Universidade Federal de Ouro Preto
Escola de Minas
Departamento de Engenharia de Produção, Administração e Economia

João Vinícius Frugencio de Souza

Template Estrutural Open-Source para Chatbots Conversacionais via WhatsApp Cloud API

Ouro Preto
2025

João Vinícius Frugencio de Souza

Template Estrutural Open-Source para Chatbots Conversacionais via WhatsApp Cloud API

Monografia apresentada ao Curso de Engenharia de Produção da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Produção.

Orientador: Prof. Me. Cristiano Luís Turbino de França e Silva

Ouro Preto
2025

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA DE PRODUCAO, ADMINISTRACAO E ECON

FOLHA DE APROVAÇÃO

João Vinícius Frugencio de Souza

Template Estrutural Open-Source para Chatbots Conversacionais via WhatsApp Cloud API

Monografia apresentada ao Curso de Engenharia de Produção da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Engenheiro de Produção

Aprovada em 18 de Dezembro de 2025

Membros da banca

Mestre - Cristiano Luís Turbino de França e Silva - Orientador(a) Universidade Federal de Ouro Preto
Doutor - Helton Cristiano Gomes - Universidade Federal de Ouro Preto
Doutor - Yã Grossi Andrade - Universidade Federal de Ouro Preto

Cristiano Luís Turbino de França e Silva, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 18/12/2025.

Documento assinado eletronicamente por **Cristiano Luis Turbino de Franca e Silva**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 18/12/2025, às 19:51, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Yã Grossi Andrade**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 24/12/2025, às 17:49, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **1035101** e o código CRC **301DA2D2**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591540 - www.ufop.br

Dedico este trabalho à minha avó que foi total exemplo de força, sabedoria e amor incondicional, cuja presença e palavras sempre me inspiraram a seguir com fé e determinação.

Agradecimentos

Agradeço primeiramente a Deus, pela força, sabedoria e oportunidades concedidas ao longo desta caminhada.

Ao Professor **Cristiano Luís**, meu orientador, por ter acreditado em meu potencial desde o início, por me conceder meu primeiro projeto e por ter me apresentado a programação e ciencia de dados — áreas que transformaram completamente minha trajetória acadêmica e profissional. Sem sua orientação e paciência, este trabalho e todos os meus atuais projetos, de fato, não existiriam.

Ao Professor **Helton Gomes**, pela constante dedicação à criação de projetos inovadores e por incentivar os alunos a explorarem novas ideias e tecnologias com propósito.

Ao Professor **Gustavo Nikolaus**, pela amizade, apoio e por sempre estar disposto a ajudar dentro e fora da sala de aula.

Ao Professor **Yã Grossi**, pela amizade e pelas parcerias em projetos que tanto contribuíram para meu crescimento técnico e pessoal.

Ao Professor **André Luís**, por compartilhar ensinamentos valiosos sobre empreendedorismo e por inspirar caminhos que sigo até hoje.

Estendo meus agradecimentos ao amigo **Luciano Lages Torres**, que acreditou no meu trabalho e confiou em sua aplicação prática, possibilitando a concretização e validação de meus conhecimentos academicos.

À **República Tabor**, meu lar durante a graduação, onde encontrei amizade, companheirismo e apoio.

Aos amigos que caminharam comigo ao longo desta trajetória: **Thiago**, parceiro de trabalho e de tantas realizações em conjunto; **Talles Francisco**, meu amigo e colega de graduação, por sua amizade leal e incentivo constante; **Fernando Alzamora** e **Matheus Rezende**, pela parceria e pelos momentos de aprendizado compartilhado; e à **Marielly Araújo** e **Pedro Augusto**, por sua amizade e presença constante.

A todos os professores do **Departamento de Engenharia de Produção da Escola de Minas (DEPRO)**, que contribuíram de forma direta ou indireta para minha formação acadêmica e profissional, deixo meu sincero reconhecimento e gratidão.

Por fim, a todos que, de alguma forma, estiveram presentes nesta caminhada — oferecendo apoio, amizade, conhecimento ou inspiração —, meu mais profundo **muito obrigado**.

"Nada é mais poderoso do que uma ideia cujo tempo chegou."
Victor Hugo

Resumo

Este trabalho apresenta um template estrutural open-source para o desenvolvimento de chatbots conversacionais integrados à WhatsApp Cloud API, com foco em arquitetura modular, reutilização e escalabilidade. A pesquisa é de natureza aplicada, com abordagem qualitativa e caráter experimental. A proposta é disponibilizar ao leitor uma base de código que abstrai camadas de servidor (Node.js/Express), controle de mensagens e códigos utilitários, de modo que o desenvolvedor concentre seus esforços apenas na lógica de negócio do bot. A validação experimental empregou Ngrok para homologação de webhooks e testes de desempenho, estabilidade e tratamento de falhas, incluindo o envio/recebimento de texto, imagens e botões interativos. Como aplicação prática, o template foi utilizado em uma empresa de engenharia em Belo Horizonte para vistorias veiculares via WhatsApp: o motorista responde a um checklist guiado (placa obrigatória, itens de segurança e integridade, fotos e observações), e as respostas são armazenadas em banco de dados, no caso deste projeto, optou-se pelo PostgreSQL. Uma API REST desenvolvida em Python, abastece um dashboard web que permite consultar vistorias por placa, visualizar fichas, gerar relatórios e baixar PDFs. Os resultados indicam que a solução é estável, escalável e extensível, confirmando sua adequação como infraestrutura base para projetos de automação conversacional e como contribuição reutilizável à comunidade.

Palavras-chave: chatbot; WhatsApp Cloud API; arquitetura modular; open-source; Node.js; PostgreSQL; FastAPI; automação conversacional; checklist de vistoria.

Abstract

This work presents an open-source structural template for developing conversational chatbots integrated with the WhatsApp Cloud API, with emphasis on modular architecture, reuse, and scalability. The research is applied in nature, with a qualitative and experimental approach. The proposal is to provide a codebase that abstracts the server layer (Node.js/Express), message handling, and utility modules so that developers can focus primarily on the bot's business logic. Experimental validation used Ngrok for webhook homologation and for testing performance, stability, and failure handling, including the sending and receiving of text, images, and interactive buttons. As a practical application, the template was deployed in an engineering company in Belo Horizonte to support vehicle inspections via WhatsApp: drivers answer a guided checklist (mandatory license plate, safety and integrity items, photos, and comments), and the responses are stored in a PostgreSQL database. A REST API developed in Python feeds a web dashboard that allows users to query inspections by license plate, view records, generate reports, and download PDFs. The results indicate that the solution is stable, scalable, and extensible, confirming its suitability as a base infrastructure for conversational automation projects and as a reusable contribution to the community.

Keywords: chatbot; WhatsApp Cloud API; modular architecture; open-source; Node.js; PostgreSQL; FastAPI; conversational automation; vehicle inspection.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
DELETE	Método HTTP DELETE (remoção de recurso)
GET	Método HTTP GET (leitura de recurso)
Git	Sistema de controle de versão distribuído
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
POST	Método HTTP POST (envio/criação de recurso)
PUT	Método HTTP PUT (atualização de recurso)
REST	Representational State Transfer
SI	Sistema de Informação
TI	Tecnologia da Informação
URL	Uniform Resource Locator

Sumário

1 Introdução

A comunicação online faz parte do dia a dia das pessoas, empresas e órgãos públicos, e uma grande parte dessas interações acontece por aplicativos de mensagem, em especial o WhatsApp. Nesse sentido, os chatbots surgem como uma forma simples de automatizar atendimentos, coletar dados e oferecer serviços sem exigir que o usuário instale novos aplicativos ou acesse websites no navegador. Porém, por trás dessa conversa aparentemente "natural", existe toda uma estrutura técnica que precisa ser planejada, desenvolvida e mantida. Este trabalho apresenta uma visão geral dessa estrutura e propõe um template estrutural open-source para criação de chatbots conversacionais integrados à WhatsApp Cloud API.

