
Generación de historias por secciones utilizando
Large Language Model
Story generation by sections 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
Story generation by sections 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

1 de septiembre de 2025

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 evaluación 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, historIAs, escritura asistida, IA, Llama, Transformer, token.

Abstract

Story generation by sections using Large Language Models

The improvement of *Large Language Models* (LLMs) has enabled their use in creative domains such as story and narrative generation. This work presents an application called **historIAs** for the creation of stories assisted by LLMs, through a workflow structured into sections: *plot and symbolic thread*, *world description*, *main characters*, *setting description*, *story structure* and finally, the writing of chapters. This system allows the user to personalize and edit the story both before and during the writing process. All information is stored in a database, enabling the user to resume the story at any moment.

To assess the quality of the generated stories, evaluations from different users were conducted, aiming to contrast the narrative production of the model with human writing and to identify strengths and limitations. Finally, the work discusses the role of LLM-based story generation as a creative aid for writers in their storytelling process.

Keywords

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

Índice

1. Introducción	1
1.1. Motivación	2
1.2. Objetivos	2
1.3. Estructura de la memoria	3
2. Estado de la Cuestión. Los LLMs en la generación de historias	5
2.1. Orígenes y fundamentos generales de la Inteligencia Artificial	5
2.2. Fundamentos técnicos de los LLMs	6
2.3. LLMs en la generación de historias	10
3. Descripción del Trabajo	13
3.1. Recursos para la creación de historÍAs: Groq, Flet y MongoDB	14
3.2. Diseño de la aplicación historÍAs	16
3.2.1. Modos de generación de historias: rápido y detallado	16
3.2.2. Diagrama de flujo y componentes de la aplicación historÍAs	17
3.2.3. Interacción del usuario con el modelo LLM	19
3.2.4. Almacenamiento en la base de datos	22
3.3. Diseño y gestión de <i>prompts</i>	24
3.4. Interfaz de usuario	29
3.4.1. Pantallas y funcionalidades	29
3.4.2. Arquitectura de la interfaz	32
4. Desarrollo y evaluación de historias generadas	35
4.1. Construcción del mensaje y gestión de la petición	36
4.2. Ejemplo de una historia en el <i>modo detallado</i>	38
4.3. Ejemplo de una historia en el <i>modo rápido</i>	44
4.4. Evaluación de las historias	46
5. Conclusiones y Trabajo Futuro	49
A. Modelos Groq	53

B. Ejemplo de una historia y encuestas de los usuarios	55
B.1. Ejemplo de historia en modo detallado	55
B.2. Encuestas realizadas por los usuarios	59
C. Introduction	69
D. Conclusion and Future Work	73
Bibliografía	77

Índice de figuras

2.1. Número de artículos científicos relacionados con los LLMs en los últimos años	7
2.2. Precisión según la temperatura	9
2.3. Línea del tiempo de lanzamientos de LLMs	10
3.1. Esquema general de componentes para la aplicación historIAs	13
3.2. Diagrama de flujo <i>Flujo principal</i>	18
3.3. Diagrama de componentes de historIAs	19
3.4. Diagrama de secuencia	20
3.5. Ejemplo de almacenamiento en la base de datos: <i>info_usuarios</i> . . .	22
3.6. Ejemplo almacenamiento en la base de datos para el <i>modo detallado: sección</i>	23
3.7. Ejemplo de almacenamiento en la base de datos para el <i>modo detallado: contexto</i>	23
3.8. Ejemplo de almacenamiento en la base de datos para el <i>modo rápido: capítulos</i>	24
3.9. Pantalla de inicio de sesión	30
3.10. Pantalla de registro de usuario	30
3.11. Pantalla principal	31
3.12. Pantalla de creación de historia en modo detallado	32
3.13. Pantalla de creación de historia en modo rápido	33
3.14. Organización de las pantallas en Flet	34
4.1. Historia de ejemplo: <i>Trama e hilo simbólico</i>	38
4.2. Historia de ejemplo: Modificar - <i>Trama e hilo simbólico</i>	39
4.3. Historia de ejemplo: Modificación - <i>Trama e hilo simbólico</i>	39
4.4. Historia de ejemplo: Características - <i>Personajes principales</i>	40
4.5. Historia de ejemplo: <i>Personajes principales</i>	40
4.6. Panel derecho	41
4.7. Historia de ejemplo: <i>Personajes principales</i> - Editar.	41
4.8. Historia de ejemplo: <i>Modo detallado</i> - Capítulo 1.	42
4.9. Historia de ejemplo: <i>Modo detallado</i> - Capítulo 4.	43
4.10. Historia de ejemplo: <i>Capítulo 4</i>	43

4.11. Historia de ejemplo: <i>Cuestionario inicial</i>	44
4.12. Historia de ejemplo: <i>Modo rápido</i> - Capítulo 1.	45
4.13. Historia de ejemplo: <i>Modo rápido</i> - Capítulo 4.	45
5.1. Comparación de arcos narrativos entre humanos y modelos de lenguaje	50
D.1. Comparison of narrative arcs between humans and language models .	74

Índice de tablas

3.1. Límites de uso de los modelos empleados (a fecha de realización). . .	15
A.1. Tabla LLM Groq	53

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* ¹) 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.

¹https://en.wikipedia.org/wiki/Large_language_model

1.1. Motivación

Actualmente existen múltiples páginas web y aplicaciones que permiten generar historias de manera automática mediante inteligencia artificial (Summarizer [24], Wrizzle [29]) 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 reiniciar 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 este modo, se busca ayudar y guiar a los usuarios en la creación de sus propias historias, ofreciendo un alto nivel de detalle.

En el análisis de una obra literaria, ya sea narración, poesía, etc., es habitual considerar, entre otros, los elementos que la configuran: la trama, la simbología, los personajes principales y los atributos que los definen, los escenarios, el desenlace y la estructura narrativa (ver Sección 2.2 para un mayor desarrollo). Tomando estos elementos como referencia, en este trabajo, la creación de historias se organiza en secciones clave: la *trama e hilo simbólico*, que marca el rumbo general de la obra, los *personajes principales*, donde se especifican 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 característica 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:

- Usar un LLM que permita la generación de historias, asegurando una buena gestión de *tokens*, entendidos como las pequeñas unidades en que los modelos dividen el texto para procesarlo, optimizando el proceso de modo que no se supere 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 (MongoDB²) 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 limitaciones 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 inteligencia artificial 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 apartado 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**.

²<https://www.mongodb.com/>

- **Capítulo 4. Desarrollo y evaluación de historias generadas:** En este capítulo se muestra un ejemplo de la creación de una historia en el *modo detallado* y su posterior evaluación mediante unas encuestas.
- **Capítulo 5. Conclusiones y trabajo futuro:** Presenta una reflexión final acerca de la generación narrativa por parte de los LLMs, las mejoras introducidas mediante la creación por secciones y finalmente se exponen una serie de líneas para trabajos futuros.
- **Apéndice A y B:** En el apéndice A se muestra la tabla con todos los LLMs disponibles en **Groq**. En el apéndice B, Se proporciona la historia completa generada en el ejemplo de capítulo 4, junto con un par de ejemplos de evaluaciones resueltas.
- **Apéndice C y D:** Estos dos apéndices corresponden a la parte en inglés de la introducción y la conclusión y trabajo futuro.
- **Bibliografía:** Se muestra las referencias utilizadas en el proyecto.

Estado de la Cuestión. Los LLMs en la generación de historias

Los LLMs (*Large Language Models*) han transformado radicalmente el procesamiento del lenguaje y la generación de texto. A lo largo de este capítulo se presenta un análisis 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

Se puede definir la Inteligencia Artificial como la “disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico”¹

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 [26], 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” (IA por sus siglas en español) [17]. 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) [19] y el General Problem Solver (1959) [20]. Sin

¹R. Asale and Rae. Diccionario de la lengua española <https://dle.rae.es/>.

embargo, debido a las limitaciones computacionales y la falta de datos provocaron un estancamiento en el desarrollo.

A partir de las décadas de 1980 y 1990, el desarrollo de algoritmos de aprendizaje automático (*machine learning*) permitió que las máquinas empezaran a identificar patrones y a mejorar su rendimiento a partir de datos, sin requerir reglas programadas explícitamente. Sin embargo, no fue hasta los años 2000's cuando el *machine learning*² pudo ser implementado de manera práctica, en parte gracias a la disponibilidad de grandes volúmenes de datos y la aparición de hardware económico con un gran poder de cálculo como fueron las primeras *Unidades de Procesamiento Gráfico* (GPU³). Gracias a estos grandes avances tecnológicos, se pudieron construir las redes neuronales con cientos de capas de neuronas, dando nacimiento al termino *Deep Learning* (aprendizaje profundo). El *deep learning* permite que modelos computacionales, formados por capas de procesamiento, aprendan representaciones de los datos [13]. Utiliza nodos o “neuronas” interconectadas, organizado en capas, que permite aprender de sus errores y mejorar continuamente. Gracias a este sistema, se puede generar y entrenar modelos a partir de millones de datos.

En 2017 se presenta el artículo de investigación *Attention Is All You Need* donde se introduce la arquitectura *Transformers* [27]. 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.

2.2. Fundamentos técnicos de los LLMs

La gran mayoría de LLMs se basan en los *Transformers*⁴, que consiste en una arquitectura basada exclusivamente en mecanismos de atención, prescindiendo por completo de recurrencias y convoluciones [27]. Desde los inicios hasta ahora, los modelos han escalado en parámetros y capacidades. A finales de 2024, Meta presentó *Llama 3.3-70b*⁵, un modelo de 70 mil millones de parámetros, optimizado para tareas multilingües y generación creativa.

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

²https://en.wikipedia.org/wiki/Machine_learning

³https://en.wikipedia.org/wiki/Graphics_processing_unit

⁴<https://huggingface.co/docs/transformers/main/es/index>

⁵https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_3/

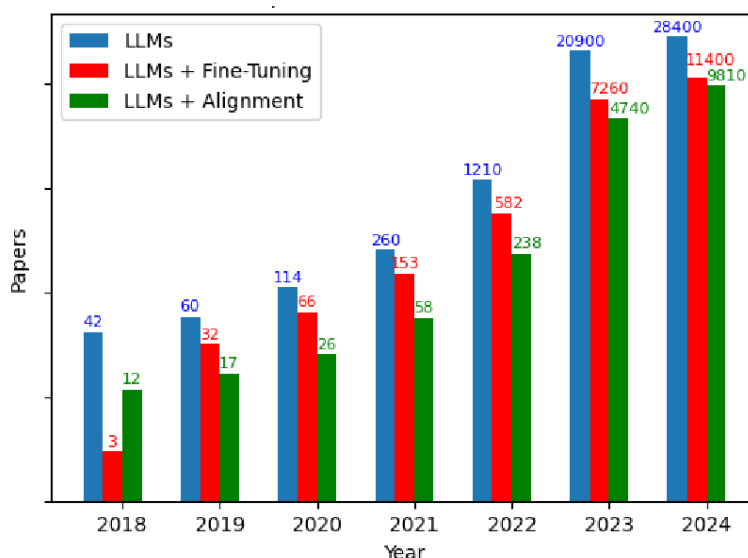


Figura 2.1: Número de artículos científicos relacionados con los LLMs en los últimos años (fuente del gráfico [18])

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).

La arquitectura del *Transformer* 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*. Este intercambio de información se realiza mediante el mecanismo de *atención*, que permite al modelo identificar qué partes de la entrada son más relevantes para generar cada *token* de salida.

La función de atención vincula una consulta y un conjunto clave-valor a una salida, donde la salida es la suma ponderada de los valores del conjunto clave-valor, 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*.

El mecanismo de atención constituye el núcleo de esta arquitectura, permitiendo al modelo enfocarse dinámicamente en las partes más relevantes de la secuencia de entrada al generar cada elemento de la salida. Este mecanismo compara cada *token* generado (la *query*) con todos los *tokens* de entrada (las *keys* y *values*) mediante un producto escalar, normalizado y transformado en una distribución de probabilidad. Los resultados se utilizan para ponderar los valores, generando una representación contextualizada. Además, el uso de múltiples cabezas de atención (*multi-head attention*) permite al modelo capturar simultáneamente distintos patrones y relaciones dentro de la secuencia, lo que mejora la comprensión contextual y la capacidad de generalización. Esta atención distribuida y paralelizable es lo que hace del Transformer una arquitectura tan poderosa y eficiente (para un mayor detalla consultar [27]).

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 serie 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, \dots, x_m , se busca generar los siguientes n *tokens* para obtener la sucesión $x_1, \dots, x_m, \dots, x_{m+n}$. Holtzman, Buys, et al. en su trabajo [11] consideran que los modelos calculan la probabilidad

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

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

Junto a la *tokenización*, también resulta fundamental la configuración de los parámetros de generación. Estos parámetros permiten variar el comportamiento del modelo durante la respuesta. Existen numerosos parámetros ajustables, pero en este proyecto se emplea únicamente los más básicos:

- **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 de *temperature* 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, como así muestra el trabajo de Renze y Guven (2024). En el gráfico de la Figura 2.2, para cada *temperature* se muestra la precisión de la respuesta, calculada como el cociente entre el número de respuestas correctas en diez intentos por cada problema [22]. Se observa que se mantiene estable hasta el valor 1 donde empieza a decaer (ver Figura 2.2).

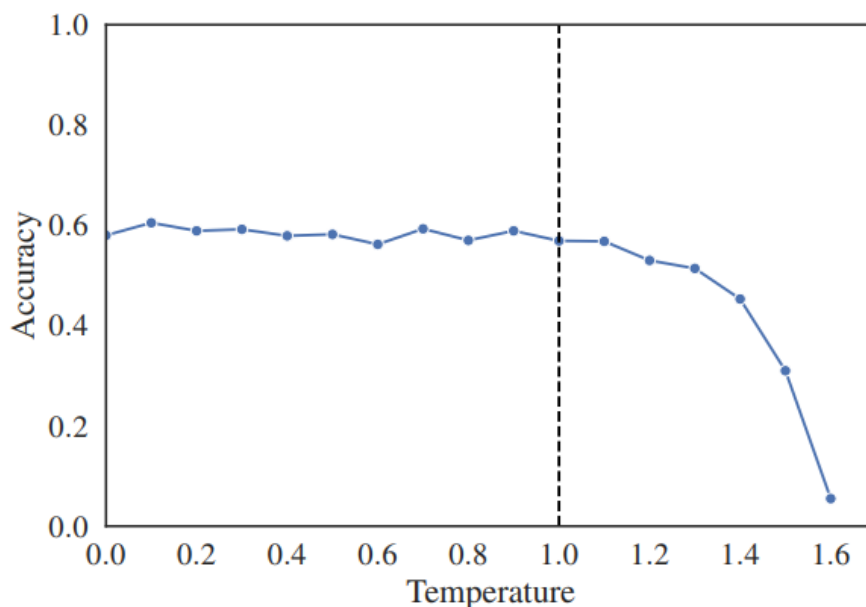


Figura 2.2: Precisión según la temperatura (de 0,0 a 1,6) para GPT-3.5 [22]).

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.

- **max_tokens**: Define el número máximo de *tokens* que puede devolver el modelo de lenguaje en una respuesta. Se ha establecido este máximo en 4.096 *tokens* lo cual permite una salida completa del modelo evitando cortes a mitad de la generación. Esta cantidad debe de convivir con el *prompt del sistema* (ver Sección 3.3 para más detalle) y el mensaje generado, por eso se debe de ajustar cuidadosamente el máximo de *tokens* por respuesta para no desbordar el límite del modelo.

Estos parámetros ofrecen un equilibrio entre coherencia y creatividad, y permiten adaptar la salida del modelo a los requisitos específicos de la narrativa en cada sección.

En la Figura 2.3 se presenta una línea del tiempo con los hitos más relevantes de los LLMs. Además de apreciarse el incremento de propuestas de LLMs pueden encontrarse los nombre de los LLMs más significativos que han surgido en los últimos años. Los recuadros azules representan modelos preentrenados (*pre-trained*), es decir, un modelo que se entrena con una gran variedad de datos y luego se adapta a tareas específicas [5]. Mientras que los naranjas corresponden a modelos afinados con instrucciones (*instruction-tuned*), modelos preentrenados ajustados para aprender a seguir instrucciones en lenguaje natural [28]. 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.

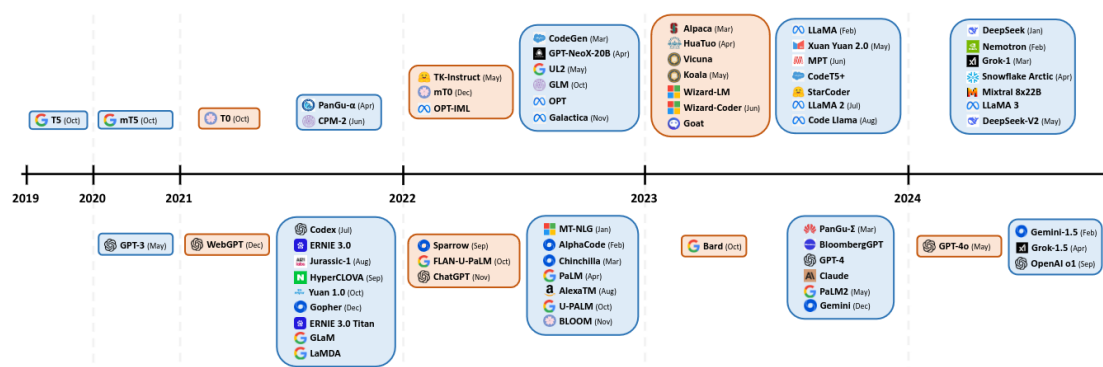


Figura 2.3: Línea del tiempo de lanzamientos de LLMs (fuente del gráfico [18])

2.3. LLMs en la 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. Esta mejora no solo permite tareas clásicas (resúmenes, reescritura, reformulación), sino que da paso a la generación narrativa. A partir de un texto o instrucción que se introduce al LLM (*prompt*), el modelo es capaz de construir una historia desarrollando escenas, personajes y mantener la coherencia a lo largo de la generación.

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) [21], 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) [8] 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) [10] 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 desafíos 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 [30]. 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 guía para producir el texto final. De esta forma se intenta que el modelo conserve el “plan” hacia donde transcurre 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, proponemos un enfoque inspirado en los sistemas generadores clásicos que sentaron las bases de la generación automática de historias. Ya el primer generador, Novel Writer (Klein, 1973) [12], partía de una descripción del mundo donde ocurre la acción y una completa caracterización de los personajes. Poco después, Tale-Spin (Meehan, 1977) [15] introdujo la planificación guiada por metas de los personajes y ofreció un modo interactivo en el que el usuario podía definir participantes y hechos del mundo. Más adelante, Author (Dehn, 1981) [4] incorporó explícitamente las metas del autor como criterio de construcción de la historia, asumiendo que toda narración se orienta por una trama concebida en la mente del autor, incluso cuando esas intenciones no se formulan de manera explícita. Para un análisis más detallado y un repaso exhaustivo de estos y otros sistemas generadores de historias, puede consultarse la Tesis de Eugenio Pablo Concepción Cuevas, donde se profundiza ampliamente en estos enfoques y su evolución [3].

En base a lo anterior el proyecto propone un sistema de generación de historias guiado por secciones. Se compone de las siguientes secciones: *trama e hilo simbólico*, *personajes principales*, *descripción de escenarios*, *descripción del mundo* y *estructura de capítulos*. 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**.

Descripción del Trabajo

Este capítulo presenta el desarrollo completo de **historIAs**, la aplicación propuesta para la creación de historias asistida por modelos de lenguaje. En primer lugar, se describen las herramientas y recursos empleados, así como los criterios de su elección: **Groq** para la inferencia del LLM, **Flet** para la interfaz de usuario y **MongoDB** para la base de datos. Además se introducen dos vistas de la arquitectura de la aplicación: (1) un diagrama de flujo que recorre el ciclo de creación de una historia y (2) un diagrama de componentes que muestra la relación e interacción entre las distintas partes del sistema. Antes de entrar en detalle, se presenta un esquema (ver Figura 3.1) de la distribución de los módulos.

