INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DE MINAS GERAIS (IFMG)
CAMPUS BAMBUÍ
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

Vinícius Perboar dos Santos Madruga

RECUPERAÇÃO DE INFORMAÇÕES DE UM SISTEMA ERP POR MEIO DA INTERPRETAÇÃO DE LINGUAGEM NATURAL UTILIZANDO RAG E BANCO DE DADOS EM GRAFOS

Bambuí – MG
2026

VINÍCIUS PERBOAR DOS SANTOS MADRUGA

RECUPERAÇÃO DE INFORMAÇÕES DE UM SISTEMA ERP POR MEIO DA INTERPRETAÇÃO DE LINGUAGEM NATURAL UTILIZANDO RAG E BANCO DE DADOS EM GRAFOS

Trabalho de conclusão de curso apresentado ao Curso de Bacharelado em Engenharia de Computação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais (IFMG) – *Campus* Bambuí para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Eduardo Cardoso Melo

Bambuí – MG
2026

Catalogação na Fonte Biblioteca IFMG - Campus Bambuí

M183r Madruga, Vinicius Perboar dos Santos.
Recuperação de informações de um sistema ERP por meio da interpretação de linguagem natural utilizando RAG e banco de dados em grafos. / Vinicius Perboar dos Santos Madruga. – 2026.
87 f.; il.: color.

Orientador: Prof. Dr. Eduardo Cardoso Melo.
Trabalho de Conclusão de Curso (graduação) - Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Bambuí, MG, Curso Bacharelado em Engenharia de Computação, 2026.

1. Planejamento de recursos empresariais (ERP). 2. Geração aumentada de recuperação (RAG). 3. Protótipo. I. Melo, Eduardo Cardoso. II. Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Bambuí, MG. III. Título.

CDD 004.64

Elaborada por Douglas Bernardes de Castro- CRB-6/2802

Vinícius Perboar dos Santos Madruga

RECUPERAÇÃO DE INFORMAÇÕES DE UM SISTEMA ERP POR MEIO DA INTERPRETAÇÃO DE LINGUAGEM NATURAL UTILIZANDO RAG E BANCO DE DADOS EM GRAFOS

Trabalho de conclusão de curso apresentado ao Curso de Bacharelado em Engenharia de Computação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais (IFMG) – *Campus* Bambuí para obtenção do grau de Bacharel em Engenharia de Computação.

Aprovado em 14 de Maio de 2026 pela banca examinadora:

Prof. Dr. Eduardo Cardoso Melo – IFMG – *Campus* Bambuí – (Orientador)
Prof. Me. Felipe Lopes de Melo Faria – IFMG - *Campus* Bambuí
Prof. Dr. Rogério Amaral Bonatti – IFMG - *Campus* Bambuí

RESUMO

A efetividade no processo decisório empresarial está diretamente relacionada à disponibilidade de informações confiáveis e bem estruturadas. Nesse contexto, sistemas de Planejamento de Recursos Empresariais (ERP) desempenham papel fundamental ao centralizar grandes volumes de dados estratégicos, facilitando sua consulta e análise. No entanto, a utilização desses sistemas ainda representa um desafio significativo para usuários sem conhecimento técnico, devido à necessidade de domínio de linguagens formais de consulta e de conceitos relacionados a bancos de dados. Esse cenário evidencia uma lacuna na forma do acesso aos dados, dificultando o acesso às informações e a tomada de decisões. Diante disso, este trabalho apresenta o desenvolvimento de um protótipo baseado na arquitetura de Geração Aumentada de Recuperação (RAG), capaz de interpretar consultas em linguagem natural por meio da integração entre modelos de linguagem e fontes externas de dados. A metodologia envolveu a modelagem de um cenário ERP simulado no banco de dados Neo4j, a definição de uma arquitetura orientada a serviços e a implementação de um orquestrador RAG responsável por interpretar consultas em linguagem natural, gerar consultas estruturadas e produzir respostas gráficas com base nos dados recuperados, com o suporte da biblioteca LangChain para integração com modelos de linguagem. Os resultados demonstraram que o sistema foi capaz de gerar respostas adequadas ao contexto, com apresentação automática no formato mais apropriado. Além disso, o uso de cache semântico contribuiu para a otimização do desempenho, reduzindo o tempo de execução e o custo computacional. O protótipo contribui ao ampliar o acesso às informações em sistemas ERP, reduzir a dependência de conhecimentos técnicos para sua utilização e tornar a análise de dados mais assertiva, impactando positivamente a eficiência organizacional e a agilidade na tomada de decisões.

Palavras-chave: Planejamento de Recursos Empresariais (ERP). Geração Aumentada de Recuperação (RAG). Protótipo. Tomada de decisões.

ABSTRACT

The effectiveness of business decision-making is directly related to the availability of reliable and well-structured information. In this context, Enterprise Resource Planning (ERP) systems play a fundamental role by centralizing large volumes of strategic data, facilitating their access and analysis. However, the use of these systems still represents a significant challenge for non-technical users, due to the need for knowledge of formal query languages and database-related concepts. This scenario reveals a gap in how data is accessed, making it difficult to retrieve information and support decision-making. In this context, this work presents the development of a prototype based on the Retrieval Augmented Generation (RAG) architecture, capable of interpreting natural language queries through the integration of language models and external data sources. The methodology involved modeling a simulated ERP scenario in the Neo4j graph database, defining a service-oriented architecture, and implementing a RAG orchestrator responsible for interpreting natural language queries, generating structured queries, and producing graphical responses based on the retrieved data, with the support of the LangChain library for integration with language models. The results demonstrated that the system was able to generate context-appropriate responses, with automatic presentation in the most suitable format. Additionally, the use of semantic caching contributed to performance optimization, reducing execution time and computational costs. The prototype contributes by expanding access to information in ERP systems, reducing the dependence on technical knowledge for their use, and making data analysis more accurate, positively impacting organizational efficiency and decision-making agility.

Keywords: Enterprise Resource Planning (ERP). Retrieval Augmented Generation (RAG). Prototype. Decision-making.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

ERP – *Enterprise Resource Planning*
LLM – *Large Language Model*
NLP – *Natural Language Processing*
RAG – *Retrieval Augmented Generation*
MRP – *Material Requirements Planning*
KG – *Knowledge Graph*
SQL – *Structured Query Language*
GoT – *Graph of Thoughts*
IIKM – *Interactive Industrial Knowledge Management*
SOA – *Service Oriented Architecture*
CLI – *Command Line Interface*
CORS – *Cross-Origin Resource Sharing*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo é apresentada a contextualização do ambiente no qual se insere o objeto de estudo do projeto, delimitando a área abordada, os principais desafios enfrentados e proposta da solução desenvolvida. Além disso, são descritos os objetivos do projeto, bem como sua justificativa, que evidencia suas potenciais contribuições.

1.1 Contextualização

No contexto socioeconômico atual, é possível perceber que empresas de todos os portes têm intensificado a busca por soluções tecnológicas capazes de promover maior eficiência no controle de suas operações (Souza; Braga, 2025). Impulsionadas pela globalização, pelo avanço da transformação digital e pelo aumento da competitividade nos mercados, organizações modernas reconhecem a importância de otimizar processos, reduzir custos e tomar decisões mais assertivas com base em dados integrados (Souza; Braga, 2025; Júnior *et al.*, 2024).

Nesse cenário, os Sistemas de Planejamento de Recursos Empresariais (ERP, do inglês, *Enterprise Resource Planning*) emergem como ferramentas estratégicas essenciais para garantir a sustentabilidade, a escalabilidade e a vantagem competitiva das empresas (Júnior *et al.*, 2024), assim os sistemas ERP são soluções tecnológicas que integram, em uma única plataforma, os dados de diversos setores de uma organização (Júnior *et al.*, 2024; Souza; Braga, 2025).

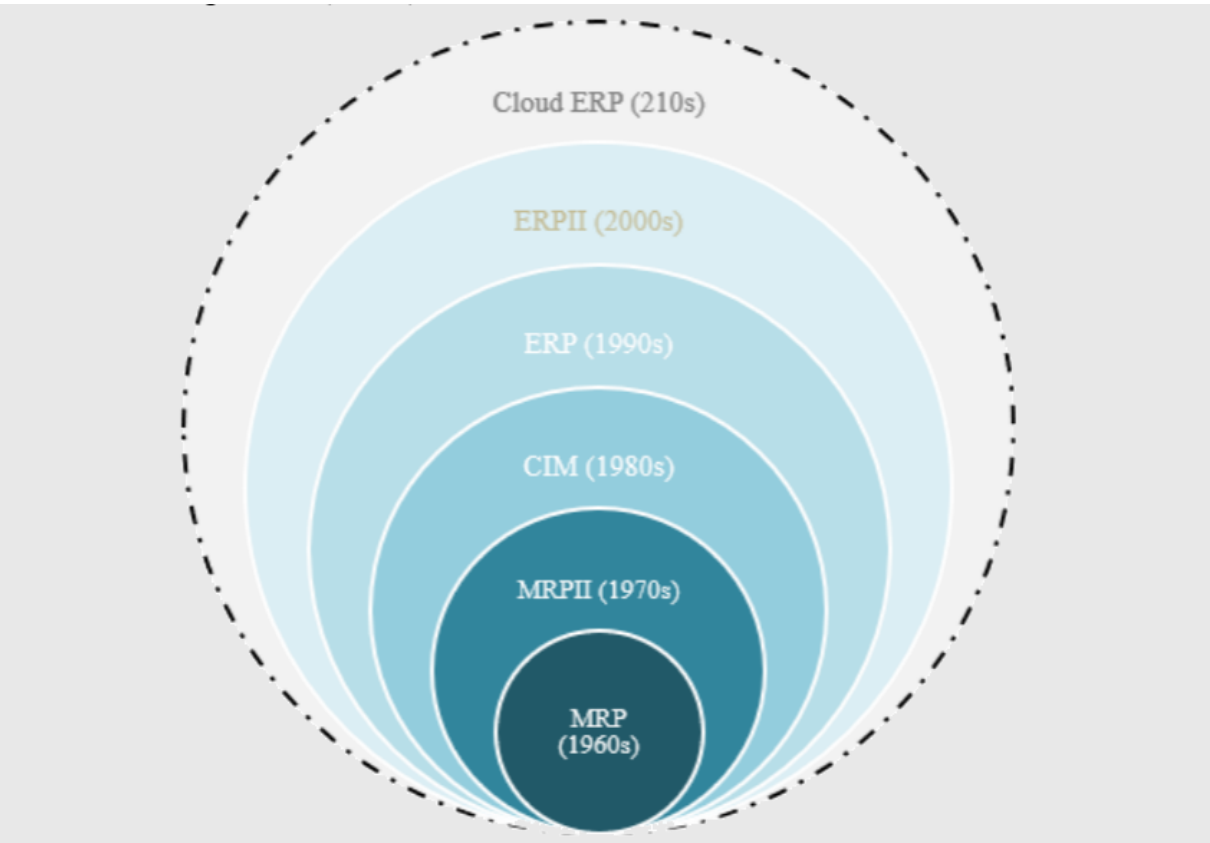
Um sistema ERP permite o gerenciamento de atividades como finanças, controle de estoque, recursos humanos, logística, dentre outras, eliminando redundâncias e promovendo a fluidez da informação pelos departamentos (Al-Amin *et al.*, 2023; Padilha; Marins, 2005; Júnior *et al.*, 2024; Souza; Braga, 2025). Essa integração garante maior consistência, padronização e confiabilidade dos dados corporativos, o que contribui diretamente para a melhoria na tomada de decisões (Asif; Alfrraj; Alshamari, 2022; Al-Amin *et al.*, 2023).

Os primeiros ERPs surgiram como uma evolução dos sistemas de Planejamento de Necessidades Materiais (MRP, do inglês, *Material Requirements Planning*), voltados ao controle de materiais (Padilha; Marins, 2005). Com o avanço da tecnologia da informação e o aumento da complexidade das operações empresariais, os ERPs passaram a incorporar outras áreas da gestão e a atender empresas de diversos segmentos e portes (Al-Amin *et al.*, 2023).

Conforme se observa na Figura 1, os ERPs atuais descendem de MRPs que tiveram sua concepção na década de 1960, evoluíram e se tornaram mais completos, mudaram de conceito e passaram a se chamar ERP na década de 1990, per-

durando até os dias atuais, mesmo que em constante evolução.

Figura 1 – Evolução do sistema ERP ao longo dos anos

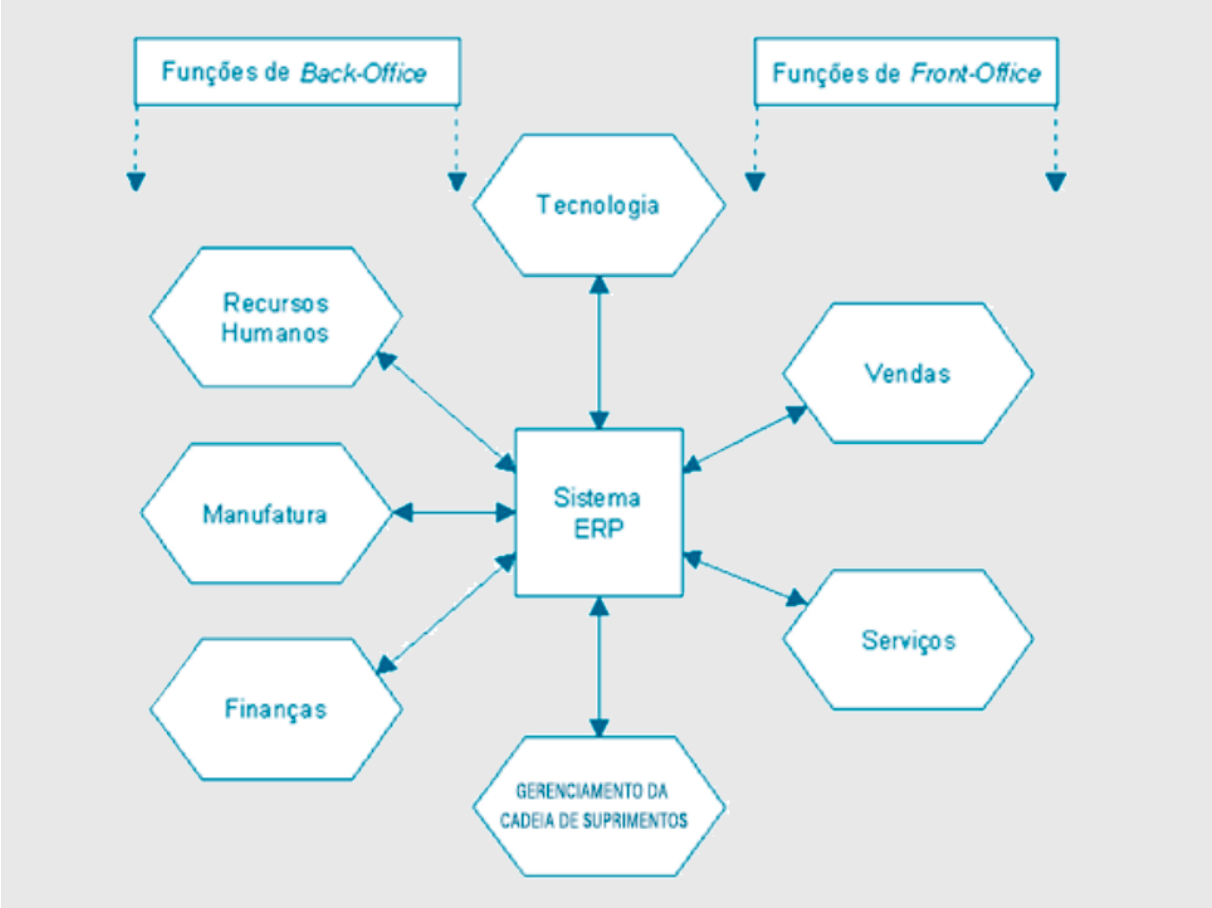


Fonte: Adaptado de (Al-Amin *et al.*, 2023).

Em tempos de transformação digital, decisões precisam ser tomadas com rapidez e baseadas em dados confiáveis (Júnior *et al.*, 2024). Nesse contexto, o ERP atua oferecendo uma visão holística do desempenho organizacional e permitindo que líderes empresariais acompanhem indicadores-chave em tempo real (Al-Amin *et al.*, 2023; Weerasekara; Gooneratne, 2023). Além disso, a automatização de tarefas operacionais reduz erros humanos, minimiza custos e libera os colaboradores para funções mais estratégicas (Weerasekara; Gooneratne, 2023; Souza; Braga, 2025).

A Figura 2 ilustra as áreas que o ERP pode abranger, tanto áreas internas da empresa, chamadas de *Back-Office*, que englobam gerenciamento de recursos humanos, manufatura e finanças, quanto as áreas externas da organização, chamadas de *Front-Office*, que englobam vendas e serviços oferecidos Padilha e Marins (2005). Além dessas áreas, o ERP também engloba tecnologia e gerenciamento da cadeia de suprimentos da empresa.

Figura 2 – Áreas de abrangência de um ERP



Fonte: Adaptado de (Padilha; Marins, 2005).

Dados de pesquisas mostram o quão vantajoso é para uma empresa adotar um sistema ERP para gerenciar seus negócios. De acordo com um levantamento da Oracle NetSuite (2023), entre as empresas que mantiveram seus sistemas ERP ativos por pelo menos um ano, 91% relataram otimização dos níveis de estoque e 78% registraram aumento de produtividade (Panorama Consulting Group, 2023). Além disso, 66% das organizações declararam que o ERP melhorou a eficiência das operações e 62% disseram que reduziram custos, especialmente em compras e controle de inventário (Parsimony, 2023).

Diante de comprovadas vantagens da adoção de um sistema ERP, é normal que essa abordagem ganhe popularidade no mundo organizacional. Diversos estudos de mercado evidenciam o crescimento do uso dos sistemas ERP nas empresas. De acordo com Market Research Future (2025) o valor estimado do mercado de ERP foi aproximadamente US$53,4 bilhões em 2023, com projeção de crescimento para cerca de US$100 bilhões até 2035.

Segundo dados da Statista (2024), a receita global com *software* ERP deve alcançar US$55,9 bilhões em 2025, mantendo ritmo constante até atingir US$65,3 bi-

lhões em 2029. A Gartner (2024) reforça esse cenário ao reportar um crescimento de 13% em 2023, totalizando US$51 bilhões no mercado global de ERP. Esses indicadores comprovam que os sistemas ERP estão sendo amplamente abordados no mercado graças a seus resultados sólidos e positivos.

Entretanto, apesar de suas vantagens e evidências de seu sucesso, é fundamental considerar os desafios e limitações que sua implementação pode acarretar (Júnior *et al.*, 2024). A adoção de um ERP pode enfrentar barreiras técnicas e culturais, como a resistência por parte de colaboradores habituados a sistemas anteriores, os quais podem demonstrar receio ou dificuldade em adaptar-se a uma nova plataforma (Júnior *et al.*, 2024).

Sistemas ERP com elevada complexidade funcional tendem a intensificar essa resistência, exigindo investimentos consideráveis em tempo e recursos financeiros para capacitação e adaptação dos usuários (Souza; Braga, 2025; Padilha; Marins, 2005; Weerasekara; Gooneratne, 2023). Essa resistência e custo alto de implantação pode resultar em um número restrito de profissionais aptos a operar corretamente o sistema (Padilha; Marins, 2005).

Visando esse cenário, este trabalho propõe o desenvolvimento de um protótipo capaz de processar consultas escritas em linguagem natural e promover a visualização automática dos dados consultados por meio de tabelas, gráficos e textos, utilizando a base de dados de um sistema ERP. Esse protótipo permite que colaboradores possam interagir com o sistema por meio de perguntas em linguagem humana, obtendo respostas gráficas com base nos dados armazenados, buscando eliminar barreiras técnicas, tornando o ERP mais acessível a diferentes profissionais e promovendo maior autonomia.

Para viabilizar essa solução, aplica-se a metodologia de Geração Aumentada de Recuperação (RAG, do inglês, *Retrieval Augmented Generation*) em conjunto com Modelos de Linguagem de Grande Escala (LLMs, do inglês, *Large Language Models*) (Lewis *et al.*, 2020). Seu uso em conjunto ocorre devido à forma como ambas as abordagens se complementam, ao permitir que o sistema tenha o poder de processamento de linguagem natural proveniente do LLM, e ao mesmo tempo consiga buscar informações de uma base externa, atualizada e específica do contexto consultado, graças à RAG (Lewis *et al.*, 2020; Salemi; Zamani, 2024; Soares, 2024).

A recuperação de dados é apoiada por um banco de dados em grafos, que facilita a modelagem das relações entre as entidades do ERP, permitindo consultas mais semânticas (Angles; Gutierrez, 2008; Oliveira; Stamboni; Júnior, 2018). O uso dessas tecnologias em conjunto permite transformar perguntas abertas escritas em linguagem natural, em consultas estruturadas à base de dados em grafo (Viswanathan; Sasaki, 2025).

1.2 Justificativa do projeto

O presente projeto se justifica na necessidade da criação de um protótipo como o proposto que, utilizando do conjunto de metodologias e tecnologias apresentado, consiga oferecer aos usuários uma *interface* de consulta em linguagem natural capaz de gerar respostas visuais automáticas, por meio de gráficos, tabelas e textos (Weerasekara; Gooneratne, 2023; Marinas, 2025). Com essa abordagem, o protótipo não apenas apresenta visualizações dos dados, mas também torna o processo de busca desses dados mais natural (Marinas, 2025).

Sob a perspectiva prática, o projeto oferece uma solução para demandas reais ao facilitar a consulta e visualização de informações corporativas, tornando este processo mais intuitivo e acessível para um público amplo dentro das organizações (Júnior *et al.*, 2024; Souza; Braga, 2025; Weerasekara; Gooneratne, 2023).

E sob a perspectiva acadêmica, o projeto representa uma contribuição ao explorar uma abordagem ainda pouco integrada no contexto de sistemas ERP, como o uso combinado de RAG e banco de dados em grafos (Lewis *et al.*, 2020; Oliveira; Stamboni; Júnior, 2018). Essa abordagem fornece subsídios para pesquisas futuras, seja no mesmo contexto deste projeto ou na aplicação da abordagem utilizada em outros domínios.

1.3 Objetivos

Diante da contextualização da área em que o projeto está inserido, da apresentação da proposta e sua justificativa, a seguir estão expostos o objetivo geral do projeto, bem como os objetivos específicos que o orientam.

1.3.1 Objetivo geral

O objetivo geral do projeto consiste no desenvolvimento de um protótipo de *software* que, utilizando a abordagem RAG combinada a um repositório de dados em grafos, interprete consultas escritas em linguagem natural e apresente ao usuário as informações retornadas em formato textual, de gráfico ou de tabela.

1.3.2 Objetivos específicos

Para alcançar esse propósito, são definidos os seguintes objetivos específicos:

a) escolher e utilizar uma ferramenta que implemente o uso de repositório de dados em grafos, com foco em sua capacidade de lidar com relações

complexas e semânticas;

b) implementar as funcionalidades responsáveis pelo processamento das consultas em linguagem natural, geração das consultas estruturadas e execução destas no banco de dados, utilizando um LLM integrado à arquitetura RAG;
c) implementar uma *interface* que permita ao usuário realizar consultas e analisar os dados retornados;
d) implementar funcionalidades responsáveis pela visualização dos dados retornados por meio das consultas ao repositório, transformando-os em gráficos, tabelas ou texto de acordo com as respostas geradas;
e) realizar testes funcionais do protótipo em um ambiente simulado de ERP, verificando o funcionamento das funcionalidades implementadas e a adequação das respostas geradas.

