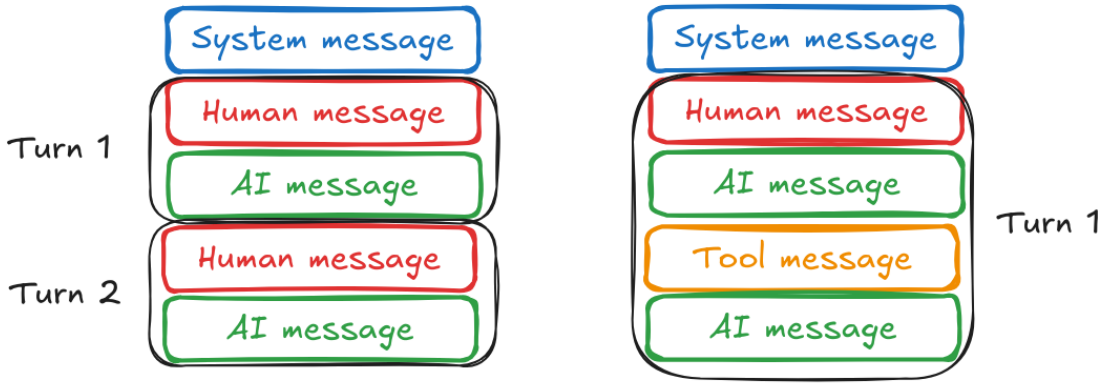
Introducción a LangChain

Gestión de mensajes

José Orlando Maldonado Bautista

Gestión de mensajes en LangChain

- En LangChain, los mensajes estructuran la conversación para modelos tipo chat.
- Los más usados: **SystemMessage** (reglas/rol) y **HumanMessage** (instrucción del usuario).
- Separarlos mejora consistencia, portabilidad entre proveedores y control del contexto.
- Regla práctica: primero define el “marco” con SystemMessage; luego da la instrucción con HumanMessage.

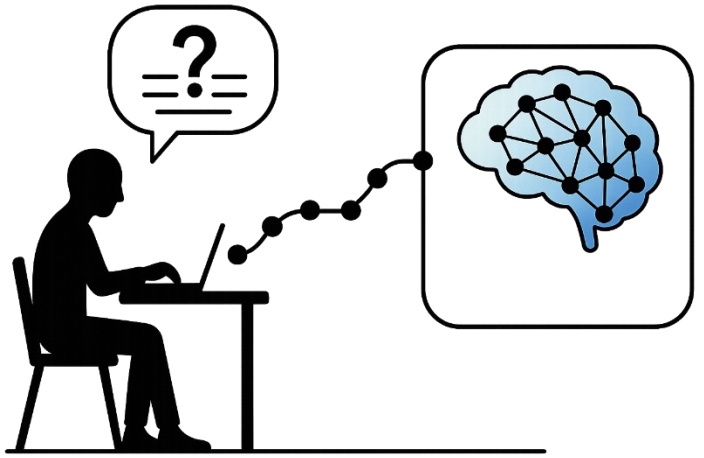


SystemMessage

- **Definición:** mensaje usado para indicar al modelo cómo debe comportarse y para aportar contexto adicional. *Nota*: no todos los proveedores de chat models lo soportan.
- **Uso típico:** contenido que dirige la conversación (p. ej., instrucciones iniciales o marco de comportamiento).

HumanMessage

- **Definición:** representa la entrada del usuario que interactúa con el modelo, usualmente en forma de texto u otra entrada interactiva.
- **Uso típico:** contenido que constituye la entrada enviada por el usuario al modelo.



Ejemplo:

```python
from langchain_core.messages import SystemMessage, HumanMessage
from langchain_ollama.chat_models import ChatOllama

# 1) Definimos el marco de comportamiento (SystemMessage)
# y la instrucción del usuario (HumanMessage)
messages = [
    SystemMessage(content=(
        "Eres un docente experto en didactica de la matemática, "
        "responde en español de forma clara y concisa. "
        "Cuando sea útil, usa viñetas."
    )),
    HumanMessage(content=(
        "Explica cómo resolver una sistema de ecuaciones de 2x2, "
        "por el método de sustitución, "
        "termina con un ejemplo numérico breve."
    )),
]

# 2) Inicializamos el chat model local de Ollama
chat = ChatOllama(model="llama3.2:3b", temperature=0.3)

# 3) Invocamos el modelo con la conversación
resp = chat.invoke(messages)
print(resp.content)
```

Clases de modelos en LangChain (Para conectar y enviar solicitudes)

Clases de Texto (LLM)

Ej: OpenAI, OllamaLLM, GoogleGenerativeAI.

