Escuela
Politécnica
Superior

Enriquecimiento de datos de negocios a partir de Google Maps usando Procesamiento del Lenguaje Natural

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Daniel Carreres González
Tutor:
Iván Gadea Sáez

Universitat d'Alacant
Universidad de Alicante

Julio 2025

Enriquecimiento de datos de negocios a partir de Google Maps usando Procesamiento del Lenguaje Natural

Una propuesta de automatización del enriquecimiento de datos georreferenciados mediante scraping y modelos de lenguaje natural en entorno local

Autor
Daniel Carreres González

Tutor
Iván Gadea Sáez
Departamento del tutor

Grado en Ingeniería Informática

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2025

Preámbulo

El objetivo de este Trabajo de Fin de Grado es desarrollar una solución automatizada para enriquecer los datos de Google My Maps, mediante técnicas de web scraping y procesamiento de lenguaje natural obtener respuestas a cuestiones sencillas que pueda realizar un usuario. El sistema permitirá extraer información de manera directa de las páginas webs de los negocios y mediante un modelo de lenguaje previamente entrenado responder de manera específica las preguntas realizadas por el usuario de una manera rápida y eficiente. Todo ello, manteniendo un enfoque en escalabilidad, reutilización de datos y optimización de procesos mediante almacenamiento inteligente.

La idea viene motivada por la necesidad de una herramienta que facilite obtener información rápida y completamente veraz acerca de distintos negocios o cualquier sitio con web y registrado en Google My Maps en un mismo momento. De este modo, podríamos, por ejemplo, consultar los horarios de varios restaurantes que nos pudieran interesar a la vez, revisar el menú de cada uno, en definitiva, consultar cualquier cosa que podríamos ver en las páginas web de cada uno de los sitios pero obteniendo la respuesta para cada uno de ellos de una vez.

Para abordar este objetivo, se ha diseñado una arquitectura modular compuesta por una extensión de navegador, una aplicación de línea de comandos, un sistema de scraping con Scrapy y BeautifulSoup, y una capa de inteligencia basada en modelos de lenguaje natural (LLM) autoalojados, a través de LMStudio. La extensión de navegador permite insertar de forma automática la URL del sitio web de cada punto de interés (POI) en el campo de descripción de Google My Maps. Posteriormente, el archivo KML exportado de dicho mapa se utiliza como fuente para realizar scraping de todas las URLs vinculadas.

Los contenidos extraídos se almacenan y gestionan en una base de datos, evitando duplicidades y optimizando recursos mediante un sistema de control de URLs ya procesadas. El usuario, desde una interfaz por la línea de comandos, puede realizar una consulta en lenguaje natural (por ejemplo, “¿Qué tipo de negocio es?”), y el sistema selecciona las páginas web más prometedoras hasta la profundidad marcada, consulta al LLM y devuelve una respuesta clara y estructurada. Se ha diseñado también un mecanismo de ranking con prompts específicos para determinar qué páginas tienen más probabilidad de contener la respuesta adecuada, reduciendo la carga computacional y mejorando la precisión.

Se han evaluado distintos modelos LLM (como DistilGPT2, LLaMA 2, Nous Hermes y Qwen2.5), valorando su rendimiento en términos de precisión, velocidad y capacidad de ejecución local en dispositivos con recursos limitados.

Los resultados muestran que la solución propuesta permite extraer información útil y veraz

directamente desde las webs de los negocios, sin depender de bases de datos intermedias ni de datos desactualizados. Además, se han identificado oportunidades de mejora futura como el soporte para otros idiomas, la integración con asistentes de voz y la mejora del ranking mediante aprendizaje supervisado. Este trabajo contribuye al desarrollo de herramientas accesibles basadas en IA que permiten explotar mejor los datos georreferenciados y mejorar la experiencia de los usuarios.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1. Introducción

Hoy en día, se está rodeado de datos por todas partes. Son usados constantemente, en ocasiones sin querer, para tomar decisiones tanto en lo personal como en lo profesional. Sin embargo, no siempre hacen falta análisis complejos ni bases de datos enormes. A veces lo que se requiere es algo mucho más simple, como el horario de apertura de un local, su número de teléfono o su web.

Para eso, herramientas como Google My Maps, un servicio lanzado por Google en 2007 que permite crear mapas personalizados a los usuarios para uso propio o compartido, permiten marcar ubicaciones, trazar líneas y añadir formas en Google Maps. Herramientas como esta pueden ser útiles, pero pueden resultar insuficientes. Es posible almacenar establecimientos en un mapa, pero, para obtener más información sobre cada ubicación, es necesario realizar una búsqueda en su página web. Y si hay muchos sitios, eso se vuelve pesado y poco eficiente.

El formato nativo utilizado por Google My Maps para manejar los puntos de interés (POI, por sus siglas en inglés) es el KML. Los archivos KML (Keyhole Markup Language File) son archivos con un formato basado en XML (Extensible Markup Language) para almacenar datos geográficos y su contenido relacionado. Este formato permite almacenar puntos, líneas, polígonos e imágenes en un solo archivo.

Se propone el desarrollo de una herramienta que permita al usuario seleccionar varios negocios en Google My Maps y, a partir de ahí, obtener información más detallada y específica directamente desde sus páginas web. Todo ello formulando una única consulta, escrita en lenguaje natural, del tipo: "¿Cuál es el horario, la dirección y el teléfono de este sitio?".

Para responder a las preguntas de los usuarios a partir de los archivos, se emplea un modelo de lenguaje natural (LLM, por sus siglas en inglés), que extrae las respuestas solicitadas a partir del HTML de las páginas web. Los modelos de lenguaje natural (LLM) son algoritmos de inteligencia artificial diseñados para reconocer y generar texto. Se entrenan con grandes cantidades de datos para aprender patrones en el lenguaje. Los LLM utilizan redes neuronales, que son estructuras que permiten a las máquinas procesar la información de manera similar al cerebro humano. A través de un proceso llamado aprendizaje profundo, estos modelos mejoran continuamente al aprender de sus errores.

El web scraping, técnica que se emplea para obtener el código HTML de cada una de las páginas web y explorar sus diferentes niveles, es el proceso de extraer datos de páginas web de manera automatizada mediante herramientas o programas especializados. El proceso consiste en acceder a la página web y, según el caso, extraer todos los datos para un posterior refinamiento o solo una parte de ellos, dependiendo de los requerimientos específicos

de la aplicación. Hoy en día, el web scraping tiene múltiples aplicaciones, desde recopilar información de competencia en el sector empresarial hasta monitorear redes sociales. Una de las ventajas del web scraping es que permite realizar tareas que serían extremadamente laboriosas de forma manual. Por ejemplo, recolectar datos de cientos o miles de páginas de productos de diferentes tiendas online, sin tener que acceder a cada página individualmente.

Una vez introducidos algunos de estos conceptos relevantes para comprender la arquitectura propuesta y el desarrollo de la aplicación, se procede a explicar cómo estas preguntas realizadas a un terminal nos devolverán unas respuestas extraídas directamente de las páginas web de los sitios.

El proyecto se divide en dos partes. La primera es una extensión de navegador que recoge los puntos seleccionados en el mapa y extrae la URL de cada negocio colocándola en el campo descripción de cada ubicación. Esa información se guarda en un archivo KML, que sirve como enlace entre la herramienta del usuario y el siguiente paso del proceso.

La segunda parte consiste en una aplicación que toma ese KML como entrada, visita automáticamente las páginas web, extrae hasta cierta profundidad su código HTML y, usando un modelo de lenguaje natural previamente entrenado, extrae los datos que el usuario ha pedido.

En lugar de entrenar un modelo propio, se opta por integrar modelos existentes que se puedan usar en local, sin necesidad de depender de servicios externos ni gastar recursos innecesarios. Esto también ayuda a mantener el control sobre los datos tratados y facilita la reutilización del sistema por parte de otras personas o empresas con menos medios técnicos.

Uno de los aspectos clave del proyecto es la flexibilidad: el usuario no tiene que saber nada de scraping ni de modelos de lenguaje. Solo debe preparar su mapa en My Maps, lanzar la aplicación con un par de comandos, y recibir como resultado la información que necesita lanzada por terminal, también almacenada en un pequeño fichero JSON y en una base de datos.

Además, el sistema incluye algunas protecciones, como la limitación de profundidad en la navegación web, para evitar sobrecargar las páginas de destino o quedarse atrapado en enlaces innecesarios.

1.1. Justificación

El origen de este proyecto se basa en las limitaciones de las herramientas de mapas personalizables como Google My Maps, que, aunque útiles para marcar ubicaciones, no permiten acceder de manera eficiente a información detallada sobre los negocios o puntos de interés. Esta deficiencia genera una discrepancia entre la capacidad de visualizar lugares en un mapa y la necesidad de obtener datos concretos de esos lugares, como horarios, teléfonos o servicios disponibles. En muchos casos, los usuarios se ven obligados a realizar búsquedas manuales de estos datos, lo que no solo es lento, sino también propenso a errores.

Este Trabajo de Fin de Grado se fundamenta en esa idea. Al identificar una limitación real en el uso de estos mapas personalizados, se buscó una forma de facilitar las cosas. Lo que se plantea aquí es una manera de automatizar esa búsqueda y de conseguir, de forma rápida, información útil sobre varios negocios simultáneamente.

En definitiva, el objetivo de este proyecto ha sido diseñar algo que sea fácil de usar, adaptable a diferentes contextos y que no requiera demasiados recursos para funcionar. Aprovecha tecnologías modernas que ya están disponibles, como los modelos de lenguaje y técnicas de scraping, pero sin añadir complejidad innecesaria para el usuario. Todo con la idea de hacer más útil la información que ya está en los mapas, sin tener que buscarla manualmente cada vez. luego aplícalas en tus documentos.

1.2. Marco teórico

1.2.1. KML

Los archivos KML (Keyhole Markup Language) son documentos basados en XML diseñados para representar información geográfica, y son utilizados por aplicaciones como Google Earth y Google My Maps para visualizar lugares, rutas y datos especiales.

Contienen datos geoespaciales tales como:

- Marcadores de ubicación (coordenadas GPS).
- Rutas o caminos (conectando puntos en el mapa).
- Polígonos (para delimitar áreas geográficas).
- Estilos (colores, iconos, etiquetas).
- Descripciones (textos o enlaces para cada punto).

Se trata de un lenguaje de marcado que organiza estos datos de manera jerárquica utilizando etiquetas XML.

1.2.2. LLM (Large Language Model)

Los modelos de lenguaje de gran tamaño son modelos de aprendizaje profundo preentrenados con grandes volúmenes de datos. Están basados en la arquitectura *transformer*, un tipo de red neuronal profunda que revolucionó el procesamiento del lenguaje natural y otras tareas de IA al introducir un mecanismo denominado *atención*, que básicamente decide qué partes del texto son más importantes para cada palabra, permitiendo capturar relaciones a largo plazo de manera eficiente.

Un modelo *transformer* estándar se compone de un codificador y un decodificador. El codificador recibe una secuencia de entrada y extrae representaciones internas en forma de

vectores que capturan el significado contextual de cada palabra. A partir de estas representaciones, el decodificador genera la salida. Ambos componentes utilizan capas de atención y redes neuronales *feedforward* (de avance directo) que permiten manejar secuencias complejas de manera paralela y eficiente.

El mecanismo de atención (*Self-Attention*) permite al modelo ponderar cada palabra de entrada según su relevancia en el contexto. El proceso se describe de la siguiente forma: se calculan tres vectores por palabra: *Query* (Q), *Key* (K) y *Value* (V). Primero se compara Q de una palabra con todos los K de la secuencia, se asignan pesos a cada palabra con una función *softmax* y se calcula una combinación ponderada de los V para producir la salida. Este proceso se repite varias veces.

Las aplicaciones de los modelos de lenguaje son diversas y abarcan áreas como la traducción automática, el análisis de datos, el resumen de textos, y la creación de chatbots y asistentes virtuales. En este caso se emplearán para extraer información de un HTML previamente procesado para contestar una pregunta que realice el usuario; también se usarán para calificar en cuál de los HTML es más probable encontrar la respuesta, basándose en la URL, y buscarla en ese orden.

1.2.3. Web Scraping

El *web scraping* es una técnica utilizada para extraer datos de manera automatizada del contenido de sitios web. Mediante el uso de programas que simulan la navegación humana para acceder al contenido de una página, analizar su estructura y recuperar los datos que sean de interés. Esta práctica es ampliamente utilizada para extraer datos como opiniones en redes sociales, precios en distintos portales de venta, noticias sobre distintos temas e información sobre competidores en un mercado, entre otros.

Para profundizar en el proceso que se sigue para el *scraping*, este generalmente comienza con el envío de una petición HTTP a una página web. Una vez se recibe la respuesta, normalmente en formato HTML, el *scraper* puede analizar el documento utilizando herramientas de análisis del DOM (*Document Object Model*) para localizar los elementos deseados.

Las herramientas más comunes para realizar *web scraping* incluyen bibliotecas como `BeautifulSoup`[1] y `requests`[2], así como *frameworks* más avanzados como `Scrapy`[3], que permiten manejar múltiples páginas.

Aunque el *web scraping* es una técnica poderosa con aplicaciones extensas, presenta ciertas limitaciones legales, éticas e incluso técnicas que deben ser consideradas. Uno de ellos es el archivo `robots.txt`, un archivo de texto ubicado en la raíz de un dominio web que indica a los *bots*, en este caso al *scraper*, qué partes del sitio están permitidas y cuáles no para su

[1] `https://www.crummy.com/software/BeautifulSoup/`
[2] `https://www.python.org/`
[3] `https://www.scrapy.org/`

acceso automatizado. Aunque este archivo no impide técnicamente el acceso a las páginas prohibidas y, si se desea, puede ser eludido, establece una norma de buena conducta que debe ser respetada. Ignorarlo se considera una infracción de los términos de servicio y puede acarrear consecuencias legales.

En ocasiones, las páginas web implementan medidas *anti-scraping* para prevenir el acceso de programas diseñados para eludir estos términos de buena conducta, como los conocidos *CAPTCHAs*, límites de acceso o cambios frecuentes en la estructura del HTML.

