Generación de historias por secciones utilizando Large Language Model
Dialog generation for stories using Large Language Models

Trabajo de Fin de Grado
Curso 2024–2025

Autor
Pablo Enrique Padial Iniesta

Director
Pablo Gervás Gómez-Navarro
Gonzalo Méndez Pozo

Grado en Ingeniería de Computadores
Facultad de Informática
Universidad Complutense de Madrid

Generación de historias por secciones utilizando Large Language Model
Dialog generation for stories using Large Language Models

Trabajo de Fin de Grado en Ingeniería de Computadores

Autor
Pablo Enrique Padial Iniesta

Director
Pablo Gervás Gómez-Navarro
Gonzalo Méndez Pozo

Convocatoria: *Septiembre 2025*

Grado en Ingeniería de Computadores
Facultad de Informática
Universidad Complutense de Madrid

DIA de MES de AÑO

Dedicatoria

A mis padres y mi hermana.

Agradecimientos

A la familia y amigos por el apoyo y la ayuda.

Resumen

Generación de historias por secciones utilizando Large Language Model

La mejora de los LLMs (*Large Language Models*) han permitido su uso en aspectos creativos como la generación de historias o relatos. Este trabajo presenta una aplicación llamada **historIAs** para la creación de historias asistidas por modelos LLMs, mediante un flujo guiado por secciones: *trama e hilo simbólico, descripción del mundo, personajes principales, descripción de escenarios, estructura de la historia* y finalmente la escritura de los capítulos. Este sistema permite personalizar y editar la historia antes y durante la redacción. La información se almacena en una base de datos dando la posibilidad al usuario de continuar con la historia en cualquier otro momento.

Para evaluar la calidad de las historias, se plantea una valoración de diferentes usuarios buscando contrastar la producción narrativa del modelo con la escritura humana e identificar fortalezas y limitaciones. Finalmente se discute el encaje de la generación de historias mediante LLMs como apoyo para escritores en su proceso creativo.

Palabras clave

LLM, generación de historias, escritura asistida, IA, Llama, Transformer, token.

Abstract

Dialog generation for stories using Large Language Models

An abstract in English, half a page long, including the title in English. Below, a list with no more than 10 keywords.

Keywords

LLM, story generation, assisted writing, AI, Llama, Transformers, token.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Durante los últimos años la forma de afrontar problemas, de buscar información o de generar ideas está cambiando de manera acelerada. El auge que vive actualmente la inteligencia artificial generativa ha irrumpido de lleno en la mayoría de sectores de la sociedad. La facilidad de acceso a estas herramientas, junto con la calidad de los resultados en la generación de textos, resúmenes, imágenes, vídeos o en la automatización de tareas tediosas, ha provocado un uso generalizado de estas tecnologías incluso en actividades cotidianas. El uso, en muchas ocasiones abusivo, de modelos de inteligencia artificial en ámbitos creativos ha generado rechazo en parte de la sociedad, especialmente en sectores como la ilustración, la traducción o la composición musical, donde existe preocupación por la pérdida de valor humano y de empleo. En el caso de la *generación de historias* ocurre algo similar: muchos escritores y lectores cuestionan el valor artístico de un texto producido por una máquina, al considerar que carece de la profundidad emocional y de la experiencia vital que aporta un autor humano. Además, es importante destacar los sesgos derivados de errores de diseño o presentes en los datos de entrenamiento, que pueden llevar a reproducir estereotipos o formas de discriminación. Pese a ello, la escritura creativa generada por los LLMs (*Large Language Models)* puede entenderse también como una oportunidad, no para sustituir a escritores o novelistas, si no como apoyo a la hora de creación e imaginación. En este ámbito se presenta el proyecto, que pretende servir como guía y apoyo a autores que quieran escribir historias, ofreciendo un flujo estructurado y editable. Además, se plantea un análisis de las historias generadas con el fin de evaluar su calidad narrativa y comprobar hasta qué punto pueden considerarse comparables a las creadas por un autor humano. Esta evaluación no busca sustituir al escritor, sino valorar el potencial de la inteligencia artificial como herramienta creativa, así como identificar sus limitaciones actuales en cuanto a originalidad, coherencia y profundidad literaria.

1.1. Motivación

Actualmente existen múltiples páginas web y aplicaciones que permiten generar historias de manera automática mediante inteligencia artificial (Summarizer [1], Wrizzle [2]) que pueden crear pequeños relatos a partir de un tema y una pequeña

configuración. Sin embargo, en la mayoría de casos la personalización por parte del usuario es mínima, sin ofrecer un control real sobre el proceso creativo. Esto provoca una edición baja o inexistente donde el usuario únicamente puede reinciar la generación completa. Frente a estas limitadas opciones, la motivación principal de este proyecto es desarrollar una herramienta que no solo genere una historia completa, sino que también ofrezca un gran nivel de edición y personalización. De esta forma, ayudar y guiar a los usuarios con sus propias historias a través de un gran nivel de detalle. Para ello, la narrativa se divide en secciones clave: la *trama e hilo simbólico* que marca el rumbo general de la obra, los *personajes principales* donde se define sus atributos y descripción, los *escenarios principales* y la *estructura de los capítulos*. El objetivo es construir la historia a través secciones donde el usuario pueda reescribir, pedir modificaciones o editar manualmente cada una de ellas y así se ajuste a sus preferencias. Además, se dispone de un modo de generación rápida en el que se pretende generar la historia a partir de los datos introducidos en un cuestionario, creando historias de forma mucho más rápida, sacrificando parte de la edición y personalización caracteristica de la otra opción. Con esto surge otro análisis: evaluar si la calidad y coherencia de la historia generada varían en función del modo de creación (ver la sección 3.2.1 para más detalles).

1.2. Objetivos

El objetivo principal de este TFG consiste en desarrollar una aplicación que guíe la creación de historias mediante modelos de lenguaje, permitiendo la generación estructurada de historias y ofreciendo un alto grado de edición y personalización al usuario. Los objetivos específicos que plantea el proyecto son:

- Implementar un LLM (se utiliza el servidor `Groq`) que permita la generación de historias, asegurando una buena gestión de *tokens* para no superar el límite de uso del modelo.

- Diseñar una interfaz de usuario interactiva que permita al usuario generar, reescribir, modificar y editar manualmente cada sección de la historia para un mayor ajuste a sus preferencias.

- Diseñar el flujo narrativo dividiendo la historia en secciones clave (trama, personajes, escenarios, mundo, capítulos) que faciliten la edición y personalización.

- Almacenar las secciones generadas en una base de datos (`MongoDB`) para gestionar la persistencia de los datos y permitir que el usuario pueda retomar la historia en cualquier momento.

- Ofrecer dos modos de generación de historias: uno *detallado* y otro *rápido*, para adaptarse a las necesidades del usuario y comparar la calidad y coherencia de las historias generadas en función del modo de creación.

- Evaluar la calidad narrativa de las historias generadas, analizando su coherencia y profundidad literaria en comparación con las creadas por un autor humano.
- Identificar las limitiaciones actuales de los LLMs en la generación de historias proponiendo líneas de mejoras futuras.

1.3. Estructura de la memoria

El proyecto está dividido en varios capítulos que abordan diferentes aspectos del desarrollo y evaluación de la herramienta:

- **Capítulo 1: Introducción:** Presenta el contexto del proyecto, la motivación y los objetivos.
- **Capítulo 2: Estado de la cuestión:** Se revisan los trabajos previos y las principales líneas de investigación relacionadas con la generación automática de historias. Se expone la evolución de la IA desde Alan Turing hasta el día de hoy y los fundamentos técnicos de los LLMs. Además, se comentan varias novelas publicadas escritas por inteligencia artificial.
- **Capítulo 3: Descripción del trabajo:** Presenta el capítulo principal del proyecto donde se desarrolla todo lo relacionado con la aplicación. El capítulo inicia con una sección dedicada a exponer los recursos y herramientas principales utilizadas durante el proyecto: `Groq`, `Flet` y `MongoDB`. Seguimos con la Sección 3.2 donde se expone de forma detallada los dos modelos de creación de historias, diagramas de flujo y de componentes. Se incluye un apartardo donde se detalla el almacenamiento en la base de datos y finalmente se hace referencia a la importancia de los *prompts del sistema* para guiar al modelo, y se muestran los más relevantes. Para terminar esta sección, se detalla las pantallas de la aplicación y el funcionamiento de `Flet`.

Capítulo 2

Estado de la Cuestión

Los LLMs (Large Language Models) han tranformado radicalmente el procesamiento del lenguaje y la generación de texto. A lo largo de este capítulo se presenta un analisis de los LLMs, con un enfoque en su aplicación en la generación literaria mediante Inteligencia Artificial Generativa. Se incluye un cronograma con los hitos más relevantes, desde el surgimiento de los LLMs, gracias a la arquitectura *Transformers*, hasta el modelo *Llama 3.3-70b* empleado en el proyecto y destacando obras literarias generadas por LLMs.