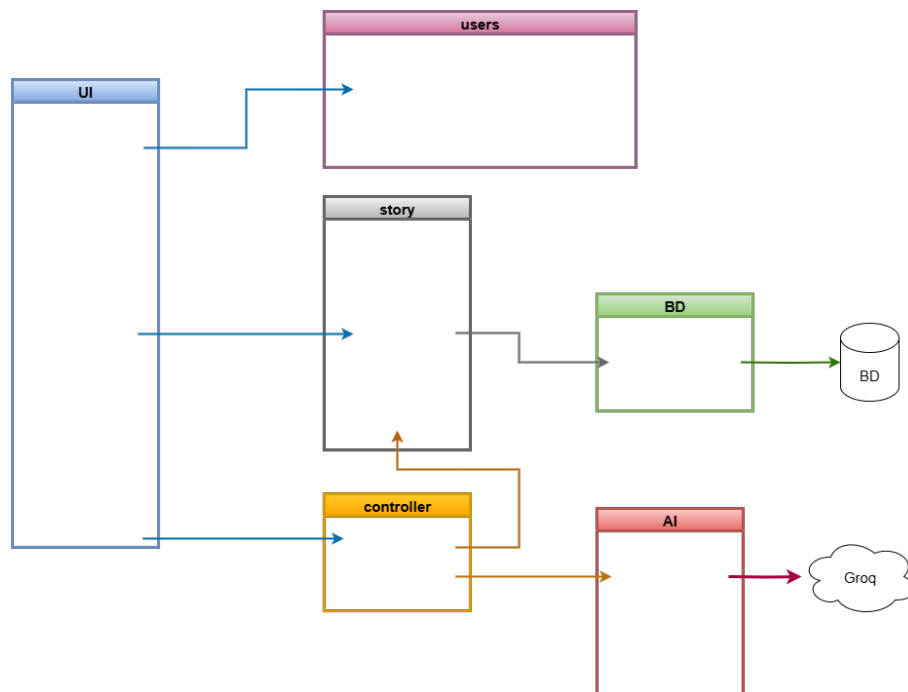


Figura 3.1: Esquema general de componentes para la aplicación **historIAs**.

A continuación, se documenta como se almacenan y para que se usan las res-

puestas en MongoDB. La persistencia de los datos es un aspecto fundamental para mantener el hilo de historia y permitir la edición. Se describen y muestran las distintas bases de datos y colecciones empleadas. Finalmente, se documenta la interfaz de usuario y su arquitectura en Flet, mostrando cómo la navegación y las vistas soportan un flujo de trabajo por secciones que favorece la edición y la personalización antes y durante la generación.

3.1. Recursos para la creación de historIAS: Groq, Flet y MongoDB

En este proyecto se han empleado tres herramientas principales que han sido determinantes para el desarrollo de la aplicación: Groq, Flet y MongoDB:

- **Groq, Inc.:** Es una empresa tecnológica desarrolladora de hardware específico para la inferencia de la inteligencia artificial [9]. Groq cuenta con el desarrollo de los procesadores **Large Processing Units (LPUs)**, que se caracterizan por ser altamente eficientes y estar optimizados para las tareas de inferencia en tiempo real. Las LPUs permiten ejecutar LLMs y otros modelos a mayores velocidades y hasta diez veces más eficiente energéticamente en comparación con las GPUs. En un principio se planteaba la idea de utilizar un LLM de forma local pero enseguida comprobamos que no teníamos el hardware necesario para utilizarlo. Tras utilizar plataformas como Google Colab [2] 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 alternativas en Microsoft Azure [1] o Amazon Web Service [23] pero finalmente optamos por Groq, un servidor menos conocido pero que ofrece una mayor velocidad y facilidad para el uso. Únicamente se precisa de una cuenta de usuario y una API Key para poder utilizar los modelos que ofrece en la version *Free*.

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 [6], 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 (request per minute)* y *RPD (request per day)* 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 (tokens per minute)* y *TPD (tokens per day)*, en especial el número de *tokens* por minuto por la demanda del “contexto” (ver la Sección 3.2.3 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 límite 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 (ver tabla completa de los modelos en el Apéndice A).

Tabla 3.1: Límites de uso de los modelos empleados (a fecha de realización).

Modelo	RPM	RPD	TPM	TPD
<code>deepseek-r1-distill-llama-70b</code>	30	1K	6K	100K
<code>llama-3.1-8b-instant</code>	30	14.4K	6K	500K
<code>llama3-70b-versatile</code>	30	1K	12K	100K
<code>llama3-8b-8192</code>	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**¹ (framework de código abierto desarrollado por *Google* para crear aplicaciones), lo que permite desarrollar aplicaciones web, de escritorio y móviles usando únicamente Python [7]. A diferencia de otras bibliotecas, **Flet** abstrae 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 usuario 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 [16]. 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 distintas 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. 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.

¹<https://flutter.dev/>

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 historIAs

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 historias: rápido y detallado

La aplicación 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 centra en 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*, *estructura de capítulos* y la propia escritura de la historia. Una vez se genera la sección, el usuario a través de botones interactúa 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 obtener resultados inmediatos, este modo presenta al usuario un breve cuestionario con distintos 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 sacrificando parte de la personalización en favor de la velocidad y simplicidad.

Con ambos modos se constituye un enfoque innovador respecto a otras aplicaciones ya existentes. Esto permite una gran flexibilidad, aportando opciones para los usuarios que deseen un control exhaustivo del proceso creativo y para los usuarios que precisen resultados más rápidos. Internamente, cada modo emplea un conjunto de **prompts** específicos (ver Sección 3.3) adaptados a los requisitos del modo.

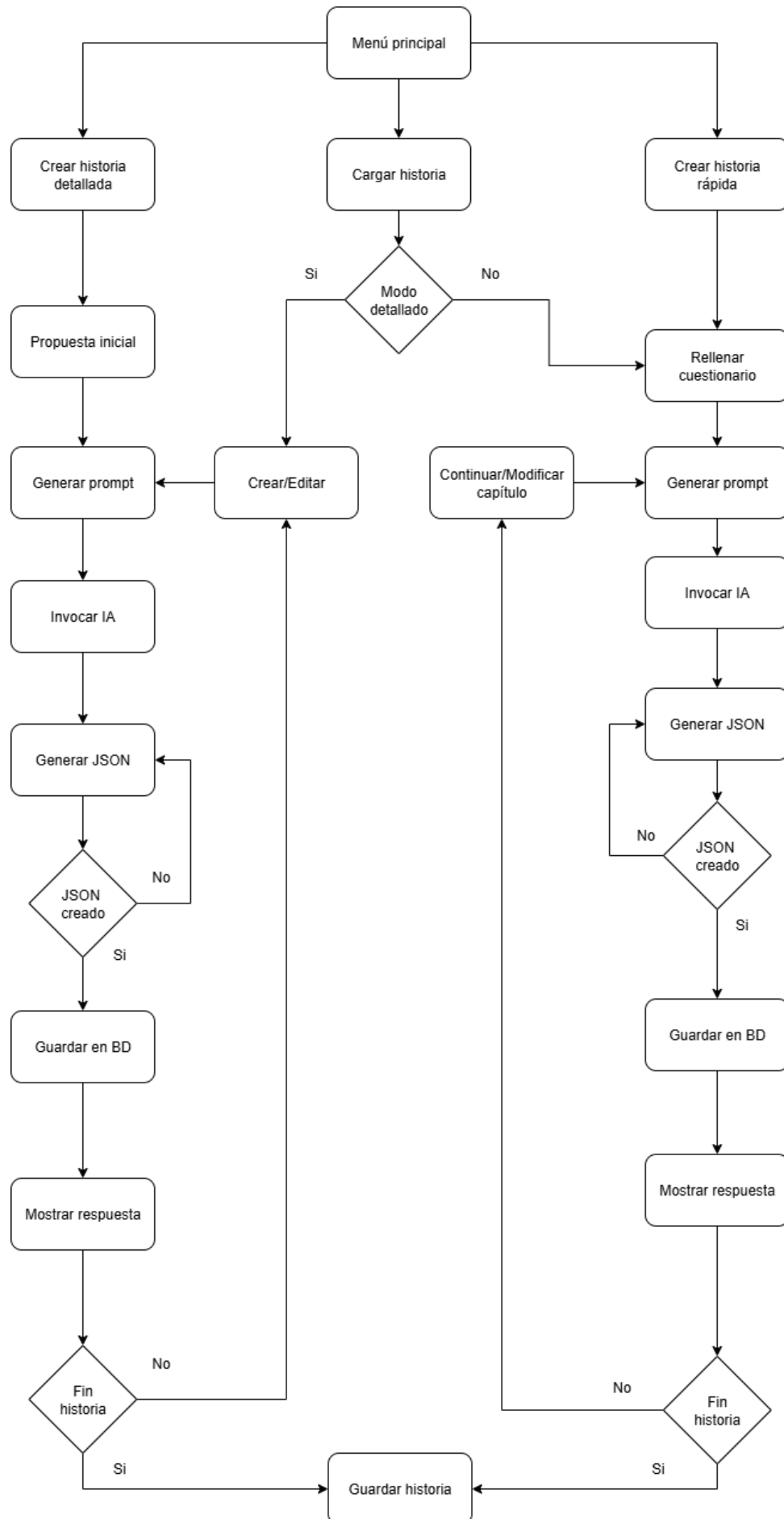
3.2.2. Diagrama de flujo y componentes de la aplicación historIAs

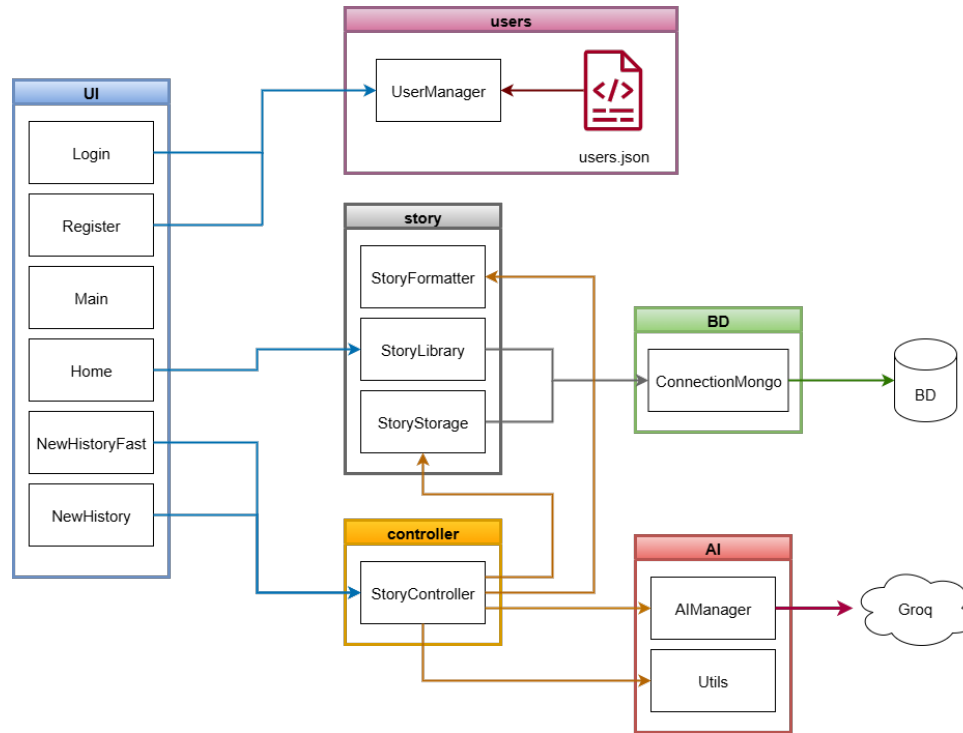
Para poder abordar con más detalle el diseño de la aplicación **historIAs**, haremos uso de distintos diagramas. Se muestra el diagrama de flujo para crear una historia (ver Figura 3.2) y se detalla su funcionamiento.

- *Flujo principal*: 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 Sección 3.2.1 para más detalles sobre los modos de creación de historias) manteniendo varias iniciadas al mismo tiempo.

A continuación se muestra el diagrama de componentes (ver Figura 3.3) que refleja la arquitectura de la aplicación. Esta dividida en distintos paquetes:

- *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 (ver Sección 3.4).
- *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 historias. **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 el usuario y **StoryStorage** crea 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 ocupa de llevar la lógica del procesamiento del mensaje, detectar la sección en 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 la clase **AIManager** encargada 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** del sistema que se usarán para generar las respuestas.

Figura 3.2: Diagrama de flujo *Flujo principal*

Figura 3.3: Diagrama de componentes de **historIAs**

3.2.3. Interacción del usuario 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` [14], 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** (generar sección o realizar modificación), 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 realizar 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 cambia es la manera en la que se crea el mensaje.

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. Cuando generamos una nueva sección o modificamos, necesitamos crear el contexto que necesita el LLM para generar la nueva respuesta. Para ello, en el *modo detallado*, se llama a la función `buscar_dependencias_bd` encargada de recuperar de la base de datos la información necesaria de las secciones anteriores y así construir el contexto. En el

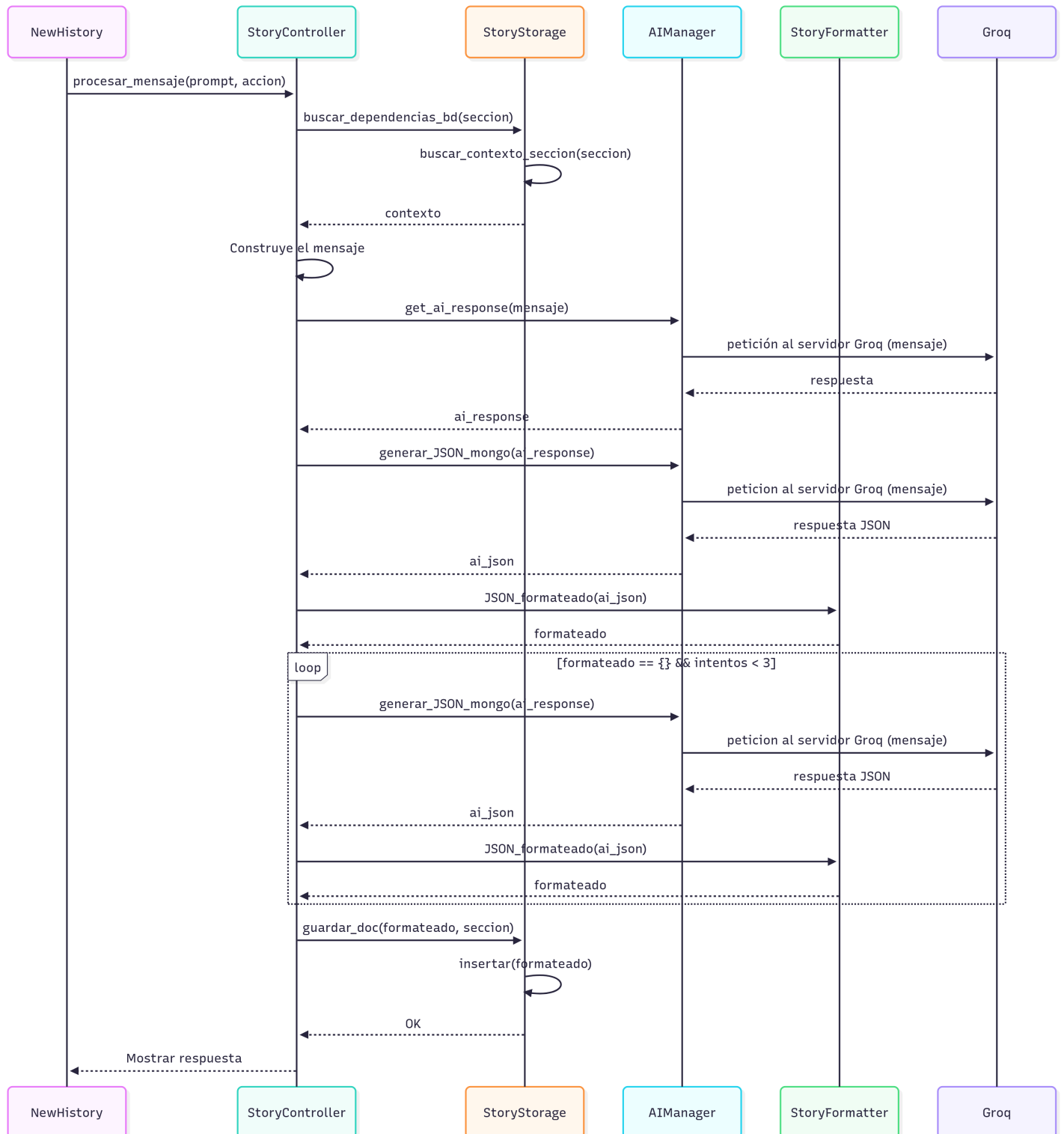


Figura 3.4: Diagrama de secuencia

caso del *modo rápido*, el contexto se crea a partir del cuestionario inicial y de los capítulos anteriores.

A continuación, se construye el mensaje de entrada que se envía al modelo. Dicho mensaje incluye el contexto generado previamente, el *input* del usuario (ya sea una propuesta inicial, una modificación o características especiales para la nueva sección) y la sección a crear. Además, se añade un *prompt del sistema* (ver Sección 3.3), que actúa como instrucción base para guiar el comportamiento del modelo y asegurar que las respuestas se ajusten al formato y estilo requeridos. En la llamada al modelo también se configuran parámetros clave, como *temperature*, que controla el grado de creatividad o aleatoriedad en la generación de texto, y *max_tokens*, que establece la longitud máxima de la respuesta (ver Sección 2.2 para más detalle). Se ha establecido el valor de *temperature* a 0,8 para proporcionar creatividad sin perder precisión y el *max_tokens* a 4096 para evitar que un capítulo se quede a mitad de la generación. De este modo, se puede equilibrar la originalidad frente a la coherencia, y limitar el tamaño de la salida para ajustarla a las necesidades. La respuesta se genera en formato *Markdown*, un lenguaje de marcado ligero que permite dar estructura y estilo al texto mediante símbolos sencillos (*#*, ***, ****). Antes de mostrarse al usuario, este contenido se procesa y se convierte en texto legible, eliminando la sintaxis propia de *Markdown* para presentar únicamente el resultado final.

Se requiere almacenar el contenido generado en MongoDB para que se pueda recuperar posteriormente. Para ello se convierte 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 esperadas (ver Sección 3.3), no siempre respetaba de forma exacta la misma estructura. Intentar depurar estas variaciones de forma manual 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 superar estas dificultades, se emplea de nuevo el LLM, al que se le indica que formatee la salida como un objeto JSON válido. De este modo, 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 guardar en la base de datos. Para solucionar estos errores, se implementa un bucle que permite al modelo generar un nuevo JSON cuando se produce un error. 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. Por tanto, el *prompt* se envía nuevamente al LLM para obtener una respuesta en el formato JSON adecuado. A continuación, la salida se convierte de texto plano a un objeto JSON mediante la función `json.loads()` y posteriormente se almacena en la base de datos de MongoDB.

3.2.4. Almacenamiento en la base de datos

Para este proyecto únicamente se precisa 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` (ver Figura 3.5) 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 en el momento de mostrar al usuario las historias que ha creado y pueda retomarla desde donde la dejó.

