PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS

ESCOLA POLITÉCNICA E DE ARTES

GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

CUSTOMIZAÇÃO DO TELEGRAM PARA APLICAÇÕES EM SAÚDE

RAFAEL DE MATOS ABE

GOIÂNIA,

2024

RAFAEL DE MATOS ABE

CUSTOMIZAÇÃO DO TELEGRAM PARA APLICAÇÕES EM SAÚDE

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 de Computação.

Orientador(a): Prof. Dr. Talles Marcelo G. de A. Barbosa

GOIÂNIA,

2024

RAFAEL DE MATOS ABE

CUSTOMIZAÇÃO DO TELEGRAM PARA APLICAÇÕES EM SAÚDE

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 de Computação, em 11/12/2024.

Orientador(a): Prof. Dr. Talles Marcelo de G. De A Barbosa

Prof. Me. Claudio Martins Garcia

Profª. Dra. Maria Emília F. Teixeira

GOIÂNIA,

2024

RESUMO

Este trabalho tem como objetivo apresentar a viabilidade de se praticar telemedicina a partir da customização de uma plataforma de propósito geral. Para isso, foi desenvolvido um *chatbot* para a plataforma de propósito geral Telegram, adotando um caso de uso de teleorientação para gestantes em situações de riscos gestacionais. Além disso, foi desenvolvido uma interface gráfica para o médico, a fim de monitorar as condições apresentadas pela interação dos pacientes com o *chatbot*. Assim, o trabalho mostra conceitos de telessaúde, telemedicina e de plataformas de propósito geral, mostrando vantagens e desafios enfrentados por essa prática, explorando vantagens de implementação e analisando riscos de segurança.

Palavras-Chave: *Chatbot*. Telegram. Telemedicina. Telessaúde.

ABSTRACT

This work aims to demonstrate the feasibility of practicing telemedicine through the customization of a general-purpose platform. For this purpose, a chatbot was developed for the general-purpose platform Telegram, adopting a use case of teleconsultation for pregnant women in situations of gestational risk. Additionally, a graphical interface was developed for physicians to monitor the conditions presented during patient interactions with the chatbot. Thus, this work explores the concepts of telehealth, telemedicine, and general-purpose platforms, highlighting the advantages and challenges of this practice, exploring implementation benefits, and analyzing security risks.

Keywords: *Chatbot. Telegram. Telehealth. Telemedicine.*

LISTA DE FIGURAS

LISTA DE SIGLAS

API	*Application Programming Interface*, Interface de Programação de Aplicação
CFM	Conselho Federal de Medicina
CSS	*Cascading Style Sheets;* Folhas de Estilo em Cascata
CSRF	*Cross-Site Request Forgery*; Falsificação de requisição entre sites
HTML	*HyperText Markup Language*; Linguagem de Marcação de Hipertexto
HTTP	*Hypertext Transfer Protocol;* Protocolo de Transferência de Hipertexto
HTTPS	*HyperText Transfer Protocol Secure*; Protocolo Seguro de Transferência de Hipertexto
IA	Inteligência Artificial
IDE	*Integrated Development Environment*; Ambiente de Desenvolvimento Integrado
LGPD	Lei Geral de Proteção de Dados
NLP	*Natural Language Processing*; Processamento de Linguagem Natural
OMS	Organização Mundial da Saúde
SQL	Structured Query Language; Linguagem de Consulta Estruturada
SO	Sistema Operacional
XSS	*Cross-Site Scripting*; Script entre sites

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, a telessaúde tem se consolidado como uma solução crucial para expandir o acesso aos serviços de saúde, especialmente, em áreas remotas e para populações com acesso limitado a recursos médicos. Este campo abrange uma gama ampla de atividades, como, educação em saúde, gestão de saúde pública e telemedicina, utilizando tecnologias digitais para conectar pacientes e profissionais de saúde de forma remota, por meio da Internet. Neste contexto, a telemedicina desempenha um papel central, possibilitando a prestação de cuidados de saúde a distância, com possibilidades de diagnóstico, tratamento e prevenção de doenças por meio da transmissão de dados biomédicos, incluindo dados de sensores e imagens. Com a pandemia de COVID19, a aceitação e adesão ao uso da telemedicina cresceram significativamente, proporcionando a redução de custos para pacientes e profissionais de saúde. (Stoltzfus *et al.*, 2023).

A telemedicina é categorizada pelo Conselho Federal de Medicina (CFM) em diferentes modalidades, cada uma atendendo às necessidades específicas de cada tipo de serviço. Entre essas categorias estão a teleconsulta, para consultas médicas não presenciais mediadas por tecnologias digitais; A teleinterconsulta, que envolve a troca de informações e opiniões entre médicos;O telediagnóstico, que possibilita a análise e emissão de laudos a distância; a telecirurgia, que utiliza equipamentos robóticos para realizar procedimentos cirúrgicos remotamente; O telemonitoramento, destinado ao acompanhamento de parâmetros de saúde de pacientes por meio de dispositivos tecnológicos; a teletriagem, para avaliação inicial e encaminhamento do paciente; e a teleconsultoria, para esclarecer procedimentos administrativos e ações de saúde. Essas categorias evidenciam a amplitude da telemedicina e a capacidade de adaptar-se a diferentes demandas, promovendo uma abordagem diversificada e integrada para melhorar o acesso aos cuidados de saúde entre a população (Conselho Federal de Medicina, 2022).

Entretanto, ao mesmo tempo que a telemedicina aumenta a acessibilidade, a implementação dessas soluções enfrenta desafios operacionais e logísticas significativas. Operacionalmente, a falta de infraestrutura robusta, especialmente em áreas rurais, a divisão digital e a aceitação com as novas tecnologias, dificultam a adoção de sistemas complexos de telemedicina (Raj e Srikanth, 2021). Adicionalmente, questões como o alto custo de desenvolvimento, manutenção de

sistemas dedicados e a necessidade de treinamento técnico adequado para os usuários, pacientes e profissionais da saúde, são desafios recorrentes. Logisticamente, o acesso físico limitado a centros médicos, a escalabilidade dos serviços em regiões com alta demanda e a gestão de dados sensíveis em conformidade com regulamentos como a Lei Geral de Proteção de Dados (LGPD) (Conselho Federal de Medicina, 2022) tornam a operacionalização ainda mais complexa.

Assim, para viabilizar os serviços de telemedicina, as plataformas de propósito geral se destacam como uma solução flexível e adaptável. Essas plataformas, como sistemas operacionais, serviços em nuvem e plataformas de comunicação, a exemplo do WhatsApp e do Telegram, oferecem uma infraestrutura multifuncional que suporta uma ampla variedade de aplicações. A flexibilidade dessas plataformas possibilita que sejam customizadas para atender demandas específicas da telessaúde, possibilitando a integração de diferentes serviços e a adaptação às necessidades locais de saúde (Nardis *et al.*, 2022). Assim, com o uso de plataformas de propósito geral, a telessaúde pode se tornar mais acessível, podendo ser uma opção viável para o atendimento remoto.

Além dessas vantagens funcionais, essas plataformas ajudam a superar barreiras significativas no campo da saúde. Por exemplo, em áreas onde pacientes enfrentam dificuldades para acessar fisicamente os serviços de saúde, essas plataformas possibilitam monitoramento remoto e comunicação direta com profissionais (Raj e Srikanth, 2021). Em locais com alta demanda e recursos médicos limitados, elas possibilitam a escalabilidade do atendimento, ampliando o alcance para mais pacientes de forma simultânea (Nascimento, 2018). Também otimizam a troca de informações entre pacientes e médicos, tornando os processos mais ágeis e eficientes (Gunawan *et al*., 2021).

Entretanto, o uso dessas ferramentas também levanta questões éticas e de privacidade, tanto para os pacientes, quanto para os médicos que podem vir a ter perda de privacidade (Leão *et al.*, 2018). A prática de telemedicina no Brasil é regulamentada pelo Conselho Federal de Medicina (CFM) e exige que os médicos sigam diretrizes éticas rigorosas, incluindo o consentimento informado, a proteção da privacidade e a confidencialidade dos dados do paciente. Com a Lei Geral de Proteção de Dados (LGPD), o tratamento de informações pessoais e de saúde exige uma série

de precauções, incluindo o uso de dispositivos exclusivos, a máxima segurança possível aplicada e a restrição do uso de aplicativos não dedicados para consultas e diagnósticos formais. Aplicativos de mensagens, como o WhatsApp ou Telegram, devem ser reservados para monitoramento, orientações e esclarecimentos emergenciais, ou seja, telemonitoramento. (Conselho Federal de Medicina, 2022).

Por fim, a incorporação de *chatbots*, agentes artificiais inteligentes projetados para simular interações humanas, apresenta-se como uma ferramenta inovadora no campo da telessaúde. Esses sistemas são capazes de interpretar e responder às mensagens de forma personalizada a partir de um conjunto pré-definido de regras e fluxos lógicos. Além disso, a capacidade de interagir com pacientes de maneira contínua e automatizada possibilita um suporte mais ágil e eficiente, promovendo o monitoramento remoto de sintomas em tempo real e facilitando intervenções proativas por parte dos profissionais de saúde (Lee et al., 2021).

Dessa forma, os *chatbots* têm demonstrado potencial para reduzir a carga de trabalho de profissionais de saúde ao lidar com questões recorrentes e simples, liberando esses profissionais para casos mais complexos (Gunawan *et al*., 2021). Quando integrados a plataformas de propósito geral, como o Telegram, esses *chatbots* se beneficiam de uma infraestrutura flexível e acessível, que possibilita a implementação sem a necessidade de criar sistemas dedicados do zero. A combinação desses fatores torna viável o desenvolvimento de soluções tecnológicas acessíveis para o telemonitoramento, destacando a complementaridade entre os *chatbots* e as plataformas de propósito geral na superação de barreiras operacionais e logísticas em telessaúde.

Logo, justifica-se estudar esse tema para explorar a viabilidade e os riscos da customização de plataformas de propósito geral para operações em telessaúde, com foco em telemedicina. Dessa forma, abordando vantagens e desafios envolvidos, as questões éticas e regulatórias, e o potencial dos *chatbots* como ferramentas para essa prática, buscando ampliar o acesso à saúde de forma acessível, segura e ética, especialmente em contextos de alta demanda e recursos limitados.

Diante do contexto, este projeto tem como desafio investigar possibilidades para o uso de plataformas de propósito geral, comumente utilizadas em redes sociais, combinadas com tecnologias que possibilitam a manipulação automática de dados

obtidos de respostas dos usuários (em inglês, *chatbots*), para aplicações de telessaúde. Portanto, no domínio das possibilidades, pretende-se responder **à existência de uma solução tecnológica que satisfaça às restrições para aplicações de telessaúde mediadas por customizações em plataformas de propósito geral, como, por exemplo, o Telegram.**

1.1 OBJETIVOS

1.1.1 Objetivo Geral

- Avaliar o uso de uma plataforma de propósito geral customizada com um *chatbot* para realização de telemedicina, com foco na plataforma Telegram.

1.1.2 Objetivos específicos

- Apresentar conceitos de telessaúde, telemedicina e plataformas de propósito geral;
- Desenvolver um *chatbot* para o Telegram, com foco em telemonitoramento;
- Desenvolver uma interface gráfica para que o médico, exibindo condições dos pacientes a partir da interação com o *chatbot*;
- Avaliar riscos de confidencialidade e privacidade de dados.

2 REFERENCIAL TEÓRICO

Este capítulo traz os principais conceitos de Telessaúde, Telemedicina e Plataformas de Propósito Geral e a visão geral do Conselho Federal de Medicina (CFM) sobre princípios éticos e confidencialidade dos dados. Além disso, traz também trabalhos relacionados ao tema.

2.1 Telessaúde

A telessaúde refere-se ao uso de tecnologias digitais para oferecer uma ampla gama de serviços relacionados à saúde abrangendo tanto a telemedicina quanto iniciativas de educação em saúde e gestão de saúde pública (Stoltzfus *et al*., 2023).

Definida pela Organização Mundial da Saúde (OMS) como “a entrega de serviços de saúde a distância por todos os profissionais de saúde utilizando tecnologias de informação e comunicação para a troca de informações válidas para diagnóstico, tratamento e prevenção de doenças e lesões”, a telessaúde emerge como uma solução poderosa para ampliar o acesso aos serviços de saúde. Ela visa conectar pacientes e profissionais de saúde superando barreiras geográficas, reduzindo custos e promovendo a continuidade do cuidado.

Além disso, a telessaúde desempenha um papel crucial em áreas de acesso limitado, promovendo uma conexão constante entre populações isoladas e a infraestrutura de saúde, possibilitando, assim, uma melhoria substancial na qualidade de vida desses indivíduos.

2.2 Telemedicina

A telemedicina é definida pela Organização Mundial da Saúde (OMS) como a prestação de serviços de saúde a distância por profissionais de saúde, utilizando tecnologias de comunicação para a troca de informações relevantes para o diagnóstico, tratamento e prevenção de doenças e lesões. Desde a primeira aplicação nos anos 1950, com a transmissão de radiografias, a telemedicina evoluiu significativamente e se expandiu para diversas especialidades, incluindo radiologia, dermatologia e psiquiatria. Essa prática tem se mostrado especialmente útil para

conectar profissionais de saúde e pacientes em regiões remotas, possibilitando consultas, diagnósticos e orientações médicas sem a necessidade de deslocamento (Stoltzfus *et al*., 2023).

Atualmente, a telemedicina representa um avanço essencial na acessibilidade ao atendimento médico, possibilitando consultas por vídeo, monitoramento remoto de saúde e até mesmo triagem de emergências. Ao reduzir os tempos de viagem e os custos associados ao deslocamento, a telemedicina facilita o acesso a serviços médicos em regiões isoladas e contribui para o controle de doenças transmissíveis ao minimizar o contato físico entre pacientes e profissionais de saúde (Stoltzfus *et al*., 2023).

2.2.1 Diferença entre os tipos de telemedicina

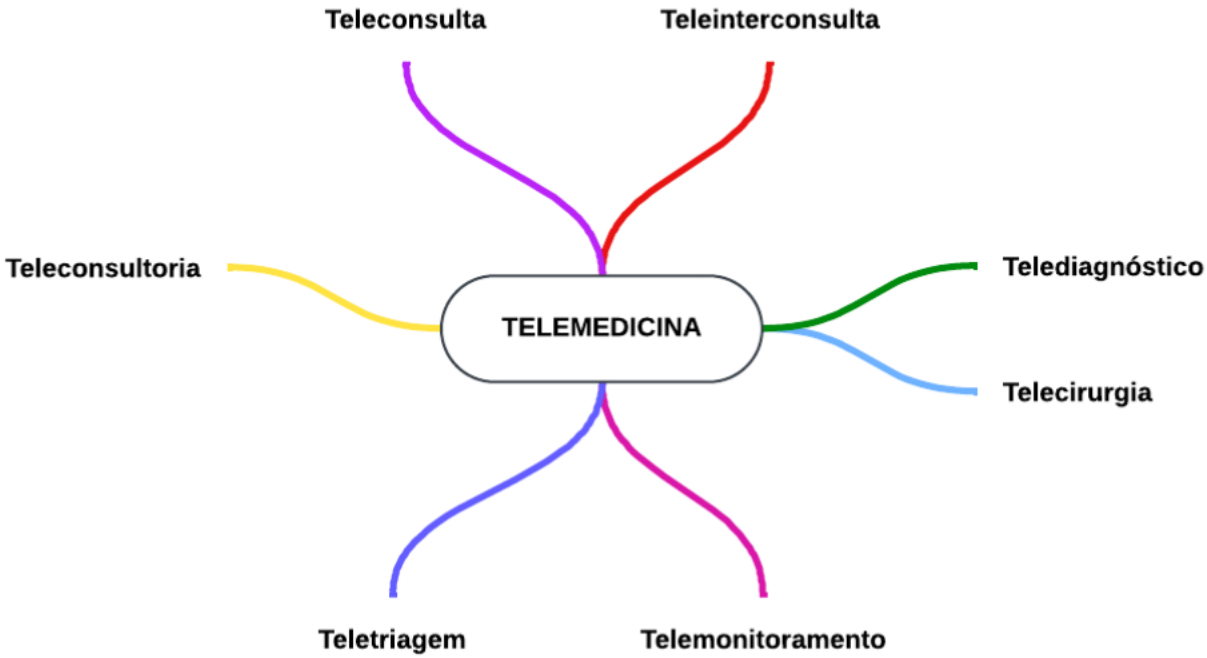
A telemedicina, conforme descrita pela Organização Mundial da Saúde (OMS), consiste na oferta de serviços de saúde em que a distância desempenha um papel crucial. Utilizando tecnologias de informação e comunicação, esses serviços possibilitam a troca de informações médicas para o diagnóstico, tratamento e prevenção de doenças e lesões, sempre visando a promoção da saúde de indivíduos e comunidades. Desde a introdução na década de 1950, a telemedicina tem evoluído significativamente, ampliando as modalidades para atender diferentes necessidades médicas e logísticas, especialmente em contextos onde o acesso presencial a serviços de saúde é limitado.

No Brasil, a Resolução CFM nº 2.314/2022 formalizou essa categorização, destacando os principais tipos de telemedicina utilizados na prática médica. A Figura 1 ilustra todos os tipos de telemedicina e definição detalhada de cada um delas são:

1. Teleconsulta: A teleconsulta consiste em consultas médicas realizadas a distância, onde médicos e pacientes interagem por videoconferências ou mensagens. Esta modalidade possibilita o diagnóstico e acompanhamento clínico remoto, sendo especialmente útil em áreas com escassez de especialistas. Além disso, possibilita a obtenção de segundas opiniões e a integração de exames para uma avaliação mais abrangente (Cadforio *et al*., 2020; Jeong *et al*., 2017).

2. Telemonitoramento (ou Televigilância): O telemonitoramento envolve a coleta e transmissão contínua de dados de saúde dos pacientes, possibilitando a observação remota de sinais vitais. Amplamente usado em doenças crônicas, como diabetes e hipertensão, reduz a necessidade de consultas presenciais, possibilitando respostas ágeis a alterações clínicas e contribuindo para a redução de hospitalizações (Cadforio *et al*., 2020).
3. Teleinterconsulta: A teleinterconsulta é a troca de informações e opiniões entre médicos para auxílio diagnóstico ou terapêutico. Essa modalidade pode ocorrer com ou sem a presença do paciente e é essencial para promover a colaboração interdisciplinar e a capacitação entre profissionais de saúde.
4. Telediagnóstico: O telediagnóstico refere-se à emissão de laudos e pareceres médicos a partir da análise de dados e imagens transmitidos a distância. Esta prática depende de especialistas com registro em áreas específicas e é amplamente usada em radiologia, cardiologia e dermatologia.
5. Telecirurgia: A telecirurgia consiste na realização de procedimentos cirúrgicos à distância, mediada por tecnologias robóticas interativas e seguras. É um avanço significativo em cirurgias de alta precisão, possibilitando que especialistas realizem procedimentos em locais remotos.
6. Teletriagem: A teletriagem é a avaliação remota de sintomas realizada por médicos para determinar o tipo de atendimento necessário. Trata-se de uma impressão diagnóstica inicial, não substituindo uma consulta médica, mas, orientando o paciente para o cuidado adequado.
7. Teleconsultoria: A teleconsultoria é um ato de consultoria entre médicos e outros profissionais de saúde, focado em orientações sobre ações administrativas e clínicas. É frequentemente usada para apoiar a organização e gestão de processos em saúde pública e privada.

Figura 1: Telemedicina e seus tipos



Fonte: Autoria própria

2.2.2 Impactos da Telemedicina

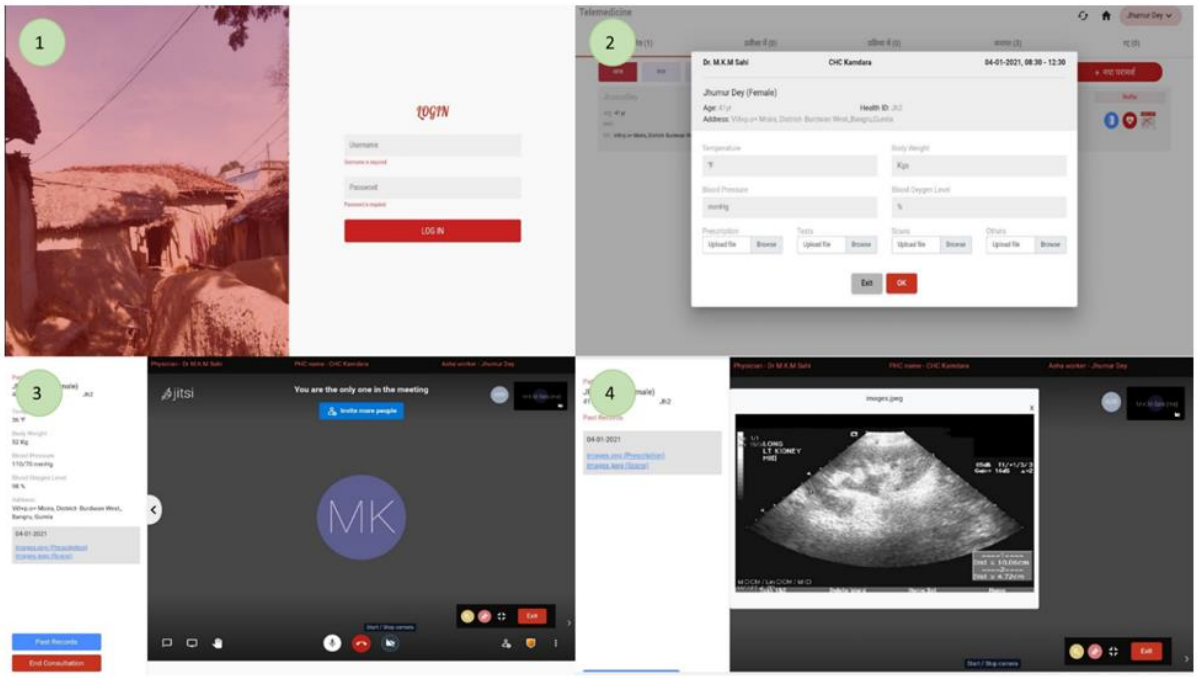
Em 2020, no contexto da pandemia de COVID-19, para melhorar o acesso à saúde em áreas rurais da Índia, Raj e Srikanth (2021) realizaram um experimento rural com um modelo de telemedicina assistida.

Este experimento notou impactos significativos na acessibilidade aos serviços de saúde, especialmente em regiões onde a distância e a infraestrutura deficiente dificultavam o atendimento. A Figura 2 mostra a metodologia aplicada, que envolveu a criação de uma plataforma customizada que possibilita a consulta remota de profissionais médicos.

Entre as principais vantagens, destacam-se a melhoria no acesso à saúde, redução de custos e tempo com deslocamentos. Por outro lado, notou-se desafios, como o receio inicial dos moradores em adotar uma nova tecnologia e a necessidade de treinamento específico para os profissionais e agentes da saúde, e para a população local.

Em conclusão, o estudo de Raj e Srikanth (2021) demonstra que a telemedicina assistida pode beneficiar significativamente populações rurais, mas, exige adaptações culturais e logísticas para garantir a aceitação e sustentabilidade do modelo.

Figura 2: Interface do Protótipo de Telemedicina Assistida



Fonte: Raj; Srikanth, 2021

Outra abordagem para a prática da telemedicina, utilizando uma plataforma já existente, foi explorada no estudo de Leão *et al.* (2018), realizado no contexto de um ambulatório de pediatria e obstetrícia na Fundação Santa Casa de Misericórdia do Pará, Brasil. O estudo investigou o uso do WhatsApp como ferramenta de comunicação médico-paciente, revelando tanto impactos positivos quanto limitações significativas.

A metodologia adotada neste estudo consistiu na aplicação de questionários qualitativos a oito médicos, com foco nas experiências e percepções sobre o uso do aplicativo para interação com pacientes. Entre as vantagens relatadas na Figura 3, destacaram-se a agilidade no esclarecimento de dúvidas, o suporte contínuo à saúde dos pacientes e a redução de visitas presenciais desnecessárias, o que contribuiu para otimizar o atendimento.

Figura 3: Vantagens do uso do WhatsApp na relação médico-paciente

Vantagens	n	%
Manter boa relação médico-paciente	2	25
Acompanhar o tratamento a distância	1	12,5
Evitar idas desnecessárias ao médico que já acompanha o paciente	1	12,5
Orientar e tirar dúvidas do paciente	3	37,5
Enviar resultados de exames	2	25
Informar ao médico sinais e sintomas novos que apareçam após a consulta	1	12,5
Emergências	3	37,5
Comunicação rápida	1	12,5

Fonte modificado de: Leão *et al*., 2018

No entanto, a Figura 4 mostra que o uso do WhatsApp também apresentou desafios. Médicos relataram a falta de limites por parte de alguns pacientes, o que resultava na trivialização do serviço médico. Além disso, 62,5% dos médicos expressaram desconforto ético, relacionado à ausência de regulamentação específica e preocupações com a privacidade dos dados compartilhados na plataforma.

Figura 4: Desvantagens do uso do WhatsApp na relação médico-paciente

Desvantagens	n	%
Falta de limites dos pacientes	3	37,5
Paciente não quer mais ir à consulta	2	25
Perda da privacidade do médico	1	12,5
Não tem amparo legal	1	12,5
Banaliza o serviço médico	2	25
Nenhuma	1	12,5

Fonte modificado de: Leão *et al*., 2018

O estudo Leão *et al.* (2018) concluiu que, embora o WhatsApp ofereça benefícios práticos, como a acessibilidade e a rapidez na comunicação, há uma necessidade urgente de regulamentações claras e melhores práticas que garantam a confidencialidade e a integridade da relação médico-paciente.

2.3 Plataformas de Propósito Geral

Plataformas de propósito geral são sistemas projetados para atender a uma ampla gama de aplicações e funcionalidades em diferentes contextos, oferecendo flexibilidade e adaptabilidade como principais características. Essas plataformas fornecem uma infraestrutura robusta capaz de integrar diversos serviços, como comunicação, processamento e visualização de dados, integração com sistemas externos e funcionalidades de segurança (De Nardis *et al*., 2022).

Segundo De Nardis *et al*. (2022), a escolha entre plataformas pode variar entre opções de código aberto ou fechado, dependendo das necessidades específicas, como controle do código-fonte, custos envolvidos e objetivos do projeto. No contexto acadêmico e industrial, plataformas de propósito geral são frequentemente utilizadas para facilitar o desenvolvimento de soluções tecnológicas complexas, aproveitando recursos já existentes, sem a necessidade de construir sistemas do zero.

Assim, plataformas de propósito geral permitem acelerar o desenvolvimento de projetos, oferecendo uma base flexível para customizações que atendam interesses, objetivos e requisitos específicos para a construção de um projeto.

2.3.1 Telegram e Chatbot

O Telegram é uma plataforma de comunicação altamente versátil, destaca-se pela flexibilidade ao possibilitar o desenvolvimento de *chatbots* personalizados por meio Interface de Programação de Aplicação (em inglês, *Application Programming Interface* - API), que suporta uma ampla gama de linguagens de programação e funcionalidades intuitivas para interação com usuários (Dr Raghu *et al*., 2023).

Um *chatbot* é um sistema automatizado projetado para interagir com usuários em linguagem natural, auxiliando em diversas tarefas, como responder perguntas frequentes, coletar informações e oferecer suporte em tempo real. Em um contexto médico, os *chatbots* podem atuar como ferramentas de triagem inicial, auxiliando pacientes a identificar sintomas e sugerindo possíveis diagnósticos com base nas informações fornecidas. Além disso, podem oferecer recomendações personalizadas, como orientações sobre cuidados preventivos ou o encaminhamento para especialistas, baseando-se em algoritmos de inteligência artificial e aprendizado de

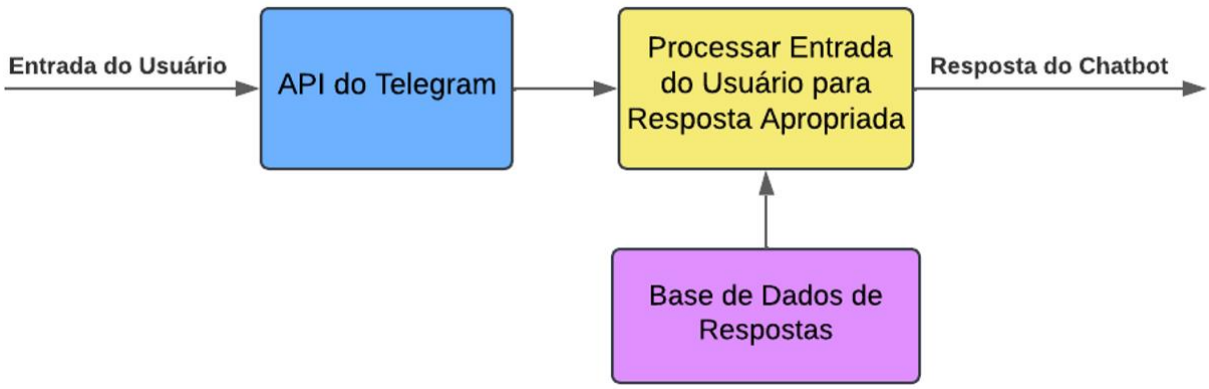
máquina. Integrados a plataformas de propósito geral, como o Telegram, esses sistemas permitem a implementação de soluções de telessaúde que são ao mesmo tempo acessíveis, personalizadas e escaláveis, atendendo a uma ampla gama de necessidades dos usuários e profissionais de saúde (Harshithan *et al.*, 2024).

O estudo de Gunawan *et al.* (2021) utilizou o Telegram para criar um *chatbot* com reconhecimento emocional, projetado para responder a mensagens dos usuários com empatia e de forma personalizada. A metodologia empregada envolveu a combinação de Python, Telegram Bot API e ferramentas de Processamento de Linguagem Natural (em inglês, *Natural Language Processing* - NLP), para implementar a análise de sentimentos e a geração de respostas emocionais (Figura 5).

O sistema foi testado com sucesso e apresentou uma interação fluida. Demonstrou a capacidade de identificar emoções como tristeza e alegria, oferecendo respostas adequadas e empáticas. Apesar do *chatbot* ainda necessitar de refinamentos, os resultados iniciais indicaram a viabilidade do uso do Telegram como plataforma para *chatbots* de saúde mental. Essa solução se mostrou uma alternativa acessível para suporte psicológico.

O estudo de Gunawan *et al.* (2021) concluiu que o Telegram é uma ferramenta poderosa para desenvolvimentos futuros em telessaúde. Ele possibilita a criação de *chatbots* que não apenas interagem, mas, também, reconhecem e respondem às emoções dos usuários.

Figura 5: Processo típico de um chatbot no Telegram



Fonte Modificada de: Gunawan *et al.*, 2021

2.4 Princípios Éticos da LGPD na Saúde

A Lei Geral de Proteção de Dados Pessoais (LGPD), Lei n. 13.709/2018, foi implementada para regular o tratamento de dados pessoais no Brasil, com o objetivo de proteger os direitos de liberdade e privacidade dos indivíduos. Esta regulamentação tem um impacto profundo na área da saúde, onde são tratados dados pessoais sensíveis, como informações médicas, que exigem um tratamento com rigor ético e legal. Na medicina, a LGPD vem reforçar o sigilo profissional já estabelecido pelo Código de Ética Médica (CEM), que protege as informações dos pacientes contra qualquer divulgação não autorizada, garantindo, assim, a confidencialidade dos dados no contexto dos cuidados médicos (MANUAL-LGPD-CRM-DF-2022) (CFM, 2022).

Para isso, a prática de telemedicina e telessaúde deve respeitar os princípios estabelecidos pela LGPD para assegurar que o tratamento dos dados de saúde siga normas éticas e legais. Entre os princípios estão:

Finalidade e Necessidade: Dados pessoais devem ser coletados somente para propósitos específicos e legítimos, limitando-se ao mínimo necessário para a prestação de cuidados.

Transparência e Livre Acesso: Pacientes possuem o direito de acesso a seus dados e devem ser devidamente informados sobre como e por que seus dados estão sendo tratados.