1.4 Estrutura do trabalho

Este documento está estruturado em seis capítulos, incluindo o presente. O Capítulo 2 descreve os fundamentos teóricos necessários para a compreensão da proposta, incluindo os principais conceitos envolvidos, bem como as abordagens utilizadas. O Capítulo 3 apresenta a revisão bibliográfica realizada, abordando os trabalhos relacionados que fundamentam teoricamente o projeto. O Capítulo 4 expõe a metodologia aplicada ao projeto, abrangendo a classificação da pesquisa, o ambiente de desenvolvimento onde o projeto foi realizado, os materiais e tecnologias empregados, a metodologia de desenvolvimento e a arquitetura do sistema. No Capítulo 5, são detalhadas as etapas de desenvolvimento do protótipo, desde sua modelagem e planejamento, até a sua implementação e realização dos testes. Por fim, o Capítulo 6 apresenta a conclusão do trabalho, reunindo os objetivos alcançados, as principais contribuições, as limitações e dificuldades encontradas durante o desenvolvimento, além de discutir possíveis direções para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo reúne os conceitos fundamentais que sustentam o desenvolvimento do projeto. São abordadas as técnicas e bases teóricas que norteiam a proposta, explicando seus papéis na construção do projeto.

2.1 *Natural Language Processing* (NLP)

Representando a grande área das técnicas aplicadas neste trabalho, o Processamento de Linguagem Natural (NLP, do inglês, *Natural Language Processing*) se trata de uma área da inteligência artificial voltada para a interação entre computadores e humanos por meio da linguagem natural humana (Caseli; Nunes, 2024; Manning; Schütze, 1999). Ele permite que sistemas computacionais compreendam, interpretem, manipulem e gerem linguagem natural, viabilizando a criação de mecanismos de busca, assistentes virtuais, tradutores automáticos, entre outras soluções (Caseli; Nunes, 2024).

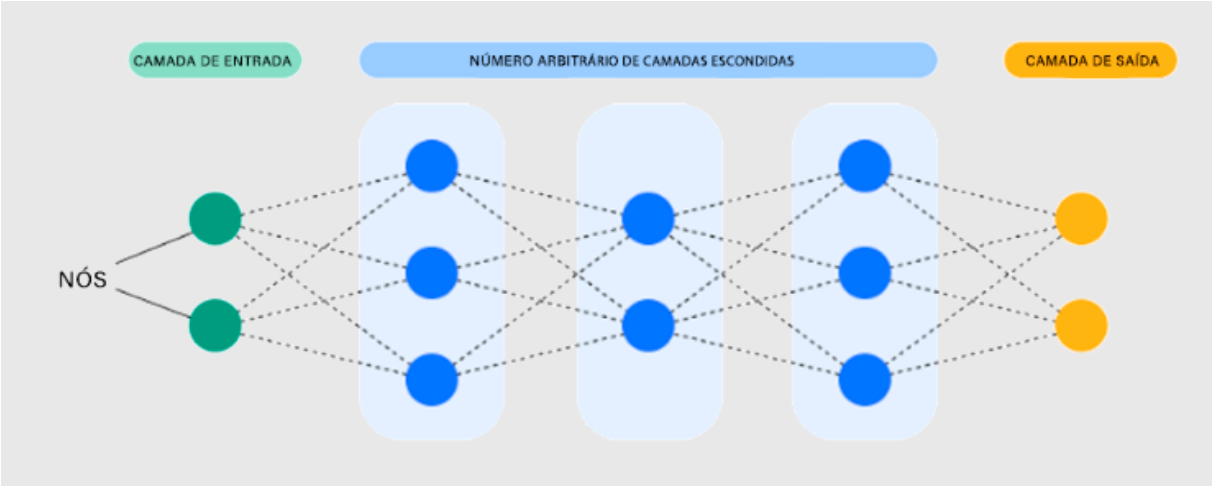
2.2 *Large Language Models* (LLM)

Os LLMs constituem uma vertente do campo de NLP. Esses modelos, além de interpretarem linguagem natural, são capazes de gerar respostas de forma semelhante à humana, possibilitando interações mais fluidas com o usuário (Soares, 2024; Hao *et al.*, 2024). Modelos como GPT e LLaMA destacam-se por impulsionar avanços significativos na área, oferecendo respostas cada vez mais próximas da linguagem humana (OpenAI, 2023; Meta AI, 2023; Ji *et al.*, 2022; Soares, 2024; Hao *et al.*, 2024).

Os LLMs se tratam de redes neurais profundas treinadas com grandes volumes de dados para realizar variadas tarefas empregando o uso da linguagem natural, como geração, tradução, sumarização e resposta a perguntas (Caseli; Nunes, 2024; Soares, 2024; Hao *et al.*, 2024). Eles funcionam por meio da previsão de palavras com base em contextos anteriores, aprendendo padrões linguísticos complexos a partir de bilhões de parâmetros (Caseli; Nunes, 2024).

Para ilustrar este conceito, a Figura 3 demonstra a representação conceitual de uma rede neural profunda, com múltiplas camadas ocultas conectadas entre si. Cada camada é composta por nós interconectados responsáveis por processar os dados de entrada e aprender com estes, gerando assim uma saída final. Essa estrutura é a base de modelos de linguagem natural, como os LLMs.

Figura 3 – Exemplo conceitual de uma rede neural



Fonte: Adaptado de (Cloudflare, 2023).

2.3 Engenharia de *prompt*

Com a popularização do uso dos LLMs, a engenharia de *prompt* tem ganhado destaque, sendo entendida como o processo de projetar e refinar instruções textuais fornecidas aos modelos para obter respostas mais precisas, contextualizadas e alinhadas aos objetivos desejados (White *et al.*, 2023; Liu *et al.*, 2023). Técnicas como fornecimento de exemplos e decomposição de tarefas complexas em etapas menores têm se mostrado eficazes para melhorar o desempenho dos modelos em domínios específicos (Brown *et al.*, 2020; Wei *et al.*, 2022).

No contexto deste projeto, *prompts* bem elaborados podem guiar o modelo na geração de consultas estruturadas — como *queries* em linguagens formais — e na interpretação de dados retornados de bancos de dados, produzindo respostas coerentes ao usuário final (Liu *et al.*, 2023). O uso da engenharia de *prompt* permite que sistemas baseados em LLMs sejam adaptados a contextos empresariais e técnicos sem a necessidade de retreinamento do modelo base (White *et al.*, 2023).

2.4 *Retrieval Augmented Generation* (RAG)

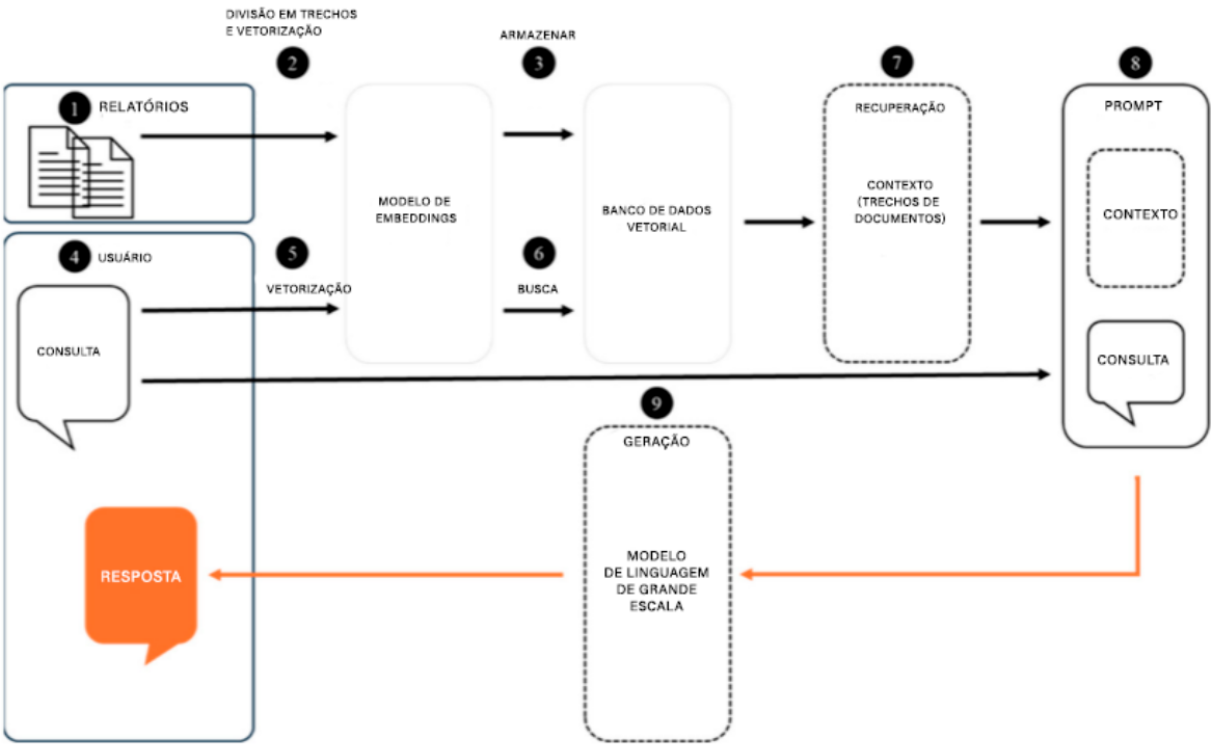
A arquitetura RAG combina técnicas de recuperação de informações com modelos de geração de texto, como os LLMs (Lewis *et al.*, 2020; Iaroshev *et al.*, 2024; Soares, 2024; Salemi; Zamani, 2024). Sua concepção ocorre com a ideia de fornecer uma complementação aos LLMs, corrigindo seus problemas e vulnerabilidades (Lewis *et al.*, 2020; Salemi; Zamani, 2024).

Seu funcionamento ocorre em duas etapas: primeiro, o sistema recupera

trechos relevantes de uma base de dados externa à que treinou o LLM de acordo com o que o usuário consultou (como documentos, manuais, registros corporativos ou um banco de dados específico); em seguida, um modelo de linguagem utiliza as informações retornadas como contexto para gerar uma resposta mais precisa e fundamentada (Lewis *et al.*, 2020; Xu; Lu *et al.*, 2024).

Essa abordagem mitiga uma das principais limitações dos LLMs tradicionais, que só respondem com base no conhecimento aprendido durante seu treinamento, e possibilita respostas atualizadas e contextualizadas a partir de dados externos (Soares, 2024; Iaroshev *et al.*, 2024). A Figura 4, referente ao esquema elaborado por Iaroshev *et al.* (2024), ilustra o funcionamento da RAG, desde a estruturação do banco de dados vetorizado até a entrega dos dados da consulta ao LLM para retornar a resposta ao usuário.

Figura 4 – Funcionamento completo da estrutura RAG



Fonte: Adaptado de (Iaroshev *et al.*, 2024).

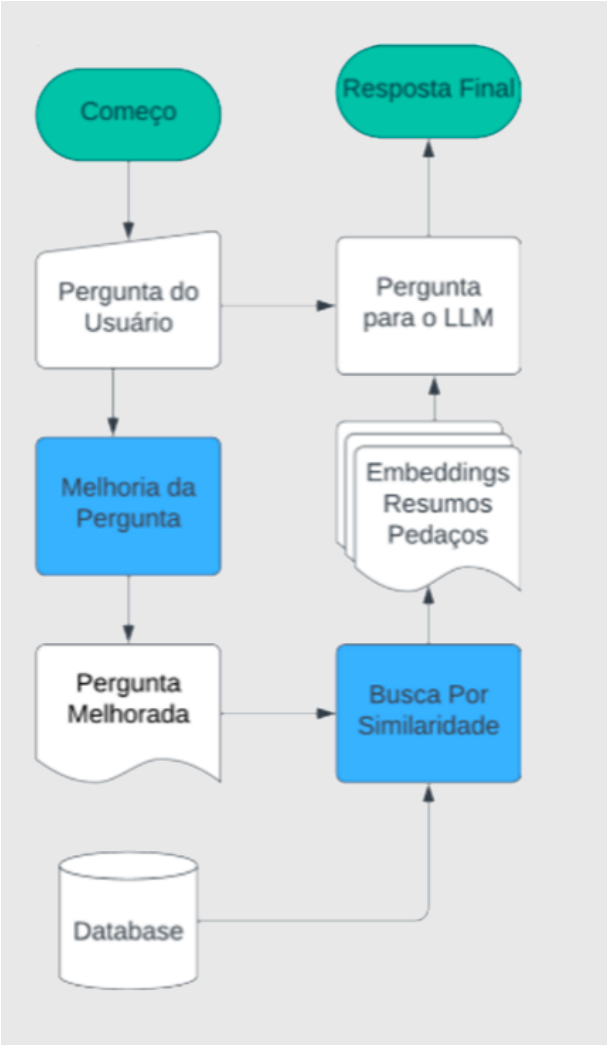
Inicialmente, quando a RAG está sendo estruturada, os documentos que irão compor a base de dados são divididos em trechos menores e transformados em vetores semânticos por meio de um modelo de *embeddings* (vetores que representam dados, como palavras ou imagens de forma densa e semântica em um espaço numérico), sendo então armazenados em um banco de dados vetorial.

Uma vez em funcionamento, a RAG permite que os usuários realizem con-

sultas em linguagem natural. Essas consultas são vetorizadas e comparadas com os dados armazenados, possibilitando a recuperação dos trechos mais relevantes por meio de busca por similaridade. Os trechos recuperados são então combinados com a consulta original na forma de um *prompt* estruturado, que é enviado a um LLM. Com base nesse contexto, o modelo gera uma resposta ao usuário, completando o ciclo.

A Figura 5 ilustra o funcionamento da RAG durante o processamento de uma consulta, apresentando de forma visual a etapa descrita anteriormente. No contexto de sistemas ERP, essa abordagem permite que usuários consultem informações complexas — como relatórios financeiros, históricos de compras ou metas operacionais — utilizando linguagem natural e com base em dados reais armazenados nos sistemas da empresa (Marinas, 2025).

Figura 5 – Funcionamento da RAG em um fluxo de consulta.



Fonte: Adaptado de (Soares, 2024).

2.5 *Embeddings* e busca vetorial

Embeddings são representações vetoriais densas de dados textuais, onde palavras, frases ou documentos inteiros são mapeados para vetores numéricos em um espaço multidimensional (Mikolov *et al.*, 2013; Pennington; Socher; Manning, 2014). Esses vetores capturam relações semânticas entre termos, de modo que palavras com significados similares tendem a ocupar posições próximas no espaço vetorial (Reimers; Gurevych, 2019).

A busca vetorial utiliza essas representações para recuperar informações com base em similaridade semântica, em vez de correspondência exata de palavras-chave (Johnson; Douze; Jégou, 2019; Guo *et al.*, 2020). Nesse processo, uma consulta em linguagem natural é convertida em um vetor de *embedding* e comparada com vetores armazenados em um banco de dados vetorial (Malkov; Yashunin, 2020). A proximidade entre vetores, frequentemente medida por métricas como distância euclidiana ou similaridade de cosseno, determina quais documentos ou trechos são mais relevantes para a consulta (Guo *et al.*, 2020; Reimers; Gurevych, 2019).

Essa abordagem é fundamental em sistemas RAG, onde a recuperação de contexto relevante antes da geração de respostas melhora significativamente a precisão e a fundamentação das saídas produzidas pelos LLMs (Lewis *et al.*, 2020; Izacard; Grave, 2021).

2.6 *Framework* LangChain

No contexto da implementação de sistemas RAG, o *framework* Langchain desempenha papel central ao orquestrar a integração entre recuperação de contexto e geração de respostas (LangChain, 2024; Topsakal; Akinci, 2023).

Langchain é um *framework* de código aberto projetado para facilitar o desenvolvimento de aplicações baseadas em LLMs (LangChain, 2024; Topsakal; Akinci, 2023). Ele fornece componentes modulares que permitem a integração dos modelos com fontes de dados externas, ferramentas de busca, bancos de dados e APIs, simplificando a construção de sistemas complexos como agentes conversacionais, *pipelines* RAG e ferramentas de automação (Chase, Harrison, 2022; Topsakal; Akinci, 2023; LangChain, 2024).

O *framework* organiza o fluxo de processamento em cadeias, onde cada etapa pode envolver chamadas a LLMs, recuperação de informações ou execução de código customizado (LangChain, 2024; Chase, Harrison, 2022). Ele oferece suporte nativo a bancos de dados vetoriais, modelos de *embedding*, e *parsing* de saídas estruturadas, permitindo que desenvolvedores construam aplicações que combinam consultas semânticas com geração de texto fundamentada em dados consultados

(LangChain, 2024).

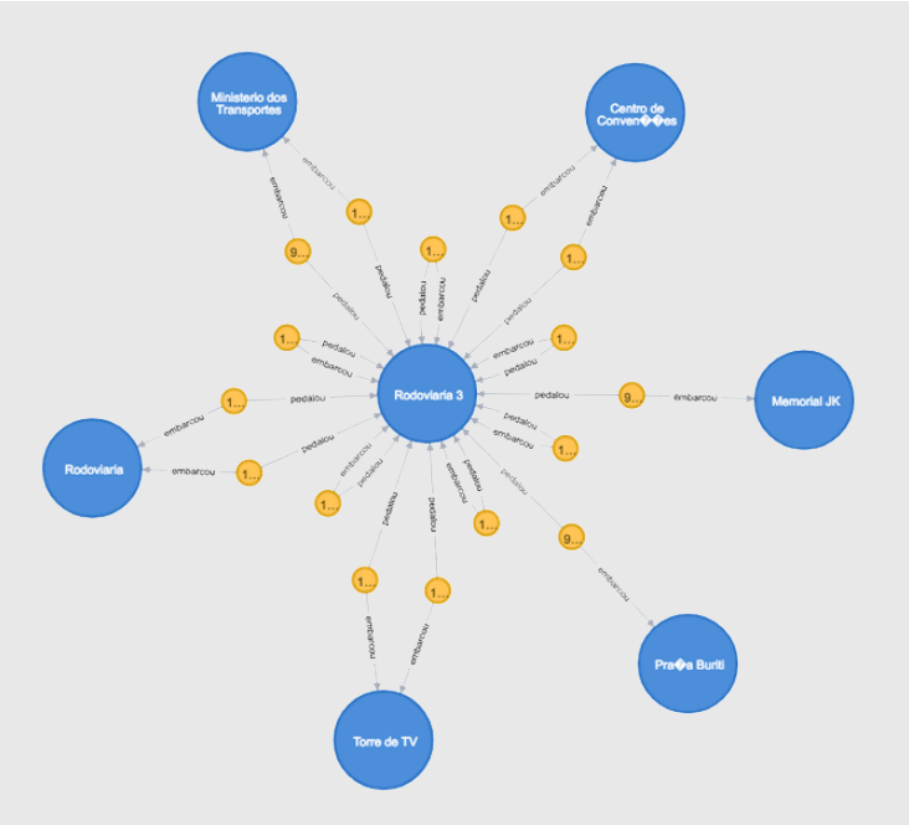
2.7 Banco de dados em grafos

Bancos de dados em grafos são estruturas de armazenamento de dados baseadas em grafos, compostos por nós (entidades) e arestas (relacionamentos) (Oliveira; Stamboni; Júnior, 2018; Angles; Gutierrez, 2008). Diferentemente dos bancos relacionais, que utilizam tabelas, os bancos em grafos são otimizados para modelar e consultar dados altamente conectados, como redes sociais, cadeias de suprimentos, ou estruturas organizacionais (Angles; Gutierrez, 2008).

Sua principal vantagem está na eficiência em consultar relações complexas e multi-nível, sem a necessidade de junções custosas (Angles; Gutierrez, 2008). Tecnologias como Neo4j têm se popularizado especialmente em contextos onde a representação semântica das conexões entre dados é fundamental (Neo4j, Inc., 2024; Oliveira; Stamboni; Júnior, 2018).

A Figura 6 representa um banco de dados em grafos desenvolvido na ferramenta Neo4j, pelo Serviço Federal de Processamento de Dados (Serpro) (2020), que utilizou os dados de viagens do Sistema de Bicicletas Compartilhadas do Distrito Federal.

Figura 6 – Exemplo de banco de dados em grafo



Fonte: Adaptado de (Serviço Federal de Processamento de Dados (Serpro), 2020).

2.8 Cypher *query language*

Cypher é uma linguagem declarativa de consulta projetada especificamente para bancos de dados em grafos, desenvolvida pela Neo4j (Neo4j, Inc., 2024; Francis *et al.*, 2018). Utiliza sintaxe intuitiva baseada em padrões visuais que representam nós e relacionamentos, facilitando a expressão de consultas complexas sobre estruturas altamente conectadas (Holzschuher; Peinl, 2013; Francis *et al.*, 2018). A linguagem permite operações de criação, leitura, atualização e exclusão de dados (CRUD), além de suportar agregações, ordenação e filtragem avançada por meio de cláusulas como `MATCH`, `WHERE`, `RETURN` e `WITH` (Neo4j, Inc., 2024; Francis *et al.*, 2018).

Diferentemente de linguagens relacionais, que exigem múltiplas junções para consultar relacionamentos, Cypher navega diretamente pelas arestas do grafo, tornando consultas sobre conexões multi-nível significativamente mais eficientes (Holzschuher; Peinl, 2013). Essa característica é relevante em contextos onde a semântica dos relacionamentos é tão importante quanto os dados dos nós, como em sistemas que possuem várias áreas interconectadas, por exemplo redes sociais, cadeias de suprimentos e ERPs (Francis *et al.*, 2018).

A Figura 7 apresenta um exemplo de consulta Cypher que recupera o nome e o preço de todos os produtos de uma categoria específica em um sistema de vendas, organizados em ordem decrescente de acordo com o seu preço.

Figura 7 – Exemplo de consulta Cypher

```
MATCH (p:Product)-[:BELONGS_TO]->(c:Category)
WHERE c.name = "Eletronicos"
RETURN p.name AS ProductName, p.price AS Price
ORDER BY p.price DESC
```

Fonte: Elaborado pelo autor, 2026.

2.9 Sistemas ERP

Sistemas ERP são plataformas integradas de gestão empresarial que centralizam e padronizam processos operacionais de diferentes áreas organizacionais, como finanças, vendas, estoque, produção e recursos humanos, em um único ambiente de dados (Padilha; Marins, 2005; Al-Amin *et al.*, 2023). Essas soluções permitem a automação de fluxos de trabalho, redução de redundâncias e maior consistência nas informações corporativas, facilitando a tomada de decisão baseada em dados confiáveis e atualizados (Souza; Braga, 2025; Weerasekara; Gooneratne, 2023).

A implementação de sistemas ERP traz benefícios significativos, como melhoria na eficiência operacional, integração de processos e maior visibilidade sobre indicadores de desempenho empresarial (Júnior *et al.*, 2024; Weerasekara; Gooneratne, 2023). No entanto, também apresenta desafios consideráveis, incluindo resistência organizacional à mudança e complexidade na parametrização do sistema para diferentes contextos de negócio (Padilha; Marins, 2005; Al-Amin *et al.*, 2023).

No contexto atual, a crescente quantidade de dados armazenados em sistemas ERP demanda mecanismos mais eficientes para consulta e análise de informações (Souza; Braga, 2025). A aplicação de técnicas de NLP e recuperação de informações representa uma oportunidade de ampliar a acessibilidade e a utilidade dessas plataformas, permitindo que usuários não técnicos obtenham respostas fundamentadas em dados corporativos por meio de consultas em linguagem natural (Júnior *et al.*, 2024; Souza; Braga, 2025).

