ASSOCIAÇÃO EDUCACIONAL DOM BOSCO
CENTRO UNIVERSITÁRIO DOM BOSCO DO RIO DE JANEIRO
CURSO DE GRADUAÇÃO EM SISTEMA DE INFORMAÇÃO

DANILO MAIA FLORENZANO
JOÃO VICTOR SOARES JORDÃO

ASSISTENTE PESSOAL INTELIGENTE: MELISSA

RESENDE - RJ
2025

Danilo Maia Florenzano
João Victor Soares Jordão

ASSISTENTE PESSOAL INTELIGENTE: MELISSA

Trabalho de Conclusão de Curso apresentado ao curso de Sistema de Informação, como requisito parcial para a obtenção do Grau de Bacharel em Sistema de Informação pelo Centro Universitário Dom Bosco do Rio de Janeiro.

Orientador(a): Professor Rafael Chiarelli Júnior

Resende - RJ

2025

Catalogação na fonte
Biblioteca Central da Associação Educacional Dom Bosco – Resende-RJ

F633 Florenzano, Danilo Maia
Assistente pessoal inteligente: Melissa. / Danilo Maia Florenzano; João Victor Soares Jordão - 2025.
109 f.

Orientador: Rafael Chiareli Júnior
Trabalho de conclusão de curso apresentado como requisito parcial à finalização do curso de Sistemas de Informação do Centro Universitário Dom Bosco do Rio de Janeiro, da Associação Educacional Dom Bosco.

1. Informática. 2. Inteligência artificial. 3. Dispositivo inteligente. 4. Assistente pessoal. I. Jordão, João Victor Soares. II. Chiareli Júnior, Rafael. III. Centro Universitário Dom Bosco do Rio de Janeiro. IV. Associação Educacional Dom Bosco. V. Título.

CDU 004.8(043)

Bibliotecária Maria Ismênia de Oliveira – CRB – 7 / 7667

DANILO MAIA FLORENZANO
JOÃO VICTOR SOARES JORDÃO

ASSISTENTE PESSOAL INTELIGENTE: MELISSA

Trabalho de Conclusão de Curso apresentado ao curso de Sistema de Informação, como requisito parcial para a obtenção do Grau de Bacharel em Sistema de Informação pelo Centro Universitário Dom Bosco do Rio de Janeiro.

BANCA EXAMINADORA

Prof.: _____

Orientador(a): Prof. Rafael Chiarelli Júnior, M.e

Prof.: _____

Convidado(a): Prof. Santiago José Franco Lopes, M.e

Prof.: _____

Convidado(a): Prof. Alexandre Polck M.e

Resende, 29 de novembro de 2025.

Dedicamos este Trabalho de Conclusão de Curso aos professores que nos acompanharam e contribuíram diretamente para nossa formação. Em especial, ao professor Rafael Chiareli Junior, nosso orientador, por fornecer as instruções, o direcionamento e o apoio necessários em cada etapa deste TCC. À professora Mônica Mara, por nos auxiliar ao longo de toda a trajetória acadêmica, sempre com paciência e dedicação. Ao professor Santiago Lopes, que, além das aulas, promoveu eventos de troca de conhecimento e nos abriu espaço para atuarmos como palestrantes, experiência indispensável para o nosso crescimento profissional. E ao professor Gabriel Brenner, coordenador do curso, que sempre demonstrou compreender as dificuldades da graduação e, além do conhecimento técnico, compartilhou lições de vida que nos motivaram a persistir até o final.

Dedicamos também à comunidade *open source*, cuja colaboração contínua e espírito de compartilhamento ampliaram nosso aprendizado e inspiraram nossa jornada ao longo da graduação.

Agradecemos a Deus, pela nossa vida, por nos fortalecer em cada etapa e por nos permitir superar todos os desafios encontrados ao longo desta jornada acadêmica.

À nossa família, que acreditou em nós desde o início, investiu na nossa educação e nos ofereceu todo o apoio necessário para que chegássemos até aqui. Pelo incentivo constante, pela compreensão nos momentos de ausência e por nos fortalecerem ao longo de toda a caminhada desta graduação.

Aos professores, pelo empenho, dedicação e pelos ensinamentos que enriqueceram nossa formação. Cada orientação, correção e palavra de incentivo contribuiu de forma significativa para o nosso desenvolvimento acadêmico e profissional.

“Uma maneira de preservar sua própria imagem é não deixar que o mundo invada sua casa. Foi um modo que encontrei de preservar ao máximo meus valores.”
— Ayrton Senna

RESUMO

Assistentes pessoais inteligentes, como Alexa da Amazon e Siri da Apple, fazem parte da rotina das pessoas comuns. Esse modelo de assistente auxilia no controle de dispositivos inteligentes – como televisões, lâmpadas, ares-condicionados, entre outros modelos de dispositivos com a tecnologia Smart –, na obtenção de informações rápidas que cooperam para pequenas tomadas de decisões, entre outras mais diversas finalidades que o usuário possa dar a essa ferramenta. A Assistente Pessoal Inteligente Melissa se apresenta como uma alternativa não proprietária, self-hosted, de código aberto, e com foco na privacidade do usuário. Assim como as alternativas proprietárias antes citadas, a Melissa também possibilita, por meio de texto ou comandos de voz, o controle de dispositivos inteligentes, acesso rápido a informações específicas, e, adicionalmente, inclui em seu núcleo um Large Language Model (LLM), uma inteligência artificial robusta, que possibilita ao usuário interagir de maneira mais natural com a assistente Melissa, permitindo que ele desenvolva diálogos sobre os mais variados temas.

Palavras-chave: assistente pessoal; código aberto; comando de voz; dispositivos inteligentes; inteligência artificial; Large Language Model; privacidade.

ABSTRACT

Intelligent personal assistants, such as Amazon's Alexa and Apple's Siri, are part of everyday life for ordinary people. These assistants help control smart devices—such as televisions, lamps, air conditioners, and other smart technology-enabled devices—while also providing quick access to information that supports small decision-making processes, among many other possible uses. The Intelligent Personal Assistant Melissa presents itself as a non-proprietary, self-hosted, open-source alternative with a strong focus on user privacy. Like the proprietary alternatives mentioned earlier, Melissa also allows users to control smart devices and quickly access specific information via text or voice commands. Additionally, it features a built-in Large Language Model (LLM), a powerful artificial intelligence system that enables users to interact with Melissa in a more natural way, allowing for conversations on a wide range of topics.

Keywords: personal assistant; open source; voice command; smart devices; artificial intelligence; Large Language Model; privacy.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AEDB	Associação Educacional Dom Bosco
ABNT	Associação Brasileira de Normas Técnicas
RJ	Rio de Janeiro
AEDB	Associação Educacional Dom Bosco
ABNT	Associação Brasileira de Normas Técnicas
RJ	Rio de Janeiro
IoT	Internet of Things (Internet das Coisas)
IA	Inteligência Artificial
LLM	Large Language Model
API	Application Programming Interface
UML	Unified Modeling Language
SQL	Structured Query Language
SGBD	Sistema de Gerenciamento de Banco de Dados
GTX	Linha de GPUs Nvidia GTX
RTX	Linha de GPUs Nvidia RTX
CT	Caso de Teste
CDU	Caso de Uso (abreviação usada nos apêndices)
HTTP	HyperText Transfer Protocol
HTTPS	Secure HyperText Transfer Protocol
HTML	HyperText Markup Language
TLS	Transport Layer Security
ERBD	Escola Regional de Banco de Dados
SBC	Sociedade Brasileira de Computação
IDE	Integrated Development Environment

Sumário

1 INTRODUÇÃO

A tecnologia tem transformado profundamente a forma como as pessoas vivem e interagem com o mundo ao seu redor. Entre as inovações mais significativas dos últimos anos, os assistentes pessoais inteligentes emergiram como interfaces naturais e acessíveis para interação humano-computador. Produtos como Amazon Alexa, Apple Siri e Google Assistant tornaram-se presença constante em milhões de residências, permitindo o controle de dispositivos inteligentes, acesso rápido a informações e automação de tarefas cotidianas através de comandos de voz ou texto. A conveniência oferecida por essas soluções é inegável, porém, sua popularização massiva trouxe consigo preocupações legítimas sobre privacidade, segurança de dados e autonomia tecnológica, uma vez que grandes volumes de informações pessoais são constantemente coletados e transmitidos para servidores remotos controlados por corporações privadas.

Diante desse cenário, o presente trabalho propõe o desenvolvimento da Assistente Pessoal Inteligente Melissa, uma alternativa não proprietária, self-hosted e de código aberto aos assistentes comerciais dominantes no mercado. O problema central que motiva esta pesquisa reside na necessidade de criar uma solução tecnológica que permita aos usuários aproveitarem os benefícios dos assistentes virtuais e dispositivos inteligentes sem comprometer sua privacidade e autonomia sobre seus dados pessoais. A Melissa opera inteiramente em infraestrutura local controlada pelo usuário, garantindo que informações sensíveis nunca deixem o ambiente doméstico sem autorização explícita, ao mesmo tempo em que oferece funcionalidades robustas comparáveis às soluções proprietárias.

O objetivo geral deste projeto é desenvolver um sistema de assistente virtual seguro, confiável e que mantenha a privacidade dos usuários como prioridade máxima. Como objetivos específicos, busca-se implementar funcionalidades de controle de dispositivos inteligentes através de comandos de voz e texto, integrar um Large Language Model de código aberto para interações naturais e contextuais, desenvolver módulos para acesso a informações externas quando autorizado pelo usuário, e criar um dispositivo IoT baseado em microcontrolador para captura e reprodução de áudio. A escolha por software livre e código aberto não é apenas uma decisão técnica, mas também ética, contribuindo para a democratização da tecnologia e empoderamento dos usuários quanto ao controle de seus dados.

Este trabalho justifica-se pela crescente demanda por soluções tecnológicas que respeitem os direitos fundamentais de privacidade dos usuários, especialmente após casos recentes de vazamentos de dados e uso indevido de informações pessoais por grandes empresas de tecnologia. O documento está estruturado apresentando inicialmente a fundamentação teórica sobre privacidade em IoT e inteligência artificial, seguida pela especificação completa do sistema Melissa incluindo arquitetura, tecnologias utilizadas e requisitos funcionais e não funcionais. Posteriormente, são abordadas as estratégias de gerenciamento de riscos, configuração, testes e implantação, finalizando com a apresentação dos resultados obtidos e considerações sobre o desenvolvimento do projeto e suas contribuições para a área de sistemas de informação.

2 FUNDAMENTAÇÃO TEÓRICA

A evolução tecnológica das últimas décadas trouxe consigo uma transformação significativa na forma como as pessoas interagem com dispositivos eletrônicos e sistemas computacionais. Os assistentes virtuais inteligentes emergiram como uma das interfaces mais naturais e acessíveis para essa interação, permitindo que usuários comuns controlem dispositivos, obtenham informações e realizem tarefas complexas utilizando comandos de voz ou texto em linguagem natural. Produtos como Amazon Alexa, Apple Siri, Google Assistant e Microsoft Cortana tornaram-se presentes em milhões de residências ao redor do mundo, integrando-se ao cotidiano das pessoas de forma cada vez mais profunda.

2.1 PRIVACIDADE DOS DADOS NO MEIO DE IOT E INTELIGENCIA ARTIFICIAL

A privacidade dos dados dos usuários de IoT é uma preocupação real. Dispositivos conectados à internet que captam informações como imagens, áudios e/ou vídeos dos ambientes residenciais com a premissa de fornecer segurança, facilidade e automação de tarefas cotidianas devem ser cuidadosamente analisados antes de serem instalados.

> Conforme Pérez Luño, o cidadão moderno está cada vez mais exposto a uma vigilância continuada e inadvertida nos aspectos mais sensíveis da sua vida. Nesse cenário, o tema direito à privacidade apresenta-se sob vários aspectos e, como visto, a internet das coisas também possibilita: a coleta, utilização, acesso, transmissão, processamento, armazenamento e o tratamento de dados pessoais. Destaca-se que os dados podem traduzir aspectos de personalidade, comportamento e preferencias36, que são úteis e de grande valor econômico na iniciativa privada. (Têmis, Gustavo e Demétrio, 2023).

A opção para o usuário final que, desacreditado da privacidade oferecida pela iniciativa privada, ainda deseja fazer a utilização de dispositivos inteligentes em sua residência é, portanto, garantir que suas informações estão trafegando somente em sua rede de comunicação local.

Ademais, a mesma preocupação e solução valem não apenas para o tópico de dispositivos inteligentes, mas também para o recente cenário de uso rotineiro de agentes de inteligência artificial. As soluções proprietárias, como ChatGPT, Claude e Gemini, são de grande valia para o esclarecimento de dúvidas gerais e também específicas, por meio de conversas com linguagem natural. No entanto, a depender do tema da conversa, o usuário fornece dados pessoais para o serviço.

> Outra desvantagem relevante é a existência de, tal como em qualquer outra tecnologia que lida com dados pessoais, um risco de violação de privacidade dos usuários. A IA do Chat GPT pode ser alvo de ataques cibernéticos, algo que eventualmente pode trazer graves consequências para os indivíduos e as organizações. Por essa razão, é importante ser cauteloso no seu uso e aplicação, garantindo que as informações fornecidas sejam verificadas e que os usuários entendam as possíveis limitações, desse modo, deve-se ainda adotar medidas de segurança de modo a proteger informações pessoas e confidenciais. (Margarida, 2023).

Grandes empresas estão sob constante vigilância de órgãos governamentais quanto ao vazamento de informações e utilização de dados sensíveis de maneira antiética. Como o caso em dezembro de 2024 em que “A Autoridade de Proteção de Dados da Itália multou a OpenAI em 15 milhões de euros (aproximadamente R$ 81 milhões) pelo uso indevido de dados pessoais para treinar o ChatGPT” (Leandro, 2024).

Casos recentes de vazamentos de dados, violações de segurança e uso não autorizado de informações pessoais por grandes corporações tecnológicas demonstram que essas preocupações não são meramente teóricas. Escândalos envolvendo o uso indevido de dados pessoais para manipulação política, discriminação algorítmica e vigilância corporativa têm levado autoridades reguladoras ao redor do mundo a implementar legislações mais rigorosas de proteção de dados, como o Regulamento Geral de Proteção de Dados (GDPR) na União Europeia e a Lei Geral de Proteção de Dados (LGPD) no Brasil.

2.2 O PAPEL DO SOFTWARE LIVRE E DE CÓDIGO ABERTO NA PRIVACIDADE DOS DADOS DO USUÁRIO

A certeza da privacidade dos dados do usuário só pode ser obtida através da utilização de softwares de código aberto e hospedagem em infraestrutura controlada. Uma vez que, apenas dessa forma é possível ter conhecimento de todo o comportamento do sistema e o gerenciamento do acesso à internet.

> É óbvio que a privacidade na internet está sendo violada, e também é evidente que a melhor forma de impedir isso não está na proibição, mas na análise dos códigos, pois eles mostram com clareza onde se encontram as fragilidades dos programas e exibem os meios utilizados para a invasão da privacidade alheia. (Cynthia, 2001).

O código aberto dá a oportunidade do usuário alterar o funcionamento, investigar e corrigir vulnerabilidades, além poder submeter suas alterações à comunidade de desenvolvedores.

A possibilidade de auto-hospedagem (self-hosting) é outro benefício crucial do software livre para a privacidade. Em vez de depender de serviços em nuvem controlados por terceiros, usuários podem executar aplicações de código aberto em seus próprios servidores ou computadores pessoais, garantindo que dados sensíveis nunca deixem o ambiente controlado pelo usuário. Essa arquitetura descentralizada reduz drasticamente a superfície de ataque e elimina intermediários potencialmente não confiáveis da cadeia de processamento de dados.

O ecossistema de software livre também promove práticas de desenvolvimento que priorizam a segurança e a privacidade desde a concepção (*security by design* e *privacy by design*). Projetos de código aberto frequentemente implementam criptografia forte, minimização de coleta de dados, anonimização de informações e outras técnicas de proteção como padrão, em vez de recursos opcionais ou posteriores. A revisão por pares inerente ao desenvolvimento de código aberto também tende a produzir código de melhor qualidade e mais seguro do que o desenvolvimento fechado.

2.3 INTELIGENCIA ARTIFICIAL GENERATIVA DE CÓDIGO ABERTO

Grandes modelos de linguagem de código aberto possibilitam que desenvolvedores criem aplicações robustas e seguras que protegem os dados sensíveis dos usuários.

Empresas de grande porte, como a Meta, enxergam benefícios em disponibilizar seus modelos para a comunidade de código aberto e declaram ter intenções de democratizar o acesso a essa tecnologia emergente.

> Com o Llama 3, pretendemos construir os melhores modelos abertos que estejam no mesmo nível dos melhores modelos proprietários disponíveis atualmente. Quisemos considerar o feedback dos desenvolvedores para aumentar a utilidade geral do Llama 3 e estamos fazendo isso enquanto continuamos desempenhando um papel de liderança no uso responsável e na implementação de LLMs (grandes modelos de linguagem). Estamos adotando o ‘espírito’ do código aberto ao lançar de forma antecipada e frequente para permitir que a comunidade tenha acesso a esses modelos enquanto eles ainda estão em desenvolvimento. (Meta, 2024).

Esta filosofia de desenvolvimento aberto e colaborativo tem implicações profundas para a democratização da tecnologia de IA. Ao disponibilizar modelos de linguagem sofisticados sob licenças permissivas, empresas como a Meta permitem que desenvolvedores, pesquisadores e organizações de todos os tamanhos experimentem, modifiquem e implementem soluções de IA sem depender de APIs

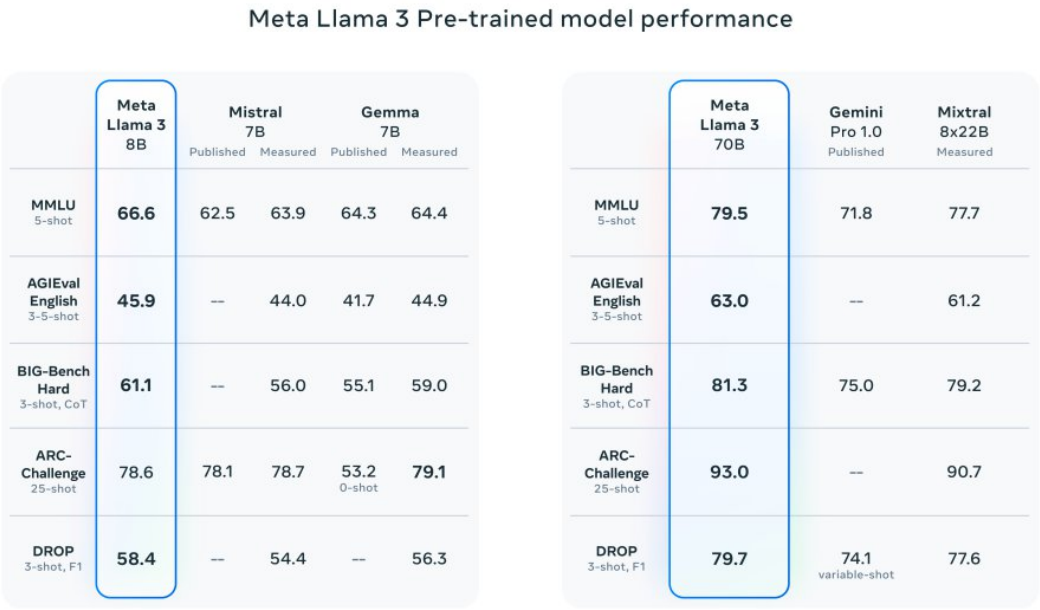
proprietárias ou serviços em nuvem controlados por terceiros. Isso é particularmente importante para aplicações que envolvem dados sensíveis, como assistentes pessoais, sistemas de saúde, ferramentas educacionais e aplicações empresariais que lidam com informações confidenciais.

Além do Llama, outras organizações têm contribuído significativamente para o ecossistema de IA de código aberto. O projeto Mistral AI, por exemplo, desenvolveu modelos altamente eficientes que competem com soluções proprietárias em diversos benchmarks de desempenho. A Google também lançou a família Gemma de modelos abertos, permitindo maior experimentação e inovação pela comunidade global de desenvolvedores. Essa diversidade de opções de código aberto cria um ambiente competitivo saudável que beneficia os usuários finais através de melhor desempenho, maior transparência e custos reduzidos.

