuanto al estilizado, next.js habilita la utilización de varios métodos como CSS Modules, Tailwind CSS y CSS-in-JS.

Por último, la experiencia de usuario y la experiencia del desarrollador también se ven mejoradas, la del usuario a través de la optimización de imágenes, scripts y demás componentes, y la del desarrollador por medio de una actualización en el soporte de TypeScript, mejorando la comprobación de tipos y habilitando una compilación más eficiente.

Astro

Astro (13) destaca como *framework* web ideal para construir páginas enfocadas al contenido o con baja demanda dinámica, Su enfoque se centra en reducir la carga y complejidad de JavaScript, lo que resulta en un tiempo de carga más rápido en comparación a la utilización de otros *frameworks*.

Astro ofrece una primera toma de contacto bastante intuitiva para el usuario, lo que lo convierte en una opción bastante llamativa. Este cuenta con un arquitectura orienta a islas, esta arquitectura basada en componentes, ofrece una mejora en cuanto a tiempos de carga, optimizando la renderización del contenido. Lo que genera una mejor experiencia de usuario.

El punto fuerte de este *framework* es su gran compatibilidad, permite la creación de componentes utilizando diversas tecnologías, por lo que es posible contar con un proyecto donde hayan componentes escritos en React, otros en Vue y también otros en lenguaje Astro, Si tenemos en cuenta esta característica migrar un proyecto a esta tecnología es muy sencillo.

La compatibilidad también existe con los *frameworks* de estilización, dándonos soporte para usar tecnologías como Tailwind CSS o Bootstrap.

Astro, al igual que Next.js, permite la renderización de páginas en el lado del servidor.

Tailwind CSS

Tailwind CSS (14) es un *framework* de CSS que ofrece clases predefinidas, lo que permite abstraerse de la liosa y tediosa experiencia de manejar CSS en crudo.

Al contrario de otros *frameworks*, tailwind no ofrece componentes preestilizados, aunque existen varios *plugins* que si que los proporcionan como es el caso de daisyUI (15).

La configuración personalizada es otro de sus puntos fuertes. A través de su archivo de configuración, los desarrolladores tienen la libertad de personalizar en profundidad las normas de estilo. También permite la creación de temas personalizados y etiquetas.

El diseño responsivo es una consideración fundamental en el desarrollo web moderno, y Tailwind CSS lo hace más accesible mediante el uso de clases predefinidas que se aplican de manera intuitiva en diferentes tamaños de pantalla.

Tailwind ofrece un enfoque modular que permite utilizar solo las clases necesarias en cada componente y no el *framework* completo. Reduciendo la carga de CSS en la pagina e incrementado el rendimiento en gran medida.

2.2.3. Tecnologías *Backend* para el desarrollo de la plataforma.

Flask

Flask (16) es un *framework* web ligero muy extendido en el desarrollo en python para la creación de *APIs* (Interfaces de Programación de Aplicaciones). Gracias a su simplicidad y facilidad de uso es un tecnología a tener en cuenta.

Flask permite definir rutas y vistas de manera intuitiva utilizando decoradores, lo que facilita la creación de puntos finales de la *API* para manejar solicitudes HTTP como GET, POST, PUT, DELETE, etc. Su sencillez es uno de sus puntos fuertes, permitiendo entender el funcionamiento de la aplicación desde el primer instante.

Otro punto a favor es la compatibilidad con diversas librerías de serialización de datos, como JSON, lo que facilita la creación de respuestas JSON para las solicitudes de la *API*.

Para la gestión de errores y excepciones se proporcionan mecanismos simples para su manejo, lo que permite devolver respuestas adecuadas en casos no deseados en las solicitudes de la *API*. Esto es útil y utilizado con frecuencia para evitar el mal uso del sistema.

Flask es altamente escalable y permite integrar fácilmente extensiones para agregar funcionalidades adicionales, como autenticación, autorización, validación de datos, entre otros, a la *API* según las necesidades del proyecto.

Por último, esta tecnología proporciona herramientas para generar documentación interactiva de la *API*, lo que facilita que otros desarrolladores comprendan cómo interactuar con ella y utilicen los puntos finales de manera efectiva. Esta tarea suele ser algo tediosa, por lo que el uso de esta funcionalidad ahorra mucho tiempo y trabajo.

FastAPI

FastAPI (17) es un *framework* moderno para la creación de *APIs*. Al igual que lo hace Flask, proporciona funcionalidades para gestionar la *API* de manera rápida y eficiente. Destaca por su alta velocidad, su comportamiento ante grandes cantidades de solicitudes y por su facilidad de uso.

FastAPI ofrece una serie de ventajas significativas para el desarrollo de *APIs*:

En primer lugar, se destaca su capacidad para mejorar la productividad. Esto se logra gracias a su sintaxis intuitiva y el sistema de decoradores, los cuales simplifican la definición de rutas y operaciones *CRUD* (**Create, Read, Update, Delete**).

Además, FastAPI se distingue por su velocidad y rendimiento excepcionales. Diseñado específicamente para ofrecer un alto rendimiento, aprovecha la capacidad de Python para el procesamiento asíncrono.

Otra característica destacada es la validación automática de datos, esta funcionalidad simplifica la validación de las solicitudes entrantes y salientes de la *API*, garantizando la integridad de los datos.

FastAPI también sobresale en la generación automática de documentación interactiva para la *API*. Esta característica facilita que otros desarrolladores comprendan cómo interactuar con ella y puedan probar los puntos finales de manera efectiva. Al igual que con Flask, esta funcionalidad permite ahorrar mucho trabajo y esfuerzo.

La compatibilidad con tipos de datos nativos de Python es otro punto a destacar. Esto significa que se pueden utilizar tipos de datos como *int, str, bool, list y dict* directamente en la definición de las rutas y modelos de datos de la *API*.

Por último, su alta escalabilidad y extensibilidad permiten agregar fácilmente funcionalidades adicionales según las necesidades específicas del proyecto, como autenticación, autorización, manejo de errores y más.

2.2.4. Decisión final de las tecnologías

LangChain vs OpenAI

Ambas opciones tienen su atractivo y permiten la interacción con los *LLM* más utilizados, Entre estos modelos encontramos GPT-3.5, GPT-4, Llama o Gemini. Cabe destacar que la *API* de OpenAI solo permite la utilización de los modelos creados por su empresa, mientras que Langchain nos ofrece un marco de posibilidades mayor en cuanto a compatibilidad.

Aún así, debido a razones de rendimiento y compatibilidad con las nuevas versiones de la librería de OpenAI, la elección elegida es la *API* de OpenAI. Gracias a ella podemos usar el modelo GPT-3.5, un modelo bastante avanzado y probado, el cual nos puede otorgar respuestas de forma rápida y precisa.

En las versiones iniciales de la biblioteca de OpenAI, Langchain ofrecía una serie de funcionalidades útiles. Sin embargo, a medida que OpenAI continuaba su desarrollo, muchas de estas funcionalidades fueron incorporadas directamente en su propia biblioteca. Este proceso resultó en la creación de una capa de abstracción adicional en Langchain, lo que generaba código más extenso para lograr los mismos resultados que se obtenían directamente con OpenAI (18).

Para entender esto podemos observar los siguientes fragmentos de código 2.1 y 2.2

```
1  from langchain.chat_models import ChatOpenAI
2  from langchain.schema import (
3      AIMessage,
4      HumanMessage,
5      SystemMessage
6  )
7
8  chat = ChatOpenAI(temperature=0)
9  chat.predict_messages([HumanMessage(content="Translate this sentence from English to
       French. I love programming.")])
10 # AIMessage(content="J'adore la programmation.", additional_kwargs={}, example=False)
```

Figura 2.1: Ejemplo usando LangChain de como traducir de inglés a francés

```
import openai

messages = [{"role": "user", "content": "Translate this sentence from English to
    French. I love programming."}]

response = openai.ChatCompletion.create(model="gpt-3.5-turbo", messages=messages,
    temperature=0)
response["choices"][0]["message"]["content"]
# "J'adore la programmation."
```

Figura 2.2: Ejemplo usando solo OpenAI de como traducir de inglés a francés

Vemos como en el fragmento de código 2.1 el proceso requiere de una capa de procesamiento adicional para realizar lo mismo que se hace en el fragmento 2.2 con menos líneas de código y menos coste computacional.

Frontend

Entre las tecnologías mencionadas se ha decidido el uso del *framework* Astro, su facilidad de uso, su sintaxis intuitiva y su compatibilidad con otras tecnologías han sido los factores principales de la decisión,

El *frontend* se hará haciendo uso de componentes Astro y el desarrollo será orientado a objetos escrito en el lenguaje TypeScript.

También se hará uso del *framework* CSS Tailwind junto con el *plugin* daisyUI para estilizar la página.

Backend

Para la creación del *Backend* y la *API* se hará uso del *framework* Flask, este es conocido y no muestra grandes desventajas frente a FastAPI.

Tras la elección de esta tecnología nos vemos obligados a usar el lenguaje Python, el cual cuenta con numerosas librerías que ayudarán en el proceso de creación del sistema.

Capítulo 3

Diseño y prototipado

El objetivo de la plataforma ChatULL es permitir al alumnado, cuerpo docente y demás usuarios de la Universidad de La Laguna, realizar cuestiones sobre las guías docentes de las distintas asignaturas ofrecidas por los grados.

A continuación se encuentra el ciclo de funcionamiento simplificado de la aplicación web (Figura 3.1).

1. **Envío del mensaje**: El usuario envía un mensaje desde su dispositivo al servidor de ChatULL, el mensaje se empaqueta y encripta con HTTPS.
2. **Procesamiento del mensaje**: El servidor recibe y procesa el mensaje. Tras esta acción se crea un nuevo mensaje especial, el *prompt*, y se envía a los servidores de OpenAI.
3. **Respuesta de OpenAI**: OpenAI analiza la petición recibida y genera una respuesta, la cual es nuevamente enviada al servidor de ChatULL.
4. **Recepción por parte de ChatULL**: El servidor recoge la respuesta y la trata, una vez finalizado el proceso, se envía al cliente para que este prosiga con el ciclo.
5. **Recepción por parte del usuario final**: El cliente recibe la respuesta tratada y la carga en la interfaz gráfica. El mensaje mostrado se encuentra en lenguaje natural sencillo y comprensible para el usuario.

Analizando el funcionamiento podemos observar cuatro nodos principales, cuyas responsabilidades se describen a continuación.

- **Cliente**: Se encarga de conectarse a la plataforma y de realizar consultas sobre las asignaturas.
- ***Frontend***: Interfaz a la que se conecta el usuario, a través de ella se envían preguntas y también se muestran las respuestas.
- ***Backend***: Procesa y analiza las solicitudes que le llegan del *Frontend*, también habilita una interfaz entre el *Frontend* y OpenAI.
- **OpenAI**: Se encarga de devolver una respuesta a la pregunta inicial del usuario a través del uso de *LLM*.

A lo largo de este apartado analizaremos el diseño y construcción tanto del *Frontend* como del *Backend*.