2. Estado del arte

2.1. Introducción al PLN

Las primeras técnicas de *procesamiento de lenguaje natural* (PLN) se basaron principalmente en métodos lingüísticos y reglas programadas, centradas en la identificación de patrones específicos en el texto. Algunas de estas reglas clásicas son:

- **Expresiones regulares (Regex)**: se usaron ampliamente para tareas como la extracción de correos electrónicos, URLs o fechas. Son rápidas y eficaces en la identificación de patrones simples; sin embargo, su capacidad para manejar la variabilidad del lenguaje es limitada, ya que no comprenden el contexto ni las relaciones entre palabras.
- **Etiquetado de Partes del Habla (POS Tagging)**: consiste en asignar a cada palabra de un texto la categoría morfosintáctica adecuada (sustantivo, adjetivo, etc.). Esta técnica utiliza gramáticas formales y algoritmos basados en reglas o modelos probabilísticos. Es útil para la extracción de relaciones sintácticas, pero no puede capturar las complejas relaciones semánticas entre palabras o frases.
- **Reconocimiento de Entidades Nombradas (NER)**: técnica clásica utilizada para identificar entidades como personas, lugares u organizaciones. Utiliza reglas lingüísticas y modelos estadísticos basados en características del texto. También tiene dificultades para manejar variaciones lingüísticas o reconocer entidades ambiguas.

Para avanzar fue crucial el desarrollo de los *word embeddings*, que permitieron representar las palabras en un espacio vectorial basado en su contexto, marcando una transición a representaciones más ricas y precisas del lenguaje. Los *word embeddings* son vectores densos, distribuidos y de longitud fija que se construyen utilizando estadísticas de coocurrencia de palabras, según la hipótesis distribucional. Estos vectores representan palabras en un espacio vectorial de manera que aquellas semánticamente relacionadas o que aparecen en contextos similares se encuentran próximas entre sí.

El avance que terminó de revolucionar el mundo del PLN fue la introducción de los modelos de *deep learning*, específicamente los *transformers*. Inicialmente se destacaron por su rendimiento en la traducción automática; sin embargo, hoy en día se han convertido en el estándar para construir sistemas de autoaprendizaje autosupervisado a gran escala. Son ampliamente utilizados en procesamiento de lenguaje natural y han comenzado a aplicarse con éxito en visión por computadora y procesamiento multimodal. Emplean *word embeddings* para capturar relaciones semánticas entre palabras y mecanismos de atención que permiten a la red neuronal enfocarse en diferentes partes de una secuencia de entrada.

Los *transformers* representan una evolución que integra los avances previos de estos subcampos del aprendizaje profundo, combinándolos para crear un modelo más potente y eficiente. Las ventajas principales de los *transformers* frente a modelos anteriores son:

- **No dependen de redes neuronales recurrentes (RNN) o convolucionales (CNN)**: a diferencia de las RNN y CNN, los *transformers* utilizan mecanismos de atención y redes neuronales de avance. No procesan las palabras en un orden específico, sino que consideran toda la secuencia en conjunto.
- **Mecanismo de Self-Attention**: este mecanismo permite capturar relaciones globales y dependencias entre las palabras de una frase, facilitando la tarea de modelar el contexto.
- **Flexibilidad**: los *transformers* constituyen arquitecturas muy flexibles, fácilmente modificables para adaptarse a una gran cantidad de tareas.

A medida que los *transformers* han madurado, han llevado a la creación de modelos de lenguaje de gran escala (*Large Language Models*, LLM), como *GPT* o *BERT*, que han mostrado un rendimiento extraordinario en una amplia variedad de tareas relacionadas con el PLN.

2.2. LLMs como herramientas para la extracción de información flexible basada en *prompts*

Los *Large Language Models* (LLMs) han demostrado ser herramientas muy poderosas para tareas complejas de extracción de información, especialmente cuando se usan de manera flexible y basada en *prompts*. Los LLMs pueden adaptarse a una variedad de tareas de extracción de datos a partir de grandes volúmenes de texto, sin necesidad de datos etiquetados o entrenamiento especializado.

El uso de *prompts* (instrucciones o preguntas del usuario) es esencial para guiar al modelo en la extracción de información específica de un texto. El modelo responde a estos *prompts* generando un mensaje que contiene la información solicitada.

Los LLMs pueden utilizarse para extraer datos de textos desestructurados (como se hace en este TFG), o para identificar relaciones entre entidades, todo a partir de un *prompt* adecuado que dirija al modelo sobre qué hacer con el texto en cuestión.

Ventajas

- **Flexibilidad**: al usar un *prompt* adecuado, los modelos pueden adaptarse a diferentes tareas sin necesidad de rediseñar el modelo. Esto permite que un solo modelo pueda realizar una amplia gama de funciones.
- **Escalabilidad**: los LLMs son capaces de procesar grandes cantidades de datos, convirtiéndolos en una opción ideal para tareas como el *web scraping*.

- **Generación de resultados estructurados**: a través de un diseño adecuado de los *prompts*, se generan salidas estructuradas fácilmente utilizables por el consumidor final.
- **Sin necesidad de datos etiquetados extensivos**: a diferencia de otros enfoques de aprendizaje, los LLMs pueden utilizarse directamente con un *prompt* bien formulado para adaptarse a una nueva tarea sin necesidad de entrenamiento adicional.

Limitaciones

- **Dependencia de la calidad del *prompt***: el rendimiento depende en gran medida de la calidad y claridad del *prompt*; si no es claro o es ambiguo, el LLM puede generar respuestas incorrectas o no deseadas.
- **Desempeño en casos complejos**: su comportamiento puede no ser del todo satisfactorio en tareas muy complejas o con información poco clara o ambigua.
- **Costos computacionales**: los LLMs pueden ser costosos en términos de recursos para entrenar y ejecutar, lo que representa un desafío cuando se pretende realizar extracción de información a gran escala en tiempo real.

2.3. Métodos de *prompt engineering* y *parsing* de resultados generados por LLMs

El *prompt engineering* y el *parsing* de resultados generados por LLMs son clave para maximizar la eficiencia y la precisión. Estos métodos permiten guiar a los modelos en la generación de respuestas específicas y útiles.

El *prompt engineering* es el proceso mediante el cual se diseñan y estructuran instrucciones o preguntas para guiar el comportamiento de los LLMs, maximizando la probabilidad de obtener respuestas útiles y relevantes.

Técnicas de *prompt engineering*

- **Prompts directos**: instrucciones claras y directas para que el modelo realice una tarea específica.
- **Prompts indirectos**: utilizan el contexto o la estructura del texto para guiar al modelo hacia la respuesta correcta.
- **Instrucciones claras y concisas**: la claridad en la formulación del *prompt* es esencial; si no es claro o es ambiguo, se pueden generar respuestas no deseadas.
- **Uso de ejemplos**: mediante el uso de ejemplos dentro del *prompt* se puede facilitar que el modelo entienda mejor la tarea. Este enfoque se denomina *few-shot learning*.
- **Prompts con contexto específico**: incluir el contexto completo o enfocar al modelo en una parte concreta del texto resulta útil especialmente con textos largos.

- **Prompts de pregunta-respuesta**: se emplean en casos donde se busca obtener respuestas directas y concretas a partir de un texto.

Métodos de *parsing* de resultados

El *parsing* de resultados se refiere al proceso de interpretar y estructurar la salida generada por los LLMs de manera que sea útil y fácilmente procesable.

- **Extracción de datos estructurados**: los resultados pueden procesarse y transformarse en datos estructurados usando *parsers* que extraen valores clave y los organizan en un formato definido.
- **Análisis de relaciones complejas**: cuando se extraen relaciones entre varias entidades, el *parsing* puede estructurar la salida en tuplas o en gráficos de conocimiento.
- **Generación de resúmenes estructurados**: el *parsing* se utiliza también para generar resúmenes de documentos extensos, ayudando a identificar los puntos clave.

2.4. Modelos de LLM adaptados a GPUs de consumo (modelos "Edge" o *Self-Hosted*)

Los LLM como *GPT*, *LLaMA* o *BERT* han demostrado tener un rendimiento excepcional en procesamiento de lenguaje natural. Sin embargo, estos modelos requieren capacidades computacionales masivas y, aunque muchos ofrecen funciones de consulta mediante API, estas suelen ser de pago.

Para hacer frente a estas limitaciones, se han desarrollado modelos "edge" o *self-hosted*, versiones más pequeñas y eficientes de los modelos grandes, optimizadas para su uso en hardware de gama media.

Estos modelos permiten a los usuarios ejecutar sistemas de lenguaje potentes de manera local, reduciendo así tanto los costes económicos como computacionales, y aumentando la privacidad y el control sobre los datos procesados.

Para reducir el tamaño de estos modelos se emplea la *cuantización*, una técnica que implica disminuir la precisión de los cálculos numéricos, lo que permite ejecutar versiones más ligeras de los LLM en dispositivos con recursos limitados. Aunque estos modelos tienen menor tamaño y rendimiento, ofrecen ventajas significativas en términos de eficiencia y autonomía.

En el caso de este TFG, era necesario un modelo que soportara cierto contexto, ya que se le iba a pasar un *prompt* con una pregunta y un HTML ligeramente procesado. Para ello se investigaron distintos LLMs mediante la plataforma LMStudio, que permite lanzar modelos en local y facilita su uso desde terminal a través de una API local.

A continuación se detallan los modelos probados y sus características más relevantes:

Modelos evaluados

- **DistilGPT2**
  - Tamaño: 82M parámetros.
  - Requerimientos: GPU de gama baja.
  - Rendimiento: Ligero y rápido, pero limitado en la generación de texto complejo.
  - Capacidad del contexto: aprox. 1,024 tokens.
- **LLaMA-2-7B**
  - Tamaño: 7B parámetros.
  - Requerimientos: GPU de gama media.
  - Rendimiento: Competente en tareas de PLN como preguntas y resúmenes.
  - Capacidad del contexto: aprox. 4,096 tokens.
- **Qwen1.5-4B-Chat-GGUF**
  - Tamaño: 4B parámetros.
  - Requerimientos: GPU de gama baja a media.
  - Rendimiento: Bueno para tareas tipo chatbot, pero limitado en generación larga.
  - Capacidad del contexto: aprox. 2,048 tokens.
- **Nous-Hermes-2-Mistral-7B-DPO-GGUF**
  - Tamaño: 7B parámetros.
  - Requerimientos: GPU de gama media, mayor carga por arquitectura DPO.
  - Rendimiento: Buen rendimiento general, más lento en tareas con alto volumen.
  - Capacidad del contexto: aprox. 4,096 tokens.
- **Qwen2.5-7B-Instruct-1M-GGUF**
  - Tamaño: 7B parámetros.
  - Requerimientos: GPU de gama media (mínimo 8 GB VRAM).
  - Rendimiento: Excelente en generación de texto y respuestas.
  - Capacidad del contexto: aprox. 4,096 tokens.

Tabla 2.1: Tabla resumen de modelos

Modelo	Tamaño	Requisitos	Rendimiento	Contexto
DistilGPT2	82M	GPU baja	Ligero y rápido, limitado para texto complejo	1,024
LLaMA-2-7B	7B	GPU media	Competente, no siempre concreto	4,096
Qwen1.5-4B	4B	GPU baja/media	Bueno para chat, limitado para textos largos	2,048
Nous-Hermes-2	7B	GPU media	Buen rendimiento, lento para grandes volúmenes	4,096
Qwen2.5-7B	7B	GPU media (8GB)	Excelente equilibrio entre rendimiento y eficiencia	4,096

3. Objetivos

El objetivo principal de este Trabajo de Fin de Grado es enriquecer la información que ofrecen los mapas de Google con datos más detallados y específicos según las necesidades del usuario, con el fin de facilitar la toma de decisiones en diversas áreas como el marketing, la gestión empresarial o la investigación de mercados.

La aplicación estará dividida en dos partes principales: una extensión, que se encargará de procesar el fichero KML de Google My Maps escribiendo las URLs de cada uno de los puntos de interés (POIs) en el campo de descripción que estos contienen; y una aplicación que realizará un proceso de *scraping* de dichas webs, almacenando su HTML procesado y pasando una pregunta realizada por el usuario a un modelo LLM, el cual devolverá la respuesta extraída directamente del contenido de esas páginas.

3.1. Objetivos específicos

Se plantean los siguientes objetivos para alcanzar el propósito final del TFG:

- Aprender acerca de Google My Maps y familiarizarse con la estructura de los ficheros KML con los que trabaja.
- Adquirir nociones básicas sobre extensiones de Google Chrome, el archivo `manifest.json` y fundamentos de JavaScript.
- Comprender el funcionamiento del DOM, cómo interactuar con elementos HTML, manejar temporizadores y promesas, así como gestionar eventos del DOM para simular clics u otras interacciones como tecleos.
- Profundizar en el funcionamiento del DOM específico de Google My Maps, identificando los elementos necesarios para simular las interacciones y escribir en los campos adecuados.
- Investigar el *web scraping* en profundidad y diseñar el método más eficiente posible, seleccionando la herramienta más adecuada para el proyecto.
- Aprender los conceptos básicos de la herramienta `Scrapy` y desarrollar un programa que obtenga el código HTML de aquellas páginas que lo permiten.
- Desarrollar un sistema que permita extraer también el contenido de distintas profundidades de la URL raíz, en función de un parámetro definido por el usuario.
- Evaluar distintos modelos LLM para determinar cuál cumple mejor los requisitos de la aplicación, mediante pruebas con preguntas y HTMLs reales.

- Diseñar una aplicación que tome el texto HTML y lo combine con la pregunta del usuario para formar un *prompt* robusto y contextualizado, que se enviará al modelo LLM.
- Realizar una evaluación cuantitativa y cualitativa de la utilidad, calidad y precisión de la aplicación desarrollada mediante pruebas exhaustivas.
- Analizar los resultados obtenidos de dichas evaluaciones, con el fin de extraer conclusiones sobre la efectividad del sistema, identificar sus limitaciones y proponer posibles mejoras para el futuro.

4. Metodología

La metodología de trabajo seleccionada para este proyecto ha sido una metodología ágil adaptada, combinando SCRUM con un enfoque de Kanban. Esta metodología promueve la entrega continua de valor, permite gestionar el trabajo de manera flexible y favorece la adaptación en cada fase del proyecto.

4.1. SCRUM adaptado

La metodología SCRUM se adapta al contexto del trabajo de la siguiente manera:

1. Roles

- **Product Owner (Estudiante)**: el estudiante asume el rol de Product Owner, definiendo los requisitos del proyecto.
- **Desarrollador (Estudiante)**: el estudiante desempeña el rol de desarrollador completo del proyecto, realizando todas las tareas propias dentro de cada sprint.
- **Scrum Master (Estudiante)**: gestiona los bloqueos, facilita reuniones y asegura el cumplimiento de los objetivos del sprint.
- **Cliente (Tutor)**: el tutor actúa como cliente, proporcionando requisitos y *feedback* durante el proceso.

