



Universidade de Brasília

Instituto de Ciências Exatas  
Departamento de Ciência da Computação

**Desenvolvendo um Agente Conversacional com  
modelos de linguagem de larga escala para ambientes  
administrativos acadêmicos**

Raphael Paula Leite Muller

Monografia apresentada como requisito parcial  
para conclusão do Bacharelado em Ciência da Computação

Orientador

Prof. Dr. Flávio de Barros Vidal

Brasília  
2023



# Dedicatória

Dedico este trabalho de graduação à minha amada família, Gabriel, Luciana e Gilson Müller, cujo apoio incondicional me sustentou em cada etapa da minha trajetória acadêmica. Vocês foram os pilares que me proporcionaram motivação e encorajamento na conclusão deste projeto e na iniciação de minha carreira profissional.

Dedico também aos amigos íntimos e minha namorada Helena, por me acompanharem na realização de tantos sonhos. Seu apoio e nossos momentos compartilhados fizeram toda a diferença em minha jornada de desenvolvimento pessoal, acadêmico e profissional.

# Agradecimentos

Agradeço ao professor orientador Flávio Vidal pelas contribuições, sejam no meu desenvolvimento profissional, enquanto colegas de equipe, quanto na execução deste trabalho, mesmo diante de todos os percalços que acompanharam o seu desenvolvimento.

# Resumo

Este trabalho propõe a implementação de um sistema *web* baseado em um agente conversacional, com o intuito de auxiliar os discentes do Departamento de Computação da Universidade de Brasília quanto a resolução de dúvidas de cunho acadêmico. Objetiva-se construir um assistente alimentado por uma base de conhecimento unificada, que seja capaz de aumentar a agilidade na resolução de perguntas frequentes e reduzir a dispersão de informações entre os meios de comunicação do departamento. O núcleo deste projeto foi construído com base no modelo de linguagem GPT-3.5-Turbo, e explora-se o seu aprimoramento a partir da técnica de *retrieval-augmented generation*, bem como a sua integração com outros módulos desenvolvidos para este projeto. Como base de conhecimento para o agente, utilizou-se o Guia do Calouro provido pelo departamento, bem como o portal de notícias de seu *website*. Assim, obtiveram-se resultados positivos na resposta de perguntas pertinentes ao contexto administrativo acadêmico, e o sistema proposto demonstrou a viabilidade da utilização da arquitetura aqui definida para o cumprimento dos objetivos definidos.

**Palavras-chave:** modelos de linguagem de larga escala, geração aumentada por recuperação, GPT, bancos de dados vetoriais, agentes conversacionais

# Abstract

This work proposes implementing a web-based conversational agent system to assist students from the Department of Computer Science at the University of Brasília in resolving academic doubts. The objective is to build an assistant powered by a unified knowledge base capable of increasing agility in answering frequently asked questions and reducing information dispersion among the department's communication channels. The core of this project was built based on the GPT-3.5-Turbo language model, and its improvement is explored through the retrieval-augmented generation technique and its integration with other modules developed for this project. The department's Freshman Guide (Guia do Calouro) and its website's news portal were used as the agent's knowledge base. Thus, positive results were obtained in answering questions pertinent to the academic-administrative context, and the proposed system demonstrated the feasibility of using the architecture defined here to achieve the defined objectives.

**Keywords:** large language models, retrieval-augmented generation, GPT, vector databases, conversational agents

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Motivação . . . . .	1
1.2	Objetivos . . . . .	2
1.2.1	Objetivos Gerais . . . . .	2
1.2.2	Objetivos Específicos . . . . .	2
1.3	Justificativa . . . . .	3
1.4	Organização do Manuscrito . . . . .	3
<b>2</b>	<b>Fundamentação Teórica</b>	<b>5</b>
2.1	Agentes Conversacionais . . . . .	5
2.2	Modelos de linguagem de larga escala (MLL) . . . . .	7
2.2.1	Transformers . . . . .	7
2.2.2	Retrieval-augmented generation . . . . .	8
<b>3</b>	<b>Requisitos Funcionais do Sistema Proposto</b>	<b>10</b>
3.1	Especificação dos requisitos de software . . . . .	10
3.1.1	REQ-1: Sistema de autenticação . . . . .	10
3.1.2	REQ-2: Agente Conversacional . . . . .	12
3.1.3	REQ-3: Modelo de Dados . . . . .	13
3.1.4	REQ-4: Interface de troca de mensagens . . . . .	14
<b>4</b>	<b>Desenvolvimento Realizado</b>	<b>16</b>
4.1	QAService . . . . .	17
4.1.1	Estrutura das consultas a MLL . . . . .	18
4.2	AuthService . . . . .	29
4.3	Bancos de dados . . . . .	32
4.4	Armazenamento de documentos . . . . .	33
4.5	Banco de dados vetorial . . . . .	34
4.6	Camada de API . . . . .	37
4.6.1	Rotas . . . . .	38

4.7	Aplicação-cliente . . . . .	43
4.7.1	Tela de login . . . . .	44
4.7.2	Menu lateral . . . . .	45
4.7.3	Formulário de chat . . . . .	47
<b>5</b>	<b>Conclusões e Trabalhos futuros</b>	<b>50</b>
	<b>Referências</b>	<b>53</b>



# Lista de Figuras

3.1	Ilustração do REQ-1 e seus sub-requisitos . . . . .	11
3.2	Ilustração do REQ-2 e seus sub-requisitos . . . . .	12
3.3	Ilustração do REQ-3 e seus sub-requisitos. . . . .	13
4.1	Exemplo de resposta insatisfatória, por parte da aplicação ChatGPT, baseada no modelo GPT, para perguntas estipuladas neste projeto. . . . .	17
4.2	Resposta do agente para a pergunta "Como o departamento pode assistir alunos para participação em eventos?". . . . .	21
4.3	Referência apresentada na resposta do agente para a pergunta "Como o departamento pode assistir alunos para participação em eventos?". . . . .	22
4.4	Resposta do agente para a primeira pergunta de referência . . . . .	25
4.5	Página 19 do Guia do Calouro, que indica o documento-fonte utilizado para a segunda pergunta de referência. . . . .	26
4.6	Página 20 do Guia do Calouro, que continua o texto sobre trancamento. . . . .	27
4.7	Resposta o agente para pergunta sobre estágio obrigatório. . . . .	28
4.8	Ilustração do processo de codificação de <i>token</i> JWT. Acessado em 17/11/2023. . . . .	31
4.9	Diagrama de entidades e relacionamentos. . . . .	33
4.10	Fluxo de busca, extração e vetorização de documentos. . . . .	36
4.11	Fluxo de persistência de mensagem de usuário e geração de resposta pelo agente. . . . .	43
4.12	Tela de login de usuários . . . . .	45
4.13	Demonstração de erro de submissão do formulário de autenticação. . . . .	45
4.14	Demonstração visual do menu lateral da aplicação. . . . .	46
4.15	Demonstração de alerta exibido em caso de falha na recuperação de sessões de conversação. . . . .	47
4.16	Demonstração do alerta em caso de falha na recuperação de mensagens . . . . .	48
4.17	Demonstração do aviso de carregamento de envio de mensagem . . . . .	48
4.18	Figura que ilustra o <i>dashboard</i> completo, incluindo o menu lateral e formulário de chat. . . . .	49

# Lista de Tabelas

4.1 Rotas implementadas para a aplicação . . . . .	40
--	----

# Capítulo 1

## Introdução

### 1.1 Motivação

Com a recente popularização dos modelos de linguagem de larga escala (do inglês *Large Language Model*, LLM), observa-se uma mudança de paradigma na maneira como os setores da sociedade interagem com computadores [1]. Isso porque a adoção desses modelos torna a interação indivíduo-computador cada vez mais natural e próxima às maneiras do raciocínio humano. Nesse contexto, uma nova onda de produtos e serviços inovadores surge de tal modo a promover disrupção em diversos ramos da indústria, e assim consolidar a inteligência artificial como um pilar tecnológico da sociedade contemporânea [2].

Um ramo industrial que demonstrou tremendo avanço na última década trata dos agentes conversacionais[3]. Isto é, aqueles capazes de compreender e responder elocuições seguindo padrões da comunicação humana. A melhora no desempenho desses agentes é perceptível mediante a evolução da inteligência artificial, em especial do campo de estudo do processamento de linguagem natural (PLN). Alguns exemplos notórios dessa tecnologia na atualidade são os assistentes virtuais implementados por bancos para triar e solucionar problemas de usuários, tais quais produtos concretos como Alexa, da empresa Amazon[4].

Esses assistentes já são amplamente conhecidos e ocasionalmente utilizados por uma parcela significativa daqueles com acesso à internet e serviços de telecomunicações. No entanto, é importante destacar que muitos setores ainda não aproveitam plenamente a tecnologia moderna ligada à inteligência artificial, que por vezes é subutilizada, conforme [2, Figure 6]. Alguns exemplos expostos pelo artigo referenciado, com base no mercado estadunidense, são serviços de assistência social, transporte, bem como a indústria de suporte e serviços administrativos. Nesse contexto, este trabalho propõe e detalha um caso de implementação de agentes virtuais focados no suporte ao usuário, com ênfase no

Departamento de Ciência da Computação (CIC) da Universidade de Brasília, a partir de uma perspectiva técnica da Engenharia de Software.

## 1.2 Objetivos

Diante da motivação exposta, definiram-se alguns objetivos de cunho geral ou específico, que orientam a realização deste trabalho e definem o escopo do que foi produzido. Nesse contexto, os tópicos apresentados neste capítulo estão intrinsecamente associados à exploração de um caso de uso prático de implantação de sistemas baseados em IA no contexto acadêmico, conforme delineado nas seguintes seções.

### 1.2.1 Objetivos Gerais

O objetivo principal deste trabalho é propor uma arquitetura computacional de um agente virtual conversacional para suporte ao corpo discente do Departamento de Ciência da Computação (CIC). Isto é, um projeto que abranja a concepção, desenvolvimento e implantação de um software baseado em mecanismos modernos de inteligência artificial (IA) e capaz de solucionar dúvidas específicas sobre temas pertinentes ao departamento. Dessa forma, espera-se conceber uma ferramenta de apoio ao trabalho humano prestado pelos encarregados desse departamento, e assim promover uma maior eficácia no suporte ao alunos.

### 1.2.2 Objetivos Específicos

A implementação do sistema proposto neste trabalho visa contribuir especialmente à comunidade de discentes do CIC. Assim, constituem-se os seguinte objetivos específicos, estes de caráter técnico:

- Documentar um projeto prático que abranja as múltiplas fases da elaboração de um sistema baseado em tecnologias e métodos modernos relacionados à IA.
- Explorar um caso de uso específico de sistemas baseados em agentes conversacionais, que se trata do suporte ao usuário, bem como as motivações para sua utilização e os seus potenciais impactos em uma comunidade.
- Disponibilizar uma implementação concreta de um sistema de código-fonte aberto à comunidade de estudantes, baseado nos aprendizados adquiridos ao longo do curso Bacharelado em Ciência da Computação e experiências corporativas do autor, com finalidade educacional.

Além disso, busca-se também oferecer uma alternativa aos tradicionais meios de comunicação e busca de informações oferecidos pelo CIC. Assim, estabelece-se o objetivo de implementar um sistema que ofereça apoio a dúvidas de caráter estudantil e pertinentes ao departamento, ao passo que centraliza o acesso à informação bem como garante eficiência e agilidade na resolução de questões pertinentes.

### **1.3 Justificativa**

Existe uma vasta gama de solicitações resolvíveis por meio do atendimento ao usuário oferecida pelo CIC. Alguns exemplos são: matrícula em estágio, dúvidas sobre atividades complementares e inscrição em monitoria. Em geral, qualquer pessoa pode buscar esse serviço, visto que o atendimento abrange tanto solicitantes internos ao departamento, quanto os demais. Além disso, o CIC conta com diversos fluxos processuais pré-definidos para cada tipo de solicitação, sejam nos seus formulários de atendimento [5], ou em suas bases de conhecimento, à exemplo do Guia do Calouro [6]. Dessa forma, é notório que, nos últimos anos, tais processos de atendimento vem apresentando uma notória modernização.

Ainda assim, sob a perspectiva de um discente do CIC, existem pontos que podem ser melhorados no atual sistema de atendimento. Em geral, o acesso a informação é descentralizado, e as solicitações por vezes podem levar dias para serem respondidas. Assim, este projeto propõe que a introdução de um agente autônomo, sob o contexto do suporte a usuários, por vezes pode representar maior agilidade da resolução de questões frequentes, bem como maior disponibilidade em comparação às jornadas de trabalho de atendentes humanos.

Além disso, o agente aqui especificado centraliza informações de múltiplos canais de comunicação do CIC. Com isso, espera-se eliminar a necessidade de consultar várias fontes por parte dos estudantes, e portanto reduzir a dispersão de informações, que por vezes pode resultar em buscas longas e ineficientes. Finalmente, a centralização pode contribuir na consistência nas informações compartilhadas, garantindo que todos tenham acesso a respostas provenientes dos mesmos documentos-fontes, que agora são gerenciados em um único repositório de dados.

### **1.4 Organização do Manuscrito**

Este manuscrito foi organizado em cinco capítulos. Seu desenvolvimento segue um fluxo linear de construção de raciocínio, que inicia-se com a exposição de um contexto sobre o tema, e as motivações para o trabalho desenvolvido, conforme o Capítulo 1. Em seguida, estabelece-se a fundamentação dos principais conceitos abordados, no Capítulo 2, de tal

forma a embasar definições técnicas e mitigar lacunas de conhecimento relacionadas aos temas discutidos.

O Capítulo 3 visa iniciar a documentação do teor prático deste trabalho, ao apresentar requisitos de *software* que foram estabelecidos para orientar, sob uma metodologia técnica, o desenvolvimento de um sistema para cumprimento dos objetivos propostos. Tal sistema é amplamente documentado no Capítulo 4, que discorre sob a implementação material dos requisitos estipulados, e concede uma visão profunda sobre os módulos desenvolvidos e os resultados obtidos.

Finalmente, o Capítulo 5 deste trabalho conclui o tema proposto, por meio do relacionamento entre metas e resultados. Além disso, introduz sugestões para continuação deste trabalho, de modo a aumentar o seu impacto dentro do contexto proposto.

# Capítulo 2

## Fundamentação Teórica

### 2.1 Agentes Conversacionais

Os agentes conversacionais, também denominados *chatbots* ou assistentes virtuais, constituem uma categoria de sistemas de inteligência artificial (IA) dedicados à interação natural com usuários humanos [7]. Esta seção se propõe a examinar de forma objetiva o conceito de agentes conversacionais, contextualizando-os em uma perspectiva histórica e analisando diferentes abordagens observadas em sua linha evolutiva.

Historicamente, os agentes conversacionais foram inicialmente implementados mediante regras programadas, seguindo roteiros e regras predefinidos e padrões de interação limitados. Um exemplo notório de agente conversacional rudimentar é o ELIZA [8], desenvolvido na década de 1960. Esse utilizava padrões predefinidos para reconhecer palavras-chave em entradas do usuário, gerando respostas pré-determinadas. Embora limitado em sua capacidade de compreender contexto e nuances da linguagem natural, ELIZA foi pioneiro na interação homem-máquina e estabeleceu os alicerces para o desenvolvimento subsequente de *chatbots* sofisticados.

