

Spring AI

컴퓨터에게 배우는 Spring 기반 LLM & RAG 서비스 개발



강사님 소개

허제민 님

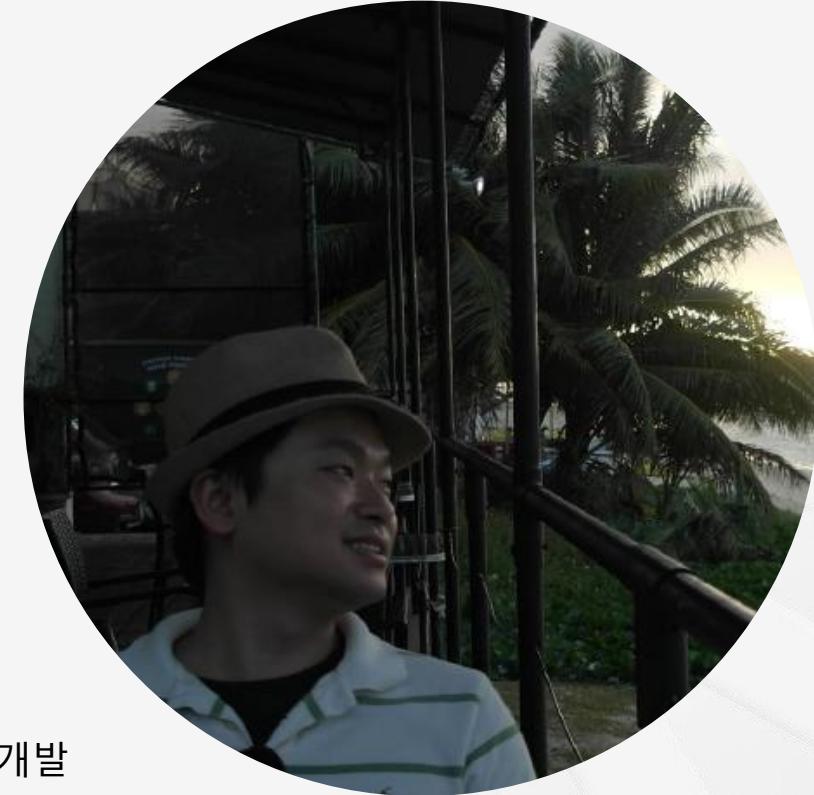
- SK플래닛 SW Engineer (2013.4 ~ 2025. 5)
- 삼성SDS 선임 / 연구원, Technical Architect (2009.2 ~ 2013.4)

[프로젝트]

- SK플래닛 Python 기반의 ML 챗봇과 RAG를 사용하는 LLM 챗봇 플랫폼 개발

[오픈소스 활동]

- Spring AI Contributor (Elasticsearch, OpenSearch, SimpleVectorStore)
- 2013 공개SW 개발자대회 금상 (2위) JMCloud-ComputeManager: AWS EC2 관리용 GUI 도구



Spring AI 소개

Spring AI 엔터프라이즈 어플리케이션 개발 프레임워크

Spring AI 엔터프라이즈 어플리케이션 개발 프레임워크

Spring 생태계와의
완벽한 통합

추상화 및 유연성



엔터프라이즈 데이터 및
API 연동

AI 통합의 간소화
(AI Models, Vector
Databases)

2025년 5월 20일 [Spring AI 1.0 GA Released](#)

Spring AI 1.0 GA 히스토리

Spring AI 1.0 GA 히스토리

2023년 8월 Spring One Spring AI 소개

- Python의 LangChain,
LlamaIndex에서 영감을 받아 Java
Spring 개발자에게 친숙한 API 제공
- 다양한 LLM(대형 언어 모델)
제공자(OpenAI, Azure OpenAI
등)와의 일관된 연동 추상화 제공

Commit 8139af4



JM-Lab authored and tzolov committed on Dec 13, 2023

Add OpenAI Stream Client implementation

- Add the latest Request and Response Classes of OpenAI's Chat Completion API
- Add spring-boot-starter-webflux dependency,
- Implement AiStreamClient interface extending AiClient,
- Implement OpenAiStreamClient using Reactor Flux
- AiStreamClient doesn't extend AiClient

赞扬 main . v1.0.0 ... v0.8.0

Spring AI 1.0 GA 히스토리

2024년 5월 Spring AI 1.0.0-M1 시작

- Spring Initializr(start.spring.io) 등록
 - 2024년 2월 29일, 0.8.x 버전
- 1.0.0-M1 ~ M5
 - ChatClient 구조 개편
ChatModel과 ChatClient로 분리
 - 여러 AI Model 지원
 - VectorStore 설정 표준화와 빌더 패턴 도입
 - ModelClient 를 Model 로 변경

Commit 9334d7d



JM-Lab authored and **tzolov** committed on Mar 20, 2024

Add Elasticsearch vector store integration

- Implement ElasticsearchVectorStore and IT.
- Add ElasticsearchAiSearchFilterExpressionConverter.
- Add dependency to BOM and module to parent pom.
- Fix ElasticsearchVectorStoreIT FilterExpression with Date type requires the use of epoch milliseconds.
- Add license formatting.

赞扬图标 main · v1.0.0 · v1.0.0-M1

Commit 46e4784



JM-Lab authored and **tzolov** committed on Jun 16, 2024 · 2 / 2

Add OpenSearch vector store integration

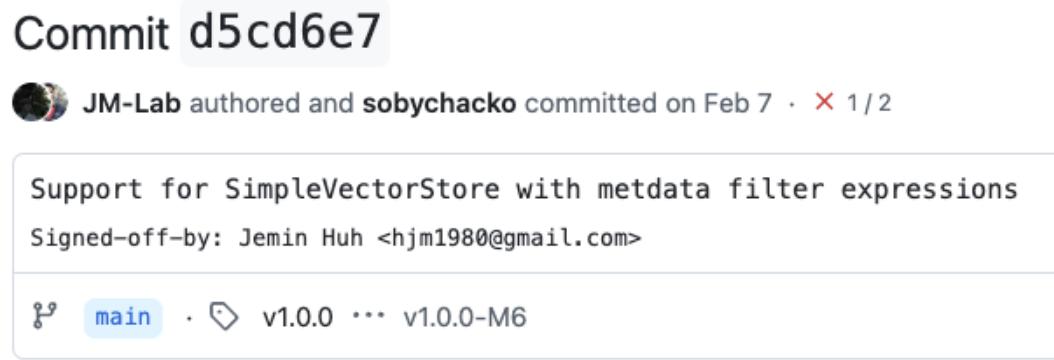
- implement OpenSearchVectorStore
- add opensearch auto-configuration and boot starter
- add documentation for OpenSearch VectorStore
- add bom dependencies
- align with new Spring AI API

赞扬图标 main · v1.0.0 · v1.0.0-M2

Spring AI 1.0 GA 히스토리

2025년 2월 Spring AI 1.0.0-M6 MCP 도입

- 1.0.0-M6 ~ M8, RC1
 - Observability 강화
 - Tool 구조 개편
 - MCP 구조 개선
 - 전체적인 모듈 구조 대개편
 - 모듈 이름 표준화
 - Deprecated API 제거: 최종 GA 전 정리



Spring AI 1.0 GA 히스토리

2025년 5월 20일

Spring AI 1.0 GA 발표

- 1.0 GA
 - 로고 변경
 - Spring AI 기반 Agent 개발 문서 및 예제
 - Building Effective Agents
 - 공식 문서 정리
 - Awesome Spring AI
 - Spring AI Powered Local CLI Chat Bot with RAG
 - Spring AI Playground

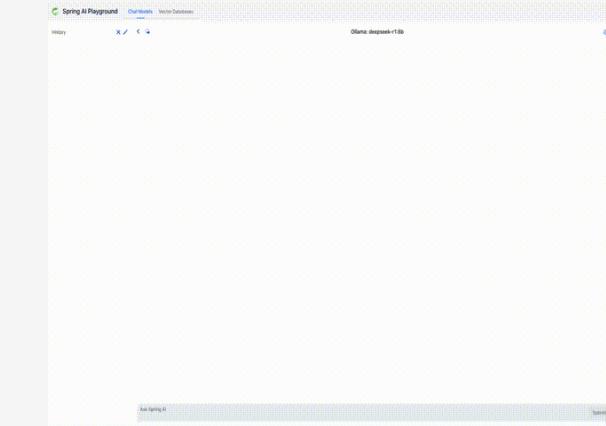


```
spring-ai-local-cli-chatbot
USER: Tell me about Hurricane Milton.

[ Search Results ]
=====
▶ 1 Document, Score: 0.74
-----
Tropical Depression Fourteen Discussion Number 1 (https://www.nhc.noaa.gov/archive/2024/al14/al142024.discus.001.shtml?) (Report). Miami, Florida: National Hurricane Center. Archived (https://web.archive.org/web/20241005151826/https://www.nhc.noaa.gov/archive/2024/al14/al142024.discus.001.shtml) from the original on October 5, 2024. Retrieved October 5, 2024.

=====
▶ 2 Document, Score: 0.74
-----
Brown, Daniel; Blake, Eric (October 7, 2024). Hurricane Milton Update Statement (https://www.n...)
...
=====

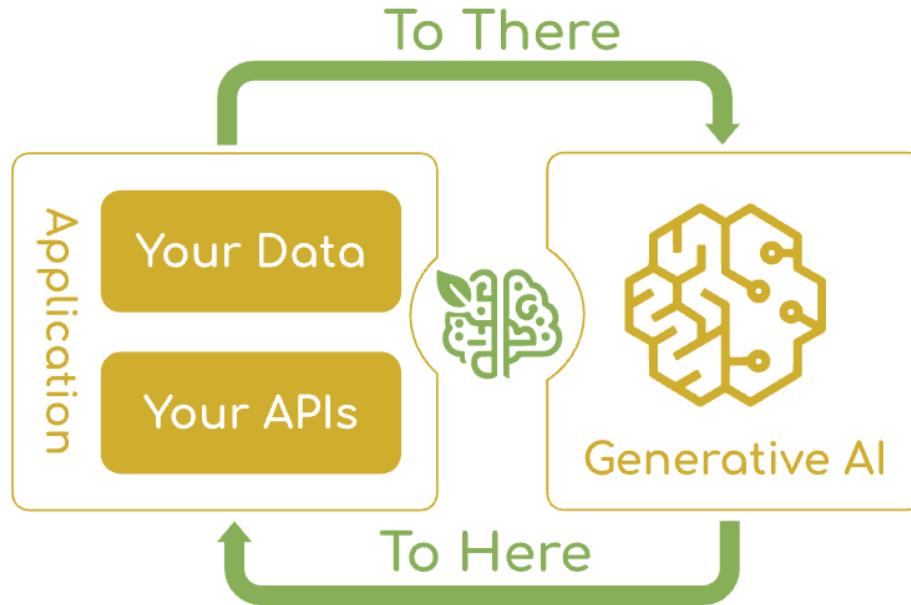
ASSISTANT: Hurricane Milton was a powerful Category 5 storm that formed in the Atlantic in 2024.
```



Spring AI vs Langchain

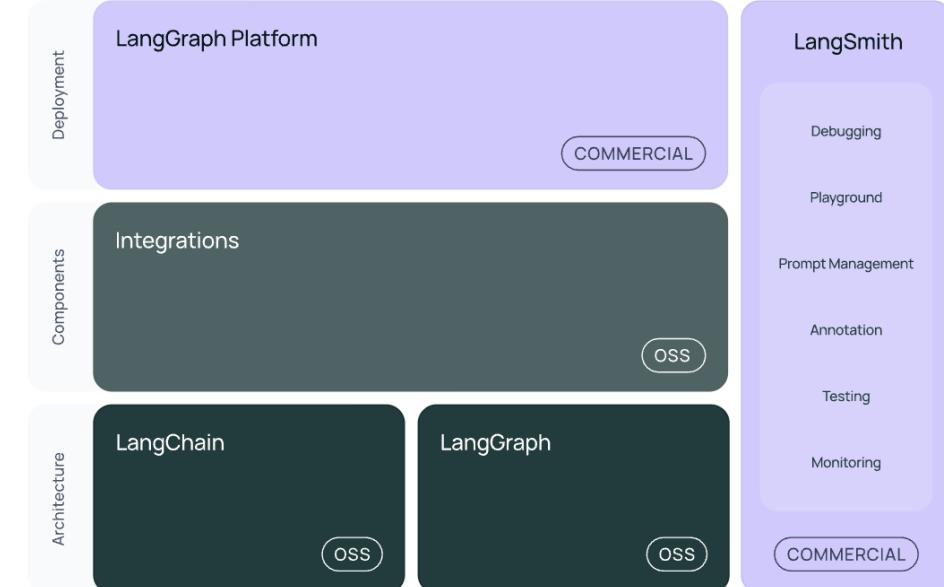
(Overview)

Spring AI vs Langchain (Overview)



Spring AI Overview

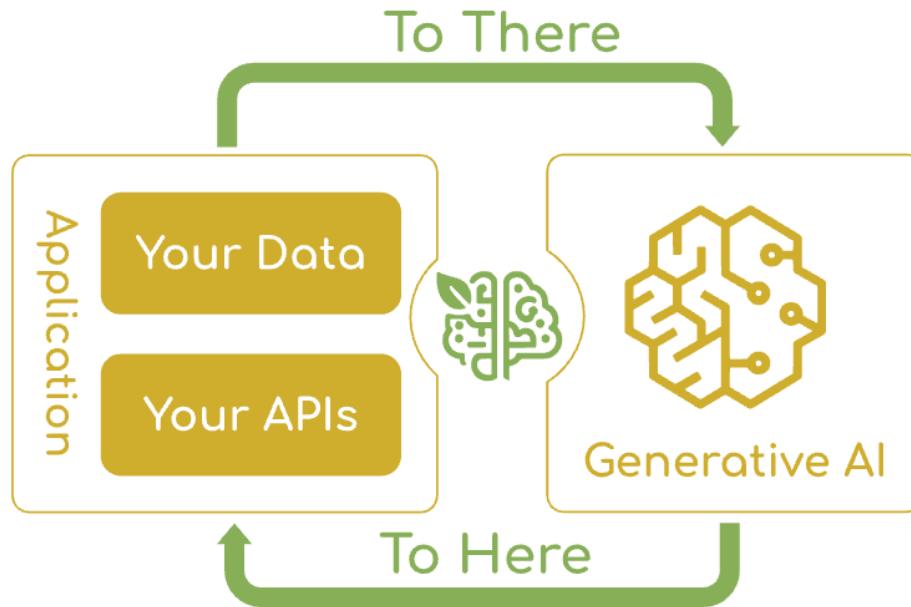
<https://docs.spring.io/spring-ai/reference/index.html>



LangChain Framework Overview

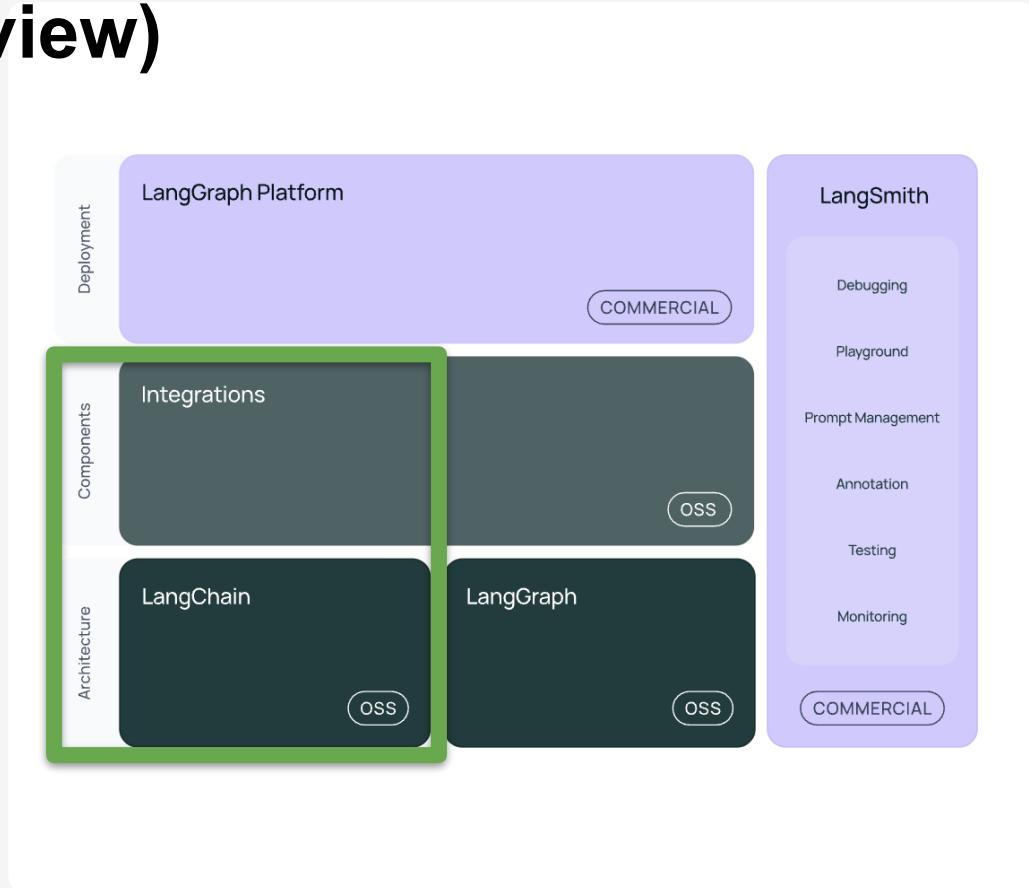
<https://python.langchain.com/docs/introduction/>

Spring AI vs Langchain (Overview)



Spring AI Overview

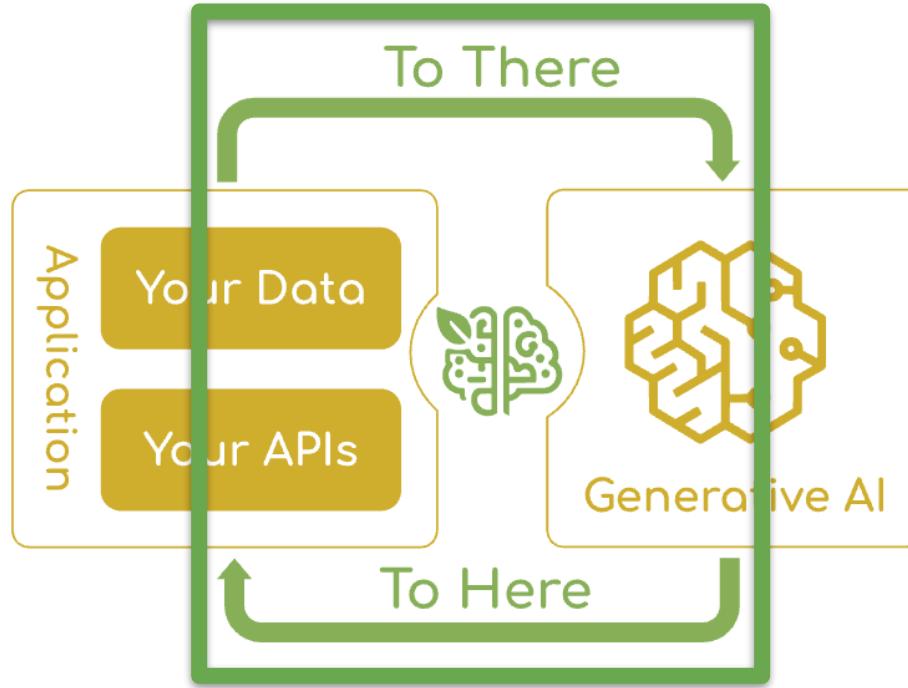
<https://docs.spring.io/spring-ai/reference/index.html>



LangChain Framework Overview

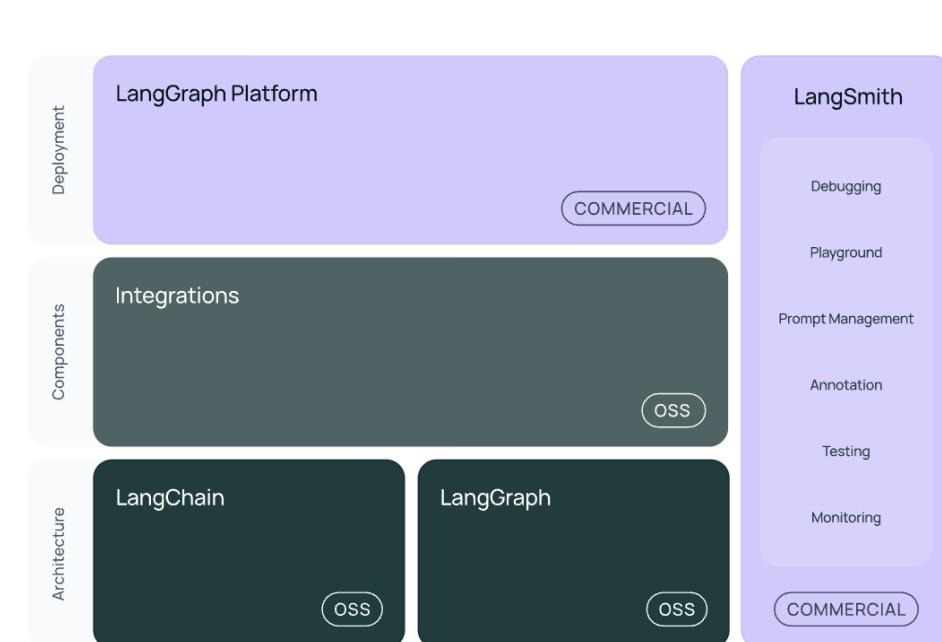
<https://python.langchain.com/docs/introduction/>

Spring AI vs Langchain (Overview)



Spring AI Overview

<https://docs.spring.io/spring-ai/reference/index.html>

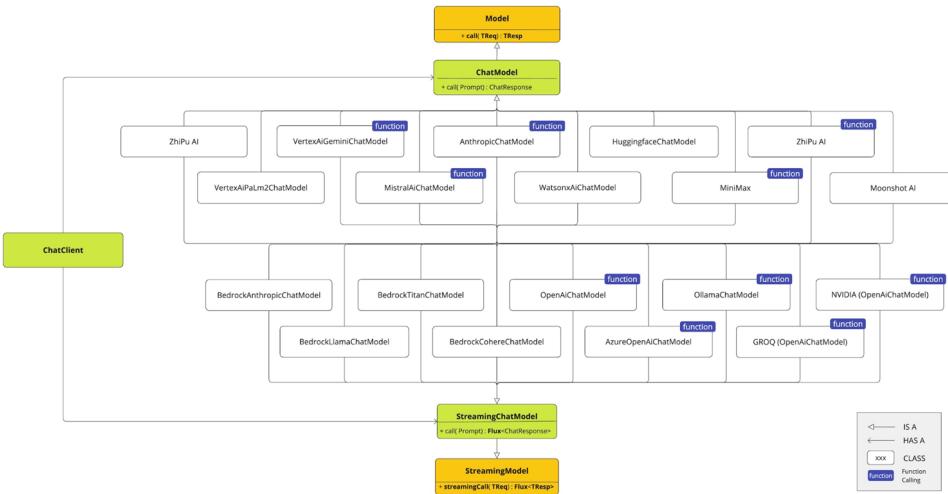


LangChain Framework Overview

<https://python.langchain.com/docs/introduction/>

Spring AI vs Langchain (Chat Model)

Spring AI vs Langchain (Chat Model)



Class hierarchy:

```
BaseLanguageModel --> BaseChatModel --> <name> # Examples: ChatOpenAI, ChatGooglePalm
```

Main helpers:

```
AIMessage, BaseMessage, HumanMessage
```

Functions

```
chat_models.base.init_chat_model()
```

Initialize a ChatModel from the model name and provider.

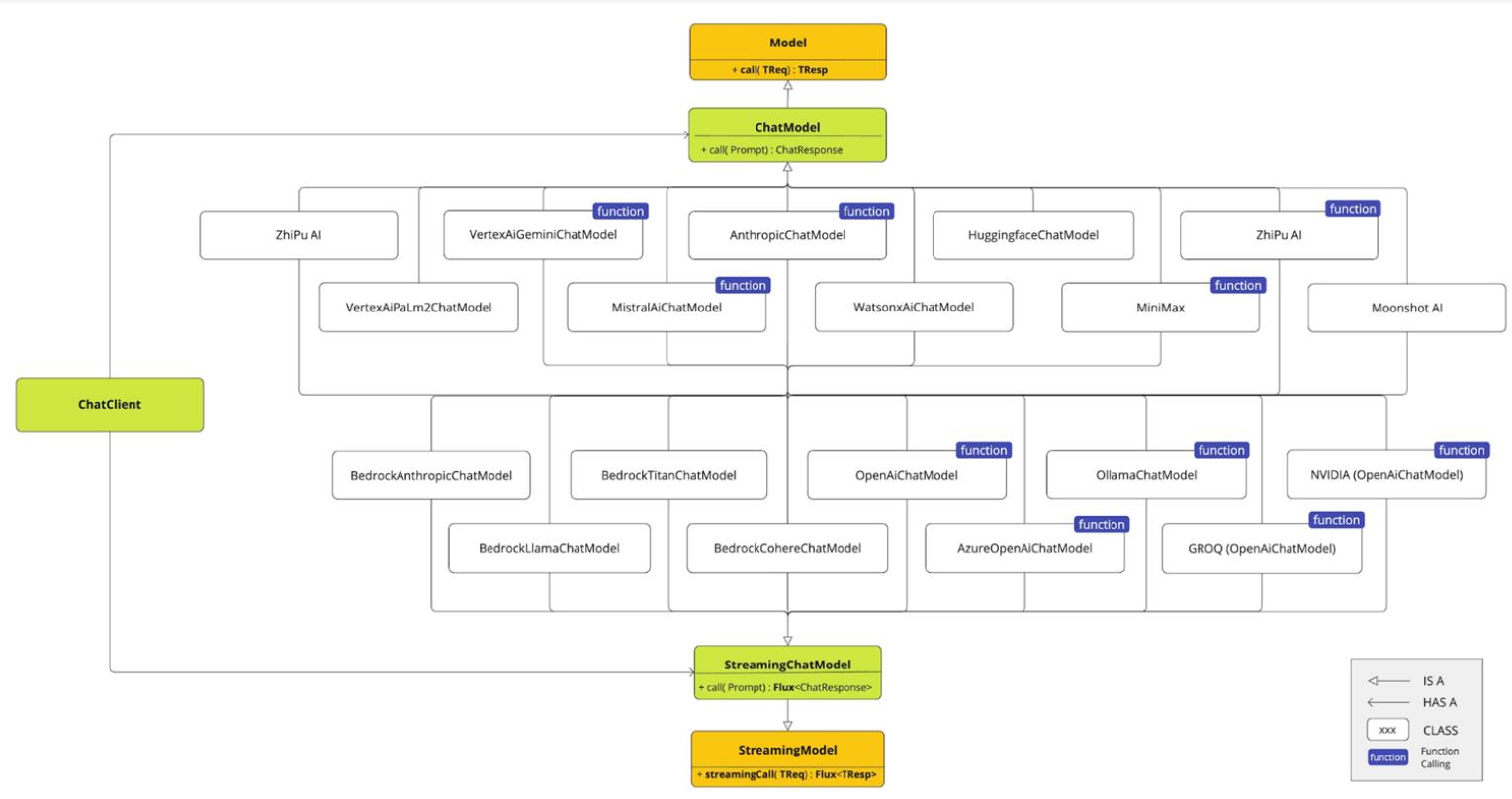
Spring AI Chat Model API Design

<https://docs.spring.io/spring-ai/reference/index.html>

LangChain chat models

https://python.langchain.com/api_reference/langchain/chat_models.html#module-langchain.chat_models

Spring AI vs Langchain (Chat Model)

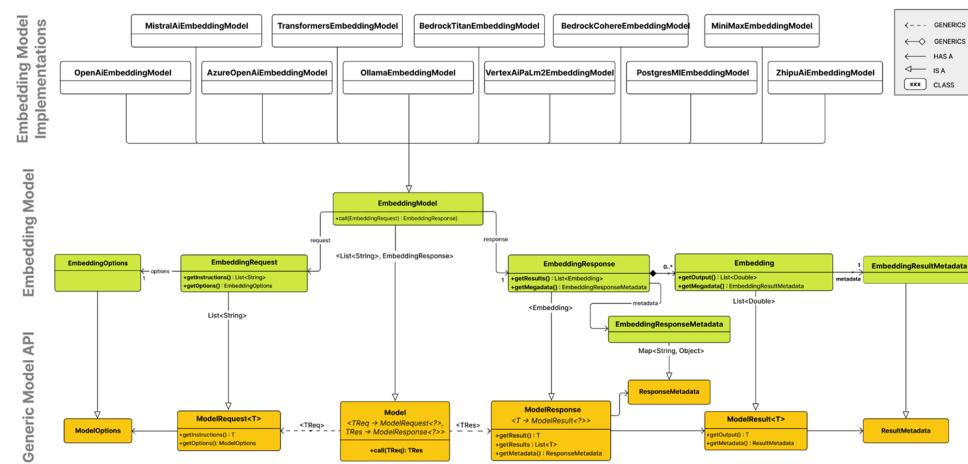


Model, Streaming Model Interface에서 시작

Model 을 지원하는 Provider 가 ChatModel, StreamingChatModel 을 구현

Spring AI vs Langchain (Embedding Model)

Spring AI vs Langchain (Embedding Model)



Class hierarchy:

```
Embeddings --> <name>Embeddings # Examples: OpenAIEmbeddings, HuggingFaceEmbeddings
```

Classes

<code>embeddings.cache.CacheBackedEmbeddings (...)</code>	Interface for caching results from embedding models.
-----------------------------------------------------------	------------------------------------------------------

Functions

<code>embeddings.base.init_embeddings (model, *[...])</code>	Initialize an embeddings model from a model name and optional provider.
--------------------------------------------------------------	-------------------------------------------------------------------------

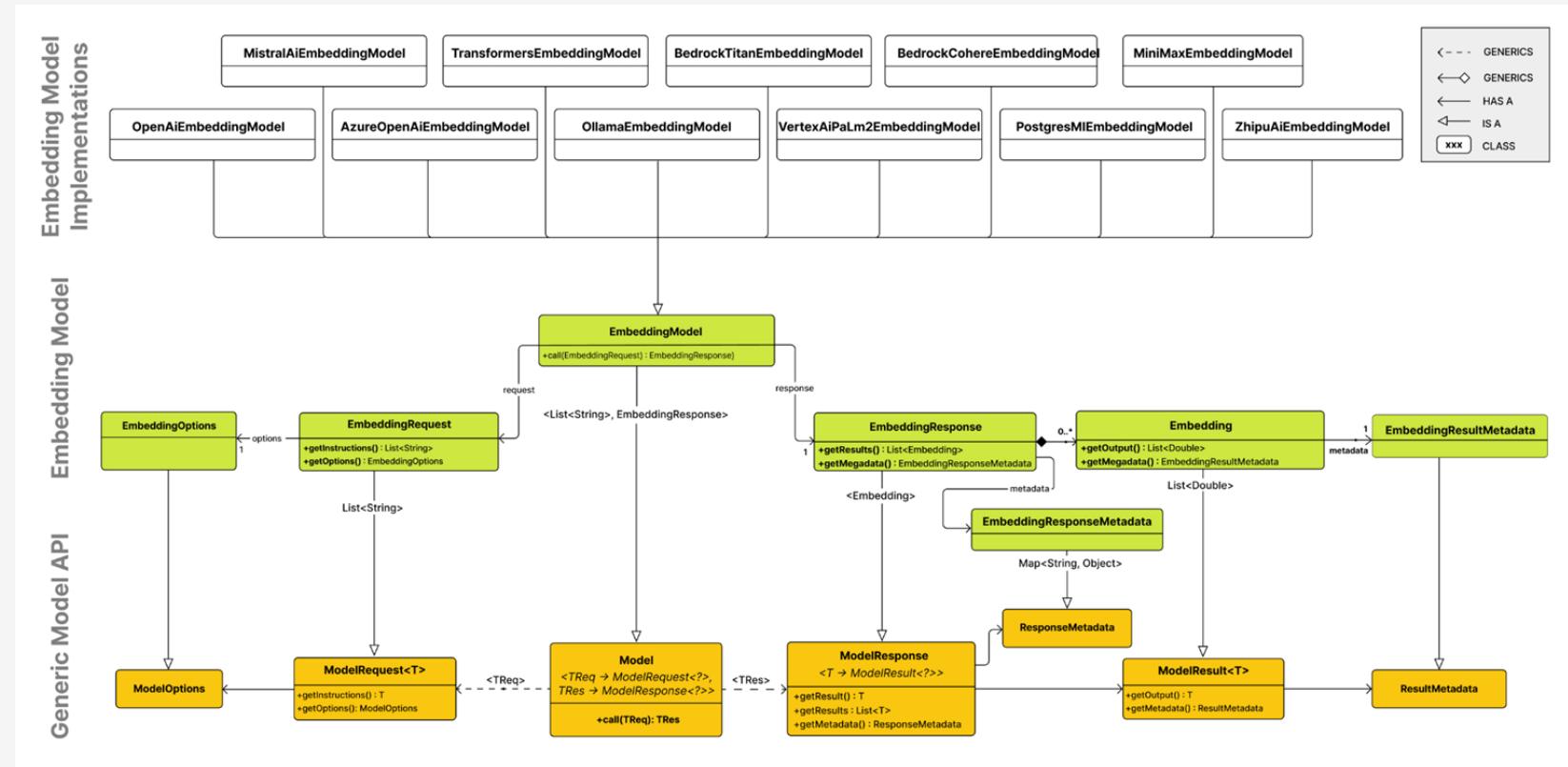
Spring AI Embeddings Model API Design

<https://docs.spring.io/spring-ai/reference/index.html>

LangChain embeddings

https://python.langchain.com/api_reference/langchain/embeddings.html#

Spring AI vs Langchain (Embedding Model)



Model Interface에서 시작

Model 을 지원하는 Provider 가 EmbeddingModel 을 구현

Spring AI vs Langchain

(Vector Databases)

Spring AI vs Langchain (Vector Databases)

```
public interface VectorStore extends DocumentWriter {  
  
    default String getName() {  
        return this.getClass().getSimpleName();  
    }  
  
    void add(List<Document> documents);  
  
    void delete(List<String> idList);  
  
    void delete(Filter.Expression filterExpression);  
  
    default void delete(String filterExpression) { ... };  
  
    List<Document> similaritySearch(String query);  
  
    List<Document> similaritySearch(SearchRequest request);  
  
    default <T> Optional<T> getNativeClient() {  
        return Optional.empty();  
    }  
}
```

vectorstores

Classes

[vectorstores.base.VectorStore\(\)](#)
[vectorstores.base.VectorStoreRetriever](#)
[vectorstores.in_memory.InMemoryVectorStore\(...\)](#)

Interface for vector store.

