UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

MICAEL RIBEIRO ROCHA

SPEAKUP: APLICAÇÃO WEB PARA A APRENDIZAGEM DE IDIOMAS APOIADA POR INTELIGÊNCIA ARTIFICIAL

CORNÉLIO PROCÓPIO

2025

MICAEL RIBEIRO ROCHA

SPEAKUP: APLICAÇÃO WEB PARA A APRENDIZAGEM DE IDIOMAS APOIADA POR INTELIGÊNCIA ARTIFICIAL

SPEAKUP: WEB APPLICATION FOR LANGUAGE LEARNING SUPPORTED BY ARTIFICIAL INTELLIGENCE

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Software do Curso de Bacharelado em Engenharia de Software da Universidade Tecnológica Federal do Paraná.
Orientador(a): Profª. Drª. Rosangela de Fátima Pereira Marquesone

CORNÉLIO PROCÓPIO

2025

MICAEL RIBEIRO ROCHA

SPEAKUP: APLICAÇÃO WEB PARA A APRENDIZAGEM DE IDIOMAS APOIADA POR INTELIGÊNCIA ARTIFICIAL

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Software do Curso de Bacharelado em Engenharia de Software da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 25/junho/2025

Rosangela de Fátima Pereira Marquesone
Doutorado em Engenharia Elétrica
Universidade Tecnológica Federal do Paraná

María Eugenia Dajer
Doutorado em Engenharia Elétrica
Universidade Tecnológica Federal do Paraná

Henrique Yoshikazu Shishido
Doutorado em Ciência da Computação.
Universidade Tecnológica Federal do Paraná

CORNÉLIO PROCÓPIO
2025

Dedico este trabalho ao meu avô Bibiano Pereira Rocha, cuja memória e exemplo de coragem continuam a me inspirar todos os dias. À minha família, pelo amor, apoio e compreensão nos momentos de ausência. Com gratidão e carinho.

AGRADECIMENTOS

Reconheço que seria impraticável mencionar nominalmente todas as pessoas que, de alguma forma, contribuíram para a realização desta etapa tão relevante em minha trajetória acadêmica e pessoal. A todos que, mesmo não citados individualmente, participaram deste percurso com gestos de apoio, palavras de incentivo ou simples demonstrações de afeto e compreensão, expresso minha sincera e profunda gratidão. Cada contribuição, por menor que tenha parecido, foi essencial para a concretização deste trabalho.

Dirijo meus agradecimentos, em primeiro lugar, à minha família, cuja presença constante, apoio incondicional e valores transmitidos foram fundamentais para minha formação. Agradeço pelo amor incansável, pela paciência durante os momentos desafiadores e pela confiança depositada em minha capacidade de seguir em frente. Sem essa base sólida, este caminho teria sido incomparavelmente mais árduo. Aos meus pais e irmã, minha eterna gratidão pela inspiração e exemplo de dedicação.

Aos amigos e amigas que estiveram ao meu lado ao longo dessa jornada, registro minha profunda estima e reconhecimento. Suas palavras de encorajamento nos momentos de incerteza, a escuta atenta diante das dificuldades e os momentos de descontração e leveza foram decisivos para que eu mantivesse o equilíbrio emocional necessário para enfrentar os desafios acadêmicos. A amizade de vocês foi um alicerce valioso e inesquecível.

À minha orientadora, manifesto especial gratidão pela orientação cuidadosa, pela disponibilidade constante para o diálogo e pela postura ética e comprometida que nortearam todas as etapas deste trabalho. Sua expertise, generosidade intelectual e dedicação foram determinantes para o meu crescimento acadêmico e pessoal. Seu exemplo profissional permanecerá como uma referência sólida em minha carreira futura.

Estendo meus agradecimentos à Universidade Tecnológica Federal do Paraná (UTFPR), instituição que me acolheu e proporcionou uma formação de excelência. Sou profundamente grato pela estrutura acadêmica, pelas oportunidades de envolvimento em projetos de ensino, pesquisa e extensão, e pelo ambiente de convivência com docentes e colegas que tanto contribuíram para minha formação integral. Essa vivência universitária foi essencial para o amadurecimento de minha visão crítica e humana.

Por fim, agradeço a todos os professores, técnicos administrativos, colegas de curso e demais membros da comunidade acadêmica que, direta ou indiretamente, participaram deste processo formativo. Cada interação, orientação, crítica construtiva e oportunidade de aprendizado deixou uma marca valiosa em minha trajetória. Levarei comigo não apenas o conhecimento adquirido, mas também os vínculos construídos e as experiências que moldaram quem sou hoje, com a certeza de que essa etapa foi apenas o início de uma caminhada comprometida com o conhecimento, a ética e a transformação social.

Primeira Lei: Um robô não pode ferir um ser humano ou, por omissão, permitir que um ser humano sofra algum mal. Segunda Lei: Um robô deve obedecer as ordens que lhe sejam dadas por seres humanos, exceto nos casos em que tais ordens contrariem a Primeira Lei. Terceira Lei: Um robô deve proteger sua própria existência desde que tal proteção não entre em conflito com a Primeira e Segunda Leis.

(ASIMOV, Isaac, 1950)

RESUMO

Este trabalho apresenta o desenvolvimento do *SpeakUp*, uma aplicação web interativa voltada ao ensino de idiomas com suporte de inteligência artificial. A proposta surgiu da necessidade de oferecer uma alternativa prática e personalizada para o aprendizado de línguas, especialmente em contextos em que o acesso a professores ou cursos presenciais é limitado. O sistema combina tecnologias como Processamento de Linguagem Natural (PLN), geração automática de respostas e correção gramatical, permitindo ao usuário praticar conversação por meio de interações com um chatbot. A metodologia adotada envolveu levantamento bibliográfico, definição dos requisitos, escolha das tecnologias e implementação da solução com base nas tecnologias React, Go e MongoDB. Como resultado, foi desenvolvido um protótipo funcional hospedado em ambiente de nuvem, capaz de adaptar o conteúdo às necessidades linguísticas e ao nível do usuário. Conclui-se que o *SpeakUp* apresenta um potencial significativo como ferramenta complementar para o ensino de línguas estrangeiras, destacando-se pela interatividade, acessibilidade e uso eficiente de recursos tecnológicos.

Palavras-chave: ensino de idiomas; plataforma web; inteligência artificial; processamento de linguagem natural; golang.

ABSTRACT

This work presents the development of SpeakUp, an interactive web application focused on language learning with the support of artificial intelligence. The project arose from the need to provide a practical and personalized alternative for language acquisition, especially in contexts where access to teachers or face-to-face courses is limited. The system integrates technologies such as *Natural Language Processing* (NLP), automatic response generation, and grammatical correction, allowing users to practice conversation through interactions with a chatbot. The methodology included a literature review, requirements definition, technology selection, and implementation using React, Go, and MongoDB. As a result, a functional prototype was developed and deployed in a cloud environment, capable of adapting content to the user's linguistic needs and proficiency level. It is concluded that SpeakUp shows significant potential as a complementary tool for learning foreign languages, standing out for its interactivity, accessibility, and efficient use of technological resources.

Keywords: language learning; web platform; artificial intelligence; natural language processing; golang.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
ARM	*Advanced Reduced Instruction Set Computing (RISC) Machine*
BLE	*Bluetooth Low Energy*
BSON	*Binary JavaScript Object Notation (JSON)*
CPU	*Central Processing Unit*
CRUD	*Create, Read, Update, Delete*
DDoS	*Distributed Denial of Service*
DNS	*Domain Name System*
GAN	*Generative Adversarial Networks*
GHz	*Gigahertz*
GPIO	*General Purpose Input/Output*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
IA	Inteligência Artificial
IDE	*Integrated Development Environment*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
LMS	*Learning Management Systems*
MER	Modelo Entidade-Relacionamento
NLP	*Natural Language Processing*
NoSQL	*Not Only Structured Query Language (SQL)*
PLN	Processamento de Linguagem Natural
PNE	Plano Nacional de Educação

RAM	*Random Access Memory*
REST	*Representational State Transfer*
RISC	*Reduced Instruction Set Computing*
SBC	*Single Board Computer*
SGBD	Sistema de Gerenciamento de Banco de Dados
SoC	*System-on-a-Chip*
SQL	*Structured Query Language*
SSH	*Secure Shell*
SSL	*Secure Sockets Layer*
UML	*Unified Modeling Language*
VM	*Virtual Machine*

SUMÁRIO

1 INTRODUÇÃO

O domínio de idiomas estrangeiros tornou-se uma competência essencial no mundo globalizado atual. A fluência em uma língua estrangeira não apenas amplia o acesso à informação e à cultura, mas também representa uma vantagem competitiva no mercado de trabalho, especialmente em setores que exigem comunicação internacional (Magazine, 2023). Além disso, o conhecimento de línguas estrangeiras é um facilitador da mobilidade acadêmica e profissional, contribuindo para a inserção em contextos multiculturais.

Entretanto, o ensino tradicional de línguas enfrenta desafios consideráveis. Entre eles destacam-se os elevados custos com cursos e materiais e a rigidez metodológica nos processos pedagógicos. Tais abordagens frequentemente seguem um currículo padronizado, que se mostra ineficaz diante da diversidade de estilos de aprendizagem e níveis de proficiência dos alunos (Reis, 2020). Esse descompasso entre as necessidades individuais e os métodos convencionais pode comprometer significativamente a eficácia do aprendizado. Visto isso, observa-se a necessidade de repensar as estratégias educacionais voltadas para o ensino de línguas, incorporando recursos tecnológicos, abordagens comunicativas e métodos adaptativos, capazes de oferecer experiências mais flexíveis, acessíveis e centradas no aluno (Paiva, 2019).

Nesse contexto, a utilização de tecnologias emergentes, especialmente aquelas baseadas em inteligência artificial, tem ganhado destaque como um caminho promissor para enfrentar esses obstáculos. Ferramentas que integram processamento de linguagem natural e aprendizado de máquina oferecem a possibilidade de criar ambientes de aprendizado mais flexíveis, adaptativos e escaláveis, permitindo que os alunos avancem no seu próprio ritmo, com *feedback* instantâneo e personalizado.

Este trabalho se insere nesse cenário ao propor o desenvolvimento do *SpeakUp*, um sistema que utiliza algoritmos de PLN para proporcionar uma experiência de aprendizado de línguas mais interativa e personalizada. Embora a plataforma tenha sido concebida para suportar diversos idiomas, sua versão inicial será focada exclusivamente no ensino da língua inglesa.

O *SpeakUp* foi projetado para se ajustar ao nível de conhecimento do usuário, oferecendo atividades que englobam prática conversacional, correção gramatical e tradução textual. Para viabilizar essas funcionalidades, o sistema integra as *Application Programming Interface* (API)'s do Gemini e da OpenAI, ferramentas de inteligência artificial generativa voltadas à compreensão e geração de linguagem natural. No desenvolvimento, utilizou-se a conta gratuita do Gemini e um crédito de 5 dólares pela OpenAI, o que permitiu explorar os recursos oferecidos por ambas as plataformas de forma acessível. A proposta é criar um ambiente de aprendizado que não apenas possibilite a aquisição de conhecimento linguístico, mas também incentive a autonomia do usuário no processo de aprendizagem.

O sistema visa, portanto, a construção de um ambiente responsivo, capaz de atender tanto àqueles que buscam aprender um idioma por conta própria quanto às instituições de ensino que desejam adotar novas tecnologias para complementar suas abordagens pedagógicas

tradicionais. Além disso, a solução oferece mecanismos de automação, permitindo que a experiência de aprendizagem seja contínua e sem interrupções.

A arquitetura do *SpeakUp* é baseada em tecnologias reconhecidas na indústria de software, como GoLang para o desenvolvimento do *backend*, React para o *frontend*, MongoDB para a persistência eficiente de dados e Docker para garantir a portabilidade e automação do ambiente de desenvolvimento e produção. Essa escolha tecnológica visa não só a escalabilidade e performance, mas também a criação de uma plataforma segura e com manutenção facilitada.

Considera-se, portanto, que a solução proposta tenha o potencial de apoiar o processo de aprendizagem de idiomas, ao mesmo tempo em que oferece um suporte para instituições de ensino que buscam integrar tecnologias em suas metodologias de ensino. Além disso, o *SpeakUp* visa contribuir para a democratização do aprendizado de línguas, tornando-o mais acessível e adaptável às necessidades dos usuários, independentemente de sua localização ou situação financeira.

1.1 Objetivos

Nesta seção são apresentados o objetivo geral e os objetivos específicos, delimitados para esse projeto.

1.1.1 Objetivo Geral

Esse trabalho tem como objetivo geral desenvolver uma aplicação web inteligente voltada ao ensino de idiomas, capaz de oferecer uma experiência por meio da interação com uma IA. A plataforma deverá integrar funcionalidades como conversação contextual, correções gramaticais automáticas e traduções em tempo real, com o objetivo de tornar o processo de aprendizado de uma nova língua mais interativo.

1.1.2 Objetivos específicos

Para atender ao objetivo geral, os seguintes objetivos específicos foram definidos:

- Implementar um assistente virtual capaz de interagir com usuários de acordo com seu nível linguístico.
- Desenvolver uma interface web responsiva e de fácil usabilidade.
- Integrar funcionalidades de correção gramatical e tradução automatizada para textos.
- Garantir a persistência segura e eficiente dos dados dos usuários utilizando banco de dados *Not Only SQL* (NoSQL).

- Utilizar ferramentas de containerização para facilitar a implantação e manutenção do sistema.
- Implementar testes unitários para o auxílio do desenvolvimento da aplicação.

1.2 Justificativa

O aprendizado de inglês se tornou uma habilidade fundamental no mundo contemporâneo, especialmente em contextos profissionais, acadêmicos e tecnológicos. Segundo a plataforma de ensino de idiomas Duolingo (2023), o inglês se manteve como o idioma mais estudado no Brasil em 2023, refletindo o interesse crescente da população por essa competência global. No entanto, esse interesse ainda não se traduz em domínio da língua: um relatório aponta que apenas 5% dos brasileiros falam inglês, sendo que apenas 1% possui fluência (British Council, 2019).

Essa lacuna entre interesse e proficiência é ainda mais preocupante quando se observa que o inglês é uma exigência em mais de 80% das vagas de emprego destinadas a profissionais com nível superior, segundo pesquisa realizada pela Catho (Catho, 2017). A dificuldade de acesso a cursos de qualidade e metodologias de ensino engajadoras contribui para esse cenário desfavorável.

Nesse contexto, o uso de tecnologias digitais oferece uma oportunidade promissora para democratizar o ensino de inglês. A iniciativa está alinhada com o Programa Educação Conectada, do Ministério da Educação, que incentiva a integração de tecnologias no ambiente escolar para melhorar o processo de ensino-aprendizagem (Brasil, 2024). Além disso, a proposta está aderente às metas do Plano Nacional de Educação (PNE), que enfatiza a necessidade de ampliação da oferta de educação de qualidade com uso de recursos tecnológicos (Brasil, 2014).

Diante disso, o desenvolvimento de uma aplicação interativa voltada para o ensino de inglês básico surge como uma solução alinhada às diretrizes educacionais do país, contribuindo para a superação das barreiras no aprendizado do idioma e para o desenvolvimento pessoal e profissional dos usuários. No entanto, é importante enfatizar que a solução proposta não tem como objetivo substituir o papel do professor de ensino de idiomas, dado que a interatividade humana consiste em uma prática primordial no processo de aprendizagem. Assim, espera-se que o *SpeakUp* seja utilizado como apoio ao professor e ao estudante, como meio de potencializar o processo de ensino-aprendizado.

1.3 Organização do trabalho

Este trabalho está organizado em seis capítulos, conforme descrito a seguir:

- **Capítulo 1 Introdução:** Apresenta o tema central da pesquisa, seus objetivos geral e específicos, a justificativa da escolha do tema e a metodologia utilizada. Também contextualiza o problema abordado e destaca a importância do desenvolvimento de

soluções com suporte em inteligência artificial. Por fim, descreve a estrutura organizacional do trabalho.

- **Capítulo 2 Referencial Teórico:** Aborda os principais conceitos relacionados à IA generativa, assistentes virtuais e tecnologias aplicadas ao ensino de idiomas. Explora fundamentos de processamento de linguagem natural, correção gramatical automatizada. Esse embasamento teórico sustenta as decisões metodológicas do projeto.

- **Capítulo 3 Materiais e Metodologia:** Descreve as linguagens de programação, *frameworks*, bibliotecas e ferramentas utilizadas no desenvolvimento do sistema. Também apresenta a metodologia adotada, os critérios para seleção das tecnologias, os processos de prototipação, implantação do sistema, testes e validação com usuários. Esta seção garante a reprodutibilidade do estudo.

- **Capítulo 4 Resultados:** Apresenta a arquitetura do sistema desenvolvido, bem como a implementação de suas funcionalidades principais, incluindo o assistente virtual baseado em IA. Detalha a integração entre os componentes, o fluxo de dados e a interface com o usuário. Também são exibidos exemplos práticos das interações realizadas na aplicação.

- **Capítulo 5 Conclusão:** Resume os resultados alcançados, destacando as contribuições da pesquisa para o campo da educação mediada por tecnologias inteligentes. Analisa as limitações encontradas durante o processo de desenvolvimento e propõe melhorias e possíveis desdobramentos futuros. Encerra com reflexões sobre o impacto social e educacional da solução.

