CODE & CHAT

CHATBOTS COM SPRING BOOT E DIALOGFLOW

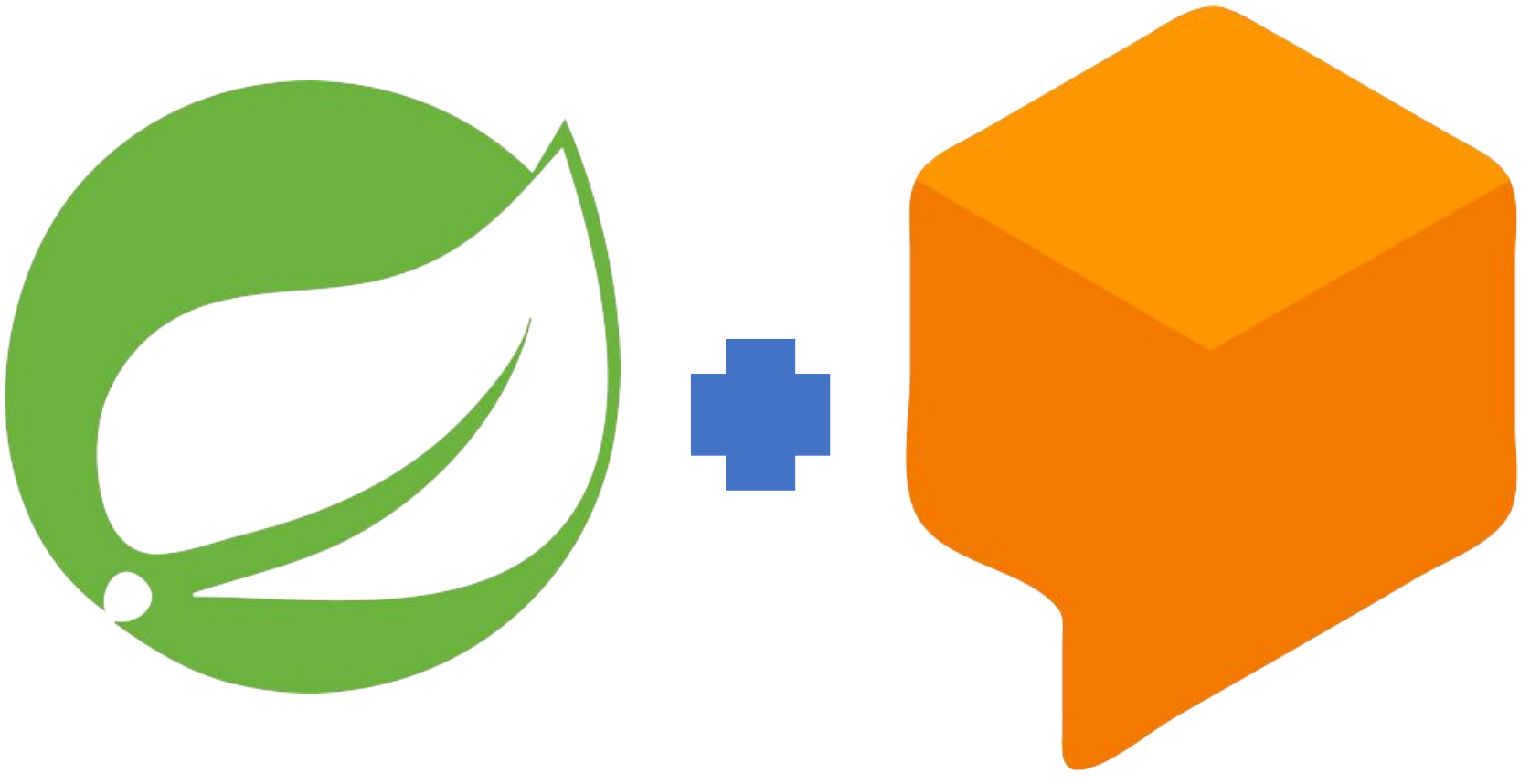
Dicas para Construir seu Próprio Assistente Digital

Aline de O. Machado

Boas vindas, Devs!

