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

Settembre 2025

SYSTEM E ROLE PROMPTING

1. *System prompting*
   - *Focus:* impostare contesto, direttive, formato utilizzati dal LLM
   - ⚠ Lasciare alla parte *user* il solo compito di fornire l'istanza della *task* da risolvere
   - Vantaggi
     - Aumento dell'efficacia con modelli contenuti
     - Possibilità di sfruttare un *system prompt* per molteplici *user prompts*
     - Maggior chiarezza ed ordine nella scrittura dei *prompt*

2. *Role prompting*
   - *Focus:* assegnare un ruolo specifico al LLM
   - ⚠ Specializzare la conversazione, utilizzando terminologia e registri tipici di specialisti del settore
   - Vantaggi
     - Informazione generata in media più rilevante ed entropica

- Sistema di risposta automatica a recensioni utente su un artefatto
  1. Creazione *string template* per istruzioni di sistema
  2. Creazione *string template* per istruzioni utente
  3. Modifiche ad `application.yml` per evitare conflitti di configurazione
  4. Modifiche ad interfaccia ed implementazione del servizio Gemini
  5. Modifica del controllore MVC per servizio Gemini
  6. *Test* delle funzionalità con Postman/Insomnia

File `set-advice-system-prompt.st`

```
Sei un esperto in materia di {artefatto}.
Rispondi in maniera coerente alla recensione fornita dallo '''user''', il cui sentiment è stato definito come
{sentiment}.
In caso di recensione positiva, rispondi in maniera entusiasta e consiglia allo '''user''' altre tre scelte simili
di {artefatto} legati a '''luoghi''' e '''persone''' riportati nella seguente lista YAML: {ner}. Se nella lista
non ci sono '''luoghi''' o '''persone''', non dare suggerimenti di {artefatto}.
In caso di recensione negativa, rispondi in maniera garbata e professionale, scusandoti per la spiacevole
esperienza.
In caso di recensione neutrale, informa lo '''user''' che rimarrai in attesa di ulteriori sue recensioni per poter
consigliare nuove scelte di {artefatto}, chiedendo quali '''luoghi''' e '''persone''' lo hanno colpito.
```

File `get-advice-user-prompt.st`

```
Rispondi a questa recensione:
'''Titolo''': {titolo}
'''Sottotitolo''': {sottotitolo}
'''Recensione''': {corpo}
```

File application.yml

```yaml
spring:
  application:
    name: demo
  ai:
    openai:
      api-key: ${GOOGLE_AI_API_KEY}
      base-url: https://generativelanguage.googleapis.com/v1beta/openai
      chat:
        completions-path: /chat/completions
        options:
          model: gemini-2.0-flash-lite
          temperature: 2
          #top-p: .99
          #top-k: 30
          #max-tokens: 100
          #max-completion-tokens: 100
          #frequency-penalty: .2
          #presence-penalty: .2
          #stop-sequences:
          #- 3
```

Interfaccia servizio Gemini

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

import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.ArtifactRequest;
import it.venis.ai.spring.demo.model.DefinitionRequest;
import it.venis.ai.spring.demo.model.DefinitionResponse;
import it.venis.ai.spring.demo.model.Question;

public interface GeminiFromClientService {

    String getAnswerFromClient(String question);

    Answer getAnswerFromClient(Question question);

    Answer getDefinitionFromClient(DefinitionRequest definitionRequest);

    Answer getCustomFormatDefinitionFromClient(DefinitionRequest definitionRequest);

    Answer getJSONUserFormatDefinitionFromClient(DefinitionRequest definitionRequest);

    DefinitionResponse getJSONOutputConverterFormatDefinitionFromClient(DefinitionRequest definitionRequest);

    Answer getSentimentForArtifact(ArtifactRequest artifactRequest);

    Answer getNERinYAMLForArtifact(ArtifactRequest artifactRequest);

    Answer getSuggestionForArtifact(ArtifactRequest artifactRequest);

}
```

Implementazione servizio Gemini - I

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

import java.util.Map;
import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.template.st.StTemplateRenderer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.io.Resource;
import org.springframework.stereotype.Service;
import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.DefinitionRequest;
import it.venis.ai.spring.demo.model.Question;

@Service
public class GeminiFromClientServiceImpl implements GeminiFromClientService {

    private final ChatClient chatClient;

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

    @Override
    public String getAnswerFromClient(String question) {
        return this.chatClient.prompt()
                .user(question)
                .call()
                .content();
    }
    ...
```

Implementazione servizio Gemini - II

```
...
@Value("classpath:templates/set-advice-system-prompt.st")
private Resource adviceSystemPrompt;

@Value("classpath:templates/get-advice-user-prompt.st")
private Resource adviceUserPrompt;

@Override
public Answer getSuggestionForArtifact(ArtifactRequest artifactRequest) {
    String sentiment = getSentimentForArtifact(artifactRequest).answer();
    String ner = getNERinYAMLForArtifact(artifactRequest).answer();
    String chatResponse = this.chatClient.prompt()
            .options(ChatOptions.builder()
            .model("gemini-2.0-flash")
            .temperature(0.3)
            .topP(1.0)
            //.topK(30)
            .maxTokens(500)
            //.frequencyPenalty(0.1)
            //.presencePenalty(0.1)
            .build())
            .system(s -> s.text(this.adviceSystemPrompt)
                    .params(Map.of("sentiment", sentiment, "ner", ner,
                            "artefatto", artifactRequest.artifact().type())))
            .user(u -> u.text(this.adviceUserPrompt)
                    .params(Map.of("titolo", artifactRequest.artifact().title(),
                            "sottotitolo", artifactRequest.artifact().subtitle(),
                            "corpo", artifactRequest.artifact().body())))
            .templateRenderer(StTemplateRenderer.builder().startDelimiterToken('{')
                    .endDelimiterToken('}')
                    .build())
            .call()
            .content();
    return new Answer(chatResponse);
}
}
```

Implementazione controllore REST

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

import org.springframework.web.bind.annotation.RestController;
...
import it.venis.ai.spring.demo.services.GeminiFromClientService;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

@RestController
public class QuestionFromClientController {

    private final GeminiFromClientService geminiService;

    ...

    @PostMapping("/client/definition/json/user")
    public Answer getJSONUserFormatDefinition(@RequestBody DefinitionRequest definitionRequest) {

        return this.geminiService.getJSONUserFormatDefinitionFromClient(definitionRequest);

    }

    ...

    @PostMapping("/client/advice")
    public Answer getSuggestionForArtifact(@RequestBody ArtifactRequest artifactRequest) {

        return this.geminiService.getSuggestionForArtifact(artifactRequest);

    }

}
```

- Sistema di promozione turistica di luoghi a partire da recensione utente
  - Ruolo del LLM: agente di viaggi
  - Suggerire tre posti da visitare per ciascun luogo estratto tramite NER dalla recenzione/descrizione dell'artefatto
  - REST API `/client/advice/place`

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

Branch: `15-spring-ai-gemini-system-role-prompting`