Embora primitivos em comparação com as tecnologias contemporâneas, o ELIZA e demais agentes conversacionais antigos desempenharam um papel significativo na história da IA, influenciando abordagens posteriores para a construção de novos modelos de assistentes virtuais [9][10]. Isso porque a evolução subsequente, impulsionada pelo avanço da aprendizagem de máquina, notadamente no campo do processamento de linguagem natural (PLN), conduziu a tecnologia a um maior grau de sofisticação [11].

Ainda sob uma perspectiva histórica, os assistentes virtuais evoluíram para utilizar abordagens mais avançadas do que as implementadas pelo ELIZA. Um exemplo notável é o ALICE (Artificial Linguistic Internet Computer Entity) [12], desenvolvido na década de 1990. Esse assistente representou uma melhoria substancial em relação ao ELIZA,

incorporando uma base de conhecimento mais extensa e uma lógica mais elaborada para responder a uma variedade maior de consultas.

O ALICE utilizava um sistema de regras hierárquicas e um mecanismo de correspondência de padrões para interpretar e responder às entradas dos usuários. Isso é baseado em uma variação da linguagem de marcação XML, denominada AIML [13]. Assim, sua arquitetura permite uma interação mais dinâmica e com melhor análise do contexto das perguntas se comparada aos *chatbots* rudimentares. No entanto, o ALICE ainda carecia da capacidade de compreensão profunda da linguagem natural presente nos modelos contemporâneos [9][10]. Assim, esse é um marco intermediário na evolução dos *chatbots*, representando um avanço significativo em relação às abordagens rudimentares, mas ainda situado em uma fase anterior à incorporação de técnicas avançadas de aprendizagem de máquina e ao poder computacional oferecido pelos modelos de linguagem de larga escala (MLL) [14].

Os modelos de linguagem de larga escala, tais como GPT-3 [9] e BERT [10], oferecem capacidades significativamente mais avançadas se comparados à ELIZA e ALICE, e representam o atual estado da arte em agentes conversacionais. Esses modelos diferenciam-se à medida que são capazes de aprender e generalizar a partir de exemplos, permitindo-lhes lidar com perguntas e interações não vistas durante o treinamento. Eles também são capazes de entender contextos complexos, nuances e ambiguidades na linguagem natural, pontos que comumente resultam em mal desempenho de modelos baseados em regras pré-determinadas [12][8]. Estes modelos, baseados em redes neurais profundas, são treinados em grandes quantidades de texto, e assim demonstram uma compreensão sem precedentes da linguagem natural em comparação com tecnologias rudimentares.

De modo a exemplificar a disrupção promovida por esses recentes modelos, toma-se o estudo de caso da aplicação do GPT no exame de licenciamento médico dos Estados Unidos [15]. Esse artigo demonstra que o GPT obteve notas suficientes ou quase suficientes para aprovação no exame, sem qualquer tipo de treinamento ou reforço adicional. Assim, os resultados da pesquisa sugerem alta capacidade de compreensão das perguntas, bem como conhecimento profundo sobre o domínio técnico de aplicação da Medicina objetivado pelo exame. Dessa forma, relaciona-se a este trabalho a medida que demonstra efeitos práticos da aplicação de IA para processamento de linguagem natural, a medida que o GPT obteve resultados positivos em um exame destinado aos humanos.

Outra aplicação bem-sucedida do GPT é ilustrada pelo modelo BloombergGPT[16], que utiliza o MLL para processar linguagem natural dentro de um contexto de finanças. O estudo obteve resultados que superam consideravelmente outros modelos existentes, e possibilita a aplicação dessa tecnologia para tarefas de classificação de notícias, análise de sentimento de mercado, reconhecimento de entidades nomeadas, dentre outros. Diferen-



temente do trabalho citado anteriormente, este artigo explora o reforço da aprendizagem do GPT por meio de treinamento sobre diversos conjuntos de dados financeiros. Assim, o GPT demonstra, mais uma vez, ótimos resultados em um domínio de problema altamente especializado.

Em síntese, os agentes conversacionais evoluíram para ferramentas sofisticadas capazes de compreender e gerar linguagem natural de maneira próxima à comunicação humana [11]. Esta análise objetiva busca fornecer uma visão abrangente das origens, do desenvolvimento histórico e das atuais abordagens tecnológicas empregadas na criação de agentes conversacionais, propiciando uma compreensão que fundamenta o desenvolvimento prático do sistema aqui documentado.

## 2.2 Modelos de linguagem de larga escala (MLL)

No contexto do avanço exponencial das tecnologias de processamento de linguagem natural (PLN), esta Seção explora uma arquitetura específica denominada *transformer*, que é comumente utilizada em MLL. Além disso, descreve uma técnica de aprimoramento de resultados de MLL denominada *retrieval-augmented generation*.

### 2.2.1 Transformers

Uma das arquiteturas mais notórias de MLL é denominada **transformer**, proposta no artigo *Attention Is All You Need* [17]. Esta arquitetura tem sido fundamental na condução de avanços significativos no campo do processamento de linguagem natural (PLN) e na formação de modelos como as versões mais recentes do GPT.

Um aspecto primordial dessa arquitetura é o mecanismo de **atenção**, que permite que o modelo pondere diferentes partes de uma sequência de entrada ao gerar uma saída. Em contraste com modelos de sequência como LSTM (Long Short-Term Memory), onde as informações de entrada são processadas sequencialmente, os *transformers* permitem que cada *token* de uma sequência interaja com qualquer outro *token*, independentemente de sua posição. Esta abordagem possibilita uma compreensão mais rica do contexto dentro de uma sequência, e isso é refletido por meio de respostas melhor elaboradas dentro da conjuntura de fatos expostos em uma entrada.

Além disso, essa arquitetura possui capacidade de realizar processamento em paralelo, o que permite um treinamento mais eficiente em hardware moderno. Ademais, demonstra bons resultados no tratamento eficiente de sequências de entrada longas, algo que é frequentemente desafiador para modelos baseados em LSTM devido ao problema do desvanecimento do gradiente [18].

No entanto, a arquitetura *transformer* apresenta alguns desafios de implementação. Isso porque a quantidade de cálculos realizados em suas previsões aumenta quadraticamente com o comprimento da sequência de entrada, visto que as partes da sequência são inter-relacionadas independentemente de ordenação, e isso torna o processamento de textos muito longos computacionalmente dispendioso. Além disso, os *transformers* são dependentes de grandes volumes de dados de treinamento, como descrito em [19, p. 12].

Não obstante suas limitações, a arquitetura *transformer* provou ser extremamente eficaz para tarefas de PLN e atualmente compõe o estado da arte para tecnologias de agentes conversacionais [14]. No contexto deste trabalho, utilizou-se o modelo GPT, que implementa a arquitetura *transformer*, como base para a construção de um agente conversacional, conforme evidenciado na Seção 4.1. Além disso, a integração deste modelo pré-treinado aliviou dificuldades relacionadas ao tempo e quantidade de dados requeridos para treinamento e obtenção de resultados expressivos.

## 2.2.2 Retrieval-augmented generation

A abordagem de *retrieval-augmented generation* (RAG) [20] é uma técnica avançada no campo de PLN que visa melhorar a qualidade e relevância das gerações automáticas feitas por um agente conversacional. Este paradigma combina duas estratégias principais: a recuperação de informações relevantes e a geração de texto.

### Fase de recuperação

A fase de recuperação é responsável por identificar informações relevantes para a resposta de um enunciado a partir de conjuntos de dados pré-existente. Em outras palavras, busca-se, em uma base de conhecimento, conteúdo que possa ser útil para embasar, contextualizar ou aprimorar uma resposta ao enunciado. Isso é frequentemente realizado por meio de modelos de busca ou sistemas de recuperação de informação. Uma das maneiras de realizar isso é por meio da representação vetorial de documentos, onde documentos são incorporados em um espaço vetorial de alta dimensão, e realiza-se busca por similaridade entre o enunciado e os documentos por meio de cálculos algébricos [21][22].

### Fase de geração

A fase de geração é encarregada de criar uma resposta coerente e relevante para um dado enunciado, baseado nas informações recuperadas na fase anterior. Assim, os textos recuperados são fornecidos como entrada para um determinado modelo de linguagem, e suas próprias capacidades de PLN permitem que o conteúdo apresentado componha uma resposta mais informada. Modelos de linguagem autoregressivos, como o GPT [9], ou

modelos de linguagem condicional, como o T5 [23], são comumente empregados nesta etapa.

## **Motivação e benefícios**

A abordagem *retrieval-augmented generation* representa um avanço significativo na capacidade de modelos de linguagem gerarem texto relevante e contextualmente informado. Ao integrar efetivamente a recuperação de informações ao processo de geração, essa técnica demonstra potencial para melhorar a qualidade e a eficácia de sistemas de processamento de linguagem natural em diversas aplicações, em especial aquelas que não possuem bom desempenho nativo de um determinado modelo de MLL.

Este projeto utiliza a premissa de que a utilização de MLL juntamente à ampliação do conhecimento via RAG formam um artifício poderoso para o cumprimento dos objetivos deste trabalho. Portanto, em posse da fundamentação aqui apresentada, o próximo passo para a concepção do software é determinar quais são os seus requisitos funcionais, de tal modo que estes conceitos possam ser materializados em uma aplicação interativa e disponível aos usuários finais.

# Capítulo 3

## Requisitos Funcionais do Sistema Proposto

Este capítulo expande o contexto técnico relativo ao planejamento e elaboração do sistema proposto. Assim, serão definidos os requisitos estabelecidos durante a concepção do sistema, os módulos que o compõem, bem como aspectos técnicos de implementação e engenharia de software empregados.

### 3.1 Especificação dos requisitos de software

Os requisitos são identificados pelo prefixo **REQ-**, seguido pelo número do requisito. Um requisito pode conter sub-requisitos atômicos e com maior grau de especificidade. Esses são identificados pelo prefixo **REQ-**, seguidos pelo número do requisito principal e o número do sub-requisito, separados por ponto. Um exemplo de requisito é **REQ-1**, e um exemplo de sub-requisito é **REQ-1.2**.

#### 3.1.1 REQ-1: Sistema de autenticação

Estabelece que o sistema deve conter um serviço de autenticação de usuários. Esse é composto pelos sub-requisitos **REQ-1.1**, que define as credenciais de acesso a serem usadas, **REQ-1.2**, que prevê o processo de login e identificação de usuários, e **REQ-1.3** que denota o cadastro de usuários. Um resumo deste requisito e seus sub-requisitos é ilustrado na imagem abaixo.

O objetivo do REQ-1 é identificar cada usuário unicamente dentro do sistema. Assim, ao utilizar a plataforma, o usuário deve ver somente os dados que lhe pertencem.

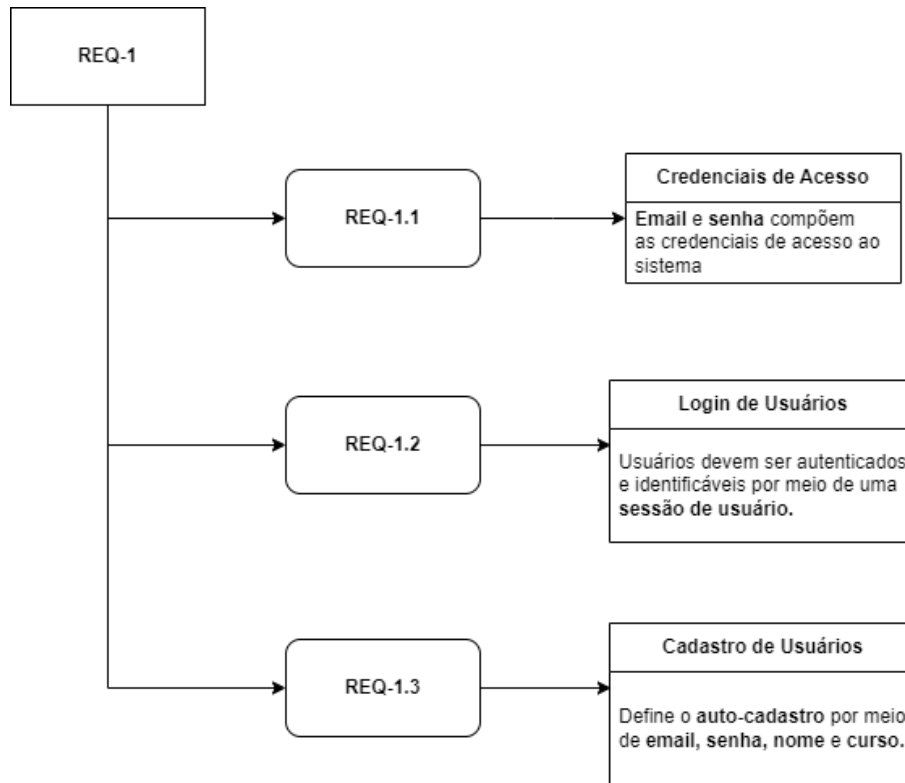


Figura 3.1: Ilustração do REQ-1 e seus sub-requisitos

### REQ-1.1: Credenciais de acesso

Cada usuário irá utilizar um **email pessoal** e uma **senha**. Os endereços de email devem ser únicos dentro do sistema. Ou seja, um mesmo email não pode representar dois usuários diferentes.

### REQ-1.2: Login de usuários

O mecanismo de login tem intuito de inicializar uma sessão de usuário. Este requisito prevê que o sistema deve implementar um módulo de login capaz de identificar qual usuário está realizando uma determinada requisição à aplicação, após o fornecimento de credenciais. Dessa forma, deve ser possível definir o escopo dos dados aos quais o requisitante possui acesso de forma segura.

### REQ-1.3: Cadastro de usuários

Para utilizar a plataforma e obter acesso ao sistema de autenticação, os usuários deverão realizar um auto-cadastro. O cadastro será composto pelo apelido, seja nome civil ou social, endereço de email, senha, e curso.

### 3.1.2 REQ-2: Agente Conversacional

O REQ-2 é o principal requisito deste trabalho, e trata da criação de um agente conversacional inteligente, baseado em mecanismos modernos de processamento de linguagem natural (PLN). Esse processo é detalhado nos sub-requisitos abaixo, e resumido na seguinte imagem:

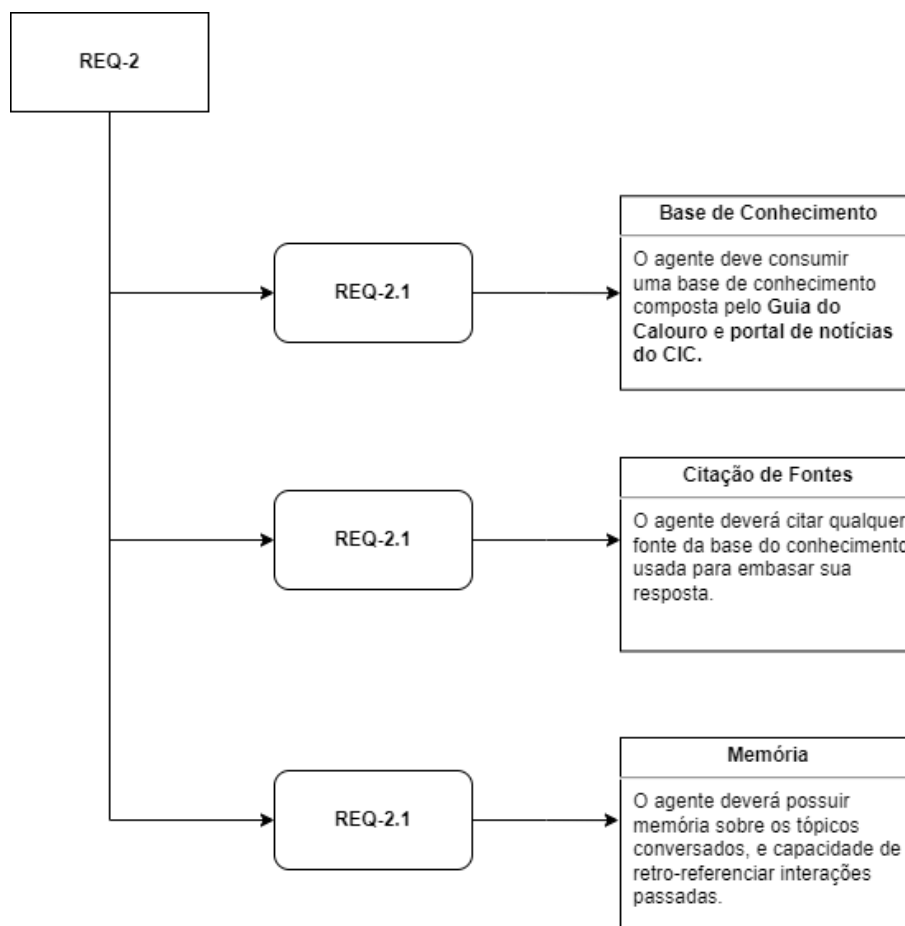


Figura 3.2: Ilustração do REQ-2 e seus sub-requisitos

#### REQ-2.1: Definição da base de dados do agente conversacional

Este requisito estipula que o agente conversacional será alimentado pelo Guia do Calouro do CIC, bem como o portal de notícias do site desse departamento. O agente deve ser capaz de responder quaisquer perguntas que possam ser solucionadas pela leitura do conteúdo textual desses arquivos.

