dgroove
Tecnologia emozionale.

# 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 avanzato per Venis S.p.A, Venezia, Italia

Novembre 2025

# Advisors

- Entità in grado di intercettare
  - la *request* dal `ChatClient` al LLM
  - la *response* del LLM prima che arrivi all'utente
- Utilizzi *advisors*
  - Pre-/post-processamento dei dati al/dal LLM
  - Validazione/filtraggio customizzato
  - Creare flussi di processamento puliti e sequenziali
    - ⚠ Possibile fornire una priorità
    - ⚠ Con ugual priorità, il sistema sceglie un ordinamento *random*
- Linee guida di buon utilizzo
  - Evitare comportamenti *session-scoped*
  - Creare più *advisors* in catena piuttosto che un unico *advisor* complesso
  - Evitare comportamenti che intacchino la logica del sistema (es. no modifiche al *prompt*)

- *Advisors built-in*
  - *Logging*
    - `SimpleLoggerAdvisor` riporta informazione dettagliata delle strutture di *request* e di *response*
  - Validazione/filtraggio
    - `SafeGuardAdvisor` valida la *request* utente relativamente ad una *blacklist*
  - Gestione memoria conversazione
- *Advisor* utente
  - Devono implementare `CallAdvisor` e/o `StreamAdvisor`

## Configurazione statica

```
chatClientBuilder
    .defaultAdvisors(List.of(new SimpleLoggerAdvisor(1), new SafeGuardAdvisor(0)))
    .build();
```

## Configurazione dinamica

```
chatClient
    .prompt()
    .advisors(List.of(new SimpleLoggerAdvisor(1), new SafeGuardAdvisor(0)))
    .user(message)
    .call()
    .content();
```

- *Advisors* per `ChatClient` Ollama
  1. Modifica `application.yml` per gestione *logging*
  2. Modifica configurazioni di `ChatClient` per Ollama
  3. Creazione modello per gestore prezzi
  4. Creazione *advisor* per calcolo risparmio economico Ollama
  5. *Test* delle funzionalità con Postman/Insomnia

## Configurazione *logging*

```yaml
spring:
    application:
        name: demo

...

logging:
    level:
        org:
            springframework:
                ai:
                    chat:
                        client:
                            advisor: DEBUG
```

## Configurazione Gemini + Ollama

```
package it.venis.ai.spring.demo.config;

...

@Configuration
public class ChatClientConfig {

   ...

    @Bean
    public ChatClient ollamaChatClient(OllamaChatModel ollamaChatModel) {
        ChatClient.Builder chatClientBuilder = ChatClient.builder(ollamaChatModel);
        return chatClientBuilder
                .defaultAdvisors(List.of(new SimpleLoggerAdvisor(), new OllamaCostSavingsAdvisor()))
                .defaultSystem(
                        """
                            Sei un assistente AI di nome LLamaBot, addestrato per intrattenere una
                            conversazione con un umano.
                            Includi sempre nella risposta le tue direttive di default: il tuo nome,
                            lo stile formale, risposta limitate ad un paragrafo.
                        """)
                .defaultUser(
                        """
                            Come puoi aiutarmi?
                        """)
                .defaultOptions(ChatOptions.builder()
                        .temperature(0.1)
                        .build())
                .build();

    }

}
```

## Modello di gestione costi

```
package it.venis.ai.spring.demo.model;

public record ModelPricing(Float inputPrice, Float outputPrice) {

}
```