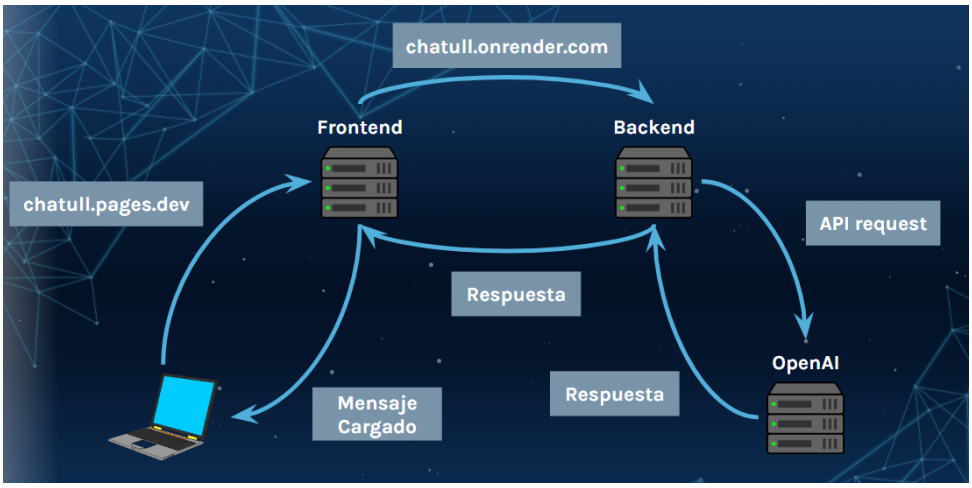

Figura 3.1: Diagrama del flujo de acción de la plataforma al realizar una pregunta

3.1. Diseño y Prototipado del *Frontend*

A lo largo de este apartado se describirá el diseño y prototipado del *Frontend* de la aplicación web. Se describirán las funcionalidades esperadas, el diseño y se mostrarán y analizarán los prototipos realizados.

3.1.1. Aspecto y funcionalidades esperadas

Como se ha mencionado anteriormente, el *Frontend* de la aplicación web se encargará de la interacción con el usuario. Por tanto, se espera que el diseño sea intuitivo y fácil de usar. Además, se espera que el usuario pueda realizar las siguientes acciones:

- Poder introducir una *API-Key* para poder acceder a la aplicación web ChatULL y poder realizar consultas.
- Poder crear un nuevo chat sobre la Guía Docente de una asignatura.
- Poder crear un nuevo chat sobre la reglamentación de Guías Docentes que existe en la Universidad de La Laguna.
- Poder realizar consultas sobre la Guía o Reglamentación en la que se encuentre.
- Poder consultar las preguntas formuladas junto con las respuestas obtenidas en otro momento de cualquiera de los chats.

A continuación, se describirá el diseño y prototipado del *Frontend* de la aplicación web.

3.1.2. Prototipado. Primera versión

Diseño

El diseño de la primera versión se centró en la creación de una interfaz sencilla y fácil de usar. Se optó por un diseño minimalista y limpio, con un un espacio de texto para

introducir el mensaje y un botón para enviarlo. Sobre estos se sitúa el espacio del chat, donde se pueden observar los mensajes enviados y las respuestas recibidas. Además, se crearon otras páginas para explicar el por qué de la existencia de la plataforma y para poder contactar con el desarrollador. En la figura 3.2 se puede ver la interfaz de la primera versión de la aplicación web.

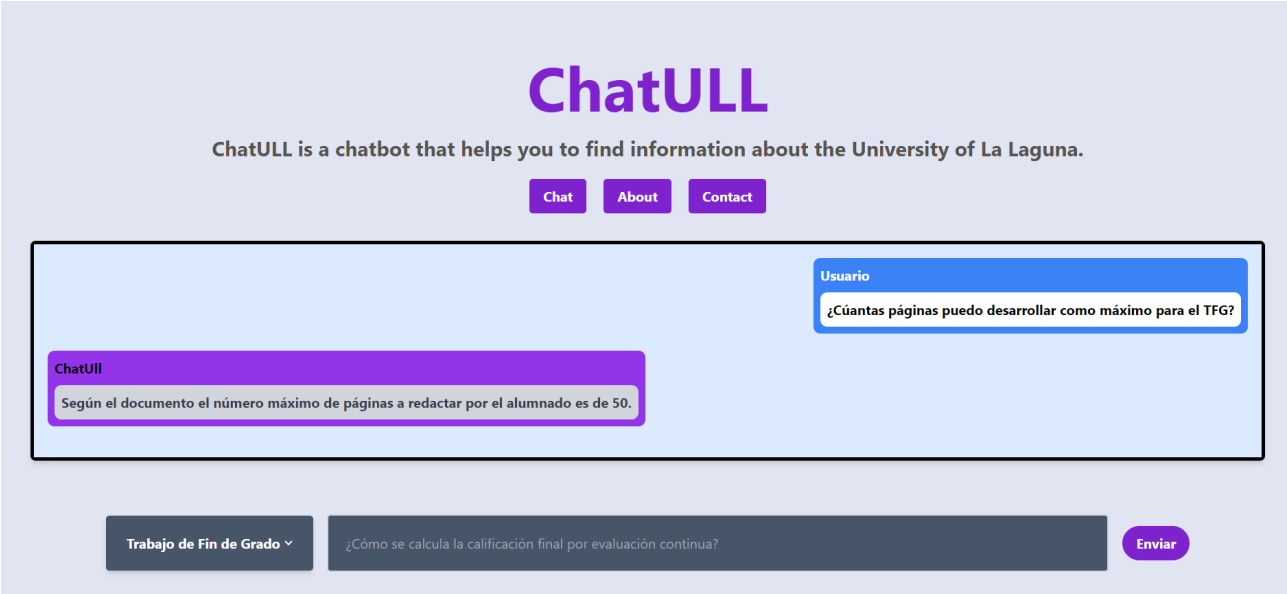

Figura 3.2: Interfaz *Frontend* Primera Versión

Funcionamiento. Función GetQuestionAnswer

El funcionamiento de la primera versión de la aplicación web era sencillo. El usuario introducía un mensaje en el espacio de texto y pulsaba el botón de enviar. El *Frontend* se encargaba de enviar el mensaje al *Backend* donde se procesaba y se devolvía una respuesta. Finalmente, esta se mostraba en la misma página, debajo del espacio de texto.

En esta primera versión el usuario final no debía introducir ninguna *API-Key* ya que era el *host* de la plataforma quien asumiría todos los costes asociados a su uso. Posteriormente con la versión final esta idea cambió forzando a interactuar con el *backend* de forma distinta.

Esta función es la encarga de trasladar la pregunta a la *API* para posteriormente cargar la respuesta y mostrarla en la interfaz.

Analizando en el fragmento de código 3.3 observamos el siguiente flujo de acción.

1. Modificamos ciertos textos y comportamientos de etiquetas para una correcta experiencia de usuario, como por ejemplo el texto del botón a “Obteniendo respuesta...” mientras se espera la respuesta del servidor y deshabilitar el botón durante la solicitud.

2. Actualizamos la *URL* de la solicitud para que contenga los parámetros necesarios, como la pregunta y la asignatura o reglamento seleccionado, adaptando la consulta para que sea compatible con la *URL* mediante la sustitución de espacios por "%20".

3. Dependiendo del tema seleccionado en el selector, construimos la *URL* de la solicitud para obtener respuestas del servidor, utilizando diferentes puntos finales para temas específicos como “Reglamentación y Normativa”.

4. Si el tema seleccionado coincide con el tema actual, cargamos la pregunta y su respuesta en la interfaz de usuario.

5. Manejamos errores en caso de que la solicitud falle, mostrando un mensaje de error en la interfaz de usuario y restaurando el estado del botón.

```
function obtenerRespuesta(input, selector, button) {
    button.disabled = true;
    button.textContent = "Obteniendo respuesta...";

    let question = input.value;
    input.value = "";

    question = question.replace(/ /g, "%20");
    let url = "";
    let actualSubject = selector.value;
    if (selector.value != "Reglamentacion y Normativa") {
        url = 'https://chatull.onrender.com/get_answer/' + '?question=' + question +
            '&subject=' + actualSubject;
    } else {
        url = 'https://chatull.onrender.com/get_teacher_answer/' + '?question=' +
            question;
    }
    loadQuestion(question.replace(/%20/g, " "));
    input.scrollIntoView();
    fetch(url)
        .then(response => response.json())
        .then(data => {
            let answer = data.answer;

            if (selector.value === actualSubject) {
                loadAnswer(answer);
                input.scrollIntoView();
            }

            button.disabled = false;
            button.textContent = "Enviar";
            })
        .catch(error => {
            console.error('Error:', error);
            button.disabled = false;
            button.textContent = "Enviar";
            loadAnswer("Error al obtener respuesta.");
        });
}
```

Figura 3.3: Función encargada de trasladar la pregunta al backend y posteriormente cargarla en la interfaz en la primera versión del código.

3.1.3. Prototipado. Segunda versión

Diseño

Respecto a la primera versión realizada se han detectado una serie de errores y se han realizado una serie de mejores. Esta segunda versión busca acercarse a las

funcionalidades extraídas al principio del capítulo y también tiene como objetivo seguir recabando los posibles errores de planteamiento no detectados.

Las mejoras realizadas son las siguientes:

- Cambio de los colores principales de la plataforma para adaptarlos a la identidad de la Universidad de La Laguna.
- Creación de un buscador en tiempo real de las asignaturas disponibles en la plataforma.
- Adición de un botón para eliminar el historial de una conversación.
- Persistencia de la conversación entre sesiones para que el usuario no pierda las consultas realizadas anteriormente.

Tras su utilización y desarrollo se han identificado los siguientes problemas a corregir en la versión final:

- Interfaz poco intuitiva para el usuario. En primera instancia puede resultar complicado entender el manejo de la plataforma, ya que no se indica como seleccionar o cambiar de guía docente y tampoco se ofrecen reglas básicas para obtener mejores resultados las consultas.
- El usuario no tiene una forma directa de saber sobre qué asignaturas ha realizado preguntas ya que únicamente se muestra cual es la materia seleccionada actualmente.
- El código cuenta con una estructura que dificulta su mantenimiento. Este inconveniente surge debido a la ausencia de clases en su desarrollo y a su naturaleza funcionalmente improvisada, creada inicialmente para garantizar su funcionamiento, con la posterior intención de migrar hacia un modelo de arquitectura del tipo Modelo-Vista-Controlador (MVC) (19).
- La plataforma web no es utilizable en dispositivos con pantallas pequeñas.

En la figura 3.4 podemos observar los cambios citados previamente.

