INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Uso de Retrieval-Augmented Generation para Auxílio em Atividades de Suporte Técnico

RUI PEDRO TELES RIBEIRO
Setembro de 2025

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Uso de Retrieval-Augmented Generation para Auxílio em Atividades de Suporte Técnico

Rui Pedro Teles Ribeiro

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Piedade Carvalho
Supervisor: José Soares

Porto, 28 de setembro de 2025

Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, Porto, 28 de setembro de 2025

Resumo

Esta dissertação investiga o uso de Retrieval-Augmented Generation (RAG) para apoiar atividades de suporte técnico em contexto empresarial, onde a rotatividade de elementos, o volume de incidentes e a dispersão do conhecimento dificultam o acesso rápido à informação. Seguindo a metodologia Design Science Research (DSR), foi concebido e implementado um protótipo composto por uma SPA em React e uma API em Spring Boot/Spring AI, suportada por uma base de dados vetorial Qdrant e uma base de dados relacional Microsoft SQL Server. A solução integra-se com o Confluence através de *webhooks*, para sincronização em tempo real (indexação, reindexação e exclusão), e inclui um processo periódico de extração de dados históricos do Control-M, desenvolvido em Java, publicando relatórios estruturados no Confluence para posterior recuperação.

A avaliação adotou o modelo Goal Question Metric (GQM) e métricas LLM-based para medir correção, relevância e consistência das respostas em cenários de documentação geral e de dados do Control-M. Os resultados evidenciam: (i) melhor relação desempenho-custo em modelos de *chat* de menor dimensão (*gpt-4.1-mini* e *gpt-4o-mini*); (ii) superioridade do modelos de *embedding* de grande dimensão (*text-embedding-3-large*); (iii) impacto negativo de *similarity thresholds* acima de 40% e ganhos com *top-k* mais elevados; (iv) melhoria na recuperação de dados do Control-M com reescrita de *queries*, embora aquém da meta de precisão de 90% definida. Complementarmente, o código apresentou boa qualidade técnica e o *feedback* dos utilizadores foi maioritariamente favorável. Conclui-se que o protótipo é eficaz para documentação geral e tecnicamente viável, sugerindo como trabalhos futuros a exploração da recuperação híbrida, de estruturas alternativas de armazenamento do conhecimento e a realização de testes operacionais.

Palavras-chave: IA, RAG, Confluence, Control-M, Suporte Técnico, Bases de Dados Vetoriais

Abstract

This dissertation investigates the use of Retrieval-Augmented Generation (RAG) to optimize technical support activities in a business context, where staff turnover, the volume of incidents, and the dispersion of knowledge make it difficult to access information quickly. Following the Design Science Research (DSR) methodology, a prototype was designed and implemented consisting of a SPA in React and an API in Spring Boot/Spring AI, supported by a Qdrant vector database and a Microsoft SQL Server relational database. The solution integrates with Confluence through webhooks, for real-time synchronization (indexing, reindexing, and deletion), and includes a periodic process of extracting historical data from Control-M, developed in Java, publishing structured reports in Confluence for later retrieval.

The evaluation adopted the Goal Question Metric (GQM) model and LLM-based metrics to measure the accuracy, relevance, and consistency of responses in general documentation and Control-M data scenarios. The results show: (i) better cost-performance ratio in smaller chat models (gpt-4.1-mini and gpt-4o-mini); (ii) superiority of large embedding models (text-embedding-3-large); (iii) negative impact of similarity thresholds above 40% and gains with higher top-k; (iv) improvement in Control-M data retrieval with query rewriting, although falling short of the defined accuracy target of 90%. In addition, the code showed of good technical quality and user feedback was mostly favorable. It is concluded that the prototype is effective for general documentation and technically feasible, suggesting as future work the exploration of hybrid retrieval, alternative knowledge storage structures, and operational testing.

Keywords: AI, RAG, Confluence, Control-M, Technical Support, Vector Databases

Agradecimentos

Agradeço ao ISEP pelo conhecimento e valores transmitidos ao longo destes 6 anos.

Agradeço à minha orientadora, Piedade Carvalho, pela ajuda contínua prestada, pelo rápido auxílio mesmo em horários apertados e por ter confiado no meu trabalho.

Agradeço ao meu supervisor, José Soares, e a toda a equipa da Purple por me terem apoidado e permitido a realização deste projeto.

Agradeço à minha mãe, pai, irmã e avô por todo o apoio e amor incondicional.

Agradeço à Érica pela paciência, dedicação, companheirismo, por ter sido a minha orientadora "não oficial"e por ter sofrido comigo ao longo deste projeto. Nunca irei esquecer o que fez por mim.

Índice

Lista de Figuras **xv**

Lista de Tabelas **xvii**

Lista de Códigos **xix**

Abreviações **xxi**

1 Introdução **1**
1.1 Organização e Contexto . . . 1
1.2 Problema . . . 1
1.3 Objetivos . . . 2
1.4 Metodologia . . . 3
1.5 Planeamento . . . 4
1.6 Considerações Éticas . . . 4
1.7 Estrutura do Documento . . . 5

2 Fundamentos Teóricos **7**
2.1 Inteligência Artificial . . . 7
2.2 Machine Learning e Deep Learning . . . 7
2.3 Large Language Models . . . 8
2.4 Fine-tuning de LLMs . . . 8
2.5 *Framework* RAG . . . 9
2.6 RAG vs. Fine-tuning . . . 10

3 Estado da Arte **13**
3.1 Revisão de Literatura . . . 13
3.1.1 Metodologia de Investigação . . . 13
3.1.2 Resultados e Análise . . . 16
3.2 Tecnologias Frontend . . . 28
3.2.1 React . . . 28
3.2.2 Angular . . . 29
3.2.3 Tabela de comparação . . . 29
3.3 Tecnologias Backend . . . 30
3.3.1 Java . . . 31
3.3.2 Python . . . 31
3.3.3 Tabela de comparação . . . 31
3.4 Frameworks RAG . . . 32
3.4.1 LangChain . . . 32
3.4.2 Haystack . . . 33
3.4.3 Spring AI . . . 34

