
Generación de diálogos para historias utilizando
Large Language Models
Dialog generation for stories using Large
Language Models



Trabajo de Fin de Grado
Curso 2024–2025

Autor

Pablo Enrique Padial Iniesta

Director

**Pablo Gervás Gómez-Navarro
Gonzalo Méndez Pozo**

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

Generación de diálogos para historias
utilizando Large Language Models
Dialog generation for stories using Large
Language Models

Trabajo de Fin de Grado en **Ingeniería de Computadores**

Autor

Pablo Enrique Padial Iniesta

Director

Pablo Gervás Gómez-Navarro

Gonzalo Méndez Pozo

Convocatoria: *Septiembre 2025*

Grado en Ingeniería de Computadores

Facultad de Informática

Universidad Complutense de Madrid

DIA de MES de AÑO

Dedicatoria

*A Pedro Pablo y Marco Antonio, por crear
TeXiS e iluminar nuestro camino*

Agradecimientos

A Guillermo, por el tiempo empleado en hacer estas plantillas. A Adrián, Enrique y Nacho, por sus comentarios para mejorar lo que hicimos. Y a Narciso, a quien no le ha hecho falta el Anillo Único para coordinarnos a todos.

Resumen

Generación de diálogos para historias utilizando Large Language Models

Un resumen en castellano de media página, incluyendo el título en castellano. A continuación, se escribirá una lista de no más de 10 palabras clave.

Palabras clave

Máximo 10 palabras clave separadas por comas

Abstract

Dialog generation for stories using Large Language Models

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

Keywords

10 keywords max., separated by commas.

Índice

1. Introduction	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Estructura de la memoria	3
2. Estado de la Cuestión	5
3. Descripción del Trabajo	7
3.1. Herramientas utilizadas	7
3.2. Diseño de la aplicación	9
3.2.1. Modos de generación de historia	9
3.2.2. Diagramas de flujo	9
3.2.3. Diagrama de componentes	12
3.2.4. Interacción con el modelo LLM	13
3.2.5. Almacenamiento en la base de datos	16
3.2.6. Gestión de prompts	18
3.3. Interfaz de usuario	19
3.3.1. Pantallas y funcionalidades	19
3.3.2. Arquitectura de la interfaz	21
4. Conclusiones y Trabajo Futuro	23
Contribuciones Personales	25
A. Título del Apéndice A	27
B. Título del Apéndice B	29

Índice de figuras

3.1. Diagrama de flujo <i>Inicio de sesión</i>	10
3.2. Diagrama de flujo <i>Flujo principal</i>	11
3.3. Diagrama de componentes	12
3.4. Diagrama de secuencia	14
3.5. Almacenamiento en la base de datos: <i>info_usuarios</i>	17
3.6. Almacenamiento en la base de datos para el <i>modo detallado: contexto</i>	17
3.7. Almacenamiento en la base de datos para el <i>modo detallado: sección</i>	18

Índice de tablas

3.1. Límites de uso de los modelos empleados.	8
3.2. Ajuste de <code>temperature</code> según el tipo de tarea	16

Introduction

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 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 LLM pueden entenderse también como una oportunidad, no para sustituir a escritores o novelistas, si no como apoyo a la hora de creación e imaginación. En este ámbito se presenta el proyecto, que pretende servir como guía y apoyo a autores que quieran escribir historias, ofreciendo un flujo estructurado y editable. Además, se plantea un análisis de las historias generadas con el fin de evaluar su calidad narrativa y comprobar hasta qué punto pueden considerarse comparables a las creadas por un autor humano. Esta evaluación no busca sustituir al escritor, sino valorar el potencial de la inteligencia artificial como herramienta creativa, así como identificar sus limitaciones actuales en cuanto a originalidad, coherencia y profundidad literaria.

1.1. Motivación

Actualmente existen múltiples páginas web y aplicaciones que permiten generar historias de manera automática mediante inteligencia artificial (Summarizer, <https://www.summarizer.org/>, wrizzle, <https://www.wrizzle.ai/es>) que pueden crear pequeños relatos a partir de un tema y una pequeña configuración. Sin em-

bargo, en la mayoría de casos 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 esta forma, ayudar y guiar a los usuarios con sus propias historias a través de un gran nivel de detalle. Para ello, la narrativa se divide en secciones clave: la *trama e hilo simbólico* que marca el rumbo general de la obra, los *personajes principales* donde se define sus atributos y descripción, los *escenarios principales*, 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. A parte, 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 se plantea otro análisis: evaluar si la calidad y coherencia de la historia generada varían en función del modo de creación (ver la sección 3.2.1 para más detalles).