Input/Output: texto plano → texto plano.

Uso: modelos antiguos (*instruct*) o prompts simples.

Tendencia: en desuso (deprecated).

Clases de Chat

Ej: ChatOpenAI, ChatGoogleGenerativeAI, ChatOllama.

Input/Output: mensajes con rol → mensajes.

Uso: chatbots, RAG, agentes, multimodalidad.

Estándar actual.

Plantillas en LangChain (Prompts templates)

Una plantilla de Prompt permite definir mensajes que pueden incluir contenido estático (texto fijo) y dinámico (variables que cambian según el contexto).

Estas plantillas son útiles para estandarizar las entradas al modelo, reducir la repetición manual y generar mensajes adaptables.

Beneficios:

Automatización: Evitan escribir mensajes repetitivos.

Flexibilidad: Incorporan variables dinámicas para personalizar la interacción.

Estandarización: Aseguran un formato consistente en los mensajes.

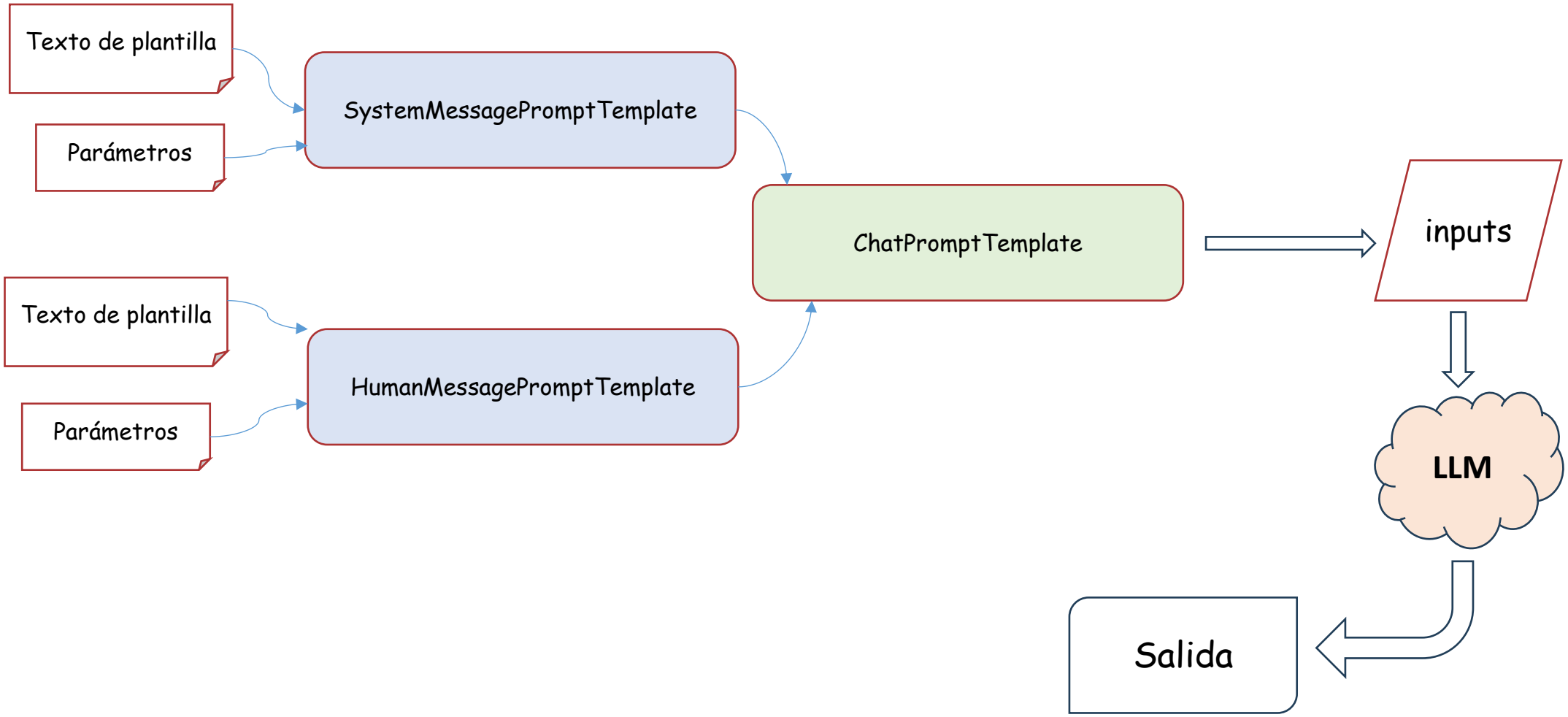
Plantillas en LangChain

Plantilla	Propósito	Uso principal
PromptTemplate	Genera mensajes básicos combinando texto estático y variables dinámicas.	Crear prompts reutilizables y personalizables con contenido dinámico.
ChatPromptTemplate	Estructura flujos conversacionales combinando mensajes de distintos tipos.	Crear interacciones complejas y organizadas entre humanos, sistema e inteligencia artificial.
SystemMessagePromptTemplate	Define mensajes del sistema para establecer contexto y reglas iniciales.	Configurar el comportamiento esperado del modelo durante la interacción.
HumanMessagePromptTemplate	Representa las consultas o entradas realizadas por un usuario humano.	Estandarizar y personalizar preguntas o comentarios del usuario.