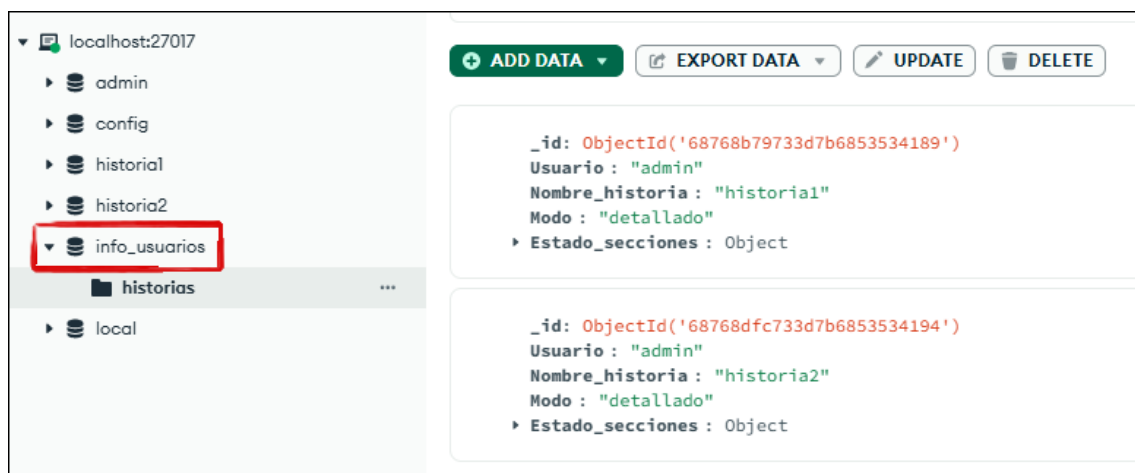


Figura 3.5: Ejemplo de almacenamiento en la base de datos: *info_usuarios*

Dentro de la base de datos de cada historia se guardan las colecciones de cada una de las secciones (ver Figura 3.6) 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` (ver Figura 3.7) guarda el contenido de cada una de las secciones en formato de texto plano para facilitar su uso en la construcción del mensaje que se le pasa al modelo.

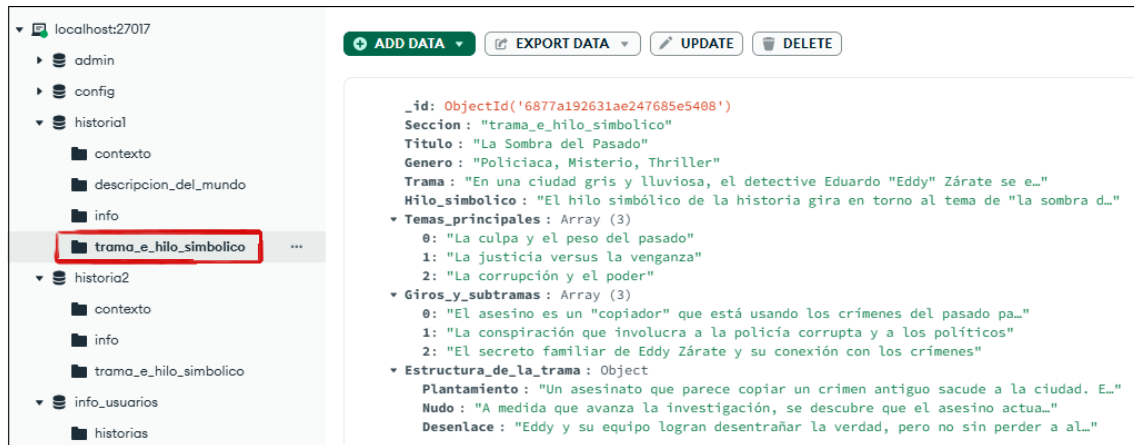


Figura 3.6: Ejemplo almacenamiento en la base de datos para el *modo detallado: sección*

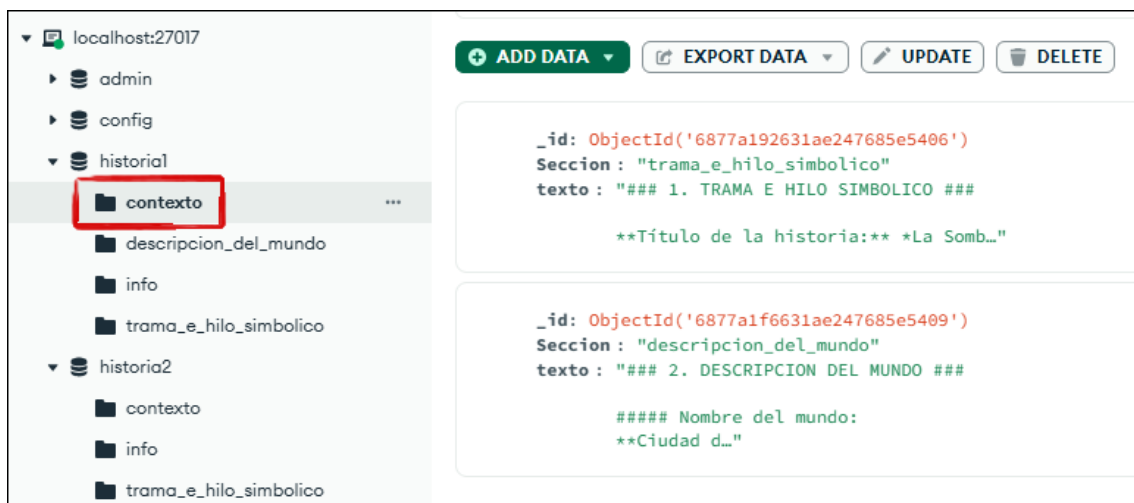


Figura 3.7: Ejemplo de almacenamiento en la base de datos para el *modo detallado: contexto*

La base de datos del *modo rápido* es más sencilla al tener que guardar únicamente los capítulos en una colección llamada *capítulos* (ver Figura 3.8). Mientras que el *modo detallado* requiere un número mayor de colecciones haciéndolo más complejo.



Figura 3.8: Ejemplo de almacenamiento en la base de datos para el *modo rápido: capítulos*

3.3. Diseño y gestión de *prompts*

Uno de los aspectos clave del proyecto es la creación de los *prompts de sistema*, que permiten guiar al modelo para que siga unas pautas concretas y actúe de una manera determinada. En estos *prompts* se formulan las reglas e instrucciones que el modelo debe cumplir a la hora de generar una respuesta. Para estructurar mejor el contenido, se emplea formato *Markdown*² y se destacan aspectos clave en mayúsculas. Conviene señalar que, aunque se definan unas reglas claras y un formato esperado, en ocasiones el modelo no se ajusta con exactitud a lo preestablecido.

A continuación se describen de forma detallada los *prompts* principales:

- **PROMPT_DETALLADO**: corresponde al *prompt* principal para el *modo detallado*. En él se define que el modelo actúe como experto en la creación de historias y se especifica cómo debe elaborar cada una de las secciones. También se fija explícitamente el idioma de salida en español, ya que en pruebas iniciales el modelo a veces devolvía resultados en inglés. La parte más relevante de este *prompt* son las reglas y ejemplos que se determinan para la construcción de las distintas secciones.

Por ejemplo, en la sección *Trama e hilo simbólico* se establecen pautas como las siguientes:

```
### REGLAS PARA LA TRAMA E HILO SIMBOLICO
- Primero se muestra el título y el género de la historia
- La trama debe estar bien desarrollada
  (estructura, temas principales, giros, subtramas, etc.)
- Se debe incluir un resumen completo de la historia
- Esquema a seguir:
  **Titulo**: (título de la historia)
  **Genero**: (género de la historia)
  Contenido con las SUBSECCIONES que se consideren necesarias
```

²<https://en.wikipedia.org/wiki/Markdown>

De esta forma, el modelo tiende a generar siempre una respuesta estructurada y detallada.

En el caso de la sección *Personajes principales*, el *prompt* indica que cada personaje se represente mediante una ficha detallada, con un máximo de cuatro personajes:

REGLAS PARA LA DESCRIPCION DE PERSONAJES PRINCIPALES

- Se mostrarán los personajes principales de la historia
- Habrá un máximo de 4 personajes principales
- La ficha de cada personaje debe seguir el siguiente esquema:
 - ****Nombre del personaje****
 - ****Rol**** (protagonista, antagonista, secundario, mentor, etc.)
 - ****Edad y sexo****
 - ****Personalidad**** (temperamento, valores, forma de actuar)
 - ****Descripción**** (altura, complexión, color de ojos/pelo, marcas distintivas, vestimenta habitual, etc.)
 - ****Historia**** (origen, familia, eventos clave de su vida)
 - ****Desarrollo personal**** (cómo evolucionará a lo largo de la historia, desafíos a los que se enfrentará, cambios en personalidad o creencias)
 - ****Relaciones**** (relaciones importantes con otros personajes: conflictos, aliados, enemigos, familiares, etc.)
 - ****Información adicional**** (otros datos relevantes)
- ****Si el usuario modifica un personaje****, se actualizará su ficha y se mostrará nuevamente ****cambiando solo lo solicitado**** junto con el resto de personajes.

El límite de cuatro personajes se estableció por cuestiones de eficiencia: en pruebas iniciales, el modelo generaba más de los esperados y se alcanzaba el límite de *tokens*. Además, se establece que en caso de modificación, el modelo debe actualizar únicamente lo solicitado y volver a mostrar la lista completa de personajes.

De manera análoga, para la sección *Descripción de escenarios* se define una ficha con campos adaptados a los lugares de la historia.

Finalmente, la sección *Estructura de capítulos* cuenta con sus propias reglas, en las que se fija una duración de cuatro capítulos y se proporciona un ejemplo de formato:

REGLAS PARA LA ESTRUCTURA DE LOS CAPITULOS

- DEBE TENER 4 CAPITULOS
- En cada capítulo se deben mostrar:
 - los personajes que aparecen
 - los escenarios implicados
 - una sinopsis breve de lo que ocurre
- La sinopsis debe ser un resumen detallado de unas 5 líneas
- Los personajes y escenarios deben aparecer

```

    CON EL MISMO NOMBRE que se registró en sus secciones
    (con nombre completo o compuesto)
## EJEMPLO:
- Seccion: estructura_de_capitulos
- Capítulos:
    - Nombre: (nombre del capítulo)
    - Sinopsis: (sinopsis del capítulo)
    - Personajes: (personajes que participan)
    - Escenarios: (escenarios implicados)
- Es fundamental que los personajes y escenarios
  que aparecen en los capítulos ya estén creados
  y registrados en sus secciones correspondientes

```

Con estas instrucciones, se consigue que el modelo mantenga un estilo uniforme y un nivel de detalle elevado en cada generación, reduciendo la variabilidad entre ejecuciones y aumentando la consistencia narrativa.

- *PROMPT_JSON*: este *prompt* se utiliza para controlar que la salida del modelo se produzca exclusivamente en formato JSON válido. Se emplea cuando la información generada debe almacenarse en la base de datos y, por tanto, requiere un formato estructurado y coherente. Se especifica que el modelo no debe incluir bloques de código (“`json`”) ni texto adicional fuera del propio objeto JSON. Además, se definen los nombres de las secciones en formato *snake_case* (*trama_e_hilo_simbolico*, *personajes_principales*), sin tildes ni caracteres especiales, y se remarcan los campos obligatorios.

En el caso de personajes o escenarios se incluyen en una lista de objetos con el campo “Personajes” o “Escenarios” respectivamente. A continuación se presenta un esquema de como se generan los escenarios en formato JSON:

```

"Escenario": [
{
  "Nombre": "Isla del Viento y la Luna",
  "Localización": "En medio de las Azules Aguas,
    a muchos días de navegación de Eldoria.",
  ...
},
{
  "Nombre": "Puerto de Brumas",
  "Localización": "En la costa occidental de Eldoria,
    rodeado de acantilados y neblina perpetua.",
  ...
},
...
]

```

Este esquema asegura que la información pueda ser procesada directamente

mediante `json.loads()` y almacenada en MongoDB sin necesidad de transformaciones adicionales.

- **PROMPT_CAPITULOS**: Tras desarrollar todas las secciones y pasar al momento de la escritura, se ha decidido crear un nuevo *prompt* más detallado para escribir los capítulos. De esta forma se separa la parte de las secciones de la generación de la historia, evitando así *prompt* extremadamente largos y ahorrando número de *tokens* innecesarios. Se especifica una serie de reglas a seguir para mantener un estilo narrativo y coherencia con la historia que se esta creando:

###REGLAS PARA LA ESCRITURA DE LOS CAPÍTULOS:

1. ****Estilo narrativo****:

- Mantén un tono coherente con el género y estilo general de la historia.
- Equilibra bien ****descripción****, ****acción**** y ****diálogo****, sin sobrecargar ninguna sección.

2. ****Coherencia con el universo****:

- Utiliza exclusivamente personajes, escenarios y elementos que hayan sido definidos previamente.
- No introduces personajes nuevos sin permiso del usuario.
- Todos los eventos deben respetar la lógica interna del universo y las relaciones previamente establecidas.

3. ****Construcción de personajes****:

- Asegúrate de que cada personaje actúe según su ****personalidad****, ****historia**** y ****relaciones**** establecidas.
- Sus decisiones deben sentirse creíbles y alineadas con su desarrollo personal.

4. ****Ritmo y tensión narrativa****:

- La estructura del capítulo debe tener un desarrollo claro: introducción, desarrollo, clímax o giro, y un cierre parcial o cliffhanger que motive a leer el siguiente.
- El ritmo debe ser dinámico pero natural, sin escenas innecesarias ni saltos abruptos.

5. ****Revisión al final****:

- Al concluir el capítulo, haz una breve pregunta al usuario sobre si desea mantenerlo como está o realizar cambios.
Si quiere cambios, procede como se indicó al inicio.

De esta forma se consigue que la historia siga lo creado previamente en las secciones, manteniendo la coherencia y el estilo.

- **PROMPT_ESCRITURA_RAPIDA**: este *prompt* se utiliza en el *modo rápido*. Indica al modelo que genere la historia completa a partir del cuestionario

inicial del usuario, mostrándola capítulo a capítulo. En las pruebas iniciales se observó que, aun fijando un máximo de cuatro capítulos, el LLM continuaba la narración, por ello se añadió una instrucción explícita que obliga a finalizar la historia al llegar al capítulo cuatro.

Objetivo

Generar una narración original, coherente y creativa que incluya un título atractivo y capítulos bien estructurados (SIEMPRE 4 CAPITULOS), siguiendo la información proporcionada y añadiendo detalles propios para enriquecer la trama. Es IMPRESCINDIBLE que la historia TERMINE en el CAPITULO 4.

Además, se establecen normas generales para estandarizar coherencia, tono y formato, incluido el estilo de los diálogos:

Reglas Generales

1. Lee atentamente los datos proporcionados (tema, género, ambientación, protagonista, tono, longitud).
2. Si algún dato no está presente, invéntalo de forma coherente con el resto.
3. Mantén la coherencia en nombres, lugares y hechos a lo largo de toda la narración.
4. Utiliza descripciones ricas y diálogos naturales para dar vida a los personajes y escenarios.
5. Los diálogos deben comenzar con el símbolo "-" y terminar con el símbolo "-" para cada intervención.
6. Ajusta el tono general (triste, alegre, misterioso, etc.) según lo indicado por el usuario.
7. Usa un lenguaje narrativo adaptado al género elegido.
8. No incluyas explicaciones, metas, ni comentarios fuera de la narración.
9. Si el usuario te pide una modificación o reescritura, muestra de nuevo ese capítulo

Con este *prompt* se obtiene una historia completa y coherente sin pasar por las secciones detalladas.

- **PROMPT_CAPITULO_A_JSON**: Este *prompt* se encarga únicamente de convertir el capítulo a formato JSON. Al igual que antes se ha decidido separar la parte de las secciones de la escritura para optimizar el número de *tokens*. Únicamente se pide que convierta el capítulo en un objeto JSON válido con las siguientes claves:

- "Seccion" → capítulo_x.
- "Titulo" → título breve y fiel.
- "Contenido" → TODO el texto del capítulo, íntegro y SIN modificar.

Es importante enfatizar en que el modelo no debe cambiar nada del texto original que no sea solicitado, ya que en versiones preliminares de la aplicación, en ocasiones modificaba parte de la sección. También es importante detallar que no se debe introducir texto adicional. De esta forma conseguimos convertir el capítulo a JSON prácticamente en la totalidad de los casos sin producir ningún error, listo para guardar en la base de datos.

3.4. Interfaz de usuario

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

3.4.1. 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 así como mostrando capturas de pantalla de la aplicación (Figuras de *login* 3.9, de *registro* 3.10, de *Mis historIAs* 3.11, del *modo detallado* 3.12 y del *modo rápido* 3.13).

3.4.1.1. Inicio de sesión

Pantalla de inicio donde se solicita 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**.

3.4.1.2. 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 el usuario y lo dirige automáticamente al menú principal.