2 REFERENCIAL TEÓRICO

Este capítulo tem como objetivo apresentar os fundamentos teóricos que embasam o desenvolvimento do sistema proposto. São discutidos os principais conceitos relacionados à IA, com ênfase em IA generativa, assistentes virtuais, correção gramatical automatizada e tecnologias educacionais aplicadas ao ensino de idiomas. A compreensão desses conceitos é essencial para contextualizar as escolhas metodológicas e tecnológicas adotadas no trabalho.

2.1 Inteligência Artificial

A IA é um campo interdisciplinar da ciência da computação que busca criar sistemas capazes de executar tarefas que normalmente exigiriam inteligência humana. A IA pode ser definida como "*The study of agents that receive percepts from the environment and perform actions*" (Russell; Norvig, 2016) (em tradução livre: "o estudo de agentes que recebem percepções do ambiente e realizam ações"). Essa definição destaca o comportamento inteligente como sendo o resultado da interação entre percepção e ação.

Por outro lado, uma definição mais abrangente considera a IA como um conjunto de técnicas computacionais baseadas em princípios matemáticos, estatísticos e lógicos, capazes de aprender padrões e tomar decisões com base em dados (Li; Link; Hegelheimer, 2022). Essas tarefas incluem raciocínio lógico, aprendizado, percepção sensorial e compreensão de linguagem natural.

Dentre as principais categorias existentes na área de IA, destacam-se:

- **Aprendizado Supervisionado:** nesse tipo de aprendizado, o modelo é treinado a partir de dados rotulados, aprendendo a prever saídas com base nas entradas obtidas. Tal abordagem é comum em aplicações de classificações de texto e reconhecimento de padrões.

- **Aprendizado Não Supervisionado:** nesse aprendizado, os dados não precisam estar rotulados. O modelo faz a descoberta de padrões ocultos, como agrupamentos (*clusters*) e associações. Essa área é comumente utilizada em análises exploratórias e sistemas de recomendação.

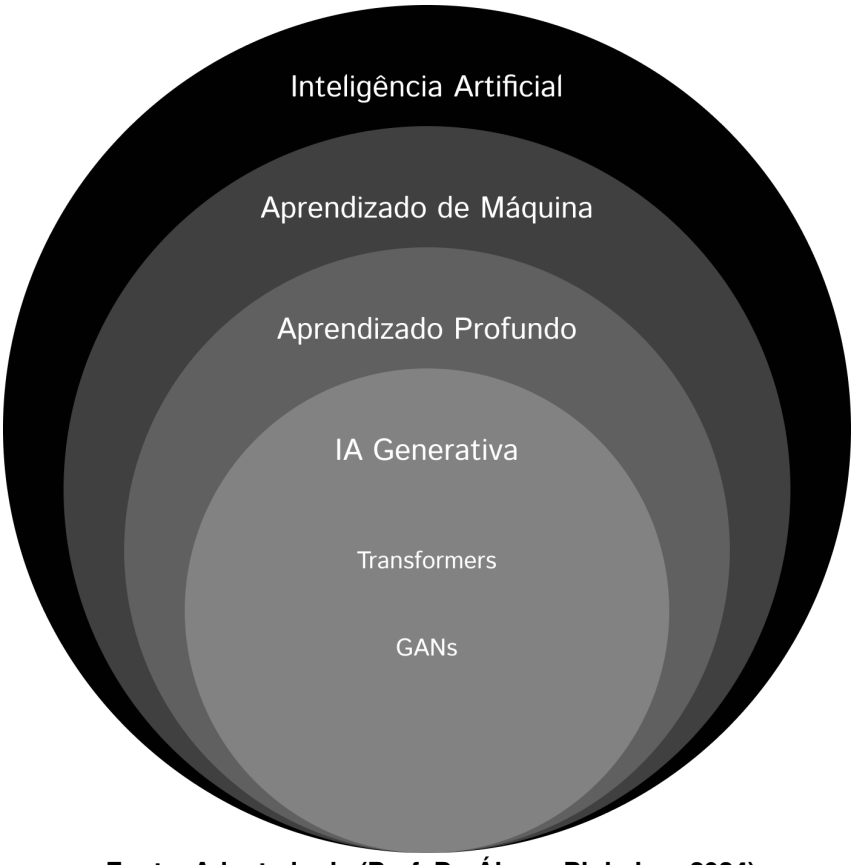
- **Aprendizado por Reforço:** nessa abordagem, um agente aprende a tomar decisões ao interagir com um ambiente, recebendo recompensas ou penalidades. Tal estratégia tem sido aplicada em jogos, robótica e sistemas adaptativos.

- **PLN:** ramo da IA que permite a interpretação, compreensão e geração de linguagem humana. Essencial em tradutores automáticos, *chatbots* e corretores gramaticais.

- **IA Generativa:** trata-se de um subcampo da IA, inserido nas camadas do Aprendizado de Máquina e, mais especificamente, do Aprendizado Profundo. Conforme ilustrado na

Figura 1, a IA Generativa se localiza como uma especialização dentro deste ecossistema hierárquico. Nela, redes neurais profundas, como os *Transformers* e as *Generative Adversarial Networks* (GAN)s (Redes Adversariais Generativas), são utilizadas para criar novos conteúdos originais, como textos, imagens, vídeos ou músicas, a partir de grandes volumes de dados de treinamento. Esses modelos são capazes de aprender padrões complexos presentes nos dados e, com base nisso, gerar amostras que simulam características reais ou plausíveis, ampliando as possibilidades de aplicação da inteligência artificial em diversas áreas.

Figura 1 – Relacionamento entre as áreas da IA



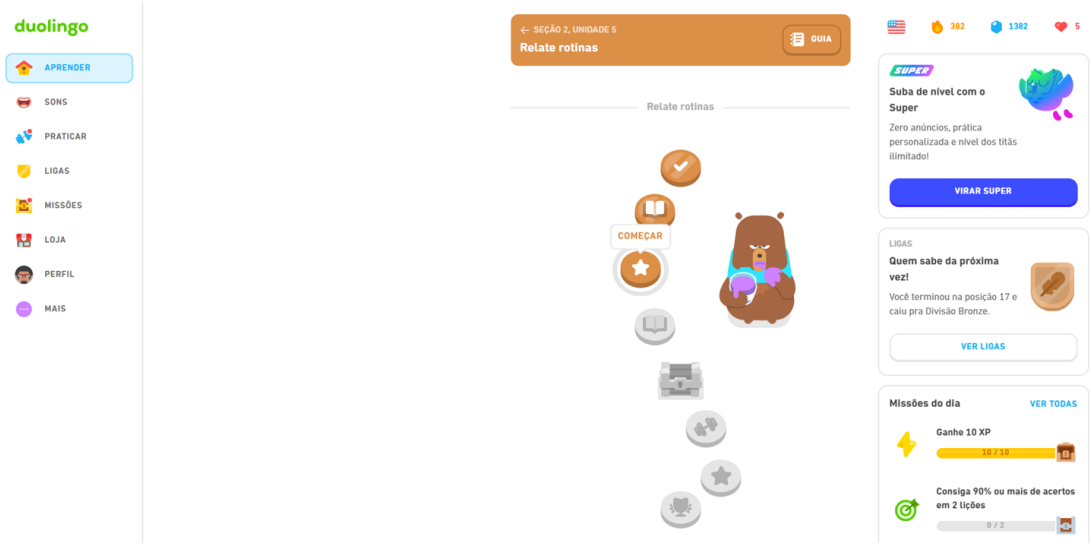
Fonte: Adaptado de (Prof. Dr. Álvaro Pinheiro , 2024).

2.2 Assistentes Virtuais no Apoio ao Aprendizado

Um assistente virtual educacional é um sistema inteligente baseado em IA, desenvolvido para interagir com estudantes por meio de linguagem natural, com o objetivo de apoiar o processo de aprendizagem. Essas ferramentas oferecem respostas em tempo real, orientações personalizadas e esclarecimento de dúvidas, atuando como suporte complementar ao ensino tradicional. Quando integrados a plataformas educacionais, os assistentes virtuais contribuem para o aumento da autonomia dos alunos, promovendo maior engajamento, personalização do conteúdo e aprendizado contínuo.

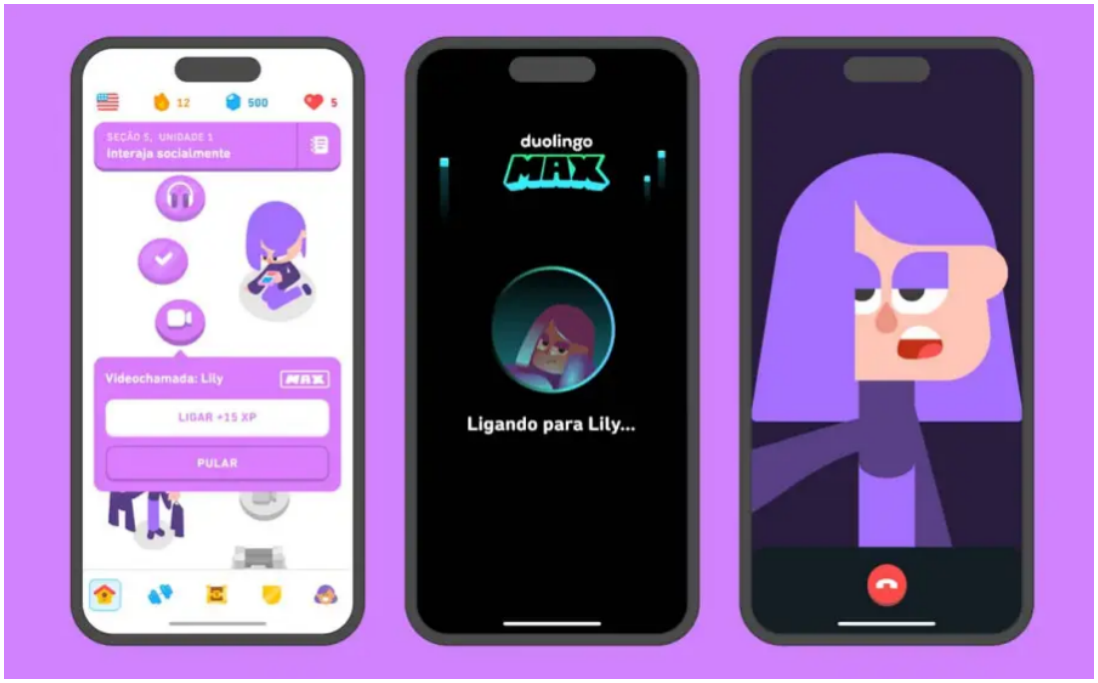
Ferramentas como o Duolingo (Figura 2), por exemplo, têm introduzido assistentes virtuais para praticar conversação e oferecer suporte personalizado (Figura 3). Esses agentes virtuais não apenas respondem perguntas, mas também analisam erros comuns, incentivam boas práticas e guiam o estudante em trilhas de aprendizagem adaptativas (USO. . . , 2023). O *SpeakUp*, sistema desenvolvido neste trabalho, insere-se nessa categoria, com o diferencial de utilizar IA generativa para realizar correções gramaticais e adaptar seu comportamento com base no histórico de interações do usuário.

Figura 2 – Tela inicial do Duolingo



Fonte: Duolingo (2025).

Figura 3 – Assistente virtual do Duolingo



Fonte: (Tecnoblog, 2024).

2.3 Correção Gramatical Automatizada

A correção gramatical automatizada é uma das aplicações mais impactantes da IA no ensino de idiomas. Ferramentas como LanguageTool, QuillBot e Grammarly utilizam técnicas de PLN para identificar e corrigir erros gramaticais, ortográficos e de estilo. Além da detecção de erros, algumas dessas plataformas oferecem explicações detalhadas, auxiliando no aprendizado (Wang *et al.*, 2024; LanguageTool, 2025).

Com o auxílio da IA, essas ferramentas são capazes de analisar o contexto semântico da frase, evitando correções errôneas que sistemas baseados em regras fixas poderiam cometer. Além disso, alguns sistemas também conseguem adaptar suas sugestões conforme o nível do usuário ou o estilo de escrita desejado (formal, acadêmico, coloquial, etc.).

2.4 Trabalhos Relacionados

Nesta seção é apresentada uma revisão de trabalhos e tecnologias relacionadas ao desenvolvimento do sistema *SpeakUp*, com o objetivo de contextualizar a proposta dentro do estado da arte e identificar lacunas que justificam sua realização. São analisados sistemas voltados ao ensino de idiomas com apoio de tecnologias digitais, bem como pesquisas que aplicam IA e técnicas de PLN, no campo educacional.

Nos últimos anos, plataformas como Duolingo e Babbel têm ganhado notoriedade por oferecerem aprendizado de línguas. Essas ferramentas utilizam estratégias de gamificação, repetição espaçada e microlições como estratégias pedagógicas. No entanto, embora populares, ainda apresentam limitações, principalmente no que diz respeito à prática de conversação espontânea, uma competência essencial para a fluência.

Estudos mais recentes apontam o potencial das tecnologias digitais na transformação do ensino. Huang *et al.* (2024) destacam que o uso adequado dessas tecnologias pode promover uma aprendizagem mais significativa, sustentável e personalizada. Eles defendem que ferramentas baseadas em inteligência artificial, quando integradas de maneira planejada ao processo educacional, podem proporcionar avanços expressivos no ensino de línguas.

O ChatGPT, desenvolvido pela OpenAI, tem sido objeto de diversos estudos acadêmicos. Wang e Zhang (2023) investigaram o uso da ferramenta em um estudo de caso voltado ao aprendizado de idiomas, observando impactos positivos na motivação dos alunos e no estímulo à produção oral. De forma complementar, Alghamdi e Alghamdi (2023) analisaram a percepção de estudantes sauditas ao utilizar o ChatGPT no ensino de inglês como língua estrangeira, concluindo que a ferramenta contribuiu para a autonomia e a confiança dos aprendizes.

Adicionalmente, Lee e Kwon (2022) exploraram a aplicação de assistentes conversacionais baseados em IA para promover a prática oral em ambientes virtuais de aprendizagem. O estudo demonstrou que tais ferramentas podem ser eficazes ao estimular interações mais naturais entre alunos e agentes conversacionais, resultando em maior engajamento e retenção do conteúdo.

Outro exemplo relevante é o trabalho de Zhou e Li (2023), que propôs a utilização de técnicas de PLN para adaptar automaticamente o nível de complexidade das perguntas e respostas durante diálogos com estudantes, personalizando o conteúdo conforme o nível de proficiência do usuário. Essa abordagem se alinha à proposta do *SpeakUp*, que busca ajustar suas interações com base no progresso individual do aprendiz.

Além disso, Chen e Sun (2021) conduziram uma pesquisa longitudinal com estudantes de língua inglesa utilizando um *feedback* educacional durante um semestre letivo. Os resultados indicaram melhora significativa na fluência oral e na confiança dos participantes, especialmente quando o *chatbot* fornecia *feedback* corretivo e sugestões contextuais.

Apesar dos avanços observados, ainda há desafios a serem considerados. Smith e Doe (2023) alertam para questões relacionadas à ética, ao controle de conteúdo e à necessidade de mediação pedagógica ao empregar modelos de linguagem como o ChatGPT no ensino. Segundo os autores, sem o devido acompanhamento docente, há riscos de desinformação, dependência tecnológica e superficialidade na aprendizagem.

Diferentemente das soluções já existentes, o sistema *SpeakUp* propõe uma abordagem integrada, que reúne prática de conversação em tempo real, *feedback* automatizado e tradução de mensagens. Ao incorporar os benefícios identificados nos trabalhos anteriores e mitigar suas limitações, esta proposta busca preencher uma lacuna no ensino de idiomas e oferecer uma solução com embasamento técnico e pedagógico.

Com base na análise dos trabalhos e ferramentas existentes, observa-se que, embora plataformas como Duolingo e Babbel ofereçam contribuições relevantes para o ensino de idiomas, ainda apresentam limitações no que se refere à personalização do conteúdo, à prática de conversação adaptativa e à correção gramatical contextualizada. Nesse sentido, a Tabela 1 resume as principais funcionalidades dessas soluções e evidencia os diferenciais do sistema *SpeakUp*, que propõe uma abordagem mais interativa e centrada no usuário.

Tabela 1 – Comparativo entre ferramentas de ensino de idiomas com o SpeakUp.

Funcionalidade	**Duolingo**	**Babbel**	**SpeakUp**
Prática de conversação	Roteirizada	Limitada	Sim (texto e adaptada ao usuário)
Correção gramatical	Parcial	Não possui	Sim
Adaptação ao nível do aluno	Sim (estática)	Sim	Sim, com base no histórico
Tradução de mensagens	Não	Não	Sim
Geração de conteúdo com IA	Não	Não	Sim (voltado ao ensino)

Fonte: Elaborado pelo autor (2025).

3 MATERIAIS E METODOLOGIA

Neste capítulo é apresenta a metodologia adotada, as tecnologias empregadas e o planejamento definido para o desenvolvimento do projeto. É também descrito o processo de implementação do sistema, detalhando as etapas seguidas ao longo de sua construção. As decisões técnicas são expostas com base nas demandas identificadas durante o processo.

3.1 Metodologia

A metodologia adotada neste trabalho pode ser caracterizada como híbrida, combinando práticas da gestão ágil com adaptações específicas ao contexto do desenvolvimento do sistema. O processo foi estruturado de forma iterativa, com entregas incrementais de funcionalidades. Essa abordagem possibilitou maior flexibilidade, permitindo a incorporação de mudanças ao longo do ciclo de desenvolvimento.

