PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E DE ARTES
GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

CHATBOT DE SERVIÇOS INTEGRADO AO WHATSAPP: UM COMPARATIVO DE MODELOS DE IAS

MARCOS VINICIUS PEREIRA LEMES

GOIÂNIA
2025

MARCOS VINICIUS PEREIRA LEMES

CHATBOT DE SERVIÇOS INTEGRADO AO WHATSAPP: UM COMPARATIVO DE MODELOS DE IAS

Trabalho de Conclusão de Curso apresentado à Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia da Computação.

Orientador(a):

Prof. Me. Rafael Leal Martins

Banca examinadora:

Prof. Me. Fabricio Schlag

Prof. Me. Fernando Gonçalves Abadia

GOIÂNIA

2025

MARCOS VINICIUS PEREIRA LEMES

CHATBOT DE SERVIÇOS INTEGRADO AO WHATSAPP: UM COMPARATIVO DE MODELOS DE IAS

Trabalho de Conclusão de Curso aprovado em sua forma final pela Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, para obtenção do título de Bacharel em Engenharia da Computação, em _____/_____/_____.

_____
Orientador(a): Prof. Me. Rafael Leal Martins

_____
Prof. Me. Fabricio Schlag

_____
Prof. Me. Fernando Gonçalves Abadia

GOIÂNIA
2025

DEDICATÓRIA

Dedico esse trabalho, em primeiro lugar, a Deus, por me conceder força, saúde e sabedoria em todo esse percurso. Aos meus pais, pelo amor incondicional, apoio diário e fé incessante em meu potencial, mesmo nos piores momentos. Aos meus amigos e colegas de faculdade, pela parceria, incentivo e momentos de diversão que descontrairam meu aprendizado. E a todos os demais que, de alguma forma, contribuíram para a conclusão do trabalho.

AGRADECIMENTOS

A Deus, por permitir-me chegar até aqui com saúde, paciência e força em cada etapa. Aos meus pais e demais familiares, pelo amor, incentivo e apoio incondicional em todos os momentos de minha vida acadêmica e pessoal. Ao professor Rafael Leal Martins, meu orientador, pela dedicação, disponibilidade, paciência e pelas valiosas orientações para elaboração deste trabalho. Sua contribuição foi fundamental para a conclusão deste projeto. Aos amigos e colegas de curso, que tornaram os anos de aprendizado, discussões, trabalhos e afinidades mais leves e prazerosos. Aos demais professores e colaboradores da instituição que, com conhecimento e experiências, contribuíram para a minha formação. E a todos que, de alguma forma, direta ou indiretamente, colaboraram para a realização deste trabalho, o meu sincero muito obrigada.

RESUMO

Este trabalho apresenta o desenvolvimento de um chatbot no whatsapp com a utilização de modelos de IA generativa para realização de serviços via HTTP por meio de mensagens em linguagem natural. A solução foi implementada utilizando o framework Laravel e integrado à API oficial do WhatsApp Business. O trabalho explica a importância da utilização do WhatsApp na execução de serviços para empresas, em seguida explica detalhes das ferramentas utilizadas e, em seguida, é mostrado como essas ferramentas interagem com os modelos de IA Generativas. O principal objetivo deste trabalho foi avaliar a viabilidade da integração de modelos de IA ao chatbot, realizando um comparativo entre os modelos a fim de entender as limitações e o potencial uso dessa integração. Os modelos de IA foram testados e comparados em diferentes cenários, evidenciando vantagens e limitações de cada tecnologia. Após testes na implementação foi concluído que é possível a integração do chatbot com os modelos de IA, apesar de cada IA demonstrar limitações diferentes o projeto aponta potencial para melhorias.

Palavras-chave: chatbot, inteligência artificial, Laravel, WhatsApp Business API, NLP, serviços automatizados, análise comparativa.

ABSTRACT

This work presents the development of a chatbot for WhatsApp using generative AI models to perform services via HTTP through natural language messages. The solution was implemented using the Laravel framework and integrated with the official WhatsApp Business API. The project explains the importance of using WhatsApp for service execution in companies, followed by details about the tools used, and how these tools interact with generative AI models. The main objective of this work was to evaluate the feasibility of integrating AI models into the chatbot, performing a comparison between the models to understand their limitations and the potential use of this integration. The AI models were tested and compared in different scenarios, showing the advantages and limitations of each technology. After the implementation tests, it was concluded that integrating the chatbot with AI models is possible, although each AI showed different limitations. The project also points out potential for future improvements.

Keywords: chatbot, artificial intelligence, Laravel, WhatsApp Business API, natural language processing, automated services, comparative analysis.

LISTA DE ILUSTRAÇÕES

Figura 1 - Número de usuários exclusivos do whatsapp..... 18
Figura 2.1 – Diagrama de fluxo dos componentes de um chatbot..... 23
Figura 2.2 – Representação visual dos principais componentes de um chatbot.....25
Figura 4.1 – Diagrama representando a interação entre as camadas do padrão MVC... 29
Figura 4.2 – Estrutura da arquitetura do Eloquent ORM, ilustrando o padrão ActiveRecord e o relacionamento entre modelos e tabelas..... 30
Figura 7.1 - Envio de mensagem pela API do WhatsApp Business.....38
Figura 7.2 – Recebimento da mensagem enviada pela API do WhatsApp Business..... 39
Figura 7.3 – Definição das rotas de verificação e recebimento de mensagens no Laravel..... 40
Figura 7.4 – Autenticação do webhook da API do WhatsApp no servidor Laravel..... 40
Figura 7.5 – Função receiveMessage para o processamento de mensagens recebidas..... 41
Figura 7.6 – Código para criação automática de um novo contato no banco de dados.....42
Figura 7.7 – Código de validação e registro do aceite da LGPD pelo usuário.....43
Figura 7.8 – Envio da mensagem para a IA e interpretação do retorno.....45
Figura 7.9 – Função montarJsonDeParametros responsável por construir o prompt com parâmetros e processar o retorno da IA em formato JSON..... 47
Figura 7.10 – Função executarIaOpenAi responsável por enviar o prompt à API do ChatGPT e processar a resposta gerada.....50
Figura 7.11 - Exemplo de requisição à API Gemini utilizando comando cURL..... 52
Figura 7.12 – Função executarIaGemini responsável pela integração entre a aplicação e a API do Gemini..... 52
Figura 7.13 – Função executarIaCohere responsável pela integração com a API da Cohere e processamento da resposta..... 55
Figura 7.10 – Arquivo .env com variáveis de configuração da API..... 59
Figura 7.11 – Exemplo de migration para criação de tabelas no banco de dados.....60
Figura 7.12 – Código para execução do modelo de IA configurado.....61
Figura 7.13 – Geração e gerenciamento de tokens de autenticação para serviços.....62
Figura 8.1 - Teste de Envio de Intenção com IA ChatGPT.....68
Figura 8.2 - Teste de Recebimento de Resposta do Serviço da VIA Chat GPT.....69
Figura 8.3 – Interação com o ChatGPT em mensagem com alto grau de erro ortográfico.....70
Figura 8.4 – Interação com o Gemini em mensagem com uso de gírias..... 71
Figura 8.5 – Exemplo de resposta adequada do ChatGPT diante de parâmetros inválidos.....72
Figura 8.6 - Código Com Ajuste Para Montagem Da Mensagem da API de Serviço no Gemini.. 73
Figura 8.7 - Teste de Envio de Intenção com IA Gemini..... 74
Figura 8.8 - Teste de Recebimento de Resposta do Serviço da VIA Gemini..... 75
Figura 8.9 – Interação com o Gemini em mensagem com alto grau de erro ortográfico..... 76
Figura 8.10 – Interação com o Gemini em mensagem com uso de gírias..... 77
Figura 8.11 – Exemplo de resposta adequada do Gemini diante de parâmetros inválidos.....78
Figura 8.12 - Teste de Envio de Intenção com IA Cohere.....79
Figura 8.13 - Teste de Recebimento de Resposta do Serviço da VIA Cohere.....80
Figura 8.14 – Interação com o Cohere em mensagem com alto grau de erro ortográfico.....81
Figura 8.15 – Interação com o Cohere em mensagem com uso de gírias.....82
Figura 8.16 – Exemplo de resposta adequada do Cohere diante de parâmetros inválidos.....83

Figura 8.17 – Retorno da API informando erro devido à ausência de parâmetro obrigatório..... 84
Figura 8.18 – Trecho de código responsável por interromper a requisição ao detectar ausência de parâmetros obrigatórios.....85

LISTA DE SIGLAS

IM	Instant Messenger; Aplicativos de Mensagens Instantâneas
HTTP	Hypertext Transfer Protocol, Protocolo de Transferência de Hipertexto
URI	Uniform Resource Identifier, Identificador Uniforme de Recurso
API	Application Programming Interface, Interface de Programação de Aplicações
REST	Representational State Transfer, Transferência Representacional de Estado
JSON	JavaScript Object Notation, Notação para Objeto em JavaScript
IA	Inteligência Artificial
URL	Uniform Resource Locator; Localizador Uniforme de Recursos
NLP	Natural Language Processing, Processamento de Linguagem Natural
CRM	Customer Relationship Management, Gestão de Relacionamento com o Cliente
ORM	Object-Relational Mapping, Mapeamento Objeto-Relacional
MVC	Model-View-Controller; Modelo-Visão-Controlador
LGPD	Lei Geral de Proteção de Dados Pessoais
NLP	Natural Language Processing, Processamento de Linguagem Natural
LLM	Large Language Model, Modelos de Linguagem de Grande Escala
GAN	Generative Adversarial Network, Rede Generativa Adversária
VAE	Variational Autoencoder, Autoencoder Variacional

SUMÁRIO

RESUMO **9**
1 INTRODUÇÃO **17**
1.1 Justificativa 18
1.2 Objetivos 19
1.2.1 Objetivo geral 19
1.2.2 Objetivos específicos 19
2 CHATBOT **20**
2.1 História Do ChatBot 20
2.2 Componentes de um ChatBot 21
2.3 Como Funciona o ChatBot 23
3 WHATSAPP BUSINESS **25**
3.1 Webhook 25
4 LARAVEL **26**
4.1 Padrão MVC 27
4.2 Eloquent ORM 28
5 API **29**
5.1 API REST 30
5.2 Endpoint De API 31
5.3 Autenticação e Autorização de uma API 31
5.3.1 Token de Autenticação 31
6 INTELIGÊNCIA ARTIFICIAL **31**
6.1 Inteligência Artificial Generativa 32
6.2 OpenAi **34**
6.2.1 ChatGPT 34
6.2.2 API ChatGPT 34
6.3 Gemini **35**
6.4 Cohere **35**
7 ESTUDO DE CASO **36**
7.1 Configuração da API do Whatsapp Business 36
7.2 Desenvolvimento do Servidor Laravel 38
7.3 Integração da Inteligência Artificial no Servidor Laravel 43
7.3.1 Integração com o ChatGPT 47
7.3.2 Integração com o GEMINI 50
7.3.3 Integração com o Cohere 52
7.4 Estrutura do Banco de Dados 56
7.4.1 Tabela chat.servico 56
7.4.2 Tabela chat.token 57
7.4.3 Tabela chat.contato 57
7.5 Elementos da solução e relacionamento entre eles 58

8 TESTES REALIZADOS..... 65
8.1 Metodologia dos Testes..... 65
8.2 Descrição dos Testes Realizados..... 65
8.2.1 Teste com ChatGpt..... 66
8.2.2 Teste Com Gemini..... 71
8.2.3 Teste Com Cohere..... 77
8.3 Teste Faltando Parâmetros Obrigatórios..... 82
8.4 Testes De Segurança..... 85
8.5 Conclusão dos Testes..... 86
9 CONSIDERAÇÕES FINAIS..... 87
9.1 Discussão..... 88
9.2 Limitações..... 88
9.3 Conclusão..... 89
9.4 Sugestão de Trabalhos Futuros..... 90
REFERÊNCIAS..... 91

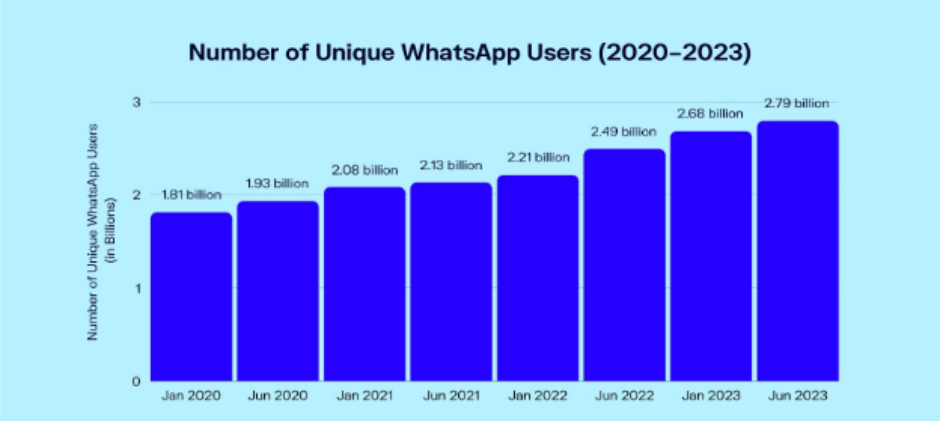
1 INTRODUÇÃO

Os aplicativos de mensagens instantâneas estão cada vez mais presentes no dia-a-dia da população. No Brasil, o WhatsApp está há anos como o mais utilizado, com cerca de 99% das pessoas que possuem smartphone tendo o aplicativo baixado.(PURZ, 2025)

O WhatsApp é cada vez mais buscado para novos propósitos desde o uso pessoal para se aproximar de amigos e familiares a uso comercial para divulgar algum produto ou serviço, Segundo BIANCHI (2025) mais de 93% da população brasileira está sendo usuário deste aplicativo onde já fez parte do estilo de vida do brasileiro.

Com o aumento do uso do Whatsapp surgiu a oportunidade das empresas ficarem mais próximas de seus clientes utilizando esse canal de comunicação com o cliente trazendo vários benefícios para empresa e mais acessibilidade ao cliente. Na Figura 1 mostra como o número de usuários do WhatsApp tem subido a cada ano.

Figura 1 - Número de usuários exclusivos do whatsapp.



fonte:https://timelines.ai/pt/whatsapp-user-statistics-and-revenue-analysis/

Segundo SCHENDES (2023) 95% das empresas utilizam o WhatsApp além da plataforma de mensagem o investimento das empresas em chatbots para automatização de seus serviços vem tendo um grande crescimento.

No seu nível mais baixo os chatbots trata-se de um software que simula uma conversas humanas pré-programada podendo ser tão simples quanto programas rudimentares que respondem consulta simples ou fazer o uso de inteligência artificial (IA) para melhor responder seus clientes onde aprendem e evoluem a cada

novo cliente nesse sentido é uma excelente ferramenta tendo em vista que pode conversar com milhares de clientes ao mesmo tempo isso irá aumentar a produtividade e satisfação de seu cliente.(INTELBRAS, 2023; ORACLE, 2020)

Para a melhor utilização de um chatbot é preciso de uma estratégia adequada a área que ele será colocado, de uma forma simples uma estratégia se trata de um plano de alocação de recursos destinados a nos levar da posição atual no presente para uma posição mais favorável no futuro.(CRUZ, 2018)

Escolher uma estratégia adequada é fundamental para maximizar os benefícios de um chatbot em qualquer área de aplicação. Como mencionado por Schmitz, uma estratégia bem definida é essencial para direcionar os recursos de forma eficaz. Ao alinhar o uso do chatbot com os objetivos e metas estratégicas da organização, é possível obter vantagens significativas, como aumento da eficiência operacional, melhoria da experiência do cliente e diferenciação competitiva.

Os chatbots desempenham um papel crucial na interação entre empresas e clientes, oferecendo suporte contínuo e personalizado. Ao adotar uma abordagem de assistência humana estilizada, esses chatbots podem compreender as necessidades dos clientes e fornecer respostas imediatas em linguagem natural. Sua disponibilidade 24 horas por dia, capacidade de processar grandes volumes de dados e personalização tornam-nos um canal de comunicação eficaz e conveniente para os clientes, promovendo uma experiência ao cliente excepcional e fortalecendo o relacionamento entre cliente e empresa.(CRUZ, 2018)

1.1 Justificativa

Esse estudo é relevante porque segundo Dino o crescimento constante de tecnologias como chatbot está cada vez maior e com projeção de crescimento de até 23% até 2028, que estão remodelando profundamente a maneira como as empresas interagem com seus clientes e fornecendo serviços. Os chatbots oferecem uma solução eficiente e acessível para empresas de todos os tamanhos, permitindo uma comunicação instantânea e personalizada através de plataformas populares como o Whatsapp. Ao adotar chatbots as empresas podem oferecer uma experiência ao cliente altamente conveniente, fornecendo acesso 24 horas por dia, 7 dias por semana, a serviços e informações essenciais. Além disso, o uso de chatbots tem o potencial de

reduzir significativamente os custos operacionais, automatizando tarefas repetitivas de atendimento ao cliente e liberando recursos para atividades mais estratégicas. (DINO, 2023)

Diante deste contexto, este projeto tem por questão de pesquisa: Como os chatbots podem ser efetivamente utilizados para fornecer uma ampla gama de serviços à população por meio da plataforma de mensagens WhatsApp?

1.2 Objetivos

1.2.1 Objetivo geral

O objetivo deste trabalho é desenvolver e avaliar um chatbot integrado ao WhatsApp Business, utilizando modelos de inteligência artificial generativa, para automatizar a execução de serviços oferecidos pela empresa Megasoft por meio de mensagens em linguagem natural.

1.2.2 Objetivos específicos

- Implementar um chatbot funcional utilizando o framework Laravel e a API oficial do WhatsApp Business;
- Utilizar a APIs da OpenAi, Gemini e Cohere para auxílio na compreensão de contexto da conversa.
- Utilizar o Framework Laravel para trabalhar com as requisições utilizadas pelo bot.
- Implementar os serviços de consulta de certidão negativa de débitos, emissão de nota fiscal eletrônica e consulta de alvará/habite-se.
- Desenvolver funcionalidades para emissão de extrato fiscal, certidão de regularidade, geração de guias de IPTU e exibição da ficha cadastral do contribuinte..

2 CHATBOT

Chatbots são programas de computador de conversa avançada criados para interação de máquina e usuários em linguagem natural, são capazes de manter um diálogo coerente como se fosse um interlocutor humano. Com algoritmos de aprendizado avançado eles podem responder a várias consultas simultaneamente, liberando os humanos para tarefas mais estratégicas. Esses sistemas imparciais podem resolver desde dúvidas simples até questões mais complexas, agindo como verdadeiros assistentes virtuais. (ANDRION 2019; CRUZ, 2018).

2.1 História Do ChatBot

O conceito de interação entre robôs e humanos teve seu início em 1940 com o livro "Eu, Robô", que apresentava nove contos explorando esse tema (IBRAMERC, 2017; LEADSTER).

Em 1950, o pioneiro da Ciência da Computação propôs um teste para determinar se um humano seria capaz de conversar com um computador sem perceber que estava interagindo com uma máquina (IBRAMERC, 2017; LEADSTER, [s.d.]).

O primeiro chatbot verdadeiro surgiu em 1966 com o Eliza, criado por Joseph Weizenbaum. Esta ferramenta funcionava por correspondência e substituição de padrões para simular conversas. Sua abordagem era fundamentada na empatia e ausência de julgamento, buscando entender as profundezas do indivíduo e tornando-se um local onde os usuários confiavam seus pensamentos mais profundos (IBRAMERC, 2017; LEADSTER, [s.d.]).

O surgimento do Eliza deu origem a vários outros ao longo dos anos, e hoje em dia a interação com ferramentas de chatbots e assistentes virtuais é parte integrante do cotidiano dos usuários (IBRAMERC, 2017; LEADSTER, [s.d.]).

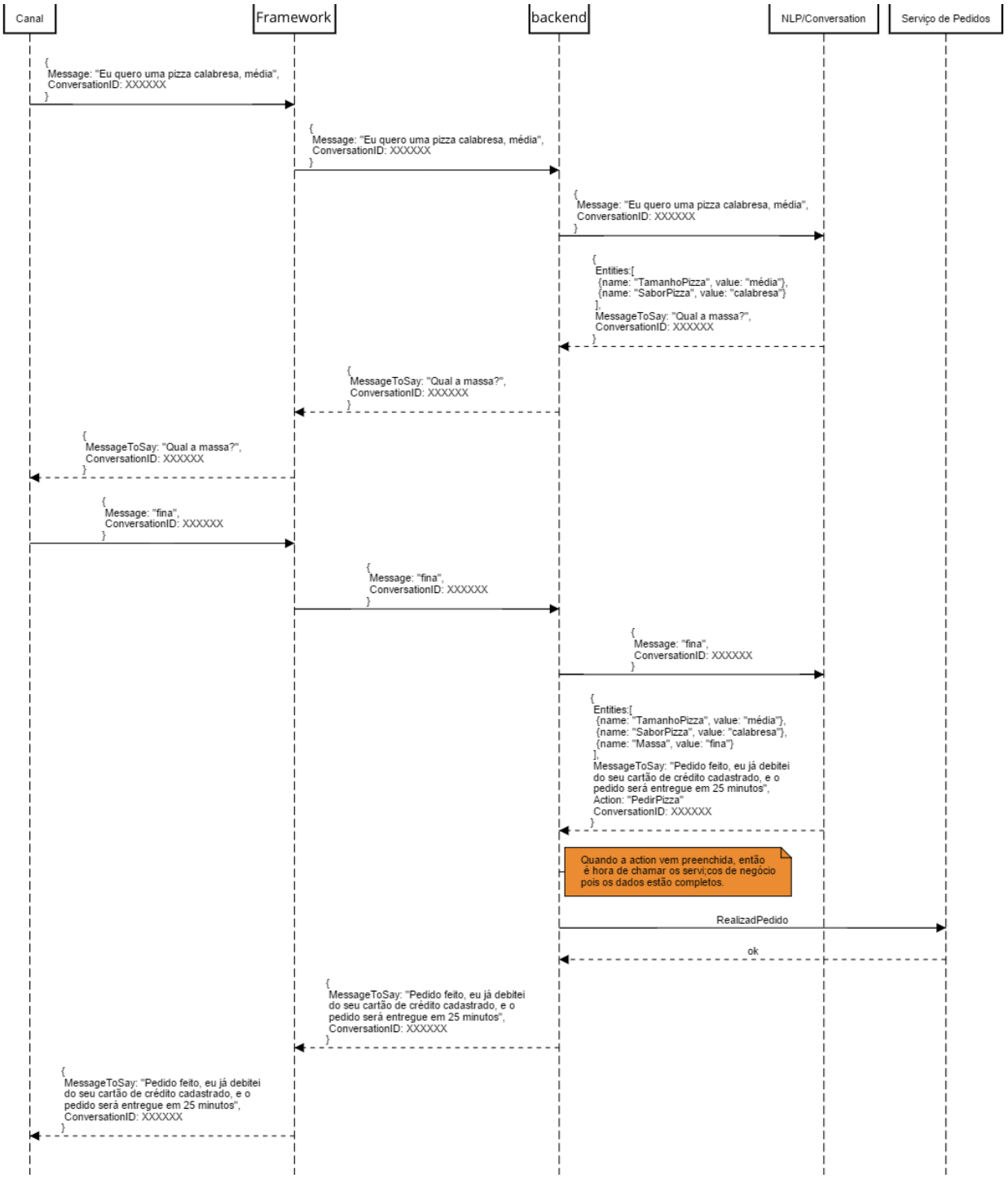
O Eliza é considerado um marco na história da inteligência artificial e no campo da comunicação homem-máquina, pois demonstrou que era possível simular uma conversa com um computador de forma convincente, mesmo utilizando regras simples. Seu impacto foi significativo para os estudos de NLP (Processamento de Linguagem Natural) e influenciou o desenvolvimento de assistentes virtuais modernos.(IBRAMERC, 2017; LEADSTER, [s.d.]).