Chatbot usando Spring Boot e Dialogflow - Dicas para Construir seu Próprio Assistente Digital

Olá, pessoal! Estou muito animada em ter vocês aqui nessa jornada para dominar chatbots usando Spring Boot e Dialogflow. Este ebook é especialmente para vocês, programadores juniores, que querem construir algo incrível e funcional. Vamos trabalhar juntos e, ao final, vocês terão um assistente digital prontinho!



Sumário

Sumário

01

INTRODUÇÃO

Chatbots são ferramentas incríveis que podem automatizar tarefas, fornecer suporte e melhorar a experiência do usuário. Eles são úteis em diversos contextos, desde atendimento ao cliente até assistentes pessoais. Acredito que aprender a construir um chatbot irá aprimorar suas habilidades e abrir muitas portas na sua carreira.

1.1 Por que um desenvolvedor escolheria fazer um projeto de Chatbot usando Spring Boot e Dialogflow?

Escolher fazer um projeto de Chatbot usando Spring Boot e Dialogflow pode ser motivado por diversas razões, dependendo das necessidades específicas do desenvolvedor ou da equipe de desenvolvimento. Aqui estão algumas razões pelas quais vocês, devs, podem optar por essa combinação:

a. Robustez e Escalabilidade

Spring Boot é conhecido por sua robustez e capacidade de lidar com aplicações empresariais de grande escala. Se o chatbot faz parte de uma aplicação maior que já utiliza Spring Boot, integrá-lo pode ser uma escolha natural para manter a consistência tecnológica e aproveitar a infraestrutura existente.

b. Ecossistema Java

Compatibilidade com o Ecossistema Java: Empresas que já utilizam tecnologias baseadas em Java para suas aplicações podem achar mais conveniente continuar usando Spring Boot para o backend, evitando a necessidade de introduzir uma nova linguagem ou framework na pilha tecnológica.

c. Integração com Serviços Empresariais

Facilidade de Integração: Spring Boot facilita a integração com outros serviços e sistemas empresariais, como bancos de dados, serviços de autenticação e APIs internas. Isso é crucial para chatbots que precisam acessar informações de múltiplas fontes.

d. Segurança

Recursos de Segurança: Spring Boot fornece recursos robustos de segurança, incluindo autenticação e autorização, que podem ser importantes para chatbots que manipulam dados sensíveis ou realizam transações.

e. Capacidades de NLP

Dialogflow oferece capacidades avançadas de Processamento de Linguagem Natural (NLP) que permitem ao chatbot entender e responder a uma ampla gama de consultas de maneira eficaz. Isso pode ser vantajoso para criar experiências de usuário mais naturais e interativas.

f. Experiência do Desenvolvedor

Conhecimento Prévio: Desenvolvedores que já têm experiência com Spring Boot podem encontrar mais fácil e eficiente continuar usando essa tecnologia, em vez de aprender um novo framework ou linguagem.

g. Soluções Personalizadas

Personalização e Controle: Usar Spring Boot permite maior personalização e controle sobre a lógica de negócios e o fluxo de dados, o que pode ser necessário para aplicações mais complexas e específicas.

h. Soluções Empresariais Completas

Quando se trata de soluções empresariais completas, que vão além de um simples chatbot, combinar Spring Boot com Dialogflow pode oferecer uma solução robusta e escalável.

1.2 Exemplos de Cenários onde essa Escolha é Vantajosa

Suporte ao Cliente de Empresas

Chatbots que precisam acessar sistemas de CRM, bancos de dados internos, ou serviços de suporte ao cliente já existentes em Java.

Automação de Processos Internos

Empresas que automatizam processos internos como requisições de TI, agendamento de reuniões, ou consulta de dados financeiros.