### REQ-2.2: Citação de arquivos-fontes

O agente será alimentado por uma base de conhecimento que consiste em documentos HTML e PDF. O agente deverá referenciar quaisquer arquivos-fontes utilizados para embasar a resposta. No caso de documentos PDF, a resposta deve conter o título e página. Já no caso de páginas HTML, deve conter a sua URL.

### REQ-2.3: Memória sobre conversas passadas

Conforme uma conversa é construída, o agente deve ser capaz de compreender menções a mensagens previamente enviadas. Isto é, o agente deve possuir memória sobre os tópicos que foram conversados, e capacidade de referenciá-los quando solicitado pelo usuário.

### REQ-2.4: Ordenação das mensagens

As conversas entre humano e agente ocorrerão por meio de **ciclos**. Esses são iniciados com uma mensagem do humano, e encerram com uma resposta do agente. Logo, todo ciclo possui duas mensagens, sendo a primeira do usuário e a última do *chatbot*. A ordenação dessa conversa deve ser preservada. Isto é, para um dado ciclo de conversação, a mensagem do agente não deverá, em qualquer hipótese, anteceder a mensagem do humano.

### 3.1.3 REQ-3: Modelo de Dados

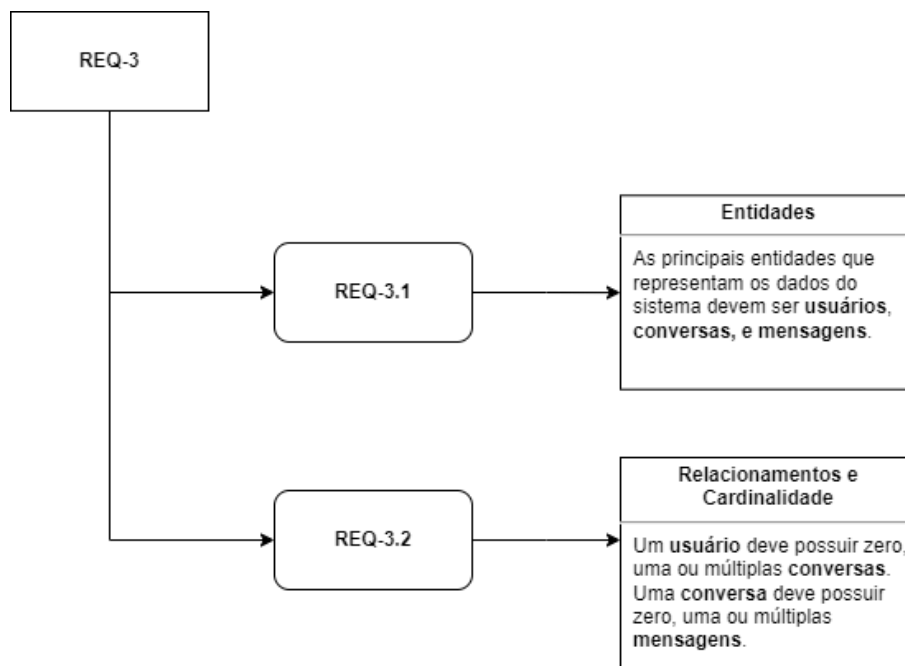


Figura 3.3: Ilustração do REQ-3 e seus sub-requisitos.

Este requisito especifica as entidades do modelo de dados do sistema, e a forma como interagem. Ressalta-se que este requisito não visa especificar a tecnologia de persistência, mas sim os conceitos-chave que devem prover estrutura para os dados armazenados.

### **REQ-3.1 Entidades**

Estipula a criação de três entidades principais, sendo **usuários**, **sessões de conversação** e **mensagens**.

### **REQ-3.2 Cardinalidade e relacionamentos**

Um único usuário poderá criar múltiplas sessões de conversação, e cada sessão de conversação pode conter um número arbitrário de mensagens.

### **3.1.4 REQ-4: Interface de troca de mensagens**

Este requisito prevê a criação de uma interface de chat que permita:

- O envio e recebimento de mensagens pelo usuário.
- O recebimento de mensagens pelo agente conversacional.
- A visualização do histórico de mensagens de uma sessão de conversação.
- A criação de uma nova sessão de conversação.
- A visualização das sessões de conversação existentes.
- A autenticação de usuários.
- O cadastro de usuários.

Os requisitos propostos neste Capítulo orientam o desenvolvimento do sistema. Esses tem o intuito de especificar as funcionalidades a serem implementadas, bem como o resultado que se pretende alcançar a nível de *software*, sem necessariamente ditar tecnologias específicas ou decisões arquiteturais mais aprofundadas. Comumente, requisitos funcionais são debatidos com as partes interessadas em um projeto de sistema, e realizam-se ciclos de definição, validação e verificação para atestar a sua correta concretização. No caso deste projeto, por se tratar de uma iniciativa independente, esses requisitos foram definidos pelo autor, tendo em vista os objetivos propostos, mas sem diálogo com outras partes interessadas. Assim, ressalta-se que este Capítulo possui influência subjetiva e não necessariamente reflete os interesses do CIC ou de outros atores relacionadas.



Nesse sentido, a implementação destes requisitos é um processo separado, e configura o próximo passo da elaboração do *software*. Assim, o capítulo seguinte discute as realizações concretas destes requisitos a nível de código, e trata da arquitetura de *software* proposta para sanar as metas estabelecidas. Isso é realizado por meio da descrição dos módulos implementados, suas responsabilidades, e os resultados alcançados.

# Capítulo 4

## Desenvolvimento Realizado

Com o objetivo de atender o requisito de geração de resposta automática (REQ-2), definido na Seção 3.1.2, optou-se pela utilização de um modelo de linguagem de larga escala (MLL). Isso porque, como verificado na Seção 2.1, essa classe de modelos representa uma alternativa moderna e eficiente para tarefas de PLN e geração inteligente de texto.

Neste projeto, adotou-se o modelo **GPT 3.5 Turbo** como base para o agente conversacional. A classe de modelos GPT é provavelmente a mais notória do presente, e é ativamente utilizada em aplicações das mais distintas áreas do conhecimento. Por isso, essa escolha baseia-se nos seguintes fatores: disponibilidade via API amplamente documentada, viabilidade de custos para a execução deste projeto, suporte a língua portuguesa, e eficiência comprovada em aplicações do tipo pergunta e resposta. Vale ressaltar que neste trabalho, os termos GPT e GPT-3.5 Turbo são utilizados intercambiavelmente.

Ainda assim, a escolha levou em consideração pontos negativos. Por se tratar de um modelo proprietário, não existe total transparência sobre dados utilizados no treinamento do GPT. Além disso, em testes preliminares, atestou-se que o modelo possui baixa precisão em perguntas relacionadas ao CIC e sua base de conhecimento disponível on-line, conforme verificado na figura 4.1. Isso sugere que tais dados não foram alimentados ao GPT durante o seu treinamento.

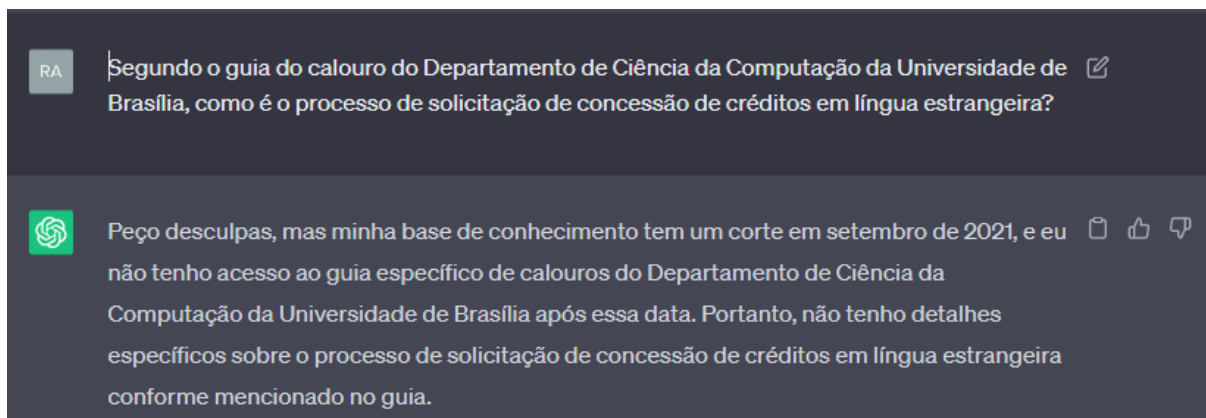


Figura 4.1: Exemplo de resposta insatisfatória, por parte da aplicação ChatGPT, baseada no modelo GPT, para perguntas estipuladas neste projeto.

A situação acima exemplifica que o modelo GPT por si só não é suficiente para cumprir os requisitos estabelecidos. Apesar de compor o artifício básico para a geração de respostas inteligentes, existem diversos outros pontos a serem trabalhados na implementação de um sistema que efetivamente cumpra os objetivos descritos. Assim, as subseções deste capítulo descrevem os demais módulos implementados para o cumprimento integral do propósito deste trabalho.

## 4.1 QAService

O primeiro ponto a ser observado para a utilização do GPT neste trabalho é a sua capacidade de integração a outros sistemas. Conforme a documentação desse modelo, esse conta com uma API REST, que pode ser consumida por chamadas HTTP, e mantida pela OpenAI. Dentre suas capacidades está gerar uma resposta para um dado enunciado utilizando o GPT, e esse é exatamente o comportamento buscado neste trabalho. Em outras palavras, deseja-se utilizar o GPT para formular respostas às perguntas dos utilizadores.

Sabe-se que a API do GPT habilita a integração deste modelo com o sistema aqui proposto. Ainda assim, esse processo costuma envolver um estudo preliminar de aspectos como metodologia de autenticação esperada pela API, formato das respostas e requisições, e rotas disponibilizadas para cada recurso. De modo a facilitar a utilização dessa API e garantir mais robustez às suas chamadas, optou-se por usar uma biblioteca externa denominada **Langchain**.

A biblioteca Langchain contém diversas funcionalidades. Uma delas é o encapsulamento de diversos MLL em uma interface única, a qual atua como uma fachada para as chamadas de API de cada MLL, e abstrai a sua utilização em métodos simples, sem exigir

conhecimento profundo de sua implementação. Em outras palavras, um único método dessa biblioteca encapsula toda a lógica necessária para realizar uma chamada à API de um MLL com robustez e simplicidade, incluindo aspectos como geração de cabeçalhos, tratamento de erros e manipulação de dados entre diferentes formatos.

Outro ponto notável da biblioteca é que essa facilita a integração de um MLL com ferramentas utilitárias externas. Alguns exemplos são o suporte a recursos como memória, conexão a base de conhecimento externa, e ferramentas de análise para *debugging*. Recursos como esses passam a ser grandes aliados na construção de agentes conversacionais sofisticados, e sua utilização neste trabalho é discutida em maiores detalhes nas subseções seguintes.

Dada a contextualização acima, implementou-se um serviço denominado **QAService**, ou serviço de perguntas e respostas, que encapsula a lógica para consulta do modelo GPT. Em termos técnicos, o serviço possui as seguintes responsabilidades subjacentes:

- Implementar a técnica de RAG por meio de consultas à base de conhecimento do projeto, descrita em 4.4.
- Gerenciar a memória de conversas passadas do agente conversacional, conforme o tópico 4.1.1.
- Construir a consulta, também chamada de *prompt*, a ser transmitida para o agente conversacional.

### 4.1.1 Estrutura das consultas a MLL

#### Fornecimento de documentos-fontes

Apesar do GPT ser um componente fundamental para o fluxo de conversação projetado, constatou-se que esse não possui conhecimento sobre dados recentes do site do CIC ou do guia do calouro desse departamento. Isso indica que provavelmente esses dados não foram consumidos durante o pré-treinamento realizado pelos seus mantenedores. Logo, propôs-se a utilização da técnica de *retrieval-augmented generation* (RAG) para fornecer o devido contexto ao MLL e assim obter respostas satisfatórias para as consultas.

A técnica RAG determina que uma etapa importante da construção de um *prompt* para o MLL passa a ser o fornecimento de dados que embasem a resposta do modelo. Assim, foi definida uma seção do *prompt* especificamente para a disponibilização de documentos-fontes. Cada documento fornecido consiste em um texto de até 1000 caracteres, e segue a seguinte estrutura:

```
01 | /START OF DOCUMENT
02 | DOCUMENT: {page_content}
```

```
03 | URL: {url}
04 | PAGE_NUMBER: {page}
05 | /END OF DOCUMENT
```

Nesse exemplo, **page\_content**, **url** e **page** são conteúdos injetados dinamicamente, e variam para cada documento-fonte. O processo de busca, extração e injeção de dados e metadados no *template* acima é melhor detalhado em 4.5.

## Memória e janela de contexto

Por padrão, a implementação do GPT fornecida via API não possui memória. Isto é, cada consulta é realizada em um contexto de execução diferente, e não é possível referenciar consultas prévias nativamente. Isso significa que a aplicação consumidora do GPT torna-se responsável por armazenar e disponibilizar uma memória para as interações. Essa responsabilidade é designada ao **QAService**, que implementa toda a lógica necessária para o funcionamento desse recurso. Vale ressaltar que o GPT, enquanto API, possui recursos diferentes em relação à aplicação **ChatGPT**, que por sua vez possui um mecanismo proprietário de memória.

Conversas de natureza humana transcendem simples ciclos de pergunta e resposta. Enquanto os ciclos independentes frequentemente se concentram em trocas superficiais de informações, as conversas significativas buscam explorar tópicos de forma mais aprofundada. Assim, mensagens passadas de uma conversa tem grande impacto nas próximas interações a serem realizadas entre interlocutores. Por isso, o conceito de **memória** torna-se relevante em qualquer conversa, seja entre pessoas, ou entre homem e máquina, o que se revela na forma de um **histórico de mensagens**.

