Project 1(NovaAI)

# Spring AI

Spring AI is an extension of the Spring Framework designed to integrate artificial intelligence (AI) capabilities into Spring-based applications. It provides tools for building AI-driven solutions, including machine learning (ML), deep learning (DL), and other intelligent systems, using the familiar Spring ecosystem.

Here’s how it works:

1. **Integration with AI Libraries**: Spring AI integrates with popular AI and ML libraries such as TensorFlow, PyTorch, and Deeplearning4j. It abstracts the complexities of working directly with these libraries and provides a seamless experience within the Spring environment.
2. **Spring Boot Compatibility**: It leverages Spring Boot for creating microservices that can expose AI models as RESTful APIs. This allows you to easily deploy AI models and make them available for real-time predictions in your applications.
3. **Machine Learning Pipelines**: Spring AI provides tools to create machine learning pipelines within a Spring-based architecture. You can define data ingestion, training, and prediction workflows, integrate these pipelines into business logic, and easily manage them using Spring’s dependency injection and lifecycle management.
4. **Model Deployment and Inference**: With Spring AI, trained models can be deployed and served as services. These models can perform inference on new data, making predictions, classifications, or decisions based on the trained model.
5. **Integration with Data Sources**: You can easily integrate Spring AI with data sources such as databases, Kafka, or even real-time data streams, making it suitable for both batch and real-time AI use cases.
6. **Scalability and Performance**: Leveraging the Spring Cloud ecosystem, Spring AI applications can be easily scaled and deployed in cloud environments, ensuring high availability and performance.

In summary, Spring AI combines the power of AI with the robustness of the Spring Framework, making it easier to build, deploy, and manage AI applications in a production environment.

* On Api Platfomr buid a project and get a api key and add it to application props in ur project

1. Firstly well use chat model api offered by spring AI

Prompts  
**Prompts** in the context of AI, particularly generative models like GPT, refer to the input text or instructions provided by a user to guide the AI in generating a response. A prompt sets the context or defines the task the AI is expected to perform.

### Examples of Prompts:

1. **Instructional Prompts**:
   * "Explain the theory of relativity in simple terms."
   * "Write a poem about the ocean."
2. **Conversational Prompts**:
   * "What is the capital of France?"
   * "How can I bake a chocolate cake?"

# Chat service

@Service  
public class ChatService {  
 private final ChatModel chatModel;  
  
 public ChatService(ChatModel chatModel) {  
 this.chatModel = chatModel;  
 }  
  
 public String getResponse(String prompt) {  
 return chatModel.call(prompt);  
 }  
 //call method of chatModel sends a promt to AIchat Model which then return a response  
  
 public ChatResponse getResponseOptions(String prompt) {  
 return chatModel.call(  
 new Prompt(prompt,  
 OpenAiChatOptions.*builder*()  
 .withModel("gpt-4-o")  
 .withTemperature(0.4)  
 .build()  
 ));  
 }

1. **ChatModel (Dependency Injection):**
   * ChatModel is a class injected into the ChatService via the constructor. It likely handles communication with the AI model.
   * The ChatModel class includes a call() method used to send prompts and receive responses.
2. **Method: getResponse(String prompt)**
   * **Purpose:**  
     Sends a simple string prompt to the AI model and returns the AI's response as a String.
   * **How it works:**  
     The call() method of chatModel is invoked with the given prompt string. This method sends the prompt to the AI and returns the response.

java

Copy code

public String getResponse(String prompt) {

return chatModel.call(prompt);

}

1. **Method: getResponseOptions(String prompt)**
   * **Purpose:**  
     Sends a prompt with additional options (like temperature and model selection) to the AI model and returns a more detailed ChatResponse.
   * **How it works:**
     + A Prompt object is created. This encapsulates the input prompt string and additional settings specified via OpenAiChatOptions.
     + OpenAiChatOptions.builder() is used to configure:
       - withModel("gpt-4-o"): Specifies the AI model to use (e.g., GPT-4 in this case).
       - withTemperature(0.4): Sets the "creativity" or randomness of the AI's response. Lower values produce more deterministic responses.
     + The call() method of chatModel is invoked with the Prompt object, and it returns a ChatResponse object, which likely contains the AI's response and other metadata.