A motivação central desta pesquisa está no fato de que muitos desenvolvedores repetem, em cada novo projeto, as mesmas etapas de configuração de servidor, integração com a API da Meta, tratamento de mensagens e conexão com banco de dados, em vez de reutilizar uma base comum e bem organizada. Ao mesmo tempo, órgãos públicos e empresas precisam de soluções confiáveis para automatizar fluxos como pesquisas de opinião e vistorias veiculares, usando um canal já conhecido pela população. Nesse contexto, O'Brien e Marakas (2013) destacam que iniciativas de automação e uso de canais digitais só geram resultados consistentes quando apoiadas em processos bem definidos e bases de dados confiáveis.

O objetivo deste trabalho é propor e implementar um template estrutural open-source para o desenvolvimento de chatbots conversacionais integrados à WhatsApp Cloud API. Para isso, têm-se os seguintes objetivos específicos: mapear os requisitos mínimos de infraestrutura e de lógica de negócio para esse tipo de solução; definir uma arquitetura modular em camadas, separando servidor, integração com a API, controle de mensagens e acesso a dados; implementar e documentar esse template em código aberto, de forma organizada e reutilizável; e validar sua aplicação em um caso real de vistoria veicular, avaliando sua estabilidade, seu comportamento sob múltiplas requisições e a facilidade de adaptação a novas lógicas de conversação.

A metodologia de pesquisa é de natureza aplicada, com abordagem qualitativa e caráter descritivo e exploratório. O estudo é conduzido como pesquisa-ação, na qual o autor desenvolve, testa e ajusta o template em ciclos sucessivos de implementação e validação, utilizando tecnologias de desenvolvimento web, banco de dados e controle de versão.

2 Referencial teórico

Este capítulo apresenta os conceitos que sustentam o desenvolvimento deste trabalho, abordando sistemas de informação, comunicação automatizada, arquitetura modular, integração entre aplicações, computação em nuvem e fundamentos relacionados ao uso de software livre.

2.1 Sistemas de Informação e Automação de Processos

Laudon (2022) trata os sistemas de informação como arranjos coordenados de pessoas, processos e tecnologia, responsáveis por coletar, processar, armazenar e distribuir informações para apoiar as operações rotineiras e a tomada de decisão nas organizações. Seguindo a mesma linha de pensamento, M.Stair e W.Reynolds (2015) definem os sistemas de informação como uma combinação organizada de pessoas, software, hardware e dados, ressaltando que o valor do sistema não está apenas na tecnologia em si, mas na capacidade de transformar dados em informação útil para quem toma decisões. Já Potter (2007) chamam atenção para a dimensão gerencial desses sistemas, ao defender que eles devem ser pensados como instrumentos para apoiar a estratégia e não apenas como ferramentas operacionais, aproximando a linguagem da TI da linguagem do negócio.

Na mesma direção, Prado e Souza (2014) reforça a ideia de que os sistemas de informação são sistemas sociotécnicos, resultantes da interação entre tecnologia, pessoas e processos organizacionais, e só fazem sentido quando conectados a objetivos concretos da organização. Da mesma forma, Rezende (2011) enfatiza que o planejamento de TI deve estar integrado ao planejamento estratégico, de modo que a informação circule entre áreas distintas e dê suporte às decisões. Audy (2005) destacam que, à medida que cresce a necessidade de integrar dados e serviços, aumenta também o espaço para automação de rotinas e para o uso de canais digitais de atendimento.

Nesse cenário, O'Brien e Marakas (2013) lembram que iniciativas de automação só geram resultados consistentes quando estão acopladas a processos bem definidos e a bases de dados confiáveis, estrutura que, segundo os autores, permite que canais automatizados respondam de forma padronizada, registrem o histórico das interações e alimentem os demais sistemas corporativos com informações atualizadas.

2.2 Chatbots e Comunicação Automatizada

Raj (2019) mostra que um chatbot pode ser estruturado como uma aplicação que combina processamento de linguagem natural, regras de negócio e um canal de entrega (mensageiro, web ou aplicativo), de modo que o usuário interaja em linguagem natural e o sistema responda de forma programática. Nessa mesma linha, Cruz, Alencar e Schmitz

(2019) destacam que assistentes virtuais inteligentes e chatbots articulam essa camada técnica com preocupações de experiência do cliente, desenhando diálogos e fluxos de atendimento consistentes ao longo do tempo. De forma mais prática, Frizzarin e Frizzarin (2023) evidenciam como bots em mensageiros podem ser construídos para executar tarefas específicas, como consultas, notificações, apoio ao comércio eletrônico e pesquisas de opinião, reforçando a ideia de chatbots como serviços automatizados acoplados a um canal de comunicação.

No contexto dos aplicativos de mensagem, Gonçalez (2020) apontam que o canal de comunicação passa a desempenhar papel central na jornada do usuário, influenciando diretamente o desenho dos fluxos conversacionais, as formas de interação e os requisitos de integração com sistemas legados. Os autores destacam que, em plataformas amplamente utilizadas, como mensageiros móveis, aspectos como tipos de mensagem suportados, mecanismos de autenticação e políticas de uso definidos pelo provedor da plataforma condicionam a forma como os chatbots são projetados e implantados.

Sob a perspectiva dos sistemas de informação, Laudon (2022) ressalta que iniciativas de automação em canais digitais de atendimento dependem da integração entre processos organizacionais, recursos tecnológicos e bases de dados, de modo que as interações realizadas nesses canais possam ser registradas, recuperadas e utilizadas para apoiar a tomada de decisão. Essa integração, segundo o autor, é fundamental para que aplicações conversacionais operem de forma contínua, mantenham o histórico das interações e garantam disponibilidade e consistência na prestação do serviço.

2.3 Arquitetura de Software Modular

Pressman (2011) aponta que a modularização reduz a complexidade e aumenta a manutenibilidade do software, na medida em que diminui a dificuldade de entendimento do código. Sommerville (2011) reforça que módulos bem definidos permitem ciclos de desenvolvimento em paralelo e que a arquitetura deve deixar claras as dependências entre componentes. Na mesma linha, Wazlawick (2019) destaca que a modularidade só é efetiva quando combinada com forte coesão interna entre as funções e baixo acoplamento entre módulos, de forma que cada parte tenha uma responsabilidade bem definida e possa evoluir sem provocar efeitos imprevisíveis no restante do sistema.

Bass, Clements e Kazman (2021) trata a arquitetura como uma decomposição do sistema em componentes e conectores orientada por atributos de qualidade, como desempenho, disponibilidade e modificabilidade, mostrando que a escolha dos módulos não é apenas técnica, mas estratégica para a evolução da solução e a inserção de novos recursos. De forma complementar, Richards e Ford (2024) discute a importância de pensar em módulos como unidades que combinam coesão, granularidade adequada e contratos bem definidos, permitindo que alterações estruturais sejam feitas de modo incremental, sem reescrever o sistema inteiro.

No contexto brasileiro, há também uma preocupação prática em conectar modularidade com cenários reais de desenvolvimento. Santana (2024) argumenta que uma arquitetura modular bem desenhada facilita atacar requisitos não funcionais, como escalabilidade, segurança e confiabilidade, porque cada módulo pode ser dimensionado e protegido de forma específica. Em termos de organização do código, isso implica, segundo o autor, separar claramente partes voltadas à infraestrutura, à integração com serviços externos e à lógica de negócio, evitando estruturas muito rígidas e pouco flexíveis.

2.4 Integração de Sistemas, APIs e WhatsApp Cloud API

Segundo Alves (2014) e Audy (2005), uma API (Application Programming Interface) pode ser entendida como um "contrato" de software que define quais operações podem ser realizadas, como devem ser chamadas e em qual formato os dados serão trocados. Em geral, ela especifica endpoints, métodos HTTP, formatos de envio (geralmente em JSON), códigos de retorno e mecanismos de autenticação. Na prática, é essa camada que permite que dois sistemas — muitas vezes desenvolvidos em linguagens diferentes, hospedados em infraestruturas distintas e pertencentes a organizações diferentes — consigam "conversar" de forma previsível e segura. Grande parte das APIs contemporâneas segue o estilo REST (Representational State Transfer), em que cada recurso é identificado por uma URL, operações como GET, POST, PUT e DELETE são utilizadas de forma consistente e o servidor não precisa manter o estado da interação entre uma requisição e outra.