Segurança e Prevenção: É imprescindível que profissionais e instituições de saúde adotem medidas de segurança que protejam os dados contra acessos não autorizados, perdas e vazamentos.

Consentimento Específico: O tratamento de dados sensíveis, especialmente na área da saúde, requer consentimento claro e específico do paciente, destacando as finalidades do uso desses dados (MANUAL-LGPD-CRM-DF-2022) (CFM, 2022).

Além disso, para garantir que a prática da telemedicina e telessaúde siga as regulamentações éticas, algumas regras específicas são destacadas:

Sigilo Médico: A proteção ao prontuário médico e a outros documentos sensíveis é um imperativo reforçado pela LGPD, exigindo que apenas os profissionais diretamente envolvidos no tratamento tenham acesso a esses dados (MANUAL-LGPD-CRM-DF-2022) (CFM, 2022).

Utilização de Plataformas Seguras: No âmbito da telemedicina, é essencial o uso de plataformas digitais que assegurem a segurança e a confidencialidade dos dados. Para orientações informais, o uso de aplicativos como WhatsApp é permitido, mas, a realização de diagnósticos e consultas formais deve ser evitada nessas plataformas para proteger a privacidade dos dados sensíveis. Assim, a Resolução CFM determina que o sistema utilizado para registro e compartilhamento de dados médicos deve atender aos requisitos de segurança, como o Nível de Garantia de Segurança 2 (NGS2) no padrão ICP-Brasil ou outros padrões legalmente aceitos. Essas plataformas devem garantir interoperabilidade e proteção contra acessos não autorizados (CFM, 2022).

Consentimento Informado e Registro de Acesso: Os pacientes devem ser informados sobre o uso e o compartilhamento de seus dados em plataformas digitais e têm o direito de revogar seu consentimento a qualquer momento. Além disso, deve haver registros de todas as interações e acessos aos dados, para fins de auditoria e controle, bem como o registro do consentimento, podendo ser obtido por meio eletrônico, e deve fazer parte do Sistema de Registro Eletrônico de Saúde (SRES) do paciente (MANUAL-LGPD-CRM-DF-2022) (CFM, 2022).

Assim, a aplicação conjunta da LGPD e do Código de Ética Médica exige que os profissionais de saúde cumpram as regulamentações técnicas e assumam um compromisso ético de proteger a privacidade dos pacientes, assegurando que os dados sensíveis sejam tratados com confidencialidade. A Autoridade Nacional de Proteção de Dados (ANPD) e o Conselho Federal de Medicina (CFM) atuam como órgãos reguladores que garantem a conformidade da prática da telemedicina e telessaúde no Brasil com os padrões éticos e legais de proteção de dados (MANUAL-LGPD-CRM-DF-2022) (CFM, 2022).

3 MATERIAIS E MÉTODOS

Este trabalho caracteriza-se como uma pesquisa básica ou fundamental, com foco no aprofundamento de conhecimentos em telemedicina e na customização de plataformas de propósito geral para aplicações em telessaúde.

Quanto à natureza, trata-se de uma pesquisa experimental, que inclui a manipulação de variáveis para avaliar a eficácia de soluções desenvolvidas, como o chatbot integrado ao Telegram e sua interface gráfica (Gil, 2017).

Quanto à forma de abordagem, é uma pesquisa quantitativa, que busca descrever e analisar dados coletados por meio de experimentos, com base em métodos sistemáticos. Além disso, possui caráter descritivo, ao detalhar as funcionalidades do sistema, e explicativo, ao explorar relações de causa e efeito nas implementações realizadas (Fontelles, 2009).

Quanto aos objetivos, trata-se de uma pesquisa exploratória, pois investiga um campo ainda pouco abordado, como a customização de plataformas populares para telemedicina (Wazlawick, 2014).

Quanto aos procedimentos técnicos, esta pesquisa é predominantemente bibliográfica, fundamentada em artigos científicos, livros e outros materiais relevantes. A pesquisa bibliográfica permitiu compreender o estado da arte em telemedicina, plataformas de propósito geral e chatbots, além de embasar as escolhas tecnológicas e metodológicas realizadas (Wazlawick, 2014).

Quanto ao desenvolvimento no tempo, é uma pesquisa transversal, realizada em um período específico, com coleta de dados limitada a esse intervalo de tempo (Fontelles, 2009).

Etapas da Pesquisa:

a) Revisão Bibliográfica: Foi realizada uma análise de materiais já publicados, como livros, artigos científicos e recursos disponíveis em bases de dados como ACM Digital Library, IEEE e PubMed. O objetivo foi identificar conceitos-chave em telemedicina, telemonitoramento e customização de plataformas de propósito geral.

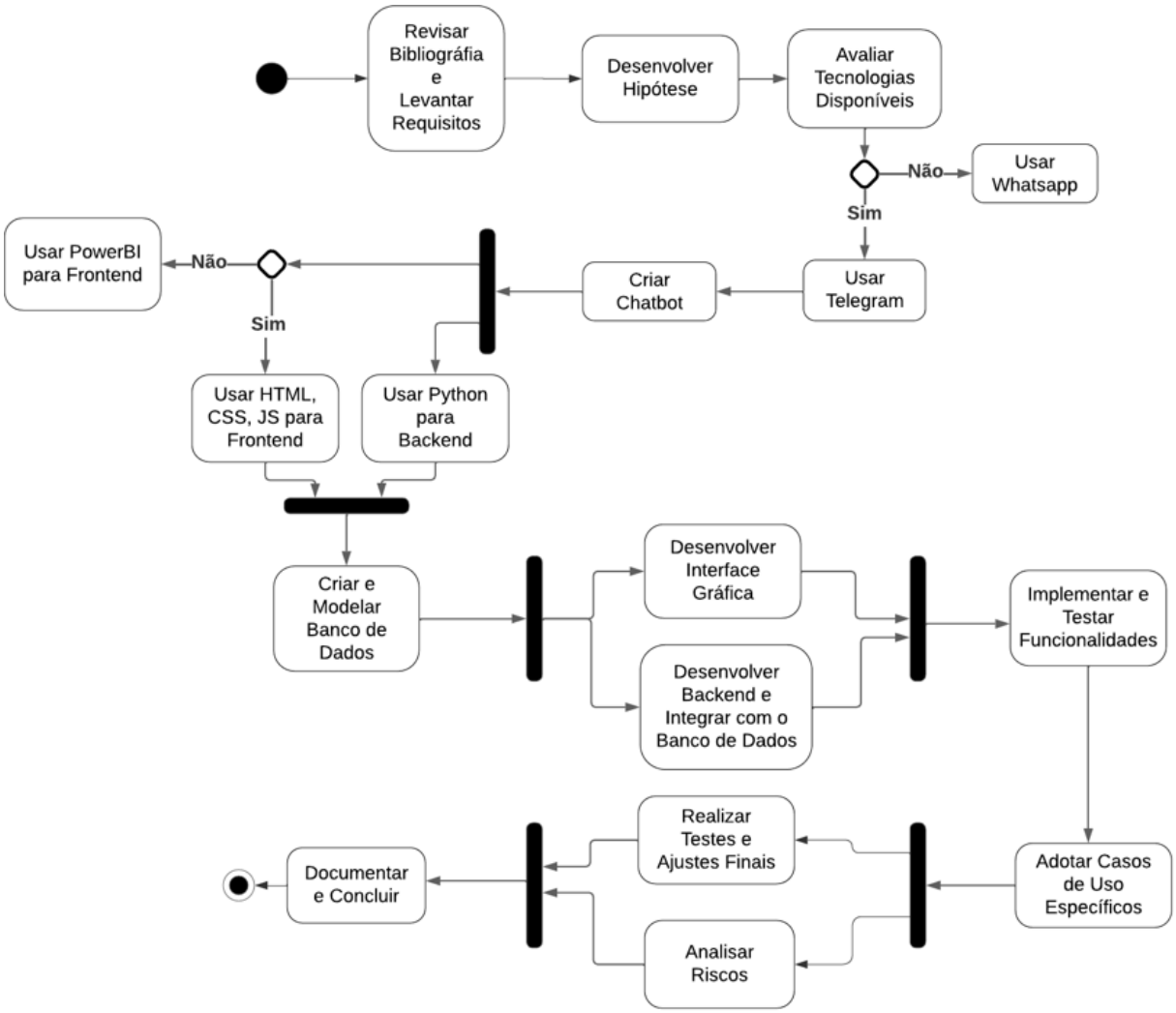
b) Formulação do Problema: Definiu-se a seguinte questão de pesquisa: Como customizações em plataformas de propósito geral, como o Telegram, podem viabilizar uma solução tecnológica que satisfaça às restrições para aplicações de telessaúde?
c) Desenvolvimento Experimental: A pesquisa envolveu a criação de um *chatbot* no Telegram, integrado a uma interface gráfica e a um banco de dados modelado para armazenar interações e mídias dos pacientes. Foram realizados testes em um ambiente local configurado com máquina específica, ferramentas de desenvolvimento (Visual Studio Code e Python) e análise de dados (PgAdmin e DBeaver).
d) Coleta e Análise de Dados: Os dados foram coletados por meio de interações simuladas no sistema, armazenados em um banco SQL, e analisados para avaliar o atendimento aos requisitos do sistema e compará-los com estudos relacionados.
e) Redação do Relatório: As etapas e resultados da pesquisa foram documentados para elaboração deste trabalho, conforme normas da ABNT.
f) Ambiente de Desenvolvimento: O ambiente experimental foi configurado em um notebook com as seguintes especificações:
    a. Hardware: Notebook Acer Aspire 3 A315-41-R2MH, processador AMD Ryzen 5 2500U, 16 GB de RAM.
    b. Software: Sistema Operacional Windows 10, Visual Studio Code versão 1.94, Python versão 3.12.2, PgAdmin 4 e DBeaver para análise de dados.
    c. Simulações: Utilizou-se um ambiente local para replicar condições reais de uso, com protocolos HTTPS e banco de dados relacional em PostgreSQL.

3.1 Processo de Desenvolvimento Adotado

O desenvolvimento do sistema foi baseado em um processo estruturado que priorizou a adaptação às necessidades específicas do projeto. As etapas foram realizadas de forma sequencial, com a aplicação de ajustes e melhorias conforme necessários, possibilitando a evolução do sistema de acordo com os requisitos identificados. Para isso, foi desenvolvido um diagrama de atividades (Figura 6), que

descreve o fluxo de atividades e as principais decisões dentro de um sistema, para ilustrar a sequência lógica das etapas realizadas durante o desenvolvimento do projeto.

Figura 6: Diagrama de Atividades



Fonte: Autoria Própria

O processo adotado seguiu as etapas representadas no diagrama de atividades da Figura 6. Cada uma dessas etapas foi definida e implementada conforme descrito abaixo:

Revisar Bibliografia e Levantar Requisitos foi o primeiro passo. Essa etapa consistiu em pesquisar sobre sistemas de telemedicina e formas de customizar uma plataforma de propósito geral, identificando ferramentas e plataformas adequadas

ao projeto. Também foram analisadas características de sistemas de telemedicina, comparando vantagens e desvantagens de métodos como criar um sistema do zero, usar plataformas de comunicação de forma não customizada ou automatizar orientações médicas por meio de converas entre pacientes e *chatbots*.

Desenvolver Hipótese foi o segundo passo. Com base na revisão bibliográfica e nos requisitos levantados, foi elaborada a hipótese de que um chatbot integrado a uma plataforma de comunicação popular poderia ser viável para telemonitoramento. Essa abordagem visou explorar custos reduzidos de implantação e simplicidade de engajamento, considerando também riscos de confiabilidade e privacidade ao usar uma plataforma existente. A ideia incluiu criar uma interface gráfica para médicos, onde fossem apresentados dados das interações entre pacientes e o chatbot.

Assim, foram analisados requisitos importantes para que o sistema funcionasse como um protótipo para a prática de uma telemedicina acessível. O *chatbot* deveria monitorar situações por meio de perguntas com respostas simples, enquadrando-se na categoria de telemonitoramento. Para isso, foi adotada a metodologia proposta por Lima (2023), que elaborou um fluxo de perguntas para diagnóstico de gestantes em situações de possível risco.

Além disso, o sistema deveria ser capaz de receber mídias, como fotos, áudios ou vídeos. Essa funcionalidade aumentaria a acessibilidade e diversificaria as mensagens enviadas pelos usuários. Dessa forma, mesmo usuários não letrados, mas, capazes de gravar áudios ou enviar fotos e vídeos, poderiam acessar o telemonitoramento. Por fim, era importante desenvolver um método para transcrição de áudio em texto. Isso possibilitaria a análise do conteúdo de áudios e vídeos, bem como a geração de legendas para vídeos.

Avaliar Tecnologias Disponíveis foi a etapa seguinte. Foi avaliada uma plataforma de comunicação popular, com testes iniciais no WhatsApp, mas, a impossibilidade de obter licença para a API levou à escolha do Telegram. Essa plataforma foi selecionada pela eficiência, flexibilidade e suporte para criação de *bots* personalizados via API.

Criar *Chatbot* foi o quarto passo. Utilizando a API BotFather, foi desenvolvido um chatbot baseado em regras para simular conversas humanas e atender às demandas de monitoramento.

Usar Python para *Backend* e Integrar com Banco de Dados incluiu a

implementação do backend responsável por gerenciar o *chatbot*, salvar informações no banco de dados e interagir com a interface gráfica.

Usar Linguagem de Marcação de Hipertexto (em inglês, *HyperText Markup Language* - HTML), Folhas de Estilo em Cascata (em inglês, *Cascading Style Sheets* - CSS) e JavaScript para Frontend começou com a criação de gráficos no *Power BI*, que foram substituídos por uma interface gráfica personalizada, oferecendo maior flexibilidade e controle.

Criar e Modelar Banco de Dados consistiu em estruturar tabelas para armazenar dados de usuários, respostas e mídias. O PostgreSQL foi escolhido pela eficiência e suporte a operações complexas, como consultas avançadas e grandes volumes de dados.

Implementar e Testar Funcionalidades verificou se o fluxo principal do sistema estava correto. Essa etapa incluiu validar que o chatbot coletava informações, armazenava no banco e exibia os dados em gráficos na interface gráfica. Outras funcionalidades, como transcrição de áudio e legendagem de vídeos, também foram implementadas e testadas.

Adotar Casos de Uso Específicos focou na lógica de monitoramento de gestantes em situações de risco deselvolvidas por Lima (2023), com fluxos específicos implementados no chatbot e na interface gráfica.

Realizar Testes e Ajustes Finais garantiu que todas as etapas, desde a interação do paciente até a exibição dos dados na interface gráfica, funcionassem corretamente.

Documentar e Concluir finalizou o processo. Todas as etapas foram detalhadamente registradas, com análises de riscos e resultados que comprovaram a viabilidade do sistema de telemonitoramento, apesar dos desafios relacionados à confiabilidade e privacidade.

3.2 Projeto

Este capítulo apresenta os principais artefatos desenvolvidos durante o projeto, responsáveis por documentar e organizar as funcionalidades e a arquitetura do sistema. Esses artefatos foram fundamentais para garantir clareza e alinhamento entre as etapas de planejamento, modelagem e implementação.

Assim, os artefatos apresentados guiaram o desenvolvimento do sistema e serviram como ferramentas de validação contínua, garantindo que cada etapa do projeto estivesse alinhada ao objetivo de criar uma solução eficiente e facilmente adaptável às necessidades dos usuários

Os diagramas e modelagens que serão apresentados e descritos neste capítulo desempenham um papel essencial na transição do objetivo geral para uma solução prática e funcional. Eles detalham os fluxos de interação, a estrutura de dados e as principais funcionalidades do sistema, assegurando que cada elemento esteja bem definido e alinhado aos requisitos levantados.

3.2.1 Diagrama de Caso de Uso

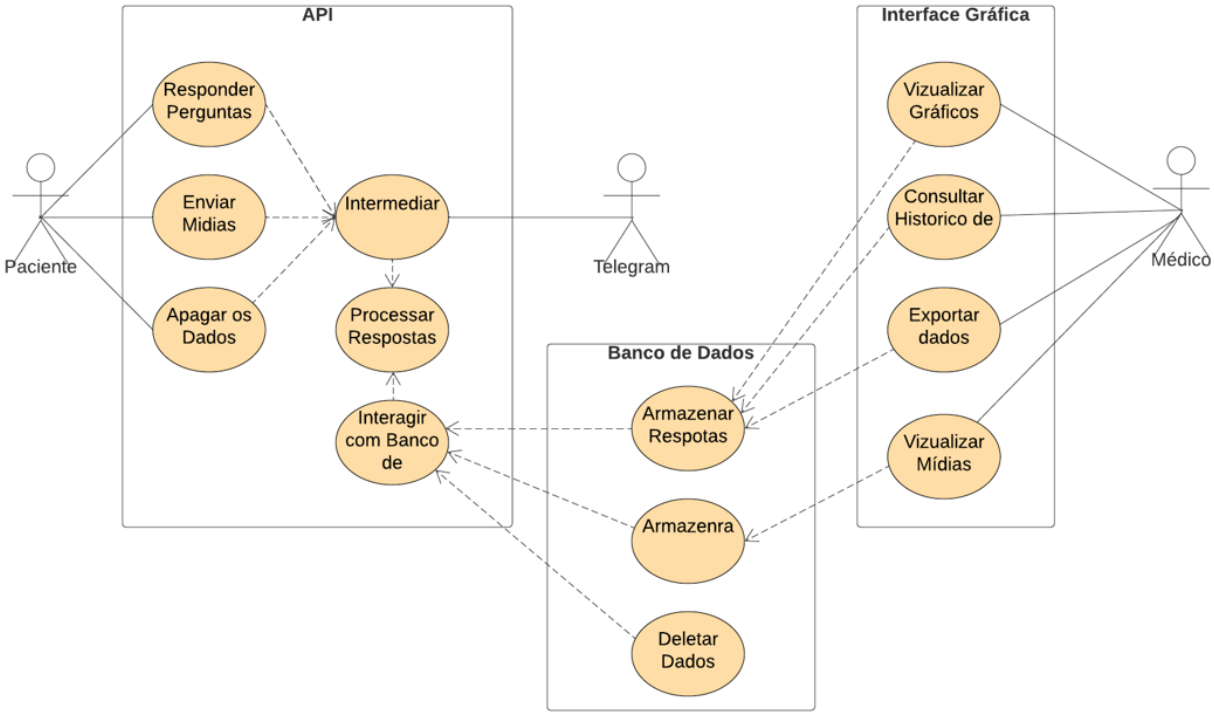
O Diagrama de Caso de Uso é uma ferramenta essencial na modelagem de sistemas, pois proporciona uma visão clara e compreensível das interações entre os atores (usuários ou sistemas externos) e os casos de uso (funcionalidades principais). Ele serviu como um planejamento geral do sistema, representando uma visão macro, focado nos atores e as interações principais com o sistema.

O Diagrama de Caso de Uso, descrito na Figura 7, apresenta uma visão geral das interações entre os atores (usuários e sistemas externos) e os principais casos de uso do sistema. Ele foi utilizado para mapear as funcionalidades oferecidas pelo sistema e como elas são acessadas pelos diferentes atores. Ele identifica os seguintes atores e casos de uso:

- Atores:
    - Paciente: Responde perguntas, envia mídias e solicita exclusão de dados.
    - Telegram: Intermedia mensagens entre paciente e *chatbot*, por meio da API.
    - Banco de Dados: Armazena e fornece dados para relatórios.
    - Médico/Operador: Visualiza gráficos, consulta históricos e exporta dados.
- Casos de Uso Principais:

- Paciente interage com o *chatbot*, por meio da API com o intermediador Telegram, para responder perguntas ou enviar mídias.
- A API processa as interações e armazena informações no banco de dados.
- Médico acessa a interface gráfica para visualizar gráficos e consultar informações.

Figura 7: Diagrama de Caso de Uso



Fonte: Autoria Própria

Esse diagrama foi crucial para alinhar os objetivos do sistema com as necessidades dos usuários e operadores, possibilitando compreender as funcionalidades principais e como cada ator contribui para o funcionamento do sistema.

3.2.2 Diagrama de Classe

O Diagrama de Classe detalha a modelagem das principais classes do banco de dados, os atributos, métodos e os relacionamentos entre elas. Assim, transitando para a modelagem do sistema, conectando funcionalidades do caso de uso às classes responsáveis por implementá-las.

O Diagrama de Classe, apresentado na Figura 8, modela as principais classes relacionadas ao banco de dados e os métodos associados. Ele detalha as tabelas centrais do sistema: usuarios, respostas e midias.

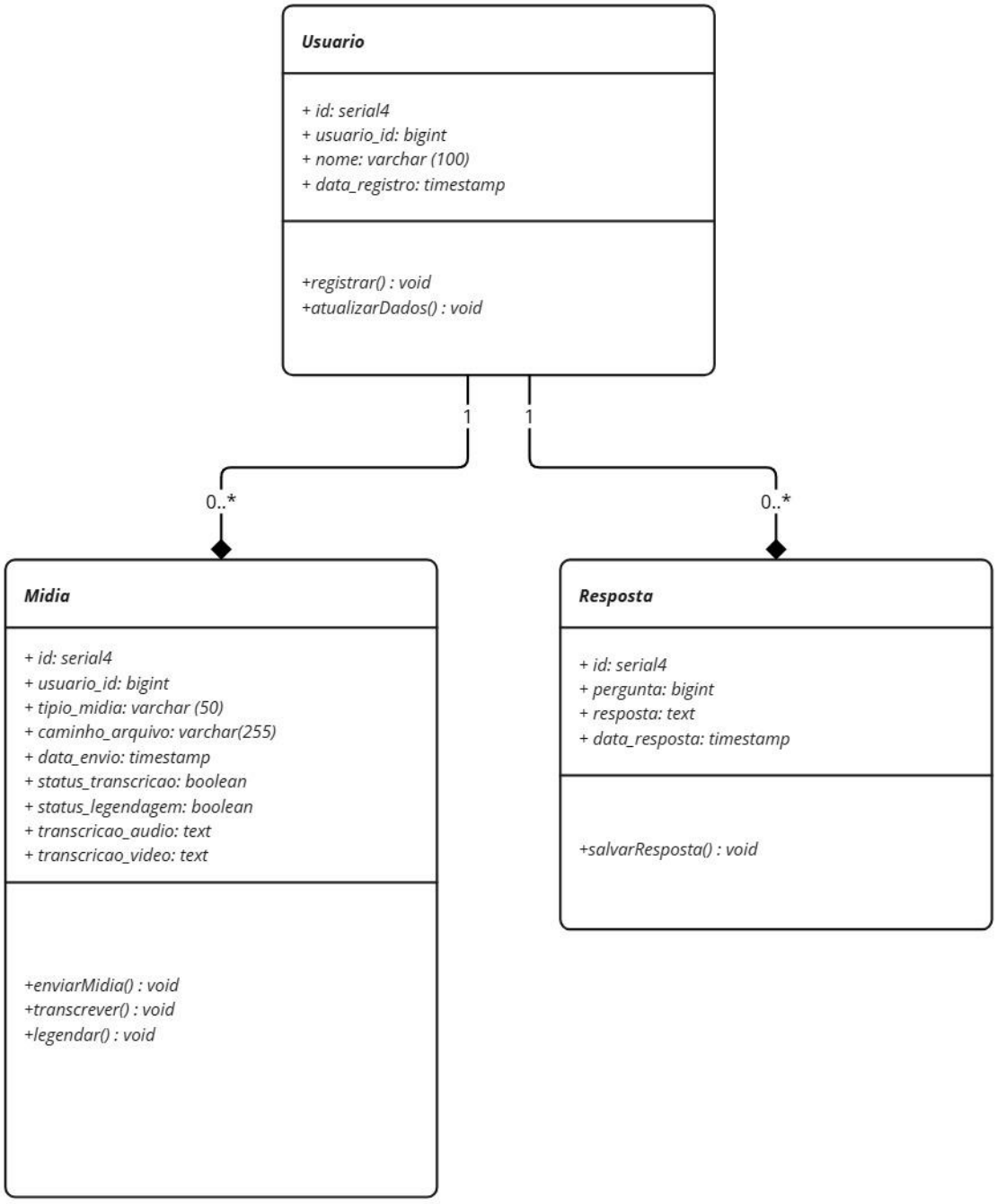
- Classes Representadas:
  - Usuário: Representa os pacientes e armazena informações como nome, ID no Telegram e data de registro.
  - Resposta: Registra as perguntas e as respectivas respostas fornecidas pelos pacientes às perguntas do *chatbot*.
  - Mídia: Contém informações sobre os arquivos enviados, se a mídia já foi transcrita ou legendagem, o caminho onde foi salvo.

Cada classe inclui métodos específicos, como:

- registrar() e atualizarDados() para a classe Usuario.
- salvarResposta() na classe Resposta.
- enviarMidia(), transcrever() e legendar() na classe Mídia.

A modelagem das classes assegurou a consistência entre o armazenamento e a recuperação de dados, facilitando a integração com o *backend*.

Figura 8: Diagrama de Classe



Fonte: Autoria Própria

3.2.3 Diagrama de Entidade-Relacional

O Diagrama Entidade-Relacionamento (DER) é um modelo conceitual utilizado para estruturar visualmente os dados de um sistema. Ele foi utilizado para modelar o banco de dados, evidenciando as entidades e os relacionamentos. Esse artefato foi

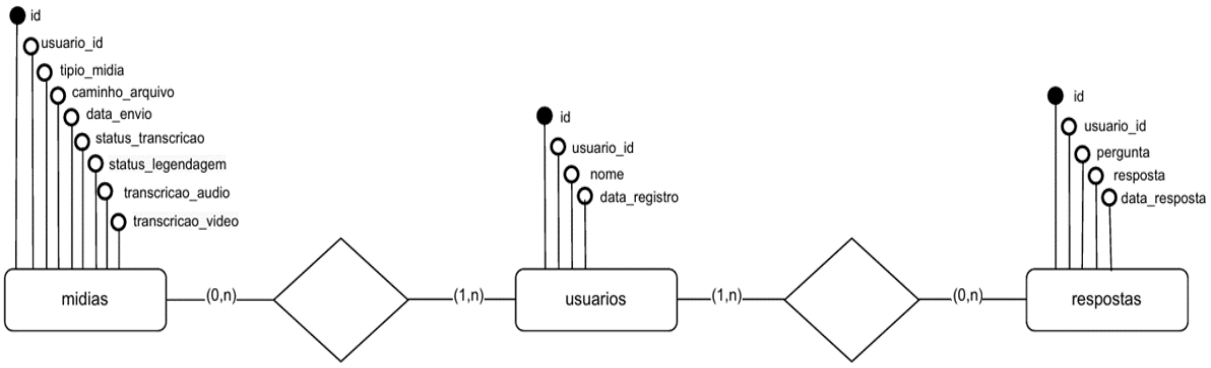
essencial para estruturar as informações que seriam armazenadas e processadas no sistema.

O DER, mostrado na Figura 9, modela a estrutura do banco de dados e os relacionamentos. As principais entidades incluem:

- Usuários: Representa os pacientes.
- Respostas: Registra as respostas fornecidas pelos pacientes.
- Mídias: Armazena informações sobre os arquivos enviados pelos pacientes.

Onde cada paciente pode estar associado a nenhuma ou várias respostas e mídias, e as respostas e mídias estão diretamente ligadas a um usuário, garantindo integridade referencial, garantindo a integridade referencial. A implementação dessa modelagem foi feita no PostgreSQL, que assegurou compatibilidade com o sistema, consultas otimizadas e armazenamento confiável.

Figura 9: Diagrama de Entidade-Relacional



Fonte: Autoria Própria

3.2.4 Diagrama de Sequência

O Diagrama de Sequência demonstra como os diferentes componentes do sistema interagem ao longo do tempo, representando cenários específicos. No caso do sistema proposto, o diagrama foi utilizado para ilustrar o fluxo de interação entre o paciente, o Telegram, o *bot*, o banco de dados e o médico. Esse diagrama evidencia

os passos realizados desde o envio de uma resposta ou mídia pelo paciente até a consulta de gráficos e diagnósticos pelo médico.

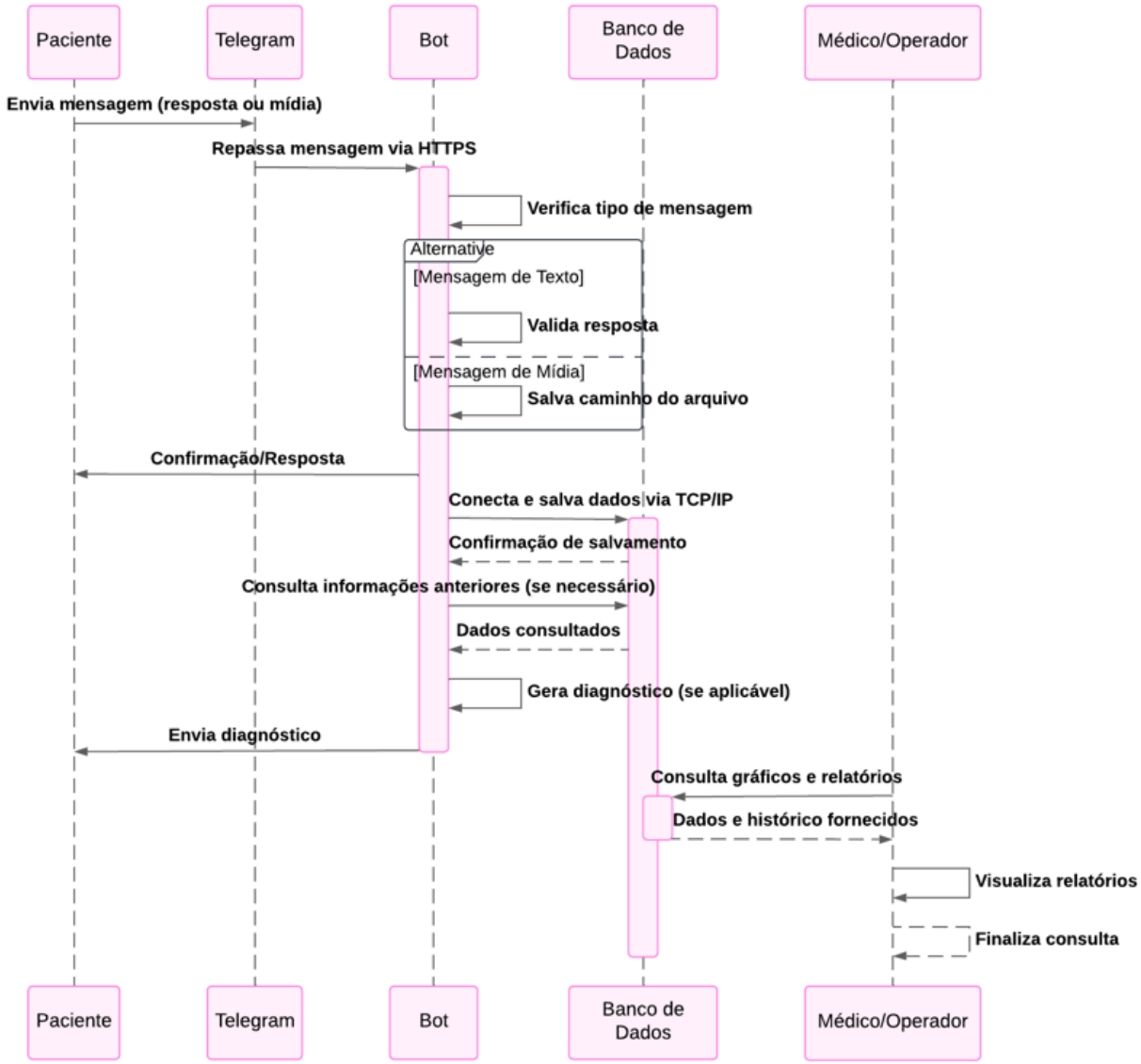
O Diagrama de Sequência, representado na Figura 10, ilustra a interação dinâmica entre os principais componentes do sistema ao longo do tempo. Ele detalha o fluxo de informações desde o envio de uma resposta ou mídia pelo paciente, passando pelo processamento no *backend* (*bot*) e armazenamento no banco de dados, até a consulta de informações pelo médico via interface gráfica.

