

SPRING AI

FOR YOUR ORGANIZATION

The GCP Vertex AI Edition

With Spring AI V1.0 GA

MUTHUKUMARAN NAVANEETHAKRISHNAN

Spring AI for Your Organization

GCP Vertex AI Edition

Muthukumaran Navaneethakrishnan

This book is available at https://leanpub.com/springai

This version was published on 2025-06-13



This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2025 Muthukumaran Navaneethakrishnan

To Balasundari Navaneethakrishnan.

Contents

1: 1 Introduction **1**
1.1 Evolution of AI Frameworks in Java 2
1.2 Emergence of Spring AI 3
1.3 Introduction to Generative AI 3
1.4 Overview of Spring AI 4
1.5 Introduction to Google Cloud's Vertex AI 5
1.6 Goals of the Book (What You Will Learn) 6
1.7 Who Should Read This Book 6
1.8 Prerequisites 8
1.9 What's Next 9

2: Creating ChatBot with Spring AI **10**
2.1 Chapter Highlights: 10
2.2 How to Initialize the project 12
2.3 Creating a RESTful Chat Endpoint 14
2.4 Connecting to Vertex AI Gemini 16
2.5 Implementing the Chat Endpoint 16
2.6 How to Test the Application 17
2.7 Configuring Chat Options for Vertex AI 18
2.8 How to Test the Application 22
2.9 Understanding API Interaction with Vertex AI 23
2.10 How to Deploy in GCP Cloud Run 26
2.11 Review Questions 29
2.12 Answers to Review Questions 30
2.13 What We Learned 31

© 2024 Muthukumaran Navaneethakrishnan

3: Building Context-Aware Chatbots with Spring AI: Memory Management and Conversational Continuity 32
3.1 Chapter Highlights 32
3.2 The Challenge of Contextual Conversation 33
3.3 Understanding LLMs and Contextual Conversations 34
3.4 Implementing Conversation History in Spring AI 36
3.5 Managing ChatBot Conversations with Session IDs 37
3.6 Updating the ChatBot Endpoint for Conversational Context 40
3.7 Testing the Enhanced ChatBot with Session History 43
3.8 Simplifying Chat History Management with Advisors 44
3.9 Creating a DELETE Endpoint to Reset Chat Session History 48
3.10 Understanding System Prompts in Chatbot Responses 50
3.11 What We Learned 55

4: Creating Structural Data Bots 56
4.1 Chapter Highlights: 56
4.2 Make Inventory to be queried by Natural Language 56
4.3 Update Dependencies 56
4.4 Setting up an SQL database in GCP Cloud SQL 56
4.6 Getting Ready for Natural Language Processing 58
4.7 From Natural Queries to SQL 58
4.8 JSON Conversion of SQL Results 59
4.9 SQL to Natural Language Response 59
4.10 Bringing It All Together: The Endpoint 59
4.11 Testing the Endpoint 60
4.12 Exercise: Implement it using CSV instead of JSON 60
4.13 Improving SQL Query Flexibility for User Queries 61
4.14 Handling multiple questions 61
4.15 What We Learned 63

5: LLM Tool Calling with Spring AI 64
5.1 Chapter Highlights: 64

© 2024 Muthukumaran Navaneethakrishnan

5.2 Understanding LLM Tool Calls 64
5.3 Implementing Tool Calling with REST API 66
5.4 Java Implementation with Direct API Calls 67
5.5 Handling Multiple Tool Calls 68
5.6 Simplifying with Spring AI 68
5.7 Advanced Tool Calling: Sequential Tool Routing 69
5.8 Few Words 70
5.9 What We Learned 70

6: Building Chatbots with Text and PDF Files 71
6.1 Chapter Highlights: 71
6.2 Use Case: Answering Questions About Vacuum Cleaners based on text file 71
6.3 Use Case: Answering Questions About Laptop based on pdf manual 73

7: Building Chatbots with Multimedia Capabilities 76
7.1 Chapter Highlights: 76
7.2 Use Case: Answering Questions About Coupons from an Image . . 77
7.3 Invoking the Chatbot with Image Data 77
7.4 Testing 78
7.5 Use Case: Answering Questions based on a Customer Care Audio . 79
7.6 Testing 80
7.7 Use Case: Answering Questions from a Vaccum Cleaner Advertisement video 80
7.8 Invoking the Chatbot with Video Data 80
7.9 Testing 81

8: Using RAG to Make LLMs Smarter with Internal Data 82
8.1 Integrating Internal Data with LLM 82
8.2 Chapter Highlights: 82
8.3 How to Create Embeddings 83
8.4 Vector Database 84

© 2024 Muthukumaran Navaneethakrishnan

8.5 Making Similar Documents as Conversational 88
8.6 Exercise: Modify Chatbot to Search All Records When Department Is Not Specified 89

9: Building Internal Knowledge based assistant 91
9.1 Chapter Highlights: 91
9.2 Integrating Google Cloud Storage Bucket 91
9.3 Building a Chatbot with Internal Documents 92
9.4 Adding File References on Chat 96

10: PDF Documents & Image Embeddings 98
10.1 Chapter Highlights: 98
10.1 Saving PDF Document Pages as Image Embeddings 98
10.2 Updating Chat Endpoint to use Image embeddings 99

11: Accessing other models in GCP Model Garden 101
11.1 Chapter Highlights: 101
11.1 Setting Up GCP 101
11.2 Setting Up Gradle and Configuration 101
11.3 Setting Up Authentication Tokens for Llama Requests 101

12: Whats Next 104

© 2024 Muthukumaran Navaneethakrishnan

Acknowledgments

Thank you to Shaama M for making this new revision possible, Especially the way you helped me with the migration from the old version to the new one. Your support has been invaluable.

Thank you to Madhana Gopala Ramachandran and Agalya for their invaluable support in testing code, fixing bugs, and editing.

Special thanks to Peter Uballed for encouraging me to complete this project. Without that push, this draft might have remained unfinished for another year.

I am also deeply grateful to my mentors, Prabagar Ganapathy and Senthilkumar Sundararajan, who have always been there when I needed guidance and have helped me take the next step.

Lastly, heartfelt thanks to my family and friends for their unwavering support throughout this journey.

© 2024 Muthukumaran Navaneethakrishnan

Preface

I began exploring Spring AI and Vertex AI with the simple goal of gaining a deeper understanding. Along the way, I realized I had valuable insights to share with the Spring community, which led me to write this book. This isn't an "I'm an expert, look at me" kind of project—it's a practical, dev-to-dev guide. In the middle of writing, I even found myself contributing to the Spring AI framework.

If you're familiar with Spring and eager to add AI-driven features like chatbots, natural language data querying, multimedia capabilities, RAG, text embeddings, multimedia embeddings, and using Model Garden—all specifically with GCP Vertex AI—this Spring AI handbook walks you through each step. With hands-on examples, we'll cover everything from setting up your environment to building intelligent, context-aware systems. Each chapter builds on the last, aiming to make your Spring applications not just smarter, but more intuitive and user-focused.

I hope this book becomes your go-to guide for integrating AI into your Spring projects using GCP Vertex AI, helping you build products quicker and more efficiently. Let's dive in together and explore the exciting possibilities!

© 2024 Muthukumaran Navaneethakrishnan

Trademark Notice

This document and the associated project incorporate technologies developed by various organizations and are used solely for educational and instructional purposes. Trademark acknowledgments are as follows:

- Spring is a trademark of Broadcom Inc. and/or its subsidiaries.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.
- Google Cloud Platform (GCP) and Vertex AI are trademarks of Google LLC. No affiliation with or endorsement by Google LLC is implied.
- PostgreSQL and the Slonik Logo are trademarks or registered trademarks of the PostgreSQL Community Association of Canada.
- Any other trademarks mentioned are the property of their respective owners.

This project and its documentation are independent of the companies and entities listed and do not imply endorsement. The use of the above trademarks is strictly for identification purposes and does not signify any association with Oracle Corporation, Broadcom Inc., Google LLC, the PostgreSQL Community Association of Canada, or any other mentioned affiliates.

© 2024 Muthukumaran Navaneethakrishnan

Chapter 1: 1 Introduction

Artificial Intelligence (AI) has evolved significantly, impacting every domain and reshaping how software is built and deployed. Java, a language renowned for its stability and enterprise focus, has been a key player in AI's history, contributing through frameworks like AIML for rule-based chatbots and Deeplearning4j for deep learning. However, with the rise of Python and its dominance in AI development, Java developers faced a gap in leveraging AI's latest advancements, especially with generative AI models like GPT.

This book bridges that gap by introducing **Spring AI**, a framework that seamlessly integrates AI into the Spring ecosystem. Spring AI simplifies complex tasks such as maintaining conversation history, implementing Retrieval-Augmented Generation (RAG), and using LLM functions, enabling developers to focus on building robust, scalable, and intelligent applications.

By leveraging tools like **Google Cloud's Vertex AI**, this book offers practical guidance on how Java developers can embrace the capabilities of generative AI without abandoning their existing ecosystem. From building chatbots to creating enterprise-ready AI applications, this book provides a comprehensive approach to mastering AI integration with Spring, transforming how we approach modern software development.