2.1. Orígenes y fundamentos generales de la Inteligencia Artificial

La Inteligencia Artificial es un campo de la informática que busca desarrollar sistemas capaces de realizar tareas que, tradicionalmente, requerían de inteligencia humana, como el razonamiento, el aprendizaje o la comprensión del lenguaje. Se busca diseñar sistemas capaces de tomar decisiones y aprender a partir de ejemplos y experiencias.

Las primeras ideas sobre la Inteligencia Artificial se remontan a mediados del siglo XX, con hitos como el *Test de Turing* propuesto por Alan Turing en 1950 [4], considerado uno de los padres de la computación. Se trata de una prueba para evaluar la capacidad de una máquina de exhibir un comportamiento inteligente similar o indistinguible al de un ser humano.

Unos años más tarde en 1956, durante la conferencia de Dartmouth, *Dartmouth Summer Research Project on Artificial Intelligence*, organizada por John McCarthy, fue acuñado el término "inteligencia artificial" [5]. Durante las décadas de 1950 y 1960, comenzaron a surgir los primeros programas de IA centrados en la resolución de problemas lógicos y el procesamiento simbólico, como el Logic Theorist (1956) [6] y el General Problem Solver (1959) [7]. Sin embargo, debido a las limitaciones computacionales y la falta de datos provocaron un estancamiento en el desarrollo.

A partir de los años 80 y 90, el desarrollo de los algoritmos de *machine learning* (aprendizaje automático) permitió que los patrones "aprendieran" a partir de datos.

No fue hasta los años 2000's cuando pudo ser implementado, en parte, gracias

a la disponibilidad de grandes volumenes de datos y la disponibilidad de hardware económico con un gran poder de cálculo como fueron las primeras *Unidades de Procesamiento Gráfico* (GPU). Gracias a estos grandes avances tecnológicos, se pudieron construir las redes neuronales con cientos de capas de neuronas, dando nacimiento al termino *Deep Learning* (redes neuronales profundas), mediante las cueles se puede generar y entrenar modelos a partir de miles de datos.

En 2017 se presenta el artículo de investigación *Attention Is All You Need* donde se introduce la arquitectura *Transformers*[14]. La combinación de grandes volúmenes de datos, potentes unidades de procesamiento gráfico (GPU) y arquitecturas avanzadas como *Transformers* ha impulsado la actual generación de LLMs, capaces de producir texto coherente, contextual y creativo a una escala sin precedentes. La evolución de los LLMs ha marcado un punto de inflexión en la inteligencia artificial, permitiendo la generación de texto con coherencia, estilo y contexto. En particular, su aplicación en narrativa literaria ha abierto nuevas posibilidades creativas, desafiando límites de la autoría y la imaginación humana. En la Figura 2.1 se observa el aumento significativo de artículos científicos relacionados con los LLMs.

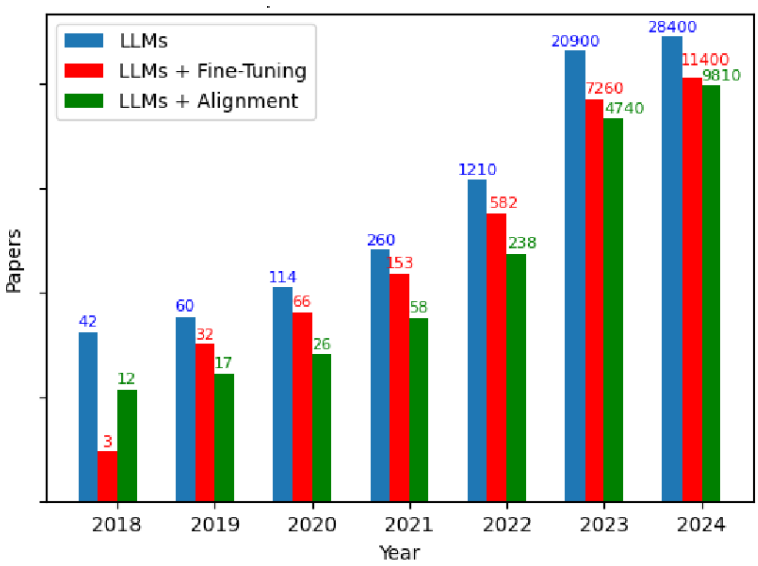


Figura 2.1: Número de artículos científicos relacionados con los LLMs en los últimos años (fuente del gráfico [10])

2.1.1. Fundamentos técnicos de los LLMs

Los LLMs se basan en la arquitectura *Transformers*, cuyos modelos se entrenan de forma auto-supervisada leyendo cantidades colosales de texto y aprendiendo a predecir la siguiente palabra de una frase. Desde BERT hasta GPT-5, los modelos han escalado en parámetros y capacidades. A finales de 2024, Meta presentó *Llama 3.3-70b*, un modelo de 70 mil millones de parámetros, optimizado para tareas multilingües y generación creativa.

La arquitectura del *Transformers* se basa en un *encoder* y un *decoder*. El *encoder* es la parte del *Transformers* encargada de leer los *tokens* de entrada previamente

convertidos en vectores de incrustación (*embedding*) y poder representar sus características semánticas.

El *decoder* contiene un mecanismo que combina la información interna del *decoder* con la información proveniente del *encoder*. La función mapea una consulta y un conjunto clave-valor a una salida, donde la salida es la suma ponderada de los valores, con los pesos determinados por una función de compatibilidad entre la consulta y las claves correspondientes. Así, cada *token* generado integra el historial parcial de salida con la información del input. De esta forma, el resultado, es un proceso donde cada nuevo *token* se produce a partir de la información previa generada y la representación del *input* proveniente del *encoder*. Este diseño, paso a paso, es especialmente eficaz en tareas secuencia-a-secuencia como traducción o resumen.

En la Figura 2.2 se muestra el funcionamiento del *Transformers*.

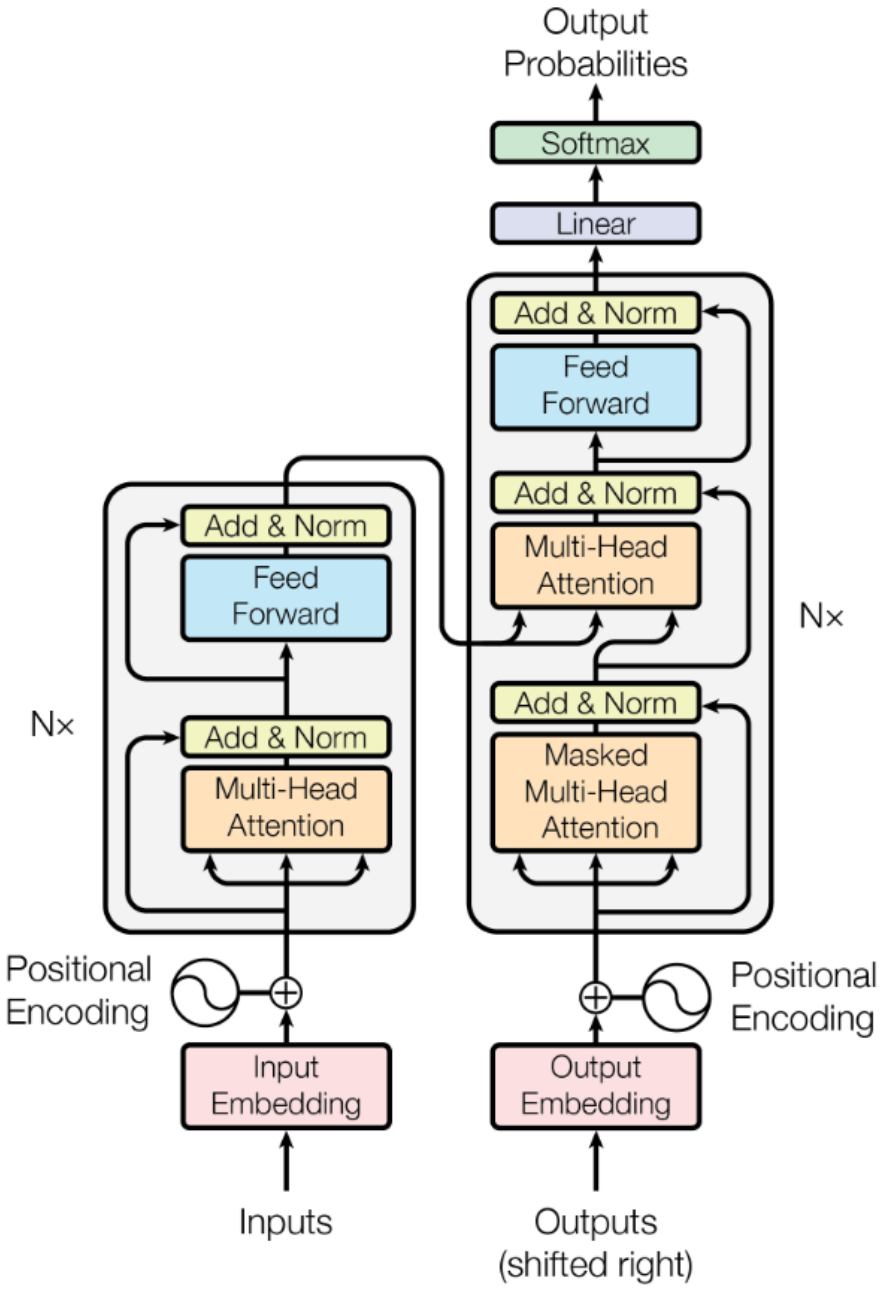


Figura 2.2: Arquitectura *Transformers* (fuente de la imagen [15])