2.2 Componentes de um ChatBot

Um chatbot é composto por elementos básicos que facilitam sua operação. O usuário, que pode ser um cliente de uma empresa ou um funcionário, está presente no chatbot para interação através de algum dos canais do bot. O canal é a interface através da qual o chatbot se comunica com o usuário, como por exemplo, os serviços de mensagens instantâneas (IM). Com a variedade de canais disponíveis, é necessário um framework para gerenciar essas interações. Normalmente, o framework do chatbot está integrado ao seu backend. O backend é responsável por responder às diversas interações do usuário, dando vida ao bot. Nele, são implementadas todas as regras de negócio e serviços oferecidos pelo chatbot. Além disso, é no backend que o processamento de linguagem natural (NLP) é aplicado, transformando a linguagem natural em estruturas tecnologicamente utilizáveis.(FARIA, 2017)

A Figura 2.1 ilustra o fluxo principal de funcionamento de um chatbot, desde a entrada do usuário até o processamento da resposta. Ela demonstra como os elementos interagem – o canal de entrada, o framework do bot, o backend, e a camada de NLP – para fornecer respostas adequadas.

Figura 2.1 – Diagrama de fluxo dos componentes de um chatbot.



Fonte: FARIA (2023)

2.3 Como Funciona o ChatBot

Para que um chatbot funcione, três componentes básicos são necessários. São eles: o canal, o conteúdo e o software. Esses componentes são utilizados para receber a mensagem do usuário, analisá-la, reconhecê-la e, finalmente, extraí-la todos os dados importantes e necessários para que a resposta do chatbot possa ser adequada para satisfação do usuário. Dessa forma, a cada mensagem, o chatbot pode guiar a conversa para que, cada vez mais, se resolva o problema que o usuário gostaria.(BLIP, 2024)

O canal de um chatbot pode ser definido como o ponto de contato com a empresa. É importante sempre oferecer disponibilidade para os canais mais utilizados pelos seus clientes, pois será através deles que seus clientes poderão entrar em contato e enviar suas mensagens.(PINEDA, 2022)

O conteúdo inclui todos os recursos que podem ser utilizados na comunicação, tais como texto, emojis e até mesmo arquivos de mídia. O usuário desempenha o papel de emissor ao iniciar a conversa através do canal escolhido, enquanto o chatbot atua como receptor, fornecendo um feedback em resposta.(BLIP, 2024, THEODORO, 2019)

O software é o programa responsável por analisar o conteúdo recebido do usuário, independentemente do canal. Ele interpreta e analisa todas as informações para formular uma resposta adequada ao usuário.(BLIP, 2024)

Além disso, o componente de software do chatbot desempenha um papel central na condução da conversa, pois é responsável por interpretar as mensagens recebidas e formular respostas adequadas. Isso é feito por meio de técnicas de Processamento de Linguagem Natural (NLP), que permitem ao chatbot compreender a linguagem humana em sua forma escrita, levando em consideração o contexto, a intenção e os dados fornecidos pelo usuário (STRYKER; HOLDSWORTH, 2024).

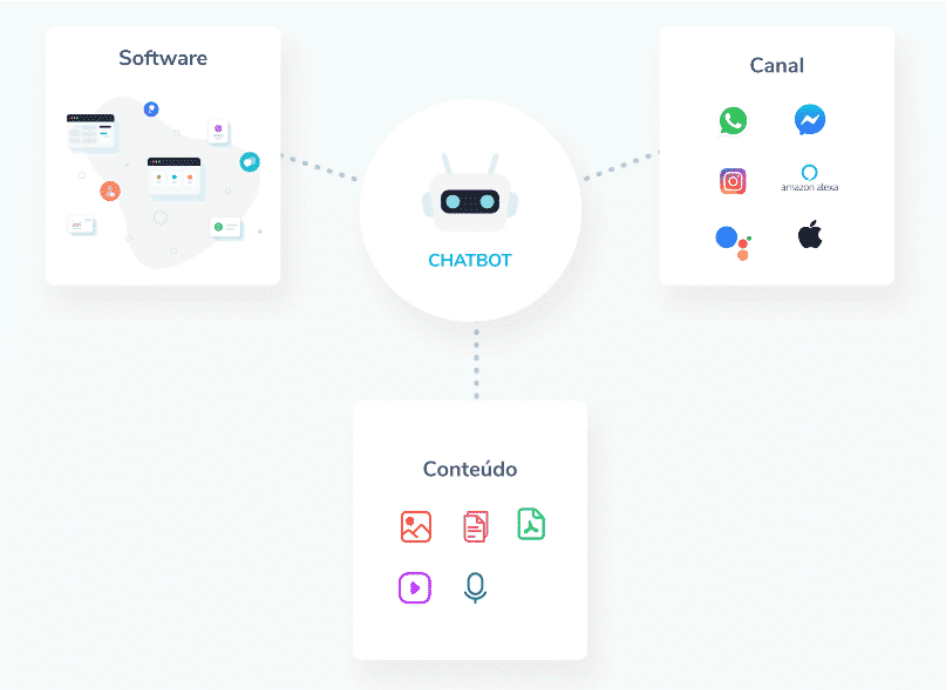
O NLP envolve etapas como a identificação da intenção da mensagem, a extração de entidades específicas (como nomes, documentos ou datas), a análise sintática e a geração de respostas em linguagem natural. Essas capacidades são essenciais para que a conversa flua de maneira natural, sem exigir comandos rígidos ou menus pré-definidos (BLIP, 2023). A combinação dessas técnicas com algoritmos de

machine learning permite que o chatbot aprenda e evolua a partir das interações, tornando-o cada vez mais eficaz.

A mensagem enviada pelo usuário corresponde ao conteúdo principal da comunicação, que, segundo os elementos da comunicação, é decodificada pelo chatbot (receptor), processada e convertida em uma resposta adequada (THEODORO, 2024). Já o canal, como o WhatsApp, representa o meio pelo qual essa troca ocorre, sendo um fator crucial para alcançar os usuários com conveniência e agilidade (PINEDA, 2024).

Com isso, a arquitetura do chatbot se consolida como uma estrutura interativa e inteligente, capaz de fornecer informações, executar serviços e adaptar-se às necessidades do usuário, promovendo uma experiência de atendimento mais eficiente e personalizada. Na Figura 2.2 é mostrado uma representação visual dos componentes do chatbot.

Figura 2.2 – Representação visual dos principais componentes de um chatbot.



Fonte: BLIP(2024)

3 WHATSAPP BUSINESS

O WhatsApp é uma das plataformas de comunicação mais populares do mundo, permitindo que bilhões de pessoas troquem mensagens de forma rápida e segura. Para atender às necessidades comerciais, a Meta desenvolveu o WhatsApp Business, uma solução voltada para empresas que desejam se comunicar de forma eficiente com seus clientes.(META, 2025?)

A Plataforma do WhatsApp Business não é um aplicativo, mas uma API que possibilita a comunicação em escala. Com suas Apis, empresas podem conectar agentes humanos e chatbots para proporcionar interações automatizadas e personalizadas com clientes.(META, 2025?)

A API do WhatsApp Business foi pensada para empresas que precisam se comunicar com seus clientes de forma eficiente. Diferente do aplicativo original, o WhatsApp Business oferece uma infinidade de recursos adaptados às necessidades das empresas como integração com sistemas de CRM (Customer Relationship Management), que são plataformas voltadas para a gestão do relacionamento com o cliente, permitindo o registro, acompanhamento e análise de interações comerciais e de atendimento..(KOMMO, 2020)

A API do WhatsApp Business permite que as empresas enviem mensagens em quatro categorias: marketing, autenticação, utilitários e serviços. Para iniciar conversas, em muitos casos, é necessário utilizar modelos de mensagens previamente aprovados pela Meta, especialmente em situações que envolvem o envio de comunicações ativas, como promoções, notificações ou autenticações.(KOMMO, 2020)

A automatização proporcionada pela API é crucial para empresas que buscam aumentar sua eficiência operacional. Ao integrar chatbots, é possível criar fluxos de atendimento que gerenciam desde a captação de leads até a entrega de notificações importantes, como status de pedidos ou confirmações de agendamentos.(HOLANDA, 2025)

3.1 Webhook

Webhooks são mecanismos que possibilitam a comunicação automática entre dois sistemas distintos. Eles funcionam como “gatilhos” baseados em eventos, ou seja,

ao ocorrer determinada ação em um sistema (como o recebimento de uma mensagem), ele envia uma requisição HTTP do tipo POST para uma URL previamente configurada em outro sistema. Essa abordagem elimina a necessidade de consultas periódicas (polling) à API, promovendo eficiência e respostas em tempo real. No contexto de aplicações como a API do WhatsApp Business, os webhooks permitem que servidores de aplicação, como o Laravel, recebam notificações automáticas sempre que uma nova mensagem é enviada ao número empresarial, ou quando há atualização no status da conversa (META, 2025?; CALADO, 2022).

Ao invés de precisar verificar constantemente se há novidades na API, os webhooks enviam as informações automaticamente assim que um evento ocorre. Isso torna a integração mais eficiente, reduzindo o número de requisições desnecessárias e melhorando a velocidade do atendimento. Com essa tecnologia, empresas podem automatizar processos, oferecer respostas mais rápidas e criar experiências mais personalizadas para seus clientes.(META, 2025?)

4 LARAVEL

Laravel é um framework para desenvolvimento de aplicações web em PHP, que utiliza o padrão MVC (Model-View-Controller), facilitando a organização do código e a manutenção do sistema. Ele oferece uma sintaxe elegante e expressiva, permitindo que os desenvolvedores escrevam código de forma mais clara e concisa. O framework inclui diversas funcionalidades integradas, como sistema de rotas, autenticação, cache, filas, envio de e-mails, testes automatizados, entre outras, o que o torna uma ferramenta poderosa para construção de aplicações modernas (MAIA, 2023).

Segundo Neto (2025), Laravel se destaca por oferecer uma estrutura robusta e modular que atende desde aplicações simples até sistemas corporativos complexos. Sua arquitetura baseada em serviços facilita a escalabilidade e a manutenção a longo prazo, o que o torna uma escolha frequente em projetos profissionais. Além disso, seu sistema de pacotes permite estender funcionalidades sem comprometer a estrutura central do projeto (NETO, 2025).

Laravel possui uma comunidade ativa e uma vasta documentação, o que facilita a resolução de problemas e a aprendizagem contínua (LARAVEL, 2025). Além disso, o

framework é constantemente atualizado, incorporando as melhores práticas de segurança e desenvolvimento de software moderno (PECORARO, 2015).

De acordo com Neto (2025), o Laravel se destaca por sua abordagem centrada no desenvolvedor, que proporciona uma curva de aprendizado mais suave para iniciantes, sem abrir mão da potência necessária para projetos mais complexos. O autor ressalta que a integração nativa com recursos como autenticação, cache, filas e testes automatizados torna o Laravel uma solução completa para o desenvolvimento de sistemas web modernos (NETO, 2025).

4.1 Padrão MVC

O padrão MVC foi criado em 1978 por Reenskaug no Palo Alto Research Laboratory Xerox (PARC), utilizando a linguagem Smalltalk-76 para proporcionar maior controle para os usuários sobre suas informações.Desde então, a arquitetura MVC tem sido fundamental para os desenvolvedores na construção das suas aplicações, separando-as em três componentes principais modelo, visão e Controlador.(LEMOS, 2013)

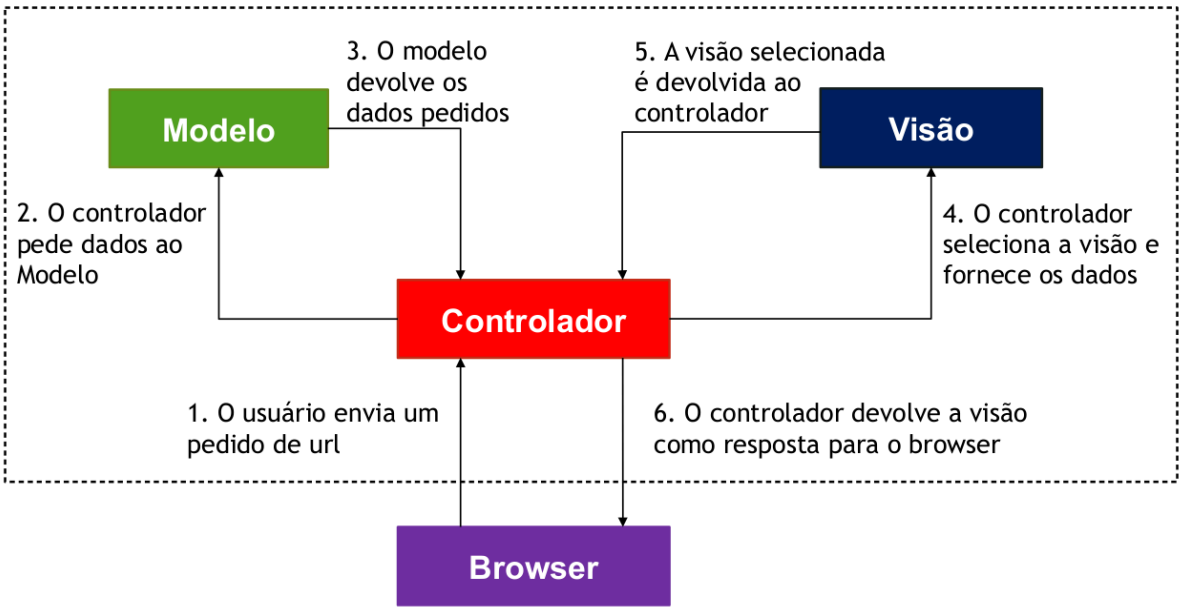
O modelo representa o estado de um aplicativo, juntamente com qualquer lógica de negócios ou operações que serão executadas por ele. O modelo contém a comunicação com os dados, que podem ser armazenados em arquivos XML, banco de dados ou outro meio com a mesma finalidade. A lógica de negócios, ou qualquer outra lógica de implementação, deve ser encapsulada no modelo para persistir o estado do aplicativo. É no modelo que reside a responsabilidade por todas as operações de CRUD. (LEMOS, 2013, SMITH, 2024)

A visão, por sua vez, é responsável por exibir as informações para o usuário final através da sua interface, a interface é responsável por toda interação com o usuário. Todas as informações são exibidas ao usuário através da visão, fornecendo e recebendo dados passados ou fornecidos pelo usuário. Ele pode enviar ou receber essas informações de muitas maneiras, como formulários, tabelas, grids e outras interações. A visão não contém regras de negócio, mas toda regra é passada do modelo através do controlador.(LEMOS, 2013, MARCIO, 2011)

O controlador administra o fluxo da aplicação, servindo como intermediário entre o modelo e a visão. Dessa forma, toda requisição fornecida pelo usuário através da visão é entregue ao controlador, que processará qual operação da camada de modelo será utilizada.(LEMOS, 2013, MARCIO, 2011)

Na figura 4.1 é mostrado como que os componentes do modelo MVC se interagem com o usuário.

Figura 4.1 – Diagrama representando a interação entre as camadas do padrão MVC



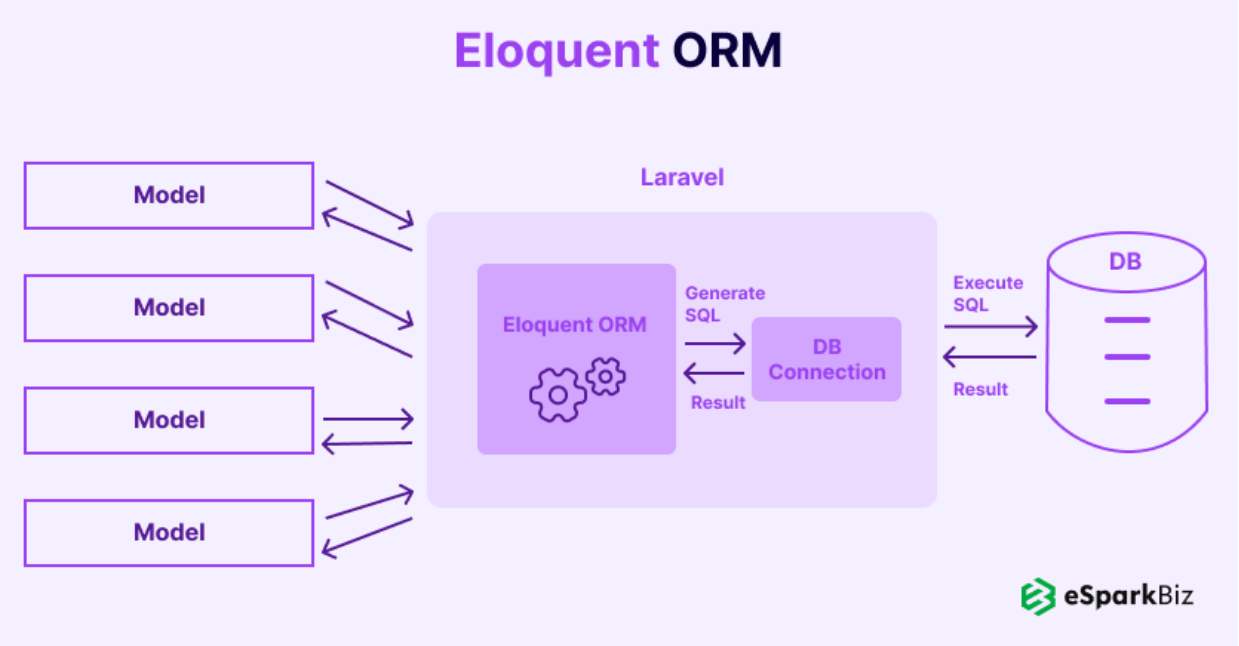
Fonte: https://vaidegrails.wordpress.com/2015/06/19/classes-de-dominio-entendendo-o-modelo-mvc

4.2 Eloquent ORM

O Laravel inclui o Eloquent, um mapeador objeto-relacional ou ORM que torna mais agradável a interação com bancos de dados por meio da implementação do ActiveRecord Pattern. Ele envolve cada tabela do banco com um wrapper permitindo que cada instância de um Modelo represente uma linha específica dessa tabela. Com métodos e propriedades que correspondem a cada célula de uma linha, o Eloquent também facilita a definição de relacionamentos entre Modelos, permitindo acessar métodos de Modelos relacionados, além de manter a sincronização automática entre as

instâncias de Modelos e dos registros no banco de dados assim atualizando e deletando as linhas necessárias. Na Figura 4.2 é mostrado como a estrutura da arquitetura do Eloquent ORM funciona, mostrando o relacionamento entre modelos e tabelas.(RODRIGUES, 2018)

Figura 4.2 – Estrutura da arquitetura do Eloquent ORM, ilustrando o padrão ActiveRecord e o relacionamento entre modelos e tabelas.



Fonte: https://www.esparkinfo.com/blog/features-of-laravel.html

5 API

Uma API (Interface De Programação de Aplicação) é um conjunto de definições e protocolos que permite a comunicação entre diferentes softwares ou sistemas. Elas fornecem uma maneira padronizada para que programas diferentes possam interagir mesmo que tenham sido desenvolvidos de forma independente. Quando um sistema deseja acessar funcionalidades de outro, ele faz isso por meio de chamadas a uma API.(MELO, 2021)

Além de padronizar a comunicação entre sistemas, as APIs promovem a reutilização de funcionalidades já desenvolvidas, contribuindo para a redução do tempo e dos custos de desenvolvimento. Elas são amplamente empregadas em diferentes contextos computacionais, como aplicações web, sistemas móveis, interações entres plataformas e em arquiteturas distribuídas, como microsserviços. Por meio de chamadas específicas, uma aplicação pode requisitar dados ou funcionalidades de outra, sem a necessidade de conhecer sua implementação interna.(EGON, 2024)

Para ilustrar, pode-se recorrer a uma analogia frequentemente utilizada na literatura técnica: uma API funciona como um cardápio de um restaurante. O cardápio descreve os pratos disponíveis, e o cliente escolhe o que deseja consumir. O garçom, nesse contexto, atua como intermediário entre o cliente e a cozinha, transmitindo o pedido e entregando o resultado. Da mesma forma, a API disponibiliza um conjunto de operações que podem ser solicitadas por um sistema, processando as requisições conforme padrões estabelecidos e retornando as respostas adequadas.(EGON, 2024)

5.1 API REST

As APIs REST são, atualmente, as interfaces mais flexíveis e amplamente utilizadas na Web, o que contribui para sua popularidade entre os desenvolvedores e arquitetos de sistemas modernos. Esse tipo de API define um conjunto de princípios e operações que permitem que clientes acessem recursos e dados hospedados em servidores por meio do protocolo HTTP.(EGON, 2024)

Cada recurso de uma API REST é representado por uma URI e pode ser manipulado por meio de métodos HTTP padronizados, como GET, POST, PUT, PATCH e DELETE. Essas operações viabilizam a criação, leitura, atualização e exclusão de dados, promovendo a interoperabilidade entre sistemas distintos.(EGON, 2024)

As APIs REST seguem o princípio de comunicação stateless, em que cada requisição do cliente ao servidor deve conter todas as informações necessárias para seu processamento, sem depender de estados armazenados entre as requisições. Isso contribui para maior escalabilidade, manutenção simplificada e melhor desempenho em aplicações distribuídas.(EGON, 2024)

São as API mais flexíveis encontradas na Web atualmente e consequentemente se tornaram a mais popular. As APIs REST definem um conjunto de funções onde os clientes podem usar para acessar os dados do servidor enviando e recebendo informações. As APIs REST podem ser utilizadas para integrar novas aplicações e projetos com sistemas de software existentes.(MELO, 2021)

5.2 Endpoint De API

Os endpoints de uma API são URLs que atuam como pontos de contato entre o cliente e o servidor, permitindo que o cliente acesse os dados ou funcionalidades do servidor. Cada endpoint corresponde a um recurso específico da API que é utilizado por métodos HTTP para realizar suas operações. Além disso, os endpoints podem exigir parâmetros, cabeçalhos e autenticação para garantir que as solicitações sejam processadas corretamente e com segurança.(NOSOWITZ; GOODWIN, 2024)

5.3 Autenticação e Autorização de uma API

A autenticação e a autorização são dois conceitos fundamentais na segurança de APIs REST. Enquanto a autenticação verifica a identidade do usuário ou sistema que está tentando acessar a API, a autorização determina quais recursos esse usuário ou sistema pode acessar. Esses processos são essenciais para proteger dados sensíveis e garantir que apenas usuários autorizados tenham permissão para interagir com determinados recursos de API.(COSTA, 2023)

5.3.1 Token de Autenticação

Um token de autenticação é um valor criptografado gerado pelo servidor após a validação das credenciais do usuário. Esse token é enviado junto a cada requisição subsequente para provar que o usuário já foi autenticado. Essa abordagem é amplamente utilizada por sua facilidade de implementação e escalabilidade.(COSTA, 2023)

6 INTELIGÊNCIA ARTIFICIAL

A Inteligência Artificial (IA) é um ramo da ciência da computação voltado ao desenvolvimento de sistemas capazes de simular o raciocínio humano, por meio de algoritmos que processam grandes volumes de dados (MORAIS; CASTELO BRANCO, 2023).Seu objetivo é permitir que máquinas aprendam, analisem, interpretem e tomem decisões de forma autônoma.