1.1 Evolution of AI Frameworks in Java

Java has been a key player in the development of artificial intelligence (AI) applications, offering a robust ecosystem of libraries and frameworks. Early AI tools in Java included:

- **AIML (Artificial Intelligence Markup Language)** (1995): AIML's first widely recognized implementation was in Java. The ALICE project (Artificial Linguistic Internet Computer Entity), which introduced AIML, heavily relied on Java for building its chatbot infrastructure. Java's platform independence and robust ecosystem made it a popular choice for early AI-related projects, including ALICE. The original interpreter for AIML, **Program D**, was also written in Java and became the foundation for many AIML-based chatbots. Java's simplicity and scalability at the time contributed to the adoption of AIML for creating rule-based conversational agents.
- **Apache OpenNLP** (2010): A library for natural language processing tasks, including entity recognition, tokenization, and text classification.
- **Stanford CoreNLP** (2014): A Java-based framework for natural language processing, offering robust tools for part-of-speech tagging, named entity recognition, dependency parsing, and sentiment analysis.
- **Deeplearning4j** (2014): A suite for building deep neural networks in Java, with support for distributed training and GPU acceleration.

1.1.1 Rise of Python in AI

The landscape of AI development shifted with the introduction of **PyTorch** (2016). PyTorch's dynamic computation graph, native C bindings, and strong

© 2024 Muthukumaran Navaneethakrishnan

GPU support made it a favorite for building and training deep learning models. Python's simplicity and vast ecosystem of AI libraries like TensorFlow, Scikit-learn, and PyTorch propelled it to the forefront of AI development, particularly with the emergence of generative AI.

1.2 Emergence of Spring AI

Generative AI and large language models (LLMs) have introduced transformative opportunities for developers. While Python has dominated due to its rich AI libraries, the introduction of generative models through APIs brought challenges such as:

- Managing **conversation history**.
- Implementing **Retrieval-Augmented Generation (RAG)** workflows.
- Leveraging **LLM functions** for complex tasks.

Manually handling these complexities can lead to inefficiencies. Recognizing this gap, the Spring community launched **Spring AI**.

Spring AI extends Spring's core principles—portability, modularity, and the use of **Plain Old Java Objects (POJOs)**—to simplify AI integration. Features include:

- Built-in support for conversation history and RAG workflows.
- Unified API for multiple AI providers like **OpenAI**, **Google Vertex AI**, and **Microsoft Azure**.
- Seamless integration with the **Spring ecosystem**, allowing developers to leverage familiar tools and methodologies.

By abstracting the complexities of AI, Spring AI empowers developers to create scalable, enterprise-ready AI applications efficiently.

© 2024 Muthukumaran Navaneethakrishnan

1.3 Introduction to Generative AI

Generative AI focuses on creating new content, such as text, images, or audio, based on patterns learned from training data. Unlike traditional AI models that predict or classify, generative AI learns **underlying structures** and generates **creative outputs**.

For example:

- **Traditional AI**: Predicts the next word in a sentence or identifies an image as a cat or dog.
- **Generative AI**: Composes entire paragraphs, generates realistic images, or creates music.

Generative AI opens up vast possibilities for innovation, such as:

- Chatbots producing human-like responses.
- Art and music creation.
- Summarizing and analyzing complex datasets.

This shift in capabilities pushes AI from **prediction to creation**, transforming applications in industries like education, entertainment, and customer service.

1.4 Overview of Spring AI

Spring AI is a Java framework that integrates artificial intelligence capabilities into applications. Key features include:

© 2024 Muthukumaran Navaneethakrishnan

- A unified API supporting chat, text-to-image, and embedding models from providers like **OpenAI**, **Google Vertex AI**, and **Microsoft Azure**.
- Seamless compatibility with the Spring ecosystem, enabling modular and scalable application design.
- Easy switching between AI components with minimal code changes, promoting flexibility.

Spring AI is designed for **enterprise-level applications**, offering:

1. **Portability**: Avoid vendor lock-in with support for multiple AI services.
2. **Scalability**: Grow AI capabilities alongside business needs.
3. **Efficiency**: Leverage Spring's existing infrastructure to save development time.

With Spring AI, Java developers can incorporate AI into applications without needing to switch to Python or learn new frameworks.

1.5 Introduction to Google Cloud's Vertex AI

Vertex AI on **Google Cloud** provides powerful tools to integrate AI into applications, such as:

- Chatbots with memory for conversational continuity.
- Natural language queries for SQL-like interactions.
- Handling diverse inputs like text, PDFs, and images.

Vertex AI works seamlessly with other Google Cloud services, including:

- **Cloud Run**: For scalable, serverless deployments.

© 2024 Muthukumaran Navaneethakrishnan

- **Cloud SQL**: For database integration.
- **Cloud Storage**: For managing multimedia files.

This integration makes Vertex AI an ideal platform for deploying intelligent, data-driven applications.

1.6 Goals of the Book (What You Will Learn)

This book is a practical guide for Spring developers to integrate **Generative AI** into their applications. By the end of this book, you will learn to:

- **Integrate AI into Java Applications**: Discover how Spring AI bridges traditional Java development with AI.
- **Build AI-Powered Chatbots**: Create chatbots capable of understanding and responding naturally.
- **Handle Complex Data Formats**: Process text, PDFs, and multimedia inputs.
- **Leverage Retrieval-Augmented Generation (RAG)**: Enhance chatbot intelligence using internal data.
- **Utilize LLM Functions**: Streamline workflows with advanced AI functionalities.
- **Optimize and Deploy AI Applications**: Deploy enterprise-ready applications on **Google Cloud Platform (GCP)** using services like Cloud Run.

This combination of conceptual knowledge, practical implementation, and deployment strategies makes this book a comprehensive resource for integrating AI with Spring.

© 2024 Muthukumaran Navaneethakrishnan

1.7 Who Should Read This Book

This book caters to:

1. **Backend Developers**: Whether seasoned or new, this book enhances your skills for building AI-driven applications.
2. **Managers Returning to Technical Roles**: Gain practical insights into integrating AI into projects.
3. **Java Developers Exploring AI**: Transition to AI development without switching to Python.
4. **Engineers Building Scalable AI Solutions**: Access hands-on examples and real-world use cases.

Whether you're diving into AI for the first time or looking to enhance your expertise, this book equips you to navigate the evolving landscape of Generative AI.

© 2024 Muthukumaran Navaneethakrishnan

1.8 Prerequisites

1.8.1 Technical Requirements

To get started, you need the **Java Development Kit (JDK)** version 17 or higher. You can verify the installation with the following command:

```
java -version
```

The preferred build tool for this book is **Gradle**, which can be checked using:

```
gradle -version
```

You also need the **Google Cloud SDK (gcloud CLI)** installed. Verify its installation with:

```
gcloud --version
```

For development, an IDE like **IntelliJ IDEA** or Eclipse is recommended. Lastly, ensure you have an active Google Cloud Platform (GCP) account. You can sign up here.

1.8.2 Knowledge Requirements

- Familiarity with **Java** & **Spring Boot**.
- Understanding of **generative AI** concepts and cloud platforms.

© 2024 Muthukumaran Navaneethakrishnan

1.8.3 GCP Setup

To set up your Google Cloud Platform (GCP) environment, start by visiting the GCP Console and creating a new project named `myspringai`. Once the project is created, navigate to the API Library and enable the **Vertex AI API**. This ensures the project is ready to integrate AI functionalities.

Next, configure the **gcloud CLI** on your system. Begin by authenticating your GCP account using the following command:

```
gcloud auth login
```

This will open a browser for authentication. After successful authentication, set the newly created project as the default for your gcloud CLI:

```
gcloud config set project myspringai
```

Finally, configure the default region for Cloud Run to `us-central1`:

```
gcloud config set run/region us-central1
```

This setup ensures that your environment is ready to deploy AI-enabled applications using Vertex AI and other GCP services.

1.9 What's Next

This chapter covered setting up Java, Gradle, and the Google Cloud SDK, along with creating a GCP project, enabling the Vertex AI API, and configuring the gcloud CLI. With these essentials in place, the next chapter will guide you through building a Spring AI chatbot, starting with project setup and designing a RESTful API for chatbot interactions, while integrating Vertex AI with Spring Boot.

© 2024 Muthukumaran Navaneethakrishnan

Chapter 2: Creating ChatBot with Spring AI

We begin our journey into building a Spring AI chatbot by establishing the project's foundation. This section will guide you through the initial setup, design considerations, and integration steps. We'll utilize Spring Boot to develop an application that interacts with Google Cloud Platform (GCP) and leverages its powerful AI capabilities. Through this process, we'll construct a chatbot that can engage users in conversation and respond to their inquiries.

2.1 Chapter Highlights:

1. **Project Initialization with Spring AI:**
    - We start by initializing our project using Spring Initializr, selecting essential dependencies such as Spring Web, Vertex AI Gemini, and Google Cloud Support. This lays the groundwork for building a robust Spring Boot application.
2. **Setting Up the Gradle Build File:**
    - Next, we configure the Gradle build file, adding necessary repositories and properties. This step ensures that our project is equipped with the right tools and dependencies needed for seamless integration with GCP and Vertex AI.
3. **Designing the ChatBot Endpoint:**

- Here, we focus on creating the endpoint for our ChatBotController. This section guides you through creating a structured endpoint that processes user queries and generates appropriate responses, making the chatbot interactive and functional.

