Integrating (L)LM into Applications

Andrés Muñoz / Iván Ruiz

Contents

- LLM as a web service
- Content generation
- Intelligent services
- Chatbots
- AI web services

LLM as a web service

Access to Models via WS

REST API

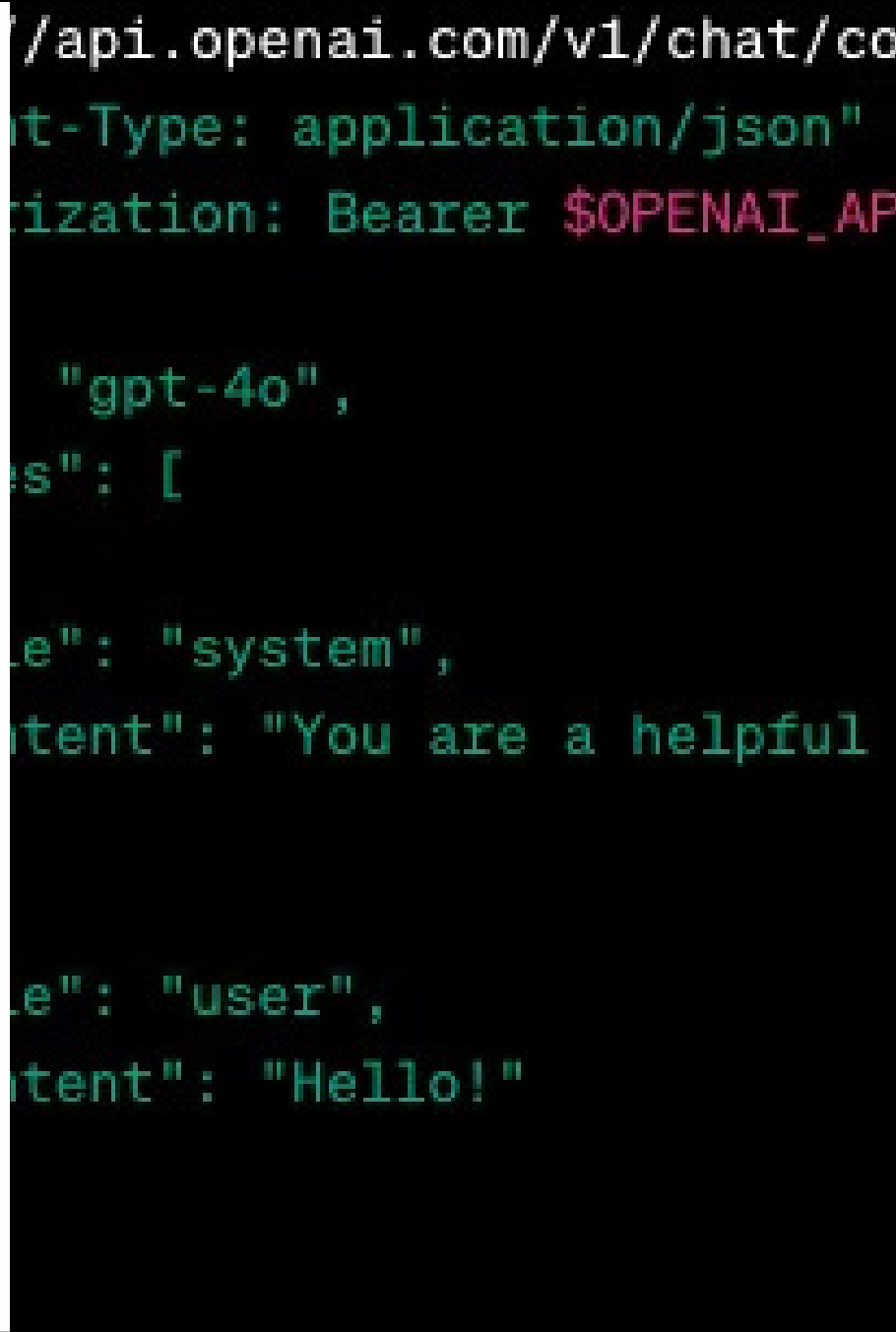
Major providers offer access via REST API.

Facilitators

It eliminates the need to train or deploy any LLM

Black box

We avoid the internal details of how the language model is built (neural network architecture)



Invoking GPT4o via 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."
            }
        ]
    }'
```

Invoking GPT4 via Python SDK

platform.openai.com

OpenAI Platform

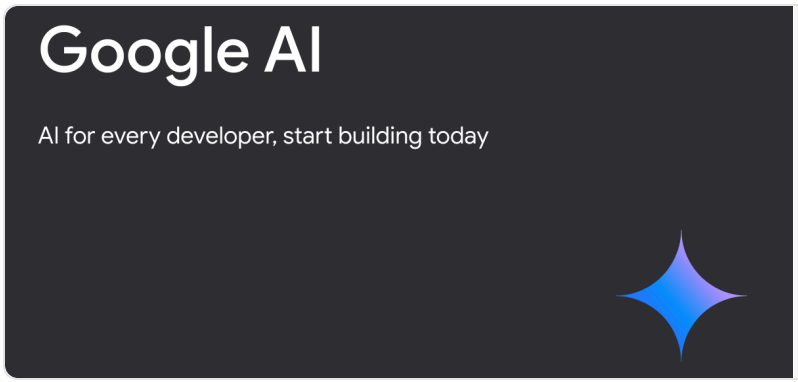
Explore developer resources, tutorials, API docs, and dynamic examples to get the most out of OpenAI's platform.

```
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)
```

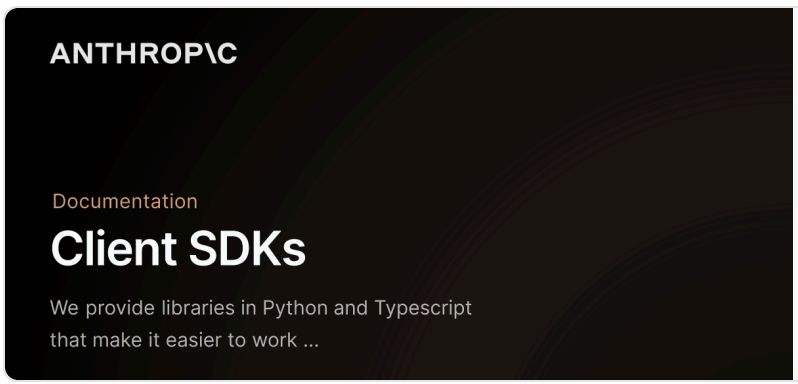
Major LLM Provider SDKs



Google AI for Developers

Gemini Developer API | Gemma open models | Google AI for Develope...