O termo foi introduzido por John McCarthy em 1956, mas os fundamentos da área foram estabelecidos anteriormente por Alan Turing, considerado o pai da computação moderna (MORAIS; CASTELO BRANCO, 2023).

A IA pode ser classificada em dois tipos principais: a **IA fraca**, voltada para tarefas específicas, como assistentes virtuais; e a **IA forte**, que busca simular capacidades cognitivas humanas completas, como o raciocínio e a autoconsciência (MORAIS; CASTELO BRANCO, 2023).

Entre suas abordagens estão:

- a **IA simbólica**, baseada em regras e lógica formal;
- a **IA conexionista**, inspirada no funcionamento do cérebro humano (ex.: *deep learning*);
- e a **IA evolucionária**, que utiliza mecanismos similares à seleção natural (MORAIS; CASTELO BRANCO, 2023).

A IA já está presente em diversas áreas, como saúde, indústria, educação e comunicação, mas também levanta questões éticas e sociais quanto à sua regulamentação e ao impacto no mercado de trabalho (MORAIS; CASTELO BRANCO, 2023).

6.1 Inteligência Artificial Generativa

A Inteligência Artificial Generativa (IA generativa) é um subcampo da IA que se concentra na criação de novos conteúdos, como textos, imagens, vídeos, músicas, códigos e dados sintéticos, com base em entradas fornecidas e padrões aprendidos durante o treinamento (DATA SCIENCE ACADEMY, 2025). Essa tecnologia utiliza redes neurais artificiais e algoritmos de aprendizado profundo para gerar saídas originais que imitam o estilo ou estrutura dos dados com os quais foi treinada (DATA SCIENCE ACADEMY, 2025).

Diferente da IA discriminativa, voltada para classificação ou previsão, a IA generativa cria novas informações a partir do aprendizado, ampliando significativamente o escopo de aplicação da inteligência artificial (DATA SCIENCE ACADEMY, 2025).

Modelos como **GPT-4**, **DALL·E** e **Stable Diffusion** são exemplos notáveis dessa tecnologia.

Entre os principais modelos estão:

- **LLMs (Large Language Models)**, como o GPT-4, usados para gerar textos contextuais;
- **Transformers**, eficientes no processamento de linguagem natural;
- **GANs (Redes Adversárias Generativas)**, amplamente utilizados na criação de imagens e vídeos;
- **VAEs (Autoencoders Variacionais)**, que reduzem e recriam dados complexos.

A IA generativa possui aplicações práticas em diversas áreas: produção de conteúdo textual, desenvolvimento de software, design gráfico, tradução automática, geração de dados sintéticos, educação, saúde, publicidade, finanças e jogos (DATA SCIENCE ACADEMY, 2025).

Um dos aspectos essenciais da interação com sistemas generativos é o uso de **prompts** — comandos escritos em linguagem natural com o objetivo de orientar a IA na criação de uma resposta específica (VALERI; TOLEDO, 2024). O prompt é analisado com base nos parâmetros aprendidos durante o treinamento e influencia diretamente a precisão e relevância da resposta gerada.

A elaboração de prompts eficazes envolve:

- Fornecimento de contexto adequado;
- Clareza e especificidade na formulação;
- Uso de exemplos concretos;
- Realização de ajustes com base nas respostas (VALERI; TOLEDO, 2024).

Por exemplo, ao utilizar um prompt como *“Gere uma imagem no estilo futurista de uma cidade submersa com torres translucidas”*, modelos como o **DALL·E 2** podem produzir uma imagem original com base na descrição textual (VALERI; TOLEDO, 2024).

Além disso, o uso contínuo e bem estruturado de prompts permite ao usuário obter **respostas mais coerentes, completas e contextualizadas**, otimizando o uso da

IA generativa em ferramentas como o ChatGPT, Gemini, ou Cohere (VALERI; TOLEDO, 2024).

Diferentemente do prompt de comando utilizado em sistemas operacionais, que exige sintaxe técnica e precisa, o prompt em IA generativa é mais flexível e conversacional, tornando a tecnologia acessível mesmo a pessoas sem formação técnica (VALERI; TOLEDO, 2024).

6.2 OpenAi

Desde que foi lançado em novembro de 2022 o ChatGPT tem cativado a atenção global, com o lançamento da API ChatGPT da OpenAI podemos facilmente integrar recursos de IA convencional em nossos aplicativos e sistemas. (ALI, 2023)

6.2.1 ChatGPT

O Generative Pré-trained Transformer conhecido como GPT é uma série de modelos de linguagem desenvolvida pela OpenAI. Esses modelos, que foram aprimorados desde o GPT-1 até o GPT-4, são treinados com grandes volumes de texto e com um uso bem mais sofisticado para tarefas linguísticas específicas que têm destaque na geração de textos coerentes prevendo palavras subsequentes, facilitando a interpretação de contexto. O ChatGPT é uma IA baseada em todos esses modelos, que se comunica em linguagem natural para ser seguro, confiável e informativo.(ALI, 2023)

6.2.2 API ChatGPT

A API ChatGPT da OpenAI fornece acesso aos modelos de IA conversacional, permitindo que os desenvolvedores integrem essas avançadas capacidades de processamento de linguagem natural em seus próprios aplicativos. Com essas APIs é possível criar funcionalidades inovadoras e úteis, como chatbots, assistentes virtuais, automação de fluxos de trabalho de suporte ao cliente e geração de conteúdos variados.(ALI, 2023)

A API ChatGPT destaca-se na sua compreensão de linguagem natural, tendo uma alta capacidade de gerar respostas contextualmente relevantes e responder a perguntas de acompanhamento, tornando-a uma ferramenta poderosa para melhorar a interatividade e eficiências de projetos.(ALI, 2023)

6.3 Gemini

O Gemini e uma nova tecnologia de Inteligência artificial multimodal generativa desenvolvido pela Google DeepMind se destacando como uma nova linha de modelos de IA voltadas para múltiplas modalidades e raciocínio avançado, prometendo superar modelos de IA especializados em uma única tarefa e expandindo as fronteiras de IA.(CARMAGO, 2025)

O Gemini não apenas manipula palavras como as outras IA, mas também compreende o contexto, conceitos, emoções, intenções e relações subjacentes aos dados. Essa capacidade de compreensão contextual, juntamente com seus mecanismos avançados de atenção, permite que o Gemini responda com mais fidelidade às consultas dos usuários, fornecendo saídas mais consistentes e relevantes.(TAVARES, 2023)

6.4 Cohere

Fundada em 2019, a Cohere é uma empresa de tecnologia baseada em inteligência artificial que desenvolve modelos de linguagem natural para atender ao ambiente corporativo. Distinguindo-se da concorrência por possuir um modelo de AI aberto, a empresa oferece uma IA integrada que é segura e personalizada o suficiente para uma inserção eficaz e, ao mesmo tempo, mantém o controle de privacidade e interações adequadas. (DAVID, 2024).

Um dos principais produtos da empresa é o modelo **Command R+**, lançado em 2024. Com 104 bilhões de parâmetros e uma janela de contexto de até 128 mil tokens, o Command R+ foi projetado para tarefas de geração aumentada por recuperação (RAG), permitindo a geração de textos baseados em informações recuperadas em tempo real de bases de dados específicas (GOMEZ, 2024).

O Command R+ destaca-se por seu desempenho em tarefas empresariais, como sumarização de documentos, resposta a perguntas baseadas em informações e geração de respostas multilíngues. Além disso, o modelo é capaz de interagir com diversas aplicações, como calculadoras para resolver problemas matemáticos ou clientes de e-mail para compartilhar documentos, focando em aplicações práticas no ambiente de negócios (BOOTH, 2024).

A Cohere também lançou a iniciativa **Aya**, visando expandir o desempenho de modelos de linguagem em idiomas além do inglês. Os modelos Aya Expanse 8B e 35B, por exemplo, foram projetados para oferecer capacidades multilíngues de última geração, superando modelos semelhantes de empresas como Google e Meta em testes de benchmark multilíngues (DAVID, 2024).

A estratégia da Cohere de fornecer modelos com pesos abertos e foco em aplicações empresariais tem atraído parcerias com grandes empresas, como Oracle e Accenture, destacando a utilidade dos modelos da Cohere em ambientes profissionais (BOOTH, 2024).

7 ESTUDO DE CASO

Esse capítulo detalha como o chatbot foi desenvolvido, serão apresentados detalhes da implementação de cada ferramenta utilizada no desenvolvimento. É apresentado como as ferramentas que compõem a construção do chatbot interagem, detalhando como o fluxo de informações é tratado desde a mensagem inicial até a resposta gerada pela IA enviada ao usuário.

7.1 Configuração da API do Whatsapp Business

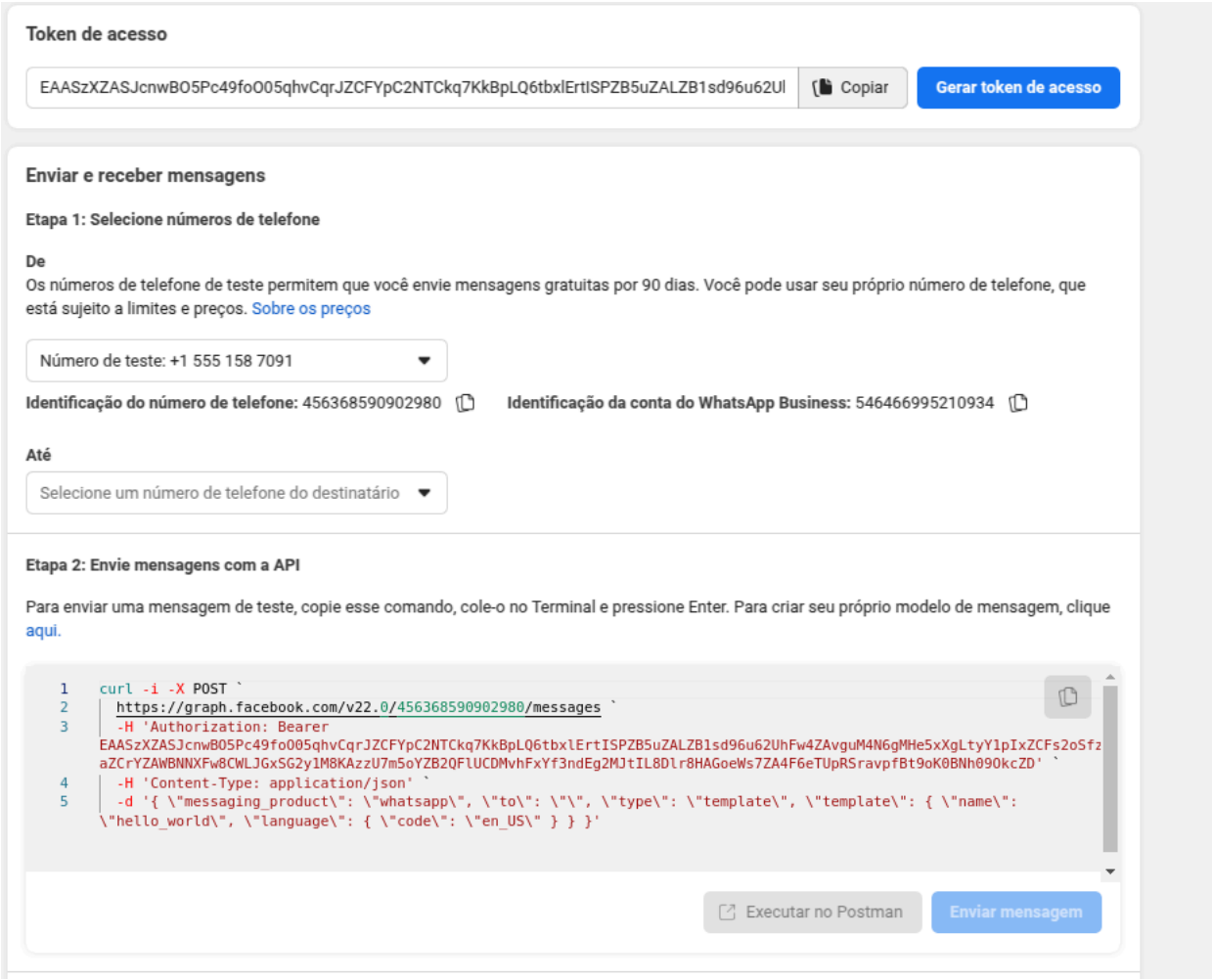
Este tópico detalha o processo de configuração da API do WhatsApp Business, um componente fundamental para a implementação do chatbot.

Antes de iniciar a configuração, é essencial garantir que os pré-requisitos sejam atingidos. Primeiramente é necessário uma conta de desenvolvedor da Meta obtida através da plataforma de desenvolvedores da Meta. Em seguida é necessário criar um aplicativo do tipo Empresa dentro da plataforma.

Após a criação do aplicativo, é possível adicionar o produto do WhatsApp entre os diversos produtos disponibilizados pela Meta. Esse processo permite a criação automática de uma conta de teste do WhatsApp Business. Dentro do produto WhatsApp, é necessário cadastrar um número de destinatário que será utilizado para os testes. Esse número deve ser verificado por meio de um código de confirmação enviado pelo WhatsApp. Uma vez confirmado, o número estará apto a receber mensagens enviadas pela API.

Para testar a API, basta acessar o painel do produto WhatsApp e enviar uma mensagem para o número cadastrado. A API disponibiliza vários modelos de teste, sendo o mais básico o tradicional “Hello World”. O envio pode ser feito diretamente pela meta conforme mostrado na Figura 7.1 ou através de qualquer ferramenta que permite realizar uma requisição HTTP POST.

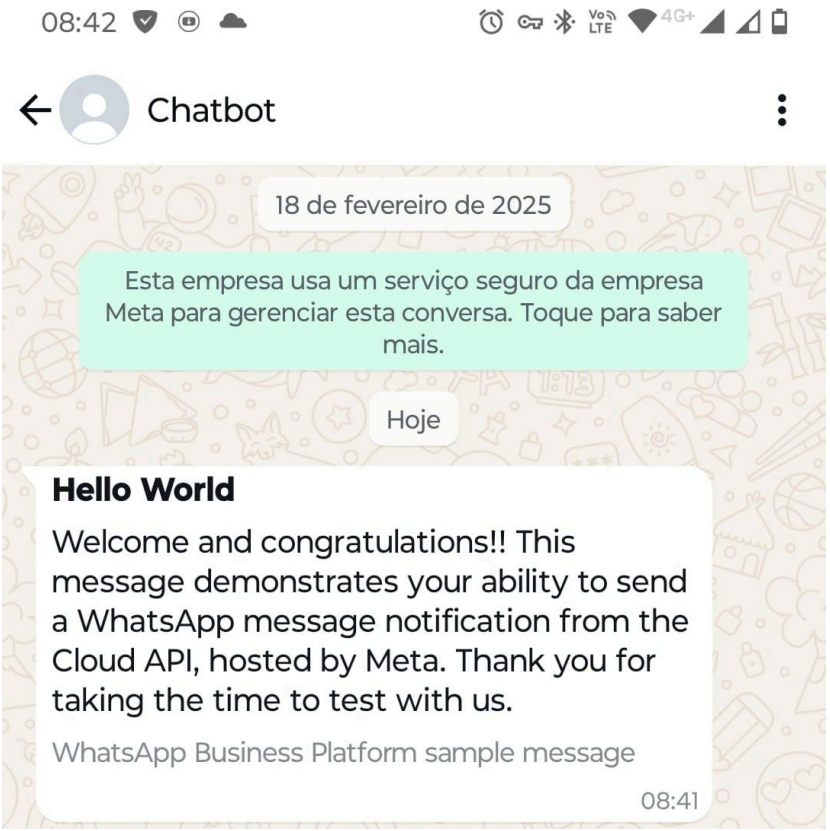
Figura 7.1 - Envio de mensagem pela API do WhatsApp Business.



Fonte: Elaborada pelo Autor

Ao enviar a mensagem ilustrada na Figura 7.2, recebemos um modelo simples de “Hello World” no número de destinatário que foi escolhido. Esse processo permite validar o funcionamento da API e garante que o ambiente esteja corretamente configurado para a continuidade d o desenvolvimento do chatbot.

Figura 7.2 – Recebimento da mensagem enviada pela API do WhatsApp Business



Fonte: Elaborada pelo Autor

7.2 Desenvolvimento do Servidor Laravel

Neste tópico, será descrito o processo de criação de um servidor Laravel para receber mensagens enviadas pela API do WhatsApp Business através do webhook. Este servidor é essencial para processar e responder as mensagens recebidas.

A criação do servidor foi realizada utilizando o framework PHP Laravel, versão 11. O laravel foi escolhido por sua robustez, facilidade de integração com APIs externas e sua arquitetura MVC, que favorece a organização do código e a separação de responsabilidades.

A primeira etapa do desenvolvimento foi a criação de uma rota pública que pudesse ser utilizada pela API do WhatsApp Business para a verificação do webhook. Esse processo é necessário para validar o endpoint e garantir que ele está apto a receber mensagens. Para isso, foi criado uma rota do tipo GET, responsável por retornar o challenge enviado pela Meta sempre que uma requisição de verificação é feita.

Conforme a Figura 7.3 essa rota foi configurada no arquivo routes/api.php, apontando para o controller WhatsAppController.

Figura 7.3 – Definição das rotas de verificação e recebimento de mensagens no Laravel

```php
<?php

use App\Http\Controllers\Api\UserController;
use App\Http\Controllers\WhatsAppController;
use Illuminate\Support\Facades\Route;

Route::get( uri: '/users', [UserController::class, 'index']);

Route::get( uri: '/webhook', [WhatsAppController::class, 'verificarToken']);
Route::post( uri: '/webhook', [WhatsAppController::class, 'receiveMessage']);
```

Fonte: Elaborada pelo Autor

No método verifyToken, o servidor compara o token recebido na requisição com o token definido previamente no ambiente da aplicação. Se os tokens coincidirem, o servidor retorna o parâmetro hub_challenge, confirmando que o webhook foi verificado com sucesso. Na Figura 7.4 mostra o trecho do código responsável por fazer essa verificação buscando o token do arquivo .env.

Figura 7.4 – Autenticação do webhook da API do WhatsApp no servidor Laravel

```php
1 usage
public function verificarToken(Request $request)
{
    Log::info( message: 'Dados recebidos no Webhook de Verificação:', $request->all());
    if ($request->input( key: 'hub_mode') === 'subscribe' && $request->input( key: 'hub_verify_token') === env( key: 'WHATSAPP_ACCESS_TOKEN')) {
        Log::info( message: "Validação de Token Realizada com Sucesso");
        return response($request->input( key: 'hub_challenge'), status: 200);
    } else {
        Log::error( message: "Erro na Validação de Token");
        return response( content: 'Forbidden', status: 403);
    }
}
```

Fonte: Elaborada pelo Autor

Com essa verificação implementada corretamente, o webhook pode ser registrado na plataforma da Meta, permitindo que o servidor Laravel receba as mensagens enviadas pelo WhatsApp Business API.

Após a verificação do token, foi criada a rota responsável por receber as mensagens enviadas pela API do WhatsApp Business. Essa rota, do tipo POST, serve como ponto de entrada do webhook e é acionada sempre que uma nova mensagem é enviada ao número cadastrado no produto WhatsApp da Meta.

A Configuração da rota foi realizada no arquivo routes/api.php, apontando para o método receiveMessage do WhatsAppController.

O método receiveMessage foi desenvolvido para processar o payload enviado pela Meta. A estrutura da requisição contém diversas informações, como número do remetente, o tipo de mensagem e o conteúdo da mensagem. O servidor realiza o parsing desses dados e, com base nisso, inicia o fluxo de processamento da solicitação. Na figura 7.5 é mostrado a função receiveMessage, e como ficou a implementação do método principal desse projeto.

Figura 7.5 – Função receiveMessage para o processamento de mensagens recebidas

```php
public function receiveMessage(Request $request)
{
    if (!$request->input( key: 'entry.0.changes.0.value.statuses') && $request->input( key: 'entry.0.changes.0.value.messages')) {
        Log::info( message: 'Dados recebidos no Webhook:', $request->all());
        $contato = $this->contatoCommand->criarContato($request);
        $mensagem = $this->tratarMensagemService->montarMensagem($request);
        HistoricoDeMensagem::create(['hi_mensagem' => $mensagem['mensagemTexto']]);

        if ($this->contatoCommand->contatoAceitouLgpd($contato, $mensagem)) {
            $intencao = $this->iaService->identificarIntencao($contato, $mensagem);

            if ($intencao == 'N') {
                $this->whatsAppService->enviarMensagemDeTexto($contato, $mensagem, texto: "Não consegui identificar a intenção da mensagem");
                $this->enviarListaDeApiSeNecessario($contato);

            } else if ($intencao == 'C') {
                $parametros = $this->iaService->montarJsonDeParametros($contato, $mensagem);
                $retornoDaExecucao = $this->executarServicoService->executar($contato->servico, $parametros);
                $this->whatsAppService->enviarMensagemDeTexto($contato, $mensagem, $retornoDaExecucao);

            } else {
                $this->logService->logInfo( txtTitle: "serviço encontrado", [$intencao]);
                $servicoAtual = Servico::where('sv_codigo', $intencao)->first();
                $this->contatoCommand->adicionarServico($contato, filter_var($intencao, filter: FILTER_VALIDATE_INT));
                $mensagemDaApiDeServico = $this->iaService->montarMensagemDaApiDeServico($servicoAtual);
                $this->whatsAppService->enviarMensagemDeTexto($contato, $mensagem, $mensagemDaApiDeServico);
            }
        }
    }
}
```

Fonte: Elaborada pelo Autor

Antes de qualquer interação com a inteligência artificial, o servidor verifica se o contato que enviou a mensagem já está registrado no banco de dados.Caso não esteja,

um novo registro é criado automaticamente com as informações recebidas. Na Figura 7.6 mostra como é criado o cadastro de contatos no banco de dados, com base nos próprios dados vindos da API do WhatsApp, podemos identificar o nome e telefone do usuário, o número é usado para buscar no banco o contato se não houver será cadastrado um novo usando os dados mencionados.

