



ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS



FUNDAMENTOS INTELIGENCIA ARTIFICIAL

ASIGNATURA:	Fundamentos de Inteligencia Artificial
PROFESOR:	Ing. Vanessa Guevara
PERÍODO ACADÉMICO:	2025-B

Proyecto

TÍTULO:

Chat box Inteligente Con IA



Manual Técnico

Instalación del entorno y ejecución del microservicio

1.1 Requisitos del sistema

Para la implementación del microservicio se utilizaron las siguientes herramientas:

- Sistema operativo: Windows
- Python versión 3.10 o superior
- Editor de código: Visual Studio Code
- Terminal de comandos

1.2 Instalación de Python

1. Descargar Python desde el sitio oficial:
<https://www.python.org/downloads/>
2. Durante la instalación se debe marcar la opción:
“Add Python to PATH”.
3. Verificar la instalación ejecutando en la terminal:

```
C:\Users\Oscar>python --version
Python 3.13.9
```

1.3 Creación del entorno virtual

Se creó un entorno virtual para aislar las dependencias del proyecto.

```
PS C:\Users\Oscar\Documents\microservicio-ia-python> python -m venv venv
```

Activar el entorno virtual:

Windows

```
PS C:\Users\Oscar\Documents\microservicio-ia-python> venv\Scripts\activate
(venv) PS C:\Users\Oscar\Documents\microservicio-ia-python>
```

1.4 Instalación de librerías

Con el entorno virtual activo se instalaron las dependencias necesarias:

```
(venv) PS C:\Users\Oscar\Documents\microservicio-ia-python> pip install fastapi uvicorn pydantic python-dotenv google-genai
```

Guardar dependencias:

```
o (venv) PS C:\Users\Oscar\Documents\microservicio-ia-python> pip freeze > requirements.txt
```

1.5 Ejecución del microservicio

Para iniciar el servidor local:

```
o (venv) PS C:\Users\Oscar\Documents\microservicio-ia-python> uvicorn main:app --reload
>>
INFO: Will watch for changes in these directories: ['C:\\Users\\Oscar\\Documents\\microservicio-ia-python']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [2560] using StatReload
INFO: Started server process [1376]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

Acceso al servicio:

```
http://127.0.0.1:8000/
```

Documentación automática:

```
127.0.0.1:8000/docs
```

Explicación del Modelo y Configuración

2.1 Selección del modelo

Para este proyecto se definió el uso del modelo **GPT-4o-mini**, debido a las siguientes razones:

- Alta capacidad de comprensión del lenguaje natural
- Bajo costo computacional
- Velocidad de respuesta adecuada
- Idoneidad para asistentes conversacionales académicos

Para fines académicos, la arquitectura del sistema se diseñó para poder integrar modelos reales de IA sin modificar la lógica principal del API.

```
model="gemini-2.5-flash",
```

2.2 Configuración del System Prompt

Se implementó un **System Prompt** para definir el rol del asistente virtual **MiDesk**, con el objetivo de:

- Restringir las respuestas al contexto académico
- Evitar respuestas fuera del sistema
- Simular un comportamiento entrenado del asistente

```
SYSTEM_PROMPT = """
Eres MiDesk, un asistente virtual del escritorio MiDesk.
Ayudas con organización académica, tareas y notas.
Si la pregunta NO es sobre MiDesk responde:
"Solo puedo ayudar con temas relacionados a MiDesk."
"""
```

Ejemplo de reglas definidas:

- Ayudar con organización de tareas y notas
- Rechazar preguntas fuera del entorno MiDesk
- Responder de manera clara y breve

2.3 Parámetros configurados

Se definieron parámetros de generación de texto:

```
"temperature":0.3,"max_tokens":300},"metricas":{"tiempo_respuesta_ms":2943}}]
```

- **temperature** = 0.3 → respuestas consistentes
- **max_tokens** = 300 → límite de longitud de respuesta

Estos parámetros permiten controlar la variabilidad y extensión de las respuestas generadas.

2.4 Métricas obtenidas

Se implementaron métricas básicas para evaluar el desempeño del sistema:

- Tiempo de respuesta del endpoint
- Conteo aproximado de tokens (palabras)
- Registro de métricas en consola
- Devolución de métricas en formato JSON

Estas métricas permiten analizar la eficiencia del microservicio.