4. **Integrating Vertex AI for Intelligent Responses:**

   - We explore the integration of Vertex AI into our Spring Boot application. This includes autowiring the `VertexAiGeminiChatModel` and modifying the chatbot's functionality to utilize this client for intelligent, AI-powered interactions.

5. **Practical Application and Deployment:**

   - Finally, we put theory into practice by deploying our application to GCP Cloud Run. This step marks the transition from development to a live environment, where the chatbot can interact with real users.

By the end of this chapter, you will have a fully functional Spring AI chatbot, integrated with GCP's Vertex AI, ready to respond to user queries with intelligence and context.

© 2024 Muthukumaran Navaneethakrishnan

2.2 How to Initialize the project

We're about to create a Spring Boot application that interacts with GCP to respond to user queries. This setup involves incorporating specific dependencies:

- Open Spring Initializr at start.spring.io and create a project named 'hello-spring-ai'
- Choose Gradle as your build tool (although Maven is also an option, this book will focus on Gradle)
- Include the following dependencies:
  - Spring Web
  - Lombok
  - Vertex AI Gemini
  - Google Cloud Support

After configuring these settings, click "Generate" to create the project and then open it in your IDE.

The generated Gradle file will include an additional repository for downloading Spring AI-related artifacts and configurations for managing dependencies related to GCP and Vertex AI Gemini:

```
repositories {
    mavenCentral()
    maven { url 'https://repo.spring.io/snapshot' }
}
```

The additional repository specifies the Spring snapshot repository, which is necessary for obtaining Spring AI-related dependencies as on when writing this book.

© 2024 Muthukumaran Navaneethakrishnan

Additionally, the Gradle properties (ext) will define versions for Spring Cloud, GCP, and Spring AI to ensure compatibility and stability:

```
ext {
    set('springAiVersion', "1.0.0-SNAPSHOT")
    set('springCloudGcpVersion', "5.4.3")
    set('springCloudVersion', "2023.0.0")
}
```

Besides the regular dependencies such as Spring Web and Lombok, your Gradle build file will also include:

```
implementation 'com.google.cloud:spring-cloud-gcp-starter'
implementation 'org.springframework.ai:spring-ai-vertex-ai-gemini-spring-b\
oot-starter'
```

These dependencies are essential for managing GCP configurations and leveraging Vertex AI capabilities within your Spring Boot application. To further enhance API testing and documentation, we will also include the following dependency:

```
implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.6.0'
```

This addition allows you to test the REST APIs directly from a browser by navigating to `http://{host}:{port}/swagger-ui.html`, typically accessible at `http://localhost:8080/swagger-ui.html`. The Springdoc OpenAPI UI integration facilitates seamless API testing and documentation management.

© 2024 Muthukumaran Navaneethakrishnan

2.2.1 Configure VertexAI Gemini Integration

Modify the `application.properties` file to set up the necessary properties for Vertex AIi Gemini:

```
spring.ai.vertex.ai.gemini.projectId=myspringai
spring.ai.vertex.ai.gemini.location=us-central1
spring.ai.vertex.ai.gemini.chat.options.model=gemini-1.5-flash
```

Ensure that the `projectId` matches your Google Cloud project ID. Select the `location` based on your project. Here are the details regarding the available model options:

- **gemini-1.5-pro**: A more advanced multimodal model designed for complex tasks, offering superior performance in reasoning and nuanced tasks. While it has a higher cost (approximately $7.00 for input and $21.00 for output per million tokens) and slower output speed, it excels in complex reasoning, creative writing, and coding.
- **gemini-1.5-flash**: A lightweight Multimodal model optimized for speed and efficiency, featuring fast output speed (163.6 tokens per second) and more cost-effective pricing($0.53 per million tokens). While it may not match Pro's performance in nuanced tasks, it excels in rapid responses, long context handling, and scalability.

For the most current model options, refer to Google Cloud's model versioning guide.

For this book, we would be using `gemini-1.5-flash` model due to its relevance, comprehensive features, and affordability also we will be using .

© 2024 Muthukumaran Navaneethakrishnan

2.3 Creating a RESTful Chat Endpoint

In this section, we'll design a RestController for our chat application, the core component that handles user requests and responses. Our goal is to create an interactive and functional chatbot.

Before we create an endpoint, we must structure the input and output formats. For testing purposes, we will establish a POST endpoint that accepts a JSON containing a 'question' property and returns a JSON with 'question' and 'answer' properties.

Define a request record named `ChatBotRequest` with a 'question' property:

```
public record ChatBotRequest(String question) {}
```

Define a response record named `ChatBotResponse` with 'question' and 'answer' properties:

```
public record ChatBotResponse(String question, String answer) { }
```

© 2024 Muthukumaran Navaneethakrishnan

Implement a RestController named `ChatBotController` that listens on the `/api/chat` endpoint with a method named `askQuestion`:

```
@RestController
public class ChatBotController {

    @PostMapping("/api/chat")
    public ChatBotResponse askQuestion(@RequestBody ChatBotRequest
    chatBotRequest) {
        return null;
    }
}
```

2.4 Connecting to Vertex AI Gemini

With the Vertex AI Gemini dependency included, a `VertexAiGeminiChatModel` class is available as a bean. This class has a method named 'call' which takes a string (the question) and returns a response (the answer).

To obtain an answer, use:

```
    String answer = vertexAiGeminiChatModel.call(question);
```

Spring automatically constructs the URL for the Vertex AI API based on the configured properties and uses the gcloud application credentials to execute the API call.

2.5 Implementing the Chat Endpoint

Now that the basic chat application setup is complete, it's time to integrate it with the Vertex API. This integration is crucial for enabling our application to send user queries to Vertex AI and receive responses:

Inject the `VertexAiGeminiChatModel` bean for autowiring:

© 2024 Muthukumaran Navaneethakrishnan

```
import org.springframework.ai.vertexai.gemini.VertexAiGeminiChatModel;

private final VertexAiGeminiChatModel vertexAiGeminiChatModel;

public ChatBotController(VertexAiGeminiChatModel vertexAiGeminiChatModel) {
    this.vertexAiGeminiChatModel = vertexAiGeminiChatModel;
}
```

Update the `askQuestion` method to utilize the 'call' method and construct the response:

```
@PostMapping("/api/chat")
public ChatBotResponse askQuestion(@RequestBody ChatBotRequest chatBotRequ\
est) {

    String answer = vertexAiGeminiChatModel.call(chatBotRequest.question()\
);
    return new ChatBotResponse(chatBotRequest.question(), answer);
}
```

2.6 How to Test the Application

To test the API, start the Spring Boot application and execute the following curl command:

```
curl -X POST http://localhost:8080/api/chat \
     -H "Content-Type: application/json" \
     -d '{"question": "What's your name?"}'
```

Alternatively, you can utilize the Swagger UI for a comprehensive and interactive testing experience. Follow these steps:

1. Start the Spring Boot application.
2. Navigate to `http://localhost:8080/swagger-ui.html` in your browser.

© 2024 Muthukumaran Navaneethakrishnan

3. Locate the `/api/chat` endpoint within the Swagger UI.
4. Enter your request payload (e.g., `{"question": "What's your name?"}`) and execute the request.

You should receive a response similar to:

```
{
  "question": "What's your name?",
  "answer": "I am a chatbot built with Spring AI and Vertex AI."
}
```

Code The source code for the above section is available on github

By following these steps, you can successfully design, implement, and test a RESTful chat controller in your Spring Boot application, integrating it with GCP's Vertex AI to provide intelligent and responsive chat functionalities.

2.7 Configuring Chat Options for Vertex AI

This section covers how to customize chat interactions in our Spring application using `VertexAiGeminiChatOptions`. Configuring these options is essential for tailoring AI responses to specific requirements and managing the behavior of the chatbot.

2.7.1 Key Configuration Options

We will focus on two primary options:

- **`MaxOutputTokens`**: This sets the maximum number of tokens (words and parts of words) that the AI model can generate in a response. Limiting

© 2024 Muthukumaran Navaneethakrishnan

the token count ensures that responses are concise and relevant, which helps manage verbosity. Each token is approximately four characters long, equating to about 60-80 words for 100 tokens.

- **`Temperature`**: This determines the randomness of AI responses. A lower value results in more predictable responses, while a higher value increases creativity. A temperature setting of 0.5 strikes a balance, allowing for reliable yet dynamic answers.

© 2024 Muthukumaran Navaneethakrishnan

2.7.2 Implementing `ChatOptions` in Chat Interaction

Chat configuration options are encapsulated within the `ChatOptions` object, which guides the AI to produce responses that are aligned with the user's input:

Building `ChatOptions`:

```
ChatOptions options = VertexAiGeminiChatOptions.builder()
    .maxOutputTokens(10)
    .temperature(0.5)
    .build();
```

Creating a `Prompt` with Chat Options:

```
Prompt prompt = new Prompt(messages, options);
```

Calling the Chat Client: Pass the configured prompt to the `VertexAiGeminiChatModel`

```
ChatResponse chatResponse = vertexAiGeminiChatModel.call(prompt);

String answer = chatResponse.getResult().getOutput().getText();
```

This method sends the user's question to Vertex AI and receives a tailored response based on the configured chat options. Note that the response is now an object, and the answer needs to be manually extracted from this object.