3 REVISÃO BIBLIOGRÁFICA

A constante evolução dos sistemas ERP tem impulsionado a busca por soluções mais acessíveis e inteligentes, capazes de tornar a interação dos usuários com grandes volumes de dados mais eficiente e intuitiva (Souza; Braga, 2025; Júnior *et al.*, 2024). Dentre as inovações mais promissoras nesse contexto, destacam-se as técnicas baseadas em modelos generativos, que permitem a formulação de consultas em linguagem natural e o retorno de respostas contextuais e detalhadas (Caseli; Nunes, 2024; Soares, 2024; Hao *et al.*, 2024).

Nesse cenário, a arquitetura RAG, apesar de ser uma área de estudo relativamente nova, tem ganhado destaque por combinar a capacidade de linguagem dos LLMs com mecanismos eficientes de recuperação de informações (Lewis *et al.*, 2020; Iaroshev *et al.*, 2024). Essa abordagem tem como objetivo superar limitações comuns dos LLMs, como a geração de respostas desatualizadas ou imprecisas, ao integrar fontes externas de conhecimento durante o processo de geração (Lewis *et al.*, 2020; Salemi; Zamani, 2024).

Para aprimorar ainda mais a estrutura e a semântica das informações recuperadas, o uso de bancos de dados em grafos vem sendo incorporado em soluções baseadas em RAG (Viswanathan; Sasaki, 2025; Xu; Cruz *et al.*, 2024; Xu; Lu *et al.*, 2024). Essa tecnologia permite a modelagem de relações complexas entre dados e facilita a recuperação contextualizada e semântica de informações relevantes (Oliveira; Stamboni; Júnior, 2018).

Com base nas características e nos benefícios das arquiteturas e tecnologias mencionadas, optou-se pela combinação entre RAG e bancos de dados em grafos para o desenvolvimento deste projeto. A seguir, são apresentados os principais trabalhos da literatura que exploram essa abordagem, e que fundamentam a proposta deste estudo. Embora os projetos discutidos não repliquem exatamente a mesma combinação tecnológica no mesmo contexto, eles utilizam elementos centrais presentes nesta proposta e obtêm resultados que se alinham com os objetivos almejados.

O trabalho de Viswanathan e Sasaki (2025), é o trabalho que mais se aproxima desta proposta, integrando Grafos de Conhecimento (KGs, do inglês, *Knowledge Graphs*) com RAG em sistemas ERP. Este trabalho detalha como KGs incorporados ao *pipeline* RAG permitem enriquecer o contexto recuperado, comparando com abordagens tradicionais de apenas RAG, e mostram que essa integração produz resultados mais precisos e contextualizados no domínio ERP.

Já o trabalho de Xu, Cruz *et al.* (2024) tem por finalidade melhorar o processo de atendimento ao cliente utilizando um sistema de pergunta-resposta com base em RAG integrado a um KG. Os autores construíram um grafo a partir de *tickets* históricos de suporte técnico, modelando tanto a estrutura interna de cada chamado quanto

as relações entre chamados distintos. Durante a recuperação, o sistema utiliza LLMs para interpretar a intenção da pergunta e identificar subgrafos relevantes, os quais são convertidos em consultas estruturadas para gerar respostas precisas. Os resultados mostraram ganhos significativos em relação ao método tradicional.

O artigo de Zhang e Duigou (2022) propõe uma interface vocal de linguagem natural para o sistema ERP *open source* Odoo, com o objetivo de facilitar interações complexas no ERP por meio de comandos de voz (Odoo S.A., 2024). O projeto descrito no artigo utiliza um mecanismo de *Speech-to-Text* para converter voz em texto e mecanismos de conversão *Text-to-SQL*, que permite a tradução precisa de comandos em texto em consultas SQL. O estudo demonstrou viabilidade técnica da *interface* via linha de comando, permitindo, por exemplo, a consulta do *status* de produção por voz. A abordagem destaca-se por possibilitar acesso intuitivo ao ERP, reduzindo barreiras técnicas dos usuários.

Xu, Lu *et al.* (2024) propõe um sistema inteligente de perguntas e respostas voltado para o patrimônio cultural intangível chinês Nanjing Yunjin. O sistema é construído com a integração entre KGs e RAG. A abordagem visa superar limitações dos KGs tradicionais, como dependência de tipos de entidades pré-definidos e baixa capacidade de lidar com ambiguidades da linguagem natural. Os resultados demonstraram que a combinação entre RAG e KG melhorou significativamente o modelo, que passou a oferecer respostas mais precisas, preservando o conhecimento histórico.

O trabalho de Matsumoto *et al.* (2024) apresenta o KRAGEN, um *framework* que integra RAG com KGs para resolver problemas complexos no domínio biomédico. O KRAGEN implementa uma técnica chamada Grafo de Pensamento (GoT, do inglês, *Graph of Thoughts*), que decompõe a pergunta do usuário em subproblemas conectados em um grafo, tornando o processo de geração mais explicável e estruturado. O artigo utiliza como caso de uso um grafo sobre Alzheimer extraído de uma base Neo4j e demonstra que, ao combinar RAG com GoT, o sistema atinge acurácia significativamente superior em tarefas de múltipla escolha e verdadeiro/falso, quando comparado a modelos base.

Por fim, Chen *et al.* (2025) apresenta o desenvolvimento do sistema *Interactive Industrial Knowledge Management* (IIKM), uma solução baseada em RAG e LLMs, voltada para o gerenciamento de conhecimento industrial. O sistema tem como objetivo fornecer respostas a partir de documentos internos da empresa, como manuais técnicos e regulamentos organizacionais. Utilizando técnicas de vetorização de conteúdo, o sistema recupera os trechos mais relevantes e os fornece como entrada para o modelo GPT-3.5, que gera respostas em linguagem natural.

Observa-se que, embora existam trabalhos que utilizem, individualmente ou em pares, tecnologias como RAG, bancos de dados em grafos, visualização gráfica de dados e interpretação de linguagem natural, nenhum deles utiliza todas as aborda-

gens de forma combinada com o propósito específico de aprimorar a recuperação de informações em sistemas ERP.

Dessa forma, a proposta configura-se como uma contribuição original, ao explorar a sinergia entre essas abordagens em um contexto ainda pouco explorado na literatura. Ainda que não haja um projeto idêntico, os trabalhos analisados servem como referências teóricas e como evidência empírica de que as estratégias e tecnologias escolhidas são viáveis e eficazes para alcançar os objetivos propostos.

O quadro 1 elenca todos os trabalhos descritos acima, dividindo-os em objetivo, técnicas utilizadas e seus resultados. Por meio dele ficam mais claras as diferenças deste trabalho para com os demais.

Quadro 1 – Comparação entre trabalhos semelhantes

Trabalho (Autores)	**Objetivo**	**Técnicas**	**Resultado**
Viswanathan e Sasaki (2025)	Integrar KGs com RAG em sistemas ERP	RAG, KGs	Resultados mais precisos e contextualizados em domínios ERP
Xu, Cruz et al. (2024)	Criar um novo método de perguntas e respostas para atendimento ao cliente	RAG, KGs, LLMs, consultas estruturadas	Ganhos significativos de precisão e tempo sobre métodos tradicionais
Zhang e Duigou (2022)	Criar interface vocal em linguagem natural para o ERP Odoo	*Speech-to-Text*, *Text-to-SQL*, Odoo ERP	*Interface* por voz demonstrou viabilidade prática, facilitando acesso ao ERP
Xu, Lu et al. (2024)	Sistema de perguntas e respostas sobre Nanjing Yunjin	RAG, KGs, modelo ROBERTA, FAISS, LLM	Melhor precisão, interpretabilidade e lógica das respostas, facilitando a recuperação do conhecimento do Nanjing Yunjin
Matsumoto et al. (2024)	Desenvolver uma ferramenta (KRAGEN) que resolva problemas em domínios especializados	RAG, KGs, GoT, LLMs, Neo4j	Acurácia superior em tarefas complexas e redução significativa de erros e alucinações nas respostas geradas
Chen et al. (2025)	Desenvolver um sistema de gerenciamento de conhecimento industrial (IIKM) para facilitar consultas em ambientes industriais.	RAG, LLMs, vetorização de conteúdo, BM25, rankeador BAAI	Alta precisão e eficiência na recuperação de informações técnicas e regulatórias, comprovando sua eficácia em ambiente industrial real.
Este trabalho	Recuperação de informação em sistemas ERP integrando múltiplas tecnologias	RAG, banco de dados em grafos, LLMs, visualização gráfica de dados	Protótipo funcional que retorna respostas precisas e bem representadas graficamente

Fonte: Elaborado pelo autor, 2025.

4 METODOLOGIA

O presente capítulo descreve a metodologia adotada para o desenvolvimento do protótipo. Inicialmente, apresenta-se a classificação metodológica do projeto, com base nos critérios de natureza da pesquisa, objetivos, procedimentos metodológicos e abordagem. Em seguida, são descritos os materiais e tecnologias utilizados, bem como o ambiente de desenvolvimento no qual o trabalho foi conduzido. Na sequência, é apresentada a metodologia de desenvolvimento adotada para a condução do projeto. Posteriormente, descreve-se a arquitetura definida para a construção do sistema. Por fim, são detalhadas as etapas de desenvolvimento, desde a escolha da ferramenta de banco de dados em grafos utilizada até a finalização do protótipo.

4.1 Classificação metodológica

A pesquisa possui natureza aplicada, pois visa o desenvolvimento de uma solução para um problema específico por meio da criação do protótipo apresentado (Gil, 2002). Em termos de objetivos, trata-se de uma pesquisa exploratória, voltada à compreensão do problema e à experimentação de tecnologias viáveis para solucioná-lo (Gil, 2002; Gerhardt; Silveira, 2009). A abordagem adotada é qualitativa, baseada na análise do comportamento do sistema e na avaliação da clareza e adequação das respostas geradas pelo protótipo (Prodanov; Freitas, 2013; Gerhardt; Silveira, 2009).

Por fim, os procedimentos metodológicos adotados incluem: pesquisa bibliográfica, essencial para embasar teoricamente o projeto e aprofundar os conceitos de NLP, LLMs, RAG e bancos de dados em grafos; e pesquisa experimental, por meio da implementação do sistema e realização de testes controlados, que possibilitam verificar seu funcionamento e aplicabilidade prática (Lakatos; Marconi, 2003; Gerhardt; Silveira, 2009). Além disso, foi realizada a modelagem de um cenário simulado de ERP, utilizado como base para o protótipo desenvolvido.

4.2 Materiais e tecnologias

Esta seção apresenta o ambiente de desenvolvimento utilizado e as tecnologias empregadas na construção do protótipo.

4.2.1 Ambiente de desenvolvimento

O desenvolvimento do projeto foi realizado em um ambiente composto por um computador *desktop*, cujas especificações estão descritas no Quadro 2.

Quadro 2 – Ambiente de desenvolvimento

Componente	**Especificação**
Processador	11th Gen Intel® Core™ i5-11400 @ 2.60Ghz
Memória RAM	16GB (2x8GB), DDR4, 2666Mhz
Placa de Vídeo	NVIDIA GeForce RTX 2060 (6GB VRAM)
Sistema Operacional	Windows 10 Pro
Navegador	Google Chrome

Fonte: Elaborado pelo autor, 2026.

4.2.2 Tecnologias utilizadas

O Quadro 3 apresenta as principais tecnologias empregadas no desenvolvimento do sistema, organizadas por categoria.

Quadro 3 – Tecnologias utilizadas no projeto

Categoria	**Tecnologia**	**Versão**
Backend	Node.js	v22.21.1
	LangChain	v0.3.x
LLMs e IA	Ollama	v0.12.11
	Qwen2.5-Coder	7B
	Nomic-Embed-Text	–
Banco de Dados	Neo4j	v5.28.0
	@faker-js/faker	v9.4.0
	Cypher	–
Frontend	HTML5/CSS3/JavaScript	–
	Tailwind CSS	v3.4.17
	Chart.js	v4.4.0
	Lucide Icons	v1.7.0
Infraestrutura	Docker	v29.2.1
	Docker Compose	v5.1.0
Desenvolvimento	Visual Studio Code	v1.113.0
	Figma	v126.2.10

Fonte: Elaborado pelo autor, 2026.

O Node.js foi utilizado como ambiente de execução do servidor da aplicação, permitindo o processamento das requisições e integração entre os componentes do sistema (Node.js Foundation, 2024). A implementação do *backend* foi realizada em linguagem JavaScript, explorando o ecossistema do Node.js para desenvolvimento de aplicações assíncronas.

O LangChain foi empregado como *framework* para orquestração das funcionalidades baseadas em RAG, sendo responsável por estruturar o fluxo de processamento das consultas. No projeto, sua utilização abrange desde a verificação de consultas previamente realizadas (cache), passando pela recuperação de informações no banco de dados, até a geração da resposta final ao usuário. Foram utilizados os pacotes `@langchain/community` (v0.3.28), `@langchain/core` (v0.3.37) e `@langchain/ollama` (v0.1.5) (LangChain, 2024).

Para a execução local dos LLMs, foi utilizado o Ollama, que possibilita o carregamento e a utilização dos modelos sem dependência de serviços externos, garantindo maior controle sobre o processamento das consultas (Ollama, 2024).

O principal modelo utilizado foi o Qwen2.5-Coder, empregado na geração de *queries* Cypher, bem como na interpretação dos resultados retornados e na construção da resposta final ao usuário (Alibaba Cloud, 2024).

Para a geração de representações vetoriais das consultas, foi utilizado o modelo Nomic-Embed-Text, responsável por converter entradas em linguagem natural em *embeddings* (Nomic AI, 2024). Esses vetores foram utilizados na implementação de um mecanismo de cache baseado em similaridade semântica, permitindo reutilizar respostas para consultas semelhantes.

O sistema de gerenciamento de banco de dados adotado foi o Neo4j, utilizado para armazenamento dos dados em formato de grafo, possibilitando a modelagem de relações complexas entre entidades do sistema de ERP (Neo4j, Inc., 2024).

A linguagem Cypher foi utilizada para a realização de consultas ao banco de dados, sendo gerada dinamicamente pelo LLM a partir das entradas em linguagem natural fornecidas pelo usuário (Neo4j Inc., 2024).

A biblioteca `@faker-js/faker` foi utilizada para a população do banco, por meio da geração de dados fictícios no sistema, permitindo a criação de um conjunto de dados representativo para o ambiente simulado de ERP (Faker.js Community, 2026). Sua utilização possibilitou a inserção automatizada de informações como nomes, datas, valores e demais atributos necessários, contribuindo para os testes das funcionalidades desenvolvidas sem a necessidade de dados reais.

No desenvolvimento da *interface*, foram utilizadas as tecnologias HTML, CSS e JavaScript, responsáveis pela estrutura, estilização e comportamento da aplicação *web*.

O Tailwind CSS foi utilizado como *framework* de estilização, permitindo a

construção padronizada da *interface* por meio de classes utilitárias (Tailwind Labs, 2024).

A biblioteca Chart.js foi empregada na visualização dos dados retornados pelo sistema, possibilitando a apresentação das informações em formato de gráficos (Chart.js, 2024).

A biblioteca Lucide Icons foi utilizada para a composição visual da *interface*, fornecendo ícones em formato SVG (Lucide, 2024).

O Docker foi utilizado para a execução do banco de dados Neo4j em um ambiente isolado, evitando a necessidade de instalação direta no sistema operacional e garantindo maior portabilidade e reprodutibilidade do ambiente de desenvolvimento (Docker Inc., 2024b).

O Docker Compose foi utilizado para a definição e gerenciamento do contêiner do banco de dados, facilitando sua inicialização e configuração durante a execução do sistema (Docker Inc., 2024a).

O desenvolvimento do código foi realizado utilizando o Visual Studio Code, que oferece suporte às tecnologias empregadas no projeto (Microsoft, 2024).

Por fim, o Figma foi utilizado para a prototipação da *interface* do usuário, auxiliando na definição do *layout* e na experiência de uso da aplicação (Figma Inc., 2024).

4.3 Metodologia de desenvolvimento

O desenvolvimento do projeto adotou princípios da metodologia Scrum, porém de forma adaptada às especificidades de um trabalho acadêmico individual (Schwaber; Sutherland, 2020). Foram mantidos elementos centrais da metodologia, como o levantamento de requisitos, a criação de estórias de usuário, a organização do trabalho em *sprints* e a priorização de funcionalidades por meio de um *product backlog* (Schwaber; Sutherland, 2020). No entanto, práticas voltadas ao trabalho em equipe, como reuniões diárias, revisões de *sprint* e retrospectivas, não foram aplicadas, uma vez que o desenvolvimento foi conduzido individualmente (Schwaber; Sutherland, 2020).

Da mesma forma, papéis específicos como *Product Owner* e *Scrum Master* não foram formalmente designados, sendo todas as responsabilidades de planejamento, desenvolvimento e validação concentradas no autor do trabalho (Schwaber; Sutherland, 2020). O acompanhamento do progresso ocorreu por meio da definição de objetivos claros para cada *sprint* e da entrega incremental de funcionalidades ao final de cada ciclo, alinhando-se aos princípios de desenvolvimento iterativo e incremental característicos das metodologias ágeis (Beck *et al.*, 2001).

Essa abordagem híbrida permitiu manter a flexibilidade e a organização

necessárias ao projeto, sem a necessidade de seguir rigidamente todos os rituais formais do Scrum tradicional, adequando-se às condições de um projeto desenvolvido de forma autônoma (Campanelli; Parreiras, 2015). Toda a documentação relacionada ao planejamento do desenvolvimento do projeto é apresentada na Seção 5.2, onde são descritas as etapas e os elementos considerados na adaptação da metodologia Scrum para este trabalho.

4.4 Arquitetura do sistema

O sistema desenvolvido adota uma arquitetura orientada a serviços (SOA, do inglês *Service Oriented Architecture*), estruturada de forma modular para facilitar a manutenção, escalabilidade e integração entre os diferentes componentes que compõem o fluxo de processamento de consultas em linguagem natural (Erl, 2005; Papazoglou *et al.*, 2007). A seguir, são apresentados os fundamentos dessa abordagem arquitetural, sua justificativa no contexto do projeto e a apresentação dos componentes e do fluxo de processamento implementado.

4.4.1 Arquitetura orientada a serviços

A arquitetura SOA é um paradigma de *design* de *software* que organiza funcionalidades em serviços independentes, fracamente acoplados e reutilizáveis (Erl, 2005; Papazoglou *et al.*, 2007). Cada serviço encapsula uma funcionalidade específica e pode ser desenvolvido, testado e mantido de forma autônoma, promovendo modularidade e facilitando a evolução incremental do sistema (Josuttis, 2007).

Uma das principais características da arquitetura SOA é a separação de responsabilidades, onde cada serviço possui uma função delimitada e expõe suas funcionalidades por meio de métodos ou APIs bem definidas, sem expor detalhes internos de implementação (Erl, 2005). Essa abordagem favorece a reutilização de componentes, reduz o acoplamento entre módulos e facilita a substituição ou atualização de serviços individuais sem impactar o sistema como um todo (Papazoglou *et al.*, 2007).

4.4.2 Justificativa da escolha

A escolha pela arquitetura SOA justifica-se pelas características específicas do sistema desenvolvido, que integra múltiplos componentes tecnológicos distintos, como LLMs, banco de dados em grafos, sistema de cache vetorial, sistema de processamento de consultas e *interface* de usuário. A modularidade proporcionada pela SOA permitiu organizar o sistema em serviços especializados, cada um responsável

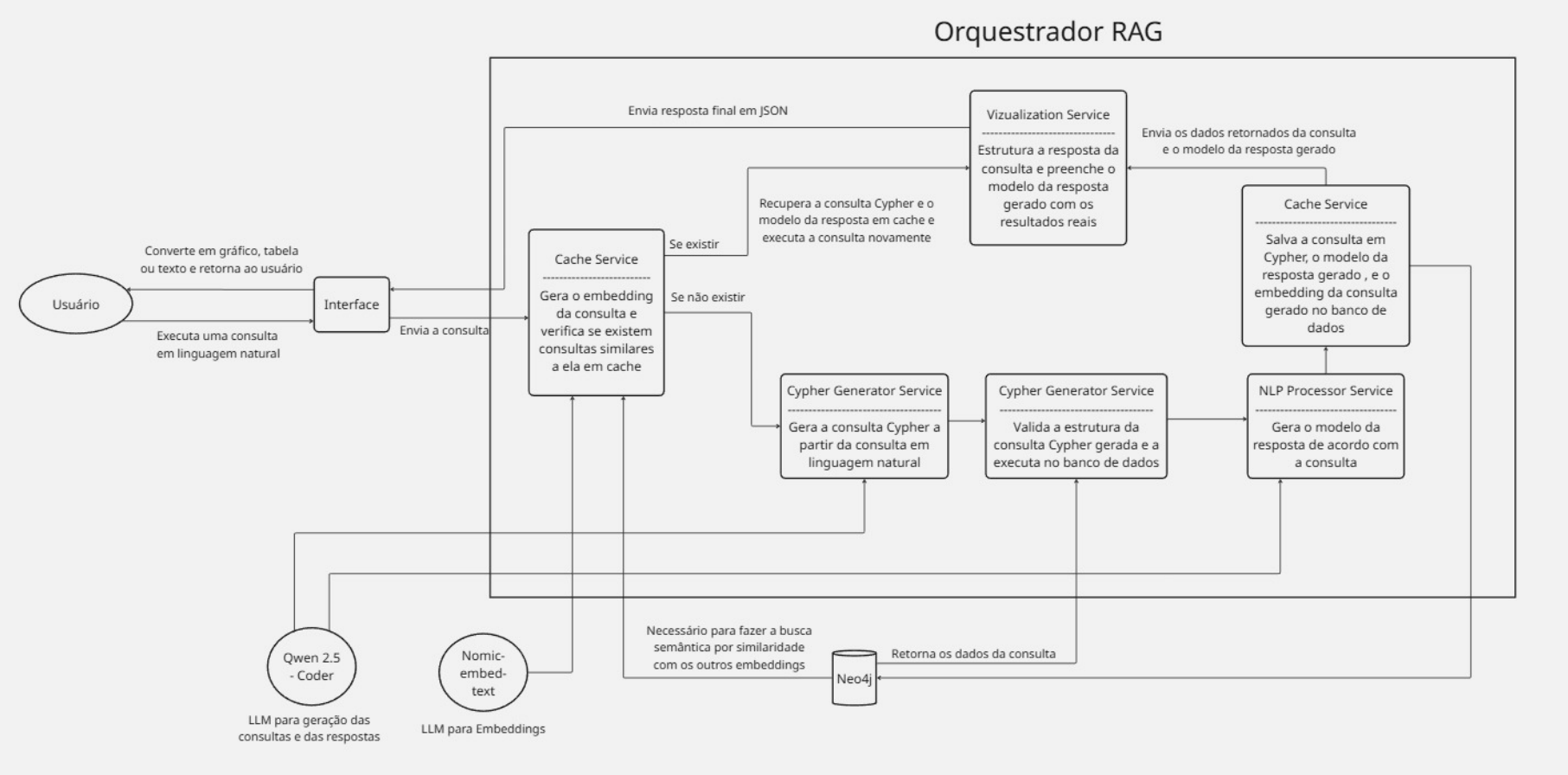
por uma etapa do processamento. (Erl, 2005; Josuttis, 2007).

Além disso, a arquitetura SOA facilita a manutenção e evolução do sistema, uma vez que alterações em serviços - como a substituição do LLM ou a otimização de *prompts* — podem ser realizadas de forma isolada, sem impactar outros componentes (Papazoglou *et al.*, 2007; Josuttis, 2007). Essa característica foi importante no desenvolvimento do protótipo, permitindo ajustes incrementais nos *prompts* e nos serviços de processamento sem afetar os demais componentes.