1.2. Objetivos

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

- Implementar un LLM a través de **Groq** que permita la generación de historias, asegurando una buena gestión de *tokens* para no superar el límite de uso del modelo.
- Diseñar una interfaz de usuario interactiva que permita al usuario generar, reescribir, modificar y editar manualmente cada sección de la historia para un mayor ajuste a sus preferencias.
- Diseñar el flujo narrativo dividiendo la historia en secciones clave (trama, personajes, escenarios, mundo, capítulos) que faciliten la edición y personalización.
- Almacenar las secciones generadas en **MongoDB** para gestionar la persistencia de los datos y permitir que el usuario pueda retomar la historia en cualquier momento.
- Ofrecer dos modos de generación de historias: uno detallado y otro rápido, para adaptarse a las necesidades del usuario y comparar la calidad y coherencia de las historias generadas en función del modo de creación.

- Evaluar la calidad narrativa de las historias generadas, analizando su coherencia y profundidad literaria en comparación con las creadas por un autor humano.
- Identificar las limitaciones actuales de los LLM 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. (Continuar....)
- **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, para detallar mejor el funcionamiento interno de la aplicación, 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*.

Capítulo 2

Estado de la Cuestión

En el estado de la cuestión es donde aparecen gran parte de las referencias bibliográficas del trabajo. Una de las formas más cómodas de gestionar la bibliografía en L^AT_EX es utilizando **bibtex**. Las entradas bibliográficas deben estar en un fichero con extensión *.bib* (con esta plantilla se proporciona el fichero *biblio.bib*, donde están las entradas referenciadas más abajo). Cada entrada bibliográfica tiene una clave que permite referenciarla desde cualquier parte del texto con los siguiente comandos:

- Referencia bibliografica con cite: ?
- Referencia bibliográfica con citep: (?)
- Referencia bibliográfica con citet: ?

Es posible citar más de una fuente, como por ejemplo (???)

Después, L^AT_EX se ocupa de rellenar la sección de bibliografía con las entradas **que hayan sido citadas** (es decir, no con todas las entradas que hay en el *.bib*, sino sólo con aquellas que se hayan citado en alguna parte del texto).

Bibtex es un programa separado de latex, pdf_latex o cualquier otra cosa que se use para compilar los *.tex*, de manera que para que se rellene correctamente la sección de bibliografía es necesario compilar primero el trabajo (a veces es necesario compilarlo dos veces), compilar después con bibtex, y volver a compilar otra vez el trabajo (de nuevo, puede ser necesario compilarlo dos veces).

Descripción del Trabajo

3.1. Herramientas utilizadas

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

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

Tabla 3.1: Límites de uso de los modelos empleados.

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**, lo que permite desarrollar aplicaciones web, de escritorio y móviles usando únicamente Python. A diferencia de otras bibliotecas, **Flet** abstraer la complejidad del **frontend**, simplificando las complejidades y proporcionando componentes listos para usar sin necesidad de tener conocimientos previos en **frontend**.

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

- *MongoDB*: En el proyecto se ha utilizado **MongoDB** como sistema de gestión de base de datos. 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 cambiantes y jerárquicas. Un aspecto clave es que el desarrollador trabaja siempre con documentos en formato JSON. Cuando se inserta un documento, **MongoDB** se encarga automáticamente de convertirlo a BSON para almacenarlo de manera eficiente. De igual modo, al recuperar la información, **MongoDB** transforma el BSON de vuelta a JSON para que pueda manipularse de forma sencilla en la aplicación.

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

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

3.2. Diseño de la aplicación

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

3.2.1. Modos de generación de historia

La 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 enfoca a tener una mayor personalización de la historia por parte del usuario. El proceso de creación se divide en varias secciones predeterminadas, tales como trama e hilo simbólico, descripción del mundo, personajes principales, desarrollo de escenarios, 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 de forma manual sacrificando parte de la personalización en favor de la velocidad y simplicidad.

