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

Colaborador

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

Colaborador

Convocatoria: *Febrero/Junio/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

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

"Frase célebre dicha por alguien inteligente"
— Autor

Según la normativa para Trabajos de Fin de Grado[1], la memoria incluirá una portada normalizada con la siguiente información: título en castellano, título en inglés, autores, profesor director, codirector si es el caso, curso académico e identificación de la asignatura (Trabajo de fin de grado del Grado en - nombre del grado correspondiente-, Facultad de Informática, Universidad Complutense de Madrid). Los datos referentes al título y director (y codirector en su caso) deben corresponder a los publicados en la página web de TFG.

La memoria debe incluir la descripción detallada de la propuesta hardware/software realizada y ha de contener:

a. un índice,

b. un resumen y una lista de no más de 10 palabras clave para su búsqueda bibliográfica, ambos en castellano e inglés,

c. una introducción con los antecedentes, objetivos y plan de trabajo,

d. resultados y discusión crítica y razonada de los mismos, con sus conclusiones,

e. bibliografía.

Para facilitar la escritura de la memoria siguiendo esta estructura, el estudiante podrá usar las plantillas en LaTeX o Word preparadas al efecto y publicadas en la página web de TFG.

La memoria constará de un mínimo de 25 páginas para los proyectos realizados por un único estudiante, y de al menos 5 páginas más por cada integrante adicional del grupo. En este número de páginas solo se tiene en cuenta el contenido correspondiente a los apartados c y d del punto anterior.

La memoria puede estar escrita en castellano o inglés, pero en el primer caso la introducción y las conclusiones deben aparecer también en inglés. Las memorias de

[1] `https://informatica.ucm.es/file/normativatfg-2021-2022?ver` (ver versión actualizada para cada curso académico)

los TFG matriculados en el grupo I deberán estar escritas íntegramente en inglés, excepto por lo especificado en los puntos 1 y 2 anteriores (título, resumen y lista de palabras clave).

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.

Todo el material no original, ya sea texto o figuras, deberá ser convenientemente citado y referenciado. En el caso de material complementario se deben respetar las licencias y *copyrights* asociados al software y hardware que se emplee. En caso contrario no se autorizará la defensa, sin menoscabo de otras acciones que correspondan.

1.1. Motivación

Introducción al tema del TFG.

1.2. Objetivos

Descripción de los objetivos del trabajo.

1.3. Plan de trabajo

Aquí se describe el plan de trabajo a seguir para la consecución de los objetivos descritos en el apartado anterior.

1.4. Explicaciones adicionales sobre el uso de esta plantilla

Si quieres cambiar el **estilo del título** de los capítulos del documento, edita el fichero `TeXiS\TeXiS_pream.tex` y comenta la línea `\usepackage[Lenny]{fncychap}` para dejar el estilo básico de LaTeX.

Si no te gusta que no haya **espacios entre párrafos** y quieres dejar un pequeño espacio en blanco, no metas saltos de línea (\\) al final de los párrafos. En su lugar, busca el comando `\setlength{\parskip}{0.2ex}` en `TeXiS\TeXiS_pream.tex` y aumenta el valor de 0,2*ex* a, por ejemplo, 1*ex*.

TFGTeXiS se ha elaborado a partir de la plantilla de TeXiS[2], creada por Marco Antonio y Pedro Pablo Gómez Martín para escribir su tesis doctoral. Para explicaciones más extensas y detalladas sobre cómo usar esta plantilla, recomendamos la lectura del documento `TeXiS-Manual-1.0.pdf` que acompaña a esta plantilla.

El siguiente texto se genera con el comando `\lipsum[2-20]` que viene a continuación en el fichero .tex. El único propósito es mostrar el aspecto de las páginas

[2] `http://gaia.fdi.ucm.es/research/texis/`

usando esta plantilla. Quita este comando y, si quieres, comenta o elimina el paquete *lipsum* al final de `TeXiS\TeXiS_pream.tex`

1.4.1. Texto de prueba

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta

tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis

congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.