Serviços Financeiros

Aplicações que requerem um alto nível de segurança e integração com sistemas de backend complexos.

1.3 Benefícios de Usar Spring Boot com Dialogflow

- **Escalabilidade**: Capacidade de suportar uma grande quantidade de usuários e transações.
- **Manutenção Simplificada**: Uso de uma única tecnologia para backend e integração com outros serviços.
- **Segurança Avançada**: Implementação robusta de autenticação e autorização.
- **Desenvolvimento Rápido**: Uso de Spring Boot permite o desenvolvimento rápido de APIs e serviços.

Veja que a escolha de usar Spring Boot com Dialogflow para um chatbot pode ser uma decisão estratégica para desenvolvedores e empresas que desejam aproveitar as vantagens da robustez e escalabilidade do Spring Boot, juntamente com as capacidades avançadas de NLP do Dialogflow. É especialmente relevante em contextos empresariais onde a integração com outros sistemas baseados em Java é crucial.

Neste projeto, vamos criar um chatbot funcional que pode responder a perguntas básicas e realizar tarefas simples. Utilizaremos o Spring Boot para construir o backend e o Dialogflow para gerenciar a lógica do diálogo. Vamos lá?

02

PREPARANDO O AMBIENTE

Antes de começarmos a codar, precisamos preparar nosso ambiente de desenvolvimento.

2.1 Instalando Ferramentas Necessárias

Vamos precisar das seguintes ferramentas:

1. Java JDK: Para rodar o Spring Boot.
2. Maven: Para gerenciar dependências do nosso projeto.
3. IDE: Recomendamos o IntelliJ IDEA ou o Eclipse.
4. Conta no Dialogflow: Para criar e gerenciar nosso chatbot.

2.2 Configurando o Projeto Spring Boot

Vamos criar um novo projeto Spring Boot usando o Spring Initializr. Siga os passos abaixo:

1. Acesse Spring Initializr.
2. Configure o projeto com as seguintes opções:
    a. **Project**: Maven
    b. **Language**: Java
    c. **Spring Boot:** 2.6.3 (ou a versão mais recente)
    d. **Group:** com.seuprojeto
    e. **Artifact:** chatbot
    f. **Dependencies:** Web, Lombok
3. Clique em "Generate" para baixar o projeto.

2.3 Criando sua Conta no Dialogflow

Acesse Dialogflow.

1. Crie uma conta ou faça login.
2. Crie um novo agente, nomeando-o como quiser.

Perceba que configurar o ambiente corretamente é essencial para o sucesso do nosso projeto. Então, dedique um tempo para garantir que tudo está instalado e configurado corretamente.

03

CONSTRUINDO O BACKEND COM SPRING BOOT

Vamos explorar como criar um projeto do zero, entender a estrutura do nosso projeto e configurar o aplicativo para que tudo funcione perfeitamente. O Spring Boot é uma ferramenta poderosa que nos permitirá desenvolver rapidamente um backend robusto e escalável, essencial para a funcionalidade do nosso chatbot. É hora de codificar e dar vida ao nosso assistente digital!

3.1 Criando um Projeto Spring Boot

Com o projeto baixado do Spring Initializr, descompacte e abra na sua IDE preferida. Agora, vamos entender a estrutura básica do projeto.

Nosso projeto terá a seguinte estrutura:

```
src/
├── main/
│   ├── java/
│   │   └── com/seuprojeto/chatbot/
│   │       ├── ChatbotApplication.java
│   │       ├── controller/
│   │       └── service/
│   └── resources/
│       └── application.properties
```

3.2 Configuração do Aplicativo

No arquivo **application.properties**, configure a porta para o nosso servidor Spring Boot:

```
server.port=8080
```

04

INTEGRANDO COM DIALOGFLOW

Agora que temos nosso backend pronto, é hora de integrar com o Dialogflow. Esta é uma plataforma que permite criar interfaces de conversa para diversas plataformas, incluindo chatbots. Vamos criar nosso primeiro assistente digital inteligente!