3.4.4 Tabela de comparação . . . 35
3.5 Sistemas Externos . . . 37
3.5.1 Confluence . . . 37
3.5.2 Control-M . . . 37
3.6 Métricas de Avaliação . . . 38
3.6.1 ROUGE . . . 38
3.6.2 BERTScore . . . 39
3.6.3 LLM-based . . . 39
3.7 Trabalhos Relacionados . . . 39
3.7.1 RAG Chatbot para a OptiMicro Technologies . . . 39
3.7.2 Sistema RAG de Recomendação para Suporte Técnico . . . 40
3.7.3 Conclusões . . . 41

4 Análise **43**
4.1 Solução . . . 43
4.2 Modelo de Domínio . . . 44
4.3 Engenharia de Requisitos . . . 44
4.3.1 Requisitos Funcionais (RF) . . . 45
4.3.2 Requisitos Não Funcionais (RNF) . . . 49
4.3.3 Casos de Uso . . . 50

5 Desenho **55**
5.1 Modelo Arquitetural . . . 55
5.2 Escolhas Tecnológicas . . . 56
5.3 Contexto C4 . . . 56
5.4 Containers C4 . . . 58
5.5 Componentes C4 . . . 58
5.5.1 Sistema RAG - SPA . . . 59
5.5.2 Sistema RAG - API . . . 59
5.5.3 B2C Core - Report . . . 63
5.6 Diagramas de Implantação . . . 63
5.7 Diagramas de Sequência . . . 65
5.7.1 Pipeline RAG . . . 65
5.7.2 Extração de dados do Control-M . . . 67

6 Implementação **69**
6.1 Estrutura . . . 69
6.1.1 Módulo SPA . . . 69
6.1.2 Módulo API . . . 71
6.1.3 Módulo Report - Extração Control-M . . . 74
6.2 Casos de Uso . . . 76
6.2.1 Pipeline RAG - Indexação/Sincronização com o Confluence (UC5) . . . 76
6.2.2 Pipeline RAG - Recuperação assistida (UC1) . . . 79
6.2.3 Extração de dados do Control-M (UC6) . . . 81
6.3 Interface Gráfica . . . 82
6.3.1 Página de Recuperação . . . 82
6.3.2 Página de Gestão de Documentos . . . 83

7 Avaliação da Solução **85**
7.1 Metodologia . . . 85

7.1.1 Abordagem de Avaliação da Qualidade do Sistema RAG 86
7.1.2 Estrutura GQM 87
7.2 Análise dos Resultados 88
7.2.1 G1 e G2 - Análise da Qualidade do Sistema RAG 88
7.2.2 G3 - Análise de *Feedback* dos Utilizadores 94
7.2.3 G4 - Análise da Qualidade do Código 96

8 Conclusão **103**
8.1 Trabalho Realizado 103
8.2 Limitações 104
8.3 Trabalho Futuro 104
8.4 Apreciação Final 105

Bibliografia **107**

Anexo A Prompts **111**

Anexo B Algoritmos **115**

Anexo C Sessão de Apresentação Interna **117**

Anexo D Modelação e Dados **119**

Lista de Figuras

1.1 Metodologia DSR (Peffers et al., 2007) . . . 3
1.2 Estrutura WBS e diagrama de Gantt . . . 4

2.1 Arquitetura simplificada do RAG moderno . . . 9
2.2 Comparação entre FT e RAG a nível de adaptação do modelo e dependência de conhecimento externo (Y. Gao et al., 2023) . . . 11

3.1 Processo de chunking (adaptado de Špeletić et al., 2024) . . . 23
3.2 Processo de recuperação e geração exclusivamente com modelo de *embedding* (Adaptado de Špeletić et al., 2024) . . . 25
3.3 Processo de recuperação híbrida . . . 26
3.4 Arquitetura do chatbot RAG da OptiMicro (H.-C. Lee et al., 2024) . . . 40
3.5 Arquitetura do sistema de recomendação de resolução de incidentes proposto por Isaza et al., 2024b . . . 41

4.1 Modelo de domínio . . . 44
4.2 Diagrama de casos de uso . . . 51

5.1 Diagrama de contexto C4 . . . 57
5.2 Diagrama de containers C4 do sistema RAG . . . 58
5.3 Diagrama de componentes C4 da SPA . . . 59
5.4 Abordagem com *polling* (Bhandari, 2024) . . . 60
5.5 Abordagem com Webhooks . . . 61
5.6 Diagrama de componentes C4 da API . . . 62
5.7 Diagrama de componentes C4 do módulo de extração de dados do Control-M 63
5.8 Alternativas de implantação do sistema RAG . . . 64
5.9 Diagrama de implantação do módulo de extração de dados do Control-M . 65
5.10 Diagrama de sequência do mecanismo de sincronização com o Confluence - UC05 . . . 66
5.11 Diagrama de sequência da recuperação assistida por IA - UC01 . . . 67
5.12 Diagrama de sequência da extração de dados do Control-M - UC06 . . . 68

6.1 Diagrama de implementação da SPA . . . 69
6.2 Diagrama de implementação da API . . . 72
6.3 Uso da abstração Vector Store . . . 73
6.4 Diagrama de implementação do Report . . . 75
6.5 Definição do horário de execução . . . 82
6.6 Interface de recuperação . . . 83
6.7 Interface de visualização e gestão de páginas . . . 84

7.1 Estrutura do modelo GQM . . . 85
7.2 Processo de avaliação . . . 86

7.3 Desempenho de modelos de *Chat* 90
7.4 Desempenho de modelos de *Embedding* 91
7.5 Desempenho por ST e TK 92
7.6 Desempenho com e sem reescrita de *query* 93
7.7 Cobertura de testes 100
7.8 Complexidade ciclomática 100
7.9 Índice de manutenibilidade 100

C.1 Evidência de sessão interna de apresentação da solução 117
C.2 Evidência de questionário retrospetivo realizado após a sessão interna . . . 118

