Pontifícia Universidade Católica
do Rio de Janeiro

João Pedro Barros Botelho

ChatBot com Base de Conhecimento Própria: Desafios e Soluções utilizando IA Generativa

Projeto Final de Graduação

CENTRO TÉCNICO CIENTÍFICO - CTC
DEPARTAMENTO DE INFORMÁTICA
Curso de Graduação em Engenharia da Computação

Orientador: Prof. Augusto Cesar Espíndola Baffa

Rio de Janeiro
Dezembro de 2024

Pontifícia Universidade Católica
do Rio de Janeiro

João Pedro Barros Botelho

ChatBot com Base de Conhecimento Própria: Desafios e Soluções utilizando IA Generativa

Relatório de Projeto Final, apresentado ao programa Engenharia da Computação da PUC-Rio como requisito parcial para a obtenção do título de Bacharel em Engenharia da Computação.

Prof. Augusto Cesar Espíndola Baffa
Orientador
Departamento de Informática – PUC-Rio

Rio de Janeiro, 18 de Dezembro de 2024

João Pedro Barros Botelho

Graduando em Engenharia da computação pela PUC-Rio.

Ficha Catalográfica

Barros Botelho, João Pedro

ChatBot com Base de Conhecimento Própria: Desafios e Soluções utilizando IA Generativa / João Pedro Barros Botelho; orientador: Augusto Cesar Espíndola Baffa. – 2024.

45 f: il. color. ; 30 cm

Relatório Projeto Final (graduação) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2024.

Inclui bibliografia

1. IA Generativa – Teses. 2. Inteligência Artificial. 3. IA Generativa. 4. ChatBot. I. Baffa, Augusto. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

"Failure cannot cope with persistence."
- Napoleon Hill

Agradecimentos

À minha mãe, **Gelcira**, que apesar de todas as dificuldades, sempre buscou o melhor para o meu bem estar e dando todo o apoio durante meus estudos.

Ao meu pai, **Carlos**, que apesar de não ter visto esta conquista, foi muito importante para o caminho até aqui.

Aos meus amigos de colégio que até hoje nos mantemos unidos e apoiando uns aos outros.

Aos meus companheiros de trabalho do meu estágio no ONS, que foram importantes no incentivo da minha carreira profissional com suas experiências e no meu reconhecimento.

Ao meu orientador, **Augusto Baffa**, que apesar do meu semestre conturbado, me auxiliou no momento que mais precisei.

Resumo

Barros Botelho, João Pedro; Baffa, Augusto. **ChatBot com Base de Conhecimento Própria: Desafios e Soluções utilizando IA Generativa**. Rio de Janeiro, 2024. 45p. Projeto Final de Graduação – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho aborda o desenvolvimento de um chatbot utilizando inteligência artificial generativa, integrando tecnologias como WikiJs, Llama Index e Qdrant, além de ferramentas de infraestrutura como FastAPI e Docker. A solução proposta visa criar uma base de conhecimento própria, eficiente e escalável, capaz de atender a demandas específicas de organizações ou até de usuários comuns. O estudo analisa os desafios enfrentados na implementação, como o gerenciamento de dados de origem da Wiki e assim como a transformação destes arquivos. O resultado foi uma interface web amigável ao usuário final, onde a sua implementação foi modularizada e que pode ser aproveitada em projetos futuros.

Palavras-chave

Inteligência Artificial; IA Generativa; ChatBot.

Abstract

Barros Botelho, João Pedro; Baffa, Augusto (Advisor). **ChatBot with Proprietary Knowledge Base: Challenges and Solutions Using Generative AI**. Rio de Janeiro, 2024. 45p. Relatório de Projeto Final II – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This work addresses the development of a chatbot using generative artificial intelligence, integrating technologies such as WikiJs, Llama Index, and Qdrant, along with infrastructure tools like FastAPI and Docker. The proposed solution aims to create an efficient, scalable, and proprietary knowledge base capable of meeting the specific demands of organizations or even common users. The study analyzes the challenges faced during implementation, such as managing Wiki source data and transforming these files. The result was a user-friendly web interface for the end user, with a modularized implementation that can be leveraged in future projects.

Keywords

Artificial Intelligence; Generative AI; ChatBot.

Sumário

Lista de figuras

Lista de algoritmos

Lista de Abreviaturas

IA – Inteligência Artificial
ISO – *International Organization for Standardization*
ANI – Inteligência Artificial Estreita
AGI – Inteligência Artificial Geral
ASI – Superinteligência Artificial
API – *Application Programming Interface*
LLM – *Large Language Models*
PLN – Processamento de Linguagem Natural
ASGI – *Asynchronous Server Gateway Interface*
URL – *Uniform Resource Locator*
AWS – Amazon Web Services

1
Introdução

O século XXI tem se consolidado como a era da revolução tecnológica digital, e temos a Inteligência Artificial (IA) despontando como uma das inovações mais animadoras e impactantes para a sociedade. O termo Inteligência Artificial é definido pela Organização Internacional de Normalização(ISO) como "um campo técnico e científico dedicado a sistemas de engenharia que geram saídas como conteúdo, previsões, recomendações ou decisões para um conjunto de objetivos definidos por humanos"(INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2022). No entanto, essa definição, embora importante para fins de padronização, não define toda a complexidade do que é a IA, que se caracteriza também pela capacidade de aprendizado, adaptação e autonomia, buscando até simular a inteligência e a ação humana em diversos aspectos.

A concepção da Inteligência Artificial não está restrita ao tempo atual, autores como Stuart Russell e Peter Norvig, por exemplo, já em 1995 definem IA como "o estudo de agentes que recebem percepções do ambiente e executam ações"(RUSSELL; NORVIG, 1995), o que destaca a capacidade da IA de interagir com o mundo e tomar decisões de forma independente. Podemos citar também o papel do cinema, onde desde o século passado tem exercido um papel crucial na construção da idealização e da percepção da IA na sociedade. O filme "2001: Uma Odisseia no Espaço"(1968), de Stanley Kubrick, destaca-se como um marco nesse contexto, apresentando uma visão futurista da IA que antecipou diversas funcionalidades hoje presentes em nosso cotidiano, como o reconhecimento de voz, o processamento de linguagem natural e a tomada de decisões autônomas.

Em meio ao amplo universo da Inteligência Artificial, este projeto se concentra em uma área específica: a IA generativa. Diferentemente da IA tradicional, que se baseia em algoritmos e regras para executar tarefas pré-definidas, a IA generativa utiliza modelos capazes de gerar novos dados semelhantes aos utilizados em seu treinamento e até mesmo durante seu funcionamento, criando conteúdos originais como textos, imagens, áudios e vídeos.

Vale destacar o crescente interesse do mercado na Inteligência Artificial. Uma pesquisa da McKinsey & Company ("The state of AI in early 2024: Gen AI adoption spikes and starts to generate value") revela que 72% das empresas do mundo utilizam alguma Inteligência Artificial em 2024, com a IA

generativa presente em 65% dos negócios. Esse dado reforça a importância de pesquisas que explorem as aplicações e novos desenvolvimentos utilizando a IA generativa, o que foi um incentivo para o desenvolvimento deste projeto. A seguir, na Figura 1, é disponibilizada uma imagem que demonstra a evolução da utilização da IA e da IA Generativa pelas organizações entre os anos de 2017 e 2024.

Figura 1.1: Uso da Inteligência Artificial nas organizações
Fonte: McKinsey & Company, *The state of AI in early 2024: Gen AI adoption spikes and starts to generate value*, 2024.

Essa ascensão da IA generativa se manifesta de forma notável na popularização dos chatbots, que podem ser definidos como interfaces conversacionais que utilizam Inteligência Artificial para interagir com usuários de forma natural e intuitiva. O ChatGPT, desenvolvido pela OpenAI, despontou como um marco nesse cenário, demonstrando a capacidade da IA de gerar textos e conversas coerentes e criativos, traduzir idiomas, responder a perguntas complexas, resumir livros e até mesmo gerar códigos programacionais. Outros exemplos de chatbots que têm ganhado destaque são o Gemini, desenvolvido pelo Google, e o Meta AI, da Meta, cada um com suas características específicas.

1.1 Justificativa

Embora a crescente utilização de chatbots de IA generativa tenha impulsionado o surgimento e aprimoramento dos modelos de linguagem como o GPT da OpenAI, o LaMDA e o Gemini do Google, e o BlenderBot e o Llama da Meta, esta variedade não garante satisfazer a necessidade de todos os usuários e organizações. O principal limitador é a dificuldade de atender as necessidades de usuários em cenários onde se buscam informações específicas, informações essas que não estão disponíveis publicamente e podem ser restritas a um ambiente organizacional, como documentos internos, relatórios financeiros, históricos de clientes ou de acessos a um *website* ou até protocolos e políticas de segurança. Em um mundo tão conectado como estamos vivendo nos tempos atuais, a informação passou a ser um ativo poderoso, isso se dá pelo fato de que ela pode ser um potencializador na melhoria de serviços e produtos, além das tomadas de decisões estratégicas e na eficiência operacional.