Figura 7.6 – Código para criação automática de um novo contato no banco de dados

```
1 usage
public function criarContato(Request $request)
{
    $numero = $request->input( key: 'entry.0.changes.0.value.contacts.0.wa_id');
    $nome = $request->input( key: 'entry.0.changes.0.value.contacts.0.profile.name');

    $this->logService->logInfo( txtTitle: 'CADASTRO DE CONTATO', [$numero, $nome]);

    $contato = Contato::firstOrCreate(
        ['con_numero' => $numero],
        ['con_nome' => $nome, 'con_aceitou_lgpd' => false]
    );

    $this->logService->logInfo( txtTitle: 'CONTATO ', [$contato]);

    return $contato;
}
```

Fonte: Elaborada pelo Autor

Em seguida, o sistema verifica se o usuário já aceitou os Termos de Uso e a Política de Privacidade, conforme exigido pela LGPD (Lei Geral de Proteção de Dados). Essa validação é realizada por meio do método `contatoAceitouLgpd`, que checa se o campo `con_aceitou_lgpd` do contato está marcado como verdadeiro.

Caso o usuário ainda não tenha fornecido o consentimento, o sistema envia automaticamente uma mensagem solicitando essa autorização. Se a resposta do usuário indicar concordância, o sistema confirma o aceite, registra essa informação no banco de dados e envia uma mensagem de boas-vindas. A partir desse ponto, o contato está apto a interagir com os serviços oferecidos pelo chatbot.

Na Figura 7.7 mostra como é feito a verificação da mensagem LGPD, enquanto o usuário não aceitar os termos o uso dos serviços ficará indisponível.

Figura 7.7 – Código de validação e registro do aceite da LGPD pelo usuário

```php
public function contatoAceitouLgpd($contato, $mensagem)
{
    if (!$contato->con_aceitou_lgpd) {
        if (!$this->contatoJaRecebeuMensagemDeLgpd($contato)) {
            $this->logService->logInfo( txtTitle: 'MENGAGEM RECEBIDA ', [$mensagem]);

            if ($this->mensagemDoUsuarioConfirmaLgpd($mensagem)) {
                $this->whatsAppService->enviarMensagemDeTexto(
                    $contato,
                    $mensagem,
                    texto: "Obrigado por aceitar nossa política de privacidade. Agora no que posso te ajudar."
                );
                $contato->con_aceitou_lgpd = true;
                $contato->save();
                return false;
            }
            $this->whatsAppService->enviarMensagemDeTexto(
                $contato,
                $mensagem,
                texto: "Para continuar, é necessário aceitar a nossa política de privacidade."
            );
        }
        $this->whatsAppService->enviarMenu(
            $contato,
            $mensagem,
            codigo_do_menu: 0
        );
    }

    return $contato->con_aceitou_lgpd;
}
```

Fonte: Elaborada pelo Autor

7.3 Integração da Inteligência Artificial no Servidor Laravel

Após a validação do webhook da API do WhatsApp Business, o servidor Laravel está preparado para receber notificações de novas mensagens enviadas por usuários. O processamento se inicia com a verificação do token de autenticação enviado pela Meta, que garante que a requisição é legítima.

Quando o webhook é acionado, o sistema extrai do corpo da requisição (payload) informações fundamentais, como o número de telefone do usuário, o tipo da mensagem e o conteúdo textual. Antes de qualquer interação com a camada de inteligência artificial (IA), o sistema realiza duas verificações:

1. **Verificação de cadastro do contato:** Se o número de telefone do usuário já está presente na tabela `chat.contato`. Caso não esteja, um novo registro é automaticamente criado.
2. **Consentimento de uso de dados:** É verificado se o campo `con_aceitou_lgpd` está marcado como verdadeiro. Caso contrário, o sistema envia uma solicitação automática de consentimento, conforme exigido pela Lei Geral de Proteção de Dados (LGPD).

Com as validações concluídas, a mensagem do usuário é encaminhada para o módulo de interpretação por inteligência artificial. Esse módulo é responsável por analisar a linguagem natural utilizada e determinar a intenção da mensagem. O sistema oferece suporte a múltiplos modelos de IA, sendo possível alternar entre ChatGPT (OpenAI), Gemini (Google) e Cohere, por meio da variável de configuração definida no arquivo `.env`.

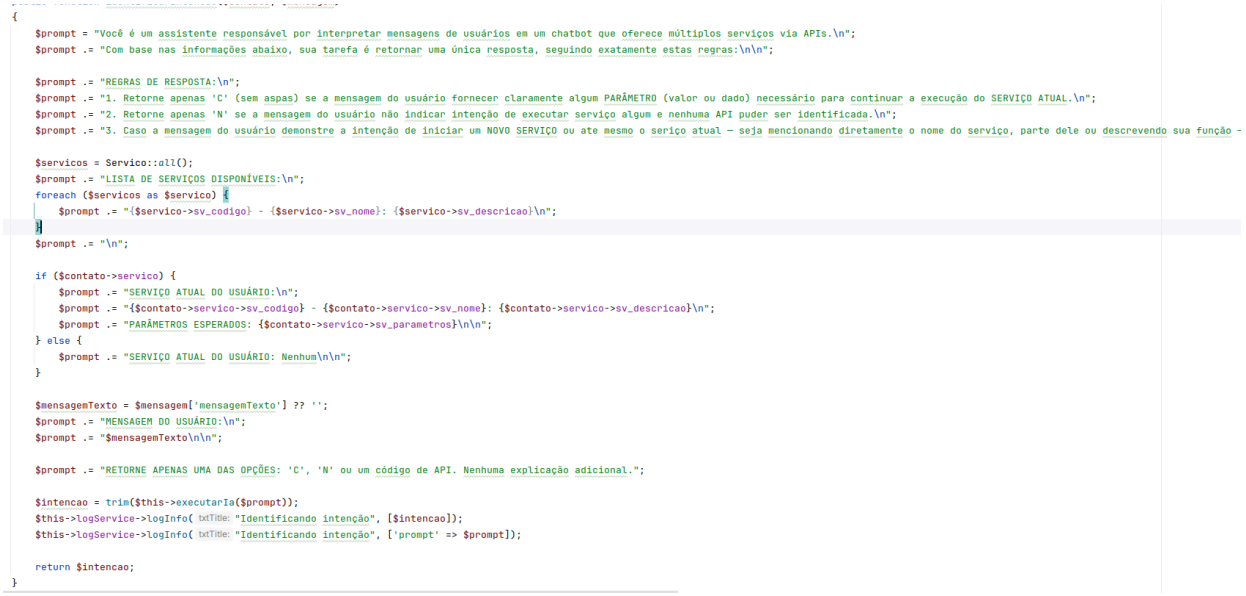
A integração com o provedor de IA é realizada por meio de requisições HTTP POST. O corpo dessas requisições é construído dinamicamente e contém apenas um prompt textual, formatado em linguagem natural.

Esse prompt é montado de forma programática no backend e inclui, em seu conteúdo:

- A **mensagem original enviada pelo usuário**;
- A **lista de serviços disponíveis**, extraída da tabela `chat.servico`;
- **Instruções claras** para que a IA identifique, com base no conteúdo, qual serviço corresponde à intenção expressa pelo usuário.

A Figura 7.8 ilustra um exemplo prático da construção desse prompt, demonstrando como as informações são organizadas em linguagem natural antes de serem enviadas à API de inteligência artificial.

Figura 7.8 – Envio da mensagem para a IA e interpretação do retorno



```
{
    $prompt = "Você é um assistente responsável por interpretar mensagens de usuários em um chatbot que oferece múltiplos serviços via APIs.\n";
    $prompt .= "Com base nas informações abaixo, sua tarefa é retornar uma única resposta, seguindo exatamente estas regras:\n\n";

    $prompt .= "REGRAS DE RESPOSTA:\n";
    $prompt .= "1. Retorne apenas 'C' (sem aspas) se a mensagem do usuário fornecer claramente algum PARÂMETRO (valor ou dado) necessário para continuar a execução do SERVIÇO ATUAL.\n";
    $prompt .= "2. Retorne apenas 'N' se a mensagem do usuário não indicar intenção de executar serviço algum e nenhuma API puder ser identificada.\n";
    $prompt .= "3. Caso a mensagem do usuário demonstre a intenção de iniciar um NOVO SERVIÇO ou ate mesmo o seriço atual — seja mencionando diretamente o nome do serviço, parte dele ou descrevendo sua função -

    $servicos = Servico::all();
    $prompt .= "LISTA DE SERVIÇOS DISPONÍVEIS:\n";
    foreach ($servicos as $servico) {
        $prompt .= "{$servico->sv_codigo} - {$servico->sv_nome}: {$servico->sv_descricao}\n";
    }
    $prompt .= "\n";

    if ($contato->servico) {
        $prompt .= "SERVIÇO ATUAL DO USUÁRIO:\n";
        $prompt .= "{$contato->servico->sv_codigo} - {$contato->servico->sv_nome}: {$contato->servico->sv_descricao}\n";
        $prompt .= "PARÂMETROS ESPERADOS: {$contato->servico->sv_parametros}\n\n";
    } else {
        $prompt .= "SERVIÇO ATUAL DO USUÁRIO: Nenhum\n\n";
    }

    $mensagemTexto = $mensagem['mensagemTexto'] ?? '';
    $prompt .= "MENSAGEM DO USUÁRIO:\n";
    $prompt .= "$mensagemTexto\n\n";

    $prompt .= "RETORNE APENAS UMA DAS OPÇÕES: 'C', 'N' ou um código de API. Nenhuma explicação adicional.";

    $intencao = trim($this->executarIa($prompt));
    $this->logService->logInfo( txtTitle: "Identificando intenção", [$intencao]);
    $this->logService->logInfo( txtTitle: "Identificando intenção", ['prompt' => $prompt]);

    return $intencao;
}
```

Fonte: Elaborada pelo Autor

Ao receber o prompt, a inteligência artificial (IA) realiza a análise semântica da mensagem e retorna uma resposta estruturada. Essa resposta pode assumir uma das seguintes formas:

- **Código de um serviço específico** (ex.: `SV001`), quando a intenção do usuário corresponde diretamente a um serviço cadastrado no sistema;

- **Caractere “C”**, utilizado para indicar que a mensagem é uma continuação de uma interação anterior — normalmente o envio de parâmetros complementares;

- **Caractere “N”**, quando a IA não consegue identificar uma intenção válida, seja por falta de clareza, ambiguidade ou ausência de correspondência com os serviços disponíveis.

Nos casos em que a IA retorna o caractere “N”, o chatbot informa ao usuário que não foi possível compreender a solicitação. Se essa situação ocorrer três vezes consecutivas, o sistema responde automaticamente com a lista completa de serviços disponíveis, como forma de guiar o usuário para uma nova tentativa de interação bem-sucedida.

Quando a IA identifica corretamente um serviço (retornando seu código), o sistema armazena esse código no campo `con_servico` do contato ativo e consulta, na tabela `chat.servico`, os parâmetros necessários para a execução do serviço solicitado. Esses parâmetros — tanto obrigatórios quanto opcionais — são então utilizados para construir um novo prompt que é enviado à IA. Esse prompt instrui a IA a redigir uma mensagem de solicitação clara e humanizada, pedindo ao usuário os dados exigidos.

Esse processo de montagem do prompt é realizado pela função `montarJsonDeParametros`, que concatena instruções específicas com os parâmetros esperados e o conteúdo da mensagem do usuário. A resposta da IA, que deve conter somente os dados solicitados em formato JSON, é então tratada e convertida em um objeto PHP para uso na requisição ao serviço.

A Figura 7.9 ilustra essa função, destacando tanto a criação do prompt quanto o tratamento posterior do JSON retornado pela IA.

Figura 7.9 – Função montarJsonDeParametros responsável por construir o prompt com parâmetros e processar o retorno da IA em formato JSON

```php
public function montarJsonDeParametros($contato, $mensagem)
{
    $servico = $contato->servico;
    $mensagemString = "O usuário selecionou um serviço que necessita desses parâmetros:\n";
    $mensagemString .= $servico->sv_parametros . "\n\n";
    $mensagemString .= "Com base nisso, identifique os parâmetros na mensagem do usuário e retorne somente um JSON com as informações fornecidas pelo usuário, ";
    $mensagemString .= "formate cade parâmetro de acordo com a informação obtida:\n'";
    $mensagemString .= $mensagem['mensagemTexto'] . "'\n\n";

    $resposta = $this->executarIa($mensagemString);


    TRATAMENTO DO JSON RECEBIDO PELA IA PARA SER UTILIZADO EM REQUISIÇÕES HTTP
    $resposta = str_replace( search: '`', replace: '', $resposta);
    $resposta = str_replace( search: 'json', replace: '', $resposta);
    $resposta = json_decode($resposta, associative: true);

    return $resposta;
}
```

Fonte: Elaborada pelo Autor

Conforme o usuário envia as informações solicitadas, o conteúdo de cada mensagem é novamente processado pela IA. O objetivo é transformar os dados recebidos em um objeto JSON estruturado, com os campos devidamente preenchidos de acordo com as exigências do serviço. Essa estrutura é validada com base nas regras definidas previamente no campo `sv_parametros` da tabela de serviços.

Após todos os parâmetros estarem coletados e organizados, o sistema executa a função `executaServico`, responsável por realizar uma requisição HTTP ao endpoint especificado no campo `sv_url`, utilizando o método definido (`GET`, `POST`, etc.). O retorno da requisição — geralmente em formato JSON — é processado e enviado novamente à IA, que interpreta os dados e gera uma mensagem de resposta final personalizada para o usuário.

Essa mensagem final, já formatada de forma clara e acessível, é então enviada pelo chatbot através da API do WhatsApp Business. O processo se conclui com o envio de uma notificação de leitura, garantindo que o usuário tenha ciência do término da operação.

Essa abordagem adaptativa, baseada em inteligência artificial, elimina a necessidade de menus rígidos ou comandos específicos, proporcionando uma **experiência fluida, contextual e acessível** mesmo para usuários sem conhecimentos técnicos.

7.3.1 Integração com o ChatGPT

Neste tópico é descrito o processo de integração da aplicação chatbot com a API do ChatGPT, fornecida pela empresa OpenAI. Essa integração permite que o sistema utilize um modelo de linguagem natural para interpretar mensagens enviadas pelos usuários e gerar respostas inteligentes, adaptadas ao contexto da conversa.

O primeiro passo consiste em acessar o portal oficial da OpenAI, disponível em https://platform.openai.com/, onde estão disponíveis a documentação técnica, os recursos de gerenciamento de chaves e os modelos disponíveis para integração.

Para habilitar a comunicação com a API, é necessário:

1. Criar uma conta no portal;
2. Cadastrar uma forma de pagamento válida;
3. Gerar uma chave de API (API Key);
4. Adquirir créditos, consumidos conforme o volume de tokens processados por requisição.

O modelo de inteligência artificial utilizado neste projeto foi o gpt-3.5-turbo, disponibilizado pela OpenAI. A escolha se deu por sua alta disponibilidade, bom desempenho e baixo custo operacional, características que o tornam adequado para aplicações em ambientes de produção e desenvolvimento.

As interações com esse modelo são realizadas por meio de requisições HTTP do tipo POST, enviadas ao endpoint oficial da API da OpenAI. Para que a requisição seja válida e processada corretamente, é necessário incluir dois cabeçalhos principais:

O primeiro é o cabeçalho de autorização, no formato conhecido como *Bearer Token*. Nesse campo, a aplicação envia a chave de API gerada no painel da OpenAI, que funciona como um identificador seguro e pessoal do desenvolvedor. Sem essa chave, a requisição é recusada pelo servidor.

O segundo cabeçalho obrigatório é o que define o tipo de conteúdo da requisição, especificando que os dados estão estruturados em formato JSON. Essa configuração garante que a API consiga interpretar corretamente as instruções e parâmetros fornecidos pela aplicação.

Com esses dois elementos – autenticação e estrutura de conteúdo – a aplicação é capaz de enviar mensagens para o modelo de linguagem, receber as respostas geradas e, assim, utilizar a IA de forma integrada ao fluxo do chatbot. Essa integração é essencial para permitir que a aplicação interprete mensagens em linguagem natural e responda de forma dinâmica e contextualizada aos usuários.

A montagem e envio do corpo da requisição para a API do ChatGPT são realizados por uma função específica no backend da aplicação, responsável por preparar os dados e realizar a chamada ao serviço externo de inteligência artificial. Essa função, além de construir a estrutura da requisição, também processa a resposta recebida da IA, extraindo apenas o conteúdo relevante que será utilizado pelo chatbot.

A Figura 7.10 apresenta a função `executarIaOpenAi`, que encapsula todo o processo de comunicação com a API da OpenAI. Nessa função, o conteúdo textual do prompt previamente montado com base na mensagem do usuário é enviado ao modelo `gpt-3.5-turbo`.

Figura 7.10 – Função executarIaOpenAi responsável por enviar o prompt à API do ChatGPT e processar a resposta gerada

```php
private function executarIaOpenAi($prompt)
{
    $response = Http::baseUrl( url: "")->withHeaders([
        'Authorization' => 'Bearer ' . env( key: "OPENAI_API_KEY"),
        'Content-Type' => 'application/json'
    ])->post(env( key: "OPENAI_API_URL"), [
        "model" => "gpt-3.5-turbo",
        "messages" => [
            ["role" => "user", "content" => $prompt]
        ]
    ]);

    $this->logService->logInfo( txtTitle: "Resposta da IA OpenAI", [$response]);

    return $response->json()['choices'][0]['message']['content'];
}
```

Fonte: Elaborada pelo Autor

Os cabeçalhos de autenticação são definidos com base nas variáveis de ambiente armazenadas no arquivo `.env`, garantindo a segurança e a flexibilidade na gestão das credenciais da API. A requisição é enviada utilizando o método P0ST, conforme exigido pela documentação da OpenAI, e os dados são estruturados em conformidade com o formato esperado.

Após o envio da requisição, o conteúdo da resposta é extraído, sendo isolada apenas a parte textual gerada pela IA. Esse texto representa a interpretação do modelo sobre o prompt enviado e será utilizado como resposta direta ao usuário ou como instrução intermediária no fluxo da aplicação.

Esse processo é fundamental para o funcionamento do chatbot, pois transforma a mensagem do usuário em um dado semântico compreendido pelo modelo de IA, e permite que o sistema responda de forma autônoma, contextualizada e eficiente.

7.3.2 Integração com o GEMINI

Este tópico apresenta o processo de integração da aplicação chatbot com a API do Gemini, um serviço de inteligência artificial desenvolvido pela Google, utilizado para

interpretar mensagens em linguagem natural e gerar respostas automatizadas de forma contextual.

O acesso à API é feito por meio da plataforma oficial da Google, disponível em: https://ai.google.dev/, onde são disponibilizadas as instruções de uso, recursos de autenticação e a documentação técnica detalhada. A documentação específica da API pode ser acessada diretamente no endereço: https://ai.google.dev/gemini-api/docs?hl=pt-br.

Para utilizar a API Gemini, é necessário gerar uma chave de autenticação (API Key). Isso é feito por meio da plataforma Google AI Studio, acessível em: https://aistudio.google.com/apikey?hl=pt-br.

Após aceitar os termos de uso, o desenvolvedor pode criar uma nova chave, que será utilizada na autenticação das requisições feitas à API.

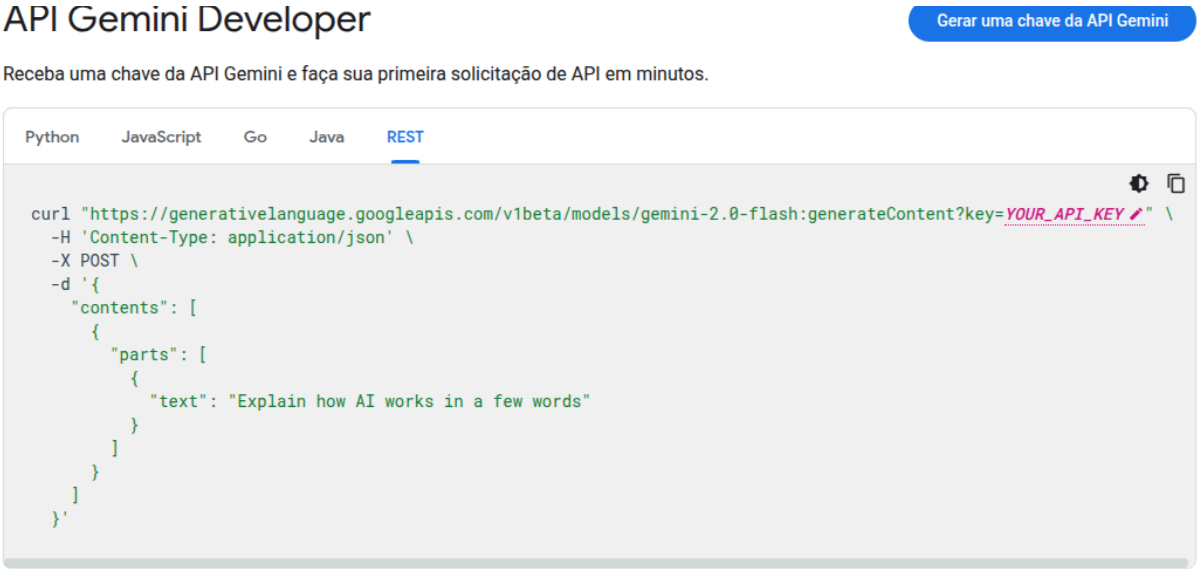
A comunicação com a API do Gemini ocorre por meio de requisições HTTP do tipo POST, enviadas a um endpoint definido no painel de integração da plataforma. As requisições incluem:

- Um **cabeçalho de autenticação**, que utiliza o padrão `Bearer Token` com a chave da API;
- A **definição do modelo**, como o `gemini-pro`, que foi o adotado neste projeto;
- Um **corpo estruturado**, que inclui o conteúdo da mensagem enviada pelo usuário.

A Figura 7.11 apresenta um exemplo de requisição feita via comando cURL, utilizado nos testes iniciais de conectividade com a API do Gemini durante o desenvolvimento do sistema.

Já a **Figura 7.12** mostra a função `executarIaGemini`, responsável por realizar a integração direta entre a aplicação e o modelo de linguagem. Essa função é utilizada para enviar o prompt construído pelo backend, receber a resposta gerada pela IA e extrair o conteúdo relevante da estrutura retornada pela API.

Figura 7.11 - Exemplo de requisição à API Gemini utilizando comando cURL



Fonte: Elaborada pelo Autor

Figura 7.12 – Função executarIaGemini responsável pela integração entre a aplicação e a API do Gemini

```
1 usage
private function executarIaGeimini($prompt)
{
    return $this->client->post( url: env( key: 'GEMINI_API_URL') . '?key=' . env( key: 'GEMINI_API_KEY'), [
        'contents' => [
            [
                'parts' => [
                    [
                        'text' => $prompt,
                    ]
                ]
            ]
        ]
    ])->json()['candidates'][0]['content']['parts'][0]['text'];
}
```

Fonte: Elaborada pelo Autor

A estrutura da resposta recebida pela aplicação segue o padrão definido pela documentação da Google. O texto gerado pelo modelo de linguagem é acessado a partir dos campos aninhados da resposta JSON e, após extraído, é utilizado pelo chatbot como resposta final ao usuário ou como parte de uma lógica de coleta de parâmetros.