© 2024 Muthukumaran Navaneethakrishnan

2.7.3 Integrating Chat Options in the API Endpoint

Now, incorporate the chat options into the `ChatBotController` to effectively manage user requests:

```
@PostMapping("/api/chat")
public ChatBotResponse askQuestion(@RequestBody ChatBotRequest chatBotRequ\
est) {

    String question = chatBotRequest.question();

    // Setup options
    ChatOptions options = VertexAiGeminiChatOptions.builder()
            .maxOutputTokens(100)
            .temperature(0.5)
            .build();
    Prompt prompt = new Prompt(question, options);

    // Invoke Gemini client
    ChatResponse chatResponse = vertexAiGeminiChatModel.call(prompt);

    // extract answer
    String answer = chatResponse.getResult().getOutput().getText();

    // return response
    return new ChatBotResponse(chatBotRequest.question(), answer);
}
```

This configuration allows the Spring Boot application to fully leverage Vertex AI's capabilities, ensuring nuanced and well-configured AI responses for each interaction.

© 2024 Muthukumaran Navaneethakrishnan

2.8 How to Test the Application

To ensure the functionality of the updated chatbot, start the Spring Boot application and follow these steps to test the API using the Swagger UI:

1. **Access the Swagger UI**: Navigate to `http://localhost:8080/swagger-ui.html` in your web browser.
2. **Send a POST request**: Select the `/api/chat` endpoint, and use the following JSON payload in the Swagger UI's interactive interface:

```
POST http://localhost:8080/api/chat
content-type: application/json

{
  "question": "write a short story in simple words about freedom"

}
```

The expected result should look something like this:

```
{
  "question": "write a short story in simple words about freedom",
  "answer": "In a distant land, nestled amidst towering mountains and spar\
kling rivers ...."
}
```

This response, truncated due to the `MaxOutputTokens` setting, demonstrates how you can control the verbosity of AI responses. Experiment with different `Temperature` settings to see how the AI's creativity varies

> **Code** The source code for the above section is available on github

© 2024 Muthukumaran Navaneethakrishnan

2.9 Understanding API Interaction with Vertex AI

When you invoke the `vertexAiGeminiChatModel.call(prompt)` method in your Spring Boot application, it triggers an API call to Google Cloud's Vertex AI service.

2.9.1 The API Request to Vertex AI

When you invoke `vertexAiGeminiChatModel.call(prompt);`, it sends an API request to Vertex AI like this:

```
POST https://us-central1-aiplatform.googleapis.com/v1/projects/myspringai/\
locations/us-central1/publishers/google/models/gemini-1.5-flash:generateCo\
ntent
Authorization: Bearer {{gcloud_token}}
Content-Type: application/json

{
  "contents": [{
    "role": "user",
    "parts": [
      {
        "text": "write a short story in simple words about freedom"
      }
    ]
  }]
}
```

Endpoint and Authorization: The request is sent to your project's specific Vertex AI endpoint, using the `gemini-1.5-flash` model in the `us-central1` region. It includes an `Authorization` header with your `gcloud_token`, obtained via:

© 2024 Muthukumaran Navaneethakrishnan

```
gcloud auth application-default print-access-token
```

Payload: The body of the request contains the user's prompt formatted as a JSON object:

- **`contents`**: An array that encapsulates the messages within the conversation.
  - **`role`**: Designates the sender of the message; in this case, set to `"user"` for inputs provided by the user.
  - **`parts`**: An array within each message detailing the content.
  - **`text`**: The actual content of the user's prompt.

This structure informs Vertex AI that the message is from the user and includes the text that needs processing. The `parts` array is flexible, allowing for the message to be divided into several segments if necessary.

2.9.2 API Response from Vertex AI

Vertex AI processes the request and returns a response similar to this:

```
{
  "candidates": [
    {
      "content": {
        "role": "model",
        "parts": [
          {
            "text": "In a distant land, nestled amidst towering mountains \
and sparkling rivers..."
          }
        ]
      },
      "finishReason": "STOP",
```

© 2024 Muthukumaran Navaneethakrishnan

```
      "safetyRatings": [...],
      "avgLogprobs": -0.53441956718012973
    }
  ],
  "usageMetadata": {
    "promptTokenCount": 9,
    "candidatesTokenCount": 265,
    "totalTokenCount": 274
  },
  "modelVersion": "gemini-1.5-flash"
}
```

Here's what each part means:

- **`candidates`**: An array of responses generated by the AI. Typically, you'll use the first candidate.
  - **`content`**:
    - **`role`**: Indicates the sender of the message. Here, `"model"` means it's the AI's response.
    - **`parts`**: Contains the generated text.
      - **`text`**: The actual response from the AI.
  - **`finishReason`**: Explains why the AI stopped generating text. `"STOP"` means it completed as expected.
  - **`avgLogprobs`**: The average log probability of the generated tokens, reflecting confidence.
- **`usageMetadata`**:
  - **`promptTokenCount`**: Tokens used in your input.
  - **`candidatesTokenCount`**: Tokens used in the AI's response.
  - **`totalTokenCount`**: Total tokens used in the interaction.
- **`modelVersion`**: The specific AI model used.

© 2024 Muthukumaran Navaneethakrishnan

```
String answer = chatResponse.getResult().getOutput().getText();
```

This retrieves the text from the first candidate's content.Understanding this response structure allows you to effectively map the data to your application's classes and manage additional features such as safety ratings or token usage.

> The main distinction with GCP Vertex AI libraries is that they use gRPC instead of REST APIs for communication.

2.10 How to Deploy in GCP Cloud Run

Deploying your Spring Boot application to GCP Cloud Run involves a straightforward process that starts right from your project's root directory.

2.10.1 Step-by-Step Deployment Guide

1. Create the `project.toml` File:

- Since our application utilizes Java 17, we must ensure that Cloud Run builds it using the correct Java version. Create a `project.toml` file in your project's root folder and add the following configuration:

```
[[build.env]]
    name = "GOOGLE_RUNTIME_VERSION"
    value = "17"
```

- This file configures the Cloud Run build environment to use Java 17, aligning with our application's requirements.

Quick Fact:

© 2024 Muthukumaran Navaneethakrishnan

The `project.toml` file is essential in customizing the build settings for Cloud Run deployments. It specifies critical configurations like the runtime environment. In Java-based projects for GCP, this file instructs Cloud Run to utilize a specified Java version. Our example uses Java 17, aligning the build process with the application's requirements

2. Initiate the Deployment:

- Execute the following command in your terminal:

```
gcloud run deploy
```

3. Verify Source Code Location:

- During the deployment, `gcloud` will inquire about the source code location. Since we are executing from the project root, simply confirm the default location when prompted.

4.Provide Deployment Details:

- You will be asked a series of questions to configure your deployment:
  - **Service Name**: Assign a name to your Cloud Run service.
    * **Container Registry**: Choose whether to create a new container registry or use an existing one.
    * **Allow Unauthenticated Access**: Opt for 'yes' to make your service publicly accessible on the internet, allowing easy access without authentication.

© 2024 Muthukumaran Navaneethakrishnan

5. **Get Your Service URL**:

   - Once deployment completes, Cloud Run provides a URL where your application is now hosted, marking the transition from local development to a live service.

6. **Validate Your Deployment**:

   - Test the deployment using the service URL to ensure your application functions as expected. You can substitute the local testing URL with the Cloud Run service URL in any testing tool like curl or Postman.

And that's it! Your Spring Boot application, configured for Java 17, should now be up and running on Cloud Run.

© 2024 Muthukumaran Navaneethakrishnan

2.11 Review Questions

1 When deploying a Spring Boot application to Cloud Run that uses Java 17, what is the purpose of creating a `project.toml` file?

```
a) To specify the version of Java used for building the application.
b) To define the application's routes and endpoints.
c) To configure the database connections.
d) To set up application-specific environment variables.
```

2. In the context of deploying to Cloud Run, what does setting 'Allow Unauthenticated Access' to 'yes' do?

```
a) It enables the application to connect to GCP services without creden\
tials.
b) It makes your service publicly accessible on the internet.
c) It disables all security features of the application.
d) It allows users to bypass login screens in the application.
```

© 2024 Muthukumaran Navaneethakrishnan

2.12 Answers to Review Questions

Here are the answers to the review questions from the previous section:

1. Answer: a) To specify the version of Java used for building the application.

- Explanation: The `project.toml` file is used in Cloud Run deployments to specify certain build configurations, including the version of Java to be used. In this case, creating this file with the specified content informs GCP Cloud Run to build the application using Java 17.

2. Answer: b) It makes your service publicly accessible on the internet.

- Explanation: Setting 'Allow Unauthenticated Access' to 'yes' during the deployment process on Cloud Run makes the service publicly accessible on the internet. This means that anyone can access the service without needing authentication, which is particularly important for services meant to be available to the general public.