4.1 Criando seu Primeiro Assistente

1. No Dialogflow, crie um novo assistente e configure o idioma e a região.
2. Use o nome que preferir e clique em "Create".

4.2 Configurando Intents e Entidades

Intents são as intenções do usuário que nosso chatbot deve reconhecer e responder. Vamos criar uma intent simples de saudação.

1. Clique em "Create Intent".
2. Nomeie como "Greeting".
3. Adicione frases de exemplo como "Olá", "Oi", "Bom dia".
4. Configure a resposta padrão para algo como "Olá! Como posso ajudar?".

05

DESENVOLVENDO A LÓGICA DO CHATBOT

Neste capítulo, vamos nos concentrar no desenvolvimento da lógica do chatbot, que envolve receber e processar mensagens, além de responder de maneira inteligente. Vamos ver como podemos usar o Spring Boot para conectar nosso backend ao Dialogflow, garantir que o chatbot compreenda as intenções do usuário e forneça respostas adequadas. Também abordaremos o tratamento de erros e exceções para assegurar uma experiência suave e eficiente para os usuários. Vamos colocar a mão na massa e fazer o nosso chatbot brilhar!

5.1 Recebendo e Processando Mensagens

No nosso backend Spring Boot, vamos criar um controlador para receber e processar mensagens do Dialogflow.

```
@RestController
@RequestMapping("/webhook")
public class WebhookController

    @PostMapping
    public ResponseEntity<String> handleWebhook(@RequestBody
String payload)
        // Processar a mensagem aqui
        return ResponseEntity.ok("Resposta do chatbot");
    }
}
```

5.2 Respondendo com Inteligência

Vamos integrar com o Dialogflow para processar a mensagem e retornar uma resposta apropriada. Adicione a dependência do Dialogflow no **pom.xml**:

```
<dependency>
    <groupId>com.google.cloud</groupId>
    <artifactId>google-cloud-dialogflow</artifactId>
    <version>2.2.0</version>
</dependency>
```

5.3 Tratamento de Erros e Exceções

Sempre é importante tratar erros e exceções para evitar que o usuário tenha uma má experiência. Adicione um tratamento básico de exceções no controlador.

06

TESTANDO E DEPURANDO

Testar e depurar o código são passos essenciais para garantir que nosso chatbot funcione como esperado. Com testes unitários e integração, podemos verificar se cada parte do código está correta. Vamos garantir que tudo funcione direitinho e que possamos resolver qualquer problema que surgir.

6.1 Testes Unitários e Integração

Primeiro, vamos escrever alguns testes unitários para garantir que cada parte do nosso código funcione corretamente. Para isso, utilizaremos o JUnit e o Mockito.

Adicione as dependências no seu `**pom.xml**`:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>3.6.28</version>
    <scope>test</scope>
</dependency>
```

Agora, vamos criar um teste simples para o noss `**WebhookController**`. Criem um novo arquivo chamado `**WebhookControllerTest.java**` em `**src/test/java/com/seuprojeto/chatbot/controller**`.

6.2 Debugging

Depurar o código é essencial para encontrar e corrigir problemas. Vamos usar a ferramenta de depuração da nossa IDE.

1. Pontos de interrupção: Coloquem pontos de interrupção no código onde acham que pode haver problemas.
2. Modo de depuração: Iniciem a aplicação no modo de depuração.
3. Examinem variáveis: Quando o código parar no ponto de interrupção, examinem as variáveis para ver se contêm os valores esperados.

Por exemplo, vamos colocar um ponto de interrupção no método **`handleWebhook`** do nosso **`WebhookController`**.