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

3.2.2. Diagramas de flujo

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

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

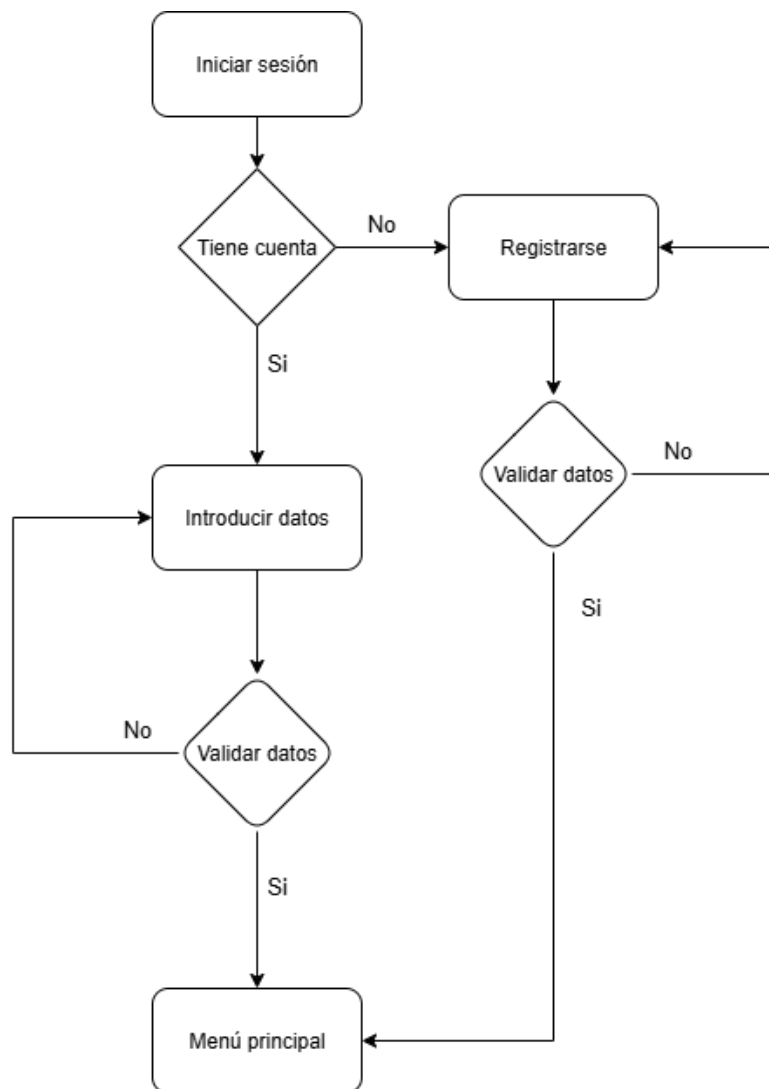
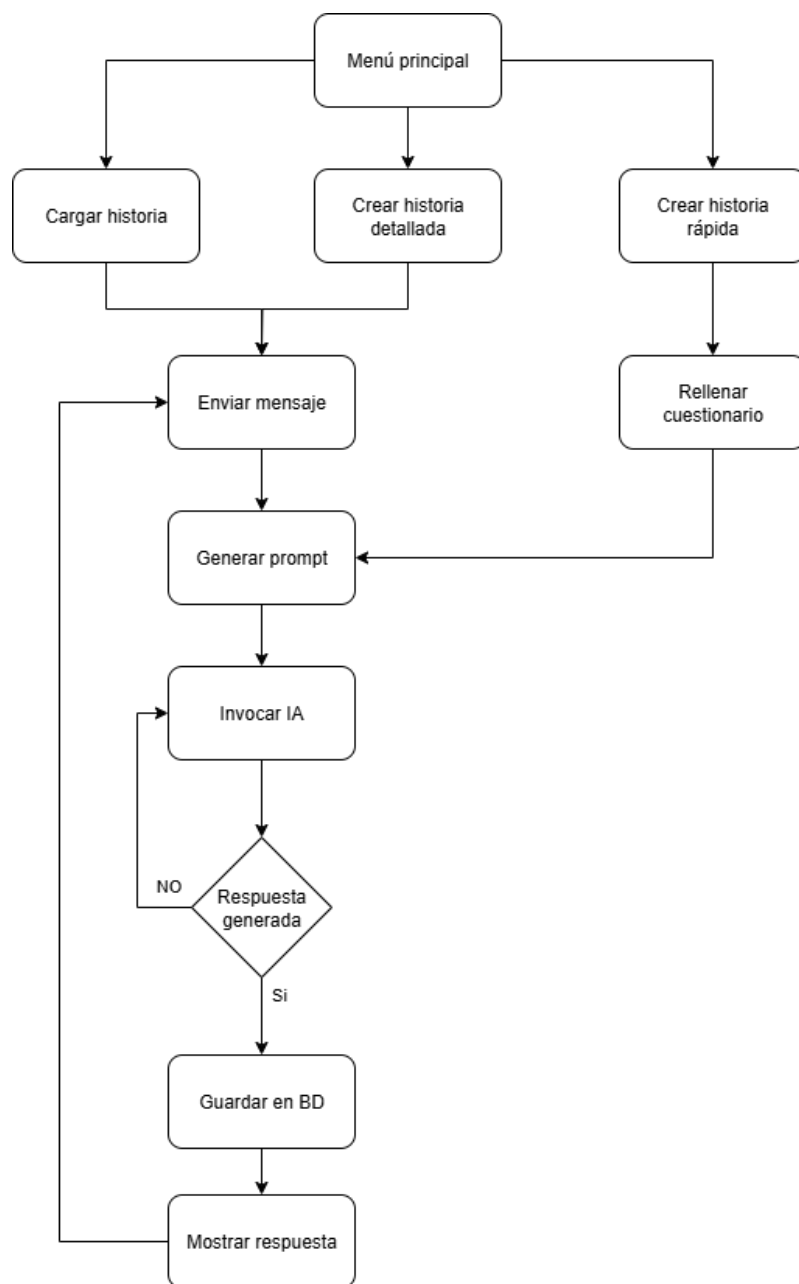