Curabitur tellus magna, porttitor a, commodo a, commodo in, tortor. Donec interdum. Praesent scelerisque. Maecenas posuere sodales odio. Vivamus metus lacus, varius quis, imperdiet quis, rhoncus a, turpis. Etiam ligula arcu, elementum a, venenatis quis, sollicitudin sed, metus. Donec nunc pede, tincidunt in, venenatis vitae, faucibus vel, nibh. Pellentesque wisi. Nullam malesuada. Morbi ut tellus ut pede tincidunt porta. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam congue neque id dolor.

Donec et nisl at wisi luctus bibendum. Nam interdum tellus ac libero. Sed sem justo, laoreet vitae, fringilla at, adipiscing ut, nibh. Maecenas non sem quis tortor eleifend fermentum. Etiam id tortor ac mauris porta vulputate. Integer porta neque vitae massa. Maecenas tempus libero a libero posuere dictum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aenean quis mauris sed elit commodo placerat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus rhoncus tincidunt libero. Etiam elementum pretium justo. Vivamus est. Morbi a tellus eget pede tristique commodo. Nulla nisl. Vestibulum sed nisl eu sapien cursus rutrum.

Nulla non mauris vitae wisi posuere convallis. Sed eu nulla nec eros scelerisque

pharetra. Nullam varius. Etiam dignissim elementum metus. Vestibulum faucibus, metus sit amet mattis rhoncus, sapien dui laoreet odio, nec ultricies nibh augue a enim. Fusce in ligula. Quisque at magna et nulla commodo consequat. Proin accumsan imperdiet sem. Nunc porta. Donec feugiat mi at justo. Phasellus facilisis ipsum quis ante. In ac elit eget ipsum pharetra faucibus. Maecenas viverra nulla in massa.

Nulla ac nisl. Nullam urna nulla, ullamcorper in, interdum sit amet, gravida ut, risus. Aenean ac enim. In luctus. Phasellus eu quam vitae turpis viverra pellentesque. Duis feugiat felis ut enim. Phasellus pharetra, sem id porttitor sodales, magna nunc aliquet nibh, nec blandit nisl mauris at pede. Suspendisse risus risus, lobortis eget, semper at, imperdiet sit amet, quam. Quisque scelerisque dapibus nibh. Nam enim. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nunc ut metus. Ut metus justo, auctor at, ultrices eu, sagittis ut, purus. Aliquam aliquam.

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 LaTeX 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, LaTeXse 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, pdflatex 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).

Capítulo 3

Descripción del Trabajo

3.1. Flujo detallado

Este apartado describe de forma detallada, paso a paso, el flujo de ejecución desde que un usuario inicia la aplicación hasta que envía un mensaje en el *chat*.

3.1.1. Ejecución inicial

El usuario inicia la aplicación ejecutando el comando `python start.py` y lanza el main principal `ft.app(target=main, view=ft.AppView.FLET_APP)`. *Flet* crea la ventana raíz ("/") y llama a `UI/Main.py` que es la encargada de gestionar todas las rutas de la aplicación usando `flet-route`.

Código 3.1: Inicio de la aplicación

```
if __name__ == "__main__":
    ft.app(target=main, view=ft.AppView.FLET_APP)
```

3.1.2. Autentificación

Nada más iniciar la aplicación se ejecuta `UI/Login.py`, que corresponde con la ventana raíz de la aplicación. En esta primera pantalla se muestra una breve descripción de la aplicación junto con un *Inicio de Sesión* donde cada usuario tendrá que identificarse para acceder.