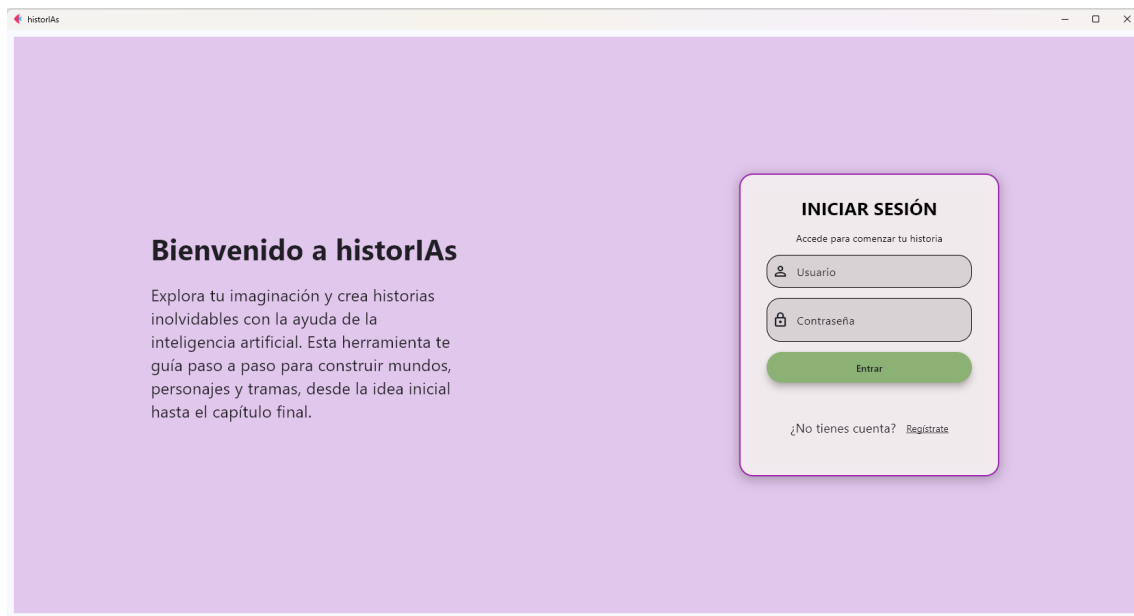


Figura 3.9: Pantalla de inicio de sesión

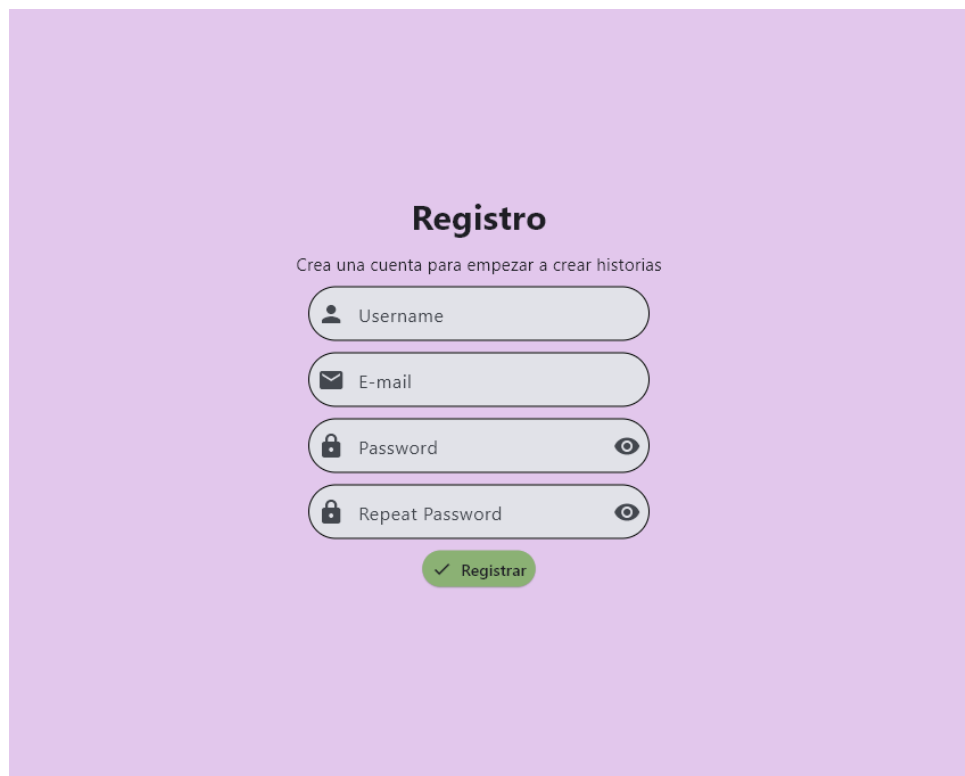


Figura 3.10: Pantalla de registro de usuario

3.4.1.3. Página principal

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

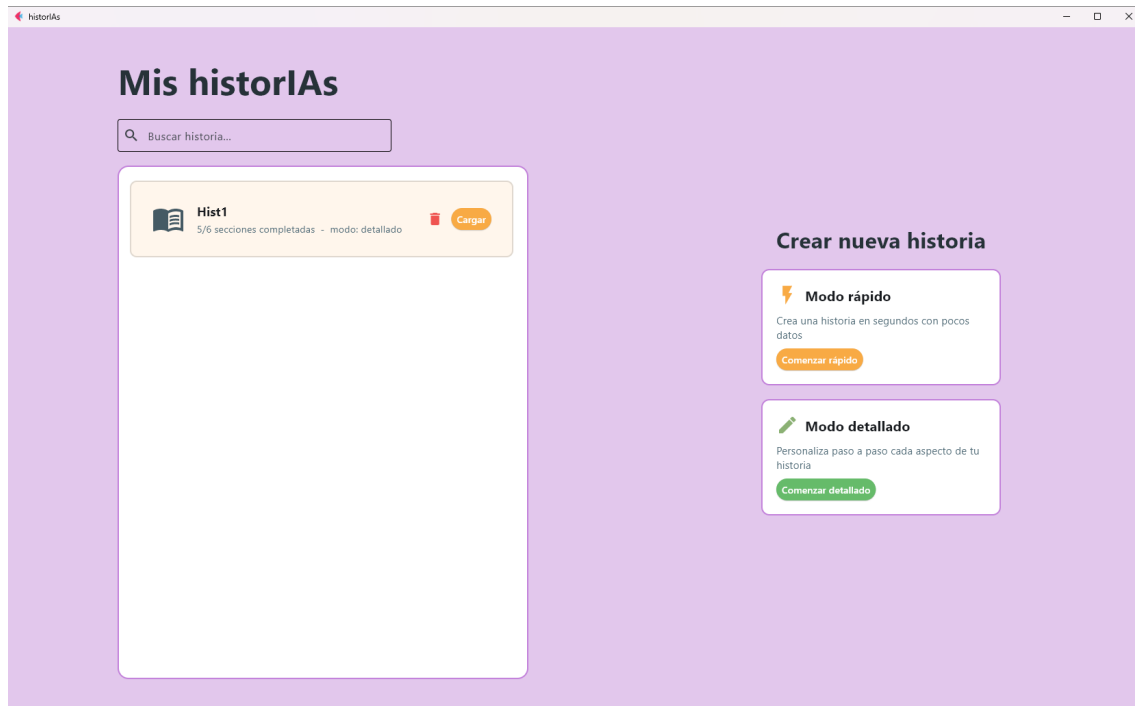


Figura 3.11: Pantalla principal

- *Cargar y eliminar*: Cada usuario cuenta con el área *Mis historIAS* donde se visualiza su historial de historias 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 solicita al usuario introducir un nombre para guardar la historia en el historial.

3.4.1.4. Comenzar 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 historia. 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: (1) *Ver*, permite en cualquier momento visualizar la sección deseada y, mediante la acción *Editar*, realizar modificaciones de forma manual. (2) *Generar*, posibilita seleccionar la sección que se desea crear, con la opción de especificar alguna características concretas para orientar la generación.

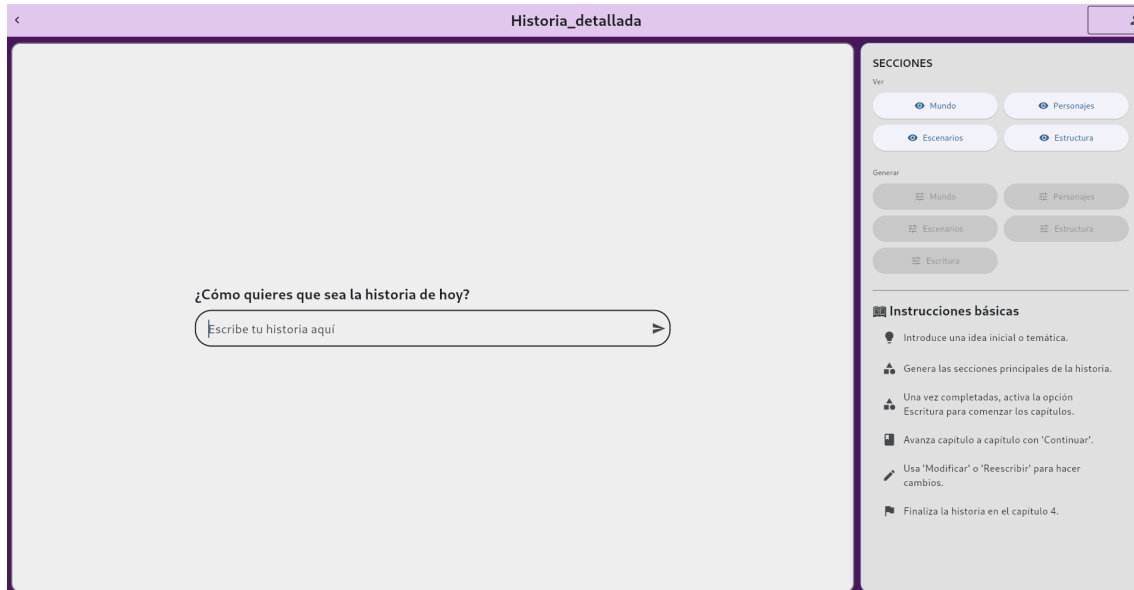


Figura 3.12: Pantalla de creación de historia en modo detallado

- *Instrucciones*: Espacio donde se presenta una breve guía para facilitar el uso de la aplicación

3.4.1.5. Comenzar 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 crea 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.

3.4.2. 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 aplicación.

3.4.2.1. 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


Historia_rapida


Crea tu historia en segundos


Completa solo lo imprescindible y déjate sorprender

Tema

 Género

 Ambientación

 Protagonista principal

 Tono

Longitud de la historia ▾


 ¡Generar historia!

Figura 3.13: Pantalla de creación de historia en modo rápido

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.

3.4.2.2. 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 (ver Figura 3.14).

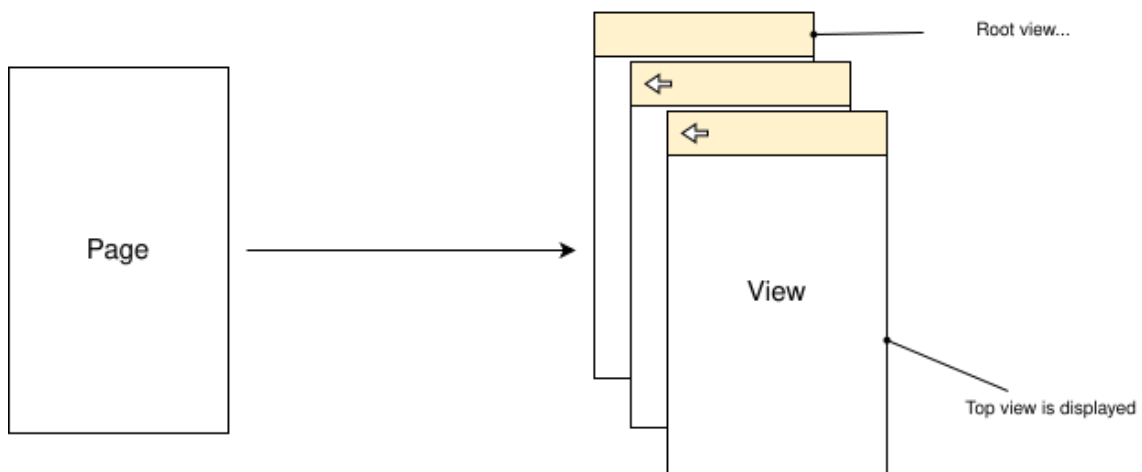


Figura 3.14: Organización de las pantallas en Flet

Se organizan como una lista de vistas donde la última de la lista es la que se muestra actualmente.

En resumen, la arquitectura de la interfaz combina la flexibilidad de **Flet** con la simplicidad de **flet_route**, lo que permite organizar la aplicación en vistas independientes y navegables de forma intuitiva. De esta manera, cada pantalla queda claramente separada y el usuario puede recorrer el flujo completo de creación de historias, desde el inicio de sesión hasta la edición y generación de la historia, sin percibir la complejidad técnica que hay detrás.

Capítulo 4

Desarrollo y evaluación de historias generadas

En este capítulo se expone un ejemplo de creación de una historia en el *modo detallado* y en *modo rápido* y se detalla todo el funcionamiento para generar una nueva historia. Se da un enfoque visual a través de capturas de pantalla para mostrar el resultado generado y se comentan aspectos importantes a tener en cuenta durante la creación de historias.

Código 4.1: Envío y procesamiento asíncrono de mensajes

```
def send_message(e, prompt_override=None, accion="crear",
seccion="trama_e_hilo_simbolico"):
    prompt_usuario = entrada.value.strip()
    prompt = prompt_override if prompt_override is not None
    else prompt_usuario

    if accion == "crear" and prompt_override is None:
        self.ultimo_prompt = prompt_usuario

    threading.Thread(target=procesar_en_segundo_plano, args=(
        prompt, accion, seccion,)).start()

def procesar_en_segundo_plano(prompt, accion, seccion):

    respuesta = controller.procesar_mensaje(prompt, self.
        ultimo_prompt, accion, seccion)

    page.run_thread(lambda: (mostrar_respuesta(respuesta,
        seccion), page.update()))
```

4.1. Construcción del mensaje y gestión de la petición

En esta sección se describe cómo se construye el mensaje que se envía al modelo y cómo se gestiona la petición desde la interfaz hasta la obtención y presentación de la respuesta. En el Código 4.1 se recogen dos de las funciones implicadas en este flujo: una de ellas en el punto de entrada desde la UI `send_message(...)` y la otra relativa a la ejecución en hilo de trabajo `procesar_en_segundo_plano(...)`. Ambas se activan cada vez que el usuario realiza una acción.

La función `send_message(...)` resuelve el *prompt* de trabajo y selecciona la acción solicitada por el usuario: crear nueva sección (*crear*), modificar una sección existente (*modificar*) o reescribir una sección (*reescribir*). Este paso es clave porque condiciona cómo `StoryController` compondrá el mensaje final para el LLM.

El *prompt* del usuario se construye de dos formas: (1) Cuando se comienza una nueva historia, el usuario debe de introducir la propuesta inicial, en este caso se toma `prompt_usuario`. (2) En el resto de acciones, mediante los botones, se utiliza `prompt_override`.

En la acción *reescribir*, el usuario no necesita teclear nada: la propia aplicación genera una instrucción base indicando lo que tiene hacer el modelo:

```
"No me ha gustado la respuesta anterior. A continuación te
paso la sección que has generado. Por favor escribe
otra sección distinta pero manteniendo la coherencia
de la propuesta inicial.\n\n"
```

Para la acción *crear*, el usuario puede aportar o no características adicionales. Si escribe un mensaje, se incorpora y si no, se envía vacío. En *modificar*, el usuario debe indicar explícitamente qué quiere cambiar, y ese contenido se usa como *prompt*.

Una vez se tienen los *prompts* se llama a la función `procesar_mensaje(...)` que conecta con la clase `StoryController` encargada de crear el contexto y realizar la petición al servidor de `Groq`.

Es importante mantener el mayor detalle a la hora de construir el *prompt*, por tanto, enviar únicamente el *prompt* generado anteriormente, puede llegar a ser confuso para el LLM. Para solucionar este problema, la función `procesar_mensaje(...)` de `StoryController` gestiona de forma personalizada el mensaje que se envía al modelo, de cada una de las tres acciones. Para cada una de las acciones son distintos los mensajes para las secciones o para los capítulos. A continuación se muestran, tal como se emplean en la implementación, los mensajes creados para cada acción.

- *Acción crear*: Se integra el contexto de historia y la sección objetivo, junto con las características opcionales introducidas por el usuario:

```
mensaje = f"{self.contexto}\n\nAhora toca hacer la seccion:
{seccion}\nCaracteristicas para la sección: {user_input}"
```

El contexto (ver Sección 3.2.3) garantiza coherencia con el resto de secciones. En el caso de los capítulos, se refuerza adicionalmente el cierre en el cuarto capítulo para evitar que no finalice, como se ha observado en pruebas:

```
if self.chapter == 4:
    mensaje = (
        f"\nLos capítulos anteriores son:\n{self.contexto}\n"
        f"Los capitulos anteriores son: capitulo 1,
        capitulo 2 y capitulo 3\n"
        f"Debes TERMINAR la historia con el CAPÍTULO {self.chapter}. "
        f"Este capítulo debe ser el FINAL de la historia. "
        f"No repitas capítulos anteriores ni reinicies desde el 1."
    )
else:
    mensaje = (
        f"\nLos capítulos anteriores son:\n{self.contexto}\n"
        f"El capítulo anterior es el: {self.chapter-1}\n"
        f"Continúa la historia con el CAPÍTULO {self.chapter}. "
    )
```

- *Acción modificar*: Se envía al modelo la sección previamente creada como referencia, junto con la instrucción de cambio solicitada:

```
mensaje = f"\nContexto de la historia:\n{self.contexto}\nSección actual
\n\n{contenido_seccion}\nQuiero realizar el siguiente cambio:
{user_input}\nModifica la sección anterior para que incluya
este cambio. La seccion es: {seccion}\n"
```

- *Acción reescribir*: Del mismo modo, se aporta la sección generada para que el modelo produzca una nueva versión coherente con la propuesta inicial:

```
mensaje = f"\n\nCONTENIDO DEL MENSAJE\n\n{self.contexto}\n{user_input}
\nLa seccion antigua es:{seccion_antigua}\n
Prompt:{ultimo_prompt}Cambia la seccion {seccion}\n"
```

Finalmente estos mensajes se utilizan en la llamada al servidor de **Groq** para generar la respuesta del modelo y mostrarla al usuario. En la Sección 3.2.3 se puede encontrar en más detalle todo lo referido a la creación del contexto y al guardado en la base de datos.

Con este esquema conseguimos: (1) claridad para el modelo (contexto e instrucciones bien delimitados), (2) control explícito de la estructura (refuerzo de cierre en capítulo 4) y (3) coherencia narrativa al reutilizar el contexto y las secciones previas. Además, la separación entre la capa de UI (`send_message(...)`), la ejecución en segundo plano y el controlador (`StoryController`) facilita el mantenimiento y la extensión del sistema sin impactar en la experiencia del usuario.

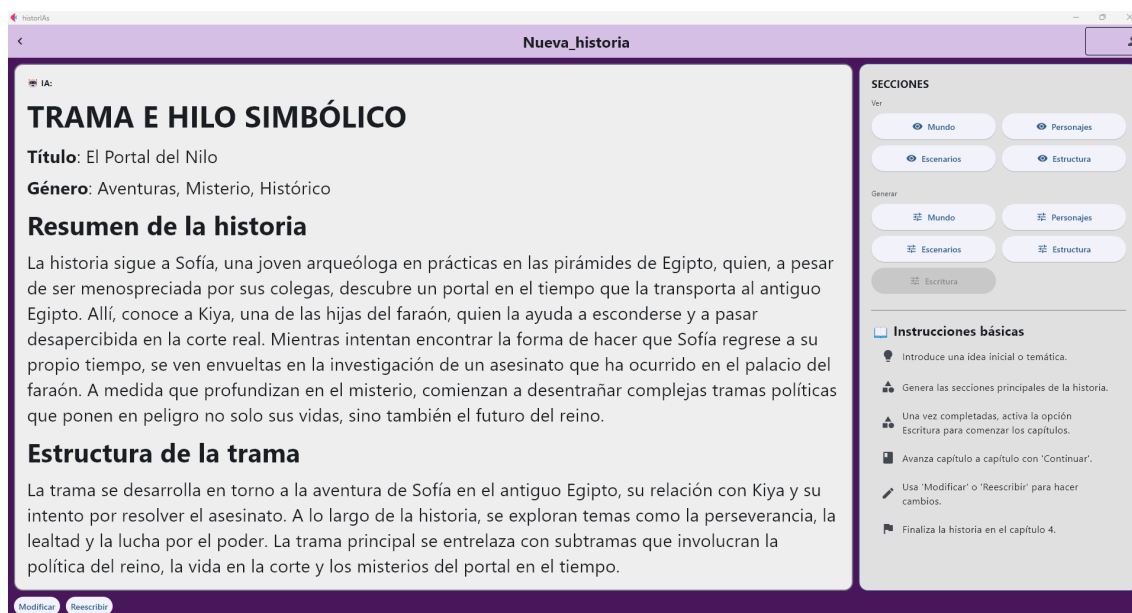


Figura 4.1: Historia de ejemplo: *Trama e hilo simbólico*. Mensaje introducido: ‘*Una arqueóloga de prácticas en las pirámides de Egipto, a la que menosprecian un poco, descubre sin querer un portal en el tiempo y viaja al antiguo Egipto donde conoce a una de las hijas del faraón que la ayuda a esconderse y pasar desapercibida y mientras intentan ayudarse Para encontrar la forma de volver a su tiempo, trata de resolver un asesinato que ha habido en el palacio del faraón y empiezan a desentrañar tramas políticas.*’

4.2. Ejemplo de una historia en el *modo detallado*

Una vez se comienza una nueva historia, se dispone de un cuadro de texto (ver Figura 3.12) donde se necesita que el usuario introduzca las características de su historia. Es válido desde una descripción mínima o nula (“*Quiero crear una historia*”, “*Genera una historia de ciencia ficción*”) a una descripción mucho más detallada como veremos en el ejemplo. Cuanto más se detalle más se ajusta la respuesta a la idea del usuario. Se envía este *prompt* y tras unos segundos de espera, el modelo nos muestra la *trama e hilo simbólico* que se trata de la primera de las secciones y única donde el usuario no puede elegir el orden. En la Figura 4.1 vemos la respuesta generada. Como se puede observar, el modelo introduce los aspectos que el usuario pidió. Como se ha comentado anteriormente, el modelo suele seguir una estructura en la respuesta aunque en ocasiones introduce variaciones. Una vez se muestra la primera sección, el resto de botones de las secciones se desbloquean para que sea el usuario el que elija la siguiente. El botón de *Escritura* da paso al desarrollo de los capítulos y se desbloquea cuando se completan el resto de secciones.