© 2024 Muthukumaran Navaneethakrishnan

2.13 What We Learned

By the end of this chapter, we've accomplished quite a bit. Here's a quick recap of what we've learned:

- **GCP Setup**: We started by creating a new GCP project and enabling Vertex AI, setting the foundation for our AI-powered application.
- **gcloud CLI Configuration**: We installed and configured the gcloud CLI, connecting our local development environment to the GCP project.
- **Spring Boot Application Creation**: We then stepped into the world of Spring Boot, creating a new project with essential dependencies for our AI application.
- **Designing a Chat Controller**: Our focus shifted to designing a RestController, crucial for handling user interactions within our application.
- **Vertex AI Integration**: We integrated Vertex AI into our Spring Boot application, enabling us to process and respond to user queries with AI-driven insights.
- **Application Deployment**: Finally, we deployed our application to Cloud Run, bringing our project to life and making it accessible for real-world testing and use.

© 2024 Muthukumaran Navaneethakrishnan

Chapter 3: Building Context-Aware Chatbots with Spring AI: Memory Management and Conversational Continuity

In this chapter, we tackle the challenge of building a chatbot that can maintain context throughout a conversation. The ability to remember user information across multiple interactions is crucial for creating a natural and engaging chatbot experience. Without this feature, chatbots can appear disconnected and impersonal, which ultimately limits their usefulness. This chapter focuses on enhancing our chatbot endpoint to support contextual conversations, enabling more personalized interactions with users.

3.1 Chapter Highlights

- **Challenges with Contextual Conversations**: We explore the limitations of traditional chatbots, which struggle to remember previous exchanges, and the significance of maintaining context in conversational AI.
- **Implementing Conversation History**: Learn how to use the `ChatBotHistoryManager` to maintain chat history effectively by storing messages in a list and managing them through session IDs.
- **Utilizing Spring AI Advisors**: Simplify chat history management using the `MessageChatMemoryAdvisor`, which streamlines handling request and response cycles by automatically managing conversation memory.

- **Endpoint Enhancements**: Update the chatbot's API to support session-aware, contextually rich interactions.
- **Resetting Session History**: Create a DELETE endpoint that allows users to clear session history, providing flexibility in managing conversations.

3.2 The Challenge of Contextual Conversation

Imagine having a conversation where each new sentence seems to forget the last one. Frustrating, right? That's the challenge with our current chatbot. While it's great for single questions, it struggles to remember past interactions. For instance, if you tell the bot your name, it won't recall it in your next chat. Here's how this looks:

3.2.1 Example: Traditional Chatbot Limitation

Question 1: Informing the Bot About Your Name

```
POST http://localhost:8080/api/chat
Content-Type: application/json

{
  "question": "My name is Mr. Rich, remember it."
}
```

Response:

© 2024 Muthukumaran Navaneethakrishnan

```
{
  "question": "My name is Mr. Rich, remember it.",
  "answer": "Nice to meet you, Mr. Rich."
}
```

Question 2: Asking the Bot to Recall Your Name

```
POST http://localhost:8080/api/chat
Content-Type: application/json

{
  "question": "Do you remember my name?"
}
```

Response:

```
{
  "question": "Do you remember my name?",
  "answer": "I do not have access to personal information, including names\
."
}
```

This forgetfulness is due to how Large Language Models (LLMs), like our Vertex AI, are built. They don't automatically remember previous conversations, which can lead to disjointed and unsatisfying interactions.

3.3 Understanding LLMs and Contextual Conversations

In this section, we will explore how Large Language Models (LLMs) handle contextual conversations and why this is crucial for a more engaging chatbot experience.

© 2024 Muthukumaran Navaneethakrishnan

3.3.1 How LLMs Handle Context

LLMs, like Vertex AI, manage context by maintaining entire conversation histories. This allows the chatbot to generate responses that consider previous interactions. However, without specific implementation, the chatbot may lose this context, resulting in responses that seem unaware of previous dialogue.

3.3.2 Why Context Matters

Contextual conversations are essential for creating engaging chatbots. Users often reference previous parts of the conversation or expect the chatbot to recall specific details. By enabling context, we create a more natural and intuitive user experience.

To maintain context in a conversation, Large Language Models (LLMs) like ours rely on three essential types of messages:

- **SystemMessage:** This is the initial setup message that defines the LLM's role.
- **UserMessage:** These are the queries or inputs sent by the user.
- **AssistantMessage:** These are the responses generated by the LLM.

It's crucial to include all previous messages when sending a new query to the LLM. This comprehensive approach enables the LLM to understand and respond effectively, keeping track of the ongoing conversation.

So the Endpoint should be designed to accept all the messages , something like this

© 2024 Muthukumaran Navaneethakrishnan

```
POST https://somedomain.com/some-endpoint
Content-Type: application/json

[
  {"role": "system", "content": "You are my personal assistant"},
  {"role": "user", "content": "My name is Mr. Rich, remember it"},
  {"role": "assistant", "content": "Noted, Mr. Rich."},
  {"role": "user", "content": "Do you remember my name?"}
]
```

Here, the conversation is tracked systematically, with the latest query being added to the end of the array. This format has become standard for LLMs to effectively manage and maintain conversation context.

3.4 Implementing Conversation History in Spring AI

In Spring AI, maintaining a conversation history is managed using a list of `Message` objects. These objects store messages from both the user and the assistant in chronological order. Let's look at how to implement this:

```
import org.springframework.ai.chat.messages.AssistantMessage;
import org.springframework.ai.chat.messages.Message;
import org.springframework.ai.chat.messages.UserMessage;

// Initialize the list to store conversation messages
List<Message> messages = new ArrayList<>();
// Adding System message from history
messages.add(new SystemMessage("You are my personal assistant"));
// Adding user message for history
messages.add(new UserMessage("My name is Mr Rich, remember it"));
// Adding assistant response for history
messages.add(new AssistantMessage("Nice to meet you Mr. Rich."));
// Adding another user message to test memory recall
messages.add(new UserMessage("Do you remember my name?"));
```

By storing and recalling conversation history, chatbots can provide responses that are relevant and personalized.

© 2024 Muthukumaran Navaneethakrishnan

Modifying the Vertex AI Interaction: Initially, our interaction with Vertex AI was straightforward, accepting a string as a question and returning a string as an answer:

```
String answer = vertexAiGeminiChatModel.call(question);
```

To incorporate the conversation context, we now use a `Prompt` object containing our list of `Message` objects. The method returns a `ChatResponse`, from which we extract the answer:

```
import org.springframework.ai.chat.model.ChatResponse;

List<Message> messages = new ArrayList<>();
messages.add(new SystemMessage("You are my personal assistant"));
messages.add(new UserMessage("My name is Mr Rich, remember it"));
messages.add(new AssistantMessage("Nice to meet you Mr. Rich."));
messages.add(new UserMessage("Do you remember my name?"));

Prompt prompt = new Prompt(messages);

ChatResponse chatResponse = vertexAiGeminiChatModel.call(prompt);
String answer = chatResponse.getResult().getOutput().getText();
```

This method ensures the response from Vertex AI considers the entire conversation, allowing for more context-aware interactions.

3.5 Managing ChatBot Conversations with Session IDs

Depending on clients (or frontends or react applications) to provide complete conversation histories is not always ideal. A more efficient and secure method involves managing chat histories on the server using session IDs.

Storing Chat History: Chat history for each session is stored in a `ConcurrentHashMap`:

© 2024 Muthukumaran Navaneethakrishnan

```
ConcurrentHashMap<String, List<Message>> chatBotHistory = new ConcurrentHa\
shMap<>();
```

Retrieving Chat History: To access the chat history for a specific session, implement `getChatHistory` method to retrieve from the the `ConcurrentHashMap`

```
public List<Message> getChatHistory(String sessionId) {
    return chatBotHistory.getOrDefault(sessionId, new ArrayList<>());
}
```

Check SessionId is New: To check if a session ID is new, use the following method. This can be useful for setting a `SystemMessage`

```
public boolean isNewSession(String sessionId) {
    return !chatBotHistory.containsKey(sessionId);
}
```

Add System Message to session: To add a new System Message , implement `addSystemMessage` as below :

```
public void addSystemMessage(String sessionId, String message) {
    List<Message> messages = new ArrayList<>();
    messages.add(new SystemMessage(message));
    chatBotHistory.put(sessionId, messages);
}
```

The `SystemMessage` should be the first one for any session, so we create a new list and set it as the first message.