Build with Gemini 2.0 Flash, 2.5 Pro, and Gemma using the Gemini API and Google AI Studio.



Anthropic

Client SDKs – Anthropic

We provide libraries in Python and Typescript that make it easier to work with the Anthropic API.

Industry Leading, Open-Source AI | Llama by Meta

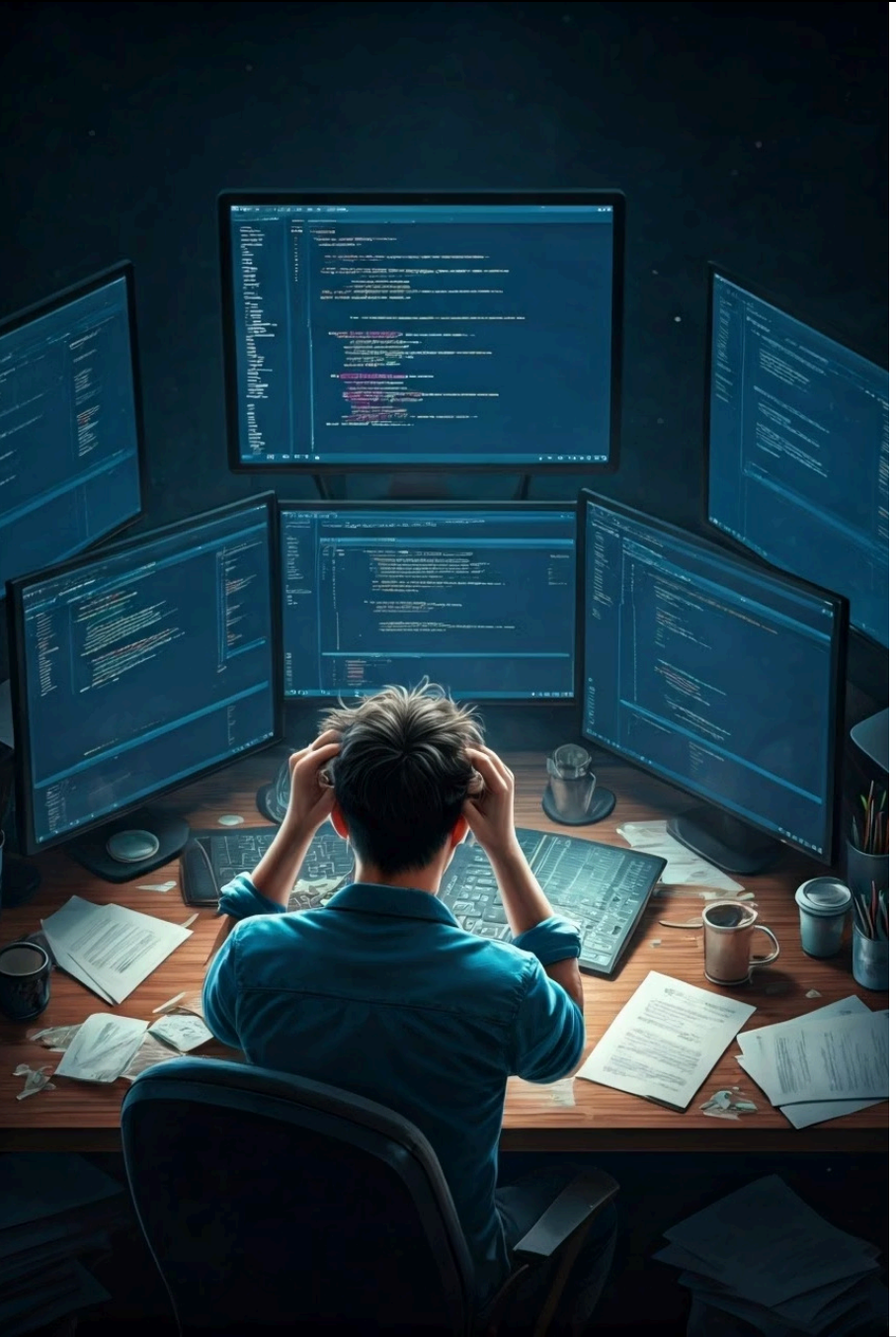
Industry Leading, Open-Source AI | Llama by Meta

Discover Llama 4's class-leading AI models, Scout and Maverick. Experience top performance, multimodality, low costs, and unparalleled efficiency.

docs.mistral.ai

Clients | Mistral AI Large Language Models

We provide client codes in both Python and Typescript.



Challenges

Prompt Complexity

Difficulty in creating effective prompts.

Model Evolution

Variable performance depending on task and model. Constant emergence of new models.

Code Rewriting

Need to adapt code when changing LLM.

LLM frameworks

Unified interface

Instead of interacting directly with the model, the framework offers a unified interface to access its functionalities.

Simplified development

LLM frameworks simplify development by allowing you to work with different language models without changing the application code.

"Similar" to ORMs

Just as an ORM facilitates access to data from different data sources, an LLM framework allows you to work with different language models without changing the application code.

High-Level Generic (Frameworks) SDKs


LangChain


Semantic Kernel


haystack
by deepset


LlamaIndex

LLM Tasks

Content Generation

Create original new text from a given input.

Summarisation

Condensing lengthy texts.

Translation

Conversion between different languages.