```
@RestController
@RequestMapping("/webhook")
public class WebhookController {

    @PostMapping
    public ResponseEntity<String> handleWebhook(@RequestBody
String payload) {
        // Coloque um ponto de interrupção aqui
        System.out.println("Recebido: " + payload);
        return ResponseEntity.ok("Resposta do chatbot");
    }
}
```

```java
package com.seuprojeto.chatbot.controller;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.web.servlet.MockMvc;

import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest(WebhookController.class)
public class WebhookControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testHandleWebhook() throws Exception {
    String requestBody = "{ \"queryResult\": { \"queryText\": \"Oi\" } }";

    mockMvc.perform(post("/webhook")
            .contentType("application/json")
            .content(requestBody))
            .andExpect(status().isOk());
    }
}
```

6.3 Ferramentas de Teste

Para testar nossas APIs, podemos usar o Postman, uma ferramenta poderosa e fácil de usar. Vamos fazer isso juntos!

1. Instale o Postman: Baixem e instalem o Postman se ainda não tiver.
2. Crie uma nova requisição: Configurem uma nova requisição POST para **`http://localhost:8080/webhook`**.
3. Cabeçalhos: Configure o cabeçalho **`Content-Type`** para **`application/json`**.
4. Corpo da requisição: Adicione o seguinte JSON no corpo da requisição:

```
{
  "queryResult": {
    "queryText": "Oi"
  }
}
```

5. Envie a requisição: Clique em "Send" e verifiquem se a resposta é **200 OK** e a mensagem "Resposta do chatbot".

Dicas para Depuração:

Logs: Usem logs para acompanhar o que está acontecendo no código.

Divisão e Conquista: Quebrem problemas grandes em partes menores e solucionem uma de cada vez.

Documentação: Consultem a documentação e comunidades online para possíveis soluções.

07

MELHORIAS E PRÓXIMOS PASSOS

Vamos adicionar algumas funcionalidades extras ao nosso chatbot para torná-lo mais útil e interativo. Além disso, que tal adicionar uma integração com uma API externa para obter informações em tempo real? Vamos usar a API de previsão do tempo como exemplo.

7.1 Adicionando um Intent de Despedida

Vamos criar um novo intent no Dialogflow para que o chatbot possa responder a despedidas.

1. No Dialogflow, clique em "Create Intent".
2. Nomeie como "Goodbye".
3. Adicione frases de exemplo como "Tchau", "Até mais", "Adeus".
4. Configure a resposta padrão para algo como "Tchau! Volte sempre!".

7.2 Adicionando um Intent de Perguntas Frequentes

Vamos adicionar um intent para responder a perguntas frequentes.

1. Crie um novo intent no Dialogflow chamado "FAQ".
2. Adicione frases de exemplo como "Quais são seus horários de atendimento?", "Como posso falar com um atendente?".
3. Configure as respostas adequadas para cada pergunta.

No backend, precisamos atualizar o controlador para lidar com esses novos intents.

```
@RestController
@RequestMapping("/webhook")
public class WebhookController {

    @PostMapping
    public ResponseEntity<String> handleWebhook(@RequestBody WebhookRequest
request) {
    String intentName =
request.getQueryResult().getIntent().getDisplayName();
    String response;

    switch (intentName) {
        case "Greeting":
            response = "Olá! Como posso ajudar?";
            break;
        case "Goodbye":
            response = "Tchau! Volte sempre!";
            break;
        case "FAQ":
            response = handleFaq(request);
            break;
        default:
            response = "Desculpe, não entendi sua pergunta.";
            break;
    }
```

```
    WebhookResponse webhookResponse = new WebhookResponse();
    webhookResponse.setFulfillmentText(response);
    return ResponseEntity.ok().body(webhookResponse.toString());
    }

    private String handleFaq(WebhookRequest request) {
    String question = request.getQueryResult().getQueryText();

    if (question.contains("horários de atendimento")) {
        return "Nosso horário de atendimento é das 9h às 18h de segunda a
sexta.";
    } else if (question.contains("falar com um atendente")) {
        return "Você pode falar com um atendente ligando para 0800-123-456.";
    } else {
        return "Desculpe, não tenho uma resposta para essa pergunta.";
    }
    }
}
```

7.3 Integrando com a API de Previsão do Tempo

1. Crie uma conta em um serviço de previsão do tempo, como OpenWeatherMap, e obtenham uma chave de API.
2. Adicione uma dependência para fazer requisições HTTP no **`pom.xml`**:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

3. Crie uma classe de serviço para fazer a chamada à API:

```
@Service
public class WeatherService {

    private final RestTemplate restTemplate;
    private final String apiKey = "SUA_CHAVE_API_AQUI";
    private final String apiUrl =
"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={apiKey}";

    public WeatherService(RestTemplate restTemplate) {
    this.restTemplate = restTemplate;
    }
```

```
    public String getWeather(String city) {
    try {
        String url = apiUrl.replace("{city}", city).replace("{apiKey}",
apiKey);
        ResponseEntity<String> response = restTemplate.getForEntity(url,
String.class);
        return parseWeatherResponse(response.getBody());
    } catch (Exception e) {
        return "Não consegui obter a previsão do tempo. Tente novamente mais
tarde.";
    }
    }

    private String parseWeatherResponse(String responseBody) {
    // Parse o JSON e extraia a informação desejada
    // Retorne a string com a previsão do tempo formatada
    }
}
```

4. Atualize o controlador para lidar com pedidos de previsão do tempo:

```
@RestController
@RequestMapping("/webhook")
public class WebhookController {

    private final WeatherService weatherService;

    public WebhookController(WeatherService weatherService) {
    this.weatherService = weatherService;
    }
```

```
@PostMapping
    public ResponseEntity<String> handleWebhook(@RequestBody WebhookRequest
request) {
    String intentName = request.getQueryResult().getIntent().getDisplayName();
    String response;

    switch (intentName) {
        case "Greeting":
            response = "Olá! Como posso ajudar?";
            break;
        case "Goodbye":
            response = "Tchau! Volte sempre!";
            break;
        case "FAQ":
            response = handleFaq(request);
            break;
        case "Weather":
            String city =
request.getQueryResult().getParameters().get("geo-city").getStringValue();
            response = weatherService.getWeather(city);
            break;
        default:
            response = "Desculpe, não entendi sua pergunta.";
            break;
    }
```

```
    WebhookResponse webhookResponse = new WebhookResponse();
    webhookResponse.setFulfillmentText(response);
    return ResponseEntity.ok().body(webhookResponse.toString());
    }

    private String handleFaq(WebhookRequest request) {
    String question = request.getQueryResult().getQueryText();