1.2 Objetivo

Diante desta problemática, este trabalho de conclusão de curso (TCC) se propõe a utilizar o WikiJs como ferramenta para o desenvolvimento de uma base de conhecimento própria e segura, oferecendo uma alternativa para a implementação de um chatbot personalizado e eficaz. O WikiJs possui uma interface amigável, além de recursos de colaboração, facilitando a criação, organização e atualização do conteúdo da base de conhecimento, permitindo que o chatbot se adapte de forma dinâmica as necessidades dos usuários. Além disso, este trabalho contribui para o avanço do conhecimento na área de IA generativa, explorando o potencial dessa tecnologia na construção de soluções inovadoras que beneficiem a sociedade.

1.3 Organização deste trabalho

Para alcançar o objetivo proposto, este Trabalho de Conclusão de Curso(TCC) se estrutura da seguinte forma:

- Capítulo 1 - Introdução: Apresenta o contexto geral do trabalho, a problemática, a justificativa, os objetivos e a estrutura do trabalho.
- Capítulo 2 - Revisão Literária: Aborda os conceitos e fundamentos teóricos relacionados a Inteligência Artificial, IA generativa, Llama Index, QDrant, WikiJS e Docker.

– Capítulo 3 - Metodologia: Descreve a metodologia utilizada no desenvolvimento do chat, incluindo as etapas de estudos, de levantamento de requisitos e das ferramentas necessárias.

– Capítulo 4 - Resultados: Apresenta os resultados obtidos com o desenvolvimento do chat, incluindo testes, avaliações.

– Capítulo 5 - Conclusões: Discute as conclusões do trabalho, assim como suas contribuições, as limitações encontradas e as sugestões de melhorias futuras.

2
Revisão Literária

Este capítulo apresenta os conceitos e fundamentos teóricos que sustentam este trabalho, explorando as áreas de Inteligência Artificial (IA), IA generativa, Llama Index, QDrant, WikiJS e Docker.

2.1
Inteligência Artificial: Uma Visão Geral

A Inteligência Artificial (IA) está cada vez mais presente no nosso dia a dia, impulsionada pelos avanços tecnológicos e pela grande quantidade de dados disponíveis. Mas afinal, o que é Inteligência Artificial? Definir IA não é tão trivial quanto parece, pois não existe uma única resposta ou uma definição formal que englobe tudo. Isso se deve principalmente à sua característica multifacetada, além das diferentes definições ao longo da história, influenciadas por evoluções técnicas, ou por diferentes áreas de conhecimento e outros fatores. Autores como (ERTEL, 2024) exploram essa complexidade, mostrando a diversidade de abordagens em IA. O artigo de (WANG, 2019) também contribui para essa discussão, aprofundando o debate sobre a definição de IA. Neste trabalho, vamos considerar a Inteligência Artificial (IA) como sistemas computacionais que, autônomos ou semiautônomos, realizam tarefas complexas que até simulam a inteligência humana. São variados os tipos de sistemas, mas em geral, vão além da simples execução de tarefas predefinidas, eles aprendem com a experiência, se adaptam a novas situações, tomam decisões complexas e interagem com o mundo de forma independente, buscando alcançar objetivos definidos.

2.1.1
Tipos de Inteligência Artificial e suas Aplicações

Segundo a (Alura, 2023), a IA pode ser classificada de diferentes maneiras, sejam elas de acordo com as funcionalidades, ou pelas suas capacidades ou pelos objetivos dos sistemas.

2.1.1.1
Classificação por Capacidade

Uma das classificações mais comuns se baseia nos níveis de inteligência em que esses sistemas demonstram, usando a mente humana como parâmetro de comparação, e é estruturada da seguinte forma:

- **IA de máquina reativa:** Sistemas que reagem a estímulos do ambiente, ou seja, levam em consideração somente os dados disponíveis no presente momento, não possuindo memória ou capacidade de aprendizado. São os sistemas mais antigos e básicos de IA, podemos citar os que jogam xadrez, onde temos o exemplo mais conhecido, o supercomputador criado pela IBM na década de 90, o *Deep Blue*, que foi capaz de derrotar o campeão mundial de Xadrez, Garry Kasparov, em 1997. As máquinas reativas são úteis em casos onde não é necessário utilizar experiências passadas ou ser levado em consideração o histórico de outras execuções.
- **IA de memória limitada:** Sistemas que podem armazenar informações e utilizá-las para tomar decisões futuras, mas com capacidade de aprendizado limitada. A limitação se deve principalmente a sua base de conhecimento, a necessidade de avaliação sobre o sistema por parte do usuário, além dos aprendizados dos padrões de utilização. Esse tipo de IA é o usado neste projeto, utilizando uma base de conhecimento prévia para fornecer respostas, além de que a partir do histórico de conversas é possível refinar a sua utilização. Outros exemplos incluem os sistemas de reconhecimento facial e carros autônomos.
- **IA com teoria da mente:** Sistemas que podem entender e simular as emoções, crenças e intenções de outros agentes, sejam eles seres humanos ou outras máquinas. Esse entendimento é importante para a realização de ações de forma natural e contextualizada conforme a mente humana faria. Essa área ainda está em desenvolvimento, principalmente pela dificuldade da Inteligência Artificial compreender a natureza humana, mas representa um importante passo.
- **IA autoconsciente:** Sistemas que possuem consciência de si mesmos e de suas capacidades. Ainda é um realidade e objetivo distante da IA atual, mas levanta questões importantes sobre o futuro da Inteligência Artificial, com debates sobre questões éticas e morais.

2.1.1.2 Classificação por Funcionalidade e Aplicações

No mesmo artigo (Alura, 2023), demonstra uma outra forma de classificarmos a IA, que é com base em sua funcionalidade, ou seja, o quão amplo é o conjunto de tarefas que um sistema pode realizar e suas possíveis aplicações:

- **IA estreita (ANI):** Sistemas que se destacam em tarefas específicas, como reconhecimento facial, tradução de idiomas e jogos de xadrez. A principal característica está no fato que uma vez programado para exercer

determinada função, a IA irá atuará de maneira repetitiva apenas na mesma tarefa. A maioria das aplicações de IA hoje em dia estão nesta categoria.

- **IA geral (AGI):** Sistemas ainda teóricos que possuem inteligência equivalente a humana, sendo capazes de realizar tarefas de nível complexo. Apesar dos estudos em andamento e treinamento de modelos, ainda não foram desenvolvidos sistemas AGI.
- **IA superinteligente (ASI):** Sistemas hipotéticos com inteligência que supera a humana em todos os aspectos, sejam eles criatividade científica, sabedoria geral e habilidades sociais. A ASI ainda é um conceito teórico com grandes implicações e consequências para o futuro da humanidade.

2.1.1.3 Classificação por Técnicas e Abordagens

Temos também uma forma de categorizar a IA é pelas técnicas e abordagens utilizadas para construir os sistemas:

- ***Machine Learning* (Aprendizado de Máquina):** Sistemas que aprendem com dados, sem serem explicitamente programados. Aplicações: sistemas de recomendação, detecção de fraudes, previsão de demanda.
- ***Deep Learning* (Aprendizado Profundo):** Sistemas que utilizam redes neurais artificiais com múltiplas camadas para aprender padrões complexos em dados. Aplicações: reconhecimento de imagens, processamento de linguagem natural, análise de dados.
- **Processamento de Linguagem Natural (PLN):** Sistemas que permitem que computadores entendam e se comuniquem em linguagem humana. Aplicações: tradução automática, análise de sentimentos, chatbots.
- **Visão Computacional:** Sistemas que permitem que computadores observem e interpretem imagens. Aplicações: reconhecimento facial, detecção de objetos, análise de imagens médicas.

Outras áreas de aplicação da IA incluem:

- **Robótica:** Robôs industriais, robôs autônomos, robôs assistentes.
- **Planejamento e otimização:** Logística, escalonamento de tarefas, otimização de recursos.

- **Finanças:** Detecção de fraudes, análise de risco, investimento automatizado.
- **Saúde:** Diagnóstico médico, descoberta de medicamentos, monitoramento de pacientes.

2.2 IA generativa: Explorando Novas Fronteiras

A IA generativa, foco deste trabalho, representa uma área emergente da IA que se concentra em gerar novos dados e conteúdos originais, a partir de modelos treinados em grandes conjuntos de dados. Esses modelos aprendem os padrões e as características dos dados de treinamento e são capazes de criar novos dados que se assemelham aos originais, mas com variações e detalhes diferentes.