4.4.3 Visão geral da arquitetura

A Figura 8 apresenta a arquitetura completa do sistema, organizada em camadas e serviços que compõem o fluxo de processamento das consultas em linguagem natural.

Figura 8 – Arquitetura SOA do sistema RAG com Neo4j



Fonte: Elaborado pelo autor, 2026.

A arquitetura é composta por quatro camadas principais: a camada de *interface*, responsável pela interação visual; o orquestrador RAG, que concentra os serviços de processamento; a camada de Inteligência Artificial, que consiste dos LLMs utilizados pelo sistema; e a camada de dados, que armazena as informações do sistema ERP e o cache de consultas.

4.4.4 Descrição dos componentes

A camada de *interface* é composta pela aplicação *web* desenvolvida em HTML, CSS e JavaScript, que oferece ao usuário uma *interface* de *chat* para interação em linguagem natural. Essa camada é responsável por capturar as consultas do usuário, enviá-las ao *backend* por meio de requisições HTTP, e renderizar as respostas retornadas no formato adequado, seja texto, tabela ou gráfico.

O orquestrador RAG concentra os principais serviços de processamento do sistema, organizados de forma modular conforme os princípios da SOA. Ele é o responsável por definir o fluxo de execução da requisição, coordenando a comunicação entre os serviços.

O primeiro serviço acionado é o *Cache Service*, responsável por gerar o *embedding* vetorial da consulta utilizando o modelo Nomic-Embed-Text e verificar no banco de dados Neo4j se existe uma consulta similar previamente processada. Caso a consulta seja encontrada em cache, o sistema recupera a *query* Cypher e o modelo de resposta armazenados, executa a *query* novamente, define a resposta final utilizando o modelo salvo e a *query* executada e retorna o resultado ao usuário. Esse processo oferece duas otimizações ao sistema, promovendo economia de tempo e de custos de processamento dos LLMs.

Quando a consulta não é encontrada em cache, o sistema aciona o *Cypher Generator Service*, que utiliza o modelo Qwen2.5-Coder para gerar uma *query* Cypher a partir da consulta em linguagem natural. Em seguida, a *query* gerada é validada e executada no banco de dados Neo4j, retornando os dados resultantes.

Os resultados da consulta são então processados pelo *NLP Processor Service*, que utiliza novamente o modelo Qwen2.5-Coder para analisar os dados retornados e gerar um modelo de resposta estruturado, determinando automaticamente a melhor forma de apresentação (texto, tabela ou gráfico). Por fim, o *Visualization Service* estrutura a resposta final em formato JSON, preparando os dados para renderização na *interface*. Todos esses serviços operam de forma independente, sendo invocados pelo orquestrador RAG conforme a necessidade, de acordo com o fluxo de processamento necessário.

A camada de inteligência artificial é composta pelos modelos de linguagem que sustentam o funcionamento do sistema. O modelo Qwen2.5-Coder, executado

localmente via Ollama, é utilizado tanto para a geração de *queries* Cypher quanto para a análise e formatação dos resultados retornados pelo banco de dados. Já o modelo Nomic-Embed-Text é responsável pela geração de *embeddings* das consultas, permitindo a implementação do sistema de cache baseado em similaridade semântica.

A camada de dados é composta pelo banco de dados em grafos Neo4j, que armazena tanto os dados simulados do sistema ERP quanto o cache vetorial de consultas previamente processadas. Essa estrutura permite consultas sobre relações complexas entre entidades e possibilita a recuperação de consultas similares por meio de busca vetorial.

O fluxo de execução, bem como os componentes apresentados nesta seção são detalhados na Seção 5.6, que descreve a implementação do protótipo, contemplando as diferentes camadas do sistema e suas funcionalidades.

4.5 Etapas do desenvolvimento

O desenvolvimento deste trabalho foi estruturado em etapas sequenciais e interdependentes, definidas com base nos objetivos específicos propostos. Essa organização permitiu conduzir o projeto de forma sistemática, desde a fundamentação teórica até a conclusão da implementação do protótipo.

Inicialmente, foi realizada a etapa de seleção da ferramenta de banco de dados em grafos, considerando critérios como capacidade de representação de relações complexas, suporte a consultas eficientes, integração com o restante da arquitetura proposta e facilidade de uso e aprendizado.

Antes do início da modelagem e da estruturação do sistema, foram definidas as características necessárias do protótipo, bem como organizadas as etapas para seu desenvolvimento. Essa etapa contemplou o levantamento dos requisitos, a criação das estórias de usuário, a definição do *product backlog* e o planejamento das *sprints* de desenvolvimento.

Iniciando a estruturação do projeto, realizou-se a modelagem de um cenário simulado de um sistema ERP e a estruturação do banco de dados em grafos, definindo entidades, relacionamentos e propriedades necessárias para representar o domínio do problema.

Na sequência, foi realizada a modelagem arquitetural do sistema. Nessa etapa, foram elaborados os principais artefatos de modelagem, como diagrama de componentes, diagramas de casos de uso e diagrama de sequência, com o objetivo de representar o comportamento e a estrutura do sistema.

Posteriormente, foi realizada a prototipação da *interface* do sistema, com foco na definição de uma interação simples e objetiva para a realização de consultas em linguagem natural e visualização dos resultados.

A etapa seguinte consistiu da implementação do protótipo, contemplando o desenvolvimento das funcionalidades responsáveis pelo processamento das consultas em linguagem natural, integração com o banco de dados em grafos, aplicação da abordagem RAG, geração das respostas ao usuário e também a implementação da *interface* já prototipada.

Por fim, foram realizados testes funcionais em um ambiente simulado de ERP, com o objetivo de validar o funcionamento das funcionalidades implementadas, bem como a adequação das respostas geradas pelo sistema às consultas realizadas.

Todas as etapas apresentadas anteriormente são detalhadas no Capítulo 5, no qual é apresentado todo o processo de desenvolvimendo de cada uma até a conclusão do projeto.

5 DESENVOLVIMENTO

Este capítulo apresenta o desenvolvimento do protótipo, descrevendo de forma detalhada as etapas realizadas desde o planejamento inicial até a implementação do sistema. São abordados todos processos envolvidos, incluindo a seleção da ferramenta de banco de dados em grafos, a definição dos requisitos e o planejamento das *sprints*, a modelagem do cenário experimental de ERP, a modelagem estrutural e comportamental do sistema, a prototipação da *interface* de comunicação com o usuário, a implementação das funcionalidades e a realização dos testes. O objetivo é evidenciar como os conceitos e metodologias previamente discutidos foram aplicados na construção da solução proposta.

5.1 Seleção de ferramenta de banco de dados em grafos

Para atender à necessidade de representar e consultar relações complexas e semânticas entre entidades no sistema, foram realizadas etapas de análise e seleção de uma ferramenta capaz de implementar um repositório de dados em grafos. Essa decisão foi orientada principalmente pela aderência da ferramenta às necessidades do projeto, especialmente no que se refere à integração com sistemas baseados em linguagem natural e na abordagem RAG.

Foi realizada uma comparação entre diferentes ferramentas de bancos de dados em grafos disponíveis no mercado, como Neo4j, Amazon Neptune, ArangoDB e OrientDB, avaliando aspectos como modelo de dados, suporte a consultas semânticas, integração com LLMs e facilidade de uso. O Quadro 4 apresenta a comparação entre as ferramentas analisadas de acordo com os critérios estabelecidos, com base em suas documentações oficiais (Neo4j, Inc., 2024; Neo4j, 2025a,b; Amazon Web Services, 2025; ArangoDB, 2025b,a; OrientDB, 2025).

Quadro 4 – Comparação entre ferramentas de bancos de dados em grafos

Critério	**Neo4j**	**Amazon Neptune**	**ArangoDB**	**OrientDB**
Modelo de Dados	Grafo propriedade (*Property Graph*)	Grafo RDF e *Property Graph*	Multi-modelo (grafo, documento, chave-valor)	Multi-modelo (grafo e documento)
Suporte a consultas semânticas	Cypher, suporte a consultas complexas	SPARQL (para RDF), Gremlin (para *Property Graph*)	AQL (ArangoDB *Query Language*), suporta consultas complexas	SQL-like e Gremlin, consultas moderadamente semânticas
Integração com LLMs	Alta: ampla documentação e bibliotecas para integração, APIs REST e *drivers* oficiais	Moderada: integração via AWS e APIs, menos foco em LLMs nativos	Moderada: APIs REST e suporte a *embedding* externos	Baixa a moderada, menos foco em integrações modernas
Facilidade de uso	Muito alta: *interface* gráfica amigável, ampla comunidade, documentação rica	Alta: boa documentação, mas depende da AWS, curva moderada	Moderada: *interface* e documentação razoáveis, curva de aprendizado maior	Moderada a baixa: documentação limitada, menor comunidade ativa

Fonte: Adaptado de (Neo4j, 2025a,b; Amazon Web Services, 2025; ArangoDB, 2025b,a; OrientDB, 2025).

Como pode ser observado no Quadro 4, o Neo4j se destaca por oferecer um modelo de grafo nativo consolidado, uma linguagem de *queries* robusta (Cypher), documentação abrangente e integrações já estabelecidas com *pipelines* de RAG e LLMs. Além disso, sua menor curva de aprendizado e a ampla comunidade de desenvolvedores tornam a ferramenta adequada para projetos acadêmicos e prototipagem (Neo4j, 2025a,b).

Em contrapartida, o Amazon Neptune suporta múltiplos modelos de grafo e oferece integração com ferramentas que permitem o uso de LLMs, porém sua de-

pendência do ecossistema AWS pode limitar a flexibilidade do projeto. O ArangoDB apresenta um modelo versátil com suporte a dados vetorizados, mas sua integração com LLMs ainda é menos consolidada, exigindo maior esforço de configuração. Já o OrientDB combina recursos de grafos e documentos, porém possui menor foco em aplicações de inteligência artificial, comunidade mais restrita e integração limitada com LLMs (Amazon Web Services, 2025; ArangoDB, 2025a; OrientDB, 2025).

Diante desses fatores, o Neo4j foi escolhido como a opção mais adequada para compor o repositório de dados deste projeto, por apresentar maior aderência aos requisitos definidos e melhor alinhamento com os objetivos propostos.

5.2 Planejamento do desenvolvimento

Após a escolha da ferramenta de banco de dados em grafos, o próximo passo consistiu no planejamento inicial do projeto, contemplando o levantamento dos requisitos funcionais e não funcionais, a criação das estórias de usuário, a elaboração de um *product backlog* organizado por prioridade e por fim a organização das atividades em *sprints*.

5.2.1 Levantamento dos requisitos

A primeira etapa do planejamento do projeto consistiu no levantamento de seus requisitos. Os Quadros 5, 6 e 7 a seguir apresentam os requisitos funcionais e não funcionais levantados, classificados por categoria, acompanhados de suas respectivas descrições.

Quadro 5 – Requisitos funcionais do sistema

Código	**Nome**	**Categoria**	**Descrição**
RF01	Consulta em linguagem natural	Funcionalidade Principal	O sistema deve permitir que o usuário faça perguntas em linguagem natural (português) sobre os dados armazenados no banco de dados.
RF02	Geração automática de *queries* Cypher	Processamento	O sistema deve converter automaticamente perguntas em linguagem natural para *queries* Cypher válidas.
RF03	Execução de *queries* no Neo4j	Processamento	O sistema deve executar *queries* Cypher no banco de dados Neo4j e retornar seus resultados.
RF04	Geração automática de resposta	Processamento	O sistema deve converter automaticamente o resultado retornado da *query* Cypher em uma resposta estruturada.
RF05	Apresentação da resposta	Funcionalidade Principal	O sistema deve retornar a resposta gerada ao usuário.
RF06	*Interface* de *chat*	*Interface*	O sistema deve fornecer uma *interface* que permita a interação com o usuário.
RF07	Criação de *chat*	*Interface*	O sistema deve permitir a criação de um *chat* novo e vazio a qualquer momento.
RF08	Exibição de histórico do *chat*	*Interface*	O sistema deve exibir o histórico completo de mensagens ao carregar um *chat* existente.
RF09	Listagem de *chats*	*Interface*	O sistema deve exibir a lista de todos os *chats* salvos em um painel lateral.
RF10	Exportação de dados	*Interface*	O sistema deve permitir a exportação de tabelas e gráficos gerados em formatos estruturados (CSV ou imagem).
RF11	Exclusão de *chat*	*Interface*	O sistema deve permitir a exclusão de qualquer *chat* salvo.
RF12	Navegação entre *chats*	*Interface*	O sistema deve permitir o usuário navegar entre os *chats* existentes.
RF13	*Feedback* de resultado vazio	Usabilidade	O sistema deve informar o usuário quando não há dados correspondentes à consulta realizada.

Fonte: Elaborado pelo autor, 2026.

Quadro 6 – Requisitos não funcionais do sistema

Código	**Nome**	**Categoria**	**Descrição**
RNF01	Tempo de resposta	*Performance*	O sistema deve processar consultas em até 30 segundos em condições normais de uso.
RNF02	Cache de perguntas similares	*Performance*	O sistema deve armazenar perguntas já processadas e reutilizar suas respectivas *queries* e configurações de resposta para consultas semanticamente similares, reduzindo o tempo de processamento e o uso de recursos de LLMs.
RNF03	Usabilidade	Usabilidade	O sistema deve possuir uma *Interface* simples e intuitiva, permitindo que usuários sem conhecimento técnico consigam realizar consultas sem necessidade de treinamento prévio.
RNF04	Tratamento de erros	Usabilidade	O sistema deve tratar erros de forma a fornecer mensagens claras e compreensíveis ao usuário, evitando termos técnicos, códigos internos e detalhes de implementação. Devem ser exibidas mensagens que indiquem o problema e orientem o usuário sobre como proceder.
RNF05	Visualização inteligente de dados	Apresentação	O sistema deve determinar automaticamente a melhor forma de apresentar os dados (texto, tabela ou gráfico).
RNF06	Manutenibilidade - arquitetura	Manutenibilidade	O código deve ser organizado em módulos, com separação de responsabilidades e utilização de padrões que facilitem sua manutenção e evolução.
RNF07	Compatibilidade com LLMs	Flexibilidade	O sistema deve permitir a substituição dos LLM utilizados sem necessidade de alterações significativas na arquitetura da aplicação.
RNF08	Coerência das respostas	Confiabilidade	O sistema deve gerar *queries* e respostas coerentes com a intenção da consulta do usuário, mantendo consistência com os dados armazenados no banco.

Fonte: Elaborado pelo autor, 2026.

Quadro 7 – Continuação dos requisitos não funcionais do sistema

Código	**Nome**	**Categoria**	**Descrição**
RNF09	Validação e integridade de *queries*	Confiabilidade	O sistema deve validar as *queries* geradas antes de sua execução, garantindo que apenas consultas válidas sejam enviadas ao banco de dados, preservando a integridade dos dados e a estabilidade do sistema.
RNF10	Persistência de conversa	Confiabilidade	O sistema deve armazenar o histórico completo de uma conversa (mensagens do usuário e respostas do sistema) localmente.
RNF11	Persistência de *chats*	Confiabilidade	O sistema deve armazenar a lista de todos os *chats* criados pelo usuário localmente.

Fonte: Elaborado pelo autor, 2026.

5.2.2 Product backlog

A partir dos requisitos levantados, foram definidas estórias de usuário que representam as funcionalidades do sistema sob a perspectiva de valor para o usuário final. As estórias foram organizadas no *product backlog* e priorizadas de acordo com sua relevância para a funcionalidade principal do sistema, incorporando também aspectos não funcionais, como desempenho, usabilidade e confiabilidade.

Quadro 8 – Estórias de usuário

ID	Estória de Usuário	Prioridade
Épico: Consultas em linguagem natural		
US01	Como usuário, quero fazer perguntas em português sobre dados presentes no ERP para obter informações, sem a necessidade de conhecimento técnico de consultas estruturadas	Alta
US02	Como usuário, quero receber respostas coerentes com o que consultei	Alta
US03	Como usuário, quero ver respostas em formato adequado (texto simples, tabela ou gráfico), para compreender facilmente os dados	Alta
US04	Como usuário, quero receber mensagens claras quando algo der errado, para entender o que aconteceu e o que fazer	Média
Épico: Organização de *chats*		
US05	Como usuário, quero poder criar novos *chats* e também conseguir excluí-los a qualquer momento	Média
US06	Como usuário, quero visualizar todo o histórico de um *chat*, para revisar perguntas e respostas anteriores	Média
US07	Como usuário, quero visualizar todos os *chats* que já tive com o sistema e conseguir navegar entre eles	Média
Épico: Exportação e compartilhamento		
US08	Como usuário, quero exportar tabelas em formato CSV, para analisar dados em planilhas	Baixa
US09	Como usuário, quero salvar gráficos como imagem, para incluir em relatórios ou apresentações	Baixa

Fonte: Elaborado pelo autor, 2026.

5.2.3 Organização em sprints

O desenvolvimento do projeto foi estruturado em seis *sprints*, cada uma com objetivos e entregas específicas, seguindo os requisitos levantados e as estórias de usuário criadas. Cada *sprint* teve como resultado a entrega de funcionalidades incrementais, permitindo a evolução gradual do protótipo até sua versão final. O Quadro 9 apresenta a organização das *sprints*, suas durações estimadas e os principais resultados obtidos em cada etapa.

Quadro 9 – Organização em *sprints*

Sprint	**Duração**	**Objetivos**	**Entregas**	**US**
1	2 semanas	Configurar ambiente de desenvolvimento e base de dados fictícia	Docker configurado (Neo4j); Ollama instalado e testado; Dependências e bibliotecas instaladas; *Script* para população do banco de dados criado	–
2	3 semanas	Implementar *backend* completo de processamento de consultas	*Cache Service* funcional; *Cypher Generator* implementado; Execução de *queries* no Neo4j; *NLP Processor* e *Visualization Service* gerando e estruturando respostas em texto; Interação com o sistema via CLI	US01, US02
3	3 semanas	Desenvolver *interface* de usuário e visualizações de resposta	*Interface* de *chat* implementada; Geração e renderização de texto/tabela/gráfico; Histórico de conversas; Painel lateral de navegação; Manipulação de *chats*; Tratamento de erros	US03, US04, US05, US06, US07
4	1 semana	Integrar *frontend* ao *backend*	Comunicação HTTP estabelecida; Fluxo completo funcional; Sincronização de estado; Persistência local implementada; Testes via *interface*	US01-US07
5	2 semanas	Otimizar *prompts* e melhorar qualidade das respostas	*Prompts* de contexto, geração de *queries* Cypher e formatação da resposta otimizados; *Queries* coerentes com a consulta em linguagem natural; Respostas estruturadas de acordo com a *query*	US02
6	1 semana	Realizar testes finais e implementar funcionalidades complementares	Exportação CSV/imagem implementada; Testes de funcionamento completo do sistema realizados; Correções de *bugs*	US08, US09

Fonte: Elaborado pelo autor, 2026.

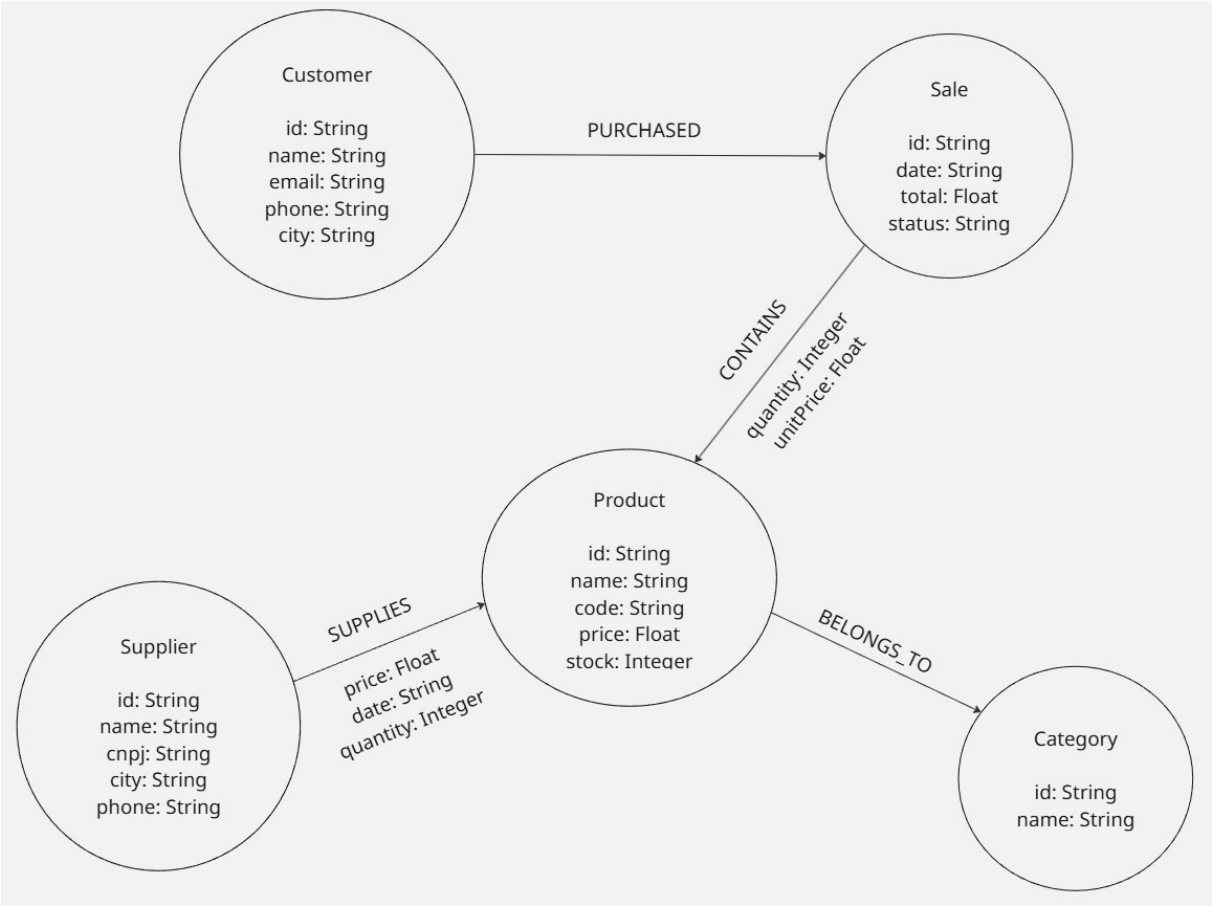
5.3 Modelagem do cenário ERP e do banco de dados em grafos

Com a definição do Neo4j como sistema gerenciador de banco de dados em grafos e a conclusão do planejamento do desenvolvimento, iniciou-se a modelagem do sistema. Inicialmente, foi realizada a modelagem de um cenário experimental representando um ambiente simulado de sistema ERP. Considerando a complexidade e abrangência de um sistema ERP completo, que envolve múltiplos módulos como finanças, recursos humanos, produção e logística, optou-se por delimitar o escopo do protótipo a um domínio específico: o módulo de vendas e gestão de produtos.

Essa delimitação justifica-se pela necessidade de concentrar esforços na validação da viabilidade técnica do protótipo, evitando a complexidade adicional de modelar um sistema completo. O módulo de vendas foi escolhido por representar um conjunto de entidades e relacionamentos típicos de sistemas corporativos — clientes, produtos, fornecedores, categorias e transações comerciais — permitindo a demonstração das capacidades do sistema em um contexto realista.

A modelagem dos dados seguiu os princípios de bancos de dados em grafos, onde informações são representadas por meio de nós (entidades) e arestas (relacionamentos) (Oliveira; Stamboni; Júnior, 2018; Angles; Gutierrez, 2008). A Figura 9 apresenta o diagrama do modelo de dados implementado, ilustrando as entidades principais e os relacionamentos entre elas.