Classification

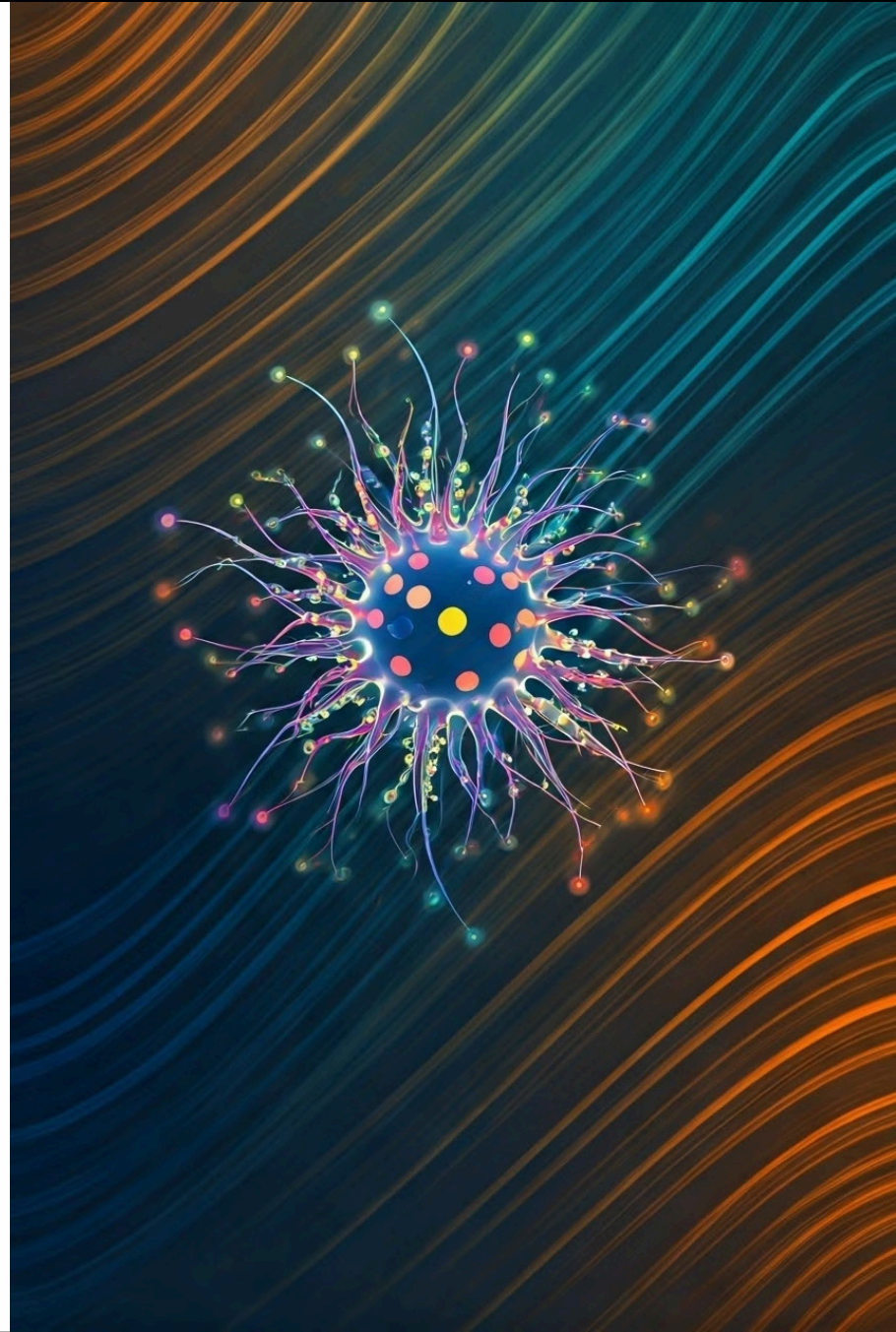
Automatic categorisation of texts.

Text Analysis

Identify relevant elements in a text, extract features, etc.

Q&A

Generating answers to specific questions.



Content Generation

Introduction to LangChain4j

LangChain4j

Supercharge your Java application with the power of LLMs

Features of LangChain4J

LangChain4J is a Java framework that simplifies the interaction with large language models (LLMs) and enables the creation of cutting-edge AI applications.

Ease of Use

Simplifies the development of natural language-based applications.

Abstraction

LangChain4J offers a unified interface to work with different LLMs and vector databases

Adaptability

Modular and flexible structure to connect LLMs with different data sources, execute complex workflows and build intelligent agents.

Scalability

Can handle large volumes of data and natural language requests.

Integrations

Connects LLMs with databases, files and other external resources.

Documentation

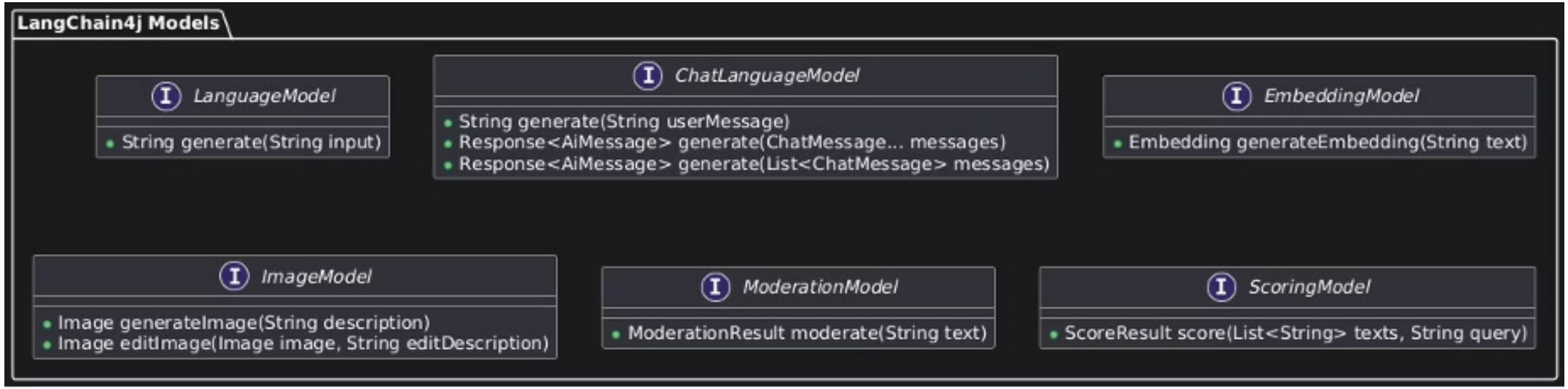
Provides comprehensive documentation and examples to facilitate learning.