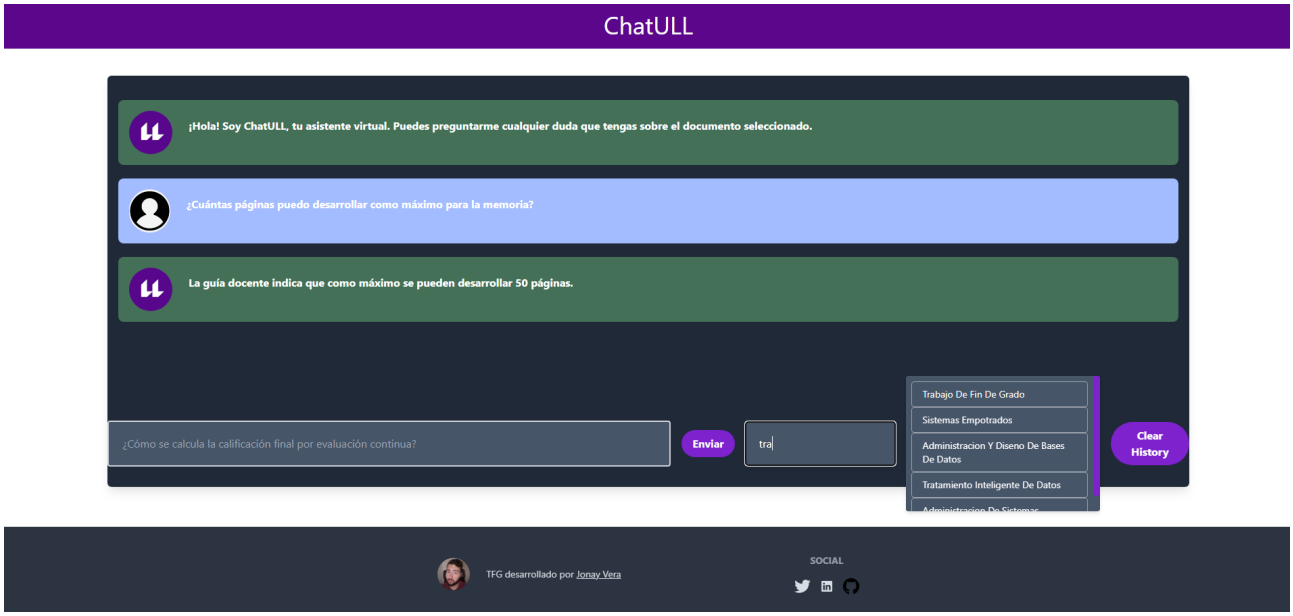

Figura 3.4: Interfaz *Frontend* Segunda Versión

Funcionamiento

Utilizamos *localStorage* (20) para almacenar las conversaciones en la segunda versión de la aplicación web debido a su simplicidad y conveniencia. En lugar de crear y gestionar una base de datos, *localStorage* proporciona una forma sencilla de almacenar datos en el navegador del usuario. Esto es especialmente útil para la información relativamente simple que necesitamos almacenar, como las conversaciones de chat, ya que nos permite persistir los datos de manera fácil y rápida sin depender de un servidor o una base de datos externa.

Lista de cuestiones a tener en cuenta para mantener el funcionamiento:

- **Adaptación del formato de los datos**: Modificamos el formato de los datos de las conversaciones para que sean compatibles con el almacenamiento en *localStorage*. Para esto debemos adaptar el formato de las preguntas y respuestas a JSON (21) antes de almacenarlos y serializarlos, y luego analizarlos antes de usarlos.
- **Actualización de la lógica de carga y guardado**: Revisamos y actualizamos la lógica de carga y guardado de las conversaciones para que utilice *localStorage* en lugar de simplemente añadir una serie de etiquetas html para poder visualizar la información.
- **Manejo de límites de almacenamiento**: Consideramos y manejamos los límites de almacenamiento impuestos por *localStorage* para asegurarnos de que no excedamos los límites permitidos y gestionemos adecuadamente los datos almacenados. Al tratarse de una cantidad pequeña de información esto no será un problema para la plataforma.
- **Gestión de errores**: Contemplamos manejar los errores en situaciones donde el almacenamiento en *localStorage* pueda fallar, como cuando el almacenamiento está lleno o cuando se producen errores de serialización/deserialización.
- **Actualización de la interfaz de usuario**: Actualizamos la interfaz de usuario para reflejar los cambios en el funcionamiento de la aplicación debido al uso de *localStorage*. Esto incluye la visualización de mensajes de errores relacionados con el almacenamiento local.

Estos cambios nos permiten mantener el funcionamiento de la aplicación web mientras aprovechamos las ventajas de *localStorage* para el almacenamiento de datos local en el navegador del usuario.

A términos prácticos hemos tenido que realizar los siguientes cambios:

- **Estructura en formato JSON**: Ahora es necesario la utilización de un objeto JavaScript (como el de la figura 3.5) para facilitar la interacción con *localStorage*.
- **Código encargado de la carga, lectura y visualización (*localStorage*)**: El código encargado de cargar, leer o visualizar la información referente a las cuestiones de los usuarios ha sufrido cambios, ahora se utilizan los métodos proporcionados por el navegador para cumplir con el funcionamiento esperado (métodos presentes en la figura 3.6).
- **Código encargado de la carga, lectura y visualización (Interfaz de Usuario)**: Se han creado nuevas funciones dedicadas a visualizar la información extraída del

almacenamiento local tras seleccionar una materia en la que se habían realizado preguntas (ejemplo de función creada para la visualización del texto extraido de *localStorage* en la Figura 3.7).

```
let questionObject = {
    "question": "¿Cuántas páginas puedo desarrollar como máximo para la memoria?",
    "answer": "La guía docente indica que como máximo se pueden desarrollar 50 pá
        ginas."
};
```

Figura 3.5: Ejemplo de estructura utilizada para el almacenamiento en localStorage.

```
// Método parse() para transformar JSON en un objeto JavaScript y método getItem para
     obtener el JSON necesario
let questions = JSON.parse(localStorage.getItem(actualSubject));
questions.push(questionObject);

// Método setItem para guardar JSON en localStorage y método stringify para
    transformar un objeto JavaScript en JSON
localStorage.setItem(actualSubject, JSON.stringify(questions));
```

Figura 3.6: Ejemplo de como se accede a *localStorage* para almacenar una nueva pregunta/respuesta.

```
function loadQuestion(text) {
    let messageContainer = document.createElement("div");
    messageContainer.setAttribute("class", "message-container bg-[#a3bcff] user-
        message flex flex-row items-center rounded-lg");
    let userMessage = document.createElement("div");
    userMessage.setAttribute("class", "UserMessage text-white rounded-lg p-2 mb-5
        font-bold");
    let userPhoto = document.createElement("img");
    userPhoto.setAttribute("class", "rounded-full ml-4 mr-4 mt-4 mb-4 max-h-16 m-w-16
        ");
    userPhoto.setAttribute("src", "https://www.clipartmax.com/png/middle/434-4349876
        _profile-icon-vector-png.png");
    messageContainer.appendChild(userPhoto);
    let message = document.createElement("div");
    message.setAttribute("class", "message text-white rounded-lg");
    message.textContent = text.question;
    userMessage.appendChild(message);
    messageContainer.appendChild(userMessage);
    document.querySelector("div.chat-container").appendChild(messageContainer);
    messageContainer.scrollIntoView();
}
```

Figura 3.7: Función para la visualización de las preguntas extraidas de *localStorage*.

3.1.4. Prototipado. Aproximación final

En este subapartado analizaremos la versión final a la que se ha llegado tras evaluar diversas cuestiones y problemáticas de las anteriores aproximaciones.

3.1.5. Diseño

Respecto a la segunda versión realizada también se han detectado una serie de errores y se han realizado una serie de mejores. Esta aproximación final plantea los aspectos y funcionalidades listados al comienzo del capítulo y recoge una serie de limitaciones extraídas que se plantearán más adelantes como posibles mejores o líneas futuras de desarrollo del TFG.

Las mejoras realizadas son las siguientes:

- Todas las páginas creadas para la plataforma han sido adaptadas por completo a la temática seguida por la universidad.
- Creación de una página para poder indicar la *API-Key*, sin esta clave no se puede utilizar la plataforma. (Véase la figura 3.8)
- Cambio notable en la interfaz del chat, empezando por un listado de conversaciones creadas en la parte izquierda del panel, también se ha modificado la forma en la que se muestran los mensajes, dándole un toque más neutro y claro. Por último se ha añadido un modal para la creación de nuevos chats, el cual implementa el buscador en tiempo real (Véanse las figuras 3.9 y 3.10).
- La plataforma es totalmente utilizable en dispositivos de cualquier tamaño de pantalla.
- Se ha realizado una modificación drástica en la estructura del código, adoptando un enfoque orientado a objetos que se asemeja al patrón MVC, aunque fusionando la vista y el controlador en una sola clase. Esta decisión se tomó buscando simplificar el proceso de desarrollo, dada la reducida cantidad de componentes y partes interactivas involucradas. Más adelante se analizará esta nueva estructura.

Tras su utilización y desarrollo se han identificado las siguientes limitaciones respecto al diseño:

- El sistema de usuarios es simple por lo que no da mucha opciones de personalización al usuario. Por ejemplo no hay un perfil de usuarios donde se puedan elegir entra las *API-Keys* introducidas.
- No existe una vista previa de la guía docente, lo que permitiría el contraste de información en tiempo real si el usuario lo quisiera.

A pesar de las limitaciones identificadas en el diseño, es importante destacar que la plataforma sigue siendo funcional y cumple con los objetivos principales establecidos. Aunque el sistema de usuarios podría ofrecer más opciones de personalización y comodidad, como la capacidad de gestionar múltiples *API-Keys* o la implementación de perfiles de usuario más detallados, estas deficiencias no impiden el uso efectivo de la plataforma para su propósito previsto.

Por otro lado, la ausencia de una vista previa de la guía docente puede ser considerada como una limitación menor en comparación con las mejoras significativas realizadas en esta aproximación final. Sin embargo, proporcionar esta funcionalidad podría mejorar la experiencia del usuario al permitir un acceso más rápido y eficiente a la información relevante.

En resumen, aunque siempre hay áreas que pueden mejorarse, la versión final de la plataforma representa un avance significativo con respecto a las iteraciones anteriores, incorporando mejoras importantes en la interfaz de usuario, la funcionalidad y la estructura del código. Estas mejoras son el resultado de una cuidadosa evaluación de las cuestiones y problemáticas identificadas en las etapas anteriores del proyecto, y sientan una sólida base para posibles desarrollos futuros.