Para orientar o desenvolvimento do sistema, os requisitos foram definidos e organizados em duas categorias principais: funcionais e não funcionais. Os requisitos funcionais descrevem as funcionalidades que o sistema deve oferecer ao usuário, como operações, interações e comportamentos esperados. Já os requisitos não funcionais tratam de aspectos relacionados à qualidade do sistema, tais como desempenho, disponibilidade, segurança e usabilidade. Essa separação permitiu uma análise mais precisa das necessidades do projeto e contribuiu para um planejamento mais eficiente das entregas.

3.2 Tecnologias Utilizadas

O desenvolvimento do projeto envolveu o uso de diversas ferramentas e tecnologias, selecionadas com base em suas características, vantagens técnicas e aderência às necessidades específicas. Cada tecnologia foi escolhida não apenas por sua facilidade de uso, mas principalmente pela sua capacidade de contribuir para a qualidade, eficiência e escalabilidade do sistema desenvolvido.

A escolha das ferramentas também levou em consideração fatores como integração entre camadas, compatibilidade com diferentes ambientes e a curva de aprendizado. Ao longo do projeto, essas tecnologias permitiram otimizar tarefas, organizar o fluxo de trabalho e garantir que as entregas fossem realizadas de maneira ágil e consistente. Nesta seção, são descritas as principais ferramentas utilizadas, acompanhadas de suas definições e dos papéis que desempenharam dentro do desenvolvimento da solução proposta.

3.2.1 GoLang

Go, ou *Go Programming Language* (Donovan; Kernighan, 2015), é uma linguagem de programação desenvolvida pela Google, reconhecida por seu suporte nativo à concorrência. No projeto em questão, a linguagem foi empregada na implementação do *backend* da aplicação, sendo responsável por tratar as requisições da API, executar a lógica de negócio e gerenciar o acesso ao banco de dados.

3.2.2 React

React é uma biblioteca JavaScript desenvolvida pelo Facebook (Meta Platforms, Inc., 2024), voltada à criação de interfaces de usuário reativas e componentizadas. No projeto, foi responsável pela construção do *frontend*, permitindo a criação de telas dinâmicas, com navegação fluida e componentes reutilizáveis. A estrutura do React facilitou a separação de responsabilidades visuais e tornou possível entregar uma experiência de usuário interativa.

3.2.3 MongoDB

MongoDB é um banco de dados NoSQL que armazena dados em formato de documentos *Binary JSON* (BSON), semelhantes a objetos JSON (MongoDB, Inc., 2024). Ele foi utilizado para armazenar as informações persistentes da aplicação. Sua flexibilidade de esquema permitiu adaptações rápidas durante o desenvolvimento, além de oferecer bom desempenho em consultas, especialmente em dados com estrutura não uniforme.

3.2.4 Docker

Docker é uma plataforma de *containers* que permite empacotar e distribuir aplicações com todas as suas dependências (Docker, Inc., 2024). No projeto, foi utilizado para garantir que os ambientes de desenvolvimento e produção fossem idênticos, evitando problemas de incompatibilidade. Com Docker, foi possível levantar toda a infraestrutura necessária (*backend*, *frontend* e banco de dados) com poucos comandos, agilizando a implementação do código e simplificando a implantação (*deploy*).

3.2.5 Cloudflare

Cloudflare é uma plataforma de segurança e desempenho para aplicações web, oferecendo serviços como proteção contra ataques *Distributed Denial of Service* (DDoS), otimização de tráfego e *Domain Name System* (DNS) (Cloudflare, Inc., 2024). No projeto, utilizou-se a versão gratuita da plataforma para aumentar a segurança e melhorar o tempo de resposta da

aplicação, protegendo os *endpoints* contra acessos maliciosos e armazenando recursos estáticos. Essa abordagem foi adotada com o objetivo de garantir maior estabilidade e confiabilidade aos usuários finais, mesmo com recursos limitados.

3.2.6 Notion

Notion é uma ferramenta de produtividade que combina funcionalidades de anotações, gerenciamento de tarefas e banco de dados (Notion Labs, Inc., 2024). Foi utilizado como plataforma central de documentação do projeto, onde eram registradas reuniões, anotações técnicas e planos de trabalho. A organização das tarefas também foi feita por meio do Notion, com a criação de tabelas e listas que permitiram acompanhar o progresso de forma visual e colaborativa.

3.2.7 Figma

Figma é uma ferramenta online para *design* de interfaces e prototipagem colaborativa (Figma, Inc., 2024). No projeto, foi usada para criar os protótipos das telas da aplicação, definindo *layouts*, cores, fontes e fluxos de navegação. Por ser baseada na nuvem, essa solução permite o acompanhamento de mudanças em tempo real e a colaboração no processo de *design*.

3.2.8 Visual Studio Code

Visual Studio Code, também conhecido como VSCode, é um editor de código-fonte gratuito e multiplataforma, com suporte a diversas linguagens e extensões (Microsoft Corporation, 2024). No projeto, foi a principal *Integrated Development Environment* (IDE), oferecendo recursos como autocompletar, depuração, terminal integrado e controle de versão com Git. A variedade de extensões disponíveis também ajudou a personalizar o ambiente de desenvolvimento de acordo com as necessidades do projeto.

3.2.9 Postman

Postman é uma plataforma de desenvolvimento de APIs que permite testar, documentar e automatizar requisições *HyperText Transfer Protocol* (HTTP) (Postman, Inc., 2024). No projeto, foi utilizado para testar as rotas da API desenvolvida em Go, simulando diferentes tipos de requisições e verificando se as respostas estavam corretas. Também foi útil para automatizar testes de *endpoints* e documentar o comportamento das rotas de forma clara, o que facilitou a integração entre *backend* e *frontend*.

3.2.10 Git

Git é um sistema de controle de versão distribuído amplamente utilizado para rastrear mudanças no código-fonte durante o desenvolvimento de software (The Git Development Community, 2024). No projeto, o Git foi essencial para o versionamento do código, permitindo o controle de alterações, o trabalho em diferentes ramificações (*branches*) e o registro detalhado do histórico de desenvolvimento.

3.2.11 GitHub

GitHub é uma plataforma baseada na web que hospeda repositórios Git e facilita a colaboração entre desenvolvedores (GitHub, Inc., 2024). No contexto do projeto, foi utilizado como repositório central de código, permitindo a integração eficiente.

3.2.12 Astah

Astah é uma ferramenta de modelagem visual utilizada para criação de diagramas *Unified Modeling Language* (UML) e outros tipos de representações gráficas de sistemas (Change Vision, Inc., 2024). No projeto, foi empregada para elaborar diagramas de casos de uso, classes, contribuindo para o entendimento e documentação da arquitetura e do comportamento da aplicação.

3.3 Planejamento do Projeto

Antes de iniciar a implementação do sistema, foi realizada uma etapa fundamental de levantamento e especificação dos requisitos, com o objetivo de definir claramente o escopo funcional e não funcional da aplicação. Esta seção descreve detalhadamente os Requisitos Funcionais e Não Funcionais, que nortearam o desenvolvimento do sistema.

Os requisitos funcionais abrangem as funcionalidades esperadas pela aplicação, enquanto os não funcionais estabelecem critérios de desempenho, segurança, disponibilidade e usabilidade. A definição precisa desses requisitos serviu como base para a modelagem dos dados, estruturação da arquitetura do sistema, elaboração dos diagramas UML e construção dos protótipos da interface, garantindo um alinhamento entre as necessidades dos usuários e as soluções técnicas adotadas.

3.3.1 Requisitos Funcionais

A seguir são apresentados os requisitos funcionais do *SpeakUp*:

- **[RF01]** - O sistema deve possuir uma interface de chat onde o usuário envia mensagens e recebe respostas da IA.
- **[RF02]** - O sistema deve integrar com ao menos uma API de IA generativa.
- **[RF03]** - O sistema deve armazenar o histórico de conversas de cada usuário em um banco de dados.
- **[RF04]** - O sistema deve recuperar o histórico de conversas ao enviar uma nova mensagem para a API de IA.
- **[RF05]** - O sistema deve implementar técnicas para lidar com o limite de *tokens* da API, como resumir o histórico ou enviar apenas as últimas mensagens.
- **[RF06]** - O sistema deve permitir cadastro de usuários com e-mail e senha.
- **[RF07]** - O sistema deve permitir login com as credenciais cadastradas.
- **[RF08]** - O sistema pode sugerir tópicos ou perguntas para aprendizado de idiomas diretamente no chat.
- **[RF09]** - O sistema pode permitir que a IA responda com exemplos e explicações gramaticais básicas.
- **[RF10]** - O sistema pode permitir que o usuário solicite à IA a correção de frases.

3.3.2 Requisitos Não Funcionais

A seguir são apresentados os requisitos não funcionais do *SpeakUp*:

- **[RNF01]** - O sistema deve estar disponível continuamente (24/7).
- **[RNF02]** - O *backend* deve ser escalável para suportar múltiplos usuários simultaneamente.
- **[RNF03]** - A interface deve ser responsiva e adaptada para dispositivos móveis.
- **[RNF04]** – Todas as comunicações entre cliente e servidor devem ser criptografadas utilizando o protocolo *HyperText Transfer Protocol Secure* (HTTPS).
- **[RNF05]** – O sistema deve ser compatível com os navegadores mais utilizados (Chrome, Firefox, Safari e Edge).
- **[RNF06]** – A arquitetura do sistema deve permitir fácil manutenção e atualização de componentes individuais.

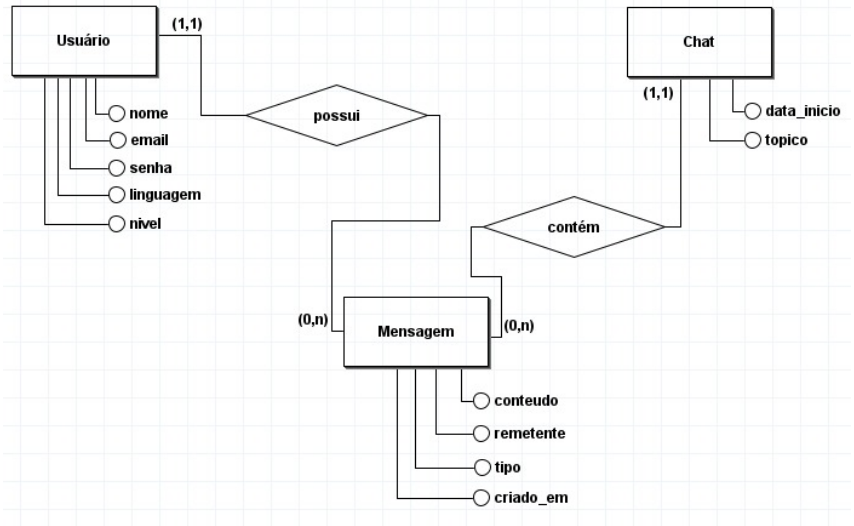
3.3.3 Modelagem de Dados

A modelagem de dados do sistema foi representada por meio do Modelo Entidade-Relacionamento (MER) (Modelo Entidade-Relacionamento), apresentado na Figura 4. Esse modelo ilustra as principais entidades do sistema — Usuário, Chat e Mensagem — bem como os relacionamentos entre elas. Cada entidade possui um conjunto de atributos relevantes: por exemplo, o Usuário contém nome, email, senha, idioma e nível de proficiência, sendo que este último é informado pelo próprio usuário através de uma autodeclaração no momento do cadastro; o Chat armazena a data de início e o tópico da conversa; e a Mensagem inclui o conteúdo, o remetente, o tipo e a data de criação.

O relacionamento “possui” indica que cada usuário pode estar associado a várias mensagens, enquanto o relacionamento “contém” mostra que cada chat reúne diversas mensagens. As cardinalidades definem com precisão as regras de participação entre as entidades, garantindo a integridade dos dados.

As regras de normalização foram aplicadas para eliminar redundâncias e assegurar a consistência das informações. Com base nesse modelo conceitual, foi gerado o modelo relacional, que serviu de referência para a implementação do banco de dados físico por meio de um Sistema de Gerenciamento de Banco de Dados (SGBD). Essa etapa foi essencial para garantir a eficiência, integridade e organização do armazenamento de dados no sistema.

Figura 4 – Modelo Entidade-Relacionamento do Banco de Dados



Fonte: Elaborado pelo autor (2025).

3.3.4 Modelagem UML

Com o objetivo de representar os aspectos funcionais e estruturais do sistema, foram elaborados diagramas utilizando a linguagem UML, conforme descrito a seguir.

O diagrama de casos de uso (Figura 5) ilustra graficamente as interações entre os atores, neste caso, o Usuário e o Sistema, e os principais casos de uso, ou seja, as funcionalidades

oferecidas pela aplicação *SpeakUp*. Esse tipo de diagrama é fundamental para o levantamento e validação dos requisitos funcionais, fornecendo uma visão clara e objetiva do que o sistema deve realizar.

Entre os casos de uso associados ao **Usuário**, destacam-se:

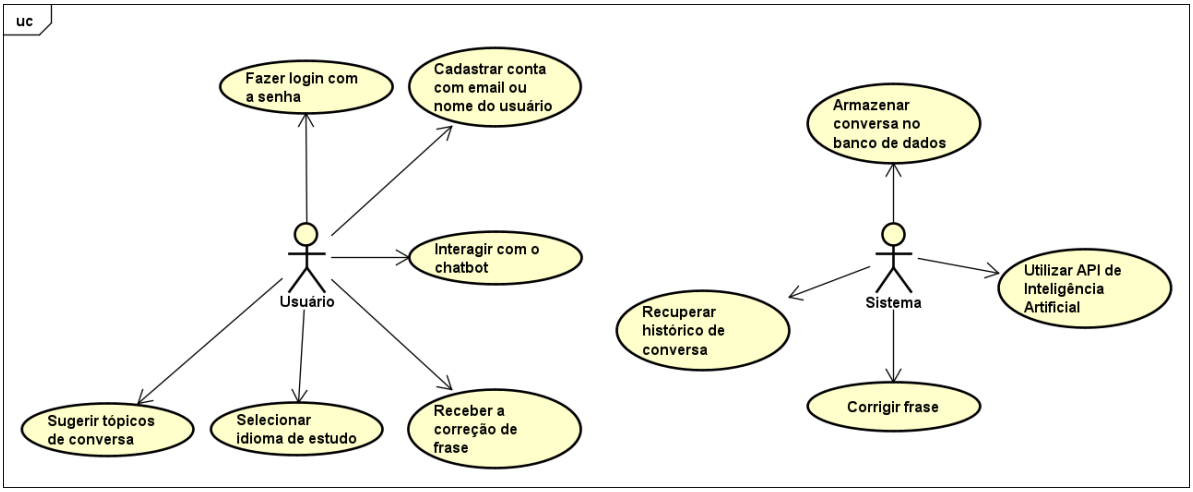
- **Fazer login com a senha**: permite ao usuário acessar sua conta por meio de autenticação com email ou nome de usuário e senha.
- **Cadastrar conta com email ou nome do usuário**: possibilita o registro de novos usuários na plataforma, incluindo informações básicas como nome, email e idioma de interesse.
- **Interagir com o chatbot**: representa a comunicação direta com o assistente virtual, por onde ocorre a troca de mensagens.
- **Sugerir tópicos de conversa**: o sistema propõe temas para auxiliar o usuário a manter o diálogo fluido e contextualizado.
- **Selecionar idioma de estudo**: o usuário escolhe o idioma que deseja praticar, influenciando toda a interação com o chatbot.
- **Receber a correção de frase**: após enviar mensagens, o usuário recebe sugestões de correção gramatical, com foco no aprendizado contínuo.

Por sua vez, os casos de uso associados ao **Sistema** incluem:

- **Armazenar conversa no banco de dados**: toda interação é registrada, permitindo manter o histórico de uso e aprendizado do usuário.
- **Recuperar histórico de conversa**: o sistema disponibiliza conversas anteriores para revisão e acompanhamento do progresso.
- **Utilizar API de Inteligência Artificial**: o sistema integra-se a uma API externa para gerar respostas inteligentes, corrigir frases e manter um diálogo natural.
- **Corrigir frase**: funcionalidade específica que processa a entrada do usuário e retorna a versão gramaticalmente correta, com base no idioma selecionado.

Essas funcionalidades, em conjunto, definem o escopo do sistema e reforçam seu propósito educacional voltado ao aprimoramento da fluência em línguas estrangeiras por meio da interação com inteligência artificial.

Figura 5 – Diagrama de Casos de Uso



Fonte: Elaborado pelo autor (2025).

O diagrama de classes descreve a estrutura estática do sistema, modelando as classes envolvidas, seus atributos, métodos e relacionamentos (Figura 6). Esse diagrama é fundamental para o desenvolvimento orientado a objetos, servindo como base para a codificação, reutilização de código e organização lógica do sistema.

No contexto do projeto *SpeakUp*, foram definidas três classes principais: **Usuario**, **Chat** e **Mensagem**, cada uma com atributos e métodos específicos que refletem suas responsabilidades dentro da aplicação.

- **Classe Usuario**: representa os usuários do sistema. Possui atributos como `id`, `nome`, `email`, `senha`, `linguagem` e `nivel`, que armazenam as informações essenciais de cada usuário. Os métodos associados incluem:
  - `+CriarUsuario(): void` — cria um novo usuário no sistema;
  - `+ResgatarUsuarioPorId(): void` — retorna um usuário a partir do seu identificador;
  - `+AtualizarUsuario(): void` — permite modificar os dados de um usuário;
  - `+ApagarUsuario(): void` — remove o usuário do sistema;
  - `+Login(): void` — autentica o usuário com base nas credenciais fornecidas.