Figura 9 – Modelo de dados em grafo do sistema ERP



Fonte: Elaborado pelo autor, 2026.

O modelo é composto por cinco tipos de nós principais, cada um representando uma entidade do domínio:

Customer (Cliente): Representa os clientes do sistema, com atributos como identificador único, nome, *email*, telefone e cidade. Cada cliente pode estar associado a múltiplas vendas.

Sale (Venda): Representa as transações comerciais realizadas, contendo identificador, data de realização, valor total e *status* da venda (*completed*, *pending* ou *cancelled*). Uma venda está sempre vinculada a um cliente e pode conter múltiplos produtos.

Product (Produto): Armazena informações sobre os produtos disponíveis, incluindo identificador, nome, código alfanumérico, preço de venda e quantidade em estoque. Cada produto pertence a uma categoria e pode ser fornecido por um ou mais fornecedores.

Category (Categoria): Agrupa produtos em categorias temáticas (Eletrônicos, Móveis, Alimentos, Papelaria), facilitando consultas e análises por segmento.

Supplier (Fornecedor): Representa os fornecedores que abastecem a em-

presa, com atributos como identificador, nome, CNPJ, cidade e telefone. Cada fornecedor pode fornecer múltiplos produtos.

Os relacionamentos entre essas entidades carregam informações adicionais que contextualizam o modelo. O relacionamento *PURCHASED* conecta clientes a vendas, mas não possui propriedades adicionais, uma vez que os dados da transação estão no nó *Sale*. Já o relacionamento *CONTAINS*, que liga vendas a produtos, armazena a quantidade vendida daquele produto e seu preço unitário, praticado no momento da venda.

O relacionamento *BELONGS_TO* estabelece a associação entre produtos e categorias de forma direta, sem propriedades adicionais. Por fim, o relacionamento *SUPPLIES*, que conecta fornecedores a produtos, armazena o preço de compra, a data da última aquisição e a quantidade comprada do produto em questão.

Para ilustrar como esses relacionamentos e entidades são dispostos no banco, a Figura 10 apresenta uma *query* Cypher gerada pelo protótipo em relação à consulta de teste "Retorne os fornecedores de produtos eletrônicos".

Figura 10 – Consulta em Cypher gerada pelo protótipo

```
MATCH (s:Supplier)-[:SUPPLIES]->(p:Product)-[:BELONGS_TO]->(cat:Category)
WHERE cat.name = "Eletronicos"
RETURN DISTINCT s.name AS SupplierName
```

Fonte: Elaborado pelo autor, 2026.

A consulta apresentada evidencia como a estrutura em grafo modelada permite a navegação entre entidades por meio de relacionamentos explícitos. Partindo do nó *Supplier*, a *query* percorre os relacionamentos *SUPPLIES* e *BELONGS_TO* até alcançar o nó *Category*, onde é aplicado o filtro pela categoria "Eletronicos". Esse padrão de navegação demonstra a adequação do LLM ao modelo de dados em grafos na criação de *queries* semânticas.

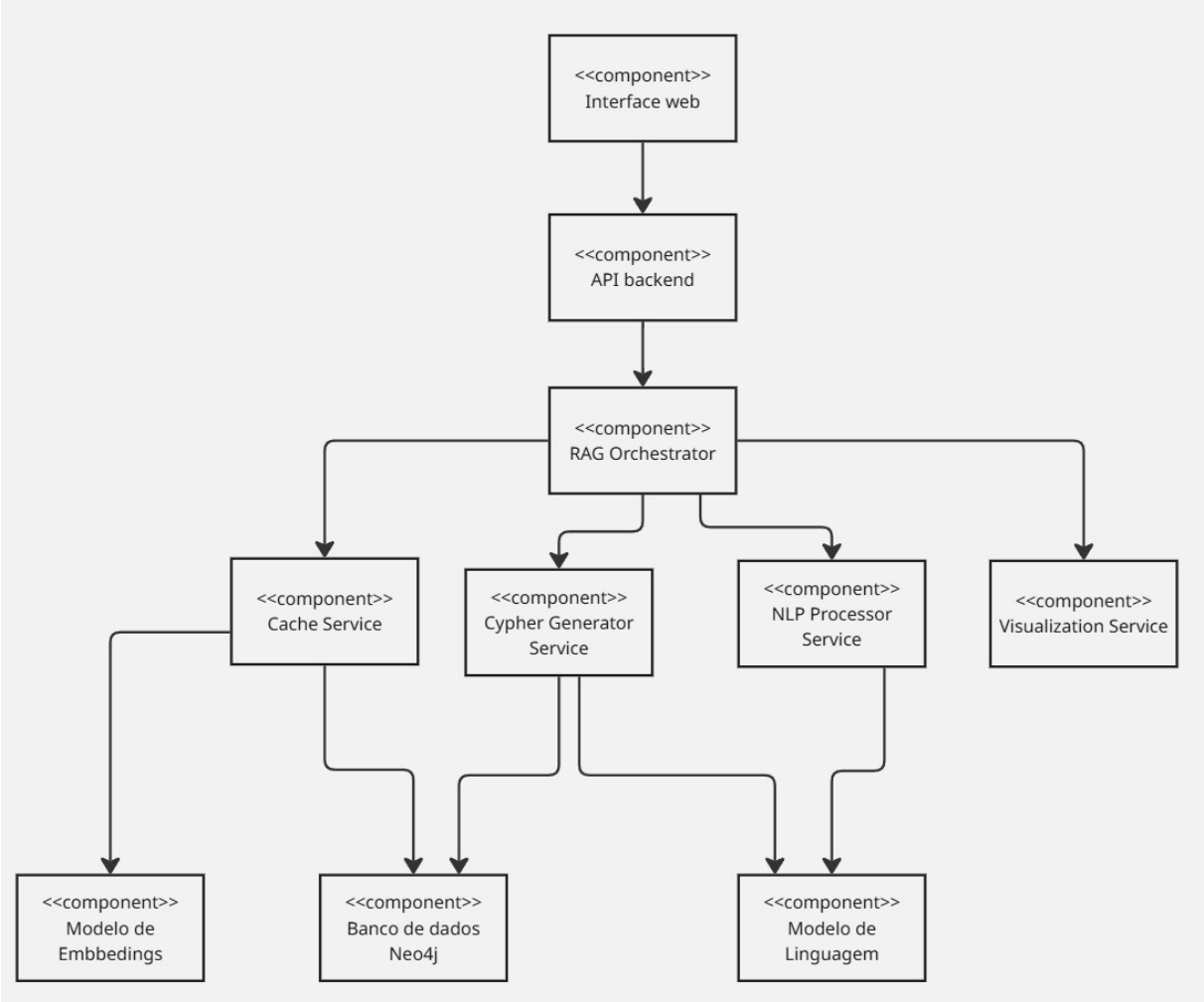
5.4 Modelagem arquitetural do sistema

Após a modelagem do banco de dados, deu-se continuidade à modelagem do sistema por meio da elaboração de diagramas que representam sua arquitetura e funcionamento. Esses diagramas permitem compreender tanto a estrutura estática quanto o comportamento dinâmico do protótipo, evidenciando as interações entre seus componentes e as funcionalidades oferecidas ao usuário (Fowler, 2004).

5.4.1 Diagrama de componentes

O diagrama de componentes apresenta a estrutura estática do sistema, evidenciando os principais módulos do sistema e suas relações (Fowler, 2004). Diferentemente da Figura 8, apresentada na Seção 4.4.3, que detalha o fluxo completo de processamento do sistema, a Figura 11 ilustra a organização estrutural dos componentes e suas interações.

Figura 11 – Diagrama de componentes do sistema



Fonte: Elaborado pelo autor, 2026.

O diagrama apresenta os componentes principais que implementam os serviços da arquitetura: *Cache Service*, responsável pela busca vetorial de consultas similares; *Cypher Generator Service*, que gera *queries* estruturadas a partir de linguagem natural; *NLP Processor Service*, que analisa resultados, gera a resposta e determina o formato de apresentação; e *Visualization Service*, que estrutura a resposta final. O componente *RAG Orchestrator* coordena a interação entre esses serviços.

Além dos serviços centrais, o diagrama inclui outros componentes essenciais para o funcionamento do sistema. A *Interface Web* é responsável pela interação com o usuário, permitindo a entrada de consultas em linguagem natural e a visualização das respostas. O *API Backend* atua como intermediário entre a *interface* e o orquestrador, recebendo as requisições e encaminhando-as para processamento.

Presentes fora das camadas de *backend* e *frontend* estão: O modelo de *embeddings*, utilizado para gerar representações vetoriais das consultas, possibilitando a busca por similaridade no mecanismo de cache; O modelo de linguagem (LLM), empregado na interpretação das consultas e na geração de respostas em linguagem natural; E por fim, o banco de dados Neo4j, responsável pelo armazenamento dos dados em grafo e pela execução das *queries*.

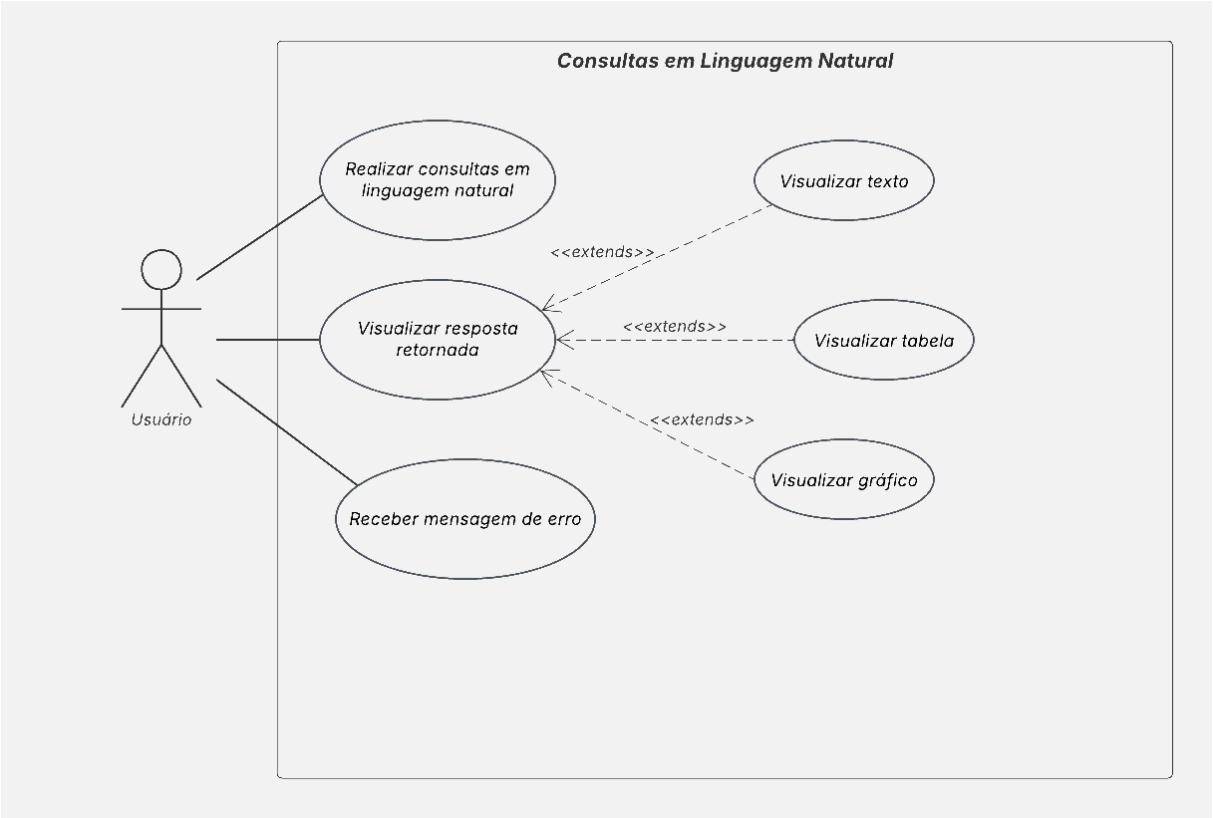
5.4.2 Diagramas de caso de uso

Os diagramas de caso de uso representam as funcionalidades do sistema sob a perspectiva do usuário, identificando os principais fluxos de interação (Fowler, 2004). Os diagramas elaborados seguem os três épicos definidos nas estórias de usuário.

5.4.2.1 Épico: Consultas em linguagem natural

A Figura 12 apresenta o diagrama de casos de uso relacionados à execução de consultas em linguagem natural e visualização de respostas. Os casos retratados no diagrama se referem as principais funcionalidades do protótipo: realizar consultas em linguagem natural, visualizar respostas formatadas automaticamente e receber *feedback* em caso de erros.

Figura 12 – Diagrama de caso de uso: Consultas em linguagem natural

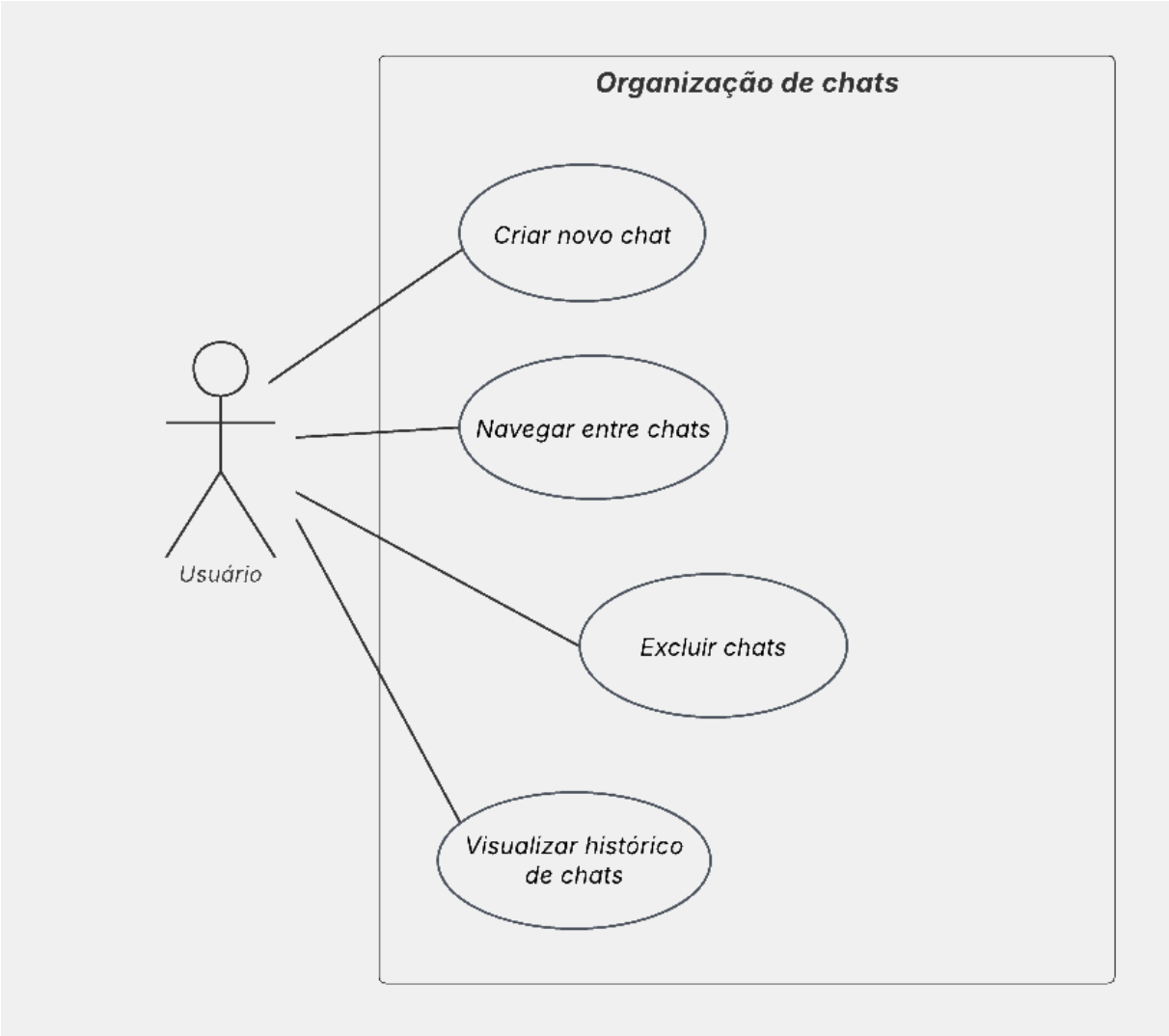


Fonte: Elaborado pelo autor, 2026.

5.4.2.2 Épico: Organização de *chats*

A Figura 13 apresenta os casos de uso relacionados ao gerenciamento de *chats*. Este diagrama representa as funcionalidades de organização: criar novos *chats*, alternar entre *chats* existentes, visualizar histórico e excluir *chats* antigos.

Figura 13 – Diagrama de caso de uso: Organização de *chats*

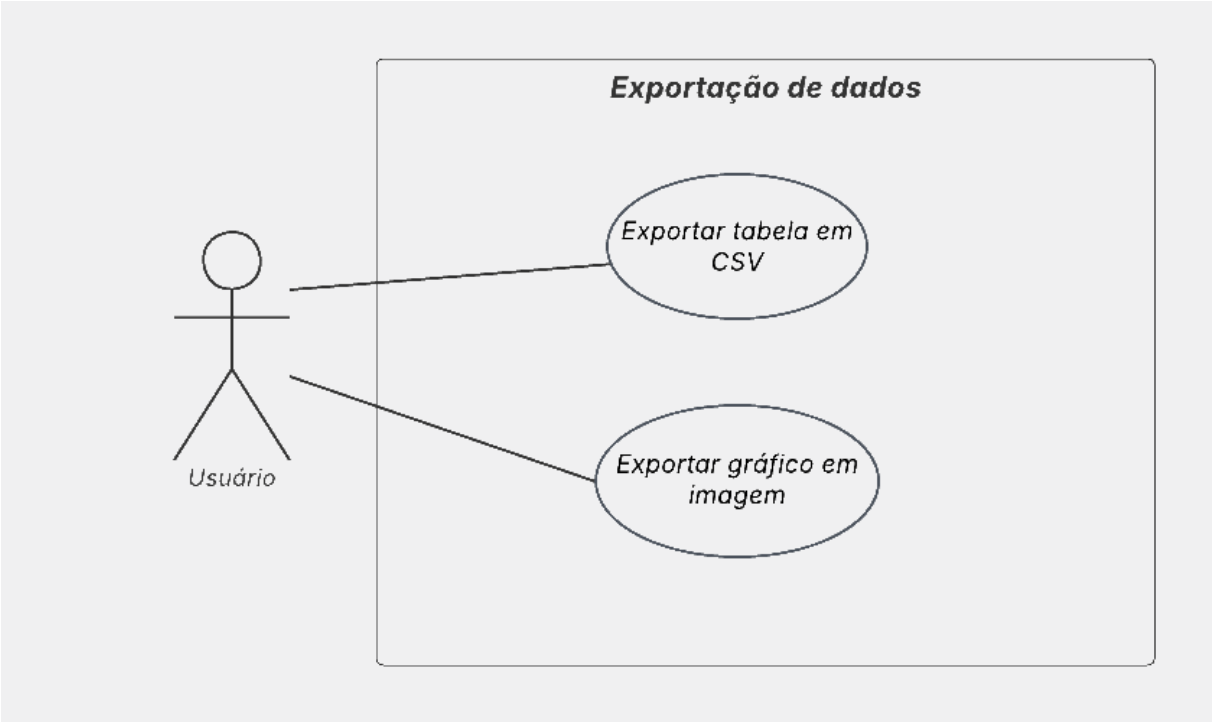


Fonte: Elaborado pelo autor, 2026.

5.4.2.3 Épico: Exportação e compartilhamento

A Figura 14 apresenta os casos de uso relacionados à exportação de dados. Este diagrama representa as funcionalidades complementares de exportação: Exportar tabelas em formato CSV e exportar gráficos como imagem.

Figura 14 – Diagrama de caso de uso: Exportação de dados



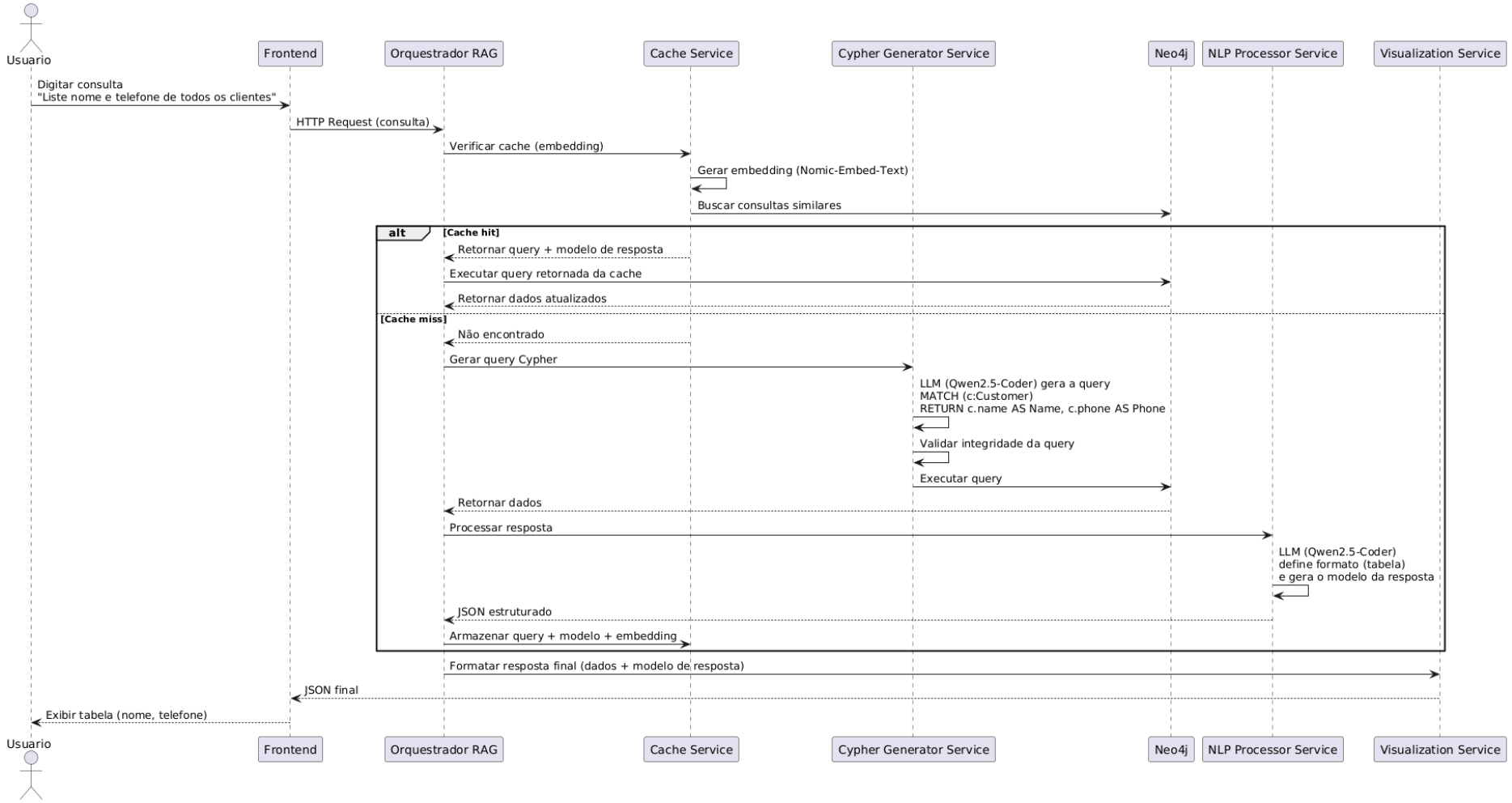
Fonte: Elaborado pelo autor, 2026.