2.2.1 Modelos de IA generativa

Existem diferentes tipos de modelos de IA generativa, cada um com suas características e aplicações:

- Modelos de Linguagem: Como o GPT e o Gemini, são capazes de gerar textos coerentes e criativos, traduzir idiomas, responder a perguntas e até mesmo gerar códigos.
- Redes Neurais Generativas (GANs): São capazes de gerar imagens, vídeos e áudios realistas, com aplicações em áreas como design, entretenimento e arte.
- Modelos de Difusão: São uma classe de modelos generativos que gradualmente adicionam ruído a dados de treinamento e, em seguida, aprendem a recuperar os dados revertendo esse processo de ruído.

2.2.2 Aplicações da IA generativa

A IA generativa tem o potencial de revolucionar diversas áreas, com aplicações que vão desde a criação de conteúdo criativo até a análise e interpretação de dados complexos. Podemos citar os exemplos de uso:

- Geração de Conteúdo: Criação de textos, imagens, músicas, roteiros de viagens e outros tipos de conteúdos criativos.
- Análise de Dados: Identificação de padrões, anomalias e *insights* em grandes bases de dados.

- Conteúdo Personalizado: Criação de experiência personalizada para os usuários, como recomendações de produtos, conteúdo e serviços.
- Medicina: Aprimoramento em diagnósticos, como nos exames de imagem de câncer de mama que tem sido possível identificar em estágios ainda iniciais.
- Pesquisa Científica: Geração de novas hipóteses, simulação de cenários e descoberta de novos conhecimentos.

2.3 Llama Index e QDrant: Ferramentas para Chatbots Inteligentes

Para o desenvolvimento do chatbot proposto neste trabalho, serão utilizadas as tecnologias Llama Index e QDrant, que oferecem recursos avançados para indexação, consulta e processamento de dados textuais.

2.3.1 Llama Index

O Llama Index é uma ferramenta de código aberto que permite indexar grandes volumes de dados textuais, como documentos, artigos, livros e notícias, para assim consultá-los de forma eficiente. A sua utilização é a partir de técnicas de processamento de linguagem natural para extrair informações relevantes dos textos e assim criar um índice que facilita a busca e o retorno das informações. Neste projeto, o *framework* do Llama Index foi implementado em um código Python local, o que permitiu a integração com outras bibliotecas e *frameworks* Python, como o FastAPI e o Streamlit. Essa abordagem facilitou o desenvolvimento e a personalização do chatbot, além de oferecer maior controle sobre o processo de indexação e consulta dos dados.

O Llama Index utiliza os *embeddings* gerados pela OpenAI para indexar os documentos da base de conhecimento. Cada documento é representado por um vetor (*embedding*) que captura seu significado semântico. O Llama Index armazena esses vetores no QDrant, um banco de dados vetorial otimizado para buscas por similaridade. Quando o usuário faz uma pergunta, o Llama Index utiliza o QDrant para encontrar os documentos mais relevantes (com base na similaridade dos *embeddings*) e os utiliza para gerar a resposta.

Com base em (CloudFlare, (s.d.)) e (Elastic, (s.d.)), podemos definir *Embeddings* como representações vetoriais densas de itens, como palavras, frases, documentos ou imagens em um espaço vetorial contínuo. Essas representações capturam as relações semânticas entre os itens, de forma que itens com significados semelhantes são mapeados para vetores próximos uns dos outros

no espaço vetorial. Essa propriedade torna os *embeddings* úteis para uma variedade de tarefas de aprendizado de máquina, como classificação de texto, resposta a perguntas e tradução automática.

2.3.2 QDrant

De acordo com a documentação oficial (Qdrant, 2024), o QDrant é uma ferramenta de código aberto de banco de dados vetorial que oferece recursos de processamento de linguagem natural e recuperação de informações. Ele permite armazenar, indexar e consultar vetores de dados, que representam o significado semântico de textos e outros tipos de dados. O QDrant utiliza algoritmos de aprendizado de máquina para encontrar os vetores mais relevantes para uma determinada consulta, permitindo a recuperação de informações precisas e contextualizadas. Para o armazenamento e consulta dos vetores de *embedding*, foi utilizado o Qdrant Cloud, um serviço de banco de dados vetorial que oferece escalabilidade, alta disponibilidade e segurança. A comunicação com o Qdrant Cloud é realizada através de sua *API*, utilizando uma *URL* e uma *API Key* para autenticação.

Os vetores de busca são utilizados para representar documentos da base de conhecimento em um espaço vetorial, cada documento é representado por um vetor que captura suas características principais. Existem dois tipos principais de vetores: densos e esparsos. Vetores densos, como os *embeddings* de texto, representam os dados em um espaço contínuo, capturando informações semânticas detalhadas. Vetores esparsos, por outro lado, representam os dados em um espaço discreto, indicando a presença ou ausência de características específicas, como palavras-chave.

O QDrant permite a criação de um índice híbrido que combina vetores densos e esparsos. Essa combinação oferece diversas vantagens, como a capacidade de representar informações semânticas complexas e realizar buscas eficientes por similaridade e por características específicas. Neste projeto, o QDrant foi configurado para armazenar tanto os *embeddings* de texto (vetores densos) quanto as palavras-chave dos documentos (vetores esparsos). Essa combinação permite que o chatbot encontre os documentos mais relevantes, considerando tanto a similaridade semântica quanto a presença de palavras-chave relevantes.

2.4 WikiJs: Construindo uma Base de Conhecimento Sólida

O WikiJs, uma plataforma de código aberto para criação de *wikis*, será utilizado para construir a base de conhecimento do chatbot. Ele oferece uma interface intuitiva para criação e edição de páginas, além de organizar os conteúdos em categorias e permitir controle de acesso aos usuários, seja com uma autenticação básica ou até mesmo com um serviço de autenticação de terceiros. Estes fatores fazem do WikiJs uma ferramenta versátil e robusta para criar uma base de conhecimento personalizada e adaptável as necessidades do chatbot.

2.5 FastAPI

Para centralizar as chamadas de API e facilitar a comunicação entre os diferentes componentes do sistema, foi utilizado o FastAPI. O FastAPI é um *framework web* moderno e de alto desempenho para construção de APIs em Python, sendo fácil de usar, com uma sintaxe limpa e intuitiva. Outra característica importante é a sua validação automática de dados via *type hints*, que é uma sintaxe especial que permite declarar o tipo das variáveis da API, oferecendo uma validação automática e deixando o código mais claro.

2.5.1 Interface de Programação de Aplicações (API)

De acordo com o artigo (redhat, 2023), uma API (Interface de Programação de Aplicações) é um conjunto de ferramentas, definições e protocolos que permite que diferentes sistemas de software se comuniquem e interajam entre si. As APIs são responsáveis por conectar soluções e serviços, sem a necessidade de saber como o outro foi implementado.

2.5.2 Integração FastAPI e Unicorn

Unicorn é um servidor web ASGI (*Asynchronous Server Gateway Interface*) de alto desempenho, conhecido por sua robustez e capacidade de lidar com um grande volume de conexões simultâneas. Ele atua como um intermediário entre o servidor web e a aplicação.

A integração entre o FastAPI e o Unicorn permite que aplicações desenvolvidas com o FastAPI sejam executadas em um ambiente de produção robusto e escalável. O Unicorn atua como um servidor web, recebendo as requi-

sições dos usuários e encaminhando-as para a aplicação FastAPI. O FastAPI, por sua vez, processa as requisições e retorna as respostas apropriadas.

2.6 Streamlit

A interface do usuário do chatbot foi desenvolvida utilizando o Streamlit, uma biblioteca Python que permite a criação de aplicações web interativas de forma rápida e simples. O Streamlit oferece componentes pré-construídos para chat, o que facilitou a implementação da interface de conversação com o chatbot. A comunicação entre o Streamlit e o FastAPI é realizada através de chamadas de API, permitindo que o usuário envie perguntas e receba respostas do chatbot em tempo real. Além de perguntas e respostas, o Streamlit foi preparado para tratar a importação dos arquivos e também do processamento dos arquivos, além de lidar com questões de credenciais e API Key.

2.7 Docker e Docker Compose

No desenvolvimento deste chatbot, a tecnologia *Docker* foi empregada para garantir a portabilidade e a escalabilidade da aplicação. No artigo (hostinger, 2024) podemos conferir a definição de *Docker* que é uma plataforma de código aberto que permite a criação de contêineres, que são ambientes isolados e autossuficientes que contêm tudo o que uma aplicação precisa para ser executada, incluindo o código-fonte, bibliotecas, dependências e configurações do sistema operacional. Diferentemente de máquinas virtuais, que virtualizam o hardware, os contêineres virtualizam o sistema operacional, compartilhando o *kernel* do sistema hospedeiro. Essa abordagem torna os contêineres mais leves, eficientes e rápidos de iniciar, o que facilita o desenvolvimento, a implantação e a escalabilidade da aplicação. No contexto do chatbot, o Docker foi utilizado para contentorizar cada componente do sistema (Streamlit, FastAPI, WikiJs, PostgreSQL), o que garante a independência entre os componentes, facilitando a gestão de dependências e permite a execução da aplicação em diferentes ambientes de forma consistente.