- Componentes Representados:
    - Paciente: Interage com o *Bot* no Telegram.
    - Telegram: Plataforma intermediária responsável pela comunicação entre paciente e *Bot*.
    - *Bot*: Representa o *chatbot*, que processa mensagens e gerencia interações com o banco de dados.
    - Banco de Dados: Armazenar respostas, mídias e informações dos pacientes.
    - Médico/Operador: Acessa os dados e relatórios pela interface gráfica.

Fluxo Principal:

1. O paciente envia uma mensagem (texto ou mídia) via Telegram.
2. A mensagem é repassada ao *chatbot* por meio do protocolo Protocolo Seguro de Transferência de Hipertexto (em inglês, *HyperText Transfer Protocol Secure* - HTTPS).
3. O *chatbot* verifica o tipo de mensagem:
    - Mensagem de texto: Válida e armazena no banco de dados.
    - Mensagem de mídia: Salva o caminho do arquivo no banco de dados.
4. Caso necessário, o *chatbot* consulta informações anteriores para gerar diagnósticos ou confirmações.
5. As informações processadas são retornadas ao paciente ou disponibilizadas para o médico na interface gráfica.
6. Esse diagrama foi essencial para validar o fluxo lógico do sistema, identificando e corrigindo potenciais redundâncias ou falhas nas interações.

Figura 10: Diagrama de Sequência



Fonte: Autoria Própria

3.3 Tecnologias e Protótipos

Este capítulo apresenta as ferramentas e tecnologias utilizadas ao longo do projeto, bem como os protótipos criados para validar conceitos, testar funcionalidades e estruturar a solução final. Assim, garantindo que o desenvolvimento de um sistema eficiente e integrado para telemedicina tenha uma seleção cuidadosa de tecnologias e a validação por meio de protótipos funcionais.

A integração entre diferentes componentes, como o *chatbot* no Telegram, o *backend* desenvolvido em Python, o banco de dados PostgreSQL e a interface gráfica, exigiu uma abordagem iterativa, na qual protótipos intermediários desempenharam um papel essencial. Desde o uso de ferramentas preliminares, como planilhas Excel e gráficos gerados no *Power BI*, até a implementação final com tecnologias web personalizadas, cada etapa foi planejada e executada com foco na eficiência e na usabilidade.

Dessa forma, será detalhado neste capítulo as principais linguagens de programação, *frameworks* (conjunto estruturado de ferramentas, bibliotecas, regras e padrões), bibliotecas e ferramentas empregadas, além de descrever como os protótipos ajudaram a alinhar os objetivos do projeto com as soluções técnicas adotadas. O resultado foi um sistema integrado que atende às necessidades de telemonitoramento, oferecendo um fluxo contínuo desde a interação do paciente com o *chatbot* até a visualização de dados pelos médicos na interface gráfica.

3.3.1 Tecnologias Utilizadas

O desenvolvimento de um sistema eficiente para telemedicina, utilizando o Telegram como plataforma base, exigiu a combinação de diversas tecnologias modernas e a integração de ferramentas, linguagens de programação, *frameworks* e bibliotecas. A escolha das tecnologias foi feita pelos objetivos do projeto, de realizar a comunicação entre paciente, *chatbot*, Telegram, API, banco de dados e interface gráfica, além de diversas funcionalidades como de legendar vídeos e transcrever áudios, de forma a validar a possibilidade de se fazer telemedicina com esse sistema. Esta seção descreve as principais tecnologias utilizadas e como elas contribuíram para atingir os objetivos do sistema.

Linguagens de Programação, que foram dividias para *banckend*, *frontend* e para consulta e gerenciamento do banco de dados:

- Python: Foi utilizado no *backend* para o desenvolvimento do *chatbot*, o processamento de mensagens e a integração com o banco de dados PostgreSQL. A escolha foi motivada pela vasta gama de bibliotecas disponíveis e pela flexibilidade que Python oferece para sistemas escaláveis e integrados.

- HTML, CSS e JavaScript: Empregados na construção do *frontend*, essas tecnologias garantiram a criação de uma interface gráfica amigável e funcional para os médicos e operadores. Combinadas, proporcionaram interatividade e personalização para a exibição dos gráficos e relatórios.
- PostgreSQL: Escolhido como banco de dados relacional pela ediciência, confiabilidade e suporte para lidar com consultas avançadas e grandes volumes de dados. O PostgreSQL foi integrado diretamente ao backend para armazenamento seguro e eficiente de dados. Sendo a visualização feita por meio do DBeaver, devido a melhor experiência de navegar entre tabelas e dados salvos.

Frameworks e Bibliotecas, onde foram fundamentais para a construção do sistema:

- Flask: *Framework* Python utilizado no *backend* para gerenciar as requisições HTTP e configurar rotas da API. A leveza e simplicidade foram determinantes para o rápido desenvolvimento do sistema.
- psycopg2: Biblioteca responsável pela comunicação entre o *backend* em Python e o banco de dados PostgreSQL. Garantiu operações eficientes e seguras no gerenciamento de dados.
- Telegram Bot API: Interface essencial para integrar o *chatbot* ao Telegram, possibilitando a interação entre pacientes e o sistema de *backend*.
- Chart.js: Biblioteca JavaScript utilizada para renderizar gráficos na interface gráfica, proporcionando uma visualização clara e dinâmica dos dados.
- MoviePy e OpenAI Whisper: Ferramentas usadas para manipulação de vídeos e transcrição de áudios enviados pelos pacientes. A combinação possibilitou a criação de legendas em vídeos e o processamento de mídia diretamente pelo sistema.
- Watchdog: Utilizada nos scripts de transcrição e legendagem para monitorar alterações no sistema de arquivos em tempo real, automatizando o processamento de mídias enviadas.

Ferramentas, em que durante o desenvolvimento, ferramentas específicas foram utilizadas para aprimorar a produtividade e o gerenciamento do projeto:

- VS Code: O principal Ambiente de Desenvolvimento Integrado (em inglês, *Integrated Development Environment* - IDE) utilizado para o desenvolvimento do sistema. Ofereceu suporte para múltiplas linguagens e integração com ferramentas de versionamento, facilitando o fluxo de trabalho.
- PgAdmin 4 e DBeaver: Ferramentas para administração e visualização do banco de dados PostgreSQL. Enquanto o PgAdmin foi usado para gerenciar a modelagem do banco, o DBeaver foi útil na inspeção e análise de dados durante os testes.
- Lucidchart: Empregado na criação de diagramas UML, como os de sequência, caso de uso e implantação. Esses diagramas foram essenciais para planejar e documentar o sistema.

Integração das Tecnologias, onde o sistema foi projetado para funcionar de forma integrada, possibilitando que cada tecnologia desempenhasse seu papel de maneira coesa:

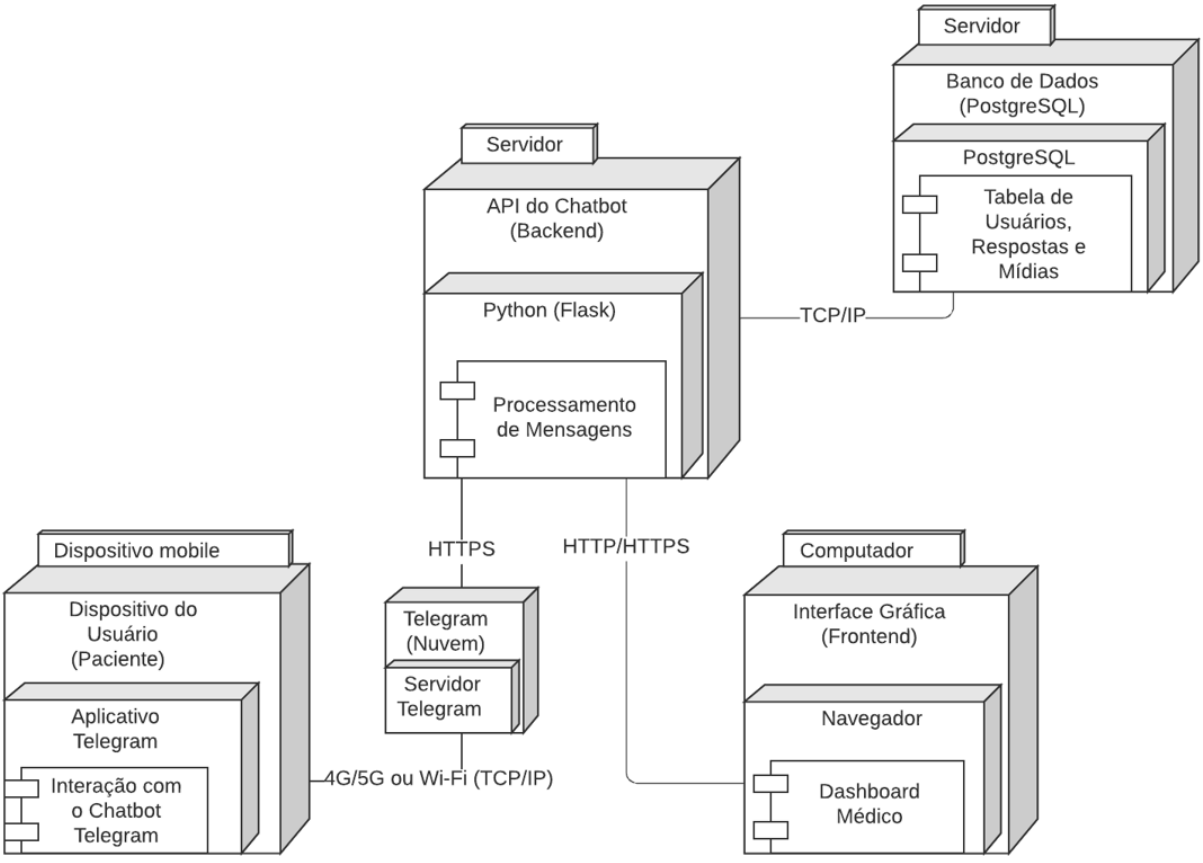
- *Backend*: Desenvolvido em Python utilizando Flask, foi responsável por processar as mensagens recebidas do *chatbot* via Telegram Bot API, armazenar as informações no banco de dados PostgreSQL e retornar os dados formatados para a interface gráfica.
- *Frontend*: A interface gráfica construída em HTML, CSS e JavaScript possibilitou que os médicos acessassem os dados armazenados, visualizassem gráficos gerados com Chart.js e exportassem relatórios personalizados.
- Banco de Dados: Modelado em PostgreSQL, o banco armazenou informações críticas, incluindo respostas dos pacientes, diagnósticos e mídias enviadas. A comunicação foi feita de forma segura e eficiente utilizando psycopg2.
- Processamento de Mídia: Scripts em Python com MoviePy e Whisper foram integrados para transcrever e legendar áudios e vídeos enviados pelos pacientes, melhorando a acessibilidade e organização das informações.
- Automação e Monitoramento: A biblioteca Watchdog foi configurada para monitorar diretórios de arquivos, iniciando automaticamente processos de transcrição e legendagem ao detectar novos envios de mídias.

3.3.1 Integração das Tecnologias

A integração das tecnologias foi realizada de forma a garantir a comunicação eficiente entre todos componentes utilizados, onde cada componente funcione de forma independente, porém integrada.

Para isso, foi feito um diagrama de implantação (Figura 11) para representar a arquitetura física do sistema, destacando como os componentes físicos e lógicos interagem entre si para viabilizar o sistema de telemonitoramento baseado no Telegram, evidenciando a distribuição de tarefas e a comunicação entre os elementos.

Figura 11:Diagrama de Implantação



Fonte: Autoria Própria

Conforme ilustrado na Figura 11, o diagrama de implantação apresenta a arquitetura geral do sistema, destacando a interação entre os componentes principais e os fluxos de dados que suportam o funcionamento do telemonitoramento por meio

do Telegram. Esse modelo foi planejado para garantir a eficiência, a escalabilidade e a integração das funcionalidades propostas.

1. Dispositivo do Usuário (Paciente):
    - O paciente utiliza um smartphone, tablet ou computador para acessar o aplicativo Telegram. O Telegram funciona como a interface principal para o envio de mensagens e mídias (áudios, vídeos ou imagens).
    - A comunicação ocorre por meio de 4G/5G ou Wi-Fi, utilizando o protocolo TCP/IP, que conecta o paciente ao servidor do Telegram.
2. Servidor do Telegram (Nuvem):
    - O Telegram atua como um intermediário, recebendo as mensagens do paciente e repassando-as para a API do *chatbot*. Essa comunicação é realizada utilizando HTTPS, garantindo a segurança e a criptografia dos dados durante o trânsito.
    - Este componente possibilita a troca contínua e confiável de mensagens entre o paciente e o *chatbot*, sendo uma das razões para a escolha do Telegram como plataforma.
3. API do *Chatbot* (*Backend*):
    - A API do *chatbot* foi desenvolvida em Python com o *framework* Flask, que facilita a criação de rotas e o gerenciamento de requisições HTTP.
    - A principal função da API é o processamento de mensagens. Após receber os dados do Telegram, a API analisa o conteúdo (texto ou mídia), executa as lógicas de negócio e interage com o banco de dados.
    - Para a comunicação com o banco de dados PostgreSQL, a API utiliza o protocolo TCP/IP, garantindo uma conexão segura e eficiente entre os componentes.
4. Banco de Dados (PostgreSQL):
    - O banco de dados PostgreSQL é responsável pelo armazenamento persistente das informações coletadas, como:
        - Respostas às perguntas feitas pelo *chatbot*.
        - Mídias enviadas pelos pacientes, incluindo transcrições de áudios e legendas de vídeos.
        - Histórico de interações, possibilitando consultas e análises futuras.

- A arquitetura do banco foi modelada para atender aos requisitos do sistema, incluindo suporte a consultas complexas e operações em tempo real, essenciais para o desempenho da aplicação.

5. Interface Gráfica (*Frontend*):
   - A interface gráfica foi desenvolvida com HTML, CSS e JavaScript, sendo acessada por médicos e operadores por meio de um navegador. Essa interface é projetada para ser responsiva e intuitiva, garantindo uma experiência de usuário eficiente.
   - As principais funcionalidades incluem:
     - Visualização de gráficos dinâmicos e relatórios detalhados.
     - Consulta de histórico de interações dos pacientes com o *chatbot*.
     - Acompanhamento das mídias enviadas, com suporte a transcrições e legendas geradas automaticamente.
6. Fluxo de Comunicação:
   - O sistema opera em um fluxo contínuo e integrado:
     1. O paciente envia uma mensagem ou mídia pelo Telegram.
     2. A mensagem é repassada para a API do *chatbot* via servidores do Telegram.
     3. A API processa os dados e os armazena no banco de dados PostgreSQL.
     4. O médico acessa a interface gráfica para consultar os dados processados, visualizar gráficos e tomar decisões clínicas.

3.3.2 Protótipos Desenvolvidos

Durante o processo de desenvolvimento, foram criados protótipos funcionais para validar conceitos, testar funcionalidades e estruturar o sistema final, desde a concepção da interação com o *chatbot*, passando por protótipos de telas para interface gráfica, até a utilização de Excel para armazenamento de dados. Os principais protótipos são descritos a seguir:

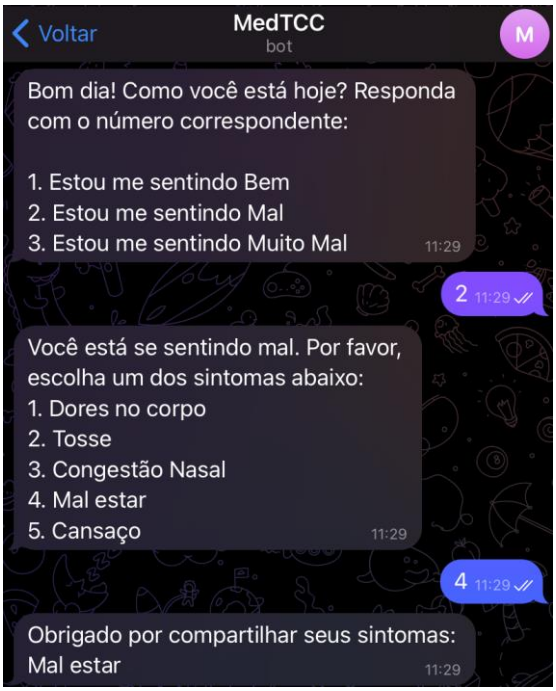
3.3.2.1 Protótipo Inicial do Chatbot

Inicialmente, para validar a possibilidade realizar telemedicina por meio do *chatbot*, foi pensado a utilização do mesmo para a prática de telemonitoramento mais precisamente. Assim, como protótipo inicial, foi realizada uma lógica muito simples de verificar os sintomas de um o paciente.

A Figura 12 demonstra um exemplo dessa interação entre paciente e *chatbot* para monitorar sintomas, em que as perguntas eram feitas dando opções de escolhas ao paciente. Dependendo do escolhido, abria-se mais possibilidades de tentar entender melhor o que o paciente estaria sentindo.

Isso foi feito a fim de que pudesse ser validado a possibilidade de se aplicar um caso onde, com base em perguntas com opções, um paciente conseguiria ser monitorado por um *chatbot*.

Figura 12: Primeira lógica de interação com o chatbot



Fonte: Autoria Própria

Posteriormente, foi adicionado uma lógica para o acompanhamento de gestantes em situações de risco. Assim, conseguindo por meio de repostas a uma

lógica de perguntas, determinar riscos de hipertensão ou até mesmo pré-eclâmpsia em gestantes (Lima, 2023).

Além disso, o *chatbot* também foi programado para receber mídias enviadas pelos pacientes. Assim, podendo enviar áudios, fotos e vídeos, que poderiam ser transcritos e legendados, depois armazenadas para que o médico tivesse acesso a interface gráfica.

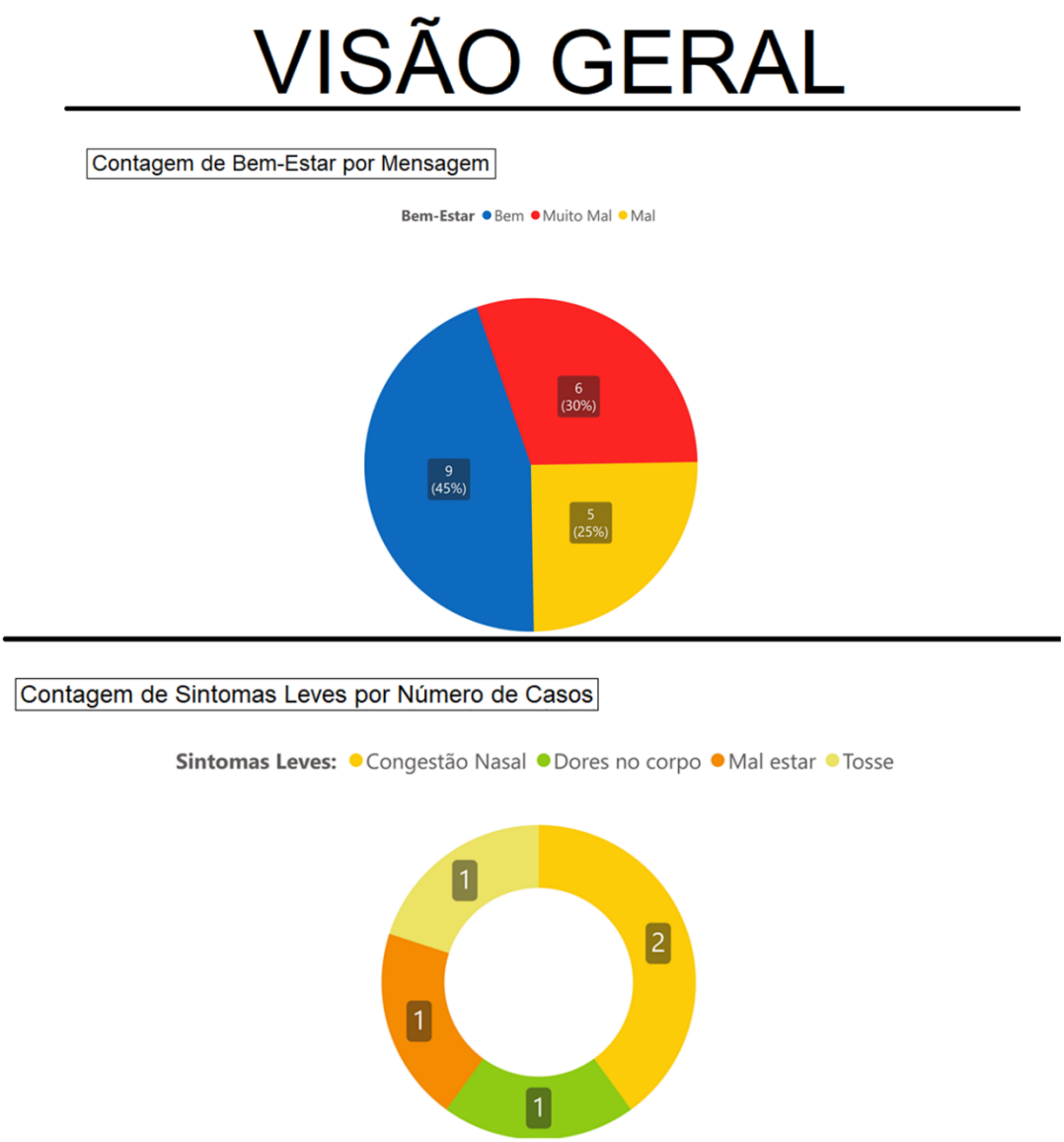
3.3.2.2 Protótipo da Interface Gráfica

A interface gráfica desempenha um papel essencial no sistema, pois é por meio dela que os médicos e operadores podem acessar e interpretar os dados coletados dos pacientes. O objetivo principal desse protótipo foi testar a viabilidade de um *dashboard*, uma interface visual que apresenta informações e métricas em um formato visualmente acessível e interativo, a fim de fornecer uma visão geral e detalhada de dados, facilitando o monitoramento, a análise e a tomada de decisões.

Dashboards geralmente utilizam gráficos, tabelas e outros elementos visuais para organizar as informações de maneira intuitiva. No nosso caso, utilizamos apenas gráficos que reunissem as informações monitoradas pelos pacientes, como bem-estar geral e sintomas leves, para que facilitassem na análise e a tomada de decisões do usuário médico.

Inicialmente, a interface gráfica foi projetada utilizando o *Power BI* (Figura 13) para a criação de relatórios e gráficos. A ideia sempre foi montar um *dashboard* para o médico poder ter, de forma visual, dados coletados e mantidos em forma de estatística de os pacientes monitorados.

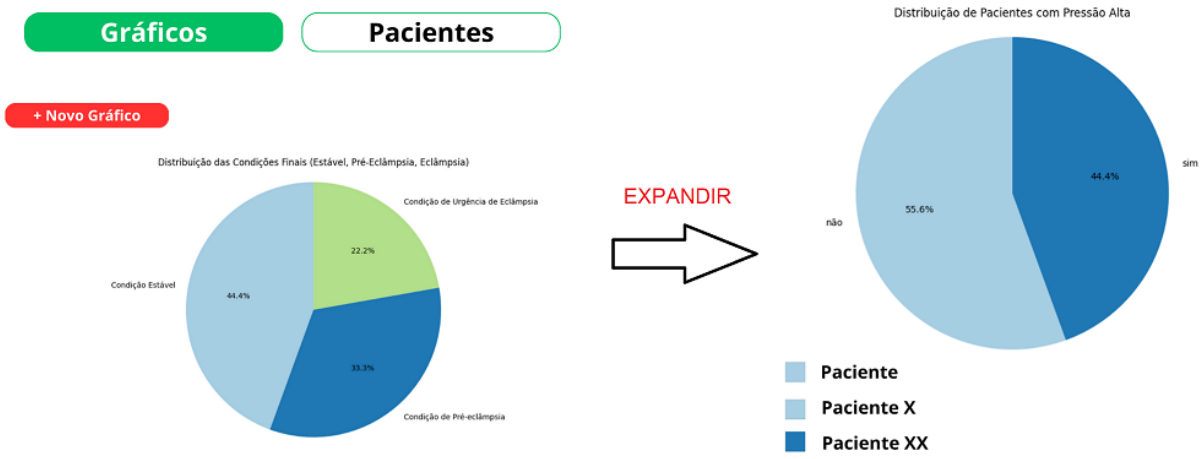
Figura 13: Protótipo de Dashboard utilizando Power BI



Fonte: Autoria Própria

No entanto, após testes de usabilidade, decidiu-se migrar para uma interface personalizada desenvolvida com HTML, CSS e JavaScript, oferecendo maior flexibilidade e personalização, e trazendo a possibilidade de dividir a interface em duas partes: a primeira contendo gráficos com base nas respostas dos pacientes, dando a possibilidade de geração de novos gráficos, bem como uma função de expansão de gráficos para ver mais detalhes de respostas (Figura 14).

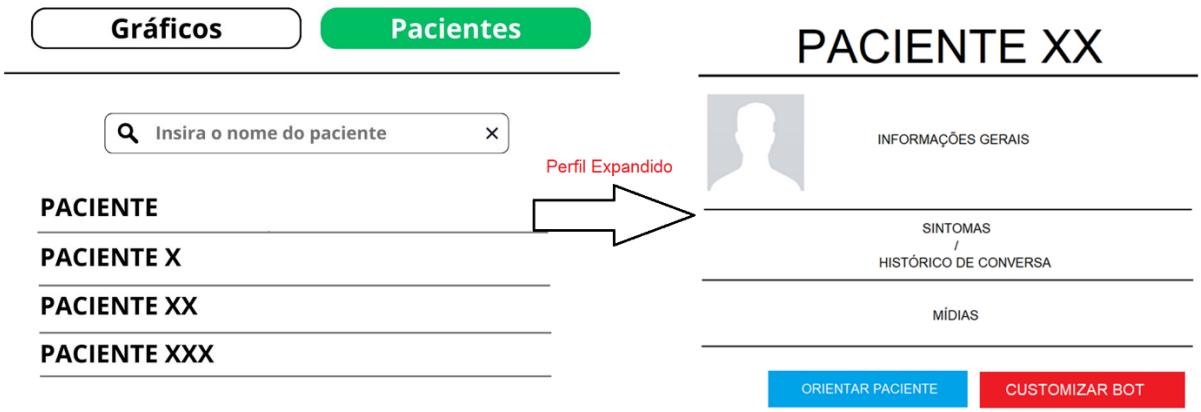
Figura 14: Protótipo de Dashboard utilizando HTML, CSS e Javascript



Fonte: Autoria Própria

A segunda aba seria para a visualização de perfil dos pacientes, contendo informações gerais, os históricos de mensagens e mídias enviadas por meio das interações com o *chatbot* (Figura 15).

Figura 15: Protótipo da aba Pacientes utilizando HTML, CSS e Javascript



Fonte: Autoria Própria

As possibilidades após a migração para HTML, CSS e Javascript iam desde possibilidade de gerar novos gráficos interativos da forma que quisesse, até a

possibilidade de orientar diretamente um paciente ou mudar a lógica das mensagens do *chatbot*.

3.3.2.3 Teste com Banco de Dados

No desenvolvimento inicial do sistema, a fase de testes utilizou diferentes abordagens para o armazenamento e a manipulação de dados, a fim de validar a melhor maneira de se guardar e manipular dados. A primeira abordagem foi utilizar o Excel, a fim de procurar uma maneira inicial simples. Após testes e validar a maneira como seriam modelados os dados, foi utilizado o PostgreSQL (PgAdmin 4) como banco de dados.

A Figura 16 mostra os dados sendo armazenados em planilhas Excel para testes iniciais. Nesse contexto, ainda se utilizava a lógica de *chatbot* da Figura 12, e a interface ainda era feita por meio do upload desses dados no *Power BI* (Figura 13).

Figura 16: Utilização do Excel para armazenar dados

	A	B	C	D
1	user_id	message	sintomas_leves	sintomas_graves
2	5688877850	Bem		
3	5688877850	Mal	Mal estar	
4	5688877850	Muito Mal		Vômito persistente
5	5688877850	Bem		
6	5688877850	Bem		
7	5688877850	Bem		
8	5688877850	Bem		
9	5688877850	Muito Mal		Perda de consciência
10	5688877850	Mal	Dores no corpo	
11	5688877850	Mal	Congestão Nasal	
12	5688877850	Mal	Congestão Nasal	
13	5688877850	Bem		
14	5688877850	Bem		
15	5688877850	Muito Mal		Dificuldade para respirar
16	5688877850	Mal	Tosse	
17	5688877850	Muito Mal		Vômito persistente
18	5688877850	Bem		
19	5688877850	Muito Mal		Dificuldade para respirar
20	5688877850	Bem		
21	5688877850	Muito Mal		Febre alta

Fonte: Autoria Própria

A Figura 17 mostra os dados sendo armazenados em planilhas no PostgreSQL, com a visualização do DBeaver.

Isso ocorreu, após a validação dos requisitos, onde se viu a importância de migrar para um sistema SQL. Após isso, o banco de dados foi modelado para atender as necessidades do sistema, com tabelas específicas para usuários, respostas e mídias (Figura 8).

Figura 17: Utilização do PostgreSQL para armazenar dados

	id	user_id	foto_enviada	pressao_alta	inchaco	cefaleia	ganho_peso	resposta_final
1	1	5.688.877.850		não	não	não	1	Condição Estável
2	2	5.688.877.850	photo_5688877850.jpg	sim	sim	sim	5	Condição de Urgência de Eclâmpsia
3	3	5.688.877.850		não	não	sim	3	Condição de Pré-eclâmpsia
4	4	5.688.877.850		não	não	não	1	Condição Estável
5	5	5.688.877.850		não	não	não	1	Condição Estável
6	6	5.688.877.850		sim	não	não	3	Condição de Pré-eclâmpsia
7	7	5.688.877.850	photo_5688877850.jpg	sim	sim	sim	10	Condição de Urgência de Eclâmpsia
8	8	5.688.877.850		não	não	não	1	Condição Estável
9	9	5.688.877.850		sim	sim	não	5	Condição de Pré-eclâmpsia
10	10	5.688.877.850		não	não	não	1	Condição Estável
11	11	5.688.877.850		não	não	não	1	Condição Estável

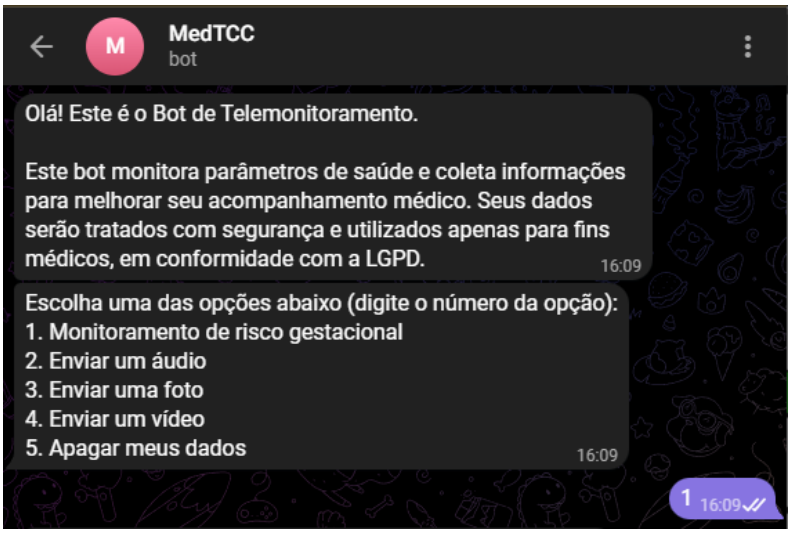
Fonte: Autoria Própria

3.3.2.4 Protótipo Final

O protótipo final do projeto consolida todas as funcionalidades planejadas ao longo do desenvolvimento, integrando *chatbot*, interface gráfica e banco de dados para um sistema eficiente de telemonitoramento voltado para gestantes em situações de alto risco.

A última versão do projeto seguiu o que os protótipos já vinham propondo: Um *chatbot* usado para monitorar gestantes em situações de alto risco, com consentimento de transparência informado, e com a função também receber mídias e apagar dados quando o usuário quiser (Figura 18).