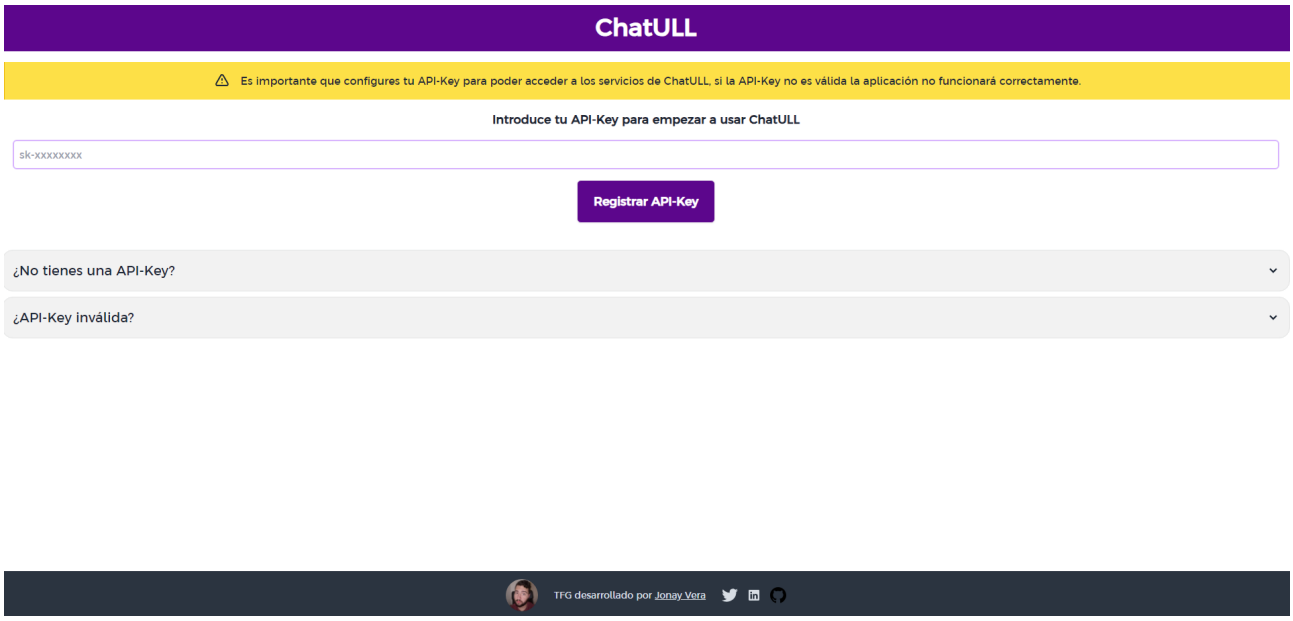

Figura 3.8: Interfaz *Frontend* Acceso con *API-Key*

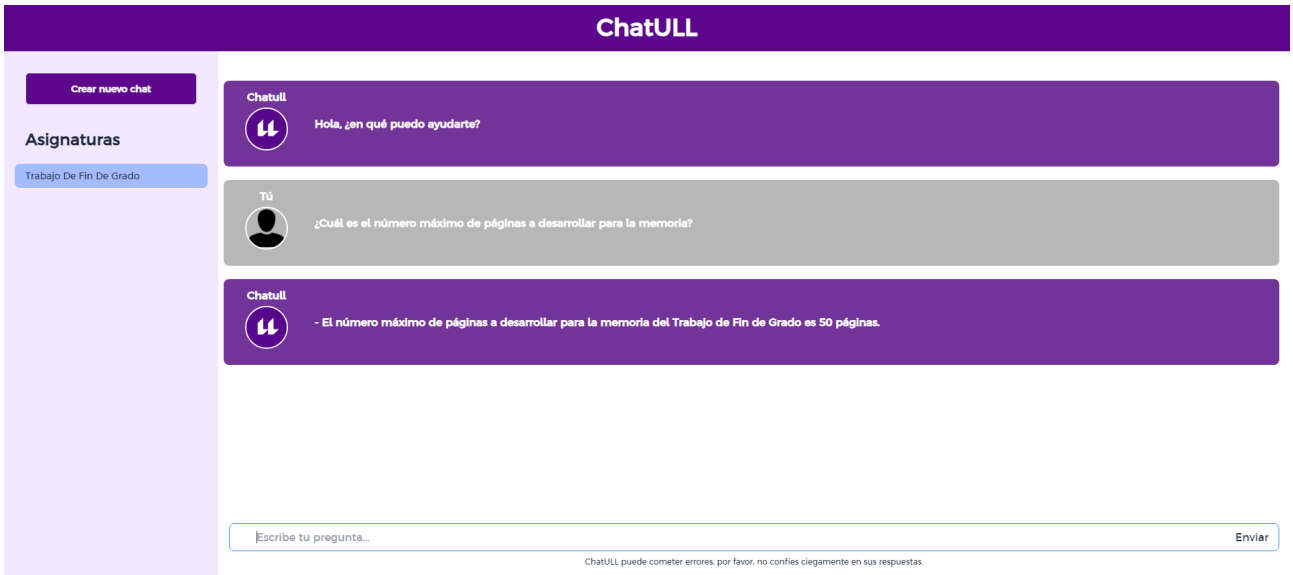

Figura 3.9: Interfaz *Frontend* Aproximación Final

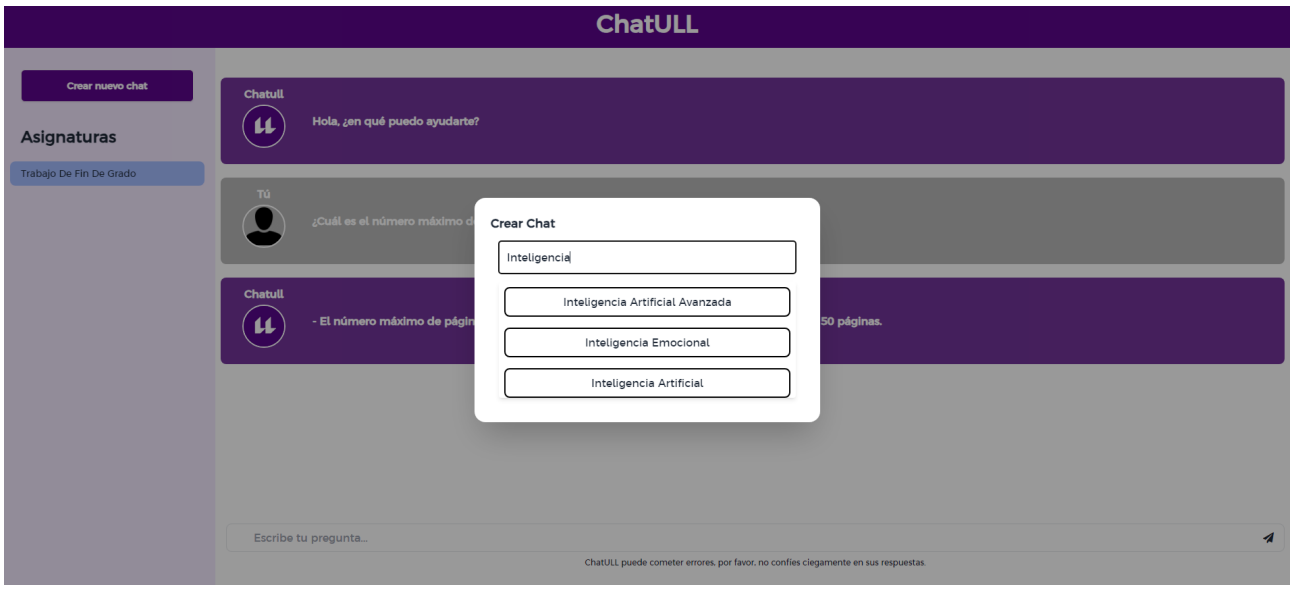

Figura 3.10: Interfaz *Frontend* Creación de un nuevo chat

Diagrama *UML*

La figura 3.11 muestra el diagrama *UML* de la estructura de clases del *Frontend* de la aplicación web. En este diagrama se pueden observar las clases y métodos principales que componen el *Frontend*, así como las relaciones entre ellas.

De este diagrama podemos extraer las siguientes conclusiones:

- La clase Message representa un mensaje enviado por el usuario o recibido del servidor. Contiene información sobre el texto del mensaje y si se trata de un mensaje de usuario o de la plataforma.
- La clase SessionController es la encargada de gestionar la sesión del usuario, incluyendo la autenticación a través de la *API-Key*.
- La clase SubjectController se encarga de gestionar las asignaturas disponibles y el menu lateral con las conversaciones activas y el modal de creación de nuevos chats.
- La clase ChatController es la clase principal que se encarga de gestionar la conversación actual, incluyendo la carga de mensajes, el envío de mensajes y la gestión de la interfaz de usuario.

En nuestra especia de MVC hemos decidido fusionar la vista y el controlador para crear tres supercontroladores que se encargan de la gestión de la interfaz de usuario y la lógica de la aplicación. Podemos tomar la parte del modelo como el almacenamiento local de las conversaciones, que se gestiona a través de los métodos para comunicarse con *localStorage* existentes en cada uno de los controladores.

Figura 3.11: Diagrama *UML* del *Frontend*

3.1.6. Clase Message

A continuación, se listan los atributos y métodos de la clase Message junto con una breve descripción de su funcionalidad.

Atributos

- **is_question_: boolean**: Indica si el mensaje es una pregunta.
- **text_: string**: Indica el texto del mensaje.

Métodos

- **constructor(text: string, is_question: boolean)**: Constructor de la clase Message. Recibe el texto del mensaje y un booleano que indica si es una pregunta.
- **BuildMessage(): HTMLElement**: Construye un elemento HTML que representa el mensaje.
- **GetText(): string**: Devuelve el texto del mensaje.
- **IsQuestion(): boolean**: Devuelve si el mensaje es una pregunta.

Funcionamiento

La clase Message se encarga de representar un mensaje enviado por el usuario o recibido del servidor. Almacena información sobre el texto del mensaje y si se trata de una pregunta o una respuesta. La clase proporciona métodos para construir un elemento HTML que representa el mensaje, obtener el texto del mensaje y comprobar si es una pregunta.

3.1.7. Clase SessionController

A continuación, se listan los atributos y métodos de la clase SessionController junto con una breve descripción de su funcionalidad.

Atributos

- **session_button_: HTMLButtonElement**: Botón de inicio de sesión.
- **session_input_: HTMLInputElement**: Campo de texto para introducir la *API-Key*.

Métodos

- **constructor(session_input_tag : string, session_button_tag: string)**: Constructor de la clase SessionController. Recibe los selectores de los elementos HTML del botón y el campo de texto.
- **setApiKey(): async**: Establece la *API-Key* introducida por el usuario.
- **GetJwt(): string**: Devuelve el *JWT* (22) almacenado en localStorage.
- **Init(): void**: Inicializa el controlador de sesión.
- **RemoveJwt(): void**: Elimina el *JWT* almacenado en localStorage.
- **SetJwtToLocalStorage(jwt: string): void**: Almacena el *JWT* en *localStorage*.

Funcionamiento

La clase SessionController se encarga de gestionar la sesión del usuario, incluyendo la autenticación a través de la *API-Key*. Almacena la *API-Key* introducida por el usuario y proporciona métodos para establecerla, obtenerla y eliminarla. La clase también proporciona métodos para inicializar el controlador de sesión y almacenar el *JWT* en *localStorage*.

3.1.8. Clase SubjectController

A continuación, se listan los atributos y métodos de la clase SubjectController junto con una breve descripción de su funcionalidad.

Atributos

- **subject_selector_: HTMLInputElement**: Selector de asignatura.
- **menu_selector_: HTMLElement**: Selector de menú.
- **modal_search_: HTMLInputElement**: Campo de búsqueda del modal.
- **modal_list_selector_: HTMLUListElement**: Selector de lista del modal.
- **subjects_: string[]**: Array que almacena las asignaturas.
- **existing_chat_subjects_: string[]**: Array que almacena las asignaturas de chat existentes.

Métodos

- **constructor(subject_selector_tag: string, menu_selector_tag: string, modal_search_tag: string, modal_list_selector_tag: string)**: Constructor de la clase SubjectController. Recibe los selectores de los elementos HTML (selector de asignatura, selector de menú, campo de búsqueda del modal y selector de lista del modal).
- **Init(): async**: Inicializa el controlador de asignaturas.
- **GetSubjectSelector(): HTMLInputElement**: Devuelve el selector de asignatura.
- **GetSelectedSubject(): string**: Devuelve la asignatura seleccionada.
- **GetSubjectsFromServer(): Promise<string[]>**: Obtiene las asignaturas del servidor.
- **LoadSubjects(): void**: Carga las asignaturas.
- **LoadExistingChatSubjects(): void**: Carga los chats existentes.
- **LoadExistingChatSubjectsToLateralMenu(): void**: Carga los chat existentes en el menú lateral.
- **LoadSubjectToLateralMenu(subject: string): void**: Crea un nuevo chat en el menú lateral.
- **SearchSubject(subject: string): string[]**: Busca asignaturas por nombre.
- **LoadSubjectsToModalList(): void**: Carga las asignaturas en la lista del modal.

Funcionamiento

La clase SubjectController se encarga de gestionar las asignaturas en la interfaz de usuario, incluyendo la carga desde el servidor, la visualización en el menú lateral y la lista del modal. Permite al usuario seleccionar una asignatura y realizar acciones relacionadas, como la creación de un nuevo chat. La clase también proporciona métodos para inicializar el controlador de asignaturas y realizar búsquedas de asignaturas.