java

Copy code

public ChatResponse getResponseOptions(String prompt) {

return chatModel.call(

new Prompt(prompt,

OpenAiChatOptions.builder()

.withModel("gpt-4-o")

.withTemperature(0.4)

.build()

)

);

}

### **Key Components Explained:**

1. **ChatModel:**
   * A class that abstracts interaction with the AI model.
   * The call() method is its primary function, sending prompts to the AI model and receiving responses.
2. **Prompt:**
   * Encapsulates the prompt to be sent to the AI and its associated options (e.g., temperature, model type).
3. **OpenAiChatOptions:**
   * Configures settings for the AI model:
     + withModel("gpt-4-o"): Specifies the model to use.
     + withTemperature(0.4): Controls randomness in the AI's responses.
4. **ChatResponse:**
   * A class representing the AI's response. Unlike a simple String, this might include:
     + The AI's textual response.
     + Metadata about the response, such as token usage or processing time.

### **Overall Functionality:**

* The ChatService class simplifies interactions with the AI model by exposing two methods:
  1. getResponse(): For simple prompts that need a basic string response.
  2. getResponseOptions(): For prompts requiring specific configurations, returning a detailed ChatResponse.

This design allows flexibility and supports both simple and advanced use cases for interacting with the AI model.

# 2.Image model API

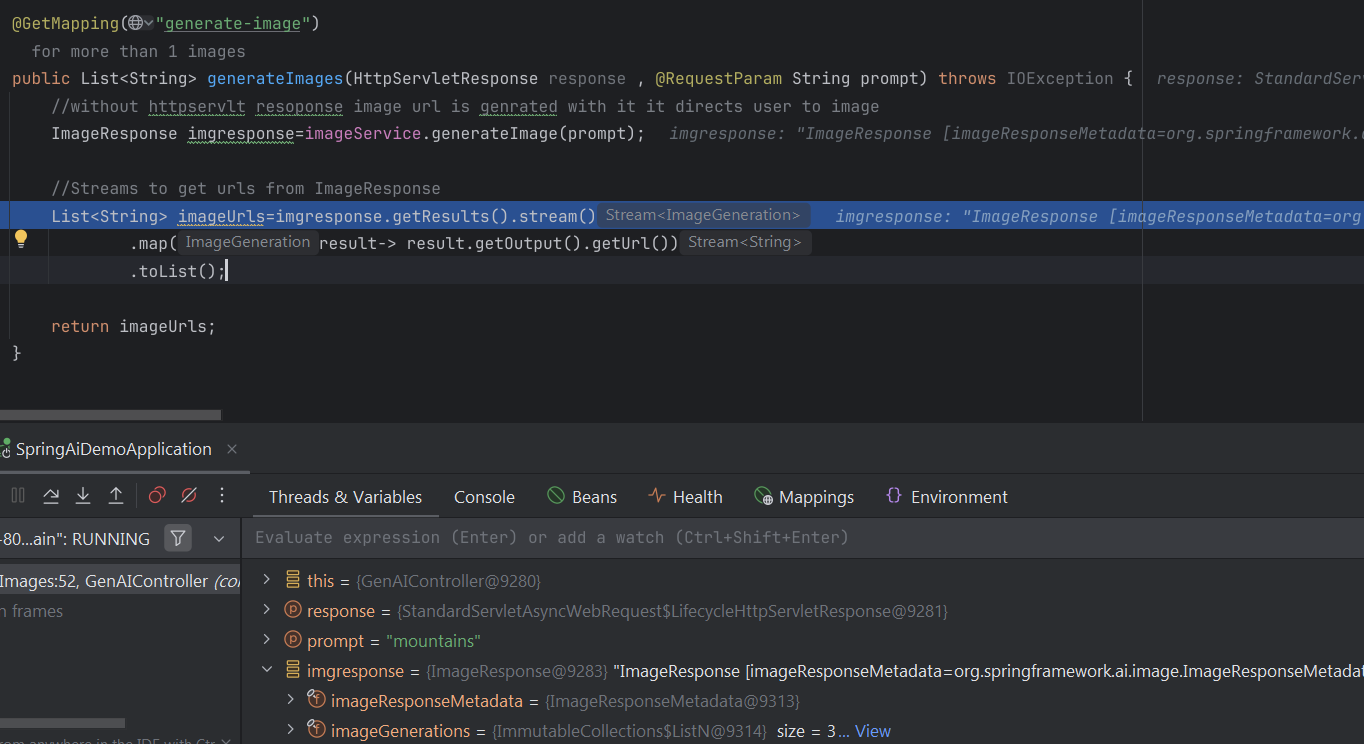
## ImageService

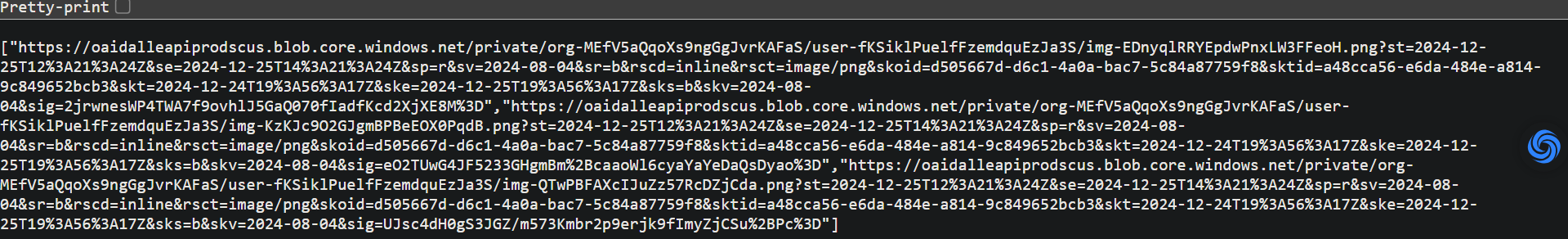
public ImageResponse generateImage(String prompt) {  
 ImageResponse response = openAiImageModel.call(  
 new ImagePrompt(prompt,  
 OpenAiImageOptions.*builder*()  
 .withQuality("hd")  
 .withN(1)  
 .withHeight(1024)  
 .withWidth(1024).build()  
  
 ));  
 return response;  
}

@GetMapping("generate-image")  
public void generateImages(HttpServletResponse response , @RequestParam String prompt) throws IOException {  
 //without httpservlt resoponse image url is genrated with it it directs user to image  
 ImageResponse imgresponse=imageService.generateImage(prompt);  
 String imgUrl=imgresponse.getResult().getOutput().getUrl();  
 response.sendRedirect(imgUrl);  
  