2.8 Considerações Finais

Este capítulo apresentou os conceitos e fundamentos teóricos que sustentam este trabalho, explorando as áreas de IA, IA generativa, Llama Index, QDrant, WikiJs. API e Docker. A revisão da literatura permitiu contextualizar o trabalho e delinear o cenário tecnológico em que se insere as tecnologias utili-

zadas. Nos próximos capítulos, será apresentado o desenvolvimento do chatbot proposto, com a descrição da metodologia, dos resultados e das conclusões.

3
Metodologia

Este capítulo descreve a metodologia utilizada na pesquisa do trabalho, aprofundando quais ferramentas são necessárias para o desenvolvimento do chatbot, assim como todas as suas técnicas e procedimentos adotados. O processo de desenvolvimento foi dividido em etapas principais:

3.1
Definição dos Requisitos

Sabendo que o objetivo principal é criar um chatbot de IA generativa com uma base de conhecimento própria, é necessária uma definição clara dos requisitos do chatbot, incluindo funcionalidades, características e restrições para entender quais ferramentas são adequadas para o seu funcionamento. Podemos definir então:

- O chatbot deve ser capaz de manter uma conversa coerente e natural com o usuário, respondendo a perguntas e fornecendo informações relevantes.
- O chatbot deve utilizar uma base de conhecimento própria, construída a partir de documentos e informações relevantes para o domínio de aplicação.
- O chatbot deve permitir que o usuário envie arquivos de texto para serem incluídos na base de conhecimento.
- O chatbot deve ser capaz de acessar, baixar e processar arquivos de texto armazenados em um bucket S3 na AWS.
- A interface do chatbot deve ser intuitiva e fácil de usar, permitindo que o usuário interaja de forma natural com o sistema.

3.2
Seleção das Ferramentas

Seleção criteriosa das ferramentas, justificando cada escolha com base nos requisitos e nas características de cada ferramenta. De acordo com as funcionalidades e requisitos, as ferramentas escolhidas foram:

3.2.1 A base de Conhecimento

Com base nos requisitos do projeto, o desafio era achar a melhor maneira ou ferramenta para a construção da base de conhecimento. Inicialmente, surgiu a ideia de utilizar uma base de documentos e arquivos de texto, seja em uma pasta local ou até mesmo em um repositório online, porém a falta de controle sobre os conteúdos dos arquivos seria um problema, pois não permitiria um controle de edição e versionamento dos conteúdos, apesar de ser uma ideia funcional para construir a base de conhecimento.

Após a decisão de fazer uma base de conhecimento colaborativa, porém com controle do conteúdo, a ideia de utilizar uma Wiki foi a mais aceita por ser uma plataforma de criação, edição e organização de páginas de forma simples e dinâmica. Dentre as diversas plataformas Wiki existentes, surgiu a plataforma WikiJs como escolhida para ser utilizada no projeto devido a sua interface intuitiva, além dos recursos de colaboração e a capacidade de lidar com documentos escritos. Além disso, a WikiJs tem filtros de moderação e permissionamento de postagens, delimitando quem pode criar uma página, assim como quem pode alterar uma determinada página de conteúdo, isso tudo com um controle de acesso por autenticação básica ou até integrada com diversos SSO existentes no mercado. Outra funcionalidade é que todo o conteúdo presente na WikiJs, além de salvo em um banco de dados PostgreSQL, pode ser salvo através das funções nativas de *backup* tanto em repositórios locais ou até em serviços de armazenamento em nuvem.

3.2.2 Indexação e Armazenamento dos Dados

Com a base de conhecimento definida, veio a principal questão de qual ferramenta escolher para lidar com os textos da base de conhecimento e fazer a função de Processamento de Linguagem Natural (PLN). O Llama Index surgiu como uma solução de *framework* ideal pelo seu desenvolvimento voltado para a indexação, a organização e a recuperação de grandes volumes de dados. São diversos os tipos de dados que o Llama Index pode processar, desde dados não estruturados, até mesmo registros de bancos de dados estruturados e gráficos de conhecimento.

O Llama Index então transforma os dados de nossa fonte em representações vetoriais numéricas, trazendo um sentido semântico para os dados e permitindo, posteriormente, uma busca otimizada das similaridades a partir dos índices criados das informações relevantes dos textos.

Outras ferramentas de indexação dos dados foram cogitadas, como por exemplo o LangChain que possui uma característica de ser modular e flexível integrando diversas formas de *LLM*, porém devido ao Llama Index ter uma facilidade de implementação e desempenho em criar e buscar índices, o mesmo foi escolhido para o projeto.

Para o armazenamento e consulta dos vetores de *embedding* gerados pelo Llama Index, o QDrant, um banco de dados vetorial de código aberto e um mecanismo de busca de similaridade de vetores, foi a escolha ideal devido a sua integração com o Llama Index, além de possuir uma otimização para buscas por similaridade e oferecer recursos avançados de processamento de linguagem natural e recuperação de informações.

3.2.3 Interface Web

Para o desenvolvimento da interface web do chatbot, buscou-se uma ferramenta que permitisse a criação de uma interface intuitiva e fácil de usar com foco na experiência do usuário. O Streamlit se mostrou a escolha ideal, pois oferece componentes pré-construídos para chat, o que facilitou a implementação da interface de conversação. Com o Streamlit, foi possível desenvolver uma interface que permite aos usuários interagirem com o chatbot de forma natural, enviando mensagens e recebendo respostas na mesma tela e ao mesmo tempo visualizando o histórico da conversa. Outra característica importante foi a facilidade de se basear em sua documentação, onde foi possível explorar os mais diversos componentes nativos no projeto.

3.2.4 Comunicação entre Componentes

Para integrar as funcionalidades do *Backend*, onde o LLama Index e o Qdrant processam a base de conhecimento, e a interface web do Streamlit utilizada pelo usuário, foi necessário desenvolver um método de comunicação entre os componentes e para isso foi desenvolvida uma API (Interface de Programação de Aplicações). Para essa comunicação, decidiu-se utilizar o framework FastAPI, que se destacou por sua alta performance no processamento das requisições, facilidade de uso com uma sintaxe clara e integração com o Unicorn, um servidor web ASGI que permite a execução do chatbot em um ambiente de produção robusto e escalável.

3.2.5 Ambiente de Execução com Docker

A necessidade de garantir a modularidade, a portabilidade e a escalabilidade da aplicação levou a escolha do Docker como ferramenta de conteinerização. O uso do Docker permite empacotar aplicações e suas dependências em contêineres, sendo unidades de software leves e portáteis que podem ser executadas em qualquer ambiente que suporte a plataforma Docker. Além disso, a utilização do Docker oferece recursos de escalabilidade, permitindo que a aplicação seja facilmente replicada para atender a um grande número de usuários conforme a necessidade.

4
Resultados

Este capítulo apresenta os resultados obtidos com o desenvolvimento do chatbot, incluindo a descrição da arquitetura do sistema, a apresentação da interface do usuário e a análise dos testes realizados.

4.1
Arquitetura do Sistema

O chatbot foi projetado com uma arquitetura modular, composto por componentes distintos que interagem entre si para fornecer as funcionalidades completas para o seu funcionamento. A figura 4.1 ilustra a arquitetura final do sistema, mostrando os principais componentes e suas interações. São 4 componentes conteinerizados, um contendo a WikiJs, outro contendo o banco de dados PostgreSQL necessário para o funcionamento da WikiJS, outro contendo o *Backend*, onde está implementado o processamento dos arquivos com a utilização do Llama Index junto com o *framework* da FastAPI e, por último, um contêiner para a interface web construída com o StreamLit. O Qdrant, também representado na ilustração da arquitetura do sistema, teve que ser tomada uma importante decisão na sua infraestrutura, pois, apesar de ser um banco de dados de código aberto e ser possível a sua utilização em um ambiente conteinerizado, foi optado por utilizar a sua arquitetura de nuvem com o Qdrant Cloud. Esta decisão levou em conta o objetivo final em termos uma aplicação completa de fácil portabilidade no acesso dos dados previamente indexados no banco, sendo possível acessar de origens diferentes, por isso é importante ressaltar que, apesar de estar representado na imagem da arquitetura do sistema, ele não se encontra na arquitetura conteinerizada local.