3.1.9. Clase ChatController

A continuación, se listan los atributos y métodos de la clase ChatController junto con una breve descripción de su funcionalidad.

Atributos

- **chat_container_: HTMLElement**: Contenedor del chat.
- **input_div_: HTMLElement**: Div de la entrada.
- **chat_input_: HTMLInputElement**: Campo de entrada del chat.
- **chat_button_: HTMLButtonElement**: Botón de enviar del chat.
- **subject_controller_: SubjectController**: Controlador de asignaturas.
- **session_controller_: SessionController**: Controlador de sesión.
- **getting_answer_: boolean**: Indica si se está obteniendo una respuesta.

Métodos

- **constructor(chat_container_tag: string, input_div_tag: string, chat_input_tag: string, chat_button_tag: string, subject_selector_tag: string, menu_selector_tag: string, modal_search_tag: string, modal_list_selector_tag: string)**: Constructor de la clase ChatController. Recibe los selectores de los elementos HTML del contenedor de chat, div de la entrada, campo de entrada del chat, botón de enviar del chat, selector de asignatura, selector de menú, campo de búsqueda del modal y selector de lista del modal.
- **Init(): void**: Inicializa el controlador de chat.
- **AddClassToChooseSubject(): void**: Añade una clase (CSS) a la etiqueta HTML de la asignatura seleccionada en el menú lateral.
- **RemoveClassToSubjects(): void**: Elimina una clase (CSS) a las etiqueta HTML de las asignaturas del menú lateral.
- **GetQuestion(): string**: Obtiene la pregunta del campo de entrada del chat.
- **ClearQuestion(): void**: Borra el contenido del campo de entrada del chat.
- **GetQuestionAnswer(): Promise<string>**: Obtiene la respuesta a la pregunta del chat desde el servidor.
- **AddMessageToChat(message: Message): void**: Añade un mensaje al contenedor de chat.
- **AddMessageToLocalStorage(message: Message): void**: Añade un mensaje al almacenamiento local del navegador.
- **AddMessage(message: Message): void**: Añade un mensaje al contenedor de chat y al almacenamiento local.

- **LoadChatFromLocalStorage(): void**: Carga el historial de chat desde el almacenamiento local del navegador.

Funcionamiento

La clase ChatController se encarga de gestionar el chat en la interfaz de usuario, incluyendo el envío y recepción de mensajes, la selección de asignaturas y la gestión de la sesión del usuario. Permite al usuario enviar preguntas al servidor y recibir respuestas, además de almacenar el historial de chat en el navegador. La clase también proporciona métodos para inicializar el controlador de chat y cargar el historial de chat desde el almacenamiento local.

3.2. Diseño y Prototipado del *Backend*

3.2.1. Análisis de estrategias

OpenAI embeddings y *Prompt engineering*

Esta estrategia se centra en el uso de *embeddings* de OpenAI (Apartado 2.2.1) para representar textos de manera numérica y facilitar la búsqueda de similitudes entre la pregunta del usuario y el contenido de los documentos. Posteriormente, se utiliza *prompt engineering* para estructurar las consultas al modelo de lenguaje y generar respuestas precisas.

Ventajas La precisión en la búsqueda de información es una ventaja clave, ya que los *embeddings* permiten encontrar fragmentos de texto relevantes basados en la similitud semántica, lo que puede mejorar la precisión de las respuestas. Además, la flexibilidad es otra ventaja importante, ya que la estrategia es adaptable a diversos tipos de documentos y temas, dado que los *embeddings* pueden representar cualquier tipo de texto. Por último, la eficiencia en la reutilización de datos es significativa, ya que al almacenar los *embeddings* generados, se puede reducir el tiempo de procesamiento en consultas futuras sobre los mismos documentos.

Desventajas Entre las desventajas se encuentra la complejidad en la implementación, ya que la generación y manejo de *embeddings*, así como el desarrollo de un sistema de búsqueda de similitud, pueden requerir una implementación técnica compleja. Además, esta estrategia requiere de altos recursos computacionales, puesto que la creación y almacenamiento de *embeddings* pueden consumir una cantidad significativa de tiempo y de espacio. Finalmente, existe una dependencia de la calidad del modelo, ya que la efectividad de las respuestas depende de la calidad del modelo de *embeddings* utilizado y de la pertinencia del *prompt engineering*.

Clasificación de la información junto a *Prompt engineering*

Esta estrategia se enfoca en clasificar la pregunta del usuario para identificar la sección relevante del documento antes de generar una respuesta. Utiliza un modelo de clasificación para determinar la sección adecuada y luego emplea *prompt engineering* para estructurar la consulta y obtener la respuesta.

Ventajas La simplicidad en la búsqueda de información es una ventaja importante, ya que al clasificar la pregunta y dirigirse directamente a la sección correspondiente del documento, se puede simplificar el proceso de búsqueda. Además, esta estrategia es particularmente eficiente en documentos estructurados, como manuales o reglamentos, donde la consistencia en la estructura facilita la localización de la información. Otra ventaja significativa es la reducción de errores, ya que la clasificación previa puede disminuir la probabilidad de errores en la búsqueda, proporcionando respuestas más precisas y pertinentes.

Desventajas Entre las desventajas se encuentra la dependencia en la precisión de la clasificación, ya que si la clasificación de la pregunta no es precisa, la respuesta generada puede no ser adecuada. Además, esta estrategia puede presentar menor flexibilidad, siendo menos efectiva con documentos no estructurados o con variabilidad en la organización del contenido.

Conclusión del análisis de estrategias

Es importante recalcar que si se busca información en documentos con la misma estructura de manera constante, la segunda versión (clasificación de la información junto a *prompt engineering*) es más adecuada. Esta estrategia permite identificar rápidamente la sección correcta del documento y proporcionar una respuesta precisa.

3.2.2. Prototipado. Primera versión usando *OpenAI embeddings*

La primera versión creada seguía la estrategia de usar *OpenAI embeddings* junto a *prompt engineering* para la obtención de respuestas precisas. Analizando el fragmento de código 3.12 podemos extraer el siguiente flujo de acción.

1. Cargamos el archivo PDF correspondiente al tema seleccionado, extrayendo el texto de cada página para su posterior procesamiento.

2. Dividimos el texto en fragmentos manejables utilizando un divisor de caracteres recursivo, configurado con un tamaño y superposición específicos para asegurar una segmentación eficiente.

3. Verificamos si existen *embeddings* previamente generados y almacenados; en caso contrario, los generamos usando *OpenAI Embeddings* y los almacenamos para uso futuro.

4. Realizamos una búsqueda de similitud para encontrar los documentos más relevantes relacionados con la pregunta, seleccionando los dos más pertinentes.

5. Inicializamos el modelo de lenguaje GPT-3.5 y definimos una plantilla del *prompt* que incluye la pregunta y los documentos relevantes.

6. Creamos una cadena de procesamiento utilizando la plantilla del *prompt* y el modelo de lenguaje para generar una respuesta precisa basada en los documentos encontrados.

7. Limpiamos la respuesta generada eliminando saltos de línea y la retornamos para su presentación en la interfaz de usuario.

```python
def answer_question(question: str, subject: str, prompt_template: str, pdf_name: str) ->
    str:
    store_name = subject

    # Load the PDF file
    pdf_reader = PdfReader(f"{pdf_name}.pdf")
    text = ""
    # Extract text from each page of the PDF
    for page in pdf_reader.pages:
        text += page.extract_text()

    # Split text into chunks for efficient processing
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=3500,
        chunk_overlap=750,
        length_function=len
    )
    chunks = text_splitter.split_text(text=text)

    # Load embeddings or generate if not exist
    if os.path.exists(f"{store_name}.pkl") and False:
        with open(f"{store_name}.pkl", "rb") as f:
            VectorStore = pickle.load(f)
    else:
        embeddings = OpenAIEmbeddings()
        VectorStore = FAISS.from_texts(chunks, embedding=embeddings)
        with open(f"{store_name}.pkl", "wb") as f:
            pickle.dump(VectorStore, f)

    # Perform similarity search to find relevant documents
    docs = VectorStore.similarity_search(question, k=2)
    docs_page_content = " ".join([d.page_content for d in docs])

    # Initialize GPT-3.5 language model
    llm = ChatOpenAI(model="gpt-3.5-turbo-0125")

    # Define prompt template
    prompt = PromptTemplate(
        input_variables=["question", "docs"],
        template=prompt_template,
    )

    # Create LLM chain
    chain = LLMChain(llm=llm, prompt=prompt)

    # Generate response using LLM chain
    response = chain.run(question=question, docs=docs_page_content)

    # Clean up response
    response = response.replace("\n", "")

    return response
```

Figura 3.12: Primera versión usando *embeddings*.

3.2.3. Prototipado. Aproximación final usando *prompt engineering*

Tras descartar la versión anterior debido a su volátil precisión en función del modo en el que se realiza la pregunta, para la versión final se siguió la segunda estrategia planteada (clasificación de la información junto a *prompt engineering*).

Guías Docentes

Dentro del marco de este trabajo, se ha llevado a cabo un riguroso análisis de las guías docentes correspondientes a una variedad de asignaturas y a la reglamentación asociada. Estos documentos, esenciales para la planificación y ejecución de la enseñanza universitaria, ofrecen una visión detallada de la estructura y el contenido de cada curso dentro del plan de estudios.

Las guías docentes se dividen en varios apartados, cada uno centrado en aspectos específicos de la asignatura. Estos apartados comprenden desde datos descriptivos de la asignatura hasta la bibliografía.

Teniendo en cuenta esta estructura se ha creado el siguiente array de secciones (figura 3.13).

```
    sections = [
        "Datos descriptivos de la asignatura",
        "Requisitos de matrícula y calificación",
        "Profesorado que imparte la asignatura",
        "Contextualización de la asignatura en el plan de estudio",
        "Competencias",
        "Contenidos de la asignatura",
        "Metodología y volumen de trabajo del estudiante",
        "Bibliografía / Recursos",
        "Sistema de evaluación y calificación",
        "Resultados de Aprendizaje",
        "Cronograma / calendario de la asignatura"
    ]
```

Figura 3.13: Secciones de una guía docente.

Para añadir mayor contexto a cada uno de los apartados de las guías se ha creado un diccionario complementario que ayudará con la tarea de clasificación (figura 3.14). Con esto se consigue que el porcentaje de error al clasificar disminuya drásticamente, ya que por si solo el nombre dela sección puede no ser suficiente para determinadas preguntas.

```
subjects_description = """
    - "Datos descriptivos de la asignatura": Información general sobre la
      asignatura, como nombre, código, créditos, etc.
    - "Requisitos de matrícula y calificación": Condiciones necesarias para
      matricularse y los criterios de evaluación.
    - "Profesorado que imparte la asignatura": Detalles sobre los profesores
      encargados de la asignatura, como nombre, correo electrónico, horarios de
      tutoría, etc.
    - "Contextualización de la asignatura en el plan de estudio": Relación de la
      asignatura con el plan de estudios.
    - "Competencias": Habilidades y conocimientos que se esperan adquirir.
    - "Contenidos de la asignatura": Temas y materias que se tratarán en la
      asignatura.
    - "Metodología y volumen de trabajo del estudiante": Enfoque de enseñanza y la
      carga de trabajo prevista.
    - "Bibliografía / Recursos": Libros, materiales o fuentes recomendadas para la
      asignatura.
    - "Sistema de evaluación y calificación": Detalles sobre cómo se evaluará y
      calificará la asignatura.
    - "Resultados de Aprendizaje": Objetivos específicos de aprendizaje.
    - "Cronograma / calendario de la asignatura": Planificación temporal de la
      asignatura.
"""
```

Figura 3.14: Descripción de las secciones de una guía docente.