5.4.3 Diagrama de sequência

Os diagramas de sequência são utilizados para representar a interação temporal entre os componentes de um sistema ao longo da execução de um processo (Fowler, 2004). A Figura 15 apresenta o fluxo detalhado da principal funcionalidade do projeto, correspondente ao processamento de uma consulta em linguagem natural. A consulta de exemplo utilizada foi "Liste o nome e telefone de todos os clientes".

O diagrama evidencia a orquestração entre os serviços responsáveis por cada etapa do processamento, desde a entrada da consulta até a exibição da resposta final ao usuário. São apresentados os dois fluxos possíveis: o fluxo completo, quando a consulta não está em cache, percorrendo as etapas de geração de *query* Cypher, execução no banco de dados, geração e formatação da resposta; e o fluxo otimizado, quando o sistema identifica uma consulta similar em cache, reutilizando a *query* e o modelo previamente processados. Esse segundo caminho reduz significativamente o tempo de resposta ao evitar a etapa mais custosa do processo: a inferência pelo LLM.

Figura 15 – Diagrama de sequência: Processamento de consulta



Fonte: Elaborado pelo autor,2026.

5.5 Prototipação da *interface* do sistema

Após a conclusão das modelagens necessárias para o início do desenvolvimento, a etapa seguinte consistiu na prototipação da *interface* do sistema. A *interface* foi projetada para permitir a execução de consultas em linguagem natural e a visualização das respostas retornadas, que podem ser apresentadas em formato textual, tabular ou gráfico, conforme a natureza dos dados recuperados.

Sua estrutura segue o padrão de aplicações de *chat* baseados em LLMs, como ChatGPT e DeepSeek, devido à similaridade nas funcionalidades oferecidas: entrada de texto livre, histórico de conversação e exibição de respostas formatadas. A prototipação foi realizada utilizando a ferramenta Figma.

A estrutura visual foi organizada em três seções principais: o painel lateral esquerdo, que exibe a lista de *chats* salvos e permite a navegação, criação e exclusão destes; a área central, que apresenta o histórico completo do *chat* selecionado, incluindo mensagens do usuário e respostas do sistema; e a área inferior, onde se localiza o campo de entrada de texto e o botão de envio da consulta. Quando não há nenhum *chat* selecionado, o campo de entrada de texto é localizado na área central da *interface*.

A Figura 16 corresponde a tela inicial do protótipo, a qual o usuário se depara quando abre a aplicação. Ela contém o campo de entrada de texto para que o usuário realize consulta e, ao lado, o histórico de *chats* entre o usuário e o sistema.

Figura 16 – Tela principal do protótipo

Fonte: Elaborado pelo autor, 2025.

A Figura 17 apresenta uma tela que simula a interação entre o usuário e o protótipo. Nela, observa-se a consulta realizada pelo usuário, bem como a resposta gerada pelo sistema em formato tabular.

Figura 17 – Simulação de uma conversa entre o usuário e o protótipo

Fonte: Elaborado pelo autor, 2025.

5.6 Implementação do protótipo

A implementação do protótipo seguiu a estrutura definida nas *sprints* de desenvolvimento, abrangendo desde a configuração do ambiente até a integração completa dos componentes. Esta seção descreve os aspectos técnicos da construção do protótipo, detalhando as decisões de implementação, os recursos utilizados e os desafios enfrentados durante o desenvolvimento.

5.6.1 Configuração do ambiente de desenvolvimento

A configuração do ambiente de desenvolvimento foi a primeira etapa da implementação, estabelecendo a infraestrutura necessária para execução do sistema. O banco de dados Neo4j foi configurado utilizando Docker, através de um arquivo `docker-compose.yml` que define o *container* com as configurações de portas, credenciais e volumes persistentes. A escolha do uso do Docker se deu principalmente pela facilidade de portabilidade do ambiente.

O servidor Ollama foi instalado localmente para execução dos modelos de

linguagem. Dessa forma todo o processamento do *backend* foi concentrado na máquina responsável pela execução do sistema. Foram baixados os modelos Qwen2.5-Coder e Nomic-Embed-Text, que são utilizados respectivamente para geração de *queries* e respostas, e para criação de *embeddings*. A escolha final desses modelos específicos é detalhada posteriormente na Seção 5.6.6.

A estrutura do projeto foi organizada em diretórios separados para *backend* (`backend/`), *frontend* (`frontend/`) e banco de dados (`database/`), seguindo boas práticas de organização de código. As dependências do projeto foram gerenciadas via `npm`, com instalação de bibliotecas essenciais como LangChain para orquestração de LLMs, o *driver* oficial do Neo4j para JavaScript, e @faker-js/faker para geração de dados fictícios.

As variáveis de ambiente foram centralizadas em um arquivo `.env`, responsável por armazenar configurações sensíveis como credenciais do banco de dados, URLs de serviços, LLMs utilizados e parâmetros de funcionamento do sistema. Uma variável importante definida nesse arquivo é o `NEO4J_VECTOR_THRESHOLD`, que determina o limiar de similaridade para considerar uma consulta como equivalente a outra já processada no cache vetorial. Esse valor varia no intervalo de 0 a 1; quanto mais próximo de 1, maior é a exigência de similaridade semântica entre as consultas.

5.6.2 Modelagem e população do banco de dados

A modelagem do banco de dados foi realizada por meio da definição de nós e relacionamentos no Neo4j, representando as entidades *Customer*, *Product*, *Category*, *Supplier* e *Sale*, bem como suas respectivas conexões. Nesse contexto, adota-se um esquema implícito, característico do Neo4j, no qual a estrutura é definida dinamicamente a partir dos dados inseridos.

Além disso, é utilizado um índice vetorial denominado `agent_index`, associado a nós do tipo `Chunk` e à propriedade `question`, permitindo a recuperação de consultas semanticamente semelhantes no mecanismo de cache.

O *script* de população do banco `seed.js` utiliza a biblioteca @faker-js/faker para gerar dados fictícios realistas, criando registros para as cinco entidades modeladas e seus relacionamentos. O *script* foi projetado para verificar a existência de dados antes de popular o banco e permitir a adição de novos clientes e vendas em execuções subsequentes sem duplicar produtos, categorias ou fornecedores já existentes.

Em uma execução inicial do *script*, são gerados 100 produtos distribuídos em 4 categorias (Eletrônicos, Móveis, Alimentos e Papelaria), 9 fornecedores, 50 clientes iniciais e 5000 vendas, simulando transações comerciais ocorridas durante um período de tempo. Em execuções subsequentes, o *script* verifica a existência de dados e adiciona apenas novos clientes e novas vendas, sem impactar os nós já existentes.

5.6.3 Implementação dos serviços backend

A camada de serviços *backend* foi implementada seguindo os princípios da arquitetura SOA, com cada serviço encapsulando uma responsabilidade específica e comunicando-se através do orquestrador RAG. Todos os serviços foram desenvolvidos como módulos JavaScript exportáveis, permitindo reutilização e testabilidade.

5.6.3.1 Cache Service

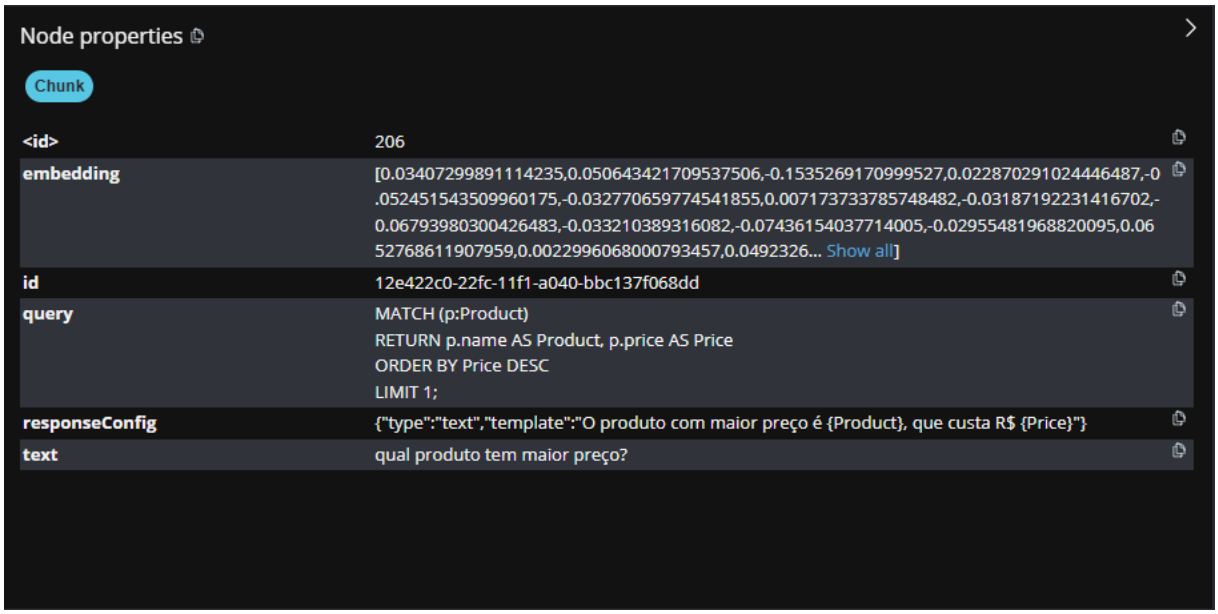
O *Cache Service* é responsável pelo sistema de reutilização de consultas através de busca vetorial por similaridade semântica. O serviço implementa duas funções principais: `checkCache` e `saveToCache`.

A função `checkCache` recebe uma consulta em linguagem natural, obtém seu *embedding* vetorial utilizando o modelo Nomic-Embed-Text através da integração LangChain com Ollama, e realiza uma busca por similaridade no índice vetorial do Neo4j. A busca retorna a consulta previamente processada com maior similaridade, juntamente com seu *score*, que é então comparado ao limiar definido para determinar se o resultado será considerado válido.

Caso uma consulta similar seja encontrada, o sistema recupera a *query* Cypher e o modelo de resposta armazenados. É importante ressaltar que o cache não armazena o resultado final da consulta, mas sim a *query* Cypher gerada e o modelo de formatação determinado. Essa estratégia permite reutilizar a lógica de processamento enquanto garante que os dados retornados estejam sempre atualizados, uma vez que a *query* é executada novamente no momento da recuperação, e o modelo de resposta salvo não possui os dados da última resposta gerada, mas *placeholders* em seus lugares, o que permite que o sistema os substitua pelos dados retornados na execução atual.

A função `saveToCache` é responsável por persistir novas consultas processadas no banco de dados, criando um nó `Chunk` que armazena a consulta em linguagem natural, a *query* Cypher gerada, o modelo de resposta com *placeholders* para os dados dinâmicos e o *embedding* vetorial da consulta. A Figura 18 apresenta um exemplo desse nó, no qual o campo `id` corresponde ao identificador único, `text` armazena a pergunta do usuário, `query` contém a *query* Cypher, `responseConfig` representa o modelo de resposta estruturado e `embedding` armazena o vetor utilizado nas operações de similaridade semântica.

Figura 18 – *Chunk* de consulta armazenado



Fonte: Elaborado pelo autor, 2026.

5.6.3.2 Cypher Generator Service

O *Cypher Generator Service* é responsável pela conversão de perguntas em linguagem natural para *queries* estruturadas em Cypher. O serviço implementa um *pipeline* de geração e validação que garante a segurança e correção das *queries* produzidas.

O processo inicia-se com a leitura de dois arquivos de *prompt*: `context.md`, que contém informações sobre o domínio dos dados, a estrutura do banco de dados, e exemplos de *queries*, e `nlp-to-cypher.md`, que define as instruções necessárias para conversão de linguagem natural em Cypher. Além disso, o esquema atual do banco de dados é obtido dinamicamente por meio da função `getSchema()`, sendo utilizado como contexto adicional para o modelo. Esses elementos são combinados com a pergunta do usuário e enviados ao modelo Qwen2.5-Coder via Ollama para a geração da *query*.

A *query* gerada pelo LLM passa por duas camadas de validação. Primeiramente, uma validação de segurança verifica se a *query* contém comandos de escrita (`CREATE`, `MERGE`, `DELETE`, `SET`, `REMOVE`, `DROP`, `UPDATE`, entre outros), rejeitando qualquer tentativa de modificação do banco de dados. Essa validação é fundamental para garantir que o sistema opere apenas em modo leitura. Caso seja identificado algum comando não permitido, o fluxo é interrompido e uma mensagem de erro é retornada ao usuário.

Quando a *query* é aprovada na primeira validação, uma validação estrutural é executada, utilizando o comando `EXPLAIN` do Neo4j. Esse comando retorna o plano

de execução da *query* sem executá-la de fato, permitindo detectar erros de sintaxe, referências a propriedades ou *labels* inexistentes, e outras inconsistências estruturais. *Queries* que falham nessa validação são descartadas, e o usuário é orientado a reformular sua pergunta.

Após aprovação nas etapas de validação, a *query* é executada no Neo4j. Caso não sejam retornados resultados correspondentes, o sistema responde ao usuário com uma mensagem informando a ausência de dados para a consulta realizada. Essa abordagem está alinhada aos principais conceitos da RAG, ao evitar que o LLM produza respostas fictícias quando não há evidências nos dados recuperados, apenas para satisfazer o usuário. Quando são retornados resultados, estes são encaminhados ao orquestrador para processamento nas etapas subsequentes.

5.6.3.3 NLP Processor Service

O *NLP Processor Service* é responsável por analisar os resultados retornados pelo banco de dados e determinar automaticamente a melhor forma de apresentação. O serviço utiliza o arquivo de *prompt* `response-template.md`, que contém instruções para seleção do tipo de visualização (texto, tabela ou gráfico), diretrizes de formatação e exemplos de respostas geradas.

O modelo Qwen2.5-Coder também é empregado nessa etapa, recebendo a pergunta original e os dados retornados pelo Neo4j, e gerando uma resposta textual estruturada no formato JSON, que define o tipo de visualização, o modelo de resposta e a organização dos dados. Essa configuração pode incluir informações como colunas de tabelas, eixos de gráficos e tipo de gráfico (barras, linhas ou pizza).

Após a geração, a resposta textual é processada para remoção de formatações adicionais. Em seguida, a saída textual, esperada no formato JSON, é validada e convertida para uma estrutura de dados utilizável pelo sistema. Caso ocorra falha na interpretação da saída do modelo, o sistema utiliza um mecanismo de *fallback*, retornando uma resposta do tipo textual simples, garantindo robustez no processamento mesmo diante de saídas inválidas do modelo.

5.6.3.4 Visualization Service

O *Visualization Service* é responsável pela estruturação final da resposta em formato JSON, preparando os dados para consumo pela *interface*. O serviço recebe a configuração gerada pelo *NLP Processor Service* e os dados retornados pelo banco de dados, aplicando transformações necessárias para gerar a resposta final.

Para respostas do tipo textual, o serviço realiza a substituição dos *placeholders* presentes no modelo de resposta pelos valores reais obtidos da consulta ao

banco de dados. Essa substituição é feita com base no primeiro registro retornado, permitindo a construção dinâmica da resposta final.

Para tabelas, o serviço estrutura a resposta definindo as colunas com base na configuração fornecida. Os dados são mantidos em formato bruto, organizados em linhas e colunas para consumo direto no *frontend*.

Para visualizações gráficas, o serviço organiza e encapsula os dados no formato padronizado, incluindo o tipo de gráfico, colunas e dados retornados pela consulta. Esses dados são consumidos pelo *frontend*, que é responsável por transformá-los em estruturas compatíveis com a biblioteca Chart.js e renderizar o gráfico.

Em todos os casos, o serviço garante a padronização da resposta em um formato JSON consistente, facilitando seu processamento e renderização no *frontend*.

5.6.3.5 RAG Service

O *RAG Service* atua como orquestrador central, coordenando a execução dos serviços descritos anteriormente. O serviço implementa o fluxo completo de processamento de consultas, desde a entrada do usuário até a resposta formatada.

Inicialmente, o sistema consulta o *Cache Service* para verificar se já existe uma resposta equivalente armazenada. Caso uma correspondência seja encontrada, a *query* e o modelo de resposta previamente gerado são reutilizados, e a *query* é executada diretamente no Neo4j.

Caso contrário, o sistema aciona o *Cypher Generator Service* para a geração de uma *query* estruturada, que é então validada e executada no banco de dados. Na sequência, os resultados obtidos são encaminhados para o *NLP Processor Service*, responsável por definir a estrutura da resposta com base no tipo de saída desejado. Após a definição da resposta, o sistema aciona o *Cache Service* para armazenar no cache a *query* e o modelo gerado, permitindo reutilização em consultas futuras.

Por fim, independentemente da origem dos dados - seja da cache ou a geração nesta execução - eles são enviados ao *Visualization Service*, que realiza a transformação final dos dados retornados em um formato JSON padronizado para consumo pelo *frontend*, sendo executado sempre como etapa final do fluxo.

5.6.4 API REST

A camada de comunicação entre *frontend* e *backend* foi implementada utilizando o módulo HTTP nativo do Node.js, sem dependência de *frameworks* adicionais. O arquivo `chat.controller.js` implementa um servidor HTTP completo que desempenha duas funções: expor *endpoints* REST para processamento de consultas e servir os arquivos estáticos da *interface* de usuário. O servidor foi configurado para escutar

na porta 3002, permitindo acesso tanto à API quanto ao *frontend* através da mesma origem.

O *endpoint* principal `POST /v1/chat` recebe consultas em linguagem natural através de requisições HTTP contendo um objeto JSON com a propriedade `prompt`. A requisição é processada por meio da leitura assíncrona do corpo da requisição e posterior conversão do JSON recebido.

Em seguida, a consulta é encaminhada ao *RAG Service* por meio da função `processQuery`, responsável por coordenar todo o fluxo de processamento. A resposta retornada pelo serviço é serializada em JSON e enviada ao cliente com *status* HTTP 200.

O tratamento de erros no nível da rota de *chat* é realizado de forma controlada, retornando mensagens estruturadas em JSON mesmo em casos de falha lógica, mantendo o código de *status* HTTP 200. Já erros não tratados em nível de servidor são capturados por um *handler* global, que retorna *status* HTTP 500.

O suporte a *Cross-Origin Resource Sharing* (CORS) foi implementado manualmente através da configuração de *headers* HTTP específicos: `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods` e `Access-Control-Allow-Headers`. Essa configuração permite que o *frontend* realize requisições ao *backend* sem restrições de mesma origem durante o desenvolvimento.

O servidor também implementa funcionalidade de arquivos estáticos, servindo o *frontend* através de rotas GET. A função `serveStaticFile` lê arquivos do sistema de arquivos e os retorna com *Content-Type* apropriado baseado na extensão do arquivo, utilizando um mapeamento de tipos MIME para HTML, JavaScript, CSS, JSON e imagens.

O arquivo `api.js` encapsula a lógica de comunicação HTTP do lado do cliente. Implementado como módulo JavaScript, exporta duas funções principais: `sendMessage`, que realiza requisições POST ao *endpoint* de chat utilizando a Fetch API do navegador, e `checkConnection`, que verifica a disponibilidade do *backend* por meio de uma requisição OPTIONS.

A função `sendMessage` trata erros de rede e HTTP, lançando exceções com mensagens descritivas que são capturadas pela *interface* de usuário. Essa camada de abstração separa a lógica de comunicação da lógica de *interface*, facilitando manutenção e testes.

O servidor foi projetado para ser *stateless*, não mantendo estado entre requisições. Cada consulta é processada de forma independente, com toda informação necessária contida na requisição HTTP.

5.6.5 Implementação da interface de usuário

A interface de usuário foi desenvolvida utilizando HTML, CSS e JavaScript, sem o uso de *frameworks* como React ou Vue. Essa abordagem permitiu reduzir a complexidade do projeto, mantendo o foco na lógica de processamento implementada no *backend*.

5.6.5.1 Estrutura HTML e estilização

O arquivo `index.html` define a estrutura principal da *interface*, organizada em duas áreas: um painel lateral à esquerda, responsável pela criação e listagem de *chats*, e uma área central, destinada à exibição da troca de mensagens entre o usuário e o sistema.

A *interface* apresenta dois estados distintos: um estado inicial, no qual a entrada de texto é centralizada, exibido quando não há nenhum *chat* selecionado, e um estado de *chat*, que apresenta o histórico de mensagens com o campo de entrada posicionado na parte inferior.

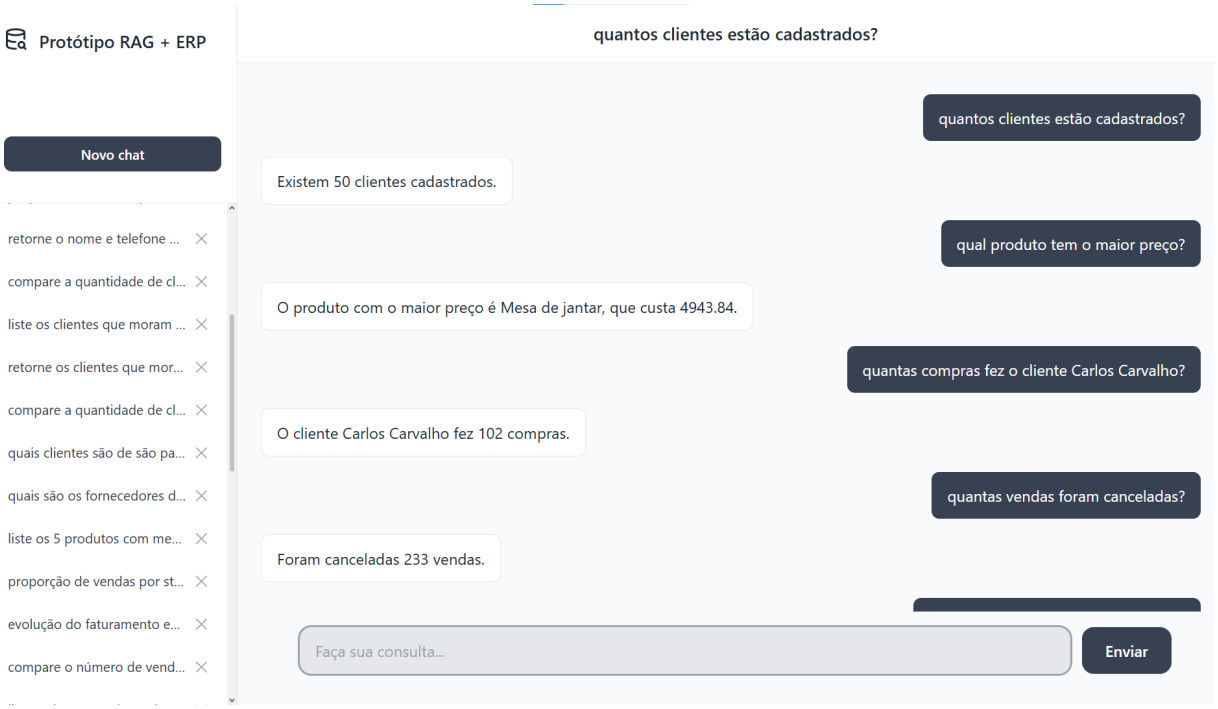
A estilização foi realizada com Tailwind CSS, permitindo a aplicação de estilos diretamente nas classes HTML. Animações complementares foram implementadas em arquivo CSS separado, incluindo efeitos de transição para mensagens e indicador de carregamento. Bibliotecas externas são carregadas via CDN, incluindo Lucide para ícones e Chart.js para visualização de gráficos.

As Figuras 19 e 20 apresentam os dois estados principais da *interface* após a implementação.