2. Sprints

El proceso se organiza en sprints de aproximadamente tres semanas, donde se avanza de forma iterativa y entregable. Cada sprint se enfoca en completar un conjunto específico de tareas, alineadas con los hitos del proyecto.

3. Adaptación continua

Siguiendo el enfoque ágil, cualquier elemento del *backlog* puede ser ajustado en cualquier momento, ya sea dividiéndolo, reordenándolo o modificándolo según las nuevas necesidades del proyecto.

Antes de comenzar el trabajo, el tutor expresa una serie de necesidades, ideas y deseos. Estos se convierten en historias de usuario que siguen el formato clásico de SCRUM. Las historias de usuario se validan y priorizan de acuerdo a la importancia y al valor que aportan al proyecto.

4.2. Planificación del sprint

Cada sprint comienza con la selección de tareas del *backlog* del producto, compuesto por todos los requisitos del trabajo. Las tareas se asignan y se priorizan en función de los objetivos de cada sprint.

4.3. Ejecución del sprint

Durante la ejecución del sprint, se utiliza la herramienta Trello para gestionar el progreso de las tareas. Al inicio de cada jornada, se realiza un *Daily Scrum* individual, donde se registra lo siguiente:

- ¿Qué hice ayer?
- ¿Qué haré hoy?
- ¿Qué problemas tengo?

Este proceso permite identificar bloqueos y ajustar prioridades de manera continua. El diario de progreso facilita la organización diaria y el seguimiento de tareas.

Figura 4.1: Imagen del tablón de tareas en Trello

4.4. Sprint Review y Feedback

Al finalizar cada *Sprint*, se presenta un avance tangible al tutor, quien proporciona *feedback* sobre el trabajo realizado. Se revisa si las tareas completadas cumplen con los objetivos

establecidos. El *feedback* recibido permite ajustar prioridades y tareas futuras para el siguiente *Sprint*.

5. Análisis y diseño

5.1. Análisis

Para comenzar con el proyecto, lo primero fue observar las partes que lo conformaban, en este caso, eran dos partes clave: la primera, **Google My Maps**, que era la aplicación en la que se marcaban los POIs y por lo tanto el origen de las URLs de las empresas a las que queríamos consultar; y la segunda, la aplicación desarrollada para ser utilizada por terminal, que debía hacer el *scraping* de las webs de los negocios marcados y contestar a las preguntas del usuario.

Comenzando por la parte de Google My Maps, en primera instancia se trató de marcar los POIs y exportar el fichero KML. Sin embargo, los campos del contenedor de información no se guardaban. Sí se guardaba un pequeño campo de descripción, por defecto vacío, en el que era posible escribir un pequeño mensaje —en este caso, la URL raíz del sitio web—.

Como no era viable asociar esta tarea de escritura de la URL al usuario, por comodidad y porque la aplicación se plantea para un número variable de negocios, se optó por una extensión de navegador que automatiza este proceso: copia del contenedor de información la URL y la transfiere al apartado de descripción del POI.

En segundo lugar, la aplicación de consulta por terminal recibiría el fichero KML para iterar por las diferentes URLs de los POIs. Para evitar realizar el *scraping* de la misma web varias veces, se planteó una base de datos en la que almacenar el código HTML de las URLs ya procesadas. Las preguntas se pasarían al modelo LLM mediante un *prompt* que incluye tanto la pregunta como el HTML procesado. El modelo respondería a partir de ese contenido, y la respuesta se almacenaría en un fichero `.json` junto con la pregunta y la URL de origen.

Para el correcto desarrollo del proyecto, el primer paso fue un análisis de las tareas necesarias para su implementación, así como la identificación de los requisitos funcionales y no funcionales. A partir de este análisis se definieron las tareas de un *Backlog* que guiaron el desarrollo del sistema.

5.1.1. Requisitos funcionales

A continuación se detallan los requisitos funcionales necesarios para la correcta implementación del sistema:

- **RF-001: Extracción de Datos de Google My Maps**
  - **Objetivo:** Cargar en la descripción del POI la URL del sitio.
  - **Descripción:** Permitir al usuario marcar distintos POIs y automatizar el copiado de la URL en el campo de descripción del POI, para la futura exportación.

- **RF-002: Web Scraping para Extracción de Datos**
  - **Objetivo:** Extraer el HTML en crudo de las URLs extraídas.
  - **Descripción:** Mediante scraping, extraer los HTML del sitio raíz accediendo hasta la profundidad marcada por el usuario.

- **RF-003: Limpieza del HTML y Almacenado**
  - **Objetivo:** Limpiar el HTML y almacenarlo.
  - **Descripción:** Eliminar etiquetas innecesarias y partes propias del formato HTML que no contienen información relevante.

- **RF-004: Organización de las URLs para contestar la pregunta**
  - **Objetivo:** Agilizar y mejorar la calidad de la respuesta.
  - **Descripción:** Enviar al LLM todas las URLs de una misma raíz para que priorice el orden de consulta según la probabilidad de contener la respuesta.

- **RF-005: Consulta de la información mediante Lenguaje Natural**
  - **Objetivo:** Permitir al usuario realizar preguntas en lenguaje natural.
  - **Descripción:** Utilizar un LLM para responder a las preguntas del usuario a partir del HTML procesado, mediante un prompt.

- **RF-006: Almacenamiento y Gestión de Datos**
  - **Objetivo:** Evitar repetir procesos y mejorar el rendimiento.
  - **Descripción:** Guardar el HTML procesado en base de datos para que no se vuelva a scrapear la misma URL.

Tabla 5.1: Resumen de requisitos funcionales

ID	Requisito	Descripción
RF-001	Extracción de Datos de Google My Maps	Cargar la URL del negocio en el campo de descripción del POI para exportación en KML.
RF-002	Web Scraping para Extracción de Datos	Obtener el HTML completo de las URLs, recorriendo enlaces hasta la profundidad especificada.
RF-003	Limpieza del HTML y Almacenado	Eliminar contenido irrelevante del HTML y almacenarlo para su posterior uso.
RF-004	Organización de URLs para la consulta	Priorizar URLs por dominio según su relevancia para responder a una pregunta.
RF-005	Consulta en Lenguaje Natural	Generar respuestas a preguntas del usuario usando prompts combinados con HTML limpio.
RF-006	Almacenamiento de Datos	Almacenar los HTML procesados para no repetir scraping innecesario.

5.1.2. Requisitos no funcionales

- **RNF-001: Rendimiento**
  - **Objetivo:** Garantizar que el sistema sea capaz de manejar grandes volúmenes de datos de manera eficiente.
  - **Descripción:** La aplicación debe procesar y almacenar los HTML extraídos minimizando el tiempo de respuesta, asegurando agilidad tanto en la extracción como en la consulta.
- **RNF-002: Seguridad de los Datos**
  - **Objetivo:** Proteger los datos procesados y almacenados.
  - **Descripción:** La aplicación debe implementar medidas para garantizar la seguridad de las URLs y HTML almacenados, evitando accesos no autorizados a la base de datos.
- **RNF-003: Usabilidad**
  - **Objetivo:** Garantizar que la interfaz sea fácil de usar.
  - **Descripción:** La interacción con la aplicación debe ser intuitiva, facilitando su uso incluso por usuarios sin experiencia técnica.
- **RNF-004: Compatibilidad**
  - **Objetivo:** Asegurar que la aplicación sea compatible con distintas plataformas.

– **Descripción:** La extensión del navegador y la aplicación deben funcionar correctamente en los navegadores principales (Chrome, Firefox, Edge) y en sistemas operativos como Windows, macOS y Linux.

Tabla 5.2: Resumen de requisitos no funcionales

ID	Requisito	Descripción
RNF-001	Rendimiento	Procesar y almacenar HTML extraído de URLs rápidamente, optimizando tiempos de respuesta en extracción y consulta.
RNF-002	Seguridad de los datos	Asegurar la protección de URLs y HTML almacenados mediante medidas contra accesos no autorizados.
RNF-003	Usabilidad	Asegurar que la aplicación sea intuitiva y permita una interacción fluida por parte del usuario.
RNF-004	Compatibilidad	Garantizar compatibilidad con los principales navegadores y sistemas operativos (Windows, macOS, Linux).

5.1.3. Tareas

Para el desarrollo de los requisitos funcionales fue necesaria la especificación de varias tareas, organizadas en un *backlog*. A continuación se detallan:

Nº	Tarea	Descripción detallada	RF asociado
1	Reunión inicial y planificación	Reunión entre tutor y estudiante para definir la metodología, herramientas y planificación del proyecto.	–
2	Investigación del DOM de Google My Maps	Análisis de la estructura del DOM de My Maps para localizar POIs y acceder al modo de edición.	RF-001
3	Estudio de extensiones de Chrome	Revisión del funcionamiento de extensiones: estructura, permisos y scripts.	RF-001
4	Inyección y pruebas de *content scripts*	Desarrollo de scripts para modificar el DOM de Google My Maps de forma segura.	RF-001
5	Extracción y validación de URLs	Lógica para capturar URLs existentes en la descripción de los POIs.	RF-001
6	Edición automatizada del POI	Acceso al modo edición del marcador para insertar la URL si no está incluida.	RF-001

7	Diseño del interfaz de extensión	Desarrollo del *popup* con botón para ejecutar el script desde la extensión.	RF-001
8	Pruebas y depuración	Validación funcional en mapas reales y ajustes de tiempos/errores.	RF-001
9	Estudio legal del scraping y robots.txt	Revisión de aspectos legales y técnicos del scraping conforme al *robots.txt*.	RF-002
10	Estudio de Scrapy y desarrollo de spiders	Aprendizaje de Scrapy y desarrollo de *spiders* con navegación por profundidad.	RF-002
11	Almacenamiento del HTML en bruto	Guardado del HTML en ficheros o base de datos para su análisis posterior.	RF-002, RF-006
12	Limpieza del HTML	Eliminación de etiquetas y elementos irrelevantes del HTML.	RF-003
13	Agrupación de URLs por dominio	Clasificación de URLs por dominio raíz para optimizar el ranking posterior.	RF-004
14	Diseño de *prompts* para ranking	Generación de *prompts* para que el LLM ordene URLs según su relevancia.	RF-004
15	Selección de URLs para consulta	Ordenación y selección final de URLs para enviar al LLM.	RF-004
16	Diseño de *prompts* para preguntas	Construcción del *prompt* con pregunta y HTML procesado.	RF-005
17	Consulta final y filtrado de respuesta	Evaluación de respuestas del LLM y selección de la más adecuada.	RF-005
18	Registro de resultados	Almacenamiento en JSON de URL, pregunta y respuesta final.	RF-006

Tabla 5.3: Tareas del backlog y su relación con los requisitos funcionales

5.2. Diseño

La arquitectura del sistema se organiza en tres capas fundamentales: presentación, negocio y datos. Esta separación facilita el mantenimiento, escalabilidad y evolución del sistema a lo largo del tiempo.

- **Capa de presentación (Frontend):** Es la interfaz con la que el usuario final interactúa. Incluye la extensión de navegador para Google My Maps y el terminal donde se realizan las consultas en lenguaje natural.

- **Capa de lógica de negocio (Backend):** Se encarga de procesar las solicitudes del frontend. En este caso, gestiona la escritura de las URLs en los POIs, el scraping de las webs y las consultas al modelo LLM.

- **Capa de datos (Base de Datos):** Responsable de almacenar toda la información relacionada con las URLs, el HTML extraído y los resultados de las respuestas del modelo.

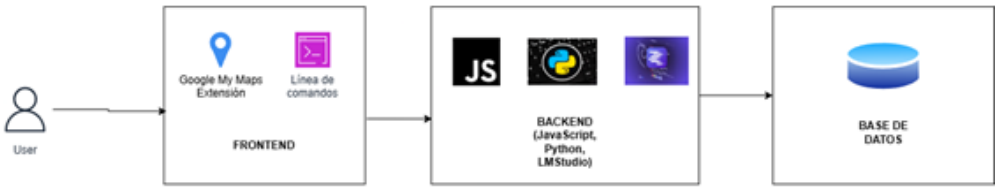

Figura 5.1: Arquitectura de la aplicación

5.2.1. Diseño General

Como resumen general del flujo completo de la aplicación, desde Google My Maps marcaremos los POIs que deseemos. La extensión se encargará de escribir la URL en la descripción de los contenedores de información de los POIs, para poder tener este dato tras la exportación en formato KML.

En cuanto a la aplicación de *scraping*, esta realiza varias comprobaciones, como verificar si la URL ya está almacenada en la base de datos, consultar el archivo `robots.txt` para verificar la viabilidad del scraping y, si es necesario, realizar el scraping del HTML, limpiarlo y almacenarlo en la base de datos.

Por último, conectaremos mediante API con un modelo LLM, en primer lugar para obtener un ranking de URLs de la misma raíz en función de la probabilidad de obtener una respuesta. Si se encuentra una respuesta satisfactoria, se inserta en un archivo JSON junto con la URL raíz correspondiente. Este proceso se repite hasta que no haya más URLs raíz que analizar, y finalmente, se devuelve el JSON con los resultados obtenidos.