Figura 3.1: Diagrama de flujo *Inicio de sesión*

- *Flujo principal*: **revisar el punto** 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) manteniendo varias iniciadas al mismo tiempo.

Figura 3.2: Diagrama de flujo *Flujo principal*

3.2.3. Diagrama de componentes

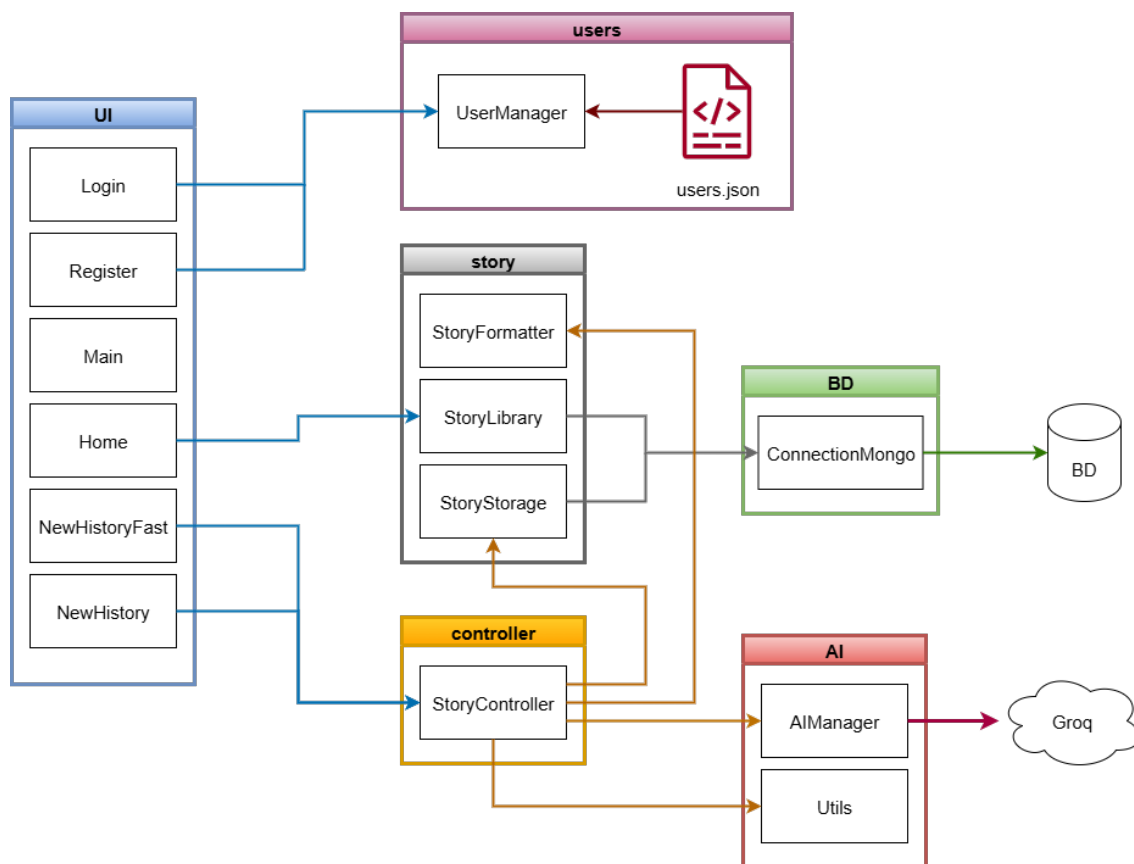


Figura 3.3: Diagrama de componentes

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

- *UI (interfaz de usuario)*: Se encuentran las clases que implementan **Flet** para crear la interfaz gráfica. Tenemos el **Main** que actúa como punto de arranque de la aplicación en **Flet** y se encarga de definir las rutas asociadas a cada pantalla. El resto de clases corresponden a las diferentes pantallas de la aplicación.
- *users*: La clase **UserManager** se encarga de la gestión de usuarios a la hora de realizar el **login** y el registro. Contiene un fichero **users.json** donde se almacena el nombre, correo y contraseña de cada usuario.
- *story*: Esta sección agrupa toda la lógica que hace posible la creación, formateo y almacenamiento de las novelas. **StoryFormatter** se encarga de todo lo relacionado con formatear el texto para convertirlo en un formato JSON que sea compatible con **MongoDB**. La clase **StoryLibrary** se encarga de gestionar las historias creadas por el usuario y **StoryStorage** se encarga de crear todo lo relacionado con la historia, como las secciones, resúmenes y almacenado en la base de datos.

- *controller*: Su función es separar la lógica de la interfaz y las llamadas al servidor **Groq**. Se encarga de llevar la lógica del procesamiento del mensaje, detectar la sección por la que se encuentra el usuario, utilizar los métodos creados en **StoryStorage** y realizar las llamadas a **AIManager** que devuelven la respuesta generada por el modelo.
- *BD (base de datos)*: Contiene la clase **ConnectionMongo** encargada de realizar las consultas en la base de datos.
- *AI*: Contiene **AIManager** encargado de establecer la conexión con el cliente **Groq** y realizar las llamadas al servidor. Además contiene un fichero **Utils** donde se almacenan los distintos **prompts** iniciales que se usarán para generar las respuestas.

3.2.4. Interacción con el modelo LLM

La interacción entre el LLM y el usuario es la parte principal de la funcionalidad de la aplicación. El sistema se apoya en la *API de Groq*, utilizando el modelo **deepseek-r1-distill-llama-70b**, para generar el contenido narrativo. Este contenido se genera siguiendo una serie de pasos que se van a explicar en esta sección. **(Explicar porque uso el modelo)** Cada vez que el usuario realiza una acción desde **NewHistory**, esta clase crea un **thread** para separar la parte de la interfaz de la petición al servidor y así evitar bloqueos en la parte de la interfaz. Mientras este *thread* está activo, bloquea temporalmente la posibilidad al usuario de 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 cambie es como se forman los mensajes.