Otro aspecto importante del proceso de los LLMs es la *tokenización*. Un *token* es la unidad mínima de texto que el modelo procesa, ya sea una palabra completa o un segmento subpalabra resultante de dividir una palabra en partes más pequeñas. El proceso de *tokenización* convierte el texto en una seria de *tokens* con un identificador único (*token ID*). El resultado es una secuencia numérica que representa el texto original en un formato manejable para el modelo. La *tokenización* es un proceso clave para el rendimiento: la eficiencia depende en gran medida del número de *tokens* en que se divide un texto. Dada una sucesión de m *tokens*, $x_1, \ldots, x_m$, se busca generar los siguientes n *tokens* para obtener la sucesión $x_1, \ldots, x_m, \ldots, x_{m+n}$. Holtzman,

Buys, et al. en su trabajo [3] consideran que los modelos calculan la probabilidad

$$P(\{x_i\}_{i=1}^{i=m+n}) = \prod_{i=1}^{m+n} P(x_i|x_1, \ldots, x_{i-1}),$$

para generar la continuación *token* a *token* mediante una estrategia de decodificación determinada, como por ejemplo la *decodificación basada en maximización*, siendo una de las más usadas.

Para un mayor recorrido histórico acerca de la IA y los LLMs, puede verse el trabajo de Dilli Hang Rai [8]. A continuación se presenta una línea del tiempo con los hitos más relevantes de los LLMs (ver Figura 2.3).

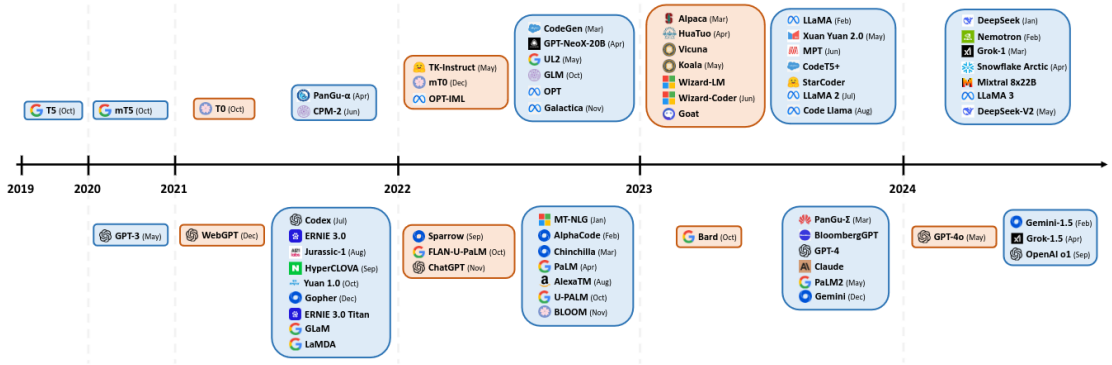


Figura 2.3: Línea del tiempo de lanzamientos de LLM (fuente del gráfico [10])

En la Figura 2.3 además de apreciarse el incremento de propuestas de LLMs pueden encontrarse los nombre de los LLMs más singnificativos que han surgido en los últimos años. Los recuadros azules representan modelos preentrenados (pretrained), mientras que los naranjas corresponden a modelos afinados con instrucciones (instruction-tuned). Los modelos en la mitad superior indican código abierto (open-source), y los de la mitad inferior son de código cerrado (closed-source). El gráfico muestra una tendencia creciente hacia los modelos afinados con instrucciones y de código abierto, destacando la evolución del panorama y las tendencias en la investigación de procesamiento del lenguaje natural.

2.2. Generación de historias

Con los últimos avances tecnológicos y la disponibilidad de millones de datos, los LLMs han alcanzado una capacidad notable para producir texto largo, fluido y contextualizado. Está mejora no solo permite tareas clásicas (resúmenes, reescritura, reformulación), sino que da paso a la generación narrativa. A partir de un simple *prompt* o una idea, el modelo es capaz de construir una historia desarrollando escenas, personajes y mantener la coherencia a lo largo de la generación. Un gran momento de los LLMs generativos se produjo en 2019, con la introducción de `GPT-2` (OpenAI), un modelo con más de un millón de parámetros entrenado con una gran cantidad de datos. La calidad de un modelo genérico, no orientado a la narrativa, a

la hora de continuar historias a partir de una frase, fue un gran avance en el uso de modelos *pre-entrenados*. Sin embargo no fue hasta finales de 2022 cuando se produjo el mayor auge de los LLMs en la sociedad, el lanzamiento para todo el mundo de `ChatGPT`, por parte de (OpenAI). Esto provocó un cambio de paradigma en la sociedad. El fácil acceso al modelo permitió que millones de personas pudiesen utilizar por primera vez un modelo LLM para todo tipo de tareas, entre ellas crear historias. Sin duda este auge explica el creciente número de artículos sobre LLMs publicados recientemente (ver gráfico 2.1).

Existen precedentes de obras escritas integramente por IA. Ya en el siglo pasado se realizó un experimento donde el programa *Racter* generó *The Policeman's Beard Is Half Constructed* (1984) [12], que consistía en fragmentos de prosa. El texto finalmente fue editado y supervisado por el humano. No fue hasta 2017 cuando Ross Goodwin condujo desde Nueva York hasta Nueva Orleans con una IA conectada a sensores instalados en el vehículo, cuyos datos convirtió en palabras. La novela *1 the road* (2018) [11] fue publicada un año mas tarde en 2018 sin ninguna edición por parte del humano. En estos últimos años se han publicado más novelas escritas por IA y pulidas por el autor, siendo uno de los primeros casos *Iris* (2023) [13] escrita por David Guisado con la ayuda de `ChatGPT`.

A pesar de la mejora de los modelos y de la consistencia y coherencia a la hora de generar texto narrativo, seguía habiendo desafios al crear tramas extensas. Estos modelos tienden a divagar, olvidar detalles mencionados o introducir incoherencias en la historia a medida que se alarga el relato. Para poder abordar estos problemas, la investigación reciente se centra en integrar técnicas de planificación narrativa con modelos generativos. Se propusieron enfoques como el de *"Plan-and-Write"* mediante un modelo jerárquico [9]. En este método, dado un tema, primero se extrae una serie de eventos clave que surgen a lo largo de la historia, para que luego este esquema sirva de guia para producir el texto final. De esta forma se intenta que el modelo conserve el "plan" hacia donde trascurre la historia mientras la redacta y así tratar de evitar fallos de consistencia o esos "olvidos" producidos cuanto más larga es la historia. Los experimentos mostraron que incluir esta fase de planificación previa mejora notablemente la fidelidad, coherencia e interés de las historias obtenidas, en comparación con generar texto de una sola pasada sin un plan previo.

Para abordar estos problemas, este proyecto propone un sistema de generación de historias guiado por secciones. Previamente el usuario define las secciones con ayuda del modelo, pudiendo editar y personalizar cada una de ellas. Una vez definidas estas secciones, el modelo genera los capítulos de la historia haciendo uso de la información almacenada en la base de datos de cada sección. De esta forma el modelo siempre tiene acceso a la información relevante de la historia como la trama, personajes o escenarios, evitando así los problemas de coherencia y consistencia que suelen surgir en relatos largos. Este proyecto ha dado lugar a la aplicación que denomino **historIAs**.

Capítulo 3

Descripción del Trabajo

3.1. Herramientas utilizadas

En este proyecto se han empleado tres herramientas principales que han sido determinantes para el desasrrollo de la apliación: *Groq, Flet* y *MongoDB*.