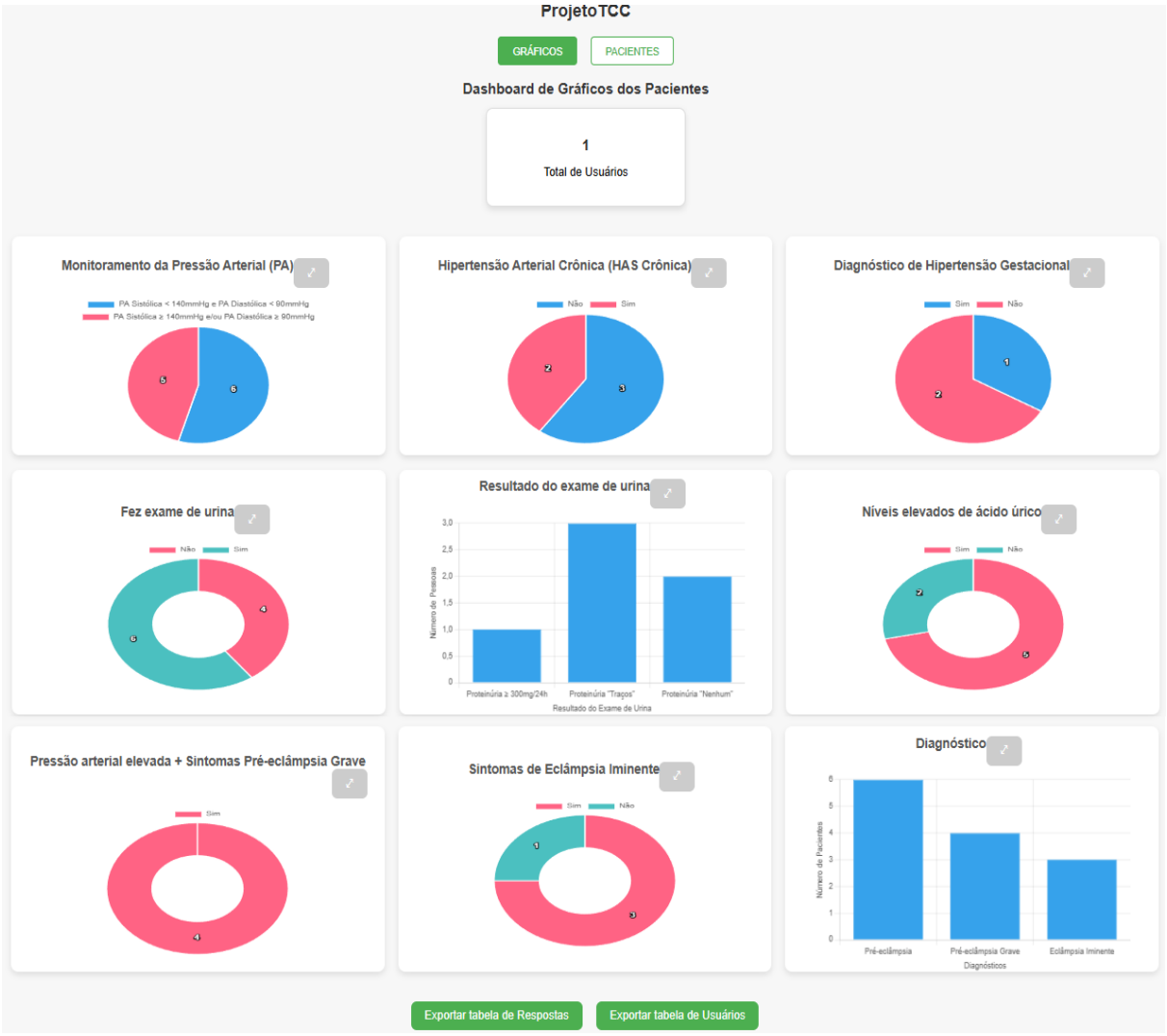
Figura 18: Interação inicial com a última versão do chatbot



Fonte: Autoria Própria

A Figura 19 exibe uma interface gráfica contendo uma aba para de *dashboard*, focada em conter gráficos que demonstrem a quantidade de respostas dadas a determinadas perguntas. Na tela principal é exibido a quantidade total de usuários, de possuir gráficos tipo pizza, donuts e em barras, e no fim da página botões para exportar esses dados no formato do Excel.

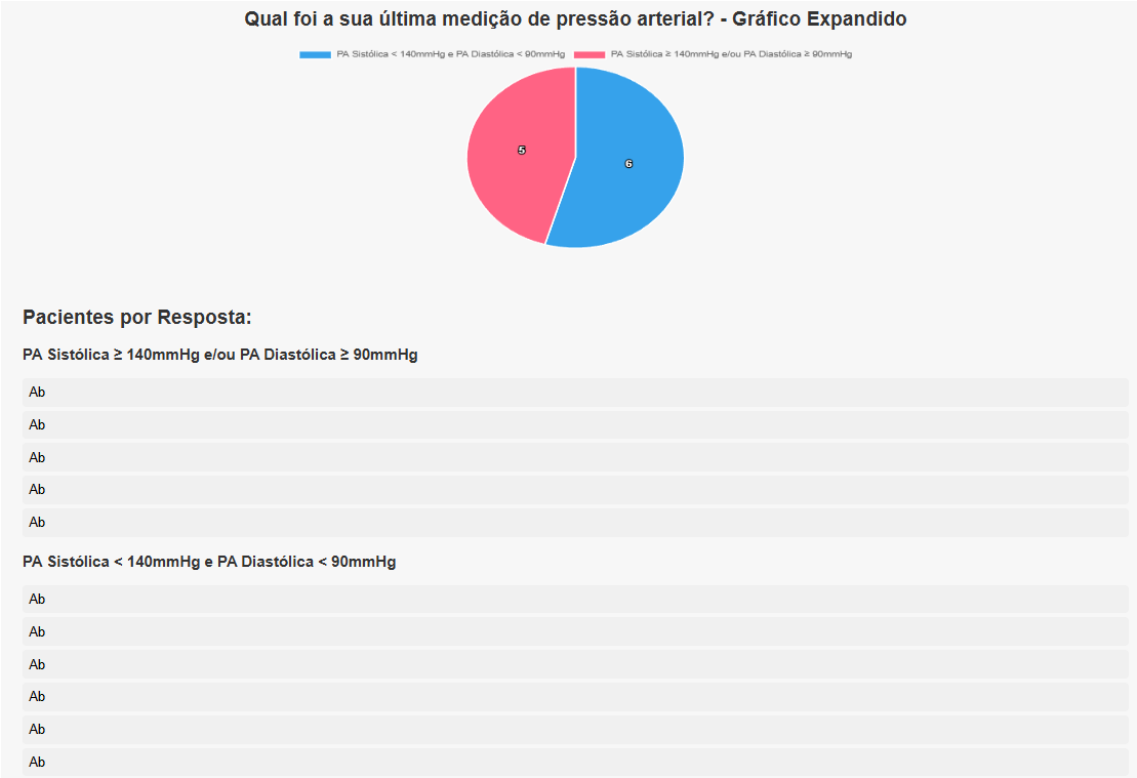
Figura 19: Aba de gráficos na versão final da Interface Gráfica



Fonte: Autoria Própria

A Figura 20 exibe uma das funcionalidades contidas na tela principal da interface gráfica, que é a de expandir o gráfico. Essa funcionalidade tem como objetivo mostrar para o usuário médico o gráfico com a informação a mais de saber exatamente que foi o paciente que deu determinada resposta. Assim, o usuário médico consegue obter mais informação para suas análises.

Figura 20: Expansão do gráfico na versão final da Interface Gráfica



Fonte: Autoria Própria

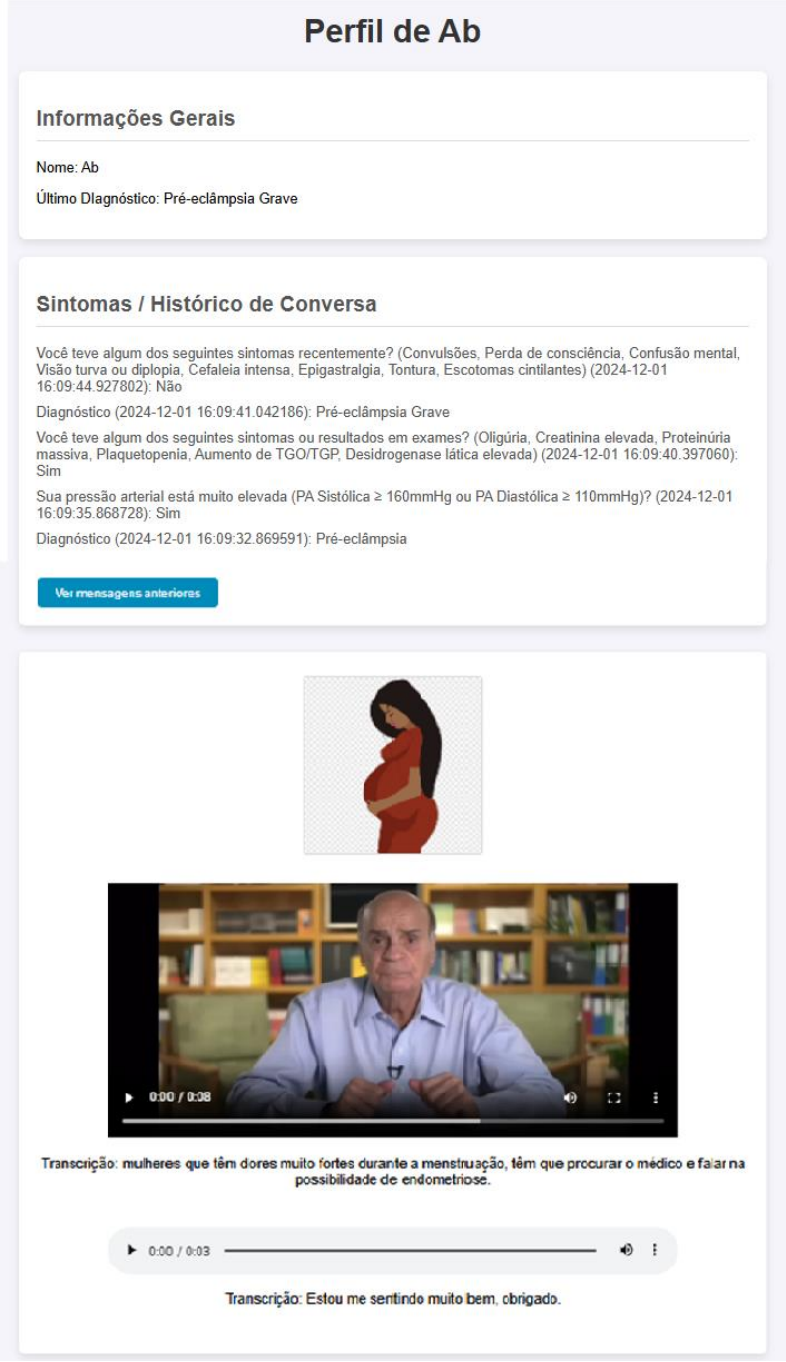
E uma outra aba que pode ser escolhida na parte superior da interface gráfica é a aba de Pacientes. A Figura 21 mostra esta aba, em que é mostrado ao usuário médico a lista com todos os pacientes registrados no sistema, ou seja, com todos que já interagiram com o *chatbot*. Além disso, está presente a funcionalidade de buscar pelo nome do paciente.

Figura 21: Aba de perfis de pacientes na versão final da Interface Gráfica

Fonte: Autoria Própria

Ao selecionar um paciente, é mostrado ao usuário médico a página contendo o perfil do paciente. A Figura 22 mostra a tela de perfil, contendo o nome do paciente, o último diagnóstico que o *chatbot* apontou que provavelmente ele estaria, o histórico de conversa com o *chatbot*. E ao final da página é exibido as mídias enviadas, bem como a transcrição do áudio e vídeo, e a presença do vídeo já legendado.

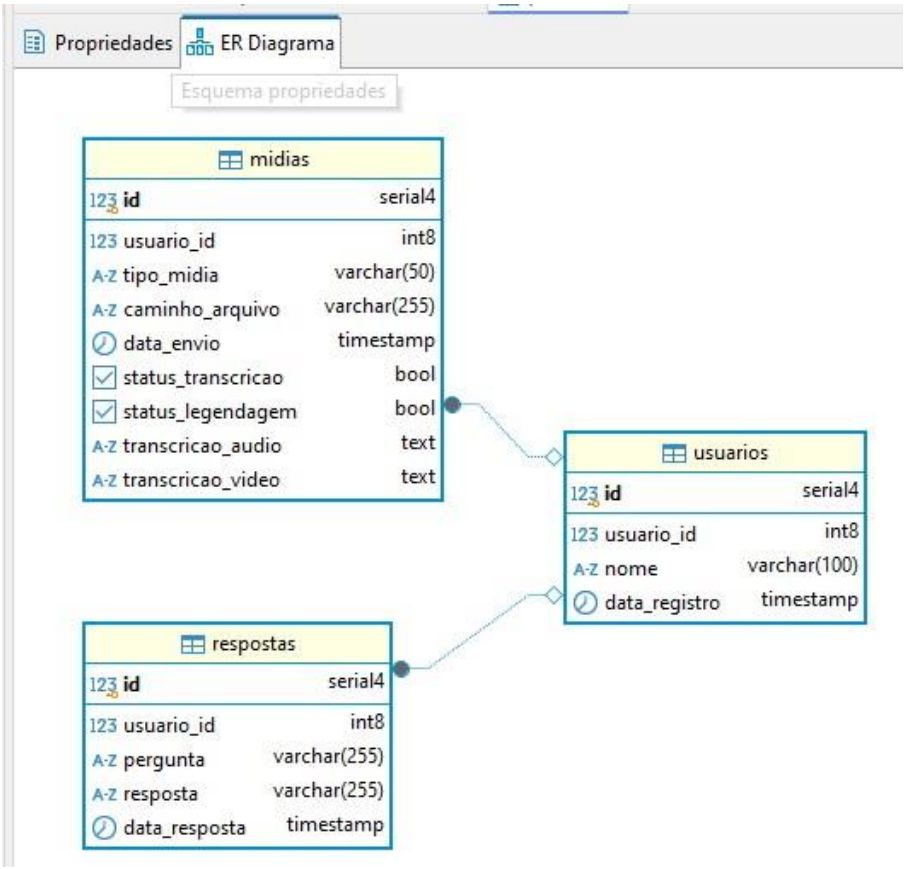
Figura 22: Perfil expandido do paciente versão final da Interface Gráfica



Fonte: Autoria Própria

Por fim, no protótipo final temos o banco de dados. A Figura 23 mostra o banco de dados modelado para o sistema. Contendo tabela para guardar informações dos pacientes usuários, outra tabela para armazenar as respostas dos usuários para as perguntas feitas pelo *chatbot*, e outra tabela para cuidar da parte de mídias que o usuário pode mandar, armazenando o tipo de mídia, se já foi transcrita ou legendada, qual a transcrição, dentre outras informações.

Figura 23: Diagrama Entidade Relacional da versão final do banco de dados



Fonte: Autoria Própria

4 Avaliação da Hipótese

Neste capítulo, será abordada a avaliação da hipótese central deste trabalho: a viabilidade de utilizar uma plataforma de propósito geral, como o Telegram, de forma customizada para suporte à telemedicina, mais especificamente para a prática de telemonitoramento. A proposta busca explorar as vantagens associadas a essa abordagem, como o baixo custo de desenvolvimento, maior acessibilidade e engajamento dos usuários, ao mesmo tempo em que considera os riscos e limitações que podem impactar a confiabilidade e a segurança do sistema.

A avaliação do sistema foi conduzida por meio de testes de ponta a ponta, que simulam cenários reais de interação entre pacientes e médicos. Esses testes foram essenciais para validar o funcionamento das funcionalidades implementadas, garantir a integração entre os componentes do sistema (*chatbot* no Telegram, *backend*, banco de dados e interface gráfica).

Além disso, a análise considerou os benefícios da escolha de uma plataforma amplamente utilizada e com características familiares aos usuários, como o Telegram, que facilita o acesso à tecnologia e elimina custos de treinamento extensivo. No entanto, também foram identificados riscos relacionados à segurança e à privacidade, inerentes ao uso de uma plataforma que não foi projetada originalmente para telemedicina.

Assim, este capítulo detalha a metodologia de avaliação utilizada, os testes realizados, os resultados alcançados e as limitações do sistema, fornecendo uma análise crítica sobre a viabilidade e os desafios de adotar uma plataforma de propósito geral para telemonitoramento.

4.1 Metodologia de Avaliação

A avaliação do sistema seguiu uma metodologia centrada em testes de ponta a ponta, cujo objetivo foi verificar a funcionalidade integral do sistema, desde a interação inicial do paciente até a exibição dos dados na interface médica. Esses testes abrangeram a integração entre o *chatbot* no Telegram, o *backend* desenvolvido em Flask, o banco de dados PostgreSQL e a interface gráfica.

O propósito da metodologia foi:

- Garantir que o sistema atingisse os objetivos de funcionalidade e integração propostos.
- Validar a hipótese de que o uso de uma plataforma de propósito geral, como o Telegram, pode ser customizado para práticas de telemonitoramento.
- Avaliar a viabilidade técnica e prática da solução frente aos riscos e desafios identificados.

4.2 Processo de Avaliação

Inicialmente, para validar a possibilidade realizar telemedicina por meio do *chatbot*, foi pensado a utilização do mesmo para a prática de telemonitoramento mais precisamente. Assim, como protótipo inicial, foi realizada uma lógica muito simples de verificar os sintomas de um o paciente.

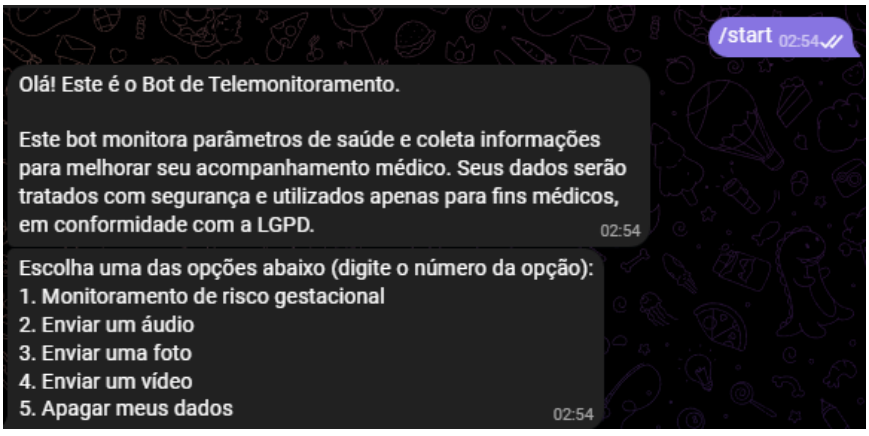
Os testes simulam cenários reais de uso, abordando:

4.2.1 Interações pelo Chatbot

Envio de mensagens textuais e mídias pelos pacientes, incluindo a validação da transcrição automática de áudios e legendagem de vídeos.

A Figura 24 representa a primeira interação do usuário com o *chatbot*, onde lhe será informado sobre o consentimento do uso de seus dados, seguindo normas da LGPD.

Figura 24:Primeira interação do usuário com o chatbot



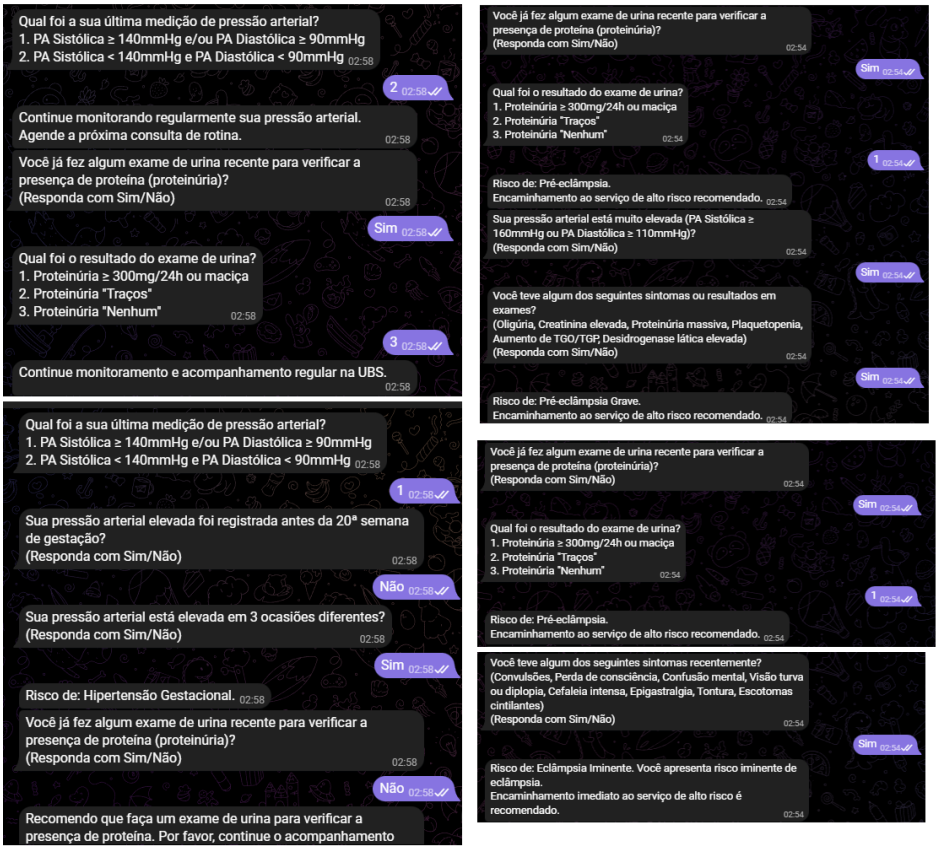
Fonte: Autoria Própria

Após essa primeira interação, será dado ao usuário paciente as opções de: Monitorar seu risco gestacional, onde será iniciado uma série de fluxos de perguntas e respostas para verificar o risco de Hipertensão Gestacional Crônica (HAS Crônica), Hipertensão Gestacional, Pré-eclâmpsia, Pré-eclâmpsia Grave e Eclâmpsia Iminente, sendo o fluxo e lógica de conversar do Apêndice A, Figuras 1, 2, 3, 4 e 5.

Enquanto isso, o sistema terá reconhecido o usuário novo e terá o incluído no banco de dados, por meio da execução de funções do *chatbot* presentes no código fonte do Apêndice B, Figura 3. Essas funções são iniciadas por meio da execução do código fonte do Apêndice B 2, Figura 2. Sendo o Apêndice B, Figura 3 também responsável por toda lógica de mandar mensagens, receber, processar e armazenar as respostas no banco de dados, que mantém a conexão com o sistema pela execução do Apêndice B, Figura 4.

A Figura 25 mostra as mais diversas interações de respostas para o monitoramento de risco em caso gestacional. Foram realizados o teste de 22 cenários de riscos mais diversos.

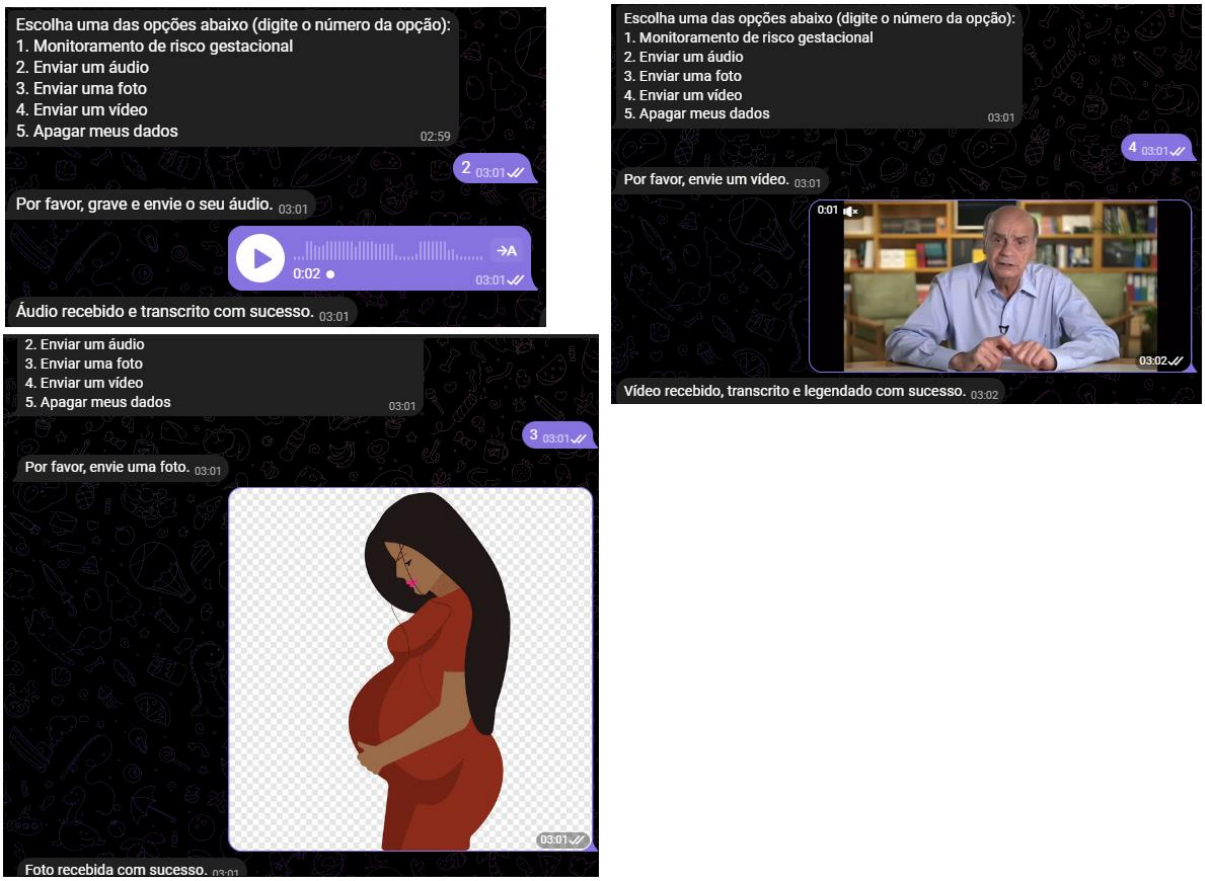
Figura 25: Diversas interações de monitoramento do usuário com o chatbot



Fonte: Autoria Própria

Além disso, foram testadas as opções de envio de mídias, sendo enviado um áudio, foto e vídeo (Figura 26). Com isso, tem-se a execução do código fonte do Apêndice B, Figuras 5 e 6, para a transcrição do áudio e vídeo, a legendagem do vídeo e os armazenamentos correspondentes.

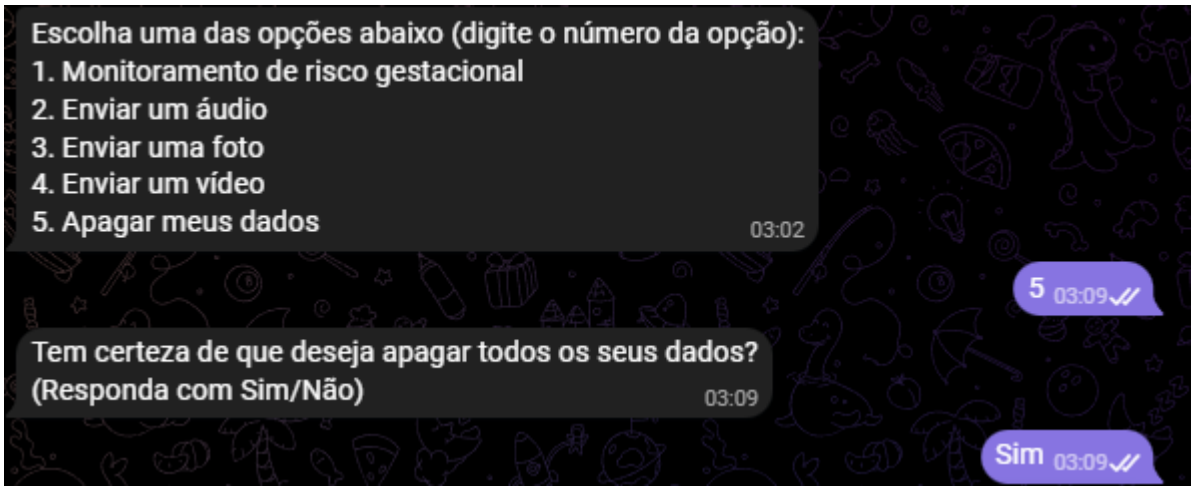
Figura 26: Envio de mídias do usuário com o chatbot



Fonte: Autoria Própria

Por fim, a Figura 27 mostra a última interação do teste, que é o pedido de deletar os dados salvos.

Figura 27: Pedido de deletar dados do usuário



Fonte: Autoria Própria

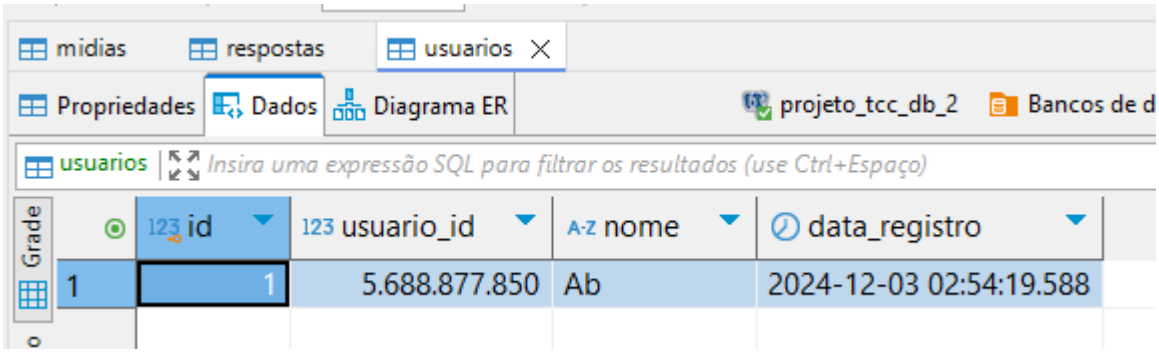
Assim, o teste de interação com o *chatbot* abrange todas as opções disponíveis ao usuário paciente.

4.2.2 Armazenamento de Dados:

Verificação do armazenamento correto no banco de dados PostgreSQL, contemplando informações textuais, transcrição de áudio e caminhos de mídias, conforme os testes realizados na seção anterior.

A Figura 28 mostra o usuário salvo na tabela de usuários. Ação feita quando um novo usuário interage com o *chatbot* pela primeira vez (Figura 24), após responder a alguma pergunta.