D.1 Diagrama de classes Unified Modeling Language (UML) - Sistema RAG . . 119
D.2 Diagrama entidade-relação da base de dados relacional composto exclusivamente pela *page* 120
D.3 Qdrant - Associação de metadados a cada vetor armazenado 120

Lista de Tabelas

1.1 Objetivos 2

2.1 Comparação entre RAG e FT (Y. Gao et al., 2023) 11

3.1 Questões de investigação 14
3.2 Bases de dados de investigação 15
3.3 Critérios de inclusão e exclusão para artigos 15
3.4 Critérios de inclusão e exclusão para *websites* 16
3.5 Comparação entre tipos de bases de dados no contexto de RAG 22
3.6 Comparação entre tecnologias frontend 30
3.7 Comparação entre tecnologias backend 32
3.8 Comparação entre frameworks RAG 36

4.1 Requisitos funcionais da sincronização com o Confluence 46
4.2 Requisitos funcionais da recuperação assistida por IA 47
4.3 Requisitos funcionais do sistema de gestão do conhecimento 48
4.4 Requísitos funcionais para a integração de dados históricos do Control-M . 49
4.5 Requisitos não funcionais 50
4.6 UC01 – Realizar pesquisa 51
4.7 UC02 – Visualizar páginas 52
4.8 UC03 – Excluir página 52
4.9 UC04 – Restaurar página 53
4.10 UC05 – Sincronizar automaticamente com o Confluence 53
4.11 UC06 – Extração de dados históricos do Control-M 54

7.1 Legenda de pontuações 87
7.2 Métricas do GQM 87
7.3 Estrutura GQM 88
7.4 Legenda de critérios de configuração dos testes 89
7.5 Comparação dos modelos de *chat* em termos de pontuação média e desvio padrão. 90
7.6 Comparação dos modelos de *embedding* em termos de pontuação média e desvio padrão. 91
7.7 Comparação de resultados face à variação de ST e TK 92
7.8 Avaliação da recuperação de dados do Control-M com ou sem reescrita de *query* 94
7.9 Questionário retrospetivo da sessão 95

8.1 Grau de realização dos objetivos definidos 104

Lista de Códigos

3.1 *Query* de investigação . . . 14
6.1 Composição do Router . . . 70
6.2 Barra de navegação . . . 70
6.3 Exemplo de *Reducer* da página de recuperação . . . 71
6.4 Exemplo de saga de recuperação . . . 71
6.5 Definição do *ConfluenceApiClient* . . . 73
6.8 Configuração dos modelos de IA . . . 74
6.9 Exemplo de *endpoint* no *Controller* . . . 74
6.10 *Query* SQL para a recolha de *jobs* falhados da base de dados do Oracle do Control-M . . . 75
6.11 *Endpoint* para criação de página no Confluence - POST /wiki/rest/api/-content . . . 76
6.12 Subscrição de *webhooks* . . . 77
6.13 Exemplo de evento de edição de página . . . 77
6.14 Tratamento dos *webhooks* . . . 78
6.15 Método de remoção de vetores . . . 78
6.16 Método de indexação de vetores . . . 79
6.17 Chamada do recuperador na SPA . . . 80
6.18 Pesquisa vetorial . . . 80
6.19 Serviço de recuperação na API . . . 81
6.20 Construção de linhas da tabela dos relatórios do Control-M . . . 81
7.1 Cenário positivo - Recuperação com documentos presentes . . . 97
7.2 Cenário negativo - Recuperação de conteúdo de página do Confluence com problema de conexão . . . 97
7.3 *Beans* do Spring Simulados . . . 98
7.4 *Webhook* de criação de página . . . 99
A.1 *Prompt* RAG elaborado - *chain-of-thought prompting* . . . 111
A.2 *Prompt* de avaliação das respostas . . . 112
A.3 *Prompt* reescrita de *query* para Control-M . . . 113
B.1 Algoritmo de experimentação automatizada . . . 115

Abreviações

ACID	Atomicidade, Consistência, Isolamento e Durabilidade.
ANN	Approximate Nearest Neighbor.
API	Application Programming Interface.
BD	Base de Dados.
BERT	Bidirectional Encoder Representations from Transformers.
BM25	Best Matching 25.
C	Modelo de chat.
CI/CD	Continuous Integration / Continuous Delivery.
CLI	Command Line Interface.
CSS	Cascading Style Sheets.
DI	Dependency Injection.
DIMEI	Dissertação de Mestrado em Engenharia Informática.
DL	Deep Learning.
DOM	Document Object Model.
DSR	Design Science Research.
DTO	Data Transfer Object.
E	Modelo de Embedding.
FT	Fine-tuning.
FTS	Full-Text Search.
FURPS+	Funcionalidade, Usabilidade, Confiabilidade, Desempenho, Suportabilidade + (restrições extra).
GPT	Generative Pre-trained Transformer.
GPU	Graphics Processing Unit.
GQM	Goal Question Metric.
HNSW	Hierarchical Navigable Small World.
HTML	HyperText Markup Language.
HTTP	HyperText Transfer Protocol.
IA	Inteligência Artificial.
ID	Identificador.

IDE	Integrated Development Environment.
IEEE	Institute of Electrical and Electronics Engineers.
IPP	Instituto Politécnico do Porto.
ISEP	Instituto Superior de Engenharia do Porto.
JDK	Java Development Kit.
JPA	Java Persistence API.
JRE	Java Runtime Environment.
JSON	JavaScript Object Notation.
JSX	JavaScript XML.
JVM	Java Virtual Machine.
LCEL	LangChain Expression Language.
LLaMA	Large Language Model Meta AI.
LLM	Large Language Model.
ML	Machine Learning.
MVC	Model–View–Controller.
N	Número de perguntas.
ND	Número de documentos.
NoSQL	Not Only SQL.
PLN	Processamento de Linguagem Natural.
RAG	Retrieval-Augmented Generation.
RDF	Resource Description Framework.
REST	Representational State Transfer.
RF	Requisito Funcional.
RNF	Requisito Não Funcional.
ROUGE	Recall-Oriented Understudy for Gisting Evaluation.
RQ	Research Questions.
RxJS	Reactive Extensions for JavaScript.
SaaS	Software as a Service.
SGBD	Sistemas de Gestão de Bases de Dados.
SLA	Service Level Agreement.
SPA	Single-Page Application.
SQL	Structured Query Language.
ST	Similarity Threshold.
TK	Top-K.
UC	Caso de Uso.
UML	Unified Modeling Language.
URL	Uniform Resource Locator.