Para este projeto, foi adotada a **configuração padrão** da API Gemini, sem personalizações adicionais nos parâmetros de geração da resposta, como temperatura,

número de candidatos ou formatação alternativa. Essa decisão visa manter a simplicidade e garantir previsibilidade nos testes e no comportamento da IA.

Como boa prática de segurança, a chave de API utilizada na integração é armazenada no arquivo de variáveis de ambiente `.env` da aplicação Laravel, mantendo a separação entre código e dados sensíveis.

A integração com o modelo Gemini permite que o chatbot compreenda a linguagem natural utilizada pelo usuário e gere respostas adaptadas ao contexto de cada interação, promovendo um atendimento mais inteligente, acessível e dinâmico.

7.3.3 Integração com o Cohere

Este tópico descreve o processo de integração da aplicação chatbot com a API da Cohere, uma plataforma de inteligência artificial especializada em modelos de linguagem natural (PLN), utilizada para interpretar mensagens dos usuários e gerar respostas contextualizadas.

O acesso à API é realizado por meio do site oficial da Cohere, disponível em: https://cohere.ai/, onde é possível encontrar informações detalhadas sobre os serviços oferecidos, os modelos disponíveis e a documentação técnica para desenvolvedores.

Para realizar a integração, é necessário criar uma conta na plataforma e, em seguida, acessar a seção de gerenciamento de chaves para gerar uma chave de API (API Key). Em ambientes de produção, a plataforma exige o cadastro de um método de pagamento válido e a aquisição de créditos, que são consumidos conforme o volume de requisições realizadas. No entanto, para fins de desenvolvimento e testes, a Cohere disponibiliza uma modalidade gratuita, que permite o uso limitado da API sem a necessidade de inserção de dados de pagamento.

A API da Cohere utiliza o padrão REST e permite o envio de requisições do tipo `POST` para um endpoint definido, onde a aplicação pode transmitir um prompt ou contexto textual. O modelo de IA processa o conteúdo recebido e retorna uma resposta estruturada, contendo o texto gerado com base na análise semântica da entrada. Esse

conteúdo pode então ser utilizado diretamente pelo chatbot para continuar a conversa com o usuário.

Para garantir a comunicação correta com a API, a aplicação deve incluir os seguintes elementos na requisição:

- **Cabeçalho de autenticação**, no formato `Bearer Token`, contendo a chave da API gerada no painel da Cohere;
- **Cabeçalho de tipo de conteúdo**, especificando que os dados estão no formato `application/json`;
- **Corpo da requisição**, estruturado com o modelo desejado, o prompt a ser processado e parâmetros opcionais de controle, como temperatura e número máximo de tokens.

A **Figura 7.13** apresenta a função `executarIaCohere`, responsável por realizar a comunicação entre a aplicação e a API da Cohere. Nessa função, o prompt previamente montado com base na interação do usuário é enviado ao modelo `command-r-plus`, por meio de uma requisição HTTP do tipo `POST`.

Os cabeçalhos utilizados na requisição incluem o método de autenticação do tipo *Bearer Token*, que recebe a chave de API armazenada no arquivo `.env`, e a especificação do tipo de conteúdo, definido como `application/json`.

O corpo da requisição contém os seguintes parâmetros:

- **`model`**: identifica o modelo de linguagem a ser utilizado, neste caso `command-r-plus`;
- **`message`**: representa o prompt textual enviado à IA;
- **`temperature`**: parâmetro que define o nível de criatividade da resposta, ajustado neste projeto para o valor 0.5, buscando equilíbrio entre consistência e variedade.

Após o envio, a resposta da IA é registrada em log e o campo de texto gerado é extraído da estrutura JSON retornada pela API. Esse conteúdo será utilizado diretamente no fluxo do chatbot, como resposta final ao usuário ou como orientação para os próximos passos da interação.

Figura 7.13 – Função executarIaCohere responsável pela integração com a API da Cohere e processamento da resposta

```php
private function executarIaCohere($prompt)
{
    $response = Http::withHeaders([
        'Authorization' => 'Bearer ' . env( key: "COHERE_API_KEY"),
        'Content-Type' => 'application/json'
    ])->post(env( key: "COHERE_API_URL"), [
        "model" => "command-r-plus",
        "message" => $prompt,
        "temperature" => 0.5
    ]);

    $this->logService->logInfo( txtTitle: "Resposta da IA Cohere", [$response]);

    return $response->json()['text'];
}
```

Fonte: Elaborada pelo Autor

A resposta da API é retornada em formato JSON, contendo um campo com o texto gerado pelo modelo. A aplicação extrai esse conteúdo e o utiliza como parte do fluxo conversacional com o usuário, seja como resposta final ou como etapa intermediária para coleta de dados adicionais.

Embora a API da Cohere ofereça suporte a parâmetros avançados de configuração, como controle de criatividade, penalização por repetição e limitação de tokens, este projeto optou por utilizar a **configuração padrão recomendada na documentação oficial**, visando simplicidade e consistência nos testes.

Assim como nas demais integrações de IA adotadas no sistema, a chave de autenticação é armazenada de forma segura no arquivo `.env` da aplicação Laravel, mantendo a separação entre código-fonte e informações sensíveis.

A integração com a API da Cohere complementa as capacidades do chatbot, oferecendo uma alternativa viável e eficiente para o uso de inteligência artificial na **compreensão e geração de linguagem natural**, ampliando a autonomia e a qualidade das interações com os usuários.

7.4 Estrutura do Banco de Dados

A estrutura do banco de dados foi desenvolvida com o objetivo de armazenar de forma organizada e eficiente as informações relacionadas aos serviços, tokens de autenticação e usuários que interagem com o chatbot. O banco foi implementado utilizando o PostgreSQL, com a utilização do esquema `chat` para agrupar logicamente as tabelas relacionadas ao módulo de atendimento inteligente.

7.4.1 Tabela `chat.servico`

A tabela `servico` tem como finalidade registrar os serviços disponíveis no sistema, que poderão ser utilizados pelos usuários por meio do chatbot. Cada serviço contém os seguintes campos:

- `sv_codigo`: chave primária, utilizada para identificação única do serviço;
- `sv_nome`: nome identificador do serviço;
- `sv_descricao`: descrição breve sobre a funcionalidade do serviço;
- `sv_url`: URL do endpoint que será acessado para a execução do serviço;
- `sv_metodo`: método HTTP utilizado na requisição (GET, POST, etc.);
- `sv_parametros`: campo do tipo `jsonb` que armazena a lista de parâmetros esperados pelo serviço, incluindo os obrigatórios e opcionais;
- `sv_parametros_fixos`: campo do tipo `jsonb` contendo parâmetros internos utilizados para controle e filtragem na requisição, não exigidos ao usuário;
- `sv_parametros_de_retorno`: campo `jsonb` que define quais campos do retorno da requisição devem ser considerados, permitindo o filtro e a formatação da resposta enviada ao usuário.

Essa estrutura possibilita a definição dinâmica dos serviços, permitindo que o comportamento do chatbot seja modificado ou ampliado diretamente pelo banco de dados, sem necessidade de alterações no código.

7.4.2 Tabela `chat.token`

A tabela `token` é responsável por armazenar informações utilizadas na autenticação de serviços externos que exigem tokens dinâmicos. Essa tabela permite o gerenciamento de múltiplos tokens, cada um com sua respectiva configuração de geração. Os campos da tabela são:

- `tk_codigo`: chave primária;
- `tk_header`: valor do cabeçalho HTTP onde o token será inserido, devendo ser único;
- `tk_metodo`: método utilizado para a obtenção do token (por exemplo, POST);
- `tk_url`: URL do endpoint de autenticação;
- `tk_campo`: campo do corpo da resposta onde o token será localizado.
- `tk_tempo_expiração`: tempo de duração do token

Essa estrutura permite que o sistema gera tokens automaticamente antes da execução dos serviços que necessitam autenticação.

7.4.3 Tabela `chat.contato`

A tabela `contato` armazena os dados dos usuários que interagem com o chatbot via WhatsApp. Ela é essencial para identificar o histórico de interações e garantir conformidade com a LGPD. Seus campos incluem:

- `con_codigo`: chave primária;
- `con_numero`: número de telefone do usuário, com valor único;
- `con_nome`: nome do contato, quando disponível;
- `con_data_de_cadastro`: data e hora em que o contato foi registrado no sistema;
- `con_data_de_alteracao`: data da última atualização do contato;
- `con_aceitou_lgpd`: campo booleano que indica se o usuário aceitou os termos de uso e privacidade exigidos pela LGPD;

- `con_servico`: chave estrangeira que referencia o serviço atualmente selecionado pelo usuário, permitindo o controle de sessões em andamento.

Essa tabela é utilizada tanto para validar os contatos e seu consentimento, quanto para manter o controle de qual serviço está sendo executado por cada usuário no momento da interação.

7.5 Elementos da solução e relacionamento entre eles

A implementação da solução foi estruturada em etapas sequenciais, visando garantir uma integração eficiente entre os componentes do sistema, com foco na comunicação automatizada via WhatsApp, processamento inteligente de mensagens por meio de modelos de inteligência artificial e interação com serviços externos. A seguir, são detalhados os elementos da solução e como se relacionam entre si.

O primeiro passo consistiu na definição e configuração das variáveis de ambiente no arquivo `.env`, contemplando informações essenciais como as credenciais da API do WhatsApp Business, bem como as chaves de autenticação das diferentes inteligências artificiais utilizadas (ChatGPT, Gemini e Cohere). Essas variáveis foram fundamentais para garantir a flexibilidade do sistema e facilitar a alternância entre diferentes serviços sem a necessidade de alterações diretas no código-fonte.

Figura 7.10 – Arquivo .env com variáveis de configuração da API

```
IA_ATUAL=COHERE

WHATSAPP_API_URL=https://graph.facebook.com/v21.0/{numberId}
WHATSAPP_ACCESS_TOKEN=
WHATSAPP_TOKEN_PERMANENTE=

GEMINI_API_KEY=
GEMINI_API_URL=https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent

OPENAI_API_KEY=
OPENAI_API_URL=https://api.openai.com/v1/chat/completions

COHERE_API_KEY=
COHERE_API_URL=https://api.cohere.ai/v1/chat
```

Fonte: Elaborada pelo Autor

Em seguida, foram criadas as **migrations** e os **models** responsáveis pela geração das tabelas no banco de dados PostgreSQL. Essas estruturas armazenam

informações relevantes, como contatos, interações, tokens e logs, garantindo a persistência e rastreabilidade das operações.

Figura 7.11 – Exemplo de migration para criação de tabelas no banco de dados

```php
return new class extends Migration {

    public function up(): void
    {
        Schema::create( table: 'chat.contato', function (Blueprint $table) {
            $table->bigIncrements( column: 'con_codigo')->primary();
            $table->string( column: 'con_numero',  length: 12)->unique();
            $table->string( column: 'con_nome',  length: 300);
            $table->timestamp( column: 'con_data_de_cadastro')->useCurrent();
            $table->timestamp( column: 'con_data_de_alteracao')->useCurrent()->useCurrentOnUpdate();
            $table->boolean( column: 'con_aceitou_lgpd')->nullable();
            $table->unsignedBigInteger( column: "con_servico")->nullable();

            $table->foreign( columns: 'con_servico')
                ->references( columns: 'sv_codigo')
                ->on( table: 'chat.servico');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists( table: 'chat.contato');
    }
};
```

Fonte: Elaborada pelo Autor

Na etapa seguinte, foi desenvolvido o código responsável pela integração com as inteligências artificiais. Foi implementado um **switch** que define qual modelo de IA será utilizado, com base na variável de ambiente previamente configurada. Cada IA possui um método específico de execução, que recebe um *prompt* e retorna o texto gerado pela IA correspondente.

Figura 7.12 – Código para execução do modelo de IA configurado

```
5 usages
private function executarIa($prompt)
{
    switch (env( key: 'IA_ATUAL')) {
        case 'GEMINI':
            return $this->executarIaGeimini($prompt);
        case 'OPENAI':
            return $this->executarIaOpenAi($prompt);
        case 'COHERE':
            return $this->executarIaCohere($prompt);
        default:
            $this->logService->logError("IA não configurada", ['ia' => env( key: "IA_ATUAL")]);
            return '';
    }
}
```

Fonte: Elaborada pelo Autor

Além disso, foram implementadas funções específicas que utilizam esta função de integração com a Inteligência Artificial, em diferentes momentos do fluxo de interação com o usuário. Cada uma dessas funções é responsável por gerar um *prompt*, o qual é enviado à IA para auxiliar na tomada de decisão e na construção das mensagens. A seguir, são descritas essas funções:

- **identificarIntencao**: analisa a mensagem recebida do usuário e determina sua intenção principal, permitindo que o sistema direcione corretamente a próxima ação. O *prompt* gerado inclui informações sobre os serviços disponíveis, as regras que a IA deve considerar e, por fim, a mensagem enviada pelo usuário.;
- **montarMensagemDaApiDeServico**: com o auxílio da IA, essa função gera uma mensagem explicativa solicitando ao usuário os parâmetros necessários para o consumo de um determinado serviço. O *prompt* contém as informações dos serviços e seus respectivos parâmetros, de modo que a IA possa retornar uma mensagem clara pedindo os dados ao usuário.
- **montarJsonDeParametros**: converte a mensagem do usuário em um objeto JSON estruturado, conforme a configuração exigida pela API do serviço. O *prompt* solicita à IA a geração desse JSON com base nas especificações dos parâmetros, viabilizando sua utilização na chamada ao serviço.
- **montarRetornoDoServico**: formata a resposta recebida do serviço externo em uma mensagem clara e compreensível para o usuário, utilizando apenas os campos relevantes definidos na configuração. O *prompt* apresenta a resposta

bruta do serviço e solicita que a IA a transforme em uma mensagem adequada ao usuário final.

Posteriormente, foi implementado o **método de geração de tokens**, o qual busca no banco de dados as informações necessárias para autenticação dos serviços externos. Esse método retorna um objeto HTTP contendo os tokens e *headers* prontos para utilização.

Figura 7.13 – Geração e gerenciamento de tokens de autenticação para serviços

```php
public function gerarTokens()
{
    foreach (Token::orderBy('tk_codigo')->get() as $token) {
        $this->logService->logInfo( txtTitle: 'Gerando Tokens', [$token->tk_url]);

        if ($this->verificarSeExisteTokenGerado($token)) {
            if ($token->tk_campo === '') {
                $this->client->withHeaders([$token->tk_header => 'Bearer ' . Cache::get( key: 'BearerToken')]);
            } else {
                $this->client->withHeaders([$token->tk_header => Cache::get($token->tk_campo)]);
            }
        } else {
            if ($token->tk_metodo === 'POST') {
                $this->logService->logInfo( txtTitle: 'Gerando Token POST', [$token->tk_url]);
                $response = $this->client->post($token->tk_url, [
                    'usuario' => 'DjHFclhn1a3oKIp4ZEiiD3uPDsLAJgAX',
                    'senha' => 'S1ojH+8FT9z!WNU8jZc+Lxwj_7YQRhPp'
                ]);
            } else {
                $response = $this->client->get($token->tk_url);
            }

            if ($token->tk_campo === '') {
                Cache::put( key: 'BearerToken', $response->body(), now()->addMinutes($token->tk_tempo_expiracao));
                $this->client->withHeaders([$token->tk_header => 'Bearer ' . $response->body()]);
            } else {
                Cache::put($token->tk_campo, $response->body(), now()->addMinutes($token->tk_tempo_expiracao));
                $this->client->withHeaders([$token->tk_header => $response->json()[$token->tk_campo]]);
            }
        }
    }

    return $this->client;
}
```

Fonte: Elaborada pelo Autor

Esse objeto HTTP é utilizado pela função **executaServico** como mostra na Figura 7.14, que realiza a requisição ao serviço selecionado pelo usuário e entrega a resposta recebida para ser processada e formatada pela IA.

Figura 7.14 – Função executaServico para consumo de serviços externos via HTTP

```php
1 usage
public function executar($servico, $parametros)
{
    $this->logService->logInfo( txtTitle: "Executando o serviço: ", [$servico]);

    $http = $this->montarHttp();

    if ($servico->sv_metodo == "POST") {
        $fixos = json_decode($servico->sv_parametros_fixos,  associative: true); // true retorna como array
        $dados = array_merge($parametros, $fixos);
        $response = $http->post($servico->sv_url, $dados);
        return $this->iaService->montarRetornoDoServico($servico, $response);

    } else if ($servico->sv_metodo == "PUT") {
        $fixos = json_decode($servico->sv_parametros_fixos,  associative: true); // true retorna como array
        $dados = array_merge($parametros, $fixos);
        $response = $http->put($servico->sv_url, $dados);
        return $this->iaService->montarRetornoDoServico($servico, $response);

    } else if ($servico->sv_metodo == "GET") {
        $fixos = json_decode($servico->sv_parametros_fixos,  associative: true); // true retorna como array
        $dados = array_merge($parametros, $fixos);
        $response = $http->get($servico->sv_url, $dados);
        return $this->iaService->montarRetornoDoServico($servico, $response);
    }

}
```

Fonte: Elaborada pelo Autor

Todos esses processos são centralizados no método **receiveMessage**, que atua como ponto de entrada principal do sistema. Esse método realiza inicialmente uma validação para garantir que a mensagem recebida é de fato uma mensagem textual (evitando o processamento de reações ou visualizações captadas pelo *webhook* do WhatsApp). Em seguida, busca ou cria o registro do contato no banco de dados e extrai do *request* as informações essenciais como a mensagem e o identificador do usuário.

Há, então, uma verificação relacionada à **LGPD**, na qual o sistema verifica se o usuário aceitou os termos de uso. Após essa etapa, é feita a chamada à função de identificação de intenção. Com base no resultado dessa identificação, três fluxos são possíveis:

1. **Intenção não encontrada**: envia uma mensagem informando que a intenção não foi reconhecida e, caso o usuário continue interagindo, apresenta uma lista de serviços disponíveis;
2. **Execução de serviço**: o usuário já escolheu um serviço e está fornecendo os parâmetros. O sistema utiliza a função `montarJsonDeParametros` para estruturar os dados e executa o serviço, utilizando `montarRetornoDoServico` para apresentar o resultado ao usuário;
3. **Escolha de serviço**: o sistema registra o serviço escolhido como o atual para aquele usuário e utiliza `montarMensagemDaApiDeServico` para solicitar os parâmetros necessários à execução do serviço.

Foi também implementado o arquivo whatsappService, responsável pela lógica de envio de mensagens via API do WhatsApp Business. Nele, destaca-se o método enviarMensagemTexto mostrado na Figura 7.15, que constrói o corpo da mensagem (em formato JSON) e o envia ao método enviarMensagem. Este, por sua vez, utiliza o método montarHttp, que monta a requisição HTTP com os dados de autenticação e URL da API, definidos nas variáveis de ambiente. Além disso, o sistema envia automaticamente uma notificação de mensagem visualizada ao WhatsApp, reforçando a interatividade.

Figura 7.15 – Envio de mensagens de texto via API do WhatsApp

```
5 usages
public function enviarMensagemDeTexto(
    $contato,
    $mensagem_recebida,
    $texto)
{

    $this->logService->logInfo( txtTitle: "Texto a ser enviado", $texto);
    $corpo_da_mensagem = [
        "messaging_product" => "whatsapp",
        "to" => $contato->con_numero,
        "type" => "text",
        "text" => [
            "body" => $texto
        ]
    ];

    $this->enviarMensagem($mensagem_recebida, $corpo_da_mensagem);
}

3 usages
private function enviarMensagem($mensagem_recebida, $corpo_da_mensagem)
{
    $http = $this->montarHttp($mensagem_recebida);

    $response = $http->post( url: "/messages", $corpo_da_mensagem);

    $this->logService->logInfo( txtTitle: "Resposta da API", $response->json());
}
```

Fonte: Elaborada pelo Autor

Em síntese, a implementação é composta por múltiplos elementos integrados de forma coesa. O fluxo inicia-se com o recebimento de mensagens via webhook do WhatsApp, passa pelo backend Laravel que centraliza a lógica de negócios, utiliza modelos de IA para interpretação e formatação de respostas, consulta serviços externos quando necessário e finaliza com o envio da resposta ao usuário. A arquitetura modular permite a manutenção isolada de componentes e possibilita futuras expansões, como a inclusão de novos modelos de IA ou integração com novos serviços.

8 TESTES REALIZADOS

Neste capítulo, são descritos os testes efetuados com o objetivo de simular uma conversa real de um usuário.Também são abordadas algumas falhas e correções realizadas nas instruções enviadas para a IA..

8.1 Metodologia dos Testes

Para avaliar a eficácia do chatbot, foram conduzidos testes funcionais consistindo no envio de mensagens simulando interações reais de usuários. O objetivo principal foi verificar a capacidade do sistema em:

- Identificar corretamente a intenção do usuário;
- Solicitar os parâmetros necessários para a execução dos serviços;
- Executar as requisições aos serviços externos;
- Formatar e retornar respostas coerentes e claras ao usuário.
- Identificar a intenção de mensagens em diferentes cenários de escrita.
- Verificar Comportamento ao Passar dados incorretos ou que não tem resultados

Os testes foram repetidos com três diferentes modelos de IA configurados no sistema: ChatGPT, Gemini e Cohere. Para cada modelo, foram realizados os mesmos testes e analisadas suas respostas.

8.2 Descrição dos Testes Realizados

Para os testes práticos, foi utilizado um fluxo simples e padronizado com o objetivo de verificar a coerência e a eficácia da integração entre o chatbot, os modelos de inteligência artificial (IA) e os serviços externos.

Cada teste consistiu nas seguintes etapas:

1. **Envio da mensagem de intenção**: o usuário enviou uma mensagem com o objetivo de executar algum serviço;

2. **Verificar se a IA escolheu o serviço correto e analisar a mensagem com os parâmetros:** após o envio da mensagem com a intenção é esperado uma mensagem de retorno da IA pedindo os parâmetros necessários para executar o

serviço.

3. **Recebimento e análise da resposta**: Após o envio dos parâmetros, espera-se uma mensagem formatada e clara com os dados retornados pelo serviço executado.
4. **Verificação da identificação de intenção em diferentes cenários:** Foram utilizados exemplos de solicitações do usuário contendo erros de digitação, gírias e expressões informais, com o objetivo de avaliar se a IA seria capaz de identificar corretamente a intenção em contextos mais naturais e cotidianos.
5. **Envio de parâmetros incorretos**: Foram enviados parâmetros incorretos de dados que não existem para avaliar se a IA é capaz de gerar respostas naturais de erros retornados pela API.

Este procedimento foi repetido com os três modelos de IA disponíveis (ChatGPT, Gemini e Cohere), sendo analisados os seguintes aspectos em cada execução:

- Clareza da mensagem de solicitação de dados (ex: como a IA pediu o CPF/CNPJ);
- Capacidade da IA em entender corretamente a intenção do usuário;
- Qualidade da resposta final gerada com base no retorno do serviço.