- *Groq*: Es una empresa tecnológica desarrolladora de hardware específico para la inferencia de la inteligencia artificial [16]. *Groq* cuenta con el desarrollo de `Large Processing Units (LPUs)`, que se caracterizan por ser altamente eficientes y estar optimizadas para las tareas de inferencia en tiempo real. Las `LPUs` permiten ejucutar LLMs y otros modelos a mayores velocidades y hasta diez veces más eficientemente energeticamente en comparación con las GPUs. En un principio se planteaba la idea de utilizar un LLM de forma local pero enseguida vimos que no teníamos el hardware necesario para utilizarlo. Tras utilizar plataformas como `Google Colab` y ver que no era viable para proyectos más grandes, se optó por hacer uso de un servidor para correr el modelo. Inicialmente se barajaron servidores más conocidos en el mercado como `Microsoft Azure` o `Amazon Web Service` pero finalmente optamos por `Groq`, un servidor menos conocido pero ofrece una mayor velocidad y facilidad para el uso. Unicamente se precisa de una cuenta de usuario y una `API Key` para poder utilizar los modelos que ofrece en la version *Free*. A continuación, en la Tabla 3.1 se muestran los modelos que se han ensayado durante el proyecto: Nos hemos centrado en los modelos de `Llama` ya que son los más descargados y mejor valorados por la comunidad según `Hugging Face`, siendo el modelo más descargado `meta-llama/Llama-3.1-8B-Instruct` con 15 millones de descargas a fecha de este TFG. Al momento de seleccionar el modelo la parte más importante era los límites de uso que ofrecía cada uno. En nuestro caso (ver datos de la Tabla 3.1)*RPM* y *RPD* no son cruciales al ser un proyecto pequeño se hace imposible superar ese número de peticiones por minuto y por día. Lo relevante son los valores *TPM* y *TPD*, en especial el número de *tokens* por minuto por la demanda del "contexto" (ver la Sección 3.2.4 para más detalles) de la historia. A medida que se avanza en la generación de la historia, se tiene que añadir más información al LLM provocando, en ocasiones, que con una sola petición se supere el limite de *tokens* de `Groq`. Por este motivo, los modelos con

un *TPM* menor de 6K en numerosas ocasiones sobrepasaban el límite permitido por el servidor. Finalmente se optó por el modelo `llama3-70b-versatile` que posee el doble de límite de *tokens* por minuto permitiendo una mayor flexibilidad a la hora de generar la historia.

Tabla 3.1: Límites de uso de los modelos empleados.

Modelo	RPM	RPD	TPM	TPD
`deepseek-r1-distill-llama-70b`	30	1K	6K	100K
`llama-3.1-8b-instant`	30	14.4K	6K	500K
`llama3-70b-versatile`	30	1K	12K	100K
`llama3-8b-8192`	30	14.4K	6K	500K

RPM: peticiones/min; **RPD**: peticiones/día; **TPM**: tokens/min; **TPD**: tokens/día. Valores sujetos a cambios según cuenta/plan.

- *Flet*: Es un `framework` de código abierto basado en `Flutter`, lo que permite desarrollar aplicaciones web, de escritorio y moviles usando únicamente Python [17]. A diferencia de otras bibliotecas, `Flet` asbtrae la complejidad del `frontend`, simplificando las complejidades y proporcionando componentes listos para usar sin necesidad de tener conocimientos previos en `frontend`.

  En el proyecto se ha utilizado Flet para implementar la interfaz de usuario encargada de guiar al autor en el proceso de creación de historias. La decisión de emplear esta herramienta se debe a que no se quería combinar Python con otros lenguajes adicionales como PHP, HTML o CSS para el desarrollo de la parte visual. En su lugar, se buscó una solución que permitiera mantener toda la aplicación en un único lenguaje. Tras analizar distintas alternativas, se optó por Flet, un framework relativamente reciente que facilita la construcción de interfaces mediante componentes ya definidos, lo que agiliza notablemente el desarrollo y asegura una integración directa con la lógica implementada en Python.

- *MongoDB*: En el proyecto se ha utilizado `MongoDB` como sistema de gestión de base de datos [18]. Se trata de una base de datos NoSQL orientada a documentos, en la que la información se almacena en formato BSON (Binary JSON). Cada base de datos posee distitnas colecciones donde se almacenan los documentos con los datos a guardar. A diferencia de los sistemas relacionales, MongoDB no requiere esquemas fijos, lo que le otorga gran flexibilidad para manejar estructuras de datos cambiantes y jerárquicas. Un aspecto clave es que el desarrollador trabaja siempre con documentos en formato JSON. Cuando se inserta un documento, MongoDB se encarga automáticamente de convertirlo a BSON para almacenarlo de manera eficiente. De igual modo, al recuperar la información, MongoDB transforma el BSON de vuelta a JSON para que pueda manipularse de forma sencilla en la aplicación.

  La elección de esta herramienta se debe a la necesidad de almacenar entidades heterogéneas como personajes, tramas, escenarios o capítulos, cuya estructura

podía variar según la historia. Con una base de datos relacional habría sido necesario definir tablas y relaciones rígidas, lo que dificultaría la evolución del modelo. En cambio, con `MongoDB` es posible guardar directamente documentos con la estructura que se necesite en cada momento, simplificando la persistencia de datos y facilitando tanto la consulta como la actualización durante el proceso de generación de historias.

3.2. Diseño de la aplicación

Este capítulo presenta el proceso de diseño de la aplicación, ofreciendo una visión global de cómo se estructura y comporta el sistema, así como el flujo de información interno.

3.2.1. Modos de generación de historia

La aplicacion cuenta con dos modos principales de creación de historias, diseñados para adaptarse a los distintos estilos de trabajo y posibilidades:

- *Modo detallado:* Este modo se enfoca a tener una mayor personalización de la historia por parte del usuario. El proceso de creación se divide en varias secciones predeterminadas, tales como trama e hilo simbólico, descripción del mundo, personajes principales, desarrollo de escenarios, estrucutra de capitulos y la propia escritura de la historia. Una vez se genera la sección, el usuario a través de botones interactua generando, revisando y modificando cada sección antes de continuar. Este enfoque permite un alto grado de personalización y control sobre el contenido, así como la posibilidad de reescribir o editar manualmente cualquier parte de la historia.

- *Modo rápido:* Orientado a optener resultados inmediatos, este modo presenta al usuario un breve cuestionario con distitnos campos opcionales. El modelo va generando los capítulos siguiendo las indicaciones proporcionadas o, en ausencia de ellas, elaborando los elementos de forma creativa. Este modo no permite realizar modificaciones de forma manual sacrificando parte de la personalización en favor de la velocidad y simplicidad.

Con ambos modos se ofrece una gran flexibilidad aportando opciones para los usuarios que deseen un control exhautivo del proceso creativo y para los usuarios que precisen resultados más rápidos pero sacrificando parte de la personalización. Internamente, cada modo emplea un conjunto de `prompts` específicos adaptados a los requisitos del modo.

3.2.2. Diagramas de flujo

Para poder abordar con más detalle el diseño de la aplicación, haremos uso de distintos diagramas. Se ha dividido el diagrama de flujo para facilitar el detalle y la comprensión:

- *Inicio de sesión:* El usuario inicia sesión para acceder a la aplicación. Al iniciar la aplicación, el usuario se encuentra la pantalla de `Login` donde debe introducir su usuario y contraseña para poder acceder al menu principal. En caso de no disponer de una cuenta, posee la opción de registrarse creando una nueva. Se validan los datos introducidos, si son correctos, se accede al *Menú principal* (`Home`).

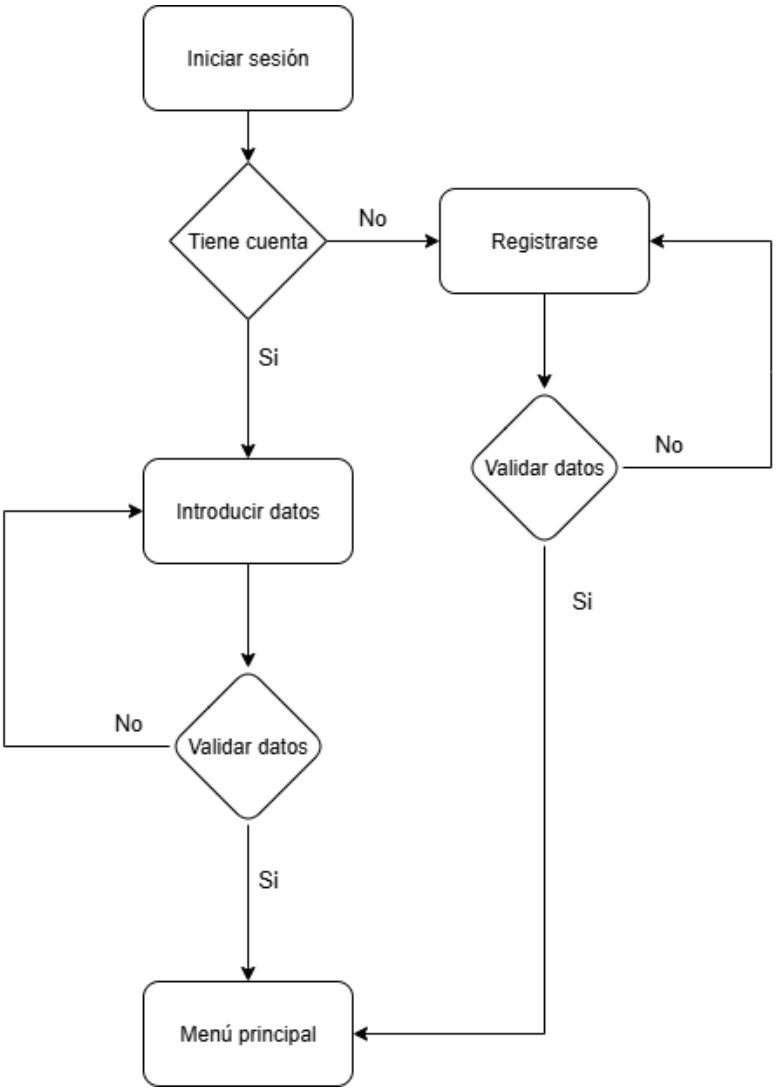


Figura 3.1: Diagrama de flujo *Inicio de sesión*

- *Flujo principal:* Enviar mensaje ya no es correcto, seria realizar accion o algo asi Una vez se accede al menú principal, el usuario puede realizar cuatro acciones principales: *eliminar historia, cargar historia, crear historia detallada* y *crear historia rápida*. En todo momento el usuario puede cargar una historia ya creada, permitiendo retomar el trabajo en cualquier momento. De igual forma, el usuario puede comenzar una nueva historia en cualquiera de ambos modos (ver Seccion 3.2.1 para más detalles) manteniendo varias iniciadas al mismo tiempo.