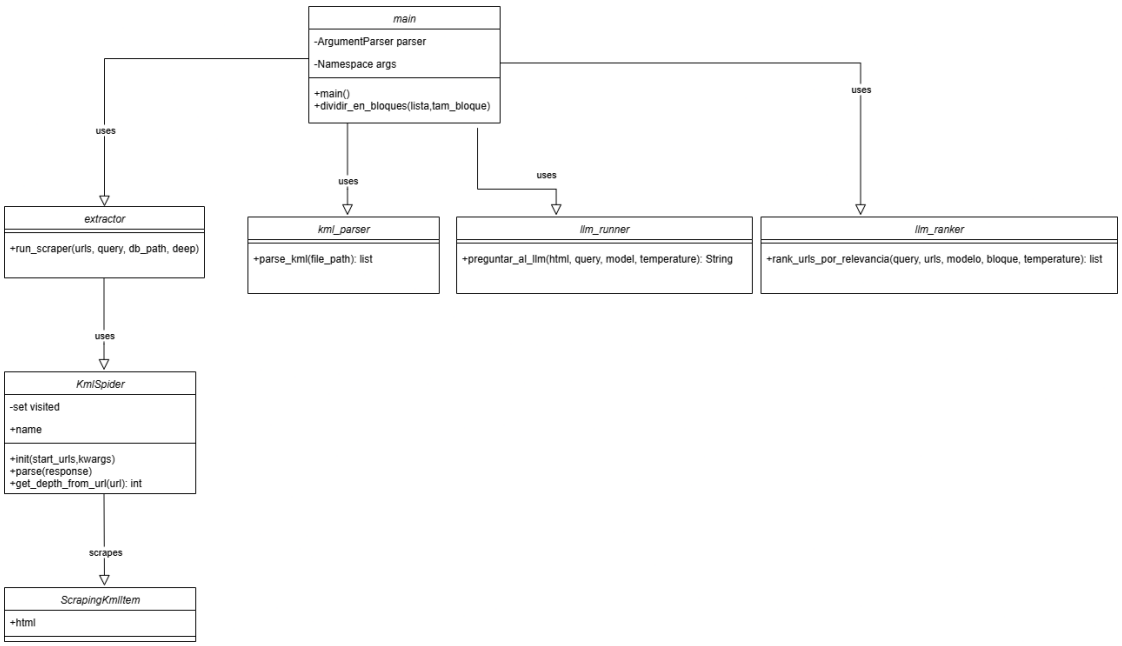

Figura 5.2: Diagrama de clases de la aplicación de línea de comandos

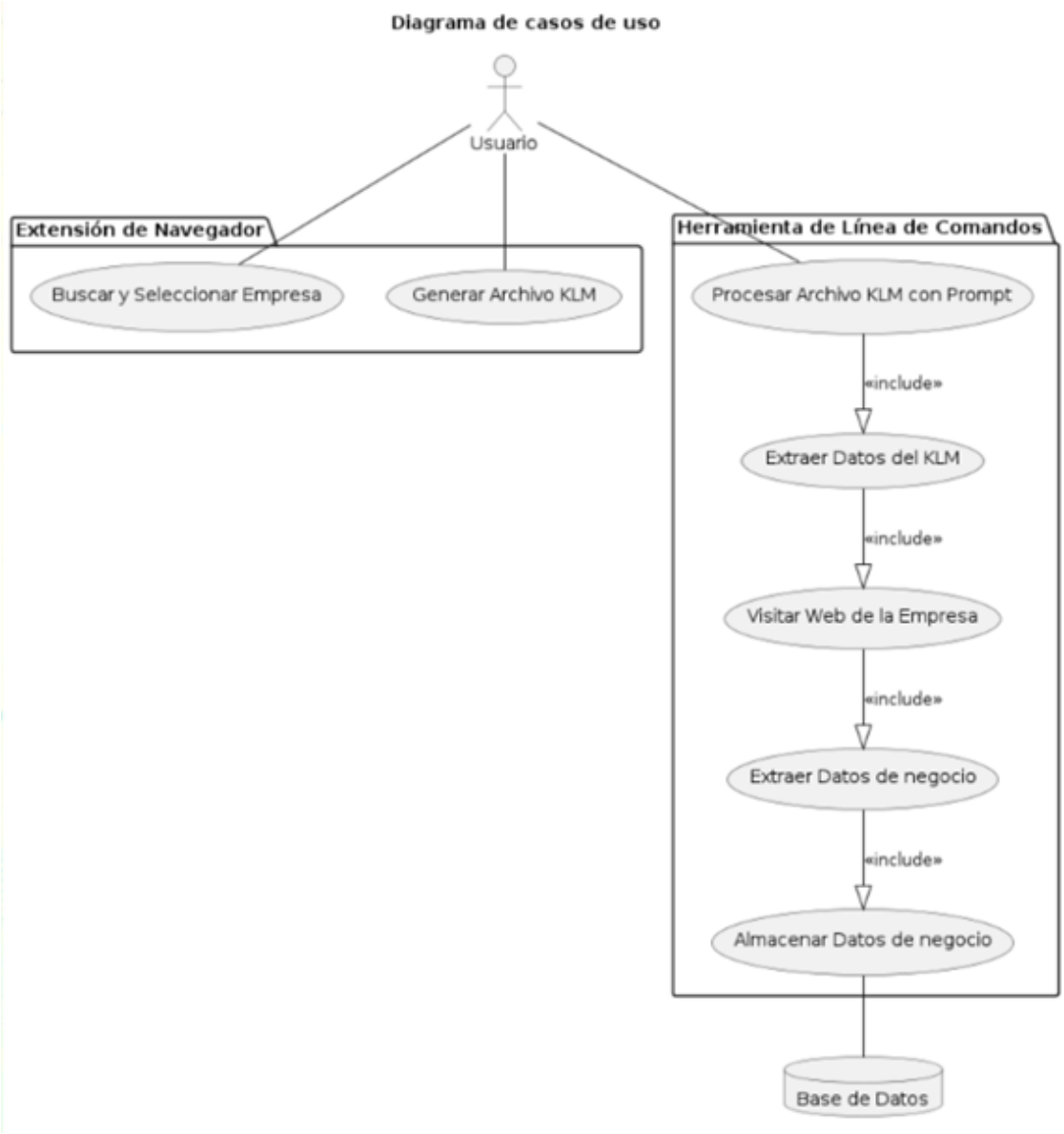

Figura 5.3: Diagrama de casos de uso: Flujo completo

5.2.2. Diseño de la extensión

El método para obtener las URLs de los distintos POIs seleccionado fue una extensión de navegador, mediante una estructura de acciones secuenciales utilizando el DOM de Google My Maps y el manejo de eventos y temporizadores. En la siguiente figura se muestra la estructura de la solución.

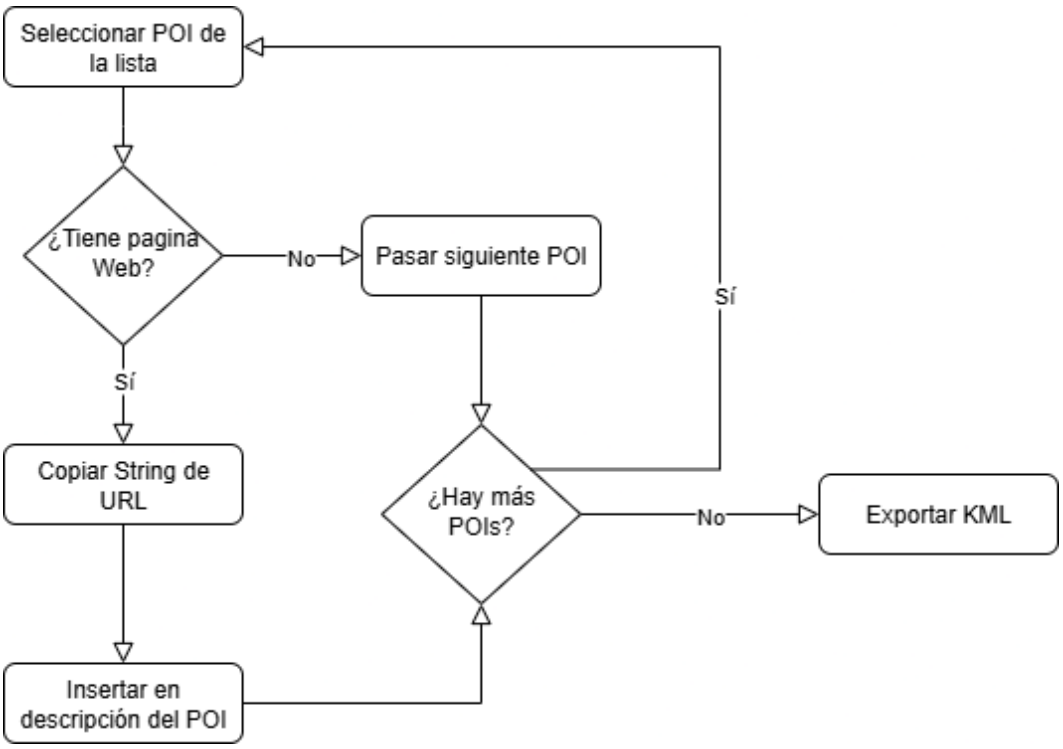

Figura 5.4: Diagrama de flujo de la extensión del navegador para Google My Maps

Las acciones mostradas en la figura anterior se describen de la siguiente manera:

- **Seleccionar POI de la lista:** se crea una lista con todos los POIs listados, identificando los elementos del DOM que representan cada uno de estos marcadores.
- **Página web:** al hacer clic sobre el POI, se despliega el contenedor de información. Se busca si contiene una URL; en función de ello se ejecuta una acción u otra.
- **Copiar *string* de URL:** si hay una URL disponible, se copia el *string* correspondiente desde el contenedor de información.
- **Insertar en descripción del POI:** se accede al modo de edición del POI y se pega la URL en el campo de descripción.
- **Exportar KML:** se exporta el mapa de Google My Maps en formato KML con las descripciones modificadas, para su posterior uso por la aplicación.

En resumen, esta extensión se encarga de automatizar el proceso de escritura de las URLs en los distintos POIs, permitiendo su recuperación tras la exportación del archivo KML.

5.2.3. Diseño aplicación Scraping

Para poder obtener la información de las URLs almacenadas, se optó por utilizar la herramienta `Scrapy`, *scrapeando* todo el HTML en crudo hasta una profundidad determinada

dentro de la raíz, indicada por el usuario mediante su solicitud por terminal. El HTML recibe un ligero procesamiento para facilitar su consulta y almacenamiento.

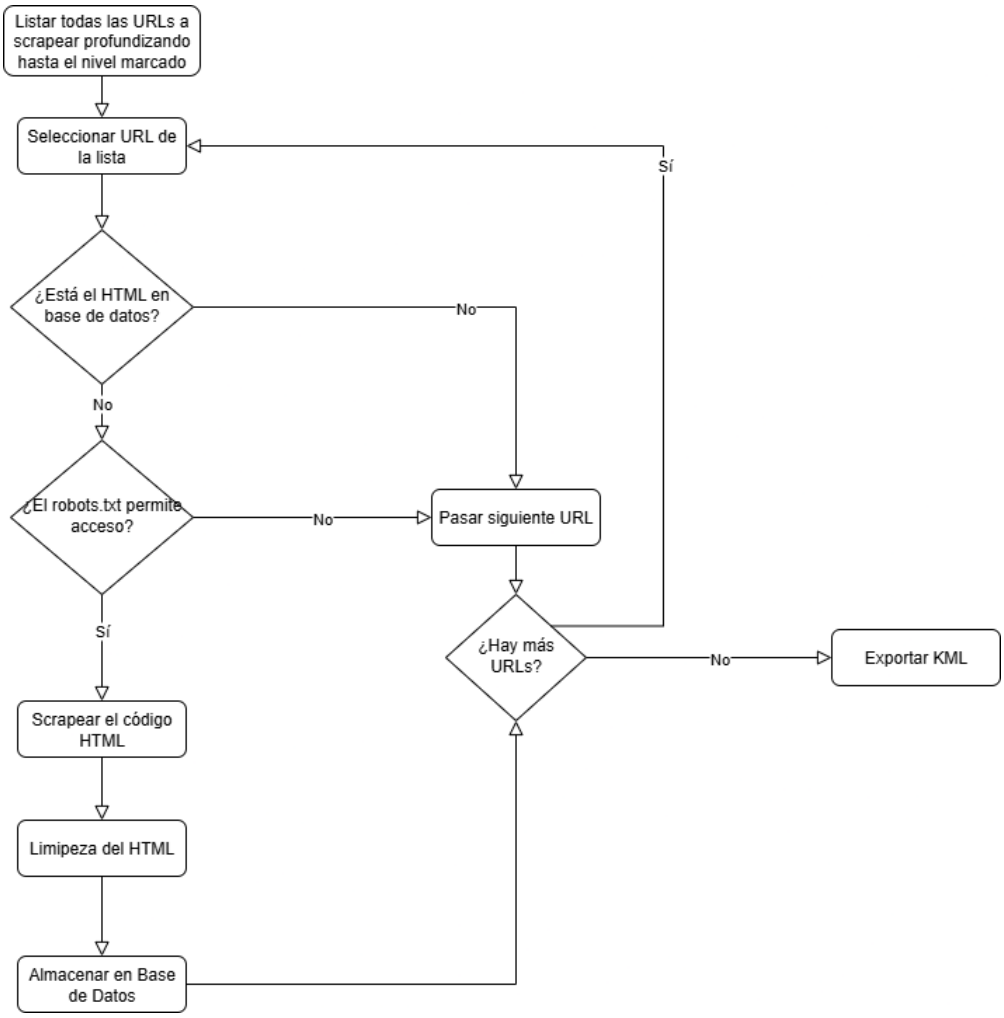

Figura 5.5: Diagrama de flujo de la aplicación de Scraping

Las tareas descritas en la figura se explican de la siguiente manera:

- **Listar todas las URLs a scrapear profundizando hasta el nivel marcado:** cogiendo el valor de la profundidad marcado por el usuario, se seleccionan todas las URL a partir de la raíz hasta esa profundidad y se listan.
- **Seleccionar URL de la lista:** se escoge por orden una URL de la lista para *scrapearla*.
- **Consultar base de datos:** se consulta si el HTML se encuentra ya en la base de datos para no tener que *scrapear* de nuevo en caso de que ya esté.
- **Consultar `robots.txt`:** se consulta el `robots.txt` de la URL para ver si permite el *scraping* o no.

- **Scrapear código HTML:** se *scrapea* el contenido total del HTML de la URL concreta.
- **Limpieza del HTML:** se realiza una limpieza del HTML eliminando espacios y algunas de las estructuras propias de los HTML, dejando simplemente información.
- **Almacenar en base de datos:** se inserta el HTML limpio en la base de datos.

En resumen, la aplicación listará las URLs a las que se podrá realizar la consulta y comprobará si su HTML está o no en la base de datos. En caso de no estar, se realizará el *scraping* con las comprobaciones pertinentes y se almacenará en la base de datos.

5.2.4. Diseño aplicación consulta al LLM

Para obtener respuesta a las consultas del usuario, utilizaremos un LLM desplegado en local mediante `LMStudio`. A este LLM se le consultará primero para realizar un *ranking* en función de las probabilidades de obtener una respuesta lo más específica posible en las siguientes URLs que se obtengan de la raíz, y después se le pasará un *prompt* con la pregunta en cuestión, un pequeño mensaje de contexto y el HTML procesado previamente.