  O relacionamento entre **Usuario** e **Chat** é de um para muitos (1:N), indicando que cada usuário pode possuir vários chats associados. Essa associação é representada pelo atributo `usuario_id` na classe **Chat**, que referencia o usuário ao qual o chat pertence.

- **Classe Chat**: modela as sessões de conversa entre o usuário e o chatbot. Seus atributos incluem `id`, `usuario_id`, `data_inicio` e `topico`. Os métodos implementados são:
  - `+CriarChat(): void` — inicia um novo chat;
  - `+ResgatarChatPorId(): void` — localiza um chat específico por seu identificador;
  - `+ResgatarChats(): void` — retorna todos os chats disponíveis;
  - `+AtualizarChat(): void` — altera as informações do chat;
  - `+ApagarChat(): void` — exclui um chat;
  - `+ResgatarChatsPorIdDoUsuario(): void` — retorna os chats vinculados a um determinado usuário.

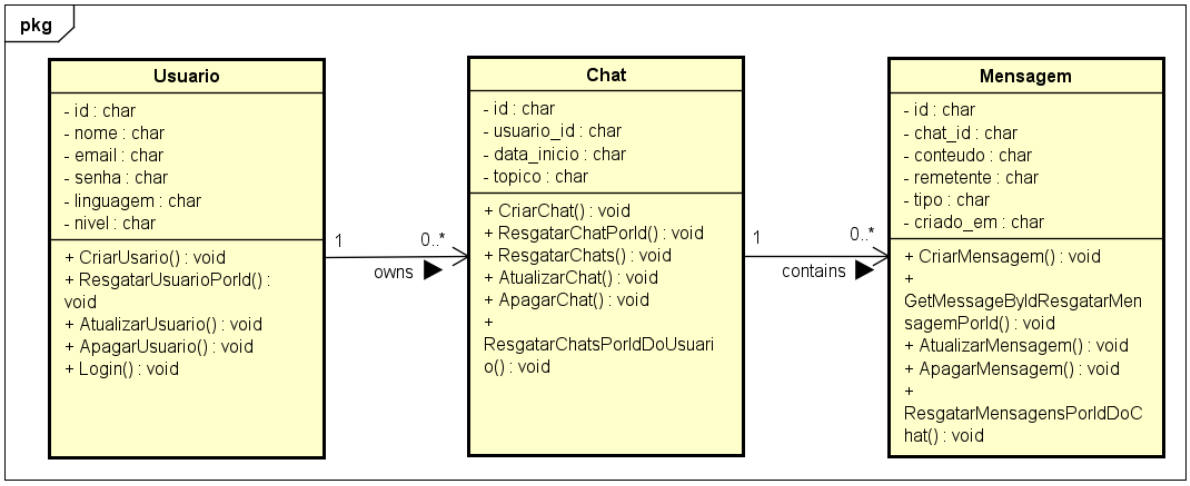
  O relacionamento entre **Usuario** e **Chat** é de um para muitos (1:N), representando que cada usuário pode possuir múltiplos chats.

- **Classe Mensagem**: representa cada mensagem trocada dentro de um chat. Os atributos incluem `id`, `chat_id`, `conteudo`, `remetente`, `tipo` e `criado_em`, que definem o conteúdo, origem e características da mensagem. Os métodos principais são:
  - `+CriarMensagem(): void` — insere uma nova mensagem no chat;
  - `+GetMessageByIdResgatarMensagemPorId(): void` — recupera uma mensagem específica;
  - `+AtualizarMensagem(): void` — permite editar uma mensagem existente;
  - `+ApagarMensagem(): void` — remove uma mensagem;
  - `+ResgatarMensagensPorIdDoChat(): void` — retorna todas as mensagens de um determinado chat.

  A associação entre **Chat** e **Mensagem** também segue uma multiplicidade de um para muitos (1:N), indicando que um chat pode conter várias mensagens.

A estrutura definida neste diagrama orienta a lógica de implementação do sistema, garantindo que as responsabilidades estejam bem distribuídas entre as classes e favorecendo a manutenção, escalabilidade e legibilidade do código.

Figura 6 – Diagrama de Classes



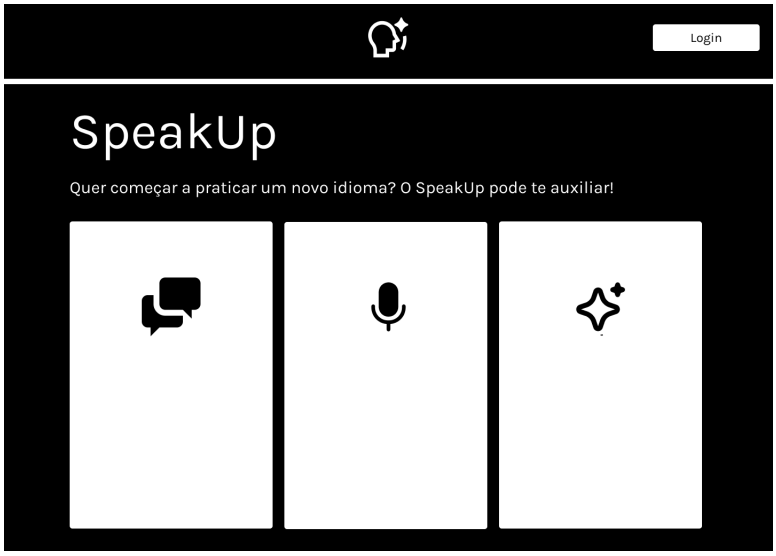
Fonte: Elaborado pelo autor (2025).

3.3.5 Prototipação da Interface

Antes da implementação da interface gráfica do *SpeakUp*, foi realizada uma fase de prototipação de alta fidelidade utilizando a ferramenta Figma. Essa etapa teve como objetivo validar com usuários-alvo aspectos fundamentais da experiência de uso, como o fluxo de navegação, a organização dos elementos visuais e a coerência da identidade visual da aplicação. Foram elaborados modelos interativos das principais telas (Index, Login e Chat), permitindo simulações de uso e coleta de *feedback*.

A tela Index, apresentada na Figura 7, exibe a página inicial com elementos introdutórios e chamadas à ação em três *cards*. O primeiro refere-se à funcionalidade do usuário ter a opção de conversar via texto com a IA. Já o segundo *card* mostra a apresentação da funcionalidade de interação com a IA via áudio, que ainda não está ativada. Por fim, o terceiro *card* refere-se a uma nota explicativa dizendo sobre os modelos de IA, como gpt4o e gemini2.5, que são utilizados na aplicação.

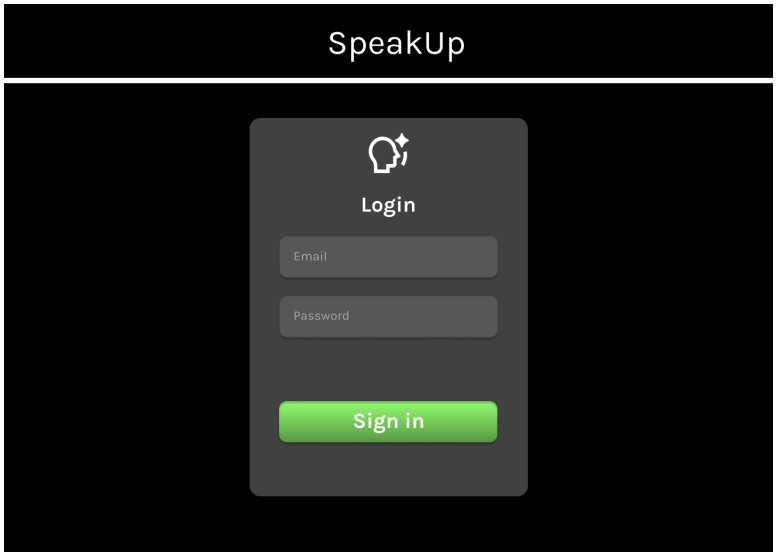
Figura 7 – Protótipo da Interface da Tela Index no Figma.



Fonte: Elaborado pelo autor (2025).

A tela Login, ilustrada na Figura 8, foi projetada para garantir um acesso simples e seguro ao sistema solicitando o email e senha do usuário.

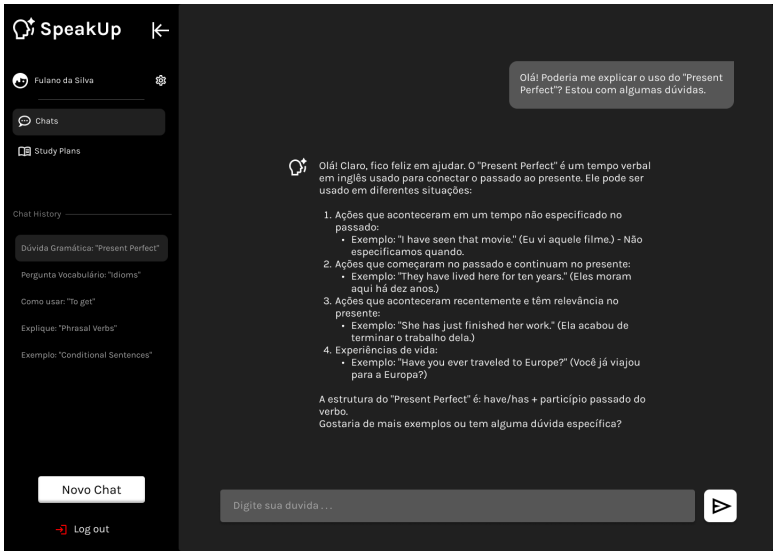
Figura 8 – Protótipo da Interface da Tela Login no Figma.



Fonte: Elaborado pelo autor (2025).

Por fim, a tela Chat, mostrada na Figura 9, representa o ambiente principal de interação, com foco na usabilidade e clareza na troca de mensagens. Essa tela foi designada para a conversa entre o usuário e a IA, tendo retornos da mesma para a correção e tradução de texto, mostrando, também, as conversas que o usuário teve anteriormente.

Figura 9 – Protótipo da Interface da Tela Chat no Figma.



Fonte: Elaborado pelo autor (2025).

3.3.6 Arquitetura do Sistema

A arquitetura do sistema adotada foi uma abordagem híbrida, baseada no modelo em três camadas (*3-tier architecture*) (Bass; Clements; Kazman, 2012), aliado ao paradigma de microsserviços (Newman, 2015). Essa combinação permitiu aproveitar as vantagens de cada estratégia, promovendo maior flexibilidade, escalabilidade e isolamento das funcionalidades. A estrutura em camadas organiza o sistema em níveis bem definidos, enquanto os microsserviços possibilitam a divisão do sistema em pequenos serviços autônomos. A seguir, serão apresentados detalhes de cada camada da arquitetura, conforme ilustrado na Figura 10.

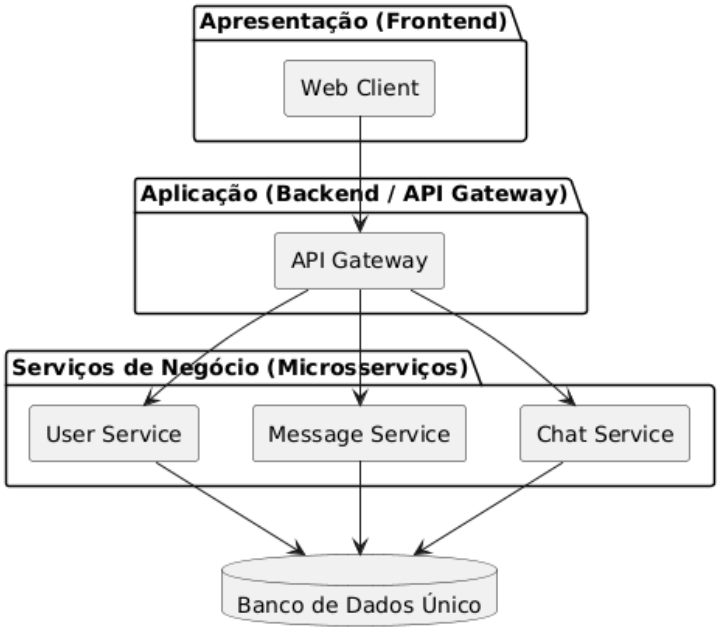
- **Camada de apresentação:** Representada no diagrama pelo bloco “Apresentação (Frontend)”, esta camada é responsável pela interface com o usuário, e foi desenvolvida utilizando a biblioteca React. Ela compreende o “Web Client”, que oferece uma aplicação web interativa, intuitiva e responsiva. Essa camada permite fácil acesso às funcionalidades do sistema e oferece uma boa experiência de uso em diferentes dispositivos, como *desktops* e *smartphones*.

- **Camada de aplicação:** Representada pelo bloco “Aplicação (Backend / API Gateway)” no diagrama, é onde está localizado o componente “API Gateway”. Essa camada atua como intermediária entre a apresentação e os serviços de negócio. Foi implementada com a linguagem Go, seguindo o padrão de microsserviços, com responsabilidades bem definidas, o que garante maior organização, isolamento de funcionalidades e facilidade de escalabilidade. Também é nesta camada que se integra os modelos de linguagem da OpenAI por meio da API, além do uso do Gemini para funcionalidades de IA.

- **Serviços de negócio (microsserviços):** Localizados entre a camada de aplicação e a de dados, estão os serviços autônomos identificados como "*User Service*", "*Message Service*" e "*Chat Service*" no diagrama. Cada um desses microsserviços é responsável por um domínio específico do sistema, garantindo coesão, independência e escalabilidade no desenvolvimento e manutenção da aplicação.
- **Camada de dados:** Representada no diagrama pelo bloco "Banco de Dados Único", essa camada é responsável pelo armazenamento persistente das informações do sistema. Utiliza o MongoDB, uma solução NoSQL que oferece flexibilidade na estrutura dos dados e excelente desempenho em consultas. Essa camada assegura integridade, consistência e segurança, servindo de base para todas as operações dos microsserviços.

Essa arquitetura promoveu a escalabilidade e a modularidade do sistema, permitindo o desenvolvimento e a implantação independente de cada serviço, além de facilitar futuras manutenções e expansões. Com a separação clara das responsabilidades, o sistema se tornou mais robusto e preparado para evoluir conforme as demandas dos usuários e do mercado.

Figura 10 – Arquitetura Híbrida do Sistema - *3-tier* com Microsserviços



Fonte: Elaborado pelo autor (2025).

3.3.7 Estrutura de Pastas

A estrutura de pastas do projeto foi planejada para garantir modularidade, escalabilidade e facilidade de manutenção, proporcionando uma separação clara entre as camadas de *backend*, *frontend* e infraestrutura.

- **root (/)**: Diretório raiz do projeto. Contém os principais módulos do sistema, incluindo o *backend*, o *frontend* e os arquivos de configuração para *containerização*.
- **backend**/: Responsável por toda a lógica de negócio e pela API, implementada em Go. Sua estrutura interna segue os princípios de organização modular de projetos Go:
  - **cmd**/: Arquivos de inicialização e execução da aplicação.
  - **docs**/: Documentação técnica relacionada ao *backend*.
  - **pkg**/: Pacote principal da aplicação, subdividido para melhor organização das responsabilidades:
    - **api**/: Definições de interfaces e contratos da API.
    - **adapters**/: Módulos de integração com serviços externos ou outras camadas do sistema.
    - **config**/: Arquivos e funções de configuração da aplicação.
    - **handlers**/: Manipuladores responsáveis pelo tratamento das requisições HTTP.
    - **middlewares**/: Middlewares aplicados ao ciclo de requisição-resposta.
    - **models**/: Estruturas de dados e definições das entidades do sistema.
    - **prompts**/: Conjunto de *prompts* utilizados nas interações com a IA.
    - **routes**/: Arquivos de definição das rotas da aplicação.
  - **public**/: Recursos estáticos expostos externamente, como arquivos públicos.
  - **main.go**: Ponto de entrada da aplicação *backend*.
- **frontend**/: Contém os arquivos relacionados ao desenvolvimento da interface web, além da configuração específica para o ambiente de containerização. Estruturado da seguinte forma:
  - **web**/: Diretório principal do *frontend*, desenvolvido com a biblioteca **React**. Inclui:
    - **src**/: Código-fonte da aplicação web, subdividido em:
      - **components**/: Componentes reutilizáveis da interface.
      - **pages**/: Páginas que compõem a navegação da aplicação, alinhadas com as rotas definidas no React.
      - **utils**/: Funções utilitárias de apoio.
      - **App.js**: Arquivo principal de inicialização da aplicação React.
    - **package.json**: Gerencia as dependências e os scripts de execução da aplicação.
- **docker-compose.yml**: Arquivo de orquestração responsável por construir e executar simultaneamente os serviços de *backend* e *frontend* em *containers* Docker.

Essa organização estrutural permite uma separação clara de responsabilidades, promovendo uma arquitetura que facilita a manutenção, a colaboração entre equipes e futuras expansões do sistema.

3.4 Implementação do Projeto

A implementação do sistema foi realizada com a linguagem Go no *backend*, utilizando MongoDB para persistência de dados. Foram criadas estruturas para representar os principais elementos da aplicação, como usuários, chats e mensagens. O complemento da explicação do processo de implementação pode ser observado nas listagens disponíveis no Anexo A.