No contexto desse trabalho, utiliza-se a premissa de que adicionar memória ao agente conversacional é uma maneira de otimizar a assertividade de suas respostas. Ao habilitar o acesso a informações contextuais passadas, o mecanismo de memória capacita o agente a recordar nuances específicas das conversas anteriores. Assim, busca-se encontrar respostas mais relevantes e com maior grau de personalização, mediante as interações de cada usuário.

Em termos técnicos, existem diversas maneiras de implementar memória em agentes conversacionais. Este trabalho explora duas metodologias: **memória sumarizada**, e **memória em janela**. A primeira busca construir um sumário de toda a conversa, que é atualizado a cada ciclo de interações realizado entre o usuário e o agente. Assim, constitui-se um contexto de longo prazo das interações mais importantes e relevantes, com baixa especificidade nas informações. Já a memória em janela contém o contexto de curto prazo, armazenando as n-ésimas últimas interações, porém com alto grau de especificidade, já que o conteúdo das mensagens é preservado por completo.

Em maiores detalhes, a memória sumarizada é construída também pelo modelo GPT, porém em uma consulta separada. Para isso, é fornecido o último sumário conhecido, bem como os 2 últimos ciclos de interações, compostos cada um por uma mensagem do usuário e uma resposta do agente. Assim, concede-se a seguinte instrução para geração do sumário:

```
01 | Progressively summarize the lines of conversation provided, adding onto
    | the previous summary returning a new summary.
02 | The conversation happens in brazilian portuguese.
03 |
04 | INSTRUCTIONS
05 | - Always include any personal facts shared by the Human, like his name
    | or course they're enrolled in.
06 | - Include one sentence that embraces each topic of the conversation.
07 | - Break each topic into a separate sentence.
08 |
09 | Current summary:
10 | {summary}
11 |
12 | New lines of conversation:
13 | {new_lines}
```

Em suma, a instrução acima descreve concisamente o processo descrito no último parágrafo. Além disso, concede orientações para que o agente extraia cada tópico que foi conversado, e sumarie-o em uma única sentença. Ademais, busca incluir no sumário qualquer fato de cunho pessoal compartilhado pelo humano, com o objetivo de tornar a conversa mais íntima e enfatizar a criação de uma personalidade gentil e humanizada. Finalmente, contém referências a conteúdos injetados pela aplicação, sendo esses: **summary**, que representa o último sumário armazenado, e **new\_lines**, que contém os dois últimos ciclos de interações realizadas pelos humanos. Para conversas recém-iniciadas, **summary** é inicializado em branco.

Ao contrário da memória sumarizada, a **memória em janela** possui maior riqueza de detalhes nas mensagens que engloba, e consiste essencialmente na reprodução dos últimos  $n$  ciclos de interação entre usuário e agente. Com isso, busca-se garantir que o contexto das últimas mensagens não se perca, permitindo à inteligência artificial responder perguntas como "*Qual foi o tema da última pergunta*", ou "*Você pode repetir exatamente o que disse?*". Neste projeto, utilizou-se o parâmetro  $n$  como 5 ciclos de interações, isto é, 10 mensagens.

Tecnicamente, a memória em janela pode conter quantos ciclos de interação forem desejados. Contudo, à medida que mais ciclos são adicionados, espera-se um aumento consistente no número de caracteres da consulta. Assim, como consequência direta, costuma-se

observar maior lentidão na geração da resposta, bem como potencial ultrapassagem do limite de caracteres permitidos pelo modelo GPT. Isso resultaria em truncamento da consulta, e eventual perda de informação. Por isso, a utilização de múltiplos mecanismos conjugados pode ser uma boa ideia, de tal forma a maximizar os benefícios individuais.

## Perguntas de referência

De modo a padronizar os testes submetidos ao GPT, foram estabelecidas três perguntas sobre assuntos discutidos no Guia do Calouro do CIC. Durante o processo de criação do *template* de consulta definido acima, essas mesmas perguntas foram repetidas, e foi realizada uma análise subjetiva dos resultados com objetivo de verificar o impacto de cada ajuste no *template*. Essas perguntas são:

- Como o departamento pode assistir alunos para participação em eventos?
- Qual o passo-a-passo para a realização de trancamento justificado de matrícula?
- Poderia fornecer informações sobre o processo de estágio obrigatório?

Ressalta-se que a motivação para utilização das perguntas acima decorre da incapacidade do modelo GPT para respondê-las apropriadamente somente com os dados utilizados em seu treinamento. Abaixo são listados alguns dos resultados obtidos a partir dessas perguntas, pelo agente conversacional desenvolvido.

## Nova Conversa



Como o departamento pode assistir alunos para participação em eventos?



O departamento pode assistir alunos para participação em eventos de duas formas. A primeira forma é realizando pagamentos diretamente com a instituição organizadora do evento, sem a intermediação do aluno. A segunda forma é oferecendo auxílio financeiro, que pode cobrir totalmente os custos de participação ou apenas uma parcela, dependendo do valor e da disponibilidade do departamento. No caso do auxílio, o estudante recebe uma quantia em sua própria conta bancária. [Documento Fonte](#) (página 17)

Figura 4.2: Resposta do agente para a pergunta "Como o departamento pode assistir alunos para participação em eventos?".

Para a primeira pergunta, sobre o auxílio do CIC a participação em eventos, obteve-se uma resposta satisfatória, conforme a figura 4.2. Isso pode ser observado na figura 4.3, que demonstra o texto original do documento-fonte (Guia do Calouro) utilizado para

embasar a resposta. Destaca-se a correta inclusão de um link para a referência, bem como a indicação da página que contém o conteúdo, conforme o requisito REQ-2.2, que trata da citação de arquivos-fontes.



## 2.4 APOIO PARA PARTICIPAÇÃO EM EVENTOS

Para eventos que requerem gastos financeiros, o CIC possui duas formas de assistir alunos que desejam participar: pagamentos de inscrições e auxílios.

Quanto à primeira modalidade, a UnB realiza pagamentos diretamente com a instituição organizadora do evento, sem intermediação do aluno.

Já quanto ao auxílio, o estudante recebe uma quantia – que pode cobrir totalmente os custos de participação ou apenas uma parcela, a depender do valor e da disponibilidade do departamento – em sua própria conta bancária.

Assim, o aluno é o responsável por efetuar o pagamento, incluindo taxas e impostos.

17

Figura 4.3: Referência apresentada na resposta do agente para a pergunta "Como o departamento pode assistir alunos para participação em eventos?".

Para a segunda pergunta, o agente também demonstrou resultado satisfatório. Observe na figura 4.4 a resposta realizada pelo agente. Essa condiz com o conteúdo textual das figuras 4.5 e 4.6, também retiradas do Guia do Calouro.



Contudo, ao observar-se a listagem de código 4.1, o último documento referencia a página 14. Essa página no Guia do Calouro discorre, também em um formato passo-a-passo, sobre o aproveitamento de estudos. Embora o formato do documento-fonte e os passos sejam similares ao trancamento, pode ser considerado um documento incluído erroneamente, já que se refere a outro tema. Isso indica uma potencial necessidade de refinamento no processo de busca de documentos-fontes, detalhado na seção 4.5.

```
01 | /START OF DOCUMENT
02 | DOCUMENT: 20
03 | Duvidas?
04 | Apos a leitura deste manual, caso ainda tenha
05 | duvidas em peticionar seu Trancamento Justificado,
06 | contate a Coordenacao de Orientacao e Supervisao
07 | das Unidades Avancadas, da Secretaria de
08 | Administracao Academica -SAA/COS, por meio do
09 | saaatendimento@unb.br
10 | Lembre-se de anexar um documento comprobatório: atestado de
11 | medico ou psicologico, declaracao de servico militar, atestado de
12 | requisicao judicial, declaracao de intercambio, entre outros.7 PASSO
    | Concluir peticionamento
13 | Consultar recibo e acompanhamento
14 | de processo 8 PASSO
15 | Atencao: o trancamento
16 | justificado pode ser
17 | protocolado ate o final do
18 | semestre letivo.
19 | URL: https://ehcvoenjkdxcazeyjzsr.supabase.co/storage/v1/object/sign/cic
    | -public-documents/guia_cic.pdf
20 | PAGE_NUMBER: 20.0
21 | /END OF DOCUMENT
22 |
23 |
24 |
25 | /START OF DOCUMENT
26 | DOCUMENT: 19
27 | 2.5 TRANCAMENTO
28 | Suspensao de atividades academicas de todo um semestre ou
29 | de uma disciplina, por motivo de impedimento involuntario.Trancamento
    | justificado
30 | 1 PASSO
31 | 2 PASSO
32 | 3 PASSO
33 | 5 PASSOEntrar no Sistema Eletronico de Informacoes
34 | (SEI)
35 | Iniciar novo processo (peticionamento)
```

```

36 | Escolher o tipo de processo
37 | 4 PASSO Formulario de peticaoamento
38 | Formulario de solicitacao
39 | 6 PASSO Verificar o codigo de opcao de curso
40 | URL: https://ehcvoenjkdxcaxeyjzsr.supabase.co/storage/v1/object/sign/cic
    | -public-documents/guia_cic.pdf
41 | PAGE_NUMBER: 19.0
42 | /END OF DOCUMENT
43 |
44 |
45 |
46 | /START OF DOCUMENT
47 | DOCUMENT: 14
48 | Em caso de Revisao, Recurso ou Adaptacao de Estudos, consultar
49 | procedimento no Manual de Aproveitamento do Estudos de Outras
50 | Instituicoes de Ensino Superior. 7 PASSO Incluir documentacao
    | comprobatória
51 | 8 PASSO
52 | 9 PASSO Inserir especificacao do documento
53 | Concluir peticaoamento
54 | Consultar recibo e acompanhamento
55 | de processo 10 PASSO6 PASSO Verificar o codigo de opcao de curso
56 | Manual5 PASSO Formulario de solicitacao
57 | URL: https://ehcvoenjkdxcaxeyjzsr.supabase.co/storage/v1/object/sign/cic
    | -public-documents/guia_cic.pdf
58 | PAGE_NUMBER: 14.0
59 | /END OF DOCUMENT

```

Listing 4.1: Listagem de código que indica os documentos utilizados na consulta ao agente conversacional para a segunda pergunta de referência. Este conteúdo é gerado dinamicamente conforme o processo de *retrieval-augmented generation*.

# Nova Conversa



Qual o passo-a-passo para a realização de trancamento justificado de matrícula



Olá! Para realizar o trancamento justificado de matrícula na Universidade de Brasília, você precisa seguir os seguintes passos:

1. Acesse o Sistema Eletrônico de Informações (SEI).
2. Inicie um novo processo de peticionamento.
3. Escolha o tipo de processo relacionado ao trancamento justificado.
4. Preencha o formulário de solicitação.
5. Verifique o código de opção de curso.
6. Inclua a documentação comprobatória necessária, como atestado médico ou psicológico, declaração de serviço militar, entre outros.
7. Conclua o peticionamento.
8. Consulte o recibo e o acompanhamento do processo.

Lembre-se de que o trancamento justificado pode ser protocolado até o final do semestre letivo. Caso tenha mais dúvidas, você pode entrar em contato com a Coordenação de Orientação e Supervisão das Unidades Avançadas, da Secretaria de Administração Acadêmica, por meio do [saaatendimento@unb.br](mailto:saaatendimento@unb.br).

Aqui está o documento que pode te ajudar com mais informações:

- [Documento Fonte](#) (página 19)

Espero ter ajudado! Se tiver mais alguma pergunta, é só me avisar. 😊

Figura 4.4: Resposta do agente para a primeira pergunta de referência

## 2.5 TRANCAMENTO

### Trancamento justificado

Suspensão de atividades acadêmicas de todo um semestre ou de uma disciplina, por motivo de impedimento involuntário.

1º PASSO

Entrar no Sistema Eletrônico de Informações (SEI)

2º PASSO

Iniciar novo processo (peticionamento)

3º PASSO

Escolher o tipo de processo

4º PASSO

Formulário de peticionamento

5º PASSO

Formulário de solicitação

6º PASSO

Verificar o código de opção de curso

Figura 4.5: Página 19 do Guia do Calouro, que indica o documento-fonte utilizado para a segunda pergunta de referência.

7º PASSO

Concluir peticionamento

8º PASSO

Consultar recibo e acompanhamento de processo

## Dúvidas?

Após a leitura deste manual, caso ainda tenha dúvidas em peticionar seu Trancamento Justificado, contate a Coordenação de Orientação e Supervisão das Unidades Avançadas, da [Secretaria de Administração Acadêmica -SAA/COS](#), por meio do [saatendimento@unb.br](mailto:saatendimento@unb.br)

*Lembre-se de anexar um documento comprobatório: atestado de médico ou psicológico, declaração de serviço militar, atestado de requisição judicial, declaração de intercâmbio, entre outros.*

Atenção: o trancamento justificado pode ser protocolado até o final do semestre letivo.

Figura 4.6: Página 20 do Guia do Calouro, que continua o texto sobre trancamento.

Finalmente, a figura 4.7 demonstra o resultado obtido para a terceira pergunta, sobre o processo de estágio obrigatório, que também foi satisfatório.



Gostaria de mais informações sobre o processo de estágio obrigatório.



Olá! Claro, vou te explicar mais sobre o processo de estágio obrigatório na Universidade de Brasília (UnB). O estágio obrigatório faz parte da grade curricular do curso de Engenharia da Computação. Para se matricular nas disciplinas Estágio Supervisionado em Engenharia da Computação 1 e 2, você precisa atender às seguintes condições:

- Ter completado o mínimo de 70 créditos obrigatórios;
- Estar cursando a partir do 6º período da graduação;
- Ter apresentado os documentos de estágio (TCE e Plano de Atividades) detalhados e assinados;
- Estagiar por 150 horas durante o semestre.

Ao final do semestre, você precisará encaminhar o Formulário de Avaliação de Estágio preenchido e assinado pelo(a) professor(a) orientador(a) e pelo(a) supervisor(a) de estágio no Formulário de Atendimento. O envio desse formulário é requisito para a atribuição de menção nas disciplinas.

Se você quiser mais informações sobre o estágio obrigatório, você pode consultar o [Documento Fonte](#) (página 25).

Espero que essas informações sejam úteis! Se você tiver mais alguma pergunta, é só me avisar. 😊

Figura 4.7: Resposta o agente para pergunta sobre estágio obrigatório.

## 4.2 AuthService

Em observância ao REQ-1, foi integrado um módulo para autenticação de usuários no sistema. Esse módulo utiliza a metodologia **json web token** (JWT), e é hospedado externamente na ferramenta de código aberto **Supabase** [24].

A autenticação de usuário é um processo altamente difundido em diversas aplicações, e existem diferentes metodologias e soluções de terceiros para facilitar a adoção dessa funcionalidade. Assim, a escolha da plataforma Supabase é embasada por motivos claros e específicos.

Primeiramente, o serviço de autenticação provido pela plataforma está em conformidade com os requisitos formais da RFC 7519 [25], protocolo que formaliza o JWT, bem como provê interfaces simples para gerenciamento das sessões de usuário. Além disso, o Supabase também suporta outros tipos de autenticação, como OAuth, que podem ser futuramente integrados a este projeto, o que favorece a extensibilidade do sistema. Finalmente, essa plataforma também possui outros serviços úteis aos demais módulos aqui apresentados, como gerenciamento de banco de dados e hospedagem de arquivos.

Na implementação realizada, o **AuthService** tem duas principais funcionalidades: gerar e decodificar *tokens* JWT. Ao requisitar a sua autenticação no sistema, o usuário deve fornecer suas credenciais, que conforme o requisito REQ-1.1, são seu email e senha. Essas são então comparadas com os dados salvos no banco de dados relacional, na tabela de usuários. Confirmada a validade das credenciais, é gerado um *token* JWT que contém o ID codificado deste usuário. Finalmente, esse *token* é retornado à aplicação-cliente, operada pelo usuário, e pode ser concedido para autenticar requisições futuras.