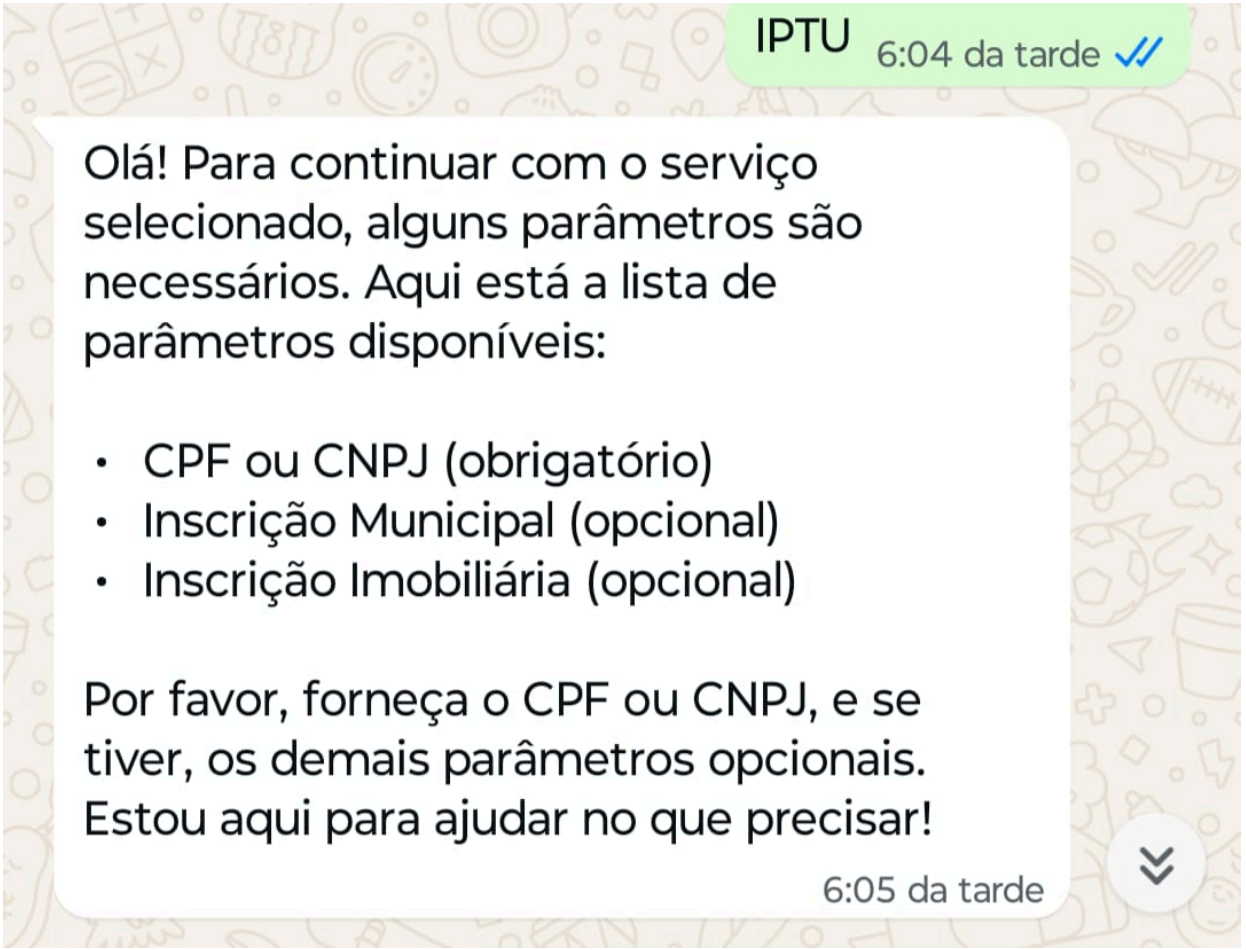
A seguir, são descritos os resultados observados para cada IA utilizada.

8.2.1 Teste com ChatGpt

O modelo da OpenAI demonstrou excelente capacidade de compreensão da intenção do usuário logo na primeira mensagem. Mesmo utilizando várias abordagens diferentes ou com erros de escrita, ele conseguiu identificar o serviço e montar a mensagem solicitando os parâmetros.

A respeito da mensagem de parâmetros, ele incluiu, em vários testes, uma mensagem dizendo que, em caso de dúvidas sobre os parâmetros, o usuário poderia perguntar. Mesmo após a alteração do prompt para evitar esse tipo de ajuda, ele continuou oferecendo esse suporte. Além disso, em alguns casos, a mensagem foi enviada sem formatação e sem quebras de linha, tornando-a difícil de ler. Na Figura 8.1 foi mostrado um exemplo do teste da resposta da intenção do serviço consultar IPTU.

Figura 8.1 - Teste de Envio de Intenção com IA ChatGPT

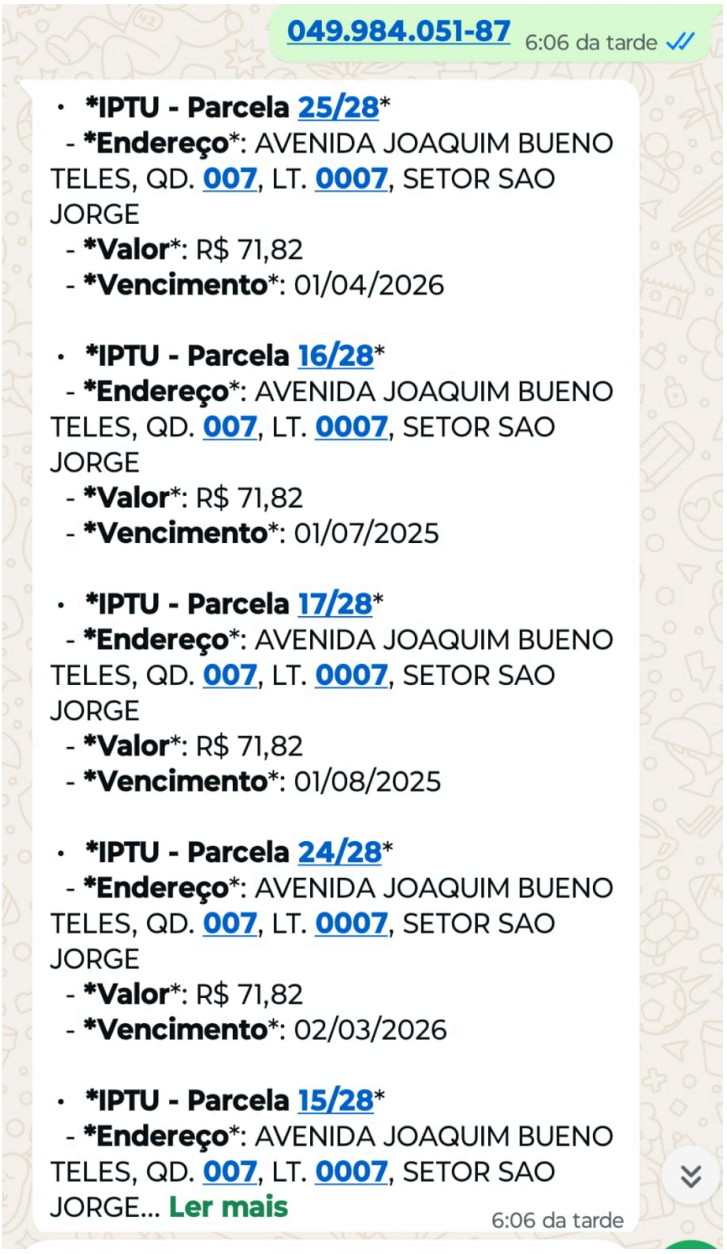


Fonte: Elaborada pelo Autor

No que diz respeito à montagem dos parâmetros e à formatação dos campos, o modelo apresentou desempenho excelente, identificando corretamente os dados fornecidos e aplicando as máscaras adequadas nos campos que exigiam formatação específica.

Na análise final da resposta do serviço, o modelo foi capaz de gerar mensagens de erro adequadas, informando corretamente quando os dados não foram encontrados ou quando algum parâmetro obrigatório estava ausente. A formatação da resposta final foi satisfatória, mantendo um padrão coerente e compreensível conforme mostrado na Figura 8.2.

Figura 8.2 - Teste de Recebimento de Resposta do Serviço da VIA Chat GPT



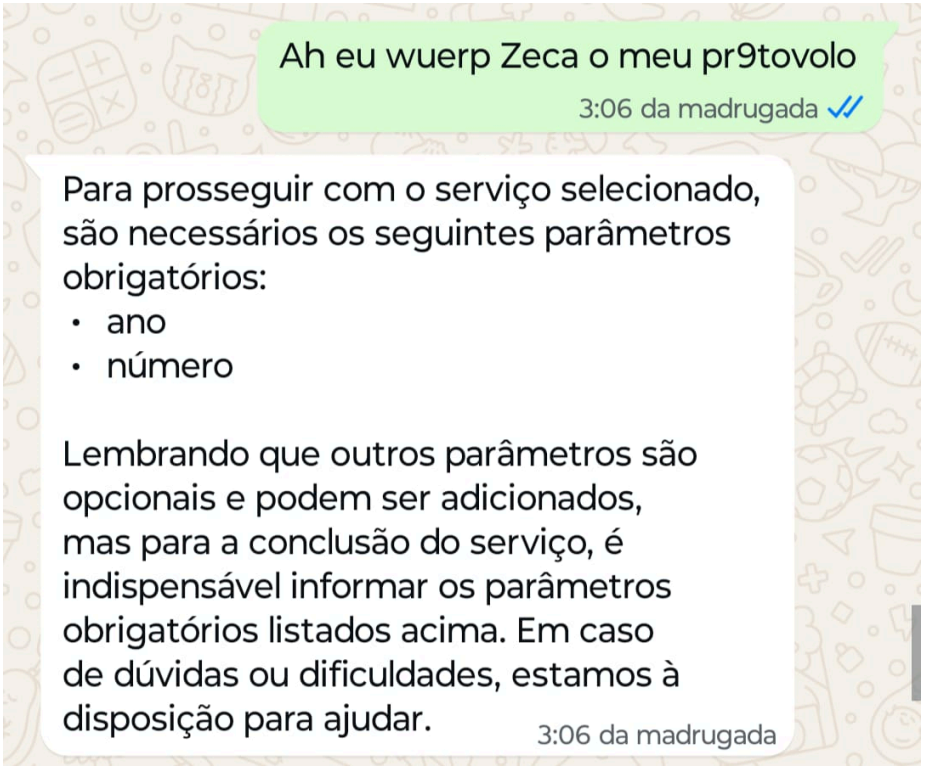
Fonte: Elaborada pelo Autor

Nos testes realizados com o modelo ChatGPT (OpenAI), foi possível observar uma maior tolerância a erros ortográficos nas mensagens enviadas pelos usuários.

Na Figura 8.3, o usuário digitou a frase: “Ah eu wuerp Zeca o meu pr9tovolo”, contendo múltiplos erros, apesar da distorção ortográfica significativa, o ChatGPT conseguiu interpretar corretamente a intenção como sendo o serviço de Consulta de Protocolo, retornando imediatamente a solicitação dos parâmetros obrigatórios para execução da tarefa, sem necessidade de retrabalho ou correção por parte do usuário.

Esse comportamento demonstra que o modelo ChatGPT possui maior robustez semântica em comparação ao Gemini, conseguindo inferir a intenção mesmo com erros mais graves na entrada textual.

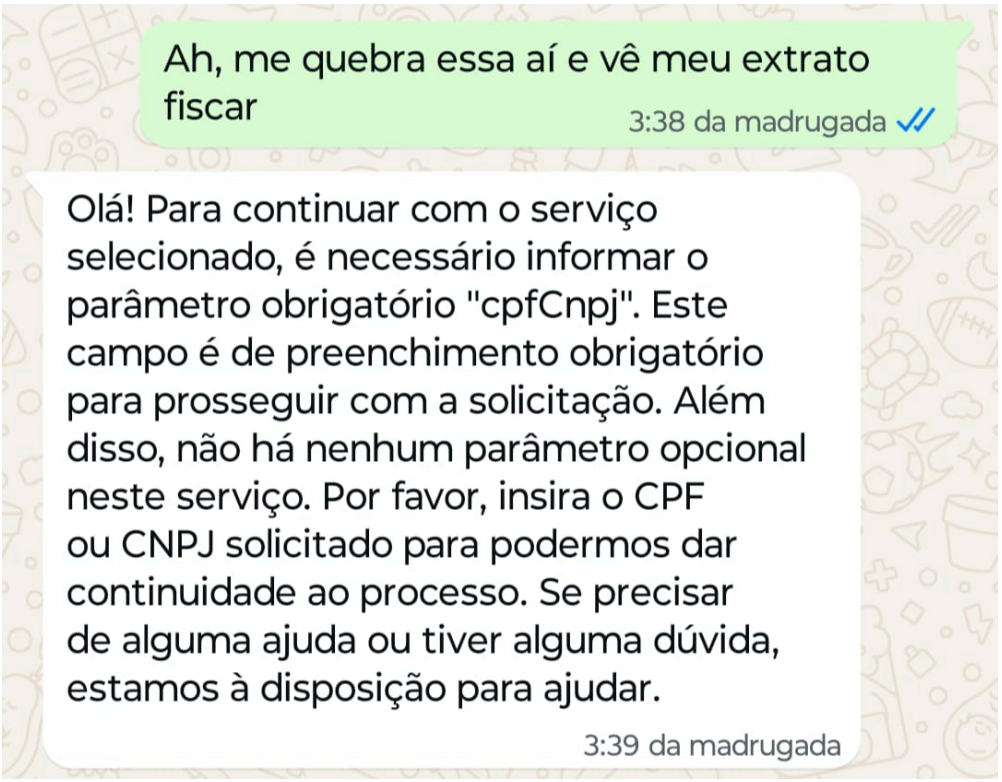
Figura 8.3 – Interação com o ChatGPT em mensagem com alto grau de erro ortográfico.



Fonte: Elaborada pelo Autor

Neste teste, foram realizadas diversas interações utilizando gírias e expressões informais, com o objetivo de verificar se o modelo da OpenAI (ChatGPT) seria capaz de compreender corretamente a intenção do usuário, mesmo quando a linguagem utilizada não segue padrões formais. A Figura 8.4 apresenta um exemplo representativo deste teste, no qual o modelo identifica com sucesso a solicitação de extrato fiscal, apesar do uso de linguagem coloquial e de um erro de digitação.

Figura 8.4 – Interação com o Gemini em mensagem com uso de gírias.



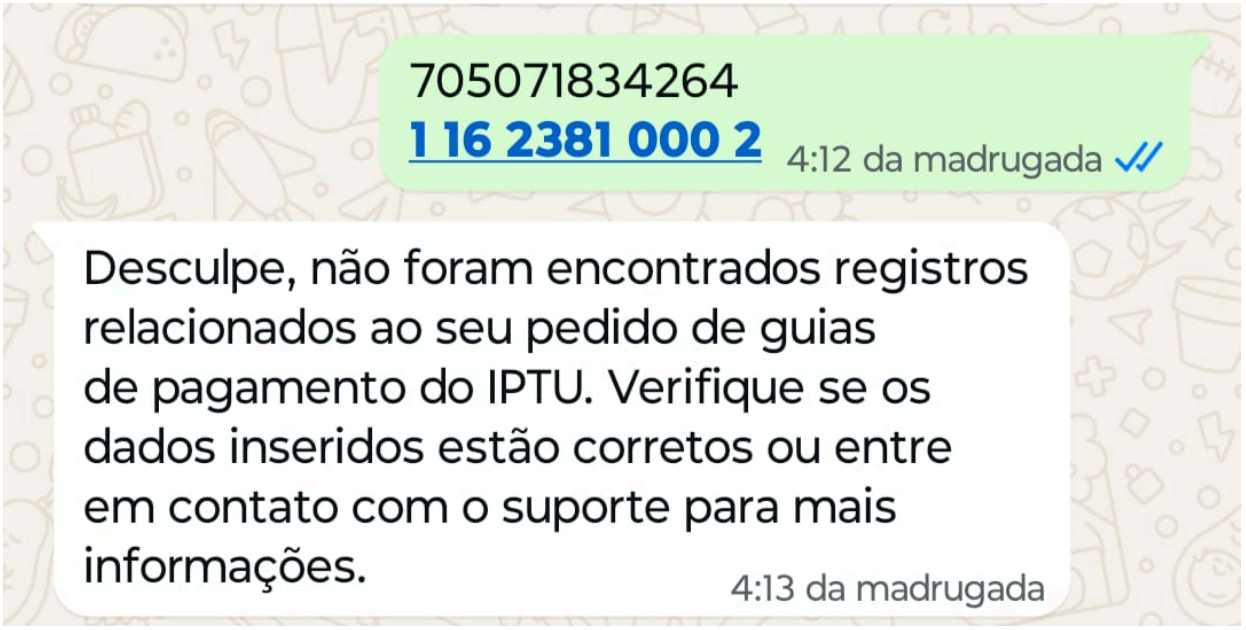
Fonte: Elaborada pelo Autor

Neste teste, o objetivo foi avaliar o comportamento do modelo ChatGPT ao receber parâmetros com dados incorretos. Mesmo com valores inválidos, o modelo foi capaz de identificar corretamente cada parâmetro, montar o JSON de requisição e executar o serviço.

Como resposta da API, foi retornado que nenhum registro foi encontrado. Diante disso, o chatbot gerou uma mensagem adequada e compreensível para o usuário final, informando a ausência de resultados de forma clara.

A Figura 8.5 apresenta um exemplo dessa interação, demonstrando a capacidade do modelo de lidar com entradas inválidas sem comprometer a experiência do usuário.

Figura 8.5 – Exemplo de resposta adequada do ChatGPT diante de parâmetros inválidos



Fonte: Elaborada pelo Autor

8.2.2 Teste Com Gemini

Os testes realizados com o Gemini demonstraram excelente capacidade de compreensão da intenção do usuário. Em alguns casos, foi necessário ajustar o prompt, adicionando mais requisitos para garantir a escolha correta do serviço.

Na montagem dos parâmetros, o Gemini utilizou campos de formatação, orientando o usuário a fornecer os dados no formato especificado no JSON. No entanto, essa informação era de uso interno e apenas a obrigatoriedade dos parâmetros deveria ser considerada. Por esse motivo, foi necessário adicionar um trecho no código para remover essas instruções de formatação do prompt. Na Figura 8.6 mostra a alteração que foi preciso fazer no código que será executada somente para API do Gemini.

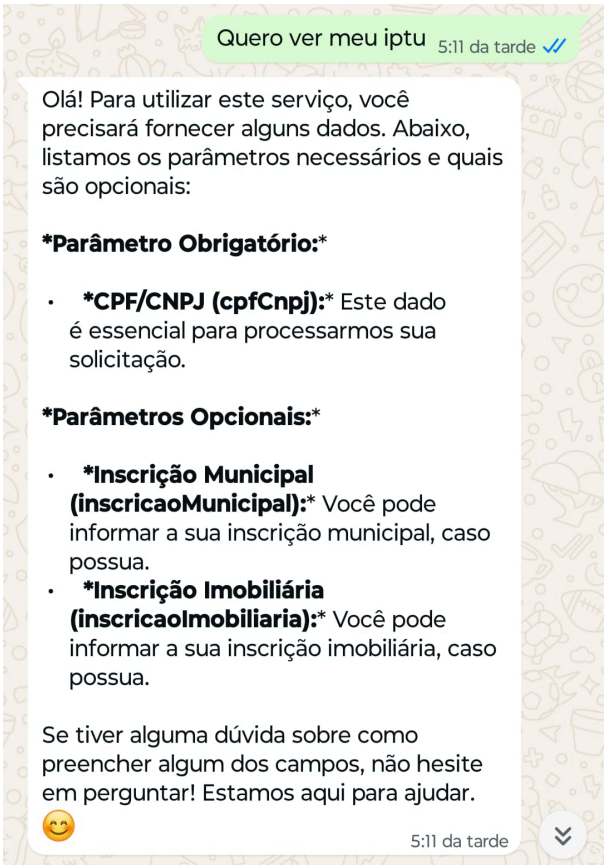
Figura 8.6 - Código Com Ajuste Para Montagem Da Mensagem da API de Serviço no Gemini

```php
public function montarMensagemDaApiDeServico($servico)
{
    // Decodifica os parâmetros JSON em um array associativo
    $parametros = json_decode($servico->sv_parametros, associative: true);

    if (env( key: 'IA_ATUAL') === 'GEMINI') {
        // Remove o campo 'formatar' de cada parâmetro
        foreach ($parametros as &$param) {
            if (isset($param['formatar'])) {
                unset($param['formatar']);
            }
        }
    }

    // Codifica novamente o JSON já sem os campos indesejados
    $parametrosLimpos = json_encode($parametros, flags: JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE);

    // Cria o prompt com instruções para a IA
    $mensagemString = "O usuário selecionou um serviço que requer parâmetros. Crie uma mensagem amigável explicando quai
    $mensagemString .= "Aqui estão os parâmetros:\n";
    $mensagemString .= $parametrosLimpos . "\n\n";

    $this->logService->logInfo( txtTitle: "montarMensagemDaApiDeServico", [$mensagemString]);

    // Executa a IA com o prompt preparado
    return $this->executarIa($mensagemString);
}
```

Fonte: Elaborada pelo Autor

As mensagens foram geradas com ótima formatação, e a utilização de emojis evidenciou uma tentativa de tornar a conversa mais humanizada. Na Figura 8.7 mostra o exemplo da mensagem recebida com a intenção do serviço de consultar IPTU.

Figura 8.7 - Teste de Envio de Intenção com IA Gemini



Fonte: Elaborada pelo Autor

No que diz respeito à montagem dos parâmetros e à formatação dos campos, o Gemini apresentou um nível semelhante ao do modelo da OpenAI, formatando corretamente todos os campos necessários.

Na análise da resposta do serviço, o desempenho foi superior ao das demais IAs. Como mostrado na Figura 8.8 além de formatar adequadamente o texto, o Gemini organizou os dados em agrupamentos para facilitar a leitura e incluiu um breve resumo com o total de resultados obtidos.

Figura 8.8 - Teste de Recebimento de Resposta do Serviço da VIA Gemini

Fonte: Elaborada pelo Autor

Durante os testes com entradas mal formuladas, foi verificado que o modelo de IA consegue interpretar a intenção do usuário mesmo diante de pequenos erros

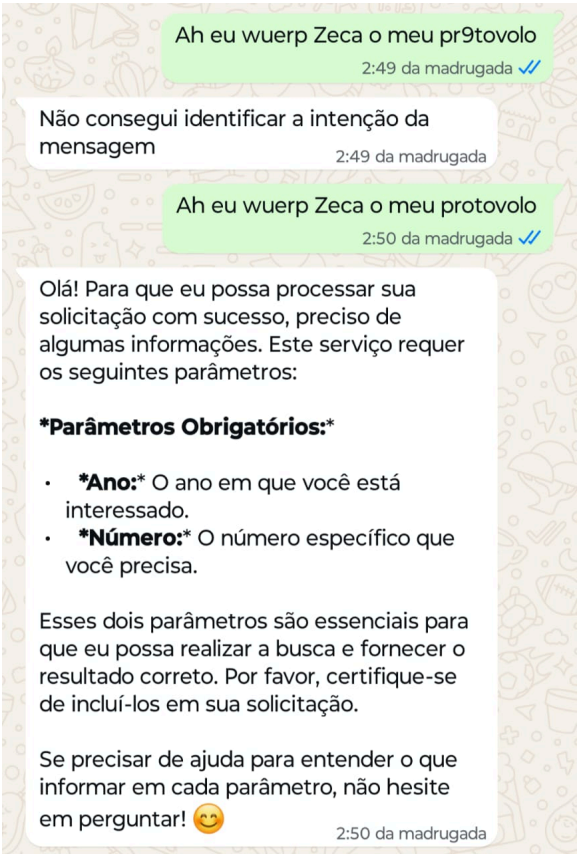
ortográficos. No entanto, quando o erro afeta partes críticas da palavra-chave, a compreensão pode falhar.