A execução local desses modelos tornou-se viável graças a ferramentas como o Ollama, um software de código aberto que simplifica drasticamente o processo de download, instalação e execução de LLMs em computadores pessoais. O Ollama oferece uma interface unificada para interagir com diversos modelos de linguagem, abstraindo as complexidades técnicas de configuração e gerenciamento de memória, permitindo que mesmo usuários sem expertise profundo em aprendizado de máquina possam executar modelos sofisticados de IA em suas próprias máquinas.

A capacidade de executar modelos de IA localmente tem implicações profundas para a privacidade. Quando um LLM é executado inteiramente no computador do usuário, todas as consultas, conversas e dados processados permanecem sob controle exclusivo desse usuário, sem nunca serem transmitidos através da internet para servidores de terceiros. Isso elimina completamente o risco de interceptação de dados em trânsito, acesso não autorizado por funcionários de empresas fornecedoras, uso de dados para treinamento de modelos futuros sem consentimento explícito, ou cumprimento de ordens judiciais ou governamentais que poderiam comprometer a privacidade do usuário.

Figura 1 - Benchmark de desempenho do Llama com outros modelos



Fonte: Meta (2024)

3 ESPECIFICAÇÃO DO SISTEMA

3.1 DESCRIÇÃO DO(S) PROBLEMA(S)

A tecnologia tem crescido dia após dia, trazendo novas ferramentas e utilidades que tornam a vida das pessoas mais prática. Entre essas novidades, destacam-se os dispositivos de Internet das Coisas (IoT) e os assistentes virtuais baseados em Inteligência Artificial (IA), que estão cada vez mais presentes dentro das casas.

Apesar de tantas facilidades, esse avanço também traz preocupações. Muitos desses dispositivos captam informações pessoais, como áudios, vídeos e dados de comportamento dos usuários, sem que eles tenham total controle sobre o que é coletado ou para onde essas informações são enviadas. Além disso, grande parte das soluções disponíveis no mercado funciona mediante servidores externos, o que aumenta os riscos de vazamento de dados e uso indevido das informações.

Outro ponto importante é que os assistentes de IA mais populares são atualmente fechados e pertencem a grandes empresas, o que dificulta a verificação do que é feito com os dados fornecidos. Casos recentes de vazamentos e multas mostram que a preocupação com a privacidade é real e precisa ser levada a sério.

Por isso, existe a necessidade de criar uma solução que permita ao usuário aproveitar as facilidades da tecnologia, mas sem abdicar da segurança e da privacidade dos seus dados. A proposta é desenvolver um sistema aberto, que rode em infraestrutura local, dê autonomia ao usuário e permita o controle total sobre as informações trafegadas.

3.2 PROPOSTA DE SOLUÇÃO

Diante dos problemas levantados, a proposta deste projeto é desenvolver um sistema de assistente virtual que seja seguro, confiável e que mantenha a privacidade dos usuários como prioridade. O sistema será baseado em software livre e de código aberto, para que os usuários possam auditar e controlar todo o funcionamento interno da solução, além de também colaborar com o projeto trazendo novas soluções e melhorias.

O servidor da assistente será executado em infraestrutura local, sem acesso obrigatório à internet, para que os dados pessoais dos usuários não sejam enviados para servidores de terceiros. O usuário terá a opção de permitir o acesso temporário

à internet apenas quando desejar usar funcionalidades externas, como previsão do tempo ou consulta de feriados.

Essa solução visa atender a um público que valoriza a tecnologia, mas não renuncia à proteção de seus dados pessoais, criando um modelo de assistente virtual que respeita a privacidade do usuário desde a sua concepção.

3.3 PARTICIPANTES DO PROJETO

O projeto está sendo desenvolvido por Danilo Maia Florenzano e João Victor Soares Jordão, ambos estudantes do curso de Sistemas de Informação na Universidade Dom Bosco em Resende-RJ. Os participantes atuam profissionalmente como desenvolvedores Back End, com experiência focada na plataforma .NET.

Durante o desenvolvimento do trabalho de conclusão de curso, ambos dividiram as responsabilidades técnicas e de documentação, aplicando seus conhecimentos práticos para a construção de uma solução bem estruturada e alinhada com os princípios de segurança e privacidade estudados.

Além de desempenharem o papel de desenvolvedores, também atuaram como stakeholders principais do projeto, sendo responsáveis por definir os requisitos, validar as funcionalidades e garantir a aderência do sistema aos objetivos propostos.

O projeto conta com a orientação acadêmica do Professor Rafael Chiarelli Júnior, que acompanha o progresso técnico e prático do trabalho, oferecendo suporte metodológico durante o desenvolvimento. Além disso, a Professora Mônica Mara atua na avaliação da parte teórica do projeto, sendo responsável por analisar a estrutura, a fundamentação e a qualidade acadêmica dos documentos elaborados.

3.4 REQUISITOS FUNCIONAIS

Os requisitos funcionais descrevem as funcionalidades que o sistema proposto deve ser capaz de executar, atendendo diretamente às necessidades dos usuários finais. Esses requisitos garantem que a assistente virtual possa realizar tanto interações básicas quanto ações mais específicas, integrando funcionalidades de automação residencial e organização de tarefas pessoais. A seguir, estão detalhados os principais requisitos funcionais levantados para o projeto:

RF01 - Resposta a Perguntas Genéricas

- A assistente deve ser capaz de responder a perguntas gerais formuladas pelo usuário em linguagem natural.
- O sistema deve utilizar o modelo de linguagem (LLM) configurado para processar e gerar respostas contextualizadas.
- As respostas devem ser coerentes, relevantes ao contexto da pergunta e apresentadas em linguagem natural fluente.
- O sistema deve manter o histórico da conversa para permitir referências a interações anteriores.
- Exemplos de perguntas genéricas incluem: "Qual a capital do Brasil?", "Como funciona a fotossíntese?", "Me conte uma curiosidade interessante".

RF02 - Acesso a Ferramentas Específicas

- A assistente deve identificar quando uma pergunta requer acesso a ferramentas especializadas além do conhecimento geral do LLM.
- O sistema deve implementar mecanismos de roteamento inteligente que direcionem consultas específicas para as ferramentas apropriadas.
- As ferramentas devem ser modulares e extensíveis, permitindo a adição de novas funcionalidades sem modificar o núcleo do sistema.
- A integração entre o modelo de linguagem e as ferramentas deve ser transparente para o usuário final.

RF02.1 - Consulta de Informações sobre Feriados

- A assistente deve permitir consultas sobre feriados nacionais, estaduais e municipais do Brasil.

- O usuário deve poder consultar feriados por período (mês específico), localização (cidade ou estado) ou nome do feriado.
- O sistema deve utilizar web scraping ou APIs públicas para obter informações atualizadas sobre feriados.
- As respostas devem incluir a data do feriado, sua natureza (nacional, estadual ou municipal) e, quando aplicável, informações adicionais sobre o significado do feriado.
- Exemplos de consultas: "Quais são os feriados em outubro?", "Quando é o Dia da Consciência Negra em São Paulo?", "Há feriados em Resende no mês de junho?".

RF02.2 - Consulta de Previsão do Tempo

- A assistente deve fornecer informações meteorológicas atualizadas para cidades brasileiras.
- O usuário deve poder consultar a previsão para o dia atual ou dias futuros.
- As informações fornecidas devem incluir temperatura atual, máxima e mínima, condições climáticas (sol, chuva, nublado), umidade e probabilidade de precipitação.
- O sistema deve utilizar web scraping de fontes confiáveis ou APIs públicas de meteorologia.
- A assistente deve ser capaz de interpretar consultas em linguagem natural como "Vai chover hoje?", "Como está o tempo em Resende?", "Qual a temperatura máxima para amanhã?".

RF03 - Gerenciamento de Tarefas Pessoais

- O sistema deve permitir que o usuário organize suas atividades através de um sistema completo de gerenciamento de tarefas.

- Todas as tarefas devem ser armazenadas de forma persistente no banco de dados MySQL.
- As funcionalidades de gerenciamento devem ser acessíveis através de comandos em linguagem natural.

RF03.1 - Agendamento de Tarefas

- A assistente deve permitir o cadastro de novas tarefas através de comandos de voz ou texto.
- O usuário deve poder especificar título da tarefa, descrição detalhada, data de vencimento e prioridade.
- O sistema deve confirmar o agendamento e fornecer um identificador único para a tarefa criada.
- Tarefas podem ser criadas com comandos como "Agende uma reunião com a equipe para sexta-feira às 14h", "Adicione uma tarefa para comprar mantimentos", "Lembre-me de ligar para o dentista amanhã".

RF03.2 - Consulta de Tarefas Agendadas

- A assistente deve permitir consultas sobre tarefas pendentes utilizando diferentes critérios de filtro.
- O usuário deve poder listar todas as tarefas, tarefas de um período específico, ou tarefas com determinada prioridade.
- As consultas devem retornar informações completas sobre cada tarefa: título, descrição, data de vencimento, prioridade e status.
- Exemplos de consultas: "Quais são minhas tarefas para hoje?", "Mostre todas as tarefas pendentes desta semana", "Tenho algo importante agendado?".

RF03.3 - Marcação de Tarefas como Concluídas

- O usuário deve poder marcar tarefas como concluídas através de comandos naturais.
- O sistema deve identificar a tarefa através de seu título, identificador único ou referência contextual.
- Tarefas concluídas devem ter seu status atualizado no banco de dados com data e hora da conclusão.
- Comandos como "Marquei como concluída a reunião com a equipe", "Concluí a tarefa de comprar mantimentos", "Finalizei todas as tarefas de hoje" devem ser interpretados corretamente.

RF03.4 - Consulta de Tarefas Concluídas

- A assistente deve permitir consultas sobre tarefas já finalizadas para fins de histórico e acompanhamento.
- O usuário deve poder filtrar tarefas concluídas por período ou buscar tarefas específicas.
- As informações retornadas devem incluir data de conclusão além dos dados originais da tarefa.

RF04 - Controle de Dispositivos Domésticos

- O sistema deve implementar funcionalidades de automação residencial através do controle de dispositivos inteligentes.
- A arquitetura deve ser modular para facilitar a adição de novos tipos de dispositivos no futuro.
- Todos os comandos de controle devem ser interpretados a partir de linguagem natural.

RF04.1 - Controle de Ar-Condicionado

- A assistente deve ser capaz de ligar e desligar aparelhos de ar-condicionado específicos.
- O usuário deve poder identificar qual aparelho deseja controlar quando há múltiplos dispositivos.
- O sistema deve permitir ajuste de temperatura dentro dos limites operacionais do aparelho (tipicamente 16°C a 30°C).
- Comandos como "Ligue o ar-condicionado da sala", "Desligue o ar do quarto", "Coloque a temperatura em 22 graus", "Abaixe a temperatura para 20" devem ser interpretados corretamente.
- O sistema deve confirmar a execução do comando e informar o novo estado do dispositivo.

RF05 - Comunicação Multimodal

- O sistema deve suportar múltiplas formas de entrada e saída de informações para atender diferentes contextos de uso e preferências dos usuários.

RF05.1 - Entrada por Texto

- A assistente deve aceitar comandos e perguntas digitados pelo usuário através de interface de terminal ou aplicativo cliente.
- O processamento de texto deve ser em tempo real, com resposta imediata após o envio da mensagem.
- Mensagens de texto devem ser armazenadas no histórico de conversação.

RF05.2 - Entrada por Áudio

- O sistema deve capturar áudio através de microfone em dispositivos clientes.

- A captura deve utilizar streaming de áudio em tempo real para o servidor.
- O áudio capturado deve ser automaticamente transcrito para texto utilizando modelos de reconhecimento de voz (como Whisper.net).
- O sistema deve ser capaz de lidar com diferentes qualidades de áudio e ruídos de fundo razoáveis.

RF05.3 - Saída por Texto

- As respostas da assistente devem ser apresentadas em formato de texto legível e bem formatado.
- O texto deve ser exibido progressivamente à medida que é gerado pelo modelo de linguagem, proporcionando feedback visual imediato.

RF05.4 - Saída por Áudio

- O sistema deve converter as respostas textuais em áudio sintetizado utilizando tecnologia de text-to-speech.
- O áudio sintetizado deve ser transmitido via streaming para os dispositivos clientes para reprodução.
- A qualidade do áudio deve ser clara e compreensível, com prosódia natural.

RF06 - Arquitetura Cliente-Servidor

- O sistema deve implementar uma arquitetura distribuída com clara separação entre componentes cliente e servidor.
- O servidor central deve concentrar toda a lógica de negócio, processamento de IA e acesso a dados.
- Clientes devem ser responsáveis apenas pela captura de entrada do usuário e apresentação de respostas.

- A comunicação deve utilizar protocolos padronizados que permitam desenvolvimento de diferentes tipos de clientes.

RF07 - Cliente IoT com Capacidades de Áudio

- O sistema deve incluir ao menos um cliente baseado em dispositivo IoT (microcontrolador).
- O dispositivo deve integrar capacidades de gravação de áudio através de microfone.
- O dispositivo deve ter capacidade de reprodução de áudio através de alto-falantes ou amplificadores.
- A comunicação com o servidor deve ser realizada através de rede Wi-Fi.
- O dispositivo deve ser compacto e adequado para instalação em ambiente doméstico.

RF08 - Criptografia Configurável

- A comunicação entre clientes e servidor deve suportar criptografia TLS/HTTPS para proteção de dados em trânsito.
- A criptografia deve estar habilitada por padrão para garantir segurança desde a instalação inicial.
- O usuário deve poder desabilitar a criptografia através de configuração, sem necessidade de reinicialização do servidor.
- Esta flexibilidade permite otimização de desempenho em redes locais fechadas onde a criptografia possa não ser crítica.

RF09 - Controle de Acesso à Internet

- O servidor deve operar por padrão em modo offline, sem realizar requisições à internet.

- O usuário deve poder habilitar acesso à internet de forma granular para ferramentas específicas.
- Ferramentas que dependem de internet (como previsão do tempo e consulta de feriados) devem funcionar apenas quando o acesso estiver explicitamente autorizado.
- A configuração de acesso à internet deve ser dinâmica, sem necessidade de reinicialização do servidor.
- O sistema deve registrar em logs todas as tentativas de acesso à internet para auditoria.

RF10 - Seleção Dinâmica de Modelo de Linguagem

- O usuário deve poder alternar entre diferentes modelos de linguagem (LLMs) disponíveis no sistema.
- A troca de modelo deve ser realizada através de configuração acessível, sem necessidade de modificar código ou reiniciar o servidor.
- O sistema deve suportar modelos de diferentes tamanhos e capacidades para atender a diferentes perfis de hardware.
- Exemplos de modelos suportados: Llama 3 (diversos tamanhos), Mistral, Gemma, entre outros compatíveis com Ollama.
- A seleção do modelo deve considerar o equilíbrio entre capacidade de resposta e recursos computacionais disponíveis.

RF11 - Histórico de Conversação

- O sistema deve registrar automaticamente todo o histórico de interações entre usuário e assistente.
- O histórico deve incluir tanto as mensagens do usuário quanto as respostas da assistente.

- Conversas devem ser armazenadas com timestamps para permitir recuperação temporal.
- O histórico deve ser persistido no banco de dados MySQL para sobreviver a reinicializações do sistema.

RF12 - Exportação de Histórico

- O usuário deve poder exportar o histórico completo de conversações.
- A exportação deve estar disponível em formatos acessíveis como texto plano, JSON ou através de envio por email.
- O comando de exportação deve ser interpretado em linguagem natural, como "Envie meu histórico de conversas por email".

3.5 REQUISITOS NÃO FUNCIONAIS

Os requisitos não funcionais descrevem como o sistema deve se comportar, estabelecendo padrões de desempenho, segurança, qualidade e estrutura de desenvolvimento. Embora não definam funcionalidades específicas, são essenciais para garantir que o sistema atenda às expectativas de estabilidade, privacidade e facilidade de manutenção. A seguir, são detalhados os principais requisitos não funcionais do projeto:

RNF01 - Requisitos de Hardware para Servidor

Configuração Recomendada:

- Placa de vídeo Nvidia GTX 1060 ou superior, ou RTX de qualquer geração, para aceleração de processamento de modelos de IA.
- Alternativamente, placa AMD compatível com ROCm para execução de modelos de linguagem.

- Processador Intel Core i5 de 7ª geração ou superior, ou AMD Ryzen 5 ou superior.
- Memória RAM de 16GB ou mais para modelos menores (até 7B parâmetros), 32GB recomendado para modelos maiores.
- Armazenamento SSD com no mínimo 100GB livres para sistema operacional, contêineres e modelos de IA.

Configuração Mínima:

- Processador Intel Core i5 de 7ª geração, Intel Xeon de 4ª geração, ou equivalente AMD.
- Processamento via CPU sem GPU dedicada, resultando em tempos de resposta significativamente maiores.
- Memória RAM de 8GB para modelos muito pequenos (até 3B parâmetros), com limitações de desempenho.
- A configuração mínima permite operação funcional, mas não é recomendada para uso regular devido à experiência de usuário comprometida.

RNF02 - Requisitos de Hardware para Cliente IoT

- O dispositivo cliente deve ser baseado em microcontrolador de baixo custo e ampla disponibilidade.
- Microcontroladores suportados: ESP32, ESP8266 ou Arduino com módulo Wi-Fi compatível.
- O dispositivo deve incluir módulo de microfone eletreto ou MEMS para captura de áudio.
- Amplificador de áudio (como PAM8403 ou MAX98357A) para reprodução através de alto-falantes.
- Alto-falantes de 3W a 5W com impedância de 4 ou 8 ohms.

- Fonte de alimentação adequada para fornecer corrente suficiente para todos os componentes (tipicamente 5V 2A).
- Opcionalmente, bateria recarregável para operação sem fio completa.

RNF03 - Robustez e Tolerância a Falhas

- O sistema deve implementar tratamento robusto de exceções em todos os componentes críticos.
- Falhas na geração de respostas pelo modelo de linguagem não devem causar travamento (crash) do servidor.
- Erros na execução de ferramentas específicas devem ser capturados e reportados de forma controlada ao usuário.
- O sistema deve implementar mecanismos de retry automático para falhas temporárias de comunicação.
- Timeouts adequados devem ser configurados para todas as operações de rede para evitar bloqueios indefinidos.
- O servidor deve ser capaz de se recuperar graciosamente de falhas parciais, mantendo outros componentes operacionais.
- Falhas de comunicação entre cliente IoT e servidor não devem corromper o estado do sistema.

RNF04 - Observabilidade e Logging

- O sistema deve implementar geração estruturada de logs para registro de eventos, erros e atividades do sistema.
- Diferentes níveis de logging devem ser suportados: DEBUG, INFO, WARNING, ERROR, CRITICAL.
- Logs devem incluir timestamps precisos, identificação do componente gerador e contexto relevante.

- Logs de erro devem incluir stack traces completos para facilitar diagnóstico de problemas.
- O sistema deve registrar todas as interações com o modelo de linguagem, incluindo prompts enviados e respostas recebidas.
- Logs devem ser rotacionados automaticamente para evitar crescimento ilimitado de arquivos.
- Embora não seja requisito crítico para operação básica, o logging é essencial para manutenção e evolução do sistema.

RNF05 - Disponibilização através de Contêineres

- Todo o servidor deve ser disponibilizado utilizando tecnologia de contêineres Docker.
- A orquestração dos serviços deve ser realizada através de arquivo docker-compose.yml.
- O arquivo docker-compose deve definir todos os serviços necessários: servidor web, Ollama, banco de dados MySQL.
- Volumes Docker devem ser utilizados para persistência de dados entre reinicializações.
- A configuração de rede entre contêineres deve ser isolada e segura.
- Variáveis de ambiente devem ser utilizadas para configurações sensíveis como senhas de banco de dados.
- O arquivo docker-compose deve incluir políticas de restart automático para resiliência.

RNF06 - Licenciamento Open Source

- Todo o código-fonte produzido deve ser disponibilizado sob licença MIT.

- A licença MIT permite uso comercial, modificação, distribuição e uso privado sem restrições significativas.
- O código deve ser hospedado em repositório público no GitHub para máxima acessibilidade.
- Documentação completa deve acompanhar o código, incluindo instruções de instalação, configuração e contribuição.
- A escolha da licença MIT facilita adoção e contribuições da comunidade sem barreiras legais.

RNF07 - Segurança e Privacidade
Isolamento de Rede:

- Cliente e servidor devem ser capazes de operar completamente offline por padrão.
- Acesso à internet deve ser explicitamente autorizado pelo usuário para funcionalidades específicas.
- O sistema deve implementar whitelist de domínios permitidos quando internet estiver habilitada.
- Requisições HTTP/HTTPS para internet devem ser registradas em logs para auditoria.