AIMessagePromptTemplate	Construye mensajes que simulan respuestas generadas por la inteligencia artificial.	Predefinir y controlar las respuestas de la IA en flujos conversacionales planificados.

Ejemplo PromtTemplate

```python
from langchain_core.prompts import PromptTemplate
from langchain_ollama.chat_models import ChatOllama

# Definir plantilla
prompt = PromptTemplate(
    input_variables=["animal", "estilo"],
    template="Cuéntame  un dato curioso sobre un {animal} en un estilo {estilo}."
)

# Modelo
chat = ChatOllama(model="llama3.2:3b", temperature=0.3)

#
mensaje = prompt.format(animal="pulpo",estilo="académico y formal")

# Ejecutar
respuesta = chat.invoke(mensaje)
print(respuesta.content)
```

Uso de PromptTemplates

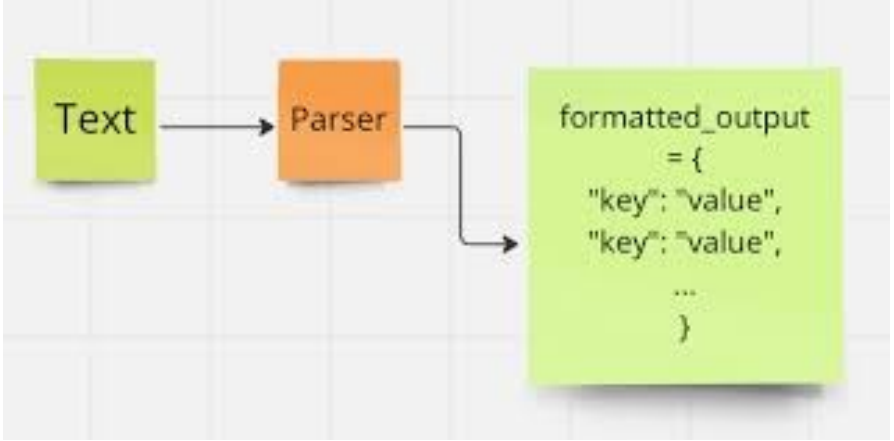


Parser: Procesamiento de datos de salida

El ***parseo*** es el proceso de interpretar y transformar datos para que sean comprensibles y utilizables por modelos de lenguaje y aplicaciones.

¿Para qué sirve?

- ✓ **Interpretar respuestas:** Extraer datos específicos en formatos como JSON, listas o tablas.
- ✓ **Validar entradas:** Verificar que los datos cumplan el formato requerido.
- ✓ **Preparar datos:** Simplificar y estructurar información para el procesamiento por parte del modelo.



Parseo: Procesamiento de datos de salida

Tareas comunes de parser:

- ✓Extraer información clave (entidades, fechas, números).
- ✓Transformar formatos (texto a JSON, XML, listas).
- ✓Dividir y normalizar texto.
- ✓Aplicar lógica de negocio según necesidades de la app.

Formando nuevas generaciones con sello de excelencia comprometidos con la transformación social de las regiones y un país en paz