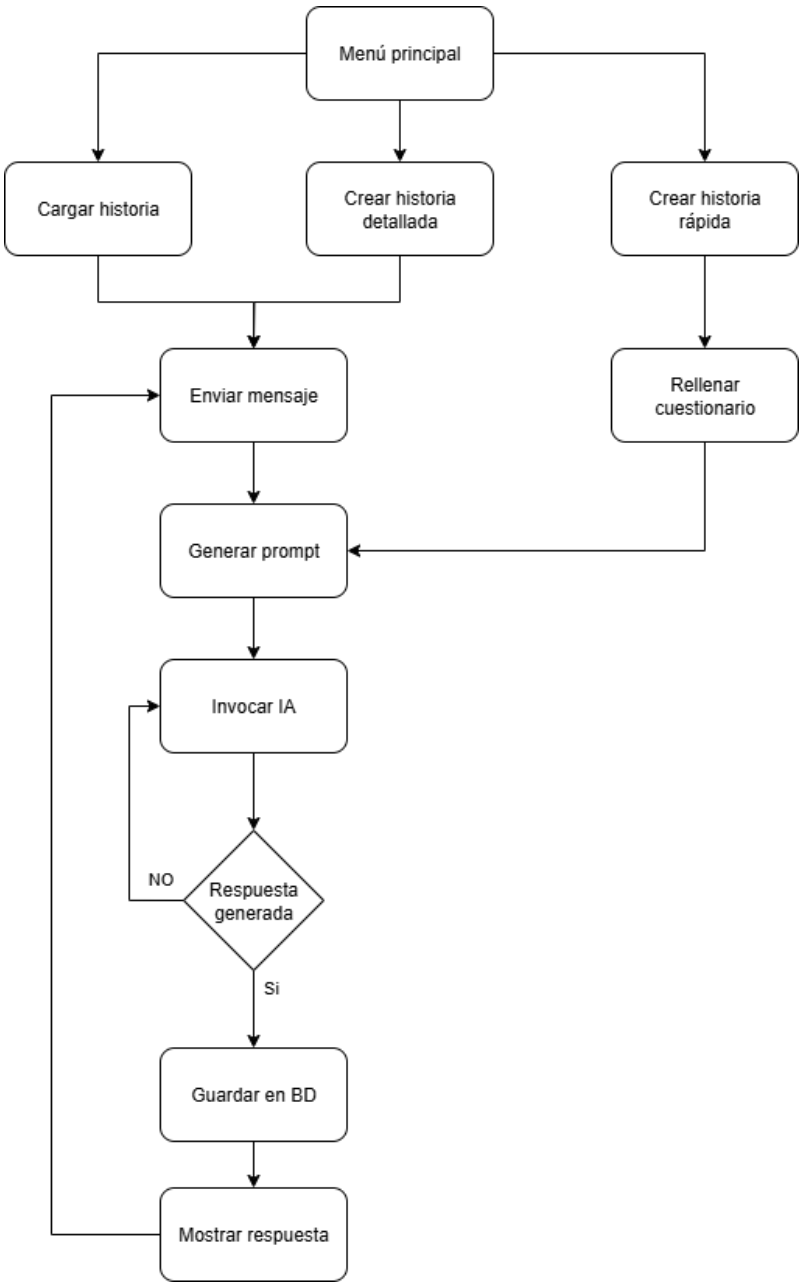


Figura 3.2: Diagrama de flujo *Flujo principal*

3.2.3. Diagrama de componentes

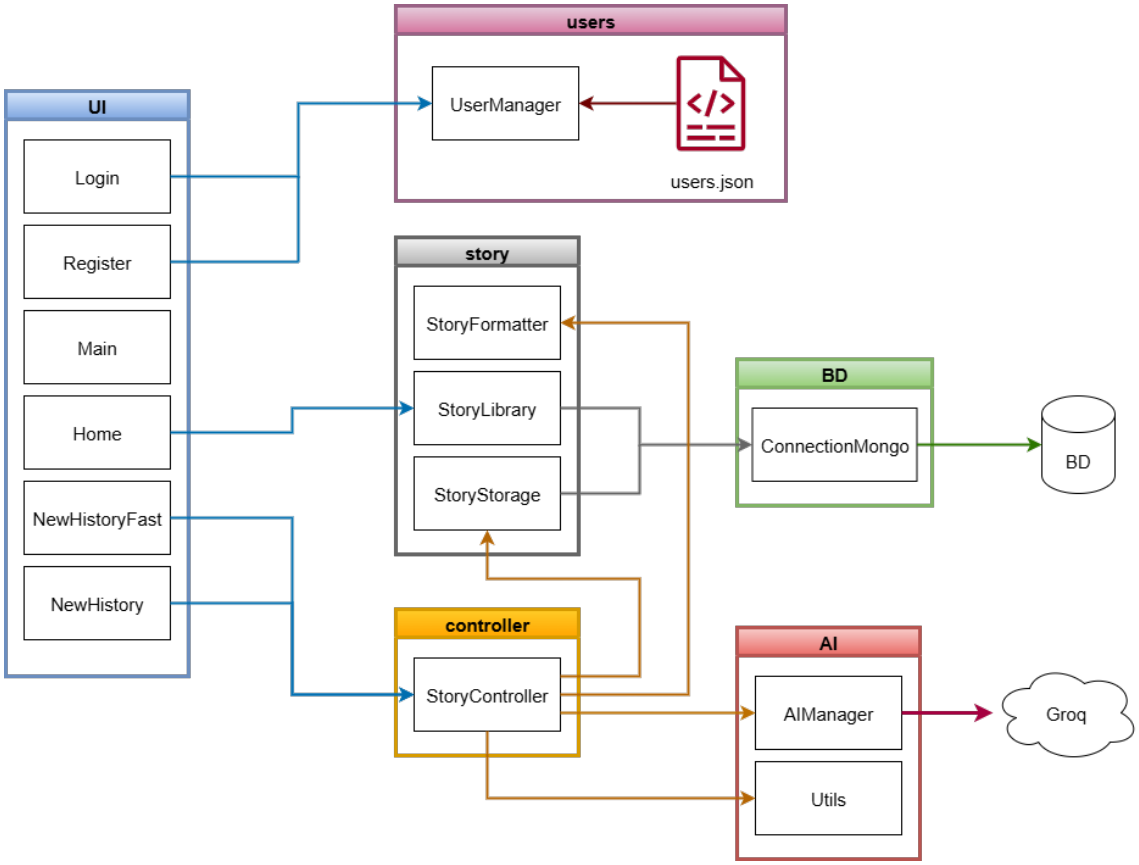


Figura 3.3: Diagrama de componentes

En este diagrama se observa como está estructurada la aplicación. Esta dividida en distintos paquete:

- *UI (interfaz de usuario)*: Se encuentran las clases que implementan `Flet` para crear la interfaz gráfica. Tenemos el `Main` que actúa como punto de arranque de la aplicación en `Flet` y se encarga de definir las rutas asociadas a cada pantalla. El resto de clases corresponden a las diferentes pantallas de la aplicación.

- *users:* La clase `UserManager` se encarga de la gestión de usuarios a la hora de realizar el `login` y el registro. Contiene un fichero `users.json` donde se almacena el nombre, correo y contraseña de cada usuario.

- *story:* Esta sección agrupa toda la lógica que hace posible la creación, formateo y almacenamiento de las novelas. `StoryFormatter` se encarga de todo lo relacionado con formatear el texto para convertirlo en un formato JSON que sea compatible con `MongoDB`. La clase `StoryLibrary` se encarga de gestionar las historias creadas por por el usuario y `StoryStorage` se encargar de crear todo lo relacionado con la historia, como las secciones, resúmenes y almacenado en la base de datos.

- *controller:* Su función es separar la lógica de la interfaz y las llamadas al servidor `Groq`. Se encarga de llevar la lógica del procesamiento del mensaje, detectar la sección por la que se encuentra el usuario, utilizar los métodos creados en `StoryStorage` y realizar las llamadas a `AIManager` que devuelven la respuesta generada por el modelo.