    if (question.contains("horários de atendimento")) {
        return "Nosso horário de atendimento é das 9h às 18h de segunda a
sexta.";
    } else if (question.contains("falar com um atendente")) {
        return "Você pode falar com um atendente ligando para 0800-123-456.";
    } else {
        return "Desculpe, não tenho uma resposta para essa pergunta.";
    }
    }
}
```

No Dialogflow, crie um novo intent chamado "Weather" e configure as frases de exemplo, como "Qual é a previsão do tempo para [cidade]?", onde [cidade] é um parâmetro.

7.4 Publicando seu Chatbot

Para publicar seu chatbot, podemos usar um serviço de hospedagem. Abaixo deixei uma lista de opções gratuitas (com certas limitações por serem gratuitas) até a data em que esse ebook foi publicado:

- **Netlify**: Ideal para projetos front-end, mas pode ser usado para back-end com funções serverless.
- **Vercel:** Excelente para projetos JAMstack e suporte a APIs serverless.
- **GitHub Pages:** Bom para projetos estáticos, com suporte a GitHub Actions para back-end.
- **Glitch:** Ambiente de desenvolvimento colaborativo e hospedagem gratuita.
- **Firebase:** Oferece hospedagem gratuita para aplicações web e funções serverless.
- **AWS Free Tier:** Amazon Web Services oferece um nível gratuito com diversos serviços, incluindo EC2 e Lambda.
- **Google Cloud Platform Free Tier:** Google Cloud oferece um nível gratuito com várias opções de hospedagem.
- **Microsoft Azure Free Tier:** Azure também oferece um nível gratuito com serviços de hospedagem variados.
- **Oracle Cloud Free Tier:** Oferece recursos gratuitos, incluindo computação e armazenamento.
- **Repl.it:** Ambiente de codificação colaborativo com opções de hospedagem para pequenos projetos.
- **Surge.sh:** Hospedagem gratuita para projetos estáticos com integração contínua.
- **Render:** Hospedagem gratuita com suporte a contêineres e aplicativos web.

08

RECURSOS E LEITURAS ADICIONAIS

Estamos chegando ao final da nossa jornada, mas o aprendizado não para por aqui! Neste capítulo, vamos compartilhar uma seleção de recursos adicionais e leituras recomendadas para que vocês possam continuar se aprimorando na criação de chatbots com Spring Boot e Dialogflow. Acredito que, ao explorar esses materiais, vocês poderão aprofundar seus conhecimentos, descobrir novas técnicas e ficar por dentro das melhores práticas do mercado.

8.1 Documentação Oficial

Para aprofundar seu conhecimento e resolver dúvidas específicas, a documentação oficial é sempre uma excelente fonte de informação:

- Documentação do Spring Boot: Spring Boot
- Documentação do Dialogflow: Dialogflow

8.2 Tutoriais e Artigos Gratuitos

A seguir, listo alguns tutoriais e artigos gratuitos de universidades e posts interessantes no LinkedIn, que podem ajudar a complementar seu aprendizado:

Universidade de Stanford:

- Curso de Processamento de Linguagem Natural: Este curso oferece uma visão geral das técnicas de NLP, que são fundamentais para o funcionamento de chatbots. Disponível gratuitamente no Coursera.

Harvard University:

- CS50's Introduction to Artificial Intelligence with Python: Embora focado em Python, este curso gratuito fornece uma base sólida em IA e chatbots, aplicável a várias linguagens e frameworks. Disponível no edX.

LinkedIn Learning:

- Build a Chatbot with Google Dialogflow: Um tutorial prático que aborda a criação de chatbots com Dialogflow. Disponível gratuitamente com a avaliação do LinkedIn Learning aqui.

Artigos no LinkedIn:

- "Creating Conversational AI: Building Chatbots with Dialogflow": Um artigo bem detalhado e popular sobre como criar chatbots com Dialogflow. Confira no LinkedIn.
- "Spring Boot for Beginners: A Comprehensive Guide": Um guia completo e acessível sobre como começar com Spring Boot, também disponível no LinkedIn.

Comunidades e Fóruns:

Participar de comunidades online e fóruns pode ser extremamente útil para obter suporte, trocar ideias e aprender com a experiência de outros desenvolvedores:

- Stack Overflow: Um excelente recurso para tirar dúvidas específicas sobre problemas de codificação. Confira a tag Spring Boot e Dialogflow.
- Reddit: O subreddit r/java e r/Chatbots são ótimos lugares para discussões e compartilhamento de conhecimento.

AGRADECIMENTOS

Espero que esse ebook ajude vocês, queridos devs, a construírem chatbots incríveis e funcionais. Lembrem-se de que cada passo é uma oportunidade de aprendizado e crescimento. Boa sorte e mãos à obra!

Este ebook foi gerado por IA e formatado por uma equipe humana. O guia completo está disponível no meu GitHub. Este conteúdo visa propósitos educacionais e não foi minuciosamente validado por um humano, podendo conter eventuais erros típicos de uma IA.

Mesmo assim, pelo que observei está tudo ok, em termos de código!

GitHub

https://github.com/alinemach/prompts-recipe-to-create-a-ebook.git

Autora: Aline de Oliveira Machado
Linkedin
Github