Base Retriever class for VectorStore.

In-memory vector store implementation.

Functions

[vectorstores.utils.maximal_marginal_relevance\(...\)](#)

Calculate maximal marginal relevance.

Spring AI Vector Databases

<https://docs.spring.io/spring-ai/reference/api/vectordbs.html>

LangChain vectorstores

[https://python.langchain.com/api_reference/core/index.html
#langchain-core-vectorstores](https://python.langchain.com/api_reference/core/index.html#langchain-core-vectorstores)

Spring AI vs Langchain (Vector Databases)

- Azure Vector Search - The [Azure](#) vector store.
- Apache Cassandra - The [Apache Cassandra](#) vector store.
- Chroma Vector Store - The [Chroma](#) vector store.
- Elasticsearch Vector Store - The [Elasticsearch](#) vector store.
- GemFire Vector Store - The [GemFire](#) vector store.
- MariaDB Vector Store - The [MariaDB](#) vector store.
- Milvus Vector Store - The [Milvus](#) vector store.
- MongoDB Atlas Vector Store - The [MongoDB Atlas](#) vector store.
- Neo4j Vector Store - The [Neo4j](#) vector store.
- OpenSearch Vector Store - The [OpenSearch](#) vector store.
- Oracle Vector Store - The [Oracle Database](#) vector store.
- PgVector Store - The [PostgreSQL/PGVector](#) vector store.
- Pinecone Vector Store - [PineCone](#) vector store.
- Qdrant Vector Store - [Qdrant](#) vector store.
- Redis Vector Store - The [Redis](#) vector store.
- SAP Hana Vector Store - The [SAP HANA](#) vector store.
- Typesense Vector Store - The [Typesense](#) vector store.
- Weaviate Vector Store - The [Weaviate](#) vector store.
- SimpleVectorStore - A simple implementation of persistent vector storage, good for educational purposes.

Spring AI VectorStore Implementations

https://docs.spring.io/spring-ai/reference/api/vectordbs.html#_vectorstore_implementations

Vectorstore	Delete by ID	Filtering	Search by Vector	Search with score	Async	Passes Standard Tests	Multi Tenancy	IDs in add Documents
AstraDBVectorStore	✓	✓	✓	✓	✓	✗	✗	✗
Chroma	✓	✓	✓	✓	✓	✗	✗	✗
Clickhouse	✓	✓	✗	✓	✗	✗	✗	✗
CouchbaseVectorStore	✓	✓	✗	✓	✓	✗	✗	✗
DatabricksVectorSearch	✓	✓	✓	✓	✓	✗	✗	✗
ElasticsearchStore	✓	✓	✓	✓	✓	✗	✗	✗
FAISS	✓	✓	✓	✓	✓	✗	✗	✗
InMemoryVectorStore	✓	✓	✗	✓	✓	✗	✗	✗
Milvus	✓	✓	✗	✓	✓	✗	✗	✗
MongoDBAtlasVectorSearch	✓	✓	✓	✓	✓	✗	✗	✗
openGauss	✓	✓	✓	✓	✗	✓	✗	✓
PGVector	✓	✓	✓	✓	✓	✗	✗	✗
PineconeVectorStore	✓	✓	✓	✗	✓	✗	✗	✗
QdrantVectorStore	✓	✓	✓	✓	✓	✗	✗	✗
Redis	✓	✓	✓	✓	✓	✗	✗	✗
Weaviate	✓	✓	✓	✓	✓	✗	✓	✗
SQLServer	✓	✓	✓	✓	✗	✗	✗	✗

All Vectorstores

Name	Description
ActiveLoop Deep Lake	ActiveLoop Deep Lake is a Multi-Modal Vector Store that stores embed...
Aerospike	Aerospike Vector Search (AVS) is an...
Alibaba Cloud OpenSearch	Alibaba Cloud Opensearch is a one-stop platform to develop intelligenc...

LangChain All vectorstores

<https://python.langchain.com/docs/integrations/vectorstores/#all-vectorstores>

Spring AI Chat Client

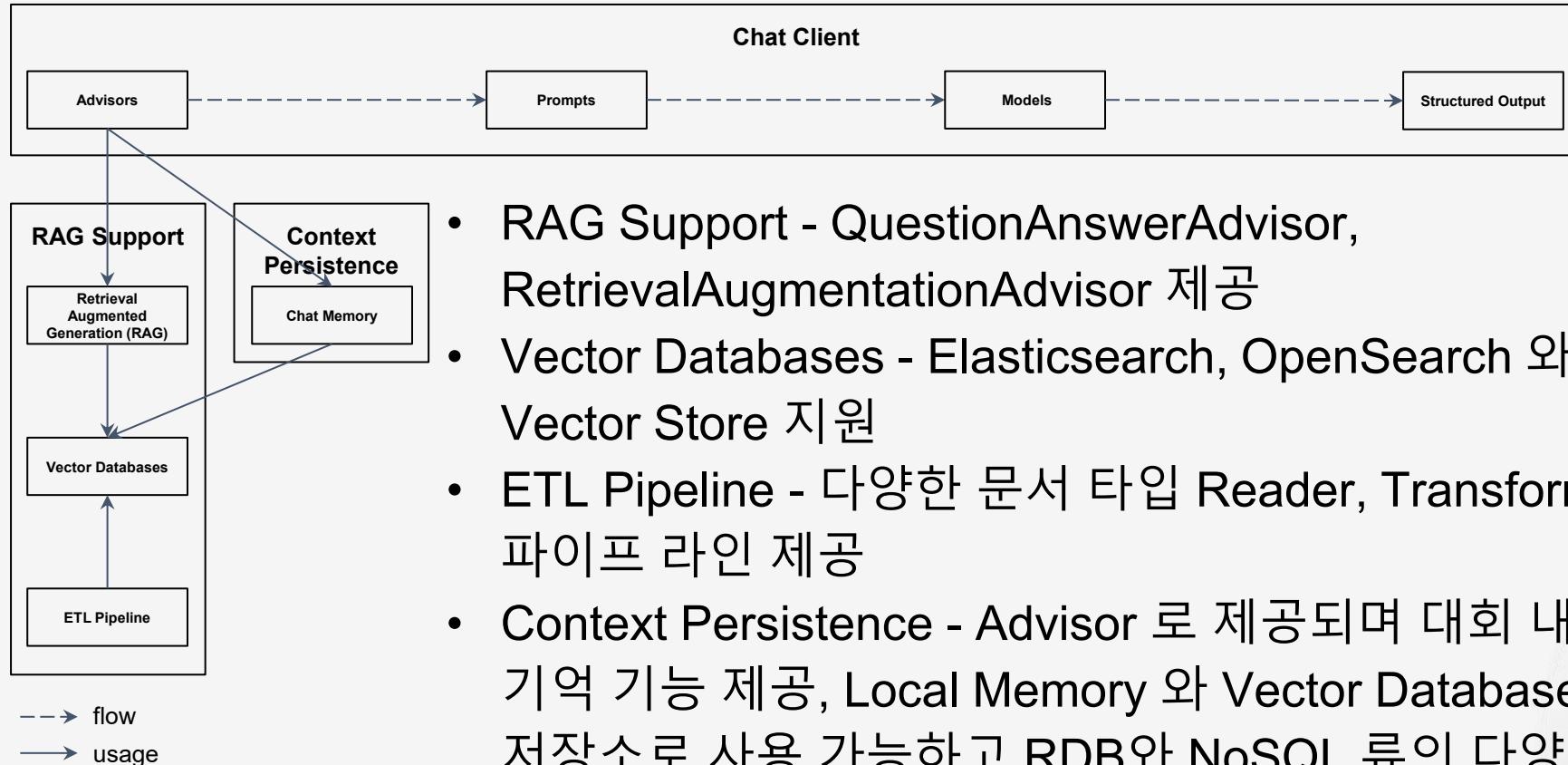
Spring AI Chat Client



- Chat Client - Advisors, Prompts, Models, Structured Output 구성
- Advisors - 최종 Prompts 를 만들기위한 전처리 작업을 수행 하여 Context 를 사용해 전달
- Prompts - Context 의 기본 입력 내용과 Context 내용을 사용하여 Chat Model 에 전달할 Prompts 생성
- Models - 최종 프롬프트를 AI Model 에 전달하여 Responses 를 받음 Responses, Streaming 설정에 따라 응답규격 변경
- Structured Output - 최종 응답을 정해진 객체 규격으로 전달 받을 수 있음

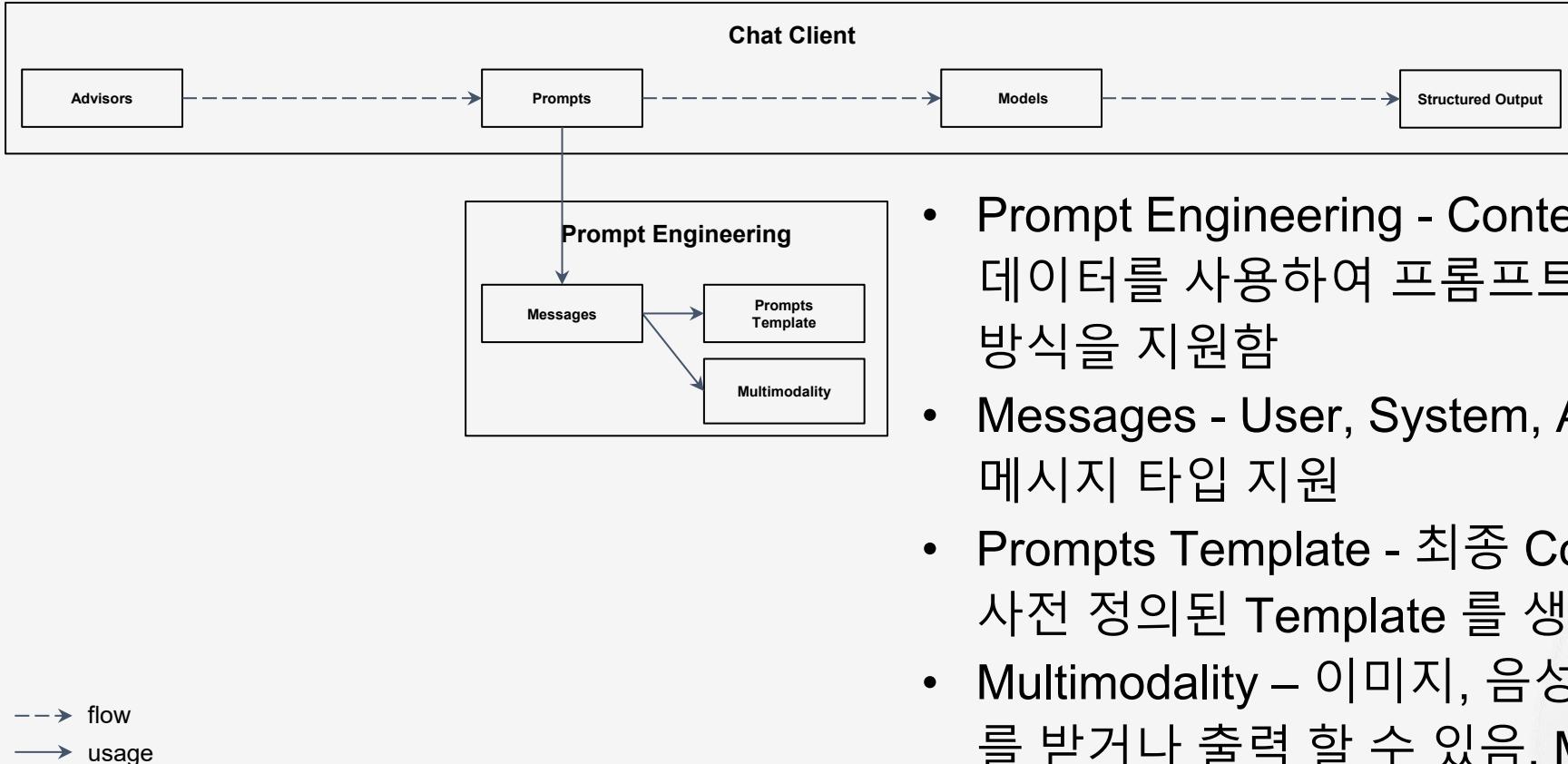
Spring AI Advisor

Spring AI Advisors



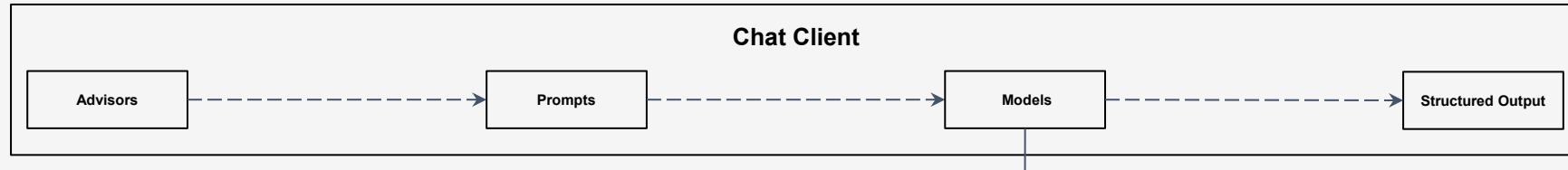
Spring AI Prompts

Spring AI Prompts

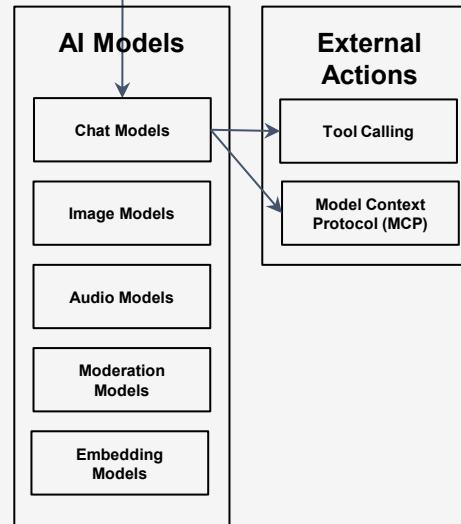


Spring AI Models

Spring AI Models



- AI Models: 다양한 종류의 모델을 추상화
- Chat Models - 대부분의 LLM 모델들 지원
- Image Models - 이미지 생성 모델 지원
- Audio Models - Open AI 의 TTS, STT
- Moderation Models - 부적절한 입출력 검열
- Embedding Models - Vector 추출



- External Actions – 모델이 외부 API 나 기능을 호출하는 방법
- Tool Calling - Chat Model 이 외부 API를 호출하는 방법
- Model Context Protocol (MCP)
 - MCP Client 와 Server 간 통신을 통해 Tool Calling 의 기능을 수행

--> flow
→ usage

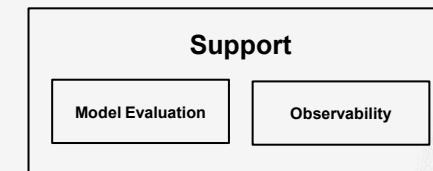
Spring AI Structured Output, Support

Spring AI Structured Output, Support



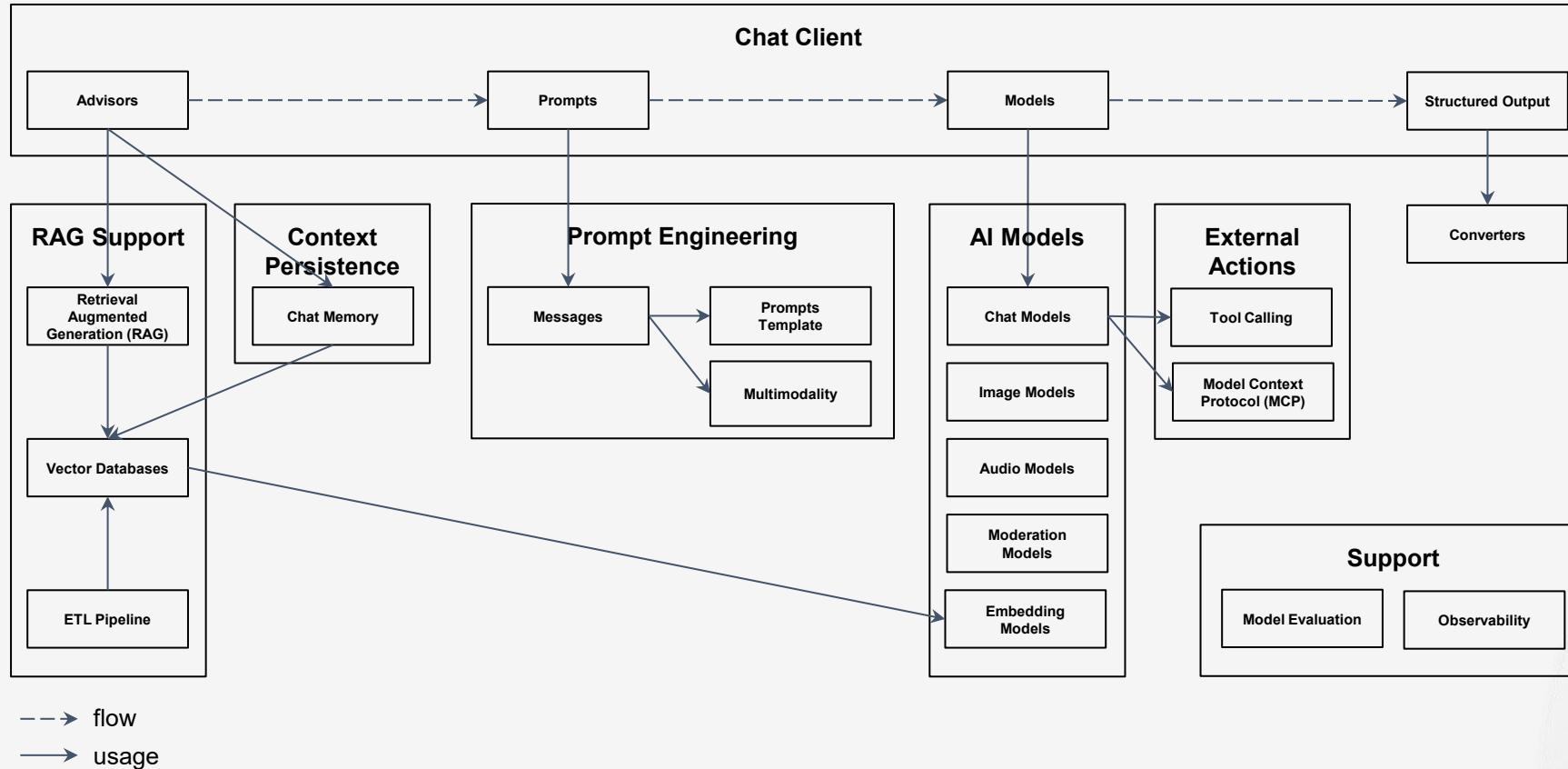
- Converters - Built-in JSON mode 를 지원하는 LLM 모델에서만 사용 가능, 응답을 원하는 형태로 변경
- Support - AI Application 개발에 필요한 기능들 제공
- Model Evaluation - 응답을 평가하는 방법 제공, Unit test 형태로 지원
- Observability - Spring 환경에서 사용되는 Actuator 를 통해 AI Model 들의 다양한 metric 제공

--> flow
—> usage



Spring AI 모듈 구성 및 관계

Spring AI 모듈 구성 및 관계



AI 기본 지식

Prompts

Prompts

- 정의
 - AI 모델이 특정한 출력을 생성하도록 유도하는 입력
- 프롬프트의 구조와 역할 (ChatGPT API)
 - 다중 입력/역할
 - 시스템(System): 모델의 행동 지침 및 맥락 설정
 - 사용자(User): 실제 질문이나 명령 입력
- Prompt Engineering
 - AI와의 대화는 SQL 등 기존 질의와 달리, 사람과 대화하듯 자연어로 소통해야 함
 - 다양한 기법과 연구가 활발히 진행 중
 - 단계별 접근: "차근차근 단계별로 생각해보세요"
 - 역할 부여: "당신은 전문가입니다" 등의 페르소나 설정
 - 예시 제공: Few-shot learning을 통한 패턴 학습
 - 제약 조건: 출력 형식, 길이, 스타일 등의 명확한 가이드라인

Tokens

Tokens

- 정의
 - AI 모델이 입력과 출력을 처리하는 가장 작은 단위

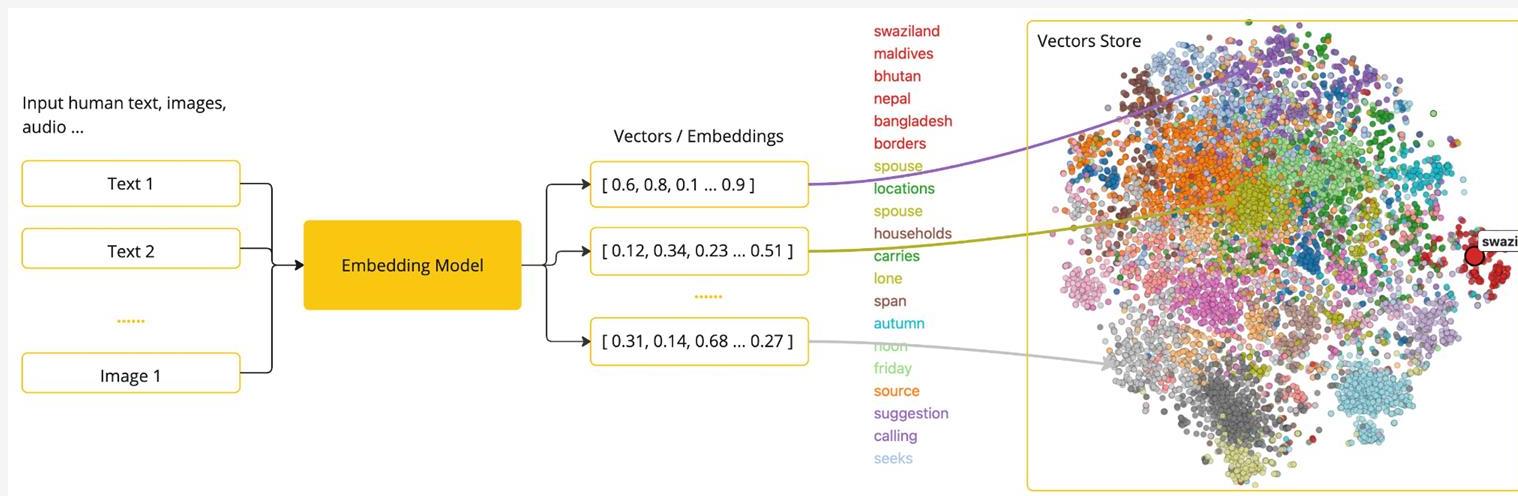
LLMs like ChatGPT generate tokens, not words. Although tokens often end up being complete words, understanding the distinction is essential for understanding how LLM settings affect their outputs, why oddities like universal adversarial triggers occur, how AI-text detectors work, etc.

- Token 변환 과정
 - 입력: 단어 → Token 변환 → AI 모델 처리
 - 출력: Token → 단어 변환 → 사용자에게 제공
- Tokens = 비용
 - OpenAI, Anthropic 등 입력과 출력에 사용된 모든 토큰이 합산되어 과금
- context window
 - AI Model 의 한 번의 호출에서 처리할 수 있는 Tokens 한계
 - 예 - GPT-3(4K), GPT-4(8K/16K/32K), Claude(100K), Meta(100만)

Embeddings

Embeddings

- 정의
 - 텍스트, 이미지, 비디오 등의 입력 간의 관계를 포착하는 숫자 벡터 표현
- 의미론적 공간(semantic space)
 - 두 임베딩 벡터 간의 수치적 거리를 계산, 데이터간 의미적 유사도를 파악
- 활용 분야
 - 텍스트 분류, 의미론적 검색, 제품 추천



Fine Tuning

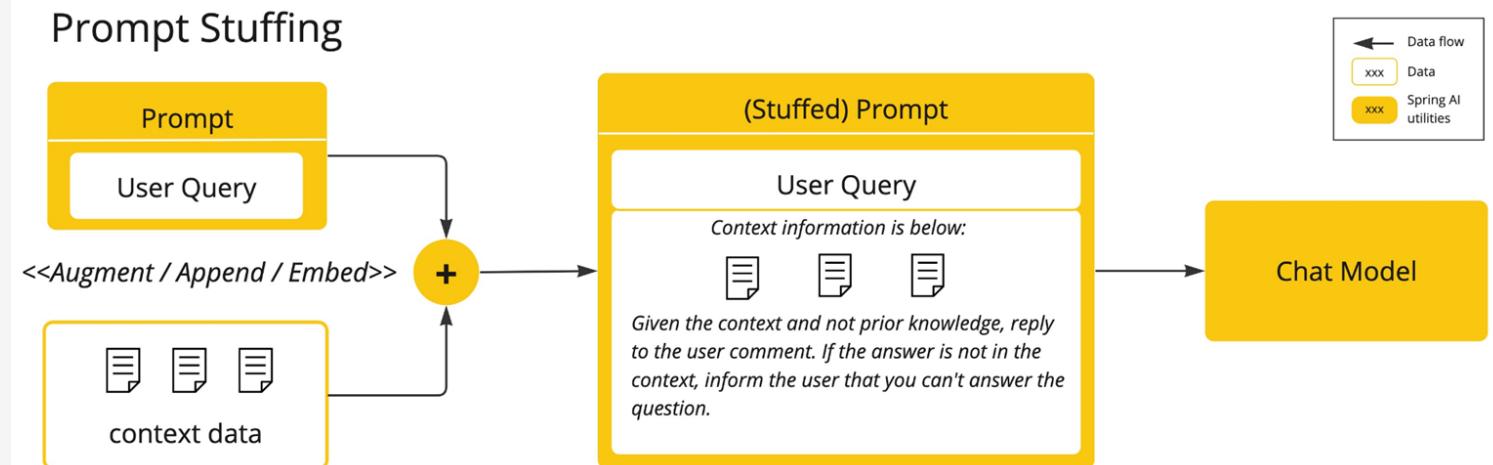
Fine Tuning

- 정의
 - 모델의 내부 가중치를 조정하여 특정 데이터에 맞게 조정하는 전통적인 기법
- AI 모델에 데이터 통합의 필요성
 - GPT 3.5/4.0의 학습 데이터는 2021년 9월까지만 포함 (약 650GB).
- 장점
 - 모델의 특정 작업에 대한 성능을 크게 향상
 - **모델의 행동이나 스타일을 미세하게 조정**
- 단점
 - 고사양 컴퓨팅 자원 필요, 시간·비용 부담 큼
 - 머신러닝 전문 지식 필요
 - 일부 모델은 파인튜닝 미지원

Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation (RAG)

- 정의
 - AI 모델이 학습하지 않은 데이터를 프롬프트에 통합하여 정확한 응답을 생성하는 기술
- 프롬프트 스터핑(Prompt Stuffing)
 - AI 모델의 토큰 제한 내에서 관련 데이터를 효과적으로 프롬프트에 삽입
- 장점
 - 실시간 정보 업데이트
 - 특정 도메인 지식 활용
 - 모델 재훈련 없이 지식 확장
 - 응답 정확성과 신뢰성 향상

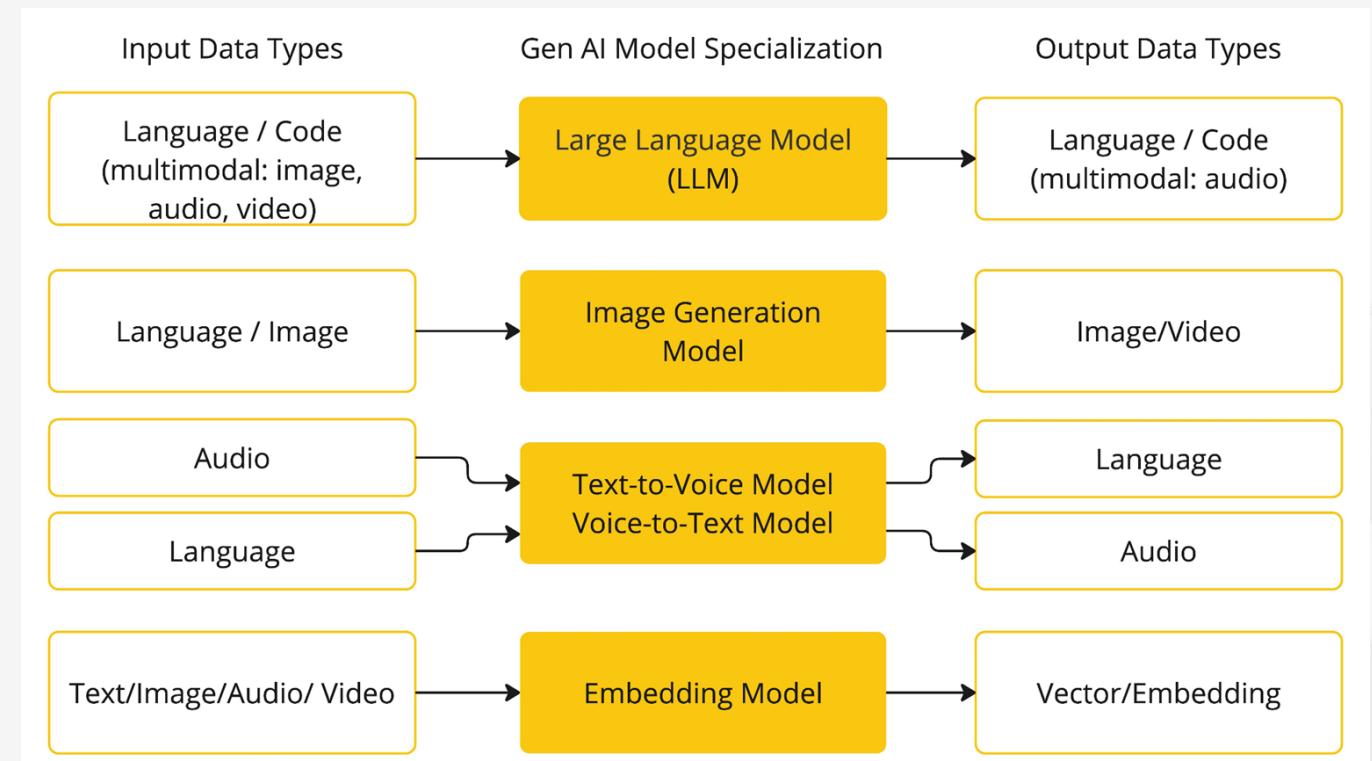


Models

(Language, Image, Audio, Embedding)

Models (Language, Image, Audio, Embedding)

- 정의
 - AI 모델은 정보를 처리하고 생성하도록 설계된 알고리즘, 인간의 인지 기능을 모방
- 사전 훈련 (pre-trained) 모델
 - GPT (Generative Pre-trained Transformer)는 사전 훈련된 대표적인 모델
 - 머신러닝이나 모델 훈련 지식 없이도 AI를 일반 개발 도구로 활용 가능



AI Model providers

AI Model providers

- API 기반 클라우드 서비스
 - 대규모 컴퓨팅 자원 및 인프라 관리 부담 없음
 - 최신 모델에 대한 접근성 용이
 - 사용량 기반 요금(토큰, 호출 횟수 등)
 - 다양한 모델(텍스트, 이미지, 오디오 등) 및 기능 제공

- 로컬 모델 실행 플랫폼 (Ollama)
 - 로컬 환경에서 대규모 언어 모델(LLM) 실행
 - 오프라인 사용

Provider	Multimodality	Tools/Functions	Streaming
Anthropic Claude	text, pdf, image	✓	✓
Azure OpenAI	text, image	✓	✓
DeepSeek (OpenAI-proxy)	text	✗	✓
Google VertexAI Gemini	text, pdf, image, audio, video	✓	✓
Groq (OpenAI-proxy)	text, image	✓	✓
HuggingFace	text	✗	✗
Mistral AI	text, image	✓	✓
MiniMax	text	✓	✓
Moonshot AI	text	✗	✓
NVIDIA (OpenAI-proxy)	text, image	✓	✓
OCI GenAI/Cohere	text	✗	✗
Ollama	text, image	✓	✓
OpenAI	In: text, image, audio Out: text, audio	✓	✓
Perplexity (OpenAI-proxy)	text	✗	✓
QianFan	text	✗	✓
ZhiPu AI	text	✓	✓
Amazon Bedrock Converse	text, image, video, docs (pdf, html, md, docx ...)	✓	✓

Spring AI 개발 맛보기

Spring AI 개발 환경 설명

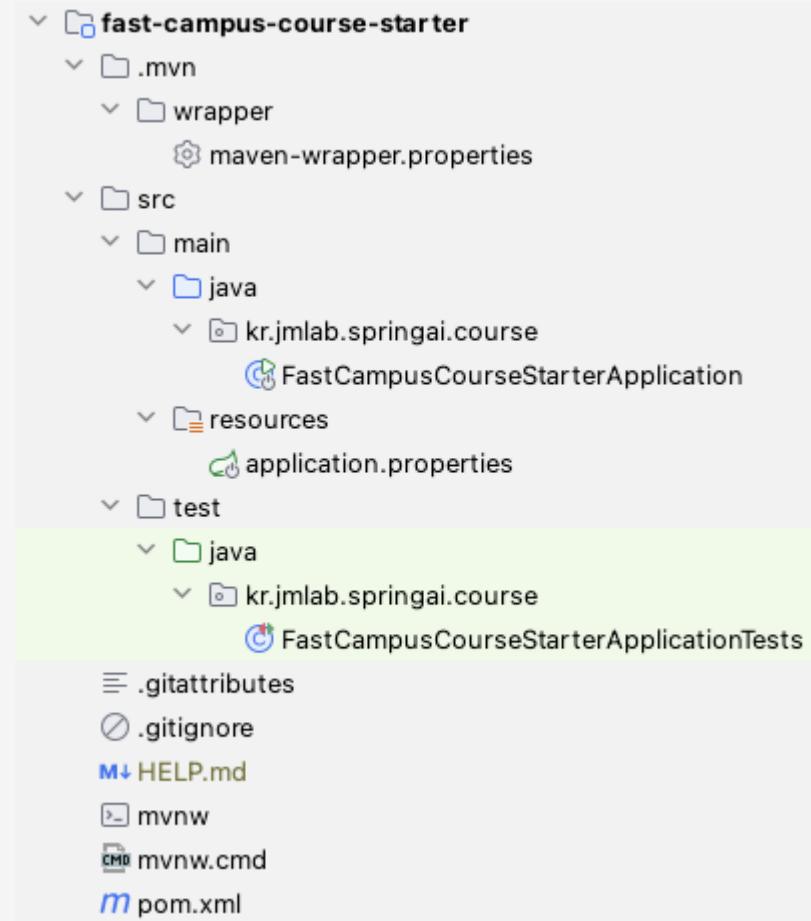
Spring AI 개발 준비

- 전체 코드
 - <https://github.com/JM-Lab/spring-ai-fast-campus-course/tree/main/fast-campus-course-starter>
- Spring Initializr
 - start.spring.io

The screenshot shows the Spring Initializr web interface. On the left, there are sections for Project (Gradle - Groovy, Gradle - Kotlin, Maven), Language (Java, Kotlin, Groovy), and Spring Boot (4.0.0 (SNAPSHOT), 3.5.1 (SNAPSHOT), 3.5.0, 3.4.7 (SNAPSHOT), 3.4.6, 3.3.13 (SNAPSHOT), 3.3.12). On the right, there's a Dependencies section with an 'ADD DEPENDENCIES...' button. Below it, there are two AI-related dependency cards: 'OpenAI' (Spring AI support for ChatGPT and DALL-E) and 'Ollama' (Spring AI support for Ollama LLMs). The 'Ollama' card is highlighted with a green border. At the bottom, there are Java version options (24, 21, 17).