Updating Chat History : After each interaction, update the chat history with the new messages:

© 2024 Muthukumaran Navaneethakrishnan

```
public void addChatHistory(String sessionId, String question, String answe\
r) {
    var existingHistory = getChatHistory(sessionId);
    var messages = new ArrayList<>(existingHistory);
    messages.add(new UserMessage(question));
    messages.add(new AssistantMessage(answer));
    chatBotHistory.put(sessionId, messages);
}
```

3.5.1 Implementing `ChatBotHistoryManager`

To efficiently handle the history management, we introduce `ChatBotHistoryManager` as a service:

```
@Service
public class ChatBotHistoryManager {

    ConcurrentHashMap<String, List<Message>> chatBotHistory = new Concurre\
ntHashMap<>();

    public void addChatHistory(String sessionId, String question, String a\
nswer) {
        var existingHistory = getChatHistory(sessionId);
        var messages = new ArrayList<>(existingHistory);
        messages.add(new UserMessage(question));
        messages.add(new AssistantMessage(answer));
        chatBotHistory.put(sessionId, messages);
    }

    public List<Message> getChatHistory(String sessionId) {
        return chatBotHistory.getOrDefault(sessionId, new ArrayList<>());
    }

    public boolean isNewSession(String sessionId) {
        return !chatBotHistory.containsKey(sessionId);
    }

    public void addSystemMessage(String sessionId, String message) {
    List<Message> messages = new ArrayList<>();
    messages.add(new SystemMessage(message));
    chatBotHistory.put(sessionId, messages);
```

© 2024 Muthukumaran Navaneethakrishnan

```
}

}
```

With `ChatBotHistoryManager`, the chat history for each session is effectively managed, ensuring a smooth and context-aware conversation experience.

3.6 Updating the ChatBot Endpoint for Conversational Context

Now, let's begin the process of updating our existing endpoint. This update is crucial to ensure that our chatbot responds with context awareness. By making this change, the chatbot will be able to maintain the continuity of the conversation, taking into account the entire dialogue history when replying. This enhancement marks a significant step towards more intelligent and personalized interactions, transforming our chatbot into a truly conversational partner

3.6.1 Modifying the ChatBot Request Model Class

Initially, our `ChatBotRequest` class contained only a single property, `question`:

```
public record ChatBotRequest(String question) {
}
```

To incorporate session tracking, we'll add a `sessionId` property. This allows us to maintain a chat history:

© 2024 Muthukumaran Navaneethakrishnan

```
public record ChatBotRequest(String question, String sessionId) {
}
```

The `sessionId` represents the identifier of the chat session, which correlates with the history managed by `ChatBotHistoryManager`.

3.6.2 Integrating ChatBotHistoryManager

Add a dependency for `ChatBotHistoryManager` in the `ChatBotController` class through constructor injection:

```
private final VertexAiGeminiChatModel vertexAiGeminiChatModel;
private final ChatBotHistoryManager chatBotHistoryManager;

public ChatBotController(VertexAiGeminiChatModel vertexAiGeminiChatModel, \
ChatBotHistoryManager chatBotHistoryManager) {
    this.vertexAiGeminiChatModel = vertexAiGeminiChatModel;
    this.chatBotHistoryManager = chatBotHistoryManager;
}
```

3.6.3 Create System Prompt for New Session

Update the chatBotHistoryManager to add a System Message when session is new

```
if(chatBotHistoryManager.isNewSession(sessionId))
    chatBotHistoryManager.addSystemMessage(sessionId, "You are my personal\
 assistant");
```

3.6.4 Building the Prompt

Construct the prompt by merging existing chat history with the new user message, if existing chat history is empty add a system Message:

© 2024 Muthukumaran Navaneethakrishnan

```
var chatHistory = chatBotHistoryManager.getChatHistory(sessionId);
var messages = new ArrayList<>(chatHistory);
messages.add(new UserMessage(question));
```

Then, use these messages to create the prompt:

```
Prompt prompt = new Prompt(messages);
```

3.6.5 Invoking Vertex AI with Context

Make the Vertex AI call with the constructed prompt:

```
ChatResponse chatResponse = vertexAiGeminiChatModel.call(prompt);
```

Retrieve the response from the chat client:

```
String answer = chatResponse.getResult().getOutput().getText();```

### 3.6.6 Updating Chat History

Add the new question and its answer to the chat history:

```java
chatBotHistoryManager.addChatHistory(sessionId, question, answer);
```

3.6.7 The Final Endpoint Implementation

The fully implemented endpoint now incorporates conversation history:

© 2024 Muthukumaran Navaneethakrishnan
```

```
    @PostMapping("/api/chat")
    public ChatBotResponse askQuestion(@RequestBody ChatBotRequest chatBot\
Request) {
        String sessionId = chatBotRequest.sessionId();
        String question = chatBotRequest.question();

        // Setup System Message for new Session
        if(chatBotHistoryManager.isNewSession(sessionId))
            chatBotHistoryManager.addSystemMessage(sessionId, "You are my \
personal assistant");

        //Combine chat history with the new question
        var chatHistory = chatBotHistoryManager.getChatHistory(sessionId);
        var messages = new ArrayList<>(chatHistory);
        messages.add(new UserMessage(question));

        // create a prompt
        Prompt prompt = new Prompt(messages);
        // call the chat client
        ChatResponse chatResponse = vertexAiGeminiChatModel.call(prompt);

        // get the answer
        String answer = chatResponse.getResult().getOutput().getText();

        // add the chat history to our local cache
        chatBotHistoryManager.addChatHistory(sessionId, question, answer);
        return new ChatBotResponse(question, answer);
    }
```

This approach effectively manages conversation history in Spring AI, using a cache with `ConcurrentHashMap`. Note that for production environments, more robust solutions like Redis or similar open-source alternatives should be used for managing conversation history.

3.7 Testing the Enhanced ChatBot with Session History

With the introduction of session IDs in our chatbot, we can now maintain a conversation history, overcoming the previous challenge. Let's test this improved functionality:

© 2024 Muthukumaran Navaneethakrishnan

Question 1: Informing the Bot About Your Name

```
POST http://localhost:8080/api/chat
Content-Type: application/json

{
  "question": "My name is Mr Rich, remember it",
  "sessionId": "cljXX1"
}
```

Response:

```
{
  "question": "My name is Mr Rich, remember it",
  "answer": "Hello Mr Rich, I will remember your name."
}
```

Question 2: Asking the Bot to Recall Your Name

```
POST http://localhost:8080/api/chat
Content-Type: application/json

{
  "question": "Do you remember my name?",
  "sessionId": "cljXX1"
}
```

Response:

```
{
  "question": "Do you remember my name?",
  "answer": "Yes, your name is Mr Rich."
}
```

Now, as demonstrated, the chatbot responds accurately and contextually, remembering previous interactions within the same session.

Code The source code for the above section is available on github

© 2024 Muthukumaran Navaneethakrishnan

3.8 Simplifying Chat History Management with Advisors

So far, we have implemented a service class to maintain chat history as a list of messages. Fortunately, Spring AI provides a similar feature called Advisors, which can wrap requests and responses, similar to Spring AOP. These advisors update the requests when they are sent and also handle responses when received, allowing custom operations to be performed.

Previously, we invoked methods in `ChatBotHistoryManager` before and after calling the chat client to manage the chat history. Instead, we can use an advisor called `MessageChatMemoryAdvisor`, which will manage messages in memory automatically. This advisor requires a `ChatMemory` interface, and for this example, we will use `InMemoryChatMemory`. However, you can also use persistent options like Redis or your database by implementing the `ChatMemory` interface.

First, we will create a `ChatMemory` bean, defined in any configuration class or in the main application class.

```
import org.springframework.context.annotation.Bean;
import org.springframework.ai.chat.memory.ChatMemory;
import org.springframework.ai.chat.memory.InMemoryChatMemory

@Bean
ChatMemory chatMemory() {
    return new InMemoryChatMemory();
}
```

Next, in the `ChatBotController` class, remove the `chatBotHistoryManager` dependency and add properties `inMemoryChatClient` of type `ChatClient` and `chatMemory` of type `ChatMemory`:

© 2024 Muthukumaran Navaneethakrishnan

```
import org.springframework.ai.chat.client.ChatClient;

@RestController
@Slf4j
public class ChatBotController {

private  final  ChatClient inMemoryChatClient;
private  final  ChatMemory chatMemory;
//... Other stuff

}
```

We can create a `MessageChatMemoryAdvisor` from `chatMemory` like this:

```
MessageChatMemoryAdvisor messageChatMemoryAdvisor = MessageChatMemoryAdvis\
or.builder(chatMemory).build();
```

Then, create an `inMemoryChatClient` by setting the default advisors:

```
inMemoryChatClient = ChatClient.builder(vertexAiGeminiChatModel)
        .defaultAdvisors(messageChatMemoryAdvisor)
        .build();
```

Now The Constructor will look like this

```
public ChatBotController(VertexAiGeminiChatModel vertexAiGeminiChatModel ,\
ChatMemory chatMemory) {

 this.chatMemory = chatMemory;
 var messageChatMemoryAdvisor = new MessageChatMemoryAdvisor(chatMemory);
 inMemoryChatClient = ChatClient.builder(vertexAiGeminiChatModel)
                            .defaultAdvisors(messageChatMemoryAdvisor)
                            .build();
}
```

In the endpoint, simply retrieve the `sessionId` from `ChatBotRequest`:

© 2024 Muthukumaran Navaneethakrishnan

```
@PostMapping("/api/chat")
public ChatBotResponse askQuestion(@RequestBody ChatBotRequest chatBotRequ\
est) {
    String sessionId = chatBotRequest.sessionId();
    String question = chatBotRequest.question();
    // To be Implemented ...
}
```