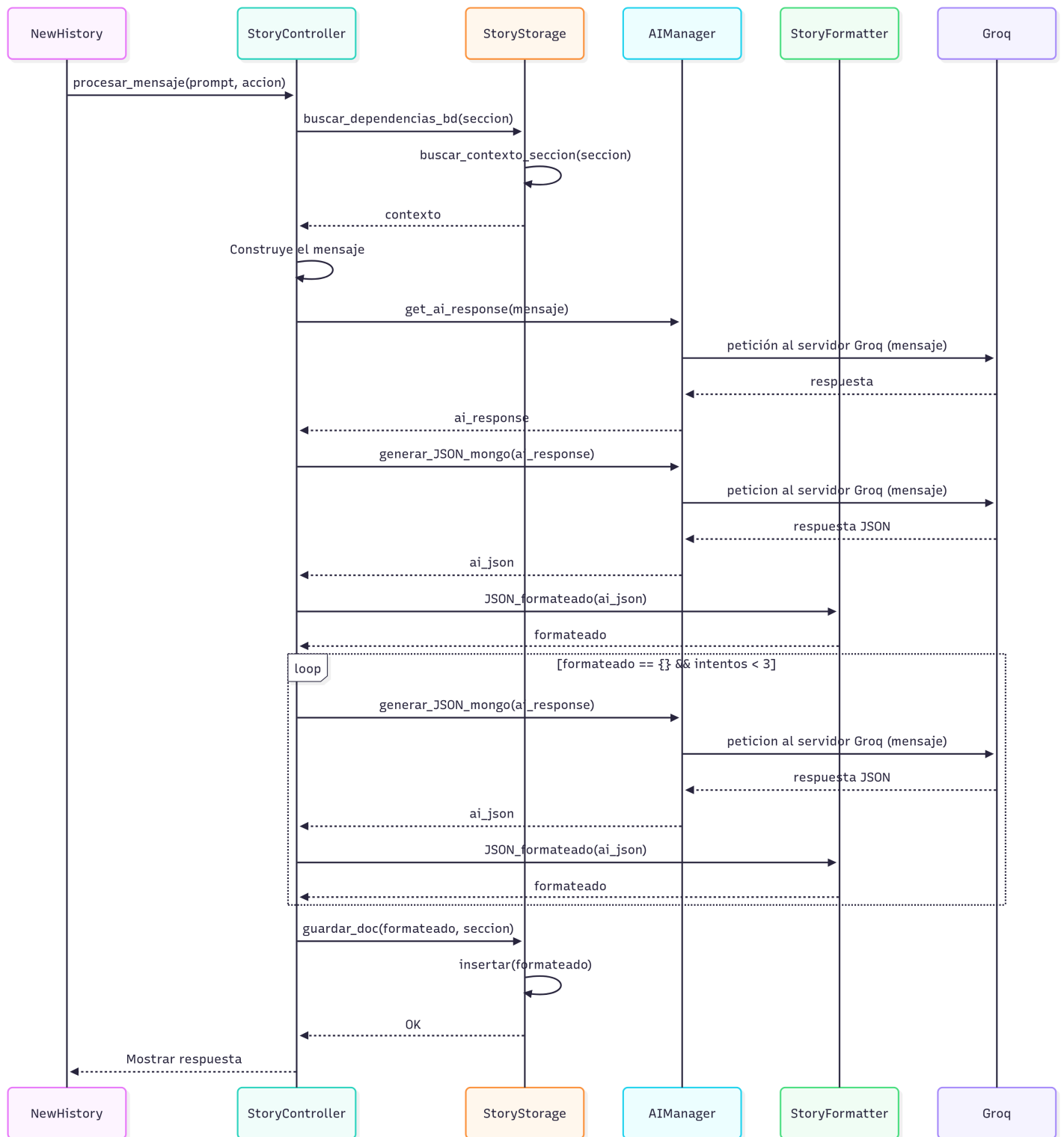


Figura 3.4: Diagrama de secuencia

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

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

Código 3.1: Llamada al cliente de Groq

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

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

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

```
return clean_ai_response(raw_response)
```