Proteção de Dados Pessoais:

- Nenhuma informação pessoal deve ser transmitida para servidores externos sem consentimento explícito.
- Dados de conversa devem ser armazenados exclusivamente no banco de dados local.
- O sistema não deve implementar telemetria, analytics ou qualquer forma de rastreamento de uso.

- Backups de dados devem ser responsabilidade do usuário, sem sincronização automática com nuvem.
  Comunicação Segura:

- Comunicação entre cliente e servidor deve suportar criptografia TLS 1.2 ou superior.

- Certificados autoassinados ou de autoridades certificadoras podem ser utilizados conforme preferência do usuário.

- O sistema deve alertar sobre comunicação não criptografada quando aplicável.

RNF08 - Princípios de Design de Software

- A arquitetura do servidor deve seguir princípios SOLID de design orientado a objetos.

- **Princípio Aberto/Fechado (Open/Closed Principle):**

    - Componentes devem estar abertos para extensão, mas fechados para modificação.

    - Novas ferramentas e funcionalidades devem poder ser adicionadas sem alterar código existente.

    - Utilização de interfaces e abstrações para permitir polimorfismo e extensibilidade.

- **Inversão de Dependência (Dependency Inversion Principle):**

    - Módulos de alto nível não devem depender de módulos de baixo nível; ambos devem depender de abstrações.

    - Abstrações não devem depender de detalhes; detalhes devem depender de abstrações.

    - Utilização de injeção de dependência para desacoplamento entre componentes.

- **Responsabilidade Única (Single Responsibility Principle):**
    - Cada classe deve ter apenas uma razão para mudar.
    - Separação clara entre lógica de apresentação, lógica de negócio e acesso a dados.
- **Segregação de Interface (Interface Segregation Principle):**
    - Clientes não devem ser forçados a depender de interfaces que não utilizam.
    - Interfaces específicas e coesas são preferíveis a interfaces grandes e genéricas.

RNF09 - Manutenibilidade

- O código deve seguir convenções de nomenclatura consistentes e idiomáticas do C#.
- Métodos e classes devem ter nomes descritivos que comunicam claramente sua função.
- Comentários devem ser utilizados para explicar decisões de design complexas, não para descrever código óbvio.
- O código deve ser formatado consistentemente utilizando ferramentas de formatação automática.
- Complexidade ciclomática deve ser mantida baixa, com métodos pequenos e focados.
- Duplicação de código deve ser minimizada através de abstração e reutilização.

RNF10 - Testabilidade

- O código deve ser estruturado de forma a facilitar testes automatizados.
- Dependências externas (banco de dados, APIs, modelo de IA) devem ser abstraídas através de interfaces para permitir mocking em testes.
- Testes unitários devem cobrir lógica de negócio crítica.
- Testes de integração devem verificar a comunicação adequada entre componentes.
- Casos de teste documentados devem ser mantidos e atualizados conforme o sistema evolui.

RNF11 - Performance

- O tempo de resposta do sistema deve ser aceitável considerando as limitações de hardware.
- Em hardware recomendado (com GPU), respostas simples devem ser geradas em menos de 5 segundos.
- Consultas a ferramentas externas (web scraping) devem ter timeout configurável, tipicamente entre 10-30 segundos.
- O sistema deve utilizar streaming de respostas para fornecer feedback progressivo ao usuário.
- Operações de banco de dados devem ser otimizadas com índices apropriados.
- Cache pode ser implementado para dados que mudam infrequentemente (como feriados).

RNF12 - Escalabilidade

- A arquitetura deve permitir escalabilidade vertical através de adição de recursos computacionais.
- Múltiplos clientes devem poder conectar-se simultaneamente ao servidor.
- O sistema deve gerenciar adequadamente recursos de memória mesmo sob carga de múltiplas requisições.
- Embora otimizado inicialmente para uso doméstico/familiar, a arquitetura não deve impedir evolução futura para cenários de maior escala.

RNF13 - Portabilidade

- O servidor deve ser executável em diferentes sistemas operacionais: Linux (várias distribuições), Windows, macOS.
- A utilização de Docker garante consistência de ambiente independentemente da plataforma hospedeira.
- O código não deve conter dependências específicas de plataforma quando alternativas multiplataforma existirem.
- Configurações específicas de plataforma devem ser externalizadas e documentadas claramente.

RNF14 - Usabilidade

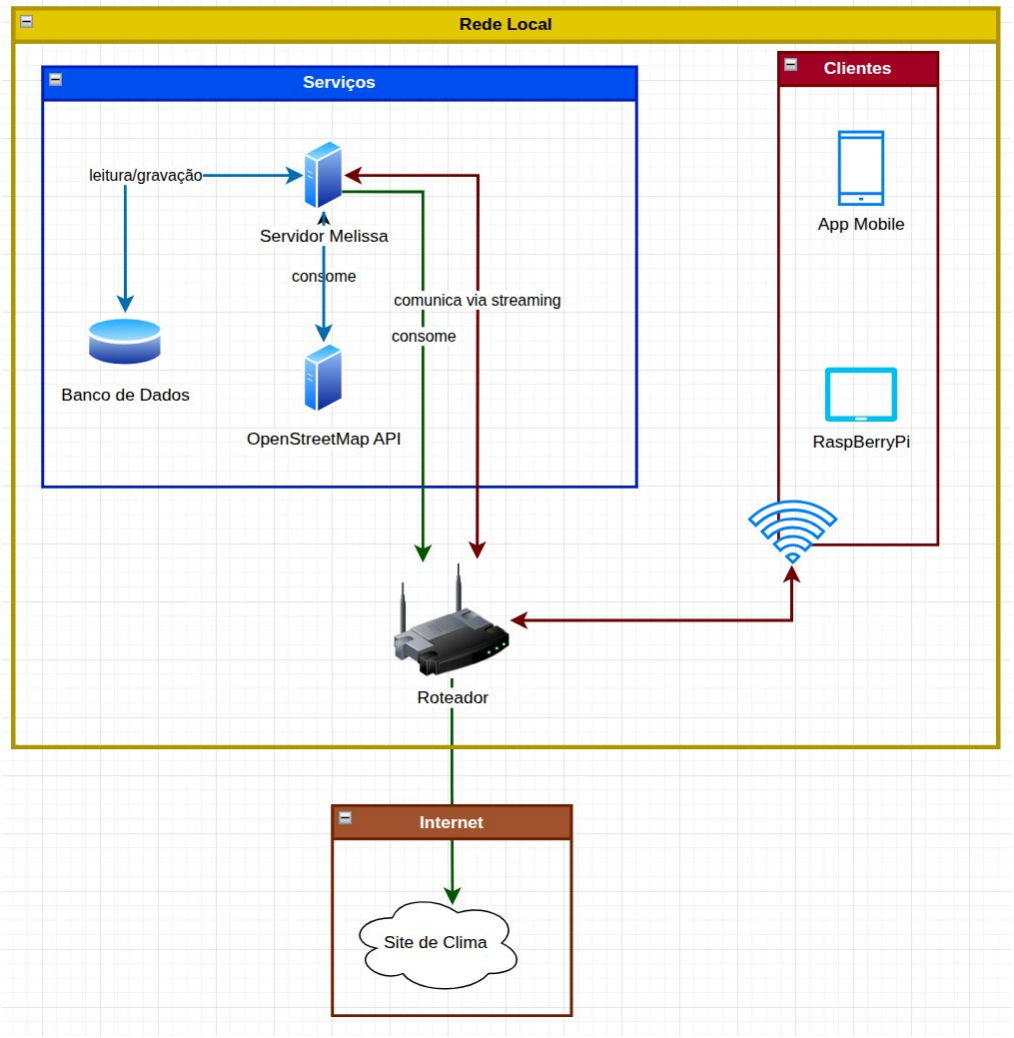
- A interface de comunicação deve aceitar comandos em linguagem natural fluente, não exigindo sintaxe específica.
- Mensagens de erro devem ser claras e orientar o usuário sobre como corrigir problemas.
- O sistema deve fornecer feedback adequado para todas as ações do usuário.

- A documentação do usuário deve ser acessível e compreensível para público não técnico.
- O processo de instalação deve ser simplificado ao máximo através do uso de Docker e scripts de configuração.

3.6 ARQUITETURA ESTRUTURAL DO SISTEMA

A arquitetura do sistema (Figura 2) foi planejada para garantir segurança, modularidade e facilidade de manutenção, respeitando os princípios da engenharia de software moderna. O sistema será dividido em duas grandes partes: servidor e cliente.

Figura 2 – Diagrama da arquitetura e comunicação do sistema



Fonte: Do próprio autor

1.
2.
3.
    3.1.
    3.2.
    3.3.
    3.4.
    3.5.
    3.6.

3.6.1. Estrutura Física

- **Servidor**: será executado em uma máquina local. Essa máquina será responsável por executar a assistente virtual, gerenciar tarefas, controlar dispositivos e disponibilizar APIs para interação com os clientes.
- **Cliente IoT**: dispositivo físico baseado em um minicomputador (RaspBerryPi) responsável por captar áudio do ambiente e enviar comandos ao servidor, além de reproduzir áudios de resposta.

3.6.2. Estrutura Lógica

A arquitetura lógica do sistema é dividida em três componentes principais: Cliente, Web Server e Core, cada um com responsabilidades bem definidas para garantir uma interação eficiente e organizada.

1. **Cliente**: O Cliente é a interface com o usuário, onde ele realiza as interações e envia os inputs. O Cliente se comunica diretamente com o Web Server, enviando solicitações e recebendo respostas.
2. **WebServer**: O Web Server é responsável por processar os inputs recebidos do Cliente, instanciando o modelo de Inteligência Artificial (IA) e interpretando as requisições. Ele direciona a solicitação para o Core de acordo com a natureza do pedido e aciona as ferramentas ou funções necessárias.
3. **Core**: O Core é o coração do sistema, contendo as regras de negócios e ferramentas específicas, como a previsão do tempo via Web Scraping, busca de feriados, e outras funcionalidades. O Core é chamado pelo Web Server para executar as funções necessárias, seja consultando uma API externa ou realizando cálculos internos.

3.6.3. Fluxo de Funcionamento

1. O Cliente envia um input (como uma pergunta ou comando) para o Web Server.

2. O **WebServer** recebe o input, instancia o modelo de IA e interpreta o comando do usuário.
3. Dependendo da solicitação, o Web Server aciona o Core, que possui as classes e métodos necessários para realizar a ação (como fazer uma consulta de previsão do tempo ou buscar feriados).
4. O Core executa a ação, retorna o resultado para o Web Server, que por sua vez envia a resposta de volta para o Cliente.

Essa divisão entre Cliente, Web Server e Core permite a escalabilidade e manutenção do sistema, além de uma clara separação das responsabilidades de cada componente.

3.6.4. Comunicação

A comunicação entre os componentes do sistema é essencial para garantir uma experiência fluida e eficiente. O sistema é dividido em três partes principais: Cliente, WebServer e Core, e a comunicação entre eles é feita de forma segura e eficiente para processar as requisições do usuário.

3.6.5. Comunicação entre Cliente e Web Server:

- **Protocolo de Comunicação (SignalR)**: A comunicação entre o Cliente e o WebServer é realizada por meio de SignalR, permitindo comunicação em tempo real. O Cliente envia mensagens de texto ou áudio, e o WebServer responde com as respostas apropriadas, seja em formato de texto ou áudio. A comunicação é contínua, com mensagens e áudios enviados e recebidos em tempo real, sem a necessidade de requisições HTTP adicionais.
- **Fluxo de Áudio (Streaming)**: O Cliente envia áudio para o WebServer via streaming. O WebServer recebe o áudio em tempo real, processa a transcrição utilizando Whisper.net, envia o texto transcrito para o Core, e então retorna a resposta ao **Cliente** também via streaming de áudio.

3.6.6. Comunicação entre WebServer e Core:

- **Interação com o Core**: A comunicação entre o WebServer e o Core é feita por meio do objeto Assistant, que é instanciado no Core. No caso específico da assistente Melissa, o WebServer envia as perguntas para a instância de

Melissa através do método Ask(). Esse método envolve a lógica para verificar a criptografia, se habilitada, e passa a pergunta para o Chat associado à Melissa.

- **Uso de Ferramentas e Lógica de Negócio**: O Core contém ferramentas como GetWeatherByLocationTool, GetBrazilianHolidaysTool e GetHolidayDateByNameTool, que são utilizadas para responder perguntas específicas. Quando o WebServer recebe uma pergunta relacionada a essas ferramentas, ele a encaminha para o Core por meio do método Ask(), que faz a consulta apropriada e retorna a resposta.
- **Modelo de IA**: O Core utiliza um modelo de IA (definido no construtor de Melissa) para processar as perguntas gerais. Quando o WebServer envia uma pergunta, a Melissa interage com o modelo de IA e, com base na lógica de negócios, decide qual ferramenta utilizar ou como gerar uma resposta usando o modelo.

3.6.7. Comunicação com Ferramentas Externas

- **Integração com APIs Externas**: O Core pode se comunicar com APIs externas para obter dados conforme necessário.
- **Web Scraping**: Algumas ferramentas no Core podem usar web scraping para obter informações de sites externos, como datas de feriados e/ou previsão do tempo, para fornecer uma resposta precisa ao WebServer.

3.7 LOGOTIPO

O logotipo do projeto Melissa foi criado como uma homenagem afetiva. O nome “Melissa” foi escolhido em referência à cachorrinha de estimação de um dos desenvolvedores, representando o carinho e a história pessoal que inspiraram o projeto. Além de refletir essa ligação emocional, o logotipo também busca transmitir a ideia de segurança, proteção e confiança, que são pilares fundamentais da proposta do sistema.

O estilo visual integra elementos simbólicos marcantes como: um cadeado em formato de “M”, fazendo referência ao nome *Melissa* (nome do projeto), com traços tecnológicos que evocam os conceitos de Segurança e Inteligência Artificial; e as

orelhinhas de cachorro, como homenagem à história afetiva que inspirou o nome do projeto.

Figura 3 - Logotipo do Projeto

Fonte: Gerado por inteligência artificial (ChatGPT, 2024)

3.8 TECNOLOGIAS E FERRAMENTAS UTILIZADAS

Durante o desenvolvimento do projeto, foram utilizadas diversas tecnologias e ferramentas que garantem o bom funcionamento do sistema, bem como o atendimento aos requisitos de segurança, privacidade e eficiência. A seguir, detalham-se as principais escolhas:

- **C# (.NET)**:

Figura 4 - Logo C# .NET

Fonte: Fischer (online)

C# é uma linguagem de programação orientada a objetos desenvolvida pela Microsoft para a plataforma .NET, amplamente utilizada no desenvolvimento de aplicações desktop, web e mobile. Utilizamos dessa linguagem e plataforma para todo o desenvolvimento da nossa aplicação. Embora o projeto tenha sido iniciado em Python, optou-se pela migração para C# devido à maior familiaridade da equipe com essa linguagem, o que reduziu significativamente a curva de aprendizado. Essa mudança proporcionou um desenvolvimento mais ágil, consistente e alinhado às competências do time, permitindo que o foco fosse direcionado ao domínio de novas tecnologias que agregam valor ao projeto.

- **Modelo de Linguagem Llama (executado via Ollama)**:

Figura 5 - Logo Ollama

Fonte: Jude (2024)

Llama é um modelo de IA LLM (Large Language Model) desenvolvido pela Meta (anteriormente Facebook), lançado em 2023 como uma alternativa de código aberto aos modelos proprietários. O Llama foi projetado para processamento de linguagem natural, sendo capaz de compreender e gerar texto de forma contextualizada, com versões que variam em tamanho e capacidade de parâmetros. Já o Ollama é uma ferramenta de código aberto desenvolvida para facilitar a execução local de modelos de IA LLM, como o Llama, Mistral e outros, diretamente em computadores pessoais. O Ollama simplifica o processo de instalação, configuração e gerenciamento desses modelos, eliminando a necessidade de infraestrutura em nuvem e permitindo que desenvolvedores executem modelos de linguagem de forma privada e offline.

Para a comunicação em linguagem natural, está sendo utilizado um modelo da família Llama, executado localmente através do Ollama. Essa escolha assegura controle total sobre o processamento de dados sensíveis, em linha com os princípios de privacidade definidos no projeto.

- **Web Scraping**:

Para obtenção de informações externas, como previsão do tempo e feriados nacionais, estão sendo desenvolvidos módulos de Web Scraping. Web Scraping é uma técnica automatizada de extração de dados de páginas web, que analisa o código HTML de sites para coletar informações específicas de

forma estruturada. Essa abordagem permite a captura de dados de fontes públicas confiáveis sem a necessidade de integração direta com APIs de terceiros, oferecendo maior autonomia e flexibilidade ao sistema.

- **Banco de Dados**:

Figura 6 - MySQL Logo

Fonte: Logos-world (2025)

O sistema utiliza um banco de dados MySQL para armazenamento de informações. MySQL é um sistema de gerenciamento de banco de dados relacional (SGBD) de código aberto, amplamente utilizado em aplicações web por sua robustez e desempenho. Por meio dele, são armazenadas informações como tarefas agendadas e registros de atividades, valorizando a segurança, a confiabilidade e a facilidade de manutenção dos dados.

- **Dispositivo IoT - Arduino:**

Figura 7 - Arduino Logo

Fonte: Wikimedia Commons (2024)

No componente IoT, está sendo utilizado um microcontrolador Arduino para interação física com o usuário. Arduino é uma plataforma de prototipagem eletrônica de código aberto, composta por hardware (placa de circuito com microcontrolador) e software (IDE de programação), amplamente utilizada em projetos de automação e IoT. No sistema, o Arduino tem como funções principais a captação de comandos de áudio e a reprodução de respostas sonoras. A escolha desta plataforma se justifica por sua flexibilidade, ampla documentação disponível na comunidade e fácil integração com sistemas embarcados.

- **Docker**:

Figura 8 - Docker Logo

Fonte: Wikipédia (2013)

A infraestrutura do servidor está disponibilizada por meio de contêineres Docker. Docker é uma plataforma de código aberto que permite empacotar aplicações e suas dependências em contêineres isolados, garantindo que o software execute de forma consistente em qualquer ambiente. A orquestração é realizada via *docker-compose*, ferramenta que gerencia múltiplos contêineres simultaneamente através de arquivos de configuração declarativos, permitindo maior portabilidade e escalabilidade da solução.

Essas tecnologias foram selecionadas visando construir uma solução segura, escalável e alinhada com os princípios de código aberto, respeitando o foco em privacidade e controle total por parte do usuário.

3.9 DEPENDÊNCIAS

As dependências do projeto referem-se a todos os elementos necessários para garantir o funcionamento correto da assistente virtual inteligente, tanto no nível de hardware e software quanto no nível humano. Estes componentes são fundamentais para atender às exigências de desempenho, segurança, robustez e privacidade definidos nos requisitos.

1.
2.
3.
   3.1.

3.2.
3.3.
3.4.
3.5.
3.6.
3.7.
3.8.
3.9.

3.9.1. Hardware

- **Servidor Central:**
    - Computador ou servidor dedicado com os seguintes requisitos:
        - **Recomendado**: Placa de vídeo dedicada (Nvidia GTX/RTX ou AMD compatível com execução local de LLMs via Ollama).
        - **Mínimo**: Processador Intel de 7ª geração (ou equivalente AMD Ryzen) com suporte a instruções modernas de virtualização e processamento paralelo.
        - Memória RAM mínima de 16GB recomendada para operação fluida.
        - Espaço em disco suficiente para armazenar modelos de linguagem (pelo menos 50 GB livres).
- **Cliente IoT:**
    - Dispositivo microcontrolado, como:
        - ESP32 com suporte a gravação e reprodução de áudio.
        - Módulo de microfone e alto-falante acoplados ao microcontrolador.
- **Infraestrutura de Rede:**
    - Roteador local com suporte a redes internas isoladas (sem acesso externo obrigatório).
    - Disponibilidade de infraestrutura para comunicação segura (certificados para criptografia TLS caso necessário).

3.9.2. Software

- **Servidor:**
    - **Sistema Operacional**: Linux ou Windows.

    - **Plataformas de Contêinerização**: Docker e docker-compose para implantação e orquestração de serviços.
      Gerenciador de Modelos de IA: Ollama para execução local dos modelos LLM.
    - **Modelos de Linguagem**: Llama 3, Mistral, Gemma ou modelos compatíveis em formato aberto.
- **Cliente IoT:**
    - **Firmware**: Programa customizado para ESP32, desenvolvido em Arduino IDE ou PlatformIO.
- **Ferramentas de Desenvolvimento:**
    - Visual Studio, Rider ou similar para codificação.
    - Git para controle de versão.
    - Repositório remoto (GitHub) para versionamento e colaboração.