WBS	Work Breakdown Structure.
XML	Extensible Markup Language.

Capítulo 1

Introdução

O presente capítulo introduz o enquadramento do projeto, apresentando a organização, o contexto e o problema que motivou a investigação. São definidos os objetivos e a metodologia adotada para a construção da solução proposta, bem como o planeamento do trabalho e as considerações éticas que orientaram a sua execução. Por fim, é descrita a estrutura global da dissertação, de forma a guiar o leitor ao longo dos capítulos seguintes.

1.1 Organização e Contexto

A Purple é uma instituição financeira com presença internacional, atuando na banca de investimentos, gestão de ativos, seguros e serviços financeiros especializados. Em Portugal, dispõe de um centro tecnológico dedicado ao desenvolvimento e operação de soluções que suportam várias linhas de negócio. O trabalho decorreu especificamente na equipa B2C, responsável por sistemas de risco de crédito.

Recentemente, a organização adotou uma plataforma corporativa de Inteligência Artificial (IA) generativa em regime Software as a Service (SaaS), baseada em modelos Generative Pre-trained Transformer (GPT), infraestrutura da qual este projeto tirou partido para prototipar um sistema que auxilie a equipa de suporte da B2C.

Este projeto de dissertação foi desenvolvido no âmbito da unidade curricular de Dissertação de Mestrado em Engenharia Informática (DIMEI) do Instituto Superior de Engenharia do Porto (ISEP), proporcionando a aplicação prática de conhecimentos adquiridos ao longo do curso, em estreita colaboração com uma organização de referência no setor financeiro.

1.2 Problema

A B2C é uma equipa responsável por diversos projetos na área da banca, com foco principal no cálculo de risco de crédito. Semanalmente, existe uma equipa de suporte rotativa composta por dois elementos da equipa de desenvolvimento. Devido à dimensão da equipa, à extensa documentação de suporte e à quantidade de desenvolvimentos constantes entregues em produção, as atividades de suporte podem tornar-se complexas, exigindo um conhecimento global do fluxo do sistema e do histórico de operações. Estas atividades incluem o acompanhamento diário de uma cadeia de trabalhos (*jobs*) no Control-M, a prestação de respostas e a resolução de incidentes provenientes de diversos canais de comunicação.

Em média, a equipa de suporte lida semanalmente entre 20 e 30 incidentes, incluindo pedidos de suporte técnico e intervenções urgentes para corrigir problemas críticos em produção.

Estes incidentes abrangem desde dúvidas operacionais até falhas que exigem resposta imediata, o que contribui para uma elevada pressão sobre os membros da equipa e reforça a necessidade de soluções que agilizem o acesso à informação e a resolução eficiente dos problemas.

O conhecimento tático está um pouco disperso devido à rotatividade semanal da equipa de suporte. A documentação existente, embora abrangente, pode ser difícil de consultar rapidamente durante situações de alta pressão.

1.3 Objetivos

O foco deste projeto é mitigar os problemas descritos na secção anterior. Para tal, propôs-se o desenvolvimento de uma solução que integre Retrieval-Augmented Generation (RAG) para auxiliar a equipa na tomada de decisão nas atividades de suporte, através da recuperação e da geração de informação contextualizada. A aplicação do RAG neste contexto permitirá otimizar a eficiência da equipa de suporte, reduzindo o tempo gasto na procura de informação e facilitando a resolução de incidentes. Além desta solução, propôs-se o desenvolvimento de um mecanismo secundário responsável pela integração de dados históricos do Control-M no sistema RAG. Os objetivos específicos deste trabalho encontram-se mais detalhados na Tabela 1.1.

Tabela 1.1: Objetivos

Identificador (ID)	Tópico	Descrição
O1	**Sistema RAG protótipo**	O objetivo principal é o desenvolvimento de um protótipo funcional que integre uma arquitetura baseada em RAG. Este protótipo incluirá uma interface simples para interação prática. O sistema deverá ser capaz de se integrar com o espaço da equipa no Confluence e fornecer respostas contextualizadas, facilitando o acesso à documentação interna;
O2	**Control-M**	Desenvolvimento de um mecanismo periódico de extração de dados históricos do Control-M e de publicação desses dados no Confluence. Desta forma, possibilita-se a consulta centralizada destes dados pelo sistema RAG;

A eficácia do sistema RAG será avaliada através de testes experimentais, utilizando métricas de desempenho para aferir a qualidade das respostas. Complementarmente, será realizada uma avaliação qualitativa com os utilizadores de suporte e uma análise do código por meio de testes automatizados e de outras métricas destinadas a medir a complexidade e a manutenibilidade. Desta forma, será assegurada a utilidade, a confiabilidade e a sustentabilidade do sistema.

1.4 Metodologia

A metodologia Design Science Research (DSR) (Figura 1.1) foi escolhida como quadro orientador para o desenvolvimento deste trabalho. Esta metodologia é particularmente adequada para investigações que visam resolver problemas do mundo real através da criação e avaliação de artefactos (Peffers et al., 2007).