Spring AI 개발 준비

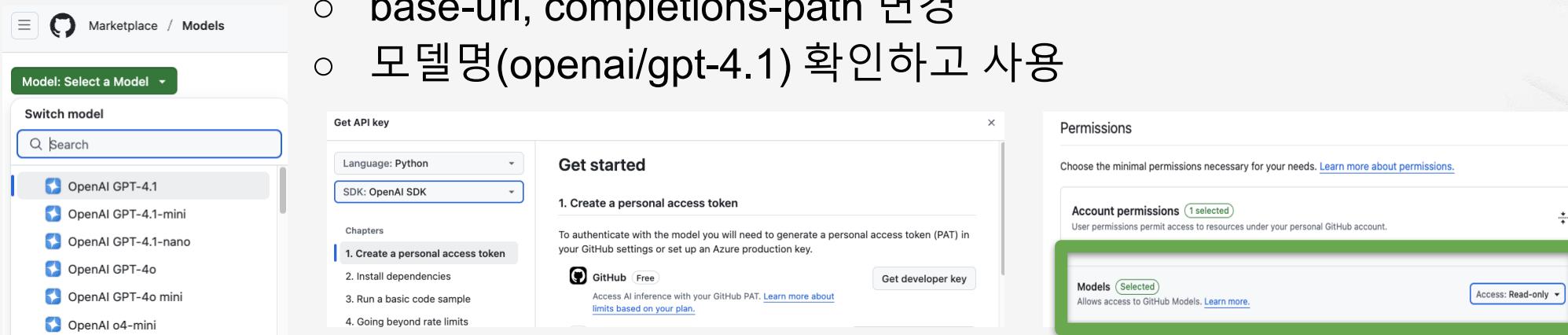
- 주요 디렉토리 구성
 - Main class
 - FastCampusCourseStarterApplication
 - application.properties
 - pom.xml
 - Spring AI 버전 BOM 일괄 관리
 - BOM (Bill of Materials)
 - 사용하려는 Chat 모델 별 의존



Spring AI 개발 환경 구성

Spring AI 개발 환경 구성

- OpenAI Key 준비
 - OpenAI 정식
 - <https://platform.openai.com/>
 - 무료 OpenAI 모델 사용
 - <https://github.com/marketplace/models>
 - OpenAI 모델은 OpenAI SDK 를 지원
 - OPENAI_API_KEY 환경 변수에 github key 사용
 - base-url, completions-path 변경
 - 모델명(openai/gpt-4.1) 확인하고 사용



Chat Client API 환경 구축

Chat Client API 환경 구축

- pom.xml
 - spring-ai-starter-model-openai 만 사용
 - spring-ai-starter-model-ollama 의존성 주석 처리
 - Swagger UI 사용 테스트
 - springdoc-openapi-starter-webmvc-ui 의존성 추가
- application.properties 구성
 - Github 제공 OpenAI 사용 구성

```
# OpenAI 호환 API 를 제공하는 Provider 의 경우 아래 내용을 적절히 수정
spring.ai.openai.api-key=${OPENAI_API_KEY}
spring.ai.openai.chat.options.model=openai/gpt-4.1-nano
# 설정하지 않으면 기본 OpenAI api 호출 주소를 사용
spring.ai.openai.chat.base-url=https://models.github.ai/inference
# OpenAI 기본값은 /v1/chat/completions, github 모델 사용시 아래와 같이 수정
spring.ai.openai.chat.completions-path=/chat/completions
```

Chat Client API 환경 구축

- 구현 및 테스트
 - <http://127.0.0.1:8080/swagger-ui/>

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.8.9</version>
</dependency>

@RestController
class SimpleChatController {

    private final ChatClient chatClient;

    public SimpleChatController(ChatClient.Builder chatClientBuilder) {
        this.chatClient = chatClientBuilder.build();
    }

    @GetMapping("/ai")
    String generation(String userPrompt) {
        return this.chatClient.prompt()
            .user(userPrompt)
            .call()
            .content();
    }
}
```

The screenshot shows the Swagger UI interface for the API. At the top, it says "OpenAPI definition v0 OAS 3.1" and "v3/api-docs". Below that, there's a "Servers" dropdown set to "http://127.0.0.1:8080 - Generated server url". The main area is titled "simple-chat-controller" and shows a "GET /ai" endpoint. Under "Parameters", there is one parameter named "userPrompt" which is required and of type "string". Under "Responses", there is a single response entry for code 200 with the description "OK" and a media type of "string". A note says "Controls Accept header." and "Example Value | Schema".

Local 개발 환경 Ollama 구성

Local 개발 환경 Ollama 구성

- Ollama Install (OS 환경에 맞게 설치)
 - <https://ollama.com/download>
- pom.xml
 - spring-ai-starter-model-ollama 사용
- application.properties 구성
 - Ollama 제공 모델 사용 구성
 - Local LLM 테스트를 위해 국내 오픈 모델 사용
 - 약 1.5GB 메모리 필요

```
# 여러 Chat 모델 사용시 auto-configurations 에서 사용할 모델 설정 필요 예: openai, ollama
spring.ai.model.chat=ollama
# Structured Output, Tool Calling 사용시 mistral (7B q4 기본 모델로 약 7GB 메모리 필요) 사용
spring.ai.ollama.chat.options.model=hf.co/rippertnt/HyperCLOVAX-SEED-Text-Instruct-1.5B-Q4_K_M-GGUF
# when_missing, always, never 설정
spring.ai.ollama.init.pull-model-strategy=when_missing
```

Local 개발 환경 Ollama 구성

- Ollama로 다양한 Local 모델 사용하기
 - <https://ollama.com/search>
 - [GGUF Hugging Face Models](#)
 - Hugging Face 및 llama.cpp 생태계에서 널리 사용되는 모델
 - 모델명 - hf.co/<username>/<model-repository>

The screenshot shows the Ollama search interface with a search bar at the top. Below it, there are three model cards:

- deepseek-r1**: A family of open reasoning models with performance approaching that of leading models, such as Q3 and Gemini 2.5 Pro. It includes tabs for Embedding, Vision, Tools, Thinking, and Popular. It has 1.5b, 7b, 8b, 14b, 32b, 70b, and 671b variants.
- gemma3**: The current, most capable model that runs on a single GPU. It includes tabs for Vision, 1b, 4b, 12b, and 27b. It has 6.2M Pulls, 21 Tags, and was updated 1 month ago.
- qwen3**: Qwen3 is the latest generation of large language models in Qwen series, offering a comprehensive suite of dense and mixture-of-

The screenshot shows the Hugging Face search interface with a search bar at the top. Below it, there are several sections:

- Main**: Tasks, Libraries, Languages, Licenses, Other.
- Tasks**: Text Generation, Any-to-Any, Image-Text-to-Text, Image-to-Text, Image-to-Image, Text-to-Image, Text-to-Video, Text-to-Speech (+42).
- Parameters**: A slider for model size from <1B to >500B, currently set to 128.
- Libraries**: PyTorch, TensorFlow, JAX, Transformers, Diffusers, Safetensors, ONNX, GGUF (selected), Transformers.js, MLX, Keras (+39).
- Models**: 37 results, sorted by Most downloads. Some listed models include:
 - mradermacher/HyperCLOVAX-SEED-Text-Instruct-0.5B-hf-i1-GGUF
 - rippertnt/HyperCLOVAX-SEED-Text-Instruct-1.5B-Q4_K_M-GGUF
 - DevQuasar/naver-hyperclovax.HyperCLOVAX-SEED-Text-Instruct...
 - Mungert/HyperCLOVAX-SEED-Text-Instruct-0.5B-GGUF
 - mradermacher/HyperCLOVAX-SEED-Text-Instruct-1.5B-hf-i1-GGUF
 - cherryDavid/HyperCLOVA-X-SEED-Vision-Instruct-3B-Llamafied...

SimpleChat Rest API 개발

SimpleChat Rest API 개발

- Spring AI

- call

- 동기 호출

- 응답

- ChatResponse

```
@GetMapping(value = "/call", produces = MediaType.APPLICATION_JSON_VALUE)  
ChatResponse call(String userPrompt) {  
    return this.chatClient.prompt()  
        .user(userPrompt)  
        .call().chatResponse();  
}
```

- stream

- 비동기 호출

- 응답

- Flux<String>

```
@GetMapping(value = "/stream", produces = MediaType.TEXT_EVENT_STREAM_VALUE)  
Flux<String> stream(String userPrompt) {  
    return this.chatClient.prompt()  
        .user(userPrompt)  
        .stream()  
        .content();  
}
```

- Server-Sent Events (SSE)

- 클라이언트 단방향 실시간 메시지 전달을 위한 HTTP 기반 스트리밍 기술

Spring AI Prompt Engineering

Prompt 구조

Prompt 구조

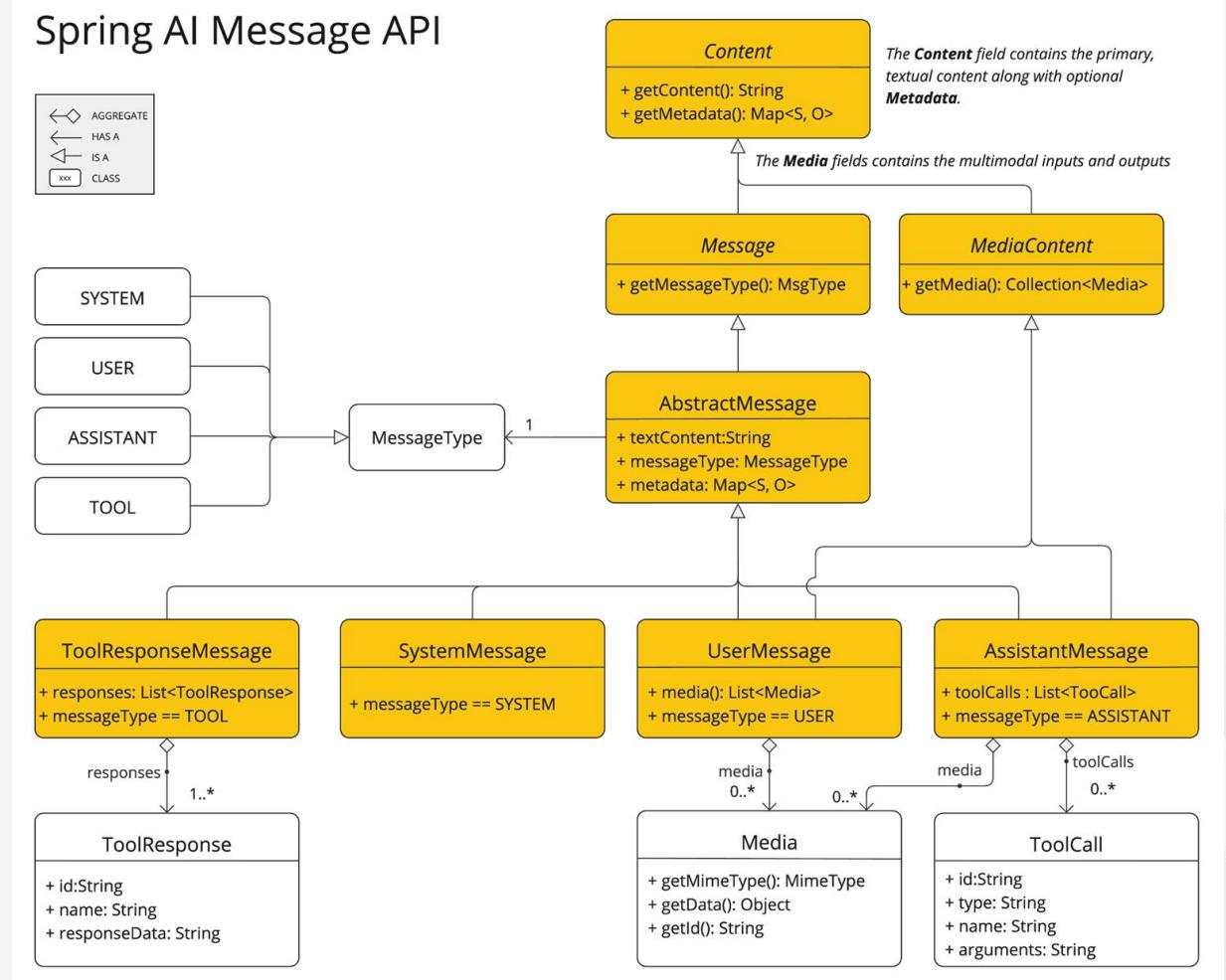
- Prompt 클래스
 - Message 객체와 요청 ChatOptions를 위한 컨테이너 역할

```
public class Prompt implements ModelRequest<List<Message>> {  
  
    private final List<Message> messages;  
  
    private ChatOptions chatOptions;  
}
```

- Prompt 사용
 - ChatModel에 요청시 Prompt 전달
 - Message 내 contents를 코드를 사용해 조작

Prompt 구조

- Message Type = Role
 - USER
 - 사용자 메시지
 - Multimodal
 - ASSISTANT
 - Model 의 응답 메시지
 - Multimodal
 - SYSTEM
 - AI의 전반적인 행동 방식과 응답 스타일을 설정
 - TOOL
 - Tool 실행 결과(추가 정보) 응답 메시지



Prompt Template

Prompt Template

- Spring AI의 구조화된 프롬프트 생성을 위한 클래스
- 주요 특징
 - {} 문법으로 템플릿 변수 식별 (커스터마이징 가능)
- 사용 예 - <> 문법으로 변경

```
PromptTemplate promptTemplate = PromptTemplate.builder()
    .renderer(StTemplateRenderer.builder().startDelimiterToken('<').endDelimiterToken('>').build())
    .template("""
        Tell me the names of 5 movies whose soundtrack was composed by <composer>.
    """)
    .build();

String prompt = promptTemplate.render(Map.of("composer", "John Williams"));
```

Chat Options

Chat Options

- Chat Model의 응답 특성을 제어할 수 있는 공통 옵션 설정 제공
- Options
 - Temperature
 - 응답이 얼마나 창의적이고 무작위적으로 생성될지 결정
 - 값의 의미:
 - 0.0~0.3: 매우 결정적, 일관된 응답 (사실 기반, 분류 등)
 - 0.4~0.7: 균형 잡힌 응답 (일반 대화, 정보 제공 등)
 - 0.8~1.0: 창의적이고 다양한 응답 (스토리텔링, 아이디어 브레인스토밍 등)

```
.options(ChatOptions.builder()
    .temperature(0.1) // Very deterministic output
    .build())
```

Chat Options

- Options
 - Output Length (MaxTokens)
 - 모델이 한 번에 생성할 수 있는 최대 토큰 수를 제한
 - 값의 의미:
 - 5~25: 단어, 짧은 구, 분류 라벨
 - 50~500: 문단, 짧은 설명
 - 1000+: 장문, 스토리, 복잡한 설명

```
.options(ChatOptions.builder()
    .maxTokens(250) // Medium-length response
    .build())
```

Chat Options

- Options
 - Sampling Controls (Top-K and Top-P)
 - Top-K **OpenAI 의 모델들은 지원 안함**
 - 다음 토큰 선택을 가장 가능성 높은 K개의 토큰으로 제한
 - 값이 클수록 다양성 증가, 작을수록 결정적
 - 큰값 : 40-50
 - Top-P (Nucleus Sampling)
 - 누적 확률이 P를 초과하는 토큰을 동적으로 선택
 - 일반적인 값: 0.8-0.95

```
.options(ChatOptions.builder()
    .topK(40)      // Consider only the top 40 tokens
    .topP(0.8)      // Sample from tokens that cover 80% of probability mass
    .build())
```

Chat Options

- Model-Specific Options

- Provider 별 고유 기능과 구성을 사용할 수 있도록 모델별 옵션 클래스도 제공

```
// Using OpenAI-specific options
OpenAiChatOptions openAiOptions = OpenAiChatOptions.builder()
    .model("gpt-4o")
    .temperature(0.2)
    .frequencyPenalty(0.5)      // OpenAI-specific parameter
    .presencePenalty(0.3)        // OpenAI-specific parameter
    .responseFormat(new ResponseFormat("json_object")) // OpenAI-specific JSON mode
    .seed(42)                   // OpenAI-specific deterministic generation
    .build();

String result = chatClient.prompt("...")
    .options(openAiOptions)
    .call()
    .content();

// Using Anthropic-specific options
AnthropicChatOptions anthropicOptions = AnthropicChatOptions.builder()
    .model("claude-3-7-sonnet-latest")
    .temperature(0.2)
    .topK(40)                  // Anthropic-specific parameter
    .thinking(AnthropicApi.ThinkingType.ENABLED, 1000) // Anthropic-specific thinking configuration
    .build();

String result = chatClient.prompt("...")
    .options(anthropicOptions)
    .call()
    .content();
```

Spring AI에서 Prompt Engineering

Spring AI Prompt Engineering 가이드

- Spring AI는 주요 프롬프트 엔지니어링 기법들을 손쉽게 구현할 수 있는 효율적인 Java API를 제공
- 가장 효과적인 접근 방법
 - 시스템 프롬프트에 few-shot 예시를 함께 사용
 - chain-of-thought과 역할 기반 프롬프트(role prompting)를 함께 사용하는 방식
- Production applications 팁
 - 다양한 파라미터(temperature, top-k, top-p)로 프롬프트를 테스트
 - 중요한 의사결정에는 self-consistency 기법을 고려
 - Spring AI의 엔티티 매핑을 활용하여 타입 안정성(type-safe)이 있는 응답을 생성
 - 애플리케이션 도메인에 특화된 지식을 제공하기 위해 문맥 기반 프롬프트(contextual prompting)를 사용

Prompt Engineering Techniques

Prompt Engineering Techniques

- 원하는 결과를 얻기 위해 LLM(거대 언어 모델)에 제공하는 입력(프롬프트)을 효과적으로 설계하고 구성하는 기술
- Zero-Shot Prompting
 - 예시 없이 모델에 작업을 지시
 - 사전 학습된 지식에 의존
 - 감성 분석, 분류 등 간단한 작업
 - Reference: Brown, T. B., et al. (2020). "Language Models are Few-Shot Learners." arXiv:2005.14165.
<https://arxiv.org/abs/2005.14165>

```
// Implementation of Section 2.1: General prompting / zero shot (page 15)
public void pt_zero_shot(ChatClient chatClient) {
    enum Sentiment {
        POSITIVE, NEUTRAL, NEGATIVE
    }

    Sentiment reviewSentiment = chatClient.prompt(""""
        Classify movie reviews as POSITIVE, NEUTRAL or NEGATIVE.
        Review: "Her" is a disturbing study revealing the direction
        humanity is headed if AI is allowed to keep evolving,
        unchecked. I wish there were more movies like this masterpiece.
        Sentiment:
        """
    )
    .options(ChatOptions.builder()
        .model("claude-3-7-sonnet-latest")
        .temperature(0.1)
        .maxTokens(5)
        .build())
    .call()
    .entity(Sentiment.class);

    System.out.println("Output: " + reviewSentiment);
}
```

Prompt Engineering Techniques

- One-Shot & Few-Shot Prompting
 - One-Shot
 - 단 하나의 예시를 제공
 - 패턴이 비교적 단순할 때 유용
 - Few-Shot
 - 여러 개(보통 3~5개)의 예시를 제공
 - 복잡한 패턴을 학습시키거나 다양한 예외 케이스를 처리 시
 - JSON 파싱 등 특정 출력 형식 요청
 - Reference: Brown, T. B., et al. (2020). "Language Models are Few-Shot Learners." arXiv:2005.14165.
<https://arxiv.org/abs/2005.14165>

```
// Implementation of Section 2.2: One-shot & few-shot (page 16)
public void pt_one_shot_few_shots(ChatClient chatClient) {
    String pizzaOrder = chatClient.prompt(""""
        Parse a customer's pizza order into valid JSON

    EXAMPLE 1:
    I want a small pizza with cheese, tomato sauce, and pepperoni.
    JSON Response:
    """
    {
        "size": "small",
        "type": "normal",
        "ingredients": ["cheese", "tomato sauce", "pepperoni"]
    }
"""

    EXAMPLE 2:
    Can I get a large pizza with tomato sauce, basil and mozzarella.
    JSON Response:
    """
    {
        "size": "large",
        "type": "normal",
        "ingredients": ["tomato sauce", "basil", "mozzarella"]
    }
"""

Now, I would like a large pizza, with the first half cheese and mozzarella.
And the other tomato sauce, ham and pineapple.
"""
    .options(ChatOptions.builder()
        .model("claude-3-7-sonnet-latest")
        .temperature(0.1)
        .maxTokens(250)
        .build())
    .call()
    .content();
}
```

Prompt Engineering Techniques

- System Prompting

- AI 모델의 전체적인 맥락, 목적, 행동 지침, 출력 형식, 윤리적 경계 등 "글로벌 미션"을 설정하는 프롬프트
- 특징:
 - 대화 전체에 일관된 규칙 적용
 - 출력 포맷(JSON, 대문자 등)이나 톤, 역할, 제약조건 등 지정
 - 멀티턴 대화에서 일관성 보장
- Reference: OpenAI. (2022). "System Message."

<https://platform.openai.com/docs/guides/chat/introduction>

```

public void pt_system_prompting_1(ChatClient chatClient) {
    String movieReview = chatClient
        .prompt()
        .system("Classify movie reviews as positive, neutral or negative. Only return the label in uppercase.")
        .user("")

        Review: "Her" is a disturbing study revealing the direction
        humanity is headed if AI is allowed to keep evolving,
        unchecked. It's so disturbing I couldn't watch it.

        Sentiment:
        """)
    .options(ChatOptions.builder()
        .model("claude-3-7-sonnet-latest")
        .temperature(1.0)
        .topK(40)
        .topP(0.8)
        .maxTokens(5)
        .build())
        .call()
        .content();
}

record MovieReviews(Movie[] movie_reviews) {
    enum Sentiment {
        POSITIVE, NEUTRAL, NEGATIVE
    }
}

record Movie(Sentiment sentiment, String name) {
}

MovieReviews movieReviews = chatClient
    .prompt()
    .system("")

        Classify movie reviews as positive, neutral or negative. Return
        valid JSON.
        """
    .user("")

        Review: "Her" is a disturbing study revealing the direction
        humanity is headed if AI is allowed to keep evolving,
        unchecked. It's so disturbing I couldn't watch it.

        JSON Response:
        """
    .call()
    .entity(MovieReviews.class);

```

Prompt Engineering Techniques

- Role Prompting

- AI에게 특정 역할, 페르소나, 전문가적 관점, 스타일을 부여하여 응답의 스타일과 깊이를 제어
- 특징:
 - "여행 가이드처럼 답변해", "데이터 과학자처럼 설명해" 등 역할 부여
 - 스타일(유머러스, 진지함 등)도 함께 지정 가능
- Reference: Shanahan, M., et al. (2023). "Role-Play with Large Language Models." arXiv:2305.16367. <https://arxiv.org/abs/2305.16367>

```
public void pt_role_prompting_1(ChatClient chatClient) {
    String travelSuggestions = chatClient
        .prompt()
        .system(""""
            I want you to act as a travel guide. I will write to you
            about my location and you will suggest 3 places to visit near
            me. In some cases, I will also give you the type of places I
            will visit.
            """")
        .user(""""
            My suggestion: "I am in Amsterdam and I want to visit only museums."
            Travel Suggestions:
            """")
        .call()
        .content();
}
```

```
public void pt_role_prompting_2(ChatClient chatClient) {
    String humorousTravelSuggestions = chatClient
        .prompt()
        .system(""""
            I want you to act as a travel guide. I will write to you about
            my location and you will suggest 3 places to visit near me in
            a humorous style.
            """")
        .user(""""
            My suggestion: "I am in Amsterdam and I want to visit only museums."
            Travel Suggestions:
            """")
        .call()
        .content();
}
```

Prompt Engineering Techniques

- Contextual Prompting
 - AI에게 추가 배경정보(도메인, 청중, 제약 등)를 전달해 더 맞춤화된 응답을 유도하는 기법
 - 특징:
 - 주 지시문을 복잡하게 만들지 않고, context 변수로 배경정보 전달
 - 특정 상황/タ겟에 최적화된 답변 생성
 - Reference: Liu, P., et al. (2021). "What Makes Good In-Context Examples for GPT-3?" arXiv:2101.06804.
<https://arxiv.org/abs/2101.06804>

```
// Implementation of Section 2.3.3: Contextual prompting
public void pt_contextual_prompting(ChatClient chatClient) {
    String articleSuggestions = chatClient
        .prompt()
        .user(u -> u.text(""""
            Suggest 3 topics to write an article about with a few lines of
            description of what this article should contain.

            Context: {context}
            """))
        .param("context", "You are writing for a blog about retro 80's arcade video games.")
    .call()
    .content();}
```

Prompt Engineering Techniques

- Chain of Thought (CoT)
 - 문제 해결 과정을 여러 단계로 나누어 논리적으로 추론하도록 유도하는 프롬팅 기법
 - 특징:
 - Zero-shot CoT - “Let's think step by step.”만 추가해도 LLM이 논리적 추론
 - Few-shot CoT - 단계별 추론 예시를 보여주면 모델이 이를 따라 더 정확하게 답을 도출
 - Reference: Wei, J., et al. (2022). "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." arXiv:2201.11903.
<https://arxiv.org/abs/2201.11903>

```
// Implementation of Section 2.5: Chain of Thought (CoT) – Zero-shot approach
public void pt_chain_of_thought_zero_shot(ChatClient chatClient) {
    String output = chatClient
        .prompt(""""
        When I was 3 years old, my partner was 3 times my age. Now,
        I am 20 years old. How old is my partner?

        Let's think step by step.
        """")
        .call()
        .content();
}

// Implementation of Section 2.5: Chain of Thought (CoT) – Few-shot approach
public void pt_chain_of_thought_singleshot_fewshots(ChatClient chatClient) {
    String output = chatClient
        .prompt(""""
        Q: When my brother was 2 years old, I was double his age. Now
        I am 40 years old. How old is my brother? Let's think step
        by step.
        A: When my brother was 2 years, I was 2 * 2 = 4 years old.
        That's an age difference of 2 years and I am older. Now I am 40
        years old, so my brother is 40 - 2 = 38 years old. The answer
        is 38.
        Q: When I was 3 years old, my partner was 3 times my age. Now,
        I am 20 years old. How old is my partner? Let's think step
        by step.
        A:
        """")
        .call()
        .content();
}
```

Prompt Engineering Techniques

- Code Prompting

- 대형 언어 모델(LLM)이 코드를 이해하고 생성하는 능력을 활용하는 특화된 기법
- 특징:
 - 코드 자동화 및 문서화, 프로토타입 개발에 탁월
 - 다양한 언어 간 코드 변환 및 학습 지원
- Reference: Chen, M., et al. (2021). "Evaluating Large Language Models Trained on Code." arXiv:2107.03374.
<https://arxiv.org/abs/2107.03374>

```
// Implementation of Section 2.9.1: Prompts for writing code
public void pt_code_prompting_writing_code(ChatClient chatClient) {
    String bashScript = chatClient
        .prompt(""""
            Write a code snippet in Bash, which asks for a folder name.
            Then it takes the contents of the folder and renames all the
            files inside by prepending the name draft to the file name.
            """
        )
        .options(ChatOptions.builder()
            .temperature(0.1) // Low temperature for deterministic code
            .build())
        .call()
        .content();
}

// Implementation of Section 2.9.2: Prompts for explaining code
public void pt_code_prompting_explaining_code(ChatClient chatClient) {
    String code = """
        #!/bin/bash
        echo "Enter the folder name: "
        read folder_name
        if [ ! -d "$folder_name" ]; then
            echo "Folder does not exist."
            exit 1
        fi
        files=( "$folder_name"/* )
        for file in "${files[@]}"; do
            new_file_name="draft_${basename "$file"}"
            mv "$file" "$new_file_name"
        done
        echo "Files renamed successfully."
    """;

    String explanation = chatClient
        .prompt()
        .user(u -> u.text("""
            Explain to me the below Bash code:
            """
            {code}
            """
        )));
        .param("code", code)
        .call()
        .content();
}

// Implementation of Section 2.9.3: Prompts for translating code
public void pt_code_prompting_translating_code(ChatClient chatClient) {
    String bashCode = """
        #!/bin/bash
        echo "Enter the folder name: "
        read folder_name
        if [ ! -d "$folder_name" ]; then
            echo "Folder does not exist."
            exit 1
        fi
        files=( "$folder_name"/* )
        for file in "${files[@]}"; do
            new_file_name="draft_${basename "$file"}"
            mv "$file" "$new_file_name"
        done
        echo "Files renamed successfully."
    """;