3.9.3. Outras Dependências

- **Documentação**:
    - Manuais de instalação, configuração e operação.
    - Guia de boas práticas de segurança e privacidade.

3.10 REFERÊNCIAS (PARA O LEVANTAMENTO INICIAL)

A necessidade de proteger a privacidade e a segurança dos dados do usuário final foi o principal ponto de partida para o desenvolvimento do sistema. Com o aumento do uso de dispositivos de Internet das Coisas (IoT) e agentes de Inteligência Artificial (IA) em ambientes domésticos e corporativos, identificou-se uma preocupação crescente com a exposição involuntária de dados pessoais sensíveis.

A partir dessa realidade, algumas necessidades específicas foram levantadas:

- **Garantia de Privacidade Local**: A utilização de dispositivos conectados à internet implica na coleta constante de dados. Muitos desses dispositivos, por serem baseados em soluções proprietárias, enviam informações para servidores externos, o que aumenta o risco de vazamentos ou usos indevidos. Assim, foi identificada a necessidade de criar soluções que

permitam que todos os dados trafeguem exclusivamente na rede local do usuário, sem dependência de serviços de nuvem de terceiros.

- **Controle Total pelo Usuário**: Um dos princípios norteadores foi dar ao usuário total transparência e controle sobre seus dados. Para isso, surgiu a exigência do uso de softwares de código aberto, permitindo auditoria completa do comportamento do sistema, além de possibilitar personalizações e correções de segurança por parte da própria comunidade ou dos usuários mais avançados.
- **Resistência a Ataques Cibernéticos**: Considerando que mesmo soluções de IA robustas como o Chat GPT podem ser alvos de ataques, ficou evidente a necessidade de desenvolver um sistema minimamente exposto a ameaças externas. Dessa forma, priorizou-se uma arquitetura que minimiza a superfície de ataque e adota práticas seguras de hospedagem e comunicação.
- **Autonomia Tecnológica**: Foi apontada também a necessidade de independência em relação às grandes corporações tecnológicas. Ao utilizar **modelos de IA de código aberto**, como Llama, Mistral e Gemma, o usuário pode usufruir das tecnologias mais modernas sem renunciar à sua soberania sobre seus dados.
- **Facilidade de Uso e Implementação**: Apesar do foco em segurança, outra necessidade importante era garantir que o sistema fosse acessível também para usuários finais que não possuem conhecimentos técnicos avançados. Por isso, as soluções propostas deveriam ser fáceis de instalar, configurar e manter.

Essas necessidades foram a base para a definição dos requisitos do sistema, sempre com a segurança e a privacidade do usuário como prioridades máximas.

4 ESTRATÉGIAS DE RISCO

4.1 LISTA DE RISCOS

Os riscos para o projeto são apenas técnicos e envolvem, principalmente, a imprevisibilidade do LLM escolhido. Dado o escasso recurso computacional, e por consequência o simples modelo utilizado, os riscos relacionados à assistente são: Ausência de resposta, alucinação na resposta, resposta formatada para texto com caracteres que não compõem uma comunicação verbal.

Ademais, o projeto pode enfrentar riscos relacionados à comunicação entre o dispositivo IoT e o servidor, ou até mesmo uma falha no hardware, como mal funcionamento do módulo de microfone, amplificador, alto-falantes ou microcontrolador.

Outro risco mapeado, é o da não inicialização do serviço Ollama, necessário para o funcionamento do sistema.

4.2 PRIORIZAÇÃO DOS RISCOS

Tabela 1: Priorização dos riscos

Risco	Prioridade	Impacto	Probabilidade
Ausência de resposta da assistente	Alta	Alto	Baixa
Alucinação na resposta da assistente	Baixa	Baixo	Média
Resposta da assistente mal formatada	Médio	Baixo	Baixa
Falha de comunicação entre dispositivo IoT e servidor	Alta	Alto	Média
Falha de hardware no dispositivo IoT	Alta	Alto	Média
Não inicialização do serviço Ollama	Alta	Alto	Alta

4.3 PLANOS DE MITIGAÇÃO

Tabela 2: Planos de mitigação

Risco	Plano de Mitigação
Ausência de resposta da assistente	Desenvolvido mecanismo de reinicialização da conversa
Alucinação na resposta da assistente	Criatividade da assistente reduzida para 0,6
Resposta da assistente mal formatada	Dada instrução inicial para a assistente estruturar a resposta simulando uma conversa verbal
Falha de comunicação entre dispositivo IoT e servidor	Desenvolvido mecanismo para permitir novas tentativas
Falha de hardware no dispositivo IoT	Projetado invólucro de plástico para preservação dos componentes elétricos
Não inicialização do serviço Ollama	Desenvolvido mecanismo para permitir novas tentativas e também mensagem de aviso sugerindo que o serviço possa estar fora do ar.

4.4 PLANOS DE CONTINGÊNCIA

Tabela 3: Planos de contingência

Risco	Plano de Contingência
Ausência de resposta da assistente	Caso a reinicialização programada não solucione, se fará necessária a reinicialização manual do servidor e do Ollama
Alucinação na resposta da assistente	Reduzir ainda mais a criatividade da assistente
Resposta da assistente mal formatada	Considerar adicionar mais instruções sobre a estrutura das respostas
Falha de comunicação entre dispositivo IoT e servidor	Conectar dispositivo ao computador e diagnosticar o problema com o auxílio das mensagens logadas no canal serial
Falha de hardware no dispositivo IoT	Utilizar multímetro para medições elétricas, diagnosticar módulo defeituoso e efetuar reparo
Não inicialização do serviço Ollama	Iniciar serviço via terminal ou arquivo executável

5 GERENCIAMENTO DE CONFIGURAÇÃO

O gerenciamento de configuração de software e hardware, tem como objetivo garantir o controle de todos os artefatos produzidos ao longo do clico de vida do projeto Melissa.

5.1 ENVOLVIDOS NO PROCESSO

Tabela 4: Envolvidos no processo

Nome	Atividade	Responsabilidades
Danilo Maia Florenzano	Desenvolvedor Backend e Embarcado	Desenvolvimentos de funcionalidades da assistente, configuração da infraestrutura e desenvolvimento do software embarcado junto à prototipagem eletrônica.
João Victor Soares Jordão	Desenvolvedor Backend	Desenvolvimentos de funcionalidades da assistente e configuração do modelo de inteligência artificial.
Rafael Chiareli Junior	Orientador	Supervisão geral do projeto.

5.2 FERRAMENTAS E INFRAESTRUTURA

O projeto Melissa conta com ferramentas consolidadas no mercado e na comunidade de desenvolvimento de software para desempenhar as funções de Gerenciamento da Configuração em todo o seu ciclo de vida.

5.3 FERRAMENTAS

Tabela 5: Ferramentas

Ferramenta	Finalidade	Versão
Google Docs	Apresentações	2025
Microsoft Word	Documentação textual	
Github	Hospedagem do Código Fonte	2025
Git	Versionamento do Código Fonte	2.45.1
Jetbrains Rider	Desenvolvimento e depuração de código	2025.1.3
Visual Sudio 2022	Desenvolvimento e depuração de código	17.14.5
Arduino IDE	Desenvolvimento e depuração de código	2.3.6

.NET	Framework de desenvolvimento	9.0.301
C#	Linguagem de desenvolvimento	13.0
C++	Linguagem de desenvolvimento	11.0
Ollama	Interface de comunicação e execução de LLMs	0.9.1
SQLite	Banco de dados SQL	3.50.1
Gaphor	Modelagem de diagramas UML	3.1.0

5.4 INFRAESTRUTURA

Tabela 6: Infraestrutura

Tipo	Descrição
Servidor Melissa	https://localhost:5179
Caminho Físico	src/Melissa/Melissa.WebServer
Servidor Ollama	https://localhost:11434

6 ESTRATÉGIA DE TESTES

As estratégias de teste do projeto consistem tanto na verificação das funcionalidades desenvolvidas pelos programadores, quanto pela verificação da integração delas com a inteligência artificial.

Tabela 7: Necessidade de Hardware

Tipo de Hardware	Detalhamento	Forma de Disponibilização	Data
Computador	Linux 6.12.39-1-MANJARO	Pessoal	03/08/2025
Dispositivo eletrônico	Projeto de autoria dos desenvolvedores utilizando ESP32 e módulos eletrônicos	Pessoal	03/08/2025

Tabela 8: Necessidade de Software

Tipo de Software	Detalhamento	Forma de Disponibilização	Data Limite
Servidor Ollama	Versão 0.9.6-1 ou Superior	Corporativo/Pessoal	-
Konsole ou emulador de terminal similar	Versão 25.04.3 ou Superior	Corporativo/Pessoal	-
Sistema embarcado Melissa	Versão 1.0 ou Superior	Corporativo/Pessoal	-

Tabela 9: Necessidade de Pessoas

Papel	Envolvimento Estimado	Período de Envolvimento no Projeto
Danilo Maia Florenzano (Módulo Hardware)	4 horas	01/03/2024 à 22/11/2025
João Victor Soares Jordão (Módulo terminal)	2 horas	01/03/2024 à 22/11/2025

6.1 PLANO DE TESTES

Os testes foram desenvolvidos e realizados de forma prática para assegurar a máxima consistência das funcionalidades levando em conta suas particularidades e natureza.

6.1.1 **LISTA DE TESTES**

Tabela 10: Lista de Teste

Caso de Uso/Requisito	Tipo de Teste	Técnica de Teste	Caso de Teste	Caso de Teste Dependente	Ator
Conhecimento da assistente sobre hora e data atual	Não-funcional	Funcional	CT001	***	Usuário
Busca de feriados por cidade, estado e mês	Funcional	Funcional	CT002	CT011 ou CT012	Usuário
Busca de feriados por nome	Funcional	Funcional	CT003	CT011 ou CT012	Usuário
Busca informações sobre cidades	Funcional	Funcional	CT004	CT011 ou CT012	Usuário
Busca informações sobre clima	Funcional	Funcional	CT005	CT011 ou CT012	Usuário
Registra histórico de conversa	Não-funcional	Não-funcional	CT006	CT001 e CT011 ou CT012	Usuário
Envia histórico de conversa por email	Funcional	Funcional	CT007	CT008 e CT011 ou CT012	Usuário
Converte áudio em texto	Não-funcional		CT008	***	Melissa
Converte texto em áudio	Não-funcional		CT009	***	Melissa
Comunica com cliente utilizando HTTP *Streaming*	Não-funcional		CT010	***	Melissa
Comunica por texto	Funcional	Funcional	CT011	***	Usuário
Comunica por voz	Funcional	Funcional	CT012	CT008, CT009 e CT010	Usuário
Grava lista de itens	Funcional	Funcional	CT013	CT011 ou CT012	Usuário
Exibe lista de itens	Funcional	Funcional	CT014	CT014 e CT011 ou CT012	Usuário
Altera lista de itens	Funcional	Funcional	CT015	CT014 e CT011 ou CT012	Usuário

- Funcionais: verificam as funcionalidades de um sistema e buscam erros de desempenho;

- Não-funcionais: analisam a usabilidade, velocidade e outros aspectos que independem das ações executadas pelo sistema.

Tabela 11: Caso de Teste CT013

Caso Nº	CT013 – Comunica por texto
Data	03/08/2025
Testador	João Victor Soares Jordão
Objetivo do Teste	Verificar se a assistente é capaz de se comunicar por texto.
Entradas	Digitar uma mensagem qualquer
Passos	1. Abrir o terminal cliente. 2. Digitar uma mensagem qualquer. 3. Apertar a tecla Enter para enviar a mensagem.
Comportamento Esperado	A assistente deve dar uma resposta

Tabela 12: Caso de Teste CT001

Caso Nº	CT001 - Conhecimento da assistente sobre hora e data atua
Data	04/08/2025
Testador	Danilo Maia Florenzano
Objetivo do Teste	Verificar se a assistente é capaz de saber sobre a data e hora atual.
Entradas	Perguntar à assistente “que horas são?” e “que dia é hoje?”.
Passos	1. Abrir o terminal cliente. 2. Digitar a primeira pergunta 3. Apertar a tecla Enter para enviar a mensagem. 4. Após a resposta da assistente, repetir passo 2 e 3 com a segunda pergunta.
Comportamento Esperado	A assistente deve responder corretamente a hora e a data atual.

Tabela 13: Caso de Teste CT002

Caso Nº	CT002 - Busca de feriados por cidade, estado e mês
Data	04/08/2025
Testador	Danilo Maia Florenzano
Objetivo do Teste	Verificar se a assistente é capaz de buscar feriados para determinada cidade ou cidade em determinado mês.
Entradas	Perguntar à assistente se há feriados em alguma cidade ou estado em determinado mês.
Passos	1. Abrir o terminal cliente. 2. Digitar a pergunta 3. Apertar a tecla Enter para enviar a mensagem.
Comportamento Esperado	A assistente deve responder corretamente se há feriados, e suas respectivas datas.

Tabela 14: Caso de Teste CT003

Caso Nº	CT003 - Busca de feriados por nome
Data	04/08/2025
Testador	João Victor Soares Jordão
Objetivo do Teste	Verificar se a assistente é capaz de buscar feriados por nome.
Entradas	Perguntar à assistente quando é a data de determinado feriado.
Passos	1. Abrir o terminal cliente. 2. Digitar a pergunta 3. Apertar a tecla Enter para enviar a mensagem.
Comportamento Esperado	A assistente deve responder corretamente a data do feriado

	requerido.

Tabela 15: Caso de Teste CT004

Caso Nº	CT004 - Busca informações sobre cidades
Data	04/08/2025
Testador	João Victor Soares Jordão
Objetivo do Teste	Verificar se a assistente é capaz de buscar informações dada uma determinada cidade do território brasileiro.
Entradas	Perguntar à assistente sobre a população, ou pontos turísticos ou administrativos de alguma cidade.
Passos	1. Abrir o terminal cliente. 2. Digitar a pergunta 3. Apertar a tecla Enter para enviar a mensagem.
Comportamento Esperado	A assistente deve responder corretamente as informações sobre a cidade requerida.

Tabela 16: Caso de Teste CT005

Caso Nº	CT005 - Busca informações sobre clima
Data	04/08/2025
Testador	João Victor Soares Jordão
Objetivo do Teste	Verificar se a assistente é capaz de responder informações sobre o clima atual dada uma determinada cidade.
Entradas	Perguntar à assistente sobre a população, ou pontos turísticos ou administrativos de alguma cidade.
Passos	1. Abrir o terminal cliente. 2. Digitar a pergunta 3. Apertar a tecla Enter para enviar a mensagem.
Comportamento Esperado	A assistente deve responder corretamente as informações sobre o clima atual da cidade requerida.

Tabela 17: Caso de Teste CT007

Caso Nº	CT006 - Registra histórico de conversa
Data	04/08/2025
Testador	Danilo Maia Florenzano
Objetivo do Teste	Verificar se a assistente é capaz de responder informações sobre o clima atual dada uma determinada cidade.
Entradas	
Passos	1. Abrir o terminal cliente. 2. Digitar a pergunta 3. Apertar a tecla Enter para enviar a mensagem.
Comportamento Esperado	A assistente deve responder corretamente as informações sobre o clima atual da cidade requerida.

7 ESTRATÉGIA DE IMPLANTAÇÃO E SUPORTE

A implantação de um sistema de assistente virtual inteligente requer planejamento cuidadoso e estratégias bem definidas para garantir que todos os componentes funcionem adequadamente em diferentes ambientes e configurações. Este capítulo detalha as necessidades de implantação do sistema Melissa, incluindo requisitos de infraestrutura, configurações necessárias e procedimentos de suporte técnico que asseguram a operação contínua e eficiente da solução.A estratégia de implantação foi concebida considerando os princípios fundamentais do projeto: privacidade, autonomia e facilidade de uso. Embora o sistema seja tecnicamente sofisticado, a implantação deve ser acessível até mesmo para usuários com conhecimentos técnicos intermediários. Para isso, foram desenvolvidos procedimentos padronizados, documentação detalhada e automações que simplificam o processo de instalação e configuração.

7.1 NECESSIDADES DE IMPLANTAÇÃO

A implantação bem-sucedida do sistema Melissa depende da satisfação de requisitos específicos de hardware, software e infraestrutura de rede. Esta seção detalha cada um desses requisitos e fornece orientações práticas para preparação do ambiente de execução.

7.1.1 **Arquitetura de implantação**

A arquitetura de implantação do sistema Melissa segue um modelo cliente-servidor descentralizado, onde o servidor opera em infraestrutura local controlada pelo usuário e múltiplos clientes podem se conectar simultaneamente para interação com a assistente. Esta arquitetura foi escolhida especificamente para maximizar a privacidade e minimizar a dependência de serviços externos.

O servidor Melissa é o componente central do sistema e deve ser implantado em um computador ou servidor dedicado dentro da rede local do usuário. Este servidor hospeda três serviços principais: o WebServer responsável pela API de comunicação e interface com clientes, o serviço Ollama que executa o modelo de linguagem de inteligência artificial, e o banco de dados MySQL para persistência de informações como tarefas agendadas e histórico de conversas. Todos esses serviços são

containerizados utilizando Docker, o que simplifica significativamente o processo de implantação e garante consistência entre diferentes ambientes de execução.

Os clientes podem ser de dois tipos principais: clientes de terminal, que são aplicações de linha de comando executadas em qualquer computador com acesso à rede local, e dispositivos IoT baseados em microcontroladores como Arduino ou ESP32, equipados com módulos de captura e reprodução de áudio. Ambos os tipos de cliente se comunicam com o servidor através de protocolo HTTP ou HTTPS, dependendo da configuração de segurança escolhida pelo usuário.

A comunicação entre componentes ocorre exclusivamente dentro da rede local, por padrão, sem qualquer tráfego saindo para a internet. O usuário tem controle total sobre quando e como o sistema acessa recursos externos. Por exemplo, funcionalidades como consulta de previsão do tempo ou busca de feriados nacionais requerem acesso à internet, mas esse acesso só é habilitado quando o usuário explicitamente configura a assistente para permitir conexões externas. Mesmo quando habilitado, apenas o servidor realiza requisições externas; os clientes continuam comunicando-se exclusivamente com o servidor local.

7.1.2 **Configuração dos servidores**

A configuração adequada do servidor é crucial para o funcionamento estável e eficiente do sistema Melissa. O processo de configuração envolve múltiplas etapas que devem ser executadas na ordem correta para garantir que todas as dependências sejam satisfeitas.

Primeiramente, o sistema operacional do servidor deve ser preparado. O Melissa é compatível com sistemas Linux (preferencialmente distribuições baseadas em Debian ou Arch Linux) e Windows 10/11 ou Windows Server. Para ambientes Linux, recomenda-se utilizar distribuições atualizadas com kernel versão 5.0 ou superior. O sistema operacional deve estar com todas as atualizações de segurança aplicadas antes de iniciar a instalação do Melissa.

O Docker e docker-compose devem ser instalados e configurados no servidor. No Linux, isso pode ser feito através dos gerenciadores de pacote nativos da distribuição. É importante adicionar o usuário que executará os contêineres ao grupo docker para permitir a execução sem necessidade de privilégios de root. Recomenda-se verificar a instalação executando comandos de teste como "docker --version" e

"docker-compose --version" para confirmar que as ferramentas foram instaladas corretamente.

O serviço Ollama deve ser instalado separadamente, pois não é containerizado junto com o servidor Melissa. Isso permite maior flexibilidade na gestão dos modelos de linguagem e melhor aproveitamento dos recursos de hardware, especialmente GPUs. A instalação do Ollama varia conforme o sistema operacional, mas geralmente envolve o download de um instalador ou script de instalação do site oficial. Após a instalação, o serviço Ollama deve ser configurado para iniciar automaticamente com o sistema operacional.

Com o Ollama instalado, o próximo passo é fazer o download dos modelos de linguagem que serão utilizados pela assistente. O comando "ollama pull" permite baixar modelos específicos, como "ollama pull llama3.2" para o modelo Llama 3.2. Recomenda-se começar com modelos menores (7B ou 8B parâmetros) e avaliar o desempenho antes de migrar para modelos maiores que exigem mais recursos computacionais. O Ollama armazena os modelos em um diretório local e os mantém em cache para acesso rápido.