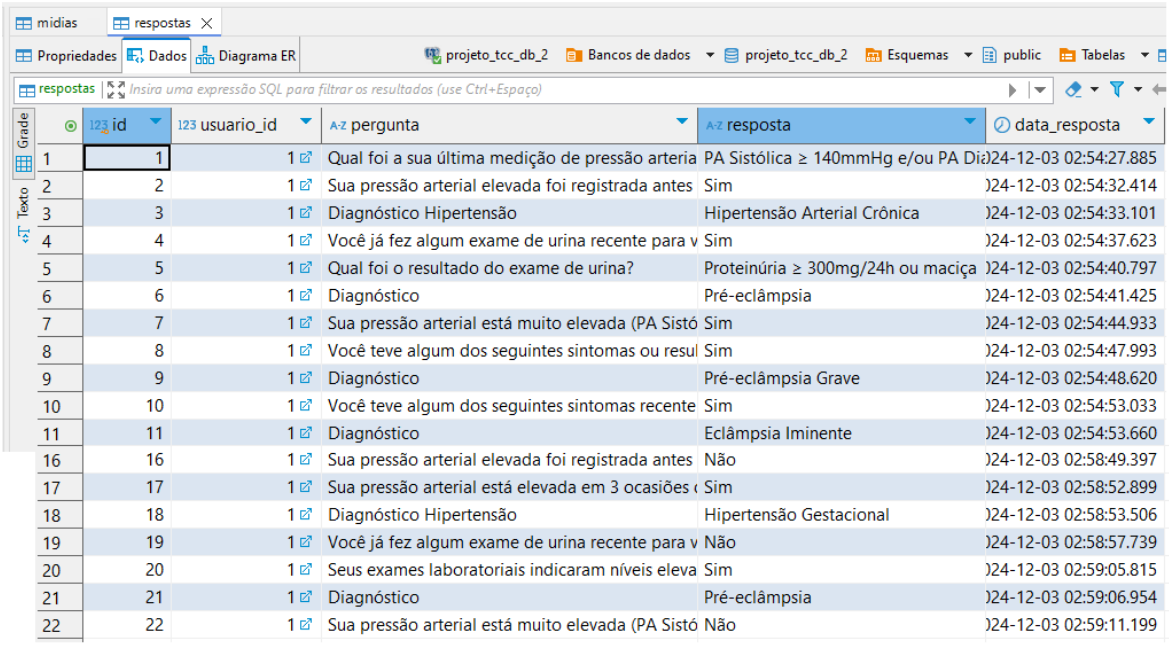
Figura 28: Tabela usuários



Fonte: Autoria Própria

A Figura 29 mostra as perguntas feitas pelo *chatbot* e as respectivas respostas enviadas pelo usuário. Ação realizada o usuário fez o teste de 22 cenários de riscos diversos com o *chatbot* (Figura 25).

Figura 29: Tabela de respostas



midias | respostas

Propriedades | Dados | Diagrama ER | projeto_tcc_db_2 | Bancos de dados | projeto_tcc_db_2 | Esquemas | public | Tabelas

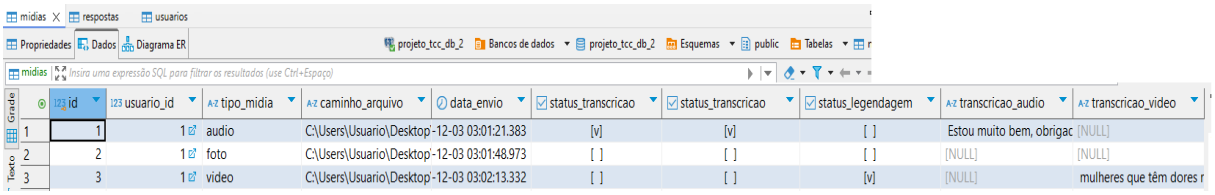
respostas | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	id	usuario_id	pergunta	resposta	data_resposta
1	1	1	Qual foi a sua última medição de pressão arteria	PA Sistólica ≥ 140mmHg e/ou PA Dia	24-12-03 02:54:27.885
2	2	1	Sua pressão arterial elevada foi registrada antes	Sim	24-12-03 02:54:32.414
3	3	1	Diagnóstico Hipertensão	Hipertensão Arterial Crônica	24-12-03 02:54:33.101
4	4	1	Você já fez algum exame de urina recente para v	Sim	24-12-03 02:54:37.623
5	5	1	Qual foi o resultado do exame de urina?	Proteinúria ≥ 300mg/24h ou maciça	24-12-03 02:54:40.797
6	6	1	Diagnóstico	Pré-eclâmpsia	24-12-03 02:54:41.425
7	7	1	Sua pressão arterial está muito elevada (PA Sistó	Sim	24-12-03 02:54:44.933
8	8	1	Você teve algum dos seguintes sintomas ou resul	Sim	24-12-03 02:54:47.993
9	9	1	Diagnóstico	Pré-eclâmpsia Grave	24-12-03 02:54:48.620
10	10	1	Você teve algum dos seguintes sintomas recente	Sim	24-12-03 02:54:53.033
11	11	1	Diagnóstico	Eclâmpsia Iminente	24-12-03 02:54:53.660
16	16	1	Sua pressão arterial elevada foi registrada antes	Não	24-12-03 02:58:49.397
17	17	1	Sua pressão arterial está elevada em 3 ocasiões (	Sim	24-12-03 02:58:52.899
18	18	1	Diagnóstico Hipertensão	Hipertensão Gestacional	24-12-03 02:58:53.506
19	19	1	Você já fez algum exame de urina recente para v	Não	24-12-03 02:58:57.739
20	20	1	Seus exames laboratoriais indicaram níveis eleva	Sim	24-12-03 02:59:05.815
21	21	1	Diagnóstico	Pré-eclâmpsia	24-12-03 02:59:06.954
22	22	1	Sua pressão arterial está muito elevada (PA Sistó	Não	24-12-03 02:59:11.199

Fonte: Autoria Própria

A Figura 30 mostra as três mídias enviadas pelo usuário. Ação realizada o usuário enviou um áudio, uma foto e um vídeo (Figura 26).

Figura 30: Tabela de mídias



midias | respostas | usuarios

Propriedades | Dados | Diagrama ER | projeto_tcc_db_2 | Bancos de dados | projeto_tcc_db_2 | Esquemas | public | Tabelas

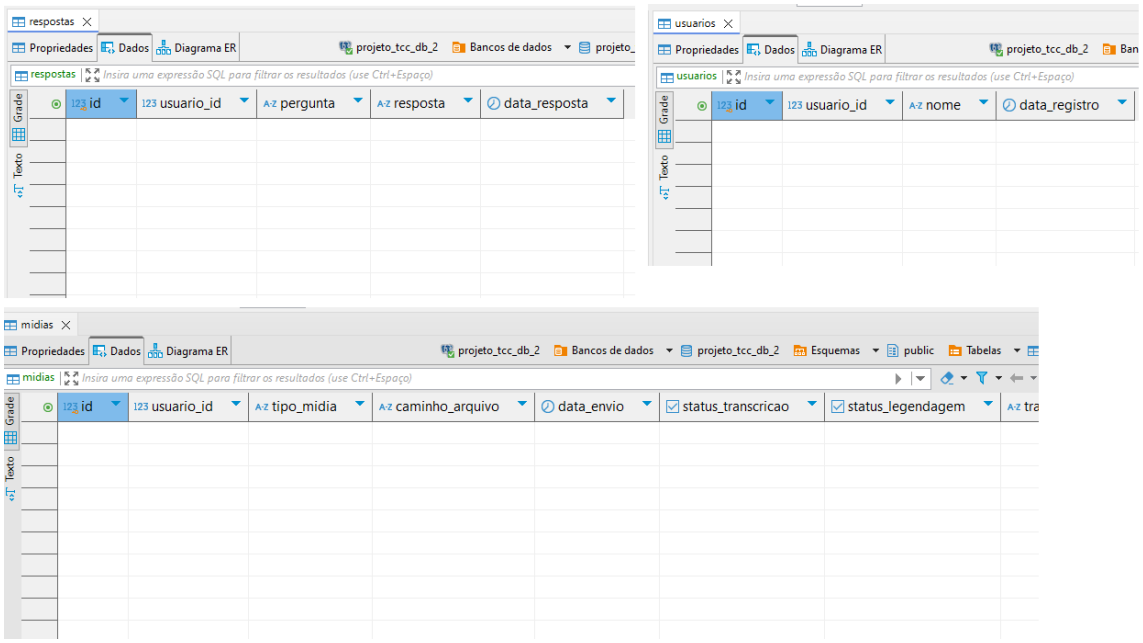
midias | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	id	usuario_id	tipo_midia	caminho_arquivo	data_envio	status_transcricao	status_transcricao	status_legendagem	transcricao_audio	transcricao_video
1	1	1	audio	C:\Users\Usuario\Desktop -12-03 03:01:21.383		[v]	[v]	[ ]	Estou muito bem, obrigac	[NULL]
2	2	1	foto	C:\Users\Usuario\Desktop -12-03 03:01:48.973		[ ]	[ ]	[ ]	[NULL]	[NULL]
3	3	1	video	C:\Users\Usuario\Desktop -12-03 03:02:13.332		[ ]	[ ]	[v]	[NULL]	mulheres que têm dores r

Fonte: Autoria Própria

A Figura 31 mostra as três tabelas vazias, que foram resultado da ação escolhida pelo usuário de apagar seus dados (Figura 27).

Figura 31: Tabelas vazias



Fonte: Autoria Própria

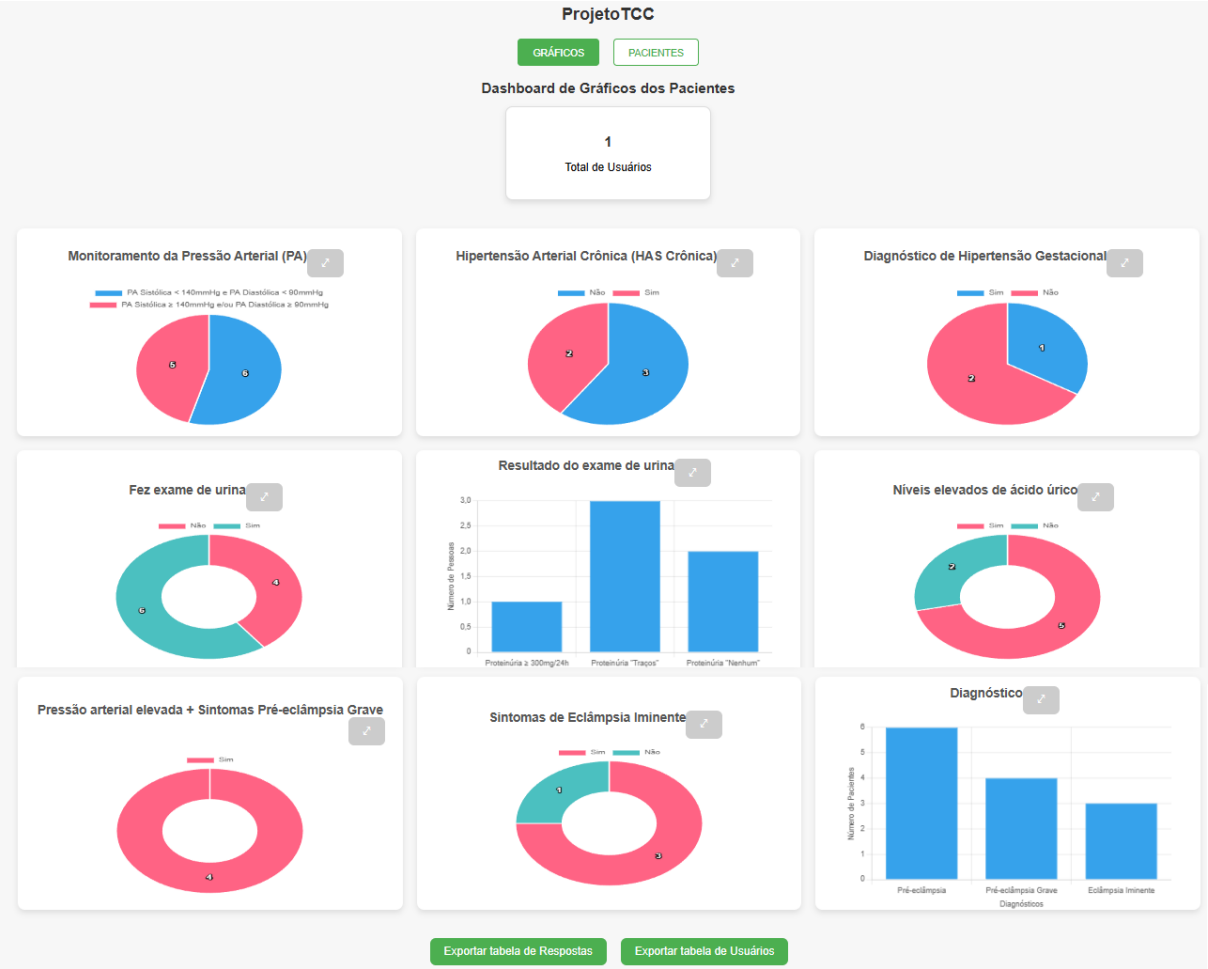
4.2.3 Consulta e Visualização:

Validação da interface gráfica, analisando se gráficos e relatórios refletem corretamente as interações registradas pelo usuário na seção 4.2.1.

Para a exibição dos gráficos e de outros elementos na interface gráfica, é necessário acessar os dados armazenados. Para isso, tem-se a execução do código fonte do Apêndice B, Figura 1, que busca as informações contidas no banco de dados e é exibido na interface gráfica por meio da execução dos códigos fontes do Apêndice B, Figura 7 e 10.

A Figura 32 é a tela inicial da interface gráfica do médico, em que tais estáticas representadas na forma de gráficos são decorrentes da interação do usuário que fez o teste de 22 cenários de riscos diversos com o *chatbot* (Figura 25).

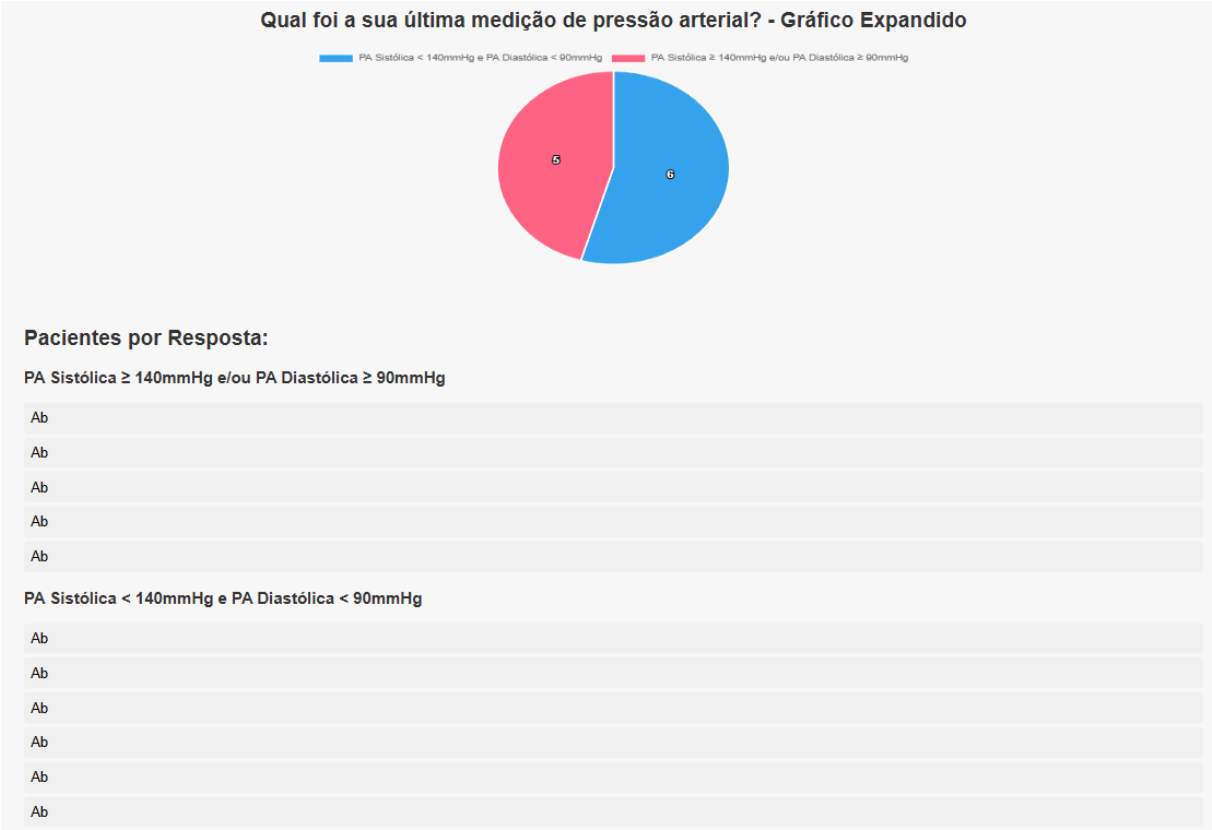
Figura 32: Aba gráficos da interface gráfica



Fonte: Autoria Própria

Ainda na Figura 32 temos algumas funções na tela, como a de expandir os gráficos, o que possibilita ver com mais detalhes os usuários que mandaram tais respostas mostradas pelo gráfico (Figura 33). Ação essa realizada por meio da execução do código fonte do Apêndice B, Figura 8 e 10.

Figura 33: Gráfico expandido

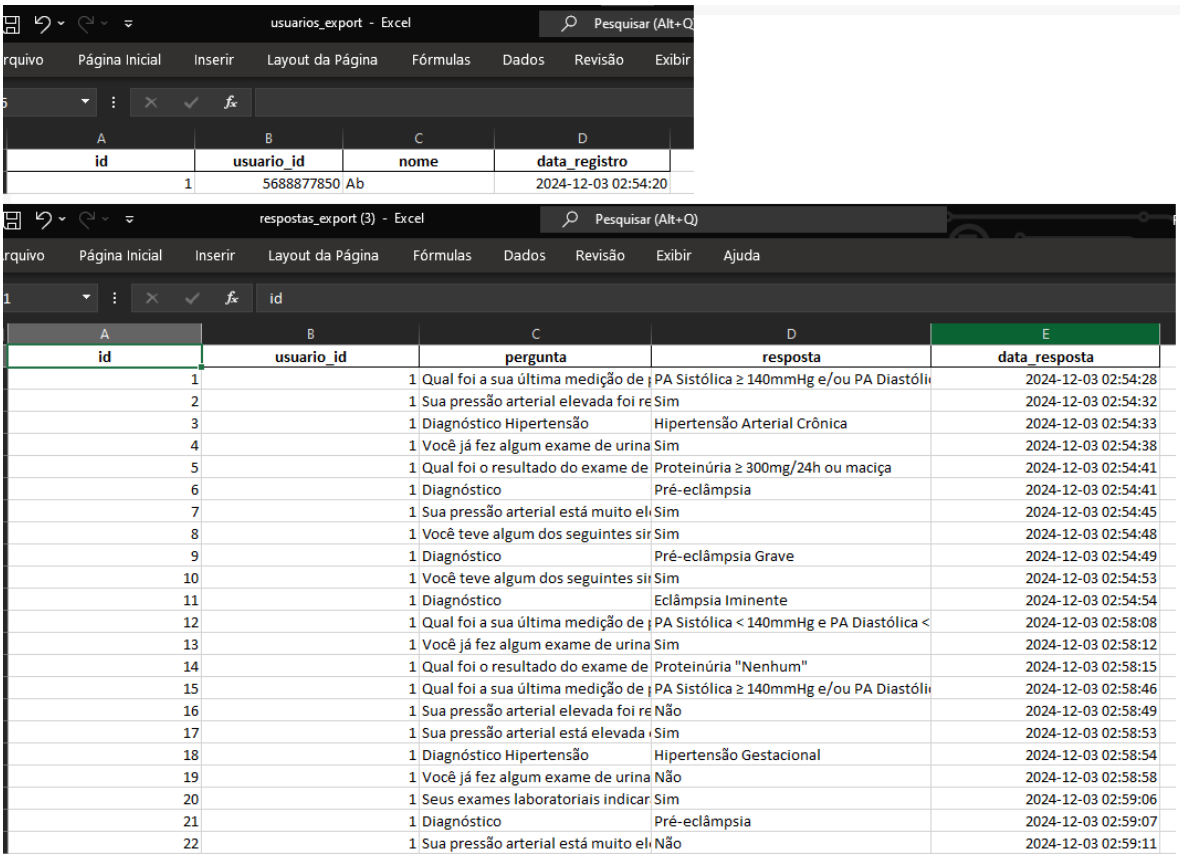


Fonte: Autoria Própria

Além disso, e ainda na Figura 32, no final da tela temos dois botões que fazem com que o médico exporte, no formato Excel, os dados armazenados na tabela respostas e usuários.

Como mostrado na Figura 34, o médico pode exportar os dados para o Excel, a fim de que ele consiga usar para fazer uma análise melhor ou até mesmo produzir relatórios.

Figura 34: Dados exportados no formato Excel



usuarios_export - Excel

id	usuario_id	nome	data_registro
1	5688877850	Ab	2024-12-03 02:54:20

respostas_export (3) - Excel

id	usuario_id	pergunta	resposta	data_resposta
1	1	Qual foi a sua última medição de	PA Sistólica ≥ 140mmHg e/ou PA Diastóli	2024-12-03 02:54:28
2	1	Sua pressão arterial elevada foi re	Sim	2024-12-03 02:54:32
3	1	Diagnóstico Hipertensão	Hipertensão Arterial Crônica	2024-12-03 02:54:33
4	1	Você já fez algum exame de urina	Sim	2024-12-03 02:54:38
5	1	Qual foi o resultado do exame de	Proteinúria ≥ 300mg/24h ou maciça	2024-12-03 02:54:41
6	1	Diagnóstico	Pré-eclâmpsia	2024-12-03 02:54:41
7	1	Sua pressão arterial está muito el	Sim	2024-12-03 02:54:45
8	1	Você teve algum dos seguintes sir	Sim	2024-12-03 02:54:48
9	1	Diagnóstico	Pré-eclâmpsia Grave	2024-12-03 02:54:49
10	1	Você teve algum dos seguintes sir	Sim	2024-12-03 02:54:53
11	1	Diagnóstico	Eclâmpsia Iminente	2024-12-03 02:54:54
12	1	Qual foi a sua última medição de	PA Sistólica < 140mmHg e PA Diastólica <	2024-12-03 02:58:08
13	1	Você já fez algum exame de urina	Sim	2024-12-03 02:58:12
14	1	Qual foi o resultado do exame de	Proteinúria "Nenhum"	2024-12-03 02:58:15
15	1	Qual foi a sua última medição de	PA Sistólica ≥ 140mmHg e/ou PA Diastóli	2024-12-03 02:58:46
16	1	Sua pressão arterial elevada foi re	Não	2024-12-03 02:58:49
17	1	Sua pressão arterial está elevada	Sim	2024-12-03 02:58:53
18	1	Diagnóstico Hipertensão	Hipertensão Gestacional	2024-12-03 02:58:54
19	1	Você já fez algum exame de urina	Não	2024-12-03 02:58:58
20	1	Seus exames laboratoriais indicar	Sim	2024-12-03 02:59:06
21	1	Diagnóstico	Pré-eclâmpsia	2024-12-03 02:59:07
22	1	Sua pressão arterial está muito el	Não	2024-12-03 02:59:11

Fonte: Autoria Própria

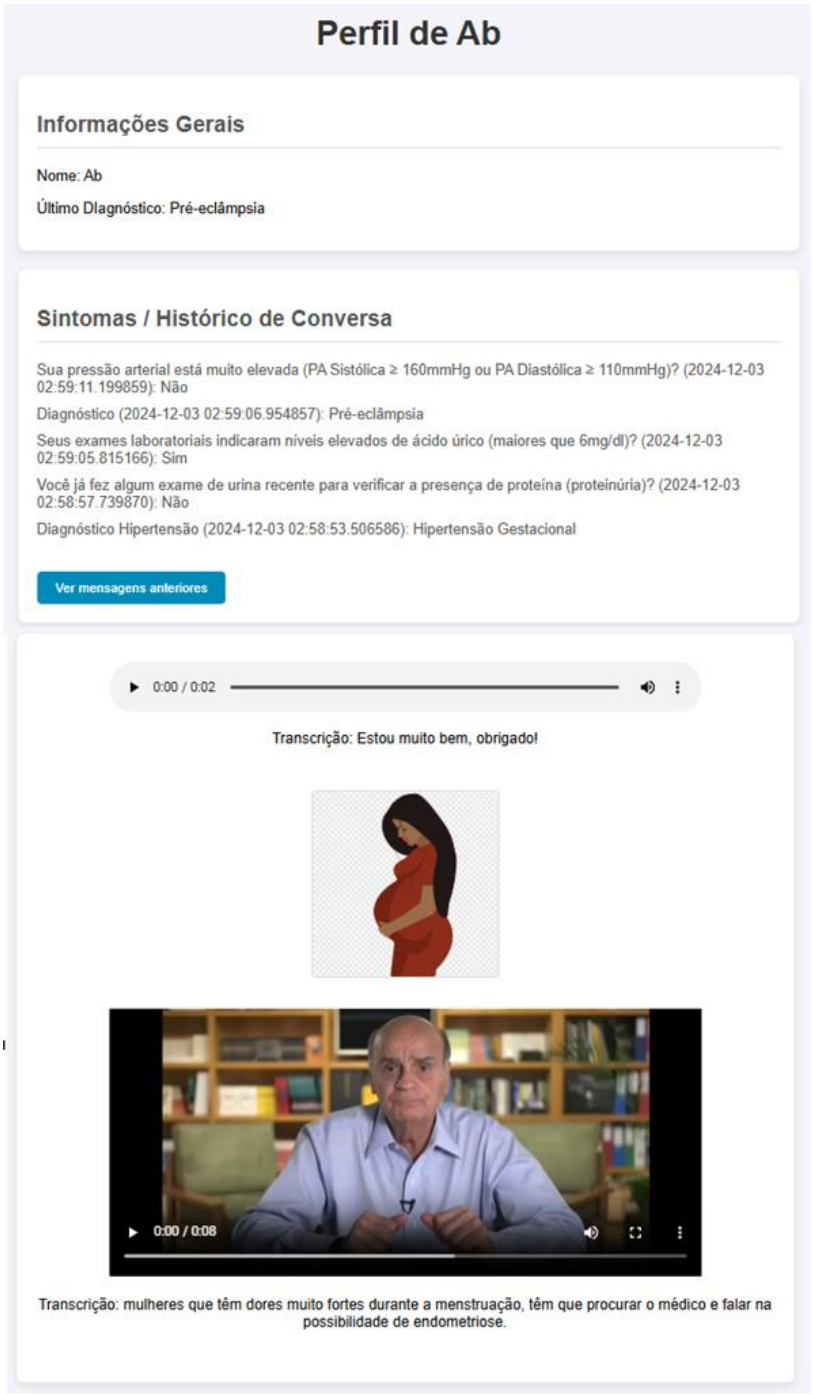
Agora mudando de aba e indo para a aba dos Pacientes, a Figura 35 mostra os pacientes salvos no banco de dados. Ação essa pela primeira interação do usuário com o *chatbot*, após ele ser salvo na tabela usuários (Figura 24).

Figura 35: Aba Pacientes

Fonte: Autoria Própria

A Figura 36 mostra o perfil do paciente, onde podemos ver as informações, seu histórico de conversas com o *chatbot*, e as mídias enviadas (Figura 26), bem como a transcrição do áudio em texto. Ação essa, realizada pela execução do código fonte dos Apêndice B, Figura 9 e 11.

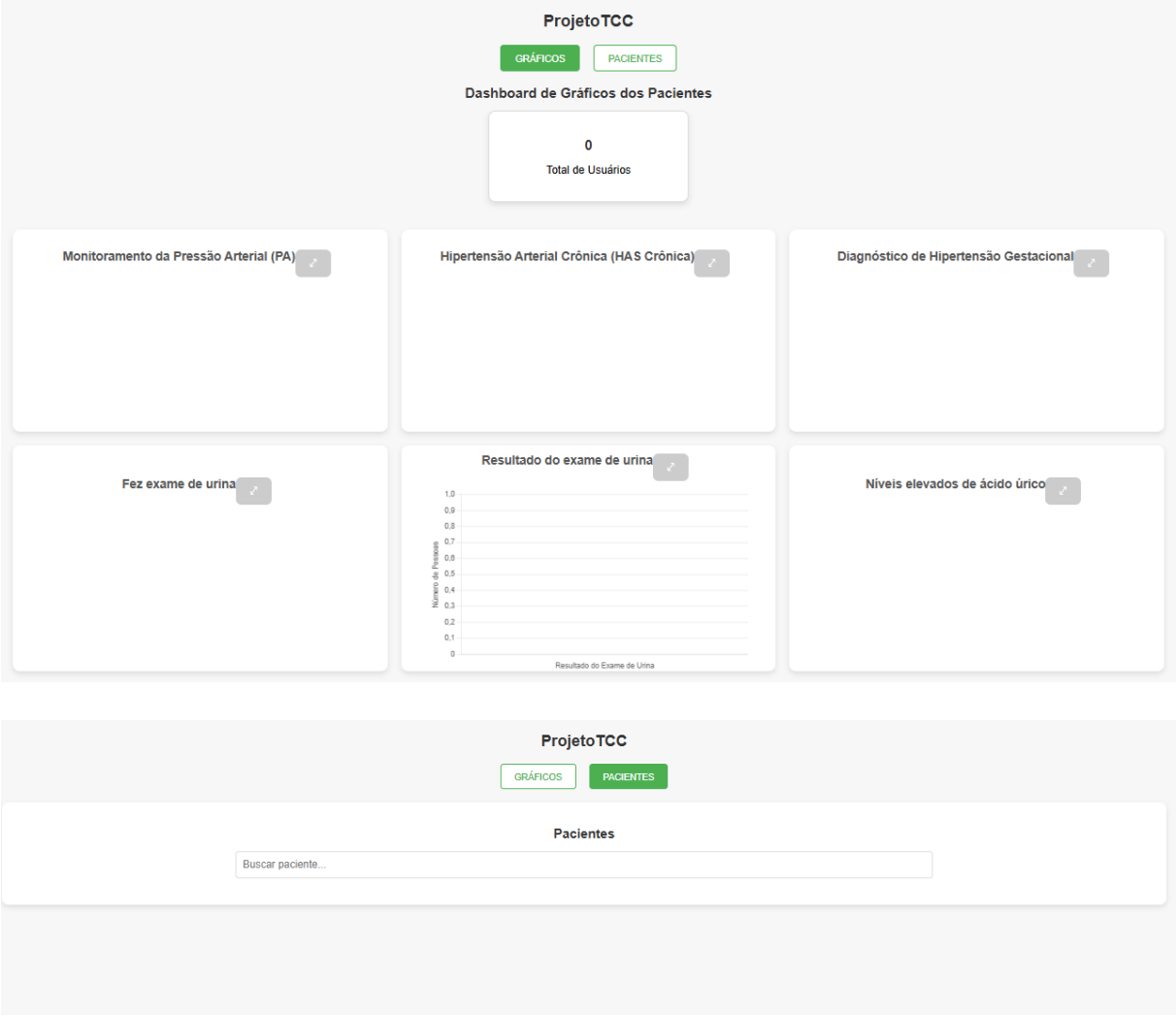
Figura 36: Perfil do paciente



Fonte: Autoria Própria

Por fim, a figura 37 mostra a interface gráfica do médico vazia de informações. Isso ocorreu em decorrência do pedido de apagar os dados realizado na Figura 27.

Figura 37: Interface gráfica vazia



Fonte: Autoria Própria.

4.3 Resultados

Podemos concluir que os testes confirmaram a viabilidade da hipótese principal: como customizações em plataformas de propósito geral, como o Telegram, pode ser uma solução tecnológica que satisfaça às restrições para aplicações de Telessaúde, mais especificamente para a prática de telemonitoramento na telemedicina. Onde os principais resultados incluem:

1. Validação Técnica: O sistema cumpriu todas as funcionalidades principais propostas, como envio de respostas e mídias, transcrição de áudio, armazenamento no banco de dados e exibição de gráficos na interface médica. Além da parte do *chatbot* de informar ao usuário paciente a possível condição de risco com base nas as respostas.
2. Análise de Vantagens: A escolha do Telegram apresentou vantagens claras, incluindo:
    a. Baixo custo de desenvolvimento: Por ser uma plataforma já existente, eliminou a necessidade de custear e construir um sistema do zero.
    b. Familiaridade e engajamento: Muitos usuários já estão habituados com plataformas de comunicação, como o Telegram, reduzindo custos de treinamento e aumentando a aceitação da solução.
    c. Acessibilidade: A plataforma é amplamente utilizada, tornando o sistema viável para diferentes perfis de pacientes.
3. Fluxo Completo Validado: O ciclo de interação do paciente com o *chatbot* em uma situação de telemonitoramento, o armazenamento no banco de dados e a consulta dos dados na interface foi executado com sucesso.

Porém, apesar da eficácia geral é necessário analisar os riscos e limitações que essa aplicação sofre, detalhados na seção seguinte:

4.4 Limitações e Análise de Riscos

Apesar do sucesso funcional do sistema, algumas limitações críticas foram identificadas que podem impactar a confiabilidade e a segurança do sistema, sendo necessário endereçá-las para garantir uma aplicação mais robusta e alinhada às normas e regulamentações de telemedicina, especialmente no contexto de uso do Telegram como plataforma principal.