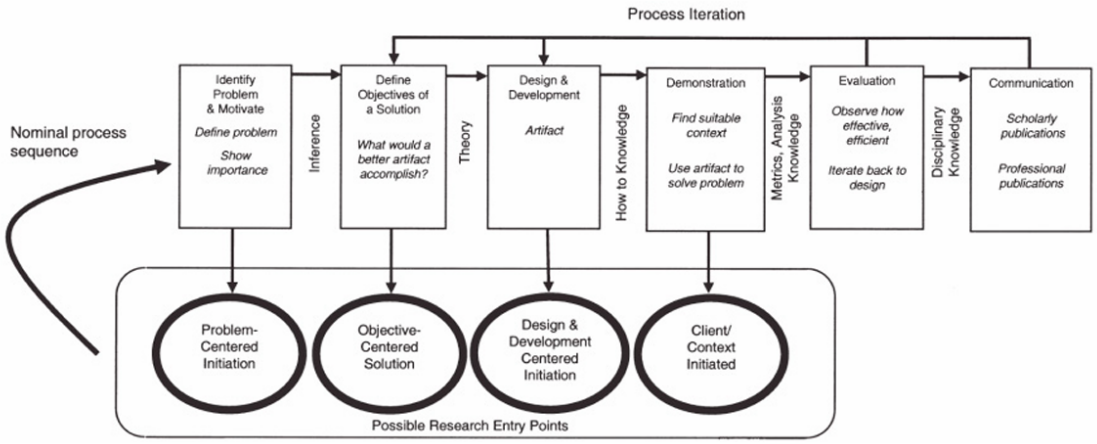


Figura 1.1: Metodologia DSR (Peffers et al., 2007)

A metodologia DSR é composta por seis etapas, que se relacionam com o presente projeto da seguinte forma:

- **Identificação e Motivação do Problema:** Foi realizada uma análise dos desafios enfrentados pela equipa de suporte técnico, nomeadamente a dificuldade em aceder rapidamente a informação precisa e a sobrecarga de pedidos;
- **Definição de Objetivos para a Solução:** Com base nos problemas identificados, foram definidos objetivos claros para o desenvolvimento de artefactos que têm o objetivo de mitigar esses desafios;
- **Desenho e Desenvolvimento:** Fase de conceção e desenvolvimento da solução proposta. Nesta etapa foram exploradas diferentes alternativas arquiteturais a fim de construir uma solução eficiente;
- **Demonstração:** A solução foi aplicada a cenários simulados e reais de suporte técnico, permitindo validar a sua aplicabilidade em tarefas como a resposta a incidentes, o esclarecimento de dúvidas técnicas e a recuperação de documentação e conhecimento relevante;
- **Avaliação:** O sistema foi avaliado com base em testes experimentais de desempenho, testes unitários e de integração;
- **Comunicação:** Os resultados desta investigação foram documentados nesta dissertação e visam contribuir tanto para o avanço do conhecimento académico na área da inteligência artificial, como para a melhoria contínua de processos em contexto empresarial.

1.5 Planeamento

O planeamento do trabalho foi estruturado através da Work Breakdown Structure (WBS) (Siami-Irdemoosa, Dindarloo e Sharifzadeh, 2015), a partir da qual se elaborou o diagrama de Gantt (Figura 1.2). A WBS organiza o projeto em sete fases, com entregáveis e tarefas associadas, e define pontos formais de revisão com a organização e com o orientador do ISEP. O diagrama de Gantt calendariza essas atividades, identifica dependências e marcos de controlo.

ID	WBS	Nome da Tarefa	Nível WBS	Duração	Início	Fim
1	**1**	**Projeto Dissertação**	**Projeto**	**134 days?**	**4/5/2025**	**10/8/2025**
2	**1.1**	**Introdução**	**Fase**	**10 days**	**4/7/2025**	**4/18/2025**
3	1.1.1	Análise do problema	Entregável	5 days	4/7/2025	4/11/2025
4	1.1.2	Definição dos objetivos	Entregável	5 days	4/14/2025	4/18/2025
5	1.1.3	Planeamento	Entregável	5 days	4/14/2025	4/18/2025
6	**1.2**	**Estado da arte**	**Fase**	**27 days**	**4/21/2025**	**5/27/2025**
7	1.2.1	Fundamentos teóricos	Entregável	5 days	4/21/2025	4/25/2025
8	1.2.2	Revisão da literatura	Entregável	20 days	4/28/2025	5/23/2025
9	1.2.3	Escolhas tecnológicas	Entregável	5 days	4/28/2025	5/2/2025
10	1.2.4	Revisão com a organização	Tarefa	1 day	5/26/2025	5/26/2025
11	1.2.5	Revisão com o orientado	Tarefa	1 day	5/27/2025	5/27/2025
12	**1.3**	**Analise da solução**	**Fase**	**10 days**	**5/28/2025**	**6/10/2025**
13	1.3.1	Modelo de domínio	Entregável	5 days	5/28/2025	6/3/2025
14	1.3.2	Requisitos funcionais	Entregável	10 days	5/28/2025	6/10/2025
15	1.3.3	Requisitos não funcionai	Entregável	10 days	5/28/2025	6/10/2025
16	1.3.4	Casos de uso	Entregável	10 days	5/28/2025	6/10/2025
17	**1.4**	**Desenho**	**Fase**	**21 days**	**6/11/2025**	**7/9/2025**
18	1.4.1	Escolha do desenho arquitetural	Tarefa	3 days	6/11/2025	6/13/2025
19	1.4.2	Vistas lógicas	Entregável	10 days	6/16/2025	6/27/2025
20	1.4.3	Vistas de processo	Entregável	5 days	6/30/2025	7/4/2025
21	1.4.4	Vistas físicas	Entregável	5 days	6/30/2025	7/4/2025
22	1.4.5	Vistas de implementaçã	Entregável	3 days	7/7/2025	7/9/2025
23	1.5	Implementação	Fase	40 days	7/10/2025	9/3/2025
24	**1.6**	**Avaliação**	**Fase**	**15 days**	**9/4/2025**	**9/24/2025**
25	1.6.1	Avaliação funcional da solução	Entregável	15 days	9/4/2025	9/24/2025
26	1.6.2	Testes de código	Entregável	10 days	9/4/2025	9/17/2025
27	1.6.3	Sessão de apresentação à organização	Tarefa	1 day	9/4/2025	9/4/2025
28	1.6.4	Avaliação qualitativa com a organização	Entregável	5 days	9/5/2025	9/11/2025
29	**1.7**	**Conclusão**	**Fase**	**10 days**	**9/25/2025**	**10/8/2025**
30	1.7.1	Validação do relatório	Entregável	5 days	9/25/2025	10/1/2025
31	1.7.2	Apresentação final	Entregável	10 days	9/25/2025	10/8/2025