Richardson, Amundsen e Ruby (2013) destacam que, quando os endpoints estão bem documentados e seguem o modelo RESTful, a comunicação entre clientes e serviços é facilitada, podendo ser complementada por fluxos de notificação baseados em *webhooks*. Nesse arranjo, em vez de o cliente consultar continuamente a API para saber se há novidades, o próprio provedor envia requisições HTTP a uma URL pública sempre que ocorre um evento relevante, como o recebimento de uma nova mensagem ou a alteração de status de um recurso. Segundo o autor, essa abordagem reduz o acoplamento entre os sistemas, melhora a escalabilidade e permite o processamento de eventos em tempo quase real.

Além desses princípios, Fowler (2018) apontam que integrações distribuídas tendem a ser mais estáveis quando os limites entre serviços são bem definidos e a comunicação entre eles segue padrões claros. Em arquiteturas orientadas a serviços, cada componente deve assumir responsabilidades específicas e expor interfaces bem projetadas, de forma que mudanças internas não quebrem o funcionamento do restante do sistema. Nessa mesma direção, Filho (2009) ressalta que a definição de interfaces estáveis e o uso de especificações bem estabelecidas são fundamentais para reduzir riscos em sistemas que dependem de integração entre diferentes componentes.

No caso da WhatsApp Cloud API, a Meta fornece um serviço *backend* em nuvem responsável por receber e encaminhar mensagens ao número de WhatsApp do negócio, disponibilizar endpoints REST para envio de mensagens, disparar *webhooks* HTTPS quando

o usuário interage e aplicar regras de segurança baseadas em *tokens* de acesso. Do ponto de vista arquitetural, as aplicações que consomem essa API assumem o papel de cliente, enquanto a infraestrutura da Meta atua como servidor, configurando um arranjo típico de arquitetura cliente–servidor em que as comunicações ocorrem por meio de requisições e respostas HTTP de forma assíncrona Tanenbaum, Feamster e Wetherall (2021). Assim, a automação conversacional ocorre por meio de uma API oficial que intermedeia, de maneira segura, o mensageiro e as aplicações que consomem esses serviços.

2.5 Engenharia de Software e Desenvolvimento do Template

Pressman (2011) destaca que processos de engenharia de software conduzidos em ciclos iterativos, envolvendo análise, projeto, implementação e testes, tendem a produzir sistemas mais estáveis, pois permitem revisar requisitos e corrigir falhas em estágios iniciais do desenvolvimento. Na mesma direção, Sommerville (2011) reforça que a clareza na separação de etapas e na definição de atividades reduz a complexidade do trabalho e facilita a manutenção de soluções que serão evoluídas ou reutilizadas por diferentes equipes ao longo do tempo.

Essa perspectiva é complementada por Bezerra (2014), ao apontar que a engenharia de software contemporânea exige não apenas processos bem definidos, mas também a capacidade de estruturar sistemas de maneira a favorecer extensões futuras. O autor observa que projetos que dependem de integração com APIs externas se beneficiam de arquiteturas modulares que isolam componentes e reduzem o impacto de mudanças inevitáveis ao longo do ciclo de vida do software, principalmente em contextos distribuídos e sujeitos a evolução tecnológica constante.

Além disso, Larman (2007) destaca que o desenvolvimento orientado a objetos, quando associado a boas práticas de análise de requisitos e a iterações curtas, contribui para maior flexibilidade na evolução de sistemas. Essa abordagem favorece a identificação de responsabilidades e a divisão do software em unidades menores e coesas, estratégia considerada fundamental para aplicações que precisam lidar com diferentes fluxos de mensagens, múltiplos tipos de eventos e regras de negócio específicas. Nessa mesma linha, Martin (2020) argumenta que a manutenção de um código limpo, com funções claras, modularização adequada e eliminação de dependências desnecessárias, é um fator determinante para a longevidade de soluções de software e para sua possibilidade de reutilização por outros desenvolvedores.

Por fim, Valente (2020) ressalta que projetos de software que pretendem ser amplamente adotados devem considerar, desde o início, a escolha de tecnologias com ecossistemas maduros, documentação acessível e forte apoio comunitário. Segundo o autor, a combinação entre uma arquitetura bem modularizada, boas práticas de engenharia de software e o uso de plataformas consolidadas aumenta a probabilidade de que o código possa ser compreendido, mantido e estendido em diferentes contextos de uso.

2.6 Computação em Nuvem e Disponibilidade

Taurion (2009) explica que a computação em nuvem fornece recursos de forma elástica e sob demanda, permitindo que aplicações voltadas à internet fiquem sempre disponíveis e escalem quando o volume de acesso cresce. Da mesma forma, Veras (2015) aponta que esse modelo de provisão remota favorece sistemas que precisam expor endpoints públicos para receber notificações (como webhooks), porque a própria infraestrutura de nuvem já oferece disponibilidade, endereço público e mecanismos de segurança necessários para esse tipo de integração. Nessa linha, Velte (2012) destaca que a nuvem permite ajustar dinamicamente capacidade de processamento e armazenamento conforme a carga do sistema, o que é essencial para manter desempenho aceitável mesmo em períodos de pico de acesso.

De forma complementar, Erl e Monroy (2024) destacam que a computação em nuvem é uma infraestrutura fundamental para sistemas modernos, pois permite alocar recursos computacionais conforme a demanda, com mecanismos nativos de redundância, recuperação automática e balanceamento de carga. Essas características tornam a nuvem especialmente adequada para aplicações que exigem operação contínua e baixa latência, já que o provedor entrega serviços de processamento, armazenamento e rede de forma integrada, reduzindo as chances de interrupções mesmo em cenários de alto uso simultâneo.

2.7 Software Livre e Colaboração

Raymond (1998) apresenta uma visão de desenvolvimento colaborativo em que o software é produzido de forma aberta e incremental, com participação ativa de uma comunidade distribuída. Nesse modelo, o código-fonte é disponibilizado sob licenças que garantem ao usuário liberdades como executar, estudar, modificar e redistribuir o software, o que o diferencia de soluções proprietárias, nas quais o código permanece fechado. Uma base ampla de usuários, desenvolvedores, pesquisadores e estudantes, distribuída e ativa, consegue encontrar erros com mais rapidez, propor melhorias e adaptar o sistema a diferentes contextos, síntese que o autor descreve como o "modelo bazar" de desenvolvimento colaborativo.

No contexto brasileiro, Silveira (2004) destaca que o software livre também assume um papel político e social, ao favorecer a democratização do acesso ao conhecimento e a redução da dependência tecnológica de grandes empresas. Ao tratar o código como bem comum, o autor argumenta que comunidades de desenvolvedores e instituições públicas podem adaptar e compartilhar soluções tecnológicas de acordo com suas próprias necessidades, fortalecendo aquilo que denomina soberania informacional e a circulação de conhecimento na sociedade.

Do ponto de vista econômico, Taurion (2004) mostra que o software livre viabiliza diferentes modelos de negócio baseados na prestação de serviços, na customização de soluções, em treinamentos e em suporte técnico especializado, em vez de uma simples venda de li-

cenças de uso. Esses modelos permitem que empresas e organizações públicas reduzam custos com licenças ao mesmo tempo em que apoiam um ecossistema local de desenvolvimento de software, criando oportunidades de inovação e especialização tecnológica.

Subramanian e Jude (2020) apresentam uma introdução sistemática aos fundamentos do software livre e de código aberto, abordando conceitos, licenças, comunidades de desenvolvimento e implicações para organizações que pretendem adotar esse tipo de solução. Os autores ressaltam que a adoção de software livre envolve uma cultura de compartilhamento, revisão e melhoria contínua, em que o código é constantemente inspecionado, corrigido e ampliado pela própria comunidade usuária.

Por fim, Guesser (2006) analisa o software livre como um fenômeno social que envolve disputas sobre propriedade intelectual, políticas públicas de tecnologia e estratégias de desenvolvimento nacional. No caso brasileiro, o autor mostra que sua adoção por órgãos públicos influencia não apenas a infraestrutura técnica, mas também políticas de transparência e abertura de código. Nesse contexto, o uso de componentes livres em arquiteturas de software favorece a criação de soluções reutilizáveis e alinhadas a princípios de colaboração e compartilhamento de conhecimento.

2.7.1 Uso do Git

O Git é um sistema de controle de versão distribuído utilizado para registrar alterações em arquivos de código-fonte e gerenciar diferentes estados de um repositório ao longo do tempo. Sua arquitetura permite a manutenção de um histórico de mudanças em repositórios locais e remotos.