En este ejemplo se decide realizar una modificación mediante el botón *Modificación* y se habilita un cuadro de diálogo donde se introduce el cambio deseado (ver Figura 4.2). Tras esto el modelo vuelve a generar la sección con el cambio introducido (ver Figura 4.3). Se genera nuevamente la sección y comprobamos como sí realiza

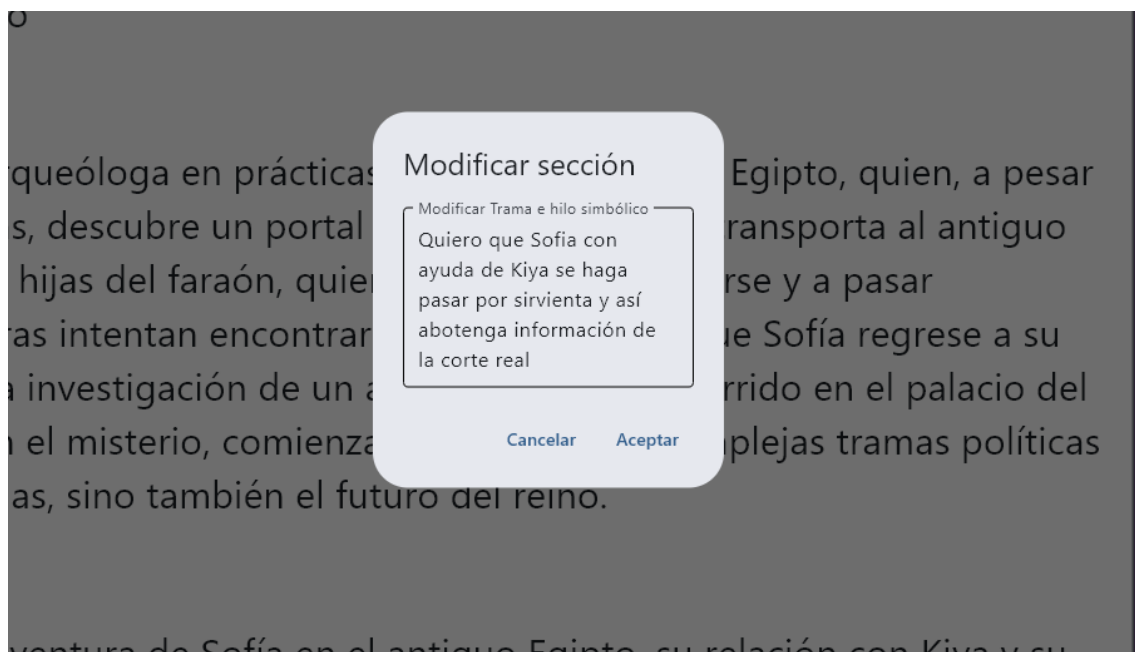


Figura 4.2: Historia de ejemplo: Modificar - *Trama e hilo simbólico*. Mensaje introducido: “*Quiero que Sofia con ayuda de Kiya se haga pasar por sirvienta y así obtenga información de la corte real.*”

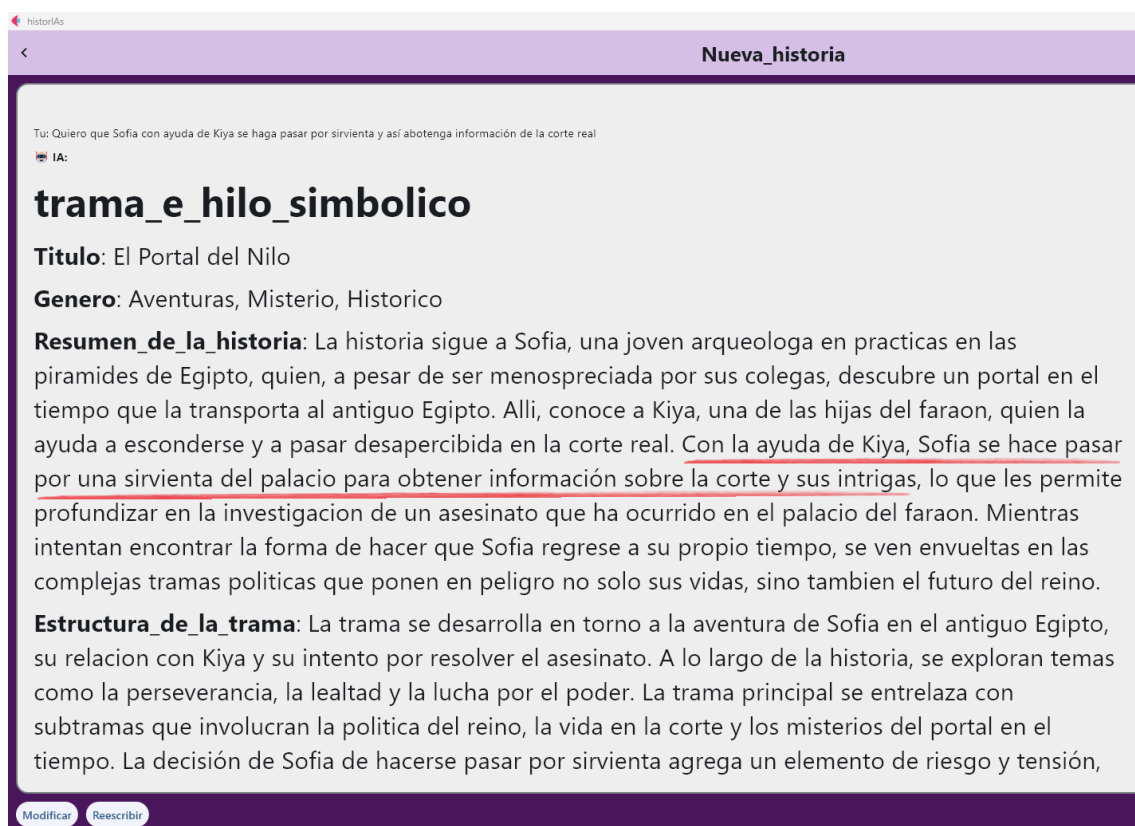


Figura 4.3: Historia de ejemplo: Modificación - *Trama e hilo simbólico*.

el cambio pedido. En ocasiones el modelo modifica el contenido global y no solo lo expresamente solicitado por el usuario.

Cuando el usuario decida, elige la siguiente sección en la parte de *Generar* a la derecha del recuadro principal. Como se muestra en la Figura 4.4, al seleccionar una sección se permite que el usuario introduzca una serie de características especiales para la sección. Esta parte se puede dejar en blanco o responder como en nuestro caso, que se pide una característica para los personajes.

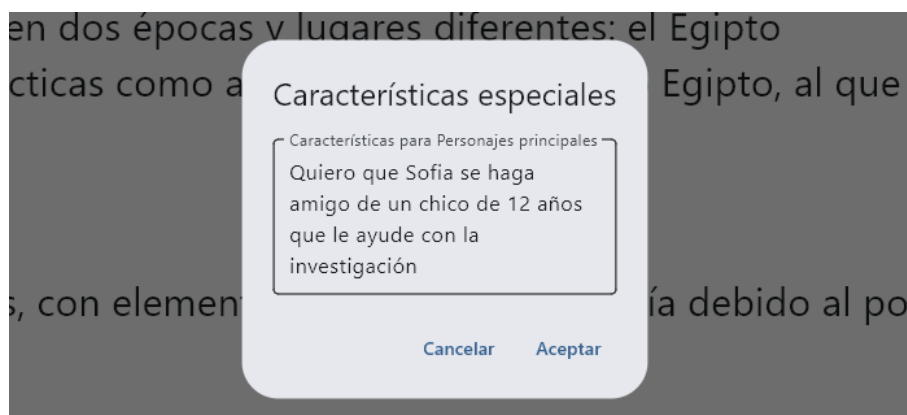


Figura 4.4: Historia de ejemplo: Características - *Personajes principales*. Mensaje introducido: “*Quiero que Sofia se haga amigo de un chico de 12 años que le ayude con la investigación*”

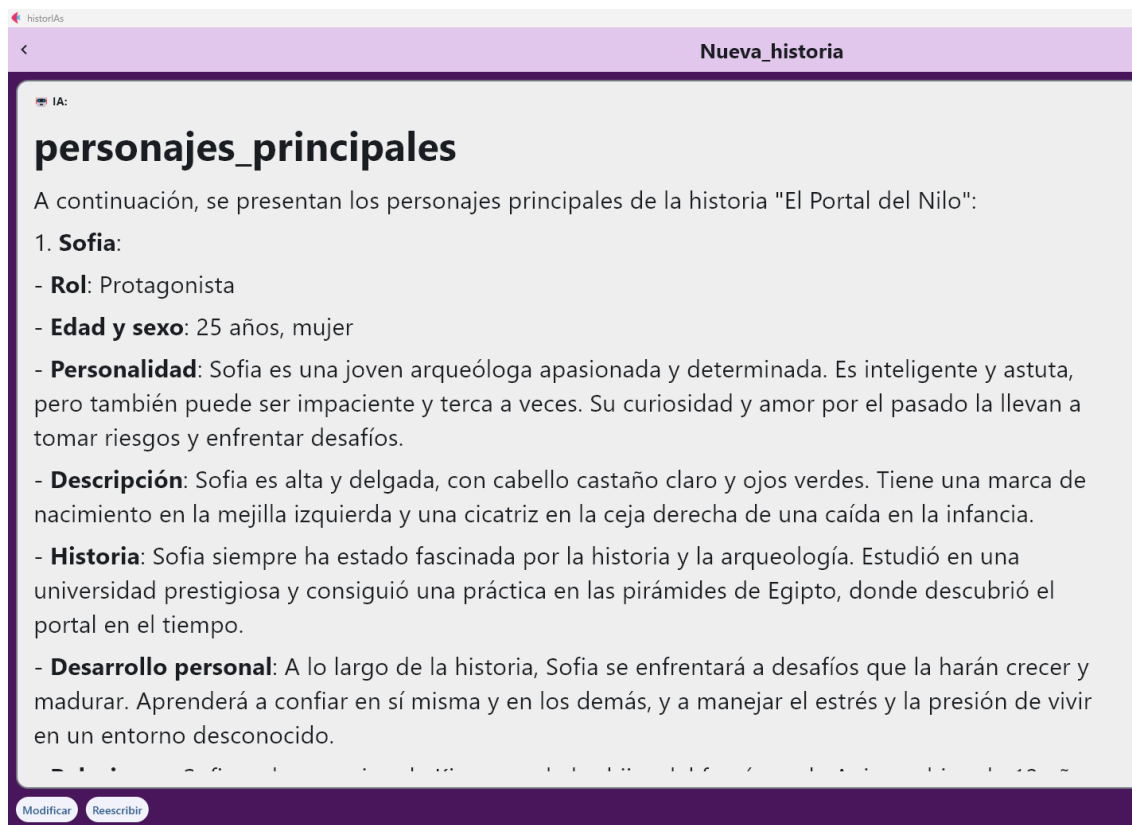


Figura 4.5: Historia de ejemplo: *Personajes principales*.

En la Figura 4.5 vemos como el modelo ha creado los personajes. Como se puede observar en el panel de la derecha, podemos hacer uso de los botones de *Ver* para observar la información de cada una de las secciones (Figura 4.6). Además, la ventana de visualización, ofrece la funcionalidad de *Editar*. Para realizar algún cambio se pulsa el botón *Editar*, lo que permite modificar de forma manual lo que el usuario desee. Se toma como ejemplo la visualización de un personaje que se edita para modificarlo. La Figura 4.7 muestra información sobre el personaje una vez modificado según lo solicitado al modelo. También se realiza esta modificación en la base de datos, aunque obviamente en el texto ya almacenado de los personajes previamente generados, no se realiza el cambio.

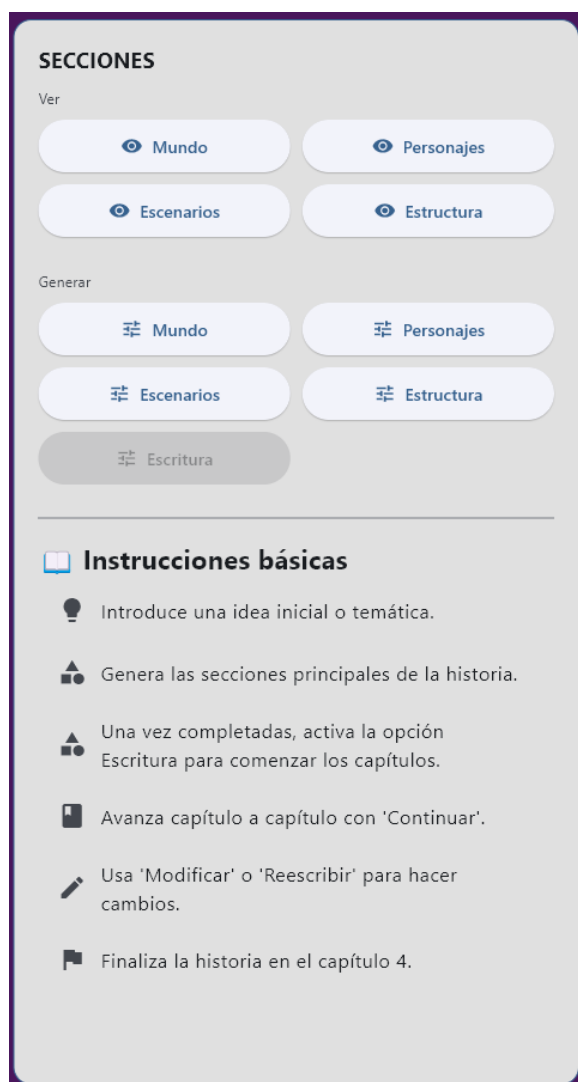
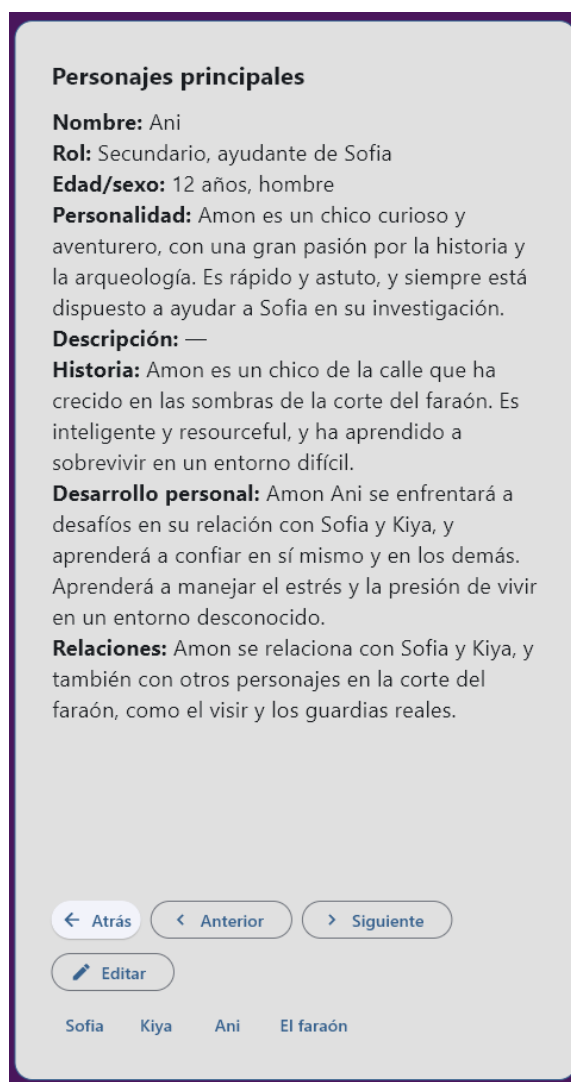


Figura 4.6: Panel derecho

Figura 4.7: Historia de ejemplo: *Personajes principales* - Editar.

Se continua con el resto de secciones hasta que se completan todas y se permite el acceso a la parte de la escritura de capítulos mediante el botón *Escritura*. Debido a

las limitaciones de *tokens* únicamente se permite realizar cuatro capítulos para no sobrepasar el límite. Como se muestra en la Figura 4.8 se eliminan los botones usados para generar las historias y se sustituye por el botón *Continuar* para avanzar los capítulos. De esta forma no se permiten realizar modificaciones ni volver a generar las secciones en mitad de los capítulos evitando futuros problemas.

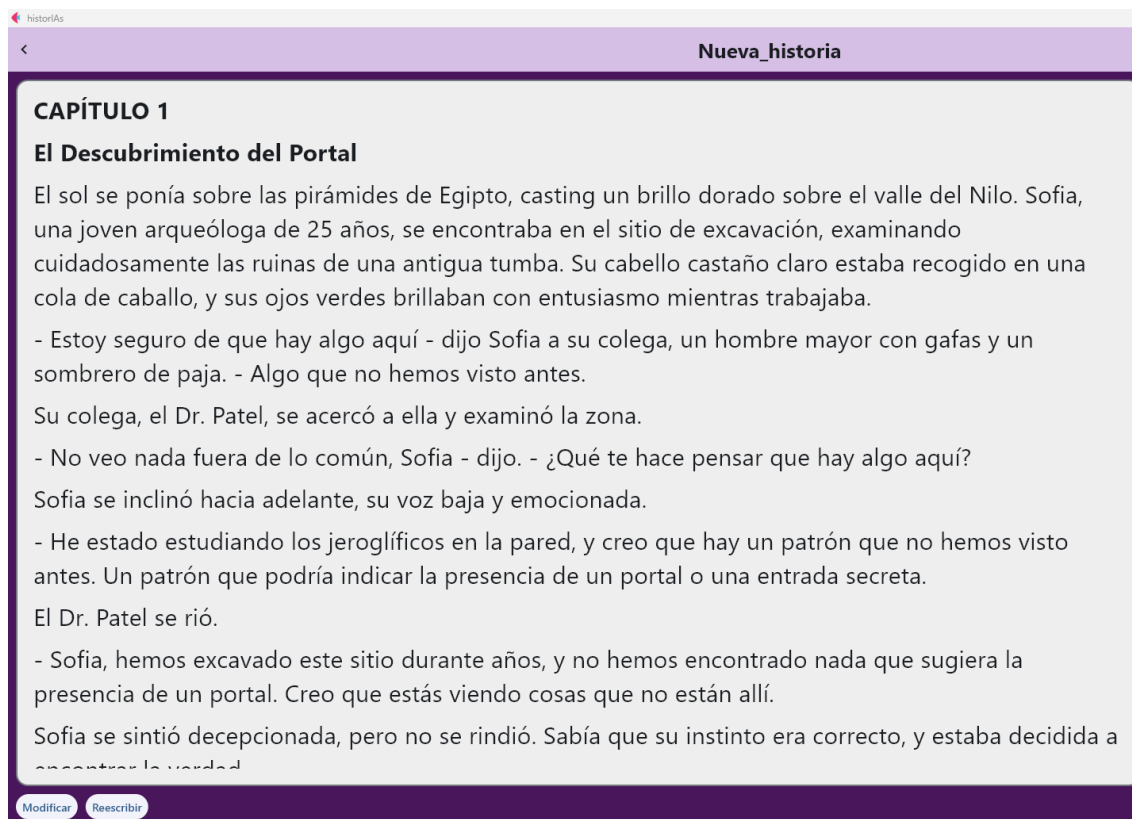
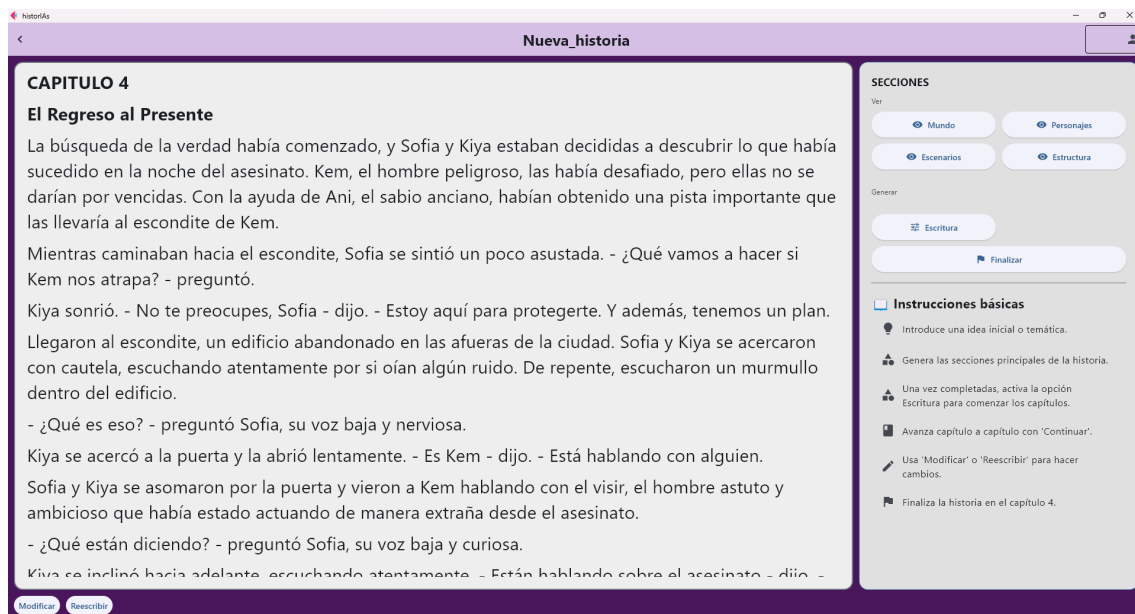
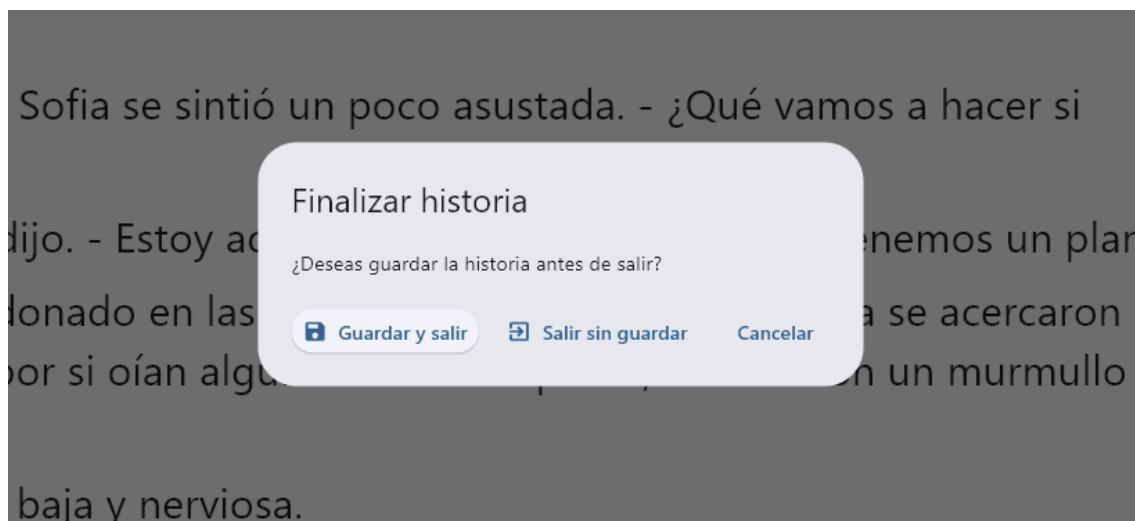