3.4.1 Modelos

A estrutura que representa o chat no sistema é definida pela Listagem 1. Nela, cada instância da estrutura Chat armazena informações essenciais sobre uma conversa, como o identificador único (`ID`), o usuário associado (`UserID`), o horário de início (`StartTime`) e o tópico da conversa (`Topic`). Essa estrutura é a base para o histórico de interações entre o usuário e o assistente virtual.

As mensagens trocadas durante o chat são armazenadas conforme a estrutura Message, apresentada na Listagem 2. Cada mensagem contém o conteúdo textual (`Content`), o remetente (`Sender`), o tipo da mensagem (`Type`, que pode representar por exemplo se é uma correção, tradução, ou texto livre), e a data de criação (`CreatedAt`). Essa estrutura foi fundamental para garantir a organização e o fluxo correto das interações no banco de dados.

Já os usuários da plataforma são representados pela estrutura User, conforme a Listagem 3. Esta estrutura contempla os campos: nome (`Name`), email (`Email`), senha (`Password`), idioma (`Language`) e nível atual de conhecimento (`Level`). Esses dados permitem designar um plano de estudos, conforme o perfil do usuário.

A modelagem dessas estruturas permitiu a integração com o MongoDB, utilizando anotações BSON para mapear os campos corretamente no banco de dados. Esse modelo de persistência foi utilizado para manter a escalabilidade do sistema e garantir a integridade dos dados, especialmente nas interações em tempo real com a IA.

Além disso, os dados foram utilizados em *endpoints* da API *Representational State Transfer* (REST) criada no *backend*, permitindo que o *frontend* — desenvolvido com React — consumisse as informações dinamicamente e atualizasse a interface conforme a interação do usuário com a plataforma.

3.4.2 Funcionalidades do Usuário

Para manipulação dos usuários no sistema, foram implementadas as seguintes funções na API:

- **Login** (Listagem 4): Permite que um usuário já cadastrado acesse o sistema informando seu email e senha. Caso as credenciais estejam corretas, o acesso é liberado.
- **CreateUser** (Listagem 5): Cria um novo usuário, cadastrando as informações dele, como: nome, email, senha, linguagem que deseja aprender e nível atual de conhecimento nessa linguagem.
- **GetUsers** (Listagem 6): Recupera todos os usuários do sistema, mostrando todos os seus dados.
- **UpdateUser** (Listagem 7): Atualiza um usuário que existe no sistema.
- **DeleteUser** (Listagem 8): Permite que o usuário possa ser excluído do sistema.

3.4.3 Funcionalidades do Chat

Para manipulação dos chats no sistema, foram implementadas as seguintes funções na API:

- **CreateChat** (Listagem 9): Inicia um novo chat, vincula-o ao usuário autenticado, define o horário de início e persiste os dados no banco de dados MongoDB.
- **GetChatById** (Listagem 10): Recupera um chat específico a partir de seu identificador único, permitindo o carregamento de conversas anteriores.
- **GetChats** (Listagem 11): Retorna todos os chats armazenados no sistema, fornecendo uma lista completa de conversas.
- **UpdateChat** (Listagem 12): Permite a atualização de informações de um chat, como o tópico da conversa, oferecendo flexibilidade na edição.
- **DeleteChat** (Listagem 13): Exclui um chat com base no seu identificador, possibilitando o gerenciamento e remoção de conversas indesejadas.
- **GetChatsByUserId** (Listagem 14): Lista apenas os chats vinculados ao usuário autenticado, permitindo uma visualização personalizada.

3.4.4 Funcionalidades da Mensagem

Para manipulação das mensagens no sistema, foram implementadas as seguintes funções na API:

- **CreateMessage** (Listagem 15): Função que permite a criação da mensagem quando o usuário for iniciar a conversa e quando a IA for respondê-lo.
- **GetMessageById** (Listagem 16): Recupera uma mensagem específica a partir de seu identificador único, permitindo o carregamento de conversas anteriores.
- **GetMessages** (Listagem 17): Retorna todos as mensagens armazenadas no banco de dados.
- **UpdateMessage** (Listagem 18): Permite a atualização de informações de uma mensagem.
- **DeleteMessage** (Listagem 19): Exclui um chat com base no seu identificador, possibilitando o gerenciamento e remoção de conversas indesejadas.
- **GetMessagesByChatId** (Listagem 20): Lista apenas as mensagens vinculados ao chat escolhido, permitindo uma visualização personalizada.

Esses *endpoints* tornam a API REST robusta e alinhada com os princípios do *Create, Read, Update, Delete* (CRUD), oferecendo ao *frontend* total controle sobre o histórico de interações do usuário com o sistema. A comunicação entre *frontend* e *backend* se dá por meio de requisições HTTP, utilizando o *framework* Gin, que facilita a criação de aplicações web e APIs no *backend* em conjunto com o MongoDB, o que proporciona agilidade e escalabilidade na manipulação de dados.

3.4.5 Funcionalidades de Inteligência Artificial

Além das operações CRUD descritas anteriormente, o sistema integra funcionalidades de IA para enriquecer a experiência de aprendizado de idiomas. Essas funcionalidades, implementadas no pacote `handlers`, utilizam a API do Gemini por meio de um conector (`connectors.NewGeminiConnector`) para processar linguagem natural. Todas as funções seguem o padrão REST, utilizando o *framework* Gin, e são protegidas por autenticação via `ApiKeyAuth`, garantindo segurança nas interações.

- **GenerateResponseDialog** (Listagem 21): Gera respostas contextuais para diálogos, considerando o histórico de mensagens do chat e a mensagem atual do usuário. Essa função utiliza *prompts* armazenados em arquivos externos (`promptDialog.txt`) e o histórico do chat recuperado do MongoDB para criar respostas personalizadas no idioma do usuário.

- **GenerateResponseCorrection** (Listagem 22): Corrige erros gramaticais em textos fornecidos pelo usuário, auxiliando no aprendizado de idiomas. O texto é processado com base em um *prompt* específico (`promptCorrection.txt`).
- **GenerateResponseTranslate** (Listagem 23): Traduz textos para o idioma especificado, utilizando um *prompt* de tradução (`promptTranslate.txt`), permitindo que os usuários aprendam vocabulário e expressões em diferentes idiomas.
- **GenerateResponseTopic** (Listagem 24): Gera tópicos de duas palavras com base em textos fornecidos, facilitando a categorização de conversas.

Essas funções utilizam *prompts* pré-definidos armazenados em arquivos de texto (`prompts/`), que são combinados com dados do usuário (como idioma e nível) e histórico de interações para gerar respostas personalizadas. A integração com o MongoDB garante que os dados processados pela IA sejam persistidos adequadamente, enquanto a comunicação com o *frontend* ocorre por meio de requisições HTTP, permitindo uma experiência dinâmica e interativa.

3.4.6 Funcionalidades dos Conectores

O sistema também conta com a implementação de conectores, cuja função é permitir a integração com diferentes APIs de IA. Essa abordagem garante flexibilidade e escalabilidade, possibilitando a adição de novos modelos no futuro de forma padronizada. A seguir, é apresentada a principal função e interface do conector implementada:

- **GenerateResponse** (Listagem 25 e Listagem 27): Gera respostas contextuais para diálogos, considerando o histórico de mensagens do chat e a mensagem atual do usuário. Essa função utiliza *prompts* armazenados em arquivos externos (como `promptDialog.txt`) e o histórico do chat recuperado do MongoDB para criar respostas personalizadas no idioma do usuário.

3.4.7 Conteinerização e Orquestração (Docker)

A conteinerização e orquestração desempenham um papel fundamental no desenvolvimento, implantação e escalabilidade do projeto *SpeakUp*. Utilizando o Docker, a aplicação é empacotada em contêineres isolados, garantindo consistência em diferentes ambientes, desde o desenvolvimento até a produção. Isso simplifica a gestão de dependências e otimiza a utilização de recursos, além de facilitar a portabilidade entre plataformas.

O arquivo docker-compose.yml na raiz do projeto demonstra como os diferentes serviços do *SpeakUp* (*backend* e *frontend* web) são definidos e orquestrados, permitindo que a aplicação seja executada com um único comando.

Para o projeto *SpeakUp*, dois `Dockerfiles` são utilizados para construir as imagens dos serviços de *backend* e *frontend* web, conforme definido no arquivo `docker-compose.yml`. Cada serviço possui seu próprio ambiente configurado, com dependências específicas e instruções para *build* e execução. O uso do Docker facilita a padronização do ambiente de desenvolvimento e implantação, evitando problemas de compatibilidade. Além disso, o `docker-compose` permite orquestrar os contêineres de forma simples, garantindo que os serviços se comuniquem corretamente. Essa abordagem torna o processo de *deploy* mais confiável e replicável.

3.4.8 Dockerfile do Backend

Este Dockerfile é responsável por criar a imagem Docker para o serviço de *backend* em GoLang. Geralmente, ele incluiria passos para:

- Definir a imagem base para o ambiente Go.
- Copiar o código fonte da aplicação para o contêiner.
- Instalar quaisquer dependências necessárias.
- Compilar a aplicação Go.
- Expor a porta em que a aplicação será executada.
- Definir o comando de inicialização da aplicação.

3.4.9 Dockerfile do Frontend

Este Dockerfile é responsável por criar a imagem Docker para o serviço de *frontend* web em ReactJS. Os passos típicos incluem:

- Definir a imagem base para o ambiente Node.js.
- Copiar o código fonte do *frontend* para o contêiner.
- Instalar as dependências do Node.js (usando npm ou yarn).
- Configurar o ambiente de desenvolvimento, se aplicável, para habilitar funcionalidades como *hot-reloading*.
- Expor a porta em que a aplicação React será servida.
- Definir o comando para iniciar o servidor de desenvolvimento do React.

3.4.10 Testes Unitários

A garantia da qualidade do software é um pilar fundamental no desenvolvimento do *SpeakUp*, e os testes unitários desempenham um papel crucial nesse processo. Eles permitem verificar o correto funcionamento de pequenas partes isoladas do código, como funções e métodos, assegurando que cada componente se comporte como esperado antes de ser integrado a outras partes da aplicação.

No *backend*, os testes unitários da API foram desenvolvidos utilizando as ferramentas de teste nativas da linguagem Go, como o pacote `testing` e a biblioteca `testify`, que auxiliam na organização e nas asserções dos testes. Essa abordagem permite validar o comportamento das funções de forma isolada, garantindo maior confiabilidade no código. Durante o processo, foram testados os principais fluxos da aplicação, incluindo manipulação de dados, autenticação e respostas HTTP. A utilização de *mocks* também foi essencial para simular dependências externas e cenários específicos. Com isso, foi possível assegurar a qualidade da API e facilitar futuras manutenções e refatorações.

A estrutura de testes em Go é simples e eficiente, com os arquivos de teste localizados no mesmo pacote do código que está sendo testado. Isso favorece a organização do projeto e permite fácil acesso às funções internas. A execução dos testes é feita de forma prática com o comando `go test`, que identifica e executa automaticamente todos os testes do pacote. Essa abordagem contribui para um desenvolvimento mais confiável e com *feedback* rápido.

Além disso, foram definidos protocolos para a verificação da cobertura dos testes unitários, utilizando o comando `go test -cover`. Essa métrica permite avaliar a proporção do código que está sendo efetivamente testada, servindo como um importante indicador da robustez da aplicação. A análise de cobertura foi incorporada ao fluxo de desenvolvimento como um critério de qualidade, incentivando a escrita de testes mais abrangentes e prevenindo lacunas no código não testado.

3.5 Implantação do Projeto

Durante o processo de implantação do *SpeakUp*, foram adquiridos conhecimentos importantes sobre ambientes de nuvem e sobre como disponibilizar uma aplicação web a partir de um servidor local. Inicialmente, foram explorados conceitos de *cloud computing*, como a configuração de máquinas virtuais, segurança de servidores e o uso de ferramentas de *proxy* reverso para disponibilizar serviços de maneira segura. Foram experimentados a exposição de aplicações web utilizando Cloudflare, configurando domínios e *Secure Sockets Layer* (SSL).

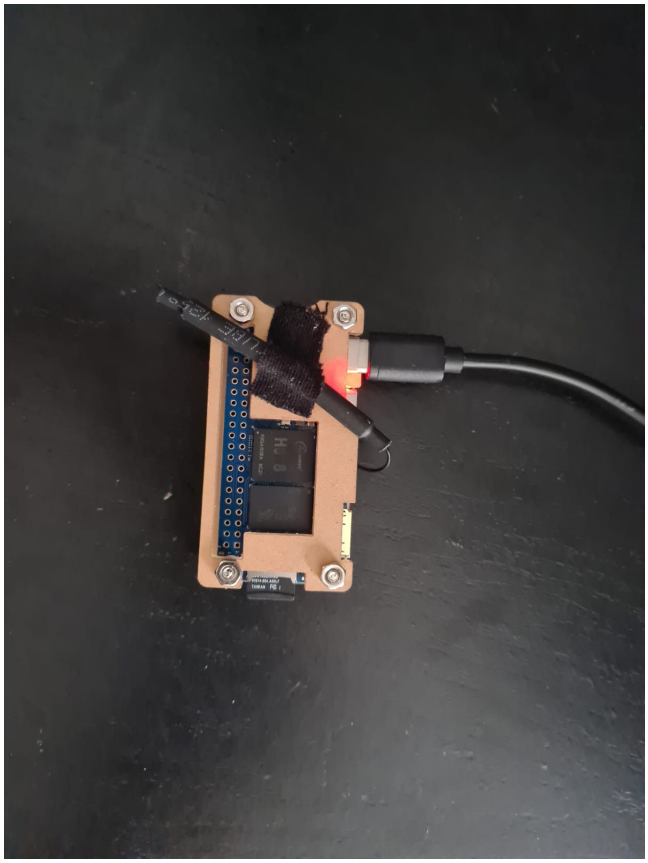
Além disso, foi utilizada a plataforma Google Cloud, especificamente a região de Oregon (us-west1), para hospedar a aplicação em uma *Virtual Machine* (VM). Esse processo envolveu a criação e configuração de instâncias com o sistema operacional desejado, definição de regras de *firewall* para liberar portas específicas (como 80 e 443). Essa experiência foi essencial para

compreender as vantagens da nuvem em termos de flexibilidade, alta disponibilidade e controle de custos.

Uma experiência significativa foi a utilização do Orange Pi Zero 2W, um mini computador de placa única (*Single Board Computer* (SBC)), como servidor local. O acesso foi realizado por meio de conexões *Secure Shell* (SSH), sendo possível acessar remotamente o dispositivo, utilizando Docker para orquestrar os contêineres da aplicação. Com o suporte do Cloudflare Tunnel, foi possível disponibilizar o site na internet de maneira segura, mesmo com *Internet Protocol* (IP) dinâmico residencial.

O Orange Pi Zero 2W, representado na Figura 11, foi o computador utilizado na implantação do projeto.

Figura 11 – Imagem do Orange Pi Zero 2W utilizado no projeto.



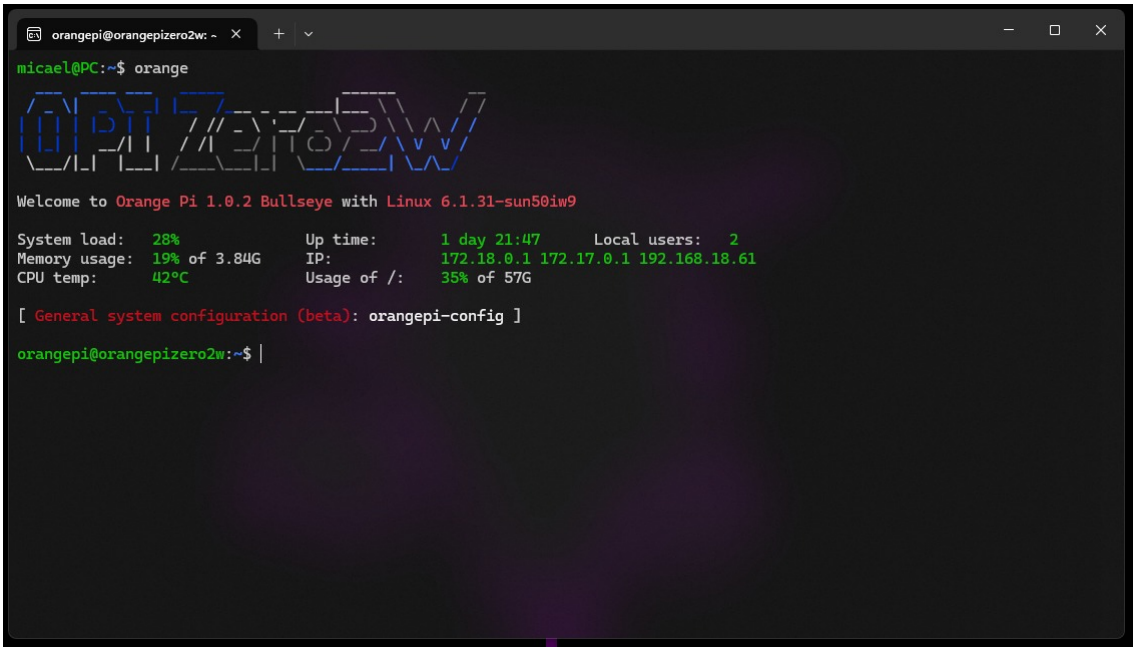
Fonte: Elaborado pelo autor (2025).