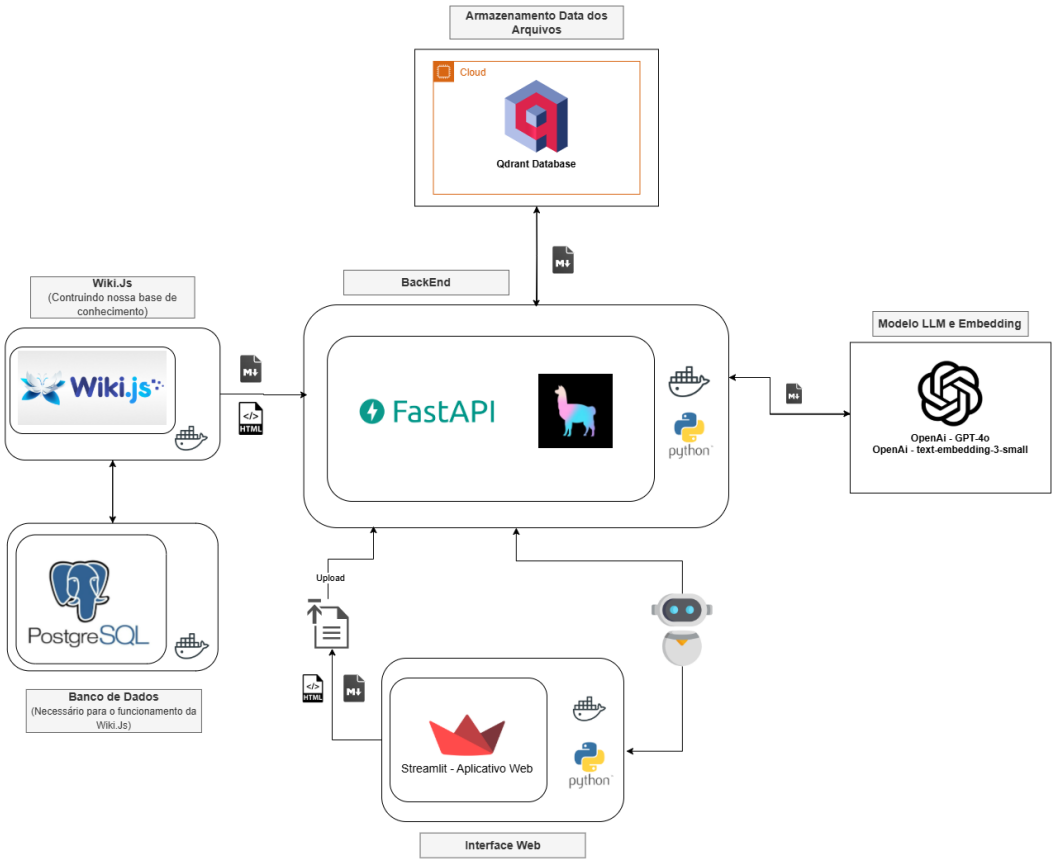

Figura 4.1: Arquitetura do Sistema

4.2 Plataforma Colaborativa

No desenvolvimento do chatbot, a construção da base de conhecimento se mostrou um importante requisito do projeto e, devido aos critérios já justificados anteriormente, foi feita a escolha de utilizar a plataforma do WikiJS. Na sua implementação, chamou a atenção a sua fácil configuração e logo após o *build* de sua imagem Docker, em seu primeiro acesso da URL já é possível cadastrar o usuário administrador e o nome da plataforma. Um exemplo de página inicial pode ser visto na figura 4.2 abaixo:

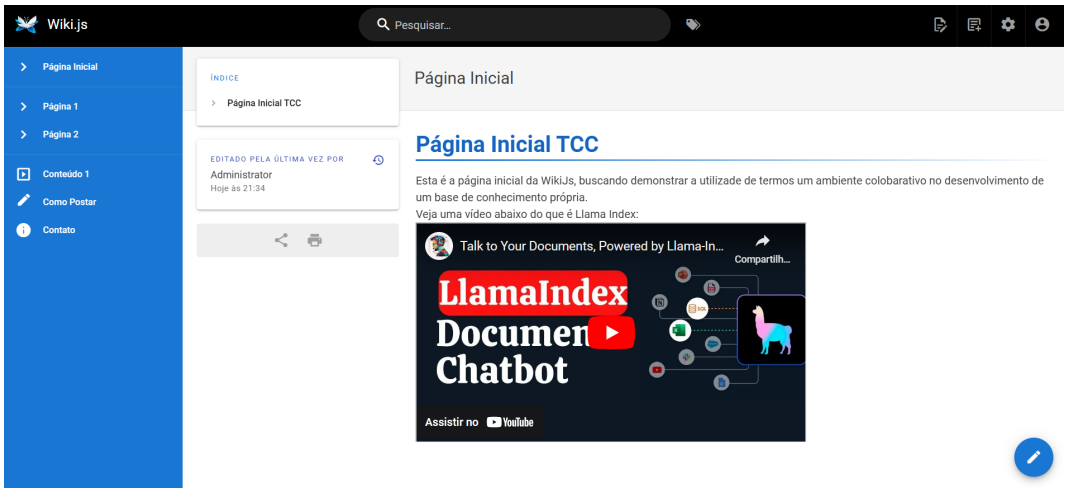

Figura 4.2: Página Inicial da WikiJS

Uma configuração que é crucial para o projeto e a integração com o *backend* é a definição do armazenamento (*Backup*) dos dados. Esta funcionalidade salva em um repositório local toda nova página presente na Wiki e suas alterações subsequentes. Este repositório está sendo compartilhado com o *backend*, a partir de configurações de volume entre o contêiner do *backend* com o da WikiJS. Esta foi a maneira de integração utilizada para os arquivos serem utilizados na nossa base de conhecimento. O exemplo de configuração pode ser encontrado na figura 4.3.

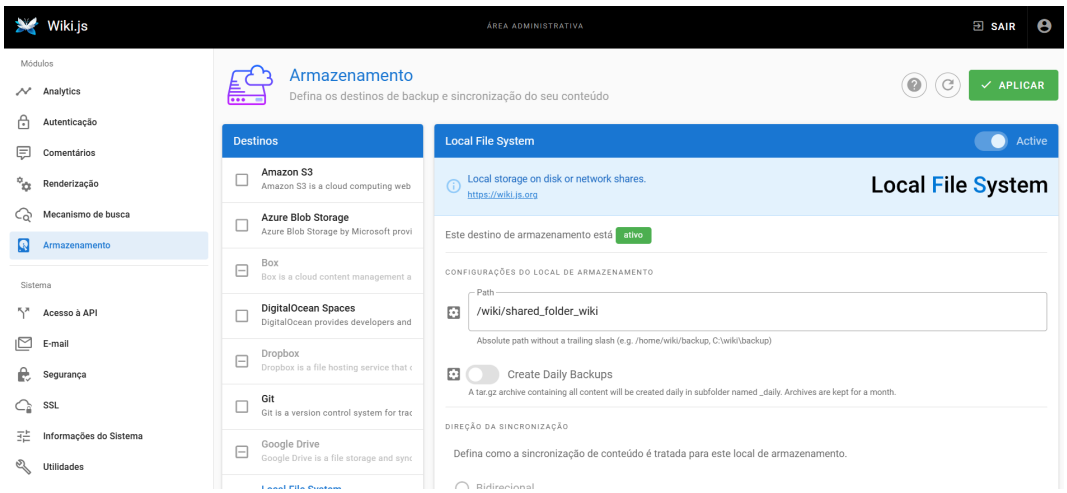

Figura 4.3: Configuração de Backup da WikiJS

4.3 Interface do Usuário

A interface do usuário do chatbot foi projetada com o objetivo de ser intuitiva e fácil de ser utilizada. A figura 4.4 apresenta o resultado final da interface inicial do chatbot, onde o usuário pode interagir com o sistema ao

entrar no site. Logo na página inicial já temos as instruções como utilizar o chatbot.