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

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

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

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

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

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

3.2.5. Almacenamiento en la base de datos

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

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

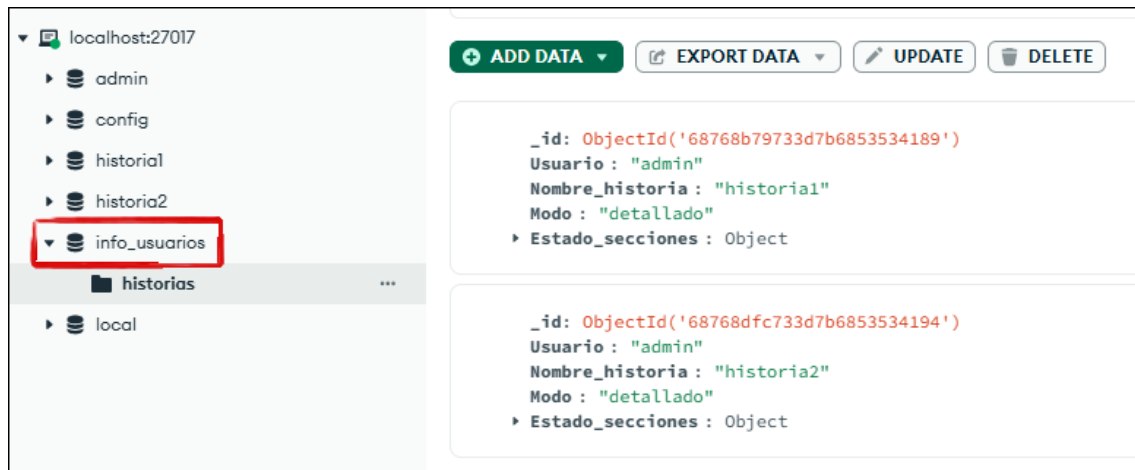


Figura 3.5: Almacenamiento en la base de datos: *info_usuarios*

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

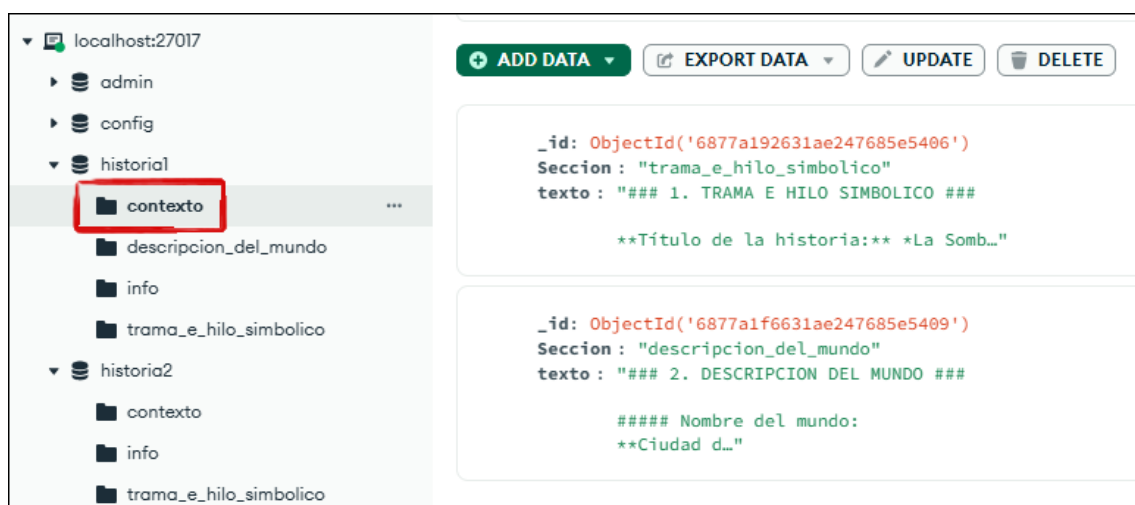


Figura 3.6: Almacenamiento en la base de datos para el *modo detallado*: *contexto*



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

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

3.2.6. Gestión de prompts

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

- **PROMPT_INICIAL**: Se trata del **prompt** principal para el modo detallado donde se especifica las reglas que tiene que seguir. A medida que se realizaban pruebas y se añadían nuevas funcionalidades este **prompt** también se iba actualizando. La parte más importante es la especificación de las secciones: *trama e hilo simbólico*, *descripción del mundo*, *personajes principales*, *descripción de escenarios* y *estructura de capítulos*. Cada una de las secciones contiene unas reglas determinadas y se proporciona un ejemplo de salida para lograr una mayor precisión en la respuesta. Tanto las secciones de personajes como escenarios, se aporta una plantilla con distintos campos para así facilitar la creación y guardado en la base de datos.