A Figura 12 mostra o resultado do terminal após acessar o Orange Pi pelo SSH. Ele se destaca por sua performance e versatilidade, especialmente em projetos de servidores locais e aplicações embarcadas. Suas principais especificações são baseadas em fontes técnicas oficiais:

- **Processador *System-on-a-Chip* (SoC)**: Allwinner H618, quad-core *Advanced RISC Machine* (ARM) Cortex-A53 de 64 bits, com frequência de até 1,5 *Gigahertz* (GHz).
- **Memória *Random Access Memory* (RAM)**: 4 GB LPDDR4, oferecendo maior capacidade de multitarefa e suporte a aplicações mais exigentes.

- **GPU**: Mali G31 MP2, compatível com OpenGL ES 1.0/2.0/3.2, OpenCL 2.0 e Vulkan 1.1, permitindo suporte a gráficos 4K.
- **Armazenamento**: Slot para cartão microSD e 16 MB de SPI Flash integrada.
- **Conectividade**:
  - Wi-Fi 5 (802.11ac)
  - Bluetooth 5.0 com suporte a *Bluetooth Low Energy* (BLE)
  - Porta Mini HDMI 2.0
  - Duas portas USB-C
  - Conector *General Purpose Input/Output* (GPIO) de 40 pinos
  - Interface de 24 pinos para expansão (suporta Ethernet 100 Mbps, portas USB adicionais, saída de vídeo composto, áudio, receptor IR, entre outros)
- **Dimensões**: 65 mm x 30 mm, tornando-o extremamente compacto e ideal para projetos com espaço limitado.
- **Sistemas Operacionais Compatíveis**:
  - Android 12 TV
  - Ubuntu 20.04 / 22.04
  - Debian 11 / 12
  - Orange Pi OS (baseado em Arch Linux)
  - Portes não oficiais do Raspberry Pi OS

Figura 12 – Interface inicial do Orange Pi Zero 2W utilizado no projeto.



Fonte: Elaborado pelo autor (2025).

Foram realizados testes para avaliar a utilização dos recursos do Orange Pi durante a execução do contêiner do projeto *SpeakUp*. A Figura 13 mostra o monitoramento do sistema utilizando a ferramenta `gotop`. Desenvolvido em GoLang, o `gotop` permitiu uma visualização clara e em tempo real do consumo de recursos. Observa-se que o uso da *Central Processing Unit* (CPU) permaneceu entre quatro por cento e oito por cento por núcleo, com consumo de memória de aproximadamente dezessete por cento (652,7 MB de 4 GB). A ausência de uso da memória *swap* e a temperatura estável indicam que o dispositivo suporta bem a execução do contêiner, demonstrando sua viabilidade como servidor local para aplicações leves.

Figura 13 – Monitoramento de recursos do Orange Pi com o container *Speakup* em execução.

Fonte: Elaborado pelo autor (2025).

Essas experiências contribuíram não apenas para a conclusão do projeto, mas também para a formação técnica do autor, proporcionando conhecimentos práticos sobre *deploy* de aplicações e infraestrutura de servidores. Ao longo do desenvolvimento, foi necessário configurar ambientes de hospedagem, gerenciar dependências, lidar com variáveis de ambiente e garantir a disponibilidade da aplicação em produção.

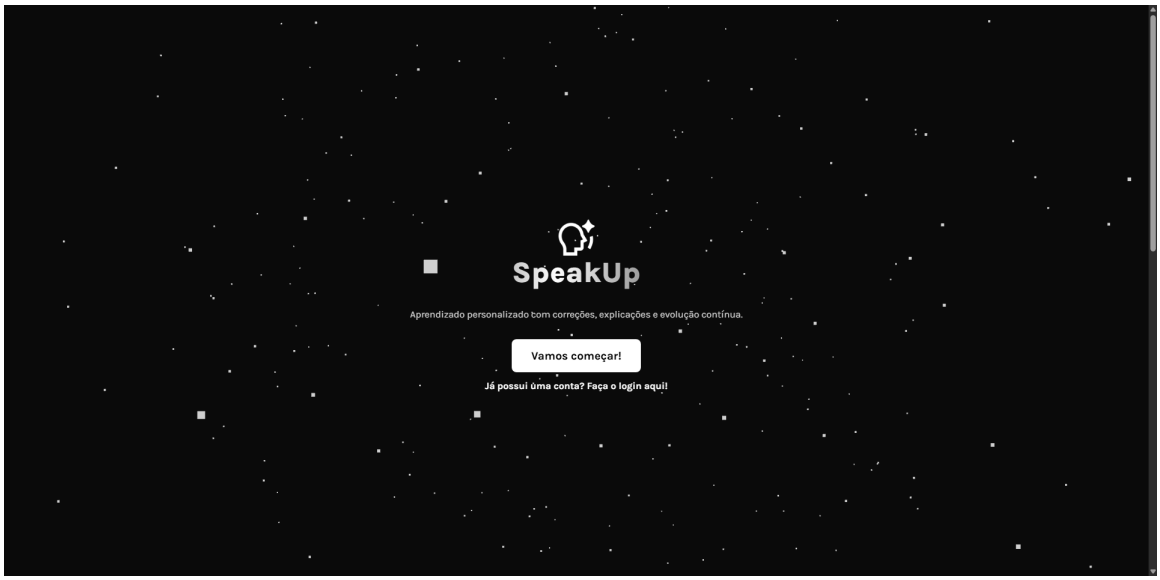
4 RESULTADOS

Este capítulo apresenta os resultados obtidos com o desenvolvimento do sistema, destacando suas principais funcionalidades, características técnicas, e o impacto da solução proposta. A seguir, serão apresentados também os aprendizados adquiridos durante o processo, especialmente relacionados ao uso de ambientes em nuvem e à implantação do sistema em servidores locais.

4.1 Apresentação do sistema

São apresentadas, a seguir, as telas do sistema, que ilustram as principais funcionalidades e o fluxo de uso da plataforma. Cada imagem mostra como o usuário interage com o sistema desde a primeira visita até o uso efetivo dos recursos disponíveis para o aprendizado de idiomas. A Figura 14 mostra a página inicial do sistema, que destaca as principais funções da plataforma de maneira clara e objetiva. Nessa tela, o visitante tem acesso à proposta da aplicação, compreendendo rapidamente seu objetivo. Também são apresentados os botões de acesso para login ou cadastro, que convidam o usuário a iniciar sua jornada de aprendizado personalizado.

Figura 14 – Tela inicial do sistema (Index).



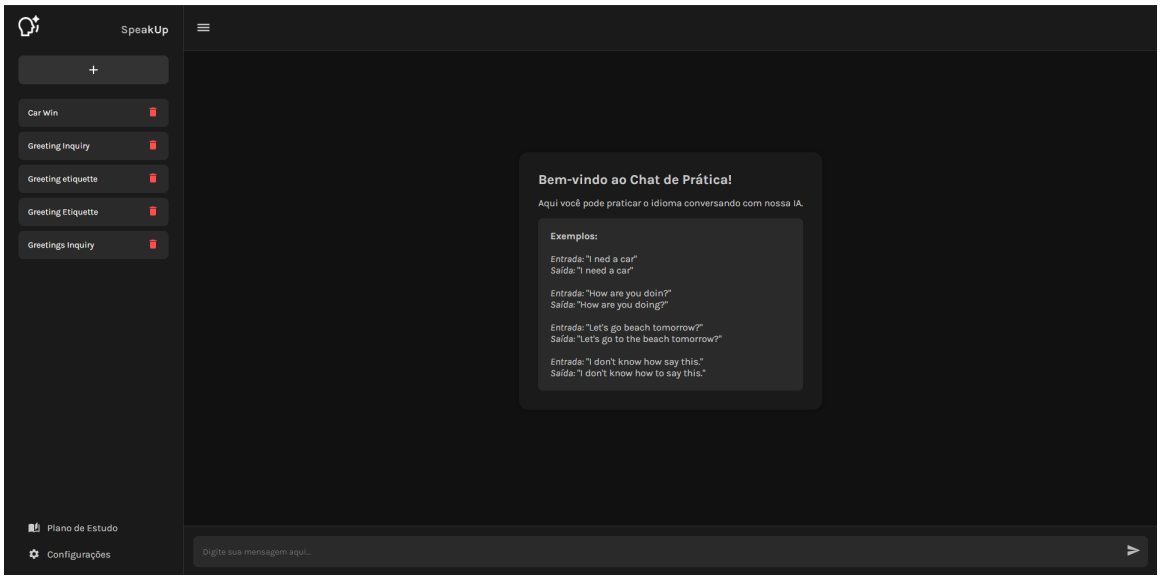
Fonte: Elaborado pelo autor (2025).

Após a autenticação, o usuário é redirecionado para a tela ilustrada na Figura 15, correspondente à página principal do sistema. Nessa interface, ele pode acessar o chat com o assistente virtual, visualizar seu plano de ensino, que é fixo para cada nível de conhecimento do usuário e modificar configurações do perfil. O *layout* dessa tela foi desenhado para proporcionar uma navegação intuitiva, com elementos bem organizados e de fácil compreensão.

A principal funcionalidade do sistema é a interação por meio da IA, conforme representado na Figura 15. A interface do chat foi desenvolvida para ser limpa e objetiva, permitindo

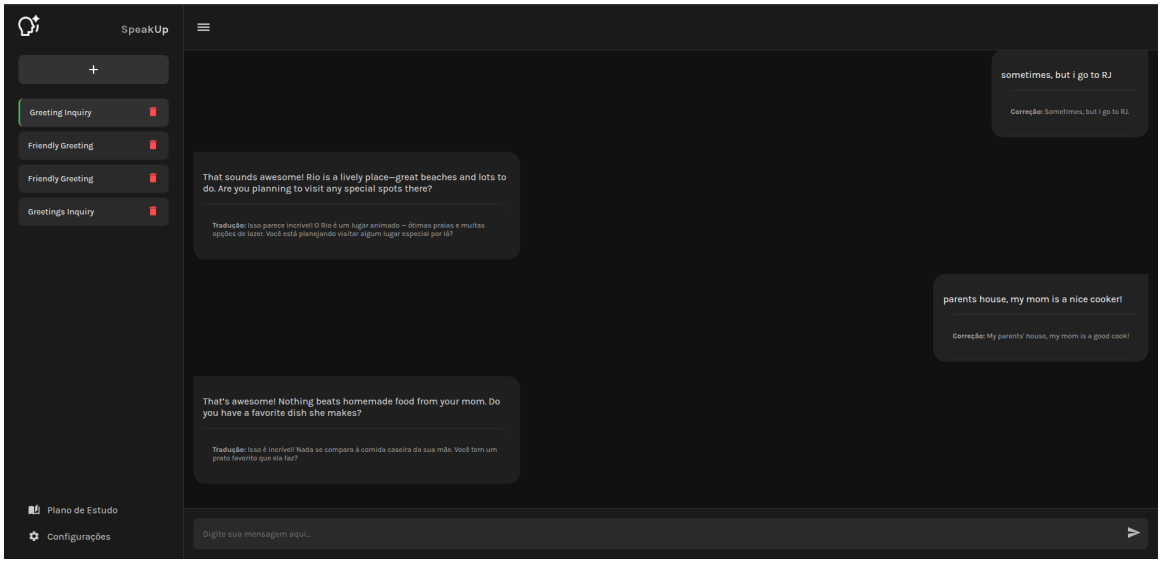
uma comunicação natural entre o usuário e o assistente. Nessa conversa, o usuário recebe correções e traduções sobre a linguagem que está aprendendo, em tempo real, como visto na Figura 16.

Figura 15 – Área de chat para interação com o chatbot.



Fonte: Elaborado pelo autor (2025).

Figura 16 – Área de chat para interação com o chatbot após a utilização do mesmo.



Fonte: Elaborado pelo autor (2025).

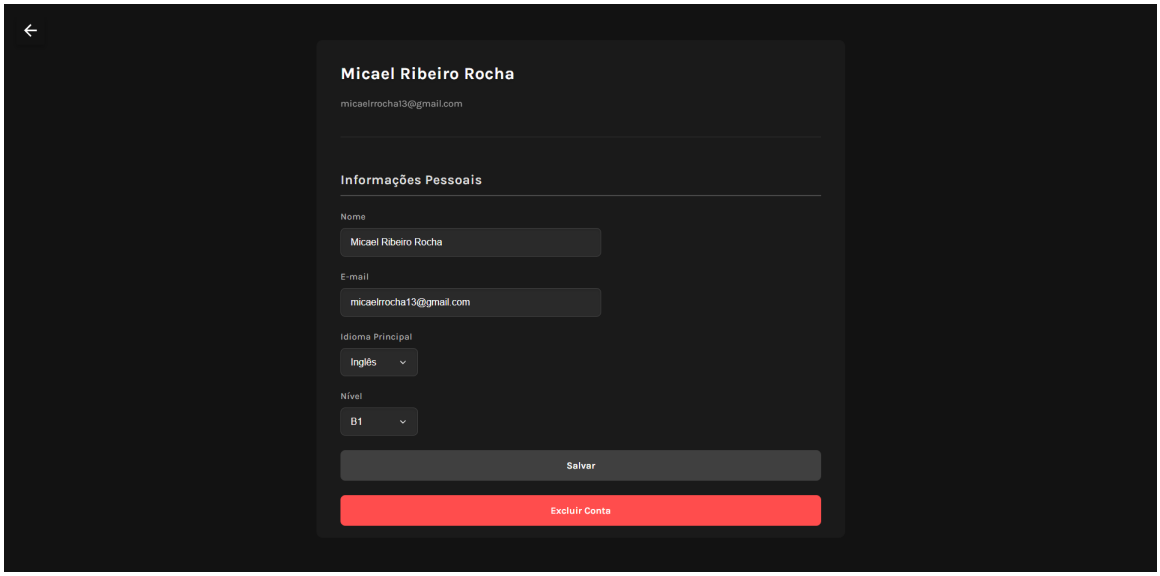
O usuário, como citado nos requisitos, terá acesso ao seu plano de estudo a qualquer momento, o mesmo que é determinado pela escolha do nível de conhecimento da língua que o usuário escolheu para seguir com o aprendizado dela.

Figura 17 – Tela de plano de ensino condizente com o nível de conhecimento do usuário.

Fonte: Elaborado pelo autor (2025).

Para que o usuário possa manter controle sobre seus dados, foi criada a tela de perfil, apresentada na Figura 18. Essa interface oferece funcionalidades básicas, como a visualização e edição de informações pessoais, incluindo nome, email, idioma em aprendizado e nível de conhecimento atual.

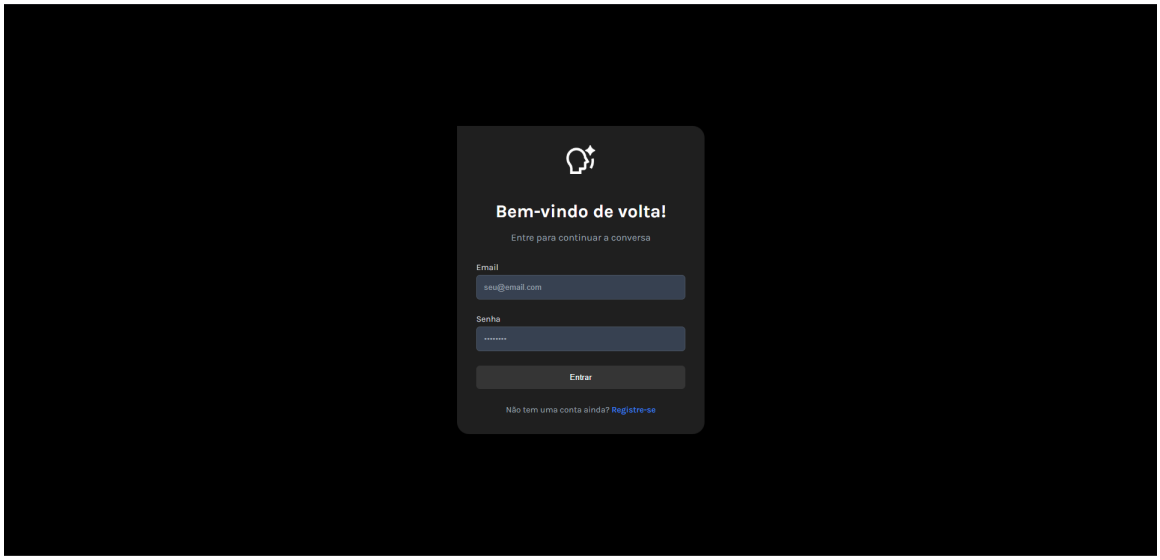
Figura 18 – Tela de edição e visualização do perfil do usuário.



Fonte: Elaborado pelo autor (2025).

As telas de autenticação foram projetadas com foco em simplicidade e segurança. A Figura 19 exibe a interface de login, onde o usuário insere suas credenciais para acessar o sistema. O formulário é direto, com campos para email e senha, e orientações claras para recuperação de acesso, caso necessário.