De acordo com Chacon e Straub (2014), o Git foi projetado para ser rápido, eficiente e seguro, oferecendo recursos como *branches*, *merges* e um modelo distribuído que reduz riscos de perda de dados e de inconsistências. Aquiles (2014) mostram como o uso disciplinado de *branches*, *tags* e repositórios remotos facilita o trabalho em equipe e a rastreabilidade das alterações em projetos reais, permitindo acompanhar quem fez cada modificação, quando e com qual propósito. De forma complementar, Silverman (2013) ressalta que o modelo distribuído do Git possibilita que cada desenvolvedor experimente localmente e só compartilhe suas mudanças quando estiverem estáveis, o que contribui para reduzir conflitos e erros de compilação durante o processo de integração.

3 Metodologia

De acordo com (VENAZI et al., 2016) , a metodologia de pesquisa deve ser classificada quanto à natureza, abordagem, objetivos e procedimentos técnicos adotados. Assim, este trabalho segue esse modelo de categorização metodológica, conforme descrito a seguir.

- natureza**:** A pesquisa é de natureza aplicada, pois busca gerar um produto prático capaz de solucionar uma necessidade real. O estudo visa desenvolver uma infraestrutura de código aberto (open-source) voltada à criação de chatbots conversacionais integrados à WhatsApp Cloud API. O resultado é um template modular e reutilizável, que reduz o esforço técnico na integração com a API e promove o reuso de código e arquitetura entre diferentes projetos.

- abordagem**:** A abordagem adotada é qualitativa, uma vez que se atuou diretamente em todas as etapas do desenvolvimento, interpretando e analisando os resultados obtidos. A observação do comportamento do chatbot, dos fluxos de mensagens e da estrutura modular foi essencial para compreender os fenômenos técnicos envolvidos na integração entre o servidor Node.js e a WhatsApp Cloud API, permitindo o aprimoramento contínuo da solução.

- objetivos**:** A pesquisa possui caráter descritivo e exploratório. É descritiva por apresentar detalhadamente a estrutura da arquitetura proposta — suas camadas, módulos e funções de integração. É também exploratória, pois busca investigar novas formas de projetar uma base genérica e flexível para chatbots, permitindo sua aplicação em diferentes contextos de automação conversacional.

- procedimentos técnicos**:** Quanto aos procedimentos técnicos, trata-se de uma pesquisa-ação, pois o autor participou diretamente do processo de desenvolvimento **e** validação do sistema. Foram utilizadas ferramentas como Node.js, Express.js**,** PostgreSQL, Visual Studio Code**,** GitLab e Ngrok. Durante a implementação, o pesquisador criou e testou módulos independentes de envio e recebimento de mensagens, controle de tokens, webhooks e gerenciamento de respostas, intervindo de forma prática para aprimorar e validar a estrutura desenvolvida.

:

4 Apresentação e discussão dos resultados

O presente capítulo apresenta a aplicação prática da metodologia proposta no capítulo anterior, detalhando as etapas de construção, estruturação e validação do template estrutural para chatbots conversacionais.

4.1 Concepção da ideia

A criação de chatbots surgiu como resposta a um problema que se tornou cada vez mais comum na atualidade: a resistência dos usuários de smartphones em instalar novos aplicativos. É frequente que a exigência de instalação de aplicativos empresariais, apenas para a execução de uma tarefa simples, iniba o usuário de realizá-la, devido à necessidade de realizar diversas etapas, como download, cadastro, login e aprendizado sobre o uso da interface. Esse processo torna-se repetitivo e cansativo, mesmo quando os aplicativos são considerados intuitivos.

Nesse contexto, o problema apresentado pelo cliente estava diretamente relacionado a essa dificuldade: como criar uma aplicação de pesquisa de opinião que fosse de rápida disseminação e que exigisse o mínimo de esforço do usuário?

A solução encontrada foi o desenvolvimento de chatbots de pesquisa integrados ao WhatsApp, plataforma amplamente popular e acessível. Atualmente, o WhatsApp é o aplicativo de mensagens mais utilizado do Brasil e um dos mais populares do mundo, com cerca de 148 milhões de usuários brasileiros ativos em 2024, o que representa aproximadamente 98% dos usuários de smartphones do país (World Population Review, 2024).

A simples divulgação de um número de telefone, que ao ser acessado abre automaticamente uma conversa no aplicativo e inicia o diálogo com o usuário, mostrou-se uma alternativa muito mais acessível e menos onerosa do que a obrigatoriedade de instalar um aplicativo próprio ou acessar um site, por exemplo.

Após a implementação de outros chatbots, como o de vistoria de veículos (criado a partir de novas demandas do mesmo cliente), foi possível observar que a estrutura básica desses sistemas se repete. As funções e a estruturação interna dos chatbots apresentavam padrões semelhantes, compostos por módulos de envio de mensagens, controle de servidor e utilitários básicos, variando apenas na lógica específica de cada aplicação, de acordo com sua finalidade.

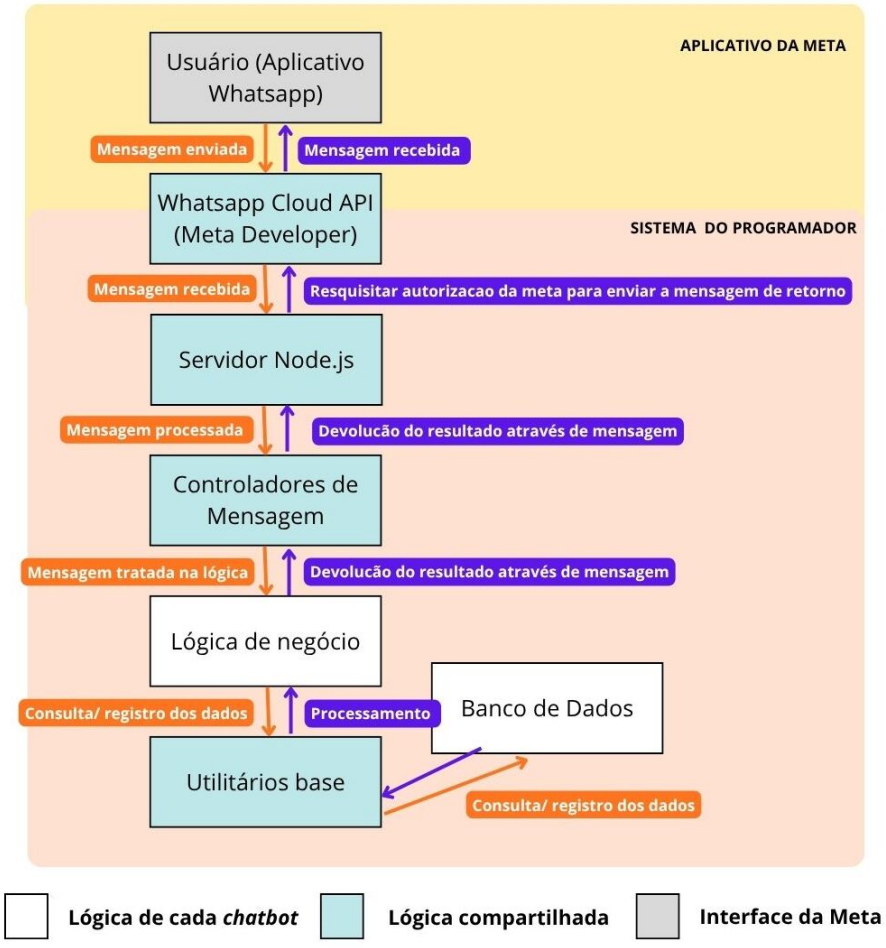
Dessa forma, surgiu a ideia central deste trabalho: disseminar essa estrutura-base, transformando-a em um template open-source que permita a outros desenvolvedores reutilizar essa infraestrutura pronta, concentrando seus esforços apenas na criação da lógica e das regras de negócio de cada chatbot, conforme suas necessidades específicas.