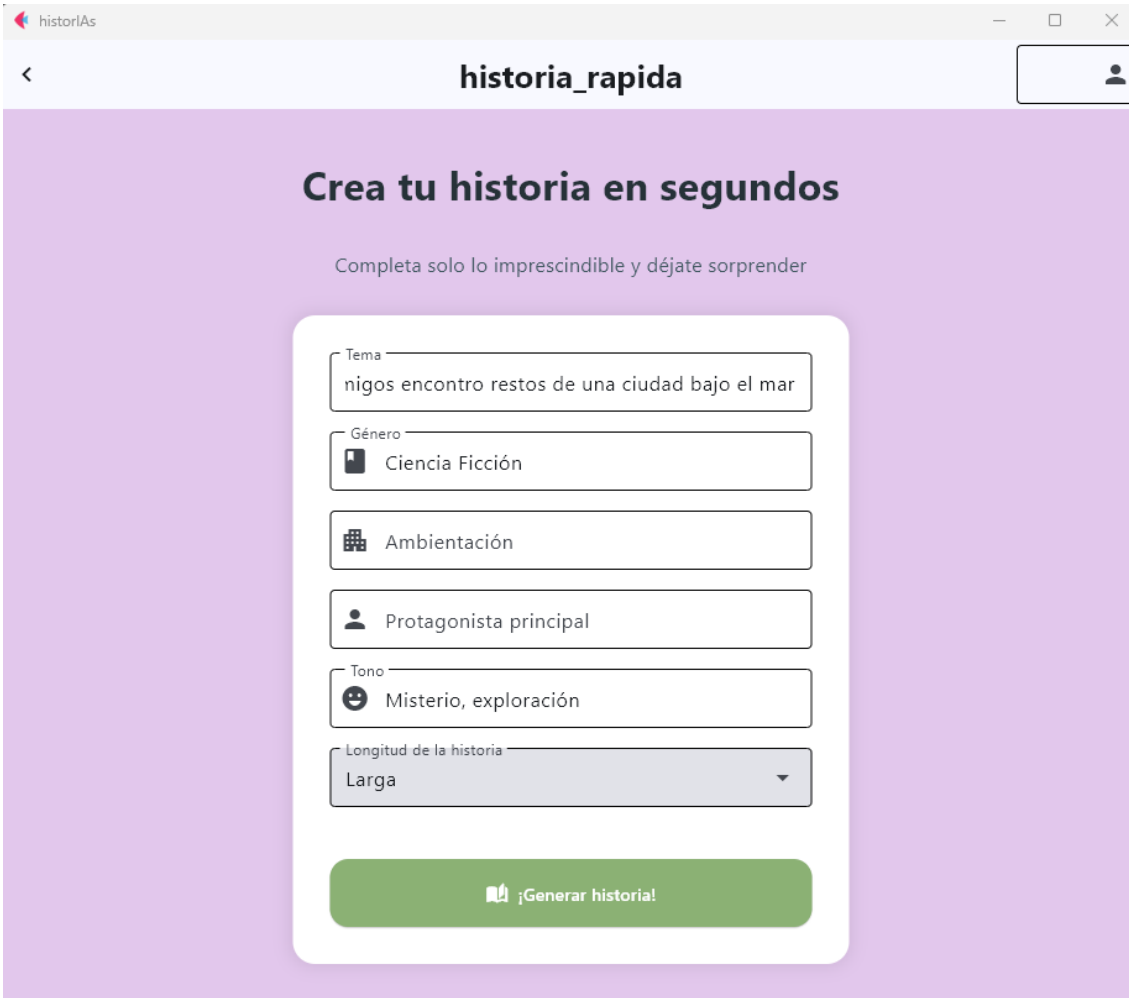
Figura 4.8: Historia de ejemplo: *Modo detallado* - Capítulo 1.

Se continua con los siguientes capítulos hasta llegar al cuarto y último capítulo donde la historia concluye. Como se observa en la Figura 4.9 al ser el último capítulo, el botón *Continuar* se sustituye por el botón *Finalizar* permitiendo que el usuario guarde la historia generada en formato pdf (ver Figura 4.10) y volviendo a la pantalla principal (ver Figura 3.11). Este sería un ejemplo de creación del *modo detallado* (ver historia completa en el Apéndice B.1). En la mayoría de pruebas realizadas, el modelo implementa de forma correcta las peticiones del usuario y mantiene un estilo de salida similar, aunque en ocasiones pueda haber algún inconveniente como por ejemplo, mostrar toda la respuesta en negrita. Se puede llegar al final generando las secciones en otro orden y realizando más o menos modificaciones. Podría ser interesante analizar si estas variaciones afectan a la calidad de la historia.

Figura 4.9: Historia de ejemplo: *Modo detallado* - Capítulo 4.Figura 4.10: Historia de ejemplo: *Capítulo 4*.

4.3. Ejemplo de una historia en el *modo rápido*

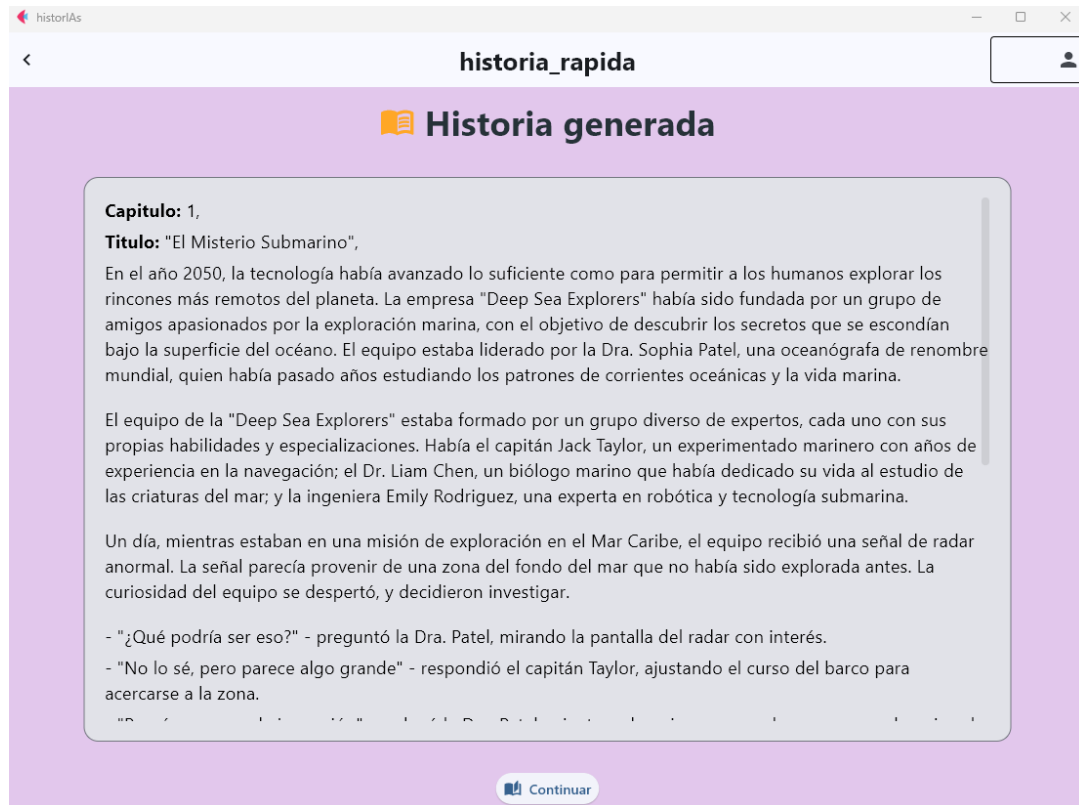
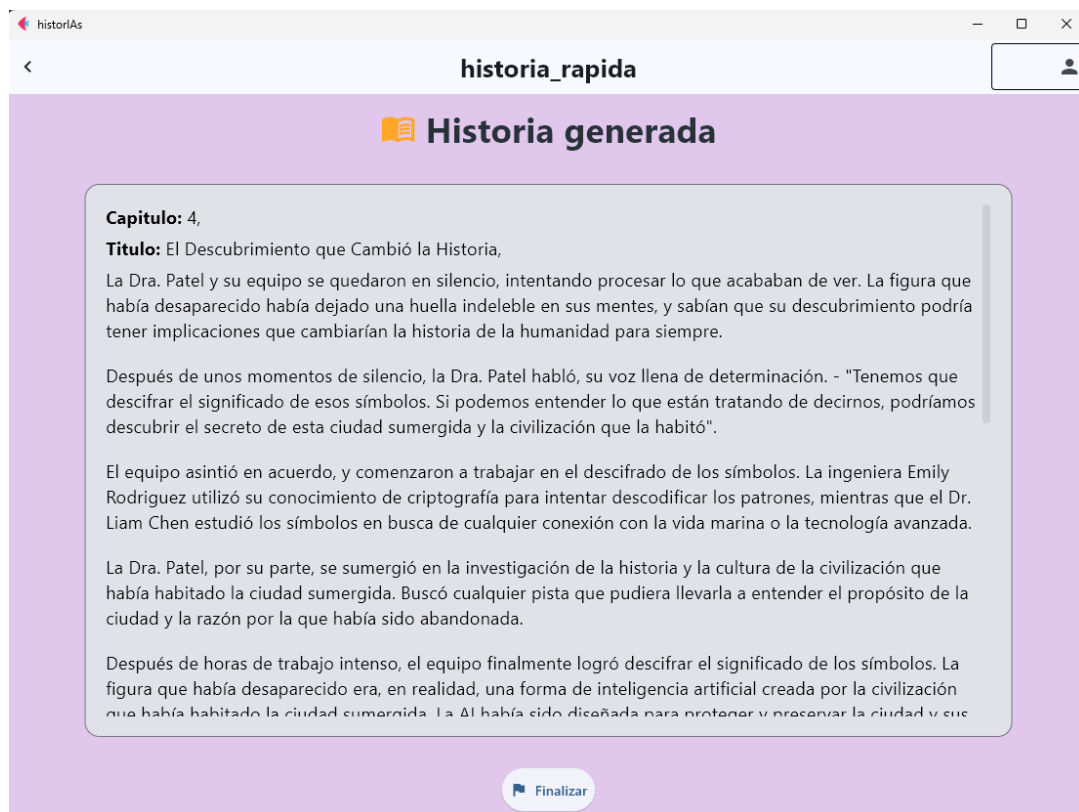
Para crear una historia en el *modo rápido*, el proceso comienza con el cuestionario inicial donde se recoge la información básica de la historia propuesta por el usuario. Como se muestra en la Figura 4.11, no es imprescindible completar todos los campos, el modelo rellena de forma coherente los datos restantes. Aun así, cuanto más detalle proporcione el usuario, mayor será la alineación de la historia resultante con sus preferencias.



The screenshot shows a web browser window with the title 'historia_rapida'. The main heading is 'Crea tu historia en segundos' (Create your story in seconds). Below it, a subtitle reads 'Completa solo lo imprescindible y déjate sorprender' (Complete only the essentials and let yourself be surprised). The form contains several input fields: 'Tema' (Topic) with the text 'nigos encontro restos de una ciudad bajo el mar'; 'Género' (Genre) with a book icon and 'Ciencia Ficción'; 'Ambientación' (Setting) with a city grid icon; 'Protagonista principal' (Main protagonist) with a person icon; 'Tono' (Tone) with a smiley face icon and 'Misterio, exploración'; and 'Longitud de la historia' (Story length) with a dropdown menu set to 'Larga'. At the bottom is a green button with a document icon and the text '¡Generar historia!'.

Figura 4.11: Historia de ejemplo: *Cuestionario inicial*.

Una vez se rellena el cuestionario se muestra el primero de los cuatro capítulos. Como se muestra en la Figura 4.12, en el *modo rápido* únicamente tenemos disponible el botón para generar el siguiente capítulo. Se generarán los siguientes capítulos hasta llegar al cuarto donde, al igual que en el *modo detallado*, el botón de *Continuar* se sustituye por el botón de *Fin* permitiendo al usuario guardar en un pdf la historia generada (ver Figura 4.13).

Figura 4.12: Historia de ejemplo: *Modo rápido* - Capítulo 1.Figura 4.13: Historia de ejemplo: *Modo rápido* - Capítulo 4.

4.4. Evaluación de las historias

Tras la creación de las historias se realizó una evaluación con lectores para comprender cómo perciben las historias creadas y qué mejoras conviene introducir en el sistema. En total se recopilaron seis encuestas: dos sobre historias generadas en *modo detallado* y cuatro sobre historias en *modo rápido*. Dado la cantidad de pruebas y el acceso limitado a usuarios, el objetivo no es extraer conclusiones estadísticas, sino analizar las reflexiones de los usuarios.

En esta sección se presenta una síntesis cualitativa de las encuestas recogidas, centrada en lo aprendido sobre lenguaje, coherencia, cierre narrativo y experiencia de uso. A continuación se muestran los puntos más relevantes.

- **Calidad de lenguaje:** Los lectores perciben un estilo demasiado sencillo y con repeticiones léxicas/sintácticas. En ocasiones aparece mezcla de registros o vocabulario poco natural. Se ha observado como expresiones e incluso frases completas, se han repetido a lo largo de los capítulos. También de forma más puntual, aparecen errores ortográficos y palabras en inglés. También se ha observado el escaso nivel descriptivo, aunque debido a la limitación de únicamente cuatro capítulos, no es algo significativo en nuestro caso.
- **Coherencia narrativa:** En general, se aprecia una buena coherencia en la evolución de la historia y entre capítulos. No obstante, en ocasiones, durante la creación de secciones se introducen escenarios o eventos que luego no llegan a materializarse en la redacción final. Queda la duda de si estos desajustes se deben a “olvidos” del LLM o a que el modelo proyecta una historia más larga y, al imponer el cierre en el capítulo cuatro, se ve obligado a cortar líneas argumentales abiertas.
- **Cierre y clímax:** Es, junto con la calidad del lenguaje, uno de los puntos más débiles. El refuerzo de los *prompts* permitió evitar que la historia se prolongara más allá del cuarto capítulo y superara el límite de *tokens*. Aun así, los finales suelen ser apresurados y simples; persiste la duda de si esta debilidad se debe, en parte, a la restricción de limitar la obra a solo cuatro capítulos.
- **Rol de la IA en la creación de historias:** La IA se percibe como un apoyo creativo eficaz para desbloquear y explorar alternativas, no como sustituto del autor. Todos los usuarios destacan su utilidad para superar bloqueos narrativos y para generar ideas, especialmente durante la construcción de secciones (trama, personajes, escenarios). Aun así, se considera imprescindible una revisión y edición humana final.
- **Modo detallado vs. modo rápido:** El *modo detallado* genera tramas más sólidas y una ambientación mejor integrada, y tiende a producir historias más largas, probablemente por el mayor contexto disponible. El *modo rápido*, en cambio, destaca por su inmediatez, permitiendo crear una historia a partir de casi cualquier idea, favoreciendo un uso más lúdico y exploratorio.

En resumen, los usuarios perciben que la calidad literaria y la carga emocional de las historias generadas aún están muy por debajo de las escritas por humanos, por lo que consideran imprescindible la supervisión y edición humana en todo momento. Aun así, valoran la creatividad y la diversión de convertir una idea propia en una historia, y señalan la creación por secciones como la parte más útil y productiva del proceso.

Conclusiones y Trabajo Futuro

En este trabajo se presenta **historIA**s, una aplicación orientada a generar historias por secciones, haciendo uso de los LLMs, para conseguir un mayor grado de personalización y edición por parte del usuario antes y durante la creación. La aplicación guarda la información en una base de datos de modo que el usuario puede retomar la historia en cualquier momento. También existe el *modo rápido* donde, tras rellenar un pequeño cuestionario con información, el modelo genera directamente la historia usando los datos introducidos.

Tras realizar numerosas pruebas se observa como las historias generadas seguían en muchas ocasiones una serie de similitudes y sesgos. Cuando se trataba de historias con temática parecida, repetía nombres de personaje y descripciones similares. Si el *prompt* no estaba muy detallado, en cualquier momento introducía aspectos mágicos y casi siempre empleaba el recurso del protagonista teniendo que derrotar al villano. La coherencia se mantenía la mayoría de ocasiones aunque hubo casos donde por ejemplo, en la sección de *descripción de escenarios* creaba un escenario y finalmente no aparecía en la historia. Al final se ha observado como estos modelos, a pesar de mantener cada vez mejor la coherencia e hilo de la historia, siguen siendo muy inferiores a la capacidad de redacción de un escritor profesional. La riqueza lingüística es más escasa repitiendo en muchas ocasiones palabras, expresiones o incluso frases completas. La calidad de las historias es muy simple, no se producen giros y sigue una estructura lineal haciendo que no produzca una gran emoción en el lector. En el estudio de Tian, Huang, et. al. [25] en la Figura 5.1 se muestra la evolución de la fortuna de los personajes: En este gráfico se expone la comparativa entre los arcos argumentales y las posiciones de los puntos de inflexión de las narrativas generadas por humanos y por LLMs. El eje vertical muestra la fortuna del personaje (de mala a buena) y el eje horizontal representa la línea del tiempo de la historia. En comparación con la escritura tradicional, los LLMs tienden a emplear arcos más alegres y menos complejos, introducen antes los puntos de inflexión en la trama y tienen menos suspense y contratiempos en sus narrativas. Esta misma tendencia se aprecia en las historias creadas con esta aplicación.