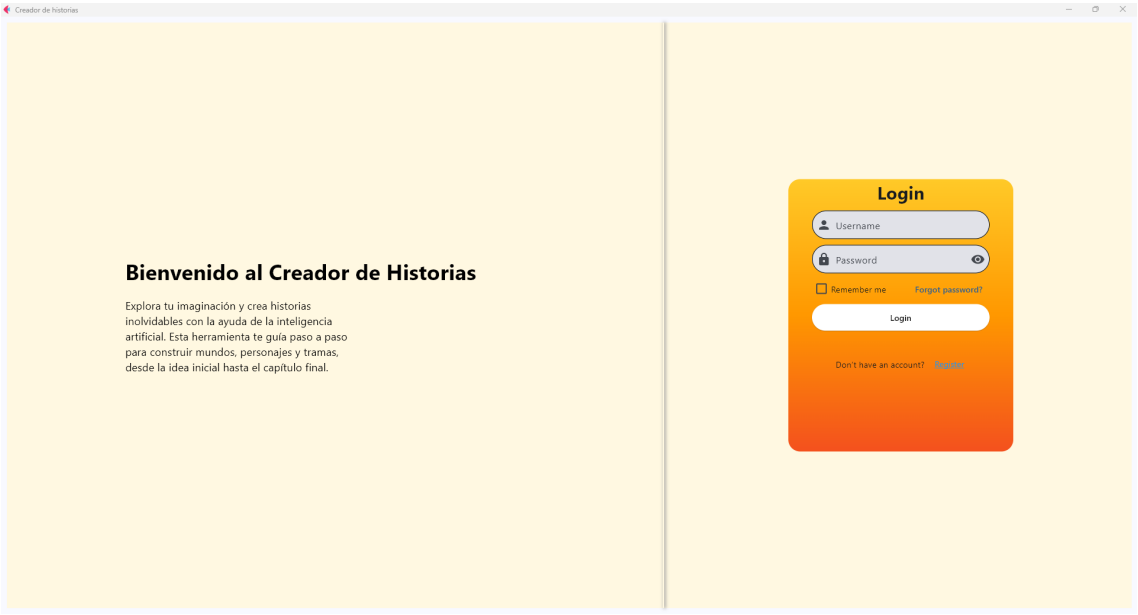

Figura 3.1

Toda la lógica encargada de gestionar los usuarios la lleva acabo `users/UserManager.py` donde se controla si el usuario y contraseña son correctas o en cambio, si se necesita crear un nuevo usuario. Una vez iniciada la sesión correctamente se pasa a la pagina ("/home"). Gracias a la funcionalidad que ofrece *Flet* con `flet.Page.client_storage`, podemos guardar datos de usuario localmente para mantener la sesión abierta.

(Falta hablar de como guardo los usuarios y contraseñas)

3.1.3. Selección de modo

Después de iniciar sesión pasamos a la página "/home" donde podremos seleccionar entre los dos tipos distintos de creación de historia. En `UI/Home.py`, haciendo uso nuevamente de `client_storage` guardo para el usuario que modo a seleccionado, si el modo rápido o detallado. (Tengo que crear para cada historia un id y guardar en que modo estaba. No se si para cada modo es la misma pantalla o cambiara) Una vez seleccionado el modo se redirige a la pantalla ("/new_history") donde se procederá a crear la nueva historia.

3.1.4. Creación de historia

Esta es la parte más importante de la aplicación donde sea crea la historia por medio de la IA y es gestionado por `UI/NewHistory`. Al enviar el *prompt* lo primero que se hace es crear un hilo para procesar la respuesta de la IA en segundo plano y así no bloquear la interfaz gráfica del usuario. Este hilo llama a `controller/StoryController.py` que se usa como controlador intermedio entre la interfaz gráfica y la lógica de la IA. Dependiendo del modo que se hubiese seleccionado se llamará a `get_ai_response_fast` en el modo rápido o `get_ai_response` en el modo detallado. Estas dos funciones pertenecen a `AI/AIManager.py` y son las encargadas de realizar la petición, en este caso, a `deepseek-r1-distill-llama-70b`.

Dependiendo de que modo sea se tiene diferente *prompt del sistema* con las reglas que debe seguir cada uno. Tras generar la respuesta y ser procesada se guarda en la base de datos para utilizarse más adelante. Para finalizar, el mensaje que ha generado la IA se le muestra al usuario por pantalla y ahora está listo para poder introducir nuevos *prompts* y reiniciar el proceso.

Figura 3.2

3.2. Interacción con la IA

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.