4.2 Estrutura e Arquitetura do Sistema

A estrutura proposta neste trabalho foi desenvolvida a partir de uma arquitetura modular em camadas, com foco em reutilização, baixo acoplamento e independência lógica entre os componentes. O template open-source foi projetado para permitir que o desenvolvedor concentre seus esforços apenas na lógica de negócio de cada chatbot, enquanto toda a infraestrutura de comunicação e integração com a WhatsApp Cloud API permanece pronta e padronizada.

A Figura 1 apresenta o fluxograma geral da arquitetura do sistema, que ilustra o fluxo completo de comunicação entre o usuário final, a API da Meta e o servidor do programador. O diagrama foi dividido em duas grandes áreas de responsabilidade: o Aplicativo da Meta e o Sistema do Programador, representadas visualmente por cores distintas no fundo.

Figura 1 – Fluxograma de comunicação sistema-usuário.



Fonte: Pesquisa direta, 2025.

Na parte superior, localizada na região amarela, encontra-se o Aplicativo da Meta,

que engloba o Usuário (aplicativo WhatsApp) e a WhatsApp Cloud API (Meta Developer). Essa camada é responsável por receber as mensagens enviadas pelos usuários e encaminhá-las aos servidores configurados pelo desenvolvedor. O WhatsApp atua como interface de comunicação, enquanto a API da Meta funciona como um intermediário oficial entre a plataforma de mensagens e o sistema externo, garantindo autenticação, entrega e segurança das requisições.

O bloco da WhatsApp Cloud API encontra-se propositalmente posicionado entre as duas áreas (meio amarelo e meio rosado), simbolizando sua dupla natureza operacional. Isso ocorre porque, embora a API seja hospedada e mantida pela Meta, toda a implementação prática — como o tratamento de erros, o gerenciamento de tokens, a configuração de endpoints e o recebimento das requisições via webhook — é realizada pelo desenvolvedor dentro do seu próprio sistema. Assim, a API funciona como um ponto de transição entre o domínio da Meta e o ambiente controlado pelo programador.

Na sequência, na área rosada, inicia-se o Sistema do Programador, composto por módulos interligados que estruturam o template proposto. De forma geral, toda a arquitetura pode ser entendida como um conjunto de blocos bem separados: (i) uma camada de servidor, responsável por receber as requisições HTTP; (ii) uma camada de integração com a WhatsApp Cloud API, que concentra tudo o que diz respeito a tokens, endpoints e detalhes do protocolo; (iii) uma camada de tratamento de mensagens e eventos, que interpreta o que chegou e encaminha para o lugar certo; e (iv) uma camada de lógica conversacional, onde ficam as regras específicas de cada chatbot. Essa decomposição ajuda a manter o código organizado e facilita tanto a manutenção quanto o reuso em outros projetos.

O primeiro módulo é o Servidor Node.js, desenvolvido com o framework Express.js, responsável por receber as mensagens encaminhadas pelo webhook da Meta, validar as credenciais e direcionar o conteúdo para o módulo apropriado.

A escolha pelo uso de Node.js com Express.js não foi aleatória. Por ser uma plataforma orientada a operações de entrada e saída assíncronas, o Node.js lida bem com cenários em que há muitas conexões simultâneas e diversas requisições leves acontecendo ao mesmo tempo, que é justamente o caso de um chatbot exposto na internet. Além disso, trata-se de uma tecnologia com amplo ecossistema de bibliotecas e uma comunidade ativa, o que facilita a resolução de problemas práticos e a evolução do template no futuro.

Em seguida, o fluxo segue para o Controlador de Mensagens, que representa a camada de orquestração das requisições. Essa camada atua como um roteador lógico, identificando o tipo de mensagem recebida (texto, imagem, botão, lista, etc.) e encaminhando-a para a Lógica de Negócio correspondente. A Lógica de Negócio, representada por um bloco branco, simboliza o núcleo variável de cada chatbot — é a parte do sistema que muda conforme a finalidade da aplicação. Enquanto as demais camadas do template permanecem fixas e compartilhadas entre diferentes projetos, a lógica é onde o desenvolvedor define as regras de decisão, os diálogos e os comportamentos personalizados do bot.

Essa separação entre lógica e infraestrutura é o que garante a modularidade e reutilização do código. Abaixo da lógica, o sistema conta com o módulo de Utilitários Base, que fornece funções genéricas, como formatação de mensagens, templates de resposta, tratamento de erros e logs. Esse módulo também realiza a ponte de comunicação com o Banco de Dados, responsável por registrar informações, armazenar respostas simuladas e processar dados necessários ao fluxo da conversa.

A interação entre os utilitários e o banco é representada por setas bidirecionais, simbolizando que há tanto consulta quanto registro de informações. Por fim, o fluxo retorna ao topo do diagrama, seguindo o caminho inverso até o usuário. A mensagem processada percorre novamente as camadas — Controlador, Servidor Node.js e WhatsApp Cloud API — até chegar ao aplicativo do usuário, que recebe a resposta de forma automática e contínua, fechando o ciclo de interação.

Em resumo, a arquitetura do template proposto foi estruturada de modo a isolar responsabilidades: a Meta mantém o ambiente de hospedagem e autenticação; o desenvolvedor controla o tratamento, a lógica e o envio das respostas; e o usuário final interage por meio de uma interface já familiar, o WhatsApp. Essa divisão clara, representada visualmente no fluxograma, reflete a proposta central do trabalho: oferecer uma base estrutural genérica, aberta e reutilizável, capaz de suportar a criação de diferentes tipos de chatbots conversacionais de forma ágil e padronizada.

4.3 Exemplo de Implementação

O código exposto na Tabela 1 exemplifica o funcionamento básico de envio de mensagens via WhatsApp Cloud API, utilizando o módulo Axios para comunicação HTTP. Vale ressaltar que o código contido neste trabalho se encontra disponível em sua íntegra e documentado na plataforma GitLab.

Na prática, o fluxo de desenvolvimento adotado neste trabalho pode ser resumido em quatro etapas principais:

a) modelar o fluxo de mensagens que o bot deve tratar, definindo quais informações serão solicitadas e em qual ordem;

b) projetar as interfaces de comunicação com a API do WhatsApp, especificando endpoints, formatos de mensagem e parâmetros de autenticação;

c) implementar os *handlers* de webhook responsáveis por receber os eventos enviados pela Meta e repassar o conteúdo para a lógica de negócio;

d) testar o endpoint real, validando assinatura, token de acesso e formatos de requisição e resposta, até garantir o funcionamento estável da integração.

Esse encadeamento ajuda a transformar o template em uma base reaproveitável de fato, e não apenas em um exemplo isolado de código.

Nesse contexto, o uso do Git se torna fundamental em práticas recentes de engenharia de software, como desenvolvimento colaborativo, integração contínua e metodologias ágeis, permitindo que equipes controlem versões de código de forma organizada. Ao manter o template versionado e publicado em um repositório GitLab, o projeto passa a estar disponível para estudo, cópia, adaptação e melhoria por outros desenvolvedores. Neste contexto de colaboração, o Git se torna um requisito básico para a difusão da ideia de software livre proposta neste trabalho.

Tabela 1 – Função em JavaScript para envio de mensagem do bot para o usuário.

Trecho de código em JavaScript:

```
export async function sendMessage(
  number: string,
  message: string,
  wmaid?: string
): Promise<any | null> {
  const ctx = "sendTextMessage";
  const body = (message || "").trim() || " ";
  const payload: any = {
    messaging_product: "whatsapp",
    to: number,
    type: "text",
    text: {
      preview_url: false,
      body,
    },
  };
  try {
    const { data } = await api.post(‘/messages‘, payload);
    console.log(‘[WA:${ctx}] OK‘, JSON.stringify(data));
    return data;
  } catch (e) {
    logAxiosError(ctx, e);
    return null;
  }
}
```

Fonte: Pesquisa direta, 2025.

Do ponto de vista da comunicação, toda a sessão do WhatsApp fica sob responsabilidade da Meta, e a aplicação desenvolvida neste trabalho conversa com a plataforma exclusivamente por meio de requisições HTTP direcionadas à WhatsApp Business Platform. Esse é exatamente o modelo descrito na documentação oficial da WhatsApp Business

Platform, que apresenta endpoints como os ilustrados nas Tabelas 2 e 3.