Otro aspecto importante del proyecto era analizar la comparativa en la generación de las historias entre el *modo rápido*, más similar a lo que hay en el mercado,

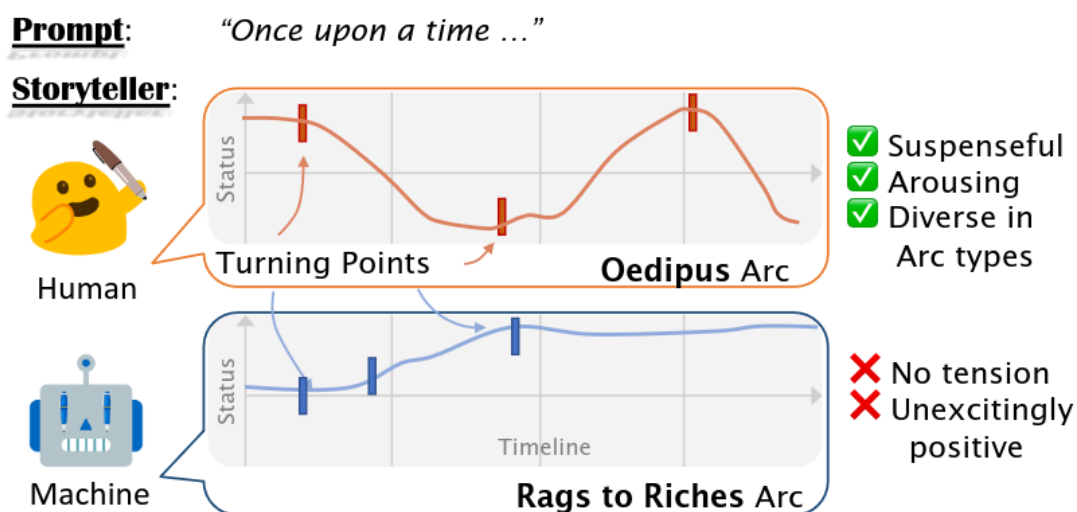


Figura 5.1: Comparación de arcos narrativos entre humanos y modelos de lenguaje (fuente del gráfico [25])

con el *modo detallado* propuesto en este proyecto. Tras ser utilizados estos modos de creación por varios usuarios, tras la evaluación aportada por los mismos (se adjuntan en el Apéndice B), se puede deducir, según sus evaluaciones, que el nivel de coherencia era similar entre ambos modos. No obstante el *modo detallado* sí crea una historia más extensa y desarrollada que el *modo rápido*. Uno de los aspectos más destacados del proyecto ha sido la funcionalidad de creación por secciones. Esta estructura no solo facilita la personalización y el control narrativo por parte del usuario, sino que también se ha observado como una herramienta muy útil para generar ideas y estimular la creatividad (ver Sección 4.2 para más detalle). Aunque la calidad literaria de los textos generados por los modelos aún presenta limitaciones, el proceso de construir una historia paso a paso resulta dinámico, entretenido y enriquecedor, según la experiencia de los usuarios que han probado la aplicación (ver Sección 4.4). En este sentido, la aplicación no solo cumple una función generativa, sino también inspiradora, pudiendo servir como punto de partida para quienes deseen crear historias.

En conjunto, los objetivos planteados se han alcanzado de forma satisfactoria, demostrando la viabilidad de la aplicación y del flujo por secciones. No obstante, el desarrollo ha revelado limitaciones y oportunidades de mejora, especialmente en calidad del lenguaje y control del cierre narrativo, que orientan el trabajo futuro.

El objetivo principal fue la implementación de un LLM. La restricción determinante vino dada por las capacidades hardware, lo que motivó el uso de un servidor para acceder al modelo. Como mejora, se propone su implantación local, a fin de minimizar la dependencia del servicio remoto y las limitaciones de los *tokens*.

También se ha podido lograr crear una aplicación que permite un gran grado de edición e interacción por parte del usuario. Sin embargo, se podrían introducir

mejoras que permitan al usuario crear más personajes y escenarios y no limitarse únicamente al máximo de cuatro establecido para no superar el límite de *tokens*. Se ha logrado diseñar un flujo narrativo dividiendo la historia en secciones, pero tras las pruebas realizadas se pensó en la posibilidad de poder aumentar el número de secciones e incluso que el usuario pueda definir previamente, nuevas secciones y cuales usar. También se podría incluir la posibilidad de modificar capítulos y no únicamente la parte de las secciones.

Se han implementado dos modos de creación bien diferenciados y se abre la posibilidad de incorporar nuevas funcionalidades, como permitir que el usuario ajuste en tiempo real parámetros de generación, por ejemplo la *temperature*, para modificar la creatividad a su gusto desde la propia aplicación.

Se ha llevado a cabo una evaluación con usuarios de la aplicación. De sus comentarios se desprende el interés poder modificar capítulos ya completados. Asimismo, se identifica como línea prioritaria mejorar la calidad del lenguaje, reforzando los *prompts* mediante técnicas de *prompting*, para reducir repeticiones, aumentar la diversidad léxica y elevar la calidad narrativa del resultado.

Apéndice A

Modelos Groq

A continuación se muestra una tabla con todos los LLMs disponibles en la versión gratuita de Groq [9] y sus limitaciones de *tokens*:

MODEL ID	RPM	RPD	TPM	TPD	ASH	ASD
allam-2-7b	30	7K	6K	500K	-	-
compound-beta	15	200	70K	-	-	-
compound-beta-mini	15	200	70K	-	-	-
deepseek-r1-distill-llama-70b	30	1K	6K	100K	-	-
gemma2-9b-it	30	14.4K	15K	500K	-	-
llama-3.1-8b-instant	30	14.4K	6K	500K	-	-
llama-3.3-70b-versatile	30	1K	12K	100K	-	-
llama3-70b-8192	30	14.4K	6K	500K	-	-
llama3-8b-8192	30	14.4K	6K	500K	-	-
meta-llama/llama-4-maverick-17b-128e-instruct	30	1K	6K	500K	-	-
meta-llama/llama-4-scout-17b-16e-instruct	30	1K	30K	500K	-	-
meta-llama/llama-guard-4-12b	30	14.4K	15K	500K	-	-
meta-llama/llama-prompt-guard-2-22m	30	14.4K	15K	500K	-	-
meta-llama/llama-prompt-guard-2-86m	30	14.4K	15K	500K	-	-
moonshotai/kimi-k2-instruct	60	1K	10K	300K	-	-
openai/gpt-oss-120b	30	1K	8K	200K	-	-
openai/gpt-oss-20b	30	1K	8K	200K	-	-
playai-tts	10	100	1.2K	3.6K	-	-
playai-tts-arabic	10	100	1.2K	3.6K	-	-
qwen/qwen3-32b	60	1K	6K	500K	-	-
whisper-large-v3	20	2K	-	-	7.2K	28.8K
whisper-large-v3-turbo	20	2K	-	-	7.2K	28.8K

Tabla A.1: Tabla LLM Groq

Apéndice **B**

Ejemplo de una historia y encuestas de los usuarios

A continuación se muestra el pdf con la historia generada en la Sección 4.2 junto con las encuestas:

B.1. Ejemplo de historia en modo detallado

Título: El enigma del símbolo

Capítulo 1: El descubrimiento del portal

El sol se ponía sobre las pirámides de Egipto, casting un brillo dorado sobre el valle del Nilo. Sofia, una joven arqueóloga de 25 años, se encontraba en el sitio de excavación, examinando cuidadosamente las ruinas de una antigua tumba. Su cabello castaño claro estaba recogido en una cola de caballo, y sus ojos verdes brillaban con entusiasmo mientras trabajaba. - Estoy seguro de que hay algo aquí - dijo Sofia a su colega, un hombre mayor con gafas y un sombrero de paja. - Algo que no hemos visto antes. Su colega, el Dr. Patel, se acercó a ella y examinó la zona. - No veo nada fuera de lo común, Sofia - dijo. - ¿Qué te hace pensar que hay algo aquí? Sofia se inclinó hacia adelante, su voz baja y emocionada. - He estado estudiando los jeroglíficos en la pared, y creo que hay un patrón que no hemos visto antes. Un patrón que podría indicar la presencia de un portal o una entrada secreta. El Dr. Patel se rió. - Sofia, hemos excavado este sitio durante años, y no hemos encontrado nada que sugiera la presencia de un portal. Creo que estás viendo cosas que no están allí. Sofia se sintió decepcionada, pero no se rindió. Sabía que su instinto era correcto, y estaba decidida a encontrar la verdad. - Quiero seguir investigando - dijo. - ¿Puedo quedarme aquí y seguir examinando la zona? El Dr. Patel se encogió de hombros. - Como quieras, Sofia. Pero no te quedes demasiado tiempo. La noche cae pronto, y no queremos que te quedes aquí sola. Sofia asintió y se quedó en el sitio, examinando cuidadosamente la zona. A medida que la

noche caía, el aire se enfrió y la oscuridad se cernió sobre el valle del Nilo. Sofia se sintió sola y un poco asustada, pero no se movió. Estaba decidida a encontrar lo que estaba buscando. De repente, escuchó un ruido detrás de ella. Se dio la vuelta y vio una figura emergiendo de la oscuridad. Era una mujer con cabello negro y ojos oscuros, vestida con ropa tradicional egipcia. - ¿Quién eres? - preguntó Sofia, sorprendida. La mujer se acercó a ella y sonrió. - Me llamo Kiya - dijo. - Y he venido a ayudarte, Sofia. Sofia se sintió confundida. - ¿Ayudarme? ¿En qué? Kiya se inclinó hacia adelante, su voz baja y misteriosa. - He estado observándote, Sofia. Sé que estás buscando algo. Algo que podría cambiar el curso de la historia. Sofia se sintió emocionada. - ¿Qué sabes? - preguntó. Kiya sonrió. - Sé que has encontrado el portal - dijo. - El portal que te llevará al antiguo Egipto. Sofia se sintió asombrada. - ¿Cómo sabes eso? - preguntó. Kiya se encogió de hombros. - Lo sé - dijo. - Y estoy aquí para ayudarte a cruzar. Sofia se sintió emocionada y asustada al mismo tiempo. Sabía que estaba a punto de embarcarse en una aventura que cambiaría su vida para siempre. - ¿Estás lista? - preguntó Kiya. Sofia asintió, su corazón latiendo con emoción. - Sí - dijo. - Estoy lista. Y con eso, Sofia y Kiya se adentraron en la oscuridad, hacia el portal que las llevaría al antiguo Egipto.

Capítulo 2: El Misterio de Julius Fuchs

Alex y Emma se miraron confundidos, aún tratando de procesar lo que acababan de ver. El símbolo brillante había desaparecido tan rápido como había aparecido, dejándolos con más preguntas que respuestas. Max, el perro de Alex, seguía mirando fijamente hacia el árbol, como si esperara que algo más sucediera.

-¿Qué acabamos de ver?- repitió Emma, su voz llena de asombro.

-No lo sé -respondió Alex, sacudiendo la cabeza-. Pero creo que acabamos de descubrir algo importante.

Emma asintió, su mente ya trabajando en la teoría. -Tenemos que investigar más sobre ese símbolo. Puede que esté relacionado con el proyecto de investigación que queremos presentar.

Alex asintió, su propia curiosidad ya despertada. -Vamos a la biblioteca universitaria. Puede que encontremos algo sobre ese símbolo.

Emma se levantó del banco, recogiendo su carpeta y su laptop. -Vamos. Quiero saber qué significa ese símbolo.

Alex se levantó también, llamando a Max para que lo siguiera. Juntos, los tres se dirigieron hacia la biblioteca universitaria, con la intención de descubrir el secreto detrás del símbolo misterioso.

Una vez en la biblioteca, Emma y Alex se dirigieron directamente a la sección de historia y física. Emma comenzó a buscar en los estantes, mientras que Alex se sentó en una mesa con su laptop, intentando encontrar información en línea sobre el símbolo.

Después de varios minutos de búsqueda, Emma encontró un libro sobre la historia del Proyecto Manhattan. -¡Eureka! -exclamó, mostrando el libro a Alex.

Alex se acercó a ella, interesado. -¿Qué dice?

Emma comenzó a hojear el libro, buscando la información que necesitaban. -Aquí dice que Julius Fuchs, un físico alemán, trabajó en el Proyecto Manhattan durante la Segunda Guerra Mundial.

Alex se sorprendió. -¿Julius Fuchs? ¿No es el tipo que fue asesinado en Ravenswood en los años 40?

Emma asintió. -Ese mismo. Dice que su asesinato fue un misterio que nunca se resolvió.

Alex se inclinó hacia adelante, interesado. -¿Y qué relación tiene con el símbolo que vimos?

Emma continuó leyendo. -Dice que Fuchs trabajó en un proyecto secreto en Ravenswood, relacionado con la física nuclear. Puede que el símbolo sea una pista sobre ese proyecto.

Alex se sintió emocionado. -¡Eso es! Tenemos que investigar más sobre el proyecto secreto de Fuchs.

Emma asintió, su mente ya trabajando en la teoría. -Vamos a buscar más información sobre el proyecto. Puede que encontremos una conexión entre el símbolo y el asesinato de Fuchs.

Alex se levantó, decidido. -Vamos a resolver este misterio.

Emma sonrió, lista para la aventura. -Vamos a hacerlo.

Capítulo 3: La Pista del Pasado

Alex y Emma se sumergieron de lleno en la investigación sobre el proyecto secreto de Julius Fuchs. Pasaron horas en la biblioteca universitaria, buscando cualquier información que pudiera llevarlos a la verdad. Max, el perro de Alex, se quedó dormido a sus pies, cansado de tanto caminar y esperar.

Después de lo que parecía una eternidad, Emma finalmente encontró un documento que llamó la atención de Alex. Era una carta escrita por Fuchs mismo, dirigida a un colega suyo en el Proyecto Manhattan.

-“¡Eureka!” -exclamó Emma, mostrando la carta a Alex.

Alex se acercó a ella, emocionado. -¿Qué dice? -preguntó, ansioso por saber más.

Emma comenzó a leer la carta en voz alta. La carta hablaba sobre un experimento secreto que Fuchs había estado trabajando en Ravenswood, relacionado con la física nuclear. Pero lo que realmente llamó la atención de Alex y Emma fue la mención a un lugar llamado “El Archivo”.

-¿Qué es El Archivo? -preguntó Alex, intrigado.

Emma se encogió de hombros. -No tengo idea. Pero puede que sea una pista importante.

Alex asintió, decidido. -Vamos a encontrar El Archivo. Puede que contenga la clave

para resolver el misterio del asesinato de Fuchs.

Emma asintió, lista para la aventura. -Vamos a hacerlo.

Juntos, Alex y Emma salieron de la biblioteca universitaria, decididos a encontrar El Archivo y descubrir la verdad sobre el proyecto secreto de Julius Fuchs.

Capítulo 4: La Verdad Oculta

Alex y Emma se dirigieron al parque central, decididos a encontrar El Archivo y descubrir la verdad sobre el proyecto secreto de Julius Fuchs. Max, el perro de Alex, los siguió de cerca, como si supiera que estaban cerca de resolver el misterio.

Después de buscar durante horas, finalmente encontraron un edificio antiguo y abandonado en el parque central. La puerta estaba cerrada con candado, pero Alex encontró una forma de abrirla.

-Dentro -dijo Emma, emocionada.

Alex y Emma entraron en el edificio, encontrando una habitación llena de archivos y documentos antiguos. Emma comenzó a buscar mientras Alex examinaba los archivos.

-¡Eureka! -exclamó Emma, encontrando un archivo etiquetado como "Proyecto Fuchs".

Alex se acercó a ella, emocionado.

-¿Qué dice? -preguntó.

Emma comenzó a leer el archivo en voz alta. El archivo reveló que Julius Fuchs había estado trabajando en un proyecto secreto para desarrollar una bomba atómica en Ravenswood. Sin embargo, Fuchs había descubierto que el proyecto era demasiado peligroso y había decidido abandonarlo. Pero antes de que pudiera hacerlo, fue asesinado por alguien que quería continuar con el proyecto.

-¡Eso explica todo! -dijo Alex, emocionado.

Emma asintió.

-Sí, pero hay algo más. El archivo también menciona que Fuchs había escondido la información del proyecto en un lugar seguro, conocido como "El Archivo".

Alex y Emma se miraron, emocionados.

-Eso debe ser el lugar donde encontramos el símbolo -dijo Alex.

Emma asintió.

-Sí, y creo que sé dónde está.

Emma llevó a Alex y Max a un lugar en el parque central donde había un árbol con un agujero en la corteza. Emma metió la mano en el agujero y sacó un disco duro antiguo.

-Eso debe ser El Archivo -dijo Alex, emocionado.

Emma asintió y comenzó a examinar el disco duro. Después de unos minutos, encontró la información que estaban buscando.

-¡Lo hice! -exclamó Emma.

Alex se acercó a ella, emocionado.

-¿Qué dice? -preguntó.

Emma comenzó a leer la información en voz alta. El archivo reveló que Julius Fuchs había escondido la información del proyecto en un lugar seguro, para evitar que cayera en malas manos.

También mencionaba que Fuchs había dejado una pista para que alguien pudiera encontrar la información en el futuro.

-¡Eso es increíble! -dijo Alex, emocionado.

Emma asintió.

-Sí, y creo que hemos resuelto el misterio.

Alex y Emma se miraron, satisfechos de haber resuelto el misterio del asesinato de Julius Fuchs. Max, el perro de Alex, ladró suavemente, como si estuviera contento de haber ayudado a resolver el misterio.

B.2. Encuestas realizadas por los usuarios

Se han realizado seis encuestas pero en el apéndice pondremos únicamente la correspondiente a la historia de ejemplo, una extra del *modo rápido* y la plantilla vacía.

Checklist de evaluación: usabilidad e historia generada

Rango de Edad: menor de 15 ☐ 15 – 25 ☐ 26 – 40 ☒ 41 – 60 ☐ más de 60 ☐
Modo usado: Rápido ☐ Detallado ☒

A) Evaluación de la historia generada

(1 = muy bajo · 5 = excelente)

1. Coherencia global de la trama: 1 ☐ 2 ☐ 3 ☐ 4 ☒ 5 ☐
2. Consistencia entre capítulos: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☒
3. Desarrollo de personajes: 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5 ☐
4. Decisiones y conflictos: 1 ☐ 2 ☐ 3 ☒ 4 ☐ 5 ☐
5. Subtramas: 1 ☒ 2 ☐ 3 ☐ 4 ☐ 5 ☐
6. Transiciones entre escenas/capítulos: 1 ☐ 2 ☐ 3 ☐ 4 ☒ 5 ☐
7. Ritmo de la historia: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☒
8. Originalidad/interés de la historia: 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5 ☐
9. Calidad del lenguaje: 1 ☒ 2 ☐ 3 ☐ 4 ☐ 5 ☐
10. Mundo/ambientación: 1 ☐ 2 ☐ 3 ☐ 4 ☒ 5 ☐
11. Clímax y desenlace: 1 ☒ 2 ☐ 3 ☐ 4 ☐ 5 ☐
12. Sesgos/estereotipos: 1 ☐ 2 ☐ 3 ☐ 4 ☒ 5 ☐
13. Impacto emocional: 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5 ☐

Comentarios sobre la historia (puntos fuertes, cabos sueltos, sugerencias):

El lenguaje utilizado era muy repetitivo y algunas estructuras gramaticales eran dudosas.
 En la descripción de la estructura se ha hecho referencia a eventos que debían suceder en los capítulos que no han llegado a ocurrir y en la descripción de escenarios aparecía el apartamento de los protagonistas que no ha llegado a aparecer realmente en la historia.