- *BD (base de datos):* Contiene la clase `ConnectionMongo` encargada de realizar las consultas en la base de datos.

- *AI:* Contiene `AIManager` encargado de establecer la conexión con el cliente `Groq` y realizar las llamadas al servidor. Además contiene un fichero `Utils` donde se almacenan los distintos `prompts` iniciales que se usarán para generar las respuestas.

3.2.4. Interacción con el modelo LLM

La interacción entre el LLM y el usuario es la parte principal de la funcionalidad de la aplicación. El sistema se apoya en la *API de Groq*, utilizando el modelo `llama-3.3-70b-versatile` [19], para generar el contenido narrativo. Este contenido se genera siguiendo una serie de pasos que se van a explicar en esta sección. Cada vez que el usuario realiza una acción desde `NewHistory`, esta clase crea un `thread` para separar la parte de la interfaz de la petición al servidor y así evitar bloqueos en la parte de la interfaz. Mientras este *thread* está activo, bloquea temporalmente la posibilidad al usuario de realiazar nuevas acciones. Esto permite que el usuario no pueda generar múltiples mensajes simultáneamente mientras se genera una respuesta, evitando así colisiones y duplicación de peticiones al servidor. Además el *thread* se conecta con `StoryController` que se encarga de realizar el procesamiento del mensaje introducido. El modo rápido de creación de historias funciona de igual forma, lo único que cambie es como se forman los mensajes.

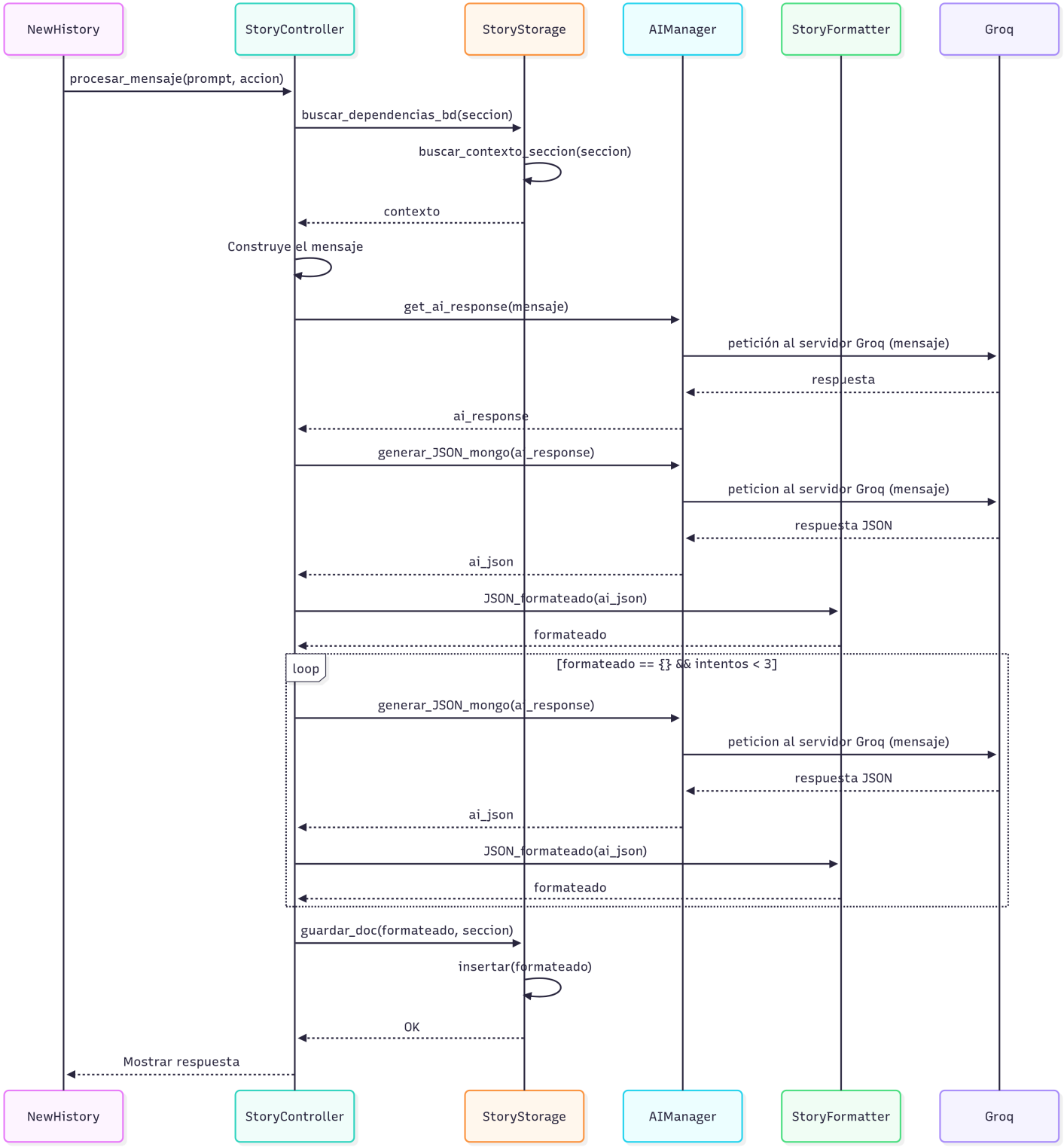


Figura 3.4: Diagrama de secuencia

Como se observa en la Figura 3.4, `StoryController` realiza la función de puente entre la parte de la interfaz y de la lógica. Esta clase se encarga de actualizar en todo momento la sección en la que se encuentra el usuario. Con dicha sección lla-

mamos a `buscar_dependencias_bd` de `StoryStorage` encargada de buscar para la sección actual que otras secciones se necesita como contexto el modelo para generar el `prompt`. Para cada sección se hace una búsqueda en la base de datos y se crea el contexto que se devuelve a `StoryController`. En el caso del modo rápido donde directamente se crean los capitulos, el contexto se forma pasando los capitulos anteriores. Ahora se construye el mensaje formado por el contexto creado anteriormente, la sección actual y el `input` del usuario. Con este mensaje se realiza una petición al servidor y genera una respuesta en formato texto. Para poder almacenar el contenido de la historia en `MongoDB`, se necesita convertir el texto a formato JSON. Durante el desarrollo nos encontramos con un problema fundamental: la respuesta generada por el LLM. Aunque en la mayoría de casos seguía las normas esperada, no siempre respetaba de forma exacta la misma estructura. Intentar depurar estas variaciones de forma manual haciendo uso, por ejemplo de `json.loads()` junto con expresiones regulares, resultó ser una tarea extremadamente compleja provocando una gran cantidad de errores ocasionados por un mínimo cambio inesperado en la respuesta. Para poder solucionar estas dificultades se opta por hacer uso nuevamente del LLM enviándole directamente la respuesta generada y pidiendo que la convierta en un objeto JSON válido. De este modo, la el modelo actúa como "formateado" de su propio `output`, liberándonos de la necesidad de escribir código de `parsing` extremadamente específico. Al igual que antes, el modelo en ocasiones comete errores y no siempre genera un JSON válido, lo que provoca un fallo al querer guardar en la base de datos. Para solucionar estos errores, se implementa un bucle que permite al modelo generar un nuevo JSON en cada iteración. Además, se incorpora un contador para evitar que el proceso se prolongue indefinidamente. Tras varias pruebas, comprobamos que al limitar el bucle a tres iteraciones se consigue una tasa de acierto prácticamente perfecta. Con este nuevo mensaje se llama a `get_ai_response` de `AI/AIManager.py` donde se realiza la llamada al cliente de *Groq*.

Código 3.1: Llamada al cliente de Groq

```
def get_ai_response(messages:str)->str:
    """Obtiene la respuesta de Groq AI y la limpia antes de
       mostrarla."""
    # Agregar mensaje del sistema
    messages.insert(0, {
        "role": "system",
        "content": PROMPT_INICIAL
    })

    completion = client.chat.completions.create(
        model="deepseek-r1-distill-llama-70b",
        messages=messages,
        temperature=0.8,
        max_tokens=4096,
    )

    # Obtener la respuesta sin <think>...</think>
    raw_response = completion.choices[0].message.content
    return clean_ai_response(raw_response)
```

Se añade al mensaje el *prompt inicial* que contiene la estructura y las reglas que debe de seguir el modelo para generar la historia. Seguidamente se hace la llamada al modelo configurando varios parámetros que influyen directamente en el comportamiento de la generación:

- `temperature`: Este parámetro controla el nivel de creatividad en la generación del texto por parte del modelo de lenguaje. Funciona ajustando la distribución de probabilidad con la que el modelo selecciona la siguiente palabra (*token*) en cada paso de la generación. Los valores varían entre 0 y 2 siendo 0 menos creatividad y más conservador y 2 más aleatoriedad. Normalmente se utilizan valores comprendidos entre 0 y 1, en nuestro caso se ha optado por seleccionar 0,8 para conseguir una gran creatividad pero sin perder la precisión a la hora de generar la respuesta. (Añadire unas referencias y graficas de articulos)

Caso de uso	**Rango**	**Descripción y ejemplo**
Generación de código	0,0 – 0,3	Alta precisión y coherencia. Ideal para scripts, funciones matemáticas y automatización.
Chatbots y asistentes	0,4 – 0,7	Equilibrio entre naturalidad y control. Útil para asistentes virtuales conversacionales.
Narrativa creativa	0,8 – 1,0	Estilo más libre y expresivo. Adecuado para cuentos, diálogos y contenido literario.
Exploración artística	>1,0	Generación altamente impredecible y experimental. Usado en arte generativo o literatura abstracta.