LangChain4j: AI Models

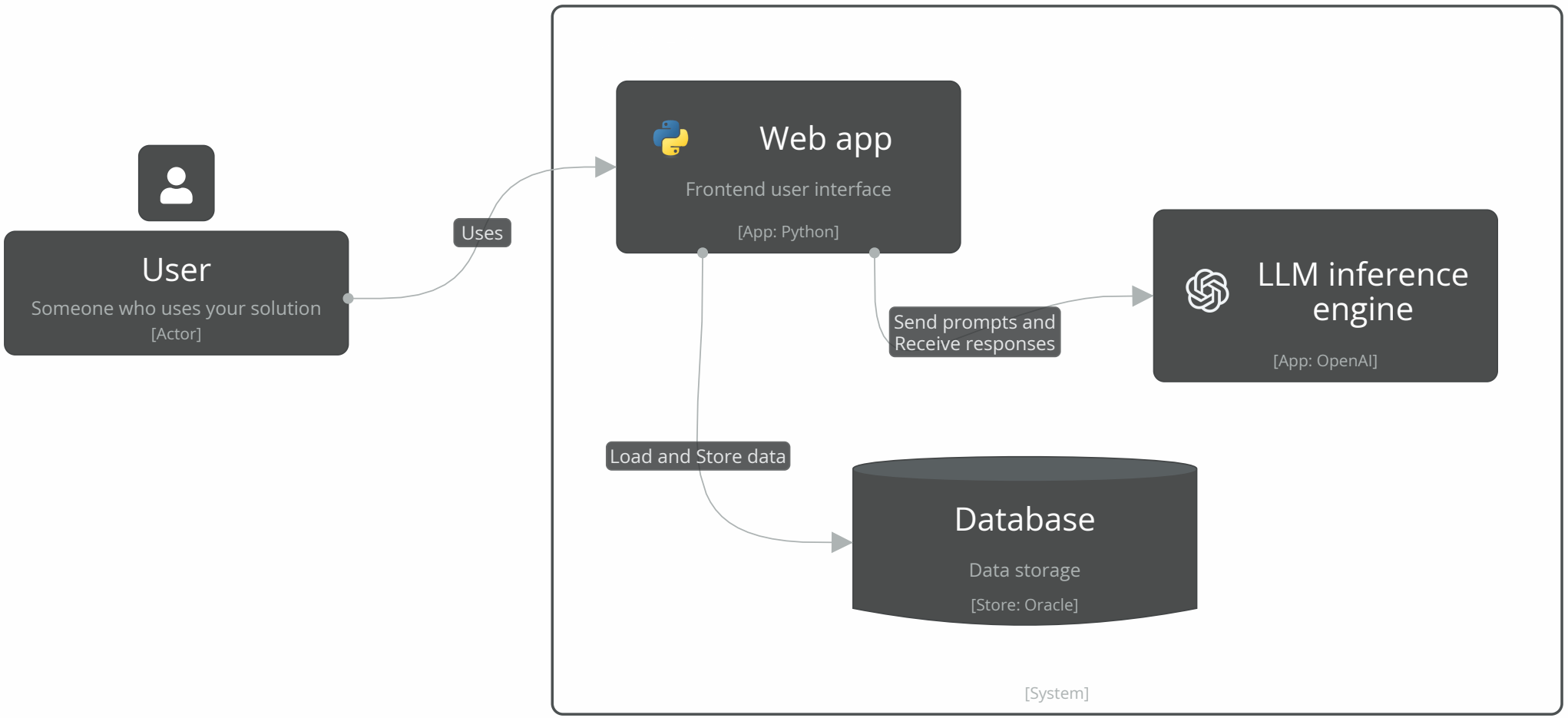
Language model providers, such as OpenAI or Google, use proprietary APIs. LangChain4j offers a unified API that simplifies the interaction with these models, avoiding the need to learn and implement specific APIs for each provider.

Thanks to this unified API, you can easily switch between different models without having to rewrite the code. Currently, LangChain4j supports more than 15 LLM providers.

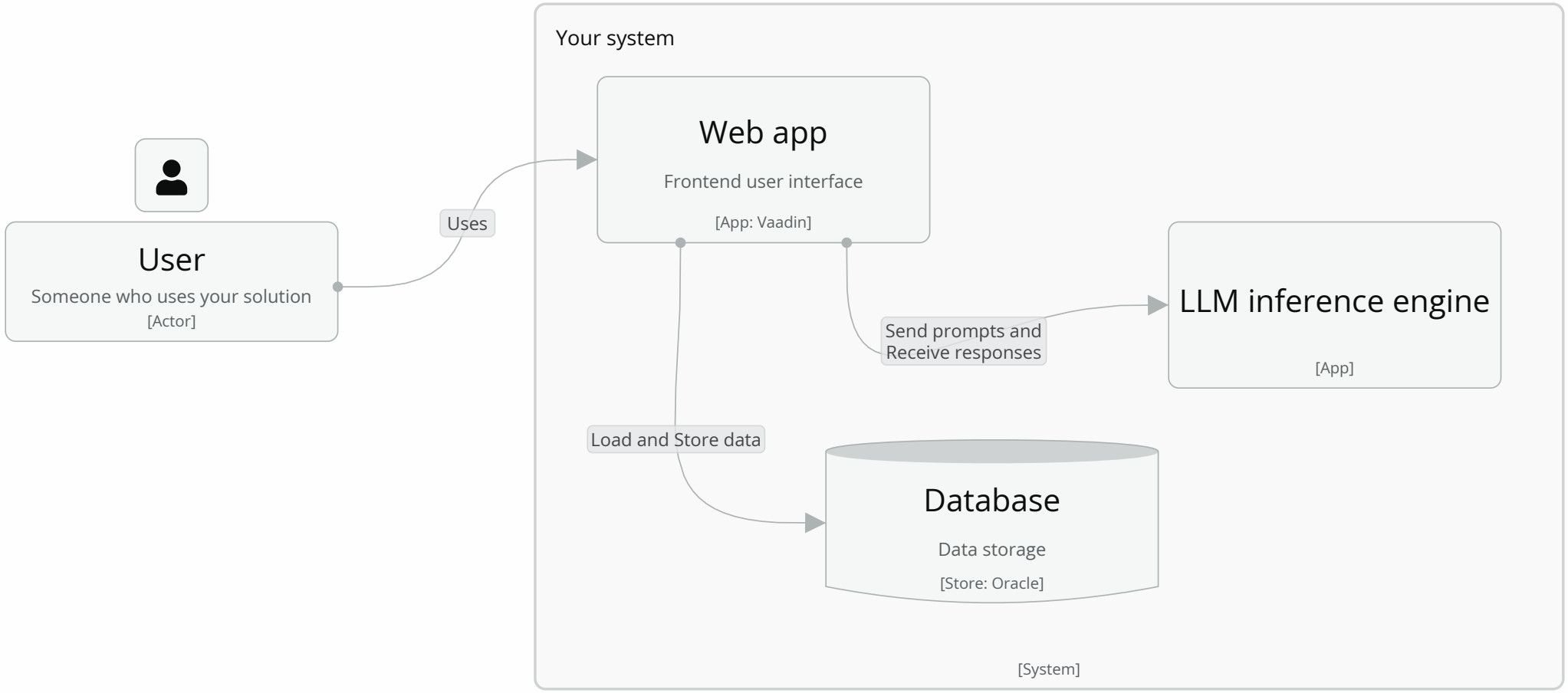
In addition to language models, LangChain4j allows you to work with other AI models, such as image models, embedding generation, etc.



Basic Architecture of an AI App



Basic Architecture of an AI App



Ex1. Text Generator

Ex2. Image Generator

Ex. 3 Multimodal

Creating Intelligent Services

Intelligent Services

We can expand the capabilities of our applications by integrating services that leverage LLMs.

These services, implemented as methods in object-oriented languages, act as intermediaries between the application and the LLM.