The `MessageChatMemoryAdvisor` uses a default parameter called `CHAT_MEMORY_CONVERSATION_ID_KEY` to manage conversation sessions. To make it specific to our session, we replace this parameter with our session ID. When configuring the advisors, we set up the system prompt, add the user's question, update the advisor with the session ID, and then create the chat request:

```
import static org.springframework.ai.chat.memory.ChatMemory.CONVERSATION_I\
D;

var chatRequest = inMemoryChatClient
        .prompt()
        .system("You are my personal assistant")
        .user(question)
        .advisors(a -> a.param(CONVERSATION_ID, sessionId));
```

Now, get the response by invoking the `call` method:

```
        var chatResponse = chatRequest.call().chatResponse();
        String answer = chatResponse.getResult().getOutput().getText();```

The updated endpoint will look like this:

```java

    @PostMapping("/api/chat")
    public ChatBotResponse askQuestion(@RequestBody ChatBotRequest chatBot\
Request) {
        String sessionId = chatBotRequest.sessionId();
        String question = chatBotRequest.question();


        var chatRequest = inMemoryChatClient
```

© 2024 Muthukumaran Navaneethakrishnan
```

```
                .prompt()
                .system("You are my personal assistant")
                .user(question)
                .advisors(a -> a.param(CHAT_MEMORY_CONVERSATION_ID_KEY, se\
ssionId));


        var chatResponse = chatRequest.call().chatResponse();
        String answer = chatResponse.getResult().getOutput().getText();    \
     return new ChatBotResponse(question, answer);
      }
```

You can test the updated implementation in the same way as before. The new approach is more concise and leverages Spring AI's built-in capabilities for managing conversation history.

> **Code** The source code for the above section is available on github

3.9 Creating a DELETE Endpoint to Reset Chat Session History

We will create an endpoint to delete the conversation history, allowing users to reset the session and start fresh. In this example, we're using `MessageWindowChatMemory` as our `ChatMemory` implementation. Let's first define it in our application:

```
@Bean
ChatMemory chatMemory() {
    return MessageWindowChatMemory.builder().build();
}
```

Next, we'll add a DELETE endpoint to clear the conversation history by invoking the `chatMemory.clear` method:

© 2024 Muthukumaran Navaneethakrishnan

```
@DeleteMapping("/api/chat/{sessionId}")
public ResponseEntity<Void> deleteChatHistory(@PathVariable String session\
Id) {
    chatMemory.clear(sessionId);
    return ResponseEntity.noContent().build();
}
```

3.9.1 Testing the Endpoint

1. **Initial Context Setup:**

   - Start by establishing a conversation context where the chatbot is informed of the user's name:

     ```
     POST http://localhost:8080/api/chat
     Content-Type: application/json

     {
       "question": "My name is Mr. Rich, remember it",
       "sessionId": "cljXX1"
     }
     ```

   - The expected response should acknowledge the user's name:

     ```
     {
       "question": "My name is Mr. Rich, remember it",
       "answer": "Hello Mr. Rich, I will remember your name."
     }
     ```

2. **Clear the Session History:**

   - Use the `DELETE` endpoint to clear the history for the session:

     ```
     DELETE http://localhost:8080/api/chat/cljXX1
     ```

3. **Verify Chatbot's Memory Reset:**

   - After clearing the session history, test if the chatbot has forgotten the context:

© 2024 Muthukumaran Navaneethakrishnan

```
POST http://localhost:8080/api/chat
Content-Type: application/json

{
  "question": "Do you remember my name?",
  "sessionId": "cljXX1"
}
```

- Expect a response indicating the chatbot doesn't remember the user's name, such as:

```
{
  "question": "Do you remember my name?",
  "answer": "I'm sorry, I don't recall your name."
}
```

Code The source code for the above section is available on github

3.10 Understanding System Prompts in Chatbot Responses

In developing advanced chatbots, it's often beneficial to guide the bot's responses into specific formats. System prompts is helpful in achieving this. Imagine a scenario where you want your chatbot to do a factCheck , It should respond with only 'True', 'False', or 'Not sure' to any statement. Let's see how system prompts can enable this functionality.

3.10.0.1 Example Scenario: True/False/Not Sure Responses

Initial Request without System Prompt:

Consider the question: "Rich Hickey Created Clojure". Without a system prompt, the chatbot might provide a detailed answer:

© 2024 Muthukumaran Navaneethakrishnan

```
POST /api/chat
Content-Type: application/json

{
  "question": "Rich Hickey Created Clojure",
  "sessionId": "cljXXx1"
}
```

Typical Response:

```
{
  "question": "Rich Hickey Created Clojure",
  "answer": "That's not entirely accurate. Rich Hickey is the primary crea\
tor of Clojure..."
}
```

Modifying the Request with a Specific Prompt: To constrain the chatbot to respond with ‘True’, ‘False’, or ‘Not sure’, modify the question:

```
POST /api/chat
Content-Type: application/json

{
  "question": "Rich Hickey Created Clojure, you should reply only either T\
rue or False or Not sure",
  "sessionId": "cljXXx1"
}
```

Constrained Response:

```
{
  "question": "Rich Hickey Created Clojure, you should reply only either T\
rue or False or Not sure",
  "answer": "True"
}
```

© 2024 Muthukumaran Navaneethakrishnan

3.10.1 Implementing System Prompts for Uniform Response Style

To consistently receive responses in this True/False/Not sure format, use a system prompt at the beginning of each session. For example:

```
if(chatBotHistoryManager.isNewSession(sessionId))
    chatBotHistoryManager.addSystemMessage(sessionId, "You should only ans\
wer true or false or not sure going forward.");
```

Let's create a specialized endpoint, `/api/fact-check`, to handle these specific types of queries. This endpoint will ensure that all responses adhere to the format set by our system prompt

```
@PostMapping("/api/fact-check")
public ChatBotResponse factCheck(@RequestBody ChatBotRequest chatBotReques\
t) {
    String sessionId = chatBotRequest.sessionId();
    String question = chatBotRequest.question();


    if (chatBotHistoryManager.isNewSession(sessionId))
        chatBotHistoryManager.addSystemMessage
        (sessionId,
        "You are my personal assistant, you should only answer true or fal\
se or not sure going forward.");

    //Combine chat history with the new question
    var chatHistory = chatBotHistoryManager.getChatHistory(sessionId);
    var messages = new ArrayList<>(chatHistory);


    messages.add(new UserMessage(question));

    // create a prompt
    Prompt prompt = new Prompt(messages);


    // call the chat client
    ChatResponse chatResponse = vertexAiGeminiChatModel.call(prompt);
```

© 2024 Muthukumaran Navaneethakrishnan

```
    // get the answer
    String answer = chatResponse.getResult().getOutput().getText();

    // add the chat history to our local cache
    chatBotHistoryManager.addChatHistory(sessionId, question, answer);

    return new ChatBotResponse(question, answer);

}
```

Now, any question asked via this endpoint will be answered strictly within the True/False/Not sure format:

Question:

```
POST /api/fact-check
Content-Type: application/json

{
  "question": "The sum of 1 + 2 = 5",
  "sessionId": "cljXX1"
}
```

Response:

```
{
  "question": "The sum of 1 + 2 = 5",
  "answer": "False"
}
```

With this setup, all responses from the `/api/fact-check` endpoint will align with our specified format, thanks to the initial system prompt. This approach is particularly useful for chatbots functioning in scenarios where binary or straightforward answers are preferred.

Code The source code for the above section is available on github

© 2024 Muthukumaran Navaneethakrishnan

3.10.2 Customizing System Prompts

Customize the system prompt to suit your chatbot's specific needs. This approach is particularly useful in scenarios like automated FAQs, quick decision-making tools, or streamlined customer service interactions, where straightforward answers are preferred.

In summary, system prompts offer a powerful way to control the style and format of your chatbot's responses, ensuring consistency and relevance in its interactions.

© 2024 Muthukumaran Navaneethakrishnan

3.11 What We Learned

- How to update the chatbot endpoint to maintain conversation context using session IDs.
- How to modify the request model to support session tracking, enabling the chatbot to remember previous interactions.
- How to use `ChatBotHistoryManager` to manage and store conversation history efficiently.
- How to implement system prompts to set the context for new sessions.
- How to use Spring AI's `MessageChatMemoryAdvisor` to simplify chat history management without manually invoking history methods.
- How to create a DELETE endpoint to clear session history, allowing users to reset their conversations.