- **PROMPT_JSON**: Aunque se le asigne una serie de reglas y ejemplos a la IA no siempre genera la respuesta siguiendo el mismo formato. Esto dificultaba mucho a la hora de convertir el texto de la respuesta en formato JSON ya que se quería que se respetase cada campo generado y no fuese únicamente texto plano en un único campo. Finalmente se vio que la mejor forma de conseguirlo era que la IA convirtiese su respuesta a texto formato JSON y así únicamente se tuviese que pasar de texto a JSON. Para ello se crea otro *prompt del sistema*

al que se le da toda la información y reglas que necesita para convertir de el mensaje inicial en un *string* con un formato JSON válido. Se ha ido ajustando y perfeccionando el *prompt* porque inicialmente en muchas ocasiones podía cambiarte el contenido o te encapsulaba el JSON entre comillas o símbolos que producían error a la hora de convertir el texto.

- **PROMPT_ESCRITURA_RAPIDA:** Este es el **prompt** dedicado a al modo de escritura rápida donde se indica que debe de crear la historia a partir de la información introducida en el cuestionario.
- **PROMPT_CAPITULOS:** Tras desarrollar todas las secciones y pasar al momento de la escritura, se ha decidido crear un nuevo *prompt* más detallado de como escribir los capítulos. Así, además, sirve para gestionar mejor la limitación de *tokens* ya que si se incluyese en el mismo *prompt* este gastaría más *tokens* limitando así la respuesta. En este apartado se especifica que tiene que escribir los capítulos de uno en uno utilizando la información que se le pasa como contexto.