Cada vez que el usuario envía un mensaje por la interfaz, este es gestionado por el método `send_message` de `UI/NewHistory.py`, que lanza un *thread* con `threading.Thread(target=procesar_en_segundo_plano, args=(prompt,)).start()` para poder gestionar la respuesta de la IA en segundo plano y no bloquear mientras la interfaz.

Código 3.2: Crear thread

```
def send_message(e):
    prompt = entrada.value.strip()
    if prompt == "":
        return
    print(f"Mensaje enviado: {prompt}")
    scroll_marker = ft.Text("", key="scroll-target")
    mensajes.controls.append(scroll_marker)

    entrada.value = ""
```

```
    zona_carga.visible = True
    page.update()
    mensajes.controls.append(ft.Text(f"Tu: {prompt}"))
    threading.Thread(target=procesar_en_segundo_plano, args=(
       prompt,)).start()
```

Mientras este *thread* está activo, bloquea temporalmente la posibilidad al usuario de poder enviar nuevos mensajes, deshabilitando el campo de entrada (`entrada.disble = True`) y el botón de envío (`btn_send.disable = True`). Esto permite que el usuario no pueda generar múltiples mensajes simultáneamente mientras se está generando una respuesta, evitando así colisiones y duplicación de peticiones a la IA.

Código 3.3: Procesar mensaje en segundo plano

```
def procesar_en_segundo_plano(prompt):
   print("Estoy procesando la respuesta")
   entrada.disabled = True
   btn_send.disabled = True
   btn_copy.disabled = True

   respuesta = controller.procesar_mensaje(prompt, page)
   page.run_thread(lambda: mostrar_respuesta(respuesta))
   page.update()
```

Además este *thread* llama a `controller.procesar_mensaje` que se encuentra en `controller/StoryController`. Allí se realiza los siguientes pasos: (De momento solo está para el modo detallado)

Código 3.4: Procesar mensaje de la IA

```
def procesar_mensaje(self, user_input, page):
        modo = self.elegir_modo(page)
        if modo == "rapido":
            ai_response = get_ai_response_fast([{"role": "user
               ", "content": mensaje}])
        else:
            print(f"Sección actual: {self.seccion}")
            self.estado_secciones[self.seccion] = True
            nueva_seccion = detectar_seccion(user_input, self.
               seccion, self.estado_secciones)
            self.seccion = nueva_seccion
            self.contexto = ss.buscar_dependencias_bd(self.
               seccion)
            mensaje = f"{self.contexto}\n{user_input}"
            ai_response = get_ai_response([{"role": "user", "
               content": mensaje}])

        ai_json = generar_JSON_mongo(ai_response)
        formateado = sf.JSON_formateado(ai_json)

        texto = sf.json_texto(ai_response, self.seccion)
```

```
        ss.guardar_texto(texto)
        ss.guardar_doc(formateado, self.seccion)

        return ai_response
```

3.2.1. Detección de sección narrativa

Una clave de la aplicación es que la salida se organiza en bloques narrativos definidos: trama, persoanjes, escenarios, etc. Para lograr esto es imprescindible que el sistema identifique correctamente a que parte de la historia corresponde cada mensaje del usuario para poder guiar la conversación y almacenar cada respuesta en su colección correspondiente de la base de datos. Además, un aspecto muy importante, es que debemos detectar esta sección antes de que el LLM genere la respuesta ya que, como veremos más adelante, es un aspecto crucial a la hora de generar el contexto necesario. Esta detección de secciones se realiza gracias a la función `detectar_seccion` de `AI/AIManager.py`, que realiza una llamada a la IA con un `PROMPT_SECCTION` pidiendo que analice el mensaje enviado por el usuario y así poder detectar la sección.