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

Structured Responses

Sometimes we want the LLMs to return structured data, such as objects (beans, POJOs), that can be processed by other parts of our system. The problem is that originally the LLMs generate sequences of tokens.

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

Special Prompt

With a specific prompt, we can ask the LLM to provide the data in the desired format, such as XML or JSON. Although this approach works with all LLMs, it does not guarantee that the appropriate format will always be obtained.

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.

JSON Mode

Some of the most recent LLMs already incorporate a JSON mode, which by means of an additional parameter in the web service, allows you to specify the required format.

```
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
            }
          },
          "final_answer": {
            "type": "string"
          }
        },
        "required": ["steps", "final_answer"],
        "additionalProperties": false
      }
    }
  }
}
```

Workflow of a service (powered by AI)

1

Process input parameters

The service receives the client's input parameters.

2

Construct the prompt

A prompt is created with the necessary information for the LLM to understand the request.

3

Invoke the LLM

The service sends the prompt to the LLM to obtain a response.

4

Obtain the LLM's response

The LLM processes the prompt and returns a response to the service.

5

Prepare the response

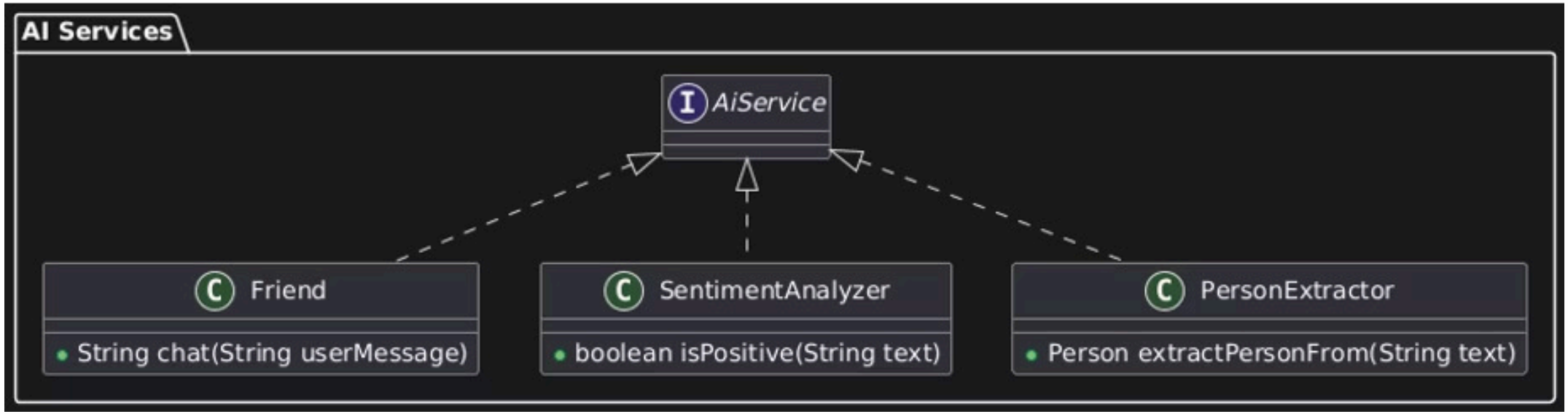
The service formats the LLM's response according to the client's request.

LangChain4j: AI Services

It works similarly to **Spring Data JPA**, where you define a declarative interface, and **LangChain4j** provides a proxy object that implements it.

It's like a component of the service layer that offers AI capabilities.