Tabla 3.2: Ajuste de `temperature` según el tipo de tarea

- `max_tokens`: Define el número máximo de *tokens* que puede devolver el modelo de lenguaje en una respuesta. Se ha establecido este tope a 4.096 *tokens* lo cual corresponde aproximadamente entre 2.000 y 2.500 palabras. Esta cantidad debe de convivir con el *prompt del sistema* y el mensaje generado, por eso se debe de ajustar cuidadosamente el máximo de *tokens* por respuesta para no desbordar el tope.

Ya con el mensaje generado, se guarda en la base de datos y se muestra la respuesta al usuario.

3.2.5. Almacenamiento en la base de datos

Para este proyecto unicamente se necesita guardar el texto generado en cada sección, por eso se ha decidido usar una base de datos no relacionada, facilitando su uso frente a una base de datos SQL. Cada sección de la historia se guarda en su propia colección, lo que facilita las consultas. Se crea una base de datos `info_usuarios` donde se almacenan documentos con la información más importante de cada historia:

usuario, nombre de la historia, modo y estado de las secciones. Esto ayuda a la hora de mostrar al usuario las historias que ha creado y pueda retomarla desde donde la dejó.

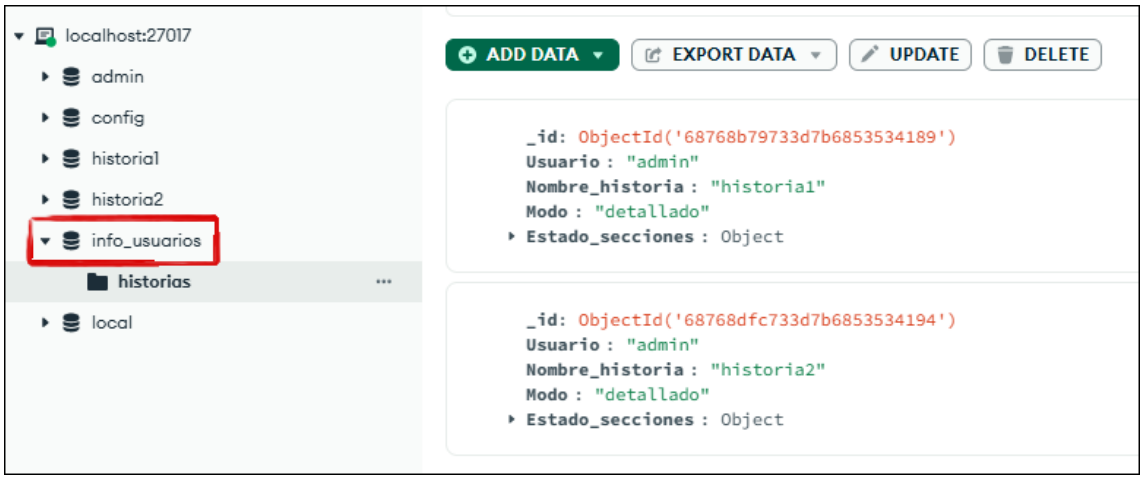


Figura 3.5: Almacenamiento en la base de datos: *info_ usuarios*

Dentro de la base de datos de cada historia se guarda las colecciones de cada una de las secciones y además, dos colecciones especiales como son: `info` y `contexto`. La colección `info` almacena la información de usuario, modo y estado de las secciones de la historia. Mientras que `contexto` guarda el contenido de cada una de las secciones en formato de texto plano y para facilitar su uso para construir el mensaje que se le pasa al modelo.

Figura 3.6: Almacenamiento en la base de datos para el *modo detallado*: *contexto*

Figura 3.7: Almacenamiento en la base de datos para el *modo detallado*: *sección*

La base de datos del *modo rápido* es más sencilla al tener que guardar unicamente los capitulos en una colección llamada `capitulos`. Mientras que el *modo detallado* requiere un número mayor de colecciones haciéndolo más complejo. Añadir imagen modo rapido La gerarquía en MongoDB: la base de datos ->colecciones ->documentos

3.2.6. Gestión de prompts

Uno de los aspectos clave en el proyecto es la creación de los *prompt de sistema* para que el modelo pueda seguir una seria de pautas y actue de determinada manera. En estos *prompts* se explicitan las reglas e instrucciones que el modelo debe seguir a la hora de generar una respuesta. Es importante comentar que aunque se concreten unas reglas y formato deseado, en ocasiones no se ajusta a las reglas preestablecidas. A continuación se explican los *prompts* más importantes:

- PROMPT_INICIAL: Se trara del `prompt` principal para el modo detallado donde se especifica las reglas que tiene que seguir. A medida que se realizaban pruebas y se añadian nuevas funcionalidades este `prompt` también se iba actualizando. La parte más importnte es la especificación de las secciones: *trama e hilo simbólico, descripción del mundo, personajes principales, descripción de escenarios* y *estructura de capitulos*. Cada una de las secciones contiene unas reglas determinadas y se proporciona un ejemplo de salida para lograr una mayor preción en la respuesta. Tanto las secciones de personajes como escenarios, se aporta una plantilla con distintos campos para así facilitar la creción y guardado en la base de datos.

- PROMPT_JSON: Se invoca tras la generación del contenido narrativo del LLM para transformar el texto en formato JSON válido, listo para insertarlo en la base de datos de MongoDB. Se describe una serie de reglas para poder conseguir el formato JSON deseado donde el nombre de la sección se guarda en un campo *Sección* o secciones como personajes y escenarios se contruyen a partir de listas. Con este `prompt` evitamos errores y permite controlar los

pequeños cambios de formato que pueda producir el modelo al generar la respuesta.

- PROMPT_ESCRITURA_RAPIDA: Este es el `prompt` utilizado para el modo rápido. Al no tener las secciones y directamente escribir los capítulos, se necesitaban otras normas e instrucciones a la hora de construir la respuesta. Se especifica la importancia de utilizar los datos introducidos en la encuesta inicial para crear la historia.
- PROMPT_CAPITULOS: Tras desarrollar todas las secciones y pasar al momento de la escritura, se ha decidido crear un nuevo *promt*, más detallado, de como escribir los capítulos. De esta forma lo separamos del `prompt` donde se especifican las secciones logrando así, una mayor precisión. De igual forma, sirve para gestionar mejor la limitación de *tokens* al separar el `prompt` y no usar uno de mayor longitud.

3.3. Interfaz de usuario

Esta sección describe la aplicación desde el punto de vista del usuario final. El objetivo es que cualquier lector, tenga o no base técnica, entienda cómo se crea una novela de principio a fin usando la herramienta, y por qué la interfaz está organizada del modo en que lo está.

3.3.1. Pantallas y funcionalidades

En este apartado se describen las vistas principales de la aplicación, explicando las funcionalidades que puede realizar el usuario en cada una de ellas:

3.3.1.1. Inicio de sesión

Pantalla de inicio donde se pide al usuario iniciar sesión a través de su usuario y contraseña para poder acceder a la aplicación. En caso de no estar registrado se puede crear una nueva cuenta mediante el botón `Register`.

3.3.1.2. Registro de usuario

En esta vista se solicitan los datos necesarios para crear una nueva cuenta: Username, E-mail, Password y Repeat Password. Una vez validados los datos, el sistema registra al usuario y lo dirige automáticamente al menú principal.

3.3.1.3. Página principal

Esta pantalla muestra el menú principal estructurado en dos bloques principales:

- *Cargar y eliminar:* Cada usuario cuenta con el área *Mis historias* donde se visualiza su historial de novelas junto con el progreso alcanzado. Para cada una de las novelas, el usuario puede retomarla desde el punto en que la dejó o eliminarla si no desea conservarla.

- *Empezar nueva historia:* Existe la opción de iniciar una nueva historia ya sea en *modo detallado* o *modo rápido*. En ambos casos se pide al usuario introcudir un nombre para guardar la historia en el historial.