O processo de decodificação, por sua vez, é iniciado toda vez que uma requisição do usuário contém um *token* JWT. Esse é então decodificado, e assim obtém-se acesso à carga de dados que fora codificada, que nesse caso é composta pelo ID do usuário. Assim, esse identificador único de usuário é fornecido ao contexto de execução da aplicação, e é usado pelos demais serviços para determinar o escopo dos dados e permissões às quais o usuário têm acesso.

### Geração de JWT

Conforme a visão geral do processo de geração de JWT, essa seção melhor define o algoritmo de codificação implementado por esse meio de autenticação.

O *token* JWT é composto por três elementos principais: um cabeçalho, uma carga de dados, e uma assinatura, descritos abaixo:

1. **Cabeçalho:** normalmente contém o tipo de *token* utilizado, que no caso é o JWT propriamente dito, e o algoritmo utilizado para assiná-lo.

Toma-se como exemplo o algoritmo HMAC SHA256. Assim, o cabeçalho pode ser descrito como um objeto JSON no seguinte formato:

```
01 | {
02 |   "alg": "HS256",
03 |   "typ": "JWT"
04 | }
```

Em seguida, realiza-se a codificação do JSON em formato Base64, obtendo-se o seguinte resultado:

```
01 | eyJhbGciOiAiSFMyNTYiLCJkaWVzIjoiKpXVCJ9
```

2. **Carga de dados:** esse valor é arbitrariamente definido pela aplicação, e pode ser usado em diferentes contextos. Como esse projeto utiliza o JWT para caráter de autenticação, definiu-se que a carga de dados conterà o identificador único de usuário. Assim, considera-se um UUID arbitrário e obtém-se a seguinte carga de dados, em formato JSON:

```
01 | {
02 |   "id": "550e8400-e29b-41d4-a716-446655440000"
03 | }
```

Analogamente ao cabeçalho, esse objeto também é codificado em formato Base64, e resulta em:

```
01 | eyJpZCI6IjU1MGU4NDAwLWUyOWItdFkNC1hNzE2LTQ0NjY1NTQ0
02 | MDAwMCJ9
```

3. **Assinatura:** consiste da aplicação do algoritmo especificado para assinatura do cabeçalho e carga de dados, separados por ponto, sob a utilização de uma chave secreta.

Considera-se o algoritmo **HMAC SHA256**, por exemplo. Também considera-se:

$$content = base64(cabeçalho) + "." + base64(carga\_de\_dados)$$

Então, a assinatura pode ser representada por:

$$HMACSHA256(content, chave\_secreta) \quad (4.1)$$

Tomando-se o cabeçalho e carga de dados do exemplo acima, e a chave secreta **segredo\_secreto**, a assinatura do *token* JWT seria:

```
01 | 8tkhSxLJPtL8BicqJsXSM-_dFxmYVeVcVv1974x8YgQ
```



Descritas as três partes que o compõem, o *token* final pode ser obtido a partir da concatenação do cabeçalho, carga de dados e assinatura, respectivamente, separados por um ponto. O *token* resultante do exemplo acima pode então ser descrito como:

```
01 | eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjU1MGU4
02 | NDAwLWUyOWItNDkNC1hNzE2LTQ0NjY1NTQ0MDAwMCJ9.8tkh
03 | SxLJPTl8BicqJsXSM-_dFxmYVeVcVv1974x8YgQ
```

A Figura 4.8 abaixo ilustra uma demonstração do processo de codificação do JWT descrito acima por meio de um sistema interativo. Esse pode ser acessado no *website JWT.io*.

The image shows a screenshot of the JWT.io website interface. It is divided into two main sections: 'Encoded' and 'Decoded'.

- Encoded:** Labeled 'PASTE A TOKEN HERE'. It contains a text area with the following JWT token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjU1MGU4NDAwLWUyOWItNDkNC1hNzE2LTQ0NjY1NTQ0MDAwMCJ9.8tkhSxLJPTl8BicqJsXSM-_dFxmYVeVcVv1974x8YgQ`. The token is color-coded: header in purple, payload in blue, and signature in cyan.
- Decoded:** Labeled 'EDIT THE PAYLOAD AND SECRET'. It displays the decoded components of the token:
  - HEADER: ALGORITHM & TOKEN TYPE:** `{ "alg": "HS256", "typ": "JWT" }`
  - PAYLOAD: DATA:** `{ "id": "550e8400-e29b-41d4-a716-446655440000" }`
  - VERIFY SIGNATURE:** Shows the HMACSHA256 function being applied to the header and payload, with a text input field containing 'segredo\_secreto' and a checkbox for 'secret base64 encoded'.

Figura 4.8: Ilustração do processo de codificação de *token* JWT. Acessado em 17/11/2023.

## Decodificação do JWT

Após a geração de um JWT, este costuma ser servido à aplicação-cliente requisitante, e pode ser posteriormente utilizado para autenticar determinadas ações. Para isso, é necessário decodificar o *token* fornecido em uma requisição, e realizar uma verificação de integridade.

Primeiramente, é necessário decodificar o cabeçalho, representado em *Base64*, com o intuito de definir-se o algoritmo de assinatura empregado. Assim, extrai-se a sub-cadeia de caracteres que compreende o primeiro caracter do *token*, até o primeiro "." encontrado. Ressalta-se que o ponto é apenas um delimitador e não deve ser incluído na sub-cadeia. Ainda considerando o exemplo acima, para geração de *token*, a decodificação da representação *Base64* do cabeçalho exporia o algoritmo *HMAC SHA256*.

Uma vez que o algoritmo de assinatura seja obtido, reaplica-se a equação 4.1 e compara-se a assinatura obtida com aquela fornecida no próprio *token*, composta pela sub-cadeia de caracteres que inicia após o segundo ponto até o final da cadeia. Caso as assinaturas coincidam, atesta-se a integridade do *token* fornecido. Assim, a carga de dados pode ser decodificada e utilizada para qualquer fim proposto. Nesta aplicação, o identificador de usuário contido na carga é usado para fins de filtragem de dados.

### 4.3 Bancos de dados

Conforme abordado no requisito REQ-3, um recurso fundamental da aplicação proposta é a capacidade de persistir e recuperar dados de usuários e suas respectivas sessões de conversação. Assim, esta seção aborda a solução de bancos de dados explorada para a satisfação desse requisito, e visa embasar as decisões arquiteturais para armazenamento de dados no sistema.

A escolha de banco de dados para esta aplicação foi embasada a partir da discussão de diferentes tecnologias de bancos de dados e suas particularidades. Assim, analisou-se distintos cenários e características.

O primeiro fator analisado foi o grau de estruturação dos dados. Conforme o requisito REQ-3.1, o sistema conta com três entidades principais, sendo essas **usuários**, **conversas** e **mensagens**. Além disso, tais entidades são associadas de acordo com os mesmos graus de cardinalidade, segundo o REQ-3.2, aonde um único **usuário** pode conter zero, uma ou múltiplas **sessões de conversação**, que por sua vez possuem a mesma cardinalidade em relação às **mensagens**. A identificação deste padrão durante o estabelecimento de requisitos é importante, visto que quando existe uma estrutura pré-definida e a identificação prévia dos relacionamentos entre entidades, favorece-se o uso de um modelo relacional de dados.

Ainda no âmbito do requisito REQ-3, destaca-se a garantia da integridade referencial como um critério significativo na escolha do banco de dados. Isso ocorre devido à clara hierarquia das entidades, onde a validade de todos os relacionamentos é crucial para aumentar a confiabilidade das operações. Essa garantia assegura a associação correta entre os dados dos usuários, suas conversas e as mensagens que as compõem.

Outra preocupação considerada na escolha foi a escalabilidade. Embora possa ser um fator limitante em bancos de dados relacionais quando comparados com sistemas NoSQL, no contexto apresentado, a escalabilidade não é um problema inicial. Isso porque a capacidade e desempenho dos bancos de dados relacionais devem ser suficientes para lidar com a carga esperada em fase inicial. Futuramente, também considera-se a possibili-

dade de migração para estratégias híbridas que envolvam soluções relacionais e NoSQL combinadas, porém isso não foi considerado relevante para o presente momento.

Assim, ao considerar os aspectos supracitados, fica claro que um banco de dados relacional se alinha com os requisitos e as características arquiteturais do sistema proposto. Essa tecnologia oferece uma combinação robusta de integridade de dados, suporte maduro, e conformidade com transações ACID, que tratam sobre isolamento, consistência, atomicidade e durabilidade de dados e transações, tornando esta uma escolha sólida para fundamentar as operações deste projeto. Logo, a partir dessa escolha, constitui-se o seguinte modelo de dados, que relaciona as entidades detectadas e seus relacionamentos.

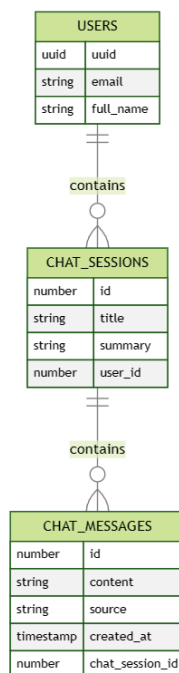


Figura 4.9: Diagrama de entidades e relacionamentos.

## 4.4 Armazenamento de documentos

O Supabase Storage [26] é um componente do Supabase. Esse serviço é especialmente dedicado ao armazenamento de arquivos na nuvem. A plataforma utiliza tecnologias renomadas, como o Amazon S3 e o Backblaze, para garantir um serviço robusto e confiável.

Ao integrar o Amazon S3, por exemplo, o Supabase Storage aproveita a escalabilidade e a durabilidade excepcionais desse serviço de armazenamento de objetos. Isso permite a persistência de uma grande quantidade de dados de maneira eficiente e segura, com

suporte a vários tipos de arquivos, e aproveitando a infraestrutura poderosa da Amazon. Isso torna-se um importante ponto que embasa a escolha do Supabase Storage para este trabalho.

O Supabase Storage pode ser operado tanto via API, quanto via interface de usuário. No âmbito deste trabalho, o acesso via API foi um fator fundamental, visto que diversos dados foram extraídos por meio de *scrapping*, em especial as informações do portal de notícias do CIC, e armazenados neste serviço por meio de *scripts* automáticos. Além disso, como evidenciado na seção seguinte (4.5), este sistema realiza um processo de vetorização nos documentos extraídos, o que é realizado por meio de uma rotina de código, que por sua vez depende do acesso de leitura via rede para carregar os arquivos em memória.

Diante das características acima, destaca-se a capacidade do Supabase Storage para atuar como base de conhecimento unificada, conforme proposto nos objetivos deste trabalho. Em suma, escolheu-se essa tecnologia devido à versatilidade nas formas de acesso, seja via API ou interface de usuário, o suporte a diversos formatos de arquivos, os mecanismos de segurança para acesso dos dados e o alto índice de disponibilidade do sistema.

## 4.5 Banco de dados vetorial

Além de MLL, outro tipo de modelo mostra-se útil para a implementação da técnica de RAG em aplicações de pergunta e resposta baseadas em IA. Tratam-se de modelos voltados para a transformação de palavras em vetores. Alguns exemplos notórios de modelos para esse intuito são **Word2Vec** [27] e **ADA-002** [28].

A premissa base para utilização desses é de que palavras podem ser representadas por vetores em um espaço de alta dimensão, e quanto menor a distância entre dados vetores, maior a correlação semântica entre as palavras que esses representam. Logo, é possível usar artifícios da álgebra linear para metrificar a similaridade do significado de palavras ou sentenças a partir de suas representações vetoriais.

No âmbito deste projeto, é sabido que os modelos de MLL propostos não possuem um desempenho satisfatório para a resposta de perguntas dentro do contexto estabelecido. Um exemplo é descrito na figura 4.1, que demonstra a incapacidade do modelo de responder uma das perguntas formuladas para validação do projeto, devido a falta de treinamento com o documento-fonte em questão.

Exposto o problema acima como motivação, considera-se que uma dada pergunta do usuário pode ser comparada com textos-fontes a partir da transformação de ambos em vetores. Logo, estabelece-se a hipótese de que tal comparação matemática permite a identificação dos documentos de maior relevância para a resposta de uma pergunta.

Assim, ao fornecê-los ao MLL por meio da técnica de RAG, os resultados tornar-se-ão mais satisfatórios.

Uma vez que o cálculo da distância entre vetores pode ser útil para o cumprimento dos objetivos propostos, faz-se necessário buscar artifícios computacionais que viabilizem este processo. Isso porque a vetorização pode se tornar dispendiosa, tanto em termos de custos quanto tempo de processamento, à medida que as coleções de documentos-fontes crescem. Assim, os bancos de dados vetoriais passam a ser uma alternativa para a persistência dos vetores, ao passo que evitam a necessidade de geração contínua de documentos vetorizados a cada consulta do usuário.

No contexto deste projeto, foi utilizado o banco de dados vetorial **Pinecone** [29], também referido aqui como **VectorDB**. Trata-se de uma ferramenta que têm recebido grande atenção por parte da comunidade de desenvolvedores na área da IA. Dentre as principais funcionalidades desse banco de dados, destacam-se: 1) integração facilitada com o modelo ADA-002, escolhido como o mecanismo de vetorização para este projeto, 2) suporte nativo a consulta de similaridade por meio de distância euclidiana (L2), produto interno, e distância entre cossenos, e 3) motor de busca que permite consultas complexas baseadas não só na similaridade entre vetores, mas também em diferentes filtros de metadados.

Feitas as considerações acima, a figura 4.10 ilustra o processo de processamento e vetorização de documentos que integram a base de conhecimento do projeto. Dessa maneira, é possível visualizar o papel do banco de dados vetorial no processo de consultas baseadas em RAG, bem como observar a integração entre alguns módulos do sistema em maiores detalhes.

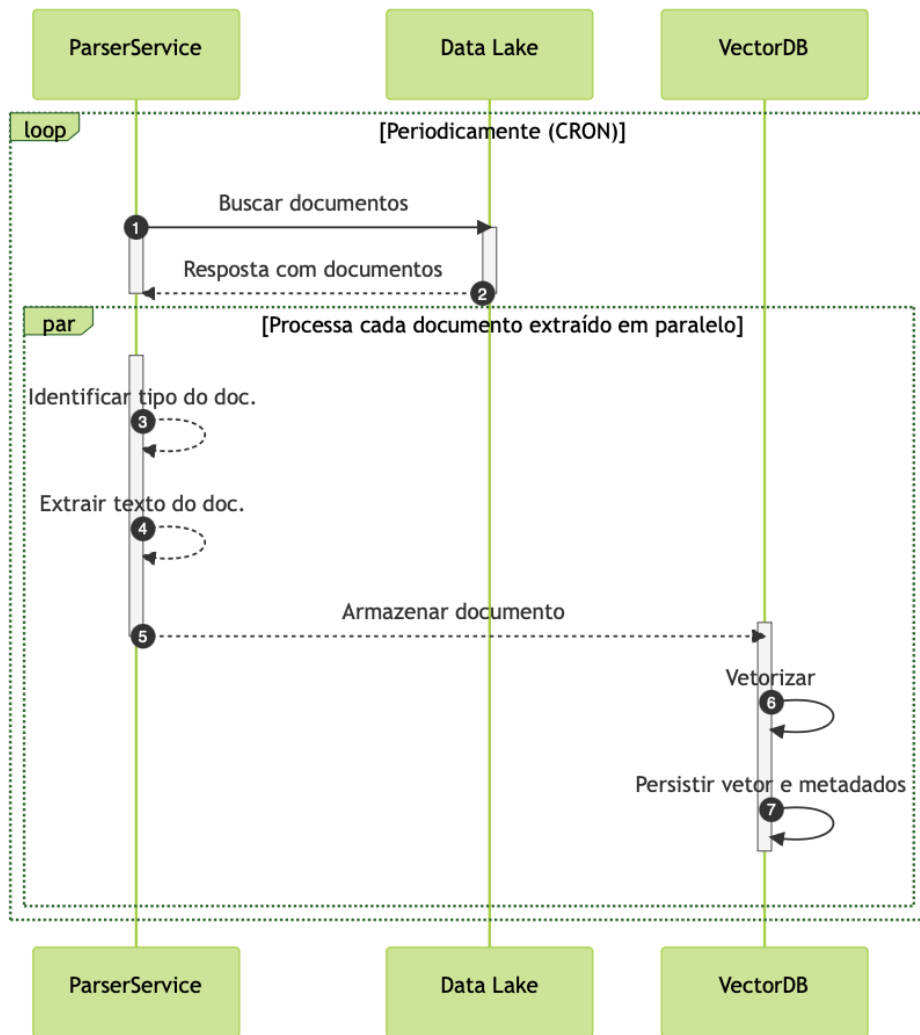


Figura 4.10: Fluxo de busca, extração e vetorização de documentos.