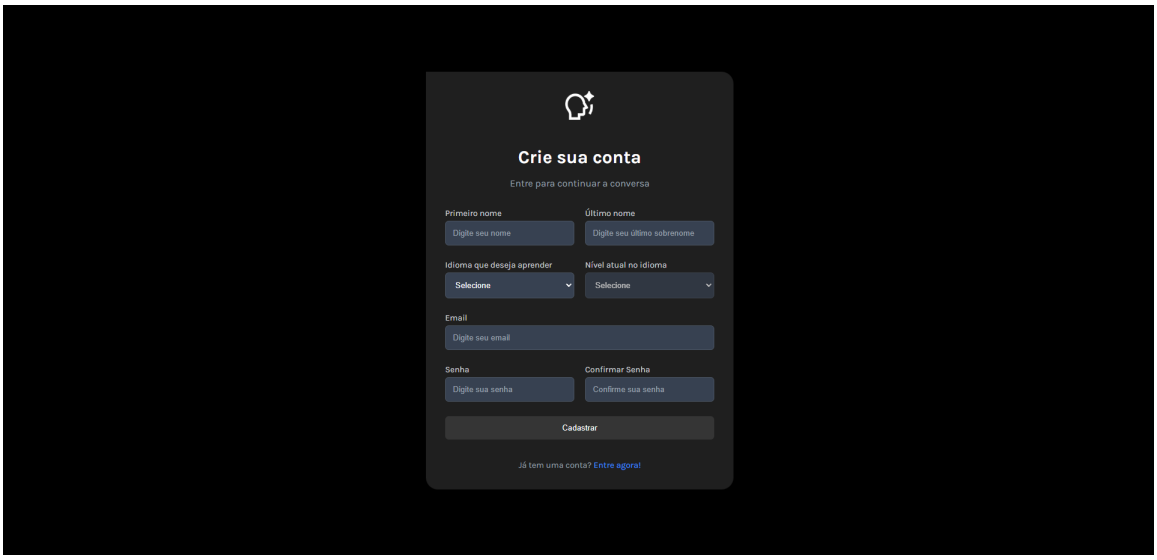
Figura 19 – Tela de login do sistema.



Fonte: Elaborado pelo autor (2025).

Por fim, a Figura 20 mostra a tela de cadastro, responsável por permitir que novos usuários criem uma conta rapidamente. Nela, são solicitadas informações como nome, email, senha, linguagem desejada e o nível atual de conhecimento. Esses dados são fundamentais para personalizar a experiência de aprendizado oferecida pela plataforma.

Figura 20 – Tela de registro de novos usuários.



Fonte: Elaborado pelo autor (2025).

5 CONCLUSÃO

Este trabalho apresentou o desenvolvimento do *SpeakUp*, uma aplicação web dedicada ao ensino de idiomas, impulsionada pelos avanços da IA. Sua concepção fundamentou-se em uma arquitetura robusta, utilizando tecnologias estabelecidas, como GoLang para o *backend*, React para o *frontend* e Docker para conteinerização e orquestração, garantindo escalabilidade, eficiência e facilidade de manutenção à solução proposta.

A partir dos resultados apresentados, considera-se que o *SpeakUp* atingiu os objetivos delimitados para esse projeto, promovendo o auxílio no aprendizado de idiomas. Por meio de funcionalidades essenciais como a correção gramatical automatizada, a tradução instantânea das respostas geradas pela IA para o português e a simulação de diálogos interativos, o sistema proporciona uma experiência de aprendizado imersivo. A aplicação da IA permite que o sistema aprenda continuamente com as interações dos usuários, adaptando-se às necessidades específicas de cada indivíduo e personalizando o processo de aprendizagem para se alinhar ao ritmo e estilo cognitivo do estudante. Dessa forma, espera-se que o *SpeakUp* ofereça conteúdo teórico criando um ambiente dinâmico e interativo.

Um dos destaques do *SpeakUp* é sua abordagem intuitiva. A interface foi projetada para ser amigável e de fácil navegação, ampliando seu alcance a usuários com diferentes níveis de familiaridade tecnológica e minimizando barreiras técnicas ao aprendizado. No que tange ao impacto social, o projeto visa democratizar o acesso ao aprendizado de novos idiomas, tornando essa experiência mais acessível e menos dependente de recursos financeiros elevados, como ocorre com os cursos tradicionais. Ao incorporar IA, o *SpeakUp* não apenas reduz a barreira do custo, mas também oferece uma alternativa eficaz e de não custo que pode alcançar diversos públicos e contextos, desde estudantes que buscam autonomia no aprendizado em casa até comunidades com acesso limitado a recursos educacionais formais. Isso é especialmente relevante em regiões onde o acesso a professores qualificados ou a cursos presenciais é restrito, promovendo a inclusão digital e educacional.

Adicionalmente, a compatibilidade e a viabilidade do *SpeakUp* em hardwares acessíveis, como o Orange Pi 2W, um microcomputador de baixo custo com 4 GB de memória RAM, reforçam o compromisso com a inclusão digital. A capacidade de executar o *SpeakUp* em dispositivos compactos e econômicos possibilita sua implementação em contextos com recursos limitados, como escolas públicas ou comunidades remotas, sem comprometer o desempenho do sistema.

5.1 Trabalhos Futuros

Para consolidar e expandir o impacto do *SpeakUp*, diversas possibilidades de evolução podem ser consideradas:

- **Expansão da simulação de diálogos:** Incluir cenários temáticos, como entrevistas de emprego, viagens internacionais e situações acadêmicas, para diversificar as oportunidades de prática.
- **Melhoria da usabilidade e acessibilidade da interface:** Realizar testes com usuários de diferentes perfis para aprimorar continuamente a interface, com foco em acessibilidade para pessoas com deficiências.
- **Ampliação da compatibilidade com hardware de baixo custo:** Otimizar o desempenho da aplicação em microcomputadores e dispositivos móveis, visando ampliar o acesso em regiões com infraestrutura tecnológica limitada.
- **Avaliação aprofundada do impacto social:** Estabelecer parcerias com instituições educacionais para mensurar e documentar os impactos reais do uso do *SpeakUp* em ambientes de ensino formal e informal.
- **Gamificação do processo de aprendizado:** Introduzir elementos de jogos, como conquistas, rankings e desafios diários, para aumentar o engajamento e a motivação dos usuários ao longo do tempo.
- **Criação de relatórios de progresso:** Implementar funcionalidades que permitam ao usuário acompanhar sua evolução de forma clara, com estatísticas de desempenho e recomendações de estudo.
- **Integração com plataformas de ensino:** Explorar a integração com *Learning Management Systems* (LMS), como Moodle ou Google Classroom, ampliando a aplicabilidade do sistema em contextos acadêmicos.
- **Suporte Multilíngue Expandido:** Ampliar o número de idiomas disponíveis para prática e tradução, tornando o sistema útil a uma audiência global e multicultural.
- **Implementação de teste de nivelamento:** Desenvolver um mecanismo de avaliação diagnóstica inicial para que usuários que não saibam seu nível no idioma possam realizar uma prova e, a partir dos resultados, receber um plano de ensino adequado à sua proficiência.
- **Realização de testes de usabilidade e aprendizagem em larga escala:** Conduzir estudos com um número maior e mais diversificado de usuários, utilizando métodos quantitativos e qualitativos para avaliar a eficácia do sistema, a usabilidade da interface e o impacto real no aprendizado de idiomas. Esses testes permitirão ajustes baseados em evidências para aumentar a efetividade do *SpeakUp*.

5.2 Considerações Finais

O desenvolvimento do *SpeakUp* mostrou a viabilidade técnica de uma solução educacional baseada em IA, aliada a um propósito social relevante. Ao integrar tecnologias com estratégias pedagógicas centradas no usuário, o sistema viabiliza uma nova abordagem para o aprendizado de idiomas. Essa combinação tecnológica e pedagógica reforça o papel do *SpeakUp* como uma ferramenta de apoio ao ensino de idiomas.

Cabe ressaltar, por fim, que o futuro da aplicação não se limita ao avanço técnico, mas requer também um compromisso contínuo com aspectos éticos, sociais e educacionais. É essencial que o *SpeakUp* continue sendo aprimorado a partir do diálogo com seus usuários, respeitando suas necessidades e especificidades. A escuta ativa da comunidade, aliada à busca constante por inovação responsável, poderá manter o sistema relevante frente às mudanças tecnológicas e pedagógicas. Além disso, torna-se fundamental considerar o impacto social e cultural gerado pela aplicação, sobretudo em contextos com acesso limitado à educação formal. O projeto, portanto, se propõe não apenas a ensinar idiomas, mas a promover inclusão, equidade e autonomia educacional.

REFERÊNCIAS

. *2023 Duolingo Language Report*. [*S.l.*]: , 2023. Acessado em: 3 maio 2025. Disponível em: <https://blog.duolingo.com/2023-duolingo-language-report/>.

ALGHAMDI, A.; ALGHAMDI, A. The impact of chatgpt on english language learning: A study of saudi efl students. **Journal of English Language Teaching**, v. 66, n. 1, p. 3–15, 2023. Disponível em: https://journals.eltai.in/jelt/article/view/JELT660103/1055.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. 3rd. ed. Boston, MA: Addison-Wesley Professional, 2012. ISBN 978-0321815736.

Brasil. **Plano Nacional de Educação (PNE)**. [*S.l.*]: , 2014. Acessado em: 3 maio 2025. Disponível em: https://pne.mec.gov.br/18-planos-subnacionais-de-educacao/543-plano-nacional-de-educacao-lei-n-13-005-2014.

Brasil. **Programa de Inovação Educação Conectada**. [*S.l.*]: , 2024. Disponível em: https://educacaoconectada.mec.gov.br/.

British Council. **O ensino de inglês no Brasil: o que dizem os indicadores**. [*S.l.*]: , 2019. Relatório Técnico. Disponível em: https://www.britishcouncil.org.br/sites/default/files/estudo_oensinodoinglesnaeducacaopublicabrasileira_0.pdf.

Catho. **Inglês é exigido em mais de 80
nível superior**. [*S.l.*]: , 2017. Acessado em: 3 maio 2025. Disponível em: https://www.catho.com.br/carreira-sucesso/nao-ter-ingles-pode-significar-perda-de-dinheiro/.

Change Vision, Inc. **Astah – Professional UML Tool**. [*S.l.*]: , 2024. https://astah.net. Acessado em: 3 maio 2025. Disponível em: https://astah.net/.

CHEN, Y.; SUN, L. Improving spoken english with ai chatbots: A longitudinal study on language learners. **ReCALL**, v. 33, n. 3, p. 253–270, 2021.

Cloudflare, Inc. **Cloudflare – Helping Build a Better Internet**. [*S.l.*]: , 2024. https://www.cloudflare.com. Acessado em: 3 maio 2025. Disponível em: https://www.cloudflare.com/.

Docker, Inc. **Docker – Empowering App Development for Developers**. [*S.l.*]: , 2024. https://www.docker.com. Acessado em: 3 maio 2025. Disponível em: https://www.docker.com/.

DONOVAN, A. A. A.; KERNIGHAN, B. W. **The Go Programming Language**. Addison-Wesley Professional, 2015. ISBN 978-0134190440. Disponível em: https://www.gopl.io/.

Figma, Inc. **Figma – The Collaborative Interface Design Tool**. [*S.l.*]: , 2024. https://www.figma.com. Acessado em: 3 maio 2025. Disponível em: https://www.figma.com/.

GitHub, Inc. **GitHub: Where the world builds software**. [*S.l.*]: , 2024. https://github.com. Acessado em: 3 maio 2025. Disponível em: https://github.com/.

HUANG, R. *et al.* Understanding the role of digital technologies in education: A review. **Sustainable Operations and Computers**, v. 3„ p. 100015, 2024. Disponível em: https://www.sciencedirect.com/science/article/pii/S2666557324000156.

LanguageTool. **LanguageTool – AI-based grammar checker**. [*S.l.*]: , 2025. Acessado em: 11 maio 2025. Disponível em: https://languagetool.org/.

LEE, J.; KWON, H. Ai-powered conversational agents for language practice: Opportunities and challenges. **Computer Assisted Language Learning**, v. 35, n. 4, p. 389–410, 2022.

LI, J.; LINK, S.; HEGELHEIMER, V. Artificial intelligence in language education: A systematic review. **Computer Assisted Language Learning**, v. 35, n. 8, p. 1485–1511, 2022.

MAGAZINE, T. **Competitividade no mercado de trabalho aumenta com domínio de língua estrangeira**. [*S.l.*]: , 2023. Disponível em: https://www.terramagazine.com.br/competitividade-no-mercado-de-trabalho-aumente-com-dominio-de-lingua-estrangeira.

Meta Platforms, Inc. **React – A JavaScript library for building user interfaces**. [*S.l.*]: , 2024. https://reactjs.org. Acessado em: 3 maio 2025. Disponível em: https://reactjs.org/.

Microsoft Corporation. **Visual Studio Code – Code Editing. Redefined.** [*S.l.*]: , 2024. https://code.visualstudio.com. Acessado em: 3 maio 2025. Disponível em: https://code.visualstudio.com/.

MongoDB, Inc. **MongoDB – The Developer Data Platform**. [*S.l.*]: , 2024. https://www.mongodb.com. Acessado em: 3 maio 2025. Disponível em: https://www.mongodb.com/.

NEWMAN, S. Building microservices: Designing fine-grained systems. **O'Reilly Media**, „, 2015. Available as a book.

Notion Labs, Inc. **Notion – One workspace. Every team.** [*S.l.*]: , 2024. https://www.notion.so. Acessado em: 3 maio 2025. Disponível em: https://www.notion.so/.

PAIVA, V. L. M. O. **Tecnologia no ensino de línguas: avanços e perspectivas**. [*S.l.*]: Editora UFMG, 2019.

Postman, Inc. **Postman – The Collaboration Platform for API Development**. [*S.l.*]: , 2024. https://www.postman.com. Acessado em: 3 maio 2025. Disponível em: https://www.postman.com/.

Prof. Dr. Álvaro Pinheiro . **IA: Subcampos da Inteligência Artificial e a Ciência de Dados**. [*S.l.*]: , 2024. https://www.youtube.com/watch?v=w8SdS-QYQ50&t=89s.

REIS, L. M. Desafios no ensino de línguas estrangeiras no brasil. **Revista de Educação Linguística**, v. 5, n. 2, p. 34–49, 2020.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3. ed. Pearson, 2016. Disponível em: https://thuvienso.hoasen.edu.vn/handle/123456789/8967.

SMITH, J.; DOE, J. Chatgpt in language education: Opportunities and challenges. **ERIC Journal**, v. 58, n. 2, p. 45–60, 2023. Disponível em: https://files.eric.ed.gov/fulltext/EJ1426851.pdf.

Tecnoblog. **Duolingo terá videochamadas com personagem usando IA**. [*S.l.*]: , 2024. https://www.mundoconectado.com.br/inteligencia-artificial/duolingo-videochamadas-ia-lily-android/.

The Git Development Community. **Git – Distributed Version Control System**. [*S.l.*], 2024. Acessado em: 3 maio 2025. Disponível em: https://git-scm.com/.

USO do GPT-4 no Duolingo. [*S.l.*]: , 2023. https://blog.duolingo.com/duolingo-max/.

WANG, I. X. *et al.* Neural automated writing evaluation with corrective feedback. **arXiv preprint arXiv:2402.17613**, „, 2024. Disponível em: https://arxiv.org/abs/2402.17613.

WANG, L.; ZHANG, W. Exploring the use of chatgpt in language learning: A case study. **Canadian Journal of Learning and Language Studies**, v. 1, n. 1, p. 10–20, 2023. Disponível em: https://www.cjlls.ca/index.php/cjlls/article/view/87/72.

ZHOU, X.; LI, W. Adaptive language learning with nlp: Dynamic adjustment of question complexity based on learner proficiency. **Journal of Educational Technology Society**, v. 26, n. 1, p. 45–59, 2023.

ANEXOS

ANEXO A – Listagens do código-fonte

Listagem 1 – Estrutura que representa um chat do sistema

```go
package models
type Chat struct {
    ID string json:"id" bson:"id"
    UserID string json:"user_id" bson:"user_id"
    StartTime string json:"start_time" bson:"start_time"
    Topic string json:"topic" bson:"topic"
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 2 – Estrutura que representa uma mensagem do sistema

```go
package models
type Message struct {
    ID string json:"id" bson:"id"
    ChatID string json:"chat_id" bson:"chat_id"
    Content string json:"content" bson:"content"
    Sender string json:"sender" bson:"sender"
    Type string json:"type" bson:"type"
    CreatedAt string json:"created_at" bson:"created_at"
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 3 – Estrutura que representa o usuário do sistema

```go
package models
type User struct {
    ID string json:"id" bson:"id"
    Name string json:"name" bson:"name"
    Email string json:"email" bson:"email"
    Password string json:"password" bson:"password"
    Language string json:"language" bson:"language"
    Level string json:"level" bson:"level"
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 4 – Função responsável por logar um usuário existente.