© 2024 Muthukumaran Navaneethakrishnan

Chapter 4: Creating Structural Data Bots

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.1 Chapter Highlights:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.2 Make Inventory to be queried by Natural Language

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.3 Update Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.3.1 Required Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.4 Setting up an SQL database in GCP Cloud SQL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.4.1 Create a Database in GCP

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.4.2 Import and execute the SQL file

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.4.3 Setting up project

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.5.1 Create Model class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.5.2 Create a Repository class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

4.5.3 Create an Endpoint to Test

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.5.4 Test The endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.6 Getting Ready for Natural Language Processing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.7 From Natural Queries to SQL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.7.1 Organizing Prompts in the Resources Folder

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.7.2 Building the SQL Generator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

4.7.3 Implementing the Service Class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.7.5 Fine-Tuning Version 2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.8 JSON Conversion of SQL Results

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.8.1 Testing and Output

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.9 SQL to Natural Language Response

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

4.10 Bringing It All Together: The Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.11 Testing the Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.12 Exercise: Implement it using CSV instead of JSON

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.12.1 Objective

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.12.2 Task Description

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

4.12.3 Implementation Guidance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.12.4 Testing the Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.13 Improving SQL Query Flexibility for User Queries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.14 Handling multiple questions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.14.1 Adjusting the Prompt for Enhanced Functionality

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

4.14.1.1 Modifications include:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

4.14.2 Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

4.15 What We Learned

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

Chapter 5: LLM Tool Calling with Spring AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.1 Chapter Highlights:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.2 Understanding LLM Tool Calls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.2.1 What is Tool Calling?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.2.2 The Tool Calling Solution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.2.3 How Tool Calling Works

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.2.4 Implementation Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

5.3 Implementing Tool Calling with REST API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.3.1 Define Available Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.3.2 Receive Tool Call Request

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.3.3 Execute Function and Return Results

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.3.4 Receive Natural Language Response

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

5.4 Java Implementation with Direct API Calls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.4.1 Service Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.4.2 Key Helper Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.4.3 Main Implementation Logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.4.4 Testing the Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

5.5 Handling Multiple Tool Calls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.5.1 Implementation Challenges

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.6 Simplifying with Spring AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.6.1 Understanding Spring AI Tool Annotations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.6.2 Integrating Tools in Controllers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.6.3 Benefits of Spring AI's Approach

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

5.7 Advanced Tool Calling: Sequential Tool Routing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.7.1 Understanding Sequential vs. Parallel Tool Calls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.7.2 Creating SQL Tools with Routing Logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.7.3 Controller Implementation with Routing Instructions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.7.4 Observing Sequential Tool Routing in Action

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.7.5 The Power of Sequential Tool Routing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

5.7.6 Behind the Scenes: Raw API Calls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.8 Few Words

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

5.9 What We Learned

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

Chapter 6: Building Chatbots with Text and PDF Files

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.1 Chapter Highlights:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.2 Use Case: Answering Questions About Vacuum Cleaners based on text file

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.2.1 Setting Up System Message

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.2.2 Making File Ready for Chat

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.2.3 Formulating the User Message with Media

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.2.4 Querying the LLM Client

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.2.5 Endpoint implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.2.6 Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

6.3 Use Case: Answering Questions About Laptop based on pdf manual

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.3.1 Setting Up System Message

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.3.2 Preparing the PDF File for Chat

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.3.3 Formulating the User Message with Media

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.3.4 Querying the LLM Client

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

6.3.5 Endpoint implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

6.3.6 Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

6.3.7 What We Learned:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

Chapter 7: Building Chatbots with Multimedia Capabilities

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.1 Chapter Highlights:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.2 Use Case: Answering Questions About Coupons from an Image

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.3 Invoking the Chatbot with Image Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.3.1 Setting Up System Message

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.3.2 Making Image Ready for Chat

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.3.3 Formulating the User Message with Image Input

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

7.3.4 Querying the LLM Client with Images

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.3.5 Endpoint Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.4 Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

7.5 Use Case: Answering Questions based on a Customer Care Audio

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.5.1 Setting Up System Message

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.5.2 Setting Up Media Objects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.5.3 Formulating the User Message with Audio

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.5.4 Querying the LLM Client with Audio

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

7.5.5 Endpoint Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.6 Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.7 Use Case: Answering Questions from a Vaccum Cleaner Advertisement video

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.8 Invoking the Chatbot with Video Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.8.1 Setting Up System Message

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

7.8.2 Making Video to be Ready for Chat

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.8.3 Formulating the User Message with Video Media

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.8.4 Querying the LLM Client with Video

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.8.5 Endpoint Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.9 Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

7.9.1 What We Learned:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

Chapter 8: Using RAG to Make LLMs Smarter with Internal Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.1 Integrating Internal Data with LLM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.2 Chapter Highlights:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.2.1 The Challenge of Specificity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.2.2 Bridging the Gap: Retrieval Augmented Generation (RAG)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.2.3 The Role of Embeddings and Vector Databases

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.3 How to Create Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.3.1 Embedding Models available in Vertex AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.3.2 Create embeddings from VertexAI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.3.3 The Embeddings Endpoint Definition

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.3.4 Setting Up Embeddings With Spring AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

8.3.5 Adding dependency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.3.6 Configuring the Embedding Model in `application.properties`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.3.7 Implement Endpoint to convert text to Embedding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.3.8 Testing the Embedding Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4 Vector Database

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.1 The Role of Postgres as a Vector Database

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

8.4.2 Setting Up Postgres PGVector in Spring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.3 Managing Unstructured Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.4 Inserting Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.5 Testing Insert Document

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.6 Inserting Multiple Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.7 Testing the Insertion of Multiple Documents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

8.4.8 Retrieving Relevant Data - Usecase

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.9 Retrieving Relevant Data - Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.10 Testing the Similarity Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.11 Refining the Retrieval Process with Limits and Maximum Distance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.11.1 Enhancing the SimilaritySearchRequest

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.11.2 Configuring the SearchRequest for limit

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

8.4.11.3 Configuring the SearchRequest for Maximum Distance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.11.4 Updating the retrieval endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.12 Testing the Enhanced Similarity Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.13 Organizing Departmental Data - Usecase

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.13.1 The Challenge of Data Segregation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.13.2 Organizing Unstructured Data by Department

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

8.4.14 Organizing Departmental Data - Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.15 Inserting Data with Department

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.16 Retrieving Data with Department

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.4.17 Testing the Similarity Search By department

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.5 Making Similar Documents as Conversational

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.5.1 Implementing Answers Based on Relevant Documents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

8.5.2 Testing the Conversational Search by Department

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.6 Exercise: Modify Chatbot to Search All Records When Department Is Not Specified

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.6.1 Objective

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.6.4 Implementation Guidance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

8.6.5 Testing the Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

8.6.6 What We Learned:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

Chapter 9: Building Internal Knowledge based assistant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.1 Chapter Highlights:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.2 Integrating Google Cloud Storage Bucket

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.2.0.1 Setting Up the `StorageBucketService` Class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.2.0.2 Creating the Upload Method

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.2.0.3 Creating the getFile Method

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.2.0.4 Creating the deleteFile Method

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.2.1 Testing File Upload and Download

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3 Building a Chatbot with Internal Documents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.1 Workflow Overview

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.1.1 Configuring the Vector Store

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

9.3.1.2 Create an Endpoint with Dependency Injection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.2 Extract Text from Files

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.3 Create an Upload Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.4 Testing the Document Upload Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.5 Uploading Larger Documents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.6 Interacting with Internal Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

9.3.6.1 Constructing the Query Expression

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.6.1.1 Adding Extension Criteria

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.6.1.2 Adding Category Criteria

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.6.1.3 Finalizing the Query Expression

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.6.1.4 Creating the Search Request

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.6.1.5 Creating Chat Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

9.3.7 Testing the Internal Data end point

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.8 Streamlining Chat Interactions without Advisors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.9 Extract and Prepare Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.9.1 Fetch Contextual Documents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.9.2 Compile the Context History

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.9.3 Set Up Prompts for the AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

9.3.9.4 Initialize the Chat Client

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.3.10 Detailed Endpoint Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.4 Adding File References on Chat

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.4.1 Creating a File Download Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.4.2 Testing the File Download Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.4.3 Enhancing Chat Interactions with Document References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

9.4.3.1 Key Enhancements:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.4.3.2 Building Context with URLs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.4.3.3 Adjusting the Chat Prompt

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.4.3.4 Enhanced Response Output

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.4.4 Testing Chat with Document Reference endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

9.4.5 What We Learned:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

Chapter 10: PDF Documents & Image Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.1 Chapter Highlights:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.1 Saving PDF Document Pages as Image Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.1.1 Updating the Configuration for Image Embedding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.1.2 Transforming PDF Resources into Image Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.1.2.1 Detailed Process of Converting PDFs to Images

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.1.3 Building Component for PDF Embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.1.4 Processing Single Pdf Image

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.1.5 Implementing processPdfToDocuments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.1.6 Building the Upload Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.1.7 Testing the Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

10.2 Updating Chat Endpoint to use Image embeddings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.2.1 Modifications for Image Embedding Support

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.2.2 Creating a Method to Get Media

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.2.3 Update the Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.2.4 Testing the Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

10.2.5 What We Learned:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

Chapter 11: Accessing other models in GCP Model Garden

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

11.1 Chapter Highlights:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

11.1 Setting Up GCP

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

11.2 Setting Up Gradle and Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

11.3 Setting Up Authentication Tokens for Llama Requests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

11.3.1 Creating the Interceptor Class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

11.3.2 Adding the Authorization Header

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

11.3.3 Creating a Bean to Use Interceptor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

11.3.4 Creating an Endpoint to Use Llama

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

11.3.5 Testing the Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

11.3.6 What We Learned:

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.

© 2024 Muthukumaran Navaneethakrishnan

Chapter 12: Whats Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/springai.