Después de establecer la estructura de las guías docentes y proporcionar una descripción detallada de cada sección, se ha desarrollado un sistema de clasificación para identificar la sección correspondiente a una pregunta específica dentro del contexto de la guía docente.

Este sistema de clasificación se ha implementado a través del uso de un modelo de lenguaje y está diseñado para reconocer las secciones específicas mencionadas anteriormente. El sistema recibe una pregunta relacionada con la asignatura y devuelve el nombre de la sección a la que pertenece dicha pregunta, utilizando como referencia el array de secciones definido previamente.

El *prompt* de clasificación proporcionado (ver Figura 3.15) establece las instrucciones para el uso adecuado del sistema, enfatizando la importancia de proporcionar respuestas precisas y evitar información adicional que pueda interferir con el proceso de clasificación.

Este enfoque de clasificación automatizada facilita la organización y el análisis de las preguntas relacionadas con las guías docentes, lo que permite una mejor comprensión de los temas y preocupaciones más comunes planteados por los estudiantes en relación con cada sección de las guías.

```
    subjects_promp_classify = """
        Eres un modelo de lenguaje especializado en clasificar preguntas según su
            sección correspondiente.
        Solo proporcionarás el nombre de la sección a la que pertenece la pregunta.
        Debes usar la sección exacta tal como aparece en la lista, sin agregar puntos,
            comas, etc. La respuesta debe ser precisa.
        Si añades información adicional, el sistema no podrá entenderlo correctamente y
            no funcionará adecuadamente, así que ten cuidado.

        Ejemplos:
            Pregunta: ¿Cuáles son los horarios de tutoría de Israel?
            Respuesta: Profesorado que imparte la asignatura

            Pregunta: ¿Cómo se evalua la asignatura?
            Respuesta: Sistema de evaluación y calificación

        Secciones: {sections}

        Descripción de las secciones para ayudarte a clasificar la pregunta:
            {description}

        Pregunta: {question}
    """
```

Figura 3.15: Ejemplo de *prompt* para la clasificación de preguntas entre los apartados de una guía docente.

Este análisis meticuloso de las guías docentes ha sido fundamental para comprender en profundidad la organización y el contenido de las asignaturas en el contexto del plan de estudios. Asimismo, ha permitido identificar patrones y características comunes que han servido como base para el desarrollo de este Trabajo de Fin de Grado (TFG).

Reglamentación

Al igual que se ha realizado con las guías docentes, también se ha llevado un estudio para identificar patrones dentro de la reglamentación llegando a la extracción de los siguientes capítulos (ver Figura 3.16).

```
    regulation_sections = [
        "CAPÍTULO I. OBJETO Y ÁMBITO DE APLICACIÓN",
        "CAPÍTULO II. RÉGIMEN DE CONVOCATORIAS",
        "CAPÍTULO III. EVALUACIÓN",
        "CAPÍTULO IV. CALIFICACIONES"
    ]
```

Figura 3.16: Capítulos de la reglamentación de guías docentes.

Siguiendo los mismos pasos también se ha creado un diccionario que almacena una breve descripción para cada uno de los capítulos contemplados (ver Figura 3.17).

```
regulation_description = """
    - "CAPÍTULO I. OBJETO Y ÁMBITO DE APLICACIÓN": Define el propósito y alcance de la regulación.
    - "CAPÍTULO II. RÉGIMEN DE CONVOCATORIAS": Detalla el proceso de convocatorias en el contexto de la regulación.
    - "CAPÍTULO III. EVALUACIÓN": Establece los criterios y procesos relacionados con la evaluación.
    - "CAPÍTULO IV. CALIFICACIONES": Describe cómo se asignan y registran las calificaciones.
"""
```

Figura 3.17: Descripción de los capítulos de la reglamentación de guías docentes.

El *prompt* utilizado para la clasificación es el mismo que se ha analizado en el apartado anterior, su carácter reutilizable y genérico brindan una mayor usabilidad y control del código.

Answer Helper

Answer Helper es la función encargada de la obtención de la respuesta final, a través de ella se llevan a cabo los procesos necesarios para tratar la pregunta y recibir una respuesta usando la interfaz de comunicación existente con OpenAI utilizando su *API*.

Tras analizar la función (Figura 3.18) obtenemos las siguientes conclusiones sobre su comportamiento.

1. Inicializamos el cliente de OpenAI con la clave *API* proporcionada para realizar las solicitudes necesarias a los modelos de lenguaje.

2. Medimos el tiempo de inicio de la operación para calcular posteriormente la duración de la misma.

3. Creamos una solicitud al modelo `gpt-3.5-turbo` de OpenAI para clasificar la pregunta en la sección correcta, utilizando un *prompt* que incluye la pregunta, las secciones disponibles y una descripción del proceso de clasificación.

4. Procesamos la respuesta del modelo para extraer y limpiar la sección correcta de los documentos, eliminando caracteres no deseados y formatos innecesarios.

5. Intentamos obtener el contenido de la sección clasificada; si la sección no está disponible, proporcionamos un mensaje de error indicando que no hay información disponible.

6. Realizamos una segunda solicitud al modelo `gpt-3.5-turbo` con un *prompt* que incluye la pregunta y el contenido de la sección clasificada, para generar una respuesta detallada.

7. Calculamos la duración total de la operación restando el tiempo de inicio del tiempo de finalización.

8. Procesamos la respuesta final del modelo para formatearla adecuadamente, reemplazando caracteres específicos para mejorar la legibilidad.

9. Retornamos la respuesta generada y la duración de la operación.

```python
    def answer_question(question, docs_page_content, classify_model, question_model,
        classify_description, api_key):
    client = OpenAI(api_key=api_key)

    start_time = time.time()

    completion = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "user", "content": classify_model.format(question=question,
                sections="".join(docs_page_content.keys()),
                description=classify_description)}
        ]
    )

    correct_section = completion.choices[0].message.content

    correct_section = re.sub(r"\d+\.", "", correct_section)
    correct_section = re.sub(r"\n", "", correct_section)
    correct_section = re.sub(r"Respuesta:", "", correct_section)
    correct_section = re.sub(r"respuesta:", "", correct_section)
    correct_section = re.sub(r"\,", "", correct_section)
    correct_section = re.sub(r'"', "", correct_section)
    correct_section = correct_section.strip()


    try:
        section_content = docs_page_content[correct_section]
    except:
        section_content = "No hay información disponible, prueba a preguntar de otra
            manera."

    completion = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "user", "content": question_model.format(question=question,
                section_content=section_content)},
        ]
    )
    response = completion.choices[0].message.content
    end_time = time.time()

    duration = end_time - start_time

    response = re.sub(r"\;", "\n", response)

    return response, duration
```

Figura 3.18: Función encargada de realizar las consultas a la *API* de OpenAI

Validación

Para la validación de la aplicación se han implementado dos mecanismos fundamentales: la verificación de la *API-Key* y la validación de tokens *JWT* (*JSON Web Token*). A continuación, se describen y explican estos mecanismos y su propósito.

La Figura 3.19 muestra una función que valida la *API-Key* proporcionada por el usuario.

```python
def check_api_key(api_key):
    client = OpenAI(api_key=api_key)
    try:
        client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "user", "content": "Prueba de API key"}
            ]
        )
        return True
    except:
        return False
```

Figura 3.19: Validación de la *API-Key* introducida por el usuario

La función **check_api_key** se encarga de verificar la validez de una *API-Key* proporcionada. Utiliza la biblioteca de OpenAI para intentar crear una solicitud a la *API*. Si la *API-Key* es válida, la función devuelve *True*. Si se produce algún error (por ejemplo, una *API-Key* incorrecta), la función captura la excepción y devuelve *False*.

La Figura 3.20 muestra el decorador creado para la validación del *JWT*.

```python
# Decorador para verificar y decodificar el token JWT
def token_required(func):
    def wrapper(*args, **kwargs):
        token = request.headers.get('Authorization')
        if not token:
            return jsonify({'error': 'Token de autenticación faltante'}), 401

        try:
            payload = jwt.decode(token, SECRET_KEY, algorithms=['HS256'])
            kwargs['api_key'] = payload['api_key']
        except jwt.ExpiredSignatureError:
            return jsonify({'error': 'Token expirado'}), 401
        except jwt.InvalidTokenError:
            return jsonify({'error': 'Token inválido'}), 401

        return func(*args, **kwargs)

    return wrapper
```

Figura 3.20: Decorador creado para la validación del *JWT*

Este decorador verifica si la solicitud contiene un token *JWT* válido en su encabezado. Si el token no está presente, devuelve un error indicando que falta el token de autenticación. Si el token está presente, intenta decodificarlo utilizando una clave secreta y el algoritmo HS256. Si la decodificación es exitosa, extrae la *API-Key* del *payload* del *token* y la pasa a la función envuelta. Si el token ha expirado o es inválido, retorna un mensaje de error adecuado.

Estos mecanismos aseguran que solo los usuarios autenticados y autorizados puedan acceder a las funcionalidades de la *API*, garantizando así la seguridad y el correcto funcionamiento de la aplicación.

3.2.4. *API*: Rutas

La *API* creada para la interacción entre el *Frontend* y OpenAI incluye las siguientes rutas:

- **/set_api_key** (POST):
  - **Descripción:** Esta ruta permite establecer una *API-Key* para autenticación.
  - **Estructura de entrada:** Se espera recibir una clave de *API* en formato *JSON* en el cuerpo de la solicitud (Figura 3.21).

```
{
    "api_key": "your_api_key_here"
}
```

Figura 3.21: Estructura de entrada en formato *JSON* para /set_api_key.

  - **Respuesta:** Devuelve un token *JWT* que se utilizará para autenticar las solicitudes posteriores a la *API* (Figura 3.22).

```
{
    "message": "API key guardada exitosamente",
    "jwt": "your_jwt_token_here"
}
```

Figura 3.22: Respuesta en formato *JSON* para /set_api_key.

- **/get_answer** (GET):
  - **Descripción:** Utilizada para obtener respuestas a preguntas relacionadas con asignaturas (guías docentes). Se debe aportar el *JWT* de autenticación el el *Header* de la solicitud.
  - **Estructura de entrada:** Los parámetros de pregunta y materia se esperan en la *URL* de la solicitud (Figura 3.23).

```
/get_answer?question=your_question_here&subject=your_subject_here
```

Figura 3.23: Estructura de entrada en la *URL* para /get_answer.

  - **Respuesta:** Devuelve una respuesta en formato *JSON* que contiene la respuesta generada por el sistema para la pregunta dada (Figura 3.24).

```
{
    "answer": "your_generated_answer_here"
}
```

Figura 3.24: Respuesta en formato *JSON* para /get_answer.

- **/get_regulation_answer** (GET):
  - **Descripción:** Similar a **/get_answer**, pero utilizado para obtener respuestas a preguntas relacionadas con regulaciones. También se debe aportar el *JWT* de autenticación el el *Header* de la solicitud.
  - **Estructura de entrada:** El parámetro de pregunta se espera en la *URL* de la solicitud (Figura 3.25).

```
/get_regulation_answer?question=your_question_here
```

Figura 3.25: Estructura de entrada en la *URL* para /get_regulation_answer.