AI Services handles common operations such as: Formatting inputs and analysing outputs of LLMs, as well as support for memory, tool use and RAG



Exercise 4. Creating a translator

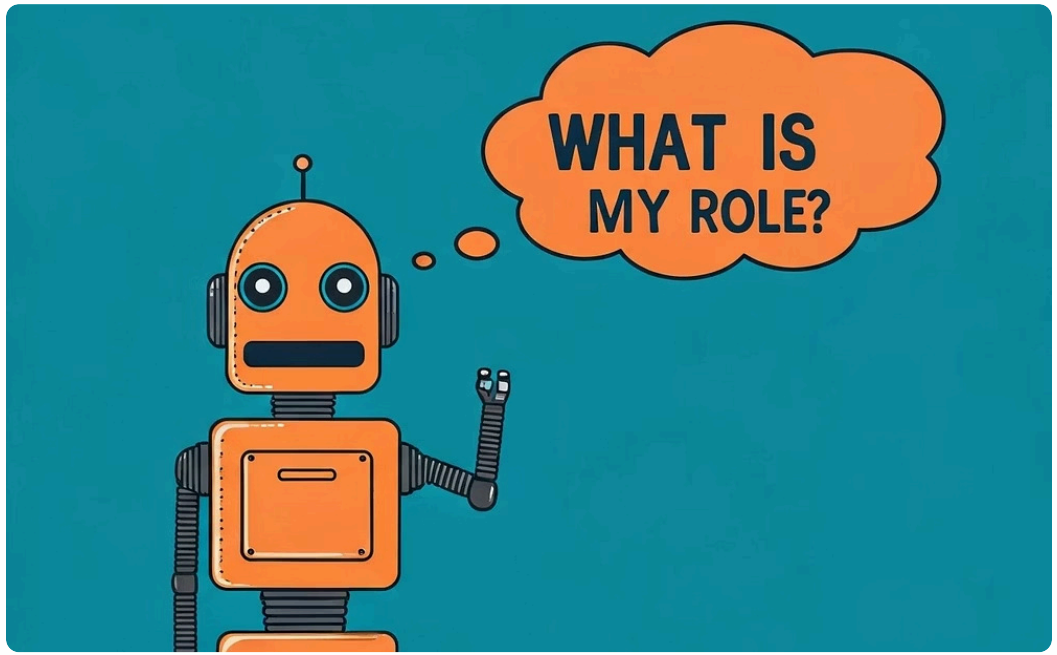
Ex 5. Creating a text summariser

Ex 6. Creating a *sentiment* analyser

Exercise 7. Creating a data extractor

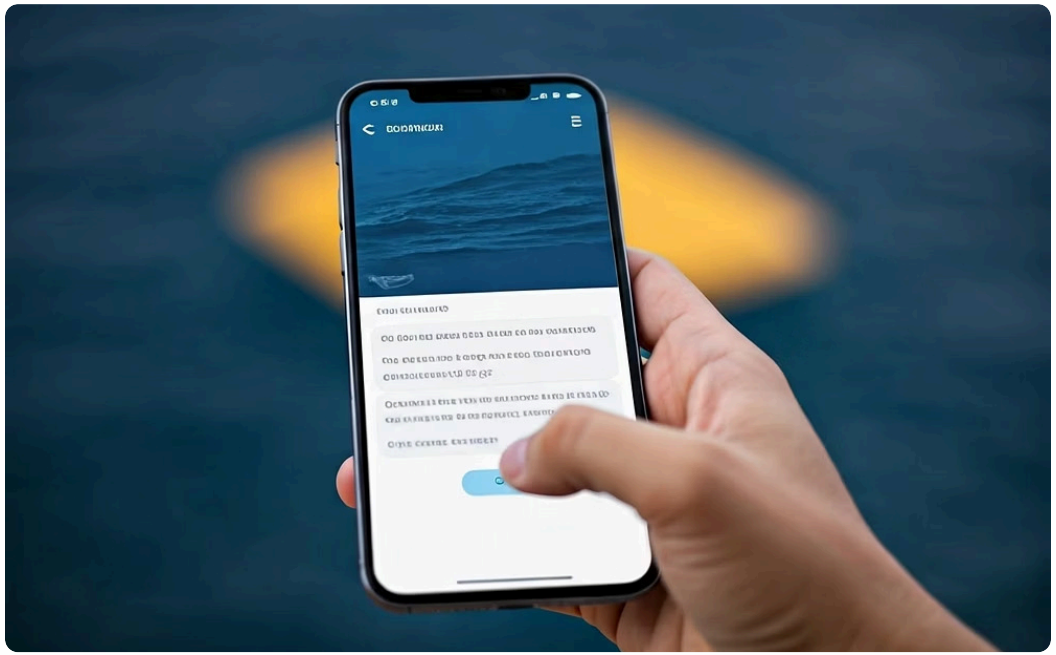
Chatbots

Considerations



Description of the bot's role

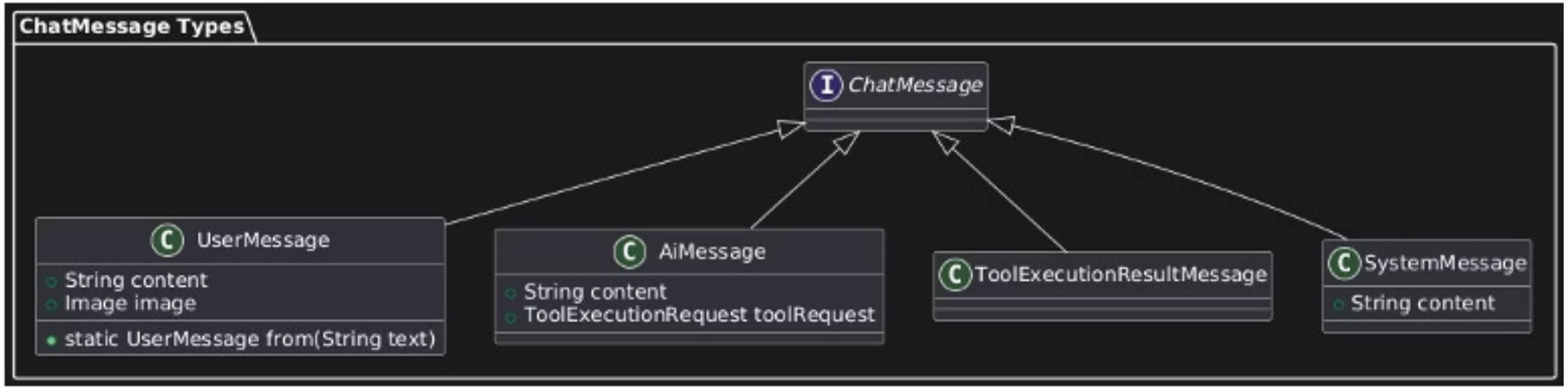
A **system prompt** should be provided that describes the bot's role, for example: "You are a very funny chatbot that you can chat and have fun with".



Output token flow

Responses should be emitted in **streaming**, without waiting for the inference process to complete, to avoid interruptions in the user experience.

LangChain4J: Types of Messages



Ex 8. Creating a chat

Important Limitation of LLMs

User:

"Where is the University of Cádiz?"

Assistant:

"The University of Cádiz (UCA) is located in the city of Cádiz, in the autonomous community of Andalusia."

User:

"What did you ask me before?"

Assistant:

"I don't know"

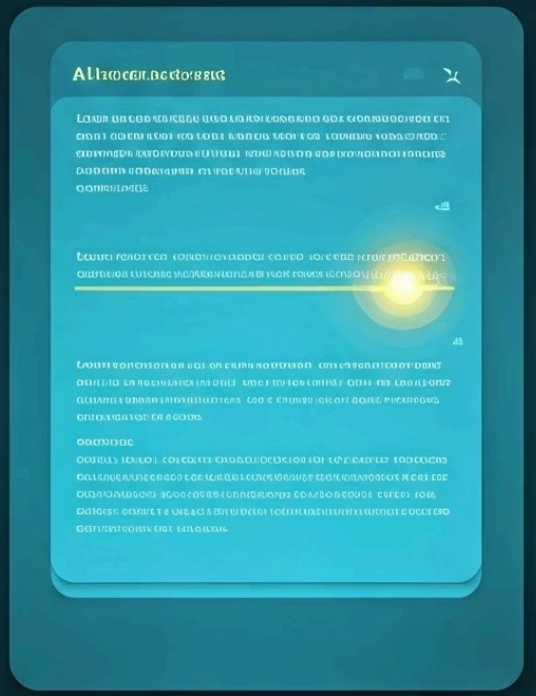
Conversation History

Stateless

LLMs do not store information between interactions. They do not remember past conversations. This limits their ability to maintain a coherent context in conversations.

Need for Memory

We need a mechanism to store the user's previous conversation history. Memory allows the system to better contextualise questions and provide more accurate responses.



Memory of the conversations

Technique

To overcome this limitation, in each interaction with the LLM, a part of the conversation maintained between the user and the AI is sent.

Context window

Amount of information that the AI can receive and generate in a given model inference.

Limitation

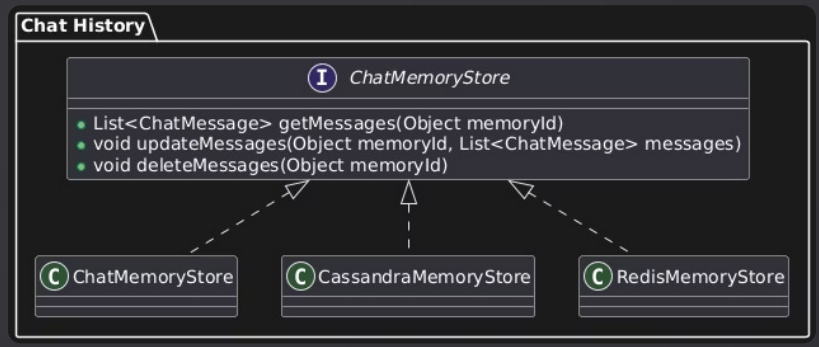
LLMs have a limited context window, which means they can only process a fixed amount of tokens in each interaction.