}

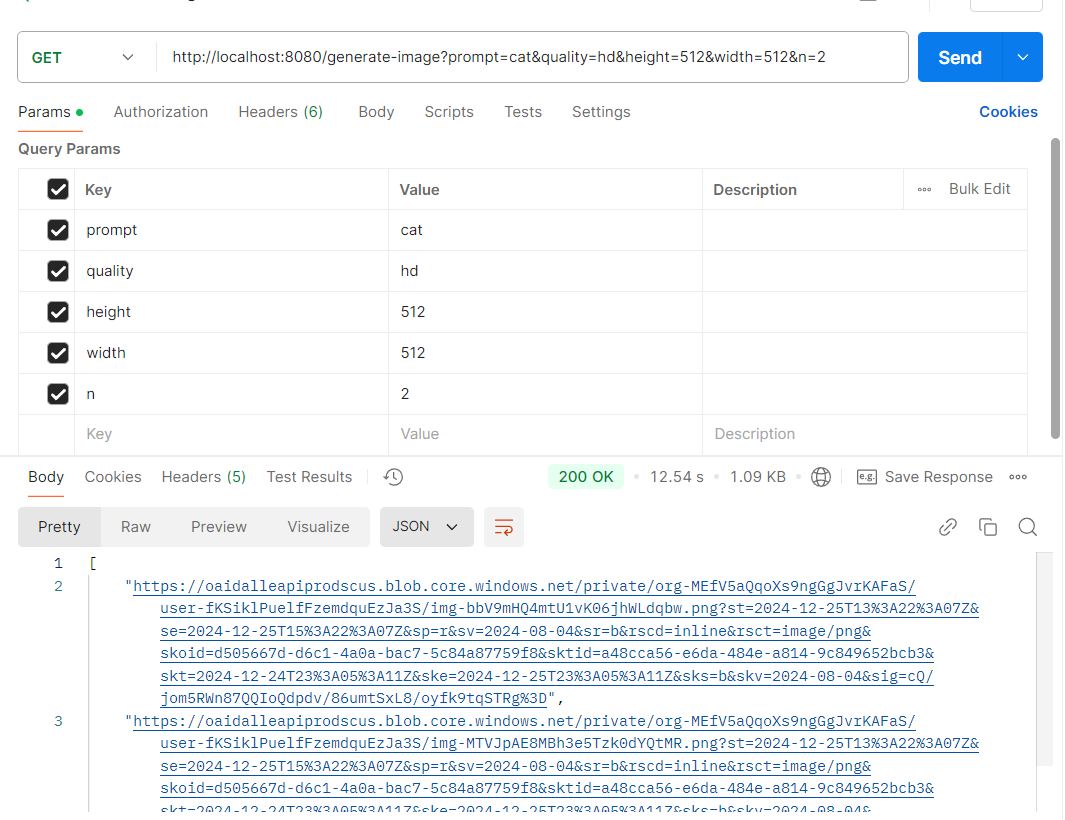
* For dall-e3 n=1 is only valid
* Urls remain for 60 seconds only
* Vivid – generates more dramatic images
* Normal – generates more normal humaly images

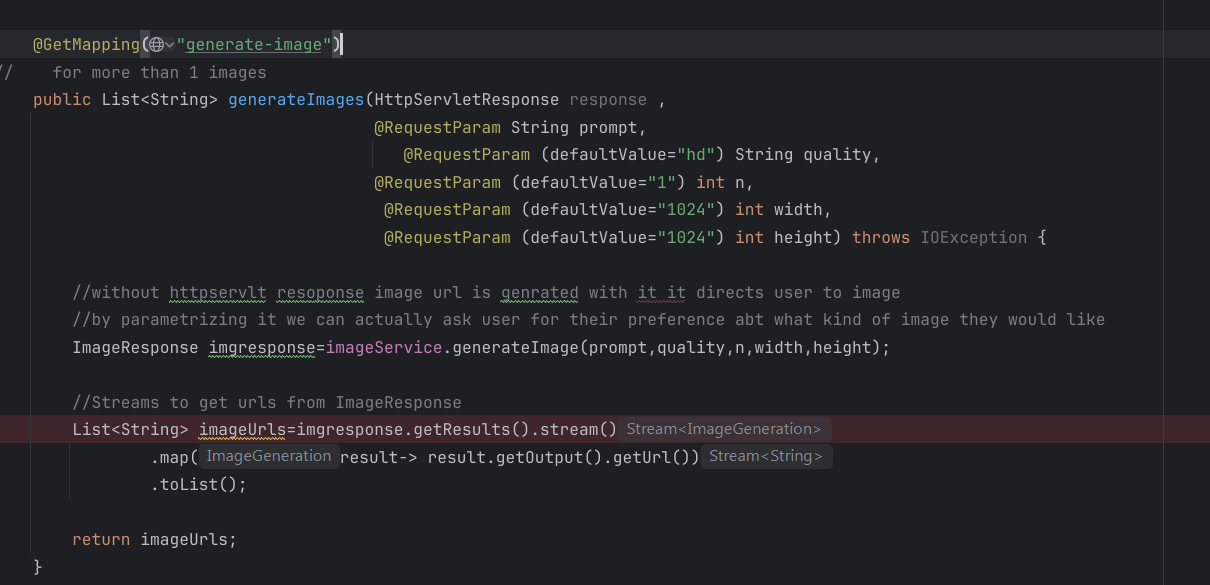
Multiple images generation using dall-e-2 model