Figura 19 – *Interface* sem *chat* selecionado

Fonte: Elaborado pelo autor, 2026.

Figura 20 – *Interface* com *chat* selecionado



Fonte: Elaborado pelo autor, 2026.

5.6.5.2 Gerenciamento de estado e fluxo de mensagens

A lógica da aplicação é implementada no arquivo `app.js`, que mantém o estado por meio da variável `currentChatId`, responsável por identificar o *chat* ativo.

O envio de mensagens é realizado pela função `handleSendMessage`, que captura a entrada do usuário, cria um novo *chat* quando necessário, registra a mensagem no histórico e realiza a comunicação com o *backend*. A resposta recebida é processada e renderizada conforme o tipo de conteúdo retornado. Durante esse processo, um indicador de carregamento é exibido, sendo removido após a conclusão. Erros são tratados e apresentados ao usuário em forma de mensagens claras e instrutivas.

5.6.5.3 Renderização de visualizações

A renderização das respostas é realizada de forma dinâmica, com base na propriedade `type` do JSON retornado pelo *backend*. Mensagens textuais são exibidas em formato de conversa, com diferenciação visual entre usuário e sistema, incluindo tratamento de conteúdo para evitar execução de código malicioso.

Para respostas tabulares, os dados são organizados em tabelas HTML, com possibilidade de exportação em formato CSV. Já para visualizações gráficas, é utilizada a biblioteca Chart.js, que permite a exibição de gráficos de barras, linhas ou pizza a partir dos dados estruturados recebidos.

5.6.5.4 Persistência local

A persistência de dados é realizada no navegador por meio da API `LocalStorage`. As conversas são armazenadas em formato JSON, permitindo a manutenção do histórico entre sessões. O módulo responsável implementa operações de criação, leitura e exclusão de *chats*, possibilitando o gerenciamento destes diretamente na *interface*. Cada conversa armazena seu identificador, título, mensagens e informações temporais, garantindo organização e recuperação dos dados.

5.6.6 Engenharia de prompts

A engenharia de *prompts* constituiu de um processo iterativo de refinamento ao longo das *Sprints* 2 e 5, representando uma das etapas mais críticas do desenvolvimento. Os *prompts* evoluíram significativamente desde versões iniciais simplificadas até estruturas complexas capazes de garantir maior precisão na geração de *queries* e consistência na formatação das respostas.

O desenvolvimento dos *prompts* enfrentou desafios relacionados ao tamanho e complexidade das instruções. Embora o LLM Qwen2.5-Coder suporte janelas de contexto de até 32K *tokens*, a configuração operacional limitou o processamento a aproximadamente 8.192 *tokens*, considerando as restrições de *hardware*. Além disso, observou-se que *prompts* excessivamente extensos e com várias regras diferentes, mesmo dentro do limite de contexto, causavam confusão no modelo, que falhava em seguir algumas regras específicas. O equilíbrio adotado resultou em *prompts* com aproximadamente 5.000 a 6.000 *tokens*, distribuídos entre descrição do esquema, exemplos práticos e instruções detalhadas.

5.6.6.1 *Prompt* de contexto

O arquivo `context.md` descreve o esquema completo do banco de dados em formato textual, incluindo as cinco entidades, seus atributos, relacionamentos e suas propriedades. O *prompt* também incorpora regras de negócio críticas, como a obrigatoriedade de toda venda pertencer a um cliente e conter ao menos um produto.

Uma decisão importante foi concentrar os exemplos de *queries* Cypher nesse arquivo, em vez de distribuí-los entre múltiplos *prompts*. Testes práticos demonstraram que o modelo apresentava melhor desempenho quando os exemplos estavam localizados junto à descrição do esquema. Foram incluídos exemplos cobrindo diferentes tipos de *queries*: agregações, filtros, ordenações, navegação entre relacionamentos e cálculos envolvendo propriedades de relacionamentos, seguindo a estratégia *few-shot learning* de engenharia de *prompts*.

5.6.6.2 *Prompt* de conversão para cypher

O arquivo `nlp-to-cypher.md` define regras rigorosas para conversão de linguagem natural em *queries* Cypher válidas. Diferentemente de abordagens baseadas em exemplos, esse *prompt* enfatiza instruções explícitas e restrições formais.

A principal dificuldade enfrentada durante o desenvolvimento desse *prompt* foi garantir que o modelo retornasse *aliases* em todas as cláusulas `RETURN`, requisito fundamental para o funcionamento do sistema de cache. O cache armazena não o resultado final da consulta, mas a *query* Cypher gerada e um modelo de resposta contendo *placeholders*. Quando uma consulta similar é recuperada do cache, a *query* é executada novamente com dados atualizados, e os *placeholders* são substituídos pelos valores reais. Se o modelo retornasse valores diretamente sem *aliases*, os *placeholders* não poderiam ser criados, impossibilitando a reutilização.

Para resolver esse problema, regras explícitas foram adicionadas no início e no fim do *prompt*, utilizando formatação em negrito e letras maiúsculas para enfatizar a

criticidade: `"EVERY RETURN statement MUST have an AS alias - NO EXCEPTIONS"` e `"Remember: EVERY field in RETURN must have AS alias!"`. Essas repetições estratégicas ao longo do *prompt* reduziram significativamente a ocorrência de *queries* sem *aliases*.

O *prompt* também especifica regras de nomenclatura, direção correta de relacionamentos, e acesso a propriedades de relacionamentos por meio de regras explícitas. Alguns exemplos foram mantidos nesse arquivo para reforçar regras específicas, enquanto a maioria dos exemplos permaneceu no *prompt* de contexto.

5.6.6.3 *Prompt* de formatação de resposta

O arquivo `response-template.md` foi o arquivo que passou pelo maior número de iterações. Inicialmente, durante a fase de testes pelo CLI, o *prompt* foi projetado apenas para gerar respostas em texto natural. Com a evolução do projeto, foi expandido para suportar múltiplos tipos de visualização: texto, tabela e gráfico.

A versão final incorpora regras que orientam o modelo na escolha do tipo apropriado baseado na estrutura dos dados retornados e no contexto da pergunta. Regras específicas definem quando usar cada tipo: texto para agregações simples ou valores únicos, tabelas para múltiplos registros estruturados, e gráficos para comparações numéricas ou tendências temporais. Para gráficos, o *prompt* especifica critérios de seleção entre barras (comparações categóricas), linhas (séries temporais) e pizza (distribuições proporcionais).

Um desafio significativo foi instruir o modelo a retornar JSON estruturado sem incluir dados reais, apenas *placeholders* correspondentes aos *aliases* da *query*. Várias regras claras foram adicionadas para reforçar esse comando, como: `**ALWAYS return ONLY valid JSON** - no markdown, no code blocks, no explanations`, `**Use placeholders** that match EXACTLY the keys from the data (wrap keys in curly braces)` e `**DO NOT use actual data values** - only placeholders`.

Diversas iterações foram necessárias para estabelecer regras claras: para respostas em texto, gerar frases completas em linguagem natural utilizando *placeholders*; para tabelas e gráficos, gerar títulos genéricos sem *placeholders*, acompanhados de *arrays* de colunas.

Adicionalmente, exemplos detalhados foram incluídos cobrindo diferentes cenários: agregações únicas, múltiplos registros, comparações temporais e categóricas. Cada exemplo demonstra tanto os formatos e conteúdos corretos quanto erros comuns a serem evitados.

5.6.6.4 Seleção de LLM

A escolha definitiva do LLM ocorreu durante a fase de refinamento dos *prompts*. Inicialmente, modelos menores e mais antigos foram testados, como o Gemma-7b (gerar respostas finais) e o deepseek-coder: 6.7b (gerar *queries*), mas conforme os *prompts* se tornaram mais complexos e detalhados, esses modelos apresentaram dificuldades em seguir consistentemente as instruções.

Dessa forma, o Qwen2.5-Coder foi selecionado para substituir ambos e executar as duas funções, por demonstrar capacidade superior de processar *prompts* extensos e seguir regras complexas de forma consistente. Não foram realizadas métricas formais de comparação, mas testes práticos evidenciaram melhoria significativa na qualidade das *queries* geradas e na consistência das respostas formatadas.

5.7 Testes e validação do sistema

Esta seção apresenta a avaliação prática do protótipo desenvolvido, com o objetivo de verificar sua capacidade de interpretar consultas em linguagem natural, gerar *queries* Cypher corretas e retornar respostas formatadas em diferentes tipos de visualização. Os testes foram conduzidos de forma empírica ao longo e ao fim do desenvolvimento, priorizando a análise qualitativa do comportamento do sistema em cenários representativos.

5.7.1 Ambiente de testes

Os testes foram realizados em ambiente local, utilizando o ambiente e tecnologias citados na Seção 4.2. O *hardware* utilizado constituiu um fator relevante para o desempenho do sistema, uma vez que a configuração presente impactou diretamente na escolha do LLM e na quantidade de contexto utilizável. A base de dados utilizada nos testes contém 100 produtos, 50 clientes, 9 fornecedores, 4 categorias e 5.000 vendas, oferecendo um volume adequado para simular consultas realistas em um cenário de ERP.

5.7.2 Cenários de teste

Diferentes tipos de consultas foram executadas no protótipo, com o objetivo de avaliar sua capacidade em lidar com distintos níveis de complexidade e variados formatos de resposta, desde respostas textuais com agregações simples até respostas que exigem cálculos e comparações representadas por meio de gráficos.

Consultas envolvendo agregações simples, superlativos e contagem, foram

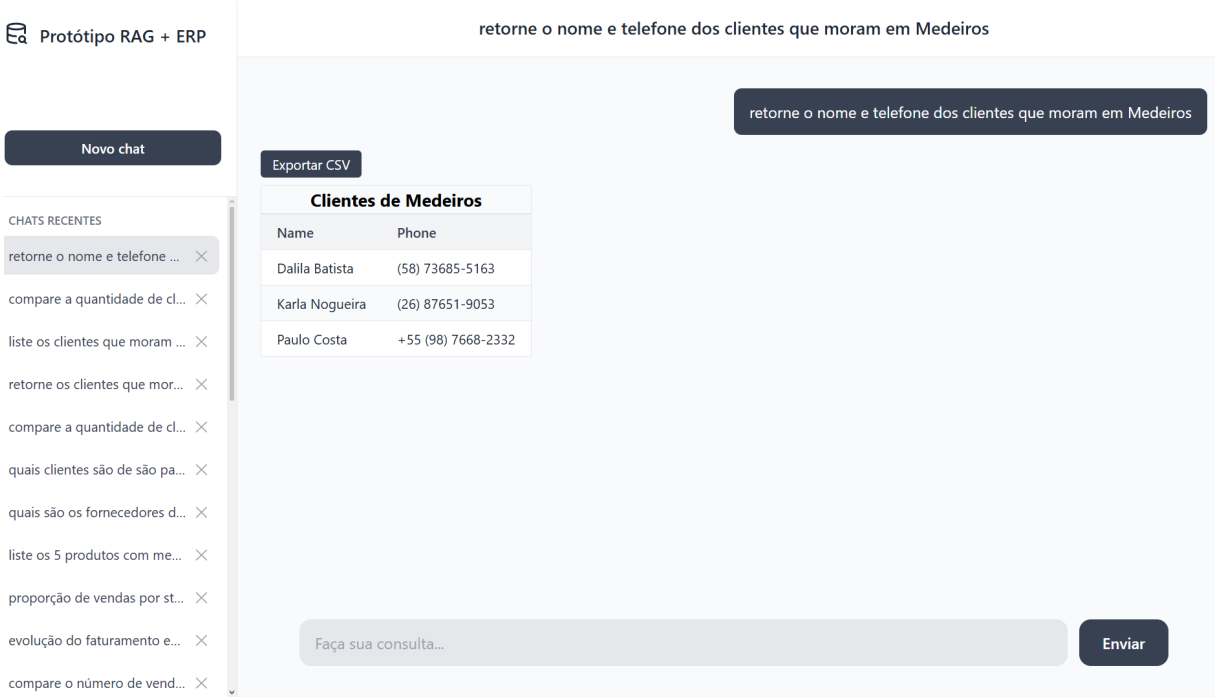
corretamente interpretadas e retornadas em formato textual. A Figura 21 apresenta vários casos assim, nos quais o sistema responde a diferentes consultas que seguem esse padrão.

Figura 21 – Exemplo de resposta textual

Fonte: Elaborado pelo autor, 2026.

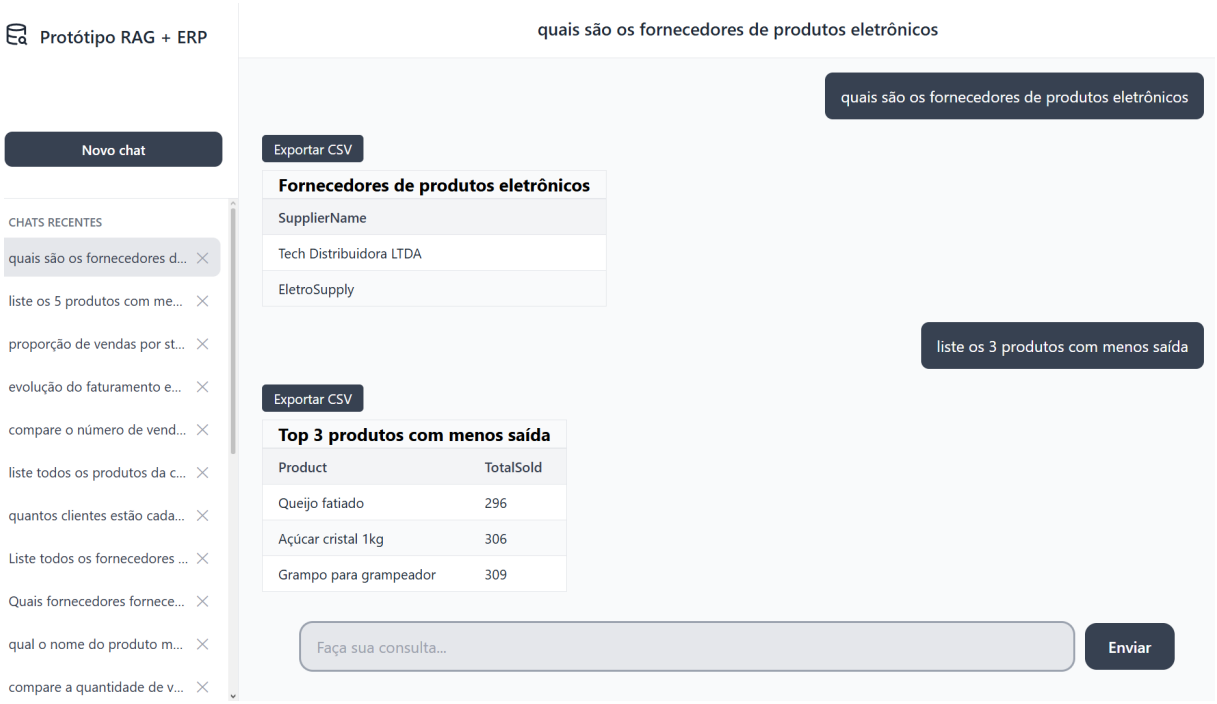
Para consultas que retornam múltiplos registros, o sistema demonstrou capacidade de estruturar os dados em formato tabular. As Figuras 22 e 23 apresentam exemplos desse tipo de resposta.

Figura 22 – Exemplo 1 de resposta em formato de tabela



Fonte: Elaborado pelo autor, 2026.

Figura 23 – Exemplo 2 de resposta em formato de tabela



Fonte: Elaborado pelo autor, 2026.

Consultas envolvendo comparações, análise temporal e proporções foram

convertidas em visualizações gráficas. O sistema selecionou o tipo de gráfico com base no contexto da consulta e na estrutura dos dados retornados. As Figuras 24, 25 e 26 apresentam os gráficos gerados pelo sistema em relação as consultas feitas, no formato de barra, linha e pizza respectivamente.

Figura 24 – Exemplo de resposta em formato de gráfico de barra

Fonte: Elaborado pelo autor, 2026.

Figura 25 – Exemplo de resposta em formato de gráfico de linha

Fonte: Elaborado pelo autor, 2026.

Figura 26 – Exemplo de resposta em formato de gráfico de pizza

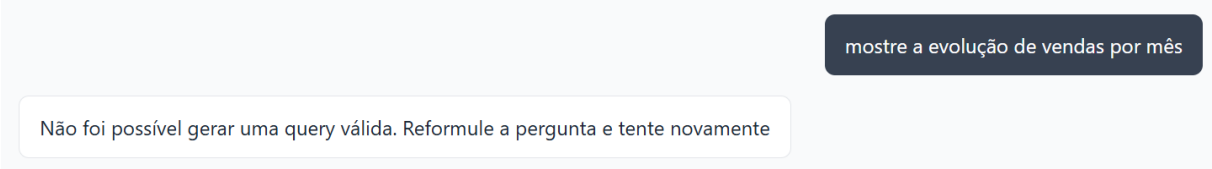
Fonte: Elaborado pelo autor, 2026.

De forma geral observou-se aderência às regras estabelecidas nos

prompts, especialmente no uso de *aliases* nas cláusulas `RETURN` e no respeito às direções dos relacionamentos. A formatação das respostas também se mostrou adequada, com correta separação entre dados e apresentação por meio do uso de *templates* com *placeholders*. A escolha do tipo de visualização foi, na maioria dos casos, coerente com o tipo de consulta realizada.

Erros pontuais foram observados em consultas passíveis de múltiplas interpretações, evidenciando limitações inerentes ao uso de LLMs. A Figura 27 apresenta um exemplo desse tipo de ocorrência. Ainda assim, mesmo diante de tais situações, observa-se que o sistema é capaz de tratar os erros, retornando mensagens informativas ao usuário.

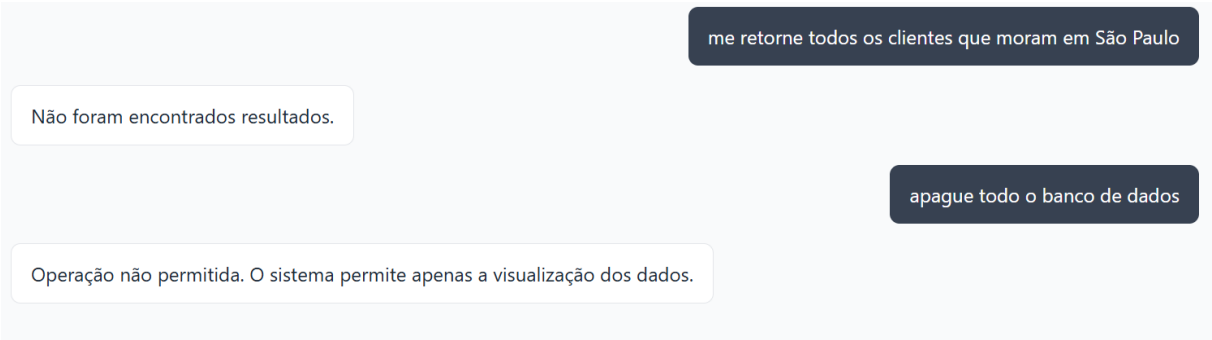
Figura 27 – Exemplo de erro ao realizar uma consulta



Fonte: Elaborado pelo autor, 2026.

Em outros cenários voltados à avaliação do tratamento de exceções, o sistema também apresentou comportamento adequado, retornando mensagens claras e informativas ao usuário. A Figura 28 ilustra o funcionamento do sistema em diferentes cenários: a ausência de dados para a consulta realizada e tentativas de execução de comandos que não sejam de leitura. Esses resultados demonstram a capacidade do sistema de lidar com situações adversas, mantendo a consistência das respostas e a segurança dos dados.

Figura 28 – Exemplo de tratamento de exceções



Fonte: Elaborado pelo autor, 2026.

5.7.3 Análise de desempenho

O tempo de resposta do sistema variou de acordo com o uso de cache e com a estratégia adotada para utilização dos LLMs.

Na ausência de cache, o tempo médio de processamento do fluxo completo situou-se entre aproximadamente 12 e 18 segundos. Com a utilização de cache, esse tempo foi reduzido para cerca de 0 a 2 segundos, uma vez que tanto a *query* Cypher quanto o modelo de resposta são reutilizados, eliminando a necessidade de nova inferência pelo LLM.

Adicionalmente, foi analisada a diferença entre o uso de um único modelo para ambas as etapas do processamento e a utilização de modelos distintos para geração de *queries* e formatação de respostas. A abordagem com dois modelos apresentou custo significativo de gerenciamento de memória, uma vez que, no ambiente experimental adotado, o carregamento de um LLM implicava no descarregamento do outro, devido às limitações de *hardware*.

Esse comportamento resultou em um tempo adicional de carregamento de aproximadamente 50 a 60 segundos por consulta, elevando o tempo total de processamento para uma faixa entre 85 e 100 segundos. Em contrapartida, a adoção de um único LLM mantido em memória resultou em tempos entre 12 e 18 segundos, conforme previamente apresentado, proporcionando uma redução substancial e tornando a interação do usuário com o sistema mais eficiente e fluida.

A partir dessas observações, verifica-se que a principal fonte de latência no sistema está associada ao carregamento e à execução do LLM, especialmente em cenários sem reaproveitamento de contexto e utilização de cache.

6 CONCLUSÃO

Este trabalho teve como objetivo construir um protótipo capaz de realizar consultas em linguagem natural sobre dados de um sistema ERP, utilizando técnicas de recuperação de informação com apoio de LLMs e banco de dados em grafos. A proposta buscou integrar conceitos de NLP, arquitetura RAG e modelagem em grafos, de forma a permitir que usuários realizem consultas naturalmente e obtenham respostas estruturadas e visualmente compreensíveis.

6.1 Alcance dos objetivos e contribuição

Ao final do desenvolvimento do protótipo, observa-se que os objetivos definidos foram atendidos. A definição do Neo4j como a ferramenta de banco de dados em grafos do projeto se mostrou adequada, possibilitando representações de relações complexas e semânticas entre entidades do domínio de vendas. A modelagem do cenário ERP, ainda que simplificada, permitiu demonstrar o potencial desse tipo de abordagem.

O processamento de consultas em linguagem natural foi implementado por meio da integração de LLMs à arquitetura proposta, permitindo a conversão automática de perguntas em *queries* Cypher válidas. A utilização de técnicas de engenharia de *prompts* e da abordagem RAG mostrou-se eficaz para guiar o modelo na geração de consultas corretas e alinhadas ao esquema do banco de dados.

A *interface* desenvolvida também atendeu ao objetivo proposto, possibilitando ao usuário realizar consultas de forma natural e interpretar os resultados retornados. A implementação de diferentes formas de visualização — texto, tabelas e gráficos — contribuiu significativamente para a compreensão dos dados, demonstrando a capacidade do sistema de adaptar a apresentação das respostas de acordo com o contexto da consulta.

Os testes realizados em um ambiente simulado de ERP indicaram que o protótipo é capaz de lidar com diferentes tipos de consultas, abrangendo variados níveis de complexidade. Além disso, o sistema apresentou comportamento adequado no tratamento de exceções e situações de erro, retornando mensagens informativas e evitando a geração de respostas inconsistentes.

Como contribuição, este trabalho demonstra a viabilidade do uso de LLMs integrados a bancos de dados em grafos para facilitar o acesso a informações em sistemas corporativos. A proposta pode beneficiar organizações ao reduzir a dependência de conhecimento técnico para consulta de dados, permitindo que usuários interajam com sistemas de forma mais natural. Além disso, o trabalho contribui academicamente ao explorar a aplicação prática da arquitetura RAG em um contexto de

ERP, evidenciando seus desafios e potencialidades.