```
func Login(c *gin.Context) {
  client := config.GetMongoClient()
  var user models.User
  if err := c.ShouldBindJSON(&user); err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
    return
  }
  collection := client.Database("speakup").Collection("users")
  var result models.User
  err := collection.FindOne(c, bson.M{"email": user.Email}).Decode(&result)
  if err != nil {
    c.JSON(http.StatusUnauthorized, gin.H{"error": "Invalid email"})
    return
  }
  err = bcrypt.CompareHashAndPassword([]byte(result.Password), []byte(user.Password)
    )
  if err != nil {
    c.JSON(http.StatusUnauthorized, gin.H{"error": "Invalid password"})
    return
  }
  token, err := middlewares.GenerateJWT(result.ID, result.Email, result.Language,
    result.Level)
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to generate token"
    })
    return
  }
  c.Set("authToken", token)
  c.JSON(http.StatusOK, gin.H{"message": "Login successful", "token": token})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 5 – Função responsável por criar um novo usuário no sistema

```go
func CreateUser(c *gin.Context) {
  client := config.GetMongoClient()
  var user models.User
  if err := c.ShouldBindJSON(&user); err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
    return
  }
  collection := client.Database("speakup").Collection("users")
  var existingUser models.User
  err := collection.FindOne(c, bson.M{"email": user.Email}).Decode(&existingUser)
  if err == nil {
    c.JSON(http.StatusConflict, gin.H{"error": "User with this email already exists"
    })
    return
  }
  if err != mongo.ErrNoDocuments {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Database error"})
    return
  }
  user.ID = uuid.New().String()
  hashedPassword, err := bcrypt.GenerateFromPassword([]byte(user.Password), bcrypt.
    DefaultCost)
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to hash password"
    })
    return
  }
  user.Password = string(hashedPassword)
  _, err = collection.InsertOne(c, user)
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to create user"})
    return
  }
  c.JSON(http.StatusOK, gin.H{"message": "User created successfully"})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 6 – Função responsável por resgatar todos os usuários no sistema

```go
func GetUsers(c *gin.Context) {
  client := config.GetMongoClient()
  id := c.Param("id")
  collection := client.Database("speakup").Collection("users")
  var user models.User
  err := collection.FindOne(c, bson.M{"id": id}).Decode(&user)
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to get user"})
    return
  }
  c.JSON(http.StatusOK, gin.H{"user": user})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 7 – Função responsável por atualizar um usuário no sistema

```go
func UpdateUser(c *gin.Context) {
  client := config.GetMongoClient()
  id := c.Param("id")
  var user models.User
  if err := c.ShouldBindJSON(&user); err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
    return
  }
  updateDoc := bson.M{}
  if user.Name != "" {
    updateDoc["name"] = user.Name
  }
  if user.Email != "" {
    updateDoc["email"] = user.Email
  }
  if user.Language != "" {
    updateDoc["language"] = user.Language
  }
  if user.Level != "" {
    updateDoc["level"] = user.Level
  }
  if user.Password != "" {
    hashedPassword, err := bcrypt.GenerateFromPassword([]byte(user.Password), bcrypt
    .DefaultCost)
    if err != nil {
      c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to hash password
    "})
      return
    }
    updateDoc["password"] = string(hashedPassword)
  }
  if len(updateDoc) == 0 {
    c.JSON(http.StatusOK, gin.H{"message": "No fields to update"})
    return
  }
  collection := client.Database("speakup").Collection("users")
  _, err := collection.UpdateOne(c, bson.M{"id": id}, bson.M{"$set": updateDoc})
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to update user"})
    return
  }
  c.JSON(http.StatusOK, gin.H{"message": "User updated successfully"})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 8 – Função responsável por deletar um usuário no sistema.

```go
func DeleteUser(c *gin.Context) {
  client := config.GetMongoClient()
  id := c.Param("id")
  collection := client.Database("speakup").Collection("users")
  _, err := collection.DeleteOne(c, bson.M{"id": id})
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to delete user"})
    return
  }
  c.JSON(http.StatusOK, gin.H{"message": "User deleted successfully"})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 9 – Função responsável por criar um novo chat no sistema

```go
func CreateChat(c *gin.Context) {
  var chat models.Chat
  if err := c.ShouldBindJSON(&chat); err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
    return
  }

  chat.ID = uuid.New().String()
  chat.UserID = middlewares.GetUserIDFromContext(c)
  chat.StartTime = time.Now().Format(time.RFC3339)

  db := config.GetMongoClient()
  collection := db.Database("speakup").Collection("chats")
  _, err := collection.InsertOne(c, chat)
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to create chat"})
    return
  }

  c.JSON(http.StatusCreated, chat)
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 10 – Função para buscar um chat específico pelo seu identificador

```go
func GetChatById(c *gin.Context) {
  id := c.Param("id")
  db := config.GetMongoClient()
  collection := db.Database("speakup").Collection("chats")
  var chat models.Chat
  err := collection.FindOne(c, map[string]string{"id": id}).Decode(&chat)
  if err != nil {
    c.JSON(http.StatusNotFound, gin.H{"error": "Chat not found"})
    return
  }

  c.JSON(http.StatusOK, chat)
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 11 – Função que retorna todos os chats cadastrados

```go
func GetChats(c *gin.Context) {
  db := config.GetMongoClient()
  collection := db.Database("speakup").Collection("chats")
  cursor, err := collection.Find(c, map[string]string{})
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to get chats"})
    return
  }

  var chats []models.Chat
  if err := cursor.All(c, &chats); err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to get chats"})
    return
  }

  c.JSON(http.StatusOK, chats)
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 12 – Função responsável por atualizar os dados de um chat

```go
func UpdateChat(c *gin.Context) {
  id := c.Param("id")
  var chat models.Chat
  if err := c.ShouldBindJSON(&chat); err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
    return
  }
  db := config.GetMongoClient()
  collection := db.Database("speakup").Collection("chats")
  _, err := collection.UpdateOne(c, map[string]string{"id": id}, chat)
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to update chat"})
    return
  }
  c.JSON(http.StatusOK, chat)
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 13 – Função que realiza a exclusão de um chat com base no ID

```go
func DeleteChat(c *gin.Context) {
  id := c.Param("id")
  db := config.GetMongoClient()
  collection := db.Database("speakup").Collection("chats")
  _, err := collection.DeleteOne(c, map[string]string{"id": id})
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to delete chat"})
    return
  }
  c.JSON(http.StatusOK, gin.H{"message": "Chat deleted successfully"})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 14 – Função que retorna todos os chats pertencentes a um usuário

```go
func GetChatsByUserId(c *gin.Context) {
  id := middlewares.GetUserIDFromContext(c)
  db := config.GetMongoClient()
  collection := db.Database("speakup").Collection("chats")
  cursor, err := collection.Find(c, map[string]string{"user_id": id})
  if err != nil {
    c.JSON(http.StatusInternalServerError,
        gin.H{"error": "Failed to get chats"})
    return
  }
  var chats []models.Chat
  if err := cursor.All(c, &chats); err != nil {
    c.JSON(http.StatusInternalServerError,
        gin.H{"error": "Failed to get chats"})
    return
  }
  c.JSON(http.StatusOK, gin.H{"chats": chats})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 15 – Função que permite a criação de uma mensagem no sistema.

```go
func CreateMessage(c *gin.Context) {
  var message models.Message
  if err := c.ShouldBindJSON(&message); err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
    return
  }
  message.ID = uuid.New().String()
  message.CreatedAt = time.Now().Format(time.RFC3339)
  db := config.GetMongoClient()
  collection := db.Database("speakup").Collection("messages")
  _, err := collection.InsertOne(c, message)
  if err != nil {
    c.JSON(http.StatusInternalServerError,
        gin.H{"error": "Failed to create message"})
    return
  }
  c.JSON(http.StatusCreated, message)
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 16 – Função que permite o resgate de uma mensagem no sistema.

```go
func GetMessageById(c *gin.Context) {
  id := c.Param("id")
  db := config.GetMongoClient()
  collection := db.Database("speakup").Collection("messages")
  var message models.Message
  err := collection.FindOne(c, map[string]string{"id": id}).Decode(&message)
  if err != nil {
    c.JSON(http.StatusNotFound, gin.H{"error": "Message not found"})
    return
  }
  c.JSON(http.StatusOK, message)
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 17 – Função que permite o resgate de todas as mensagens no sistema.

```go
func GetMessages(c *gin.Context) {
  db := config.GetMongoClient()
  collection := db.Database("speakup").Collection("messages")
  cursor, err := collection.Find(c, map[string]string{})
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to get messages"})
    return
  }
  var messages []models.Message
  if err := cursor.All(c, &messages); err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to get messages"})
    return
  }
  c.JSON(http.StatusOK, messages)
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 18 – Função que permite a atualização de uma mensagem no sistema.

```go
func UpdateMessage(c *gin.Context) {
  id := c.Param("id")
  var message models.Message
  if err := c.ShouldBindJSON(&message); err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
    return
  }
  db := config.GetMongoClient()
  collection := db.Database("speakup").Collection("messages")
  _, err := collection.UpdateOne(c, map[string]string{"id": id}, bson.M{"$set":
    message})
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to update message"
    })
    return
  }
  c.JSON(http.StatusOK, gin.H{"message": "Message updated successfully"})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 19 – Função que permite apagar uma mensagem no sistema.

```go
func DeleteMessage(c *gin.Context) {
  id := c.Param("id")
  db := config.GetMongoClient()
  collection := db.Database("speakup").Collection("messages")
  _, err := collection.DeleteOne(c, map[string]string{"id": id})
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to delete message"
    })
    return
  }
  c.JSON(http.StatusOK, gin.H{"message": "Message deleted successfully"})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 20 – Função que permite resgatar mensagens de determinado chat.

```go
func GetMessagesByChatId(c *gin.Context) {
  chatId := c.Param("id")
  db := config.GetMongoClient()
  collection := db.Database("speakup").Collection("messages")
  cursor, err := collection.Find(c, map[string]string{"chat_id": chatId})
  if err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to get messages"})
    return
  }
  var messages []models.Message
  if err := cursor.All(c, &messages); err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to get messages"})
    return
  }
  c.JSON(http.StatusOK, messages)
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 21 – Função que gera uma resposta de diálogo usando IA

```
func GenerateResponseDialog(c *gin.Context) {
    promptPath := filepath.Join("prompts", "promptDialog.txt")
    promptBytes, err := os.ReadFile(promptPath)
    if err != nil {
        c.JSON(http.StatusInternal monomericServerError, gin.H{"error": "Failed to load prompt: " + err.Error()})
        return
    }
    prePrompt := string(promptBytes)
    var request struct {
        Message string `json:"message"`
        ChatID  string `json:"chat_id"`
    }
    if err := c.ShouldBindJSON(&request); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }
    db := config.GetMongoClient()
    collection := db.Database("speakup").Collection("messages")
    cursor, err := collection.Find(c, map[string]string{"chat_id": request.ChatID})
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to get chat history"})
        return
    }
    var messages []models.Message
    if err := cursor.All(c, &messages); err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to get chat history"})
        return
    }
    var chatHistory strings.Builder
    for _, msg := range messages {
        chatHistory.WriteString(fmt.Sprintf("%s: %s\n", msg.Sender, msg.Content))
    }
    connector := connectors.NewGeminiConnector()
    resumeHist, err := connector.GenerateResponse(context.Background(), "Format the following chat history to only show user response and AI response: "+chatHistory.String())
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    fullPrompt := fmt.Sprintf("%s\nChat history:\n%s\nATENTION! All Before this point is system instructions and chat history, to generate your response consider the current user message -> = %s\nAnswer me in this language: %s",
        prePrompt, resumeHist, request.Message, middlewares.GetLanguageFromContext(c))
    dialogueResp, err := connector.GenerateResponse(context.Background(), fullPrompt)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    c.JSON(http.StatusOK, gin.H{"response": dialogueResp})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 22 – Função que corrige erros gramaticais em textos

```go
func GenerateResponseCorrection(c *gin.Context) {
    promptPath := filepath.Join("prompts", "promptCorrection.txt")
    promptBytes, err := os.ReadFile(promptPath)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to load prompt: " + err.Error()})
        return
    }
    prompt := string(promptBytes)
    var request struct {
        Message string `json:"message"`
    }
    if err := c.ShouldBindJSON(&request); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }
    connector := connectors.NewGeminiConnector()
    correctionResp, err := connector.GenerateResponse(context.Background(), "Answer me in this language: " + middlewares.GetLanguageFromContext(c) + prompt + request.Message)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    c.JSON(http.StatusOK, gin.H{"response": correctionResp})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 23 – Função que traduz textos para o idioma especificado

```go
func GenerateResponseTranslate(c *gin.Context) {
    promptPath := filepath.Join("prompts", "promptTranslate.txt")
    promptBytes, err := os.ReadFile(promptPath)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Erro ao carregar o prompt: " + err.Error()})
        return
    }
    prompt := string(promptBytes)
    var req struct {
        Message string `json:"message" binding:"required"`
    }
    if err := c.ShouldBindJSON(&req); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Dados inválidos: " + err.Error()})
        return
    }
    connector := connectors.NewGeminiConnector()
    response, err := connector.GenerateResponse(context.Background(), prompt+req.Message)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Erro ao gerar tradução: " + err.Error()})
        return
    }
    c.JSON(http.StatusOK, gin.H{"response": response})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 24 – Função que gera tópicos de duas palavras

```
func GenerateResponseTopic(c *gin.Context) {
    var request struct {
        Message string `json:"message"`
    }
    if err := c.ShouldBindJSON(&request); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }
    connector := connectors.NewGeminiConnector()
    topicResp, err := connector.GenerateResponse(context.Background(), "Please
    generate a topic for the following text, return only words, generate with 2
    words only: "+request.Message)
    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }
    c.JSON(http.StatusOK, gin.H{"response": topicResp})
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 25 – Interface para conectores de IA

```
package connectors

import "context"

type AIConnector interface {
    GenerateResponse(ctx context.Context, message string) (string, error)
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 26 – Função do conector para o modelo Gemini

```go
package connectors

import (
    "context"
    "log"
    "os"
    "github.com/google/generative-ai-go/genai"
    "google.golang.org/api/option"
)

type GeminiConnector struct {
    client *genai.Client
    model  *genai.GenerativeModel
}

func NewGeminiConnector() *GeminiConnector {
    apiKey := os.Getenv("GEMINI_API_KEY")
    ctx := context.Background()
    client, err := genai.NewClient(ctx, option.WithAPIKey(apiKey))
    if err != nil {
        log.Fatal(err)
    }

    model := client.GenerativeModel("gemini-2.0-flash")

    return &GeminiConnector{
        client: client,
        model:  model,
    }
}

func (g *GeminiConnector) GenerateResponse(ctx context.Context, message string) (
    string, error) {
    resp, err := g.model.GenerateContent(ctx, genai.Text(message))
    if err != nil {
        return "", err
    }

    if resp != nil && len(resp.Candidates) > 0 && len(resp.Candidates[0].Content.
    Parts) > 0 {
        if text, ok := resp.Candidates[0].Content.Parts[0].(genai.Text); ok {
            return string(text), nil
        }
    }

    return "", nil
}

func (g *GeminiConnector) Close() {
    g.client.Close()
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 27 – Arquivo docker-compose.yml utilizado no *SpeakUp*.

```
services:
backend:
build:
context: ./backend/go
volumes:
- ./backend/go:/app
ports:
- "8080:8080"
environment:
- JWT_KEY=secret
command: ["/go/bin/air"]

ui-web:
build:
context: ./frontend/web
volumes:
- ./frontend/web:/app
- /app/node_modules
ports:
- "3000:3000"
environment:
- NODE_ENV=development
- CHOKIDAR_USEPOLLING=true
depends_on:
- backend
```

Fonte: Elaborado pelo autor (2025).

Listagem 28 – Exemplo de teste unitário para o *handler* de usuário (simplificado).

```go
package handlers

import (
  "bytes"
  "encoding/json"
  "net/http"
  "net/http/httptest"
  "speakup/config"
  "speakup/models"
  "testing"

  "github.com/gin-gonic/gin"
  "github.com/stretchr/testify/assert"
  "go.mongodb.org/mongo-driver/bson"
  "go.mongodb.org/mongo-driver/bson/primitive"
  "go.mongodb.org/mongo-driver/mongo/integration/mtest"
  "golang.org/x/crypto/bcrypt"
)

func TestCreateUser(t *testing.T) {
    mt := mtest.New(t, mtest.NewOptions().ClientType(mtest.Mock))

    gin.SetMode(gin.TestMode)

    t.Run("should create user successfully", func(t *testing.T) {
        mt.Run("success", func(mt *mtest.T) {
            mt.AddMockResponses(
                mtest.CreateCursorResponse(0, "speakup.users", mtest.FirstBatch),
            )

            insertedID := primitive.NewObjectID()
            mt.AddMockResponses(
                mtest.CreateSuccessResponse(
                    bson.E{Key: "ok", Value: 1},
                    bson.E{Key: "insertedId", Value: insertedID},
                ),
            )

            config.SetMongoClient(mt.Client)

            user := models.User{
                Name:     "John Doe",
                Email:    "john.doe@example.com",
                Password: "password123",
                Language: "en",
            }

            userJSON, _ := json.Marshal(user)
            req, _ := http.NewRequest(http.MethodPost, "/user", bytes.NewBuffer(
    userJSON))
            req.Header.Set("Content-Type", "application/json")

            w := httptest.NewRecorder()
            c, _ := gin.CreateTestContext(w)
            c.Request = req

            CreateUser(c)

            assert.Equal(t, http.StatusOK, w.Code)

            var response map[string]string
            err := json.Unmarshal(w.Body.Bytes(), &response)
            assert.NoError(t, err)
            assert.Equal(t, "User created successfully", response["message"])
        })
    })
}
```

Fonte: Elaborado pelo autor (2025).

Listagem 29 – Workflow de testes com GitHub Actions.

```yaml
name: Run Tests

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  backend-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Go
        uses: actions/setup-go@v4
        with:
          go-version: '1.22'
      - name: Run Backend Tests
        run: |
          cd backend/go
          go mod tidy
          go test ./...

  frontend-web-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'
      - name: Install Frontend Dependencies
        run: |
          cd frontend/web
          npm install
      - name: Run Frontend Tests
        run: |
          cd frontend/web
          npm test -- --watchAll=false
```

Fonte: Elaborado pelo autor (2025).