Código 3.5: Funcion para detectar la seccion narrativa

```
def detectar_seccion(texto: str, seccion: str,
   estado_secciones: dict) -> str:
    seccion_prompt = [
        {"role": "system", "content": f"{PROMPT_SECCION} \n\n
           Estado de las secciones: {json.dumps(
           estado_secciones, indent=2)}"},
        {"role": "user", "content": f"Seccion actual: {seccion
           } -> {texto}"}
    ]

    detectar_seccion = get_ai_response_others(seccion_prompt)
    return detectar_seccion
```

Otro problema es que el usuario puede mandar mensajes de confirmación simplemente para pasar a la siguiente sección y continuar con el orden preestablecido. Esto se soluciona creando un diccionario `estado_secciones` inicializando cada sección a `False` y cambiando a `True` cuando se completa. De esta forma en `detectar_seccion` se construye un *prompt estructurado* en formato de chat que incluye el diccionario `estado_secciones`, la sección actual y mensaje del usuario, y se le pide a la IA que cuando reciba un mensaje de confirmación busque en el diccionario la primera sección a *False* que corresponde a la siguiente que falta por completar.

Código 3.6: Diccionario secciones

```
self.estado_secciones = {
    "INICIO": False,
    "TRAMA E HILO SIMBOLICO": False,
    "DESCRIPCION DEL MUNDO": False,
    "DESCRIPCION DE ESCENARIOS": False,
```

```
    "PERSONAJES PRINCIPALES": False,
    "PERSONAJES SECUNDARIOS": False,
    "ANALISIS DE LA HISTORIA": False,
    "ESTRUCTURA DE CAPITULOS": False,
    "ESCRITURA DE CAPITULOS": False
}
```

3.2.2. Búsqueda de contexto narrativo

El modelo LLM no tiene memoria a largo plazo, y cada llamada es completamente independiente. Por eso es necesario incluir el contexto de la historia junto con el *prompt* del usuario. El problema radica en que al utilizar *Groq* la llamada a la IA esta restringida a un máximo de 6000 *tokens* que incluye: el *prompt* del sistema, el *prompt* del contexto, el *input del usuario* y la propia respuesta del LLM. Eso implica que no se pueda mantener toda la historia en memoria en una sola petición y sea necesario seleccionar cuidadosamente qué partes de la historia incluir como contexto, para que la IA disponga de la información más relevante. Para ello se utiliza el diccionario `dependencias` donde a cada una de las secciones le corresponde, a su vez, una lista de secciones que se tiene que pasar como contexto a la hora de crear dicha sección.

Código 3.7: Dependencias entre secciones

```
dependencias = {
 "TRAMA E HILO SIMBOLICO": [],
 "DESCRIPCION DEL MUNDO": ["TRAMA E HILO SIMBOLICO"],
 "PERSONAJES PRINCIPALES": ["TRAMA E HILO SIMBOLICO"],
 "PERSONAJES SECUNDARIOS": ["TRAMA E HILO SIMBOLICO", "
    PERSONAJES PRINCIPALES"],
 "DESCRIPCION DE ESCENARIOS": ["TRAMA E HILO SIMBOLICO", "
    DESCRIPCION DEL MUNDO"],
 "ANALISIS DE LA HISTORIA": ["TRAMA E HILO SIMBOLICO", "
    DESCRIPCION DEL MUNDO", "PERSONAJES PRINCIPALES"],
 "ESTRUCTURA DE CAPITULOS": ["TRAMA E HILO SIMBOLICO", "
    ANALISIS DE LA HISTORIA"]
}
```

Este diccionario se utiliza en `buscar_dependencias_bd` para que dada una sección busque en la base de datos las secciones necesarias para pasarlas como contexto y devolver un *string* formado por todas ellas. (Tengo que ver que pasa si quieres hacer una seccion y no has hecho antes las secciones de las que depende)

Código 3.8: Buscar dependencias en la base de datos

```
def buscar_dependencias_bd(seccion_actual: str) -> str:
    """Busca en la base de datos las secciones necesarias para
        el contexto."""
    secciones_necesarias = dependencias.get(seccion_actual,
       [])
    contexto = ""
```

```
    mongo.seleccionar_coleccion("CONTEXTO")
    if mongo.hay_documentos():
        for seccion in secciones_necesarias:
            texto = mongo.devolver_documentos(seccion)
            contexto += texto
    text = contexto.replace("\n", " ")
    return text
```

3.2.3. Composición del mensaje final

Este contexto se junta con el *input del usuario* para usarlo finalmente en la IA y pueda generar el texto narrativo:

```
            mensaje = f"{self.contexto}\n{user_input}"
ai_response = get_ai_response([{"role": "user", "content": mensaje}])
```

Con este nuevo mensaje se llama a `get_ai_response` de `AI/AIManager.py` donde se realiza la llamada al cliente de *Groq*.