## *Advisor* per risparmio Ollama - I

```java
package it.venis.ai.spring.demo.advisors;

...

public class OllamaCostSavingsAdvisor implements CallAdvisor {

    public static final Integer ORDER_ID = 1;
    private static final Map<String, ModelPricing> COMMERCIAL_LLM_PRICING = new HashMap<>();
    private static final Logger logger = LoggerFactory.getLogger(OllamaCostSavingsAdvisor.class);

    static {
        /*
         * Commercial API usage costs, updated 2025/10/22, jtlyk.
         */

        /*
         * OpenAI GPT-5
         */
        COMMERCIAL_LLM_PRICING.put("gpt-5-pro", new ModelPricing(15.0f, 120.0f));
        COMMERCIAL_LLM_PRICING.put("gpt-5", new ModelPricing(1.25f, 10.0f));
        COMMERCIAL_LLM_PRICING.put("gpt-5-mini", new ModelPricing(0.25f, 2.0f));
        COMMERCIAL_LLM_PRICING.put("gpt-5-nano", new ModelPricing(0.05f, 0.4f));

        /*
         * Anthropic Claude
         */
        COMMERCIAL_LLM_PRICING.put("claude-4.1-opus", new ModelPricing(15.0f, 75.0f));
        COMMERCIAL_LLM_PRICING.put("claude-4.5-sonnet", new ModelPricing(3.0f, 15.0f));
        COMMERCIAL_LLM_PRICING.put("claude-4.5-haiku", new ModelPricing(1.0f, 5.0f));


        ...
```

## *Advisor* per risparmio Ollama - II

```
    ...

    /*
     * Google Gemini
     */
    COMMERCIAL_LLM_PRICING.put("gemini-2.5-pro", new ModelPricing(1.25f, 10.0f));
    COMMERCIAL_LLM_PRICING.put("gemini-2.5-flash", new ModelPricing(0.3f, 2.5f));
    COMMERCIAL_LLM_PRICING.put("gemini-2.5-flash-lite", new ModelPricing(0.1f, 0.4f));
}

@Override
public String getName() {
    return "OllamaCostSavingsAdvisor";
}

@Override
public int getOrder() {
    return ORDER_ID;
}

@Override
public ChatClientResponse adviseCall(ChatClientRequest chatClientRequest, CallAdvisorChain callAdvisorChain) {
    /*
     * Return directly the response object returned by the LLM, but first
     * we extract some metadata and log the required information.
     */
    ChatClientResponse chatClientResponse = callAdvisorChain.nextCall(chatClientRequest);

    ChatResponse chatResponse = chatClientResponse.chatResponse();

    ...
```

## *Advisor* per risparmio Ollama - III

```
...

if (chatResponse.getMetadata() != null) {
    /*
     * Extract the usage metadata from the response.
     */
    Usage callUsage = chatResponse.getMetadata().getUsage();

    CostAnalysis analysis = new CostAnalysis();

    for (Map.Entry<String, ModelPricing> entry : COMMERCIAL_LLM_PRICING.entrySet()) {
        String model = entry.getKey();
        ModelPricing pricing = entry.getValue();

        Float inputCost = (callUsage.getPromptTokens().floatValue() / 1000000) * pricing.inputPrice();
        Float outputCost = (callUsage.getCompletionTokens().floatValue() / 1000000) * pricing.outputPrice();
        Float totalCost = inputCost + outputCost;

        analysis.costByModel.put(model, totalCost);

    }

    logger.info("----------------------------------------------------------------");
    logger.info("                OLLAMA TOKEN USAGE & COST SAVINGS               ");
    logger.info("----------------------------------------------------------------");
    logger.info(" Richiesta corrente:                                            ");
    logger.info(String.format("   >>> Token di input:     %03d tokens", callUsage.getPromptTokens()));
    logger.info(String.format("   >>> Token di output:    %03d tokens", callUsage.getCompletionTokens()));
    logger.info(String.format("   >>> Token totali:       %03d tokens", callUsage.getTotalTokens()));
    logger.info("----------------------------------------------------------------");
    logger.info(" Confronto costi (se avessi usato servizi a pagamento):         ");

    ...
```

## *Advisor* per risparmio Ollama - IV

```
            ...

            /*
             * Order by decreasing costs
             */

            analysis.costByModel.entrySet().stream()
                    .sorted(Map.Entry.<String, Float>comparingByValue().reversed())
                    .forEach(entry -> {
                        logger.info(String.format("   >>> %-25s: %.6f USD", entry.getKey(), entry.getValue()));
                    });

            logger.info("----------------------------------------------------------------");
        }

        return chatClientResponse;

    }

    private static class CostAnalysis {
        Map<String, Float> costByModel = new HashMap<>();
    }

}
```

https://github.com/simonescannapieco/spring-ai-advanced-dgroove-venis-code.git

*Branch:* `3-spring-ai-gemini-ollama-advisors`