    String pythonCode = chatClient
        .prompt()
        .user(u -> u.text("""
            Translate the below Bash code to a Python snippet:
            """
            {code}
            """
        )));
        .param("code", bashCode)
        .call()
        .content();
}
```

Advanced Prompt Engineering Techniques

Advanced Prompt Engineering Techniques

- Step-Back Prompting
 - 복잡한 문제를 바로 풀지 않고, 먼저 관련 배경지식이나 원리를 추출한 뒤 이를 바탕으로 세부 질문을 해결하는 2단계 접근법입니다.
 - Reference: Zheng, Z., et al. (2023). "Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models." arXiv:2310.06117.
<https://arxiv.org/abs/2310.06117>
- Self-Consistency
 - 동일한 문제에 대해 여러 번(다양한 샘플링으로) 답변을 생성한 뒤, 가장 많이 나온 답변(다수결)을 최종 정답으로 선택하는 방식입니다.
 - Reference: Wang, X., et al. (2022). "Self-Consistency Improves Chain of Thought Reasoning in Language Models." arXiv:2203.11171. <https://arxiv.org/abs/2203.11171>

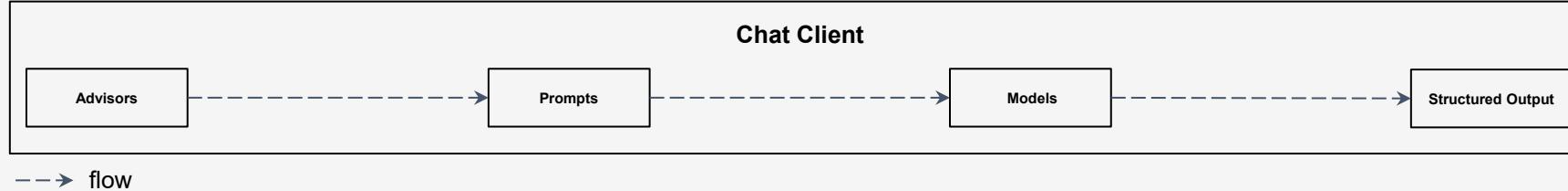
Advanced Prompt Engineering Techniques

- Tree of Thoughts (ToT)
 - 문제 해결을 여러 경로(branch)로 나눠서 동시에 탐색하며, 각 경로를 평가·선택·가지치기(pruning)하는 방식입니다. 복잡한 문제에서 다양한 해결책을 동시에 탐색하고, 최적의 답을 찾는 데 효과적입니다.
 - Reference: Yao, S., et al. (2023). "Tree of Thoughts: Deliberate Problem Solving with Large Language Models." arXiv:2305.10601. <https://arxiv.org/abs/2305.10601>
- Automatic Prompt Engineering
 - AI가 스스로 다양한 프롬프트를 생성·평가·최적화하여, 사람이 직접 프롬프트를 설계하는 과정을 자동화하는 기법입니다.
 - Reference: Zhou, Y., et al. (2022). "Large Language Models Are Human-Level Prompt Engineers." arXiv:2211.01910. <https://arxiv.org/abs/2211.01910>

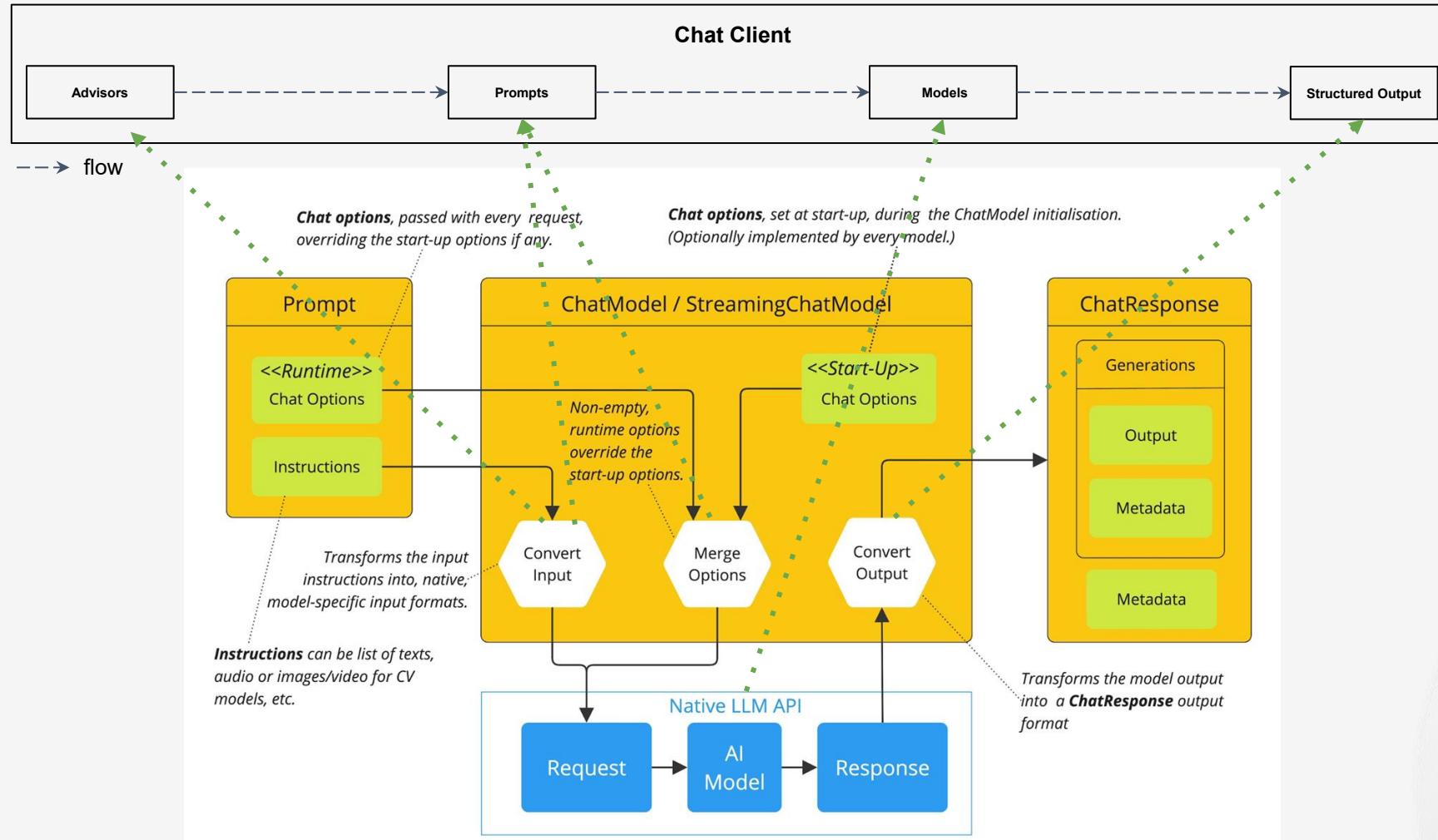
Spring AI Chat Architecture

Chat Flow

Chat Flow



Chat Flow

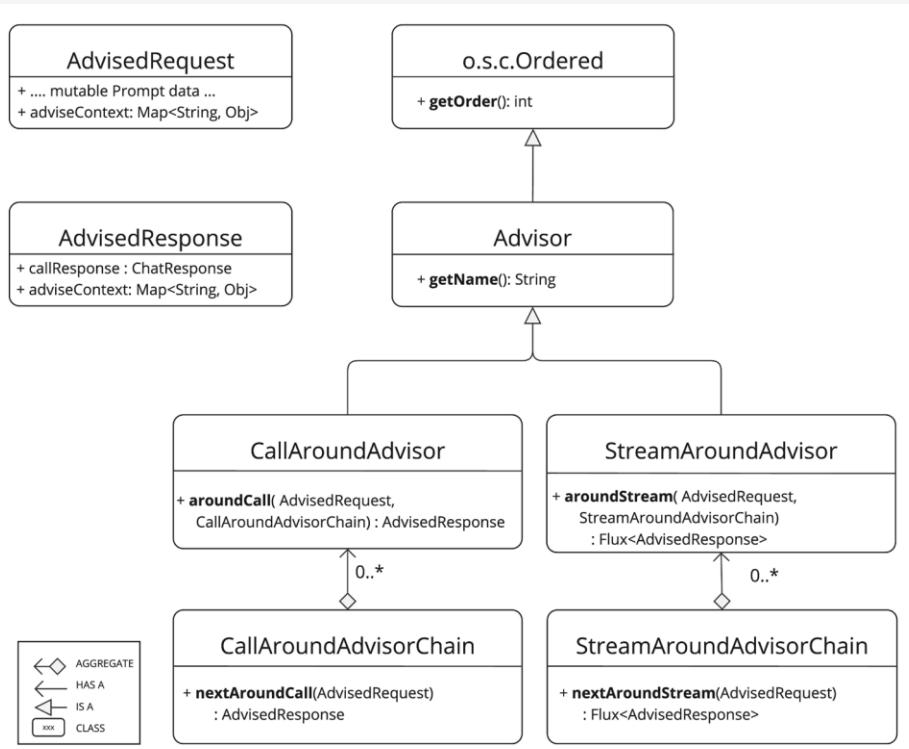


Advisors API

Advisors API

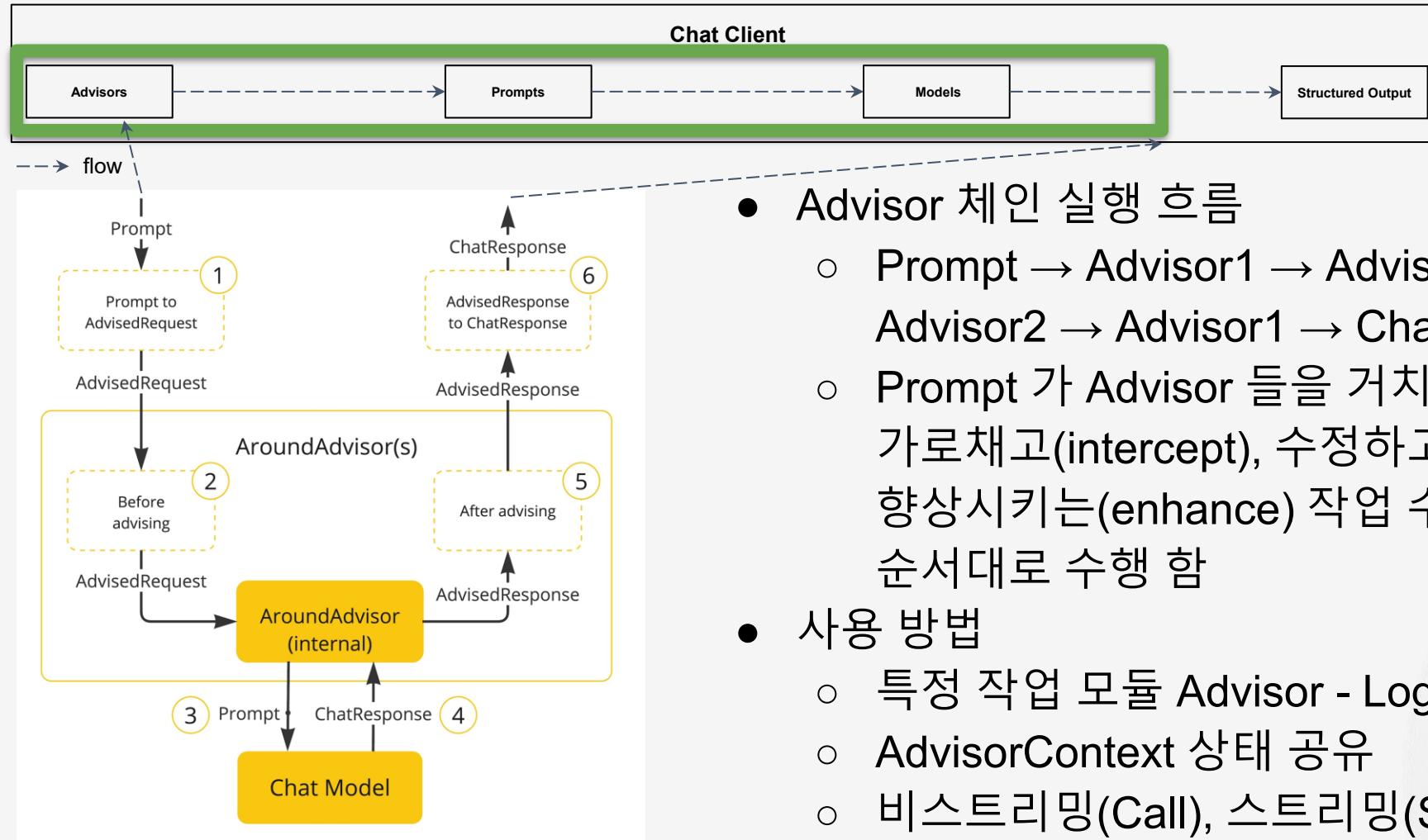


---> flow



- **Advisors**
 - AI 모델 입력 전과 출력 후
가로채고(intercept), 수정하고(modify),
향상시키는(enhance) 유연한 방법
- **Non-Streaming**
 - CallAroundAdvisor,
CallAroundAdvisorChain
- **Streaming**
 - StreamAroundAdvisor,
StreamAroundAdvisorChain
- **AdvisedRequest, AdvisedResponse**

Advisors API



- **Advisor 체인 실행 흐름**

- **Prompt → Advisor1 → Advisor2 → Model → Advisor2 → Advisor1 → ChatResponse**
- **Prompt 가 Advisor 들을 거치며 가로채고(intercept), 수정하고(modify), 향상시키는(enhance) 작업 수행되고 응답도 순서대로 수행 함**

- **사용 방법**

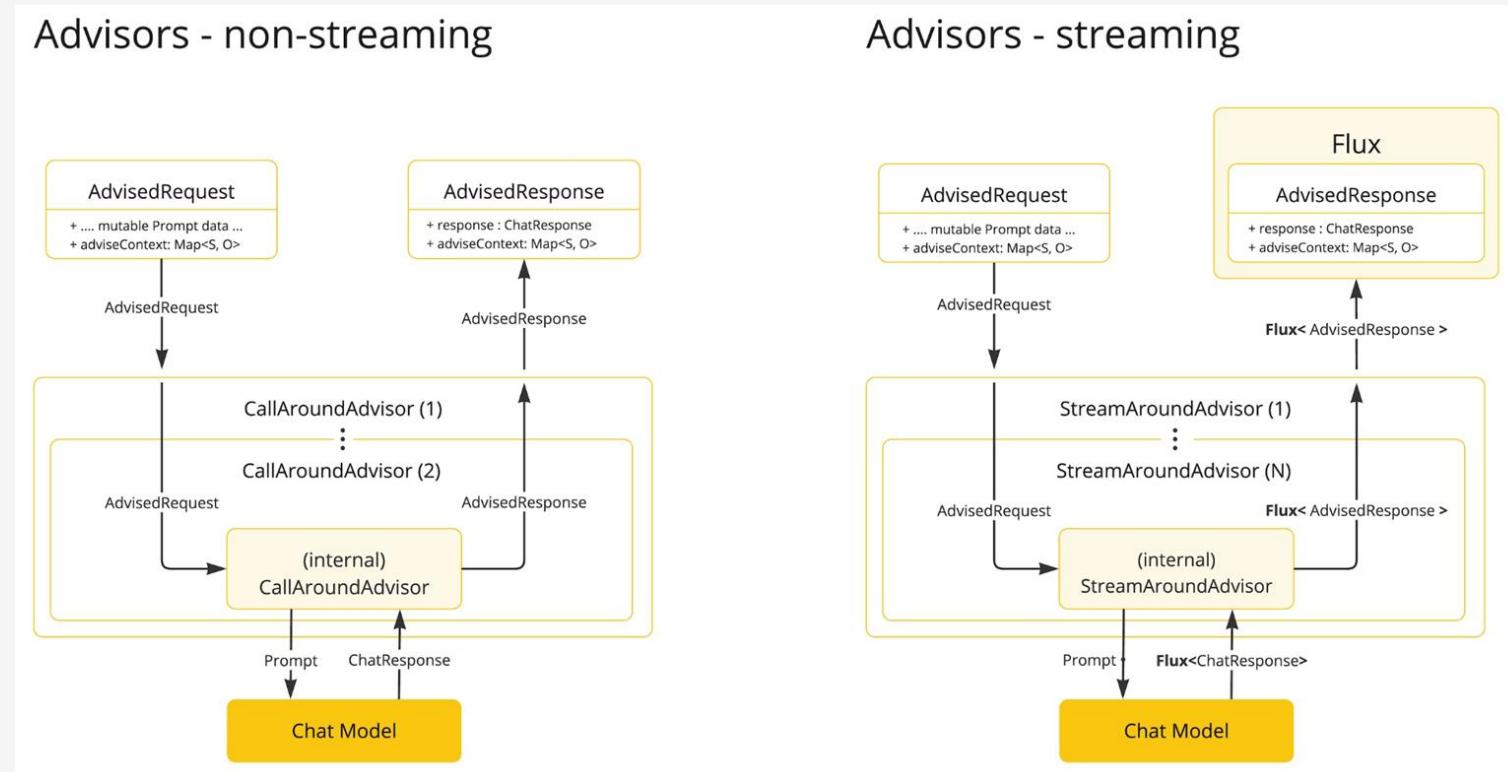
- **특정 작업 모듈 Advisor - Log, RAG, Memory**
- **AdvisorContext 상태 공유**
- **비스트리밍(Call), 스트리밍(Stream) 고려**

Advisors API

- Advisor 실행 순서 (Order)
 - getOrder() 메서드의 반환값에 의해 결정
 - 낮은 값일수록 우선순위가 높음 (먼저 실행).
 - 스택(Stack)과 유사한 동작 방식
 - 요청 처리 시: order 값이 가장 낮은 Advisor 가장 먼저 요청 처리
 - 응답 처리 시: order 값이 가장 낮은 Advisor 가장 마지막에 응답 처리
- Spring Ordered 인터페이스 사용해서 정렬
 - 동일한 order 값을 가진 Advisor들의 실행 순서는 보장되지 않음
 - Ordered.HIGHEST_PRECEDENCE (Integer.MIN_VALUE): 가장 높은 우선순위
 - Ordered.LOWEST_PRECEDENCE (Integer.MAX_VALUE): 가장 낮은 우선순위
 - ChatModel 호출이 실제로는 가장 낮은 우선순위 Advisor로 등록됨
 - 양방향 우선 처리: 요청과 응답 모두에서 가장 먼저 처리되길 원한다면, 별도의 Advisor를 사용하고 order 값과 AdvisorContext를 활용

Advisors API

- Advisors non-streaming, streaming 처리 비교



- BaseAdvisor Interface
 - before, after, getOrder 메소드만 구현하면 non-streaming, streaming 둘다 지원

Advisors API

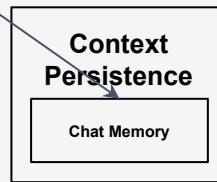
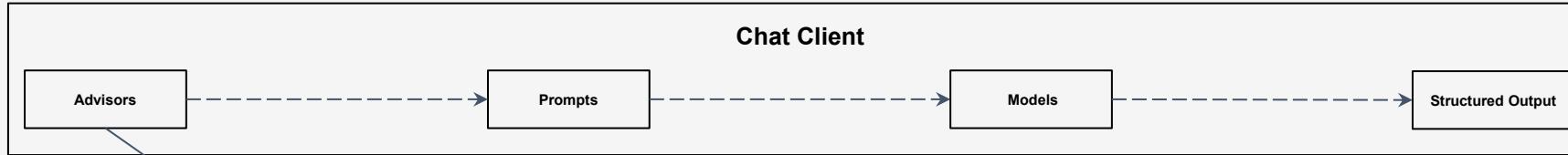
- Spring AI 내장 Advisor
 - SimpleLoggerAdvisor
 - 요청과 응답을 로깅하지만 수정하지 않는 간단한 Advisor, 구현 참고 좋음
 - ReReadingAdvisor
 - 사용자 프롬프트를 {Input_Query}\nRead the question again: {Input_Query} 형태로 변형하여 LLM의 추론 능력을 향상시키는 기법을 적용하는 Advisor
 - ChatMemoryAdvisor
 - Message, Prompt, VectorStore 기반 대화 기록 추가 기능
 - RAG 관련 Advisor
 - QuestionAnswerAdvisor - VectorStore 사용 질의응답 기능
 - RetrievalAugmentationAdvisor - Modular RAG 를 구현한 Advisor
 - SafeGuardAdvisor - 유해하거나 부적절한 콘텐츠 방지
 - 보안 관련 기능 추가시 참고하기 좋은 Advisor

Chat Memory

Chat Memory

- 정의
 - 여러 상호작용에 걸쳐 대화의 맥락을 유지하기 위해 메시지를 저장 검색하는 기능
- 개념 구분
 - Chat Memory: LLM이 대화의 맥락 인식을 위해 유지하는 정보
 - Chat History: 사용자와 모델 간 교환된 전체 대화 기록
- 지원 메모리 유형 (1.0 GA 버전 기준)
 - MessageWindowChatMemory
 - 최대 N개 메시지만 유지, 초과 시 오래된 메시지부터 삭제(기본값 20개)
- 여러 저장소 종류 지원
 - InMemoryChatMemoryRepository: 메모리 기반, 기본값
 - JdbcChatMemoryRepository: 관계형 DB(JDBC) 기반, 영구 저장
 - CassandraChatMemoryRepository: 분산 DB, TTL 지원, 대규모/감사 목적 적합
 - Neo4jChatMemoryRepository: 그래프 DB, 관계형 데이터 활용

Chat Memory



- Advisor를 통해 ChatClient에 적용
 - 기본으로 InMemoryChatMemoryRepository 사용

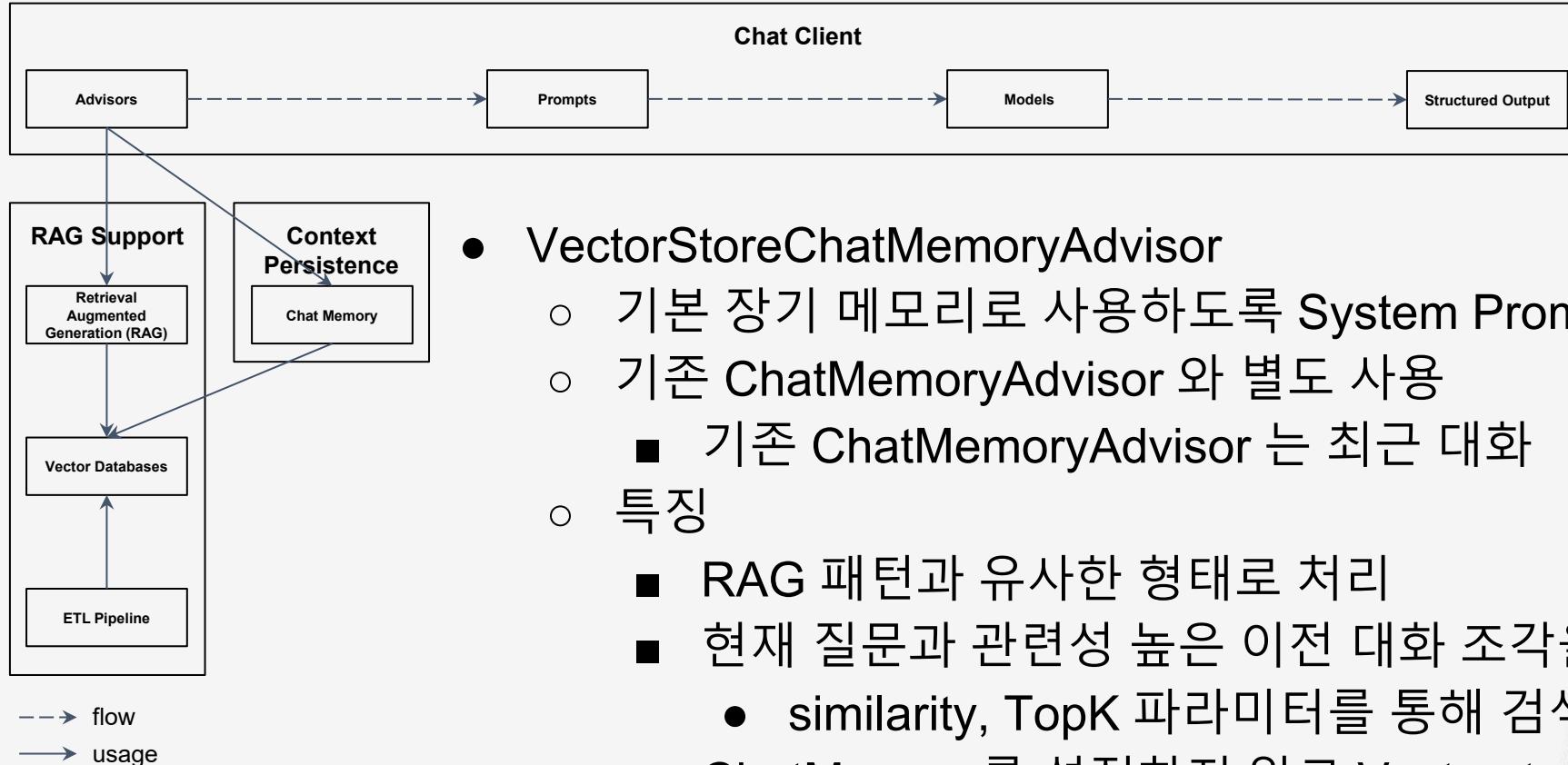
```
ChatMemory chatMemory = MessageWindowChatMemory.builder().build();
```

```
ChatClient chatClient = ChatClient.builder(chatModel)
    .defaultAdvisors(MessageChatMemoryAdvisor.builder(chatMemory).build())
    .build();
```

- Spring AI는 다양한 Advisor를 통해 메모리 동작을 유연하게 구성
 - MessageChatMemoryAdvisor
 - 메시지 컬렉션 형태로 프롬프트에 포함
 - PromptChatMemoryAdvisor
 - 시스템 프롬프트에 텍스트로 추가

--> flow
-> usage

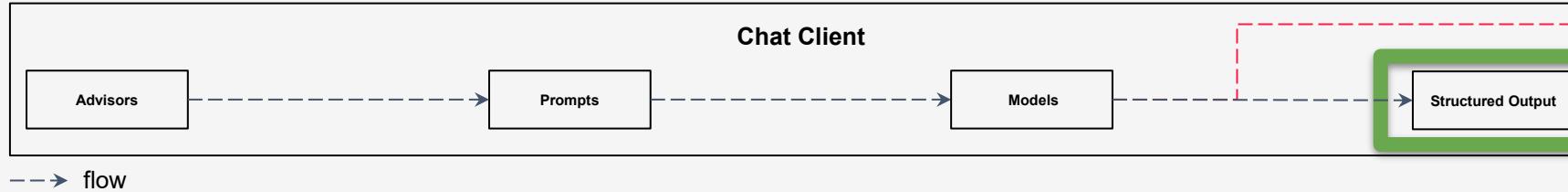
Chat Memory



- **VectorStoreChatMemoryAdvisor**
 - 기본 장기 메모리로 사용하도록 System Prompt 설정
 - 기존 ChatMemoryAdvisor 와 별도 사용
 - 기존 ChatMemoryAdvisor 는 최근 대화
 - 특징
 - RAG 패턴과 유사한 형태로 처리
 - 현재 질문과 관련성 높은 이전 대화 조각을 검색
 - similarity, TopK 파라미터를 통해 검색 설정
 - ChatMemory를 설정하지 않고 Vectorstore 사용
 - Vector Databases에 구현된 Vectorstores 사용

Structured Output

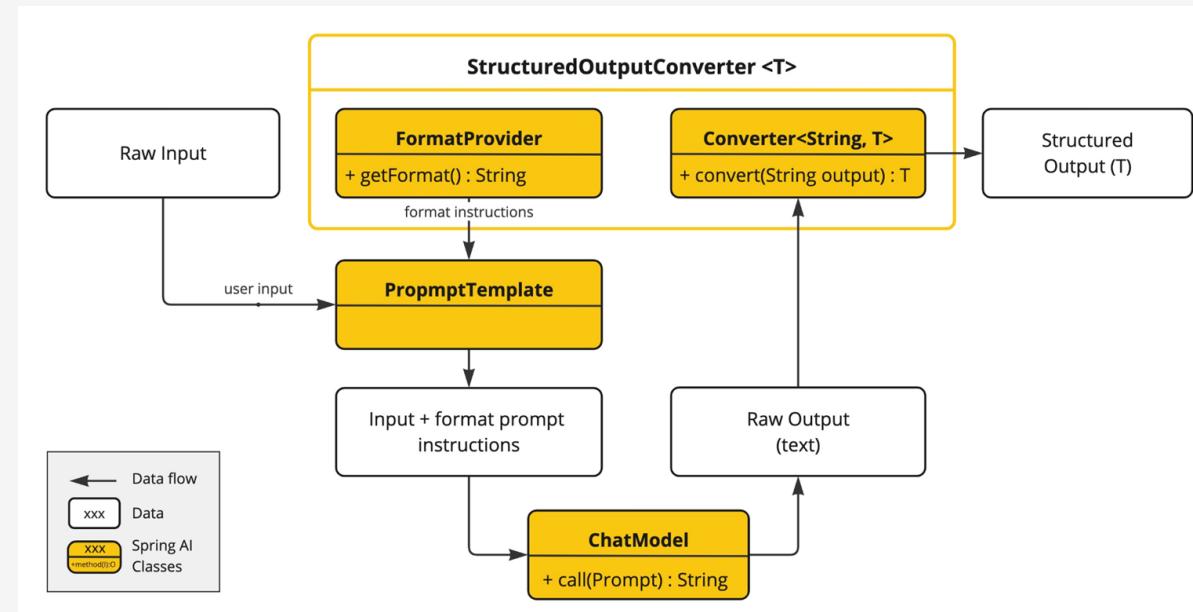
Structured Output



- Structured Output (구조화된 데이터)의 필요성
 - LLM은 자유 형식의 텍스트(String)로 응답하여 파싱이 필요함
 - 타입 안전성(Type Safety) 확보하여 명확한 Java 타입으로 데이터 처리
- StructuredOutputConvertor
 - Chat Client에서 entity()에 타입 설정으로 자동으로 적절한 컨버터가 설정됨
 - 설정하지 않을 경우 Raw Output이 바로 전달
 - stream 사용시 최종 완료 후 Convertor 수행
 - LLM 응답을 파싱하기 위한 반복적인 코드 제거 파싱 로직 간소화
 - LLM이 항상 원하는 구조로 응답하지 않을 수 있으므로, 결과 검증 로직 구현 필요
 - OpenAI, Azure OpenAI, Ollama, Mistral AI 등에서 구조화 출력 옵션 지원

Structured Output

- StructuredOutputConvertor 처리 상세
 - 프롬프트에 형식 지침을 추가
 - LLM이 원하는 출력 구조를 생성하도록 유도
 - 컨버터가 이를 지정된 구조화된 형식으로 변환



```

record ActorsFilms(String actor, List<String> movies) {}

ActorsFilms actorsFilms = ChatClient.create(chatModel).prompt()
    .user(u -> u.text("Generate the filmography of 5 movies for {actor}."))
    .param("actor", "Tom Hanks")
    .call()
    .entity(ActorsFilms.class);

BeanOutputConverter<ActorsFilms> beanOutputConverter =
    new BeanOutputConverter<>(ActorsFilms.class);

String format = this.beanOutputConverter.getFormat();

String actor = "Tom Hanks";

String template = """
    Generate the filmography of 5 movies for {actor}.
    {format}
""";

Generation generation = chatModel.call(
    new PromptTemplate(this.template, Map.of("actor", this.actor, "format", this.format)).create()).getResults();

ActorsFilms actorsFilms = this.beanOutputConverter.convert(this.generation.getOutput().getText());

```

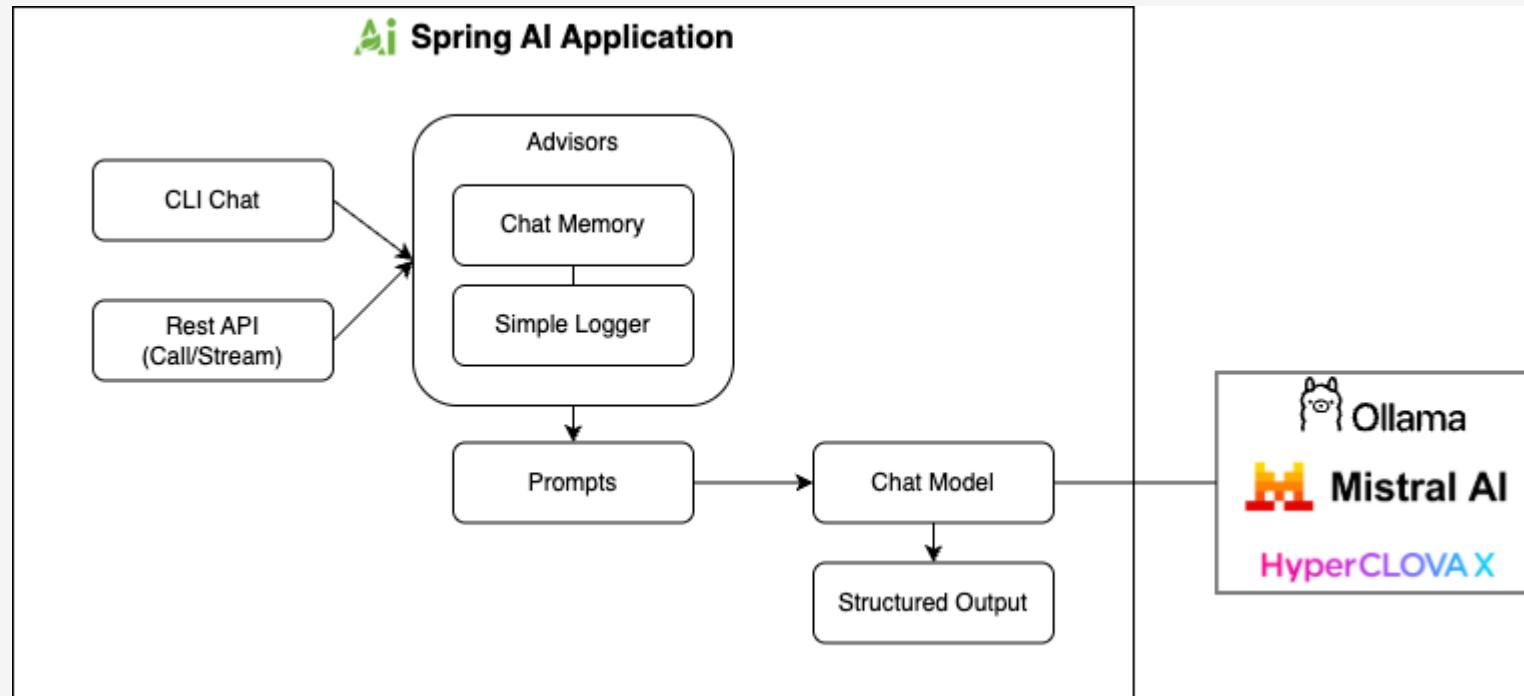
The code snippet shows the creation of a prompt for generating an actor's filmography, the execution of the prompt, and the conversion of the generated text into a structured `ActorsFilms` object using a bean output converter.

Spring AI Chat 개발

개발 환경 구축 (Auto Configuration)

개발 환경 구축 (Auto Configuration)

- 전체 코드
 - <https://github.com/JM-Lab/spring-ai-fast-campus-course/tree/main/fast-campus-course-chat>
- Spring AI 개발 맛보기와 동일한 구성



Chat Memory Advisors 개발

Chat Memory Advisors 개발

- FastCampusCourseChatConfig

```
// 디버깅 및 모니터링에 유용하며, 기본 로깅 포맷과 커스터마이징 기능을 지원
@Bean
public SimpleLoggerAdvisor simpleLoggerAdvisor() {
    return SimpleLoggerAdvisor.builder().build();
}
```

```
# SimpleLoggerAdvisor 등의 Advisor에서 DEBUG 로그 출력
logging.level.org.springframework.ai.chat.client.advisor=DEBUG
```

```
// MessageWindowChatMemory를 사용해 최근 메시지를 유지하는 채팅 메모리
// 초과 시 오래된 메시지를 순차적으로 제거하며, SystemMessage는 보관하지 않음
@Bean
public ChatMemory chatMemory() {
    return MessageWindowChatMemory.builder()
        .maxMessages(10) // 최대 보관 메시지 수: 10
        .build();
}
```

```
// ChatClient 호출 전후에 chatMemory를 이용해 대화 내역을 프롬프트에 자동으로 주입하거나 응답 저장을 수행
@Bean
public MessageChatMemoryAdvisor messageChatMemoryAdvisor(ChatMemory chatMemory) {
    return MessageChatMemoryAdvisor.builder(chatMemory)
        .build();
}
```

Stream 출력 CLI Chatbot 개발

Stream 출력 CLI Chatbot 개발

- ChatService

```
@Service
public class ChatService {

    private final ChatClient chatClient;

    public ChatService(ChatClient.Builder chatClientBuilder, Advisor[] advisors) {
        this.chatClient = chatClientBuilder.defaultAdvisors(advisor).build();
    }

    private ChatClient.ChatClientRequestSpec buildChatClientRequestSpec(String conversationId, Prompt prompt) {
        return chatClient.prompt(prompt)
            .advisors(AdvisorSpec advisor -> advisor.param(ChatMemory.CONVERSATION_ID, conversationId));
    }

    public Flux<String> stream(String conversationId, Prompt prompt) {
        return buildChatClientRequestSpec(conversationId, prompt).stream().content();
    }
}
```

- FastCampusCourseChatConfig

```
# CLI 모드 활성화 여부
spring.application.cli=true

@ConditionalOnProperty(prefix = "spring.application", name = "cli", havingValue = "true")
@Bean
public CommandLineRunner cli(@Value("${spring.application.name}") String applicationName, ChatService chatService) {
    return String[] args -> {

        LoggerContext context = (LoggerContext) LoggerFactory.getLoggerFactory();
        context.getLogger("ROOT").detachAppender(name: "CONSOLE");

        System.out.println("\n" + applicationName + " CLI bot");
    }
}
```

Chat Rest API 개발 (call, stream)

Chat Rest API 개발 (call, stream)

- ChatService

```
public ChatResponse call(String conversationId, Prompt prompt) {
    return buildChatClientRequestSpec(conversationId, prompt).call().chatResponse();
}
```

- ChatController

```
@RestController
@RequestMapping(value = "/chat")
class ChatController {

    private final ChatService chatService;

    public ChatController(ChatService chatService) {
        this.chatService = chatService;
    }

    public record PromptBody(
        @NotEmpty @Schema(description = "대화 식별자", example = "conv-1234") String conversationId,
        @NotEmpty @Schema(description = "사용자 입력 프롬프트", example = "안녕하세요, 오늘 날씨 어때요?") String userPrompt,
        @Nullable @Schema(description = "시스템 프롬프트(선택)", example = "You are a helpful assistant.") String systemPrompt,
        @Nullable @Schema(description = "채팅 음성(선택)", implementation = DefaultChatOptions.class) DefaultChatOptions chatOptions
    ) {}
}
```

```
@PostMapping(value = "/call", produces = MediaType.APPLICATION_JSON_VALUE)
ChatResponse call(@RequestBody @Valid PromptBody promptBody) {
    Prompt.Builder promptBuilder = getPromptBuilder(promptBody);
    return this.chatService.call(promptBody.conversationId, promptBuilder.build());
}
```