B) Opinión: escritura tradicional e IA

(1 = muy en desacuerdo · 5 = muy de acuerdo)

1. Ayuda en bloqueos creativos: 1 ☐ 2 ☐ 3 ☒ 4 ☐ 5 ☐
2. IA como apoyo (no sustituto): 1 ☐ 2 ☐ 3 ☒ 4 ☐ 5 ☐

3. Calidad comparable a autor humano: 1 ☒ 2 ☐ 3 ☐ 4 ☐ 5 ☐
4. Estructura de capítulos y subtramas: 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5 ☐
5. Revisión humana final imprescindible: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☒
6. Recomendar IA a principiantes (como apoyo): 1 ☐ 2 ☐ 3 ☒ 4 ☐ 5 ☐

Comentarios/opinión libre:

Puede ayudar a generar una idea general o a crear escenarios y personajes pero no es fiable para llevar a cabo el desarrollo completo de la historia.

C) Observaciones finales sobre la aplicación**Lo mejor:**

Los botones hacen que sea intuitivo. Es fácil de usar.

Lo peor:**Sugerencias de mejora:**

No aparecen todas las secciones desarrolladas en la aplicación en el pdf de la historia. Sería interesante tener la descripción de personajes, escenarios y el mundo, especialmente cuando en la historia no se describe a los personajes.

Reflexión final sobre la experiencia:

Falta mejorar algunos aspectos pero en términos generales es una herramienta muy interesante con posibilidades de futuro.

Checklist de evaluación: usabilidad e historia generada

Rango de Edad: menor de 15 ☐ 15 – 25 ☐ 26 – 40 ☐ 41 – 60 ☒ más de 60 ☐
Modo usado: Rápido ☒ Detallado ☐

A) Evaluación de la historia generada

(1 = muy bajo · 5 = excelente)

1. Coherencia global de la trama: 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5 ☐
2. Consistencia entre capítulos: 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5 ☐
3. Desarrollo de personajes: 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5 ☐
4. Decisiones y conflictos: 1 ☒ 2 ☐ 3 ☐ 4 ☐ 5 ☐
5. Subtramas: 1 ☒ 2 ☐ 3 ☐ 4 ☐ 5 ☐
6. Transiciones entre escenas/capítulos: 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5 ☐
7. Ritmo de la historia: 1 ☐ 2 ☐ 3 ☒ 4 ☐ 5 ☐
8. Originalidad/interés de la historia: 1 ☐ 2 ☐ 3 ☒ 4 ☐ 5 ☐
9. Calidad del lenguaje: 1 ☒ 2 ☐ 3 ☐ 4 ☐ 5 ☐
10. Mundo/ambientación: 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5 ☐
11. Clímax y desenlace: 1 ☒ 2 ☐ 3 ☐ 4 ☐ 5 ☐
12. Sesgos/estereotipos: 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5 ☐
13. Impacto emocional: 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5 ☐

Comentarios sobre la historia (puntos fuertes, cabos sueltos, sugerencias):

Se le olvido que hacia calor (una de las condiciones de la historia)

Repite las mismas palabras todo el rato.

No es muy "razonable" en algunas cosas, no posible, aunque la historia era contemporanea y realista.

La historia es entretenida y si quieres una idea para escribir tu propia historia, da buenas ideas.

B) Opinión: escritura tradicional e IA

(1 = muy en desacuerdo · 5 = muy de acuerdo)

1. Ayuda en bloqueos creativos: 1 ☐ 2 ☐ 3 ☐ 4 ☒ 5 ☐
2. IA como apoyo (no sustituto): 1 ☐ 2 ☐ 3 ☒ 4 ☐ 5 ☐

3. Calidad comparable a autor humano: 1 ☒ 2 ☐ 3 ☐ 4 ☐ 5 ☐
4. Estructura de capítulos y subtramas: 1 ☒ 2 ☐ 3 ☐ 4 ☐ 5 ☐
5. Revisión humana final imprescindible: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☒
6. Recomendar IA a principiantes (como apoyo): 1 ☐ 2 ☒ 3 ☐ 4 ☐ 5 ☐

Comentarios/opinión libre:

C) Observaciones finales sobre la aplicación

Lo mejor:

Puede dar ideas interesantes para crear tu propia historia (una ayuda para empezar).

Lo peor:

No tiene mucho vocabulario, no crea muchos personajes secundarios, no describe los personajes en profundidad. No da una final real a la historia.

Sugerencias de mejora:

Aumentar MUCHO el vocabulario, mas trama. Aunque sea una historia corta, que tenga desenlace.

Reflexión final sobre la experiencia:

Ha sido divertido. He podido crear más historias y puede ofrecer ideas interesantes.

Checklist de evaluación: usabilidad e historia generada

Rango de Edad: menor de 15 ☐ 15 – 25 ☐ 26 – 40 ☐ 41 – 60 ☐ más de 60 ☐
Modo usado: Rápido ☐ Detallado ☐

A) Evaluación de la historia generada

(1 = muy bajo · 5 = excelente)

1. Coherencia global de la trama: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
2. Consistencia entre capítulos: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
3. Desarrollo de personajes: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
4. Decisiones y conflictos: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
5. Subtramas: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
6. Transiciones entre escenas/capítulos: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
7. Ritmo de la historia: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
8. Originalidad/interés de la historia: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
9. Calidad del lenguaje: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
10. Mundo/ambientación: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
11. Clímax y desenlace: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
12. Sesgos/estereotipos: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
13. Impacto emocional: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Comentarios sobre la historia (puntos fuertes, cabos sueltos, sugerencias):

B) Opinión: escritura tradicional e IA

(1 = muy en desacuerdo · 5 = muy de acuerdo)

1. Ayuda en bloqueos creativos: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
2. IA como apoyo (no sustituto): 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

3. Calidad comparable a autor humano: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
4. Estructura de capítulos y subtramas: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
5. Revisión humana final imprescindible: 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐
6. Recomendar IA a principiantes (como apoyo): 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Comentarios/opinión libre:

C) Observaciones finales sobre la aplicación

Lo mejor:

Lo peor:

Sugerencias de mejora:

Reflexión final sobre la experiencia:

Introduction

In recent years, the way of tackling problems, searching for information, or generating ideas has been changing rapidly. The current boom in generative artificial intelligence has fully broken into most sectors of society. The ease of access to these tools, together with the quality of the results in generating texts, summaries, images, videos, or in automating tedious tasks, has led to the widespread use of these technologies even in everyday activities. The use, often excessive, of artificial intelligence models in creative fields has generated rejection in part of society, especially in sectors such as illustration, translation, or musical composition, where there is concern about the loss of human value and employment. In the case of *story generation* something similar occurs: many writers and readers question the artistic value of a text produced by a machine, considering that it lacks the emotional depth and life experience that a human author provides. In addition, it is important to highlight the biases arising from design errors or present in the training data, which can lead to reproducing stereotypes or forms of discrimination. Despite this, creative writing generated by LLMs (*Large Language Models*) can also be understood as an opportunity, not to replace writers or novelists, but as support for creation and imagination. In this context the project is presented, which aims to serve as a guide and support for authors who want to write stories, offering a structured and editable workflow. In addition, an analysis of the generated stories is proposed in order to evaluate their narrative quality and check the extent to which they can be considered comparable to those created by a human author. This evaluation does not seek to replace the writer, but to assess the potential of artificial intelligence as a creative tool, as well as to identify its current limitations in terms of originality, coherence, and literary depth.

Motivation

There are currently multiple websites and applications that allow stories to be generated automatically using artificial intelligence (Summarizer [24], Wrizzle [29]) that can create short stories from a topic and a small configuration. However, in most

cases personalization by the user is minimal, without offering real control over the creative process. This results in little or no editing where the user can only restart the entire generation. Faced with these limited options, the main motivation of this project is to develop a tool that not only generates a complete story, but also offers a high level of editing and personalization. In this way, to help and guide users with their own stories through a high level of detail. To this end, the narrative is divided into key sections: the *plot and symbolic thread* that sets the general course of the work, the *main characters* where their attributes and description are defined, the *main settings*, and the *structure of the chapters*. The objective is to build the story through sections where the user can rewrite, request modifications, or manually edit each one so that it fits their preferences. In addition, there is a quick generation mode intended to generate the story from the data entered in a questionnaire, creating stories much more quickly, sacrificing part of the editing and personalization characteristic of the other option. This gives rise to another analysis: to evaluate whether the quality and coherence of the generated story vary depending on the creation mode (see Section 3.2.1 for more details).

Objectives

The main objective of this TFG is to develop an application that guides the creation of stories using language models, allowing the structured generation of stories and offering a high degree of editing and personalization to the user. The specific objectives proposed by the project are:

- Implement an LLM (the **Groq** server is used) that enables story generation, ensuring good *token* management so as not to exceed the model's usage limit.
- Design an interactive user interface that allows the user to generate, rewrite, modify, and manually edit each section of the story for a better fit to their preferences.
- Design the narrative flow by dividing the story into key sections (plot, characters, settings, world, chapters) that facilitate editing and personalization.
- Store the generated sections in a database (**MongoDB**) to manage data persistence and allow the user to resume the story at any time.
- Offer two modes of story generation: a *detailed* one and a *quick* one, to adapt to the user's needs and to compare the quality and coherence of the stories generated depending on the creation mode.
- Evaluate the narrative quality of the generated stories, analyzing their coherence and literary depth in comparison with those created by a human author.
- Identify the current limitations of LLMs in story generation, proposing lines of future improvements.

Structure of the report

The project is divided into several chapters that address different aspects of the development and evaluation of the tool:

- **Chapter 1: Introduction:** Presents the project's context, motivation, and objectives.
- **Chapter 2: State of the question:** Previous work and the main lines of research related to automatic story generation are reviewed. The evolution of AI from Alan Turing to the present day and the technical foundations of LLMs are presented. In addition, several published novels written by artificial intelligence are discussed.
- **Chapter 3: Description of the work:** Presents the main chapter of the project where everything related to the application is developed. The chapter begins with a section devoted to presenting the main resources and tools used during the project: `Groq`, `Flet` and `MongoDB`. We continue with Section 3.2, where the two story-creation models, flowcharts, and component diagrams are presented in detail. A section is included that details storage in the database and finally reference is made to the importance of *system prompts* to guide the model, and the most relevant ones are shown. To finish this section, the screens of the application and the operation of `Flet` are detailed.
- **Chapter 4: Development and evaluation of generated stories:** In this chapter an example of creating a story in *detailed mode* is shown and its subsequent evaluation through surveys.
- **Chapter 5: Conclusions and future work:** Presents a final reflection on narrative generation by LLMs, the improvements introduced through section-based creation, and finally a set of lines for future work.

Conclusion and Future Work

This work presents **historIAs**, an application designed to generate stories in sections, leveraging LLMs to achieve a higher degree of personalization and user editing both before and during creation. The application stores information in a database so that users can resume a story at any time. There is also a *quick mode* in which, after completing a short questionnaire, the model directly generates the story using the information provided.

After conducting numerous tests, it was observed that the generated stories frequently exhibited a series of similarities and biases. When the themes were similar, the model repeated character names and produced similar descriptions. If the *prompt* was not very detailed, it would introduce magical elements at any point and almost always relied on the trope of the protagonist having to defeat the villain. Coherence was maintained in most cases, although there were instances where, for example, in the *setting description* section, it created a setting that ultimately did not appear in the story. Ultimately, it was observed that these models, despite increasingly maintaining the coherence and thread of the story, are still far inferior to the writing ability of a professional author. Linguistic richness is more limited, often repeating words, expressions, or even whole sentences. The stories are quite simple in quality; there are no twists, and they follow a linear structure, which fails to evoke strong emotion in the reader. In the study by Tian, Huang, et al. [25], Figure D.1 shows the evolution of the characters' fortune:

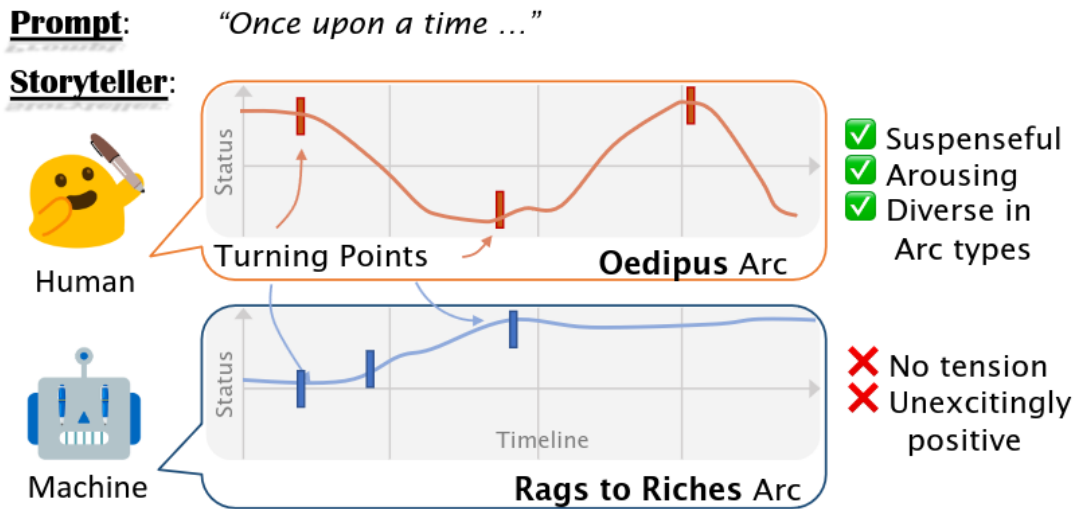


Figura D.1: Comparison of narrative arcs between humans and language models (chart source [25])

This graph presents a comparison between the narrative arcs and the positions of turning points in human- and LLM-generated narratives. The vertical axis shows a character’s fortune (from bad to good), and the horizontal axis represents the story’s timeline. Compared with traditional writing, LLMs tend to employ happier and less complex arcs, introduce turning points earlier, and include less suspense and fewer setbacks in their narratives. The same trend is apparent in the stories created with this application.

Another important goal of the project was to compare story generation in *quick mode*—more similar to what is currently on the market—with the project’s proposal, the *detailed mode*. After numerous tests, coherence levels were similar, although the *detailed mode* did produce longer and more developed stories. One of the most notable features of the project has been the section-by-section creation workflow. This structure not only facilitates personalization and narrative control for the user, but has also proven to be a very useful tool for generating ideas and stimulating creativity. Although the literary quality of the model outputs still shows limitations, the step-by-step process of building a story is dynamic, enjoyable, and enriching, according to users who tested the application. In this sense, the application serves not only a generative function but also an inspirational one, acting as a starting point for writers to develop deeper and more original narratives from the ideas produced by the system.

For this reason, continuing with the section-based approach is a promising line of work. To that end, the author could be given more flexibility to choose which sections to create, rather than being limited to the five predefined ones. Another interesting direction is to provide greater flexibility to define a series of events or actions that the user wants to occur during the story, as well as to allow modifications to sections or chapters the user has already completed. The greatest challenge throughout the project was the model’s token limit imposed by Groq; a larger token

budget would allow for more sections and chapters. Another improvement would be to let the user introduce new characters; due to the token limit, the system is currently restricted to a maximum of four characters. It would also be valuable to include an initial configuration where aspects such as type of novel, target audience, narrative style, etc. are specified. Finally, the system prompts could be improved by applying prompting techniques.

Bibliografía

- [1] Microsoft Azure. Microsoft azure, n.d. URL <https://azure.microsoft.com/es-es>.
- [2] Google Colab. Google colab, n.d. URL <https://colab.research.google.com/>.
- [3] Eugenio Pablo Concepción Cuevas. *Un modelo de generación automática de historias con múltiples tramas*. Tesis Doctoral, Universidad Complutense de Madrid, 2023. URL <https://hdl.handle.net/20.500.14352/102817>.
- [4] Natalie Dehn. Memory in story invention. pages 213–215, 1981.
- [5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL <https://arxiv.org/abs/1810.04805>.
- [6] Hugging Face. Hugging face, n.d. URL <https://huggingface.co/>.
- [7] Flet. Flet, n.d. URL <https://flet.dev/>.
- [8] R. Goodwin. *1 the road*. Jean Boîte Editions, 2018. ISBN 978-2365680271.
- [9] Groq. Groq, n.d. URL <https://groq.com/>.
- [10] D. Guisado Morcillo and ChatGPT. *Iris*. David Guisado Morcillo, 2023. ISBN 978-8409477142.
- [11] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. In *Proc. of the South African Forum for Artificial Intelligence Research (FAIR). Extended abstracts*, Cape Town, South Africa, December 2019. doi:10.48550/arXiv.1904.09751. URL <https://ceur-ws.org/Vol-2540/>.
- [12] Sheldon Klein, John F. Aeschlimann, David F. Balsiger, Steven L. Converse, Claudine Court, Mark Foster, Robin Lao, John D. Oakley, and Joel Smith. Automatic novel writing: A status report. Technical Report Technical Report #186, University of Wisconsin-Madison, Computer Sciences Department, Madison, WI, July 1973.

- [13] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. doi:10.1038/nature14539. URL <https://hal.science/hal-04206682>.
- [14] Llama3.3. Llama3.3, n.d. URL https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_3/.
- [15] James R. Meehan. Tale-spin, an interactive program that writes stories. pages 91–98, 1977.
- [16] MongoDB. Mongoddb, n.d. URL <https://www.mongodb.com/>.
- [17] J. Moor. The dartmouth college artificial intelligence conference: The next fifty years. *AI Magazine*, 27:87–91, 2006.
- [18] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *ACM Trans. Intell. Syst. Technol.*, 16(5), August 2025. ISSN 2157-6904. doi:10.1145/3744746.
- [19] A. Newell and H. Simon. The logic theory machine—a complex information processing system. *IRE Transactions on Information Theory*, 2(3):61–79, 1956. doi:10.1109/TIT.1956.1056797.
- [20] A. Newell, J. C. Shaw, and H. A. Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pages 256–264, 1959. URL http://findingaids.library.cmu.edu/repositories/2/archival_objects/22561.
- [21] Racter. *The Policeman’s Beard is Half Constructed*. Grand Central Pub, 1984. ISBN 978-0446380515.
- [22] M. Renze and E. Guven. The effect of sampling temperature on problem solving in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7346–7356, 2024. doi:10.48550/arXiv.2402.05201.
- [23] Amazon Web Service. Amazon web service, n.d. URL <https://aws.amazon.com/es/>.
- [24] Summarizer. Summarizer, n.d. URL <https://www.summarizer.org/>.
- [25] Y. Tian, T. Huang, M. Liu, D. Jiang, A. Spangher, M. Chen, J. May, and N. Peng. Are large language models capable of generating human-level narratives? *arXiv*, 2024. doi:10.48550/arXiv.2407.13248.
- [26] A. M. Turing. Computing machinery and intelligence. *Mind*, LIX(236):433–460, 1950. doi:10.1093/mind/LIX.236.433.
- [27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. doi:10.48550/arXiv.1706.03762.

-
- [28] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2022. URL <https://arxiv.org/abs/2109.01652>.
 - [29] Wrizzle. Wrizzle, n.d. URL <https://www.wrizzle.ai/es>.
 - [30] Lili Yao, Nanyun Peng, Ralph Weischedel, Kevin Knight, Dongyan Zhao, and Rui Yan. Plan-and-write: Towards better automatic storytelling. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7378–7385, Jul. 2019. doi:10.1609/aaai.v33i01.33017378.