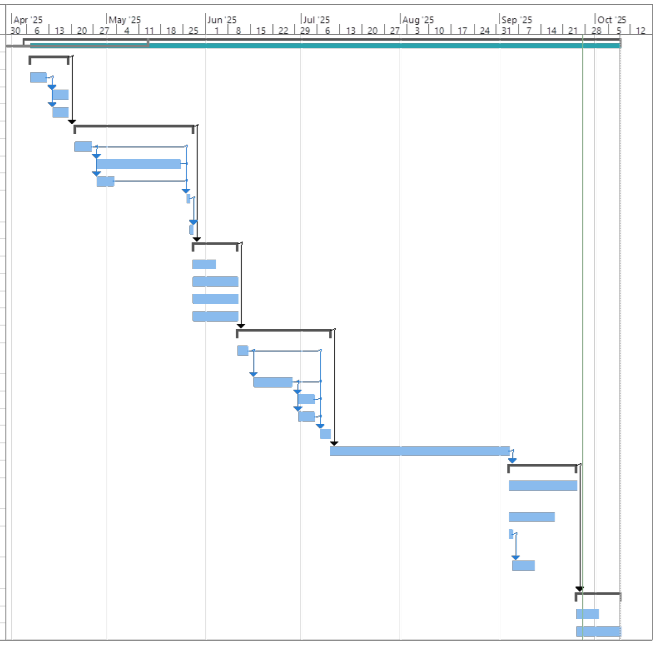


Figura 1.2: Estrutura WBS e diagrama de Gantt

Este plano garante rastreabilidade entre objetivos, entregáveis e calendário, suportando a monitorização do avanço do projeto ao longo do ciclo de vida.

1.6 Considerações Éticas

Instituto Politécnico do Porto (IPP)

No âmbito do presente trabalho, foram respeitadas as orientações éticas definidas pelo Código de Boas Práticas e de Conduta do IPP, nomeadamente os artigos 6.º, 8.º e 10.º.

- **Artigo 6.º:** Esta dissertação foi desenvolvida seguindo os princípios de honestidade, assiduidade e responsabilidade, assegurando um contributo positivo para a comunidade académica e evitando qualquer forma de plágio, fraude ou comportamento que possa comprometer a credibilidade científica;
- **Artigo 8.º:** Foi assumido o compromisso formal de garantir a originalidade do trabalho, sendo que todas as fontes de informação utilizadas encontram-se devidamente citadas e referenciadas, respeitando integralmente os direitos de autor e as regras de referenciação adotadas pelo IPP;

- **Artigo 10.º:** Relativamente às boas práticas na investigação, a dissertação foi conduzida de acordo com elevados padrões de integridade científica, dando-se prioridade à veracidade, precisão e à transparência na apresentação dos resultados.

IEEE

Foram respeitadas as normas éticas do Institute of Electrical and Electronics Engineers (IEEE) Software Engineering Code of Ethics (Gotterbarn, Miller e Rogerson, 1997), garantindo integridade, proteção de dados e responsabilidade na comunicação científica. Todos os procedimentos procuraram assegurar que os resultados são fiéis, verificáveis e reproduzíveis, evitando qualquer prática que pudesse comprometer a credibilidade ou a ética académica e profissional.

Confidencialidade e Privacidade

Todas as referências a marcas, serviços, colaboradores e documentação interna foram devidamente anonimizadas, recorrendo-se a nomes fictícios. Este cuidado teve como objetivo assegurar a conformidade com os acordos de confidencialidade da Purple, bem como proteger dados sensíveis, respeitando os princípios de privacidade e segurança da informação.

Uso de IA

Foi utilizado um serviço de IA disponibilizado internamente pela organização, com o objetivo de apoiar a melhoria da clareza, coesão e correção gramatical do texto da dissertação. Todo o uso da IA foi feito de forma transparente, garantindo que a autoria e a integridade intelectual do trabalho permanecem inalteradas, respeitando os princípios éticos da investigação.

1.7 Estrutura do Documento

Este documento está organizado em capítulos que descrevem, de forma progressiva, o problema estudado, a solução proposta, a sua implementação e a avaliação realizada. A estrutura é a seguinte:

- **Capítulo 1 – Introdução:** Apresenta a motivação, problema, objetivos e planeamento do trabalho;
- **Capítulo 2 – Fundamentos Teóricos:** Explora os conceitos teóricos essenciais ao tema em estudo;
- **Capítulo 3 – Estado da Arte:** Inclui uma revisão da literatura e análise de conceitos e boa práticas de aplicação do RAG, relacionado com os objetivos do projeto;
- **Capítulo 4 – Análise:** Apresenta o modelo de domínio, define os requisitos funcionais e não funcionais e descreve os casos de uso;
- **Capítulo 5 – Desenho:** Introduz a visão arquitetural do sistema, adotando o modelo C4 para contextualizar os componentes e as suas responsabilidades a diferentes níveis de abstração;
- **Capítulo 6 – Implementação:** Descreve detalhes da implementação prática dos módulos desenvolvidos, incluindo excertos de código, configurações e detalhes de integração;

- **Capítulo 7 – Avaliação da Solução:** Apresenta a metodologia experimental, as métricas utilizadas, os resultados obtidos na avaliação da recuperação e da geração de respostas, bem como uma análise da qualidade técnica;
- **Capítulo 8 – Conclusão:** Analisa em que medida os objetivos inicialmente definidos foram alcançados, sintetiza as principais contribuições do trabalho, discute as dificuldades encontradas ao longo do desenvolvimento e apresenta sugestões para trabalhos futuros;
- **Anexo A – Prompts:** Apresenta *prompts* relevantes para a solução proposta;
- **Anexo B – Algoritmos:** Apresenta excertos de código relevantes, usados para complementar a descrição da avaliação da solução;
- **Anexo C – Sessão de Apresentação Interna:** Documenta a sessão de apresentação realizada com utilizadores internos, apresentando evidências e feedback recolhido;
- **Anexo D – Modelação e Dados:** Inclui artefactos de modelação de baixo nível, tais como diagramas UML e modelos entidade-relação, que suportaram a análise e o desenho da solução.