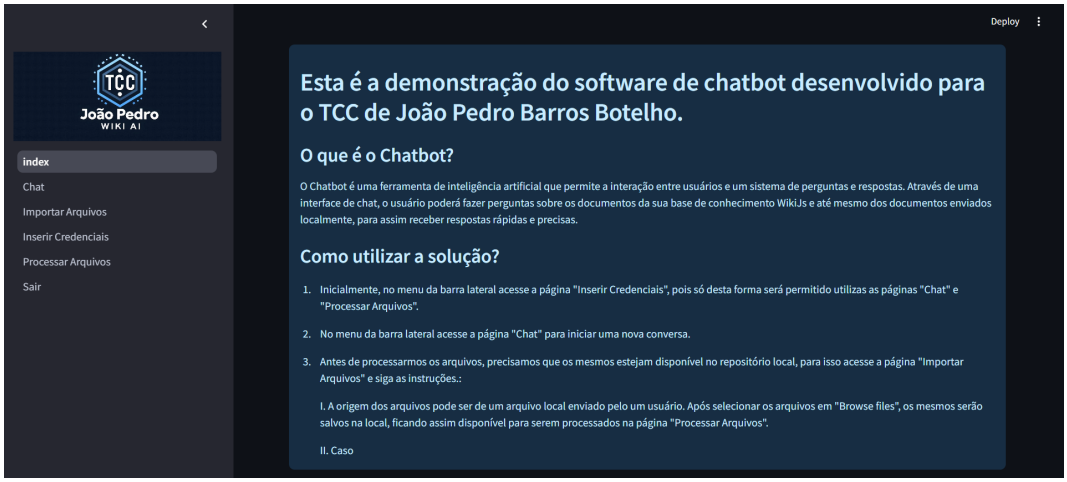

Figura 4.4: Página Inicial

Para o funcionamento do chatbot é necessário definir qual *API Key* do OpenAI e qual banco Qdrant Cloud será utilizado. Pensando nisso, foi desenvolvida uma página para definição destes parâmetros, onde é aberta a possibilidade de trocar o banco de dados vetorial utilizado conforme necessidade, além de ser possível também a troca da API Key da OpenAI. A API Key é utilizada no processo de autenticação dos serviços da OpenAI, onde é necessário no nosso projeto na criação dos índices dos arquivos de texto. No ambiente produtivo, estes parâmetros devem ser definidos como variável de ambiente no *Backend*, porém em um ambiente de desenvolvimento, ter uma maneira fácil de trocar estes parâmetros é útil.

Figura 4.5: Página de Inserir Credenciais

Com as variáveis da API Key do OpenAI e do Qdrant Cloud definidas, podemos acessar a página que tem a função de processar os arquivos. Temos três opções nessa página para aumentar a nossa base de conhecimento, que

são os arquivos da Wiki, os arquivos enviados pelo usuário por *upload* ou até mesmo arquivos de um repositório S3 da AWS. Foi adicionada também uma funcionalidade para visualizar os arquivos já processados. O resultado final da página pode ser encontrado na figura 4.6.

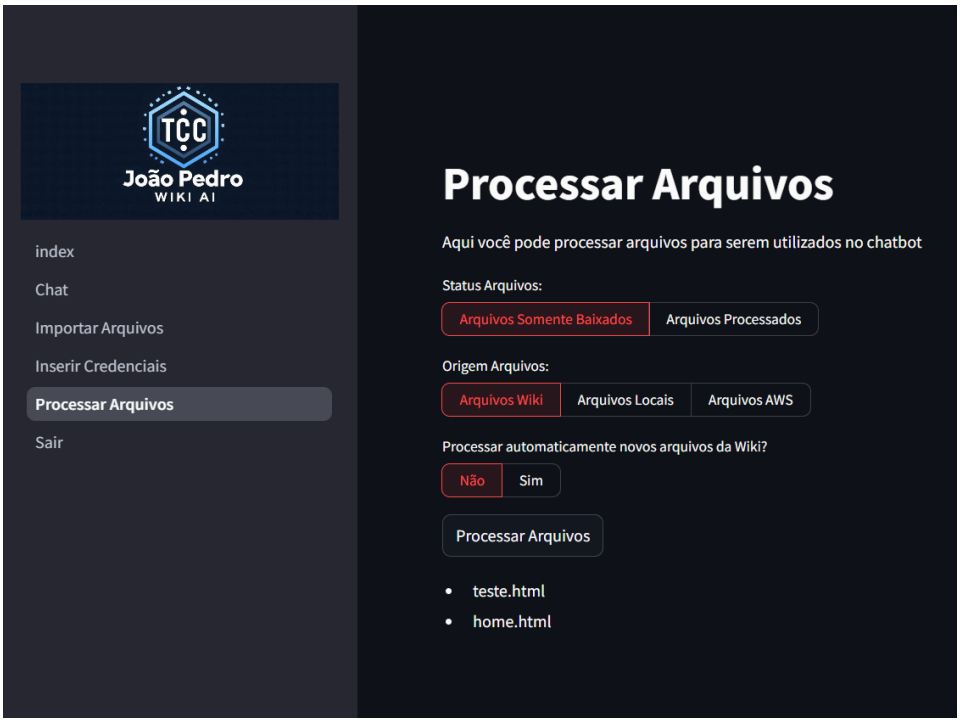

Figura 4.6: Página Processar Arquivos

Por fim, temos o resultado da página onde são feitas as interações de chat do usuário com a IA. Ao entrar na página, o chatbot estará esperando a primeira pergunta do usuário e cada nova pergunta é mantida no histórico de conversação e seu contexto é utilizado ainda na conversa. É possível iniciar uma nova conversa clicando no botão "Novo Chat". Também no canto inferior esquerdo, é possível selecionar a origem dos dados, como padrão são utilizados os dados dos arquivos da Wiki, porém se queremos falar sobre os arquivos que foram enviados manualmente pelo usuário ou sobre os arquivos importados do S3 da AWS somente é necessário alterar na seleção. A figura 4.7 demonstra o resultado final da página.

Figura 4.7: Página Chatbot

4.4 Implementação do Chatbot

Nesta sessão vamos abordar os principais códigos e estratégias para o desenvolvimento do chatbot. O chatbot foi implementado utilizando a linguagem de programação Python e diversas bibliotecas, como llama_index, qdrant_client, openai, boto3, fastapi, Streamlit, entre outras. O código foi organizado em arquivos com funções separadas, como pré-processamento de texto, geração de embeddings, indexação de documentos e interface do usuário.

4.4.1 Pré-processamento de Texto

O módulo de pré-processamento de texto foi responsável por limpar e preparar os dados textuais dos arquivos para a base de conhecimento com geração dos vetores. Entre as funcionalidades da limpeza, inclui-se o download das imagens e a conversão das mesmas para Base64, estratégia esta para que as imagens sejam levadas em consideração no contexto da indexação dos dados. Todos os arquivos finais ficam no formato Markdown, isso se deve ao fato da indexação dos dados ser feita de maneira mais otimizada em uma estrutura simplificada como a do Markdown, retirando uma complexidade da análise da estrutura, focando apenas no texto bruto. O pseudocódigo pode ser encontrado no Algoritmo 1.

Algoritmo 1: Processar Arquivos para Markdown

```
1 Pseudocódigo para processamento de arquivos HTML e Markdown
2
3 1. Função: listarArquivosExistentes(caminhoDownload)
4    Entrada: caminhoDownload
5    Processo:
```

```
6         - Tente:
7             - Retorne a lista de arquivos no diretório '
                  caminhoDownload'
8         - Caso contrário:
9             - Exiba mensagem de erro "Diretório não encontrado"
10            - Retorne lista vazia
11    Retorno: lista de arquivos ou lista vazia
12
13 2. Função: separarArquivosPorTipo(arquivos)
14    Entrada: arquivos
15    Processo:
16        - Inicialize listas 'arquivosHTML' e 'arquivosMarkdown'
17        - Para cada arquivo em 'arquivos':
18            - Se o arquivo terminar com ".html", adicione à lista '
                  arquivosHTML'
19            - Caso contrário, se terminar com ".md", adicione à lista
                   'arquivosMarkdown'
20    Retorno: listas 'arquivosHTML' e 'arquivosMarkdown'
21
22 3. Função: processarConteudoHTML(arquivosHTML, caminhoProjeto,
      caminhoDownload, decideCaminho)
23    Entrada: arquivosHTML, caminhoProjeto, caminhoDownload,
          decideCaminho
24    Processo:
25        - Crie o diretório "Pre_Processados" se não existir
26        - Para cada arquivo em 'arquivosHTML':
27            - Leia o conteúdo HTML do arquivo
28            - Converta o conteúdo HTML para Markdown
29            - Substitua links de imagem por dados em base64, se
                  necessário
30            - Salve o conteúdo processado no diretório "
                  Pre_Processados"
31            - Se 'decideCaminho' for diferente de "wiki":
32                - Exclua o arquivo HTML original
33    Retorno: nenhum
34
35 4. Função: processarArquivosMarkdown(arquivosMarkdown,
      caminhoProjeto, caminhoDownload, decideCaminho)
36    Entrada: arquivosMarkdown, caminhoProjeto, caminhoDownload,
          decideCaminho
37    Processo:
38        - Crie o diretório "Pre_Processados" se não existir
39        - Para cada arquivo em 'arquivosMarkdown':
40            - Leia o conteúdo Markdown do arquivo
41            - Substitua links de imagem por dados em base64, se
                  necessário
42            - Salve o conteúdo processado no diretório "
                  Pre_Processados"
43            - Se 'decideCaminho' for diferente de "wiki":
44                - Exclua o arquivo Markdown original
45    Retorno: nenhum
```

```
46
47 5. Função: iniciarProcessamento(decideCaminho)
48    Entrada: decideCaminho
49    Processo:
50        - Determine os caminhos do projeto e de download baseados em
              'decideCaminho'
51        - Liste os arquivos existentes no diretório de download
52        - Separe os arquivos em listas de HTML e Markdown
53        - Processe as listas de HTML e Markdown
54    Retorno: nenhum
55
56 6. Função: iniciarProcessamentoAWS(bucket, credenciaisAWS)
57    Entrada: bucket, credenciaisAWS
58    Processo:
59        - Configure o cliente S3 com as credenciais AWS
60        - Liste os arquivos existentes no diretório de download
61        - Separe os arquivos em listas de HTML e Markdown
62        - Processe as listas de HTML e Markdown, integrando imagens
              do S3
63    Retorno: nenhum
64
65 7. Função: principal()
66    Processo:
67        - Configure variáveis de acesso AWS e o nome do bucket (caso
              necessário)
68        - Liste os arquivos existentes no diretório de download
69        - Separe os arquivos em listas de HTML e Markdown
70        - Processe as listas de HTML e Markdown, integrando imagens
              do S3, se necessário
71    Retorno: nenhum
72
73 8. Se este arquivo for executado diretamente:
74    - Execute a função 'principal'
```