LangChain4J: Memory Store

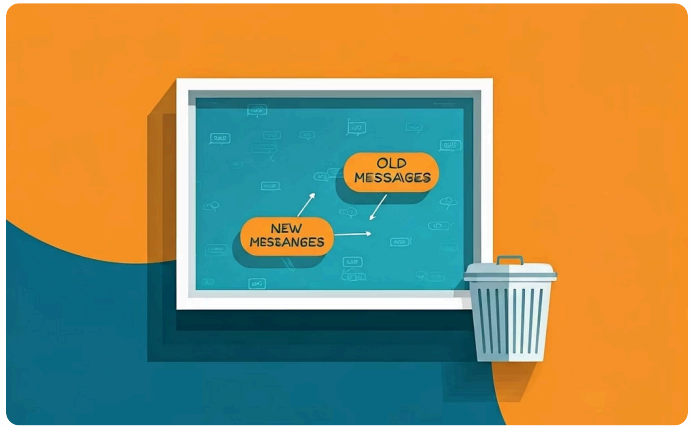
The memory store is a key component in LangChain4j that stores relevant information from previous conversations.

This allows the AI to remember previous information, improving the quality of interactions with the user.

Memory stores can implement different persistence strategies, from databases to files.

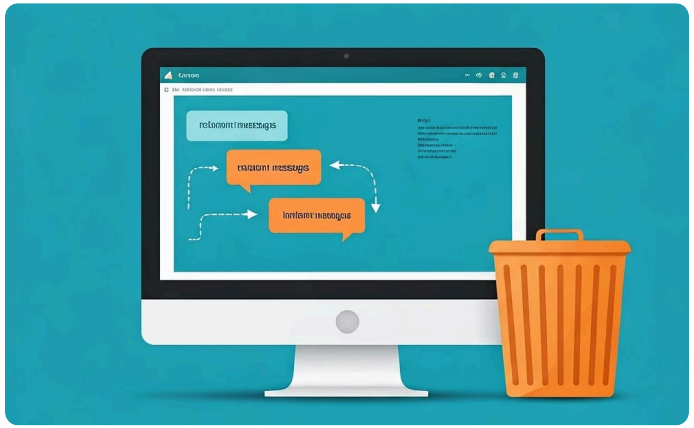


Memory Strategies



Old Messages

Discard the oldest messages.



Less Relevant Messages

Discard the less relevant messages.

Summarise Content

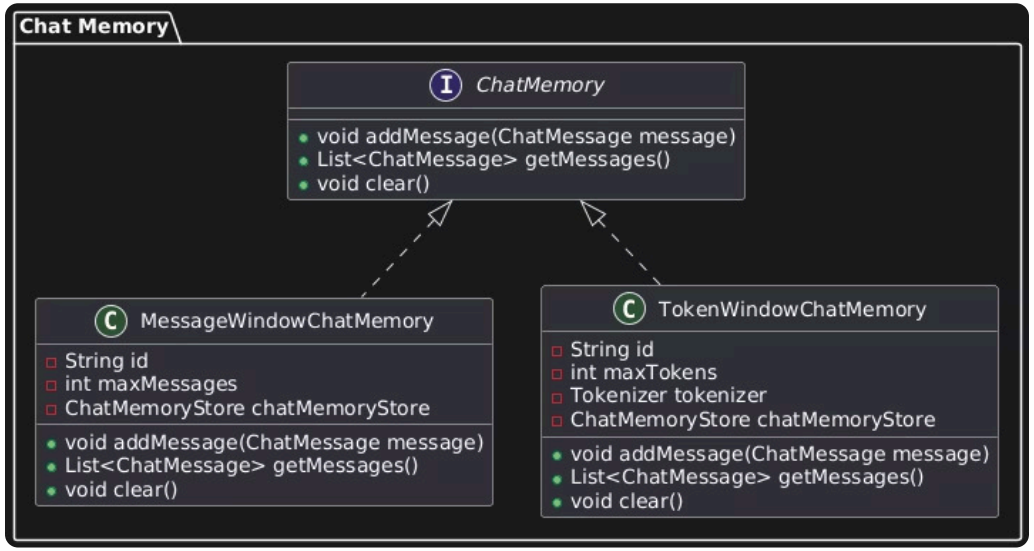
Summarise the content of the messages.

LangChain4j: Memory Management

LangChain4j offers different strategies for managing conversation memory.

These mechanisms allow the assistant to remember relevant information from previous conversations, improving the quality of the responses.

Memory can be configured to store information of different types and sizes.