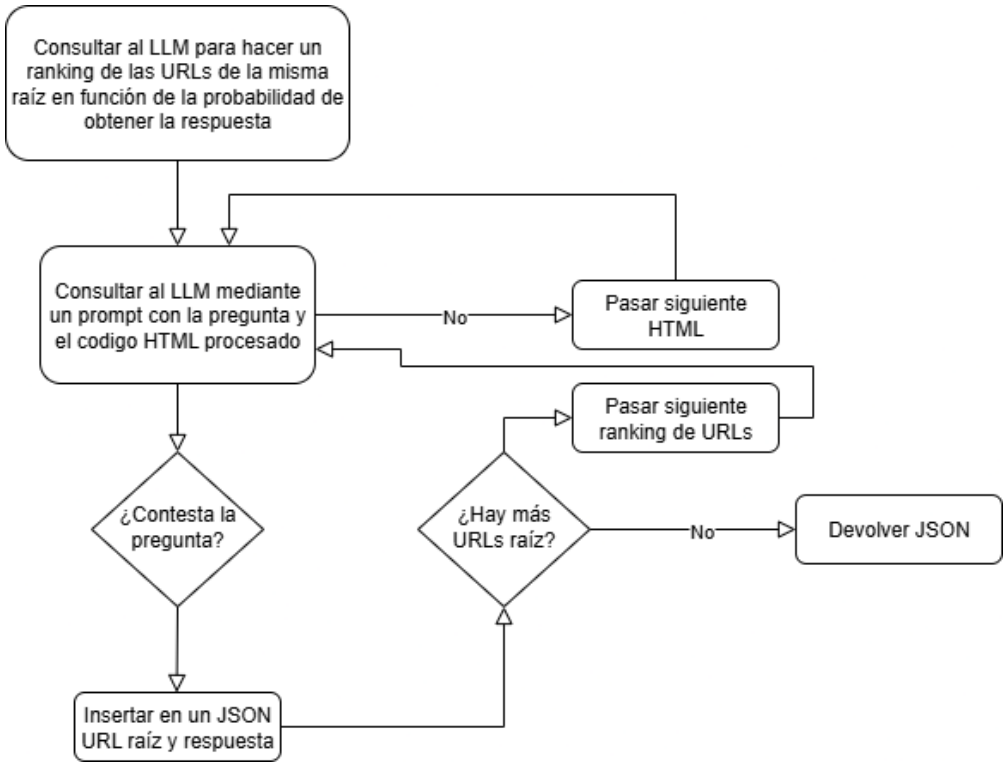

Figura 5.6: Diagrama de flujo de la aplicación de conexión con el LLM

El diagrama mostrado está formado por los siguientes procesos:

- **Consultar al LLM para hacer un ranking de las URLs de la misma raíz:** Consultar al LLM con la pregunta y el código HTML procesado para obtener un *ranking* de las URLs de la misma raíz, basado en la probabilidad de obtener una respuesta.

- **Consultar al LLM mediante un prompt:** Enviar al LLM un *prompt* con la pregunta y el código HTML procesado.
- **Contesta la pregunta:** El LLM evalúa si es capaz de contestar la pregunta con el HTML proporcionado.
- **Insertar en un JSON URL raíz y respuesta:** Si la respuesta es satisfactoria, insertar la URL raíz y la respuesta en un objeto JSON.
- **¿Hay más URLs raíz?:** Comprobar si existen más URLs raíz que analizar. Si es así, continuar con el análisis de la siguiente URL raíz. Si no, proceder al siguiente paso.
- **Devolver JSON:** Al finalizar el proceso de análisis, devolver el objeto JSON con los resultados obtenidos.

6. Desarrollo

6.1. Desarrollo de la extensión de navegador

El desarrollo de la extensión de navegador con el objetivo de automatizar el proceso para escribir desde el contenedor de información la URL raíz del negocio en cuestión en el campo descripción del POI fue ordenado y tuvo las siguientes fases:

- **Investigación inicial:** El primer paso fue estudiar un poco acerca de cómo hacer esta extensión, la estructura de las extensiones de navegador y la estructura del DOM de Google My Maps, así como elegir las tecnologías a utilizar.
- **Fase de desarrollo:** Fase de desarrollo de la aplicación, simulando el flujo completo de copiar la URL y pegarla en el campo de descripción mediante eventos y simulaciones de clics.
- **Validación de la extensión:** Fase final en la que se realizaron pruebas y se detectaron excepciones no manejadas como cuando no había una URL en el contenedor de información y otros casos.

6.1.1. Investigación inicial

El primer paso para llevar a cabo este proyecto fue investigar las opciones disponibles para crear extensiones de Chrome. Después de buscar varias alternativas, se tomó la decisión de que *JavaScript* era la mejor opción. Era el estándar para este tipo de tareas y tenía bastante documentación disponible, por lo que fue posible el desarrollo de manera rápida.

Para la investigación se comenzó a revisar tutoriales en vídeos en plataformas como *YouTube*, se leyeron documentos acerca del tema y se recurrió a modelos de lenguaje como *ChatGPT* cuando surgieron dudas. Con estos recursos, fue posible comprender cómo funcionan las extensiones de Chrome y cómo se utilizan los *content scripts*. Estos scripts se inyectan directamente en las páginas web y permiten analizarlas, extraer información y modificarlas dinámicamente según sea necesario.

También se realizó una investigación acerca del archivo `manifest.json`. Este archivo es fundamental porque define la configuración de la extensión, incluyendo los permisos que necesita y los scripts que se deben ejecutar. Después de revisar la documentación de Chrome, se concluyó que `manifest.json` es el archivo que le dice al navegador cómo gestionar la extensión.

La investigación continuó por los *content scripts*, fragmentos de código JavaScript que se inyectan en las páginas web. Estos scripts permiten interactuar con el DOM (*Document Object*

Model) de la página, que es una representación estructurada de los elementos HTML de la página web. Gracias al DOM, fue posible seleccionar y modificar los elementos necesarios para trabajar con los marcadores y las descripciones en Google My Maps.

6.1.2. Desarrollo de la extensión

Una vez comprendido el funcionamiento general de las extensiones de Chrome y los conceptos fundamentales asociados, se procedió al desarrollo progresivo de la extensión, abordando cada componente clave por separado.

El archivo `manifest.json` constituye el núcleo de configuración de cualquier extensión de Chrome. En este proyecto, se utilizó para definir los permisos necesarios (como acceso al contenido de páginas de Google My Maps), así como para registrar los scripts que debían ejecutarse y las acciones vinculadas al icono de la extensión. Esta configuración permite establecer la comunicación entre el popup de la extensión y el contenido de la página.

El archivo `content.js` implementa la lógica principal de la extensión. Este script se inyecta directamente en la página de Google My Maps y se encarga de recorrer los marcadores del mapa, extraer posibles URLs y, si es necesario, añadirlas al campo de descripción del marcador correspondiente.

Durante el desarrollo, se probó con distintas formas de identificar los marcadores en el DOM, utilizando tanto `querySelectorAll()` con diferentes combinaciones de clases como selectores XPath copiados directamente desde las herramientas de desarrollador de Chrome. Por ejemplo, se probaron selectores como:

- Selector Full XPath
- Selector XPath
- Selector JSPath

Este tipo de selectores acabaron por presentar un problema claro: no eran generales y era difícil marcar todos los POIs de esta manera, acababa por quedar fuera alguno de una forma u otra, o, en algunos casos, ni siquiera funcionaban en más de uno. Probando, se acabó por identificar un patrón común y consistente: en cada POI había un atributo personalizado `fl_id`. Este atributo se mantiene constante entre diferentes sesiones y mapas. Por ello, se acabó optando por utilizar el siguiente selector CSS:

- `#ly0-layer-items-container > div[fl_id]`

Permitiéndonos guardar todos los marcadores en una lista de estos de la siguiente manera:

```
// Selecciona todos los marcadores
const markers = document.querySelectorAll('#ly0-layer-items-container > div[↩
    ↪ fl_id]');
console.log("Marcadores encontrados:", markers.length);
```

Código 6.1: Fragmento de código de captura de marcadores

Se trata de un selector por atributo, que permite iterar de forma generalizada sobre todos los POIs dentro del contenedor de POIs en el lateral de Google My Maps. Gracias a esta solución, fue posible incluir en una lista todos los marcadores y poder iterar sobre ellos gracias a la función `querySelectorAll()`.

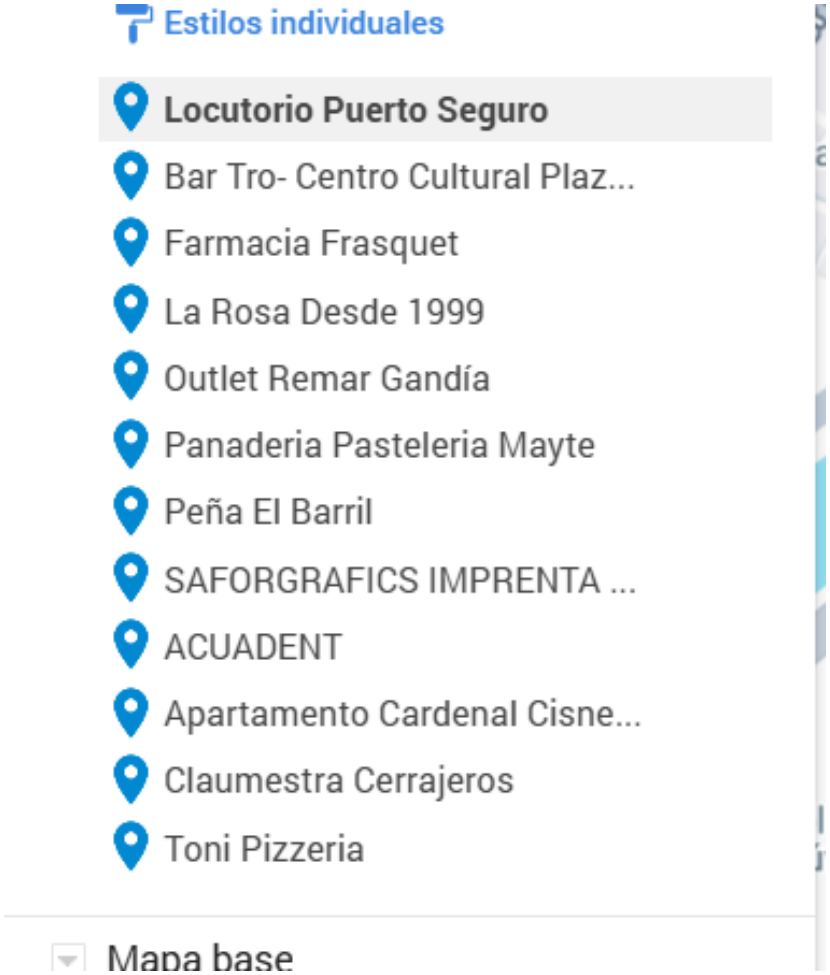

Figura 6.1: Lista de marcadores en Google My Maps

Una vez identificados estos marcadores, fue necesario simular un clic para poder abrir el contenedor de información de un POI concreto. Gracias a la función `simulateRealClick()` sobre el marcador por el que estamos iterando fue una tarea sencilla. Es importante remarcar que, debido a problemas de sincronización si se ponían todos los eventos uno detrás de otro, se utilizan algunos controladores como:

```
1 await new Promise(resolve => setTimeout(resolve, 2000));
```

Código 6.2: Fragmento de código para gestión de sincronización mediante temporizador

Que marca un punto de espera hasta que, en este caso, se abre el contenedor de información y algunos controladores de tiempo sobre todo para esperar entre dos clics.

Una vez abierto el contenedor de información, lo que necesitaba era encontrarlo, afortunadamente, todos seguían una misma estructura y no eran diferentes entre los distintos marcadores, facilitando bastante el resto del proceso:

```
// 3. Localiza el contenedor del infowindow
const infowindowEl = document.querySelector('#map-infowindow-content');
```

Código 6.3: Fragmento de código captura de contenedor de información

De una forma similar se pudo encontrar tanto la URL:

```
let linkEl = infowindowEl.querySelector('a');
const editBtn = document.querySelector('#map-infowindow-edit-button');
```

Código 6.4: Fragmento de código para obtención de enlace URL y botón de edición

En este caso haciendo una limpieza porque, en caso de que no hubiera una URL, cogía el enlace de la ubicación en Google Maps, así que en los casos en los que la URL fuera de Google Maps simplemente la dejaría vacía.

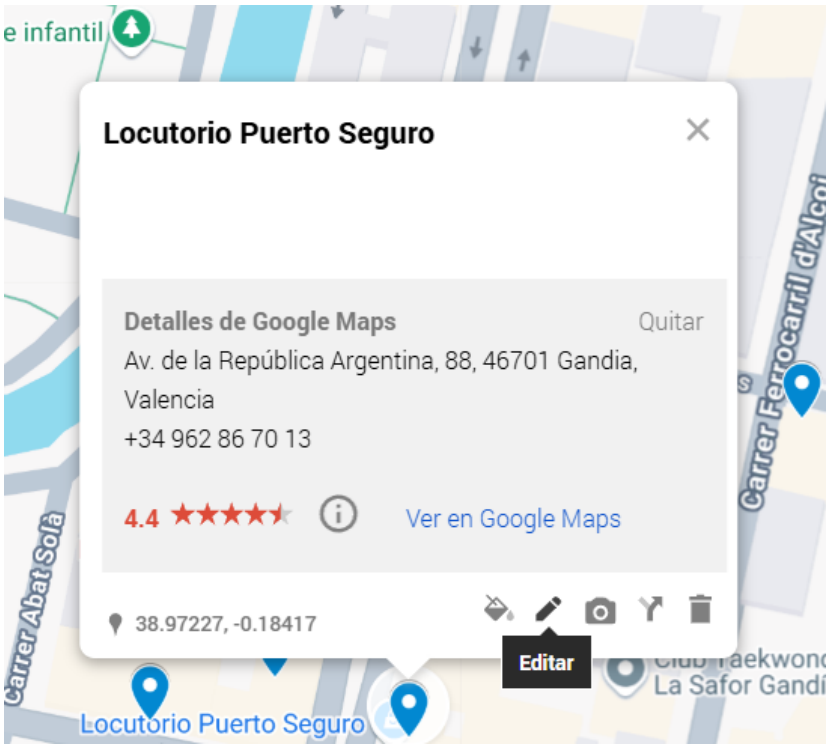

Figura 6.2: Contenedor de información de un POI en My Maps

Llegábamos a un punto clave que era identificar el campo de descripción donde teníamos que copiar la URL previamente copiada. Encontrar el campo fue sencillo, con un formato similar a los anteriores y de nuevo general para todos los marcadores:

```
const descripcionField = document.querySelector('#map-infowindow-attr-description-value');
```

Código 6.5: Fragmento para capturar el campo descripción

Para la inserción de la URL en el campo era deseable manejar si había escrito algo previamente y qué, por lo que se diseñó una lógica mediante la cual la URL solo se insertaba si no estaba ya presente evitando duplicidades exactas en los marcadores. En caso de que no fuera la URL lo que ya había, simplemente se concatena la URL a la cadena que ya hubiera. Por último, había que pulsar un botón de guardar cambios que se identificó como `#map-infowindow-done-editing-button > div`, sobre el botón simplemente se simula un clic. Para cerrar el contenedor de información simplemente se simula un tecleo a la tecla Escape. Tras esto se itera al siguiente marcador si lo hubiera.

