



ICT Training Center

Il tuo partner per la Formazione e la Trasformazione digitale della tua azienda



SPRING AI

GENERATIVE ARTIFICIAL INTELLIGENCE CON JAVA

Simone Scannapieco

Corso base per Venis S.p.A, Venezia, Italia

Settembre 2025

SPRING AI: INTRODUZIONE

- ➔ Framework per integrare l'Intelligenza Artificiale nelle applicazioni Spring Boot
- ➔ Astrazione unificata per diversi provider AI (OpenAI, Azure OpenAI, AWS Bedrock, Google Vertex AI)
- ➔ Integrazione nativa con l'ecosistema Spring esistente
- ➔ Supporto per Vector Stores, Embedding e Retrieval-Augmented Generation (RAG)

- ➔ **Problema:** ogni provider AI ha API e modelli diversi
- ➔ **Soluzione Spring AI:** interfaccia unificata e portabile
- ➔ **Principi Spring:**
 - ➔ Dependency Injection
 - ➔ Auto-configuration
 - ➔ Testabilità
 - ➔ Observability
- ➔ **Integrazione** con Spring Boot, Spring Security, Spring Data

ARCHITETTURA DI SPRING AI

- ➔ **ChatClient**: interfaccia principale per conversazioni AI
- ➔ **Fluent API** per costruire richieste complesse
- ➔ Supporto per **streaming**, **function calling** e **multimodal**

```
@RestController
public class ChatController {

    private final ChatClient chatClient;

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

    @PostMapping("/ai/chat")
    public String chat(@RequestBody String userInput) {
        return chatClient.prompt()
            .user(userInput)
            .call()
            .content();
    }
}
```


- ➔ **Vector Store:** database ottimizzati per similarity search
- ➔ Supporto per **Redis, Chroma, Weaviate, Pinecone**
- ➔ Integrazione con **Spring Data** patterns

```
@Configuration
public class VectorStoreConfig {

    @Bean
    public VectorStore vectorStore(EmbeddingClient embeddingClient) {
        return new SimpleVectorStore(embeddingClient);
    }

    @Bean
    public DocumentRetriever documentRetriever(VectorStore vectorStore) {
        return VectorStoreDocumentRetriever.builder()
            .vectorStore(vectorStore)
            .topK(5)
            .similarityThreshold(0.7)
            .build();
    }
}
```

- ➔ **EmbeddingClient**: interfaccia per modelli di embedding
- ➔ Trasforma testo in **rappresentazioni vettoriali** per similarity search
- ➔ Supporto per **OpenAI, Ollama, Azure OpenAI**

```
@Service
public class DocumentService {

    private final EmbeddingClient embeddingClient;
    private final VectorStore vectorStore;

    public void addDocument(String content, Map<String, Object> metadata) {
        // Crea embedding del contenuto
        List<Float> embedding = embeddingClient.embed(content);

        // Salva nel vector store
        Document document = new Document(content, metadata);
        document.setEmbedding(embedding);

        vectorStore.add(List.of(document));
    }
}
```

VANTAGGI DI SPRING AI

- ➡ **Portabilità tra provider:** cambio di provider AI senza modificare codice business
- ➡ **Integrazione Spring nativa:** sfrutta tutte le funzionalità dell'ecosistema
- ➡ **Auto-configuration:** configurazione automatica basata su dependencies
- ➡ **Testabilità migliorata:** mock e test integration semplificati
- ➡ **Observability:** metriche e monitoring integrati
- ➡ **Gestione errori:** retry policies e fallback automatici

```
// Configurazione per OpenAI
@Configuration
public class OpenAIConfig {
    @Bean
    public ChatClient chatClient() {
        return ChatClient.builder()
            .chatModel(new OpenAiChatModel(apiKey))
            .build();
    }
}

// Configurazione per Azure OpenAI
@Configuration
public class AzureConfig {
    @Bean
    public ChatClient chatClient() {
        return ChatClient.builder()
            .chatModel(new AzureOpenAiChatModel(endpoint, apiKey))
            .build();
    }
}
```

- ➔ `application.properties` definisce tutto
- ➔ `@ConditionalOn*` annotations per configurazione condizionale

```
# OpenAI Configuration
spring.ai.openai.api-key=${OPENAI_API_KEY}
spring.ai.openai.chat.model=gpt-4
spring.ai.openai.chat.temperature=0.7

# Vector Store Configuration
spring.ai.vectorstore.redis.uri=redis://localhost:6379
spring.ai.vectorstore.redis.index=spring-ai-index
spring.ai.vectorstore.redis.prefix=doc:

# Embedding Configuration
spring.ai.openai.embedding.model=text-embedding-ada-002
```

```
@SpringBootTest
class ChatServiceTest {

    @MockBean
    private ChatClient chatClient;

    @Autowired
    private ChatService chatService;

    @Test
    void shouldGenerateResponse() {
        // Given
        when(chatClient.prompt().user("Hello").call().content())
            .thenReturn("Hello! How can I help you?");

        // When
        String response = chatService.chat("Hello");

        // Then
    }
}
```

SVANTAGGI E LIMITAZIONI

- ➡ **Progetto giovane:** API ancora in evoluzione (versione 1.x)
- ➡ **Astrazione overhead:** layer aggiuntivo rispetto alle API native
- ➡ **Dipendenza dall'ecosistema Spring:** lock-in tecnologico
- ➡ **Supporto provider limitato:** non tutti i provider sono supportati
- ➡ **Funzionalità avanzate:** alcuni provider offrono feature specifiche non astratte
- ➡ **Learning curve:** necessario apprendere nuovi concetti AI oltre Spring

- ➡ **Breaking changes:** possibili modifiche incompatibili tra versioni
- ➡ **Documentazione incompleta:** esempi e best practices in sviluppo
- ➡ **Community piccola:** meno risorse e troubleshooting disponibili

Spring AI è attualmente in versione 1.0.1. Sebbene l'API principale sia stabile, alcune funzionalità potrebbero cambiare nelle future versioni minor.

Considerazione per progetti enterprise

```
// Chiamata diretta OpenAI (senza Spring AI)
OpenAiService service = new OpenAiService("sk-...");
CompletionRequest request = CompletionRequest.builder()
    .model("gpt-4")
    .prompt("Explain Spring Boot")
    .temperature(0.7)
    .maxTokens(150)
    .build();
CompletionResult result = service.createCompletion(request);

// VS Spring AI (più semplice ma con overhead)
ChatClient client = ChatClient.builder().build();
String response = client.prompt()
    .user("Explain Spring Boot")
    .call()
    .content();
```

Semplicità vs Performance: trade-off da valutare