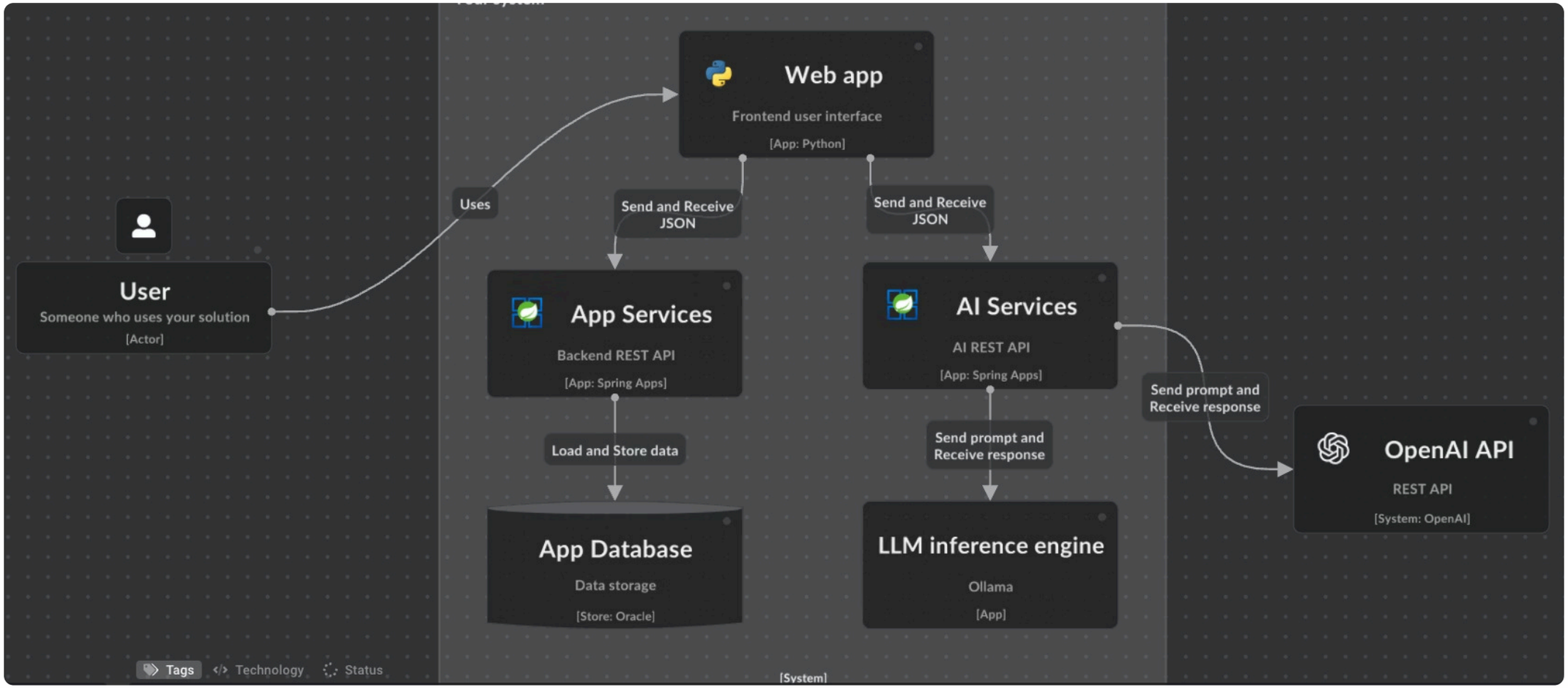
Ex 9. Creating a chat (with memory)

Creating web services with AI

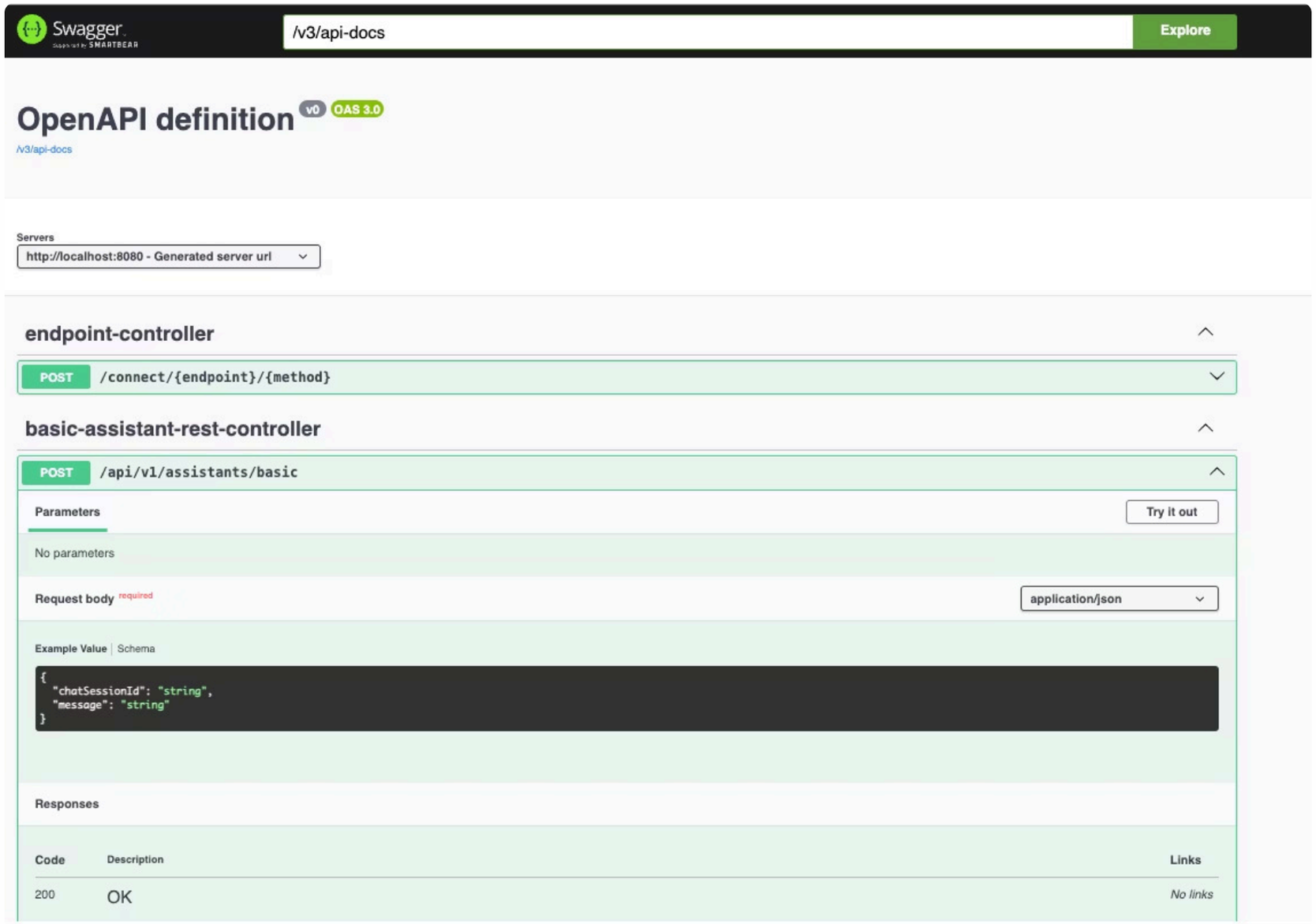
Limitations of the basic architecture

- It is common to change the LLM based on needs. For example, due to the emergence of a more powerful, cheaper or lower latency model.
- If we need to change the LLM that supports an "intelligent" service (for example, to summarise text), etc., we will have to update the connection strings (URL, API Key, etc.) in all the applications that use it.
- If we refactor the system prompt used in any AI service, it will also be necessary to update it in each of the affected applications.
- If we wanted to replicate the same AI service in applications written in different programming languages, we would have to rewrite it in each affected app.
- For all these reasons, it is more appropriate to opt for a service-oriented architecture for our AI services.

Architecture (SOA) of an AI app



Web AI Services



Ex 10. Creating a chat web service

Summary

The integration of LLMs into applications is becoming increasingly common. The ability of LLMs to understand and generate natural language opens up a world of possibilities for applications that can interact with users in a more comfortable way.

It is important to choose the right LLM for the task, and then integrate the AI into the application in a safe and efficient manner. LangChain4J offers a solution to tackle these challenges.