Na Figura 8.9, observa-se um caso em que o usuário tenta acessar o serviço de *Consulta de Protocolo*, utilizando inicialmente a palavra digitada incorretamente como “Ah eu wuerp Zeca o meu pr9tovolo”. O chatbot, nesse momento, retornou uma mensagem informando que não conseguiu identificar a intenção.

Na tentativa seguinte, após a correção parcial para “Ah eu wuerp Zeca o meu protovolo”, a IA foi capaz de reconhecer a intenção desejada e respondeu corretamente solicitando os parâmetros obrigatórios (Ano e Número) para processar a requisição.

Esse teste demonstra que, embora haja certa tolerância a erros de digitação, existe um limite semântico a partir do qual a IA deixa de compreender a intenção do usuário.

Figura 8.9 – Interação com o Gemini em mensagem com alto grau de erro ortográfico.

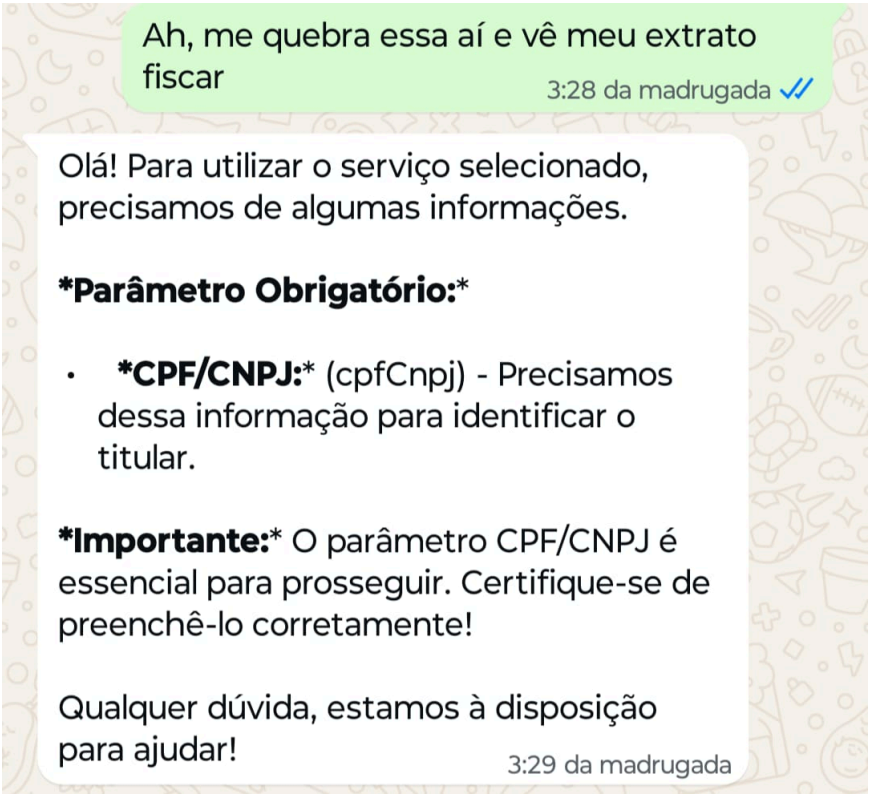


Fonte: Elaborada pelo Autor

No teste com o uso de gírias, o modelo Gemini demonstrou boa capacidade de interpretação, conseguindo identificar corretamente a intenção do usuário mesmo quando a solicitação foi formulada com expressões informais e coloquiais. A Figura 8.10 apresenta um exemplo dessa interação, na qual o usuário utiliza diversas gírias e até um erro de digitação, mas ainda assim a IA compreende que se trata de um pedido de emissão de extrato fiscal.

Foram realizados outros testes semelhantes, utilizando diferentes variações de linguagem informal, e em todos os casos o modelo foi capaz de interpretar a intenção corretamente. Isso evidencia a robustez do Gemini na compreensão de linguagem natural em contextos menos formais, como é comum em aplicativos de mensagens.

Figura 8.10 – Interação com o Gemini em mensagem com uso de gírias.



Fonte: Elaborada pelo Autor

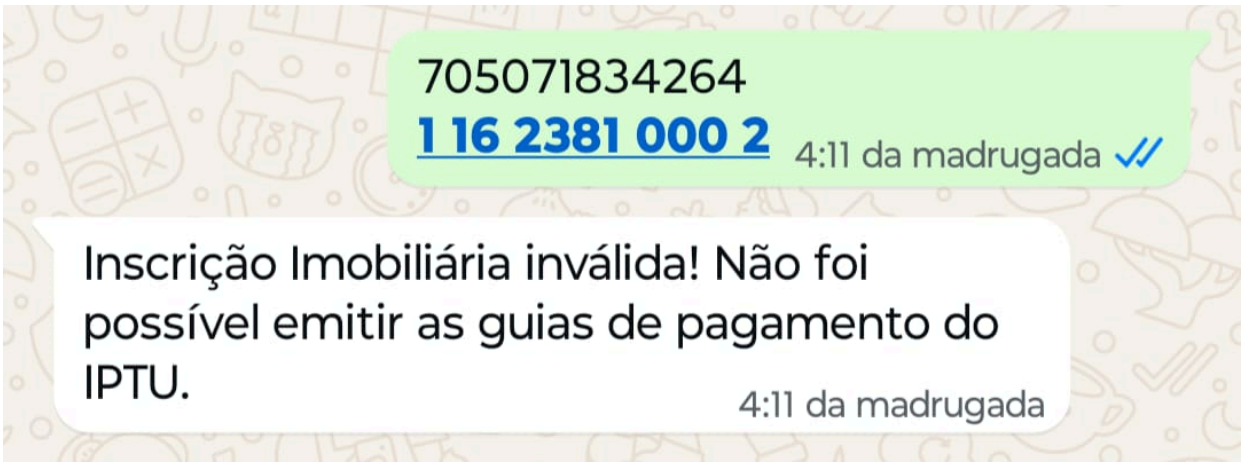
No teste com o modelo Gemini, foram enviados parâmetros com dados incorretos com o objetivo de avaliar a capacidade da IA de lidar com respostas negativas ou falhas

provenientes da API. Nesse caso, mesmo com dados inválidos, o modelo foi capaz de interpretar corretamente a resposta do serviço.

A IA tratou esse retorno de forma apropriada, gerando uma mensagem clara para o usuário final, sugerindo que os dados informados poderiam estar incorretos ou desatualizados. Esse comportamento demonstra que, embora o modelo possa não validar os dados diretamente, ele é capaz de compreender respostas da API e repassá-las ao usuário de forma compreensível.

A Figura 8.11 ilustra essa interação, mostrando a resposta recebida do serviço e a forma como o Gemini comunicou o problema ao usuário.

Figura 8.11 – Exemplo de resposta adequada do Gemini diante de parâmetros inválidos



Fonte: Elaborada pelo Autor

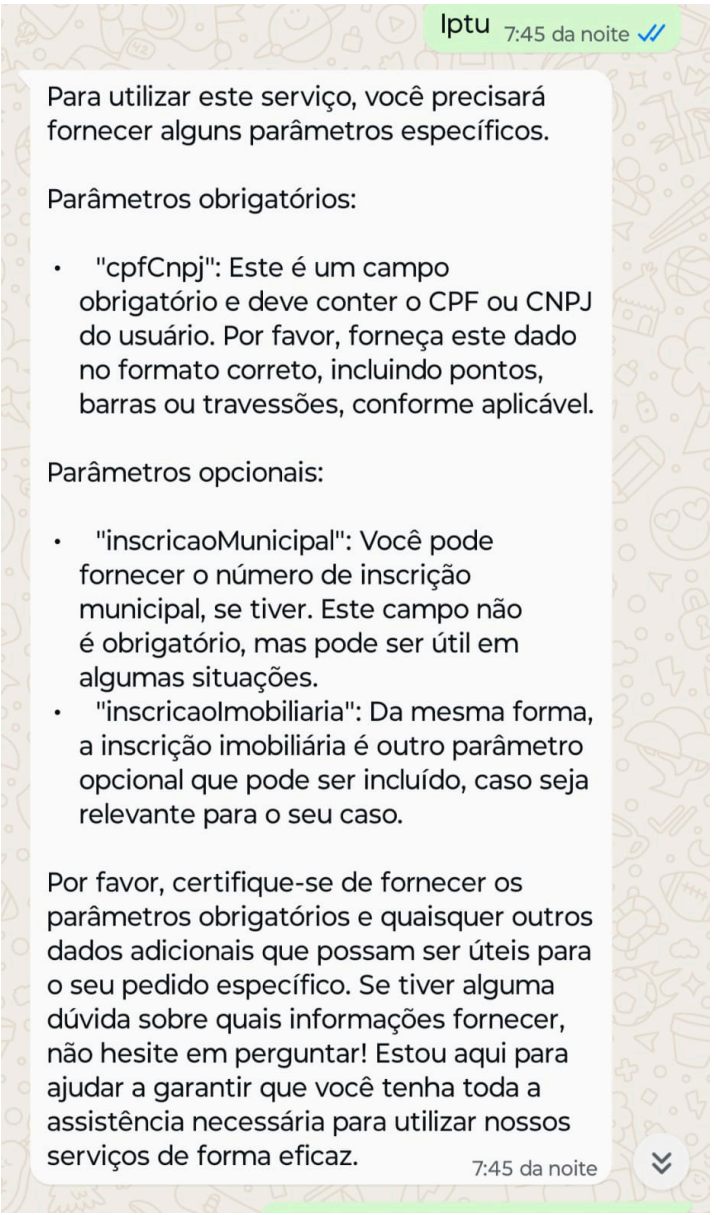
8.2.3 Teste Com Cohere

Os testes realizados com o Cohere demonstraram um nível de compreensão da intenção do usuário equivalente ao das demais IAs.

Na montagem dos parâmetros, o Cohere gerou mensagens com excesso de

informações, deixando de ser breve e objetivo como as outras IAs. Embora bem escritas e formatadas, essas mensagens tornaram-se difíceis de ler devido à sua complexidade. Na Figura 8.12 mostra um exemplo da resposta à intenção do serviço de consultar o IPTU.

Figura 8.12 - Teste de Envio de Intenção com IA Cohere



Fonte: Elaborada pelo Autor

No que se refere à montagem dos parâmetros e à formatação dos campos, o modelo Cohere apresentou desempenho equivalente ao dos demais modelos de inteligência artificial avaliados.

Na etapa de análise da resposta do serviço, seu desempenho foi semelhante ao do Gemini conforme a Figura 8.13, realizando agrupamentos dos resultados e apresentando um breve resumo do conteúdo retornado. No entanto, a formatação final da resposta ficou aquém da obtida com o modelo Gemini.

Figura 8.13 - Teste de Recebimento de Resposta do Serviço da VIA Cohere

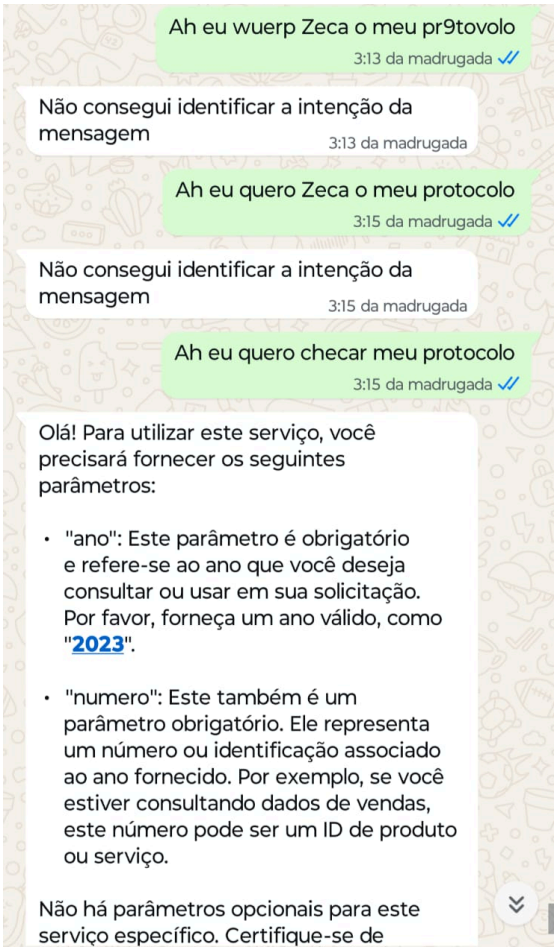
Fonte: Elaborada pelo Autor

Durante os testes com a IA da Cohere, foi observado que o modelo apresentou maior dificuldade em compreender mensagens com erros ortográficos, mesmo quando os erros foram mínimos. Na Figura 8.14, é mostrado um exemplo em que o usuário envia a frase: “Ah eu wuerp Zeca o meu pr9tovolo”, contendo diversos erros de digitação.

A IA não conseguiu identificar a intenção da mensagem. Em seguida, mesmo após correções significativas, como: “Ah eu wuerp Zeca o meu protocolo”, a resposta ainda foi negativa. Apenas quando a mensagem foi reformulada como “Ah eu quero checar meu protocolo” ou seja, com palavras mais próximas de um padrão formal, o modelo foi capaz de reconhecer corretamente o serviço e prosseguir com a solicitação dos parâmetros obrigatórios.

Outros testes também foram realizados para validar esse comportamento. Em uma das situações, a mensagem “Ah eu gostaria de checar meu protovolo” (com erro apenas na palavra "protocolo") foi corretamente interpretada. Isso indica que a IA da Cohere possui baixa tolerância a erros acumulados, mas consegue compreender a intenção do usuário quando a estrutura da frase está próxima da linguagem padrão e o erro está isolado.

Figura 8.14 – Interação com o Cohere em mensagem com alto grau de erro ortográfico.

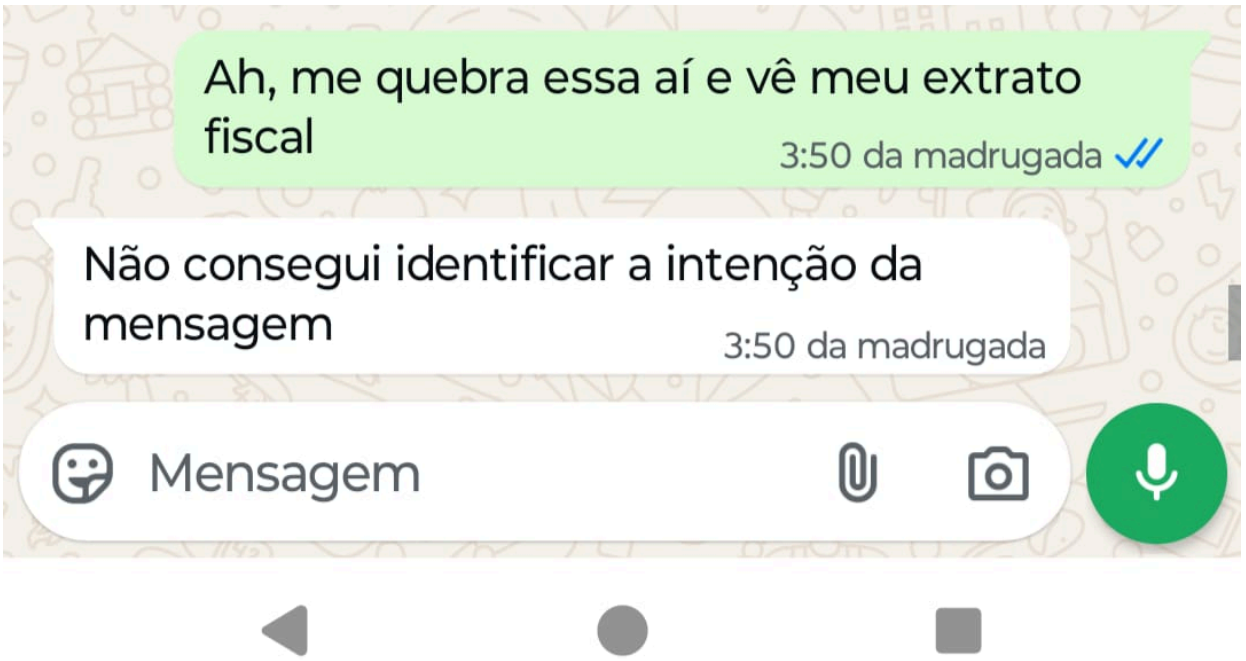


Fonte: Elaborada pelo Autor

No teste com o modelo da Cohere, foi observada uma limitação significativa na interpretação de mensagens com gírias ou linguagem informal. Apesar de a intenção da mensagem ser clara para um falante humano solicitação de extrato fiscal, o modelo não foi capaz de compreendê-la corretamente.

Embora em alguns casos o modelo consiga interpretar solicitações informais, o desempenho foi inferior em comparação com os modelos da OpenAI (ChatGPT) e do Gemini, que demonstraram maior tolerância a desvios linguísticos e gírias. A Figura 8.15 ilustra esse comportamento.

Figura 8.15 – Interação com o Cohere em mensagem com uso de gírias.



Fonte: Elaborada pelo Autor

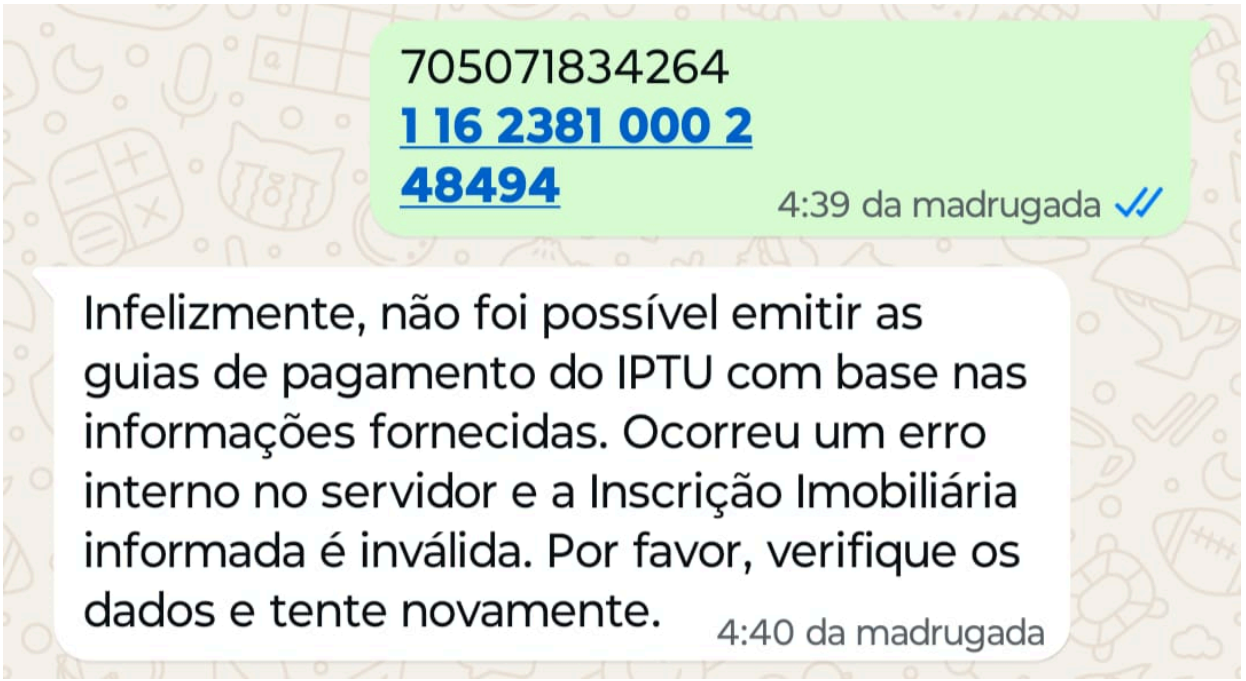
No teste com o modelo Cohere, foram fornecidos parâmetros incorretos com o objetivo de avaliar a capacidade da IA em lidar com respostas de erro provenientes da API. Ao executar o serviço com os dados inválidos.

Apesar de não validar diretamente os dados enviados, o modelo da Cohere foi capaz de interpretar corretamente a resposta do serviço e comunicar de forma adequada ao usuário que os dados informados provavelmente estavam incorretos ou não correspondiam a nenhum registro disponível.

Esse comportamento demonstra que, embora o modelo possa ter limitações na formatação técnica da requisição, ele é capaz de tratar retornos negativos da API e fornecer uma resposta útil e compreensível ao usuário.

A Figura 8.16 apresenta um exemplo dessa situação, evidenciando como o modelo lidou com a ausência de resultados no serviço consultado.

Figura 8.16 – Exemplo de resposta adequada do Cohere diante de parâmetros inválidos



Fonte: Elaborada pelo Autor

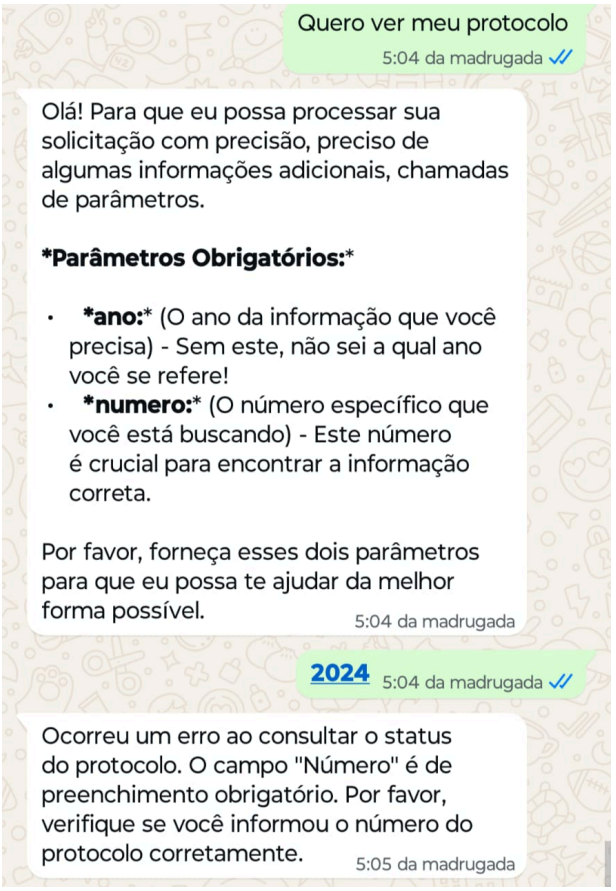
8.3 Teste Faltando Parâmetros Obrigatórios

Durante os testes realizados, foi observado um comportamento importante: quando o usuário informava apenas parte dos parâmetros obrigatórios deixando de preencher um ou mais campos obrigatórios a IA ainda assim montava o JSON e enviava a requisição para a API, mesmo com a ausência de informações essenciais.