O diagrama acima mostra os passos que antecedem a escrita de dados no banco de dados vetorial. Esse fluxo pode ser dividido em três etapas principais: 1) busca de documentos no sistema de armazenamento de arquivos (passos 1 e 2), 2) extração de texto dos documentos (*parsing*) (passos 3 e 4), e 3) vetorização e persistência de vetores em um banco vetorial (**VectorDB**), conforme os passos 5, 6 e 7. Na imagem acima, ressalta-se que **ParserService** refere-se a um serviço de menor relevância, implementado no contexto do *script* ilustrado na imagem.

A primeira etapa é realizada por meio de chamadas HTTP ao serviço de armazenamento de documentos, também chamado de *data lake*, descrito em 4.4. Como resultado, uma coleção composta por documentos em formatos PDF e HTML é carregada na memória da aplicação. Devido aos documentos possuírem estruturas diferentes, o processo de extração de textos desses é feito por meio de serviços de processamento distintos. São

esses as bibliotecas **PyPDF** para arquivos PDF, e **Unstructured** para arquivos HTML. A função de identificar o tipo e o tipo de extrator de texto apropriado é delegada ao módulo **ParserService**.

Após a extração dos textos de documentos-fontes, obtém-se uma coleção de *strings* carregadas na memória da aplicação. Esse estado é alcançado logo após o passo de número 4. Em geral, as representações de texto resultantes possuem tamanhos variáveis, sejam curtos ou longos. Contudo, visto que o modelo escolhido, GPT, possui um limite para a quantidade de *tokens* em cada consulta, faz-se necessário dividir tais textos em tamanhos menores, de forma a obter maior previsibilidade na utilização de *tokens*, e evitar que cadeias de texto sejam truncadas.

Para sanar essa dificuldade, aplicou-se a técnica de *chunking*, de modo a reduzir as *strings* a blocos de texto com tamanhos similares. Neste projeto, especificamente, utilizaram-se *chunks* com 1000 caracteres, sendo os últimos 200 compartilhados entre *chunks* vizinhos. O compartilhamento de parte do texto é uma técnica conhecida como *overlapping*, e evita que o contexto de um bloco de texto seja perdido próximo ao limite deste *chunk*. Finalmente, os *chunks* resultantes são submetidos ao banco vetorial, que aplica um algoritmo de conversão de textos em vetores, e realiza a persistência desses. Vale ressaltar que além dos vetores, também são armazenados metadados como nome do documento-fonte, URL, caso disponível, e número da página do texto no arquivo original. Esses metadados são fundamentais para que, durante a aplicação da técnica de RAG em uma consulta, a busca de documentos relevantes retorne as fontes, de tal modo a satisfazer o REQ-2.3 (seção 3.1.2).

## 4.6 Camada de API

Em um sistema projetado para a *web*, é comum haver a separação da camada de código responsável pela geração de interfaces de usuário (UI), muitas vezes referida como *front-end*, da camada de código que executa as principais regras de negócio e lida intensamente com busca e processamento de dados, também conhecida como *back-end*. Um modelo semelhante é implementado neste projeto, no qual a maior parte das cargas de trabalho são executadas em uma **aplicação-servidor** (*back-end*), fora da máquina do cliente, enquanto a UI é gerenciada pela **aplicação-cliente** (*front-end*), melhor descrita em 4.7. Alguns exemplos de cargas de trabalho realizadas no *back-end* são: consultas ao banco de dados, geração de resposta automática pelo *chatbot*, bem como coleta e vetorização de documentos da base de conhecimento.

Diversos dados são produzidos durante a operação de sistemas, e frequentemente faz-se necessário realizar seu consumo e exibição na aplicação-cliente. Porém, a separação

do sistema em duas camadas introduz um novo problema, que é a comunicação entre cliente e servidor, que agora são executados em máquinas distintas, e possivelmente utilizando diferentes linguagens de programação. Assim, é necessário implantar uma camada intermediária que possibilite a comunicação entre ambos serviços.

Uma solução para comunicação cliente-servidor é a adoção de uma camada de **interface de programação de aplicações** (API). Essa define um conjunto de regras e convenções que permitem a interoperabilidade e comunicação entre aplicações distintas. No âmbito deste trabalho, a API define a utilização do protocolo HTTP para transmissão de conteúdo pela rede, bem como o formato JSON para estruturação dos dados transferidos, e assim viabiliza o consumo de dados produzidos na aplicação-servidor pela aplicação-cliente, ou vice-versa. Agora, isso passa a ser possível mesmo quando essas são executadas em endereços de rede distintos, ou utilizam arcabouços de código diferentes.

Existem diversas bibliotecas e *frameworks* que facilitam a implementação de APIs, independentemente da linguagem de programação escolhida. Alguns são notórios pela robustez, vasta gama de funcionalidades, e suporte nativo a diferentes formatos de dados como XML e JSON. Um exemplo é o framework **Ruby on Rails**. Ao mesmo tempo, outros são conhecidos pela simplicidade e performance, de modo a oferecer uma abordagem mais minimalista. Um exemplo é o *framework* **Express**, para linguagem JavaScript.

Diversas tecnologias foram consideradas para a camada de API desse projeto, e a escolha final foi a biblioteca **FastAPI** [30]. Esse é um *framework* moderno, com foco em alto desempenho e velocidade de desenvolvimento, além de conformidade com o padrão OpenAPI [31], amplamente aceito pela comunidade de desenvolvedores. Além disso, um motivo determinante é a utilização de Python como linguagem de programação base para execução do *framework*. Isso porque essa linguagem oferece um extenso suporte a aplicações centradas em dados e inteligência artificial, tornando-a compatível com o módulo **QAService**, descrito em 4.1, que pode ser considerado o núcleo deste trabalho.

### 4.6.1 Rotas

Diferentes rotas HTTP foram projetadas para permitir tanto leitura quanto escrita de dados por meio da camada de API. Isso significa que a API oferece diferentes pontos de acesso que podem ser requisitados por meio da internet. Esses são unicamente identificados por meio de um endereço específico, bem como um método do protocolo HTTP, que no âmbito deste trabalho pode ser do tipo **GET** ou **POST**.

Além disso, a API desenvolvida estabelece que todas as rotas implementadas requeiram a passagem de um cabeçalho de autorização, conhecido como *Authorization*, que deve conter obrigatoriamente o JWT que autentica uma sessão de usuário. Conforme descrito na seção 4.2, o *token* utilizado neste sistema possui dados codificados que incluem o iden-



ificador único de um usuário. Isso é de suma importância para determinar-se o escopo dos dados aos quais um usuário requisitante possui acesso. Assim, evita-se a exposição de dados de um utilizador aos demais, visto que sessões de conversa normalmente tem um cunho particular. Em outras palavras, a autenticação não somente restringe a utilização da API para clientes selecionados, mas também identifica o escopo dos dados a serem enviados em requisições HTTP.

Conforme a especificação do protocolo HTTP, o método GET é utilizado para leitura de dados a partir da aplicação requisitante, ou seja, a aplicação-cliente. Assim, optou-se por padronizar a nomenclatura desses métodos seguindo o padrão abaixo:

- Para requisitar-se coleções de recursos, é utilizado o nome do recurso no plural, que deve coincidir com o nome da tabela do banco de dados que o persiste. A título de exemplo, para obter-se todas as sessões de conversação de um usuário, utiliza-se a rota **/chat\_sessions**.
- Para requisitar-se um recurso específico, utiliza-se o nome do recurso no plural, seguido pelo identificador único do recurso. Em linhas gerais, a rota observa o seguinte padrão **/nome\_do\_recurso/<id>**. Ou seja, para obter-se a sessão de conversação com ID 5, a rota **/chat\_sessions/5** deve ser utilizada. Ressalta-se que na aplicação desenvolvida não há exemplos tangíveis dessa convenção. Porém, em um cenário de um sistema que é continuamente desenvolvido, existe uma alta probabilidade de utilização dessa regra.
- Existe outra convenção que determina o aninhamento de recursos. Essa regra permite a especificação o escopo dos dados a serem inseridos por meio da própria URL. A estrutura de rotas aninhadas segue o formato **/recurso\_pai/<id>/recurso\_novo**.

Um exemplo é a recuperação de mensagens de chat. Para os casos de uso aqui estabelecidos, essas sempre são recuperadas sob o contexto de sua sessão de conversação, que atua como recurso-pai. Logo, para recuperação de todas as mensagens vinculadas à sessão de conversação de ID 5, utiliza-se **/chat\_sessions/5/chat\_messages**.

Já para as rotas do tipo POST, utiliza-se uma convenção ligeiramente diferente, seguindo os critérios a seguir:

- Usualmente, também utiliza-se o nome do recurso a ser criado no plural. Assim, fornece-se os dados a serem inseridos no banco de dados em formato JSON, no corpo da requisição. Por exemplo, para criar-se uma nova sessão de conversação, utiliza-se a rota **/chat\_sessions**.

- Também é possível utilizar rotas aninhadas, assim como nas requisições GET, seguindo a mesma estrutura.
- Uma exceção é a rota de autenticação de usuários. Como os *tokens* não são persistidos, e portanto não se vinculam a qualquer recurso do banco de dados, utiliza-se um nome arbitrário para sua criação, fora da convenção estabelecida. No caso, a rota é denominada **/sign\_in**.

Detalhados os diferentes métodos HTTP e seus padrões de utilização no âmbito deste projeto, a camada de API proposta implementa as seguintes rotas:

Método HTTP	Rota
GET	/chat_sessions
GET	/chat_sessions/<id>/chat_messages
POST	/chat_sessions
POST	/chat_sessions/<id>/chat_messages
POST	/users
POST	/sign_in

Tabela 4.1: Rotas implementadas para a aplicação

### GET /chat\_sessions

Em suma, essa rota retorna todas as sessões de conversação vinculadas ao usuário solicitante, bem como todas as mensagens pertencentes a cada sessão. Portanto, realiza-se uma consulta SQL com a seguinte estrutura:

```

01 | SELECT
02 |     chat_sessions.*
03 | FROM
04 |     chat_sessions
05 | WHERE
06 |     chat_sessions.user_id = <id do usuario>

```

Listing 4.2: Consulta de recuperação de sessões de conversação

O retorno dessa consulta é convertido em uma lista de dicionários da linguagem Python, utilizando um adaptador externo, na qual cada dicionário representa uma linha de dados específica. Em seguida é codificada como JSON e retornada como resposta da requisição.

## GET /chat\_sessions/<id>/chat\_messages

Essa rota habilita a recuperação de todas as mensagens de chat correspondentes a uma determinada sessão de conversação, identificada como <id>. Inicialmente, realiza-se uma consulta similar a 4.2. O motivo é determinar se o usuário requisitante possui acesso ao recurso solicitado. Caso não haja nenhuma sessão retornada, isso é um indicativo de que o recurso solicitado não existe, ou que o usuário requisitou um recurso não-autorizado. Nesse sentido, a requisição falha com um estado de erro 404.

Quando a requisição acima é bem sucedida, entende-se que a sessão de conversação solicitada é válida. Portanto, prossegue-se para realização da seguinte consulta, com o objetivo de recuperar as mensagens de chat vinculadas:

```
01 | SELECT
02 |     chat_messages.*
03 | FROM
04 |     chat_messages
05 | WHERE
06 |     chat_messages.chat_session_id = <id>
07 | ORDER BY
08 |     chat_messages.created_at
```

Listing 4.3: Consulta de recuperação de mensagens a partir de uma sessão de conversação

Analogamente à requisição de sessões de conversação, a lista de objetos retornada é processada para uma estrutura de dados de lista compatível com a linguagem Python, e em seguida retornada como resposta à requisição em formato JSON.

## POST /chat\_sessions

Essa rota é utilizada para criar uma nova sessão de conversação entre o usuário e o agente. A chamada HTTP para essa rota depende de uma ação ativa por parte do usuário, que é o clique no botão "Nova conversa" (figura 4.14). Assim, o servidor cria e persiste uma nova sessão de conversação. Além disso, é feita uma solicitação ao módulo **QAService** para que uma mensagem de boas vindas e iniciação da conversa seja gerada.

O conceito de sessão de conversação é importante, sob uma perspectiva de persistência de dados, por um principal motivo: permitir a criação de diferentes janelas de contexto entre o usuário e o agente. Conforme uma conversa é redigida, constitui-se uma memória dos tópicos conversados, e esse recurso é alimentado ao agente para que este recorde as perguntas e respostas realizadas, bem como outros tópicos pertinentes. Isso garante uma maior fluidez à conversa, visto que o conteúdo de mensagens prévias pode ser interpretado pelo agente para responder perguntas futuras. Um exemplo clássico é realizar a pergunta "Qual é o meu nome?" a um agente conversacional. Caso este não se recorde

do nome previamente fornecido, isso indica uma potencial ausência de memória em sua implementação.

Contudo, conforme a memória da conversa cresce, espera-se que mais caracteres sejam utilizados ao informá-la durante uma consulta ao agente, e que portanto haja um aumento no tempo de processamento de cada consulta. Espera-se também que haja maior ruído conversacional na memória, isto é, tópicos que não sejam de relevância para a geração de uma resposta. Assim, utilizar uma única memória para todas as mensagens do usuário e do agente torna-se ineficiente. Portanto, uma solução explorada neste trabalho é a implementação do suporte a múltiplas sessões de conversação, comparáveis a conversas de aplicativos de chat via SMS ou WhatsApp, por exemplo, onde cada sessão de conversação possui uma memória independente.

Para concretizar o assunto acima, considera-se o seguinte exemplo: o usuário Pedro inicia uma sessão de conversação com o agente. Nessa sessão, Pedro expõe a seguinte fala: "Sou estudante de Ciência da Computação na UnB". A sessão de conversação continua normalmente, e momentos depois, Pedro indaga: "Qual é o meu curso?". Espera-se que o agente responda algo como "Ciência da Computação".