```
private static Prompt.Builder getPromptBuilder(PromptBody promptBody) {
    List<Message> messages = new ArrayList<>();
    Optional.ofNullable(promptBody.systemPrompt).filter(Predicate.not(String::isBlank))
        .map(String systemPrompt -> SystemMessage.builder().text(systemPrompt).build()).ifPresent(messages::add);
    messages.add(UserMessage.builder().text(promptBody.userPrompt).build());
    Prompt.Builder promptBuilder = Prompt.builder().messages(messages);
    Optional.ofNullable(promptBody.chatOptions).ifPresent(promptBuilder::chatOptions);
    return promptBuilder;
}
```

```
@PostMapping(value = "/stream", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
Flux<String> stream(@RequestBody @Valid PromptBody promptBody) {
    Prompt.Builder promptBuilder = getPromptBuilder(promptBody);
    return this.chatService.stream(promptBody.conversationId, promptBuilder.build());
}
```

Structured Output Converter 개발

Structured Output Converter 개발

- ChatService

```
public enum Emotion {VERY_NEGATIVE, NEGATIVE, NEUTRAL, POSITIVE, VERY_POSITIVE}

public record EmotionEvaluation(Emotion emotion, List<String> reason) {}

public EmotionEvaluation callEmotionEvaluation(String conversationId, Prompt prompt) {
    return buildChatClientRequestSpec(conversationId, prompt).call().entity(EmotionEvaluation.class);
}
```

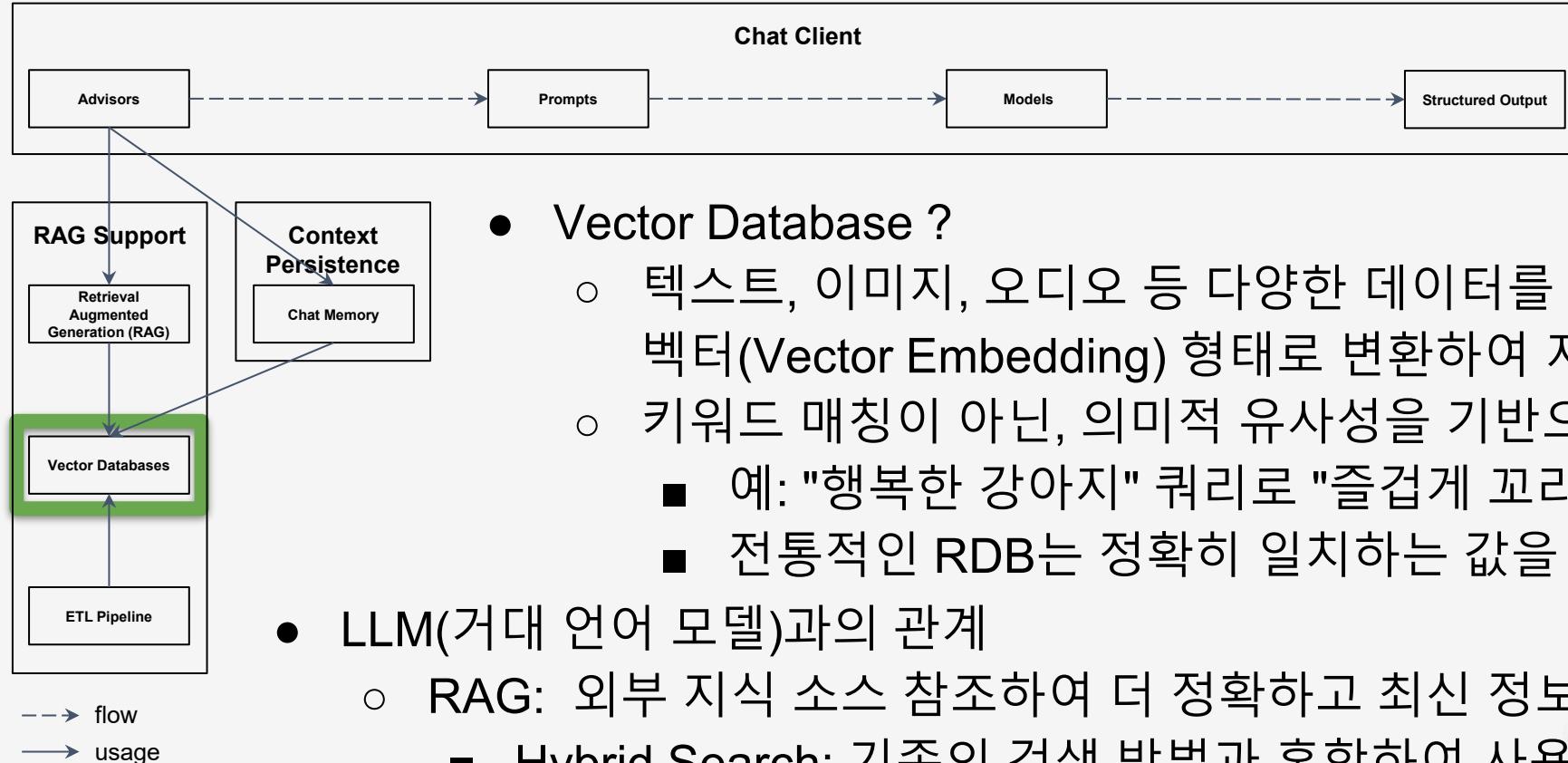
- ChatController

```
@PostMapping(value = "/emotion", produces = MediaType.APPLICATION_JSON_VALUE)
ChatService.EmotionEvaluation callEmotionEvaluation(@RequestBody @Valid PromptBody promptBody) {
    Prompt.Builder promptBuilder = getPromptBuilder(promptBody);
    return this.chatService.callEmotionEvaluation(promptBody.conversationId, promptBuilder.build());
}
```

Spring AI Vector Databases

Vector Databases

Vector Databases

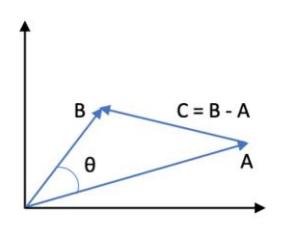


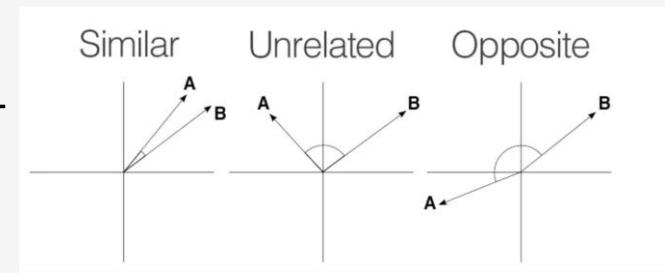
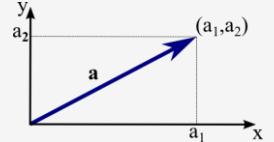
- **Vector Database ?**
 - 텍스트, 이미지, 오디오 등 다양한 데이터를 고차원 벡터(Vector Embedding) 형태로 변환하여 저장
 - 키워드 매칭이 아닌, 의미적 유사성을 기반으로 문서를 검색
 - 예: "행복한 강아지" 쿼리로 "즐겁게 꼬리 치는 개" 검색
 - 전통적인 RDB는 정확히 일치하는 값을 찾음
- **LLM(거대 언어 모델)과의 관계**
 - RAG: 외부 지식 소스 참조하여 더 정확하고 최신 정보기반 답변 생성
 - Hybrid Search: 기존의 검색 방법과 혼합하여 사용 할 수도 있음
 - 장기 기억 장치 (Long-term Memory): LLM 기반 챗봇 등이 이전 대화나 사용자 정보를 저장하고 비슷한 내용을 검색

Vector Similarity

Vector Similarity

- Vector ?
 - 벡터는 방향과 크기를 가진 수학적 객체로, AI에서는 주로 고차원 공간의 좌표
 - 2차원 벡터 예 - $[a_1, a_2]$
- 유사도(Similarity)란?
 - 두 벡터가 얼마나 비슷한지(가까운지)를 수치로 나타내는
 - 그들 사이의 각도 θ 는 유사성을 측정하는 좋은 방법
- Law of Cosines

$$a^2 + b^2 - 2ab \cos \theta = c^2$$

- Cosine Similarity
 - 코사인 유사도가 가장 직관적이고, 고차원(수백~수천 차원)에서도 잘 동작함
- Spring AI similarity score
 - 0에서 1 사이의 값으로 유사도를 정량화하며, 1에 가까울수록 더 높은 유사성을 의미

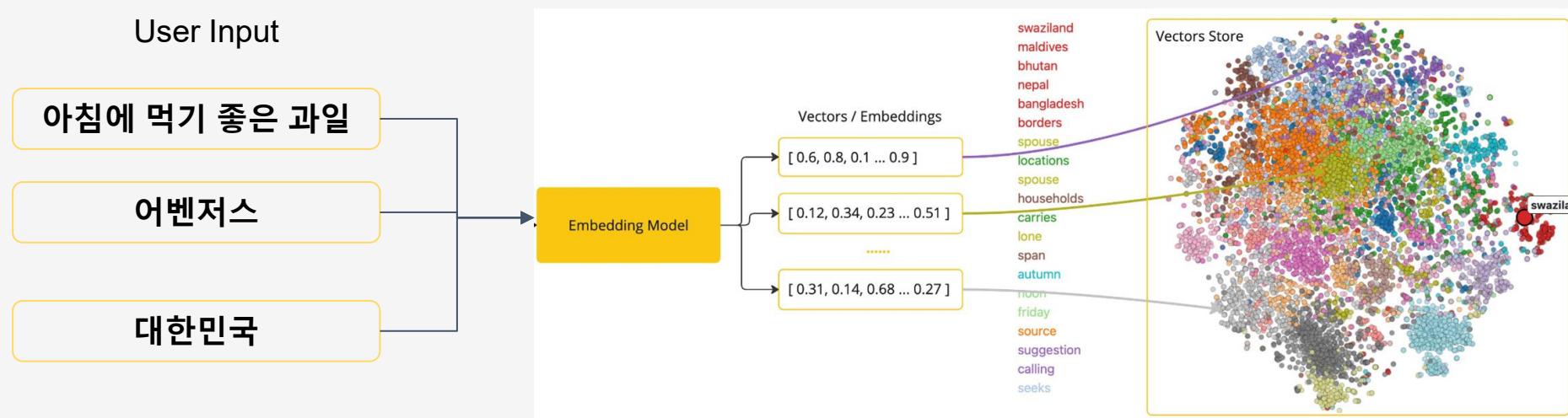


Cosine Similarity with vector components

$$\text{Similarity}(\vec{A}, \vec{B}) = \cos(\theta) = \frac{\sum_{i=1}^n \{A_i B_i\}}{\sqrt{\sum_{i=1}^n \{A_i^2\} \cdot \sum_{i=1}^n \{B_i^2\}}}$$

Vector Similarity

- Vector Store
 - Embedding Model 을 사용해 Embedding 값을 데이터를 넣어 둔 Semantic Space 실체
- Spring AI의 Vector Store
 - Vector Database 의 구현 클래스
 - 입력의 Embedding 값을 저장된 값들과 비교해서 의미 기반 검색 (Semantic Search) 제공
 - similarity score 계산



Spring AI Vector Store API

Spring AI Vector Store API

- 다양한 종류의 벡터 데이터베이스를 일관된 추상화 인터페이스 API로 사용
 - 특정 Vector Database 기술에 종속되지 않고 유연하게 애플리케이션 개발
 - 지원하는 Vector Database 종류
 - Azure Vector Search, Apache Cassandra, Chroma, Elasticsearch, MariaDB, Milvus, MongoDB Atlas, Neo4j, OpenSearch, Oracle, PgVector(PostgreSQL), Pinecone, Qdrant, Redis, SAP Hana, Typesense, Weaviate 등
 - Hybrid Search 지원?
 - 1.0 GA 에서 공식 적으로 지원하지는 않음
 - Advisor 를 통한 구현 방법
 - Vector Store 검색 결과 + 기존 검색 엔진 결과 -> 최종 검색 결과 생성

Spring AI Vector Store API

- VectorStore Interface
 - SearchRequest
 - topK, similarityThreshold, filterExpression 등 지정
 - Document
 - 텍스트와 메타데이터 포함한 기본 저장 정보
 - Document + 임베딩 벡터를 추가한 정보가 실제 저장
 - VectorStore 구현에 따라 다름
 - NoSQL 타입에서는 기본값으로 “embedding” 사용
 - Elasticsearch, OpenSearch 등

```
public interface VectorStore extends DocumentWriter {
    default String getName() {
        return this.getClass().getSimpleName();
    }

    void add(List<Document> documents);

    void delete(List<String> idList);

    void delete(Filter.Expression filterExpression);

    default void delete(String filterExpression) { ... };

    List<Document> similaritySearch(String query);

    List<Document> similaritySearch(SearchRequest request);

    default <T> Optional<T> getNativeClient() {
        return Optional.empty();
    }
}
```

Spring AI Metadata Filters

Spring AI Metadata Filters

- VectorStore 유사도 검색 시 Document의 metadata 필드를 기반으로 필터링
 - Filter.Expression 객체로 조건 지정
 - SQL과 유사한 문자열로 조건 지정
 - 예: "country == 'BG'", "genre == 'drama' && year >= 2020"
 - 다양한 연산자 지원: ==, !=, >, <, >=, <=, IN, AND, OR 등
 - 사용 예
 - 버전 관리
 - 날짜 관리

```
// Create initial document (v1) with version metadata
Document documentV1 = new Document(
    "AI and Machine Learning Best Practices",
    Map.of(
        "docId", "AIML-001",
        "version", "1.0",
        "lastUpdated", "2024-01-01"
    )
);

// Add v1 to the vector store
vectorStore.add(List.of(documentV1));

// Create updated version (v2) of the same document
Document documentV2 = new Document(
    "AI and Machine Learning Best Practices - Updated",
    Map.of(
        "docId", "AIML-001",
        "version", "2.0",
        "lastUpdated", "2024-02-01"
    )
);

// First, delete the old version using filter expression
Filter.Expression deleteOldVersion = new Filter.Expression(
    Filter.ExpressionType.AND,
    Arrays.asList(
        new Filter.Expression(
            Filter.ExpressionType.EQ,
            new Filter.Key("docId"),
            new Filter.Value("AIML-001")
        ),
        new Filter.Expression(
            Filter.ExpressionType.EQ,
            new Filter.Key("version"),
            new Filter.Value("1.0")
        )
    )
);
vectorStore.delete(deleteOldVersion);

// Add the new version
vectorStore.add(List.of(documentV2));

// Verify only v2 exists
SearchRequest request = SearchRequest.builder()
    .query("AI and Machine Learning")
    .filterExpression("docId == 'AIML-001'")
    .build();
List<Document> results = vectorStore.similaritySearch(request);
// results will contain only v2 of the document
```

Retrieval Augmented Generation (RAG)

RAG 를 사용하는 이유?

RAG 를 사용하는 이유?

- 대형 언어 모델(LLM)의 단점을 극복하기 위한 유용한 기술
 - 사실에 기반한 정확성
 - 문맥 인식 능력에 한계
- 주요 이유
 - 최신 정보 활용
 - LLM의 고정된 학습 데이터 한계 극복
 - **최신 데이터를 계속해서 업데이트 가능**
 - 환각 현상 감소
 - 검증된 출처 기반 답변 생성으로 신뢰성 향상
 - RAG 용 데이터 생성시 출처를 포함하고 이를 **검증된 출처**로 사용
 - 도메인 특화
 - 내부 문서/DB와 연동해 전문 분야 질의 처리
 - **보안이 중요한 분야 LLM 활용 가능** (망분리 환경, 전금법 등)
 - Local LLM + 내부 문서/DB 사용시 내부 자료 외부 유출 방지

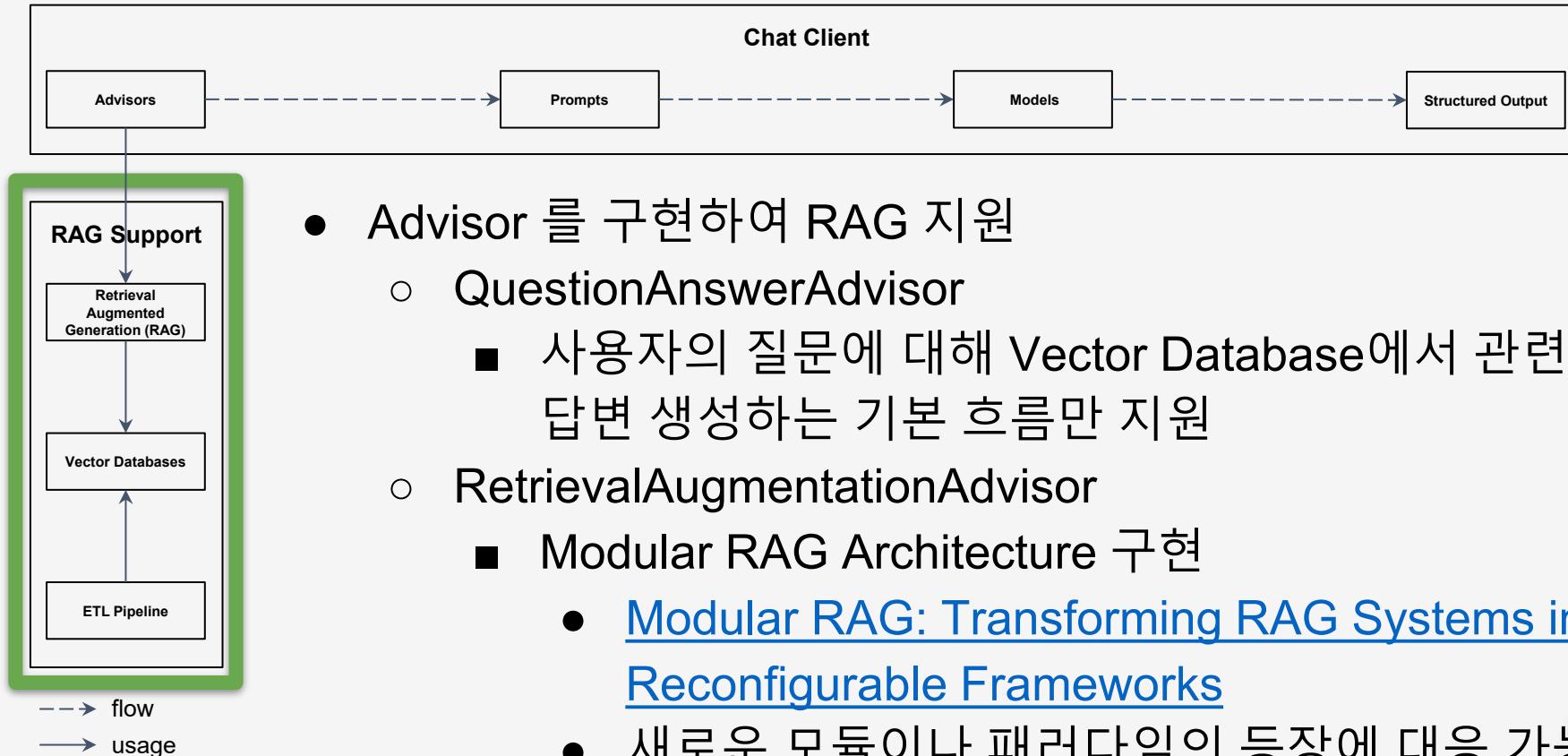
RAG를 위해 Vector Database 가 필수 인가?

RAG를 위해 Vector Database 가 필수 인가?

- NO, 필수는 아님
 - 이미 잘 정의 된 자연어 검색 방법이 있다면 기존의 방식을 사용
 - 사용자 질문 -> LLM 사용 기존 검색 쿼리로 변경 -> 검색 결과 사용
 - 데이터 API 서비스
 - Web 검색, 법률, 의료 등 전문 분야의 데이터가 API 로 제공되는 서비스 사용
- Vector Database 기반 RAG 사용 이유
 - **내부 데이터가 비정형으로 존재하고 기존 검색 시스템이 없는 경우**
 - 기존 검색 엔진 구축 보다 쉽게 구축하고 일정 수준의 결과를 얻기 쉬움
 - 다국어로 서비스를 제공하는 경우
 - 다국어 지원 Embedding Model의 경우 언어가 달라도 검색 가능
 - 기존 방식의 자연어 검색 엔진 유지 보수가 어려운 경우
 - Embedding Model 변경으로 검색 성능 향상
 - 다시 Embedding 이 필요함 하지만 기존 자연어 검색보다 간단함

Spring AI RAG 구현

Spring AI RAG 구현



- Advisor 를 구현하여 RAG 지원
 - QuestionAnswerAdvisor
 - 사용자의 질문에 대해 Vector Database에서 관련 문서를 찾아 답변 생성하는 기본 흐름만 지원
 - RetrievalAugmentationAdvisor
 - Modular RAG Architecture 구현
 - [Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks](#)
 - 새로운 모듈이나 패러다임의 등장에 대응 가능
- ETL Pipeline 지원
 - 다양한 양식에서 Chunk 추출, Vector 생성, Document로 저장 지원

Spring AI RAG 구현

- RetrievalAugmentationAdvisor 의 Modular RAG 처리 흐름
 1. Pre-Retrieval: 쿼리 변환 및 전처리
 - a. QueryTransformer(압축, 재작성, 번역 등)를 체인처럼 연결해 쿼리를 변환
 2. Query Expansion: 쿼리 확장
 - a. MultiQueryExpander는 질문을 여러 표현으로 확장해 다양한 관점의 쿼리 생성
 3. **Retrieval: 문서 검색 (필수 설정)**
 - a. VectorStoreDocumentRetriever는 다양한 조건으로 VectorStore 검색 수행
 4. Document Join & Post-Retrieval: 문서 결합 및 후처리
 - a. 리랭킹, 중복/불필요 문서 제거, 요약, 컨텍스트 길이 최적화 등으로 LLM에 전달할 정보를 정제
 5. Generation & Augmentation: 문서 기반 답변 생성
 - a. 문서 리스트를 쿼리에 컨텍스트로 추가(augment)하여, 최종 프롬프트에 반영
 - b. ContextualQueryAugmenter는 검색 된 것이 없을 경우 처리 지원

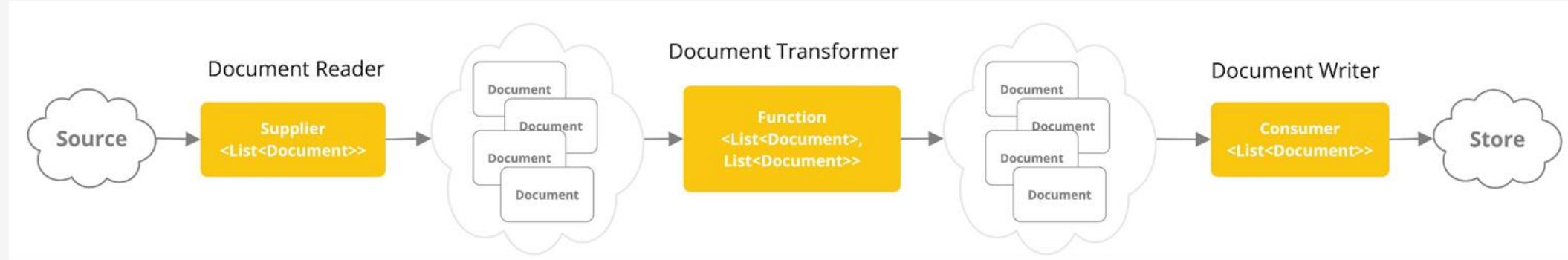
Spring AI RAG 구현

- RetrievalAugmentationAdvisor 의 Modular RAG 구현 예
 - a. 쿼리 전처리(변환)와 문서 후처리 등 다양한 고급 기능을 조합

```
Advisor retrievalAugmentationAdvisor = RetrievalAugmentationAdvisor.builder()  
    .queryTransformers(RewriteQueryTransformer.builder()  
        .chatClientBuilder(chatClientBuilder.build().mutate())  
        .build())  
    .documentRetriever(VectorStoreDocumentRetriever.builder()  
        .similarityThreshold(0.50)  
        .vectorStore(vectorStore)  
        .build())  
    .build();  
  
String answer = chatClient.prompt()  
    .advisors(retrievalAugmentationAdvisor)  
    .user(question)  
    .call()  
    .content();
```

Spring AI ETL Pipeline

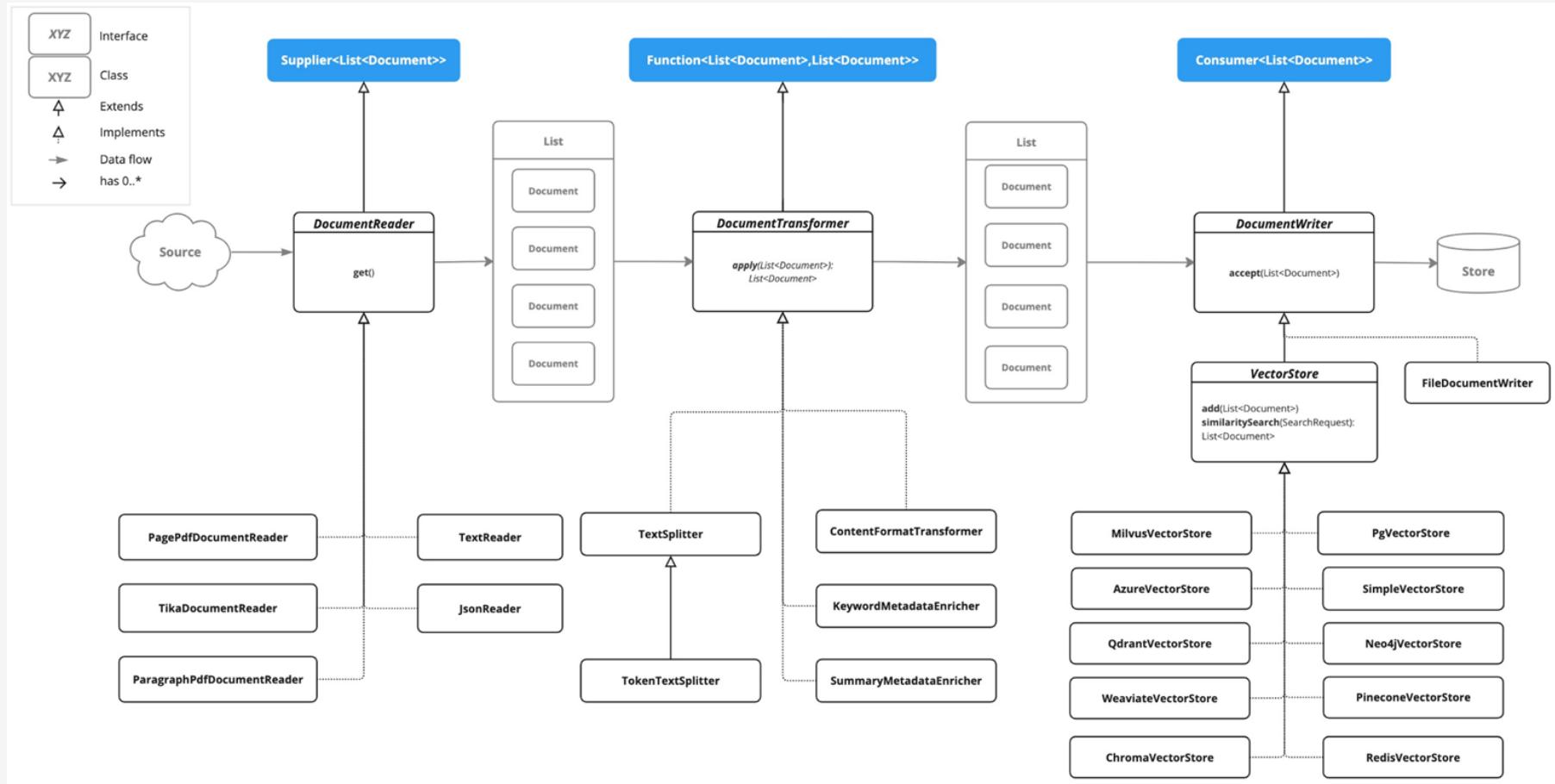
Spring AI ETL Pipeline



- 추출 (Extract)
 - DocumentReader
 - 다양한 데이터 소스(파일, DB, 웹 등)에서 Document 리스트를 생성
- 변환 (Transform)
 - DocumentTransformer
 - Document 리스트를 입력받아 청킹, 요약, 키워드 추출 등 변환 작업
- 적재 (Load)
 - DocumentWriter
 - 최종적으로 변환된 Document 리스트를 VectorStore에 저장

Spring AI ETL Pipeline

- ETL Class Diagram



Spring AI ETL Pipeline

- DocumentReader
 - 다양한 형식의 데이터를 Document 객체로 변환하는 핵심 컴포넌트
- 텍스트 및 웹 형식 DocumentReader
 - JsonReader
 - jsonKeysToUse로 특정 키의 값만 추출. JSON 포인터로 중첩 데이터 접근 가능
 - TextReader
 - 단순 텍스트 파일을 Document 1개로 변환
 - getCustomMetadata()로 정보 추가
 - JsoupDocumentReader
 - JSoup 기반 HTML selector (CSS 선택자)로 원하는 부분만 정밀하게 추출
 - MarkdownDocumentReader
 - Markdown 구조(수평선, 코드 블록 등)를 이해하고 문서를 지능적으로 분할

Spring AI ETL Pipeline

- PDF와 오피스 문서 처리 DocumentReader
 - PagePdfDocumentReader
 - 페이지 단위로 문서를 잘라 여러 Document로 만듬
 - ParagraphPdfDocumentReader
 - PDF 목차(TOC) 정보를 이용해 논리적 단락별로 Document로 만듬
 - 목차 정보 필요
 - TikaDocumentReader
 - Apache Tika를 사용해 수많은 문서 포맷을 하나의 리더로 처리 (만능 리더)
 - PDF, DOC/DOCX, PPT/PPTX, HTML 등

Spring AI ETL Pipeline

- DocumentTransformer
 - Document를 AI 모델에 더 적합한 형태로 가공하거나 정보를 추가하는 역할
 - AI의 컨텍스트 창(Context Window)에 맞춰 긴 문서를 작은 조각(Chunk)으로 나눔
- 문서 분할 DocumentTransformer
 - TokenTextSplitter
 - 텍스트를 토큰으로 변환 (OpenAI 모델과 호환되는 CL100K_BASE 인코딩 사용)
 - 단순히 자르지 않고, 문장 종결부(., ?, \n 등)에서 끊어 의미적으로 자연스러운 분할
 - 주요 파라미터
 - defaultChunkSize: 목표 청크 크기 (토큰 수, 기본값: 800)
 - minChunkSizeChars: 청크의 최소 문자 수 (기본값: 350)
 - keepSeparator: 줄바꿈 등 구분자 유지 여부 (기본값: true)
 - 필요한 경우 TextSplitter Interface 구현 필요
 - Spring AI 1.0 ver.에서 Text Length로 분할 하는 것 제공하지 않음

Spring AI ETL Pipeline

- 메타데이터 자동 생성 및 강화 DocumentTransformer
 - KeywordMetadataEnricher
 - ChatModel을 이용해 각 문서 내용의 핵심 키워드를 추출
 - metadata에 콤마(,)로 구분된 키워드 문자열이 excerpt_keywords 키추가
 - 문서 태깅, 검색 용이성 향상
 - SummaryMetadataEnricher
 - ChatModel을 이용해 각 문서의 내용을 요약
 - metadata에 section_summary 키로 요약이 추가
 - 이전/다음 문서의 요약까지 함께 추가 가능
 - prev_section_summary, next_section_summary 키 사용

Spring AI ETL Pipeline

- **DocumentWriter**
 - Document 객체 리스트를 최종적으로 외부에 저장

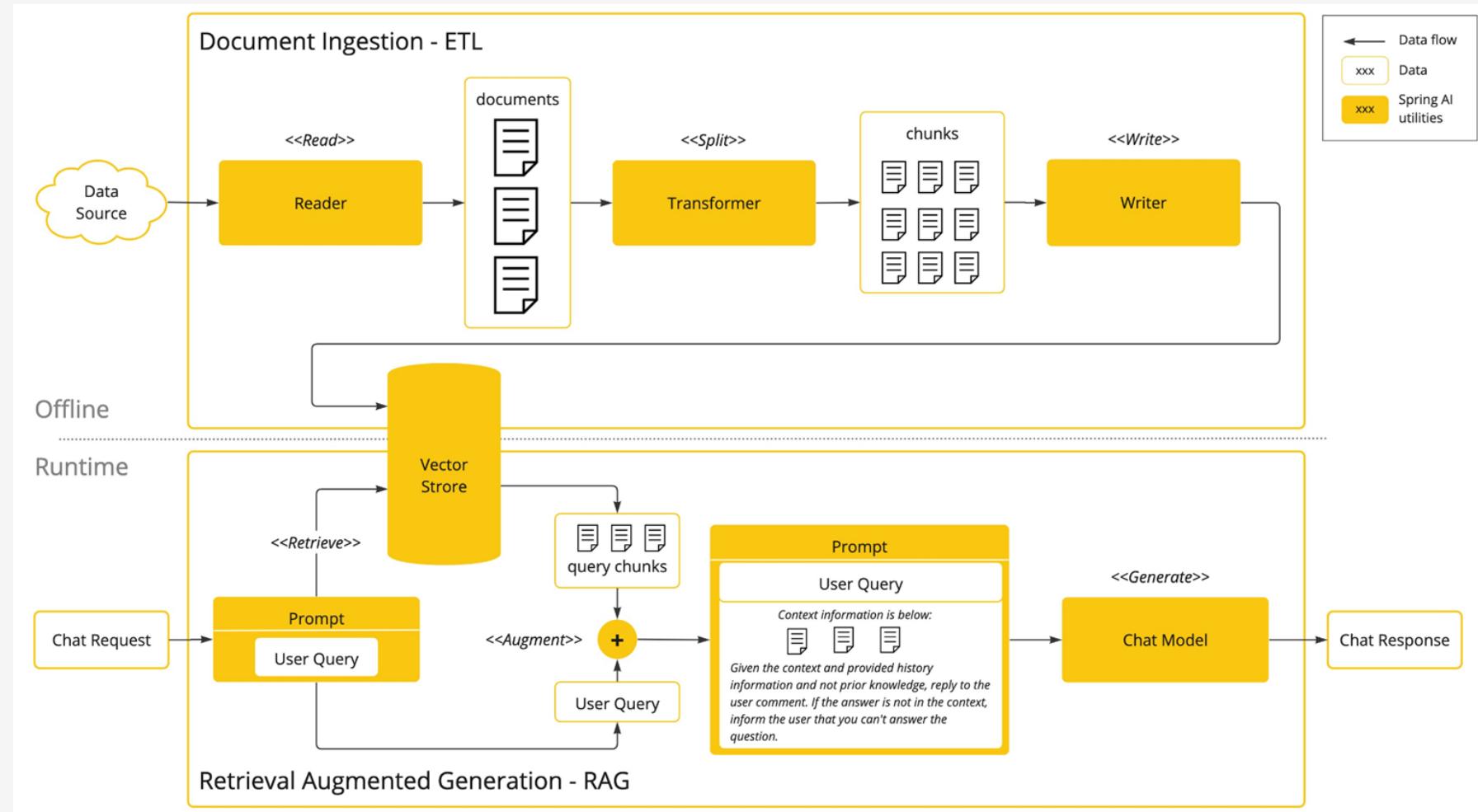
```
public interface DocumentWriter extends Consumer<List<Document>> {  
  
    default void write(List<Document> documents) { accept(documents); }  
  
}
```

- **DocumentWriter 종류**
 - **FileDocumentWriter**
 - Document 객체 리스트의 내용을 파일로 저장
 - 사람이 읽기 쉬운 텍스트 파일 형태로 변환하여 **디버깅 및 분석 용도**로 활용
 - 저장 용도가 아님
 - **VectorStore**
 - 기본으로 DocumentWriter 를 구현