Capítulo 2

Fundamentos Teóricos

Este capítulo apresenta os fundamentos teóricos que sustentam o desenvolvimento do trabalho, com foco nas principais áreas da IA relevantes para a investigação. Inicia-se com uma visão geral da IA e dos seus ramos, destacando o papel do Machine Learning (ML) e do Deep Learning (DL) no avanço recente da área. Em seguida, abordam-se os Large Language Model (LLM), descrevendo a sua evolução, arquiteturas base e técnicas de especialização, como o Fine-tuning (FT). Por fim, faz-se uma comparação entre RAG e FT, evidenciando vantagens, limitações e cenários de aplicação de cada abordagem.

2.1 Inteligência Artificial

A inteligência artificial é um ramo científico da computação que se dedica ao desenvolvimento de sistemas capazes de executar tarefas que normalmente exigiriam inteligência humana. Estes sistemas têm a capacidade de executar funções avançadas e analisar dados de grande escala a fim de gerar respostas precisas. Baseado num conceito do filósofo grego Aristóteles, a IA surgiu na década de 1950 por Allan Turing, onde o mesmo escreveu sobre a possibilidade de uma máquina pensar e imitar o comportamento humano inteligente (Machinery, 1950). Atualmente a IA é aplicada em diversos setores, como na saúde através do diagnóstico automatizado de doenças, no setor financeiro para análises de mercado e deteção de fraudes, entre outros. Recentemente a IA sofreu um rápido avanço, sendo o seu componente-chave a fundação da OpenAI em 2015 e surgimento do ChatGPT em 2022, sistema este capaz de efetuar Processamento de Linguagem Natural (PLN) e gerar respostas precisas e corretas sobre variados assuntos (Roumeliotis e Tselikas, 2023).

2.2 Machine Learning e Deep Learning

O Machine Learning é um ramo da IA centrado no desenvolvimento de algoritmos capazes de identificar padrões em dados e realizar previsões ou decisões com base nesses padrões, sem que para isso sejam explicitamente programados. Esta abordagem baseia-se na experiência, onde os modelos são treinados com dados históricos e ajustam os seus parâmetros internos para generalizar para novos dados, muitas vezes em contextos altamente variáveis e complexos (IBM, 2023a).

A aprendizagem automática pode ser agrupada em três principais paradigmas:

- Aprendizagem supervisionada, em que o modelo é treinado com exemplos rotulados (*inputs* associados a *outputs* desejados), aprendendo uma função que generaliza para novos dados;

- Aprendizagem não supervisionada, onde o objetivo é descobrir estruturas ou padrões ocultos em dados não rotulados, como agrupamentos ou relações estatísticas;
- Aprendizagem por reforço, na qual um agente interage com um ambiente, aprendendo uma política de ações com base num sistema de recompensas, com o objetivo de maximizar um retorno cumulativo.

Dentro do ML, destaca-se o Deep Learning, que se baseia no uso de redes neurais artificiais, compostas por múltiplas camadas de unidades de processamento. Estas redes têm a capacidade de modelar relações não lineares complexas, sendo particularmente eficazes em tarefas como a visão computacional, a tradução automática e o processamento de PLN.

Uma das principais vantagens da aprendizagem profunda reside na sua capacidade de extrair representações hierárquicas dos dados — as camadas iniciais aprendem caraterísticas de baixo nível, enquanto as camadas superiores capturam abstrações mais elevadas, permitindo uma compreensão mais profunda do domínio em causa. Ao contrário dos métodos tradicionais, que requerem engenharia manual de atributos, o DL automatiza esse processo, o que se revela vantajoso em cenários com grandes volumes de dados.

Estas capacidades fizeram do DL a base de várias inovações recentes em IA, em especial o LLM, que se tornou uma das áreas mais ativas e transformadoras nos últimos anos.

2.3 Large Language Models

Os LLMs representam um avanço significativo na IA. Proposta pela Google em 2017, atualmente, Transformer é a arquitetura de DL mais explorada para esta componente (Rho et al., 2024). Os Transformers foram inicialmente desenvolvidos como melhoria das arquiteturas anteriores para a tradução automática, mas desde então têm encontrado muitas aplicações, como na visão computacional e PLN. Conduziram ao desenvolvimento de sistemas pré-treinados, tais como GPT e Bidirectional Encoder Representations from Transformers (BERT) (Salıcı e Ölçer, 2024). Estes modelos são treinados através do paradigma Self-supervised Learning, no qual aprendem representações úteis dos dados sem a necessidade de rótulos manuais. No Self-supervised Learning, o próprio modelo gera os seus rótulos a partir dos dados brutos, criando tarefas preditivas auxiliares chamadas *pretext tasks*. Masked Language Modeling é um exemplo de tarefa preditiva, utilizada pelo BERT, onde palavras aleatórias são ocultadas numa frase e o modelo aprende a prever as palavras corretas, isto no contexto de PLN. Em contraste o GPT faz uso do Casual Language Modeling onde o modelo prevê a próxima palavra numa sequência de texto, dado o contexto anterior.

O aumento da escala destes modelos, tanto em volume de dados como em parâmetros, contribuiu para ganhos significativos em capacidades linguísticas, raciocínio e compreensão contextual.