  - **Respuesta:** Devuelve una respuesta en formato *JSON* (Figura 3.26).

```
{
    "answer": "your_generated_answer_here"
}
```

Figura 3.26: Respuesta en formato *JSON* para /get_regulation_answer.

- **/documents** (GET):
  - **Descripción:** Devuelve una lista de los documentos disponibles, que incluye materias de estudio y regulaciones.
  - **Estructura de entrada:** No requiere entrada.
  - **Respuesta:** Devuelve una lista de documentos en formato *JSON*, donde cada documento está representado por su nombre (Figura 3.27).

```
[
    {"name": "subject1"},
    {"name": "subject2"},
    {"name": "regulation1"}
]
```

Figura 3.27: Respuesta en formato *JSON* para /documents.

- **/logs** (GET):
  - **Descripción:** Devuelve un registro de las consultas realizadas a la *API*, incluyendo las preguntas, las respuestas y la duración de cada consulta.
  - **Estructura de entrada:** No requiere entrada.
  - **Respuesta:** Devuelve un registro en formato *JSON* que contiene información sobre las consultas realizadas, incluyendo la pregunta, la respuesta y la duración de cada consulta (Figura 3.28).

```
[
    "question1, answer1, duration1",
    "question2, answer2, duration2",
    ...
]
```

Figura 3.28: Respuesta en formato *JSON* para /logs.

3.2.5. Funciones auxiliares

Para el correcto funcionamiento de la clasificación de información es importante descomponer cada guía docente en las distintas secciones mostradas anteriormente. Por ello se ha creado una función auxiliar **GetSectionsFromPDF** (ver Figura 3.29) capaz de generar un diccionario del tipo Sección-Contenido con el que trabajar más cómodamente.

La función recibe el nombre del PDF (ruta absoluta o relativa), las distintas secciones que se esperan extraer y por último una lista de palabras, frases u oraciones que no se añadirán como contenido (filtros).

1. La función crea un objeto `PdfReader` para leer el archivo PDF (se utiliza la librería PyPDF2 (23)). Se inicializan variables para almacenar el texto extraído (`text`), la sección actual (`sectionsExtracted`) y un diccionario para almacenar las secciones con su contenido (`sections_with_text`).

2. La función itera sobre las páginas del PDF y, dentro de cada página, sobre las líneas de texto extraídas. Si se encuentra un delimitador de sección (`section_delimiter`), el texto acumulado hasta ese punto se guarda en el diccionario bajo la sección correspondiente, y se restablece la variable de texto para la siguiente sección.

3. Cuando se alcanza la última sección, el texto restante se añade a esa sección. Las líneas que contienen alguno de los filtros definidos se omiten durante la extracción.

4. La función devuelve un diccionario (`sections_with_text`) donde las claves son los nombres de las secciones y los valores son el texto correspondiente a cada una.

```python
from PyPDF2 import PdfReader
import os

def GetSectionsFromPDF(pdf_name, sections, filters):
    pdf_reader = PdfReader(f"{pdf_name}.pdf")
    text = ""
    sectionsExtracted = 1
    sections_with_text = {}
    lastSection = False
    section_delimiter = sections[sectionsExtracted]
    for page in pdf_reader.pages:
        for line in page.extract_text().split("\n"):
            if lastSection:
                text += line + "\n"
                continue
            if section_delimiter in line:
                currentSection = sections[sectionsExtracted - 1]
                sections_with_text[currentSection] = text
                text = ""
                sectionsExtracted += 1
                if sectionsExtracted == len(sections):
                    lastSection = True
                else:
                    section_delimiter = sections[sectionsExtracted]
            else:
                # si la linea no contiene ninguno de los filtros, añadirla al texto
                if not any(filter in line for filter in filters):
                    text += line + "\n"

    sections_with_text[sections[sectionsExtracted - 1]] = text
    return sections_with_text
```

Figura 3.29: Función creada para separar un PDF en distintas secciones.

3.2.6. Actualización de Guías Docentes

La actualización de las guías docentes incluye varios pasos. Primero, se obtienen los enlaces desde una página web utilizando la función **get_urls**. Luego, se descargan los archivos PDF de estas guías con la función **get_pdfs**.

Este proceso automatizado garantiza que siempre se trabaje con las versiones más recientes de las guías docentes, permitiendo mantener la información actualizada y accesible para su posterior procesamiento y análisis sin necesidad de invertir gran cantidad de recursos a la tarea.

Obtención de enlaces

Para actualizar las guías docentes es necesario primero obtener los enlaces a los documentos PDF desde una página web específica. La función **get_urls** (Figura 3.30) realiza esta tarea y devuelve una lista de enlaces directos a las guías en formato PDF.

Primero se realiza una solicitud HTTP-GET a la *URL* suministrada, a continuación haciendo uso de la librería BeautifulSoup (24), obtenemos todas las etiquetas <a> donde comúnmente se encuentran las direcciones. Finalmente, si el enlace contiene la cade-

na "view_guide" lo modificamos sustituyéndola por "print_guide", en otro caso lo descartamos.

Esto daría como resultado los enlaces directos a la descarga de las guías en formato PDF que se encuentren en la *URL* de inicio.

```
import requests
from bs4 import BeautifulSoup

def get_urls(url):
    try:
        respuesta = requests.get(url)
        respuesta.raise_for_status()

        soup = BeautifulSoup(respuesta.content, 'html.parser')

        enlaces = [enlace['href'] for enlace in soup.find_all('a', href=True)]

        enlaces = ['https://www.ull.es' + enlace.replace('view_guide', 'print_guide')
            for enlace in enlaces if '/view_guide/' in enlace]
#
        return enlaces
    except requests.exceptions.RequestException as e:
        print(f"Error al obtener la página: {e}")
```

Figura 3.30: Función creada para la obtención de *URLs* de guías docente.

Obtención de PDFs

Una vez obtenidos los enlaces a las guías docentes, es necesario descargar los documentos PDF. La función **get_pdfs** (Figura 3.31) se encarga de esta tarea.

La función recibe como parámetro el nombre de un archivo de texto que contiene los enlaces a las guías docentes. Primero, verifica si existe un directorio llamado "pdfs" y lo crea si no existe. Luego, abre el archivo de enlaces y, para cada enlace, realiza una solicitud HTTP-GET para descargar el PDF.

Durante la descarga, la función extrae el nombre original del archivo desde las cabeceras de la respuesta HTTP y lo procesa para formar un nombre de archivo más amigable, utilizando la función **to_camel_case** para transformar el nombre a *camel case*. Finalmente, guarda el PDF en el directorio "pdfs" con el nuevo nombre de archivo.

```python
import requests
import os
import time

def to_camel_case(s):
    words = s.split('_')
    return ''.join([word.capitalize() + '_' for word in words])[0:-1]

def get_pdfs(archivo_enlaces):
    if not os.path.exists('pdfs'):
        os.makedirs('pdfs')

    with open(archivo_enlaces, 'r') as archivo:
        enlaces = [linea.strip() for linea in archivo]
    contador = 1

    for enlace in enlaces:
        try:
            respuesta = requests.get(enlace)

            orig_filename =
                respuesta.headers['Content-Disposition'].split('=')[1].replace("%20",
                "_")
            subject_code = orig_filename.split('_')[2]
            filename = orig_filename.split('-')[1][1:-1]
            filename = to_camel_case(filename)
            filename = subject_code + "-" + filename

            nombre_archivo = 'pdfs/' + filename + '.pdf'

            with open(nombre_archivo, 'wb') as archivo_pdf:
                archivo_pdf.write(respuesta.content)

        except Exception as e:
            print(f"Error al descargar desde {enlace}: {e}")

        finally:
            contador += 1
            # delay para evitar errores de conexión
            if contador % 5 == 0:
                time.sleep(3)
```

Figura 3.31: Función creada para la obtención de PDFs de guías docente.

Capítulo 4

Pruebas y Resultados

4.1. Introducción a las pruebas

Para evaluar la eficacia y la percepción de la plataforma web *chatull.pages.dev*, que integra un *chatbot* diseñado para asistir a los estudiantes, docentes y demás miembros de la Universidad de La Laguna (ULL) en la obtención de información sobre las guías docentes, se creó un formulario de Google. Este formulario permitió a los usuarios proporcionar sus opiniones y experiencias al interactuar con la plataforma. Los resultados obtenidos a partir de esta encuesta se utilizaron para identificar tanto las fortalezas como las limitaciones del sistema, proporcionando una visión clara sobre su desempeño y áreas de mejora.

4.2. Pruebas Realizadas

Durante el desarrollo de la plataforma, se identificaron varias limitaciones inherentes al diseño actual del *chatbot*.

Retención de Contexto El *chatbot* no guarda el contexto de preguntas anteriores, lo que puede llevar a respuestas imprecisas o confusas en conversaciones prolongadas. Esta limitación es deliberada para mejorar el rendimiento del servidor.

Precisión de las Respuestas El *chatbot* es más preciso cuando las preguntas son de carácter individual y específicas a una sección de la guía docente. Las preguntas que requieren integrar información de varios apartados o que no son formuladas de manera particular pueden generar errores o respuestas incompletas, reflejando un uso incorrecto del sistema por parte del usuario.

Usabilidad y Navegación El *chatbot* puede no responder de forma adecuada cuando se quiere referenciar un apartado de la guía usando “`apartado 1`”, “`apartado 2`”, “`apartado final`”, `etc` en vez del nombre del apartado.

4.3. Resultados Obtenidos

Utilización de la Página Todos los usuarios indicaron que la plataforma era sencilla de utilizar, lo que demuestra una buena usabilidad y una interfaz intuitiva.

Utilidad de las Respuestas La gran mayoría de los usuarios indicó que recibieron respuestas útiles a sus preguntas. Otros mencionaron que no les fueron de ayuda. Como es bien conocido, en un diálogo con un *LLM* las preguntas deben ser formuladas correctamente para conseguir un buen funcionamiento del sistema. El *chatbot* es más preciso cuando se le hacen preguntas claras y específicas, y en algunos casos, las limitaciones se impusieron para asegurar la disponibilidad de recursos.

Intención de Uso Futuro La gran mayoría de los usuarios manifestó que volvería a utilizar la plataforma, lo que sugiere una alta satisfacción general con el servicio ofrecido.

Atracción hacia la Implementación del Chatbot La mayoría de los encuestados considera atractiva la idea de implementar un *chatbot* en la web de la universidad, mostrando un interés significativo en la tecnología propuesta.

Problemas Detectados La mayoría de los alumnos no presentó problemas al utilizar la plataforma. Esto sugiere que el sistema está bien diseñado para el público estudiantil. Por otro lado, las respuestas de los profesores fueron divididas. Aquellos que no estaban satisfechos con la plataforma solían formular preguntas de manera inadecuada para el sistema, lo que indica una necesidad de guiar mejor a los usuarios en cómo formular sus preguntas.

4.4. Conclusiones obtenidas de las pruebas

Los resultados de la encuesta muestran un interés positivo y significativo hacia la implementación del *chatbot* en la web de la ULL. La mayoría de los problemas reportados se deben a un uso incorrecto del sistema por parte de los usuarios, lo que sugiere que una mejor guía y entrenamiento en el uso del *chatbot* podría mejorar significativamente la experiencia. Con mejoras enfocadas en la retención de contexto, precisión de respuestas y resolución de errores técnicos, esta herramienta tiene el potencial de convertirse en un recurso valioso y eficiente para la comunidad universitaria.