Código 3.9: 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)
```

En este método se añade al mensaje el *prompt inicial* que contiene la estructura y las reglas que debe de seguir la IA 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 3.1: 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*, el *input* del usuario y el contexto narrativo, por eso se debe de ajustar cuidadosamente el máximo de *tokens* por respuesta para no desbordar el tope.

Finalmente, la respuesta de la IA es procesada por `clean_ai_response`, que elimina las etiquetas `<think>...</think>` para mostrar al usuario un texto limpio.

Código 3.10: Limpieza de la respuesta

```
def clean_ai_response(response:str)->str:
    """Elimina etiquetas <think>...</think> del texto."""
    return re.sub(r"<think>.*?</think>", "", response, flags=
       re.DOTALL).strip()
```

3.2.4. Generación de JSON

Una vez generada la respuesta necesitamos convertir esta cadena de texto a formato *JSON* para poder incluirlo a *MongoDB*. Vamos a generar dos tipos de *JSON*: uno sin estructurar donde únicamente se guarda una clave *Seccion* para saber a que sección pertenece, y una clave *Texto* con toda la respuesta de la IA sin procesar, para usarse a la hora de pasarle el contexto.

Código 3.11: Convertir texto a JSON

```
def json_texto(texto:str, seccion:str)->json:
    """Convierte el texto de la IA a un formato JSON para
       MongoDB."""
```

```
    return{ "Seccion": seccion, "texto": texto.strip() }
```

El otro tipo es un *JSON* estructurado donde queremos organizarlo en claves según el texto generado por el modelo. Al no seguir todas las respuestas la misma estructura, era muy difícil el poder convertir el texto, de forma normal, a *JSON*. Por eso se optó por hacer uso de la IA y realizar otra llamada. En este caso se le pide que convierta el texto en un formato *JSON* bien estructurado y así crear las claves necesarias sin importar el texto. Se llama a la función `generar_JSON_mongo` en la que tiene como parámetro la respuesta de la IA y usa como *promtp del sistema PROMPT_JSON* donde se especifica las instrucciones que tiene que seguir.

Código 3.12: Convertir texto a JSON estructurado

```
def generar_JSON_mongo(ai_response:str)->str:
    """Convertir la respuesta de la IA a formato JSON para
       poder añadirlo a Mongo"""
    JSON_prompt = [
        {"role": "system", "content": PROMPT_JSON
        },
        {"role": "user", "content": ai_response}
    ]

    mongo_JSON = get_ai_response(JSON_prompt)

    return mongo_JSON
```

Esto nos devuelve otra cadena de texto pero con un formato *JSON* bien estructurado y ya solo quedaría convertir la cadena de texto en *JSON* usando el método `JSON_formateado`.

Código 3.13: Convertir a JSON

```
def JSON_formateado(texto_ia:str)->json:
    try:
        return json.loads(texto_ia)  #Devuelve el JSON
    except json.JSONDecodeError as e:
        print("Error al decodificar JSON:", e)
        return {}  #Si no se puede decodificar, devuelve un
           diccionario vacio
```

3.2.5. Almacenamiento en la base de datos

Una vez convertido el texto a *JSON* ya solo queda guardarlo en la base de datos. Para ello usamos la función `guardar_texto` que se encarga de guardar el *JSON* no estructurado en la colección llamada "CONTEXTO".

Código 3.14: Guardar contexto

```
def guardar_texto(texto:json):
    mongo.seleccionar_coleccion("CONTEXTO")
    mongo.insertar(texto, "CONTEXTO")
```

A la hora de guardar el *JSON* estructurado en la base de datos tenemos que comprobar más posibilidades. Se contemplan distintas opciones para los distintos tipos de secciones, tenemos los escenarios y personajes que son una lista, una colección "UPDATE" para cuando se realizan cambios y una colección "OTROS" que se usa cuando hay algun tipo de error.

Código 3.15: Guardar secciones en la base de datos