3.3.1.4. Empezar historia detallada

Se trata de la pantalla principal del proyecto, dividida en tres áreas:

- *Lienzo narrativo:* Área principal donde se visualiza el contenido de la historia generado por el modelo. Al acceder por primera vez, se muestra un bloque de escritura en el que el usuario puede introducir una idea inicial o la temática de la novela. Desde este espacio también es posible emplear la opción *Reescribir* para volver a generar una sección completa, o *Modificar* para realizar cambios específicos en fragmentos concretos del texto.
- *Panel de secciones:* Zona de interacción del usuario dividida en dos partes: El primero, *Ver*, permite en cualquier momento visualizar la sección deseada y, mediante la acción *Editar*, realizar modificaciones de forma manual. El segundo, *Generar*, posibilita seleccionar la sección que se desea crear, con la opción de especificar alguna características concretas para orientar la generación.
- *Instrucciones:* Espacio donde se presenta una breve guía para facilitar el uso de la aplicación

3.3.1.5. Empezar historia rápida

En este modo, la creación de la historia se desarrolla en dos fases principales:

- *Cuestionario inicial:* Al acceder a esta pantalla, el usuario se encuentra con un breve formulario donde puede introducir datos opcionales como la temática, personajes, ambientación o estilo narrativo. Una vez enviado el cuestionario, el sistema el primer capítulo.
- *Visualización y edición de capítulos:* Tras la generación, el capítulo se muestra en el área principal. Para cada capítulo el usuario cuenta con las acciones de *Reescribir, Modificar* y *Continuar* con el siguiente capítulo.

3.3.2. Arquitectura de la interfaz

Durante este apartado se explica el funcionamiento de `Flet` a la hora de crear y navegar por las distintas pantallas de la aplicacion.

3.3.2.1. Modelo de UI en Flet

Las interfaces de usuario en `Flet` se construyen a partir de `controles` (también llamados `widgets`), que pueden organizarse en forma de árbol, donde el nodo raíz es `Page`. Cada pantalla de la aplicación se modela como una `View` asociada a una `ruta`, y la navegación consiste en cambiar la ruta activa para que `Flet` renderice la vista correspondiente. En este proyecto se utiliza la librería `flet_route` para gestionar el enrutamiento y la navegación.

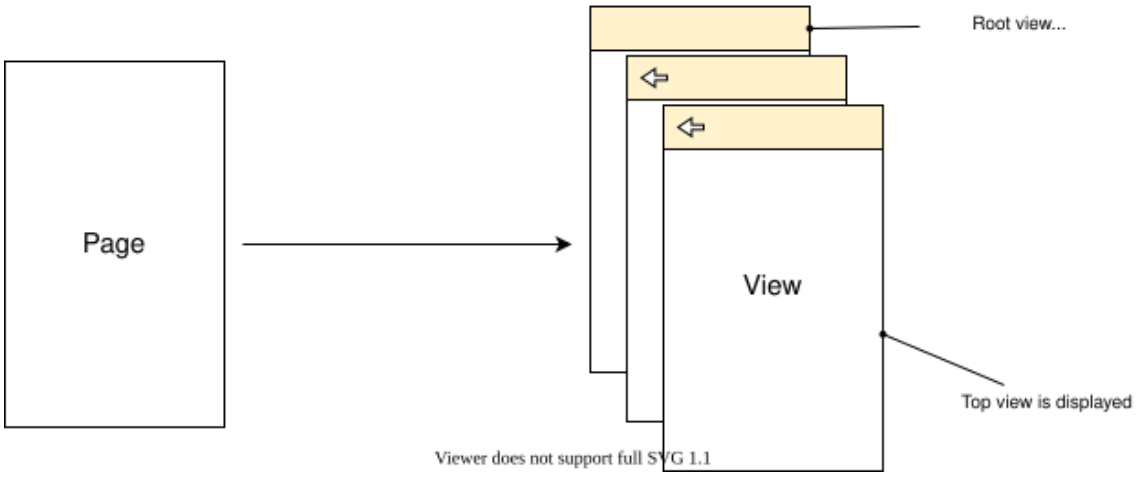
3.3.2.2. Navegacion con flet_route

La ruta de la página se encuentra a continuación del símbolo # en la URL, por ejemplo:

`https://localhost/#/nueva_hisotria`

Todas las rutas comienzan con /, por ejemplo `/home`, `/nueva_historia`. Por defecto Flet comienza en la ruta: `http://localhost:PORT/#/` y cada vez que el usuario navega entre páginas, Flet llama a `page.on_route_change`. En nuestro caso, al usar `flet_route`, se simplifica y únicamente usamos `page.go(nueva_historia)`.

Al construir varias vistas, *Page* deja de ser una única página y se convierte en un contenedor de vistas en capas una encima de otra como una pila:añadir imagen



Se organizan como una lista de vistas donde la última de la lista es la que se muestra actualmente. En el siguiente fragmento de código se muestra la lista y organización de las vistas en el proyecto:

Código 3.2: Llamada al cliente de Groq

```
import flet as ft
from flet_route import Routing, path

#Importamos las pantallas
from UI.Home import Home
from UI.Login import Login
from UI.Register import Register
from UI.NewHistory import NuevaHistoria
from UI.NewHistoryFast import NuevaHistoriaRapida
from UI.LoadHistory import CargarHistoria

def main(page:ft.Page):
    page.title="Creador de historias"
    app_routes = [
        path(url="/", clear=True, view=Login().view),
```

```
            path(url="/register", clear=True, view=Register().
                view),
            path(url="/home", clear=True, view=Home().view),
            path(url="/nueva_historia", clear=True, view=
                NuevaHistoria().view),
            path(url="/nueva_historia_rapida", clear=True, view=
                NuevaHistoriaRapida().view),
            path(url="/cargar_historia", clear=True, view=
                CargarHistoria().view)
        ]

        Routing(page=page, app_routes=app_routes)
        page.go(page.route)

    ft.app(target=main)
```

Capítulo 4

Desarrollo y evaluación de relatos generados

Capítulo 5

Conclusiones y Trabajo Futuro

Conclusiones del trabajo y líneas de trabajo futuro.

Antes de la entrega de actas de cada convocatoria, en el plazo que se indica en el calendario de los trabajos de fin de grado, el estudiante entregará en el Campus Virtual la versión final de la memoria en PDF.

Bibliografía

[1] Summarizer, https://www.summarizer.org/.

[2] Wrizzle, https://www.wrizzle.ai/es.

[3] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes and Yejin Choi, (2020). TThe curious case of Neural Text *De*generation. Published as a conference paper at ICLR 2020, arXiv:1904.09751v2 [cs.CL] 14 Feb 2020, https://doi.org/10.48550/arXiv.1904.09751.

[4] A.M. Turing, (1956). I.-Computing machinery adn intelligence. *Mind* **LIX** n.236, pp.433-460, 1950. https://doi.org/10.1093/mind/LIX.236.433.

[5] J. Moor, (2006). The Dartmouth College Artificial Intelligence Conference: The Next Fifty Years. AI Magazine. 27. 87-91.

[6] A. Newell and H. Simon, (1956). "The logic theory machine–A complex information processing system,ÏRE Transactions on Information Theory, vol. **2**, no. 3, pp. 61-79, September 1956, https://doi.org/10.1109/TIT.1956.1056797.

[7] A. Newell, J. C. Shaw and H. A. Simon, (1959). Report on a general problem-solving program. Proceedings of the International Conference on Information Processing. pp. 256-264, http://findingaids.library.cmu.edu/repositories/2/archival_objects/22561.

[8] Rai, Dilli Hang, (2024). Artificial Intelligence Through Time: A Comprehensive Historical Review. https://doi.org/10.13140/RG.2.2.22835.03364.

[9] Lili Yao, Nanyun Peng, Ralph, Weischedel, Kevin Knight, Dongyan Zhao and Rui Yan, (2019). Plan-and-Write: Towards Better Automatic Storytelling. https://doi.org/10.1609/aaai.v33i01.33017378.

[10] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, (2024). A Comprehensive Overview of Large Language Models. https://arxiv.org/abs/2307.06435.

[11] R. Goodwin, (2018). 1 the road. Ed. Jean Boîte Editions, 2018, ISBN-13: 978-2365680271.

[12] Racter, (2018). The Policeman's Beard is Half Constructed. Ed. Grand Central Pub, 1984, ISBN-13: 978-0446380515.

[13] D. Guisado Morcillo and ChatGPT, (2023). Iris. Ed. David Guisado Morcillo, 2023, IBSN-13: 978-8409477142.

[14] Transformers. https://huggingface.co/docs/transformers/main/es/.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, (2023). Attention is all you need. arXiv preprint arXiv:1706.03762. https://doi.org/10.48550/arXiv.1706.03762.

[16] Groq, https://groq.com/.

[17] Flet, https://flet.dev/.

[18] MongoDB, https://www.mongodb.com/.

[19] Llama3.3, https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_3/.

Apéndice A

Título del Apéndice A

Los apéndices son secciones al final del documento en las que se agrega texto con el objetivo de ampliar los contenidos del documento principal.

Apéndice B

Título del Apéndice B

Se pueden añadir los apéndices que se consideren oportunos.

Este texto se puede encontrar en el fichero Cascaras/fin.tex. Si deseas eliminarlo, basta con comentar la línea correspondiente al final del fichero TFGTeXiS.tex.

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes