

# Integrando (L)LM en Aplicaciones



by Ivan Ruiz Rube

# Contenidos

- LLM as a web service
- Generación de contenido
- Servicios inteligentes
- Chatbots
- Servicios web IA

# LLM as a web service

# Acceso a Modelos vía WS

## API REST

Principales proveedores ofrecen acceso vía API REST.

## Facilitadores

Nos elimina la necesidad de entrenar o desplegar ningún LLM

## Caja negra

Obviamos detalles internos de cómo está construido el modelo del lenguaje (arquitectura de red neuronal)

```
/api.openai.com/v1/chat/completions
Content-Type: application/json
Authorization: Bearer $OPENAI_API_KEY

{
  "model": "gpt-4o",
  "messages": [
    {
      "role": "system",
      "content": "You are a helpful assistant"
    },
    {
      "role": "user",
      "content": "Hello!"
    }
  ]
}
```

# Invocando a GPT4o vía curl

```
curl "https://api.openai.com/v1/chat/completions" \  
  -H "Content-Type: application/json" \  
  -H "Authorization: Bearer $OPENAI_API_KEY" \  
  -d '{  
    "model": "gpt-4o-mini",  
    "messages": [  
      {  
        "role": "system",  
        "content": "You are a helpful assistant."  
      },  
      {  
        "role": "user",  
        "content": "Write a haiku that explains the concept of recursion."  
      }  
    ]  
  }'
```






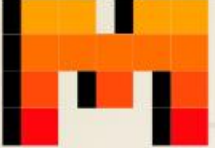

# Invocando a GPT4 vía SDK Python

```
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {
            "role": "user",
            "content": "Write a haiku about recursion in programming."
        }
    ]
)

print(completion.choices[0].message)
```

# Principales SDK de los proveedores de LLM

	<p> Google AI for Developers</p> <p><b>Gemini Developer API   Gemma open models   Google AI for Developers</b></p> <p>Build with Gemini 1.5 Flash and 1.5 Pro using the Gemini API and Google AI Studio, or access our Gemma open models.</p> <a href="#">🔗</a>
<p>Documentation</p> <p><b>Client SDKs</b></p> <p>We provide libraries in Python and</p>	<p> Anthropic</p> <p><b>Client SDKs - Anthropic</b></p> <p>We provide libraries in Python and Typescript that make it easier to work with the Anthropic API.</p> <a href="#">🔗</a>
<p><b>Llama 3.2</b></p> <p> Meta</p>	<p> Meta Llama</p> <p><b>Llama 3.2</b></p> <p>The open source AI model you can fine-tune, distill and deploy anywhere. Our latest models are available in 8B, 70B, and 405B variants.</p> <a href="#">🔗</a>
<p> <b>MISTRAL</b> <b>AI_</b></p> <p><b>Frontier AI</b></p>	<p> docs.mistral.ai</p> <p><b>Clients   Mistral AI Large Language Models</b></p> <p>We provide client codes in both Python and Typescript.</p> <a href="#">🔗</a>



# Retos

## Complejidad de Prompts

Dificultad en creación de prompts efectivos.

## Evolución de Modelos

Rendimiento variable según tarea y modelo. Constante aparición de nuevos de nuevos modelos.

## Reescritura de Código

Necesidad de adaptar código al cambiar de LLM.





# LLM frameworks

## Interfaz unificada

En vez de interactuar directamente con con el modelo, el framework ofrece una una interfaz unificada para acceder a sus a sus funcionalidades.

## Desarrollo simplificado

Los frameworks LLM simplifican el desarrollo al permitir trabajar con distintos modelos de lenguaje sin cambiar el código de la aplicación.

## "Similar" a los ORM

Al igual que un ORM facilita el acceso a acceso a datos en diferentes fuentes de fuentes de datos, un framework LLM LLM permite trabajar con distintos modelos de lenguaje sin cambiar el código de la aplicación.

# SDK Genéricos (frameworks) de Alto Nivel



# Tareas de los LLM

## Generación de contenido

Crear nuevo texto original a partir de una entrada dada.

## Resumen

Condensación de textos extensos.

## Traducción

Conversión entre diferentes idiomas.

## Clasificación

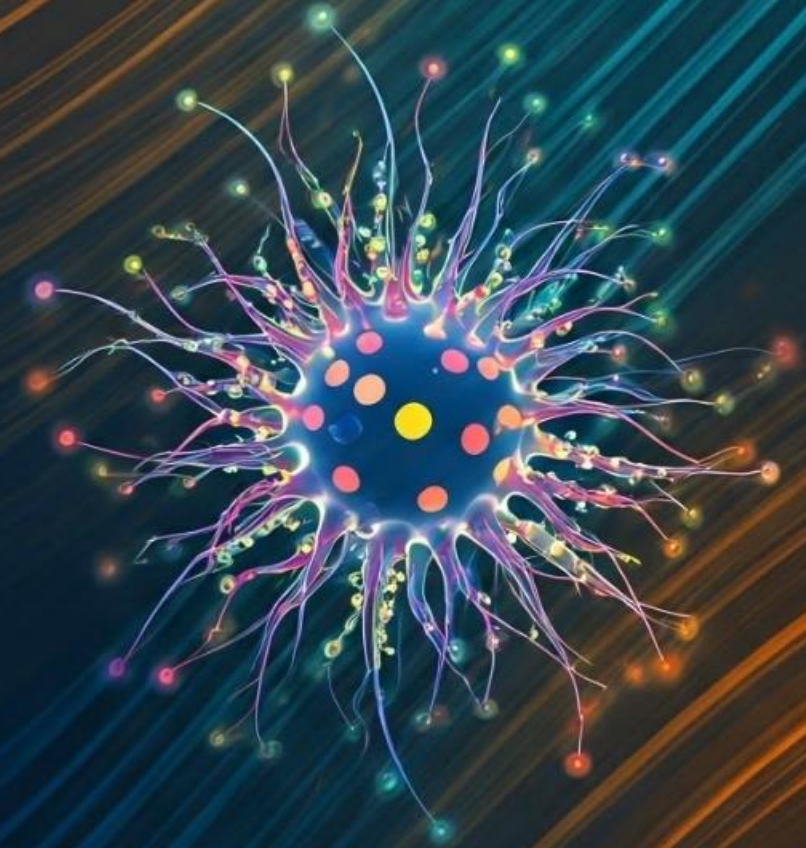
Categorización automática de textos.

## Análisis de textos

Identificar elementos relevantes en un texto, extraer características, etc.

## Q&A

Generación de respuestas a preguntas específicas.



# Generación de contenido

# LangChain4j



Supercharge your Java application with the power of LLMs

# Características de LangChain4J

LangChain4J es un framework Java que simplifica la interacción con modelos de lenguaje grandes (LLMs) y (LLMs) y permite la creación de aplicaciones de IA de última generación.

## Facilidad de uso

Simplifica el desarrollo de aplicaciones basadas en basadas en lenguaje natural.

## Abstracción

LangChain4J ofrece una interfaz unificada para para trabajar con diferentes LLM y bases de datos datos vectoriales

## Adaptabilidad

Estructura modular y flexible para conectar LLMs LLMs con diferentes fuentes de datos, ejecutar ejecutar flujos de trabajo complejos y construir construir agentes inteligentes.

## Escalabilidad

Puede manejar grandes volúmenes de datos y solicitudes de lenguaje natural.

## Integraciones

Conecta LLMs con bases de datos, archivos y otros recursos externos.

## Documentación

Proporciona una documentación completa y ejemplos para facilitar el aprendizaje.



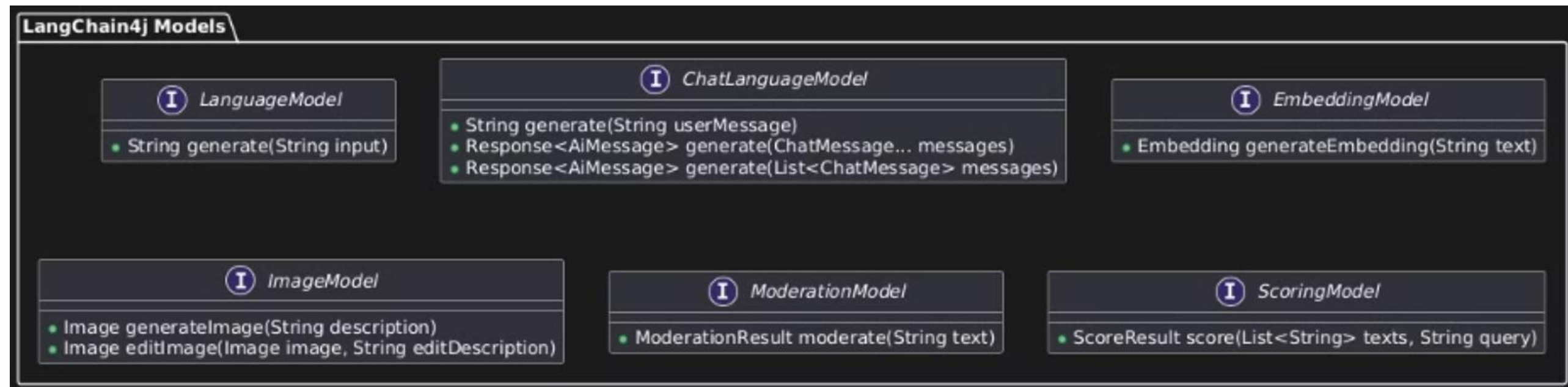


# LangChain4j: Modelos de IA

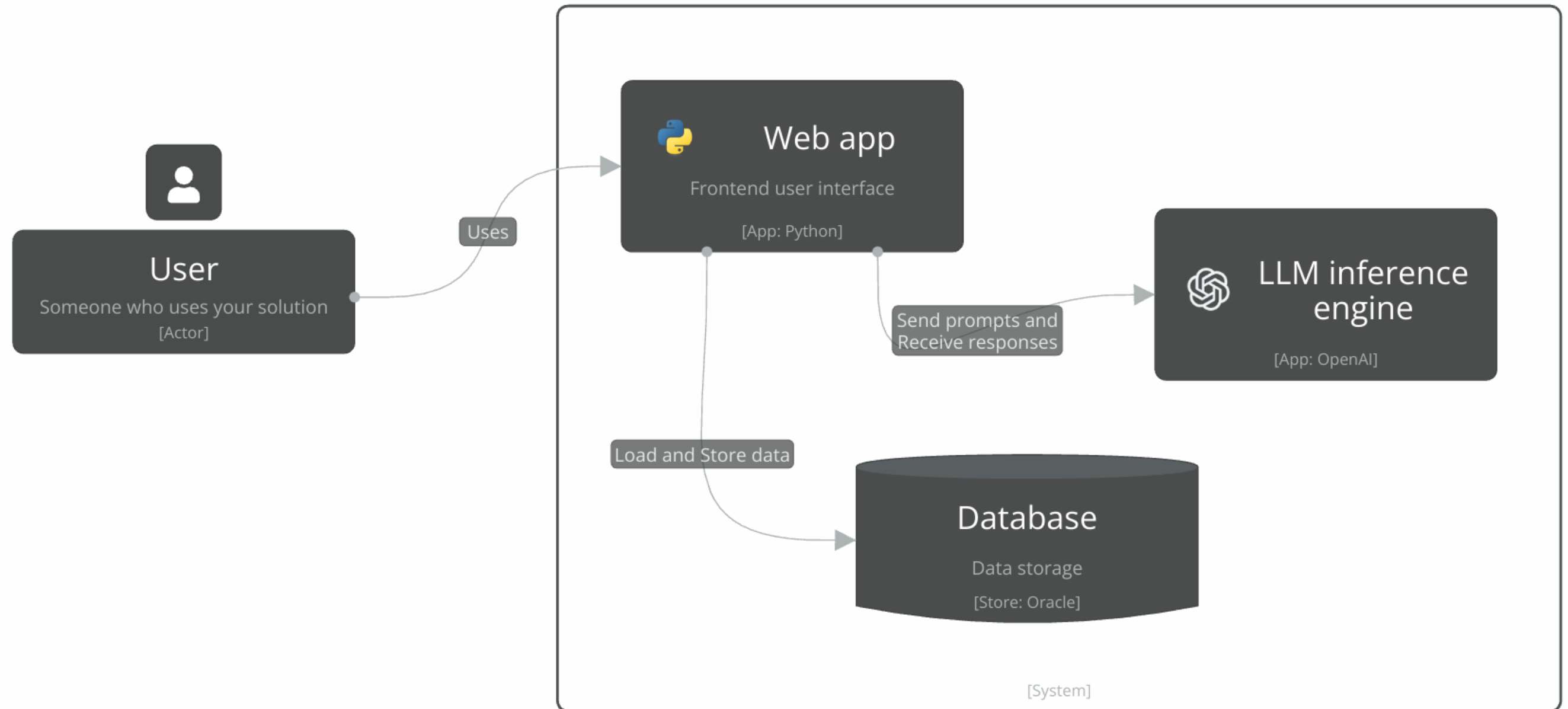
Los proveedores de modelos de lenguaje, como OpenAI o Google, utilizan API propietarias. LangChain4j ofrece una API unificada que simplifica la interacción con estos modelos, evitando la necesidad de aprender e implementar API específicas para cada proveedor.

Gracias a esta API unificada, se puede cambiar fácilmente entre diferentes modelos sin tener que reescribir el código. Actualmente, LangChain4j admite más de 15 proveedores de LLM.

Además de los modelos de lenguaje, LangChain4j permite trabajar con otros modelos de IA, como modelos de imágenes, generación de embeddings, etc.



# Arquitectura básica de una app con AI



# Ej1. Generador de textos



## Ej2. Generador de imágenes





## Ej3. Generador de textos (multimodal)



# Creación de servicios inteligentes



# Servicios inteligentes

Podemos ampliar las capacidades de nuestras aplicaciones al integrar servicios que aprovechen los LLM.

Estos servicios, implementados como métodos en lenguajes orientados a objetos, actúan como intermediarios entre la aplicación y el LLM.

```
public String translate(String userMessage, String language) {  
    // TODO: implementar  
    return "";  
}
```

# Respuestas estructuradas

En ocasiones nos interesa que los LLM devuelvan datos estructurados, como objetos (beans, POJOs), que puedan ser procesados por otras partes de nuestro sistema. El problema reside en que originalmente los LLM generan cadenas de tokens.  
tokens.

```
public Person extractPersonFrom(String userMessage)
    //OMG!
    return null;
}
```

# Prompt especial

Con un prompt específico, podemos solicitar al LLM que nos proporcione los datos en el formato deseado, como XML o JSON. Aunque este JSON. Aunque este enfoque funciona con todos los LLM, no garantiza que siempre se obtenga el formato adecuado.

```
You are an AI designed to return structured data in the form of JSON. You must follow the schema exactly and return the response in valid JSON format without additional commentary or explanation.
```

```
The schema:
```

```
{  
  "name": "string",  
  "age": "integer"  
}
```

```
Please provide the response in this exact format.
```

# Modo JSON

Algunos de los LLM más recientes incorporan ya un modo JSON, que mediante un parámetro adicional en el web service, permite i nformar del formato requerido.

```
POST /v1/chat/completions
{
  "model": "gpt-4o-2024-08-06",
  "messages": [
    {
      "role": "system",
      "content": "You are a helpful math tutor."
    },
    {
      "role": "user",
      "content": "solve 8x + 31 = 2"
    }
  ],
  "response_format": {
    "type": "json_schema",
    "json_schema": {
      "name": "math_response",
      "strict": true,
      "schema": {
        "type": "object",
        "properties": {
          "steps": {
            "type": "array",
            "items": {
              "type": "object",
              "properties": {
                "explanation": {
                  "type": "string"
                },
                "output": {
                  "type": "string"
                }
              }
            },
            "required": ["explanation", "output"],
            "additionalProperties": false
          }
        },
        "required": ["explanation", "output"],
        "additionalProperties": false
      }
    }
  },
  "final_answer": {
    "type": "string"
  }
}
```

# Flujo de un servicio (powered by IA)

1

## Procesar los parámetros de entrada

El servicio recibe los parámetros de entrada del cliente.

2

## Construir el prompt

Se crea un prompt con la información necesaria para que el LLM comprenda la petición.

3

## Invocar al LLM

El servicio envía el prompt al LLM para obtener una respuesta.

4

## Obtener la respuesta del LLM

El LLM procesa el prompt y devuelve una respuesta al servicio.

5

## Preparar la respuesta

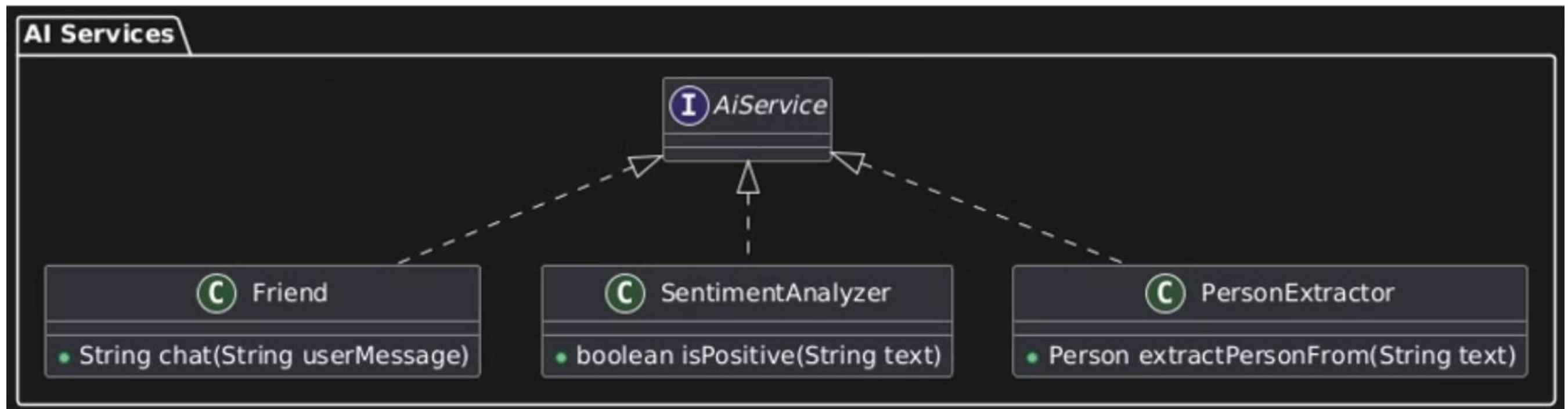
El servicio formatea la respuesta del LLM de acuerdo a la solicitud del cliente.

# LangChain4j: Servicios de IA

Funciona de manera similar a **Spring Data JPA**, donde defines una interfaz declarativa, y **LangChain4j** proporciona un objeto proxy que la implementa.

Es como un componente de la capa de servicios que ofrece capacidades de IA.

**AI Services** maneja operaciones comunes como: Formatear entradas y analizar salidas de los LLM, además de soporte para memoria, uso de herramientas y RAG





## Ej 4. Creación de un traductor



# Ej 5. Creación de un "resumidor" de textos



## Ej 6. Creación de un analizador de *sentimientos*



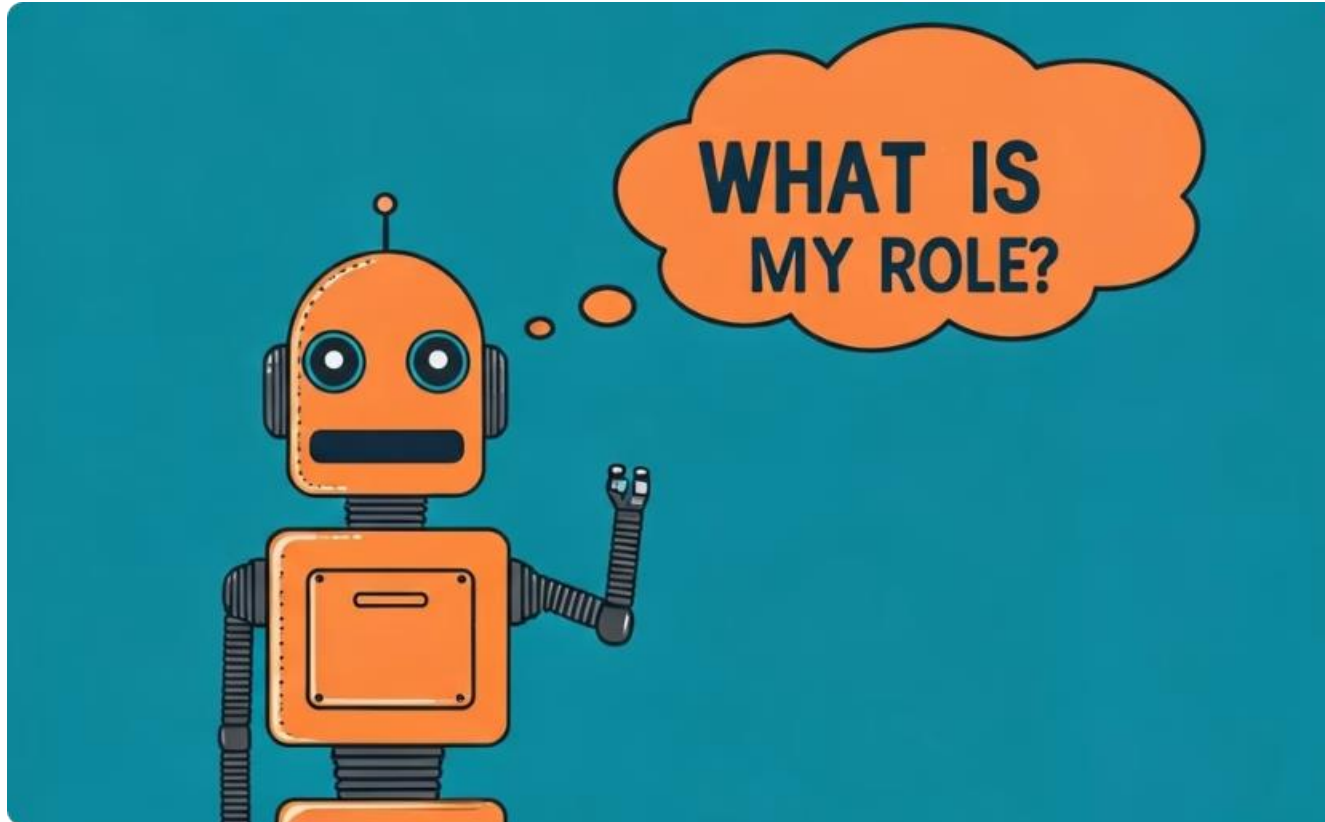


# Ej 7. Creación de un extractor de datos



# Chatbots

# Consideraciones



## Descripción del rol del bot

Se debe proporcionar un **prompt del sistema** que describa el rol del bot, por ejemplo: "Eres un chatbot muy chistoso con el que puedes charlar y entretenerte".

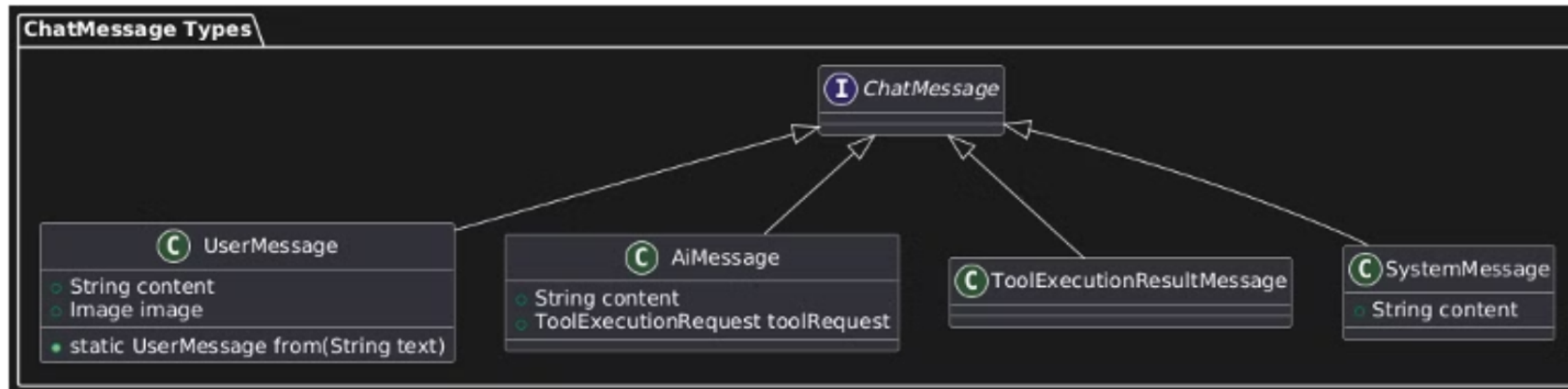


## Flujo de tokens de salida

Las respuestas deben emitirse en **streaming**, sin esperar que finalice el proceso de inferencia, para evitar interrupciones en la experiencia del usuario.



# LangChain4J: Tipos de mensajes



## Ej 8. Creación de un chat



# Limitación importante de los LLM

**Usuario:**

*“¿Dónde está la Universidad de Cádiz?”*

**Asistente:**

*“La Universidad de Cádiz (UCA) se encuentra ubicada en la ciudad de Cádiz, en la comunidad autónoma de Andalucía.”*

**Usuario:**

*“¿Qué te he preguntado antes?”*

**Asistente:**

*“No lo sé”*

# Historia de las conversaciones

## Sin Estado

Los LLM no almacenan información entre interacciones. No recuerdan conversaciones pasadas. Esto limita la capacidad de mantener un mantener un contexto coherente en las conversaciones.

## Necesidad de memoria

Necesitamos un mecanismo para guardar la información de las conversaciones previas del usuario. La memoria permite al sistema contextualizar mejor las preguntas y ofrecer respuestas más precisas. precisas.



# Memoria de las conversaciones

## Técnica

Para superar esta limitación, en cada interacción con el LLM se envía una parte de la conversación mantenida entre el usuario y la IA.

## Ventana de contexto

Cantidad de información que la IA puede recibir y generar en una determinada inferencia del modelo.

## Limitación

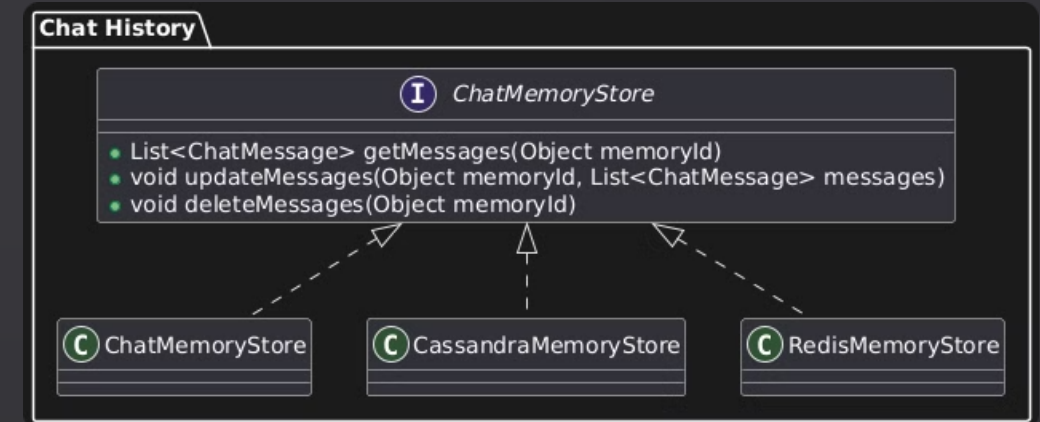
Los LLM tienen una ventana de contexto limitada, lo que significa que solo pueden procesar una cantidad fija de tokens en cada en cada interacción.

# LangChain4J: Almacén de memoria

El almacén de memoria es un componente clave en LangChain4j que guarda guarda información relevante de conversaciones anteriores.

Esto permite que la IA recuerde información anterior, mejorando la calidad de las calidad de las interacciones con el usuario.

Los almacenes de memoria pueden implementar diferentes estrategias de persistencia, desde bases de datos hasta archivos.



# Estrategias para la memoria



## Mensajes antiguos

Descartar los mensajes más antiguos.



## Mensajes menos relevantes

Descartar los mensajes menos relevantes.  
relevantes.



## Resumir contenido

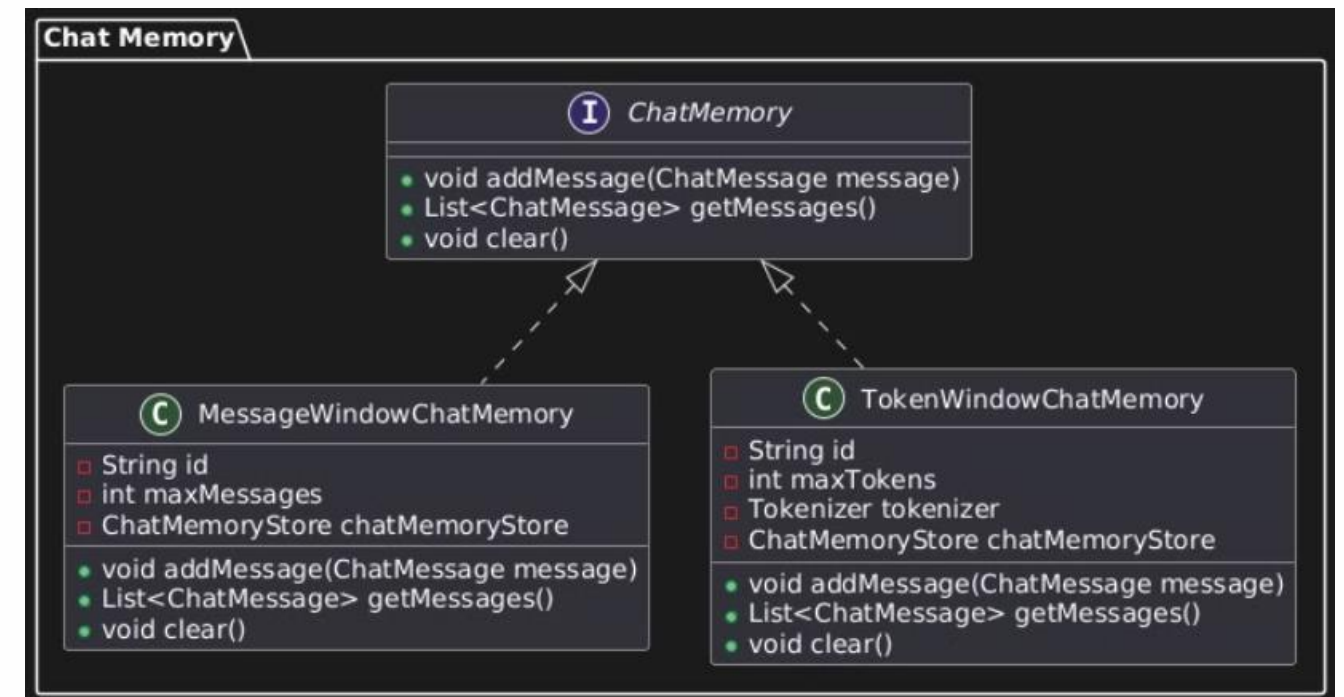
Resumir el contenido de los mensajes.

# LangChain4j: Gestión de la memoria

LangChain4j ofrece diferentes estrategias para administrar la memoria de las conversaciones.

Estos mecanismos permiten al asistente recordar información relevante de las conversaciones anteriores, mejorando la calidad de las respuestas.

La memoria se puede configurar para almacenar información de diferentes tipos y tamaños.





# Ej 9. Creación de un chat (con memoria)

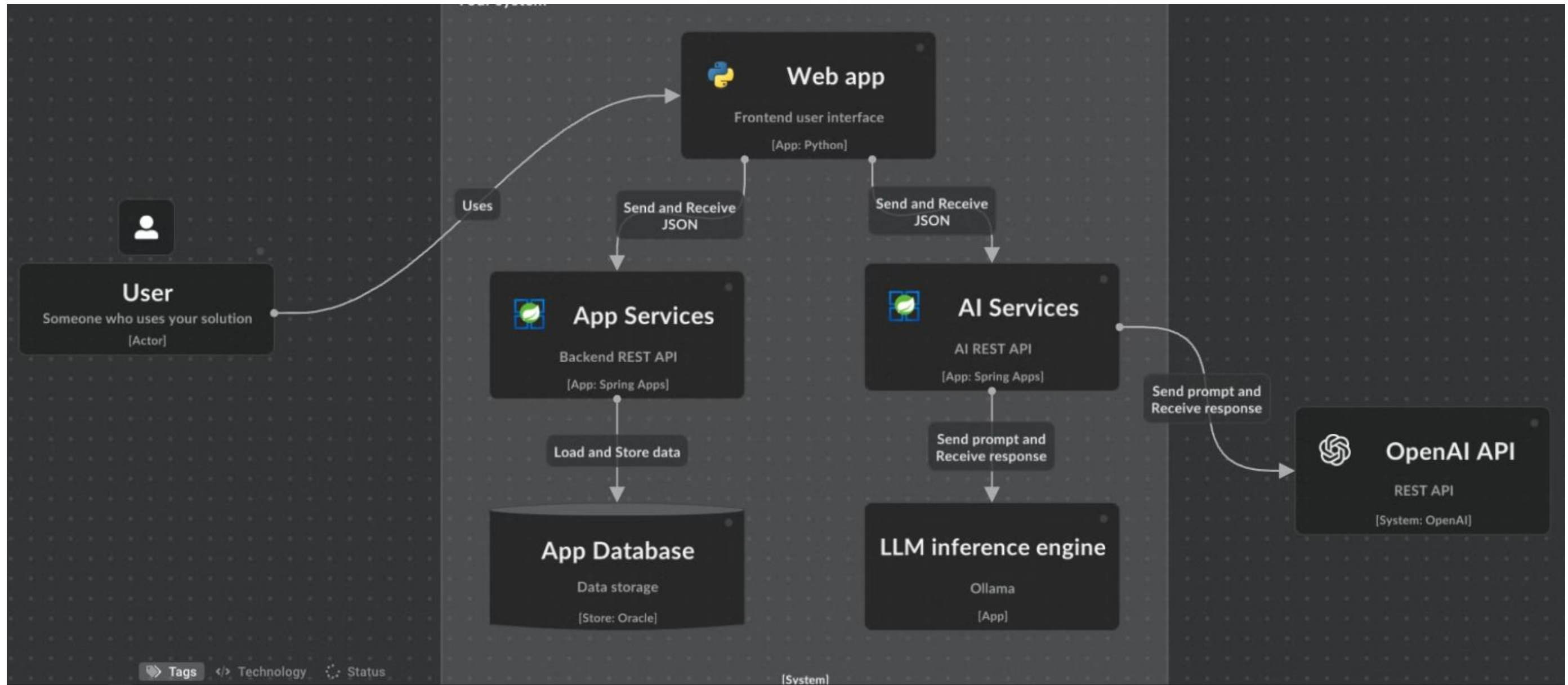


# Creación de servicios web con IA

# Limitaciones de la arquitectura básica

- Es frecuente cambiar de LLM en función de las necesidades. Por ejemplo, por la aparición de algún modelo más potente, más barato o con menor latencia.
- Si necesitamos cambiar el LLM que respalda un servicio "inteligente" (por ejemplo, resumir texto) etc., tendremos que actualizar las cadenas de conexión (URL, API Key, etc.) en todas las aplicaciones que lo utilicen.
- Si realizamos una refactorización del prompt de sistema utilizado en algún servicio de IA, será necesario actualizarlo también en cada una de las aplicaciones afectadas.
- Si quisiéramos replicar el mismo servicio de IA en aplicaciones escritas en diferentes lenguajes de programación, tendríamos que reescribirlo en cada app afectada.
- Por todo ello, es más adecuado optar por una arquitectura de servicios para nuestros servicios de IA.

# Arquitectura (SOA) de una app con AI



# Ej 10. Creación de un servicio web de chat



# Resumen

La integración de LLM en aplicaciones es algo cada vez más común. La capacidad de los LLM para comprender y generar lenguaje natural abre un mundo de posibilidades para aplicaciones que pueden interactuar con los usuarios de una manera más cómoda.

Es importante elegir el LLM adecuado para la tarea, y luego integrar la IA en la aplicación de una manera segura y eficiente. LangChain4J ofrece una solución para afrontar estos desafíos.