  
ull get three response like this

# Images as demanded by user(pareametrizing the generation)





Now user can tell how he ilikes the image to be generated

# Recipe Generator

### 5. **Prompt Template Creation**

A multi-line string (""") with placeholders ({ingredients}, {cuisine}, {dietaryInstructions}) for user input. It specifies the format of the input prompt to the AI model.

### 6. **Template Population**

PromptTemplate promptTemplate = new PromptTemplate(template);

Map<String, Object> params = Map.of(

"ingredients", ingredients,

"cuisine", cuisine,

"dietaryInstructions", dietaryInstructions

);

Prompt prompt = promptTemplate.create(params);

* **PromptTemplate**: Wraps the template string.
* **Map.of**: Creates a map with keys matching the placeholders in the template and values from user input.
* **Prompt**: Created by substituting the placeholders with actual user input.

### 7. **Calling the AI Model**

return chatModel.call(prompt).getResult().getOutput().getContent();

* **chatModel.call(prompt)**: Sends the prompt to the AI model.
* **getResult()**: Retrieves the model's response.
* **getOutput().getContent()**: Extracts the final content (the recipe) from the response.

### How It Works

1. A user specifies ingredients, cuisine, and dietary instructions.
2. These inputs are injected into a predefined prompt template.
3. The AI chat model generates a recipe based on this prompt.
4. The method returns a detailed recipe, including a title, ingredients, and cooking instructions.

### Example Usage

RecipeService service = new RecipeService(chatModel);

String recipe = service.createRecipe(

"chicken, tomatoes, onions",

"Italian",

"gluten-free"

);

System.out.println(recipe);

The AI would return something like:

Title: Gluten-Free Italian Chicken Stew

Ingredients:

- Chicken

- Tomatoes

- Onions

...

Cooking Instructions:

1. Sauté onions...

2. Add chicken and cook...

# Code

public RecipeService(ChatModel chatModel) {  
 this.chatModel = chatModel;  
 }  
  
 public String createRecipe(String ingredients,  
 String cuisine,  
 String dietaryInstructions){  
  
 String template= """  
 I want to create a recipe using the following ingredients: {ingredients}.  
 The cuisine type i prefer is {cuisine}.  
 Please consider the following dietary restrictions: {dietaryInstructions}.  
 Please provide me with detailed recipe including title, list of ingredients,   
 and cooking instructions  
 """;  
 //this is basically adding to the ingredients and info that user gives then well just use chat bot  
  
 PromptTemplate promptTemplate = new PromptTemplate(template);  
 Map<String,Object> params= Map.*of*(  
 "ingredients",ingredients,  
 "cuisine",cuisine,  
 "dietaryInstructions",dietaryInstructions  
 );  
   
 //the place holders in template need to be replacd with the map items and since there arent   
 //single values hence we are using maps sp that in place of {ingredients} all ingredients   
 //mentioned in map can be inserted there and so on...  
  
 Prompt prompt = promptTemplate.create(params);//this fits all these values into the template  
 return chatModel.call(prompt).getResult().getOutput().getContent();  
 }  
}