No contexto deste trabalho, o servidor desenvolvido faz o papel de cliente e a infraestrutura da Meta faz o papel de servidor, em um modelo clássico cliente-servidor descrito por Tanenbaum, Feamster e Wetherall (2021), em que as trocas se dão por requisições e respostas HTTP de forma assíncrona — requisito importante para suportar múltiplas conversas simultâneas. Para garantir autenticidade e integridade, utiliza-se aquilo que a própria Meta exige: *tokens* de acesso válidos e *webhooks* expostos em HTTPS com verificação. Assim, a automação conversacional só ocorre porque há uma API oficial que intermedeia, de maneira segura, o mensageiro e a aplicação desenvolvida.

Tabela 2 – Exemplo de requisição à WhatsApp Business Platform.

Exemplo de solicitação para enviar uma mensagem de texto com visualizações de links ativadas e uma string de texto do corpo que contém um link.

```
curl 'https://graph.facebook.com/v24.0/106540352242922/messages' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer EAAJB...' \
  -d '
{
  "messaging_product": "whatsapp",
  "recipient_type": "individual",
  "to": "+16505551234",
  "type": "text",
  "text": {
    "preview_url": true,
    "body": "As requested, here'''s the link to our latest product:
https://www.meta.com/quest/quest-3/"
  }
}'
```

Fonte: Pesquisa direta, 2025.

Tabela 3 – Exemplo de retorno da plataforma após envio da mensagem.

Trecho de resposta JSON retornada pela API.


```
{
  "messaging_product": "whatsapp",
  "contacts": [
    {
      "input": "+16505551234",
      "wa_id": "16505551234"
    }
  ],
  "messages": [
    {
      "id": "wamid.HBgLMTY0NjcwNDM1OTUVAgARGBI1RjQyNUE3NEYxMzAzMzQ5MkEA"
    }
  ]
}
```


Fonte: Pesquisa direta, 2025.

Esses exemplos reforçam que o código JavaScript apresentado na Tabela 1 atua como uma camada de conveniência sobre os endpoints oficiais: o desenvolvedor constrói o *payload*, envia a requisição HTTP e trata o retorno JSON, enquanto toda a manutenção da sessão, entrega de mensagens e disponibilidade da infraestrutura permanece sob responsabilidade da Meta.

4.4 Validação Experimental

Após a implementação do template, o sistema foi submetido a uma fase de validação experimental com o objetivo de testar seu desempenho, estabilidade e comportamento em situações reais de uso. Durante os testes, foi utilizada a ferramenta Ngrok, responsável por criar túneis seguros entre o servidor local e a internet, permitindo que o webhook do WhatsApp Cloud API fosse acessado publicamente durante o processo de homologação. Essa abordagem possibilitou simular requisições reais de usuários, observando o comportamento do sistema em um ambiente controlado, porém representativo de uma operação real.

Os testes contemplaram a análise de diferentes aspectos técnicos, como:

- tempo médio de resposta do servidor às requisições da API;
- estabilidade do webhook sob diferentes cargas e volumes de mensagens;
- tratamento automático de falhas, reconexões e reenvios;

- envio e recebimento de diferentes tipos de conteúdo, incluindo texto, imagem e botões interativos.

Os resultados demonstraram que o template é estável, modular e compatível com diversos contextos de aplicação, apresentando baixo tempo de resposta e comportamento previsível mesmo em situações de múltiplas requisições simultâneas. Ressalta-se que o Ngrok é utilizado apenas para fins de homologação da aplicação, não sendo o servidor que será utilizado em produção.

Ademais, os testes confirmaram a viabilidade técnica da estrutura proposta, validando sua utilização como base genérica para o desenvolvimento de chatbots personalizados.

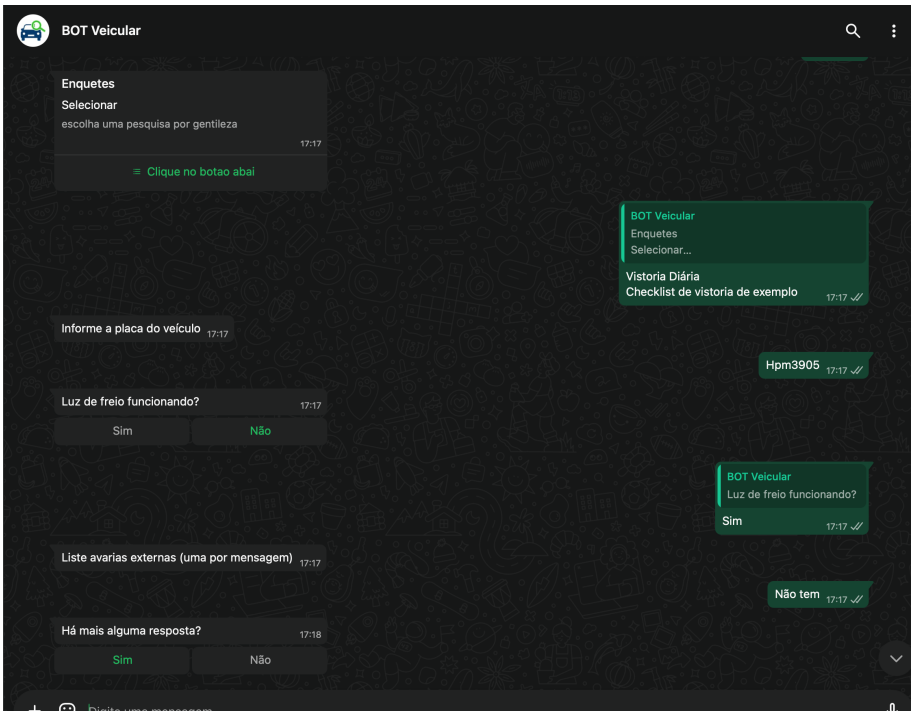
4.5 Aplicação Prática

Como parte da validação técnica e da demonstração do uso prático do template desenvolvido, foi realizada uma aplicação real do sistema em um cliente corporativo do setor de engenharia, sediado em Belo Horizonte. Essa implementação foi configurada como um estudo de campo técnico e o objetivo principal foi validar o comportamento do template em um fluxo conversacional real, com dados concretos sendo processados, armazenados e exibidos por uma aplicação web integrada.

O caso de uso escolhido foi o de vistoria veicular automatizada, no qual motoristas interagem com o chatbot via WhatsApp para responder a uma sequência de perguntas configuradas previamente no servidor. As mensagens são trocadas de forma dinâmica: o bot inicia solicitando a placa do veículo (campo obrigatório) e, em seguida, envia perguntas sobre condições do automóvel, como funcionamento de freios, existência de arranhões, irregularidades externas e outros itens do checklist de inspeção.

Cada resposta enviada pelo motorista é capturada pela WhatsApp Cloud API, tratada no servidor Node.js e posteriormente armazenada em um banco de dados PostgreSQL. A Figura 2 apresenta um exemplo real da conversa realizada no WhatsApp, mostrando o fluxo entre o bot e o motorista. Na imagem é possível observar as perguntas e respostas registradas durante uma vistoria, incluindo a placa “HPM3905”, que corresponde ao mesmo veículo exibido nos demais registros. O bot conduz a interação de maneira orientada, utilizando botões interativos, listas e mensagens de confirmação, oferecendo uma experiência conversacional fluida e intuitiva.

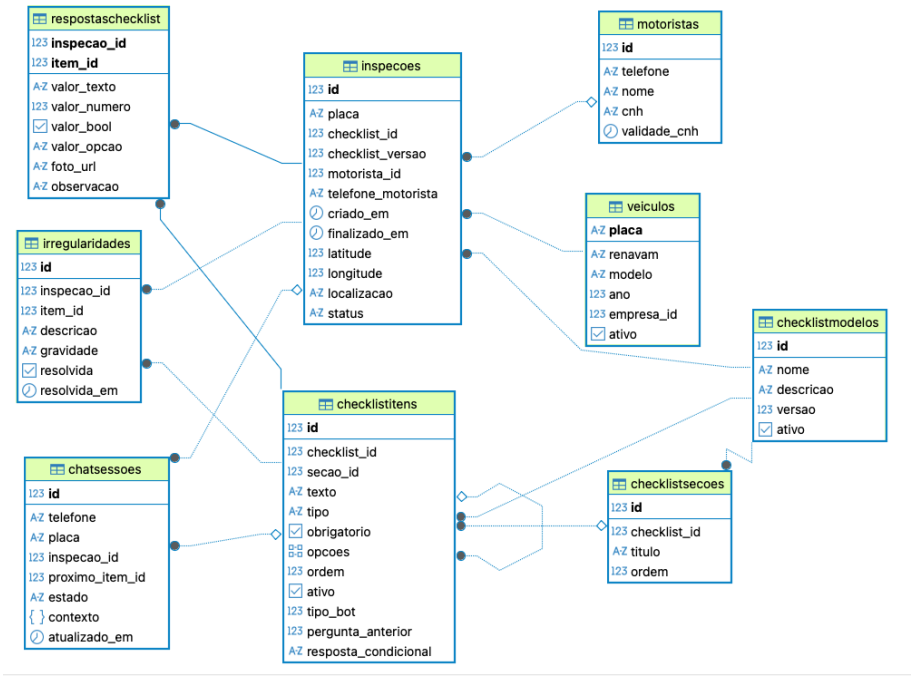
Figura 2 – Captura de tela do aplicativo WhatsApp.