```
@app.post("/chat")
def chat(req: ChatRequest):
    inicio = time.time()
    temperature = 0.3
    max_tokens = 300

    try:
        api_key = os.getenv("GOOGLE_API_KEY")
        if not api_key:
            return {"error": "GOOGLE_API_KEY no encontrada en .env"}

        client = genai.Client(api_key=api_key)

        prompt = f"{SYSTEM_PROMPT}\nUsuario: {req.mensaje}"

        response = client.models.generate_content(
            model="gemini-2.5-flash",
            contents=prompt
        )

        fin = time.time()

        return {
            "respuesta": response.text,
            "parametros_modelo": {
                "temperature": temperature,
                "max_tokens": max_tokens
            },
            "metricas": {
                "tiempo_respuesta_ms": int((fin - inicio) * 1000)
            }
        }

    except Exception as e:
        return {
            "error": "Fallo al generar respuesta",
            "detalle": str(e)
        }
```

Limpieza de Código (Código comentado)

```
from fastapi import FastAPI
from pydantic import BaseModel
import time

# Inicialización de la aplicación FastAPI
app = FastAPI(title="Microservicio IA - MiDesk")

# System Prompt que define el comportamiento del asistente
SYSTEM_PROMPT = """
Eres MiDesk, un asistente virtual que ayuda con organización
académica,
tareas y notas dentro del sistema MiDesk.
"""

# Modelo de validación del request JSON
class ChatRequest(BaseModel):
    mensaje: str

# Función que cuenta palabras (tokens aproximados)
def contar_palabras(texto: str) -> int:
    return len(texto.split())

# Función que simula la respuesta de la IA
def respuesta_simulada_midesk(mensaje: str) -> str:
    mensaje = mensaje.lower()
    if "tarea" in mensaje:
        return "Puedes organizar tus tareas creando listas por materia
y fechas."
    return "Solo puedo ayudar con temas relacionados a MiDesk."

# Endpoint POST /chat
@app.post("/chat")
def chat(req: ChatRequest):
    inicio = time.time()

    temperature = 0.3
    max_tokens = 300

    respuesta = respuesta_simulada_midesk(req.mensaje)

    fin = time.time()

    tiempo_respuesta_ms = int((fin - inicio) * 1000)
    tokens_entrada = contar_palabras(req.mensaje)
    tokens_salida = contar_palabras(respuesta)

    return {
        "respuesta": respuesta,
        "parametros_modelo": {
            "temperature": temperature,
            "max_tokens": max_tokens
        },
        "metricas": {
            "tiempo_respuesta_ms": tiempo_respuesta_ms,
            "tokens_totales_aprox": tokens_entrada + tokens_salida
        }
    }
```

Postman-Endpoint

The screenshot shows the Postman interface with a POST request to `http://127.0.0.1:8000/chat`. The request body is a JSON object with a `mensaje` field. The response is a JSON object with a `respuesta` field containing instructions on how to create a folder in MiDesk.

Request:

```
1 {
2   "mensaje": "¿Como creo una carpeta en MiDesk?"
3 }
4
```

Response:

```
1 {
2   "respuesta": "Para crear una carpeta en MiDesk, sigue estos pasos:\n\n1. Haz clic derecho en cualquier espacio vacío de tu escritorio MiDesk.\n2. En el menú contextual que aparece, selecciona la opción \"Nuevo\".\n3. Luego, haz clic en \"Carpeta\". Aparecerá una nueva carpeta con un nombre predeterminado (por ejemplo, \"Nueva Carpeta\"). Puedes escribir el nombre que desees para tu carpeta y luego presionar 'Enter' para confirmarlo.\n\n¡Así de fácil puedes organizar tus archivos y proyectos académicos en MiDesk!\",
3   "parametros_modelo": {
4     "temperature": 0.3,
5     "max_tokens": 300,
6     "metricas": {
7       "tiempo_respuesta_ms": 2943
8     }
9   }
10 }
```