4.4.2 Tratamento das Imagens

Um dos problemas encontrados durante o desenvolvimento do projeto foi que muitos arquivos possuem imagens que têm o seu conteúdo relevante na interpretação da imagem junto com o texto como um todo. A solução encontrada foi processar as imagens que agora estão no formato base64 dos arquivos e pedir uma interpretação das mesmas para o GPT da OpenAI, substituindo as imagens nos arquivos pela sua interpretação textual. A requisitação é feita por chamada API na função image_converter_with_OpenAI.py, o pseudocódigo pode ser visto no Algoritmo 2.

Algoritmo 2: Converter Imagens em Texto

```
Pseudocódigo para Processar Arquivos de Imagem com Base64 e OpenAI

1. Função: decodificarImagemBase64(base64_string)
   Entrada: base64_string
   Processo:
       - Decodificar a string Base64 em dados binários
   Retorno: dados binários da imagem

2. Função: gerarDescricaoImagem(imagem)
   Entrada: imagem (em Base64)
   Processo:
       - Criar cliente da API OpenAI
       - Enviar imagem para a API utilizando o modelo GPT-4o com uma
           mensagem   solicitando descrição
       - Receber resposta da API contendo a descrição
   Retorno: descrição gerada pela API

3. Função: formatarDescricaoParaMarkdown(descricao)
   Entrada: descricao
   Processo:
       - Criar uma string em formato Markdown com a descrição
          fornecida
   Retorno: string formatada em Markdown

4. Função: processarArquivosImagem(caminhoProjeto)
   Entrada: caminhoProjeto
   Processo:
       - Definir diretórios para pré-processados e processados
       - Criar o diretório 'Arquivos_Processados' se não existir
       - Listar arquivos Markdown no diretório 'Pre_Processados'
       - Para cada arquivo Markdown:
           a. Ler o conteúdo do arquivo
           b. Procurar por imagens em Base64 usando Regex
           c. Para cada imagem encontrada:
               - Decodificar imagem e enviar para a função '
                  gerarDescricaoImagem'
               - Substituir a imagem pela descrição gerada
           d. Salvar o conteúdo modificado no diretório '
              Arquivos_Processados'
           e. (Opcional) Excluir o arquivo Markdown original
   Retorno: nenhum

5. Função: definirChaveAPI(chaveAPI)
   Entrada: chaveAPI
   Processo:
       - Armazenar a key de API do OpenAI na variável de ambiente do
           sistema
   Retorno: nenhum

6. Função: iniciarProcessamentoArquivosImagem(chaveAPI)
   Entrada: chaveAPI
```

```
    Processo:
        - Obter caminho(path) do projeto
        - Configurar chave da API OpenAI usando 'definirChaveAPI'
        - Chamar a função 'processarArquivosImagem'
    Retorno: nenhum

7. Função: principal()
    Processo:
        - Obter caminho(path) do projeto
        - Configurar chave da API OpenAI
        - Processar os arquivos de imagem no diretório do projeto
    Retorno: nenhum

8. Se este arquivo for executado diretamente:
    - Chamar a função 'principal'
```

4.4.3 Indexação dos Dados e ChatBot

Vamos abordar agora o código process_llama_qdrant.py, o mesmo tem diversas funções, desde lidar com o histórico de conversas, até mesmo por ser o responsável de fazer as perguntas do chatbot. É neste código que a indexação dos dados é realizada utilizando o Llama Index e criando representações vetoriais (*embeddings*) para cada um dos arquivos da base de conhecimento, onde após o processo são então armazenadas em uma coleção no QDrant. O pseudocódigo está presente no Algoritmo 3.

Algoritmo 3: Indexação dos Arquivos e Recuperação dos Dados

```
Pseudocódigo para Processar e Consultar Documentos com Llama Index,
    Qdrant e OpenAI

1. Função: carregarOuCriarChatStore(caminhoArquivo)
    Entrada: caminhoArquivo
    Processo:
        - Se o arquivo de histórico do chat existir:
            - Carregar o histórico do arquivo
        - Caso contrário:
            - Criar um novo histórico de chat
    Retorno: objeto SimpleChatStore

2. Função: definirChaveAPI(openaiApiKey)
    Entrada: openaiApiKey
    Processo:
        - Definir a variável de ambiente 'OPENAI_API_KEY' com o valor
            fornecido
    Retorno: nenhum

3. Função: inicializarOuCriarColecao(qdrantClient, nomeColecao)
```

```
    Entrada: qdrantClient, nomeColecao
    Processo:
        - Verificar se o cliente ou nome da coleção estão definidos
        - Obter as coleções existentes do Qdrant
        - Se a coleção não existir:
            - Criar a coleção com parâmetros específicos
        - Inicializar o índice global com a coleção existente
    Retorno: índice global

4. Função: enviarDocumentosParaQdrant()
    Processo:
        - Obter variáveis de ambiente para a URL, sendo elas a chave
          da API e o nome da coleção Qdrant
        - Verificar se a coleção existe, caso contrário, criá-la
        - Carregar os documentos do diretório 'Arquivos_Processados'
        - Configurar o pipeline de ingestão para processar os
          documentos
        - Processar e indexar os documentos no Qdrant
    Retorno: verdadeiro se bem-sucedido, erro caso contrário

5. Função: consultarComPergunta(pergunta, nomeColecao, qdrantUrl,
    qdrantApiKey, usuarioChat, arquivoHistorico)
    Entrada: pergunta, nomeColecao, qdrantUrl, qdrantApiKey,
       usuarioChat, arquivoHistorico
    Processo:
        - Inicializar cliente e índice Qdrant, se necessário
        - Carregar ou criar o histórico de chat
        - Configurar memória de chat com histórico
        - Criar mecanismo de chat híbrido (similaridade e consulta
          vetorial)
        - Enviar pergunta para o mecanismo e obter resposta
        - Persistir o histórico atualizado em arquivo
    Retorno: resposta da pergunta

6. Função: definirVariaveisQdrant(nomeColecao, qdrantUrl,
    qdrantApiKey)
    Entrada: nomeColecao, qdrantUrl, qdrantApiKey
    Processo:
        - Definir as variáveis de ambiente para a coleção, URL e
          chave de API do Qdrant
    Retorno: nenhum

7. Função: limparVariaveisGlobais()
    Processo:
        - Redefinir as variáveis globais para 'None'
    Retorno: nenhum

8. Função: principal()
    Processo:
        - Definir a chave da API OpenAI
        - Definir as variáveis do Qdrant (URL, chave da API, coleção)
```

```
        - Enviar documentos para o Qdrant
        - Exibir resposta processada (opcional)
    Retorno: nenhum

9. Se este arquivo for executado diretamente:
    - Executar a função 'principal'
```

4.4.4 Chamadas da API

Por último, temos a função responsável pela orquestração de todos os códigos, onde temos todas as chamadas da API que foram desenvolvidas para o chatbot. O código chamadas_api.py utiliza o FastAPI e define desde os modelos de requisição até cada um dos chamados que são utilizados pela interface web do Streamlit. O pseudocódigo está representado no Algoritmo 4.