Capítulo 5

Conclusiones y líneas futuras

La integración de la inteligencia artificial en la educación, a través de chatbots avanzados para interpretar documentos académicos como Guías Docentes y reglamentos de evaluación, representa un avance significativo en la mejora de la experiencia educativa. Estos chatbots, diseñados para proporcionar asistencia personalizada en tiempo real, simplifican la búsqueda de información sobre asignaturas y procedimientos de evaluación para estudiantes, profesores y personal administrativo.

Hasta ahora, se han logrado varios hitos importantes en el desarrollo e implementación de estos chatbots. Se ha conseguido crear una herramienta eficiente que mejora significativamente el acceso a la información, ofreciendo respuestas precisas y rápidas a las consultas académicas. Este enfoque ha facilitado la comunicación en el entorno universitario, promoviendo una actividad educativa más fluida y enriquecedora.

Sin embargo, este progreso enfrenta desafíos. Por ejemplo, el chatbot no guarda el contexto de preguntas anteriores para dedicar más recursos del servidor a mejorar el tiempo de respuesta. Para abordar este inconveniente, se podría habilitar la funcionalidad de guardar contexto en el futuro, mejorando así la coherencia en las respuestas. Además, para asegurar la precisión de las respuestas, se recomienda hacer preguntas de carácter individual respecto a secciones específicas de la guía, en lugar de consultas complejas que abarquen varios temas a la vez. Un futuro desarrollo podría centrarse en optimizar los algoritmos para manejar mejor estas consultas complejas. Finalmente, para evitar errores al referenciar apartados específicos de las guías, es mejor usar el nombre del apartado en lugar de “apartado 1” o “apartado final”. Mejoras futuras podrían incluir un sistema más robusto de reconocimiento de referencias dentro de los documentos.

A pesar de estos desafíos, la implementación de esta herramienta ofrece beneficios tangibles en términos de accesibilidad, eficiencia y claridad en el ámbito educativo. La continua mejora y refinamiento de estos sistemas contribuirá a una integración más efectiva de la IA en la educación, mejorando así la experiencia didáctica para todos los involucrados. Con una actitud proactiva y un enfoque en la resolución de estos desafíos, los beneficios de los chatbots en la educación seguirán creciendo y ofreciendo un valor añadido significativo.

Capítulo 6

Summary and Conclusions

The integration of artificial intelligence in education, through advanced chatbots to interpret academic documents such as teaching guides and assessment regulations, represents a significant advance in improving the educational experience. These chatbots, designed to provide real-time personalised assistance, simplify the search for information on subjects and assessment procedures for students, teachers and administrative staff.

So far, several important milestones have been achieved in the development and implementation of these chatbots. They have succeeded in creating an efficient tool that significantly improves access to information, providing accurate and quick responses to academic queries. This approach has facilitated communication in the university environment, promoting a more fluid and enriching educational activity.

However, this progress faces challenges. For example, the chatbot does not save the context of previous questions in order to dedicate more server resources to improve response time. To address this drawback, the functionality to save context in the future could be enabled, thus improving consistency in responses. In addition, to ensure the accuracy of answers, it is recommended to ask individual questions for specific sections of the guide, rather than complex queries covering several topics at once. Future development could focus on optimising algorithms to better handle these complex queries. Finally, to avoid errors when referencing specific sections of the guides, it is better to use the name of the section rather than “section 1” or “final section”. Future improvements could include a more robust system for recognising references within documents.

Despite these challenges, the implementation of this tool offers tangible benefits in terms of accessibility, efficiency and clarity in the educational environment. The continuous improvement and refinement of these systems will contribute to a more effective integration of AI in education, thus improving the learning experience for all involved. With a proactive attitude and a focus on solving these challenges, the benefits of chatbots in education will continue to grow and offer significant added value.

Capítulo 7

Presupuesto

7.1. Recursos necesarios

Para llevar a cabo el desarrollo del proyecto, se requieren varios recursos esenciales:

- **Ordenador y acceso a internet:** Un ordenador con acceso a internet es fundamental para desarrollar el proyecto y acceder a las herramientas y plataformas necesarias.
- **Plataformas de *hosting*:** Se necesitan plataformas de *hosting* tanto para el *frontend* como para el *backend* del proyecto. Esto puede incluir servicios de alojamiento web pagados para garantizar la disponibilidad y el rendimiento óptimo de la aplicación.
- **Cuenta de GitHub:** Se requiere una cuenta en GitHub para alojar el repositorio del proyecto de forma gratuita. GitHub proporciona herramientas de control de versiones y colaboración que son esenciales para el desarrollo colaborativo del proyecto.
- **Cuenta de OpenAI con fondos:** Para utilizar los servicios de OpenAI, es necesario contar con una cuenta en su plataforma y tener fondos disponibles para acceder a las *API* y recursos computacionales necesarios para el desarrollo y la ejecución del prototipo.

En resumen, los recursos necesarios incluyen un ordenador con acceso a internet, plataformas de *hosting* para el *frontend* y el *backend*, una cuenta de GitHub para alojar el repositorio y una cuenta de OpenAI con fondos para utilizar sus servicios.

7.2. Tiempo y costes necesarios

El sueldo medio internacional de un Ingeniero Informático y la cantidad de horas de trabajo al mes son utilizados para aproximar el cómputo del coste monetario que es necesario invertir para llevar a cabo este proyecto, ya que es de carácter cualificado. Para su cálculo primero se aproxima el valor monetario de una hora de trabajo y se multiplica por las 256 horas totales para completar todas las actividades requeridas.

En españa, 39000€ al año es el salario medio de un Ingeniero Informático según el portal Glassdoor [1], que se traduce a 3250€ al mes. Suponiendo una jornada semanal de 40 horas, que al mes asciende a 160 horas, entonces:

$$salario\ por\ hora = \frac{sueldo\ al\ mes}{horas\ totales\ al\ mes} = \frac{3250\ euros}{160\ horas} = 20{,}3125\ euros/hora.$$

En la tabla 7.1 se puede observar un desglose del tiempo invertido en cada actividad del trabajo y los correspondientes costes monetarios obtenidos mediante la fórmula $salario\ por\ hora \times\ tiempo\ total\ actividad$.

Fase	Tiempo	Coste
Reuniones con los tutores	6 horas	121.875€
Revisión bibliográfica	40 horas	812.5€
Diseño del prototipo	80 horas	1625€
Codificación de la plataforma	95 horas	1929.6875€
Redacción de la memoria	35 horas	710.9375€
Total	**256 horas**	**5200€**

Tabla 7.1: Actividades con el tiempo invertido en ellas y su coste monetario.

7.2.1. Servicio de *hosting*

Para alojar el *backend* del proyecto, se puede optar por servicios de *hosting* como Render. En este caso, el costo mensual del servicio es de 7€. Este gasto es necesario para garantizar la disponibilidad y el rendimiento óptimo del *backend* de la aplicación.

Este coste podría reducirse realizando el despliegue a través de los servicios brindados por la Universidad de La Laguna.

Para alojar el *frontend* se optó por Cloudflare Pages. Este servicio garantiza un despliegue sin cortes y de forma gratuita.

7.2.2. Uso de OpenAI

Para utilizar los servicios de OpenAI, es necesario contar con una cuenta activa y fondos disponibles. En el caso de este proyecto, se ha utilizado OpenAI para generar respuestas a preguntas, y el costo asociado es aproximadamente 0.3€ por cada 100 preguntas generadas. La media de costo por pregunta es de 0.003€. Este costo puede variar dependiendo del uso específico de los servicios solicitados.

Una alternativa al uso de los servicios de OpenAI podría haber sido realizar las peticiones a un *LLM* hospedado por la ULL, de esta forma se reducirían los costes drásticamente y se evitarían problemas derivados de la utilización de servicios de terceros.

[1] Enlace: https://www.glassdoor.es/Sueldos/ingeniero-de-software-sueldo-SRCH_KO0,21.htm.

Apéndice A

Recursos de interés

A.1. Enlace al repositorio donde se encuentra el Frontend de la aplicación web

Todo el código mostrada y explicado se puede encontrar en
https://github.com/ULL-prompt-engineering/chatull-frontend

A.2. Enlace al repositorio donde se encuentra el Backend de la aplicación web

Todo el código mostrada y explicado se puede encontrar en
https://github.com/ULL-prompt-engineering/chatull-backend

Bibliografía

[1] Universidad de La Laguna, “Portal de Guías Docentes de la Universidad de La Laguna.” [Online]. Available: https://www.ull.es/apps/guias/guias/

[2] Universidad de La Laguna, “Reglamento de Evaluación y Calificación de la Universidad de La Laguna.” [Online]. Available: https://riull.ull.es/xmlui/bitstream/handle/915/7847/reglamento_evaluacion_calificacion2016.pdf?sequence=1&isAllowed=y

[3] McKinsey Explainers, “What is prompt engineering?” *McKinsey Company*, 2024, https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-prompt-engineering.

[4] DAIR.AI, “Tecnicas de prompt engineering.” [Online]. Available: https://www.promptingguide.ai/es/techniques

[5] ChatPDF, “Plataforma que permite empezar un chat con un documento pdf, permitiendo realizar preguntas y demás consultas.” [Online]. Available: https://www.chatpdf.com/

[6] AskYourPDF, “Plataforma que permite empezar un chat con un documento pdf, permitiendo realizar preguntas y demás consultas.” [Online]. Available: https://askyourpdf.com/es

[7] LangChain, “Librería LangChain.” [Online]. Available: https://python.langchain.com/docs/get_started/quickstart/

[8] Microsoft, “Explicación de qué es un LLM.” [Online]. Available: https://microsoft.github.io/Workshop-Interact-with-OpenAI-models/llms/

[9] OpenAI, “API de OpenAI.” [Online]. Available: https://platform.openai.com/docs/introduction

[10] OpenAI, “OpenAI Embeddings.” [Online]. Available: https://platform.openai.com/docs/guides/embeddings

[11] Vercel, “Framework Next.js.” [Online]. Available: https://nextjs.org/docs

[12] Jordan Walke, “Framework React.” [Online]. Available: https://es.react.dev/learn

[13] Fred Schott, “Framework Astro.” [Online]. Available: https://docs.astro.build/es/getting-started/

[14] Tailwind Labs, “Framework Tailwind CSS.” [Online]. Available: https://tailwindcss.com/

[15] daisyUI, “Plugin del Framework CSS Tailwind.” [Online]. Available: https://daisyui.com/

[16] Flask, “Framework Flask.” [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/

[17] FastAPI, “Framework FastAPI.” [Online]. Available: https://fastapi.tiangolo.com/

[18] Max Woolf, “The Problem With LangChain,” *Max Woolf’s Blog*, 2023, https://minimaxir.com/2023/07/langchain-problem/.

[19] MDN-contributors, “Explicación del Modelo Vista-ControladorExplicación del Modelo Vista-Controlador.” [Online]. Available: https://developer.mozilla.org/es/docs/Glossary/MVC

[20] MDN-contributors, “Explicación de la propiedad localStorage del navegador.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

[21] Douglas Crockford y Chip Morningstar, “Explicación del formato de los archivos JSON.” [Online]. Available: https://www.json.org/json-es.html

[22] Auth0, “Explicación de los JSON Web Tokens.” [Online]. Available: https://auth0.com/learn/json-web-tokens

[23] Mathieu Fenniak, “Librería Py2PDF2.” [Online]. Available: https://pypdf2.readthedocs.io/en/3.x/

[24] Leonard Richardson, “Librería Beautiful Soup.” [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/