



# CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

PROGRAMACIÓN PARA INTERNET

Sección: D03



## Proyecto. Better | Essay

*Profesor: Michel Emanuel López Franco*

Alumna: Katia Marlene Salcedo Huerta

Código: 210588774

# Contenido

Resumen.....	3
Objetivo general .....	3
Objetivos específicos .....	3
Arquitectura general del sistema .....	4
Corrección de ensayos – OpenRouter (GPT-4o-mini) .....	4
Generación de resúmenes – FastAPI + DistilBART (Hugging Face).....	5
Despliegue.....	7

## Resumen

Better | Essay es una aplicación web que integra tecnologías de inteligencia artificial para ayudar a los usuarios en la mejora de sus textos, tanto ensayos como resúmenes.

Se usa el modelo GPT-4o-mini por medio de OpenRouter para la corrección y reescritura de ensayos, y DistilBART CNN-12-6 (Hugging Face Transformers) para la generación de resúmenes precisos, dentro de una arquitectura en la que se utilizan: React, Node.js, FastAPI y MongoDB Atlas.

El objetivo de la plataforma es ofrecer una herramienta práctica, accesible para perfeccionar la escritura académica y profesional.

## Objetivo general

Desarrollar una aplicación web capaz de analizar, corregir y resumir ensayos y generar resúmenes claros mediante el uso de modelos de inteligencia artificial.

## Objetivos específicos

- Implementar una interfaz para ingresar y visualizar ensayos.
- Implementar una interfaz para ingresar texto extenso y visualizar resúmenes.
- Manejar usuarios y autenticación JWT.
- Utilizar MongoDB Atlas para el almacenamiento de datos en la nube.
- Desplegar los servicios en Render y Hugging Face Spaces.

## Arquitectura general del sistema

Componente	Tecnología	Función principal
Frontend	React + Vite + TypeScript	Interfaz de usuario, manejo de formularios y vistas.
Backend	Node.js + Express + TypeScript	API principal, autenticación, conexión base de datos.
Base de datos	MongoDB Atlas	Almacenamiento seguro de usuarios.
IA Corrección	OpenRouter (GPT-4o-mini)	Revisión de ensayos, corrección y sugerencias.
IA Resumen	FastAPI + Transformers (DistilBART CNN-12-6)	Generación de resúmenes de texto.
Despliegue	Render + Hugging Face Spaces	Infraestructura.

## Explicación técnica de las inteligencias artificiales

Corrección de ensayos – OpenRouter (GPT-4o-mini)

**Lenguaje:** TypeScript

El servicio de corrección se basa en el modelo GPT-4o-mini, accedido mediante la API de OpenRouter.

El código envía al modelo dos entradas: el tema y el contenido del ensayo, junto con un mensaje del sistema que define el rol del modelo como un *tutor experto en escritura*.

El prompt está hecho para producir una salida estructurada en formato Markdown, dividida en tres secciones:

1. **Correcciones y Feedback:** errores ortográficos, gramaticales y de estilo, con sugerencias específicas.

2. **Ensayo Mejorado:** una versión reescrita, clara y coherente, conservando las ideas del usuario.
  3. **Recomendaciones Generales:** consejos para mejorar redacción, cohesión y vocabulario.

#### **Fragmento clave del código:**

t.config.js    TS AI API.ts    TS AIResponse.ts X    NavMenu.tsx

frontend > src > api > TS AIResponse.ts > ...

```
1 import { openrouter } from "@lib/ai";
2 import { generateText } from "ai";
3
4 export async function generateResponse(
5   essayTopic: string,
6   essayContent: string
7 ) {
8   const result = await generateText({
9     model: openrouter("gpt-4o-mini"),
10    messages: [
11      {
12        role: "system",
13        content: `You are an expert writing tutor specialized in reviewing essays for improvement.
14
15 You will receive 2 inputs:
16`
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

^C

libel@DESKTOP-RAB1DM2 MINGW64 ~/Desktop/Marlene/Marlene/JH/Ing. Computer-Essay/backend (main)

\$ npm run dev:api

Generación de resúmenes – FastAPI + DistilBART (Hugging Face)

## Lenguaje: Python

La API de resúmenes está desarrollada con FastAPI, y utiliza el modelo sshleifer/distilbart-cnn-12-6, una versión optimizada del modelo BART, para tareas de *abstractive summarization*.

El sistema implementa una función llamada `resumir_texto_largo()` que divide el texto en fragmentos de hasta 500 palabras. Cada fragmento es procesado individualmente y luego los resultados parciales se combinan y refinan para generar un resumen final.

## Flujo del proceso:

1. El usuario envía el texto desde el frontend al endpoint /resumir.
2. FastAPI limpia el texto y lo fragmenta en partes manejables.
3. Cada parte se pasa al modelo summarizer.
4. Se unen los resultados parciales y se ajusta la longitud del resumen final.

## Fragmento clave del código:

The screenshot shows two code snippets in a code editor interface. The top snippet is from the file `main.py` and defines a class `TextRequest` with a method `limpiar_texto` that concatenates a string by splitting it into words. It also defines a function `resumir_texto_largo` that splits the text into chunks of size 500 and joins them back together. The bottom snippet is also from `main.py` and defines a function `resumir_texto_largo` with parameters `max_length=150`, `min_length=50`, and `chunk_size=500`. It also includes a `@app.post("/resumir")` decorator and a `generar_resumen` function that calls `resumir_texto_largo` on a cleaned-up version of the input text.

```
fig.js      TS AI API.ts      TS AIResponse.ts      NavMenu.tsx      AppLayout.tsx      requirements.txt      main.py
↳ main.py > ⚡ limpiar_texto
    )

    summarizer = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6")

    class TextRequest(BaseModel):
        ... . . .
        texto: str

        def limpiar_texto(texto: str) -> str:
            ... . . .
            return " ".join(texto.split())

        def resumir_texto_largo(texto: str, max_length=150, min_length=50, chunk_size=500) -> str:
            ... . . .
            palabras = texto.split()
            ... # Crear chunks de tamaño chunk_size
            chunks = [" ".join(palabras[i:i+chunk_size]) for i in range(0, len(palabras), chunk_size)]
```

  

```
fig.js      TS AI API.ts      TS AIResponse.ts      NavMenu.tsx      AppLayout.tsx      requirements.txt
↳ main.py > ⚡ generar_resumen
    def resumir_texto_largo(texto: str, max_length=150, min_length=50, chunk_size=500) -> str:
        ... . . .
        return texto_resumen

    @app.post("/resumir")
    def generar_resumen(req: TextRequest):
        ... if not req.texto.strip():
        ...     return {"error": "No se recibió texto válido."}

        ... texto_limpio = limpiar_texto(req.texto)
        ... resumen_final = resumir_texto_largo(texto_limpio)

        ... return {"resumen": resumen_final}
```

EMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

@DESKTOP-RAB1DM2 MINGW64 ~/Desktop/Marlene/Marlene/JH/Ing. Computación/Semestre 10/Programacion-Interna

## Despliegue

- **Frontend y Backend:** Render
- **IA de Resúmenes:** Hugging Face Spaces
- **Base de Datos:** MongoDB Atlas
- **Corrección de Ensayos:** OpenRouter API

Se tienen variables de entorno (.env, .env.local) que protegen las claves privadas.