```
def guardar_doc(json_mongo, seccion):
    if seccion == SeccionHistoria.UPDATE:
        mongo.seleccionar_coleccion("UPDATE")
        mongo.insertar(json_mongo, seccion)
    elif seccion == None:
        mongo.seleccionar_coleccion("OTROS")
        mongo.insertar(json_mongo, "OTROS")
    elif seccion == SeccionHistoria.ESCENARIOS.value or
       seccion == SeccionHistoria.PERSONAJES_P.value or
       seccion == SeccionHistoria.PERSONAJES_S.value:
        mongo.seleccionar_coleccion(seccion)
        if mongo.hay_documentos():
            print(f"Estoy modficando '{seccion}'")
            formateado = eliminar_seccion_json(json_mongo)
            print(formateado)
            mongo.nuevo_escenario_personaje("Seccion", seccion
               , listas_bd(seccion), formateado)
        else:
            mongo.insertar(json_mongo, seccion)
    else:
        mongo.seleccionar_coleccion(seccion)
        if mongo.hay_documentos():
            doc_previo = mongo.buscar("Seccion", seccion)   #
               Busco ese documento
            mongo.seleccionar_coleccion("UPDATE")   #Me muevo
               a la coleccion de UPDATE
            mongo.insertar(doc_previo, seccion) #Lo inserto en
                la coleccion de UPDATE
            mongo.seleccionar_coleccion(seccion)    #Cambio a
               la coleccion actual
            mongo.eliminar("Seccion", seccion)  #Elimino el
               documento antiguo
        mongo.insertar(json_mongo, seccion)
```

3.2.6. Presentación del mensaje al usuario

Una vez procesado el mensaje conforme a los pasos anteriores se hace un *return* del mensaje generado por la IA y se vuelve a `UI/NewHisotry` donde se ejecuta `mostrar_respuesta`.

Código 3.16: Mostrar respuesta al usuario

```
def mostrar_respuesta(respuesta):
    entrada.disabled = False
    btn_send.disabled = False
    zona_carga.visible = False
    btn_copy.disabled = False

    mensajes.controls.append(
        ft.Text("IA:", weight=ft.FontWeight.BOLD, selectable=
            True)
    )

    for elem in parse_respuesta_md(respuesta):
        mensajes.controls.append(elem)

    page.update()


    mensajes.scroll_to(key="scroll-target", duration=300)
```

En esta función, para terminar, se llama a `parse_respuesta_md` ya que se necesita quitar el *markdown* para que sea más atractivo cuando se muestre el mensaje al usuario.

Código 3.17: Eliminar markdown

```
def parse_respuesta_md(respuesta: str) -> list[ft.Text]:
    """
    Procesa una respuesta Markdown simulada y devuelve una
       lista de componentes Flet.
    Detecta encabezados tipo ### Título o **# Título** y
       formatea negrita/cursiva.
    """
    elementos = []
    for linea in respuesta.split("\n"):
        linea = linea.strip()
        if not linea:
            continue

        linea = corregir_estilos_linea(linea)
        # Encabezado tipo ### Título o ### **Título**
        if re.match(r"^(#{3,6})\s+(.*?)(\s+#{3,6})?$", linea):
            nivel, contenido = re.findall(r"^(#{3,6})\s+(.*?)
               (\s+#{3,6})?$", linea)[0][:2]
            contenido = re.sub(r"\*\*(.*?)\*\*", r"\1",
               contenido)  #elimina negrita Markdown
            tamaño = {3: 22, 4: 20, 5: 18, 6: 16}.get(len(
               nivel), 14)
            elementos.append(
                ft.Text(contenido, weight=ft.FontWeight.BOLD,
                   size=tamaño, selectable=True)
            )
```

```
        else:
            elementos.append(convertir_a_richtext(linea))

    return elementos
```

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.

Introduction

Introduction to the subject area. This chapter contains the translation of Chapter 1.

Conclusions and Future Work

Conclusions and future lines of work. This chapter contains the translation of Chapter 4.

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