Fonte: Pesquisa direta, 2025.

O armazenamento e a estrutura lógica dos dados coletados são realizados por meio de um banco de dados relacional PostgreSQL, cujo modelo é apresentado na Figura 3. O diagrama exibe as principais tabelas envolvidas no processo de vistoria: Motoristas (dados pessoais e CNH); Veículos (identificação, modelo, ano, status); ChecklistModelos, ChecklistItens e ChecklistSecoes (estrutura das perguntas e seções de vistoria); Inspecoes (dados das vistorias realizadas, incluindo coordenadas GPS e status); RespostasChecklist (armazenamento das respostas individuais de cada item); e Irregularidades (controle de não conformidades identificadas). Essa estrutura relacional garante integridade referencial, escalabilidade e rastreabilidade, permitindo que cada resposta no WhatsApp seja vinculada diretamente à vistoria correspondente.

Figura 3 – Esquema entidade-relacionamento do Banco de Dados.



Fonte: Pesquisa direta, 2025.

Além do armazenamento e análise dos dados, foi desenvolvida uma API REST em Python, utilizando o framework FastAPI, responsável por fornecer os dados do sistema ao painel web administrativo. Essa API permite consultas por placa, data, checklist e status, além de gerar resumos automáticos e relatórios de vistoria. A Tabela 4 ilustra parte do código da API que fornece os dados de resumo por placa:

Tabela 4 – Exemplo de endpoint da API desenvolvida em FastAPI.

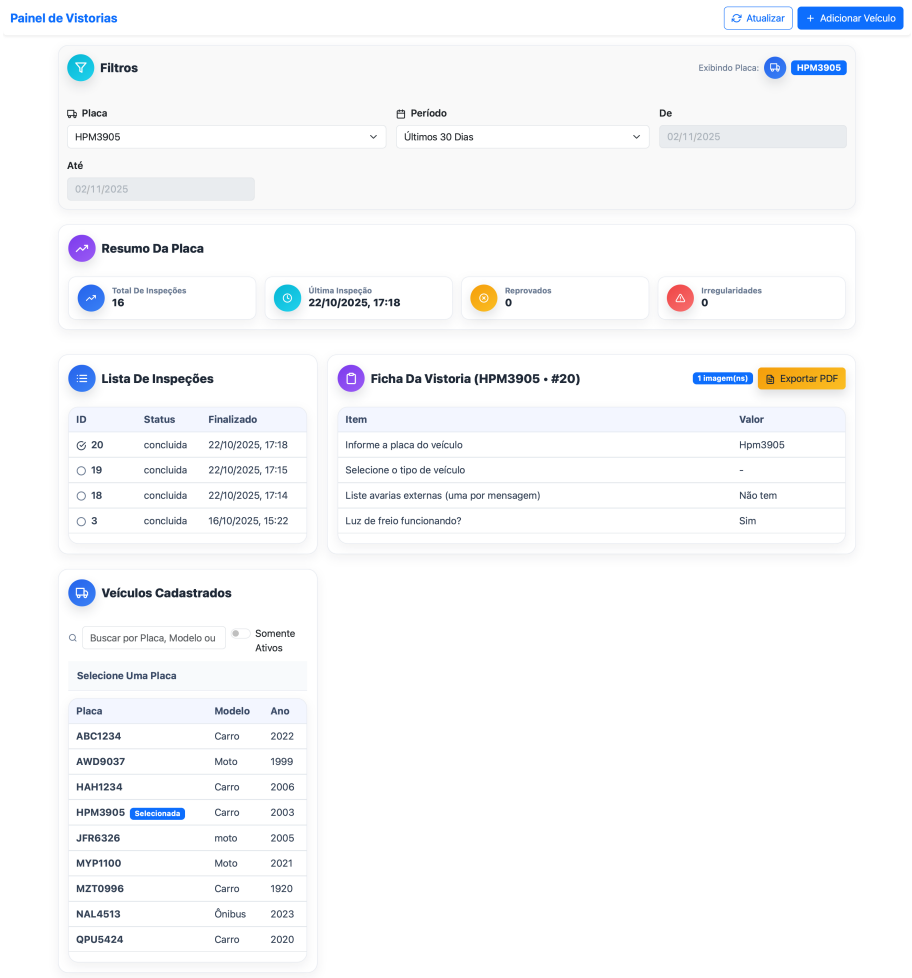
Trecho de código em Python:

```
@app.get("/placa/{placa}/resumo")
def resumo_por_placa(placa: str):
"""
Retorna um resumo agregado por placa:
- total_inspecoes
- ultima_inspecao
- itens_reprovados
- irregularidades_abertas
"""
p = _sanitize_placa(placa)
sql = """
SELECT v.placa, COUNT(i.id) AS total_inspecoes,
MAX(i.finalizado_em) AS ultima_inspecao
FROM Veiculos v
LEFT JOIN Inspecoes i ON i.placa = v.placa
WHERE v.placa = %s
GROUP BY v.placa;
"""
return _exec(sql, (p,))
```

Fonte: Pesquisa direta, 2025.

O painel web, desenvolvido com **React**, consome essa API e exibe visualmente as informações de cada veículo. A Figura 4 mostra a interface completa do painel de vistorias, na qual o administrador pode consultar o histórico por placa, visualizar o **resumo da vistoria (no exemplo, da placa HPM3905)**, acessar o checklist com todas as respostas enviadas pelo bot e **exportar o relatório em formato PDF**. O sistema também apresenta métricas consolidadas, como o total de inspeções, a data da última vistoria e indicadores de irregularidades.

Figura 4 – Captura de tela do website.



Fonte: Pesquisa direta, 2025.

Com a integração entre o chatbot, o banco de dados e o painel de controle, o sistema completo demonstra a viabilidade e a aplicabilidade do template proposto. A infraestrutura mostrou-se robusta, reutilizável e escalável, atendendo aos requisitos de modularidade e independência entre camadas. A correspondência direta entre as mensagens registradas no WhatsApp (Figura 2), os dados gravados no banco (Figura 3) e os resultados exibidos no painel (Figura 4) evidencia o funcionamento completo e validado da arquitetura desenvolvida.

Vale ressaltar que, no momento em que foi documentado este trabalho, ainda não haviam sido desenvolvidos métodos de autenticação de usuário, o que se faz necessário neste tipo de aplicação para garantir confidencialidade e privacidade dos dados. Essa aplicação prática consolida o template como uma ferramenta open-source aplicável em ambientes corporativos, reduzindo custos de desenvolvimento e acelerando a implantação de chatbots integrados à WhatsApp Cloud API.

5 Conclusões e considerações finais

Este trabalho teve como objetivo propor e implementar um template estrutural open source para o desenvolvimento de chatbots conversacionais integrados à WhatsApp Cloud API. Ao longo do texto, mostrou-se que é viável organizar essa solução em camadas bem definidas, separando a infraestrutura de comunicação da lógica de negócio. Com isso, o desenvolvedor deixa de se preocupar com detalhes repetitivos de integração e pode concentrar esforços naquilo que o chatbot precisa, de fato, realizar em cada contexto.