A configuração do servidor Melissa propriamente dito é realizada através de arquivos de configuração e variáveis de ambiente definidas no docker-compose.yml. Este arquivo especifica parâmetros como portas de rede, volumes de dados persistentes, variáveis de conexão com banco de dados e configurações de segurança. O usuário deve editar este arquivo para personalizar aspectos como a porta HTTP/HTTPS em que o servidor escutará conexões, credenciais do banco de dados, e se a criptografia TLS deve estar habilitada por padrão.

O banco de dados MySQL é inicializado automaticamente pelo docker-compose na primeira execução, mas requer configuração inicial de schema. Scripts SQL fornecidos no repositório do projeto devem ser executados para criar as tabelas necessárias e estruturas de dados. Esses scripts podem ser executados conectando-se ao contêiner do MySQL ou utilizando ferramentas de administração como phpMyAdmin ou MySQL Workbench.

Configurações de segurança adicionais incluem a geração de certificados TLS caso o usuário deseje utilizar HTTPS para comunicação criptografada. Certificados auto-assinados podem ser gerados utilizando ferramentas como OpenSSL, ou certificados emitidos por autoridades certificadoras confiáveis podem ser instalados se o servidor for acessível externamente. A configuração de firewall também é

importante para restringir acesso ao servidor apenas de dispositivos autorizados na rede local.

7.1.3 **Configuração dos clientes**

A configuração dos clientes varia significativamente dependendo do tipo de cliente sendo utilizado. Para clientes de terminal, que são aplicações de linha de comando, a configuração é relativamente simples e envolve principalmente especificar o endereço IP ou hostname do servidor Melissa na rede local.

O cliente de terminal é distribuído como um executável compilado para diferentes plataformas (Windows, Linux, macOS). Após o download, o executável deve ser colocado em um diretório acessível e, idealmente, adicionado ao PATH do sistema para facilitar sua invocação. Um arquivo de configuração JSON ou INI acompanha o cliente, contendo parâmetros como o endereço do servidor, porta de conexão, se deve utilizar HTTPS, e preferências de interface do usuário.

A primeira execução do cliente de terminal realiza uma verificação de conectividade com o servidor, testando se a comunicação está funcionando adequadamente. Se o servidor utiliza certificados TLS auto-assinados, o cliente pode solicitar confirmação do usuário para confiar no certificado na primeira conexão. Recomenda-se documentar o fingerprint do certificado para que usuários possam verificar sua autenticidade.

Para dispositivos IoT baseados em microcontroladores, a configuração é mais complexa e requer conhecimentos de eletrônica e programação embarcada. O código-fonte do firmware é fornecido no repositório do projeto e deve ser compilado utilizando a Arduino IDE ou PlatformIO. Antes da compilação, o usuário deve editar constantes no código para especificar credenciais da rede Wi-Fi local (SSID e senha) e o endereço IP do servidor Melissa.

A montagem física do dispositivo IoT segue um esquema elétrico fornecido na documentação do projeto. Componentes necessários incluem um microcontrolador ESP32, módulo de microfone com pré-amplificador (como o MAX4466 ou MAX9814), módulo amplificador de áudio (como o PAM8403), alto-falantes, fonte de alimentação adequada, botões para controle de gravação, e componentes eletrônicos básicos como resistores e capacitores. O esquema deve ser seguido cuidadosamente para evitar danos aos componentes ou mau funcionamento.

Após a montagem física e compilação do firmware, o código deve ser enviado (uploaded) para o microcontrolador através de conexão USB. Durante a primeira inicialização, o dispositivo tentará conectar-se à rede Wi-Fi configurada e, em seguida, ao servidor Melissa. Mensagens de depuração são enviadas através da porta serial e podem ser monitoradas utilizando o Serial Monitor da Arduino IDE para diagnosticar problemas de conectividade.

Calibração do módulo de microfone pode ser necessária para garantir qualidade adequada de captura de áudio. Isso envolve ajustar o ganho do pré-amplificador através de um potenciômetro presente na maioria dos módulos. O objetivo é maximizar o nível de captura sem introduzir distorção ou ruído excessivo. Testes práticos de gravação e reprodução devem ser realizados para verificar a qualidade do áudio.

7.1.4 **Infraestrutura necessária**

A infraestrutura de rede é um componente crítico para o funcionamento adequado do sistema Melissa. Embora o sistema seja projetado para operar em redes locais simples, algumas considerações de infraestrutura são importantes para garantir desempenho e confiabilidade.

O roteador de rede local deve suportar alocação de endereços IP estáticos ou, alternativamente, reservas DHCP para garantir que o servidor Melissa mantenha sempre o mesmo endereço IP. Isso evita que clientes percam conectividade caso o servidor receba um endereço diferente após reinicializações. Configurações de qualidade de serviço (QoS) no roteador podem priorizar tráfego do servidor Melissa, especialmente importante para streaming de áudio em tempo real.

A largura de banda da rede local raramente é um limitador, mas a latência pode afetar a experiência do usuário, especialmente ao utilizar comandos de voz através de dispositivos IoT. Recomenda-se que o servidor esteja conectado ao roteador através de cabo Ethernet sempre que possível, reservando conexões Wi-Fi apenas para dispositivos móveis que não podem ser cabeados. Para dispositivos IoT, a proximidade do ponto de acesso Wi-Fi é importante para garantir sinal forte e baixa latência.

Segmentação de rede pode ser considerada para maior segurança. Por exemplo, dispositivos IoT podem ser isolados em uma VLAN separada com regras de

firewall que permitem comunicação apenas com o servidor Melissa, impedindo que dispositivos comprometidos acessem outros recursos da rede doméstica. Essa configuração requer roteadores e switches gerenciáveis, sendo opcional, mas recomendada para usuários preocupados com segurança.

Backup de dados é uma consideração importante para a infraestrutura. Embora o sistema Melissa não armazene grandes volumes de dados, informações como tarefas agendadas, histórico de conversas e configurações personalizadas devem ser regularmente copiadas para mídia de backup. O docker-compose pode ser configurado para mapear volumes de dados para diretórios específicos no sistema de arquivos do servidor, facilitando procedimentos de backup automatizados utilizando ferramentas como rsync, cron jobs ou soluções de backup comerciais.

Monitoramento da infraestrutura auxilia na detecção precoce de problemas. Ferramentas de monitoramento como Prometheus e Grafana podem ser integradas para coletar métricas de desempenho do servidor, uso de recursos, latência de rede e disponibilidade dos serviços. Alertas podem ser configurados para notificar administradores quando anomalias são detectadas, permitindo intervenção antes que problemas afetem os usuários.

7.2 CRONOGRAMA DE TREINAMENTOS

Embora o sistema Melissa seja desenvolvido como uma solução self-hosted para uso individual ou familiar, é importante considerar a necessidade de documentação e materiais educativos que auxiliem os usuários na instalação, configuração e uso efetivo do sistema. O cronograma de treinamentos proposto não se refere a sessões presenciais tradicionais, mas sim à disponibilização progressiva de materiais didáticos, documentação técnica e tutoriais que capacitem diferentes perfis de usuários.

A primeira fase de materiais educativos consiste na documentação básica de instalação, direcionada a usuários com conhecimentos intermediários de informática. Esses materiais incluem guias passo a passo para instalação do Docker, configuração inicial do servidor Melissa através de docker-compose, e verificação de que o sistema está operando corretamente. Tutoriais em formato de texto com capturas de tela ilustrativas são disponibilizados no repositório GitHub do projeto, organizados por

sistema operacional (Linux e Windows) para facilitar a localização das instruções relevantes.

A segunda fase abrange materiais sobre configuração avançada e personalização do sistema. Esses recursos são direcionados a usuários com maior conhecimento técnico que desejam ajustar parâmetros do modelo de linguagem, configurar certificados TLS para comunicação segura, ou integrar ferramentas e funcionalidades customizadas. Documentação detalhada sobre a arquitetura do código, APIs disponíveis e pontos de extensão do sistema permite que desenvolvedores contribuam com novos recursos ou adaptem o sistema para necessidades específicas.

A terceira fase consiste em materiais educativos sobre a construção e configuração de dispositivos IoT clientes. Esses tutoriais incluem diagramas de circuito eletrônico, lista de componentes necessários, instruções de montagem física, e guias de configuração e gravação do firmware. Para facilitar a replicação do projeto, são disponibilizados arquivos para impressão 3D de invólucros protetivos para os componentes eletrônicos, permitindo que usuários com acesso a impressoras 3D criem dispositivos com aparência profissional e adequada para uso doméstico.

Complementando a documentação escrita, a criação de conteúdo em vídeo representa uma forma acessível de demonstrar processos de instalação e configuração. Vídeos tutoriais curtos podem ser disponibilizados em plataformas como YouTube, demonstrando visualmente cada etapa do processo de implantação. Esses vídeos são especialmente úteis para usuários que têm dificuldade com documentação técnica escrita ou que preferem aprender através de demonstrações práticas.

A comunidade de usuários desempenha papel fundamental no suporte contínuo do projeto. Através de fóruns de discussão, grupos em plataformas de mensageria ou seções de issues no repositório GitHub, usuários podem compartilhar experiências, relatar problemas, propor melhorias e auxiliar uns aos outros na resolução de dificuldades técnicas. Esse modelo de suporte comunitário é característico de projetos de código aberto e tende a gerar conhecimento coletivo valioso que beneficia todos os participantes.

Para garantir a sustentabilidade do projeto a longo prazo, é importante estabelecer processos de atualização e manutenção da documentação. À medida que novas versões do sistema são lançadas, novos recursos são adicionados ou

dependências externas são atualizadas, a documentação deve ser revisada e expandida correspondentemente. A manutenção de um changelog detalhado que documenta todas as alterações entre versões ajuda os usuários a compreenderem o que mudou e se precisam realizar ajustes em suas instalações existentes.

Finalmente, a disponibilização de ambientes de demonstração ou instâncias de teste pode auxiliar usuários indecisos a experimentarem o sistema antes de comprometerem recursos para instalação própria. Embora isso aparentemente contradiga o princípio de self-hosting, uma instância de demonstração pública com limitações claras (como não persistência de dados, limitações de uso ou restrições de funcionalidades) pode servir como ferramenta educacional que incentiva mais pessoas a instalarem suas próprias instâncias privadas após compreenderem o potencial do sistema.

8 APLICATIVO MÓVEL MELISSA

8.1 INTRODUÇÃO AO APLICATIVO

O aplicativo móvel Melissa foi desenvolvido como uma interface cliente para interação com a assistente virtual inteligente, permitindo que usuários acessem todas as funcionalidades do sistema através de dispositivos Android e iOS. O aplicativo representa a camada de apresentação do projeto, conectando-se ao servidor backend através da API REST e do hub SignalR para proporcionar uma experiência de usuário fluida, intuitiva e responsiva.

A arquitetura do aplicativo móvel foi projetada seguindo os mesmos princípios de privacidade e segurança do servidor, garantindo que todas as comunicações sejam realizadas exclusivamente com a infraestrutura local controlada pelo usuário. O aplicativo não coleta ou transmite dados para servidores de terceiros, mantendo-se fiel à proposta de autonomia digital e respeito à privacidade que fundamenta todo o projeto Melissa.

8.2 TECNOLOGIAS UTILIZADAS

O desenvolvimento do aplicativo móvel utilizou tecnologias modernas e multiplataforma que permitem a criação de uma única base de código para ambas as plataformas móveis principais. A escolha tecnológica priorizou produtividade de desenvolvimento, desempenho da aplicação e qualidade da experiência do usuário.

Framework de Desenvolvimento: O aplicativo foi desenvolvido utilizando React Native com Expo, uma combinação que permite desenvolvimento ágil de aplicações móveis nativas utilizando JavaScript e React. O Expo fornece um conjunto de ferramentas e serviços que simplificam significativamente o processo de desenvolvimento, testes e distribuição de aplicativos móveis.

Linguagem de Programação: TypeScript foi escolhido como linguagem principal de desenvolvimento, trazendo tipagem estática e recursos avançados de programação orientada a objetos para o ecossistema JavaScript. O uso de TypeScript reduz significativamente a ocorrência de erros em tempo de execução e melhora a manutenibilidade do código através de melhor autocomplete e verificação de tipos em tempo de desenvolvimento.

Gerenciamento de Estado: Para gerenciamento de estado da aplicação, foi utilizado React Context API em conjunto com hooks customizados, proporcionando

uma solução simples e eficiente para compartilhamento de dados entre componentes sem necessidade de bibliotecas externas complexas.

Comunicação com Backend: A biblioteca Axios foi utilizada para requisições HTTP à API REST do servidor Melissa, oferecendo uma interface elegante e consistente para consumo dos endpoints. Para comunicação em tempo real através do hub SignalR, foi integrada a biblioteca @microsoft/signalr que implementa o protocolo SignalR em JavaScript.

Interface do Usuário: A interface foi construída utilizando componentes nativos do React Native combinados com a biblioteca React Native Paper, que implementa as diretrizes de Material Design adaptadas para React Native, garantindo uma experiência visual consistente e profissional.

8.3 ARQUITETURA DO APLICATIVO

A arquitetura do aplicativo móvel segue o padrão de separação de responsabilidades, dividindo o código em camadas bem definidas que facilitam manutenção, testes e evolução do sistema.

Camada de Apresentação: Contém todos os componentes visuais da aplicação, incluindo telas, componentes reutilizáveis e elementos de interface. Esta camada é responsável exclusivamente pela renderização da interface e captura de eventos do usuário, delegando toda lógica de negócio para camadas inferiores.

Camada de Serviços: Implementa a comunicação com o servidor backend, encapsulando toda lógica de requisições HTTP e gerenciamento de conexões SignalR. Esta camada expõe interfaces simples que são consumidas pelos componentes de apresentação, abstraindo complexidades de protocolos de comunicação e tratamento de erros.

Camada de Estado: Gerencia o estado global da aplicação, incluindo dados do usuário, configurações, histórico de conversas e estado de conexão com o servidor. Utiliza Context API do React para tornar o estado acessível a qualquer componente da aplicação sem prop drilling.

Camada de Utilitários: Contém funções auxiliares, validações, formatadores e outras utilidades compartilhadas por diferentes partes da aplicação. Esta camada promove reutilização de código e centralização de lógica comum.

8.4 FUNCIONALIDADES PRINCIPAIS

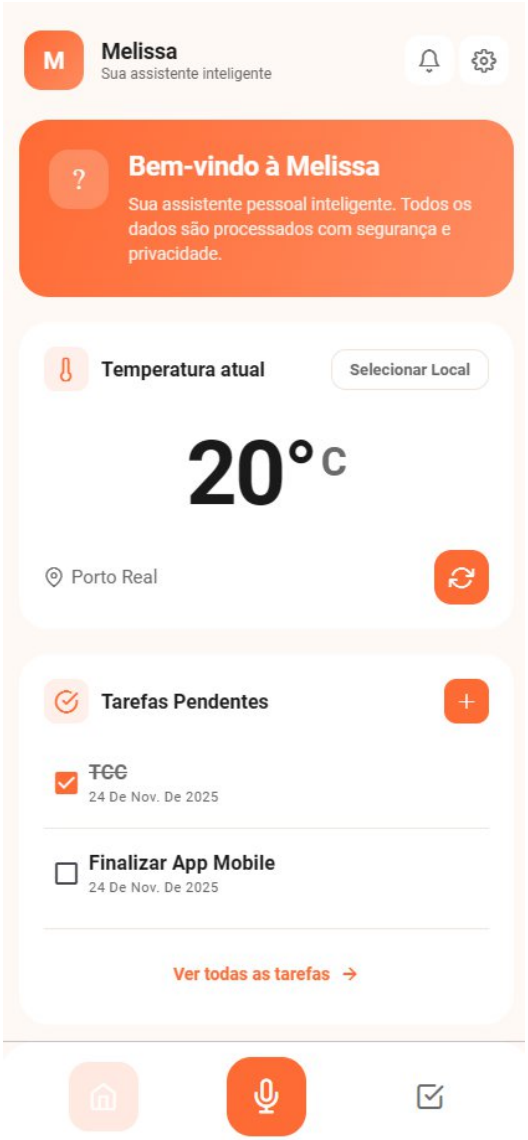
3.7.
3.8.
3.9.
3.10.

4.
5.
6.
7.
8.
8.1.
8.2.
8.3.
8.4.

8.4.1. Tela Inicial e Dashboard

A tela inicial do aplicativo apresenta um dashboard completo com acesso rápido às principais funcionalidades da assistente Melissa. O design foi cuidadosamente planejado para equilibrar densidade de informação com simplicidade visual, permitindo que o usuário identifique rapidamente o que precisa.

Figura 9 - Tela Inicial App Mobile



Fonte: Do próprio autor

O cabeçalho da aplicação exibe o nome "Melissa" junto com o subtítulo "Sua assistente inteligente", reforçando a identidade visual do projeto. No canto superior direito, ícones de notificações e configurações permitem acesso rápido a essas funcionalidades secundárias.

Um banner de boas-vindas em destaque laranja apresenta a mensagem "Bem-vindo à Melissa" seguida de uma breve descrição sobre segurança e privacidade dos dados: "Sua assistente pessoal inteligente. Todos os dados são processados com segurança e privacidade." Este banner serve tanto como elemento de boas-vindas quanto como reforço constante dos valores fundamentais do projeto.

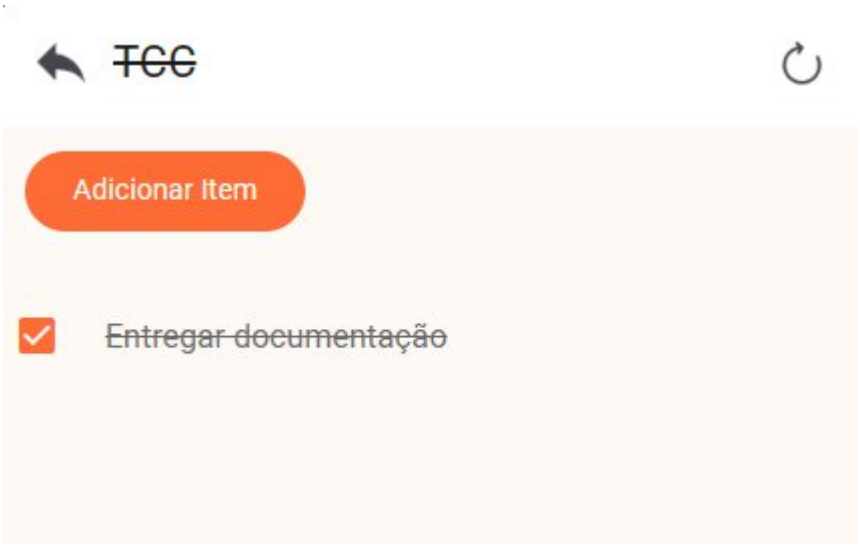
O widget de temperatura exibe a temperatura atual da localização configurada pelo usuário (20°C no exemplo), com um botão "Selecionar Local" que permite alterar a cidade para consulta meteorológica. A integração com serviços de previsão do tempo fornece informações atualizadas sempre que o usuário acessa a tela inicial.

A seção de "Tarefas Pendentes" apresenta uma visão resumida das tarefas agendadas, mostrando título, data de vencimento e status de conclusão. Um botão de adição (+) laranja no canto superior direito da seção permite criar novas tarefas rapidamente. Um link "Ver todas as tarefas" na parte inferior da seção leva o usuário para a tela completa de gerenciamento de tarefas.

8.4.2. Gerenciamento de Tarefas

O sistema de gerenciamento de tarefas (TaskList) é uma das funcionalidades centrais do aplicativo móvel, permitindo que usuários organizem suas atividades diárias de forma simples e eficiente. A interface foi projetada para minimizar o número de toques necessários para operações comuns, otimizando a produtividade.

Figura 10 - Tela de Itens da Tarefa App Mobile



Fonte: Do próprio autor

Ao acessar uma tarefa específica, o usuário visualiza todos os itens associados a ela. Cada item pode ser marcado como concluído tocando na checkbox ao lado esquerdo, que aplica uma formatação de texto tachado para indicar visualmente o status de conclusão. O botão "Adicionar Item" no topo da tela permite expansão da lista de itens conforme necessário.

Figura 11 - Tela de Tarefas App Mobile

Fonte: Do próprio autor

A tela de listagem completa de tarefas apresenta duas abas: "Abertas" e "Arquivadas", permitindo que o usuário filtre a visualização conforme o status das tarefas. Cada tarefa na lista exibe seu título, subtítulo de categorização ("Nova Lista") e dois botões de ação: "Enviar" para compartilhar a tarefa por email, e "Arquivar" para movê-la para o estado arquivado. Esta interface de swipe-actions facilita operações rápidas sem necessidade de entrar nos detalhes da tarefa.