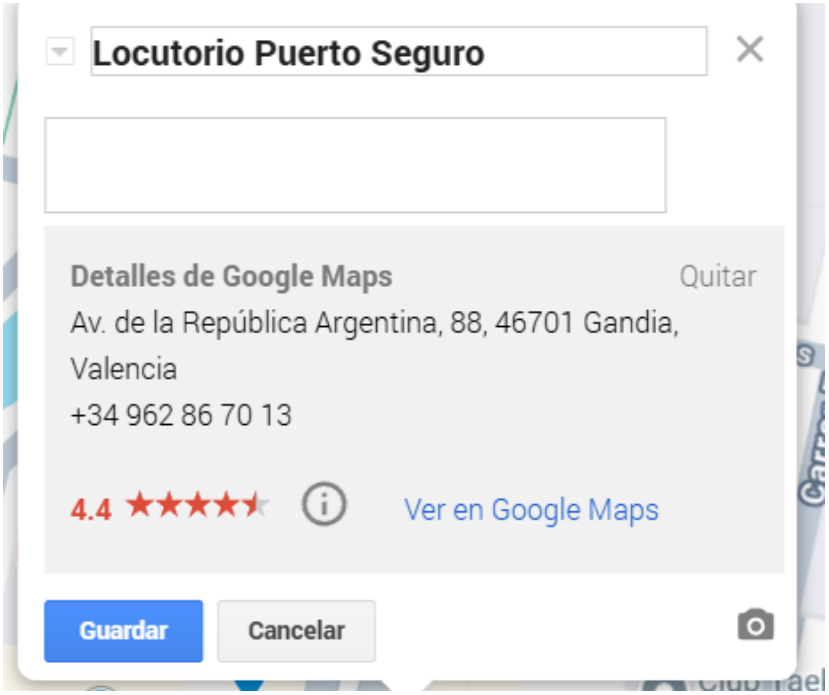

Figura 6.3: Campo descripción en un POI de My Maps

Pasando al uso de la extensión ahora que era funcional, lanzándola por consola del navegador eso sí, era el momento de desarrollar el `popup.js`, la interfaz de usuario de la extensión. Simplemente en una pequeña ventana que aparece cuando el usuario hace clic sobre la extensión en la barra de extensiones del navegador, se habilitó un botón "Iniciar proceso", que lanza la iteración descrita previamente sobre todos los marcadores.

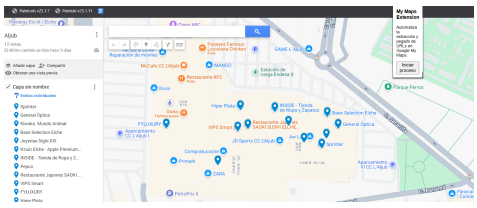

Figura 6.4: Botón "Iniciar proceso" de la extensión

Por último, se exporta el mapa con el formato KML sobre el que trabaja el resto de la aplicación.

6.2. Implementación de la herramienta por línea de comando

El desarrollo de la aplicación de línea de comandos tuvo como objetivo automatizar el proceso de consulta a las webs, extrayendo el contenido HTML de las URLs asociadas a los POI y consultando posteriormente a un LLM con un *prompt* formado por la pregunta y el HTML procesado. La implementación se dividió en tres fases principales:

- **Investigación inicial:** En primer lugar, fue necesario familiarizarse con el concepto *scraping* y sus limitaciones tanto técnicas como legales, así como buscar la mejor tecnología para el *scraping*. En este sentido, se acabó optando por *Scrapy* y su sistema de *Spiders*. Para la parte del LLM se valoraron las opciones tanto locales como remotas, optando finalmente por opciones locales gracias a las facilidades que da *LMStudio* tanto por coste como por la facilidad de conexión gracias a la API local.

- **Fase de desarrollo:** En esta etapa se construyeron dos módulos principales. Por un lado, un módulo de *scraping* con *Scrapy*, responsable de conectarse a cada URL, extraer el HTML y almacenarlo en la base de datos MySQL. Por otro lado, se desarrolló un módulo de consulta que toma como entrada las URLs almacenadas y realiza una llamada a un LLM en primer lugar para rankear las URLs y ordenarlas en función de la probabilidad de responder la pregunta, y en segundo lugar para interpretar el contenido HTML y devolver información útil como la categoría del negocio, el horario, la dirección o la oferta de productos y servicios.

- **Validación y depuración:** La fase final consistió en ejecutar pruebas completas sobre muestras de datos reales, comprobando que el *scraping* se ejecuta de forma robusta y que las respuestas del LLM se interpretan y almacenan correctamente. Durante esta etapa se depuraron errores como URLs duplicadas en base de datos, por lo que en algunos casos se realizaba *scraping* de URLs ya guardadas, y se afinaron los *prompts* enviados al modelo, mejorando además la detección de casos no válidos.

6.2.1. Investigación inicial

El primer paso era investigar la técnica de Scraping y sobre todo las distintas herramientas disponibles para ello. Como el lenguaje escogido fue Python por la gran cantidad de soporte, librerías y documentación disponible sobre él, opté por Scrapy y su sistema de Spiders. Scrapy

destacaba por su flexibilidad y cantidad de documentación y aplicaciones ya desarrolladas subidas, por lo qué fue la mejor opción.

La investigación inicial se basó en tutoriales, documentación y ejemplos de otras aplicaciones. También se recurrió al apoyo de modelos de lenguaje como ChatGPT para resolver dudas puntuales y comprender mejor el flujo de ejecución de Scrapy.

Dentro de Scrapy, el componente clave fue el diseño de un spider, que viene a ser una clase encargada de definir cómo recorrer una lista de URLs y cómo extraer los datos relevantes, aunque en este caso era extraer todo. Fue necesario configurar correctamente el archivo `settings.py` para manejar la codificación, los tiempos de espera y los middlewares necesarios para una ejecución robusta.

Para la parte de consulta al LLM, rápidamente se descartaron las opciones remotas debido sobre todo a un tema de coste. Una vez realizado el descarte se comenzó la investigación de modelos de lenguaje self-hosted y LMStudio que facilita mucho el empleo de los mismos. LMStudio es una interfaz que permite ejecutar LLMs de manera local utilizando aceleración por GPU.

6.2.2. Desarrollo de la aplicación

Para comenzar con la aplicación, el primer paso era extraer todas las URLs incluidas en el archivo KML, para ello se desarrolló un módulo aparte `kml_parser.py` que buscaba el contenido entre los placemarks de `description` y lo incluía en una lista. Durante el desarrollo, fue un problema en algunos casos en los que en el campo de descripción había más texto además de la URL, por lo que se desarrolló un filtro para incluir solo la parte deseada. Este módulo devuelve la lista por lo que ya eran accesibles todas las URLs a consultar.

```
for placemark in root.findall('.//kml:Placemark', ns):
    url_elem = placemark.find('.//kml:description', ns)
    if url_elem is not None and url_elem.text:
        matches = re.findall(r'https?://[^\s"<]+', url_elem.text)
        urls.extend([url.strip() for url in matches])
return urls
```

Código 6.6: Fragmento de código extracción de URL del campo descripción del KML

Posteriormente, era momento de scrapear los HTML de estas URLs, era necesario realizarlo hasta una profundidad sobre la raíz marcada como argumento desde el terminal. Por intentar modularizar al máximo la aplicación se separó esta parte de scraping en un módulo que sería el que llamaría al spider de Scrapy, `extractor.py`, este le envía las URLs hasta la profundidad deseada al `KmlSpider`, definido en `kml_spider.py`, que visita cada página, con un simple método usando BeautifulSoup limpia el contenido irrelevante y que no aporta información y se optó por comodidad por guardarlo en un JSON con URL y HTML.

```
process = CrawlerProcess(settings)
process.crawl(KmlSpider, start_urls=urls_filtradas, query=query, depth=deep)
```

```
process.start()
```

Código 6.7: Fragmento de código llamada al scraper desde extractor.py

Finalizado el scraping y con el JSON de URL y HTML formado, era el momento de realizar consultas al LLM. Para ello, primeramente se buscó un LLM adecuado. En un primer momento el objetivo era simplemente contestar las preguntas, por lo tanto se buscó un LLM que fuera capaz de responderlas adecuadamente para un prompt que incluyera la pregunta en cuestión y el código HTML procesado. Se probaron los siguientes modelos:

- **DistilGPT2**: Este modelo fue escogido dado que sus requerimientos eran bastante bajos y parecía tener un buen comportamiento especialmente en cuanto a velocidad de respuesta, sin embargo rápidamente fue descartado debido a su poco contexto que dificulta mucho responder de manera adecuada a las preguntas dado que en algunos casos el HTML ya sobrepasaba el contexto.
- **LLaMA-2-7B**: En este caso este modelo fue el siguiente que se probó, fue escogido por tener un mayor contexto principalmente, sin embargo, el rendimiento que tenía era pobre y las respuestas no eran del todo correctas en ocasiones.
- **Gwen1.5-4B-Cat**: Este modelo fue escogido por tener una capacidad de contexto intermedia entre el primer y el segundo caso, y por lo tanto unos requerimientos menores al anterior, el rendimiento fue correcto, funcionaba rápidamente y sin exigir demasiados recursos, en un primer lugar respondía bien, pero tras realizar varias pruebas se pudo observar que para HTMLs excesivamente largos su contexto se quedaba corto.
- **Nous-Hermes-2-Mistral-7B-DPO**: La selección de este modelo fue por dos motivos, el primero, el contexto era mayor al anterior, en este caso de 4096 tokens (el doble) y en segundo lugar, parecía estar mucho mejor optimizado que el segundo probado. Efectivamente, este modelo tenía un rendimiento mucho mejor que este y respondía adecuadamente, por lo que, de manera temporal, este fue el modelo escogido.

Sin embargo, el LLM consultaba por orden del JSON y en ocasiones se quedaba con una URL dentro de todas las extraídas por la profundidad que no respondía del todo bien la pregunta o que lo hacía de manera superficial. En primer lugar, para solucionar esto se trató de refinar el prompt al extremo pero no surtió efecto ya que seguía contestando aún no siendo una respuesta del todo precisa. Entonces, la siguiente solución que se probó y que finalmente ha quedado fue la de establecer una consulta previa al LLM que ordenaba en grupos de diez, para evitar problemas con el LLM ya que rinde mejor con estos grupos reducidos, las URLs en función de la probabilidad de responder a la pregunta, de cada grupo de diez se sacan los tres mejor posicionados y se repite este proceso de manera iterativa hasta quedarse con los diez mejores que vuelve a ordenar para su consulta final.

El prompt diseñado para establecer el orden fue el siguiente:

```
prompt = (
    "Dada la siguiente pregunta, ordena estas URLs según la probabilidad de que↩
        ↪  "
```

```
    "contengan la información solicitada, de mayor a menor. "
    "Devuélveme ÚNICAMENTE una lista de índices RELATIVOS AL BLOQUE como esta: [1, 0, 2, 5, 4, 3, 6, 8, 9, 7], la lista tendrá"
    "entre 1 y 10 elementos, en la lista coloca el mismo numero de elementos que URLs hay en el bloque, "
    "NO EXPLIQUES NADA. SOLO devuelve la lista, sin ningún otro texto. \n\n"
    f"Pregunta: {query}\n\n"
    "URLs:\n"
)


for j, url in enumerate(urls_bloque):
    prompt += f"{j}: {url}\n"

prompt += "\nRecuerda: ORDENA TODAS las URLs por relevancia. NO OMITAS NINGUNA . SOLO la lista. PASAME SOLO LA LISTA NINGUNA EXPLICACION POR FAVOR"
```

Código 6.8: Prompt para ordenar las URLs por probabilidad de contener la respuesta

Para esta tarea de ordenación fue necesario cambiar de nuevo de modelo ya que el anterior no era capaz de devolver la lista con el formato requerido y por lo tanto fue necesario buscar otro modelo que en esta ocasión cumplió con los requisitos esperados:

- **Gwen2.5-7B-Instruct-1M**: Este modelo contaba con las mismas virtudes que el anterior en cuanto a rendimiento pero estaba especializado en las preguntas y respuestas, lo que lo hacía perfecto para esta aplicación, respondía de manera satisfactoria tanto a la ordenación por lo que finalmente fue elegido y es el modelo que ha quedado finalmente.

Esta parte de orden de las URLs fue separada del resto de la consulta al LLM en otro módulo llamado `llm_ranker.py` al que se llama de manera y devuelve los tres mejor rankeados.

Una vez obtenidos las diez mejores URLs para responder a la pregunta del usuario, simplemente teníamos que llamar al otro módulo que conectaba con el LLM, `llm_runner.py`, que consulta al LLM con el siguiente prompt.

```
system_prompt = (
    "Eres un asistente que ayuda a extraer información útil de HTML. La respuesta debe ser muy precisa. "
    "Si no encuentras la respuesta, responde únicamente con: NO ENCONTRADO. "
    "Ten en cuenta que te iré pasando más HTMLs con diferentes niveles de profundidad. "
    "Si la respuesta no es perfecta, espera a más información en siguientes iteraciones. "
    "Por ejemplo, si la pregunta es sobre los ingredientes de una tarta, no digas 'crema' o 'galleta', "
    "espera a decir los ingredientes exactos como harina, leche, azúcar, etc."
)

prompt = f"""<|im_start|>system
{system_prompt}<|im_end|>
```

```
<|im_start|>user
Extrae la siguiente información del siguiente HTML:

Pregunta: {query}

HTML:
{html}
<|im_end|>
<|im_start|>assistant
\"\"\"
```

Código 6.9: Prompt de consulta al LLM para responder a la pregunta del usuario

En el prompt se especifica que conteste de manera específica a la respuesta y que no la deje parcialmente contestada y también se le explica que en caso de no poder responder adecuadamente, conteste con "NO ENCONTRADO", de esta manera filtramos y en caso de obtener esa respuesta es fácil identificar que hay que pasar a la siguiente URL en el orden marcado.