4.4.1 Confiabilidade

- Latência e Escalabilidade: Durante os testes, não foram mensuradas métricas de latência ou a capacidade do sistema em situações de uso intenso, como múltiplos pacientes interagindo simultaneamente com o *chatbot* e o *backend*. Isso representa um risco, pois um aumento no número de usuários pode comprometer a estabilidade e a performance do sistema.

- Dependência do Telegram: Como uma plataforma de propósito geral, o Telegram não foi desenvolvido especificamente para aplicações médicas. Qualquer interrupção nos serviços do Telegram, mudanças em as políticas de uso ou limitações técnicas podem impactar diretamente a operação do sistema.

4.4.2 Privacidade e Segurança

A privacidade e a segurança dos dados representam os maiores desafios do sistema, considerando as seguintes questões:

1. Segurança do Telegram:
    - Chats regulares versus secretos: As mensagens enviadas para *chatbots* no Telegram não utilizam criptografia de ponta a ponta e são armazenadas em servidores do Telegram. Isso pode expor informações sensíveis de pacientes a riscos de interceptação ou vazamentos em caso de violações de segurança nos servidores do Telegram.
    - Responsabilidade do desenvolvedor: A segurança dos dados tratados pelo *chatbot* é transferida para o desenvolvedor, que deve implementar salvaguardas adicionais para atender às exigências de segurança e privacidade.
    - Conformidade com a LGPD: Embora o Telegram forneça uma base de segurança, o armazenamento temporário de dados sensíveis na plataforma pode não estar totalmente alinhado com os requisitos de minimização de dados e proteção reforçada exigidos pela LGPD.
2. Banco de Dados e *Backend*:
    - Criptografia: Embora tenham sido implementadas medidas como consultas parametrizadas, a proteção dos dados em trânsito e essas consultas precisam de monitoramento contínuo para evitar vulnerabilidades.
    - Auditoria e Monitoramento: Logs detalhados e auditorias foram configurados, mas, a ausência de integração com sistemas de detecção de intrusão (como Snort ou Suricata) limita a capacidade de identificar e responder a atividades maliciosas em tempo real.
3. Interface Gráfica:

- Validação e Autenticação: A autenticação de médicos e operadores ainda não inclui funcionalidades avançadas, como autenticação multifator, que poderiam fortalecer o controle de acesso.
- Riscos de XSS e CSRF*:* são vulnerabilidades comuns em aplicações web, que podem comprometer a segurança do sistema. O Script entre sites (em inglês, *Cross-Site Scripting* - XSS) ocorre quando um atacante injeta *scripts* maliciosos em uma aplicação, explorando falhas na validação ou escape de entradas de usuários, possibilitando a execução de código no navegador de outros usuários. Já o Falsificação de requisição entre sites (em inglês, *Cross-Site Request Forgery* - CSRF) ocorre quando uma aplicação web é induzida a executar ações indesejadas em nome de um usuário autenticado, explorando a sessão ativa. Para mitigar esses riscos, foram implementados *tokens* anti-CSRF, que garantem que cada requisição enviada seja legítima, e políticas de cabeçalhos de segurança. No entanto, a validação das entradas de usuários permanece crucial para evitar que esses ataques ocorram e comprometam o sistema.

5 CONCLUSÕES

5.1 Considerações Finais

Este trabalho buscou responder existência de uma solução tecnológica que satisfaça às restrições para aplicações de telessaúde mediadas por customizações em plataformas de propósito geral, como, por exemplo, o Telegram**.** A solução proposta demonstrou que por meio de testes de ponta a ponto do sistema, é possível validar a viabilidade de se customizar uma plataforma já existente, como o Telegram, para práticas de telemonitoramento médico, com foco no acompanhamento de gestantes em situação de risco.

A utilidade da abordagem foi confirmada pela implementação de um sistema funcional, acessível e economicamente viável. Utilizando o Telegram, o projeto aproveitou uma plataforma amplamente difundida, eliminando a necessidade de desenvolver uma infraestrutura de comunicação do zero. Isso trouxe benefícios como:

I. Engajamento do usuário: Por ser similar a ferramentas já utilizadas, como WhatsApp, o Telegram não requer treinamento significativo.
II. Custo reduzido: Evitou-se o desenvolvimento de uma plataforma do zero, tornando o sistema acessível a ambientes com recursos financeiros limitados.
III. Flexibilidade e rapidez no desenvolvimento: A API do Telegram possibilitou a criação de um *chatbot* funcional com integração ao banco de dados e interface gráfica em um curto período.

No entanto, a originalidade do trabalho reside na abordagem proposta: substituir a criação de uma nova plataforma por uma adaptação customizada de uma já existente, garantindo flexibilidade e acessibilidade enquanto se analisa criticamente os riscos de privacidade e segurança associados.

A complexidade do sistema também deve ser ressaltada. A integração entre diversos componentes – *chatbot*, banco de dados, *backend* e interface gráfica – exigiu planejamento detalhado, modelagem robusta e testes rigorosos. Além disso, funcionalidades adicionais, como transcrição de áudio, legendagem de vídeos e visualização do perfil do paciente, enriqueceram a aplicação e demonstraram a capacidade de expansão do sistema.

Em comparação com outros estudos citados na revisão bibliográfica, que frequentemente abordam a criação de plataformas de telemedicina do zero, ou

utilizam uma plataforma já existente de forma não customizada, ou de forma customizada, mas, que não propõem uma interface gráfica para o médico que está monitorando as interações dos pacientes. Com isso, esse trabalho apresentou uma alternativa pragmática a essas revisões bibliográficas. Assim, diferentemente de sistemas proprietários que exigem altos custos iniciais, esta abordagem utilizou tecnologias acessíveis para criar um sistema funcional, porém sem comprometer a utilidade.

5.2 Propostas para trabalhos futuros

Embora o sistema tenha atendido aos objetivos iniciais, há espaço para evolução em diversas áreas, incluindo:

1) Casos de Uso Adicionais:
    - Expandir os fluxos do *chatbot* para atender outras condições médicas, como diabetes ou hipertensão crônica, ampliando o alcance do sistema.
    - Personalizar perguntas com algoritmos adaptativos, ajustando o acompanhamento conforme a condição clínica do paciente.
2) Aprimoramento de Segurança:
    - Implementar autenticação multifator e controles de acesso baseados em permissões específicas, garantindo maior proteção de dados sensíveis.
    - Realizar auditorias de segurança para identificar vulnerabilidades na comunicação entre os componentes do sistema.
3) Desempenho e Escalabilidade:
    - Ampliar os testes para medir o desempenho do sistema em cenários de alto tráfego e latência, ajustando configurações para escalabilidade.
    - Testar o sistema em plataformas de hospedagem na nuvem para identificar possíveis melhorias em eficiência e custo.
4) Análise de Dados e IA:
    - Integrar algoritmos de NLP para melhorar a análise das respostas dos pacientes, em textos e aúdios, tornando o *chatbot* mais eficiente em identificar problemas críticos.

- Adaptar o *chatbot* para suportar múltiplos idiomas e formas de comunicação mais acessíveis, incluindo textos simplificados para pacientes com pouca familiaridade técnica.

5) Testes com Outras Plataformas:
   - Avaliar a viabilidade de outras plataformas, como Microsoft Teams ou Signal, que oferecem melhores garantias de segurança, mas são menos difundidas que o Telegram.

5.3 Discussões

Apesar do sucesso do sistema, algumas questões fundamentais emergiram durante o desenvolvimento e testes:

- Falha em Utilizar o WhatsApp: A negativa de acesso à API do WhatsApp limitou a capacidade de personalização, levando à escolha do Telegram. Provavelmente isso esteja relacionado ao tipo de produto que a conta irá vender tendo acesso ao Whatsapp API, fazendo com que este trabalho não conseguisse atender aos requisitos para obter licença na conta Meta Developer.
- Limitações de Privacidade no Telegram: Embora o Telegram tenha se mostrado flexível e funcional, os chats regulares não oferecem criptografia ponta a ponta, criando riscos de confidencialidade. O armazenamento de mensagens nos servidores do Telegram pode expor dados sensíveis a vulnerabilidades ou uso indevido.
- Github: Este trabalho está disponível no github, com as devidas explicações e passa a passo de como executar. Basta acessar https://github.com/unlimitedabe/ProjetoTCC para quem quiser.

REFERÊNCIAS

BUSINESS, Meta. URGENTE: Seu acesso à publicidade foi restringido. Mensagem recebida por <advertise-noreply@support.facebook.com> em 20 mai. 2024.

CAFÓRIO, Antonio. et al. **HINT project: a BPM teleconsultation and telemonitoring platform**. In: ICIST 2020, Lecce, Itália. Anais [...]. Nova Iorque: ACM, 2020. p. 8.

CONSELHO FEDERAL DE MEDICINA. LGPD**: a Lei Geral de Proteção de Dados Pessoais e a atuação do profissional da medicina**. Brasília: CFM, 2022. 31 p. Disponível em: https://portal.cfm.org.br/bb_publicacoes/. Acesso em: 12 jun. 2024.

CONSELHO FEDERAL DE MEDICINA. **Resolução CFM nº 2.314, de 20 de abril de 2022**. Define e regulamenta a telemedicina como forma de serviços médicos mediados por tecnologias de comunicação. Diário Oficial da União: Seção 1, Brasília, DF, p. 227, 5 maio 2022. Disponível em: https://sistemas.cfm.org.br/normas/visualizar/resolucoes/BR/2022/2314. Acesso em: 26 nov. 2024.

DE NARDIS, Luca. et al. **Internet of Things Platforms for Academic Research and Development: A Critical Review**. Applied Sciences, v. 12, n. 2172, 2022.

DR RAGHU. et al. **Ultrasonic Sensor Based Door Security Camera with Wireless Data Transfer in Telegram Bot Using WIFI**. International Journal of Advanced Research in Computer Science and Software Engineering, vol. 13, no. 5, p. 56–61, 2023.

FACEBOOK. Página de teste de API no Meta for Developers. Comunicação pessoal. Acessado em: 05 mai. 2024.

FRÖHLICH, Holger. et al. **From hype to reality: data science enabling personalized medicine**. BMC Medicine, v. 16, n. 150, 2018.

FONTELLES, Mauro José; SIMÕES, Marilda Garcia; FARIAS, Samantha Hasegawa; FONTELLES, Renata Garcia Simões. **Metodologia da pesquisa científica: diretrizes para a elaboração de um protocolo de pesquisa**. Belém: Universidade da Amazônia, 2009.

GIL, Antônio Carlos. **Como Elaborar Projetos de Pesquisa**. 6. ed. São Paulo: Editora Atlas Ltda., 2017.

GUNAWAN, Teddy Surya. et al. **Development of Intelligent Telegram Chatbot Using Natural Language Processing.** In**:** International Conference on Wireless and Telematics (ICWT), 7., 2021. IEEE, 2021. Disponível em: https://ieeexplore.ieee.org. Acesso em: 07 out. 2024.

Harshithan. et al. **Medico Bot: AI-based Telemedicine System with Disease Diagnosis and Expert Recommendation System with Blockchain Implementation**. Journal of Computational and Applied Research in Telemedicine, vol. 7, no. 4, p. 120–135, 2024.

JEONG, Ji Yun. et al. **Smart Care Based on Telemonitoring and Telemedicine for Type 2 Diabetes Care: Multi-center Randomized Controlled Trial**. Telemedicine and e-Health, v. 24, n. 8, p. 1-10, ago. 2018.

LEÃO, Camila Furtado. et al. **O uso do WhatsApp na relação médico-paciente**. Revista Bioética, v. 26, n. 3, p. 412-419, 2018.

LIMA, Gabriel Fellipe Felix. **Desenvolvimento de um software para apoio ao diagnóstico da gestação de alto risco**. Programa de Iniciação Científica: PIBITI/CNPq. Orientador: Talles Marcelo Gonçalves de Andrade Barbosa. Goiânia: Pontifícia Universidade Católica de Goiás, 2023. Disponível em: https://sistemas.pucgoias.edu.br/sigep/espelhoIniciacaoCientifica/show/22534. Acesso em: 09 out. 2024.

NASCIMENTO, Artur Santos. et al. **Home Care utilization in Sergipe-Brazil: applications and mobile devices**. In: Euro American Conference on Telematics and Information Systems (EATIS ’18), 12–15 nov. 2018, Fortaleza, Brasil. **Anais** [...]. Nova Iorque: ACM, 2018.

ORGANIZAÇÃO MUNDIAL DA SAÚDE (OMS). *Consolidated telemedicine implementation guide*. Genebra: Organização Mundial da Saúde, 2022. Disponível em: https://www.who.int/publications/i/item/9789240059184#:~:text=Telemedicine%20%E2%80%93%20which%20involves%20the%20delivery,and%20extend%20coverage%20of%20services. . Acesso em: 19 mai. 2024.

STOLTZFUS, Mason. et al. **The role of telemedicine in healthcare: an overview and update**. The Egyptian Journal of Internal Medicine, v. 35, n. 49, 2023.

RAJ, Divya; SRIKANTH, T K. **Assisted Telemedicine Model for Rural Healthcare Ecosystem**. In: ACM Web Science Conference 2021 (WebSci '21 Companion), 21-25 jun. 2021, Virtual Event, United Kingdom. **Anais** [...]. Nova Iorque: ACM, 2021.

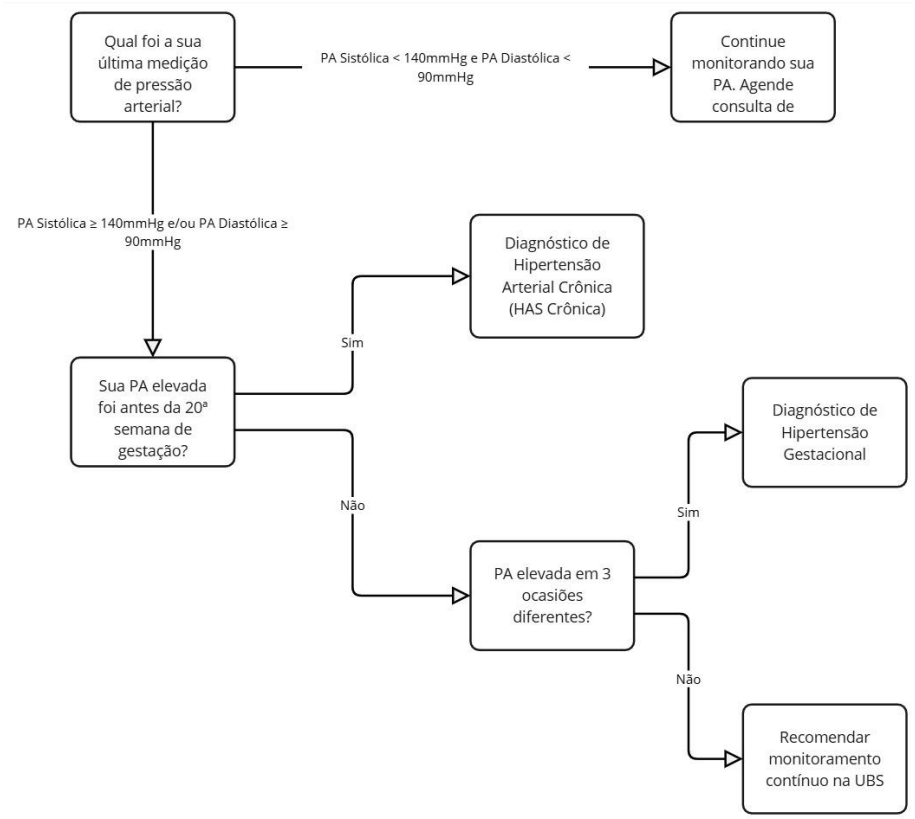
TELEGRAM. QR Code do bot TCC1BOT. Comunicação pessoal, 01 de jun. 2024.

WAZLAWICK, Raul. **Metodologia de Pesquisa para Ciência da Computação**. 2. ed. Rio de Janeiro: Elsevier, 2014.

APÊNDICE A

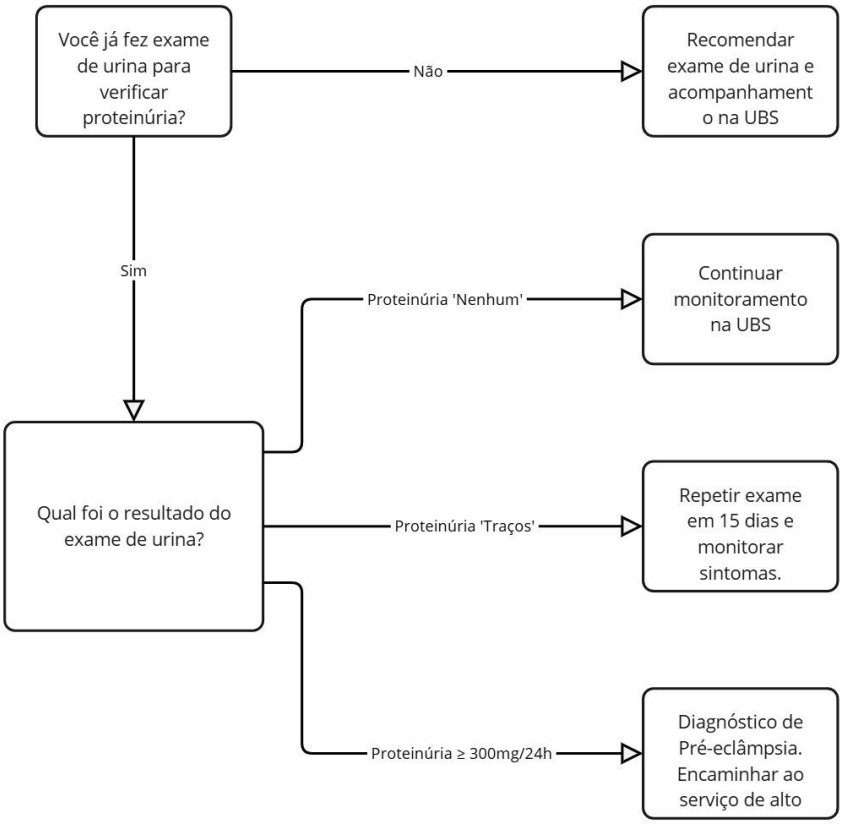
FLUXO LÓGICO PARA O MONITORAMENTO DO CHATBOT

Figura 1 – Fluxo lógico para identificação de Hipertensão Arterial Crônica ou Hipertensão Gestacional



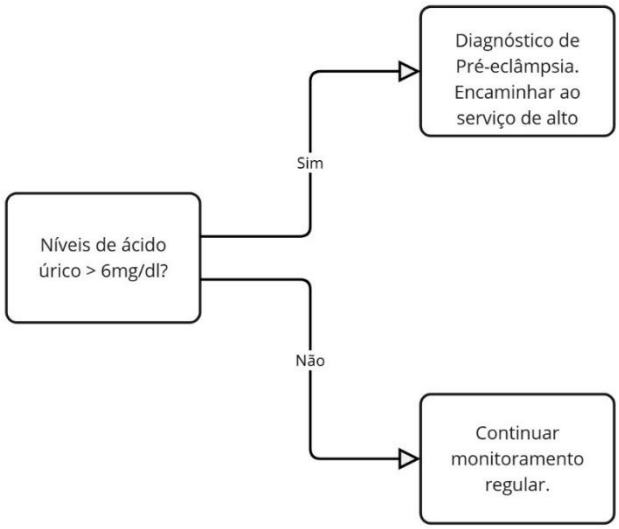
Fonte Modificada: Lima, 2023

Figura 2 – Fluxo lógico para identificação de Pré-eclâmpsia por meio do exame de urina



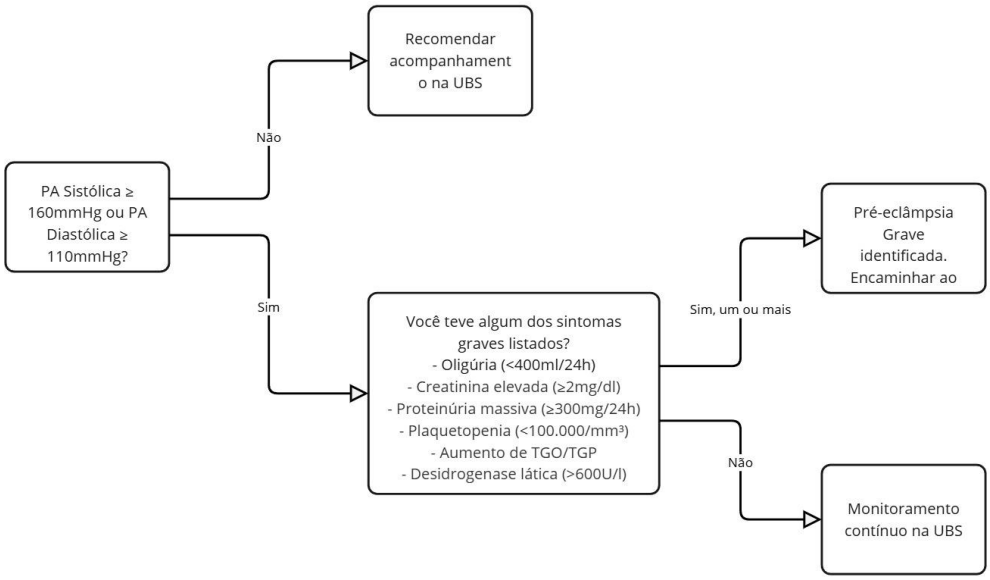
Fonte Modificada: Lima, 2023

Figura 3 – Fluxo lógico para identificação de Pré-eclâmpsia por meio do ácido úrico



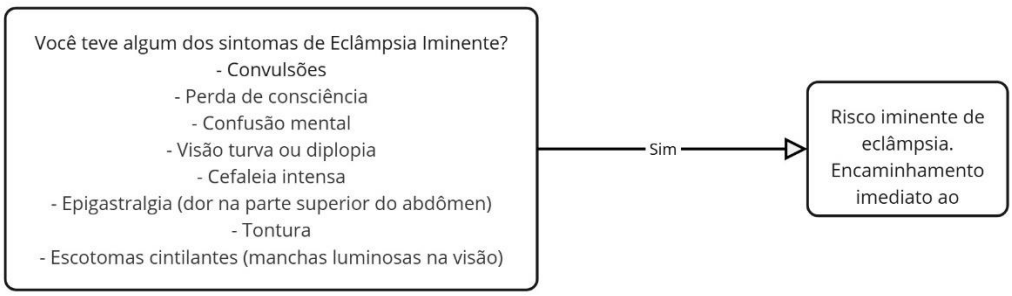
Fonte Modificada: Lima, 2023

Figura 4 – Fluxo lógico para identificação de Pré-eclâmpsia Grave



Fonte Modificada: Lima, 2023

Figura 5 – Fluxo lógico para identificação de risco Iminente de eclâmpsia.



Fonte Modificada: Lima, 2023

APÊNDICE B
CÓDIGO FONTE

Figura 1 – Código fonte api.py.

```
from flask import Flask, render_template, jsonify, session, request,
send_from_directory, abort, send_file
import secrets
import os # Biblioteca para operações relacionadas ao sistema de
arquivos
# Conexão com o banco de dados definida externamente
from database import get_db_connection
import numpy as np  # Biblioteca NumPy para manipulação de dados
numéricos
import urllib.parse  # Para decodificar strings de URL
from flask_talisman import Talisman
import pandas as pd

# Definir o caminho correto para as pastas de templates e arquivos
estáticos do frontend
template_dir = os.path.abspath(os.path.join(
    os.path.dirname(__file__), '..', 'frontend', 'templates'))
static_dir = os.path.abspath(os.path.join(
    os.path.dirname(__file__), '..', 'frontend', 'static'))
media_dir = os.path.abspath(os.path.join(
    os.path.dirname(__file__), '..', 'media'))

# Inicializa o Flask com o caminho das pastas de templates e arquivos
estáticos
app = Flask(__name__, template_folder=template_dir,
static_folder=static_dir)

# Aplica configurações de cabeçalhos de segurança
Talisman(app, content_security_policy={
    # Recursos do próprio domínio
    'default-src': "'self'",
    # Permitir scripts do domínio e CDN
    'script-src': "'self' 'unsafe-inline' https://cdn.jsdelivr.net",
    'style-src': "'self' 'unsafe-inline'",              # Permitir
estilos inline
    # Permitir imagens do domínio e base64
    'img-src': "'self' data:",
    # Permitir conexões com APIs externas
    'connect-src': "'self' https://api.exemplo.com"
})
```

```python
@app.before_request
def csrf_protect():
    if request.method == "POST":
        token = session.get("csrf_token")
        request_token = request.headers.get("X-CSRF-Token")
        if not token or not request_token or token != request_token:
            abort(403)

@app.route('/form', methods=['GET', 'POST'])
def form():
    if request.method == 'GET':
        # Gera um token CSRF para a sessão
        session['csrf_token'] = secrets.token_hex(16)
        return render_template('form.html',
csrf_token=session['csrf_token'])
    # Processa requisição POST
    return 'Formulário enviado com sucesso'

@app.route('/processar_dados', methods=['POST'])
def processar_dados():
    user_input = request.json.get('input')
    if not user_input.isalnum():
        return jsonify({'error': 'Entrada inválida'}), 400
    # Processar dado válido
    return jsonify({'message': 'Entrada processada com sucesso'})

# Função para buscar respostas no banco de dados e formatá-las para
gráficos
def obter_dados_grafico(pergunta):
    with get_db_connection() as connection:
        cursor = connection.cursor()
        query = """
        SELECT resposta, COUNT(*) FROM respostas
        WHERE pergunta = %s
        GROUP BY resposta
        """
        cursor.execute(query, (pergunta,))
        results = cursor.fetchall()
        cursor.close()

        # Extrai as respostas e as contagens para construir o gráfico
        labels = [row[0] for row in results]
        data = [row[1] for row in results]

        return {'labels': labels, 'data': data}
```

```python
# Função para buscar mídias e transcrições do banco de dados
def obter_midias():
    with get_db_connection() as connection:
        cursor = connection.cursor()
        query = """
        SELECT tipo_midia, caminho_arquivo, transcricao_audio,
transcricao_video
        FROM midias
        """
        cursor.execute(query)
        results = cursor.fetchall()
        cursor.close()

        midias = []
        for row in results:
            # Ajusta o caminho para relativo
            caminho_relativo = os.path.relpath(row[1], media_dir)
            midia = {
                'tipo': row[0],  # Tipo da mídia (áudio, imagem, vídeo)
                'caminho': caminho_relativo,  # Caminho relativo do
arquivo
                # Transcrição de áudio, se disponível
                'transcricao_audio': row[2],
                # Transcrição de vídeo, se disponível
                'transcricao_video': row[3]
            }
            midias.append(midia)

        return midias
# Rota para fornecer dados de gráfico com base em uma pergunta
@app.route('/dados_grafico')
def dados_grafico():
    pergunta = request.args.get('pergunta')  # Obtém a pergunta da
query string
    # Usa a função genérica para obter dados
    dados = obter_dados_grafico(pergunta)
    return jsonify(dados)
```

```python
# Rota para renderizar um gráfico expandido com nomes de usuários associados às respostas
@app.route('/grafico_expandido')
def grafico_expandido():
    pergunta = request.args.get('pergunta')
    with get_db_connection() as connection:
        cursor = connection.cursor()
        query = """
        SELECT resposta, nome FROM respostas
        JOIN usuarios ON respostas.usuario_id = usuarios.id
        WHERE pergunta = %s
        """
        cursor.execute(query, (pergunta,))
        results = cursor.fetchall()
        cursor.close()

        labels = {}
        for resposta, nome in results:
            if resposta not in labels:
                labels[resposta] = []
            labels[resposta].append(nome)

        return render_template('grafico_expandido.html', pergunta=pergunta, labels=labels)

# Rota para listar os pacientes e retornar em formato JSON
@app.route('/api/pacientes')
def listar_pacientes():
    with get_db_connection() as connection:
        cursor = connection.cursor()
        query = "SELECT id, nome FROM usuarios"
        cursor.execute(query)  # Sem parâmetros
        patients = cursor.fetchall()
        cursor.close()
    return jsonify([{'id': p[0], 'nome': p[1]} for p in patients])
```

```python
# Rota para renderizar o perfil de um paciente específico
@app.route('/paciente/<int:id>')
def perfil_paciente(id):
    with get_db_connection() as connection:
        cursor = connection.cursor()
        # Buscar informações do paciente
        query_patient = """
        SELECT nome, usuario_id, data_registro, id
        FROM usuarios
        WHERE id = %s
        """
        cursor.execute(query_patient, (id,))
        patient = cursor.fetchone()
        # Buscar as últimas 5 mensagens do paciente
        query_messages = """
        SELECT * FROM respostas
        WHERE usuario_id = %s
        ORDER BY data_resposta DESC
        LIMIT 5
        """
        cursor.execute(query_messages, (id,))
        ultimas_mensagens = cursor.fetchall()
        # Buscar a última conclusão de diagnóstico do paciente
        query_diagnosis = """
        SELECT resposta
        FROM respostas
        WHERE usuario_id = %s AND pergunta = 'Diagnóstico'
        ORDER BY data_resposta DESC
        LIMIT 1
        """
        cursor.execute(query_diagnosis, (id,))
        ultima_conclusao = cursor.fetchone()
        cursor.close()
    return render_template(
        'perfil_pacientes.html',
        patient=patient,
        ultimas_mensagens=ultimas_mensagens,
        ultima_conclusao=ultima_conclusao[0] if ultima_conclusao else
None
    )
```

```python
# Rota para buscar todas as mensagens de um paciente
@app.route('/paciente/mensagens-anteriores/<int:usuario_id>')
def mensagens_anteriores(usuario_id):
    with get_db_connection() as connection:
        cursor = connection.cursor()
        query = """
        SELECT * FROM respostas
        WHERE usuario_id = %s
        ORDER BY data_resposta DESC
        """
        cursor.execute(query, (usuario_id,))
        mensagens = cursor.fetchall()
        cursor.close()

    return jsonify(mensagens)

# Rota para servir arquivos de mídia
@app.route('/media/<path:filename>')
def media(filename):
    return send_from_directory(media_dir, filename)

# Rota para fornecer as mídias em formato JSON
@app.route('/obter_midias')
def obter_midias_route():
    midias = obter_midias()
    return jsonify(midias)

# Rota para servir o arquivo index_graphic.html
@app.route('/graphics')
def graphics():
    return render_template('index_graphic.html')

# Rota para contar e retornar o número total de usuários
@app.route('/total_usuarios')
def total_usuarios():
    with get_db_connection() as conn:
        if conn:
            with conn.cursor() as cursor:
                cursor.execute("SELECT COUNT(*) FROM usuarios")
                total_usuarios = cursor.fetchone()[0]

    return jsonify({'totalUsuarios': total_usuarios})
```

```python
# Rota para retornar os dados da tabela resposta
@app.route('/export/respostas_excel', methods=['GET'])
def export_excel():
    with get_db_connection() as connection:
        if connection:
            try:
                # Query para pegar os dados da tabela respostas
                query = "SELECT * FROM respostas"
                df = pd.read_sql_query(query, connection)

                # Cria um arquivo Excel temporário
                file_path = "respostas_export.xlsx"
                df.to_excel(file_path, index=False)

                # Envia o arquivo para download
                return send_file(file_path, as_attachment=True)
            except Exception as e:
                return {"error": f"Erro ao exportar: {e}"}, 500

# Rota para retornar os dados da tabela usuarios
@app.route('/export/usuarios_excel', methods=['GET'])
def export_usuarios():
    with get_db_connection() as connection:
        if connection:
            try:
                # Query para pegar os dados da tabela usuarios
                query = "SELECT * FROM usuarios"
                df = pd.read_sql_query(query, connection)

                # Cria um arquivo Excel temporário
                file_path = "usuarios_export.xlsx"
                df.to_excel(file_path, index=False)

                # Envia o arquivo para download
                return send_file(file_path, as_attachment=True)
            except Exception as e:
                return {"error": f"Erro ao exportar: {e}"}, 500

# Rota principal que renderiza a página index.html
@app.route('/')
def index():
    return render_template('index.html')

# Inicializa o servidor Flask na porta 5001 para desenvolvimento
if __name__ == '__main__':
    app.run(debug=True, port=5001)
```