O gerenciamento de tarefas implementa sincronização automática com o servidor backend através de requisições à API REST. Sempre que o usuário cria, modifica ou remove uma tarefa ou item, a alteração é imediatamente persistida no

banco de dados MySQL do servidor, garantindo que os dados estejam sempre atualizados mesmo que o aplicativo seja fechado ou o dispositivo seja reiniciado.

8.4.3. Consulta de Informações Contextuais

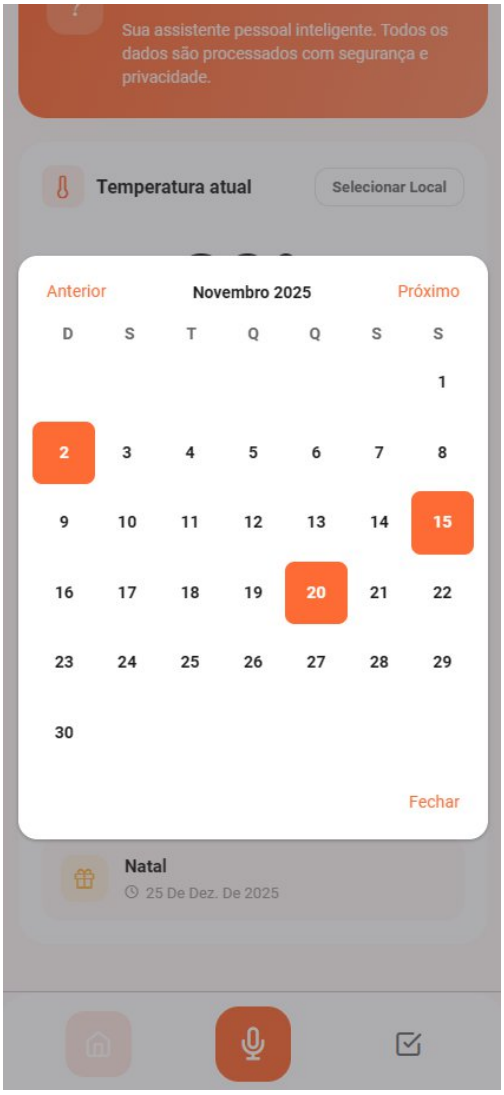
O aplicativo fornece acesso conveniente a informações contextuais relevantes para o cotidiano do usuário, como temperatura atual, previsão do tempo e datas de feriados nacionais.

Figura 12 - Tela Inicial com os Feriados App Mobile

Fonte: Do próprio autor

A seção "Próximos Feriados" exibe os feriados mais próximos da data atual, mostrando o nome do feriado e sua data. O exemplo apresenta "Natal" em 25 de dezembro de 2025. Um botão "Ver todos" permite acesso à lista completa de feriados do ano, facilitando planejamento de viagens e compromissos.

Figura 13 - Tela de Calendário dos Feriados App Mobile



Fonte: Do próprio autor

Funcionalidades que envolvem períodos de tempo, como solicitação de histórico de conversas, apresentam um componente de calendário intuitivo que permite seleção visual de datas. O calendário destaca visualmente a data atual e as datas selecionadas pelo usuário, com navegação simples entre meses através dos botões "Anterior" e "Próximo". O botão "Fechar" na parte inferior permite cancelar a seleção e retornar à tela anterior.

8.4.4. Configurações e Personalização

A tela de configurações permite que o usuário personalize aspectos importantes do funcionamento do aplicativo, particularmente o endereço do servidor Melissa ao qual deseja se conectar.

Figura 14 - Tela de Configurações App Mobile

Fonte: Do próprio autor

A seção "Servidor" apresenta um campo de texto para inserção do endereço IP ou hostname do servidor Melissa (exemplo: 192.168.1.103), seguido de um botão "Confirmar" laranja que válida e salva a configuração. Um botão "Testar Conexão" abaixo permite verificar se o servidor está acessível e respondendo adequadamente antes de confirmar a configuração.

A seção "Email" permite configuração do endereço de email que será utilizado para funcionalidades de compartilhamento e envio de histórico de conversas. Similar à configuração de servidor, um botão "Confirmar" salva o email informado.

A seção "Histórico de conversa" permite que o usuário solicite envio do histórico completo de suas interações com a assistente para o email configurado. Campos "De"

e "Até" permitem especificação do período desejado, e um botão "Solicitar" laranja envia a requisição ao servidor backend, que processará a solicitação e enviará o histórico por email.

Todas as configurações são armazenadas localmente no dispositivo utilizando AsyncStorage do React Native, garantindo que as preferências do usuário sejam mantidas entre sessões de uso do aplicativo.

8.5 INTERAÇÃO POR VOZ

Uma das funcionalidades mais inovadoras do aplicativo móvel é a capacidade de interação por comandos de voz, permitindo que o usuário converse com a assistente Melissa de forma natural e fluida, similar a assistentes virtuais comerciais como Alexa e Siri.

3.11.

8.5.

8.5.1. Captura e Processamento de Áudio

A funcionalidade de voz utiliza a API de gravação de áudio do React Native Expo (expo-av) para capturar áudio do microfone do dispositivo. Quando o usuário pressiona e mantém pressionado o botão de microfone na interface, o aplicativo inicia a gravação em formato PCM (Pulse Code Modulation) com taxa de amostragem de 16kHz, 16 bits por amostra, mono canal - configuração otimizada para processamento de voz e compatível com o serviço Whisper de transcrição utilizado pelo servidor.

Durante a gravação, o aplicativo apresenta feedback visual ao usuário através de animação do ícone de microfone ou indicador de nível de áudio, comunicando claramente que o sistema está capturando áudio. Ao soltar o botão, a gravação é interrompida e o áudio capturado é imediatamente enviado ao servidor para processamento.

8.5.2. Comunicação via SignalR Hub

Para proporcionar experiência de baixa latência e suportar streaming bidirecional de áudio, a funcionalidade de voz utiliza o hub SignalR do servidor Melissa em vez de requisições HTTP tradicionais. O SignalR estabelece uma conexão

persistente WebSocket entre o aplicativo e o servidor, permitindo comunicação em tempo real com overhead mínimo.

O fluxo de comunicação ocorre da seguinte forma:

- O aplicativo estabelece conexão com o hub SignalR em /melissa durante a inicialização
- Ao finalizar gravação de áudio, o aplicativo invoca o método remoto SendAudioChunk do hub, transmitindo o áudio bruto como array de bytes
- O servidor recebe o áudio, converte para formato WAV, realiza transcrição utilizando Whisper.net, processa a intenção do usuário através do modelo de linguagem Ollama, gera resposta em texto e a sintetiza em áudio MP3 utilizando Edge TTS
- O servidor envia o áudio de resposta de volta ao cliente através do hub SignalR
- O aplicativo recebe o áudio MP3, o decodifica e reproduz através do alto-falante do dispositivo

Esta arquitetura permite conversação natural com a assistente, onde o usuário fala, aguarda processamento e recebe resposta em áudio, criando uma experiência próxima de diálogo presencial.

8.5.3. Tratamento de Erros e Feedback

O aplicativo implementa tratamento robusto de erros para situações em que a comunicação por voz pode falhar: perda de conexão de rede durante transmissão de áudio, timeout de processamento no servidor, falhas na transcrição ou síntese de voz, ou permissões de microfone negadas pelo sistema operacional.

Em caso de erro, o usuário recebe feedback visual claro através de mensagens toast ou diálogos explicativos, com sugestões de ação para resolver o problema. Por exemplo, se permissões de microfone não foram concedidas, o aplicativo apresenta diálogo explicando a necessidade da permissão e oferecendo botão para abrir as configurações do sistema onde o usuário pode concedê-la.

8.6 ARQUITETURA DE COMUNICAÇÃO

A comunicação entre o aplicativo móvel e o servidor Melissa é crítica para o funcionamento adequado de todas as funcionalidades. A arquitetura de comunicação foi projetada para ser resiliente, eficiente e segura.

3.12.

8.6.

8.6.1. Cliente HTTP com Axios

Para operações CRUD (Create, Read, Update, Delete) tradicionais - como gerenciamento de tarefas, consulta de temperatura e feriados - o aplicativo utiliza requisições HTTP através da biblioteca Axios. Uma instância configurada do Axios é criada durante a inicialização do aplicativo, com base URL definida a partir das configurações do usuário.

A instância Axios é configurada com interceptors que adicionam automaticamente headers necessários, implementam retry logic para falhas temporárias de rede, e convertem erros de rede em objetos de erro consistentes que podem ser tratados uniformemente pela aplicação.

Exemplo de fluxo de requisição HTTP:

- Componente de interface invoca método de serviço (ex: taskService.createTask())
- Serviço constrói requisição HTTP com método, endpoint e payload apropriados
- Axios envia requisição ao servidor
- Servidor processa requisição e retorna resposta
- Axios recebe resposta, valida status code, deserializa JSON
- Serviço retorna dados ao componente chamador
- Componente atualiza interface com dados recebidos

8.6.2. Cliente SignalR

Para funcionalidades de tempo real - principalmente interação por voz - o aplicativo estabelece e mantém conexão SignalR com o servidor. A conexão é gerenciada por um serviço dedicado que encapsula toda complexidade de estabelecimento, manutenção e recuperação de conexão.

O serviço SignalR implementa:

- Conexão automática: ao iniciar o aplicativo, tenta estabelecer conexão com o hub /melissa
- Reconexão automática: em caso de perda de conexão (rede instável, servidor reiniciado), tenta reconectar automaticamente com backoff exponencial
- Heartbeat: envia mensagens periódicas para manter conexão ativa em redes com timeout agressivo de conexões idle
- Eventos: expõe sistema de eventos que permite componentes da aplicação serem notificados sobre mudanças de estado da conexão (conectado, desconectado, reconectando)

8.6.3. Segurança de Comunicação

Embora o sistema opere em rede local controlada pelo usuário, o aplicativo suporta comunicação via HTTPS/TLS quando o servidor está configurado com certificados. O Axios é configurado para validar certificados do servidor, protegendo contra-ataques man-in-the-middle.

Para ambientes de desenvolvimento ou quando o servidor utiliza certificados auto-assinados, o aplicativo pode ser configurado para aceitar certificados não confiáveis, mas essa opção é claramente marcada como insegura na interface de configuração.

8.7 GERENCIAMENTO DE ESTADO

O gerenciamento de estado do aplicativo utiliza React Context API para compartilhar estado global entre componentes sem prop drilling excessivo. Três contextos principais foram implementados:

AuthContext: gerencia estado de autenticação do usuário e configurações de servidor. Embora o sistema atual não implemente autenticação completa (pois opera em rede local confiável), a estrutura está preparada para futuras implementações de login/senha ou autenticação biométrica.

TaskContext: centraliza estado relacionado a tarefas, incluindo lista de tarefas abertas, arquivadas, tarefas carregando, e erros. Fornece métodos para criar, atualizar, deletar e arquivar tarefas, abstraindo chamadas à API e gerenciamento de estado resultante.

ConnectionContext: gerencia estado da conexão SignalR, incluindo status (conectado, desconectado, reconectando) e métodos para enviar/receber mensagens através do hub.

8.8 TESTES E QUALIDADE

O aplicativo móvel foi submetido a extensivo processo de testes para garantir qualidade e confiabilidade:

Testes Unitários: componentes de lógica pura (serviços, utilitários, validações) possuem cobertura de testes unitários utilizando Jest, framework de testes JavaScript.

Estes testes verificam comportamento correto de funções isoladas sem dependências externas.

Testes de Integração: fluxos críticos que envolvem múltiplos componentes e serviços foram testados com abordagem de integração, verificando que componentes funcionam adequadamente quando integrados.

Testes Manuais: todas as telas e funcionalidades foram extensivamente testadas manualmente em dispositivos reais Android e iOS, verificando responsividade, performance e experiência do usuário em diferentes tamanhos de tela e versões de sistema operacional.

8.9 DISTRIBUIÇÃO E INSTALAÇÃO

O aplicativo móvel está disponível para instalação através de dois métodos principais:

Build Local: desenvolvedores e usuários avançados podem clonar o repositório do projeto, instalar dependências utilizando npm ou yarn, e gerar builds locais para Android (APK) ou iOS (IPA) utilizando Expo CLI. Este método oferece máximo controle e permite customizações antes da instalação.

Expo Go: para desenvolvimento e testes rápidos, o aplicativo pode ser executado através do aplicativo Expo Go disponível nas lojas Google Play e App Store. Desenvolvedores iniciam servidor de desenvolvimento utilizando expo start e escaneiam QR code gerado com aplicativo Expo Go para carregar e testar aplicação instantaneamente.

Para distribuição em produção, o aplicativo pode ser publicado nas lojas oficiais (Google Play Store e Apple App Store) seguindo processos padrão de cada plataforma, embora isso não seja obrigatório dado o caráter self-hosted do projeto.

8.10 CONSIDERAÇÕES DE PERFORMANCE

A performance do aplicativo móvel foi cuidadosamente otimizada para proporcionar experiência fluida mesmo em dispositivos com recursos limitados:

Renderização Otimizada: componentes React Native foram otimizados utilizando técnicas como memoization (React.memo), callbacks memoizados (useCallback) e listas virtualizadas (FlatList) para minimizar renderizações desnecessárias.

Lazy Loading: componentes pesados que não são necessários na inicialização do aplicativo são carregados sob demanda, reduzindo tempo de startup.

Cache de Dados: dados que mudam infrequentemente (como lista de feriados) são cacheados localmente no dispositivo para reduzir necessidade de requisições repetidas ao servidor.

Compressão de Imagens: assets visuais foram otimizados e comprimidos para reduzir tamanho do bundle final do aplicativo.

8.11 ACESSIBILIDADE

O aplicativo foi desenvolvido com atenção a recursos de acessibilidade para garantir que possa ser utilizado por pessoas com diferentes necessidades:

Leitores de Tela: todos os componentes interativos possuem labels apropriados que são anunciados por leitores de tela como TalkBack (Android) e VoiceOver (iOS).

Contraste de Cores: a paleta de cores foi escolhida para garantir contraste adequado entre texto e fundo, facilitando leitura para pessoas com deficiências visuais.

Tamanho de Toque: botões e áreas clicáveis respeitam tamanho mínimo de 44x44 pixels recomendados por diretrizes de acessibilidade móvel.

Navegação por Teclado: em dispositivos com teclado físico ou teclado bluetooth conectado, o aplicativo suporta navegação completa sem necessidade de toque na tela.

8.12 ROADMAP E MELHORIAS FUTURAS

O aplicativo móvel Melissa representa versão inicial funcional, mas há espaço significativo para melhorias e novas funcionalidades:

Notificações Push: implementação de notificações push para alertar usuário sobre tarefas vencendo, respostas da assistente quando aplicativo está em segundo plano, ou atualizações importantes.

Widgets: criação de widgets para tela inicial do dispositivo, permitindo visualização rápida de informações como próximas tarefas ou temperatura atual sem abrir aplicativo.

Modo Offline: aprimoramento de funcionalidades offline, permitindo uso básico do aplicativo mesmo sem conexão ao servidor, com sincronização posterior quando conexão for restabelecida.

Personalização Visual: opções de temas (claro/escuro) e customização de cores primárias conforme preferência do usuário.

Integração com Calendário: sincronização bidirecional de tarefas com calendário nativo do dispositivo.

Reconhecimento de Contexto: utilização de sensores do dispositivo (localização, horário, padrões de uso) para oferecer sugestões proativas da assistente.

9 CONCLUSÃO

O desenvolvimento da Assistente Pessoal Inteligente Melissa demonstrou ser plenamente viável a criação de uma solução robusta de assistente virtual que respeita a privacidade do usuário sem comprometer funcionalidades essenciais. Ao longo deste trabalho, foi possível integrar tecnologias consolidadas de código aberto, como .NET, Ollama, MySQL e Arduino, em uma arquitetura coesa que opera inteiramente em infraestrutura local controlada pelo usuário. Os testes realizados comprovaram que o sistema é capaz de responder adequadamente a comandos de voz e texto, controlar dispositivos inteligentes, acessar informações externas quando autorizado, e manter conversas naturais através de modelos de linguagem de grande escala, tudo isso sem transmitir dados pessoais para servidores de terceiros. Os obstáculos técnicos encontrados, como a imprevisibilidade ocasional do modelo de linguagem e desafios na comunicação entre dispositivos IoT e servidor, foram mitigados através de mecanismos de reinicialização automática, tratamento robusto de exceções e implementação de estratégias de retry, resultando em um sistema estável e confiável.

Os resultados obtidos confirmam que a dicotomia frequentemente apresentada entre funcionalidades avançadas de inteligência artificial e respeito à privacidade é falsa. É possível, através de software livre e arquiteturas descentralizadas, oferecer experiências tecnológicas sofisticadas que não apenas preservam, mas fortalecem a autonomia digital dos usuários. A disponibilização do código-fonte completo sob licença MIT contribui para o movimento de democratização da tecnologia, permitindo que qualquer pessoa com conhecimentos técnicos adequados audite o funcionamento do sistema, verifique a ausência de funcionalidades ocultas de coleta de dados, ou adapte a solução para necessidades específicas. Esta transparência representa um contraste fundamental com as soluções proprietárias dominantes no mercado, onde os usuários devem confiar cegamente nas declarações de privacidade das corporações.

As experiências adquiridas durante o desenvolvimento deste projeto transcendem os aspectos puramente técnicos, abrangendo reflexões importantes sobre ética tecnológica, responsabilidade no tratamento de dados pessoais e o papel dos profissionais de tecnologia na construção de um futuro digital mais justo e respeitoso. Como trabalhos futuros, propõe-se a expansão do sistema com suporte

a múltiplos idiomas além do português, desenvolvimento de interfaces gráficas para facilitar a configuração por usuários não técnicos, integração com protocolos padronizados de automação residencial como Zigbee e Z-Wave, implementação de mecanismos de sincronização multi-dispositivo, e exploração de técnicas de fine-tuning dos modelos de linguagem para otimizar respostas em domínios específicos. Espera-se que este trabalho inspire outras instituições e pesquisadores a investigarem alternativas éticas e centradas no usuário para tecnologias que hoje são dominadas por grandes corporações, contribuindo para um ecossistema tecnológico mais diversos, transparente e alinhado com os direitos fundamentais dos cidadãos.

REFERÊNCIAS

Software e privacidade: uma defesa do código-fonte aberto na preservação do direito constitucional à vida privada. Revista do CAAP, [S. l.], v. 9, n. 1, p. 121–138, 2024. Disponível em: https://periodicos.ufmg.br/index.php/caap/article/view/47442. Acesso em: 27 abr. 2025.

CRISCUOLO, Leandro Costa. OpenAI leva multa milionária por uso indevido de dados na Itália. Olhar Digital, 20 dez. 2024. Disponível em: https://olhardigital.com.br/2024/12/20/pro/openai-leva-multa-milionaria-por-uso-indevido-de-dados-na-italia/. Acesso em: 27 abr. 2025

FIGÊNIO, Mateus R.; GOMES-JR, Luiz. Ética na era dos Modelos de Linguagem Massivos (LLMs): um estudo de caso do ChatGPT. In: ESCOLA REGIONAL DE BANCO DE DADOS (ERBD). Escola Regional de Banco de Dados (ERBD). SBC, 11 abr. 2023. Disponível em: https://sol.sbc.org.br/index.php/erbd/article/view/24351. Acesso em: 13 abr. 2025

GONÇALVES, André et al. IEEE Std 830 Prática Recomendada Para Especificações de Exigências de Software. [S.d.].

JONAS, K.; KANZOW, P.; KRETSCHMER, M. Audio streaming on the Internet. Experiences with real-time streaming of audio streams. In: ISIE '97 PROCEEDING OF THE IEEE INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS. ISIE '97 Proceeding of the IEEE International Symposium on Industrial Electronics. jul. 1997. Disponível em: https://ieeexplore.ieee.org/abstract/document/651738. Acesso em: 13 abr. 2025

JULIÃO, Gabriel; SILVA, Isabella Manoel da. Privacidade e segurança na era da IoT em residências: possibilidades e desafios. 21 jun. 2024.

LIMBERGER, Têmis; SANTANNA, Gustavo; DA SILVA GIANNAKOS, Demétrio Beck. Internet das coisas (IoT) e os direitos à privacidade e à proteção de dados do cidadão: uma necessária aproximação. Revista Brasileira de Políticas Públicas, v. 13, n. 3, p. 116, 1 dez. 2023a.