Porém, agora considera-se o cenário onde Pedro fala "Sou estudante de computação", e em seguida clica em "Nova conversa". Uma nova sessão de conversação é criada, e essa possui sua própria janela de contexto, ou memória. Portanto, na nova sessão de conversação, Pedro indaga "Qual é o meu curso?". Nesse caso, espera-se que o agente responda algo como "Não tenho acesso a essa informação.". Isso porque a informação sobre o curso de Pedro somente está disponível na sessão de conversação antiga.

## **POST chat\_sessions/<id>/chat\_messages**

Uma vez que uma sessão de conversação é criada, o usuário deve ser capaz de enviar novas mensagens ao agente, e obter as respostas geradas por esse. Em suma, a rota de criação de mensagens de chat inicia a persistência de uma mensagem do usuário no banco de dados, e em seguida, realiza consulta ao **QAService** para a geração de uma resposta. A mensagem do agente é por fim retornada como resposta à requisição HTTP, e a interface de usuário encarrega-se de exibir ambas mensagens do usuário e agente. Esse processo é ilustrado em mais detalhes na figura 4.11.

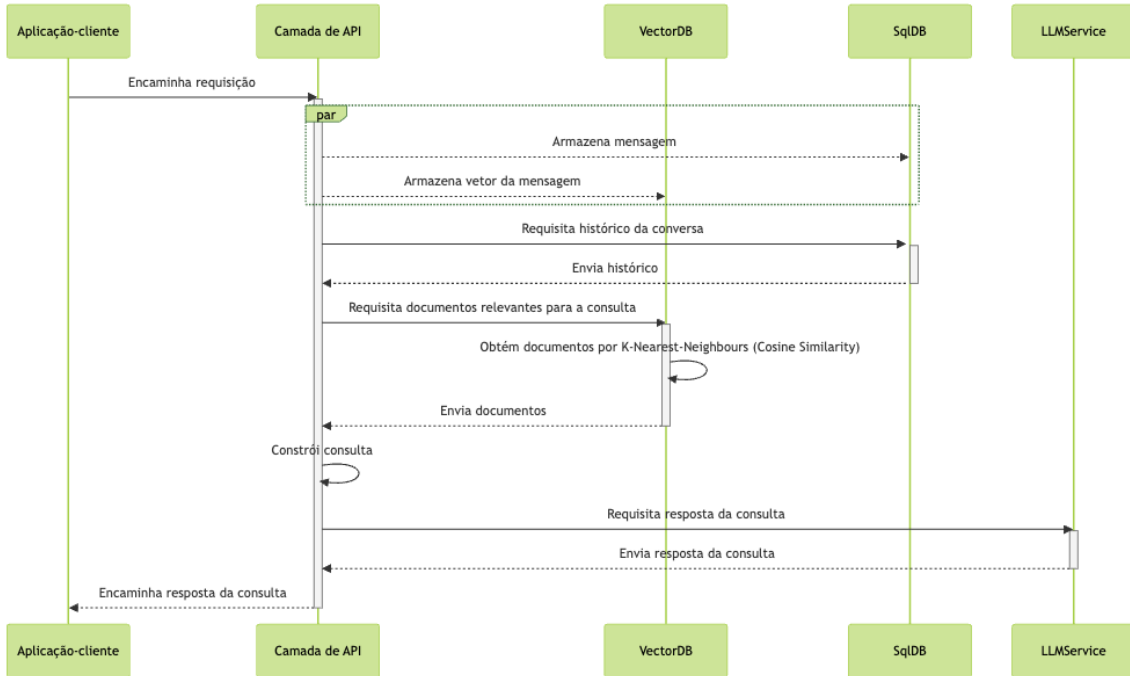


Figura 4.11: Fluxo de persistência de mensagem de usuário e geração de resposta pelo agente.

## 4.7 Aplicação-cliente

Neste trabalho, a aplicação-cliente é a porção do sistema projetada para execução diretamente na máquina do utilizador. Fundamentalmente, trata-se de uma coleção de arquivos estáticos de marcação de linguagem em HTML, bem como *scripts* escritos em *Javascript*, que permitem ao usuário interagir com o sistema via navegador. Essa aplicação disponibiliza uma interface gráfica para que o usuário possa interagir com o sistema de forma simples e visualmente agradável.

Existem diversas decisões a serem consideradas para o projeto de uma aplicação que funcione diretamente do dispositivo do cliente final. Por exemplo, o desempenho da

aplicação é um aspecto determinante para a satisfação do usuário e correta operação do sistema. Isso exige uma avaliação cuidadosa das tecnologias em relação à velocidade de renderização de conteúdo e utilização de memória. Dessa forma, a otimização da performance da aplicação em diferentes sistemas operacionais, condições de internet e sob variadas combinações de recursos, como processador e memória, é essencial para garantir uma experiência fluida aos usuários.

Outro ponto importante diz respeito à curva de aprendizado e esforço requerido para implementação ou manutenção das interfaces de usuário. Diante dos recursos disponíveis para a execução deste projeto, determinou-se que o trabalho de concepção e lapidação do agente conversacional, carga de trabalho típica da aplicação-servidor, possui maior relevância em relação à produção de interfaces complexas. Por isso, buscou-se uma solução que cumprisse os requisitos com simplicidade e rapidez de desenvolvimento.

Nesse cenário, escolheu-se uma biblioteca denominada **Streamlit** [32]. Trata-se de uma biblioteca para linguagem Python, que se destaca como uma escolha ideal devido à sua simplicidade e eficiência para realização de análise de dados interativa. Dentre suas funcionalidades está a disponibilização de diversos componentes para análise de dados como tabelas e gráficos, e até mesmo componentes de chat, motivados pela popularização dos MLL. Além disso, também disponibiliza um serviço de hospedagem de aplicações que facilita a publicação de seus aplicativos para usuários finais.

#### 4.7.1 Tela de login

Em observância ao requisito REQ-4, que descreve os requerimentos sobre a interface de usuário, implementou-se uma tela de login (figura 4.12) constituída por um formulário para entrada de email e senha. Apesar de se tratar de uma tela visualmente simples, existe uma importante lógica que condiciona a exibição dessa página: essa sempre deve ser exibida quando 1) o usuário não possuir um *token JWT* armazenado na memória do navegador, ou 2) alguma requisição à aplicação-servidor retornar status HTTP 401, indicando falha de autorização.

Em maiores detalhes, a tela possui um formulário vinculado à rota */sign\_in* da camada de API. Toda vez que é submetido, uma requisição do tipo POST é enviada, e realiza-se a checagem das credenciais do usuário no **AuthService**. Quando houver uma combinação correta de email e senha, um *token JWT* é retornado como resposta à requisição. Vale ressaltar que o armazenamento e anexação do *token* em requisições futuras é uma tarefa incumbida à aplicação-cliente. Nesse contexto, o papel do *back-end* é simplesmente gerar o *token*, e não armazená-lo.

Com base na resposta à requisição de autenticação no sistema, existem dois caminhos possíveis. O primeiro é a exibição de uma mensagem de erro caso a autenticação falhe,

conforme a figura 4.13. O segundo é o redirecionamento do usuário ao *dashboard* da aplicação, ilustrado na figura 4.18.

Deploy ⓘ

CICBOT 

**Insira suas credenciais**

Email

Senha

Entrar


Não tem uma conta?

[Cadastre-se](#)

Made with Streamlit

Figura 4.12: Tela de login de usuários

Login failed.

CICBOT 

**Insira suas credenciais**

Email

Senha

Entrar

Figura 4.13: Demonstração de erro de submissão do formulário de autenticação.

## 4.7.2 Menu lateral

O menu lateral atua como a principal área de navegação da plataforma. Isso porque permite que o usuário crie e acesse suas conversas. Assim, em termos técnicos, é a interface

gráfica que habilita o usuário a ler e escrever sessões de conversação, representadas pela entidade *chat\_sessions* no banco de dados. Esse componente é ilustrado na figura 4.14, em uma versão ampliada.

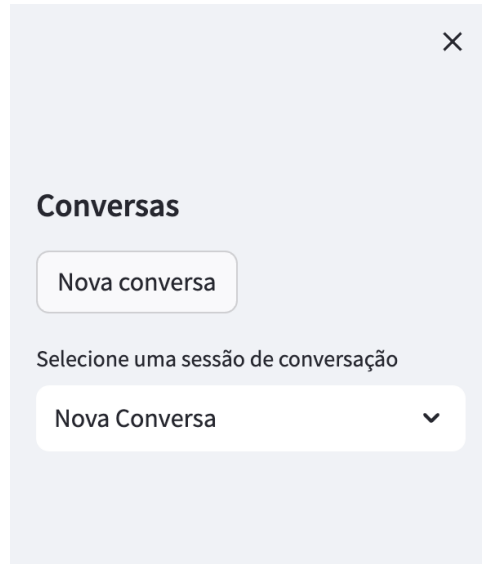


Figura 4.14: Demonstração visual do menu lateral da aplicação.

A primeira integração entre esse componente visual e a aplicação-servidor ocorre ao renderizar a página em tela. Isso dispara uma requisição do tipo GET para buscar as conversações, endereçada à rota */chat\_sessions*. Enfatiza-se que um *token* válido deve ser provido, e o conteúdo decodificado deste *token* irá determinar quais sessões de conversação pertencem ao usuário requisitante. Como resposta, em caso de sucesso, retorna-se uma lista de conversas. Essas são então dispostas visualmente em um componente do tipo seletor, indicado na figura 4.14 sob o rótulo "Selecione uma sessão de conversação". Já no caso de erro, é disposto um alerta ao usuário, conforme a figura 4.15.

Outro ponto marcante da interface atrelado à aplicação-servidor é o botão "Nova conversa". Um clique nesse botão desencadeia uma chamada do tipo POST à rota */chat\_sessions*, e analogamente à chamada de leitura de sessões de conversação, esse caminho também requer um JWT para autenticação. Assim, será persistida uma nova sessão de conversação no banco de dados, por padrão denominada "Nova conversa - ", seguido pela data e hora da requisição. Além disso, a sessão criada também contém uma referência ao identificador único do usuário, no campo *user\_id*, que atua como chave estrangeira e relaciona ambos registros. Ao ser retornada como resposta, a nova conversa é então acrescentada à última posição da lista de sessões exibidas no menu seletor.



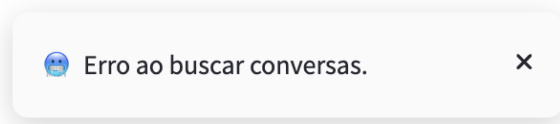


Figura 4.15: Demonstração de alerta exibido em caso de falha na recuperação de sessões de conversação.

### 4.7.3 Formulário de chat

O formulário de chat é o principal recurso da aplicação-cliente, visto que é a interface direta entre o usuário e o agente conversacional. É composto por duas seções principais: o histórico da conversa, e uma caixa de texto aonde as interações do usuário são enviadas.

A exibição do formulário de chat inicia-se a partir da seleção de uma sessão de conversação no menu lateral. Em seguida, realiza-se uma requisição do tipo **GET** à rota `/chat_sessions/<id>/chat_messages`, sendo `<id>` o identificador único do recurso a ser buscado. Vale ressaltar que essa requisição requer autenticação, logo envia-se o JWT do usuário como cabeçalho, e assim é feita uma verificação no servidor para garantir que o usuário solicitante possua permissão de leitura da sessão de conversação referenciada.

A resposta do servidor, quando bem sucedida, contém uma lista de mensagens relacionadas à sessão selecionada, armazenadas na tabela `chat_messages`. É importante notar que a ordenação correta do histórico é de suma relevância para a visualização correta da conversa, de forma a respeitar sua cronologia. Nesse aspecto, definiu-se que a aplicação servidor é responsável por ordenar os registros. Assim, para exibição das mensagens em tela, basta iterar-se pela lista. Nesse processo, cada mensagem é exibida em um balão que contém seu conteúdo textual. Além disso, de modo a diferenciar mensagens do usuário e do agente conversacional, utilizam-se ícones que representam um avatar humano e robótico, respectivamente.

Apesar do processo de solicitação de mensagem ter uma lógica simples e extensivamente testada, existe a possibilidade de falha. Embora não tenham sido evidenciados durante o desenvolvimento desta aplicação, podem haver eventuais erros pontuais, como por exemplo um cenário de indisponibilidade da API do GPT, ou até mesmo da aplicação-servidor deste trabalho. Por isso, adotou-se o tratamento de erros de requisições como uma boa prática de desenvolvimento, e assim exibe-se uma mensagem de erro caso o carregamento da sessão de conversação e suas mensagens falhe, conforme a figura 4.16.

Outra ação relevante no contexto dessa interface é o envio de mensagens. Esse processo inicia-se ao pressionar o botão de submissão ao lado da caixa de texto. Assim, desencadeia-se uma sucessão de ações. O primeiro efeito é uma requisição do tipo **POST** à rota `/chat_sessions/<id>/chat_messages`, que visa persistir a mensagem. Conforme discutido

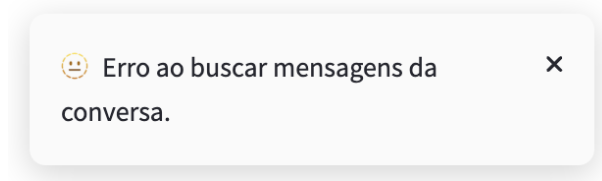


Figura 4.16: Demonstração do alerta em caso de falha na recuperação de mensagens

em maiores detalhes na subseção 4.6.1, esse processo culmina na geração de uma resposta por parte do agente conversacional, que é retornada e exibida em tela.

Embora a requisição seja a principal ação desencadeada com a submissão da mensagem, observam-se alguns efeitos adicionais na interface. Primeiramente, a caixa de texto é bloqueada, de tal modo a impedir o envio de mensagens concorrentemente. Além disso, exibe-se o aviso "*Aguarde a resposta do assistente...*", conforme a figura 4.17. A primeira medida evita uma importante condição de corrida na cronologia das mensagens, enquanto a segunda visa manter o usuário informado e tornar claro o andamento de uma carga de trabalho.

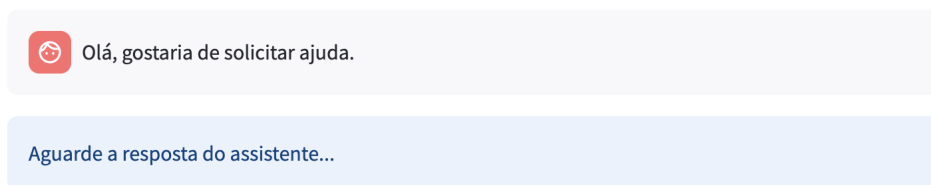


Figura 4.17: Demonstração do aviso de carregamento de envio de mensagem

Para exemplificar a situação supracitada, considera-se a primeira mensagem enviada pelo usuário como  $X$ . Tal mensagem vai, de forma assíncrona, gerar uma resposta  $Y$  após 30 segundos. Agora, supõe-se que o usuário envia uma mensagem  $I$ , 10 segundos após o envio da mensagem  $A$ . Toma-se  $I$  como uma mensagem curta, e portanto a resposta  $J$  leva apenas 5 segundos para ser gerada, também assincronamente. Nesse contexto, a ordem resultante das mensagens será:  $X$ ,  $I$ ,  $J$  e  $Y$ . Observa-se que a resposta  $Y$  para a mensagem  $X$  ficou deslocada, e isso pode gerar um possível cenário de confusão no usuário, bem como viola o funcionamento do requisito REQ-2.4, que diz respeito à ordenação dos ciclos de conversa. Em observância a esse requisito, e de modo a manter ambos ciclos de conversação intactos, é vetado o envio de mensagens enquanto existe uma consulta ao assistente em andamento. Logo, assegura-se que o exemplo acima resultaria na permutação  $X$ ,  $Y$ ,  $I$ ,  $J$ .