Fonte: Autoria Própria

Figura 2 – Código fonte app.py.

```
import subprocess  # Biblioteca para criar e gerenciar subprocessos
import os  # Biblioteca para operações relacionadas ao sistema de
arquivos

# Defina o caminho para os seus scripts
bot_script = os.path.join("bot.py")  # Script principal do bot
# Script para transcrição de áudio
transcricao_script = os.path.join("transcricao.py")
# Script para legendagem de vídeos
legendagem_script = os.path.join("legendagem.py")
# Script para API (gerenciamento de dados e rotas)
api_script = os.path.join("api.py")

def run_script(script_path):
    """Função para rodar um script em um subprocesso."""
    # Cria um subprocesso para executar o script em Python
    return subprocess.Popen(["python", script_path])

def main():
    print("Iniciando todos os processos...")
    # Inicia bot.py
    bot_process = run_script(bot_script)
    print("Bot rodando...")
    # Inicia transcricao.py
    transcricao_process = run_script(transcricao_script)
    print("Transcrição rodando...")
    # Inicia legendagem.py
    legendagem_process = run_script(legendagem_script)
    print("Legendagem rodando...")
    # Inicia api.py
    api_process = run_script(api_script)
    print("API rodando...")
    try:
        # Mantém o aplicativo rodando enquanto os subprocessos estão
ativos
        bot_process.wait()
        transcricao_process.wait()
        legendagem_process.wait()
        api_process.wait()
    except KeyboardInterrupt:
        # Se o programa for interrompido com Ctrl+C, encerra todos os
subprocessos
        print("\nEncerrando todos os processos...")
        bot_process.terminate()
        transcricao_process.terminate()
        legendagem_process.terminate()
        api_process.terminate()

if __name__ == '__main__':
    main()  # Executa a função main se o script for executado
diretamente
```

Fonte: Autoria Própria

Figura 3 – Código fonte bot.py.

```
import os  # Biblioteca para operações de sistema de arquivos
from telegram import Update  # Para interações com o Telegram
# Para configurar o bot e lidar com eventos
from telegram.ext import ApplicationBuilder, CommandHandler,
MessageHandler, ContextTypes, filters
# Funções de conexão com o banco de dados
from database import connect_db, close_connection
import atexit  # Para registrar funções a serem executadas na saída do
programa
# Função para obter conexão com o banco de dados
from database import get_db_connection
import time  # Para operações de pausa e gerenciamento de tempo
# Configuração de requisições HTTP com tempo limite
from telegram.request import HTTPXRequest
import unidecode  # Biblioteca para remover acentos
import asyncio  # Certifique-se de importar asyncio no início do
arquivo

# Definindo diretórios para salvar mídias
MEDIA_DIR = os.path.join(os.path.dirname(
    os.path.dirname(os.path.abspath(__file__))), "media")
AUDIO_DIR = os.path.join(MEDIA_DIR, "audio")
VIDEO_DIR = os.path.join(MEDIA_DIR, "video")
IMAGES_DIR = os.path.join(MEDIA_DIR, "images")

# Criar diretórios de mídia se não existirem
os.makedirs(AUDIO_DIR, exist_ok=True)
os.makedirs(VIDEO_DIR, exist_ok=True)
os.makedirs(IMAGES DIR, exist ok=True)

# Definição das perguntas e respostas do fluxo de conversa
# Perguntas e respostas mapeadas por IDs
# Estrutura de perguntas, opções e o próximo passo baseado na resposta
questions = {
    '0': {
        'text': "Escolha uma das opções abaixo (digite o número da
opção):",
        'options': {
            '1': 'Monitoramento de risco gestacional',
            '2': 'Enviar um áudio',
            '3': 'Enviar uma foto',
            '4': 'Enviar um vídeo',
            '5': 'Apagar meus dados'
```

```
        },
        'next': {
            '1': '1',  # Segue para o monitoramento
            '2': 'handle_audio',  # Solicita envio de áudio
            '3': 'handle_photo',  # Solicita envio de foto
            '4': 'handle_video',  # Solicita envio de vídeo
            '5': 'confirm_delete'  # Confirmação para apagar dados
        }
    },
    # Pergunta de confirmação para apagar dados
    'confirm_delete': {
        'text': "Tem certeza de que deseja apagar todos os seus
dados?",
        'options': {
            'sim': 'Sim, eu quero apagar meus dados salvos no sistema',
            'nao': 'Não, eu não quero apagar meus dados salvos no
sistema'
        },
        'next': {
            'sim': 'delete_data',
            'nao': 'end'  # Finaliza sem apagar
        }
    },
    '1': {
        'text': "Qual foi a sua última medição de pressão arterial?",
        'options': {
            '1': 'PA Sistólica ≥ 140mmHg e/ou PA Diastólica ≥ 90mmHg',
            '2': 'PA Sistólica < 140mmHg e PA Diastólica < 90mmHg'
        },

        'next': {
            '1': '2',  # Segue para a próxima pergunta sobre
hipertensão
            '2': 'end_monitoring'  # Instruções de monitoramento e fim
da interação
        }
    },
    '2': {
        'text': "Sua pressão arterial elevada foi registrada antes da
20ª semana de gestação?",
        'options': {
            'sim': 'Sim, foi antes da 20ª semana.',
            'nao': 'Não, foi após a 20ª semana.'
        },
```

```
        'next': {
            'sim': 'diagnostico_has_cronica',
            'nao': '3'
        }
    },
    '3': {
        'text': "Sua pressão arterial está elevada em 3 ocasiões
diferentes?",
        'options': {
            'sim': 'Sim, em 3 ocasiões diferentes.',
            'nao': 'Não, menos de 3 ocasiões.'
        },
        'next': {
            'sim': 'diagnostico_hipertensao_gestacional',
            'nao': 'end_monitoring'
        }
    },
    '4': {
        'text': "Você já fez algum exame de urina recente para
verificar a presença de proteína (proteinúria)?",
        'options': {
            'sim': 'Sim, já fiz o exame de proteinúria.',
            'nao': 'Não, ainda não fiz o exame de proteinúria.'
        },
        'next': {
            'sim': '5',  # Se sim, segue para o resultado do exame
            'nao': 'recommend_exam'  # Se não, recomenda exame e
acompanhamento
    }
},
'5': {
    'text': "Qual foi o resultado do exame de urina?",
    'options': {
        '1': 'Proteinúria ≥ 300mg/24h ou maciça',
        '2': 'Proteinúria "Traços"',
        '3': 'Proteinúria "Nenhum"'
    },
    'next': {
        '1': 'diagnostico_pre_eclampsia',
        '2': 'repeat_exam',
        '3': 'continue_monitoring'
    }
},
```

```
    '6': {
        'text': "Seus exames laboratoriais indicaram níveis elevados de
ácido úrico (maiores que 6mg/dl)?",
        'options': {
            'sim': 'Sim',
            'nao': 'Não'
        },
        'next': {
            'sim': 'diagnostico_pre_eclampsia',  # Encaminhar ao
serviço de alto risco
            'nao': 'continue_monitoring'  # Continuar monitoramento
regular
        }
    },
    '7': {
        'text': "Sua pressão arterial está muito elevada (PA Sistólica ≥
160mmHg ou PA Diastólica ≥ 110mmHg)?",
        'options': {
            'sim': 'Sim, a PA está muito elevada.',
            'nao': 'Não, está abaixo desse valor.'
        },
        'next': {
            'sim': '8',  # Segue para sintomas de pré-eclâmpsia grave
            'nao': 'continue_monitoring'
        }
    },

    '8': {
        'text': "Você teve algum dos seguintes sintomas ou resultados
em exames?\n(Oligúria, Creatinina elevada, Proteinúria massiva,
Plaquetopenia, Aumento de TGO/TGP, Desidrogenase lática elevada)",
        'options': {
            'sim': 'Sim, um ou mais desses sintomas.',
            'nao': 'Não, nenhum desses sintomas.'
        },
        'next': {
            'sim': 'diagnostico_pre_eclampsia_grave',  # Diagnóstico de
pré-eclâmpsia grave
            'nao': 'continue_monitoring'
        }
    },
    '9': {
        'text': "Você teve algum dos seguintes sintomas recentemente?
(Convulsões, Perda de consciência, Confusão mental, Visão turva ou
```

```
diplopia, Cefaleia intensa, Epigastralgia, Tontura, Escotomas
cintilantes)",
        'options': {
            'sim': 'Sim, um ou mais desses sintomas.',
            'nao': 'Não, nenhum desses sintomas.'
        },
        'next': {
            'sim': 'diagnostico_eclampsia_iminente',
            'nao': 'continue_monitoring'
        }
    }
}

# Função para enviar a pergunta ao usuário com base no ID
async def enviar_pergunta(update: Update, context:
ContextTypes.DEFAULT_TYPE, question_id: str) -> None:
    question = questions[question_id]
    # Verifica se a pergunta é do tipo Sim/Não
    # IDs das perguntas que são Sim/Não
    if question_id in ["confirm_delete", "2", "3", "4", "6", "7", "8",
"9"]:
        await update.message.reply_text(f"{question['text']}\n(Responda
com Sim/Não)")
    else:
        options_text = "\n".join(
            [f"{key}. {text}" for key, text in
question['options'].items()])
        await
update.message.reply_text(f"{question['text']}\n{options_text}")

# Função para lidar com as respostas do usuário e salvar o texto
correspondente
# Processa a resposta com base no fluxo de perguntas
# Salva resposta no banco e determina o próximo passo
async def handle_message(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    user_id = update.message.from_user.id
    user_response = update.message.text.strip().lower()
    user_response = unidecode.unidecode(user_response)  # Remove
acentos

current_question_id = context.user_data.get('question_id', '0')
current_question = questions.get(current_question_id)
```

```
    if current_question_id == '0':  # Mensagem inicial
        if user_response in ['1', '2', '3', '4', '5']:
            next_step = current_question['next'][user_response]
            if next_step == 'handle_audio':
                # Solicita envio do áudio
                await update.message.reply_text("Por favor, grave e
envie o seu áudio.")
                # Marca como esperando áudio
                context.user_data['awaiting_audio'] = True
                return
            elif next_step == 'handle_photo':
                await update.message.reply_text("Por favor, envie uma
foto.")
                context.user_data['awaiting_photo'] = True
                return
            elif next_step == 'handle_video':
                await update.message.reply_text("Por favor, envie um
vídeo.")
                context.user_data['awaiting_video'] = True
                return
            else:
                context.user_data['question_id'] = next_step
                await enviar_pergunta(update, context, next_step)
                return
        else:
            await update.message.reply_text("Opção inválida. Por favor,
digite '1', '2', '3', '4' ou '5'.")
            return

    if context.user_data.get('awaiting_audio'):
        await update.message.reply_text("Por favor, envie um arquivo de
áudio.")
        return

if context.user_data.get('awaiting_photo'):
    await update.message.reply_text("Por favor, envie uma foto.")
    return

    if context.user_data.get('awaiting_video'):
        await update.message.reply_text("Por favor, envie um arquivo de
vídeo.")
        return
```

```
    if current_question_id == 'confirm_delete':  # Confirmação de
exclusão
        if user_response in ['sim', 'nao']:
            if user_response == 'sim':
                await deletar_dados_usuario(user_id)
                await update.message.reply_text("Seus dados foram
apagados com sucesso.")
                context.user_data.clear()
            else:
                await update.message.reply_text("Operação cancelada.
Estamos à disposição caso precise!")
                context.user_data.clear()
            return
        else:
            await update.message.reply_text("Resposta inválida. Por
favor, digite 'Sim' ou 'Não'.")
            return
    # IDs das perguntas de Sim/Não
    if current_question_id in ["confirm_delete", "2", "3", "4", "6",
"7", "8", "9"]:
        if user_response in ["sim", "nao"]:
            # Salva "Sim" ou "Não" no banco de dados
            response_text = "Sim" if user_response == "sim" else "Não"
            salvar_resposta(user_id, current_question['text'],
response_text)
            next_step = current_question['next'][user_response]
        else:
            await update.message.reply_text("Resposta inválida. Por
favor, responda com 'Sim' ou 'Não'.")

    else:
        # Pergunta não é Sim/Não; verifica se a resposta é válida
(existe nas opções)
        if user_response in current_question['options']:
            response_text = current_question['options'][user_response]
            salvar_resposta(user_id, current_question['text'],
response_text)
            next_step = current_question['next'][user_response]

        else:
            await update.message.reply_text("Resposta inválida. Por
favor, selecione uma opção válida.")
            return
```

```
    # Próximos passos de acordo com o fluxo
    # Condições finais ou para próximas perguntas de proteinúria
    if next_step == 'end_monitoring':
        await update.message.reply_text("Continue monitorando
regularmente sua pressão arterial. Agende a próxima consulta de
rotina.")
        # Segue para pergunta sobre proteinúria
        context.user_data['question_id'] = '4'
        await enviar_pergunta(update, context, '4')
    elif next_step == 'diagnostico_has_cronica':
        await update.message.reply_text("Risco de: Hipertensão Arterial
Crônica (HAS Crônica).")
        salvar_resposta(user_id, 'Diagnóstico Hipertensão',
                        'Hipertensão Arterial Crônica')
        # Segue para pergunta sobre proteinúria
        context.user_data['question_id'] = '4'
        await enviar_pergunta(update, context, '4')
    elif next_step == 'diagnostico_hipertensao_gestacional':
        await update.message.reply_text("Risco de: Hipertensão
Gestacional.")
        salvar_resposta(user_id, 'Diagnóstico Hipertensão',
                        'Hipertensão Gestacional')
        # Segue para pergunta sobre proteinúria
        context.user_data['question_id'] = '4'
        await enviar_pergunta(update, context, '4')
    elif next_step == 'recommend_exam':
        await update.message.reply_text("Recomendo que faça um exame de
urina para verificar a presença de proteína. Por favor, continue o
acompanhamento regular.")
        # context.user_data['question_id'] = '5'  # Próxima etapa
        # await enviar_pergunta(update, context, '5')
        # context.user_data.clear()
        # Segue para pergunta sobre acido urico
        context.user_data['question_id'] = '6'
        await enviar_pergunta(update, context, '6')
    elif next_step == 'diagnostico_pre_eclampsia':
        await update.message.reply_text("Risco de: Pré-
eclâmpsia.\nEncaminhamento ao serviço de alto risco recomendado.")
```

```
            salvar_resposta(user_id, 'Diagnóstico', 'Pré-eclâmpsia')
            # context.user_data.clear()  # Finaliza a interação
            context.user_data['question_id'] = '7'
            await enviar_pergunta(update, context, '7')
        elif next_step == 'repeat_exam':
            await update.message.reply_text("Recomendo repetir o exame em
15 dias e monitorar sintomas.")
            # context.user_data.clear()  # Finaliza a interação
            context.user_data['question_id'] = '6'
            await enviar_pergunta(update, context, '6')
        elif next_step == 'continue_monitoring':
            await update.message.reply_text("Continue monitoramento e
acompanhamento regular na UBS.")
            # Delay de 3 segundos antes de enviar a mensagem inicial
            await asyncio.sleep(3)
            # Envia novamente a mensagem inicial
            context.user_data.clear()  # Finaliza a interação
            context.user_data['question_id'] = '0'
            await enviar_pergunta(update, context, '0')
        elif next_step == 'diagnostico_pre_eclampsia_grave':
            await update.message.reply_text("Risco de: Pré-eclâmpsia
Grave.\nEncaminhamento ao serviço de alto risco recomendado.")
            salvar_resposta(
                user_id, 'Diagnóstico', 'Pré-eclâmpsia Grave')
            # context.user_data.clear()
            context.user_data['question_id'] = '9'
            await enviar_pergunta(update, context, '9')
        elif next_step == 'diagnostico_eclampsia_iminente':
            await update.message.reply_text("Risco de: Eclâmpsia Iminente.

Você apresenta risco iminente de eclâmpsia.\nEncaminhamento imediato ao
serviço de alto risco é recomendado.")
            salvar_resposta(
                user_id, 'Diagnóstico', 'Eclâmpsia Iminente')
            # Delay de 3 segundos antes de enviar a mensagem inicial
            await asyncio.sleep(3)
            # Envia novamente a mensagem inicial
            context.user_data.clear()
            context.user_data['question_id'] = '0'
            await enviar_pergunta(update, context, '0')
        else:
            # Atribui o próximo question_id no contexto do usuário e faz a
próxima pergunta
            context.user_data['question_id'] = next_step

await enviar_pergunta(update, context, next_step)
```

```
async def handle_audio(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    user_id = update.message.from_user.id
    user_dir = os.path.join(AUDIO_DIR, f"user_{user_id}")
    os.makedirs(user_dir, exist_ok=True)

    try:
        print("Handle_audio")
        # Obtém o arquivo de áudio enviado
        usuario_id = update.message.from_user.id
        user_dir = os.path.join(AUDIO_DIR, f"user_{usuario_id}")
        os.makedirs(user_dir, exist_ok=True)

        audio_file = await update.message.voice.get_file()
        audio_path = get_next_filename(
            user_dir, f'audio_user_{usuario_id}', '.wav')
        await audio_file.download_to_drive(audio_path)

        salvar_midia(usuario_id, 'audio', audio_path)

        # Aguardar transcrição (simulada com sleep)
        time.sleep(3)

        # Retorna a confirmação e reinicia o fluxo inicial
        await update.message.reply_text("Áudio recebido e transcrito
com sucesso.")
        context.user_data.clear()  # Reseta o contexto do usuário
        context.user data['question id'] = '0'
        await enviar_pergunta(update, context, '0')
    except Exception as e:
        await update.message.reply_text("Erro ao processar o áudio. Por
favor, tente novamente.")
        print(f"Erro ao lidar com o áudio: {e}")

async def handle_photo(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    user_id = update.message.from_user.id
    user_dir = os.path.join(IMAGES_DIR, f"user_{user_id}")
    os.makedirs(user_dir, exist_ok=True)
```

```
    try:
        print("Handle_photo")
        # Obtém o arquivo de foto enviado
        photo_file = await update.message.photo[-1].get_file()
        photo_path = get_next_filename(
            user_dir, f'photo_user_{user_id}', '.jpg'
        )
        await photo_file.download_to_drive(photo_path)

        # Salva a mídia no banco de dados
        salvar_midia(user_id, 'foto', photo_path)

        # Retorna a confirmação e reinicia o fluxo inicial
        await update.message.reply_text("Foto recebida com sucesso.")
        context.user_data.clear()  # Reseta o contexto do usuário
        context.user_data['question_id'] = '0'
        await enviar_pergunta(update, context, '0')
    except Exception as e:
        await update.message.reply_text("Erro ao processar a foto. Por
favor, tente novamente.")
        print(f"Erro ao lidar com a foto: {e}")

async def handle_video(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    user_id = update.message.from_user.id
    user_dir = os.path.join(VIDEO_DIR, f"user_{user_id}")
    os.makedirs(user_dir, exist_ok=True)

    try:
        print("Handle_video")
        # Obtém o arquivo de vídeo enviado
        video_file = await update.message.video.get_file()
        video_path = get_next_filename(
            user_dir, f'video_user_{user_id}', '.mp4'
        )
        await video_file.download_to_drive(video_path)

        # Salva a mídia no banco de dados
        salvar_midia(user_id, 'video', video_path)

        # Simulação de transcrição e legendagem (substitua por chamadas
reais, se necessário)
        time.sleep(5)  # Simula tempo de processamento de transcrição
        transcricao_texto = f"Transcrição simulada para {
            os.path.basename(video_path)}"
        print(f"Transcrição completa: {transcricao_texto}")
```

```
            # Simula tempo de legendagem
            time.sleep(3)
            legendado_path =
 f"{os.path.splitext(video_path)[0]}_legendado.mp4"
            print(f"Legenda salva em: {legendado_path}")

            # Retorna a confirmação e reinicia o fluxo inicial
            await update.message.reply_text("Vídeo recebido, transcrito e
 legendado com sucesso.")
            context.user_data.clear()  # Reseta o contexto do usuário
            context.user_data['question_id'] = '0'
            await enviar_pergunta(update, context, '0')
        except Exception as e:
            await update.message.reply_text("Erro ao processar o vídeo. Por
 favor, tente novamente.")
            print(f"Erro ao lidar com o vídeo: {e}")

# Função para iniciar a conversa
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
None:
    user_id = update.message.from_user.id
    # Resetar o contexto do usuário ao iniciar uma nova conversa
    context.user_data.clear()  # Limpar as respostas anteriores

    # Obtém o nome do usuário
    nome_usuario = update.message.from_user.first_name

    # Verifica se o usuário já está no banco de dados, e insere se
necessário
    verificar_ou_inserir_usuario(user_id, nome_usuario)

    # Enviar a primeira mensagem
    await update.message.reply_text(
        "Olá! Este é o Bot de Telemonitoramento.\n\n"
        "Este bot monitora parâmetros de saúde e coleta informações
para melhorar seu acompanhamento médico. "
        "Seus dados serão tratados com segurança e utilizados apenas
para fins médicos, em conformidade com a LGPD."
    )

    # Enviar a primeira pergunta
    context.user_data['question_id'] = '0'  # Inicia na primeira
pergunta
    await enviar_pergunta(update, context, '0')
```

```
    # Enviar a primeira pergunta
    context.user_data['question_id'] = '0'  # Inicia na primeira
pergunta
    await enviar_pergunta(update, context, '0')

# Função para verificar se o usuário existe na tabela usuarios, e
inseri-lo se não existir
def verificar_ou_inserir_usuario(usuario_id, nome_usuario):
    with get_db_connection() as connection:
        if connection:
            try:
                with connection.cursor() as cursor:
                    # Verificar se o usuário já existe
                    query_check = "SELECT 1 FROM usuarios WHERE
usuario_id = %s"
                    cursor.execute(query_check, (usuario_id,))
                    if cursor.fetchone() is None:
                        # Insere o usuário
                        query_insert = "INSERT INTO usuarios
(usuario_id, nome) VALUES (%s, %s)"
                        cursor.execute(
                            query_insert, (usuario_id, nome_usuario))
                        connection.commit()
                        print(
                            f"Usuário {usuario_id} inserido no banco de
dados.")
            except Exception as e:
                print(f"Erro ao verificar ou inserir usuário: {e}")

# Função para salvar respostas no banco de dados
def salvar_resposta(telegram_user_id, pergunta, resposta):
    usuario_id = get_usuario_id_by_telegram_id(telegram_user_id)
    with get_db_connection() as connection:
        if connection:
            try:

                with connection.cursor() as cursor:
                    query_insert = """
                    INSERT INTO respostas (usuario_id, pergunta,
resposta)
                    VALUES (%s, %s, %s)
                    """
                    cursor.execute(
                        query_insert, (usuario_id, pergunta, resposta))
                    connection.commit()
                    print(f"Resposta salva no banco de dados:
{resposta}")
            except Exception as e:
                print(f"Erro ao salvar resposta: {e}")
```

```
def salvar_midia(telegram_user_id, tipo_midia, caminho_arquivo):
    usuario_id = get_usuario_id_by_telegram_id(telegram_user_id)
    if usuario_id:
        with get_db_connection() as connection:
            if connection:
                try:
                    with connection.cursor() as cursor:
                        insert_query = """
                        INSERT INTO midias (usuario_id, tipo_midia,
caminho_arquivo)
                        VALUES (%s, %s, %s)
                        """
                        cursor.execute(
                            insert_query, (usuario_id, tipo_midia,
caminho_arquivo))
                        connection.commit()
                        print(f"Mídia salva no banco de dados: {
                              caminho_arquivo}")
                except Exception as e:
                    print(f"Erro ao salvar mídia: {e}")

    # Função para organizar as pastas e nomear os arquivos
    def get_next_filename(directory, base_filename, extension):
        print("get_next_filename")
        counter = 1
        while True:
            filename = f"{base_filename}_{counter}{extension}"

            filepath = os.path.join(directory, filename)
            if not os.path.exists(filepath):
                return filepath
            counter += 1

        # Função para deletar dados do usuário
        def deletar_dados_usuario(usuario_id):
            with get_db_connection() as connection:
                if connection:
                    try:
                        with connection.cursor() as cursor:
```

```
                    # Exclui dados do usuário
                    query_delete_respostas = "DELETE FROM respostas
WHERE usuario_id = %s"
                    query_delete_usuario = "DELETE FROM usuarios WHERE
usuario_id = %s"
                    cursor.execute(query_delete_respostas,
(usuario_id,))
                    cursor.execute(query_delete_usuario, (usuario_id,))
                    connection.commit()
                    print(f"Dados do usuário {
                          usuario_id} apagados com sucesso.")
            except Exception as e:
                print(f"Erro ao apagar dados do usuário: {e}")

# Busca o id do usuário no banco de dados baseado no user_id do
Telegram.
def get_usuario_id_by_telegram_id(telegram_user_id):
    with get_db_connection() as connection:
        if connection:
            try:
                with connection.cursor() as cursor:
                    query = "SELECT id FROM usuarios WHERE usuario_id =
%s"
                    cursor.execute(query, (telegram_user_id,))
                    result = cursor.fetchone()
                    if result:
                        return result[0]
                    else:
                        print(f"Usuário com Telegram ID {
                              telegram_user_id} não encontrado.")

                        return None
            except Exception as e:
                print(f"Erro ao buscar id do usuário: {e}")
                return None

# Defina timeouts maiores
# Configuração de tempo limite para requisições HTTP do Telegram
request = HTTPXRequest(
    connect_timeout=10.0,  # Timeout para conectar (em segundos)
    read_timeout=60.0  # Timeout para leitura de grandes arquivos, como
vídeos
)
```

```
# Função principal para configurar o bot e definir os handlers de
eventos
def main() -> None:
    print("Main")
    application = ApplicationBuilder().token(

"7482188034:AAFMjb13uoC3amBEM21Mr6t9JEDMw1mYykA").request(request).buil
d()
          application.add_handler(CommandHandler("start", start))
          application.add_handler(MessageHandler(
              filters.TEXT & ~filters.COMMAND, handle_message))
          application.add_handler(MessageHandler(filters.VIDEO,
      handle_video))
          application.add_handler(MessageHandler(filters.VOICE,
      handle_audio))
          application.add_handler(MessageHandler(filters.PHOTO,
      handle_photo))

          application.run_polling()

if __name__ == '__main__':
    main()
```

Fonte: Autoria Própria

Figura 4 – Código fonte database.py.

```
import psycopg2
from contextlib import contextmanager

# Função para conectar ao banco de dados
def connect_db():
    try:
        connection = psycopg2.connect(
            host="localhost",  # Endereço do servidor
            database="projeto_tcc_db_2",  # Nome do banco de dados
            user="postgres",  # Seu usuário no PostgreSQL
            password="root1",  # Senha do PostgreSQL
            port="5435"  # Porta do PostgreSQL
        )
        return connection
    except Exception as e:
        print(f"Erro ao conectar ao banco de dados: {e}")
        return None
```

```
    # Context manager para garantir que a conexão seja fechada corretamente
    @contextmanager
    def get_db_connection():
        connection = connect_db()
        try:
            if connection is not None:
                yield connection
            else:
                print("Falha ao obter conexão.")
        except Exception as e:
            print(f"Erro durante operação no banco de dados: {e}")
        finally:
            if connection:
                connection.close()

    # Função para fechar a conexão (se for usada sem o context manager)
    def close_connection(connection):
        if connection:
            connection.close()
```

Fonte: Autoria Própria

Figura 5 – Código fonte legendagem.py.

```
import os  # Biblioteca para operações de sistema de arquivos
import whisper  # Biblioteca para transcrição de áudio com modelos de
IA
import time  # Para pausas no código
# Para monitoramento do sistema de arquivos em tempo real
from watchdog.observers import Observer
# Para lidar com eventos de arquivos
from watchdog.events import FileSystemEventHandler
# Manipulação de vídeo e legendas
from moviepy.editor import VideoFileClip, TextClip, CompositeVideoClip
import moviepy.config as config  # Configuração para MoviePy
# Funções para conexão com o banco de dados
from database import connect_db, close_connection

# Configurar o caminho do executável do ImageMagick para uso com
MoviePy
config.change_settings(
    {"IMAGEMAGICK_BINARY": "C:\\Program Files\\ImageMagick-7.1.1-Q16-
HDRI\\magick.exe"})
```

```
# Caminho da pasta onde os vídeos estão salvos
base_dir = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
video_directory = os.path.join(base_dir, 'media', 'video')

# Criar pasta para salvar vídeos legendados, caso não exista
legendado_directory = os.path.join(base_dir, 'media', 'legendado')
os.makedirs(legendado_directory, exist_ok=True)

# Carregar o modelo Whisper para transcrição
modelo = whisper.load_model("base")

# Função para verificar se o vídeo já foi legendado
def is_video_legendado(video_path):
    connection = connect_db()
    if connection:
        cursor = connection.cursor()
        # Consultar o status de legendagem no banco de dados
        check_query = """
        SELECT status_legendagem FROM midias WHERE caminho_arquivo = %s
        """
        cursor.execute(check_query, (video_path,))
        result = cursor.fetchone()

        cursor.close()
        close_connection(connection)
        if result and result[0]:  # Se o resultado for True, já foi
legendado
            return True
    return False

# Função para atualizar o status de legendagem e salvar transcrição no
banco de dados
def atualizar_status_legendagem(video_path, transcricao_texto):
    connection = connect_db()
    if connection:
        cursor = connection.cursor()
        # Atualizar status de legendagem e transcrição no banco de
dados
        update_query = """
        UPDATE midias
        SET status_legendagem = TRUE, transcricao_video = %s
        WHERE caminho_arquivo = %s
        """
        cursor.execute(update_query, (transcricao_texto, video_path))
        connection.commit()
        cursor.close()
        close_connection(connection)
        print(f"Status de legendagem e texto transcrito atualizado para
{
            video_path}")
```

```
# Classe para lidar com eventos de criação de arquivos de vídeo
class VideoHandler(FileSystemEventHandler):
    def on_created(self, event):
        if event.is_directory:
            # Monitorar novos diretórios recursivamente
            observer.schedule(VideoHandler(), event.src_path,
recursive=True)
            print(f"Novo diretório detectado e monitorado:
{event.src_path}")
        elif event.src_path.endswith(".mp4"):
            video_path = event.src_path
            print(f"Novo arquivo de vídeo detectado: {video_path}")

             if is_video_legendado(video_path):
                 print(f"Vídeo {video_path} já foi legendado
 anteriormente.")
                 return  # Evita legendagem duplicada

             try:
                 # Extrair o áudio do vídeo
                 video = VideoFileClip(video_path)
                 audio_filename =
 f"{os.path.splitext(video_path)[0]}_audio.wav"
                 audio_path = os.path.join(video_directory,
 audio_filename)
                 video.audio.write_audiofile(audio_path)

                 # Transcrever o áudio usando Whisper
                 resposta = modelo.transcribe(audio_path)
                 texto_transcricao_video = resposta['text']

                 # Salvar transcrição no banco de dados
                 atualizar_status_legendagem(
                     video_path, texto_transcricao_video)

                 # Criar clipes de legendas com timestamps
                 segments = resposta.get('segments', [])
                 if not segments:
                     print(f"Erro: Nenhuma transcrição detectada para o
 áudio {
                           audio_path}.")
                     return
```

```
                # Criação dos clipes de legenda para cada segmento de
texto
                subtitle_clips = create_subtitle_clips(segments, video)

                # Combina o vídeo com as legendas
                video_with_subtitles = CompositeVideoClip(
                    [video] + subtitle_clips)

                # Salva o vídeo legendado
                video_output =
f"{os.path.splitext(os.path.basename(video_path))[
                    0]}_legendado.mp4"

                video_output_path = os.path.join(
                    legendado_directory, video_output)
                video_with_subtitles.write_videofile(
                    video_output_path, codec='libx264', fps=24,
audio_codec="aac")

                print(f"Vídeo legendado salvo em: {video_output_path}")
                video.close()

                # Atualizar o status de legendagem no banco de dados
                atualizar_status_legendagem(
                    video_path, texto_transcricao_video)

            except Exception as e:
                print(f"Erro ao processar o vídeo {video_path}: {e}")

# Função para criar clipes de legendas
def create_subtitle_clips(segments, video):
    clips = []
    for segment in segments:
        text = segment['text'].strip()
        start = segment['start']
        end = segment['end']

        if text:  # Evitar textos vazios
            txt_clip = TextClip(text, fontsize=24,
                                color='white', bg_color='black')
            txt_clip = txt_clip.set_pos(('center',
'bottom')).set_start(
                start).set_duration(end - start)
            clips.append(txt_clip)
    return clips
```

```
# Função para monitorar o diretório de vídeo em busca de novos arquivos
def monitor_video_directory():
    print(f"Monitorando o diretório de vídeos: {video_directory}")
    event_handler = VideoHandler()
    observer.schedule(event_handler, video_directory, recursive=True)
    observer.start()

    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()

# Instancia o Observer e inicia o monitoramento
observer = Observer()
if __name__ == '__main__':
    monitor_video_directory()
```