Algoritmo 4: Chamadas API

```
Pseudocódigo para API de Processamento com FastAPI

1. Inicialização da API
    - Importar bibliotecas e módulos necessários
    - Criar uma instância do FastAPI

2. Definição dos Modelos de Requisição
    - RequestModel: 'param1: string', 'param2: inteiro'
    - StringModel: 'name: string'
    - APIKeyModel: 'openai_api_key: string'
    - S3DownloadModel: 'bucket_name', 'aws_region_name', '
        aws_access_key_id', 'aws_secret_access'
    - QdrantModel: 'collection_name', 'qdrant_url', 'qdrant_api_key'
    - QuestionModel: 'question', 'collection'
    - ListModel: 'names: lista de strings'
    - ProcessModel: 'collection', 'recorrente'

3. Endpoints da API

    3.1 Variáveis Qdrant
    - POST '/api/set_qdrant_variables':
        - Define variáveis de ambiente relacionadas ao Qdrant
        - Retorna sucesso ou erro

    - GET '/api/get_exist_Qdrant_env':
        - Verifica se variáveis de ambiente do Qdrant estão definidas
        - Retorna status

    - POST '/api/set_collection_name':
        - Define o nome da coleção Qdrant
        - Retorna sucesso
```

```
31
32  3.2 API Key OpenAI
33  - POST '/api/set_api_key':
34      - Define a chave da API OpenAI
35      - Retorna sucesso
36
37  - GET '/api/get_exist_API_Key':
38      - Verifica se a chave da API OpenAI está definida
39      - Retorna status
40
41  3.3 Credenciais AWS
42  - POST '/api/set_aws_credentials':
43      - Define as credenciais da AWS no ambiente
44      - Retorna sucesso
45
46  - GET '/api/get_status_env_aws':
47      - Verifica se as variáveis de ambiente da AWS estão definidas
48      - Retorna status
49
50  - POST '/api/set_aws_bucket':
51      - Define o nome do bucket AWS
52      - Retorna sucesso
53
54  - DELETE '/api/clear_aws_env_variables':
55      - Remove as variáveis de ambiente relacionadas à AWS
56      - Retorna sucesso
57
58  3.4 Download de Arquivos do S3
59  - POST '/api/download_from_s3':
60      - Faz download de arquivos específicos do S3
61      - Retorna status e lista de arquivos
62
63  - POST '/api/download_all_from_s3':
64      - Faz download de todos os arquivos do S3
65      - Retorna status e lista de arquivos
66
67  3.5 Processamento de Arquivos
68  - POST '/api/process_files_basic':
69      - Processa arquivos locais e integra com Qdrant
70      - Retorna status ou erro
71
72  - POST '/api/process_files_aws':
73      - Processa arquivos baixados do S3 e integra com Qdrant
74      - Retorna status ou erro
75
76  - GET '/api/process_image_files':
77      - Processa imagens usando OpenAI
78      - Retorna status
79
80  3.6 Qdrant e Perguntas
81  - GET '/api/documents_to_qdrant':
```

```
82        - Envia documentos para o Qdrant
83        - Retorna status
84
85    - POST '/api/question':
86        - Consulta o Qdrant usando uma pergunta
87        - Retorna a resposta da consulta
88
89    3.7 Listagem de Arquivos
90    - GET '/api/get_files_downloaded':
91        - Lista arquivos baixados do S3
92
93    - GET '/api/get_files_local':
94        - Lista arquivos locais
95
96    - GET '/api/get_files_wiki':
97        - Lista arquivos da Wiki
98
99    - GET '/api/get_files_processed':
100       - Lista arquivos já processados
101
102   3.8 Limpeza de Variáveis de Ambiente
103   - DELETE '/api/clear_env_variables':
104       - Remove todas as variáveis de ambiente configuradas
105       - Retorna sucesso
106
107 4. Execução Principal
108   - Define as variáveis e configurações necessárias
109   - Executa a API FastAPI
```

4.5 Testes Realizados

Para avaliar o desempenho do chatbot foram realizados testes com perguntas diversas, além dos testes das funcionalidades de maneira individual. Os testes foram divididos em três categorias principais:

- Testes de Funcionalidade: Avaliação das funcionalidades básicas do chatbot, como a capacidade de responder a perguntas simples, processar arquivos enviados pelo usuário e além de acessar informações da base de conhecimento com a conexão junto ao Qdrant Cloud.
- Testes de Usabilidade: Avaliação da interface do usuário, verificando se ela é intuitiva e fácil de usar.
- Testes de Desempenho: Avaliação do tempo de resposta do chatbot e da comunicação entre a interface web e o *backend*.

Os testes foram realizados com perguntas em cima de diferentes assuntos presentes na base de conhecimento simulada com mais de 40 arquivos presentes.

Os resultados dos testes foram utilizados para identificar os pontos fortes e fracos do chatbot, mostrando que perguntas ambíguas podem levar a ter respostas de um documento que não era o desejado. Além disso, o tempo de resposta foi elevado devido a contarmos com uma arquitetura em nuvem do Qdrant Cloud, porém em um ambiente produtivo o Qdrant poderia ser instanciado localmente, o que não seria um impeditivo para o projeto.

5
Conclusão e trabalhos futuros

Com base nos requisitos do projeto, todos os objetivos foram alcançados com sucesso. Este trabalho explorou a construção de um chatbot utilizando IA generativa, com foco na criação de uma base de conhecimento própria e na integração de diversas tecnologias. A pesquisa e a implementação demonstraram a viabilidade da aplicação da IA generativa em chatbots personalizados, respondendo a perguntas com base em informações restritas a um determinado ambiente.

O desenvolvimento do chatbot envolveu a seleção e integração de diversas ferramentas como WikiJs, Llama Index, QDrant, FastAPI e Streamlit, cada uma com suas justificativas. A WikiJs se mostrou eficaz na construção da base de conhecimento, enquanto o Llama Index e o QDrant foram essenciais para a indexação e consulta eficiente dos dados. O FastAPI e o Streamlit facilitaram a comunicação entre os componentes e a criação da interface web.

5.1
Próximos Passos

Apesar dos resultados promissores, o chatbot apresenta limitações, como a dependência da qualidade da base de conhecimento, a interpretação das imagens junto aos textos e a dificuldade em lidar com perguntas complexas. Outra limitação é utilizar um misto de informações entre a base de conhecimento privada e as informações de domínio público. A implementação deste trabalho em produção necessita de ajustes como definições das variáveis do Qdrant Cloud e da OpenAI em forma de variável de ambiente. Trabalhos futuros podem aprimorar o chatbot com a expansão da base de conhecimento, além do refinamento do modelo de linguagem e a implementação de mecanismos de aprendizado contínuo, como feedback por parte dos usuários.

O trabalho contribui para o avanço do conhecimento na área de IA generativa, não somente em demonstrar a viabilidade da construção de chatbots personalizados, mas também um incentivo para serem construídos diversos chatbots para diferentes domínios e necessidades. As ferramentas e técnicas utilizadas podem ser replicadas em projetos futuros, pois todas as ferramentas utilizadas neste projeto são de código aberto.

.

6
Referências bibliográficas

Alura. **Quais são os tipos de Inteligência Artificial (IA)? Exemplos e características**. 2023. Disponível em: https://www.alura.com.br/artigos/tipos-inteligencia-artificial-ia. Acesso em: 2024-11-15. Citado 2 vezes nas páginas 16 e 17.

CloudFlare. **O que são embeddings no aprendizado de máquina?** (s.d.). Disponível em: https://www.cloudflare.com/pt-br/learning/ai/what-are-embeddings/. Acesso em: 2024-11-05. Citado na página 20.

DataStax. **What is LlamaIndex?** 2023. Disponível em: https://www.datastax.com/guides/what-is-llamaindex. Acesso em: 2024-11-04. Nenhuma citação no texto.

Elastic. **O que são embeddings de palavras?** (s.d.). Disponível em: https://www.elastic.co/pt/what-is/word-embedding. Acesso em: 2024-11-05. Citado na página 20.

ERTEL, W. **Introduction to artificial intelligence**. [S.l.]: Springer Nature, 2024. Citado na página 16.

hostinger. **O Que é Docker e Como Ele Funciona? – Docker Explicado**. 2024. Disponível em: https://www.hostinger.com.br/tutoriais/o-que-e-docker. Acesso em: 2024-11-21. Citado na página 23.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **Information technology – Artificial intelligence – Artificial intelligence concepts and terminology**. [S.l.], 2022. Citado na página 12.

Qdrant. **O que é Qdrant?** 2024. Disponível em: https://qdrant.tech/documentation/overview/#. Acesso em: 2024-11-04. Citado na página 21.

redhat. **O que é API?** 2023. Disponível em: https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces. Acesso em: 2024-11-10. Citado na página 22.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: A modern approach**. [S.l.]: Prentice Hall, 1995. Citado na página 12.

treinaweb. **O que é FastAPI?** (s.d.). Disponível em: https://www.treinaweb.com.br/blog/o-que-e-fastapi. Acesso em: 2024-11-10. Nenhuma citação no texto.

WANG, P. On defining artificial intelligence. **Journal of Artificial General Intelligence**, De Gruyter Poland, v. 10, n. 2, p. 1–37, 2019. Citado na página 16.