A situação acima demonstra um cenário específico que é característico de sistemas de natureza assíncrona. Isto é, apesar de fluxos de trabalho muitas vezes serem claros

e intuitivos, como o simples envio de uma mensagem, nota-se que a introdução da concorrência entre linhas de execução compõe uma nova variável que adiciona complexidade ao projeto. Factualmente, a detecção proativa de situações como a descrita é um fator importante para a prevenção de *bugs* de natureza complexa, e torna-se mais relevante à medida que há criticidade na funcionalidade.

Assim, encerra-se a descrição dos módulos e dos exemplos de resultados obtidos. Nesse sentido, o capítulo seguinte apresenta conclusões obtidas pela realização deste projeto, incluindo o impacto dos serviços apresentados neste Capítulo no que tange os objetivos do trabalho.

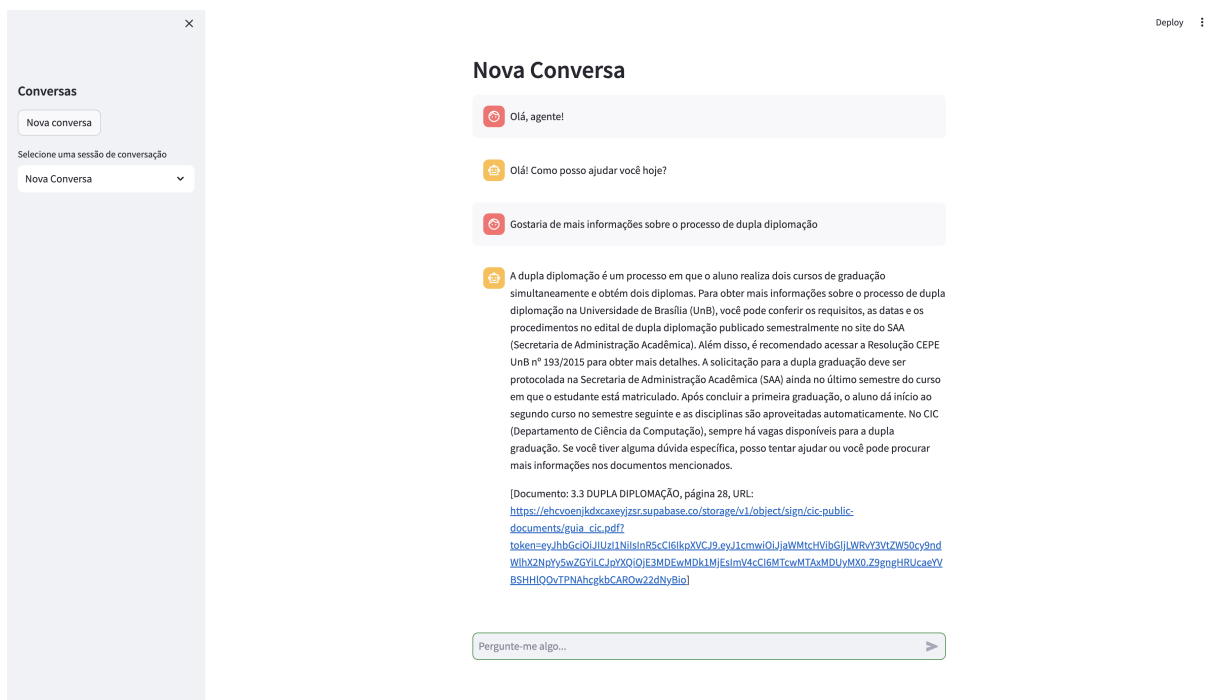


Figura 4.18: Figura que ilustra o *dashboard* completo, incluindo o menu lateral e formulário de chat.

# Capítulo 5

## Conclusões e Trabalhos futuros

Este estudo propõe a criação de um agente conversacional pautado no suporte ao discente em ambientes administrativos acadêmicos. Conforme a Seção 1.2, o projeto explora a viabilidade técnica da utilização de assistentes virtuais na resolução de questões pertinentes ao corpo estudantil do Departamento de Ciência da Computação (CIC). Além disso, possui também um cunho educativo, ao propor a disponibilização do código-fonte do trabalho, e compor assim uma ferramenta que passe a ser desenvolvida colaborativamente, de modo a permitir a aplicação dos aprendizados adquiridos nos cursos desse departamento. Assim, este Capítulo dispõe as contribuições alcançadas, sob a ótica das metas propostas para este Trabalho de Graduação.

Em vista dos módulos aprofundados no Capítulo 5, estabelece-se uma noção sobre a grande complexidade em torno da implantação de um sistema como o proposto. Mesmo com a utilização de tecnologias disruptivas como a Inteligência Artificial, verifica-se que é exigido um esforço considerável para conceber *software* que interligue-a a diferentes módulos, com variadas funções, e que por vezes requerem a aplicação de conceitos fundamentais de diversos ramos da Ciência da Computação.

Nesse sentido, o projeto parte da implementação de um núcleo baseado em mecanismos modernos de processamento de linguagem natural (PLN) [9][20]. Ainda assim, verifica que, de modo a cumprir os objetivos propostos, existe a necessidade de conceber um sistema de informação de maior complexidade. Isto é, que identifique e preencha lacunas deste núcleo baseado em IA, por meio de artifícios que ultrapassam os limites dessa vertente da Ciência. Assim, este trabalho aplica conhecimento sobre diferentes temas, tais como **persistência de dados**, observado nos módulos de banco de dados e armazenamento de arquivos, **métodos de busca e recuperação de dados**, conforme o componente de banco vetorial, **autenticação**, e **interação humano-computador**, que se releva nas interfaces de usuário desenvolvidas.

Como uma conclusão deste trabalho, destaca-se a aplicabilidade da técnica de *retrieval-augmented generation* (RAG) para fim de aprimoramento das gerações realizadas por modelos de linguagem de grande escala. Neste estudo, observou-se que o modelo GPT não possui capacidade inerente de responder a determinadas perguntas sobre o contexto proposto. Dessa forma, RAG demonstrou-se uma estratégia eficaz para expandir a capacidade desse modelo, sem a necessidade de treinamento adicional, visto que possibilitou a resposta dessas perguntas com resultados satisfatórios.

Em vista dos objetivos específicos deste trabalho, nota-se que o teor de código aberto desta aplicação é um facilitador para a execução de trabalhos futuros e proposta de novas funcionalidades ao sistema. Espera-se que a disponibilização do código, aliada à exposição deste documento, possam motivar a participação de novos mantenedores, e assim cumprir a proposta educacional deste projeto enquanto aprimora-se a qualidade e os cenários de utilização do sistema no âmbito do CIC.

Neste trabalho, também realizou-se a integração de um sistema de armazenamento de arquivos em nuvem, com o objetivo de satisfazer a criação de uma base de conhecimento unificada. Contudo, observa-se que os documentos nela inseridos representam apenas uma pequena parcela do conteúdo disponível digitalmente sob domínio do CIC. Como proposta de próximos passos ao andamento deste trabalho, expõe-se a intenção de refinar a base de conhecimento por meio da migração de mais documentos.

Além disso, busca-se estabelecer processos para garantir que essa seja consistentemente atualizada. Assim, propõe-se que a base de conhecimento implementada seja mantida em parceria com os profissionais do CIC encarregados da elaboração e curadoria daqueles documentos que possam ser úteis ao contexto deste trabalho. Dessa forma, seria possível cumprir integralmente a proposta de centralização de conteúdo.

Outra proposta de trabalho futuro é a implementação de um mecanismo de avaliação de respostas automáticas por parte dos usuários. Isso permitiria a coleta de dados de utilização, e o embasamento da qualidade do sistema por meio de análises estatísticas formais.

Embora constatado que o agente conversacional consegue responder apropriadamente as perguntas de avaliação, o critério de escolha dessas foi subjetivo do autor. Ademais, por se tratar de uma ferramenta baseada em entrada de texto livre, existem diversos outros aspectos que podem ser avaliados por meio de uma funcionalidade de avaliação das respostas automáticas. Alguns exemplos são: tom do agente, que pode exercer mais coloquialidade ou formalidade, mecanismo de memória, que diz respeito à capacidade do agente para referenciar conversas passadas, e a própria precisão das respostas para outros temas.

Finalmente, com base nas metas deste trabalho, o agente conversacional desenvolvido

demonstra ser uma ferramenta eficiente para o cumprimento dos objetivos propostos. A capacidade do sistema em responder de forma precisa e coerente as perguntas de avaliação estabelecidas não apenas atesta a sua viabilidade técnica, mas também pavimenta o caminho para modernização dos mecanismos de suporte aos discentes do CIC. Logo, fica evidente o potencial de utilização de arquiteturas baseadas neste projeto na área acadêmica, especialmente para melhorar a experiência dos alunos e a eficiência dos processos educacionais. Portanto, o presente trabalho fornece uma base sólida para futuros projetos com foco na otimização dos sistemas de suporte ao discente e na interiorização do uso de IA em contextos administrativos acadêmicos.

# Referências

- [1] Gokul, Anand: *Llms and ai: Understanding its reach and impact*. maio 2023. 1
- [2] Eloundou, Tyna, Sam Manning, Pamela Mishkin e Daniel Rock: *Gpts are gpts: An early look at the labor market impact potential of large language models*, 2023. 1
- [3] Pereira, Rodrigo, Claudio Lima, Tiago Pinto e Arsénio Reis: *Virtual assistants in industry 4.0: A systematic literature review*. Electronics, 12(19), 2023, ISSN 2079-9292. <https://www.mdpi.com/2079-9292/12/19/4096>. 1
- [4] Hardesty, Larry: *Alexa ai's natural-language-understanding papers at icassp 2022*, 2022. 1
- [5] *Página de fale conosco do cic*. <https://www.cic.unb.br/o-cic/fale-conosco>. 3
- [6] *Guia do calouro, 3rd ed.* <https://www.cic.unb.br/informacoes/documentos?download=39:ger-cartilha-da-graduacao>. 3
- [7] Adamopoulou, Eleni e Lefteris Moussiades: *An overview of chatbot technology*. páginas 373–383, maio 2020, ISBN 978-3-030-49185-7. 5
- [8] Weizenbaum, Joseph: *Eliza—a computer program for the study of natural language communication between man and machine*. Commun. ACM, 9(1):36–45, jan 1966, ISSN 0001-0782. <https://doi.org/10.1145/365153.365168>. 5, 6
- [9] Radford, Alec e Karthik Narasimhan: *Improving language understanding by generative pre-training*. 2018. <https://api.semanticscholar.org/CorpusID:49313245>. 5, 6, 8, 50
- [10] Devlin, Jacob, Ming-Wei Chang, Kenton Lee e Kristina Toutanova: *BERT: pre-training of deep bidirectional transformers for language understanding*. CoRR, abs/1810.04805, 2018. <http://arxiv.org/abs/1810.04805>. 5, 6
- [11] Khurana, Diksha, Aditya Koli, Kiran Khatter e Sukhdev Singh: *Natural language processing: state of the art, current trends and challenges*. Multimedia Tools and Applications, 82(3):3713–3744, julho 2022, ISSN 1573-7721. <http://dx.doi.org/10.1007/s11042-022-13428-4>. 5, 7
- [12] Wallace, Richard S.: *The Anatomy of A.L.I.C.E.*, páginas 181–210. Springer Netherlands, Dordrecht, 2009, ISBN 978-1-4020-6710-5. [https://doi.org/10.1007/978-1-4020-6710-5\\_13](https://doi.org/10.1007/978-1-4020-6710-5_13). 5, 6

- [13] Satu, Md. Shahriare, Md. Hasnat Parvez e Shamim-Al-Mamun: *Review of integrated applications with aiml based chatbot*. Em *2015 International Conference on Computer and Information Engineering (ICCIE)*, páginas 87–90, 2015. 6
- [14] Zhao, Wayne Xin, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian Yun Nie e Ji Rong Wen: *A survey of large language models*, 2023. 6, 8
- [15] Kung, Tiffany H., Morgan Cheatham, Arielle Medenilla, Czarina Sillos, Lorie De Leon, Camille Elepaño, Maria Madriaga, Rimel Aggabao, Giezel Diaz-Candido, James Maningo e Victor Tseng: *Performance of chatgpt on usmle: Potential for ai-assisted medical education using large language models*. *PLOS Digital Health*, 2, 2022. <https://api.semanticscholar.org/CorpusID:254876189>. 6
- [16] Wu, Shijie, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg e Gideon Mann: *Bloombergpt: A large language model for finance*, 2023. 6
- [17] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser e Illia Polosukhin: *Attention is all you need*. *CoRR*, abs/1706.03762, 2017. <http://arxiv.org/abs/1706.03762>. 7
- [18] Hochreiter, Sepp: *The vanishing gradient problem during learning recurrent neural nets and problem solutions*. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, abril 1998. 7
- [19] Yenduri, Gokul, Ramalingam M, Chemmalar Selvi G, Supriya Y, Gautam Srivastava, Praveen Kumar Reddy Maddikunta, Deepti Raj G, Rutvij H Jhaveri, Prabadevi B, Weizheng Wang, Athanasios V. Vasilakos e Thippa Reddy Gadekallu: *Generative pre-trained transformer: A comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions*, 2023. 8
- [20] Lewis, Patrick, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel e Douwe Kiela: *Retrieval-augmented generation for knowledge-intensive nlp tasks*, 2021. 8, 50
- [21] Dokmanic, Ivan, Reza Parhizkar, Juri Ranieri e Martin Vetterli: *Euclidean distance matrices: Essential theory, algorithms, and applications*. *IEEE Signal Processing Magazine*, 32(6):12–30, novembro 2015, ISSN 1053-5888. <http://dx.doi.org/10.1109/MSP.2015.2398954>. 8
- [22] Lahitani, Alfirna Rizqi, Adhistya Erna Permanasari e Noor Akhmad Setiawan: *Cosine similarity to determine similarity measure: Study case in online essay assessment*. Em *2016 4th International Conference on Cyber and IT Service Management*, páginas 1–6, 2016. 8



- [23] Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li e Peter J. Liu: *Exploring the limits of transfer learning with a unified text-to-text transformer*, 2023. 9
- [24] *Supabase Auth*. <https://supabase.com/docs/guides/auth>. [Acessado em 03/12/2023]. 29
- [25] Michael B. Jones, John Bradley e Nat Sakimura: *JSON Web Token (JWT)*. <https://datatracker.ietf.org/doc/rfc7519/>, 2015. [Acessado em 03/12/2023]. 29
- [26] *Supabase Storage*. <https://supabase.com/docs/guides/storage>. [Acessado em 03/12/2023]. 33
- [27] Mikolov, Tomas, Kai Chen, Greg Corrado e Jeffrey Dean: *Efficient estimation of word representations in vector space*, 2013. 34
- [28] *ADA-002 Reference*. <https://openai.com/blog/new-and-improved-embedding-model>. [Acessado em 03/12/2023]. 34
- [29] *Pinecone Reference*. <https://docs.pinecone.io/docs/overview>. [Acessado em 03/12/2023]. 35
- [30] *FastAPI Reference*. <https://fastapi.tiangolo.com/about/>. [Acessado em 03/12/2023]. 38
- [31] *Open API 3.1.0*. <https://spec.openapis.org/oas/v3.1.0>. [Acessado em 03/12/2023]. 38
- [32] *Streamlit Reference*. <https://docs.streamlit.io/library/get-started/main-concepts>. [Acessado em 03/12/2023]. 44