Guía de práctica: Invocar Ollama (pensando antes de pegar código)

Fecha: 12/10/2025

Objetivo: que construyas —desde cero— dos pequeñas pruebas de concepto para llamar a un modelo local de Ollama. No hay solución completa; debes decidir cómo unir las piezas.

Reglas del juego

• No se permite copiar/pegar una solución completa. Usa estos esqueletos y completa los huecos.

• Justifica en comentarios cada decisión técnica (1 línea por decisión).

• Entrega: captura de pantalla + archivo .py por cada parte + README con pasos de ejecución.

Requisitos previos

• Tener Ollama ejecutándose en http://localhost:11434 y haber hecho: ollama pull llama3.2

• Python 3.9+; instalar paquetes necesarios (tú decides cuáles).

Parte A — Usando la API HTTP con requests (sin SDK)

Construye un script llamado prueba_a.py que:

1) Defina constantes básicas (endpoint, modelo, headers).

2) Prepare una lista messages con un único turno de usuario con el mensaje: 'Describe some of the business applications of Generative AI'.

3) Envíe una petición POST al endpoint de chat de Ollama y muestre la respuesta ('message' → 'content').

4) Maneje al menos dos errores frecuentes: servidor no disponible y modelo no descargado.

```
# prueba_a.py — completa los huecos marcados con TODO
# 1) Imports que necesitas
import ____
# (Puedes añadir otros si los usas)

# 2) Constantes
OLLAMA_API = "http://localhost:11434/____"   # TODO: completa el path
correcto
HEADERS = {"Content-Type": "application/json"}
MODEL = "llama3.2"

# 3) Mensajes (formato estilo chat)
messages = [
    {"role": "user", "content": "Describe some of the business applications
of Generative AI"}
```

```
]

# 4) Construcción del payload
payload = {
    "model": MODEL,
    "messages": messages,
    "stream": ____  # TODO: ¿True o False? justifica tu elección
    # (opcional) "options": {"temperature": 0.2}
}

# 5) Envío de la petición y manejo de errores
def main():
    try:
        resp = ____.___(OLLAMA_API, json=payload, headers=HEADERS,
timeout=120)  # TODO
        resp.____()  # TODO: verificar estado
        data = resp.___()  # TODO: parsear JSON
        print(data["____"]["____"])  # TODO: acceder al texto
    except ____ as e:
        print("[ERROR] Explica aquí el posible motivo y cómo lo
verificarías.")
        print(e)

if __name__ == "__main__":
    main()
```

Checklist de comprensión

□ ¿Qué diferencia hay entre 'stream': true y false? □ ¿Dónde vive el texto de salida en la respuesta JSON? □ ¿Qué HTTP status esperarías en éxito?

Parte B — Usando el SDK: import ollama (sin requests)

Construye un script llamado prueba_b.py que haga lo mismo, pero usando el paquete 'ollama'.

Debes:

• Importar el paquete y crear la misma estructura de mensajes.

• Llamar a la función adecuada y extraer el contenido de la respuesta.

• Manejar al menos una excepción y explicar cómo la reproducirías.

```
# prueba_b.py — completa los huecos
import ____  # TODO: SDK adecuado

MODEL = "llama3.2"
messages = [
    {"role": "user", "content": "Describe some of the business applications
of Generative AI"}
]

try:
    response = ____.____(model=MODEL, messages=messages)  # TODO: función
correcta
    print(response["____"]["____"])  # TODO: claves correctas
```

```
except Exception as e:
    print("[ERROR] Explica aquí qué pasó y cómo lo comprobarías.")
    print(e)
```

Preguntas de reflexión (responde en el README)

1) ¿Qué ventajas e inconvenientes ves entre usar la API HTTP y el SDK?

2) ¿Qué campos serían útiles en 'options' para controlar el estilo de salida?

3) ¿Cómo validarías que el modelo realmente corresponde a 'llama3.2' y no a otro?

4) Si quisieras internacionalizar el prompt, ¿qué cambios harías?

Extras (opcionales, +1 punto cada uno)

- Añade un argumento por línea de comandos para cambiar el modelo.
- Guarda la respuesta en un archivo .txt con marca temporal.
- Mide el tiempo de inferencia y muéstralo al final.