LIMBERGER, Têmis; SANTANNA, Gustavo; DA SILVA GIANNAKOS, Demétrio Beck. Internet das coisas (IoT) e os direitos à privacidade e à proteção de dados do cidadão: uma necessária aproximação. | EBSCOhost. Disponível em: https://openurl.ebsco.com/contentitem/doi:10.5102%2Frbpp.v13i3.8536?sid=ebsco:plink:crawler&id=ebsco:doi:10.5102%2Frbpp.v13i3.8536. Acesso em: 13 abr. 2025b.

LIMBERGER, Têmis; SANTANNA, Gustavo Da Silva; GIANNAKOS, Demétrio Beck Da Silva. Internet das coisas (IoT) e os direitos à privacidade e à proteção de dados do cidadão: uma necessária aproximação. Revista Brasileira de Políticas Públicas, v. 13, n. 3, 16 fev. 2024.

LIVERO, Nickolas J. S.; SILVA, Fabio S. Desenvolvimento de um Assistente Virtual Baseado em Voz e LLMs para Facilitar a Interação de Estudantes com Deficiência Visual com Sistemas Operacionais. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO (SBIE). Simpósio Brasileiro de Informática na Educação (SBIE).

SBC, 4 nov. 2024. Disponível em: https://sol.sbc.org.br/index.php/sbie/article/view/31461. Acesso em: 13 abr. 2025

MACHADO, Cynthia Semíramis Figueiredo. Software e privacidade: uma defesa do código-fonte aberto na preservação do direito constitucional à vida privada. Revista do CAAP, v. 9, n. 1, p. 121–138, 2001.

MARGARIDA SOARES. Impacto do Chat GPT na sociedade. The Trends Hub, n. 3, 26 jun. 2023.

NEVES, Breno Braga et al. Explorando o Potencial e a Viabilidade de LLMs Open-Source na Análise de Sentimentos. In: CONGRESSO BRASILEIRO DE SOFTWARE: TEORIA E PRÁTICA (CBSOFT). Congresso Brasileiro de Software: Teoria e Prática (CBSoft). SBC, 30 set. 2024. Disponível em: https://sol.sbc.org.br/index.php/cbsoft_estendido/article/view/30249. Acesso em: 13 abr. 2025

OLIVEIRA, Sérgio de. Internet das Coisas com ESP8266, Arduino e Raspberry Pi 2a edição: Atualizado para ESP32. [S.l.]: Novatec Editora, 2021.

ROCHO, Rafael Selau M. UNIVERSIDADE FEDERAL DE SANTA CATARINA CENTRO DE CIÊNCIAS, TECNOLOGIA E SAÚDE - CAMPUS ARARANGUÁ CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO. [S.d.].

SRIDEVI KAKOLU; FAHEEM, Muhammad Ashraf. Building Trust with Generative AI Chatbots: Exploring Explainability, Privacy, and User Acceptance. Unpublished, 2024. Disponível em: https://rgdoi.net/10.13140/RG.2.2.27825.60006. Acesso em: 13 abr. 2025

VIVILUSOR. Apresentando Meta Llama 3: o grande modelo de linguagem de código aberto mais capaz até hoje. Sobre a Meta, 18 abr. 2024. Disponível em: https://about.fb.com/br/news/2024/04/apresentando-meta-llama-3-o-grande-modelo-de-linguagem-de-codigo-aberto-mais-capaz-ate-hoje/. Acesso em: 27 abr. 2025

FISCHER, Luiz. Algumas práticas recomendadas para desenvolvimento de aplicativos C#. [S.l.], [20--]. Disponível em: https://www.dio.me/articles/algumas-praticas-recomendadas-para-desenvolvimento-de-aplicativos-c. Acesso em: 20 nov. 2025

OPENAI. ChatGPT. Versão GPT-4. 2024. Disponível em: https://chat.openai.com. Acesso em: 17 abr. 2024

META. Apresentando Meta Llama 3: o grande modelo de linguagem de código aberto mais capaz até hoje. 18 abr. 2024. Disponível em: https://about.fb.com/br/news/2024/04/apresentando-meta-llama-3-o-grande-modelo-de-linguagem-de-codigo-aberto-mais-capaz-ate-hoje/. Acesso em: 27 abr. 2024.

JUDE, Daniel. How to Install and Configure Ollama: Run AI Models Locally. Medium, [S.l.], 2024. Disponível em: https://danieljude1992.medium.com/how-to-install-and-configure-ollama-to-run-llms-locally-d9de04588027. Acesso em: 20 nov. 2025

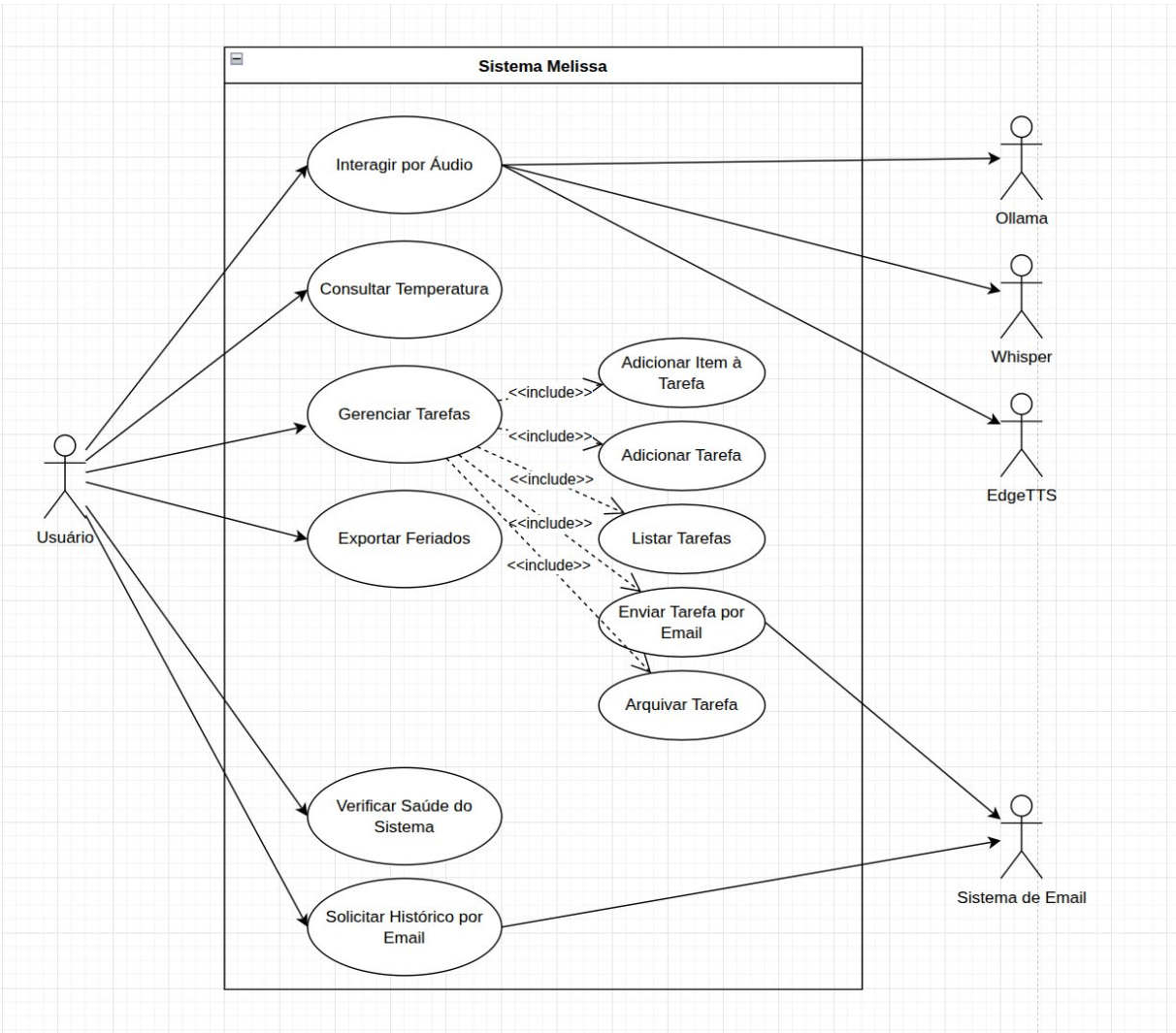
1000. Logos. Disponível em: https://1000logos.net/mysql-logo/. Acesso em: 20 nov. 2025

WIKIMEDIA COMMONS. File: Arduino IDE logo.svg. Disponível em: https://commons.wikimedia.org/wiki/File:Arduino_IDE_logo.svg. Acesso em: 20 nov. 2025

WIKIPÉDIA. Ficheiro: Docker (container engine) logo.svg. Disponível em: https://pt.wikipedia.org/wiki/Ficheiro:Docker_%28container_engine%29_logo.svg. Acesso em: 20 nov. 2025

APÊNDICE A: DIAGRAMA DE CASOS DE USO

Figura 15 - Diagrama de Casos de Uso



Fonte: Do próprio autor

APÊNDICE B: DESCRIÇÕES DE CASOS DE USO

- **CDU 01: Interagir por Áudio**

Descrição: Este caso de uso permite ao usuário interagir com a assistente através de comandos de voz. O usuário envia um fluxo de áudio para o sistema, que utiliza o serviço Whisper para transcrever a fala em texto. O texto é então processado para entender a intenção, possivelmente com o auxílio do serviço Ollama para inteligência artificial. Por fim, o sistema gera uma resposta em texto e a converte em áudio sintetizado usando o serviço EdgeTTS, retornando a resposta em voz para o usuário.

- **CDU 02: Consultar Temperatura**

Descrição: O usuário solicita a temperatura atual de uma localidade específica. O sistema recebe o nome da cidade, busca a informação em uma fonte externa e retorna o dado de temperatura para o usuário.

- **CDU 03: Gerenciar Tarefas**

Descrição: Este é um caso de uso abrangente que permite ao usuário realizar o gerenciamento completo de suas listas de tarefas. Ele serve como um ponto de entrada para diversas sub-funções, como adicionar, listar, modificar e compartilhar tarefas.

- **CDU 04: Adicionar Tarefa**

Descrição: O usuário cria uma lista de tarefas, fornecendo um título e, opcionalmente, uma descrição. O sistema armazena essa nova tarefa para gerenciamento futuro. Este caso de uso é uma especialização de "Gerenciar Tarefas".

- **CDU 05: Adicionar Item à Tarefa**

Descrição: O usuário adiciona um item específico a uma tarefa já existente. Para isso, é necessário informar o identificador da tarefa e a descrição do novo item a ser adicionado. Este caso de uso é uma especialização de "Gerenciar Tarefas".

- **CDU 06: Listar Tarefas**

Descrição: O usuário solicita a visualização de suas tarefas. O sistema retorna a lista completa de tarefas ou os itens contidos dentro de uma tarefa específica, permitindo que o usuário acompanhe suas pendências. Este caso de uso é uma especialização de "Gerenciar Tarefas".

- **CDU 07: Enviar Tarefa por Email**

Descrição: O usuário solicita que uma lista de tarefas seja enviada para um endereço de e-mail específico. O sistema formata as informações da tarefa e interage com um Sistema de Email externo para realizar o envio. Este caso de uso é uma especialização de "Gerenciar Tarefas".

- **CDU 08: Arquivar Tarefa**

Descrição: O usuário pode arquivar ou desarquivar uma tarefa existente. Esta ação serve para ocultar tarefas da lista principal sem excluí-las permanentemente, ajudando na organização. Este caso de uso é uma especialização de "Gerenciar Tarefas".

- **CDU 09: Exportar Feriados**

Descrição: O usuário solicita a exportação da lista de feriados nacionais para um arquivo de texto. O sistema gera o arquivo no servidor, que pode ser utilizado para consultas offline.

- **CDU 10: Solicitar Histórico por Email**

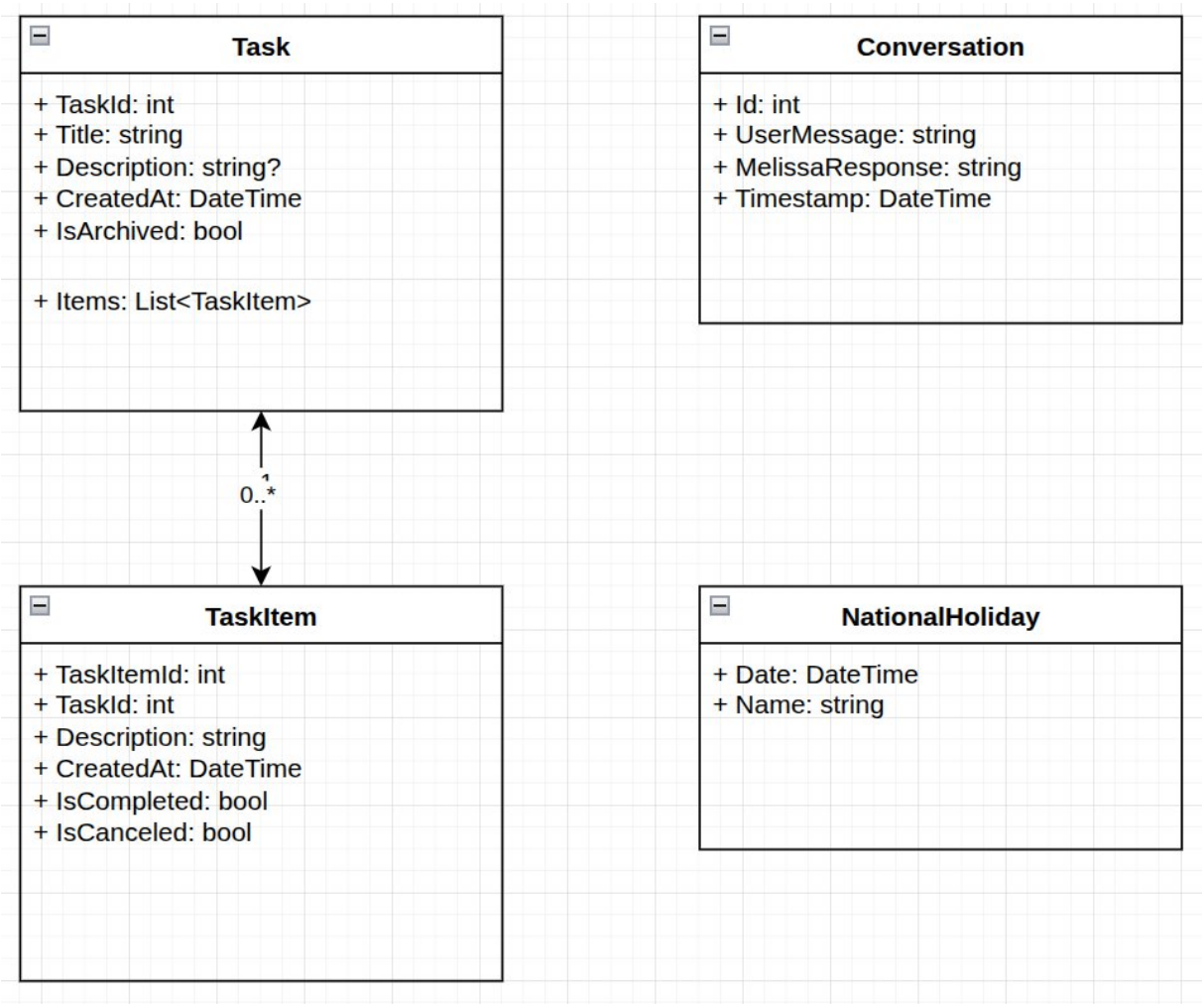
Descrição: O usuário solicita o envio de seu histórico de conversas com a assistente para um endereço de e-mail. É possível especificar um período (data de início e fim) para filtrar o histórico. O sistema coleta os dados e utiliza um Sistema de Email externo para enviar o relatório.

- **CDU 11: Verificar Saúde do Sistema**

Descrição: O usuário (ou um sistema automatizado) pode fazer uma requisição para verificar se a assistente Melissa está operacional e pronta para receber comandos. O sistema retorna um status de sucesso ou falha.

APÊNDICE C: DIAGRAMA DE CLASSES DE DADOS

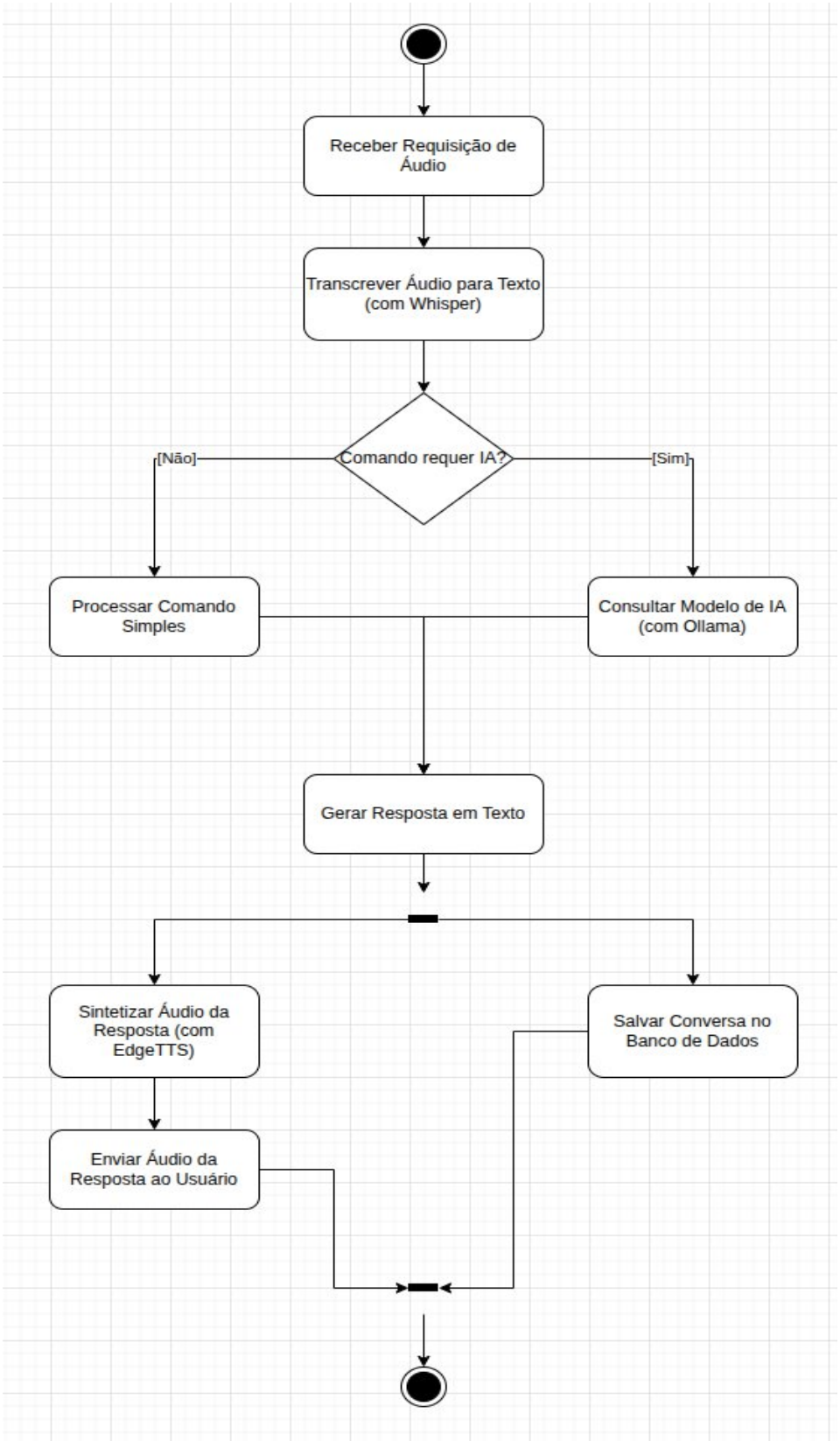
Figura 16 - Diagrama de classes de dados