Fonte: Autoria Própria

Figura 6 – Código fonte transcricao.py.

```
import os  # Biblioteca para operações de sistema de arquivos
import whisper  # Biblioteca para transcrição de áudio usando modelos
de IA
import time  # Para pausar o código
# Para monitorar o sistema de arquivos em tempo real
from watchdog.observers import Observer
# Para lidar com eventos no sistema de arquivos
from watchdog.events import FileSystemEventHandler
# Funções para conexão com o banco de dados
from database import connect_db, close_connection

# Caminho da pasta onde os áudios estão salvos
base_dir = os.path.dirname(os.path.dirname(
    os.path.abspath(__file__)))  # Caminho até /ProjetoTCC
audio_directory = os.path.join(base_dir, 'media', 'audio')
# Criar pasta para salvar transcrições, se não existir
transcription_directory = os.path.join(base_dir, 'media',
'transcricao')
os.makedirs(transcription_directory, exist_ok=True)

# Carregar o modelo Whisper para transcrição de áudio
modelo = whisper.load_model("base")
```

```
# Função para verificar se o áudio já foi transcrito, usando uma flag
no banco de dados
def is_audio_transcrito(audio_path):
    connection = connect_db()
    if connection:
        cursor = connection.cursor()
        # Consultar o status de transcrição no banco de dados
        check_query = """
        SELECT status_transcricao FROM midias WHERE caminho_arquivo =
%s
        """
        cursor.execute(check_query, (audio_path,))
        result = cursor.fetchone()
        cursor.close()
        close_connection(connection)
        if result and result[0]:  # Se o resultado for True, já foi
transcrito
            return True
    return False

# Função para atualizar o status de transcrição no banco e salvar a
transcrição
def atualizar_status_transcricao(audio_path, transcricao_texto):
    connection = connect_db()
    if connection:
        cursor = connection.cursor()
        # Atualizar status e texto de transcrição no banco de dados
        update_query = """
        UPDATE midias
        SET status_transcricao = TRUE, transcricao_audio = %s
        WHERE caminho_arquivo = %s
        """
        cursor.execute(update_query, (transcricao_texto, audio_path))
        connection.commit()
        cursor.close()
        close_connection(connection)
        print(f"Status de transcrição e texto transcrito atualizado
para {
              audio_path}")

# Classe para lidar com eventos de criação de arquivos no diretório de
áudio
class AudioHandler(FileSystemEventHandler):
    def on_created(self, event):
        if event.is_directory:
```

```
            # Caso um novo diretório seja criado, monitorá-lo
recursivamente
            observer.schedule(AudioHandler(), event.src_path,
recursive=True)
            print(f"Novo diretório detectado e monitorado:
{event.src_path}")
        elif event.src_path.endswith(".wav") or
event.src_path.endswith(".ogg"):
            audio_path = event.src_path
            print(f"Novo arquivo de áudio detectado: {audio_path}")

            # Verificar se o áudio já foi transcrito
            if is_audio_transcrito(audio_path):
                print(f"Áudio {audio_path} já foi transcrito
anteriormente.")
                return  # Evita retranscrever
            try:
                # Fazer a transcrição usando o modelo Whisper
                resposta = modelo.transcribe(audio_path)

                # Imprimir a transcrição no terminal
                print("Transcrição:")
                print(resposta['text'])

                # Obter o texto da transcrição
                texto_transcricao = resposta['text']

                # Salvar a transcrição no banco de dados
                atualizar_status_transcricao(audio_path,
texto_transcricao)

                # Salvar a transcrição em um arquivo .txt na pasta de
transcrições
                transcricao_filename = f"{os.path.splitext(
                    os.path.basename(audio_path))[0]}.txt"
                transcricao_path = os.path.join(
                    transcription_directory, transcricao_filename)

                with open(transcricao_path, 'w', encoding='utf-8') as
f:
                    f.write(resposta['text'])
                print(f"Transcrição salva em: {transcricao_path}")
except Exception as e:
    print(f"Erro ao transcrever o áudio {audio_path}: {e}")
```

```
    # Função para monitorar o diretório de áudio em busca de novos arquivos
    def monitor_audio_directory():
        print(f"Monitorando o diretório de áudio: {audio_directory}")
        event_handler = AudioHandler()
        observer.schedule(event_handler, audio_directory, recursive=True)
        observer.start()

        try:
            while True:
                time.sleep(1)  # Mantém o monitoramento em execução
        except KeyboardInterrupt:
            observer.stop()  # Para o monitoramento ao interromper o
    programa
        observer.join()
    # Instancia o Observer e inicia o monitoramento
    observer = Observer()
    if __name__ == '__main__':
        monitor_audio_directory()
```

Fonte: Autoria Própria

Figura 7 – Código fonte index.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Dashboard - ProjetoTCC</title>
    <link rel="stylesheet" href="/static/css/style_geral.css">
    <link rel="stylesheet" href="/static/css/style_graficos.css">
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/chartjs-plugin-
datalabels"></script>
    <script src="/static/js/scripts_geral.js"></script>
</head>
<body>
    <h1>ProjetoTCC</h1>
    <!-- Barra de navegação para alternar entre abas -->
    <nav>
        <button onclick="showTab('graphics')">Gráficos</button>
        <button onclick="showTab('patients')">Pacientes</button>
    </nav>
    <!-- Aba de gráficos -->
    <div id="tab-graphics" class="tab-content">
        <h2>Dashboard de Gráficos dos Pacientes</h2>
        <div class="dashboard-cards">
            <!-- Outros cards existentes -->
            <div class="card-usuario">
                <h2 id="total-usuarios">0</h2>
```

```
                <p>Total de Usuários</p>
            </div>
        </div>
        <div class="dashboard-grid">
            <div class="card">
                <h2>Monitoramento da Pressão Arterial (PA)<button
class="expand-btn" onclick="window.open('/grafico_expandido?pergunta='
+ encodeURIComponent('Qual foi a sua última medição de pressão
arterial?'), '_blank')">⤢</button></h2>
                <canvas id="graficoPizzaPressaoArterial"></canvas>
            </div>
            <div class="card">
                <h2>Hipertensão Arterial Crônica (HAS Crônica)<button
class="expand-btn" onclick="window.open('/grafico_expandido?pergunta='
+ encodeURIComponent('Sua pressão arterial elevada foi registrada antes
da 20ª semana de gestação?'), '_blank')">⤢</button></h2>
                <canvas id="graficoPizzaPressao20"></canvas>
            </div>
            <div class="card">
                <h2>Diagnóstico de Hipertensão Gestacional<button
class="expand-btn" onclick="window.open('/grafico_expandido?pergunta='
+ encodeURIComponent('Sua pressão arterial está elevada em 3 ocasiões
diferentes?'), '_blank')">⤢</button></h2>
                <canvas id="graficoPizzaPressaoEm3"></canvas>
            </div>
            <div class="card">
                <h2>Fez exame de urina <button class="expand-btn"
onclick="window.open('/grafico_expandido?pergunta=' +
encodeURIComponent('Você já fez algum exame de urina recente para
verificar a presença de proteína (proteinúria)?'),
'_blank')">⤢</button></h2>
                <canvas id="graficoDonutsExameUrina"></canvas>
            </div>
            <div class="card medium-card">
                <h2>Resultado do exame de urina <button class="expand-
btn" onclick="window.open('/grafico_expandido?pergunta=' +
encodeURIComponent('Qual foi o resultado do exame de urina?'),
'_blank')">⤢</button></h2>
                <canvas id="graficoBarrasProteinuria"></canvas>
            </div>
            <div class="card">
                <h2>Níveis elevados de ácido úrico <button
```

```
class="expand-btn" onclick="window.open('/grafico_expandido?pergunta='
+ encodeURIComponent('Seus exames laboratoriais indicaram níveis
elevados de ácido úrico (maiores que 6mg/dl)?'),
'_blank')">⤢</button></h2>
                <canvas id="graficoDonutsElevadoAcidoUrico"></canvas>
            </div>
            <div class="card">
                <h2>Pressão arterial elevada + Sintomas Pré-eclâmpsia
Grave <button class="expand-btn"
onclick="window.open('/grafico_expandido?pergunta=' +
encodeURIComponent('Você teve algum dos seguintes sintomas ou
resultados em exames?\n(Oligúria, Creatinina elevada, Proteinúria
massiva, Plaquetopenia, Aumento de TGO/TGP, Desidrogenase lática
elevada)'), ' blank')">⤢</button></h2>
                <canvas id="graficoDonutsEclampsiaGrave"></canvas>
            </div>
            <div class="card">
                <h2>Sintomas de Eclâmpsia Iminente <button
class="expand-btn" onclick="window.open('/grafico_expandido?pergunta='
+ encodeURIComponent('Você teve algum dos seguintes sintomas
recentemente? (Convulsões, Perda de consciência, Confusão mental, Visão
turva ou diplopia, Cefaleia intensa, Epigastralgia, Tontura, Escotomas
cintilantes)'), '_blank')">⤢</button></h2>
                <canvas id="graficoDonutsEclampsiaIminente"></canvas>
            </div>
            <div class="card medium-card">
                <h2>Diagnóstico <button class="expand-btn"
onclick="window.open('/grafico_expandido?pergunta=' +
encodeURIComponent('Diagnóstico'), '_blank')">⤢</button></h2>
                <canvas id="graficoBarrasDiagnosticos"></canvas>
            </div>
        </div>
        <div class="button-container">
            <button onclick="downloadExcel()">Exportar tabela de
Respostas</button>
            <script>
                function downloadExcel() {
                    window.location.href = "/export/respostas_excel";
                }
            </script>
```

```
            <button onclick="downloadUsuarios()">Exportar tabela de
Usuários</button>
            <script>
                function downloadUsuarios() {
                    window.location.href = "/export/usuarios_excel";
                }
            </script>
        </div>
    </div>
    <!-- Aba de pacientes -->
    <div id="tab-patients" class="tab-content" style="display:none;">
        <h2>Pacientes</h2>
        <input type="text" id="searchBar" placeholder="Buscar
paciente...">
        <ul id="patientList">
            <!-- Lista de pacientes será preenchida dinamicamente -->
        </ul>
    </div>
    <!-- Scripts -->
    <script src="/static/js/scripts_graficos.js"></script>
    <script src="/static/js/scripts_pacientes.js"></script>
    <script>
        // Chama os gráficos ao carregar a página
        window.onload = function() {
            showTab('graphics'); // Exibe a aba de gráficos
automaticamente
        }
    </script>
</body>
</html>
```

Fonte: Autoria Próprio

Figura 8 – Código fonte gráfico_expandido.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Gráfico Expandido</title>
    <!-- Vincula o CSS para estilizar os gráficos e layout da página --
>
    <link rel="stylesheet" href="/static/css/style_graficos.css">
    <!-- Importa a biblioteca Chart.js para renderizar gráficos -->
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/chartjs-plugin-
datalabels"></script>
</head>
<body>
    <!-- Exibe o título do gráfico expandido, representando a pergunta
selecionada -->
    <h1>{{ pergunta }} - Gráfico Expandido</h1>

    <!-- Container para o gráfico em tela cheia -->
    <div class="large-chart">
        <canvas id="graficoExpandido"></canvas>
    </div>

    <!-- Seção que lista os pacientes por resposta -->
    <div class="pacientes-lista">
        <h2>Pacientes por Resposta:</h2>

     <!-- Seção que lista os pacientes por resposta -->
     <div class="pacientes-lista">
         <h2>Pacientes por Resposta:</h2>
         <!-- Loop pelos dados das respostas, fornecendo uma lista de
 pacientes por cada resposta -->
         {% for resposta, nomes in labels.items() %}
             <h3>{{ resposta }}</h3>
             <ul>
                 <!-- Loop pelos nomes dos pacientes para cada resposta
 -->
                 {% for nome in nomes %}
                     <li>{{ nome }}</li>
                 {% endfor %}
             </ul>
         {% endfor %}
     </div>

    <!-- Importa o JavaScript para manipular e gerar o gráfico -->
     <script src="/static/js/scripts_graficos.js"></script>
 </body>
 </html>
```

Fonte: Autoria Próprio

Figura 9 – Código fonte perfil_pacientes.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Perfil do Paciente</title>
    <link rel="stylesheet" href="/static/css/style_pacientes.css">
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="/static/js/scripts_pacientes.js"></script>
</head>
<body>
    <h1>Perfil de {{ patient[0] }}</h1>

    <!-- Informações Gerais -->
    <div class="patient-details">
        <h2>Informações Gerais</h2>
        <p>Nome: {{ patient[0] }}</p>
        <p>Último DIagnóstico: {{ ultima_conclusao if ultima_conclusao
else "Nenhuma conclusão disponível" }}</p> <!-- Exibir conclusão ou
mensagem -->
    </div>

     <!-- Sintomas / Histórico de Conversa -->
     <div class="conversation-history">
         <h2>Sintomas / Histórico de Conversa</h2>
         <ul id="historico-respostas">
             {% for item in ultimas_mensagens %}
             <li>{{ item[2] }} ({{ item[4] }}): {{ item[3] }}</li> <!--
 Exibir data formatada e mensagem -->
             {% endfor %}
         </ul>
         <button id="ver-mais" onclick="mostrarMensagensAnteriores('{{
 patient[3] }}')">Ver mensagens anteriores</button>
     </div>

    <!-- Mídias -->
    <div id="midias-container">
        <h2>Mídias</h2>
        <!-- As mídias enviadas pelo paciente aparecerão aqui -->
    </div>
    <script>
        // Chamar a função para exibir as mídias quando o perfil do
paciente for carregado
        exibirMidias();
    </script>
    <script src="/static/js/"></script>
</body>
</html>
```

Fonte: Autoria Própria

Figura 10 – Código fonte scripts_graficos.js

```
// Função para criar gráfico de pizza
function criarGraficoPizza(ctx, pergunta) {
    let chart = new Chart(ctx, {
        type: 'pie',
        data: {
            labels: [],  // As labels serão preenchidas depois
            datasets: [{
                label: 'Respostas',
                data: [],
                backgroundColor: ['#36A2EB', '#FF6384', '#FFCE56'],
                hoverOffset: 4
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            plugins: {
                legend: {
                    position: 'top',
                },
                tooltip: {
                    callbacks: {
                        label: function (context) {
                            let label = context.label || '';
                            if (label) {
                                label += ': ';
                            }
                            label += context.raw || '';

            }
        }
  },
  datalabels: { // Configuração do plugin
      color: '#FFFFFF', // Cor do número
      font: {
          size: 14, // Tamanho do número
          weight: 'bold' // Peso do número
      },
      textStrokeColor: '#000000', // Cor da borda (preto)
      textStrokeWidth: 3, // Largura da borda
      formatter: (value, ctx) => {
          return value; // Exibe o valor
      },
```

```
                    align: 'center', // Alinhamento (centralizado na
fatia)
                    anchor: 'center' // Posição (âncora no centro da
fatia)
                }
            }
        },
        plugins: [ChartDataLabels] // Adiciona o plugin
    });

    // Função para buscar os dados do backend e atualizar o gráfico
    function updateChart() {
fetch(`/dados_grafico?pergunta=${encodeURIComponent(pergunta)}`)
            .then(response => response.json())
            .then(data => {
                chart.data.labels = data.labels;
                chart.data.datasets[0].data = data.data;
                chart.update();  // Atualiza o gráfico com novos dados
            })
            .catch(error => console.error('Erro ao buscar dados:',
error));
    }

    // Atualiza o gráfico a cada 30 segundos
    setInterval(updateChart, 30000);
    updateChart();  // Carrega os dados na primeira vez
}

// Função para criar gráfico de donuts
function criarGraficoDonuts(ctx, pergunta) {
    let chart = new Chart(ctx, {
        type: 'doughnut',
        data: {
            labels: [],  // As labels serão preenchidas depois
            datasets: [{
                label: 'Respostas',
                data: [],
                backgroundColor: ['#FF6384', '#4BC0C0', '#FFCE56'],
                hoverOffset: 4
            }]
        },
```

```
        options: {
            responsive: true,
            maintainAspectRatio: false,
            plugins: {
                legend: {
                    position: 'top',
                },
                tooltip: {
                    callbacks: {
                        label: function (context) {
                            let label = context.label || '';
                            if (label) {
                                label += ': ';
                            }
                            label += context.raw || '';
                            return label;
                        }
                    }
                },
                datalabels: { // Configuração do plugin
                    color: '#FFFFFF', // Cor do número
                    font: {
                        size: 14, // Tamanho do número
                        weight: 'bold' // Peso do número
                    },
                    textStrokeColor: '#000000', // Cor da borda (preto)
                    textStrokeWidth: 3, // Largura da borda
                    formatter: (value, ctx) => {
                        return value; // Exibe o valor

                     },
                     align: 'center', // Alinhamento (centralizado na
fatia)
                     anchor: 'center' // Posição (âncora no centro da
fatia)
                 }
             }
         },
         plugins: [ChartDataLabels] // Adiciona o plugin
    });
```

```
    // Função para buscar os dados do backend e atualizar o gráfico
    function updateChart() {

fetch(`/dados_grafico?pergunta=${encodeURIComponent(pergunta)}`)
            .then(response => response.json())
            .then(data => {
                chart.data.labels = data.labels;
                chart.data.datasets[0].data = data.data;
                chart.update();  // Atualiza o gráfico com novos dados
            })
            .catch(error => console.error('Erro ao buscar dados:',
error));
    }

    // Atualiza o gráfico a cada 30 segundos
    setInterval(updateChart, 30000);
    updateChart();  // Carrega os dados na primeira vez
}

// Função para criar um gráfico de barras
function criarGraficoBarra(ctx, pergunta, labelX = 'Categorias', labelY
= 'Número de Pessoas') {
    let chart = new Chart(ctx, {
        type: 'bar',
        data: {
            labels: [],  // As labels serão preenchidas depois
            datasets: [{
                label: 'Respostas',
                data: [],  // Os dados serão preenchidos depois
                backgroundColor: '#36A2EB',  // Cor das barras
                borderColor: '#36A2EB',
                borderWidth: 1
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            scales: {
                x: {
                    title: {
                        display: true,
                        text: labelX  // Rótulo do eixo X
```

```
                    }
                },
                y: {
                    beginAtZero: true,
                    title: {
                        display: true,
                        text: labelY  // Rótulo do eixo Y
                    }
                }
            },
            plugins: {
                legend: {
                    display: false  // Oculta a legenda
                },
                tooltip: {
                    callbacks: {
                        label: function (context) {
                            return `Quantidade: ${context.raw}`;
                        }
                    }
                }
            }
        }
    });

    // Função para buscar os dados do backend e atualizar o gráfico
    function updateChart() {

fetch(`/dados_grafico?pergunta=${encodeURIComponent(pergunta)}`)
            .then(response => response.json())
            .then(data => {
                // Verifica se a pergunta é "Diagnóstico" para definir
uma ordem personalizada
                if (pergunta === "Diagnóstico") {
                    const desiredOrder = ['Pré-eclâmpsia', 'Pré-
eclâmpsia Grave', 'Eclâmpsia Iminente'];
                    const orderedLabels = [];
                    const orderedData = [];

                    desiredOrder.forEach((label) => {
                        const index = data.labels.indexOf(label);
                        if (index !== -1) {
                            orderedLabels.push(data.labels[index]);
```

```
                        orderedData.push(data.data[index]);
                    }
                });

                // Atualiza os dados do gráfico com a ordem
corrigida
                chart.data.labels = orderedLabels;
                chart.data.datasets[0].data = orderedData;
            } else {
                // Se não for a pergunta "Diagnóstico", usa a ordem
padrão dos dados
                chart.data.labels = data.labels;
                chart.data.datasets[0].data = data.data;
            }

            chart.update();  // Atualiza o gráfico com os dados
processados
        })
        .catch(error => console.error('Erro ao buscar dados:',
error));
    }

    // Atualiza o gráfico a cada 30 segundos
    setInterval(updateChart, 30000);
    updateChart();  // Carrega os dados na primeira vez
}

// Função para atualizar o total de usuários
function atualizarTotalUsuarios() {
    fetch('/total_usuarios')
        .then(response => response.json())
        .then(data => {
            document.getElementById('total-usuarios').textContent =
data.totalUsuarios;
        })
        .catch(error => console.error('Erro ao buscar total de
usuários:', error));
}
```

```
// Atualiza os dados dos cards de usuários e diagnósticos
function atualizarCards() {
    atualizarTotalUsuarios();
    fetch('/dados_correlacionados')
        .then(response => response.json())
        .then(data => {
            document.getElementById('total-pre-eclampsia').textContent
= data.preEclampsia;
            document.getElementById('proteinuria-pre-
eclampsia').textContent = data.proteinuria;
            document.getElementById('acido-urico-elevado').textContent
= data.acidoUricoElevado;
            document.getElementById('pre-eclampsia-grave').textContent
= data.preEclampsiaGrave;
            document.getElementById('eclampsia-iminente').textContent =
data.eclampsiaIminente;
        })
        .catch(error => console.error('Erro ao buscar dados
correlacionados:', error));
}

// Atualiza os dados dos cards a cada 30 segundos
setInterval(atualizarCards, 30000);
atualizarCards();  // Atualiza os dados na primeira vez

// Função para expandir o gráfico ao clicar no botão
function expandirGrafico(pergunta) {
    window.location.href =
`/grafico_expandido?pergunta=${encodeURIComponent(pergunta)}`;
}

// Função específica para criar o gráfico na página expandida
function criarGraficoExpandido(pergunta, labels, data) {
    const ctx =
document.getElementById('graficoExpandido').getContext('2d');
    let chart = new Chart(ctx, {
        type: 'pie',  // Você pode ajustar o tipo do gráfico se
necessário
        data: {
            labels: labels,
            datasets: [{
                label: 'Respostas',
                data: data,
                backgroundColor: ['#36A2EB', '#FF6384', '#FFCE56'],
                hoverOffset: 4
            }]
        },
```

```
        options: {
            responsive: true,
            maintainAspectRatio: false,
            plugins: {
                legend: {
                    position: 'top',
                },
                tooltip: {
                    callbacks: {
                        label: function (context) {
                            let label = context.label || '';
                            if (label) {
                                label += ': ';
                            }
                            label += context.raw || '';
                            return label;
                        }
                    }
                },
                datalabels: { // Configuração do plugin
                    color: '#FFFFFF', // Cor do número
                    font: {
                        size: 14, // Tamanho do número
                        weight: 'bold' // Peso do número
                    },
                    textStrokeColor: '#000000', // Cor da borda (preto)
                    textStrokeWidth: 3, // Largura da borda


                     formatter: (value, ctx) => {
                         return value; // Exibe o valor
                     },
                     align: 'center', // Alinhamento (centralizado na
fatia)
                     anchor: 'center' // Posição (âncora no centro da
fatia)
                 }
             }
         },
         plugins: [ChartDataLabels] // Adiciona o plugin
    });
}
```

```
// Código para detectar a página de gráfico expandido e renderizar o
gráfico
document.addEventListener('DOMContentLoaded', function () {
    const expandidoCanvas =
document.getElementById('graficoExpandido');
    if (expandidoCanvas) {
        // Se estivermos na página do gráfico expandido, obter os dados
do backend e renderizar
        const pergunta = document.querySelector('h1').innerText.split('
- ')[0]; // Obtém a pergunta do título

fetch(`/dados_grafico?pergunta=${encodeURIComponent(pergunta)}`)
            .then(response => response.json())
            .then(data => {
                criarGraficoExpandido(pergunta, data.labels,
data.data);
            })
            .catch(error => console.error('Erro ao buscar dados:',
error));
    }
});

// Crie os gráficos para diferentes perguntas, passando o tipo de
gráfico e pergunta
const ctxPressaoArterial =
document.getElementById('graficoPizzaPressaoArterial').getContext('2d')
;
criarGraficoPizza(ctxPressaoArterial, 'Qual foi a sua última medição de
pressão arterial?');
const ctxPressao20 =
document.getElementById('graficoPizzaPressao20').getContext('2d');
criarGraficoPizza(ctxPressao20, 'Sua pressão arterial elevada foi
registrada antes da 20ª semana de gestação?');
const ctxPressao3 =
document.getElementById('graficoPizzaPressaoEm3').getContext('2d');
criarGraficoPizza(ctxPressao3, 'Sua pressão arterial está elevada em 3
ocasiões diferentes?');
const ctxExameUrina =
document.getElementById('graficoDonutsExameUrina').getContext('2d');
criarGraficoDonuts(ctxExameUrina, 'Você já fez algum exame de urina
recente para verificar a presença de proteína (proteinúria)?');
const ctxProteinuria =
document.getElementById('graficoBarrasProteinuria').getContext('2d');
criarGraficoBarra(ctxProteinuria, 'Qual foi o resultado do exame de
urina?', 'Resultado do Exame de Urina', 'Número de Pessoas');
const ctxELevadoAcidoUrico =
document.getElementById('graficoDonutsElevadoAcidoUrico').getContext('2
d');
criarGraficoDonuts(ctxELevadoAcidoUrico, 'Seus exames laboratoriais
indicaram níveis elevados de ácido úrico (maiores que 6mg/dl)?');
```

```
const ctxPreEclampsiaGrave =
document.getElementById('graficoDonutsEclampsiaGrave').getContext('2d')
;
criarGraficoDonuts(ctxPreEclampsiaGrave, 'Você teve algum dos seguintes
sintomas ou resultados em exames?\n(Oligúria, Creatinina elevada,
Proteinúria massiva, Plaquetopenia, Aumento de TGO/TGP, Desidrogenase
lática elevada)');
const ctxEclampsiaIminente =
document.getElementById('graficoDonutsEclampsiaIminente').getContext('2
d');
criarGraficoDonuts(ctxEclampsiaIminente, 'Você teve algum dos seguintes
sintomas recentemente? (Convulsões, Perda de consciência, Confusão
mental, Visão turva ou diplopia, Cefaleia intensa, Epigastralgia,
Tontura, Escotomas cintilantes)');

const ctxDiagnosticos =
document.getElementById('graficoBarrasDiagnosticos').getContext('2d');
criarGraficoBarra(ctxDiagnosticos, 'Diagnóstico', 'Diagnósticos',
'Número de Pacientes');
```

Fonte: Autoria Própria

Figura 11 – Código fonte scripts_pacientes.js

```
// Função para carregar a lista de pacientes
function loadPatients() {
    fetch('/api/pacientes')  // Rota que retorna a lista de pacientes
        .then(response => response.json())
        .then(patients => {
            const patientList = document.getElementById('patientList');
            patientList.innerHTML = '';  // Limpa a lista
            patients.forEach(patient => {
                const li = document.createElement('li');
                li.textContent = patient.nome;
                // Abre o perfil do paciente em uma nova aba ao clicar
                li.onclick = () =>
window.open(`/paciente/${patient.id}`, '_blank');
                patientList.appendChild(li);
            });
        })
        .catch(error => console.error('Erro ao carregar pacientes:',
error));
}
```

```
// Função para exibir mídias relacionadas a cada paciente
function exibirMidias() {
    fetch('/obter_midias')
        .then(response => response.json())
        .then(midias => {
            const midiasContainer = document.getElementById('midias-
container');
            midiasContainer.innerHTML = ''; // Limpar o container
anterior

            midias.forEach(midia => {
                const midiaDiv = document.createElement('div');
                midiaDiv.className = 'midia-item';

                // Verifica o tipo de mídia para exibir no formato
adequado
                if (midia.tipo === 'audio') {
                    const audioElement =
document.createElement('audio');
                    audioElement.src = `/media/${midia.caminho}`;  //
Usar caminho direto
                    audioElement.controls = true;
                      midiaDiv.appendChild(audioElement);
                      // Exibir a transcrição do áudio
                      if (midia.transcricao_audio) {
                          const transcricaoAudio =
document.createElement('p');
                          transcricaoAudio.textContent = `Transcrição:
${midia.transcricao_audio}`;
                          midiaDiv.appendChild(transcricaoAudio);
                      }
                } else if (midia.tipo === 'foto') {
                      const imgElement = document.createElement('img');
                      imgElement.src = `/media/${midia.caminho}`;  //
Usar caminho direto
                      imgElement.alt = 'Imagem do paciente';
                      imgElement.style.width = '200px';
                      midiaDiv.appendChild(imgElement);
                } else if (midia.tipo === 'video') {
                      const videoElement =
document.createElement('video');
                      videoElement.src = `/media/${midia.caminho}`;  //
Usar caminho direto
```

```
                    videoElement.controls = true;
                    midiaDiv.appendChild(videoElement);
                    // Exibir a transcrição do vídeo
                    if (midia.transcricao_video) {
                        const transcricaoVideo =
document.createElement('p');
                        transcricaoVideo.textContent = `Transcrição:
${midia.transcricao_video}`;
                        midiaDiv.appendChild(transcricaoVideo);
                    }
                }
                midiasContainer.appendChild(midiaDiv);
            });
        })
        .catch(error => console.error('Erro ao buscar mídias:',
error));
}

// Função para mostrar mensagens anteriores
function mostrarMensagensAnteriores() {
    // Aqui você pode adicionar a lógica para carregar mensagens
adicionais via AJAX
    const historicoContainer = document.getElementById('historico-
respostas');
    fetch('/paciente/mensagens-anteriores')  // Ajuste conforme sua
rota
        .then(response => response.json())
        .then(mensagens => {
            mensagens.forEach(mensagem => {
                const li = document.createElement('li');
                li.textContent = `${mensagem[2]} (${mensagem[4]}):
${mensagem[3]}`;
                historicoContainer.appendChild(li);
            });
        })
        .catch(error => console.error('Erro ao carregar mensagens
anteriores:', error));
}
```

```
// Função para mostrar mensagens anteriores
function mostrarMensagensAnteriores(patientId) {
    console.error('Erro ao carregar mensagens anteriores:', patientId)
    const historicoContainer = document.getElementById('historico-
respostas');
    fetch(`/paciente/mensagens-anteriores/${patientId}`)
        .then(response => response.json())
        .then(mensagens => {
            historicoContainer.innerHTML = '';  // Limpar as mensagens
atuais para evitar duplicação
            mensagens.forEach(mensagem => {
                const li = document.createElement('li');
                li.textContent = `${mensagem[2]} (${mensagem[4]}):
${mensagem[3]}`;  // Exibir data e mensagem
                historicoContainer.appendChild(li);
            });
            document.getElementById('ver-mais').style.display = 'none';
// Esconder o botão após exibir todas
        })
        .catch(error => console.error('Erro ao carregar mensagens
anteriores:', error));
}
// Função para buscar pacientes na lista
document.getElementById('searchBar').addEventListener('input',
function() {
    const query = this.value.toLowerCase();
    const items = document.querySelectorAll('#patientList li');
    items.forEach(item => {
        item.style.display =
item.textContent.toLowerCase().includes(query) ? '' : 'none';
    });
});
// Função para exibir o perfil de um paciente
function showPatientProfile(patientId) {
    window.location.href = `/paciente/${patientId}`;  // Redireciona
para a página do perfil
}
// Carrega a lista de pacientes ao iniciar a aba
loadPatients();
```

Fonte: Autoria Própria