Por último, las respuestas obtenidas son almacenadas en un JSON, `respuestas.json`, que contiene pregunta (la cuestión que realizó el usuario), url (URL de la que se extrajo la respuesta) y por supuesto, respuesta (la respuesta que dio el LLM). Estas respuestas son también mostradas por terminal conforme se dan de manera iterativa para las diferentes URL raíz.

```
# Paso 4: Consultar al LLM
for entrada in ordenadas:
    url = entrada["url"]
    html = entrada["html"]

    print(f"\nConsultando al LLM: {url}\n")
    respuesta = preguntar_al_llm(html, args.query)

    if "NO ENCONTRADO" not in respuesta.upper() and respuesta.strip() != "":
        print(f"\ n Información encontrada en: {url}\n")
        print(f"  Respuesta: {respuesta.strip()}")
        guardar_respuesta(url, args.query, respuesta.strip())
        break
    else:
        print(f"  Información no encontrada en: {url}")
```

Código 6.10: Respuesta final de la aplicación al usuario

7. Resultados

7.1. Extensión de navegador

En primer lugar, para poder validar el correcto funcionamiento de la extensión se crearon en un Mapa de Google My Maps ciento veintidós POIs que representan negocios de toda la ciudad de Alicante, ésta muestra era suficientemente representativa ya que solo se apreciaron tres casos posibles que varían el funcionamiento de la aplicación:

- El caso normal: el campo de descripción se encuentra vacío por lo que simplemente hay que coger el campo de la URL y guardarlo dentro.
- El caso sin URL: en los casos en los que el contenedor de información no tenía URL del negocio, al principio la extensión cogía la URL del enlace "Ver en Google Maps", este funcionamiento no era deseado por lo que se filtraron todas las URLs con raíz `google.com/maps`.
- El caso con texto ya en la descripción: en este caso no era deseable eliminar el texto que había en el campo de descripción, por lo que se optó por copiarlo y concatenar al final la URL, para después copiar la cadena completa.

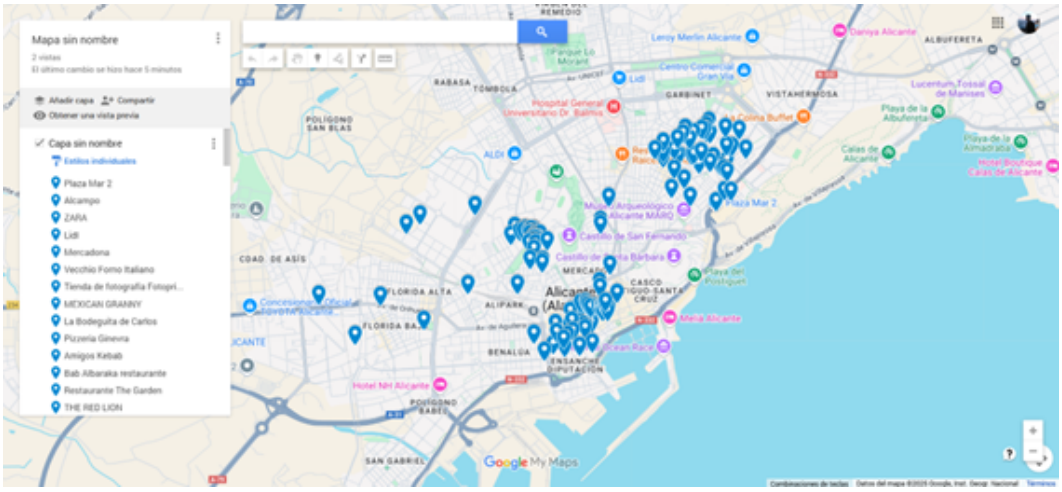

Figura 7.1: Mapa de Google My Maps con ciento veintidós POIs de la ciudad de Alicante

Para este caso con ciento veintidós POIs se pudieron extraer los siguientes resultados: el tiempo de duración del proceso completo fue de 7:57 minutos, unos 8 minutos aproximada-

mente, que representan unos 15 POIs procesados por minuto contando que pueden variar ligeramente en función de los casos que se den dentro de ese conjunto.

7.2. Aplicación línea de comando

Para continuar, se van a explicar las pruebas realizadas sobre la aplicación de línea de comando que combina el *Scraping* de las distintas URLs hasta la profundidad marcada y las consultas al LLM en primera instancia para ordenar las URLs por probabilidad de responder a la pregunta del usuario y en segunda instancia para responder a la pregunta misma a partir del HTML. Para la prueba, se marcaron en un mapa de Google My Maps algunos negocios del centro comercial L'Aljub en Elche, por tenerlos localizados geográficamente.

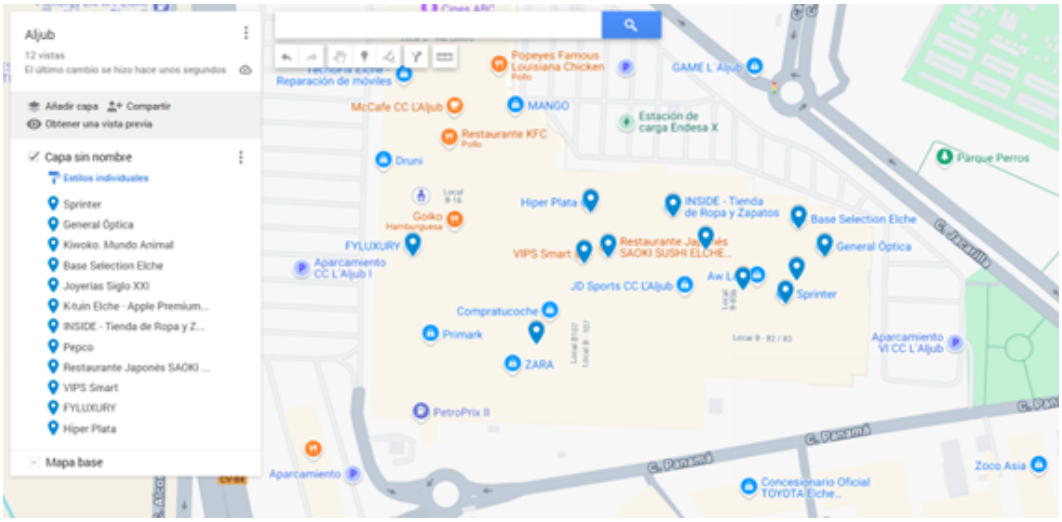

Figura 7.2: Mapa de Google My Maps del centro comercial L'Aljub

Para comenzar, realizaremos una pregunta simple como "¿qué tipo de negocio es?", buscando una respuesta que nos describa brevemente cuál es el objeto de negocio de cada uno de estos POIs.

En primer lugar se detectará qué URLs de la lista hay que scrapear y cuáles se encuentran ya en base de datos. Para esto, se verificará si la URL está registrada previamente. Este paso trata de ayudar a aumentar el rendimiento y evitar la recolección innecesaria de datos.

Figura 7.3: Detección de qué URLs son necesarias de scrapear

Tras esto, se comienza a profundizar en las diferentes URLs hasta la profundidad marcada, indicando por terminal qué URLs se están pudiendo guardar y cuáles no, indicando el motivo, como por ejemplo el caso en que lo prohíba el fichero `robots.txt`.

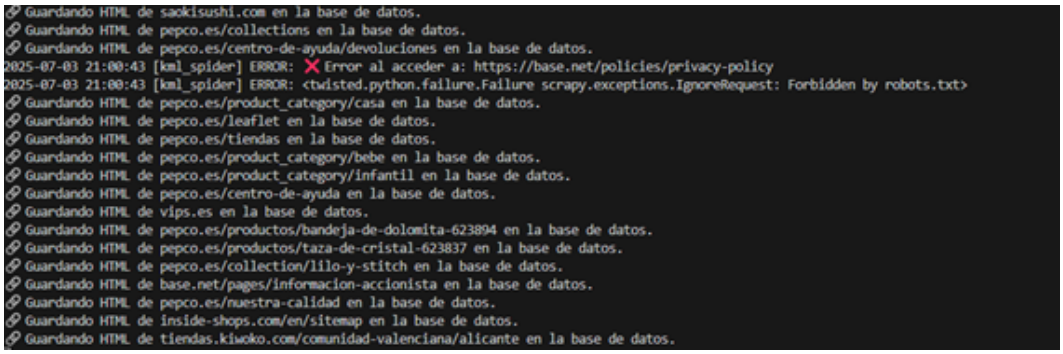

Figura 7.4: Proceso de guardado de código HTML

Una vez finalizado el *Scraping*, se agrupan las URLs extraídas por raíz, y por línea de comando se devuelve un mensaje del tipo “Procesando dominio: base.net con 19 URLs”, que muestra la raíz normalizada y el número de URLs que hay scrapeadas de ella hasta la profundidad mostrada. De cada grupo de diez se obtienen las tres mejores, repitiendo el proceso hasta obtener las diez mejores entre todas las URLs scrapeadas para responder la pregunta. Una vez ordenadas las diez mejores se realiza la consulta de la pregunta por orden secuencial en función del orden establecido y, en caso de responder a la pregunta, se coloca en un JSON con el siguiente formato:

Figura 7.5: Consulta de la pregunta secuencial al LLM

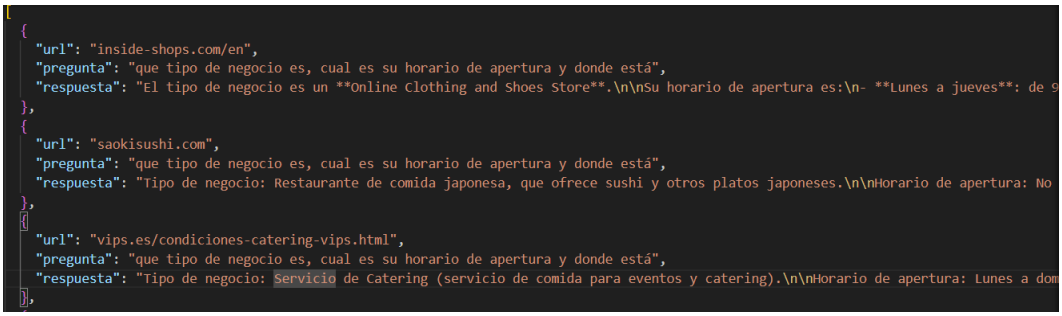

Figura 7.6: Fichero JSON de respuestas

De las pruebas realizadas, se han podido extraer las siguientes conclusiones:

En cuanto al rendimiento, se observa que al marcar una profundidad menor o igual a dos, el flujo de la aplicación es mucho más rápido y fácil de procesar, ya que no suelen generar una gran cantidad de URLs diferentes a scrapear. En casos en los que se marcan profundidades mayores o iguales a tres, el número de URLs crece de manera exponencial, sobre todo en grandes negocios y multinacionales, siendo las páginas web de estas mucho más extensas. En este caso, para Pepco se llegaron a scrapear 881 URLs diferentes con profundidad tres. Aunque el scraping se realiza solo una vez, este impacto también afecta al sistema de consultas, ya que el *ranker* debe manejar un gran número de URLs, lo que incrementa significativamente las consultas necesarias. Por ejemplo, en el caso de Pepco, se realizaron 89 consultas solo para ordenar una primera vez las URLs, siendo necesarias otras tantas hasta llegar a diez URLs a las que se pasará la consulta. En este aspecto, se puede concluir que la aplicación no responde en un tiempo razonable para grandes profundidades de grandes empresas.

En cuanto al correcto funcionamiento de la aplicación, en ocasiones el LLM tiene dificultades para inferir la respuesta a una pregunta a partir del código HTML, pese a que esta podría ser fácilmente deducible para una inteligencia humana. Esto se debe a las limitaciones

de hardware y del LLM seleccionado.

Por otro lado, en algunas ejecuciones, el modelo ha mostrado inconsistencias en las respuestas. En ocasiones encuentra la respuesta correcta en una URL, pero en otras no la considera válida en la misma URL. Si bien estos casos han sido mínimos, han existido.

Además, en casos de multinacionales o empresas grandes y distribuidas por un gran territorio, el HTML puede ser general, por lo que puede no llegar a contestar al caso del POI específico marcado en Google My Maps.

Por último, en casos en los que se pasan varias preguntas en la misma ejecución de la aplicación, se observan casos en los que el LLM no es capaz de encontrar todas las respuestas en una misma URL y por lo tanto no contesta. En este aspecto, es recomendable hacer preguntas individuales o preguntas que estén relacionadas de modo que puedan encontrarse en la misma URL.

Ejemplo:

```
python -m main --kml aljub.kml --query "que tipo de negocio es, cual es su horario de apertura y donde está" --db datos_desde_db.json --deep 2
```

Para esa petición al fichero `aljub.kml` con profundidad dos obtenemos el siguiente JSON de respuestas:

```
[
  {
    "url": "inside-shops.com/en",
    "pregunta": "que tipo de negocio es, cual es su horario de apertura y donde está",
    "respuesta": "El tipo de negocio es un **Online Clothing and Shoes Store**.\n\nSu horario de apertura es:\n- **Lunes a jueves**: de 9am a 2pm y de 3pm a 5pm.\n- **Viernes**: de 8am a 2:30pm.\n- **Sábado**: de 10am a 4pm.\n\nNo se menciona un local físico, por lo que no hay una ubicación específica."
  },
  {
    "url": "saokisushi.com",
    "pregunta": "que tipo de negocio es, cual es su horario de apertura y donde está",
    "respuesta": "Tipo de negocio: Restaurante de comida japonesa, que ofrece sushi y otros platos japoneses.\n\nHorario de apertura: No se especifica un horario en el texto proporcionado. Solo menciona novedades y promociones sin referirse a un horario de apertura o cierre.\n\nDónde está: \n1. Elche, Alicante - Centro Comercial 'LAljub, Carrer Jacarilla, 7.\n2. Cartagena, Murcia - C/ Santa Florentina, 4."
  },
  {
```

```
13     "url": "vips.es/condiciones-catering-vips.html",
14     "pregunta": "que tipo de negocio es, cual es su horario de apertura y donde
           ↪  está",
15     "respuesta": "Tipo de negocio: Servicio de Catering (servicio de comida
           ↪ para eventos y catering).\n\nHorario de apertura: Lunes a domingo.\n
           ↪ \nDónde está: El servicio es solo disponible en la Comunidad de
           ↪ Madrid. No se especifica una dirección física, sino que el servicio
           ↪ es de entrega."
16   },
17   {
18     "url": "tiendas.kiwoko.com/comunidad-valenciana",
19     "pregunta": "que tipo de negocio es, cual es su horario de apertura y donde
           ↪  está",
20     "respuesta": "El tipo de negocio es una cadena de tiendas especializadas en
           ↪  productos para mascotas (perros, gatos, conejos, roedores, reptiles
           ↪ , pájaros y peces) y servicios veterinarios. \n\nHorario de apertura
           ↪ :\n- Abre mañana a las 09:00\n- Cerrado ahora\n\nUbicación:\nLas
           ↪ tiendas están distribuidas en diferentes ciudades de la Comunidad
           ↪ Valenciana, como Carcaixent, Xativa, Valencia, etc."
21   },
22   {
23     "url": "base.net",
24     "pregunta": "que tipo de negocio es, cual es su horario de apertura y donde
           ↪  está",
25     "respuesta": "Tipo de negocio: Tienda online de ropa y calzado deportivo.\n
           ↪ \nHorario de apertura: Lunes a Viernes de 09:00- 18:00\n\nDónde está
           ↪ : No se especifica un lugar físico, ya que es una tienda online."
26   }
27 ]
```

Código 7.1: Ejemplo de JSON de respuestas generadas por la aplicación

Como se puede observar, en algunos casos contesta las tres preguntas y en otros simplemente dice a una que no se especifica, representando una inconsistencia en estos casos a depurar en el futuro.