Fonte: Do próprio autor

APÊNDICE D: DIAGRAMA DE ATIVIDADES

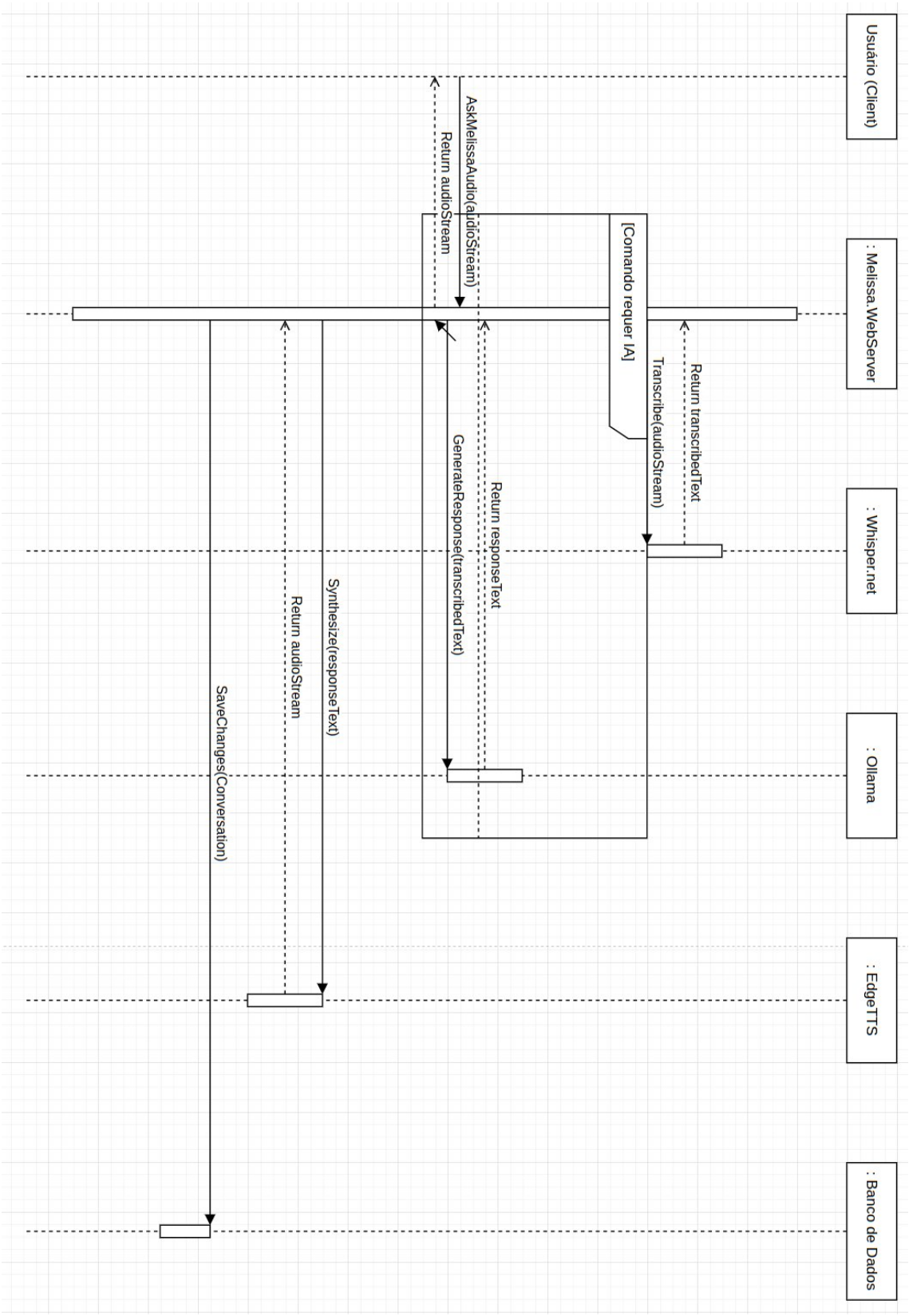
Figura 17 - Diagrama de atividades



Fonte: Do próprio autor

APÊNDICE E: DIAGRAMA DE SEQUÊNCIA

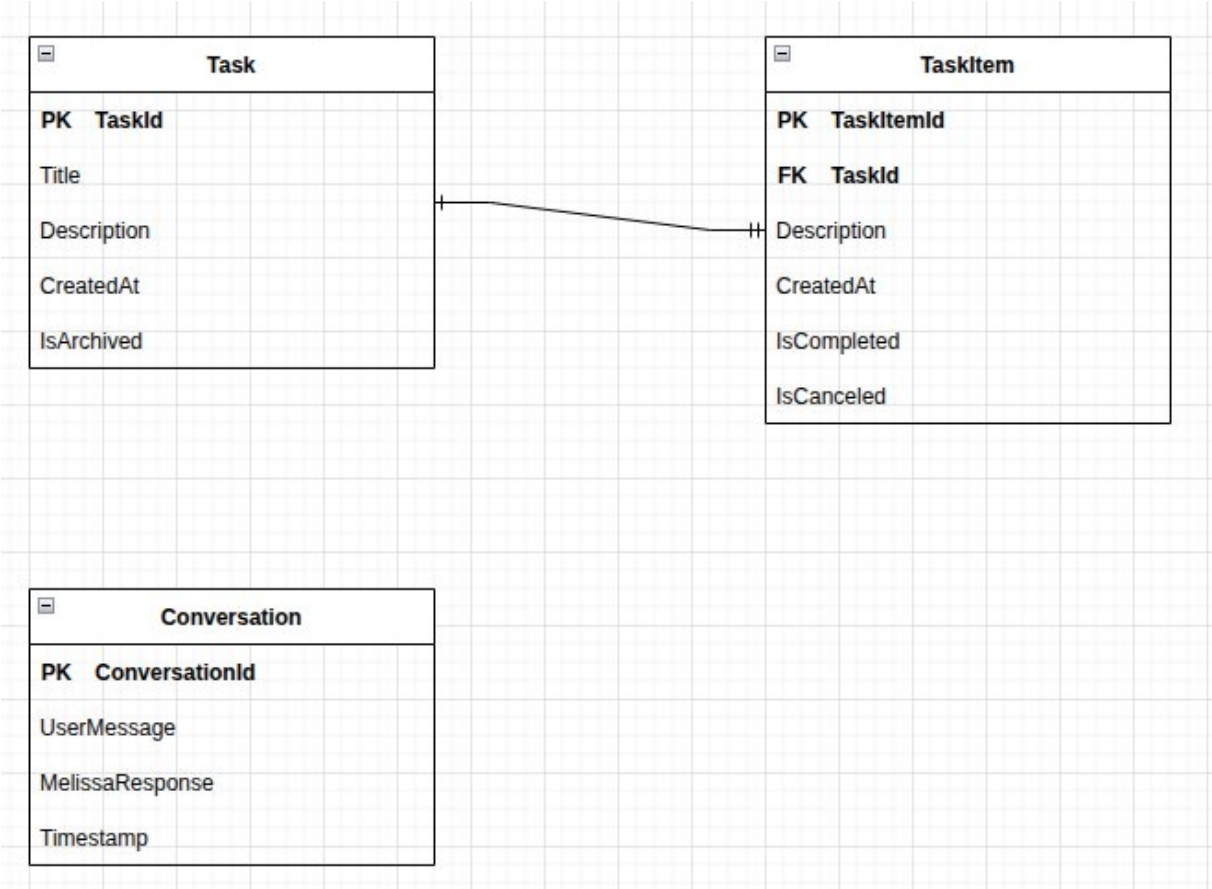
Figura 18 - Diagrama de Sequência



Fonte: Do próprio autor

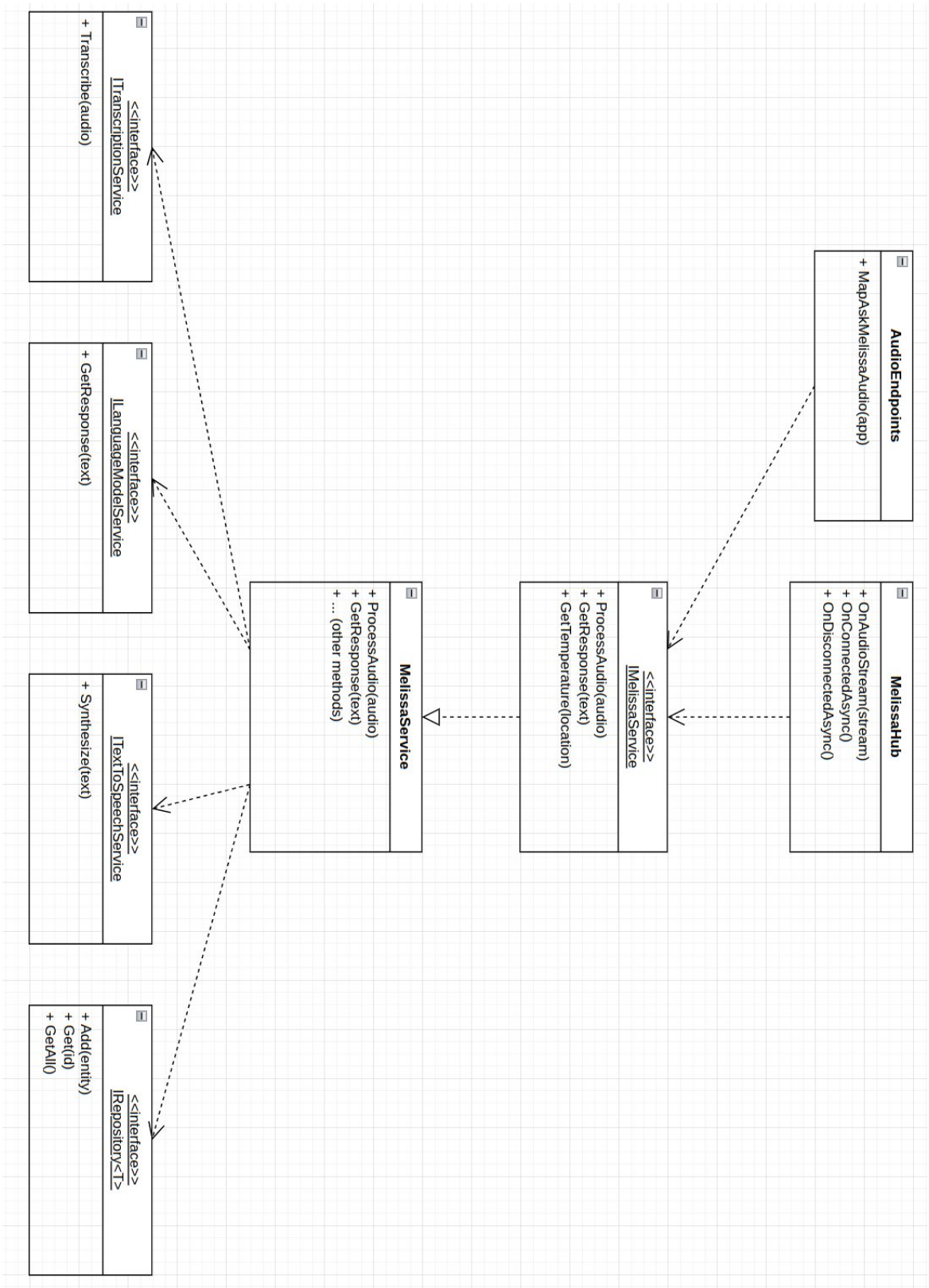
APÊNDICE F: DIAGRAMA DE ENTIDADE E RELACIONAMENTO

Figura 19 - Diagrama de Entidade e Relacionamento das Entidades de Dados



Fonte: Do próprio autor

Figura 20 - Diagrama de Entidade e Relacionamento das Classes de Serviço



Fonte: Do próprio autor

MANUAL DO USUÁRIO

Introdução

O sistema Melissa é a infraestrutura de backend para uma assistente virtual inteligente, projetada para processar comandos, gerenciar tarefas e fornecer informações em tempo real. A arquitetura é baseada em um servidor web desenvolvido em .NET, que disponibiliza uma API RESTful e um hub de comunicação para interações dinâmicas. O sistema foi construído para ser flexível, permitindo a integração com ferramentas de transcrição de áudio como o Whisper, e modelos de linguagem locais, como os disponibilizados via Ollama, para garantir a privacidade e o controle do usuário.

A interação principal com o sistema é realizada por meio de requisições HTTP, permitindo que diferentes aplicações cliente (desktop, mobile ou web) se conectem e utilizem suas funcionalidades.

Funcionalidades Principais

O sistema oferece um conjunto de funcionalidades focadas em assistência pessoal e gerenciamento de informações.

Interação por Áudio

A funcionalidade central da assistente é a capacidade de receber comandos de voz e responder de forma sintetizada. O usuário pode enviar um arquivo de áudio bruto (no formato PCM, 16kHz, 16-bit, mono) para o endpoint POST /melissa/AskMelissaAudio. O servidor, então, realiza o seguinte processo:

1. Recebe o áudio e o converte para um formato compatível (WAV).
2. Utiliza o serviço de transcrição Whisper para converter a fala em texto.
3. Processa o texto para identificar o comando ou a pergunta do usuário.
4. Gera uma resposta em texto e a sintetiza em áudio (formato MP3) usando a tecnologia Edge TTS.
5. Retorna o áudio MP3 como resposta, permitindo uma interação por voz completa.

Além disso, o sistema dispõe de um hub SignalR no caminho /melissa, que possibilita uma comunicação em tempo real e de baixa latência. Essa via é ideal para aplicações cliente que necessitam de uma conexão persistente para streaming de áudio ou para receber notificações instantâneas do servidor.

Gerenciamento de Tarefas (TaskList)

Melissa inclui um sistema robusto para gerenciamento de listas de tarefas. O usuário pode interagir com este módulo através dos seguintes endpoints:

- Criar e Gerenciar Tarefas: É possível adicionar uma nova tarefa fornecendo um título e uma descrição opcional através do endpoint POST /melissa/AddNewTask. Uma vez criada, itens específicos podem ser adicionados a ela usando POST /melissa/AddNewItemTask, informando o ID da tarefa e a descrição do item.
- Visualizar Tarefas: O usuário pode listar todas as tarefas existentes com GET /melissa/GetAllTasks ou visualizar os itens de uma tarefa específica com GET /melissa/GetAllItensByTaskId.
- Modificar Status de Itens: Itens de uma tarefa podem ser marcados como concluídos (POST /melissa/CompleteItemTask) ou cancelados (POST /melissa/CancelTaskItemById), facilitando o acompanhamento do progresso.
- Organização e Exportação: Tarefas podem ser arquivadas ou desarquivadas (POST /melissa/ArchiveTaskById e POST /melissa/UnarchiveTaskById). Para compartilhamento, uma lista de tarefas pode ser enviada por e-mail para um destinatário específico através de POST /melissa/SendTaskByEmail.

Consultas e Ferramentas Utilitárias

Para além do gerenciamento de tarefas, a assistente pode buscar e apresentar informações úteis:

- **Consulta de Temperatura:** Ao acessar GET /melissa/GetCurrentTemperatureByLocation, o usuário pode obter a temperatura atual de uma cidade informada como parâmetro.
- **Exportação de Feriados:** O sistema pode gerar um arquivo de texto (.txt) com a lista de feriados nacionais do ano corrente através do endpoint GET /melissa/ExportNationalHolidaysToTxt.
- **Histórico de Conversas:** Para fins de auditoria ou consulta, o usuário pode solicitar o envio do histórico de suas conversas com a assistente por e-mail, especificando um período de início e fim, através de POST /melissa/SendEmailConversationHistoryByPeriod.

Verificação de Status do Sistema

Para garantir que a assistente esteja operacional, o sistema fornece um endpoint de verificação de saúde (health check) em GET /health. Uma resposta com status 200 OK indica que o servidor está funcionando corretamente e pronto para receber comandos. Caso contrário, um código de erro 500 será retornado, informando que a assistente está indisponível.

Este manual descreve a perspectiva do usuário final e das aplicações cliente que consomem a API da Melissa, detalhando as capacidades e os meios de interação disponíveis para uma experiência de assistência virtual completa e funcional.

MANUAL DE INSTALAÇÃO

Introdução

Este manual fornece instruções para configurar e executar o servidor da assistente virtual Melissa. Existem dois métodos de instalação principais: local, ideal para desenvolvimento e testes, e via Docker/Docker Compose, recomendada para um ambiente de produção ou para simplificar o gerenciamento de dependências.

Requisitos do Sistema

Antes de iniciar, certifique-se de que seu ambiente atende aos seguintes requisitos:

- Sistema Operacional: Linux, macOS ou Windows.
- .NET SDK: É necessária a versão 9.0 ou superior. Você pode baixá-la no site oficial da Microsoft.
- Docker (Opcional): Para a instalação via contêineres, é necessário ter o docker e o docker-compose (ou docker compose) instalados.
- Ollama (Opcional): Se desejar utilizar modelos de linguagem locais para processamento, instale o Ollama a partir do site oficial do Ollama.

Método 1: Instalação Local (para Desenvolvimento)

Este método é ideal para desenvolvedores que desejam executar o servidor diretamente em sua máquina local para fins de teste ou contribuição.

Passo 1: Clonar o Repositório

Primeiro, clone o repositório do projeto para a sua máquina local.

```
git clone https://github.com/daniloflorenzano/Melissa.git
cd Melissa
```

Passo 2: Instalar Dependências e Compilar o Projeto

Com o .NET 9 SDK já instalado, navegue até a pasta do projeto e use os comandos do .NET CLI para restaurar as dependências, compilar e executar o servidor.

Navegue até a pasta que contém a solução
cd src/Melissa

Restaure os pacotes NuGet do projeto
dotnet restore Melissa.WebServer/Melissa.WebServer.csproj

Compile o projeto em modo de Debug
dotnet build Melissa.WebServer/Melissa.WebServer.csproj -c Debug

Execute o servidor
dotnet run --project Melissa.WebServer/Melissa.WebServer.csproj

Após a execução do último comando, o servidor estará ativo e escutando na porta 8080 (por padrão). Você pode verificar se ele está funcionando acessando http://localhost:8080/health em seu navegador ou via curl.

Configuração Local: Para desenvolvimento, as configurações como chaves de API (AllUNeedApiKey) podem ser ajustadas no arquivo src/Melissa/Melissa.WebServer/appsettings.Development.json ou através de variáveis de ambiente.

Método 2: Execução com Docker / Docker Compose

Este método utiliza contêineres para encapsular a aplicação e suas dependências, garantindo um ambiente consistente e facilitando a implantação. O repositório já inclui os arquivos necessários (Dockerfile e compose.yaml).

Passo 1: Preparar o Ambiente Docker

Certifique-se de que o Docker está em execução em sua máquina. Clone o repositório, se ainda não o fez.

```
git clone https://github.com/daniloflorenzano/Melissa.git
cd Melissa
```

Passo 2: Configurar Variáveis de Ambiente (Opcional)

O arquivo src/Melissa/compose.yaml contém as configurações para os serviços. Você pode editar este arquivo diretamente para ajustar parâmetros, como a chave AllUNeedApiKey.

Exemplo de variáveis no compose.yaml:

```
HolidaysCsvPath=/app/Data/holidays_2025.csv
AllUNeedApiUrl=https://alluneed.com.br/api/
AllUNeedApiKey= (deixe vazio para não usar ou insira sua chave)
OLLAMA_URL=http://ollama:11434
```

Passo 3: Iniciar os Contêineres

Navegue até a pasta que contém o arquivo compose.yaml e execute os comandos do Docker Compose para construir a imagem e iniciar os serviços.

```
# Navegue até a pasta correta
cd src/Melissa

# Construa a imagem e suba os serviços em primeiro plano
docker compose -f compose.yaml up --build

# Para rodar os serviços em segundo plano (detached mode)
docker compose -f compose.yaml up -d --build
```

Os serviços definidos no compose.yaml são:

melissa.webserver: O servidor da assistente, que será exposto na porta 8080.
ollama: O serviço do Ollama para executar modelos de linguagem localmente (opcional, mas recomendado).

Passo 4: Parar os Contêineres

Para parar e remover os contêineres, execute o seguinte comando no mesmo diretório:

docker compose -f compose.yaml down

Solução de Problemas Comuns (Troubleshooting)

Erro de holidays_2025.csv não encontrado: Verifique se o arquivo está presente em src/Melissa/Melissa.WebServer/Data/. Se estiver usando Docker, o caminho já está configurado no compose.yaml.
Assistente indisponível (/health retorna erro): Verifique os logs da aplicação. Se estiver usando Docker, certifique-se de que o contêiner ollama foi iniciado corretamente e que a variável OLLAMA_URL está configurada corretamente no compose.yaml.
Falha na transcrição de áudio: A aplicação tentará baixar o modelo Whisper (ggml-medium.bin) na primeira execução. Garanta que o servidor tenha acesso à internet.
Problemas com a síntese de voz (TTS): Verifique se o ambiente de execução tem as permissões e dependências necessárias para o mecanismo Edge TTS funcionar.