3.3. Interfaz de usuario

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

3.3.1. Pantallas y funcionalidades

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

3.3.1.1. Inicio de sesión

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

3.3.1.2. Registro de usuario

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

3.3.1.3. Página principal

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

- *Cargar y eliminar*: Cada usuario cuenta con el área *Mis historias* donde se visualiza su historial de novelas junto con el progreso alcanzado. Para cada una de las novelas, el usuario puede retomarla desde el punto en que la dejó o eliminarla si no desea conservarla.
- *Empezar nueva historia*: Existe la opción de iniciar una nueva historia ya sea en *modo detallado* o *modo rápido*. En ambos casos se pide al usuario introducir un nombre para guardar la historia en el historial.

3.3.1.4. Empezar historia detallada

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

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

3.3.1.5. Empezar historia rápida

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

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

3.3.2. Arquitectura de la interfaz

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

3.3.2.1. Modelo de UI en Flet

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

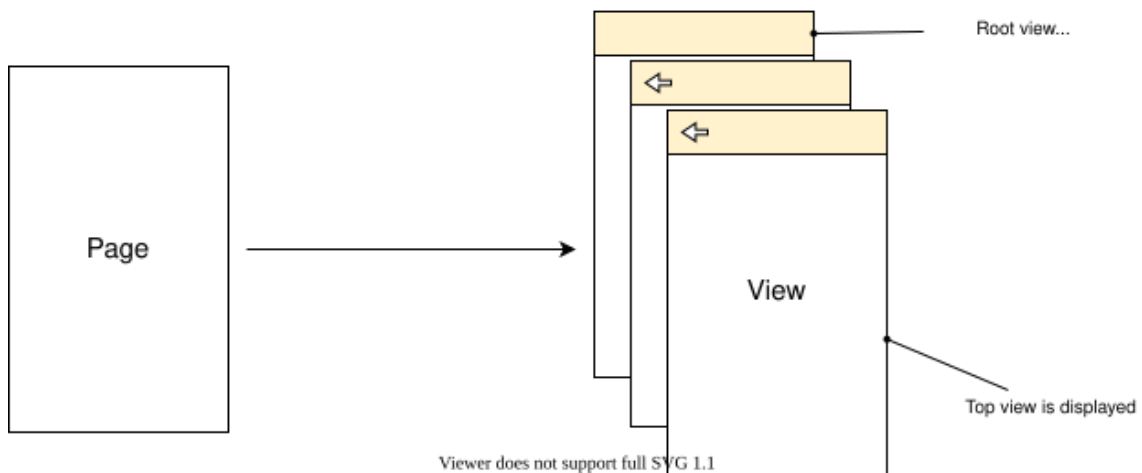
3.3.2.2. Navegacion con flet_route

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

`https://localhost/#/nueva_hisotria`

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

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



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

Código 3.2: Llamada al cliente de Groq

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

#Importamos las pantallas
from UI.Home import Home
from UI.Login import Login
```

```
from UI.Register import Register
from UI.NewHistory import NuevaHistoria
from UI.NewHistoryFast import NuevaHistoriaRapida
from UI.LoadHistory import CargarHistoria

def main(page:ft.Page):
    page.title="Creador de historias"
    app_routes = [
        path(url="/", clear=True, view=Login().view),
        path(url="/register", clear=True, view=Register().
            view),
        path(url="/home", clear=True, view=Home().view),
        path(url="/nueva_historia", clear=True, view=
            NuevaHistoria().view),
        path(url="/nueva_historia_rapida", clear=True, view=
            NuevaHistoriaRapida().view),
        path(url="/cargar_historia", clear=True, view=
            CargarHistoria().view)
    ]

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

ft.app(target=main)
```


Capítulo 4

Conclusiones y Trabajo Futuro

Conclusiones del trabajo y líneas de trabajo futuro.

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

Contribuciones Personales

En caso de trabajos no unipersonales, cada participante indicará en la memoria su contribución al proyecto con una extensión de al menos dos páginas por cada uno de los participantes.

En caso de trabajo unipersonal, elimina esta página en el fichero `TFGTeXiS.tex` (comenta o borra la línea `\include{Capitulos/ContribucionesPersonales}`).

Estudiante 1

Al menos dos páginas con las contribuciones del estudiante 1.

Estudiante 2

Al menos dos páginas con las contribuciones del estudiante 2. En caso de que haya más estudiantes, copia y pega una de estas secciones.

Apéndice A

Título del Apéndice A

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

Apéndice	B
----------	----------

Título del Apéndice B

Se pueden añadir los apéndices que se consideren oportunos.

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

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

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

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

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