8. Conclusiones y trabajo futuro

En esta sección se recogen las principales experiencias adquiridas durante el desarrollo, destacando los aprendizajes más relevantes y los retos superados. También se analizan las limitaciones encontradas a lo largo del proceso y se sugieren posibles mejoras que podrían implementarse en futuros proyectos.

8.1. Aprendizaje y experiencia en el desarrollo

A lo largo de la realización de este proyecto, se han adquirido conocimientos clave en múltiples áreas, principalmente en el desarrollo de aplicaciones automatizadas para la extracción y procesamiento de datos geoespaciales. La integración de técnicas de scraping con el procesamiento de lenguaje natural (PLN) ha sido fundamental para la creación de una solución escalable y eficiente. Estos conocimientos no solo se limitan a la programación, sino que también han involucrado la comprensión de los modelos de lenguaje, la gestión de bases de datos y el trabajo con APIs de terceros, lo que ha ampliado considerablemente el alcance del aprendizaje técnico adquirido.

El uso de herramientas como Scrapy para realizar el scraping de datos y el modelo de lenguaje natural para procesar las respuestas ha sido una parte crucial del proyecto. Aunque la teoría detrás de estas tecnologías es relativamente conocida, la implementación real ha implicado muchos desafíos prácticos, como la gestión de estructuras de datos complejas o la adaptación de modelos de lenguaje para mejorar su precisión en la extracción de respuestas relevantes.

Otro aprendizaje importante ha sido la necesidad de optimización del sistema. Durante las pruebas de rendimiento, se detectaron áreas en las que el proceso de scraping y la interacción con el modelo LLM podían mejorarse, de esta manera se optó por implementar la base de datos para no repetir el scraping por ejemplo. Este proceso de identificación y solución de cuellos de botella ha sido invaluable, ya que permite no solo mejorar la experiencia de usuario, sino también garantizar que la solución sea viable para un mayor volumen de datos.

Si tuviera que destacar un aspecto particular, sería la implementación del sistema de scraping de URLs y sobre todo la generación de respuestas con el LLM, que debido a las limitaciones propias de un LLM de tamaño reducido ha presentado bastantes dificultades y cambios de modelos. Estos elementos fueron técnicamente desafiantes, pero me proporcionaron una comprensión más profunda sobre cómo integrar y optimizar estos sistemas de manera efectiva, lo que ha sido un hito en el desarrollo de este proyecto.

En conclusión, este proyecto ha proporcionado una valiosa oportunidad para aplicar y

expandir los conocimientos adquiridos a lo largo de mi carrera, enfrentándome a retos reales en la creación de sistemas automatizados de consulta y extracción de información. Gracias a este proyecto, no solo he aprendido a programar de manera más eficiente, sino que también he adquirido una perspectiva más amplia sobre el desarrollo de sistemas complejos y la integración de diversas tecnologías.

8.2. Limitaciones y posibles mejoras

Aunque el proyecto ha sido un éxito en términos generales, durante el desarrollo han surgido diversas limitaciones. La principal de ellas ha sido el tiempo disponible. Al tratarse de un proyecto individual y con plazos relativamente ajustados, no ha sido posible dedicar el tiempo necesario para pulir todos los detalles, especialmente en áreas como la optimización del rendimiento o la mejora de la precisión del modelo de lenguaje natural.

Una de las limitaciones más notables ha sido el rendimiento del sistema en el procesamiento de grandes volúmenes de datos. Si bien el sistema es eficaz para manejar un número moderado de POIs, en casos donde los sitios web a scrapear son muy grandes o tienen una estructura compleja, los tiempos de respuesta pueden verse afectados. En estos casos, la implementación de técnicas más avanzadas de procesamiento de datos o algún tipo de filtrado de páginas a la hora de scrapear podría ayudar a mejorar la eficiencia.

En cuanto al modelo de lenguaje natural, aunque se han logrado buenos resultados en la mayoría de las consultas, existen ocasiones en las que el modelo genera respuestas inexactas o incompletas, especialmente cuando se enfrenta a preguntas complejas o cuando el HTML de la página web no está lo suficientemente estructurado. Dado que el modelo utilizado es self-hosted, es importante destacar que esta implementación puede presentar algunas limitaciones inherentes a la infraestructura local, lo que puede afectar la capacidad de interpretación y extracción de datos relevantes en algunos casos.

En términos de mejoras futuras, una sería la integración de un sistema de almacenamiento y gestión de consultas. Actualmente, el sistema es capaz de procesar las consultas en tiempo real, pero no cuenta con un sistema de almacenamiento que permita realizar consultas previas o guardar resultados entre sesiones. Implementar una base de datos que almacene las consultas realizadas y sus respuestas podría ser una gran mejora para el rendimiento y la experiencia del usuario.

Finalmente, la interfaz de usuario podría mejorarse mediante la creación de una versión gráfica que facilite la interacción con el sistema, lo que lo haría accesible a un público más amplio. Además, la incorporación de nuevas funcionalidades, como la personalización de las consultas o la integración con otros servicios de mapas, podría hacer que el sistema sea aún más útil y versátil.

8.3. Conclusión

En resumen, este proyecto ha permitido desarrollar una solución automatizada que mejora significativamente el proceso de obtención de información de Google My Maps. A pesar de las limitaciones de tiempo y el tamaño de los datos, se ha logrado crear un sistema funcional que es capaz de responder preguntas en lenguaje natural sobre puntos de interés de manera eficiente. Este trabajo ha demostrado que la combinación de técnicas de scraping y procesamiento de lenguaje natural tiene un gran potencial para automatizar la extracción y consulta de datos geoespaciales, y abre la puerta a futuras mejoras y desarrollos en este campo. Con las optimizaciones adecuadas, este sistema puede ser utilizado en una amplia gama de aplicaciones, desde la investigación de mercados hasta la gestión de negocios y más allá.

Bibliografía

1. Almeida, F., & Xexéo, G. (2019, January 25). *Word embeddings: A survey*. arXiv.org. https://arxiv.org/abs/1901.09069

2. Breuss, M. (2024, December 2). *Beautiful soup: Build a web scraper with Python*. Real Python. https://realpython.com/beautiful-soup-web-scraper-python/#step-2-scrape-html-content-from-a-page

3. Dagdelen, J., Dunn, A., Lee, S., Walker, N., Rosen, A. S., Ceder, G., Persson, K. A., & Jain, A. (2024). Structured information extraction from scientific text with large language models. *Nature Communications*, 15(1). https://doi.org/10.1038/s41467-024-45563-x

4. Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018, October 11). *BERT: Pre-training of deep bidirectional transformers for language understanding*. arXiv.org. https://arxiv.org/abs/1810.04805

5. Forero, A. F. G., & Duque, A. F. O. (2023). Evolución del procesamiento natural del lenguaje. *TecnoLógicas*, 26(56), e2687. https://doi.org/10.22430/22565337.2687

6. Gubitosa, B. (2025, February 28). Self-hosted LLM: A comprehensive guide to hosting LLMs. Plural. https://www.plural.sh/blog/self-hosting-large-language-models/

7. Kamath, U., Graham, K. L., & Emara, W. (2022). *Transformers for machine learning*. https://doi.org/10.1201/9781003170082

8. Màrquez, L., Padró, L., & Rodríguez, H. (2000). A machine learning approach to POS tagging. *Machine Learning*, 39(1), 59–91. https://doi.org/10.1023/a:1007673816718

9. Prompting techniques – Nextra. (2025, April 24). https://www.promptingguide.ai/techniques

10. Scrapy 2.13 documentation — Scrapy 2.13.2 documentation. (n.d.). https://docs.scrapy.org/en/latest/

11. System log parsing with large language models: A review. (n.d.). arXiv.org. https://arxiv.org/html/2504.04877v2

12. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, June 12). *Attention is all you need*. arXiv.org. https://arxiv.org/abs/1706.03762v7

13. Vicente, A. T. D., Camperos, M. C., & De La Mata, D. Á. C. (2024). La evolución del procesamiento del lenguaje natural y su influencia en la inteligencia artificial: Una revisión y líneas de investigación futura. *European Public & Social Innovation Review*, 10, 1–23. https://doi.org/10.31637/epsir-2025-782

14. Xiao, T., & Zhu, J. (2023, November 29). Introduction to transformers: An NLP perspective. arXiv.org. https://arxiv.org/abs/2311.17633

15. ZenRows. (2024, May 16). Scrapy in Python: Web scraping tutorial 2025. ZenRows. https://www.zenrows.com/blog/scrapy-python

A. Glosario de Acrónimos

A continuación se listan y describen los acrónimos y términos más utilizados en este trabajo:

- **KML**: Keyhole Markup Language, lenguaje basado en XML utilizado para almacenar y representar datos geográficos, utilizado por Google Earth y Google My Maps.
- **POI**: Point of Interest, punto de interés geográfico marcado en un mapa, como un negocio, monumento, restaurante, etc.
- **Google My Maps**: Herramienta de Google que permite a los usuarios crear mapas personalizados añadiendo ubicaciones, rutas y descripciones.
- **Web Scraping**: Técnica que consiste en extraer datos de páginas web de forma automatizada.
- **Scrapy**: Framework en Python para la realización de tareas avanzadas de scraping web.
- **BeautifulSoup**: Librería de Python que permite analizar y extraer información de documentos HTML y XML.
- **HTML**: HyperText Markup Language, lenguaje de marcado utilizado para estructurar contenido en la web.
- **LLM**: Large Language Model, modelo de lenguaje natural entrenado con grandes volúmenes de texto para generar o interpretar lenguaje humano.
- **Prompt**: Instrucción o entrada textual que se le proporciona a un modelo LLM para que genere una respuesta específica.
- **LM Studio**: Plataforma que permite ejecutar modelos LLM de manera local utilizando una interfaz gráfica y APIs.
- **Self-hosted**: Modelo de software que se ejecuta localmente en el equipo del usuario, en lugar de depender de servicios en la nube.
- **Prompt Engineering**: Técnica que consiste en diseñar instrucciones o entradas óptimas para maximizar la efectividad de los LLMs.
- **Parsing**: Proceso de interpretar y convertir una salida de texto (por ejemplo, de un modelo LLM) en un formato estructurado.
- **DOM**: Document Object Model, representación jerárquica de los elementos de una página web que permite interactuar y modificar su estructura desde scripts.

- **robots.txt**: Archivo que indica a los robots de indexado o scraping qué secciones de una web pueden o no ser accedidas automáticamente.
- **JSON**: JavaScript Object Notation, formato ligero de intercambio de datos, utilizado para estructurar las respuestas del sistema.
- **Extensión de navegador**: Aplicación ligera que amplía la funcionalidad de un navegador web, como Chrome o Firefox.
- **XPath / JSPath / Selector CSS**: Métodos para localizar elementos dentro del DOM de una página web.
- **Token**: Unidad mínima de información procesada por un modelo LLM, como una palabra o parte de una palabra.
- **Prompt de ranking**: Instrucción específica que permite ordenar páginas web en función de su relevancia para una pregunta dada.
- **Consultas en lenguaje natural**: Preguntas formuladas de manera conversacional, que no requieren un formato técnico.
- **Base de datos (MySQL)**: Sistema de almacenamiento estructurado de datos utilizado para guardar los HTML y respuestas.

B. Anexo I (Manual de usuario)

A continuación, se explicará el proceso que debe seguir un usuario para utilizar la información en detalle:

En primer lugar, el usuario debe abrir un mapa de Google My Maps y marcar los POIs que considere. Una vez marcados todos los deseados, se clica sobre la extensión de navegador y se pulsa el botón *Iniciar Proceso*. Una vez termine el proceso de copiado de las URLs en el campo de descripción del marcador, se podrá exportar como KML el mapa y guardar este archivo KML en la ruta base de la aplicación de línea de comando.

Una vez guardado el mapa, podremos ejecutar la aplicación de terminal mediante el siguiente comando:

```
python -m main --kml aljub.kml --query "que tipo de negocio es, cual es su horario de apertura y donde está" --db datos_desde_db.json --deep 2
```

Código B.1: Ejemplo de comando de ejecución de la aplicación desde línea de comando

Este comando, como podemos ver, cuenta con algunos argumentos que en este caso ayudarán al funcionamiento de la aplicación o le indicarán algunos aspectos para hacerla funcionar:

- `-m main`: este argumento deberá ser fijo y simplemente declara que se ejecutará el módulo llamado `main`, que realizará la llamada al resto de módulos.
- `--kml`: en este argumento podremos indicar qué fichero KML queremos que utilice la aplicación, en este caso `aljub.kml`.
- `--query`: con este argumento indicaremos la pregunta a realizar a la aplicación.
- `--db`: este argumento indica el nombre que le queremos dar a un fichero JSON donde se guardará el código HTML limpiado junto a su URL y profundidad de todas las URLs scrapeadas.
- `--deep`: muestra la profundidad a partir de la URL raíz a la que deseamos scrapear para buscar las respuestas.