```
public interface VectorStore extends DocumentWriter {
```

Spring AI RAG Process

Spring AI RAG Process

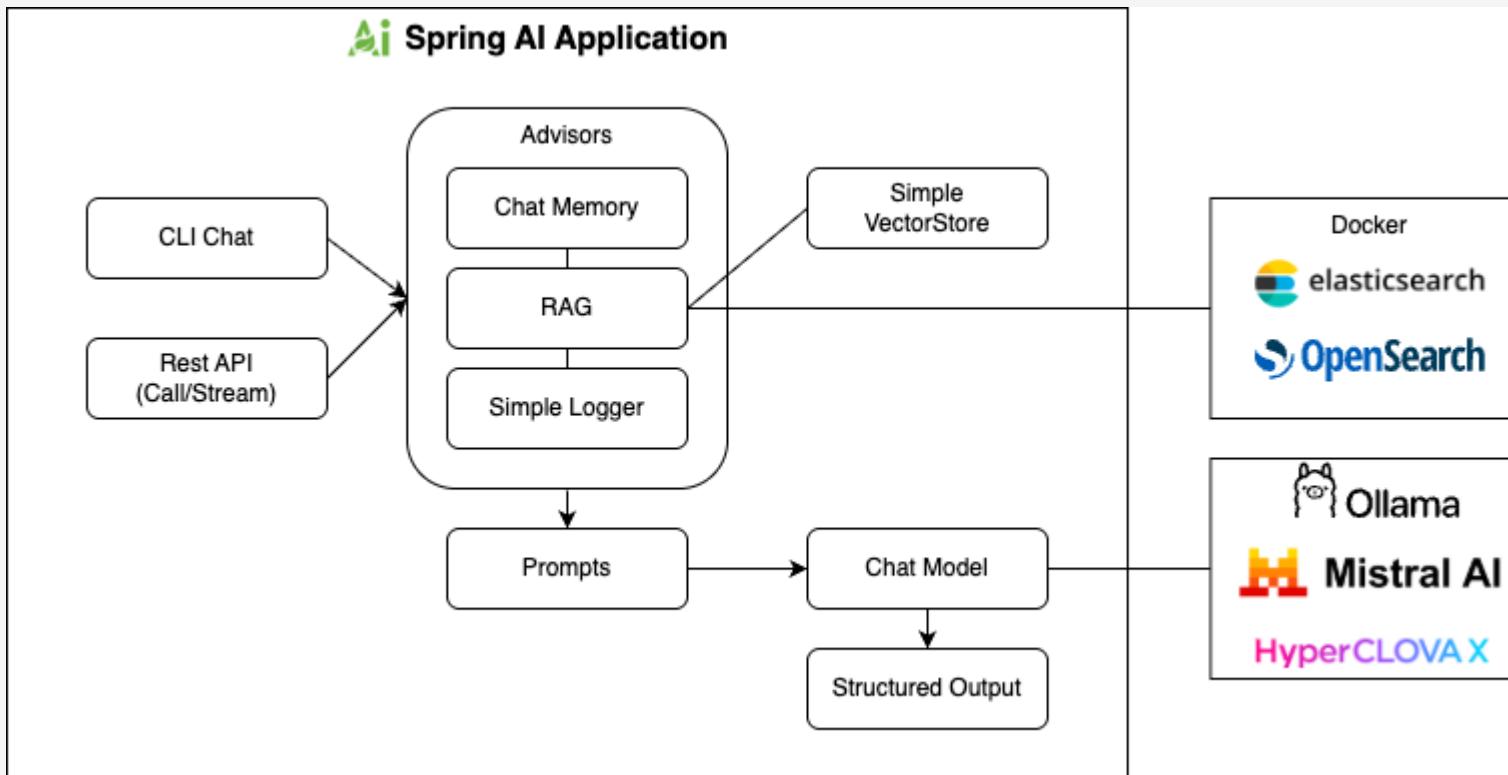


Spring AI RAG Chat 개발

개발 환경 구축 (Auto Configuration)

개발 환경 구축 (Auto Configuration)

- 전체 코드
 - <https://github.com/JM-Lab/spring-ai-fast-campus-course/tree/main/fast-campus-course-rag-chat>
- Spring AI RAG Chat 구성



개발 환경 구축 (Auto Configuration)

- pom.xml 의존성 추가

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-advisors-vector-store</artifactId>
</dependency>
```

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-tika-document-reader</artifactId>
</dependency>
```

- application.properties

- SimpleVectorStore (In-Memory) 사용시 설정 추가 없음

```
# CLI 모드 활성화 여부
app.cli.enabled=true

# RAG Data 위치
app.rag.documents-location-pattern=classpath:fastcampus-springai.pdf

spring.application.name=fast-campus-course-rag

# SimpleLoggerAdvisor 등의 Advisor에서 DEBUG 로그 출력
logging.level.org.springframework.ai.chat.client.advisor=DEBUG

# 여러 Chat 모델 사용시 auto-configurations 에서 사용할 모델 설정 필요 예: openai, ollama
spring.ai.model.chat=ollama
# Structured Output, Tool Calling 사용시 mistral (7B q4 기본 모델로 약 7GB 메모리 필요) 사용
spring.ai.ollama.chat.options.model=hf.co/rippertnt/HyperCLOVAX-SEED-Text-Instruct-1.5B-Q4_K_M-GGUF
# when_missing, always, never 설정
spring.ai.ollama.init.pull-model-strategy=when_missing

# 여러 Chat 모델 사용시 auto-configurations 에서 사용할 모델 설정 필요 예: openai, ollama
spring.ai.model.embedding=ollama
## embedding model, 기본 모델 mxbai-embed-large 는 영어 전용 모델
spring.ai.ollama.embedding.options.model=bge-m3
```

ETL Pipeline 개발

ETL Pipeline 개발

- RagConfig
 - PathMatchingResourcePatternResolver 사용
 - Ant 패턴 지원
 - **, *, ? 와 함께 file: 또는 classpath:
 - 파일 시스템에서 재귀 탐색
 - file:/data/**/*.json
 - 클래스패스 내 특정 패턴 매치
 - classpath:**/fastcampus-springai.pdf
 - DocumentReaders

```
@Bean
public DocumentReader[] documentReaders(
    @Value("${app.rag.documents-location-pattern}") String documentsLocationPattern) throws
IOException {
    return Arrays.stream(new PathMatchingResourcePatternResolver().getResources(documentsLocationPattern))
        .map(TikaDocumentReader::new).toArray(DocumentReader[]::new);
}
```

ETL Pipeline 개발

- RagConfig
 - DocumentTransformer
 - LengthTextSplitter 사용
 - chunckSize, chunkOverlap 을 가지는 TextSplitter 직접 구현
 - TextSplitter 는 DocumentTransformer 를 상속

```
@Bean
public TextSplitter textSplitter() {
    return new LengthTextSplitter(chunkSize: 1000, chunkOverlap: 200);
}
```

- KeywordMetadataEnricher
 - Metadata 에 Keyword 를 추가하는 DocumentTransformer 추가

```
@Bean
public KeywordMetadataEnricher keywordMetadataEnricher(ChatModel chatModel) {
    return new KeywordMetadataEnricher(chatModel, keywordCount: 3);
}
```

ETL Pipeline 개발

- RagConfig
 - DocumentWriter
 - jsonConsoleDocumentWriter 직접 구현
 - 최종 생성된 Chunk 를 확인
 - FileDocumentWriter 는 File 로 저장되어 즉시 확인이 안됨

```
@Bean
public DocumentWriter jsonConsoleDocumentWriter(ObjectMapper objectMapper) {
    return documents -> {
        System.out.printf("===== save chunks size %d =====\n", documents.size());
        try {
            System.out.println(objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(documents));
        } catch (JsonProcessingException e) {
            throw new RuntimeException(e);
        }
        System.out.println("=====");
    };
}
```

- SimpleVectorStore
 - 기본 제공되는 In-Memory Vectorstore 를 사용

```
@Bean
public VectorStore vectorStore(EmbeddingModel embeddingModel) {
    return SimpleVectorStore.builder(embeddingModel).build();
}
```

ETL Pipeline 개발

- RagConfig
 - ETL Pipeline 구현
 - 최초 실행시 ETL Pipeline 이 실행 되도록 함

```
@Order(1) // cli 보다 먼저 실행
@Bean
public ApplicationRunner initEtlPipeline(DocumentReader[] documentReaders, DocumentTransformer textSplitter,
    DocumentTransformer keywordMetadataEnricher, DocumentWriter[] documentWriters) {
    return ApplicationArguments args ->
        // Extract
        Arrays.stream(documentReaders).map(DocumentReader::read)
            // Transform
            .map(textSplitter)
            .map(keywordMetadataEnricher)
            // Load
            .forEach( List<Document> documents -> Arrays.stream(documentWriters)
                .forEach( DocumentWriter documentWriter -> documentWriter.write(documents)));
}
```

RAG Advisors 개발

RAG Advisors 개발

- pom.xml

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-rag</artifactId>
</dependency>
```

- RagConfig

- QuestionAnswerAdvisor

- RAG 기본 기본 구성만 지원

- RetrievalAugmentationAdvisor

- Modular RAG architecture 지원 사용 모듈 pom.xml 추가

- RAG 기본 기본 구성

```
@Bean
public RetrievalAugmentationAdvisor retrievalAugmentationAdvisor(VectorStore vectorStore) {
    RetrievalAugmentationAdvisor.Builder retrievalAugmentationAdvisorBuilder =
        RetrievalAugmentationAdvisor.builder()
            .documentRetriever(VectorStoreDocumentRetriever.builder().similarityThreshold(0.3).topK(3)
                .vectorStore(vectorStore).build());
    return retrievalAugmentationAdvisorBuilder.build();
}
```

CLI 개발

CLI 개발

- RagChatService
 - RAG 를 위한 ChatClient 개발

```
@Service
public class RagChatService {

    private final ChatClient chatClient;

    public RagChatService(ChatClient.Builder chatClientBuilder, Advisor[] advisors) {
        // RAG 에서는 검색된 내용을 기반으로 정확한 정보를 생성해야 하므로 생성 다양성을 줄이기 위해 temperature를 0.0으로 설정
        this.chatClient = chatClientBuilder.defaultOptions(ChatOptions.builder().temperature(0.0).build())
            .defaultAdvisors(advisors).build();
    }

    private ChatClient.ChatClientRequestSpec buildChatClientRequestSpec(String conversationId, Prompt prompt,
        Optional<String> filterExpressionAsOpt) {
        ChatClient.ChatClientRequestSpec chatClientRequestSpec = chatClient.prompt(prompt)
            .advisors( AdvisorSpec advisors -> advisors.param(ChatMemory.CONVERSATION_ID, conversationId));
        // filterExpression 이 있을 경우 VectorStore 검색 필터 설정
        filterExpressionAsOpt.ifPresent( String filterExpression -> chatClientRequestSpec.advisors(
            AdvisorSpec advisors -> advisors.param(VectorStoreDocumentRetriever.FILTER_EXPRESSION, filterExpressionAsOpt)));
        return chatClientRequestSpec;
    }

    public Flux<String> stream(String conversationId, Prompt prompt, Optional<String> filterExpressionAsOpt) {
        return buildChatClientRequestSpec(conversationId, prompt, filterExpressionAsOpt).stream().content();
    }

    public ChatResponse call(String conversationId, Prompt prompt, Optional<String> filterExpressionAsOpt) {
        return buildChatClientRequestSpec(conversationId, prompt, filterExpressionAsOpt).call().chatResponse();
    }

}
```

CLI 개발

- FastCampusCourseRagChatConfig
 - FastCampusCourseChatConfig 와 동일한 설정
 - SimpleLoggerAdvisor, ChatMemory, MessageChatMemoryAdvisor

```
@ConditionalOnProperty(prefix = "app.cli", name = "enabled", havingValue = "true")
@Bean
public CommandLineRunner cli(@Value("${spring.application.name}") String applicationName,
    RagChatService ragChatService,
    // app.cli.filter-expression 없으면 빈 문자열
    @Value("${app.cli.filter-expression:}") String filterExpression) {
    return String[] args -> {

        LoggerContext context = (LoggerContext) LoggerFactory.getLoggerFactory();
        context.getLogger(name: "ROOT").detachAppender(name: "CONSOLE");

        System.out.println("\n" + applicationName + " CLI bot");
        try (Scanner scanner = new Scanner(System.in)) {
            while (true) {
                System.out.print("\nUSER: ");
                String userMessage = scanner.nextLine();
                ragChatService.stream(conversationId: "cli", Prompt.builder().content(userMessage).build(),
                    // filterExpression 을 Optional로 전달
                    Optional.of(filterExpression).filter(Predicate.not(String::isBlank))
                    .doFirst(() -> System.out.print("\nASSISTANT: "))
                    .doOnNext(System.out::print)
                    .doOnComplete(System.out::println)
                    .blockLast();
            }
        }
    };
}
```

CLI 개발

- RagConfig

```
@Bean
public RetrievalAugmentationAdvisor retrievalAugmentationAdvisor(VectorStore vectorStore,
    Optional<DocumentPostProcessor> documentsPostProcessor) {
    RetrievalAugmentationAdvisor.Builder retrievalAugmentationAdvisorBuilder =
        RetrievalAugmentationAdvisor.builder()
            .queryAugmenter(ContextualQueryAugmenter.builder().allowEmptyContext(true).build())
            .documentRetriever(VectorStoreDocumentRetriever.builder().similarityThreshold(0.3).topK(3))
            .vectorStore(vectorStore).build();

    // RAG CLI 를 위해 등록
    documentsPostProcessor.ifPresent(retrievalAugmentationAdvisorBuilder::documentPostProcessors);
    return retrievalAugmentationAdvisorBuilder.build();
}
```

```
@ConditionalOnProperty(prefix = "app.cli", name = "enabled", havingValue = "true")
@Bean
public DocumentPostProcessor printDocumentsPostProcessor() {
    return ( Query query, List<Document> documents ) -> {
        System.out.println("\n[ Search Results ]");
        System.out.println("=====");

        if (documents == null || documents.isEmpty()) {
            System.out.println(" No search results found.");
            System.out.println("=====");
            return documents;
        }

        for (int i = 0; i < documents.size(); i++) {
            Document document = documents.get(i);
            System.out.printf("▶ %d Document, Score: %.2f%\n", i + 1, document.getScore());
            System.out.println("-----");
            Optional.ofNullable(document.getText()).stream()
                .map( String text -> text.split( regex: "\n" ) ).flatMap(Arrays::stream)
                .forEach( String line -> System.out.printf("%s\n", line));
            System.out.println("=====");
        }
        System.out.print("\n[ RAG 사용 응답 ]\n\n");
        return documents;
    };
}
```

- application.properties

```
# CLI 실행 시 filter-expression
app.cli.filter-expression=

# CLI 모드 활성화 여부
app.cli.enabled=true
```

Rest API 개발

Rest API 개발

- RagChatController

```
@RestController
@RequestMapping(value = "/rag")
class RagChatController {

    private final RagChatService ragChatService;

    public RagChatController(RagChatService ragChatService) { this.ragChatService = ragChatService; }

    public record RagPromptBody(
        @NotEmpty @Schema(description = "대화 식별자", example = "conv-1234") String conversationId,
        @NotEmpty @Schema(description = "사용자 입력 프롬프트", example = "안녕하세요, 오늘 날씨 어때요?") String userPrompt,
        @Nullable @Schema(description = "시스템 프롬프트(선택)", example = "You are a helpful assistant.") String systemPrompt,
        @Nullable @Schema(description = "체팅 옵션(선택)", implementation = DefaultChatOptions.class) DefaultChatOptions chatOptions,
        @Nullable @Schema(description = "VectorStore FilterExpression", example = "source == 'document1.pdf'") String filterExpression
    ) {}
}
```

```
@PostMapping(value = "/call", produces = MediaType.APPLICATION_JSON_VALUE)
ChatResponse call(@RequestBody @Valid RagPromptBody ragPromptBody) {
    Prompt.Builder promptBuilder = getPromptBuilder(ragPromptBody);
    return this.ragChatService.call(ragPromptBody.conversationId, promptBuilder.build(),
        Optional.ofNullable(ragPromptBody.filterExpression()));
}
```

```
private static Prompt.Builder getPromptBuilder(RagPromptBody ragPromptBody) {
    List<Message> messages = new ArrayList<>();
    Optional.ofNullable(ragPromptBody.systemPrompt).filter(Predicate.not(String::isBlank))
        .map( String systemPrompt -> SystemMessage.builder().text(systemPrompt).build()).ifPresent(messages::add);
    messages.add(UserMessage.builder().text(ragPromptBody.userPrompt).build());
    Prompt.Builder promptBuilder = Prompt.builder().messages(messages);
    Optional.ofNullable(ragPromptBody.chatOptions).ifPresent(promptBuilder::chatOptions);
    return promptBuilder;
}
```

```
@PostMapping(value = "/stream", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
Flux<String> stream(@RequestBody @Valid RagPromptBody ragPromptBody) {
    Prompt.Builder promptBuilder = getPromptBuilder(ragPromptBody);
    return this.ragChatService.stream(ragPromptBody.conversationId, promptBuilder.build(),
        Optional.ofNullable(ragPromptBody.filterExpression()));
}
```

상용 VectorStore 로 변경 개발

상용 VectorStore 로 변경 개발

- OpenSearch 사용

- pom.xml

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-starter-vector-store-opensearch</artifactId>
</dependency>
```

- application.properties

- AWS OpenSearch 사용

- AWS SDK

- 인증 설정 필요 없음

- AWS 내 EC2

- IAM Role 사용

- Auto Configuration

- 자동 VectorStore Bean 생성

```
# OpenSearch VectorStore 사용
spring.ai.vectorstore.opensearch.uris=<opensearch instance URIs>
spring.ai.vectorstore.opensearch.username=<opensearch username>
spring.ai.vectorstore.opensearch.password=<opensearch password>
spring.ai.vectorstore.opensearch.index-name=spring-ai-document-index
spring.ai.vectorstore.opensearch.initialize-schema=true
spring.ai.vectorstore.opensearch.similarity-function=cosinesimil

# Only for Amazon OpenSearch Service
spring.ai.vectorstore.opensearch.aws.host=<aws opensearch host>
spring.ai.vectorstore.opensearch.aws.service-name=<aws service name>
spring.ai.vectorstore.opensearch.aws.access-key=<aws access key>
spring.ai.vectorstore.opensearch.aws.secret-key=<aws secret key>
spring.ai.vectorstore.opensearch.aws.region=<aws region>
```

상용 VectorStore 로 변경 개발

- Elasticsearch 사용
 - application.properties
 - production 구분 옵션 추가

```
# 초기 데이터 로딩 여부
app.etl.pipeline.init=true

# in-memory SimpleVectorStore 사용 여부
app.vectorstore.in-memory.enabled=false
```

- RagConfig

```
@ConditionalOnProperty(prefix = "app.vectorstore.in-memory", name = "enabled", havingValue = "true")
@Bean
public VectorStore vectorStore(EmbeddingModel embeddingModel) {
    return SimpleVectorStore.builder(embeddingModel).build();
}

// ETL Pipeline
@ConditionalOnProperty(prefix = "app.etl.pipeline", name = "init", havingValue = "true")
@Order(1) // cli 보다 먼저 실행
@Bean
public ApplicationRunner initEtlPipeline(DocumentReader[] documentReaders, DocumentTransformer textSplitter,
```

상용 VectorStore 로 변경 개발

- Elasticsearch 사용

- pom.xml

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-starter-vector-store-elasticsearch</artifactId>
</dependency>
```

```
<dependency>
    <groupId>co.elastic.clients</groupId>
    <artifactId>elasticsearch-java</artifactId>
    <version>8.15.5</version>
</dependency>
```

- application.properties

```
# Elasticsearch VectorStore 사용
spring.elasticsearch.uris=<elasticsearch instance URIs>
spring.elasticsearch.username=<elasticsearch username>
spring.elasticsearch.password=<elasticsearch password>

spring.ai.vectorstore.elasticsearch.initialize-schema=true
spring.ai.vectorstore.elasticsearch.index-name=custom-index
spring.ai.vectorstore.elasticsearch.dimensions=1536
spring.ai.vectorstore.elasticsearch.similarity=cosine
```

- Auto Configuration

- VectorStore Bean 자동 생성

상용 VectorStore 로 변경 개발

- Elasticsearch 연결 테스트
 - docker 사용
 - docker run -it -p 9200:9200 -e "discovery.type=single-node" -e "xpack.security.enabled=false" -e "xpack.security.http.ssl.enabled=false" docker.elastic.co/elasticsearch/elasticsearch:8.18.3
 - Elasticsearch 인증 없이 접속할 수 있도록 실행
 - application.properties
 - 최초 실행시 데이터를 로드 했다면 이후 false 사용
 - docker 사용시 재실행시 초기화 될 수 있음

```
# 초기 데이터 로딩 여부  
app.etl.pipeline.init=false
```

Spring AI Tool Calling

Tool Calling

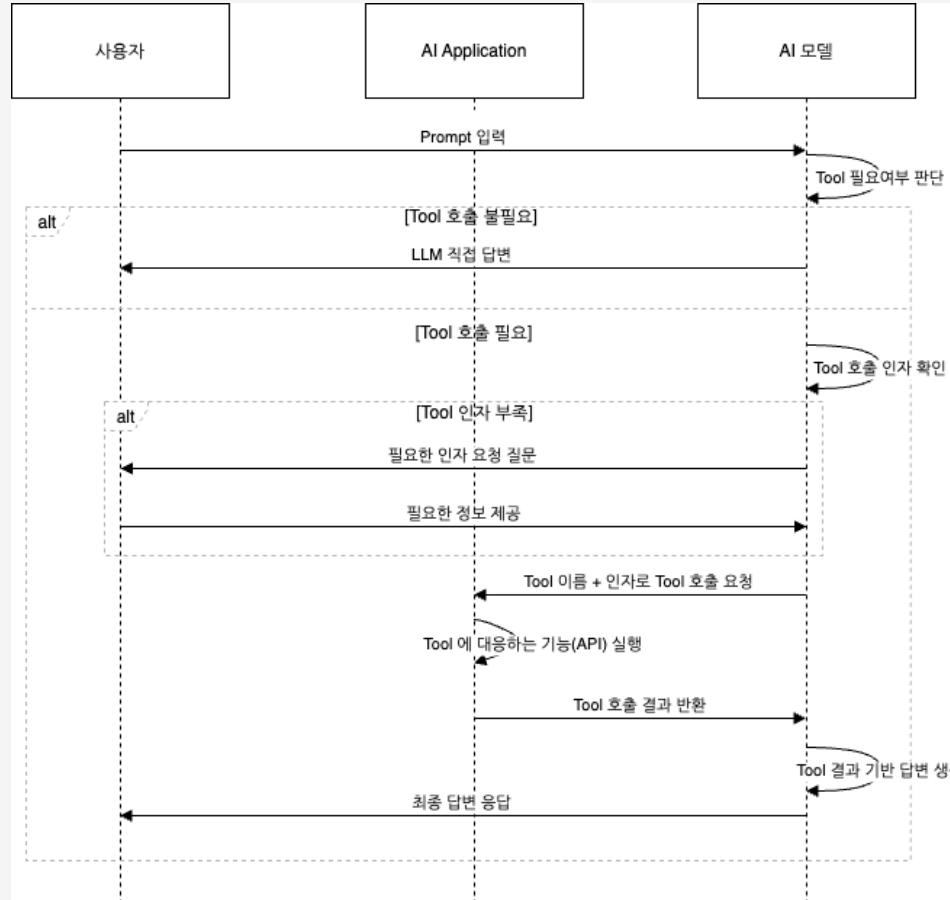
Tool Calling

- AI 모델이 외부 시스템과 상호작용할 수 있도록 하는 기능
 - Function Calling 이라고도 함
 - 주요 용도
 - 정보 검색 (Information Retrieval)
 - 작업 수행 (Taking Action)
- 필요한 것
 - AI Application 에 Tool 에 대응하는 기능 개발
 - AI 모델이 이해 할 수 있는 Tool 에 대응 하는 Tool Spec 정의
- Adviser 에서 RAG 나 API 를 호출하는 것과 다른 것인가?
 - **호출 요청을 AI 가 함**
 - 필요한 정보를 AI 가 수집
 - 다른 tool을 호출
 - 사용자에게 직접 질문으로 요청

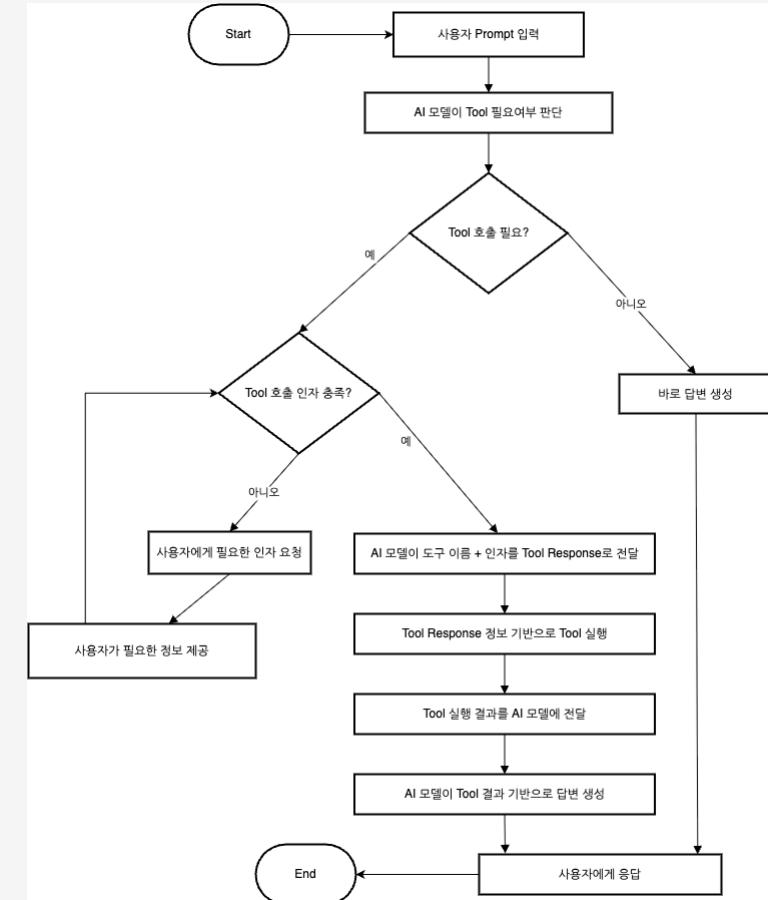
Tool Calling Process

Tool Calling Process

- Tool Calling Sequence Diagram

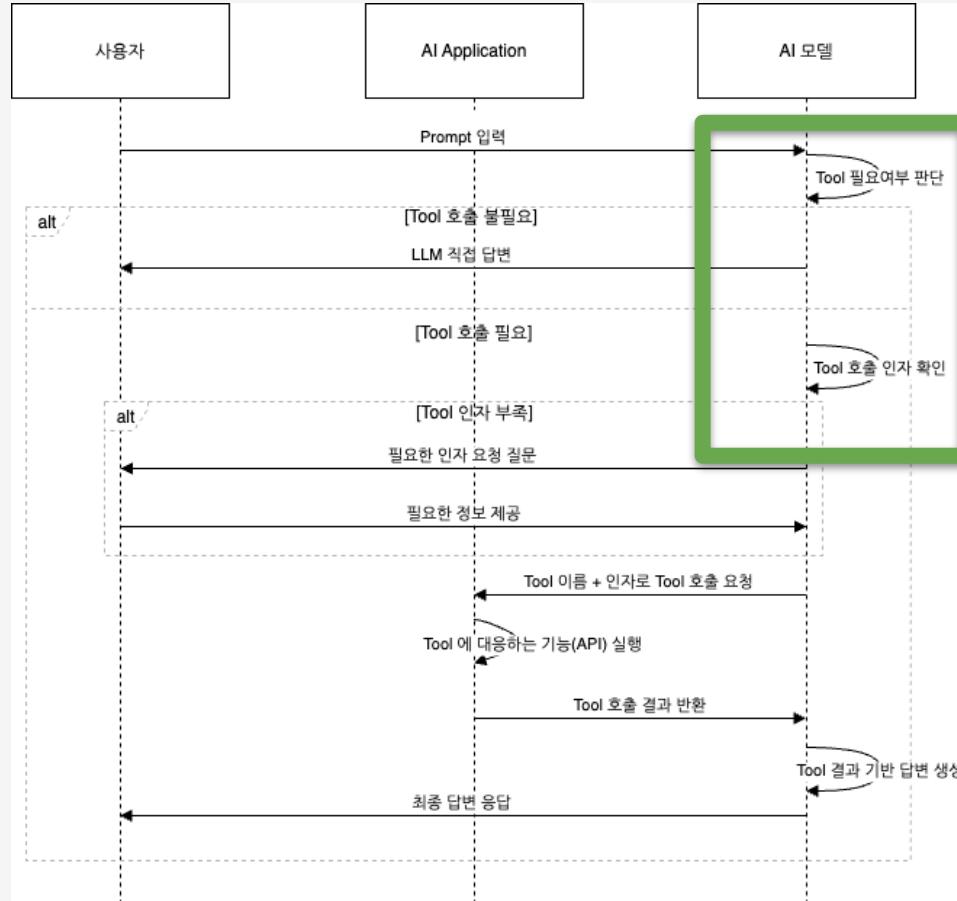


- Tool Calling Flow Diagram

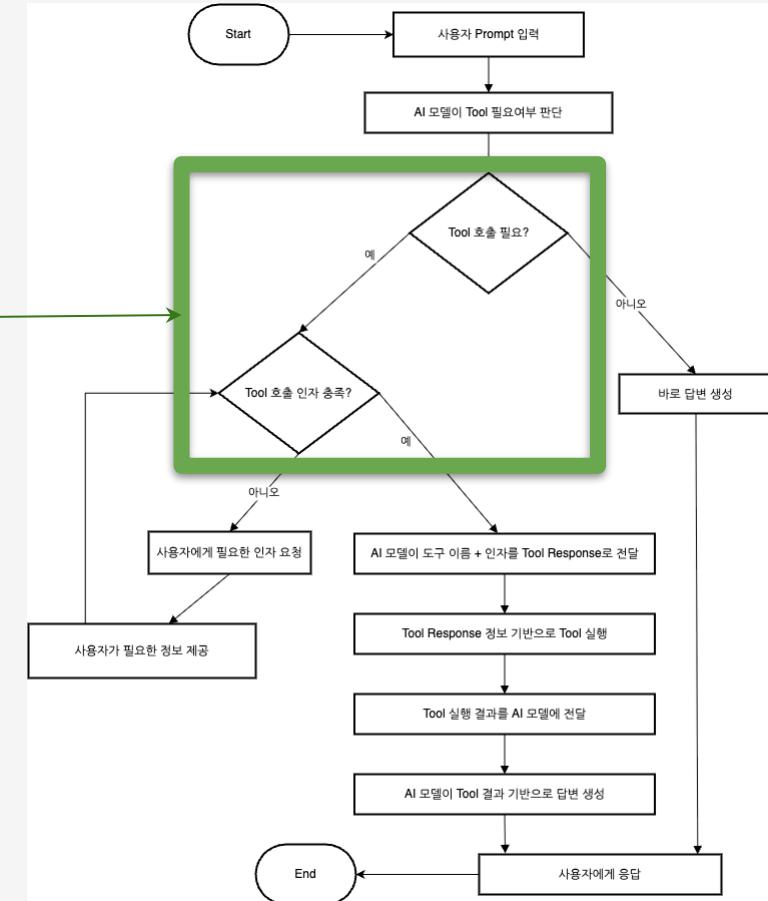


Tool Calling Process

- Tool Calling Sequence Diagram

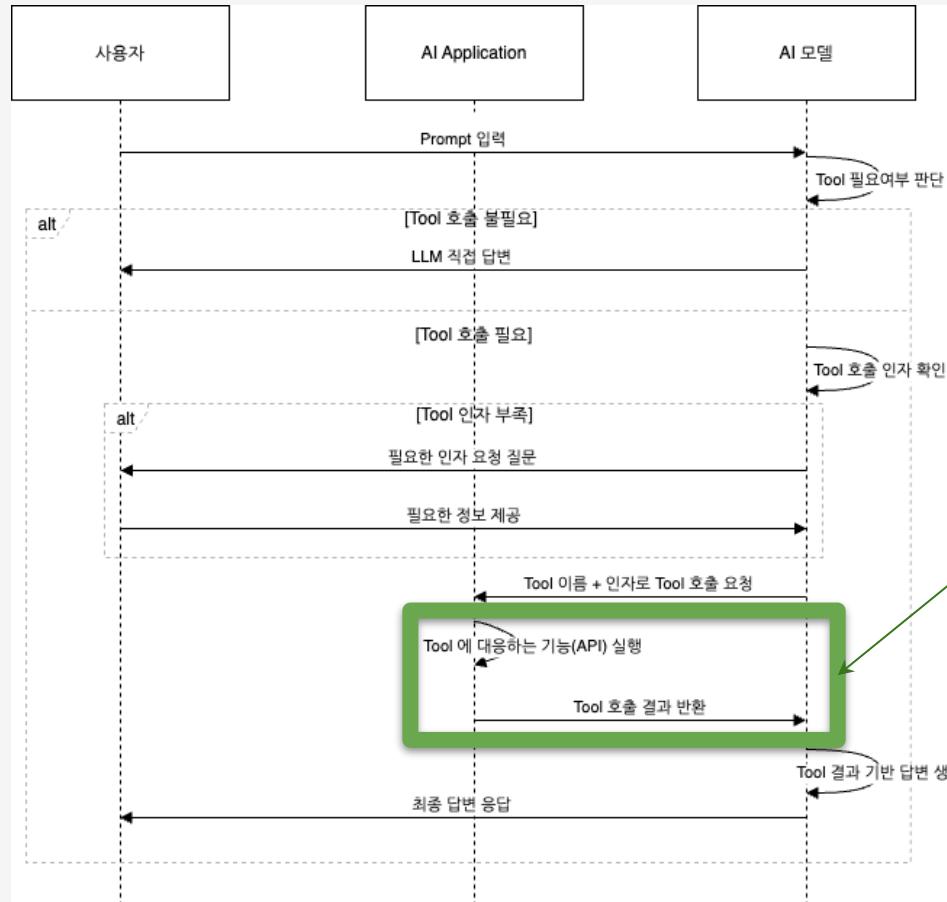


- Tool Calling Flow Diagram



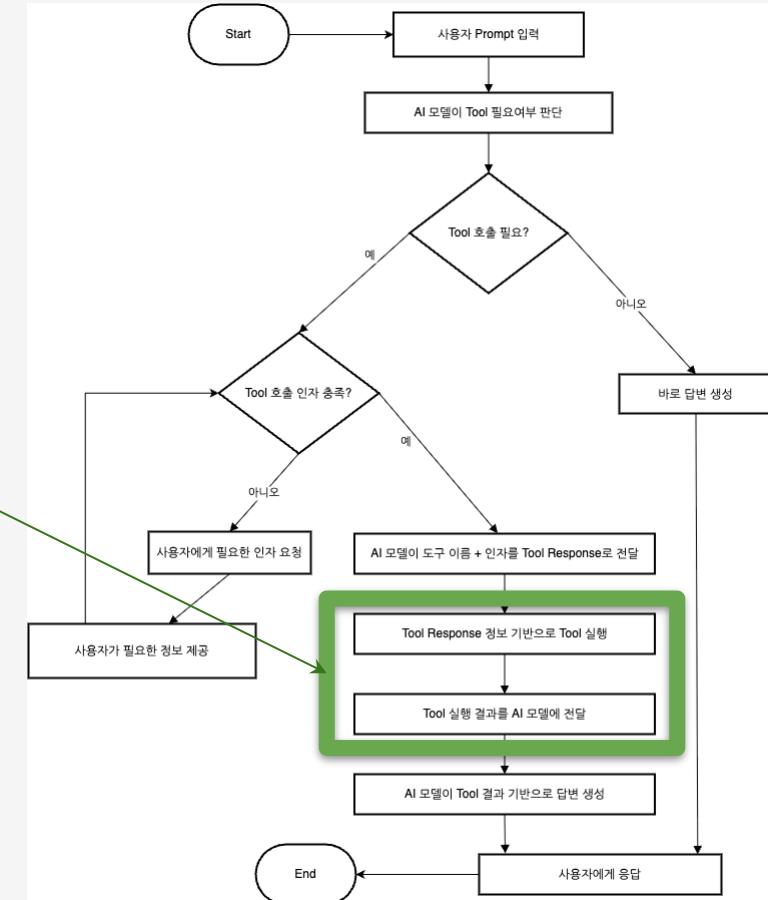
Tool Calling Process

- Tool Calling Sequence Diagram



개발 필요 부분!