Na prática, a implementação evidenciou que o template é capaz de sustentar um fluxo completo: o usuário interage com o bot pelo WhatsApp, as mensagens passam pela API da Meta, são tratadas em um servidor Node.js, registradas em um banco PostgreSQL e, posteriormente, exibidas em um painel web construído em React e abastecido por uma API. O caso de uso de vistoria veicular permitiu demonstrar esse percurso de ponta a ponta, desde a coleta das respostas até a geração de relatórios e a visualização de históricos de inspeção.

Os testes de validação indicaram que a estrutura proposta é estável e previsível, apresentando bom desempenho e comportamento consistente mesmo diante de múltiplas requisições simultâneas. A divisão em módulos — servidor, controlador de mensagens, lógica de negócio, utilitários e banco de dados — também se mostrou vantajosa para a manutenção e evolução do código. Ainda assim, ficaram evidentes alguns pontos de melhoria, como a ausência de mecanismos completos de autenticação de usuários e a necessidade de uma avaliação mais aprofundada da experiência de uso do chatbot em diferentes cenários.

Como trabalhos futuros, espera-se que esse template sirva de base para que outros desenvolvedores possam reutilizar o código, adaptar a solução às suas próprias demandas e aprimorar o desempenho do bot, especialmente em termos de segurança. A partir dessa infraestrutura já consolidada, é possível incluir autenticação, fortalecer o tratamento de dados sensíveis, adicionar camadas adicionais de logs e monitoramento, integrar modelos de inteligência artificial para processar a intenção do usuário e adequar a lógica conversacional a outros domínios além de vistorias. A perspectiva futura é de que o projeto evolua como um ponto de partida flexível, sobre o qual cada desenvolvedor possa construir seus próprios chatbots, aproveitando a o código fonte base disponibilidado neste trabalho.

Referências

ALVES, W. P. *Projetos de sistemas Web: Conceitos, estruturas, criação de banco de dados e ferramentas de desenvolvimento*. 1. ed. São Paulo: Editora Érica, 2014.

AQUILES, A. *Controlando Versões com Git e GitHub*. São Paulo: Casa do Código, 2014. ISBN 8566250532.

AUDY, G. K. d. A. e. A. C. J. L. N. *Fundamentos de Sistemas de Informação*. 1. ed. Porto Alegre, RS: Bookman, 2005.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 4. ed. [S.l.]: Addison-Wesley, 2021.

BEZERRA, E. *Princípios de Análise e Projeto de Sistemas com UML*. Rio de Janeiro: GEN LTC, 2014. ISBN 9788535226263.

CHACON, S.; STRAUB, B. *Pro Git*. 2. ed. [S.l.]: Apress, 2014. ISBN 9781484200773.

CRUZ, L. T.; ALENCAR, A. J.; SCHMITZ, E. A. *Assistentes Virtuais Inteligentes e Chatbots: Um guia prático e teórico sobre como criar experiências e recordações encantadoras para os clientes da sua empresa*. Rio de Janeiro: Brasport, 2019.

ERL, T.; MONROY, E. B. *Computação em Nuvem: Conceitos, Tecnologia, Segurança e Arquitetura*. Porto Alegre: Bookman, 2024. ISBN 8582606583.

FILHO, W. de P. P. *Engenharia de Software: Fundamentos, Métodos e Padrões*. Rio de Janeiro: LTC, 2009. ISBN 9788521616504.

FOWLER, M. *Padrões de Arquitetura de Aplicações Corporativas*. Porto Alegre: Bookman, 2018. ISBN 978-8577800643.

FRIZZARIN, P. K. L. P.; FRIZZARIN, F. B. *Chatbots para Telegram: programe seus primeiros bots usando Python*. São Paulo: Casa do Código, 2023.

GONçALEZ, F. F. *Chatbot para atendimento automatizado*. Dissertação (Dissertação de Mestrado) — Universidade Fernando Pessoa, Porto, 2020.

GUESSER, A. H. *Software Livre & Controvérsias Tecnocientíficas: Uma análise sociotécnica no Brasil e em Portugal*. 1. ed. Curitiba: [s.n.], 2006. ISBN 8536212330.

LARMAN, C. *Utilizando UML e Padrões: Uma Introdução à Análise e ao Projeto Orientados a Objetos e ao Desenvolvimento Iterativo*. Porto Alegre: Bookman, 2007. ISBN 8560031529.

LAUDON, K. C. L. e J. P. *Sistemas de Informação Gerenciais: administrando a empresa digital*. 17. ed. Porto Alegre, RS: Bookman, 2022.

MARTIN, R. C. *Código Limpo: Habilidades Práticas do Agile Software*. Rio de Janeiro: Alta Books, 2020. ISBN 978-8550811482.

M.STAIR, R.; W.REYNOLDS, G. *Princípios de Sistemas de Informação*. 3. ed. São Paulo: Cengage Learning, 2015. ISBN 8522118620.

O'BRIEN, J. A.; MARAKAS, G. M. *Administração de Sistemas de Informação*. 15. ed. São Paulo: McGraw-Hill, 2013. ISBN 8580551102.

POTTER, E. T. e R. E. *Introdução a Sistemas de Informação: Uma Abordagem Gerencial*. Rio de Janeiro: Campus, 2007. ISBN 9788535222067.

PRADO, E. P. V.; SOUZA, C. A. de. *Fundamentos de Sistemas de Informação*. Rio de Janeiro: LTC, 2014. ISBN 8535274359.

PRESSMAN, R. S. *Engenharia de Software: Uma Abordagem Profissional*. 7. ed. Porto Alegre, RS: AMGH Editora, 2011.

RAJ, S. *Construindo Chatbots Com Python*. São Paulo: Novatec, 2019.

RAYMOND, E. S. *A Catedral e o Bazar*. [S.l.: s.n.], 1998.

REZENDE, D. A. *Planejamento Estratégico Publico ou Privado*. 1. ed. São Paulo,SP: Atlas, 2011.

RICHARDS, M.; FORD, N. *Fundamentos da Arquitetura de Software: Uma Abordagem de Engenharia*. Rio de Janeiro: Alta Books, 2024.

RICHARDSON, L.; AMUNDSEN, M.; RUBY, S. *RESTful Web APIs: Services for a Changing World*. 1. ed. Sebastopol: O'Reilly Media, 2013.

SANTANA, E. F. Z. *Caixa de ferramentas da Arquitetura de Software: Como tornar suas aplicações mais escaláveis, confiáveis e seguras*. [S.l.]: Casa do Código, 2024.

SILVEIRA, S. A. da. *Software livre: a luta pela liberdade do conhecimento*. São Paulo: Fundação Perseu Abramo, 2004. ISBN 8576430037.

SILVERMAN, R. E. *Git. Guia Prático*. São Paulo: Novatec Editora, 2013. ISBN 8575223798.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo: Pearson Education, 2011.

SUBRAMANIAN, B.; JUDE, J. *Introdução ao Software Livre e de Código Aberto*. [S.l.]: Edições Nosso Conhecimento, 2020. ISBN 6200968675.

TANENBAUM, A.; FEAMSTER, N.; WETHERALL, D. *Redes de Computadores*. 6. ed. Porto Alegre, RS: Bookman, 2021.

TAURION, C. *Software livre: potencialidades e modelos de negócio*. Rio de Janeiro: Brasport, 2004. ISBN 8574521736.

TAURION, C. *Cloud Computing. Computação em Nuvem*. Rio de Janeiro: Brasport, 2009.

VALENTE, M. T. *Engenharia de Software Moderna: princípios e práticas para desenvolvimento de software com produtividade*. Belo Horizonte: [s.n.], 2020.

VELTE, A. T. *Cloud Computing. Computação em Nuvem: uma Abordagem Prática*. Rio de Janeiro: Alta Books, 2012. ISBN 8576085364.

VENAZI, D. et al. *Introdução à engenharia de produção: conceitos e casos práticos*. Rio de Janeiro: LTC, 2016.

VERAS, M. *Computação em nuvem*. 1. ed. Rio de Janeiro: Brasport, 2015.

WAZLAWICK, R. S. *Engenharia de Software: Conceitos e Práticas*. 2. ed. Rio de Janeiro: LTC, 2019.

World Population Review. *WhatsApp Users by Country 2024*. 2024. <https://worldpopulationreview.com/country-rankings/whatsapp-users-by-country>. Acesso em: 30 out. 2025.