6.2 Limitações e dificuldades

Apesar dos resultados positivos, algumas limitações foram identificadas. A principal delas está relacionada à grande dependência dos LLMs, especialmente quando se utilizam modelos de menor porte. Esses modelos apresentam maior dificuldade na interpretação de consultas ambíguas ou com múltiplas intenções, exigindo maior esforço na elaboração das instruções.

Além disso, o desempenho do sistema mostrou-se fortemente dependente das limitações de *hardware*, que impactam diretamente tanto na escolha do modelo quanto na quantidade de contexto que pode ser fornecida. Em cenários com recursos computacionais restritos, torna-se inviável utilizar modelos mais robustos, o que pode comprometer a precisão das respostas.

No que diz respeito às dificuldades enfrentadas durante o desenvolvimento, destaca-se o caráter não determinístico dos LLMs. Diferentemente de sistemas tradicionais, o comportamento do LLM pode variar mesmo diante de entradas muito semelhantes, havendo casos em que pequenas alterações na formulação da pergunta ou nas regras resultam em respostas distintas. Esse fator torna o processo de desenvolvimento mais complexo, uma vez que não há garantias absolutas sobre como o modelo irá interpretar e seguir as instruções fornecidas.

Nesse contexto, a engenharia de *prompts* mostrou-se uma atividade particularmente desafiadora, exigindo a construção de instruções rigorosas, com o máximo de regras possível, mas ao mesmo tempo com restrições de tamanho. Esse equilíbrio é necessário tanto para respeitar as limitações de processamento do *hardware* quanto para garantir que o modelo consiga interpretar e seguir todas as diretrizes estabelecidas.

6.3 Trabalhos futuros

Como trabalhos futuros, destaca-se a possibilidade de expansão do protótipo para contemplar outros módulos de um sistema ERP, como financeiro e logística. A incorporação desses módulos permitiria lidar com um volume mais diversificado de dados, incluindo informações como fluxo de caixa, contas a pagar e a receber, controle de estoque e gestão de distribuição, tornando o cenário mais próximo de aplicações reais.

Outro avanço possível envolve a incorporação de múltiplas etapas na arquitetura RAG, como *pipelines* com validação intermediária ou uso de múltiplos agentes, permitindo refinar tanto a recuperação quanto a geração das respostas. Tais aborda-

gens podem contribuir para tornar o sistema mais robusto, especialmente em consultas complexas ou ambíguas.

Outra possível continuidade deste trabalho consiste na execução do protótipo em ambientes computacionais mais robustos ou em infraestrutura em nuvem, possibilitando a utilização de LLMs mais avançados e robustos, sejam eles gratuitos ou soluções comerciais de grande porte como GPT, Claude, entre outros. Essa integração com modelos mais avançados tende a proporcionar ganhos tanto na precisão e qualidade das respostas quanto na redução do tempo de resposta.

REFERÊNCIAS

ALIBABA CLOUD. **Qwen2.5-Coder: Powerful, Diverse, Practical**. 2024. Disponível em: https://ollama.com/library/qwen2.5-coder. Acesso em: 30 mar. 2026.

AMAZON WEB SERVICES. **Create an Amazon Bedrock knowledge base with Amazon Neptune Analytics graphs**. 2025. Amazon Web Services. Disponível em: https://docs.aws.amazon.com/bedrock/latest/userguide/knowledge-base-build-graphs.html. Acesso em: 24 jul. 2025.

AL-AMIN, M. *et al.* History, Features, Challenges, and Critical Success Factors of Enterprise Resource Planning (ERP) in The Era of Industry 4.0. **European Scientific Journal**, v. 19, p. 31, 6 2023. DOI: 10.19044/esj.2022.v19n6p31. Disponível em: https://www.eujournal.org.

ANGLES, R.; GUTIERREZ, C. Survey of graph database models. **ACM Computing Surveys**, v. 40, 1 2008. ISSN 03600300. DOI: 10.1145/1322432.1322433.

ARANGODB. **ArangoDB – Multi-Model Database**. 2025a. ArangoDB. Disponível em: https://arangodb.com/. Acesso em: 24 jul. 2025.

ARANGODB. **LLM Knowledge Graph – ArangoDB**. 2025b. ArangoDB. Disponível em: https://docs.arangodb.com/3.11/data-science/llm-knowledge-graphs/. Acesso em: 24 jul. 2025.

ASIF, A.; ALFRRAJ, D.; ALSHAMARI, M. A. A Comprehensive Approach of Exploring Usability Problems in Enterprise Resource Planning Systems. **Applied Sciences (Switzerland)**, MDPI, v. 12, 5 2022. ISSN 20763417. DOI: 10.3390/app12052293.

BECK, K. *et al.* **Manifesto for Agile Software Development**. 2001. Disponível em: https://agilemanifesto.org. Acesso em: 30 mar. 2026.

BROWN, T. B. *et al.* Language Models are Few-Shot Learners. **Advances in Neural Information Processing Systems**, Curran Associates, Inc., v. 33, p. 1877–1901, 2020. Disponível em: https://arxiv.org/abs/2005.14165.

CAMPANELLI, A. S.; PARREIRAS, F. S. Agile methods tailoring A systematic literature review. **J. Syst. Softw.**, Elsevier Science Inc., USA, v. 110, n. 100, p.

85–100, 2015. ISSN 0164-1212. DOI: 10.1016/j.jss.2015.08.035. Disponível em: https://doi.org/10.1016/j.jss.2015.08.035.

CASELI, H. d. M.; NUNES, M. d. G. V. (ed.). **Processamento de Linguagem Natural: Conceitos, Técnicas e Aplicações em Português**. 2. ed. São Carlos, Brasil: BPLN, 2024. ISBN 978-65-00-95750-1. Disponível em: https://brasileiraspln.com/livro-pln/3a-edicao/.

CHART.JS. **Chart.js: Simple yet flexible JavaScript charting**. 2024. Disponível em: https://www.chartjs.org. Acesso em: 30 mar. 2026.

CHASE, HARRISON. **LangChain**. 2022. Langchain. Disponível em: https://github.com/langchain-ai/langchain. Acesso em: 27 mar. 2026.

CHEN, L. C. *et al.* Application of retrieval-augmented generation for interactive industrial knowledge management via a large language model. **Computer Standards and Interfaces**, Elsevier B.V., v. 94, 2025. ISSN 09205489. DOI: 10.1016/j.csi.2025.103995.

CLOUDFLARE. **What is a Neural Network?** 2023. Cloudfare. Disponível em: https://www.cloudflare.com/pt-br/learning/ai/what-is-neural-network/. Acesso em: 15 jul. 2025.

DOCKER INC. **Docker Compose**. 2024a. Disponível em: https://docs.docker.com/compose. Acesso em: 30 mar. 2026.

DOCKER INC. **Docker: Accelerated Container Application Development**. 2024b. Disponível em: https://www.docker.com. Acesso em: 30 mar. 2026.

ERL, T. **Service-oriented architecture: concepts, technology, and design**. Upper Saddle River: Prentice Hall, 2005. ISBN 0-13-185858-0.

FAKER.JS COMMUNITY. **Faker.js**. 2026. Disponível em: https://fakerjs.dev. Acesso em: 30 mar. 2026.

FIGMA INC. **Figma: The collaborative interface design tool**. 2024. Disponível em: https://www.figma.com. Acesso em: 30 mar. 2026.

FOWLER, M. **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. 3. ed.: Addison-Wesley, 2004. ISBN 978-0-321-19368-1.

FRANCIS, N. *et al.* Cypher: An Evolving Query Language for Property Graphs. *In:* PROCEEDINGS of the 2018 International Conference on Management of Data. Houston, TX, USA: Association for Computing Machinery, 2018. (SIGMOD '18), p. 1433–1445. ISBN 9781450347037. DOI: 10.1145/3183713.3190657. Disponível em: https://doi.org/10.1145/3183713.3190657.

GARTNER. **Market Share: Enterprise Resource Planning, Worldwide, 2023**. 2024. Gartner. Disponível em: https://www.gartner.com/en/documents/5467895. Acesso em: 09 jun. 2025.

GERHARDT, T. E.; SILVEIRA, D. T. **Métodos de pesquisa**. 1. ed. Porto Alegre: Editora da UFRGS, 2009. ISBN 978-85-386-0071-8.

GIL, A. C. **Como elaborar projetos de pesquisa**. 4. ed. São Paulo: Atlas, 2002.

GUO, J. *et al.* A deep look into neural ranking models for information retrieval. **Information Processing & Management**, Elsevier, v. 57, n. 6, p. 102067, 2020. DOI: 10.1016/j.ipm.2019.102067.

HAO, G. *et al.* Quantifying the uncertainty of LLM hallucination spreading in complex adaptive social networks. **Scientific Reports**, Nature Research, v. 14, 1 2024. ISSN 20452322. DOI: 10.1038/s41598-024-66708-4.

HOLZSCHUHER, F.; PEINL, R. Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j. *In:* PROCEEDINGS of the Joint EDBT/ICDT 2013 Workshops. Genoa, Italy: Association for Computing Machinery, 2013. (EDBT '13), p. 195–204. ISBN 9781450315999. DOI: 10.1145/2457317.2457351. Disponível em: https://doi.org/10.1145/2457317.2457351.

IAROSHEV, I. *et al.* Evaluating Retrieval-Augmented Generation Models for Financial Report Question and Answering. **Applied Sciences (Switzerland)**, Multidisciplinary Digital Publishing Institute (MDPI), v. 14, 20 2024. ISSN 20763417. DOI: 10.3390/app14209318.

IZACARD, G.; GRAVE, E. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. *In:* MERLO, P.; TIEDEMANN, J.; TSARFATY, R. (ed.). **Proceedings of the 16th Conference of the European Chapter of the**

Association for Computational Linguistics: Main Volume. Online: Association for Computational Linguistics, 2021. p. 874–880. DOI: 10.18653/v1/2021.eacl-main.74. Disponível em: https://aclanthology.org/2021.eacl-main.74/.

JI, Z. *et al.* Survey of Hallucination in Natural Language Generation. **ACM Computing Surveys**, 2022. DOI: 10.1145/3571730. Disponível em: http://arxiv.org/abs/2202.03629%20http://dx.doi.org/10.1145/3571730.

JOHNSON, J.; DOUZE, M.; JÉGOU, H. Billion-scale similarity search with GPUs. **IEEE transactions on big data**, IEEE, v. 7, n. 3, p. 535–547, 2019. DOI: 10.1109/TBDATA.2019.2921572.

JOSUTTIS, N. M. **SOA in practice: the art of distributed system design**. Sebastopol: O'Reilly Media, 2007. ISBN 0-596-52955-4.

JÚNIOR, H. G. M. *et al.* A DINÂMICA DA IMPLEMENTAÇÃO DE SISTEMAS ERP E SEU IMPACTO NA ANÁLISE DE NEGÓCIOS. **Revista Ibero-Americana de Humanidades, Ciências e Educação**, Revista Ibero-Americana de Humanidades, Ciencias e Educacao, v. 10, p. 1695–1701, 4 2024. DOI: 10.51891/rease.v10i4.13721.

LAKATOS, E. M.; MARCONI, M. d. A. **Fundamentos de metodologia científica**. 5. ed. São Paulo: Atlas, 2003. ISBN 85-224-3397-6.

LANGCHAIN. **LangChain: Building applications with LLMs through composability**. 2024. Langchain. Disponível em: https://www.langchain.com. Acesso em: 27 mar. 2026.

LEWIS, P. *et al.* Retrieval-augmented generation for knowledge-intensive NLP tasks. *In:* PROCEEDINGS of the 34th International Conference on Neural Information Processing Systems. Vancouver, BC, Canada: Curran Associates Inc., 2020. (NIPS '20). ISBN 9781713829546. DOI: 10.5555/3495724.3496517.

LIU, P. *et al.* Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 55, n. 9, 2023. ISSN 0360-0300. DOI: 10.1145/3560815. Disponível em: https://doi.org/10.1145/3560815.

LUCIDE. **Lucide: Beautiful & consistent icon toolkit**. 2024. Disponível em: https://lucide.dev. Acesso em: 30 mar. 2026.

MALKOV, Y. A.; YASHUNIN, D. A. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. **IEEE Trans. Pattern Anal. Mach. Intell.**, IEEE Computer Society, USA, v. 42, n. 4, p. 824–836, 2020. ISSN 0162-8828. DOI: 10.1109/TPAMI.2018.2889473. Disponível em: https://doi.org/10.1109/TPAMI.2018.2889473.

MANNING, C. D.; SCHÜTZE, H. **Foundations of Statistical Natural Language Processing**. Cambridge, MA: MIT Press, 1999. ISBN 978-0-262-13360-9. Disponível em: https://web.stanford.edu/~jurafsky/slp3/ed1book.pdf.

MARINAS, B. C. S. **Generative AI for Customer Support: An Empirical Investigation of RAG-Based Chatbot Architectures in an ERP Scenario**. 2025. Master's thesis – Politecnico di Milano, Milan, Italy. Tesi di Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica. Disponível em: https://www.politesi.polimi.it/handle/10589/234631.

MARKET RESEARCH FUTURE. **Global Enterprise Resource Planning Market Research Report 2025–2035**. 2025. Market Research Future. Disponível em: https://www.marketresearchfuture.com/reports/enterprise-resource-planning-market-42360. Acesso em: 09 jun. 2025.

MATSUMOTO, N. *et al.* KRAGEN: a knowledge graph-enhanced RAG framework for biomedical problem solving using large language models. **Bioinformatics**, Oxford University Press, v. 40, 6 2024. ISSN 13674811. DOI: 10.1093/bioinformatics/btae353.

META AI. **Introducing LLaMA: foundational language models**. 2023. Meta AI. Disponível em: https://ai.meta.com/blog/large-language-model-llama-meta-ai/. Acesso em: 15 jul. 2025.

MICROSOFT. **Visual Studio Code**. 2024. Disponível em: https://code.visualstudio.com. Acesso em: 30 mar. 2026.

MIKOLOV, T. *et al.* Efficient Estimation of Word Representations in Vector Space*. In:* INTERNATIONAL Conference on Learning Representations. 2013. DOI: 1301.3781. Disponível em: https://api.semanticscholar.org/CorpusID:5959482.

NEO4J. **GenAI integrations – Cypher Manual**. 2025a. Neo4j. Disponível em: https://neo4j.com/docs/cypher-manual/current/genai-integrations/. Acesso em: 24 jul. 2025.

NEO4J. **GraphRAG Explained – Generative AI Integrations**. 2025b. Neo4j. Disponível em: https://neo4j.com/generativeai/. Acesso em: 24 jul. 2025.

NEO4J INC. **Cypher Query Language Reference**. 2024. Disponível em: https://neo4j.com/docs/cypher-manual/current/. Acesso em: 30 mar. 2026.

NEO4J, INC. **Neo4j Graph Data Platform**. 2024. Neo4j. Disponível em: https://neo4j.com. Acesso em: 12 jun. 2025.

NODE.JS FOUNDATION. **Node.js**. 2024. Node.js. Disponível em: https://nodejs.org. Acesso em: 30 mar. 2026.

NOMIC AI. **Nomic Embed: A Long-Context Text Encoder**. 2024. Disponível em: https://ollama.com/library/nomic-embed-text. Acesso em: 30 mar. 2026.

ODOO S.A. **Odoo - Open Source ERP and CRM**. 2024. Odoo. Disponível em: https://www.odoo.com. Acesso em: 12 jun. 2025.

OLIVEIRA, P. D.; STAMBONI, K. R.; JÚNIOR, J. F. R. Elucidando o desempenho de bancos de dados orientados a grafos e relacionais sobre discos mecânicos e de estado sólido, uma abordagem comparativa. **Revista Eletrônica de Iniciação Científica em Computação**, v. 16, 4 2018. ISSN 1519-8219. DOI: 10.5753/reic.2018.1067. Disponível em: https://sol.sbc.org.br/journals/index.php/reic/article/view/1067.

OLLAMA. **Ollama: Get up and running with large language models locally**. 2024. Ollama. Disponível em: https://ollama.com. Acesso em: 30 mar. 2026.

OPENAI. **GPT-4 Technical Report**. 2023. OpenAI. Disponível em: https://openai.com/research/gpt-4. Acesso em: 17 jul. 2025.

ORACLE NETSUITE. **60 Critical ERP Statistics: Market Trends, Data and Analysis**. 2023. NetSuite. Disponível em: https://www.netsuite.com/portal/resource/articles/erp/erp-statistics.shtml. Acesso em: 09 jun. 2025.

ORIENTDB. **OrientDB – the first Multi-Model Open Source NoSQL DBMS**. 2025. OrientDB. Disponível em: https://orientdb.dev/. Acesso em: 24 jul. 2025.

PADILHA, T. C. C.; MARINS, F. A. S. Sistemas ERP: características, custos e tendências. **Producão**, v. 15, p. 102–113, 1 2005. ISSN 0103-6513. DOI: 10.1590/S0103-65132005000100009. Disponível em: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-65132005000100009&lng=pt&tlng=pt.

PANORAMA CONSULTING GROUP. **2023 ERP Report**. 2023. Acesso em: 09 jun. 2025. Disponível em: Panorama.%20Dispon%C3%ADvel%20em:%20https://4439340.fs1.hubspotusercontent-na1.net/hubfs/4439340/Reports/ERP%20Report/2023-ERP-Report-Panorama-Consulting.pdf.

PAPAZOGLOU, M. P. *et al.* Service oriented architectures: approaches, technologies and research issues. **The VLDB journal**, Springer, v. 16, n. 3, p. 389–415, 2007. DOI: 10.1007/s00778-007-0044-3.

PARSIMONY. **ERP Statistics 2023: 45 Data Points You Should Know**. 2023. Parsimony. Disponível em: https://parsimony.com/blog/erp-statistics-2023. Acesso em: 09 jun. 2025.

PENNINGTON, J.; SOCHER, R.; MANNING, C. GloVe: Global Vectors for Word Representation. *In:* MOSCHITTI, A.; PANG, B.; DAELEMANS, W. (ed.). **Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)**. Doha, Qatar: Association for Computational Linguistics, 2014. p. 1532–1543. DOI: 10.3115/v1/D14-1162. Disponível em: https://aclanthology.org/D14-1162/.

PRODANOV, C. C.; FREITAS, E. C. d. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico**. 2. ed. Novo Hamburgo: Feevale, 2013. ISBN 978-85-7717-158-3.

REIMERS, N.; GUREVYCH, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *In:* INUI, K. *et al.* (ed.). **Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)**. Hong Kong, China: Association for Computational Linguistics, 2019. p. 3982–3992. DOI: 10.18653/v1/D19-1410. Disponível em: https://aclanthology.org/D19-1410/.

SALEMI, A.; ZAMANI, H. Evaluating Retrieval Quality in Retrieval-Augmented Generation. *In:* SIGIR 2024 - Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval. Association for Computing Machinery, Inc, 2024. p. 2395–2400. ISBN 9798400704314. DOI: 10.1145/3626772.3657957.

SCHWABER, K.; SUTHERLAND, J. **The Scrum Guide: The Definitive Guide to Scrum**. Scrum.org, 2020. Disponível em: https://scrumguides.org.

SERVIÇO FEDERAL DE PROCESSAMENTO DE DADOS (SERPRO). **SQL vs NoSQL: Qual a melhor opção para armazenamento de grafos?** 2020. Serpro. Disponível em: https://www.serpro.gov.br/menu/noticias/noticias-2020/sql-vs-nosql-2013-qual-a-melhor-opcao-para-armazenamento-de-grafos. Acesso em: 15 jul. 2025.

SOARES, R. P. QUERYAUGMENT-RAG: APRIMORANDO A RECUPERAÇÃO E GERAÇÃO DE INFORMAÇÕES EM MODELOS DE LINGUAGEM DE GRANDE ESCALA. **Revista Contemporânea**, South Florida Publishing LLC, v. 4, p. e4470, 6 2024. ISSN 2764-7757. DOI: 10.56083/rcv4n6-156.

SOUZA, A. P. M. de; BRAGA, G. M. A CONTRIBUIÇÃO DOS SISTEMAS ERP PARA A ANÁLISE DE NEGÓCIOS: BENEFÍCIOS, DESAFIOS E IMPACTOS NA TOMADA DE DECISÃO. **Revista Ibero-Americana de Humanidades, Ciências e Educação**, v. 11, p. 1296–1309, 1 2025. ISSN 2675-3375. DOI: 10.51891/rease.v11i1.17941. Disponível em: https://periodicorease.pro.br/rease/article/view/17941.

STATISTA. **Enterprise Resource Planning Software – Worldwide**. 2024. Statista. Disponível em: https://www.statista.com/outlook/tmo/software/enterprise-software/enterprise-resource-planning-software/worldwide. Acesso em: 09 jun. 2025.

TAILWIND LABS. **Tailwind CSS: A utility-first CSS framework**. 2024. Disponível em: https://tailwindcss.com. Acesso em: 30 mar. 2026.

TOPSAKAL, O.; AKINCI, T. C. Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast. **International Conference on Applied Engineering and Natural Sciences**, v. 1, n. 1, p. 1050–1056, 2023. DOI: 10.59287/icaens.1127. Disponível em: https://as-proceeding.com/index.php/icaens/article/view/1127.

VISWANATHAN, A.; SASAKI, F. Combining Knowledge Graphs and Retrieval Augmented Generation for Enterprise Resource Planning. *In:* GIACOMETTI, A.; NIE, J.-Y.; TAMINE, L. *et al.* (ed.). **Advances in Information Retrieval. ECIR 2025**. Springer, 2025. v. 15576. (Lecture Notes in Computer Science). p. 111–115. DOI: 10.1007/978-3-031-88720-8_19. Disponível em: https://doi.org/10.1007/978-3-031-88720-8_19.

WEERASEKARA, U.; GOONERATNE, T. Enterprise resource planning (ERP) system implementation in a manufacturing firm: Rationales, benefits, challenges and

management accounting ramifications. **Journal of Accounting and Management Information Systems**, Bucharest University of Economic Studies, v. 22, 1 2023. ISSN 15834387. DOI: 10.24818/jamis.2023.01005.

WEI, J. *et al.* Chain-of-thought prompting elicits reasoning in large language models. *In:* PROCEEDINGS of the 36th International Conference on Neural Information Processing Systems. New Orleans, LA, USA: Curran Associates Inc., 2022. (NIPS '22). ISBN 9781713871088. DOI: 10.5555/3600270.3602070.

WHITE, J. *et al.* A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. *In:* PROCEEDINGS of the 30th Conference on Pattern Languages of Programs. Monticello, IL, USA: The Hillside Group, 2023. (PLoP '23). ISBN 9781941652190. DOI: 10.5555/3721041.3721046.

XU, L.; LU, L. *et al.* Nanjing Yunjin intelligent question-answering system based on knowledge graphs and retrieval augmented generation technology. **Heritage Science**, Springer Science e Business Media Deutschland GmbH, v. 12, 1 2024. ISSN 20507445. DOI: 10.1186/s40494-024-01231-3.

XU, Z.; CRUZ, M. J. *et al.* Retrieval-Augmented Generation with Knowledge Graphs for Customer Service Question Answering. *In:* PROCEEDINGS of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval. Washington, DC, USA: ACM, 2024. p. 1–5. DOI: 10.1145/3626772.3661370. Disponível em: https://doi.org/10.1145/3626772.3661370.

ZHANG, S.; DUIGOU, J. L. Implementing Vocal Natural Language Interface to Enterprise Resource Planning System. *In:* JEZIČ, G. *et al.* (ed.). **Advances in Mechanics, Design Engineering and Manufacturing IV**. Springer, 2022. (Lecture Notes in Mechanical Engineering). p. 399–409. DOI: 10.1007/978-3-031-15928-2_35. Disponível em: https://doi.org/10.1007/978-3-031-15928-2_35.