- Tool Calling Flow Diagram



Tool Specification

Tool Specification

- LLM이 외부 함수(또는 API, 도구 등)를 호출할 수 있도록 함수의 정의와 입력 파라미터, 사용 목적을 JSON Schema 형태로 명확하게 기술하는 방식
- 구성 요소
 - type: 도구의 유형(대부분 "function")
 - name: 도구(함수)의 이름
 - description: 도구의 목적 및 사용법 설명
 - parameters: 입력값의 이름, 타입, 필수 여부 등(JSON Schema로 정의)
- OpenAI에서 최초 정의 한 규격
 - AI 모델마다 다를 수 있음

Example function schema

```

1  {
2    "type": "function",
3    "function": {
4      "name": "get_weather",
5      "description": "Retrieves current weather for the given location.",
6      "parameters": {
7        "type": "object",
8        "properties": {
9          "location": {
10            "type": "string",
11            "description": "City and country e.g. Bogotá, Colombia"
12          },
13          "units": {
14            "type": "string",
15            "enum": [
16              "celsius",
17              "fahrenheit"
18            ],
19            "description": "Units the temperature will be returned in."
20          }
21        },
22        "required": [
23          "location",
24          "units"
25        ],
26        "additionalProperties": false
27      },
28      "strict": true
29    }
30  }

```

Spring AI Tool Calling

Spring AI Tool Calling

- Spring AI Tool Specification
 - Tool Callback 인터페이스를 통해 Tool 을 모델링

 - Tool Definition
 - AI 모델이 도구의 존재와 사용법을 이해하는 데 필요한 모든 정보를 제공
- ```
ToolDefinition toolDefinition = ToolDefinition.builder()
 .name("currentWeather")
 .description("Get the weather in location")
 .inputSchema("""
 {
 "type": "object",
 "properties": {
 "location": {
 "type": "string"
 },
 "unit": {
 "type": "string",
 "enum": ["C", "F"]
 }
 },
 "required": ["location", "unit"]
 }
 """)
 .build();
```
- Tool Callback
    - AI 모델이 호출할 수 있는 Tool을 정의하고 실행
    - 주요 메서드
      - getToolDefinition()
        - Tool 정의 반환
      - getToolMetadata()
        - 실행 시 사용할 메타데이터
      - call(String toolInput), call(String toolInput, ToolContext toolContext)
        - Tool 매칭 시 실행 구현

# Spring AI Tool Calling

- Spring AI Tools 구현 방법
  - @Tool 어노테이션
    - name: 없으면 메서드명 사용, 중복 불가
    - **description:** AI 모델이 이해하도록 tool 설명 상세히 작성 권장
    - returnDirect: true 일 경우 결과 직접 반환
    - resultConverter: ToolCallResultConverter 를 구현해서 결과 String 을 변환
  - @ToolParam 어노테이션
    - **description:** AI 모델이 이해하도록 param 설명 상세히 작성 권장
    - required : 기본 true, false 설정하거나 @Nullable 이 있을 경우 false

```
class DateTimeTools {

 @Tool(description = "Set a user alarm for the given time")
 void setAlarm(@ToolParam(description = "Time in ISO-8601 format") String time) {
 LocalDateTime alarmTime = LocalDateTime.parse(time, DateTimeFormatter.ISO_DATE_TIME);
 System.out.println("Alarm set for " + alarmTime);
 }

}
```

# Spring AI Tool Calling

- Spring AI Tools 특징
  - @Tool 사용하는 데에는 거의 제약이 없음
    - Spring Bean(@Component 등)내에 있으면 별도 설정 없이 AOT 컴파일 지원
    - 정적(static) 또는 인스턴스 메서드 모두 Tool로 등록 가능
    - public, protected, package-private, private 모두 허용
    - 인스턴스화 가능한 최상위 클래스, 중첩 클래스 모두 가능
  - @Tool 메서드의 파라미터 정보를 기반으로 JSON Schema를 자동 생성
    - 파라미터 개수 제한 없이 원시 타입, enum, List, 배열, Map 등 대부분 지원
    - 반환 타입 void 또는 직렬화 가능한 타입
  - 다양한 어노테이션을 통해 파라미터 설명 및 옵션 지정 가능
    - Swagger의 @Schema
    - Jackson의 @JsonProperty, @JsonPropertyDescription

# Spring AI Tool Calling

- Spring AI Tools 사용 방법
  - ChatClient 에 defaultTools 등록
    - Tool 이름 직접 사용시 defaultToolNames 사용

```
ChatModel chatModel = ...
ChatClient chatClient = ChatClient.builder(chatModel)
 .defaultTools(new DateTimeTools())
 .build();
```

```
ChatModel chatModel = ...
ChatClient chatClient = ChatClient.builder(chatModel)
 .defaultToolNames("currentWeather")
 .build();
```

- ChatClient 에 tools 등록
  - Tool 이름 직접 사용시 toolNames 사용

```
ChatClient.create(chatModel)
 .prompt("What day is tomorrow?")
 .tools(new DateTimeTools())
 .call()
 .content();
```

```
ChatClient.create(chatModel)
 .prompt("What's the weather like in Copenhagen?")
 .toolNames("currentWeather")
 .call()
 .content();
```

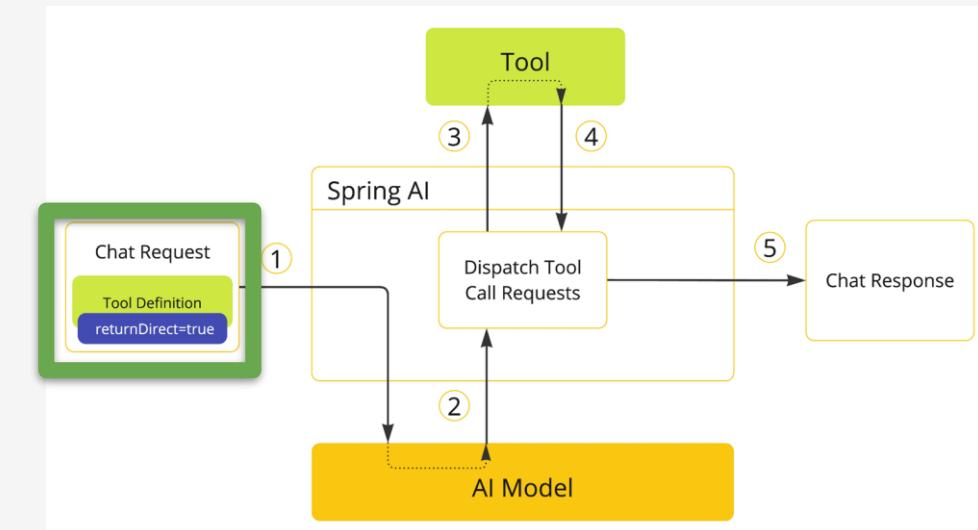
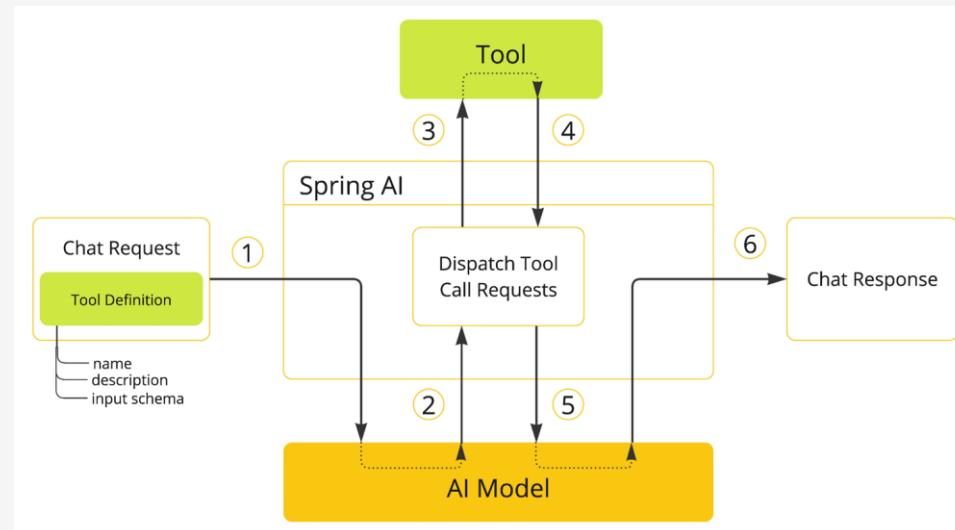
# Spring AI Tool Calling

- Spring AI Tools 구현 기타 방법
  - **@Bean 기반 동적 Tool 등록**
    - Spring Bean으로 Function, Supplier, Consumer, BiFunction 등이 Tool로 등록
    - @Bean 메서드명 = Tool 이름
    - @Description 어노테이션으로 Tool 설명 제공(미지정 시 메서드명이 설명)
  - 기존 클래스의 특정 Method를 직접 Tool로 변환
    - MethodToolCallback.Builder를 사용해 Tool의 정보를 설정
      - ChatClient.tools 에 직접 등록
    - 클래스에 정의된 일반 메서드를 Reflection을 통해 찾아 AI Tool로 포장
  - Java의 함수형 인터페이스 (Function, Supplier 등)를 Tool로 변환
    - FunctionToolCallback.Builder를 사용해 Tool의 정보를 설정
    - Function, Supplier, Consumer, BiFunction 등 함수형 객체를 Tool로 등록

# Spring AI Tool Execution

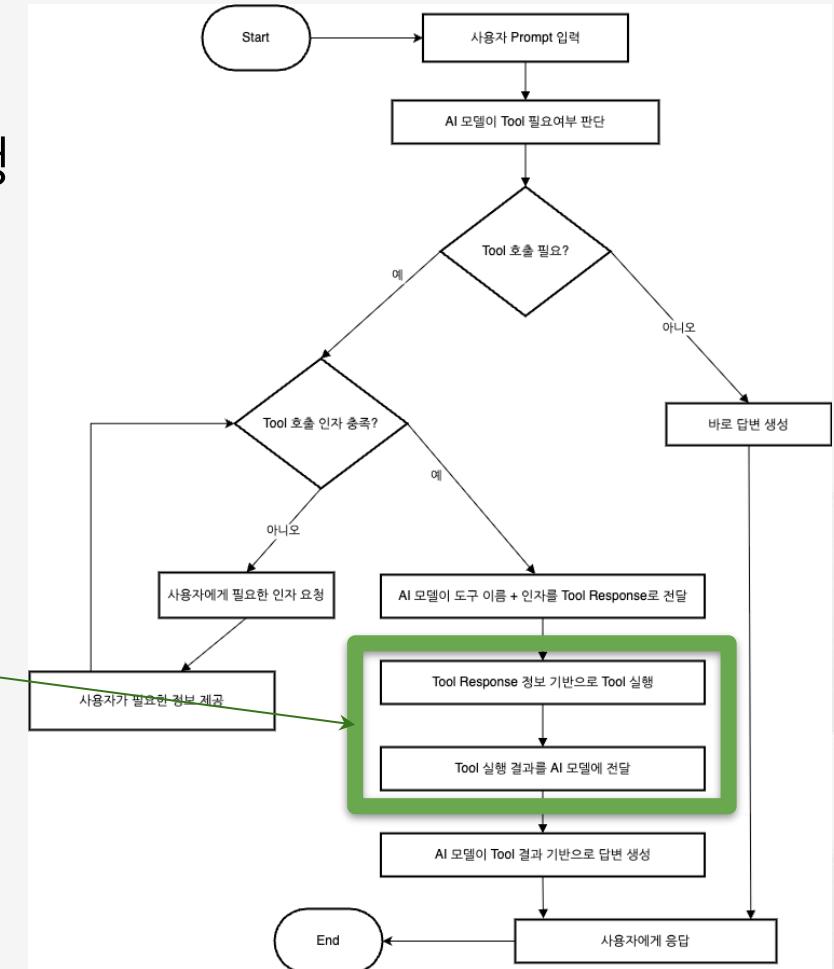
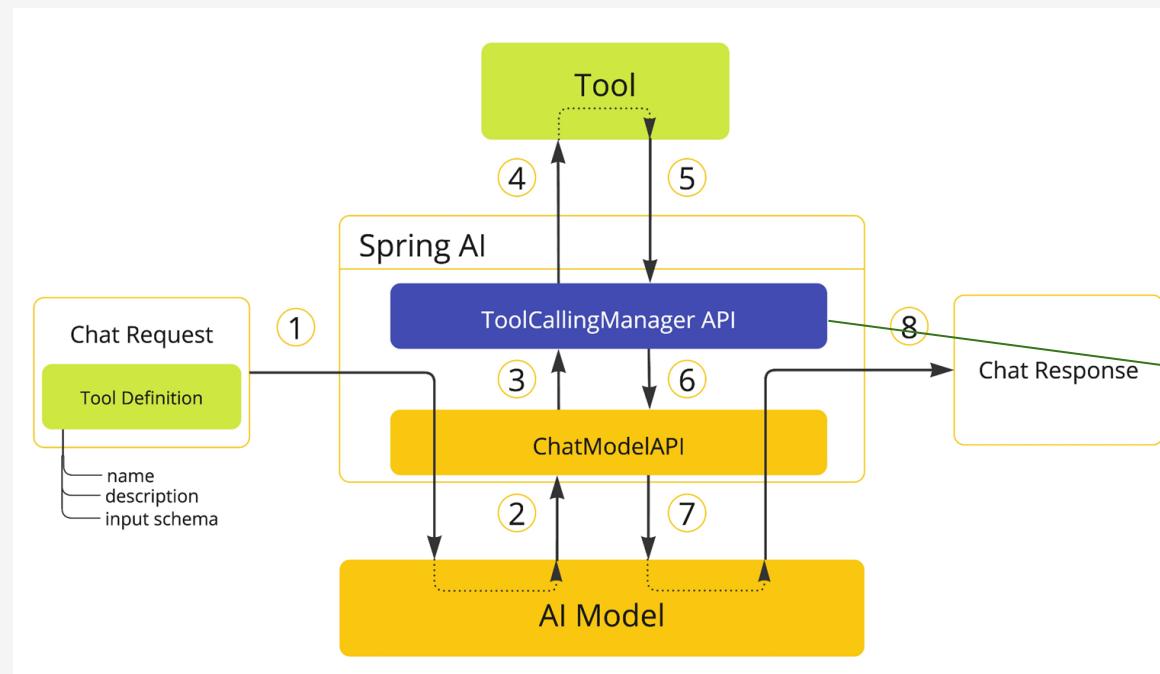
# Spring AI Tool Execution

1. 도구 정의(name, description, input schema)를 chat 요청에 포함 전달
2. AI 모델이 도구 호출 필요 시, 도구명과 인자를 응답으로 전송
3. AI 애플리케이션이 Tool 명으로 실제 개발된 Tool 실행
4. AI 애플리케이션이 Tool 결과 최종 처리 판단 (Return Direct 가능)
5. Tool 실행 결과를 모델에 전달
6. AI 모델이 Tool 결과를 활용해 최종 응답 생성



# Spring AI Tool Execution

- Spring AI Framework-Controlled Tool Execution
  - ToolCallingManager
    - 모든 Tool 호출 요청을 자동으로 가로채어, 툴 실행
    - 결과를 모델에 반환



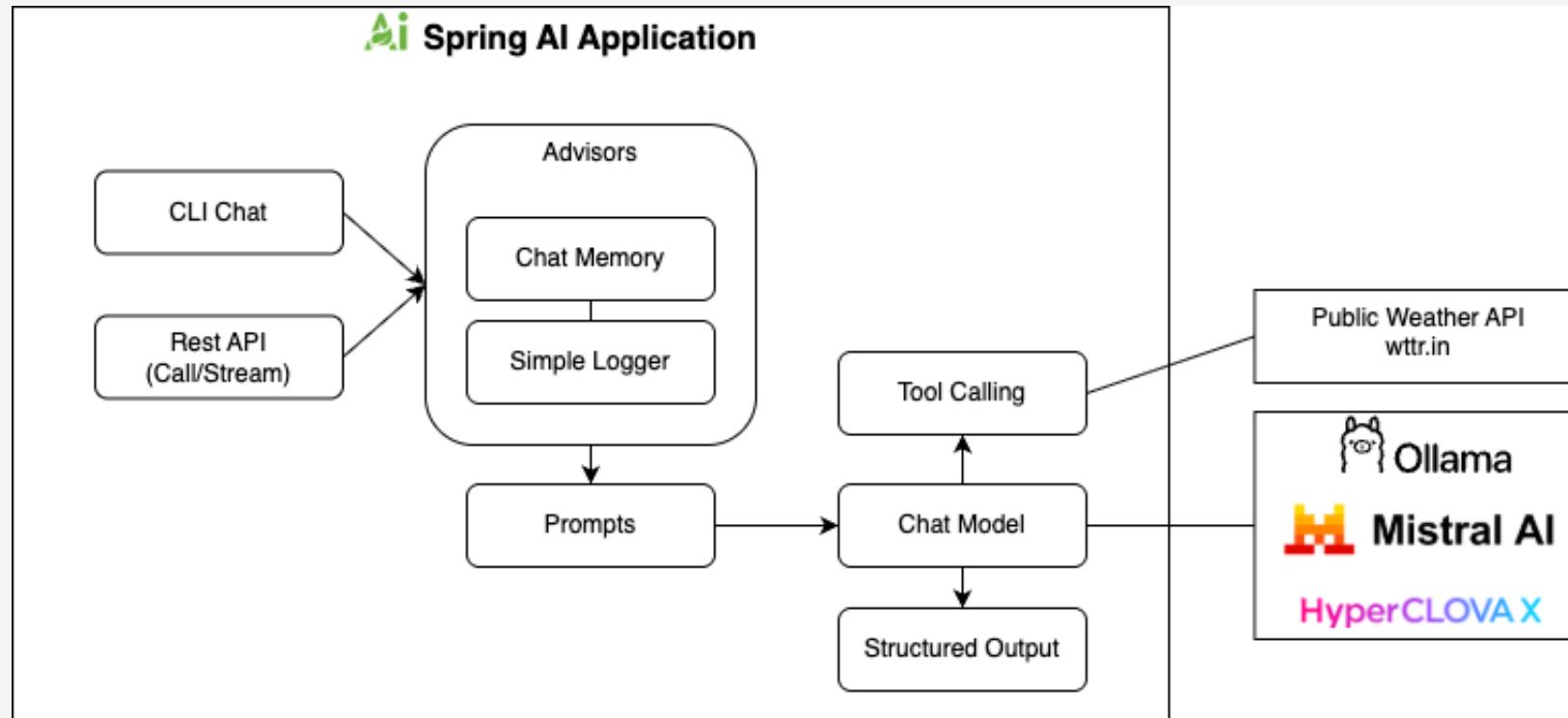
# Spring AI Tool Calling

## Chat 개발

# 개발 환경 구축 (Auto Configuration)

# 개발 환경 구축 (Auto Configuration)

- 전체 코드
  - <https://github.com/JM-Lab/spring-ai-fast-campus-course/tree/main/fast-campus-course-tool-chat>
- Spring AI Tool Chat 구성



# 개발 환경 구축 (Auto Configuration)

- pom.xml 의존성
  - fast-campus-course-chat 과 동일
- application.yaml 사용
  - tool calling 지원 모델 사용
    - ollama - mistral 모델
    - openai - gpt-4.1-nano 모델

## 1.0.0 버전에서 ollama 사용시 tool 지원 버그

- <https://github.com/spring-projects/spring-ai/pull/3372>
  - stream 출력시 NPE 에러 발생
  - 1.1.0 버전에서 패치
  - Open AI 모델 (Github 제공) 사용

```

app:
 cli:
 | enabled: true # CLI 모드 활성화 여부
 chat:
 | default-system-prompt: 한국어를 사용하는 tool 지원 AI입니다. # 기본 system prompt

logging:
 level:
 org:
 springframework:
 ai:
 chat:
 client:
 | advisor: DEBUG # SimpleLoggerAdvisor 등의 Advisor에서 DEBUG 로그 출력
 tool: DEBUG # Tool 사용 과정 DEBUG 로그 출력

spring:
 application:
 name: fast-campus-course-tool-chat # 어플리케이션 이름
 ai:
 model:
 | chat: openai # 여러 Chat 모델 사용시 auto-configurations 에서 사용할 모델 설정 필요 예: openai, ollama
 openai:
 api-key: ${OPENAI_API_KEY} # OpenAI 호출 API를 제공하는 Provider의 경우 아래 내용을 적절히 수정
 chat:
 options:
 model: openai/gpt-4.1-nano # 모델명 설정
 base-url: https://models.github.ai/inference # 설정하지 않으면 기본 OpenAI api 호출 주소를 사용
 completions-path: /chat/completions # OpenAI 기본값은 /v1/chat/completions, github 모델 사용시 아래와 같이 수정
 ollama:
 chat:
 options:
 model: mistral # Structured Output, Tool Calling 사용시 mistral (7B q4 기본 모델로 약 7GB 메모리 필요) 사용

```

# 외부 API 연동

## Tool Specification 개발

### (날씨 조회)

# 외부 API 연동 Tool Specification 개발 (날씨 조회)

- Tools
  - 날씨 연동 Tool 개발
  - wttr.in 서비스 사용
    - 인증 없는 무료 날씨 조회 서비스
      - curl wttr.in 만으로 현재 날씨와 향후 3일 간의 예보 제공
      - 다양한 포맷 출력 제공

```
@Service
public class Tools {

 private final WebClient webClient;

 public Tools(WebClient.Builder webClientBuilder) { this.webClient = webClientBuilder.build(); }

 @Tool(description = "지역 이름을 받아 현재 날씨를 조회합니다.")

 public String getWeather(@ToolParam(description = "지역 이름") String location) {
 return webClient.get().uri(UriBuilder uriBuilder -> uriBuilder.scheme("https")
 .host("wttr.in").path(location.replace(" ", "+"))
 .queryParam("lang", "ko")
 .queryParam("format", "현재+%l의+날씨는+%C+상태이며,+기온은+%t,+체감+기온은+%f,+풍속은+%W,+습도는+%h,+강수량은+%p입니다")
 .build())
 .retrieve().bodyToMono(String.class).log().block();
 }
}
```

# Tool 연동 및 Result Conversion 개발

# Tool 연동 및 Result Conversion 개발

- Tools

- 상세 날씨 연동 Tool 개발

- 일별(3일) 예보 정보 리스트 json 을 Result Conversion 사용해 받음

```
@Tool(description = "지역 이름을 받아 현재 3일간의 날씨와 천문 정보(달의 밝기, 달의 위상, 해/달의 뜨고 지는 시각)를 조회합니다.")
public WeatherResponse getWeatherDetails(@ToolParam(description = "지역 이름") String location) {
 return webClient.get().uri(UriBuilder uriBuilder -> uriBuilder.scheme("https")
 .host("wttr.in").path(location.replace(target: " ", replacement: "+"))
 .queryParam(name: "lang", ...values: "ko")
 .queryParam(name: "format", ...values: "j1") //json 출력으로 제공
 .build())
 .retrieve().bodyToMono(WeatherResponse.class).log().block();
}
```

```
// 일별 예보 (hourly 제외)
public record WeatherForecast(
 @Schema(description = "천문 정보(일출, 일몰 등)") List<Astronomy> astronomy,
 @Schema(description = "해당 날짜(yyyy-MM-dd)") String date,
 @Schema(description = "평균 기온(섭씨)") int avgtempC,
 @Schema(description = "평균 기온(화씨)") int avgtempF,
 @Schema(description = "최고 기온(섭씨)") int maxtempC,
 @Schema(description = "최고 기온(화씨)") int maxtempF,
 @Schema(description = "최저 기온(섭씨)") int mintempC,
 @Schema(description = "최저 기온(화씨)") int mintempF,
 @Schema(description = "일조 시간(시간 단위)") double sunHour,
 @Schema(description = "적설량(센티미터)") double totalSnow_cm,
 @Schema(description = "자외선 지수") int uvIndex
) {}
```

```
// 천문 정보
public record Astronomy(
 @Schema(description = "달 밝기(%)) int moon_illumination,
 @Schema(description = "달의 위상(예: Full Moon 등)") String moon_phase,
 @Schema(description = "달 뜨는 시각") String moonrise,
 @Schema(description = "달 지는 시각") String moonset,
 @Schema(description = "해 뜨는 시각") String sunrise,
 @Schema(description = "해 지는 시각") String sunset
) {}
```

```
public record WeatherResponse(
 @Schema(description = "일별(3일) 예보 정보 리스트")
 List<WeatherForecast> weather
) {}
```

# Tool 연동 및 Result Conversion 개발

- ToolChatService
  - Tool 연동을 위해 Tools Bean 을 주입 받아서 설정
  - **ToolCallingChatOptions** 사용해서 옵션 설정을 해야 함
    - internalToolExecutionEnabled 설정
      - 없거나 false면 Spring AI 가 Tool 을 관리하지 않음

```
@Service
public class ToolChatService {

 private final ChatClient chatClient;

 public ToolChatService(ChatClient.Builder chatClientBuilder, Advisor[] advisors,
 @Value("${app.chat.default-system-prompt:}") String systemPrompt, Tools tools) {
 // Tool 에서 제공한 내용을 기반으로 정보를 생성 해야 하므로 temperature를 0.2 로 설정
 this.chatClient = chatClientBuilder.defaultSystem(systemPrompt)
 .defaultTools(tools)
 .defaultOptions(ToolCallingChatOptions.builder()
 .internalToolExecutionEnabled(true)// 생략해도 true가 기본값
 .temperature(0.2)
 .build()).defaultAdvisors(advisors).build();
 }
}
```

# Tool 제어 (Return Direct, Exception Handling)

# Tool 제어 (Return Direct, Exception Handling)

- Tools
  - 날씨 연동 Tool 에 Return Direct 설정

```
@Tool(description = "지역 이름을 받아 현재 날씨를 조회합니다.", returnDirect = true)
public String getWeather(@ToolParam(description = "지역 이름") String location) {
 return webClient.get().uri(UriBuilder uriBuilder -> uriBuilder.scheme("https")
 .host("wttr.in").path(location.replace(target: " ", replacement: "+"))
 .queryParam(name: "lang", ...values: "ko")
 .queryParam(name: "format", ...values: "현재+%l의+날씨는+%C+상태이며,+기온은+%t,+체감기온은+%f,+풍속은+%W,+습도는+%h,+강수량은+%p입니다")
 .build())
 .retrieve().bodyToMono(String.class).log().block();
}
```

- ToolConfig

```
@Configuration
public class ToolConfig {

 @Bean
 public ToolCallingManager toolCallingManager() {
 return ToolCallingManager.builder().build();
 }

 @Bean
 public ToolExecutionExceptionProcessor toolExecutionExceptionProcessor() {
 // alwaysThrow를 true로 설정: 예외를 상위(chatClient 사용 Service)로 던짐
 // false(기본값): 예외를 AI 모델에 메시지로 전달
 return new DefaultToolExecutionExceptionProcessor(alwaysThrow: false);
 }
}
```

# CLI, Rest API 개발

# CLI, Rest API 개발

- FastCampusCourseToolChatConfig
  - FastCampusCourseChatConfig 동일
  - CLI 개발도 동일
- ToolChatService
  - ChatService 의 call, stream 과 동일하게 추가

```
private ChatClient.ChatClientRequestSpec buildChatClientRequestSpec(String conversationId, Prompt prompt) {
 return chatClient.prompt(prompt)
 .advisors(AdvisorSpec advisors -> advisors.param(ChatMemory.CONVERSATION_ID, conversationId));
}

public Flux<String> stream(String conversationId, Prompt prompt) {
 return buildChatClientRequestSpec(conversationId, prompt).stream().content();
}

public ChatResponse call(String conversationId, Prompt prompt) {
 return buildChatClientRequestSpec(conversationId, prompt).call().chatResponse();
}
```

# CLI, Rest API 개발

- ToolChatController

```
@RestController
@RequestMapping("/tool")
class ToolChatController {

 private final ToolChatService toolChatService;

 public ToolChatController(ToolChatService toolChatService) { this.toolChatService = toolChatService; }

 public record PromptBody(
 @NotEmpty @Schema(description = "대화 식별자", example = "conv-1234") String conversationId,
 @NotEmpty @Schema(description = "사용자 입력 프롬프트", example = "안녕하세요, 제주도 날씨 어때요?") String userPrompt,
 @Nullable @Schema(description = "시스템 프롬프트(선택)", example = "You are a helpful assistant.") String systemPrompt,
 @Nullable @Schema(description = "채팅 옵션(선택)", implementation = DefaultChatOptions.class) DefaultChatOptions chatOptions
) {}
}
```

```
@PostMapping(value = "/call", produces = MediaType.APPLICATION_JSON_VALUE)
ChatResponse call(@RequestBody @Valid ToolChatController.PromptBody promptBody) {
 Prompt.Builder promptBuilder = getPromptBuilder(promptBody);
 return this.toolChatService.call(promptBody.conversationId, promptBuilder.build());
}

private static Prompt.Builder getPromptBuilder(PromptBody promptBody) {
 List<Message> messages = new ArrayList<>();
 Optional.ofNullable(promptBody.systemPrompt).filter(Predicate.not(String::isBlank))
 .map(String systemPrompt -> SystemMessage.builder().text(systemPrompt).build()).ifPresent(messages::add);
 messages.add(UserMessage.builder().text(promptBody.userPrompt).build());
 Prompt.Builder promptBuilder = Prompt.builder().messages(messages);
 Optional.ofNullable(promptBody.chatOptions).ifPresent(promptBuilder::chatOptions);
 return promptBuilder;
}
```

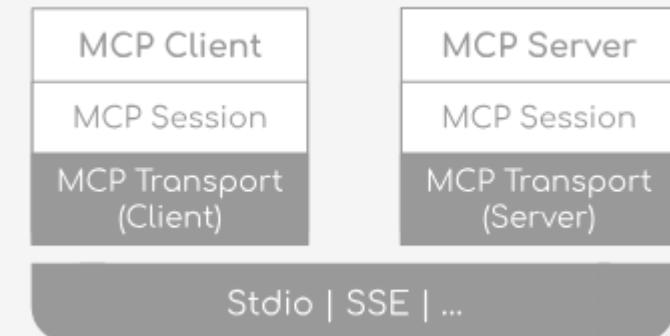
```
@PostMapping(value = "/stream", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
Flux<String> stream(@RequestBody @Valid ToolChatController.PromptBody promptBody) {
 Prompt.Builder promptBuilder = getPromptBuilder(promptBody);
 return this.toolChatService.stream(promptBody.conversationId, promptBuilder.build());
}
```

# Spring AI MCP

# MCP (Model Context Protocol)

# MCP (Model Context Protocol)

- AI 모델이 외부 도구 및 리소스와 구조적으로 상호작용할 수 있게 해주는 **표준 프로토콜**
  - **MCP는 클라이언트-서버 모델**
  - 3-layer architecture
    - 클라이언트/서버 계층
      - McpClient
        - 서버 연결·관리
      - McpServer
        - **Tools, 리소스, 다양한 기능 제공**
    - 세션 계층(Mcp Session)
      - 통신 상태 및 패턴 관리, 메시지 교환의 일관성 보장
    - 트랜스포트 계층(Mcp Transport)
      - JSON-RPC 메시지 직렬화/역직렬화
      - STDIO, Streamable HTTP 지원