Embora as APIs utilizadas nos testes possuam validações internas e retornem mensagens de erro específicas quando um parâmetro obrigatório não é informado, como demonstrado na Figura 8.17, esse comportamento pode não se repetir em outras APIs.

Algumas interfaces, especialmente menos robustas, não tratam corretamente a ausência de parâmetros obrigatórios, podendo retornar erros genéricos ou até falhas técnicas graves, como `NullPointerException`.

Figura 8.17 – Retorno da API informando erro devido à ausência de parâmetro obrigatório



Fonte: Elaborada pelo Autor

Para contornar o problema da ausência de parâmetros obrigatórios, foi realizada uma modificação tanto no prompt enviado à IA quanto na lógica de tratamento da resposta no código do chatbot.

No prompt responsável por solicitar à IA a montagem do JSON com os parâmetros do serviço, foi adicionada a seguinte instrução: "Se o usuário não forneceu algum parâmetro obrigatório, retorne no JSON o campo `erro` com uma mensagem informando para enviar também os parâmetros obrigatórios que faltam."

Com essa instrução, a IA passou a realizar uma verificação prévia antes de montar a requisição, validando se todos os campos essenciais foram informados pelo usuário. Caso identifique a ausência de algum parâmetro obrigatório, o modelo não monta o JSON de forma convencional, mas retorna um campo `erro` com uma mensagem explicativa.

Para tratar esse novo comportamento, o código do chatbot também foi ajustado. Foi implementada uma verificação que identifica a presença do campo `erro` no JSON retornado pela IA. Quando esse campo é detectado, o chatbot:

- **Interrompe a execução da requisição para a API**;
- **Exibe ao usuário uma mensagem solicitando que informe todos os dados obrigatórios antes de prosseguir**.

Esse ajuste evita o envio de requisições inválidas, reduz falhas na execução dos serviços e melhora a experiência do usuário, tornando o fluxo mais robusto e confiável.

A Figura 8.18 apresenta um trecho do código onde essa lógica de tratamento foi implementada.

Figura 8.18 – Trecho de código responsável por interromper a requisição ao detectar ausência de parâmetros obrigatórios

```php
$parametros = $this->iaService->montarJsonDeParametros($contato, $mensagem);
if (isset($parametros['erro'])) {
    $this->whatsAppService->enviarMensagemDeTexto($contato, $mensagem, $parametros['erro']);
} else {
    $retornoDaExecucao = $this->executarServicoService->executar($contato->servico, $parametros);
    $this->whatsAppService->enviarMensagemDeTexto($contato, $mensagem, $retornoDaExecucao);
}
```

Fonte: Elaborada pelo Autor

Após a implementação dessa solução, foram realizados testes com as três IAs utilizadas no projeto: ChatGPT, Gemini e Cohere. O objetivo foi verificar se a nova instrução adicionada ao prompt era compreendida corretamente por cada modelo.

O resultado foi satisfatório. Todas as IAs foram capazes de identificar a ausência de parâmetros obrigatórios e retornar mensagens informativas, solicitando ao usuário que completasse os dados faltantes antes da execução do serviço.

Um exemplo de resposta gerada foi: *"Por favor, informe também o parâmetro 'número'."*

8.4 Testes De Segurança

Com o crescimento do uso de chatbots em ambientes comerciais, torna-se fundamental avaliar não apenas sua funcionalidade e desempenho, mas também sua segurança, especialmente considerando os riscos de vazamento de dados, manipulações maliciosas e uso indevido da aplicação. A seguir, são descritos os testes de segurança realizados no chatbot desenvolvido, com foco em cenários comuns de ataque ou uso indevido.

É importante destacar que, independentemente da entrada fornecida pelo usuário, o chatbot utiliza inteligência artificial para tentar identificar a intenção por trás da mensagem. Quando essa identificação não é possível, a IA retorna o caractere **"N"**, o que indica que não houve correspondência com nenhum serviço disponível. Nesses casos, o sistema automaticamente informa ao usuário que a solicitação não foi compreendida e, após algumas tentativas mal sucedidas, envia a lista completa dos serviços oferecidos, orientando o usuário de forma segura e clara. Essa abordagem reduz significativamente o risco de ações indevidas, pois qualquer entrada que não corresponda a um serviço legítimo é tratada como neutra ou inválida, sem impacto funcional ou risco à integridade do sistema.

Este teste teve como objetivo verificar a estabilidade do chatbot diante de mensagens enviadas em grande volume e de forma repetitiva, simulando um cenário de **flood**, prática comum em ataques de negação de serviço ou tentativas de sobrecarga do sistema.

Procedimento: Foram enviadas consecutivamente 20 mensagens com o conteúdo "oi", em curto intervalo de tempo.

Resultado esperado: O sistema deveria detectar o comportamento repetitivo e limitar as respostas, evitando sobrecarga do servidor e envio de mensagens desnecessárias.

Resultado obtido: O chatbot respondeu individualmente a todas as mensagens recebidas, sem aplicar nenhuma forma de contenção ou bloqueio. Isso indica que o sistema não possui, até o momento, um mecanismo eficaz de detecção de flood ou limitação de interações consecutivas do mesmo usuário. Como consequência, o teste foi considerado não satisfatório, apontando uma possível vulnerabilidade que pode comprometer a escalabilidade e a estabilidade do serviço em ambientes com grande volume de interações.

8.5 Conclusão dos Testes

A realização dos testes permitiu avaliar com precisão a eficácia da integração entre o chatbot, os modelos de inteligência artificial (ChatGPT, Gemini e Cohere) e os serviços externos. Os resultados obtidos demonstraram que todos os modelos foram capazes de identificar corretamente a intenção do usuário, solicitar os parâmetros necessários e formatar adequadamente as respostas, com variações sutis de desempenho entre eles.

O modelo ChatGPT destacou-se pela alta taxa de acerto na identificação da intenção, mesmo diante de variações na escrita ou de mensagens ambíguas. No entanto, apresenta limitações na formatação da mensagem de solicitação de parâmetros, especialmente em relação à legibilidade.

O Gemini demonstrou desempenho consistente e, em alguns aspectos, superior, como na organização das respostas com agrupamentos e resumos que facilitaram a interpretação dos resultados. Entretanto, exigiu ajustes adicionais nos prompts para remover informações desnecessárias geradas na etapa de solicitação de parâmetros.

Já o Cohere mostrou desempenho equilibrado, acompanhando de perto os outros modelos em termos de compreensão de intenção e formatação de respostas, mas com menor tolerância a variações linguísticas.

Apesar das diferenças entre os modelos, foi possível adaptar os prompts e ajustes no código para alcançar um comportamento consistente entre eles, garantindo uma boa experiência para o usuário em todos os testes realizados. Além disso, os testes de segurança evidenciaram pontos de melhoria importantes, como a necessidade de um controle de fluxo para evitar sobrecarga por interações repetitivas. De forma geral, os resultados obtidos reforçam a viabilidade da integração entre o chatbot e múltiplos modelos de IA, desde que acompanhada de uma lógica robusta de tratamento e controle no backend.

A fim de facilitar a visualização e comparação do desempenho entre os modelos de inteligência artificial utilizados no projeto, foi elaborada a Tabela 1, com notas atribuídas a partir dos critérios definidos na metodologia de testes.

Tabela 1 - Avaliação comparativa dos modelos de IA utilizados no chatbot - Brasil, 2015.

Critério Avaliado	ChatGPT	Gemini	Cohere
Identificação da Intenção	100%	95%	74%
Solicitação de Parâmetros	90%	84%	75%
Formatação da Resposta Final	80%	97%	87%
Respostas a Parâmetros Inválidos	89%	86%	89%
Compreensão de Linguagem Informal/Gírias	100%	83%	53%
Tolerância a Erros Ortográficos	100%	74%	42%

Fonte: Autoria Própria

9 CONSIDERAÇÕES FINAIS

O avanço das tecnologias de inteligência artificial (IA) e sua integração com plataformas de mensagens, como o WhatsApp, possibilitam a automação de tarefas repetitivas e operacionais por meio de chatbots inteligentes. Este trabalho teve como objetivo principal o desenvolvimento de um chatbot voltado exclusivamente à execução de serviços automatizados, utilizando diferentes modelos de IA generativa para interpretar mensagens, solicitar parâmetros e formatar respostas a partir de serviços externos.

Ao longo do desenvolvimento, foi possível integrar com sucesso modelos como ChatGPT, Gemini e Cohere, explorando suas capacidades em etapas distintas do fluxo de execução: desde a identificação da intenção do usuário até a formatação da resposta final com os dados do serviço. A utilização de diferentes modelos de IA também permitiu realizar uma análise comparativa de desempenho em termos de compreensão contextual, geração de prompts e clareza das mensagens geradas.

9.1 Discussão

O projeto demonstrou que é viável a criação de um chatbot que, sem realizar atendimento convencional, consegue interpretar comandos, solicitar os dados necessários, executar serviços externos e retornar informações de forma clara e funcional. A separação das funções em etapas específicas — como identificação da intenção, montagem dos parâmetros, conversão para JSON e formatação do retorno — permitiu modularidade e facilitou os testes com diferentes modelos de IA.

Os resultados apontaram diferenças significativas entre os modelos. O ChatGPT apresentou excelente desempenho na interpretação da intenção, ainda que, em alguns casos, tenha mantido um comportamento assistivo mesmo após alterações no prompt. O Gemini se destacou na formatação das respostas e na organização das informações de retorno, enquanto o Cohere demonstrou boa compreensão, mas excesso de detalhes em suas mensagens, o que comprometeu a objetividade.

Esse tipo de aplicação mostra-se promissor para empresas que desejam oferecer serviços automatizados em plataformas de mensageria, sem a necessidade de operadores humanos, permitindo agilidade, precisão e escalabilidade no atendimento a demandas operacionais.

9.2 Limitações

Embora o desenvolvimento do chatbot tenha apresentado resultados satisfatórios, algumas limitações devem ser reconhecidas:

- Os fluxos de serviço testados ainda são limitados em número e complexidade, o que restringe a avaliação do sistema em cenários com múltiplos níveis de dependência entre parâmetros ou com lógica condicional mais elaborada;
- A integração com modelos de IA foi bem-sucedida, porém a aplicação ainda depende da estabilidade e disponibilidade das APIs externas, o que pode afetar a continuidade do serviço em situações de instabilidade;
- Apesar dos testes terem incluído variações linguísticas nas mensagens de entrada, todas as interações ocorreram em um ambiente controlado, com fluxo simulado. Isso pode não contemplar totalmente as variações comportamentais e contextuais de usuários reais em um ambiente produtivo;
- Embora os prompts utilizados tenham apresentado excelente desempenho até o momento, não foram realizadas avaliações em larga escala ou com usuários externos que pudessem gerar entradas inesperadas ou maliciosas;
- Em situações em que a intenção do usuário não é reconhecida corretamente, o sistema retorna uma lista de parâmetros válidos, permitindo a continuidade da interação. No entanto, esse mecanismo ainda pode ser aprimorado com estratégias mais proativas de reinterpretação da mensagem ou sugestões mais contextuais.

9.3 Conclusão

O trabalho atingiu com êxito seus objetivos, demonstrando a viabilidade técnica de um chatbot baseado em IA voltado exclusivamente à execução de serviços por meio do WhatsApp. A segmentação do fluxo em funções específicas e o uso de diferentes modelos de linguagem contribuíram para a criação de uma solução adaptável, robusta e capaz de oferecer respostas eficientes com base em entradas livres do usuário.

A análise comparativa entre os modelos de IA também revelou pontos fortes e fracos de cada abordagem, fornecendo subsídios para escolhas mais adequadas conforme o contexto de uso. Assim, conclui-se que a aplicação de IA generativa em chatbots operacionais representa um avanço significativo para a automação de

processos em mensageria, desde que bem orientada por prompts e integrada a fluxos consistentes.

9.4 Sugestão de Trabalhos Futuros

Com base nos resultados obtidos e nas possibilidades de expansão do sistema, algumas direções promissoras para trabalhos futuros incluem:

- **Implementação de entrada e saída por áudio:** Desenvolver a capacidade do chatbot de receber comandos por áudio e responder também em formato de áudio. Isso permitirá ampliar a acessibilidade e melhorar a experiência do usuário, especialmente em situações em que a digitação não é conveniente;
- **Armazenamento de histórico de conversas:** Integrar um sistema de histórico de mensagens para que a IA possa manter o contexto das interações e responder de forma mais precisa e personalizada, promovendo conversas mais fluídas e inteligentes;
- **Suporte a serviços dependentes:** Adicionar a capacidade de execução de serviços que dependem de resultados anteriores ou que envolvem múltiplas etapas interdependentes, como autenticação prévia ou a execução de tarefas sequenciais;
- **Expansão para múltiplos canais de comunicação:** Adaptar o sistema para funcionar em outras plataformas além do WhatsApp, como Telegram, Messenger ou Webchat, mantendo a lógica de serviço automatizado;

REFERÊNCIAS

ALI, Moez. **A beginner's Guide to Using the ChatGPT API.** DataCamp, 14 dez. 2023. Disponível em: https://www.datacamp.com/tutorial/a-beginners-guide-to-chatgpt-api. Acesso em 31 de maio 2024.

ANDRION Roseli. **Chatbots e computação cognitiva: cada vez mais populares e sofisticados.** Olhar Digital, 11 out. 2019. Disponível em: https://olhardigital.com.br/2019/10/11/noticias/chatbots-e-computacao-cognitiva-cada-vez-mais-populares-e-sofisticados/. Acesso em 27 abr. 2024.

BIANCHI Tiago. **WhatsApp in Brazil – Statistics & Facts. Statista.** Statista, 2025 Disponível em: https://www.statista.com/topics/7731/whatsapp-in-brazil/#topicOverview . Acesso em 13 mai. 2025.

BLIP. **Chatbot: o que é, como funciona, benefícios e cases**. Blip Blog, 11 nov. 2024. Disponível em: https://www.blip.ai/blog/chatbots/chatbot/#. Acesso em 08 jun. 2024.

BOOTH, Harry. ***Cohere Command R+: the 200 Best Inventions of 2024***. Time, 30 out. 2024. Disponível em: https://time.com/7094931/cohese-command-r/. Acesso em: 1 jun. 2025.

CALADO, Andréa. ***Entenda o que é um webhook e como ele funciona*.** Rock Content, 1 jun. 2022. Disponível em: https://rockcontent.com/br/blog/o-que-e-um-webhook/. Acesso em: 5 jun. 2025

CARMAGO, Gabriel. **GEMINI (ex-Bard): o que é e como usar a ferramenta de inteligência artificial do Google**. Rock Content, 14 mar. 2024. Disponível em: https://rockcontent.com/br/blog/gemini-inteligencia-artificial-do-google/. Acesso em 21 fev. 2025.

COSTA, Israel. **Autenticação e Autorização de uma API.** DIO, 15 jul. 2023. Disponível em: https://www.dio.me/articles/autenticacao-e-autorizacao-de-uma-api. Acesso em 27 fev. 2025.

CRUZ Leônicio Teixeira, et al. **ASSISTENTES VIRTUAIS INTELIGENTES E CHATBOTS.** 1ª ed. Rio de Janeiro: Brasport 2018.

DATA SCIENCE ACADEMY. **Guia completo sobre inteligência artificial generativa.** Data Science Academy, 29 jan. 2025. Disponível em: https://blog.dsacademy.com.br/guia-completo-sobre-inteligencia-artificial-generativa/. Acesso em 24 mai. 2025.

DAVID, Emília. ***Cohere lança novos modelos de IA para superar a divisão linguística global*.** VentureBeat, 24 out. 2024. Disponível em: https://venturebeat.com/ai/cohere-launches-new-ai-models-to-bridge-global-language-divide/. Acesso em: 1 jun. 2025.

DINO. **CHATBOTS TÊM PROJEÇÃO DE CRESCIMENTO DE 23% ATÉ 2028.** Valor Econômico, 21 nov. 2023. Disponível em: https://valor.globo.com/patrocinado/dino/noticia/2023/11/21/chatbots-tem-projecao-de-crescimento-de-23-ate-2028.ghtml. Acesso em 27 abr. 2024.

EGON. **API Rest: saiba o que é, para que serve e como usar.** Focus NFe, 31 jul. 2024. Disponível em: https://focusnfe.com.br/blog/o-que-e-api-rest. Acesso em: 06 mai. 2025.

FARIA, Luiz Carlos. **A anatomia de um chatbot.** gaGO.io, 9 abr. 2017. Disponível em: https://gago.io/blog/anatomia-de-um-chatbot/. Acesso em 04 de maio 2024.

GOMEZ, Aidan. ***Command R: Retrieval-Augmented Generation at Production Scale***. Cohere, 11 mar. 2024. Disponível em: https://cohere.com/blog/command-r. Acesso em: 1 jun. 2025.

HOLANDA, Nadynne. **O que é WhatsApp Business API, como funciona e mais!.** RD Station, 27 jan. 2025. Disponível em: https://www.rdstation.com/blog/conversacional/whatsapp-business-api/. Acesso em: 06 mai. 2025.

IBRAMERC. **Chatbots: conheça a história dessa fascinante tecnologia.** Live University, 15 dez. 2017. Disponível em: https://ibramerc.liveuniversity.com/blog/chatbots-e-a-historia-dessa-fascinante-tecnologia-2. Acesso em 04 de maio 2024.

INTELBRAS. **Chatbot: saiba o que é, para que serve e qual escolher.** Blog da Intelbras, 2023. Disponível em : https://blog.intelbras.com.br/chatbot-o-que-e/ . Acesso em 13 abr. 2024.

KOMMO. **Saiba tudo sobre o WhatsApp Business.** Kommo, 4 ago. 2020. Disponível em: https://www.kommo.com/br/blog/whatsapp-business. Acesso em: 06 mai. 2025.

LARAVEL. **Documentação Laravel 12.x**. Laravel, 24 fev. 2025. Disponível em: https://laravel.com/docs. Acesso em: 29 maio 2025.

LEADSTER. **Chatbot: o que é, história e futuro da ferramenta.** Leadster, [s.d.]. Disponível em: https://leadster.com.br/chatbot/#2. Acesso em 04 de maio 2024.

LEMOS Maxmilian, M. F.; ET AL. **Aplicabilidade da arquitetura MVC em uma aplicação web (WebApps)**. RE3C - Revista Eletrônica Científica de Ciência da Computação, v. 8, n. 1, 2013

MAIA João. **Introdução ao Laravel: Fundamentos e Estrutura Básica.** DIO, 27 dez. 2023. Disponível em: https://www.dio.me/articles/introducao-ao-laravel-fundamentos-e-estrutura-basica. Acesso em 25 de maio 2024.

MARCIO. **Padrão MVC - Java Magazine.** Java Magazine, 2011. Disponível em: https://www.devmedia.com.br/padrao-mvc-java-magazine/21995. Acesso em 08 jun. 2024.

MELO, Diego. **O que é uma API?[Guia para iniciantes].** Tecnoblog, 27 abr. 2021. Disponível em: https://tecnoblog.net/responde/o-que-e-uma-api-guia-para-iniciantes/. Acesso em 21 fev. 2025.

META**. Plataforma do WhatsApp Business**. Meta for Developers, [2025?]. Disponível em: https://developers.facebook.com/docs/whatsapp/overview. Acesso em: 21 mar. 2025.

MORAIS, Flávio Daniel Borges de; CASTELO BRANCO, Valdec Romero. ***A inteligência artificial: conceitos, aplicações e controvérsias***. In: SIMPÓSIO

INTERNACIONAL DE CIÊNCIAS INTEGRADAS DA UNAERP – CAMPUS GUARUJÁ, 20., 2023, Guarujá. Anais [...]. Guarujá: UNAERP, 2023. Disponível em: https://www.unaerp.br/documentos/5528-a-inteligencia-artificial-conceitos-aplicacoes-e-controversias/file. Acesso em: 5 jun. 2025.

NETO, Ricardo. **O que é Laravel: principais recursos, onde usar e vantagens do framework PHP.** Hostinger, 17 mar. 2025. Disponível em: https://www.hostinger.com/br/tutoriais/o-que-e-laravel. Acesso em 28 mai. 2025.

NOSOWITZ, Dan; GOODWIN, Michael. **O que é um endpoint de API?.** IBM, 5 ago. 2024. Disponível em: https://www.ibm.com/br-pt/topics/api-endpoint. Acesso em 27 fev. 2025.

ORACLE. **O que é um chatbot?.** Oracle, 2020**.** Disponível em: https://www.oracle.com/br/chatbots/what-is-a-chatbot/ . Acesso em 13 abr. 2024.

PECORARO, Christopher John. **Mastering Laravel**. Packt Pub Ltd. 2015.

PINEDA, Juliana. **Canais de atendimento: conheça os principais para usar na sua empresa.** Tray, 7 dez. 2022. Disponível em : https://tray.com.br/escola/canais-de-atendimento/. Acesso em 08 jun 2024.

PURZ, Michelly. **Os aplicativos de mensagens mais usados no Brasil**. Sinch, 2025. Disponível em: https://sinch.com/pt/blog/aplicativos-de-mensagens-no-brasil/. Acesso em: 13 abr. 2025.

RODRIGUES Mauricio. **Eloquent ORM Prático - Introdução.** Medium, 14 jun. 2018. Disponível em: https://medium.com/laraveltips/eloquent-avan%C3%A7ado-introdu%C3%A7%C3%A3o-d0515fe0f3c6. Acesso em 25 de maio 2024.

SCHENDES William. **WhatsApp é utilizado por 95% das empresas brasileiras, mostra pesquisa.** Olhar Digital, 2023. Disponível em:

https://olhardigital.com.br/2023/10/28/internet-e-redes-sociais/whatsapp-e-utilizado-por-95-das-empresas-brasileiras-mostra-pesquisa/ . Acesso em 13 abr. 2024.

SMITH, Steve. **Visão geral do ASP.NET Core MVC.** Microsoft Learn, 11 jun. 2024. Disponível em: https://learn.microsoft.com/pt-br/aspnet/core/mvc/overview?view=aspnetcore-8.0. Acesso em 08 ago. 2024.

STRYKER, Cole; HOLDSWORTH, Jim. ***O que é PLN (processamento de linguagem natural)?***. IBM, 11 ago. 2024. Disponível em: https://www.ibm.com/br-pt/think/topics/natural-language-processing. Acesso em: 5 jun. 2025.

TAVARES, Josafá: **Além do GPT-4: Por que o Google Gemini é a próxima promessa de evolução da Inteligência Artificial?.** Mindtek, 21 dez. 2023. Disponível em: https://www.mindtek.com.br/2023/12/tudo-sobre-o-gemini-nova-ia-google/. Acesso em 21 fev. 2025.

THEODORO, Juliana. **Elementos da Comunicação.** Significados.com.br, 4 jun. 2019. Disponível em: https://www.significados.com.br/elementos-da-comunicacao/. Acesso em 08 jun. 2024.

VALERI, Vitor; TOLEDO, Victor. **Prompt em IA Generativa: o que é, para que serve e como usar.** Tecnoblog, 2024. Disponível em: https://tecnoblog.net/responde/prompt-em-ia-generativa-o-que-e-para-que-serve-e-como-usar/. Acesso em 24 mai. 2025.