# MCP 와 Tool Calling 관계

# MCP 와 Tool Calling 관계

| 항목         | Tool Calling              | MCP (Model Context Protocol)                |
|------------|---------------------------|---------------------------------------------|
| Tool 등록 방식 | @Tool, ChatClient 에 직접 등록 | MCP 서버에 등록, 클라이언트가 조회                       |
| 관리 구조      | 앱 단위, 고정                  | 중앙화, 동적 업데이트 가능                             |
| Tool 변경 방법 | 툴 코드 변경 시 재시작 필요          | 신규 추가나 변경시 MCP 서버 재시작, 실시간 제공 Tool 추가/삭제 지원 |
| 아키텍처 형태    | AI 애플리케이션 내 Tool 포함       | MCP 클라이언트-서버 분리                             |
| 주 사용 시나리오  | 간단 PoC, 내부 Tool 구현        | MCP 서버로 Tool 서비스 공유, Tool 생태계 구축에 적합        |

# Spring AI MCP 구성

# Spring AI MCP 구성

- MCP Java SDK를 기반으로 Spring 애플리케이션에서 MCP를 쉽게 사용할 수 있도록 지원
  - Spring Boot 스타터를 제공하여 MCP 클라이언트와 서버를 간편하게 설정
    - spring-ai-starter-mcp-client: 클라이언트 기능 제공
    - spring-ai-starter-mcp-server: 서버 기능 제공
  - 설정으로 Client/Server 구성

```
spring:
 ai:
 mcp:
 client:
 enabled: true
 name: my-mcp-client
 version: 1.0.0
 request-timeout: 30s
 type: SYNC # or ASYNC for reactive applications
 sse:
 connections:
 server1:
 url: http://localhost:8080
 server2:
 url: http://otherserver:8081
 stdio:
 root-change-notification: false
 connections:
 server1:
 command: /path/to/server
 args:
 - --port=8080
 - --mode=production
 env:
 API_KEY: your-api-key
 DEBUG: "true"
```

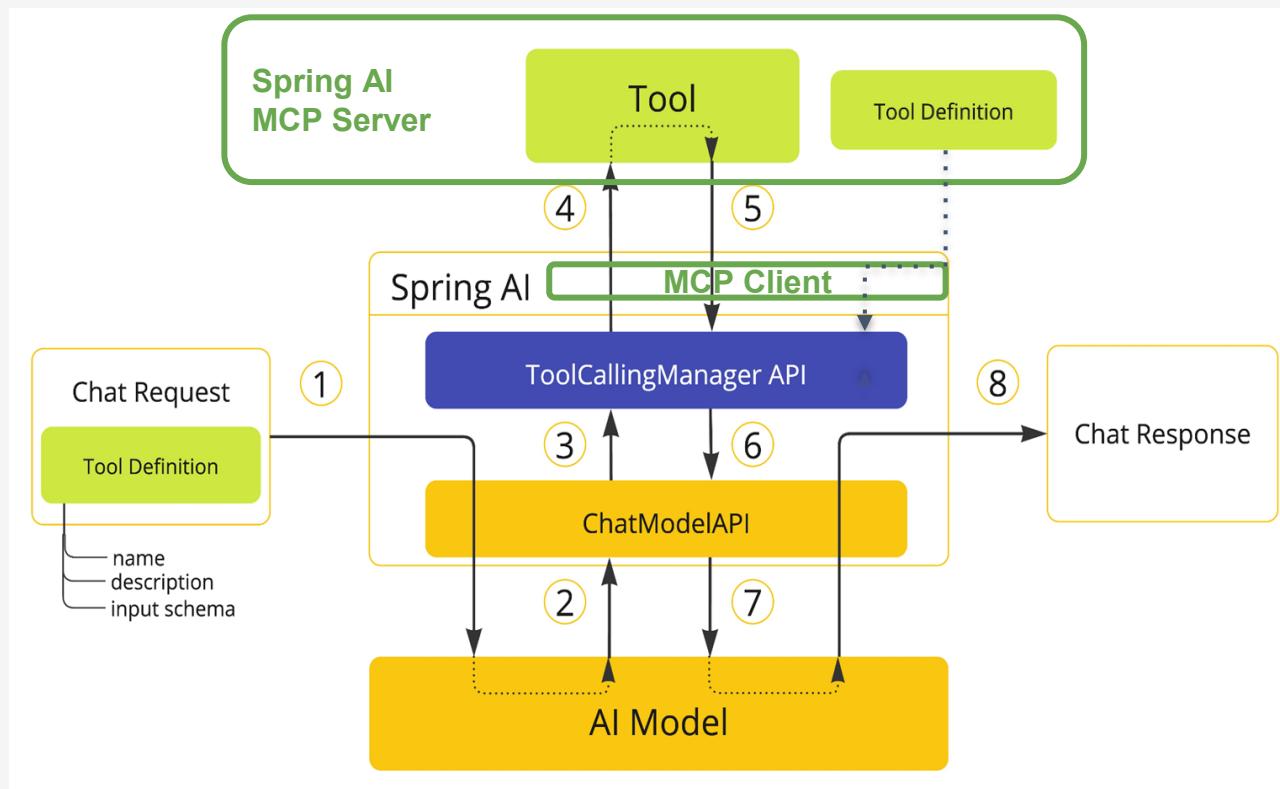
```
Using spring-ai-starter-mcp-server
spring:
 ai:
 mcp:
 server:
 name: stdio-mcp-server
 version: 1.0.0
 type: SYNC

Using spring-ai-starter-mcp-server-webflux
spring:
 ai:
 mcp:
 server:
 name: webflux-mcp-server
 version: 1.0.0
 type: ASYNC # Recommended for reactive applications
 instructions: "This reactive server provides weather information tools and resources"
 sse-message-endpoint: /mcp/messages
 capabilities:
 tool: true
 resource: true
 prompt: true
 completion: true
```

# Spring AI MCP Process

# Spring AI MCP Process

- Tool 정의는 MCP Client - Server 간에 통신으로 ToolCallingManager에 자동 등록 간접
- Tool 요청 시 MCP Client를 통해 MCP 서버의 Tool 실행 결과를 받음



# **Spring AI MCP**

# **Client & Server 구현**

# Spring AI MCP Client & Server 구현

- MCP Server 구현
  - 의존성 추가
    - 제공 tool 이 Non-blocking I/O 로 구현되고 서비스

```
<dependency>
 <groupId>org.springframework.ai</groupId>
 <artifactId>spring-ai-starter-mcp-server-webflux</artifactId>
</dependency>
```

- 제공 tool 이 Blocking I/O 로 구현되고 서비스

```
<dependency>
 <groupId>org.springframework.ai</groupId>
 <artifactId>spring-ai-starter-mcp-server-webmvc</artifactId>
</dependency>
```

- 출력을 STUDIO 만 사용할 경우
  - Local 에서 CLI 기능을 Tool 로 제공 할때
  - 보안 관점에서 가장 안전함
    - Client 가 local 에 Child process 로 Server를 실행

```
<dependency>
 <groupId>org.springframework.ai</groupId>
 <artifactId>spring-ai-starter-mcp-server</artifactId>
</dependency>
```

# Spring AI MCP Client & Server 구현

- MCP Server 구현
  - Spring Configuration

```
spring:
 ai:
 mcp:
 server:
 name: webflux-mcp-server
 version: 1.0.0
 type: ASYNC # Recommended for reactive applications
 instructions: "This reactive server provides weather information tools and resources"
 sse-message-endpoint: /mcp/messages
 capabilities:
 tool: true
 resource: true
 prompt: true
 completion: true
```

- Spring Bean 내에 @Tool 생성
- ToolCallbackProvider Bean 등록

```
@Service
public class WeatherService {

 @Tool(description = "Get weather information by city name")
 public String getWeather(String cityName) {
 // Implementation
 }

 @SpringBootApplication
 public class McpServerApplication {

 private static final Logger logger = LoggerFactory.getLogger(McpServerApplication.class);

 public static void main(String[] args) {
 SpringApplication.run(McpServerApplication.class, args);
 }

 @Bean
 public ToolCallbackProvider weatherTools(WeatherService weatherService) {
 return MethodToolCallbackProvider.builder().toolObjects(weatherService).build();
 }
 }
}
```

# Spring AI MCP Client & Server 구현

- MCP Client 구현
  - 의존성 추가
    - Blocking I/O 사용 (spring-boot-starter-web 사용)

```
<dependency>
 <groupId>org.springframework.ai</groupId>
 <artifactId>spring-ai-starter-mcp-client</artifactId>
</dependency>
```

- Non-blocking I/O 사용 (spring-boot-starter-webflux 사용)

```
<dependency>
 <groupId>org.springframework.ai</groupId>
 <artifactId>spring-ai-starter-mcp-client-webflux</artifactId>
</dependency>
```

# Spring AI MCP Client & Server 구현

- MCP Client 구현
  - Spring Configuration

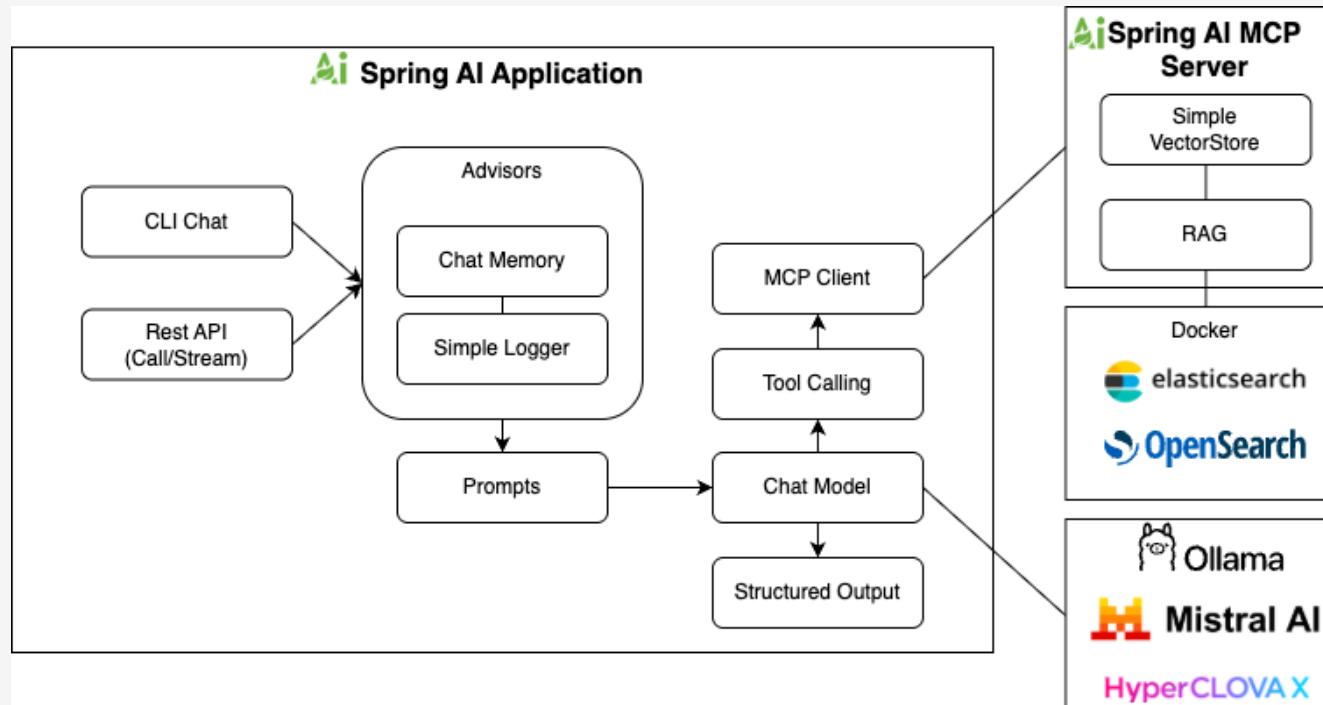
```
spring:
 ai:
 mcp:
 client:
 enabled: true
 name: my-mcp-client
 version: 1.0.0
 request-timeout: 30s
 type: SYNC # or ASYNC for reactive applications
 sse:
 connections:
 server1:
 url: http://localhost:8080
 server2:
 url: http://otherserver:8081
 stdio:
 root-change-notification: false
 connections:
 server1:
 command: /path/to/server
 args:
 - --port=8080
 - --mode=production
 env:
 API_KEY: your-api-key
 DEBUG: "true"
```

# Spring AI MCP 개발

# 개발 환경 구축 (Auto Configuration)

# 개발 환경 구축 (Auto Configuration)

- 전체 코드
  - <https://github.com/JM-Lab/spring-ai-fast-campus-course/tree/main/fast-campus-course-mcp-chat>
  - <https://github.com/JM-Lab/spring-ai-fast-campus-course/tree/main/fast-campus-course-mcp-server>
- Spring AI MCP Chat 구성



# 개발 환경 구축 (Auto Configuration)

- pom.xml 의존성
  - fast-campus-course-mcp-chat
    - fast-campus-course-tool-chat 에 mcp client 의존성에 추가

```
<dependency>
 <groupId>org.springframework.ai</groupId>
 <artifactId>spring-ai-starter-mcp-client</artifactId>
</dependency>
```

- application.yaml 사용
  - fast-campus-course-mcp-chat
    - fast-campus-course-tool-chat 과 동일
      - MCP Client 정보만 추가
    - tool calling 지원 모델 사용
      - ollama - mistral 모델
      - openai - gpt-4.1-nano 모델

```
spring:
 application:
 name: fast-campus-course-mcp-chat # 어플리케이션 이름
 ai:
 model:
 chat: openai # 여러 Chat 모델 사용시 auto-configurations에서 사용할 모델 설정 필요 예: openai, ollama
 openai:
 api-key: ${OPENAI_API_KEY} # OpenAI 호출 API를 제공하는 Provider의 경우 아래 내용을 적절히 수정
 chat:
 options:
 model: openai/gpt-4.1-nano # 모델명 설정
 base-url: https://models.github.ai/inference # 설정하지 않으면 기본 OpenAI api 호출 주소를 사용
 completions-path: /chat/completions # OpenAI 기본값은 /v1/chat/completions, github 모델 사용시 아래와 같이 수정
 ollama:
 chat:
 options:
 model: mistral # Structured Output, Tool Calling 사용시 mistral (7B q4 기본 모델로 약 7GB 메모리 필요) 사용
```

# 개발 환경 구축 (Auto Configuration)

- pom.xml 의존성
  - fast-campus-course-mcp-server
    - fast-campus-course-rag-chat 의 의존성에서 mcp-server-webmvc 사용으로 수정
      - starter-webmvc 를 포함하고 있음
      - 기존 Rag Chat 을 이용해서 RAG 기능 Tool 을 개발

```
<dependency>
 <groupId>org.springframework.ai</groupId>
 <artifactId>spring-ai-starter-mcp-server-webmvc</artifactId>
</dependency>
```

```
<dependency>
 <groupId>org.springframework.ai</groupId>
 <artifactId>spring-ai-advisors-vector-store</artifactId>
</dependency>

<dependency>
 <groupId>org.springframework.ai</groupId>
 <artifactId>spring-ai-tika-document-reader</artifactId>
</dependency>

<dependency>
 <groupId>org.springframework.ai</groupId>
 <artifactId>spring-ai-rag</artifactId>
</dependency>

<dependency>
 <groupId>org.springframework.ai</groupId>
 <artifactId>spring-ai-starter-model-llama</artifactId>
</dependency>
```

# 개발 환경 구축 (Auto Configuration)

- application.yaml 사용
  - fast-campus-course-mcp-server
    - MCP Server 정보만 추가

```
server:
 port: 8081
spring:
 application:
 name: fast-campus-course-mcp-server # 어플리케이션 이름
 ai:
 mcp:
 server:
 name: ${spring.application.name}
 version: 0.1.0
 type: SYNC
 instructions: "지역을 받아 날씨 정보를 제공하고 Spring AI 강의 정보를 RAG 를 사용해 제공하는 MCP 서비스"
 sse-message-endpoint: /mcp/messages
 model:
 chat: ollama # 여러 Chat 모델 사용시 auto-configurations 에서 사용할 모델 설정 필요 예: openai, ollama
 embedding: ollama # 여러 Embedding 모델 사용시 auto-configurations에서 사용할 모델 지정 (예: openai, ollama)
 ollama:
 init:
 pull-model-strategy: when_missing # when_missing, always, never 설정
 chat:
 options:
 model: hf.co/rippertnt/HyperCLOVAX-SEED-Text-Instruct-1.5B-Q4_K_M-GGUF # Structured Output, Tool Calling 사용시 mistral (7B q4 기본 모델로 약 76B 메모리 필요) 사용
 embedding:
 options:
 model: bge-m3 # embedding model, 기본 모델 mxbai-embed-large는 영어 전용, bge-m3로 변경
```

```
app:
 chat:
 default-system-prompt: 한국어를 사용하는 tool 지원 AI입니다. # 기본 system prompt
 etl:
 pipeline:
 init: true # 초기 데이터 로딩 여부
 vectorstore:
 in-memory.enabled: true # in-memory SimpleVectorStore 사용 여부
 rag:
 documents-location-pattern: classpath:fastcampus-springai.pdf # RAG Data 위치
```

# MCP 클라이언트 개발 (외부 MCP 서버 연동 날씨 조회)

# MCP 클라이언트 개발 (외부 MCP 서버 연동 날씨 조회)

- 사전 준비
  - mcp\_weather\_server 설치
    - pip install mcp\_weather\_server
- application.yaml
  - stdio 사용 mcp\_weather\_server 설정

```
spring:
 application:
 name: fast-campus-course-mcp-chat # 어플리케이션 이름
 ai:
 mcp:
 client:
 enabled: true # MCP Client 활성화
 type: SYNC # MCP Client 타입
 stdio:
 connections:
 weather-mcp-server:
 command: /Users/jm/myenv/bin/python # MCP 서버 실행 Python 경로
 args:
 - -m
 - mcp_weather_server # MCP 서버 실행 인자
```

# MCP 클라이언트 개발 (외부 MCP 서버 연동 날씨 조회)

- ToolConfig
  - fast-campus-course-tool-chat 과 동일 (Tools 가 없음)
- McpChatService

```

@Service
public class McpChatService {

 private static final Logger log = LoggerFactory.getLogger(McpChatService.class);

 private final ChatClient chatClient;

 public McpChatService(ChatClient.Builder chatClientBuilder, Advisor[] advisors,
 @Value("${app.chat.default-system-prompt}") String systemPrompt,
 // MCP Client 가 MCP 서버에서 자동으로 가져온 Tool 를 콜백 제공자들
 ToolCallbackProvider[] toolCallbackProviders) {
 // Tool 에서 제공한 내용을 기반으로 정보를 생성 해야 하므로 temperature를 0.2 로 설정
 ToolCallback[] toolCallbacks = Arrays.stream(toolCallbackProviders)
 .flatMap(ToolCallbackProvider provider -> Arrays.stream(provider.getToolCallbacks()))
 .peek(ToolCallback callback -> {
 ToolDefinition def = callback.getToolDefinition();
 log.info("ToolDefinition - name: {}, description: {}, schema: {}", def.name(), def.description(),
 def.inputSchema());
 })
 .toArray(ToolCallback[]::new);
 this.chatClient = chatClientBuilder.defaultSystem(systemPrompt)
 .defaultOptions(ToolCallingChatOptions.builder()
 .internalToolExecutionEnabled(true)// 생략해도 true가 기본값
 .temperature(0.2).build())
 .defaultAdvisors(advisors)
 .defaultToolCallbacks(toolCallbacks).build();
 }

 private ChatClient.ChatClientRequestSpec buildChatClientRequestSpec(String conversationId, Prompt prompt) {
 return chatClient.prompt(prompt)
 .advisors(AdvisorSpec advisors -> advisors.param(ChatMemory.CONVERSATION_ID, conversationId));
 }

 public Flux<String> stream(String conversationId, Prompt prompt) {
 return buildChatClientRequestSpec(conversationId, prompt).stream().content();
 }

 public ChatResponse call(String conversationId, Prompt prompt) {
 return buildChatClientRequestSpec(conversationId, prompt).call().chatResponse();
 }
}

```

# 로컬 MCP 서버 개발 (내부 Rest API MCP 서버로 제공)

# 로컬 MCP 서버 개발 (내부 Rest API MCP 서버로 제공)

- application.yaml
  - mcp 서버 활성화

```
server:
 port: 8081
spring:
 application:
 name: fast-campus-course-mcp-server # 어플리케이션 이름
 ai:
 mcp:
 server:
 name: ${spring.application.name}
 version: 0.1.0
 type: SYNC
 instructions: "지역을 받아 날씨 정보를 제공하고 Spring AI 강의 정보를 RAG 를 사용해 제공하는 MCP 서버"
 sse-message-endpoint: /mcp/messages
```

- FastCampusCourseMcpServerConfig

```
@Configuration
public class FastCampusCourseMcpServerConfig {

 // 디버깅 및 모니터링에 유용하며, 기본 로깅 포맷과 커스터마이징 기능을 지원
 // 기본 order는 0 이므로 1을 주면 뒤에 실행 -1 을 주면 먼저실행
 @Bean
 public SimpleLoggerAdvisor simpleLoggerAdvisor() {
 return SimpleLoggerAdvisor.builder().build();
 }

 @Bean
 public ToolCallbackProvider toolCallbackProvider(Tools tools) {
 return MethodToolCallbackProvider.builder().toolObjects(tools).build();
 }

}
```

# 로컬 MCP 서버 개발 (내부 Rest API MCP 서버로 제공)

- RagConfig
  - fast-campus-course-rag-chat에서 System.out 부분 수정

```
@Bean
public DocumentWriter logDocumentWriter(ObjectMapper objectMapper) {
 return documents -> {
 log.info("===== save chunks size {} =====", documents.size());
 try {
 log.info(objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(documents));
 } catch (JsonProcessingException e) {
 throw new RuntimeException(e);
 }
 log.info("=====");
 };
}
```

- LengthTextSplitter
  - fast-campus-course-rag-chat 그대로 사용
- RagChatService
  - fast-campus-course-rag-chat에서 stream 제거하고 사용
    - RAG api 처리 최종 결과만 사용

# 로컬 MCP 서버 개발 (내부 Rest API MCP 서버로 제공)

- Tools

- 기존 RAG Chat의 기능을 Agent-as-a-Tool 방식으로 제공
  - RequestBody에 여러 정보가 들어가는 Rest API는 바로 Tool 적용이 안됨
    - Tool은 각 ToolParam 별로 정의가 필요 함

```
@Service
public class Tools {

 Logger log = LoggerFactory.getLogger(Tools.class);

 private final WebClient webClient;

 private final RagChatService ragChatService;

 public Tools(WebClient.Builder webClientBuilder, RagChatService ragChatService) {
 this.webClient = webClientBuilder.build();
 this.ragChatService = ragChatService;
 }

 @Tool(description = "Spring AI 강의에 대해 RAG 기반 답변을 제공합니다.")
 public String ragTool(@ToolParam(description = "Spring AI 강의에 대한 질문") String userPrompt) {
 log.info("ragTool UserPrompt = {}", userPrompt);
 return this.ragChatService.call(conversationId: "mcp",
 Prompt.builder().messages(UserMessage.builder().text(userPrompt).build()).build(),
 filterExpressionAsOpt: Optional.empty())
 .getResult().getOutput().getText();
 }
}
```

- 기존 Tool인 getWeather, getWeatherDetails 그대로 사용 가능

# Spring AI Chat 에 MCP 연동 개발

# Spring AI Chat 에 MCP 연동 개발

- fast-campus-course-mcp-server 실행
  - 8081 포트 사용
- fast-campus-course-mcp-chat 에서 ast-campus-course-mcp-server 연동
  - fast-campus-course-mcp-chat 의 application.yaml

```
spring:
 application:
 name: fast-campus-course-mcp-chat # 어플리케이션 이름
 ai:
 mcp:
 client:
 enabled: true # MCP Client 활성화
 type: SYNC # MCP Client 타입
 sse:
 connections:
 local-mcp-server:
 url: http://localhost:8081
```

# Spring AI Chat 에 MCP 연동 개발

- fast-campus-course-mcp-server 빌드
  - ./mvnw clean package -DskipTests
- fast-campus-course-mcp-chat 에서 fast-campus-course-mcp-server stdio로 연동
  - fast-campus-course-mcp-chat 의 application.yaml

```
spring:
 application:
 name: fast-campus-course-mcp-chat # 어플리케이션 이름
 ai:
 mcp:
 client:
 enabled: true # MCP Client 활성화
 type: SYNC # MCP Client 타입
 stdio:
 connections:
 local-mcp-server:
 command: /usr/bin/java
 args:
 - -jar
 - /Users/jm/git/spring-ai-fast-campus-course/fast-campus-course-mcp-server/target/fast-campus-course-mcp-server-0.0.1-SNAPSHOT.jar
 - --spring.main.banner-mode=off # Spring Boot 배너 비활성화
 - --logging.pattern.console= # 콘솔 로그 패턴 비우기
 - --spring.ai.mcp.server.stdio=true # STDIO 기반 MCP 서버 활성화
 - --spring.main.web-application-type=None # 웹 서버 기능 생략 (비웹 모드)
```

# CLI, Rest API 개발

# CLI, Rest API 개발

- fast-campus-course-mcp-chat CLI 개발
  - FastCampusCourseMcpChatConfig
    - fast-campus-course-tool-chat 과 동일

```
@ConditionalOnProperty(prefix = "app.cli", name = "enabled", havingValue = "true")
@Bean
public CommandLineRunner cli(@Value("${spring.application.name}") String applicationName,
 McpChatService mcpChatService) {
 return String[] args -> {

 LoggerContext context = (LoggerContext) LoggerFactory.getILoggerFactory();
 context.getLogger("ROOT").detachAppender(name: "CONSOLE");

 System.out.println("\n" + applicationName + " CLI bot");
 try (Scanner scanner = new Scanner(System.in)) {
 while (true) {
 System.out.print("\nUSER: ");
 String userMessage = scanner.nextLine();
 mcpChatService.stream(conversationId: "cli", Prompt.builder().content(userMessage).build())
 .doFirst(() -> System.out.print("\nASSISTANT: "))
 .doOnNext(System.out::print)
 .doOnComplete(System.out::println)
 .blockLast();
 }
 };
 };
}
```

# CLI, Rest API 개발

- fast-campus-course-mcp-chat Rest API 개발
  - McpChatController
    - fast-campus-course-tool-chat 과 동일

```
@RestController
@RequestMapping(value = "/mcp")
class McpChatController {

 private final McpChatService mcpChatService;

 public McpChatController(McpChatService mcpChatService) { this.mcpChatService = mcpChatService; }

 public record PromptBody(
 @NotEmpty @Schema(description = "대화 식별자", example = "conv-1234") String conversationId,
 @NotEmpty @Schema(description = "사용자 입력 프롬프트", example = "안녕하세요, 제주도 날씨 어때요?") String userPrompt,
 @Nullable @Schema(description = "시스템 프롬프트(선택)", example = "You are a helpful assistant.") String systemPrompt,
 @Nullable @Schema(description = "채팅 음성(선택)", implementation = DefaultChatOptions.class) DefaultChatOptions chatOptions
) {}
}
```

```
@PostMapping(value = "/call", produces = MediaType.APPLICATION_JSON_VALUE)
ChatResponse call(@RequestBody @Valid PromptBody promptBody) {
 Prompt.Builder promptBuilder = getPromptBuilder(promptBody);
 return this.mcpChatService.call(promptBody.conversationId, promptBuilder.build());
}

private static Prompt.Builder getPromptBuilder(PromptBody promptBody) {
 List<Message> messages = new ArrayList<>();
 Optional.ofNullable(promptBody.systemPrompt).filter(Predicate.not(String::isBlank))
 .map(String systemPrompt -> SystemMessage.builder().text(systemPrompt).build()).ifPresent(messages::add);
 messages.add(UserMessage.builder().text(promptBody.userPrompt).build());
 Prompt.Builder promptBuilder = Prompt.builder().messages(messages);
 Optional.ofNullable(promptBody.chatOptions).ifPresent(promptBuilder::chatOptions);
 return promptBuilder;
}
```

```
@PostMapping(value = "/stream", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
Flux<String> stream(@RequestBody @Valid PromptBody promptBody) {
 Prompt.Builder promptBuilder = getPromptBuilder(promptBody);
 return this.mcpChatService.stream(promptBody.conversationId, promptBuilder.build());
}
```

# 부록

# Multimodality API

# Multimodality API

- 멀티모달리티란?
  - 인간이 텍스트, 이미지, 소리 등 여러 형태의 정보를 동시에 처리하며 배우는 방식과 유사하게, AI 모델이 다양한 데이터 입력을 함께 이해하고 처리하는 능력
    - 최신 AI 모델은 음성, 비전 등의 Multimodality 를 지원
      - OpenAI GPT-4o, Google Vertex AI Gemini 1.5, Anthropic Claude 3
      - 오픈소스 모델 - Llama 3.2, LLaVA, HyperCLOVA X SEED 3B 등
- Spring AI Message API
  - UserMessage의 media 필드에MimeType, Media URI 지정
  - **답변은 Text 만 가능, 이미지 생성 같은 것은 Image Model 을 직접 사용해야 함**

```
String response = ChatClient.create(chatModel).prompt()
 .user(u -> u.text("Explain what do you see on this picture?")
 .media(MediaTypeUtils.IMAGE_PNG, new ClassPathResource("/multimodal.test.png")))
 .call()
 .content();
```

# Observability

# Observability

- Spring 생태계의 강력한 Observability(관측 가능성) 기능을 기반으로 AI 애플리케이션의 내부 동작을 추적하고 분석
  - AI 모델과의 상호작용, 성능, 비용 등을 쉽게 파악
- 주요 특징
  - Metrics & Tracing
    - 핵심 컴포넌트의 성능 지표(Metrics)와 분산 추적(Tracing) 정보 자동 수집
- 관측 활성화 방법
  - Spring Boot Actuator, Micrometer, OpenTelemetry, Zipkin 등 연동
  - application.yml/properties에서 로그 및 트레이스 활성화 속성 설정
- 활용 사례
  - Prometheus, Grafana 등으로 메트릭/트레이스 시각화
  - 토큰 사용량, 모델 성능, 쿼리 결과 등 대시보드 구축 및 문제 추적

# AI Model Evaluation

# AI Model Evaluation

- AI Model Evaluation의 필요성
  - AI 애플리케이션 테스트 시, 생성된 응답이 사실과 부합하는지(환각 방지) 자동화 필요
    - Spring AI는 AI 모델을 사용한 평가 방법을 지원 함
    - 별도의 AI 모델을 사용할 수 있으며, 응답 생성 모델과 달라도 됨
- AI Evaluator API
  - Evaluator Interface를 구현하여 각종 평가 방법을 구현할 수 있음
  - EvaluationRequest
    - userText
      - 사용자의 원본 질문
    - dataList
      - 문맥(Context) 데이터
    - responseContent
      - 평가 대상인 AI 모델의 응답 내용

```
@FunctionalInterface
public interface Evaluator {
 EvaluationResponse evaluate(EvaluationRequest evaluationRequest);
}

public class EvaluationRequest {

 private final String userText;
 private final List<Content> dataList;
 private final String responseContent;

 public EvaluationRequest(String userText, List<Content> dataList, String responseContent) {
 this.userText = userText;
 this.dataList = dataList;
 this.responseContent = responseContent;
 }
 ...
}
```

# AI Model Evaluation

- Spring AI 내장 Evaluator
  - RelevancyEvaluator
    - RAG(검색 기반 생성) 플로우에서, AI 응답이 사용자 질문 및 문맥과 관련 있는지 평가
- Prompt Template 과 Test 코드 예

Your task is to evaluate if the response for the query is in line with the context information provided.

You have two options to answer. Either YES or NO.

Answer YES, if the response for the query is in line with context information otherwise NO.

Query:

{query}

Response:

{response}

Context:

{context}

Answer:

```

 @Test
 void evaluateRelevancy() {
 String question = "Where does the adventure of Anacletus and Birba take place?";

 RetrievalAugmentationAdvisor ragAdvisor = RetrievalAugmentationAdvisor.builder()
 .documentRetriever(VectorStoreDocumentRetriever.builder()
 .vectorStore(pgVectorStore)
 .build())
 .build();

 ChatResponse chatResponse = ChatClient.builder(chatModel).build()
 .prompt(question)
 .advisors(ragAdvisor)
 .call()
 .chatResponse();

 EvaluationRequest evaluationRequest = new EvaluationRequest(
 // The original user question
 question,
 // The retrieved context from the RAG flow
 chatResponse.getMetadata().get(RetrievalAugmentationAdvisor.DOCUMENT_CONTEXT),
 // The AI model's response
 chatResponse.getResult().getOutput().getText()
);

 RelevancyEvaluator evaluator = new RelevancyEvaluator(ChatClient.builder(chatModel));

 EvaluationResponse evaluationResponse = evaluator.evaluate(evaluationRequest);

 assertThat(evaluationResponse.isPass()).isTrue();
 }

```

# AI Model Evaluation

- Spring AI 내장 Evaluator
  - FactCheckingEvaluator
    - AI 응답(Claim)이 문서(Context)에 의해 논리적으로 뒷받침되는지 평가
- Prompt Template 과 Test 코드 예
  - GPT-4와 같은 대형 모델 대신, 비용 효율적인 사실 확인 전용 모델 Minicheck 사용

Document: {document}  
Claim: {claim}

```
@Test
void testFactChecking() {
 // Set up the Ollama API
 OllamaApi ollamaApi = new OllamaApi("http://localhost:11434");

 ChatModel chatModel = new OllamaChatModel(ollamaApi,
 OllamaOptions.builder().model(BESPOKE_MINICHECK).numPredict(2).temperature(0.0d).build())

 // Create the FactCheckingEvaluator
 var factCheckingEvaluator = new FactCheckingEvaluator(ChatClient.builder(chatModel));

 // Example context and claim
 String context = "The Earth is the third planet from the Sun and the only astronomical object known to harbor life.";
 String claim = "The Earth is the fourth planet from the Sun."

 // Create an EvaluationRequest
 EvaluationRequest evaluationRequest = new EvaluationRequest(context, Collections.emptyList(), claim);

 // Perform the evaluation
 EvaluationResponse evaluationResponse = factCheckingEvaluator.evaluate(evaluationRequest);

 assertFalse(evaluationResponse.isPass(), "The claim should not be supported by the context");
}
```

JAVA

# Spring AI Examples

# Spring AI Examples

- 공식 Spring AI 예제
  - <https://github.com/spring-projects/spring-ai-examples>
- 커뮤니티 예제 및 Spring AI 관련 자료
  - <https://github.com/spring-ai-community/awesome-spring-ai>
    - 강사 개발 Open Source
      - <https://github.com/JM-Lab/spring-ai-local-cli-chatbot>
      - <https://github.com/JM-Lab/spring-ai-playground>