2.4 Fine-tuning de LLMs

O FT de LLMs corresponde ao processo de adaptação de um modelo de linguagem previamente treinado para contextos, tarefas ou domínios específicos. Nesta abordagem, parte-se de um modelo já robusto, que foi exposto a grandes volumes de dados genéricos, e ajusta-se o seu comportamento através de um novo treino com dados mais direcionados ao objetivo pretendido. Este processo permite que o modelo aprenda nuances, terminologias e padrões

próprios do domínio de aplicação, tornando-se mais relevante e preciso para as necessidades do utilizador (X.-K. Wu et al., 2025).

Nos últimos anos, surgiram várias técnicas de FT, como Supervised Fine-tuning que é uma das abordagens mais comuns, onde o modelo é ajustado com exemplos anotados da tarefa-alvo, permitindo-lhe aprender a responder de acordo com as expectativas do domínio. Já o Full Fine-tuning implica a atualização de todos os parâmetros do modelo para concentrar o seu desempenho num domínio específico, mas exige muitos recursos computacionais e pode levar ao sobreajuste, especialmente quando se dispõe de poucos dados (X.-K. Wu et al., 2025).

2.5 Framework RAG

A *framework* RAG, introduzida por P. Lewis et al., 2020, representa uma abordagem híbrida que combina modelos de recuperação com modelos generativos pré-treinados.

Esta arquitetura, por semelhança ao FT, visa promover uma geração de linguagem natural mais contextualizada, interpretável e modular, através da incorporação dinâmica de conhecimento externo durante a inferência (Y. Gao et al., 2023).

O RAG caracteriza-se em três tipos: Naive RAG, que segue uma arquitetura simples de indexação, recuperação e geração; Advanced RAG, que introduz otimizações no pré-processamento, como segmentação de textos, adição de metadados e estruturação de índices, e no pós-processamento, como reordenação de documentos (*reranking*) e ajuste do tamanho do contexto (*context window*) para melhorar a relevância e eficiência da geração; e Modular RAG, que permite a substituição e reconfiguração de componentes da *pipeline*, oferecendo maior flexibilidade e adaptabilidade a diferentes tarefas (Eibich, Nagpal e Fred-Ojala, 2024; Y. Gao et al., 2023). A Figura 2.1 ilustra de forma resumida a arquitetura moderna dos sistemas RAG.

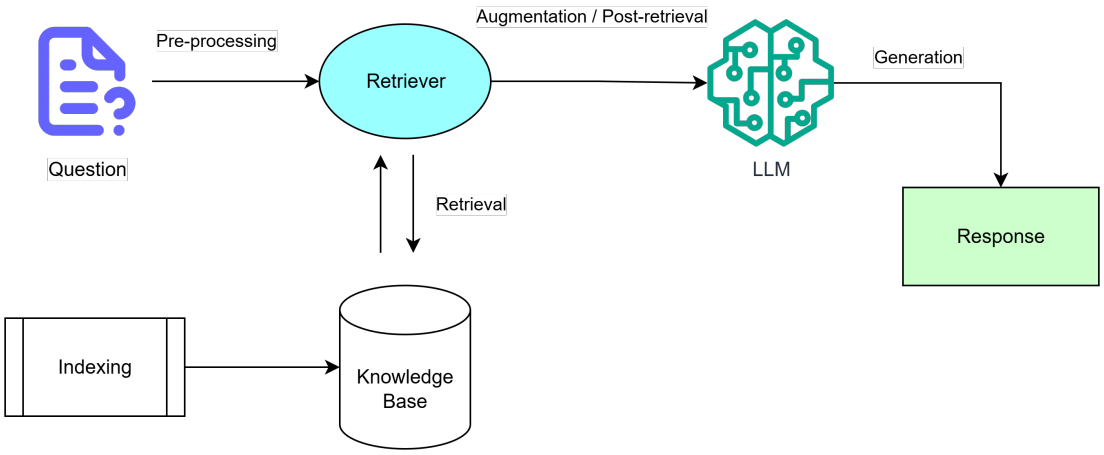


Figura 2.1: Arquitetura simplificada do RAG moderno

As fases do RAG processam-se por ordem sequencial e consistem no seguinte:

- **Indexação**: Nesta etapa, os documentos ou fontes de conhecimento são processados e convertidos em estruturas de dados. O processo de indexação pode incluir

pré-processamento dos textos, segmentação em passagens relevantes e adição de metadados para facilitar a recuperação posterior, isto numa ótica de Advanced RAG.

- **Recuperação**: Baseado numa *query* de *input*, o objetivo desta fase é percorrer o conhecimento disponível e encontrar informação semelhante. Funciona como uma espécie de motor de pesquisa e é essencial pois é o fator determinante da relevância da informação que será usada para gerar a resposta final.
- **Augmentation**: Através da *query* inicial e da informação recuperada no passo anterior, o processo de *augmentation* consiste na elaboração do *prompt* enriquecido com a informação contextual. Numa ótica de Naive RAG, a *query* apenas é concatenada com a informação contextual recuperada, mas com técnicas mais avançadas da atualidade pode existir um *reranking* de documentos ou um *Prompt Engineering* mais sofisticado para estruturar o *prompt* da forma mais adequeada para atender as necessidades do sistema.
- **Geração**: Consiste na chamada ao LLM textual e geração da resposta através do *input prompt* formulado no passo anterior. O conhecimento pré-existente é tido em conta pelo LLM permitindo que as respostas sejam mais inteligentes e informadas para o contexto em questão.

2.6 RAG vs. Fine-tuning

O RAG combina um modelo gerador com um componente de recuperação de informação, permitindo que o modelo aceda a documentos externos relevantes durante a inferência. Esta abordagem facilita a utilização de conhecimento dinâmico e atualizável sem a necessidade de retreinar o modelo, o que é particularmente útil em domínios em constante evolução. Por outro lado, o FT consiste na atualização dos parâmetros do modelo com base num conjunto de dados específico, permitindo uma especialização mais profunda para tarefas ou domínios particulares, mas com custos de computação e manutenção mais elevados.

A Tabela **??** sintetiza as principais diferenças entre estas duas abordagens, considerando fatores como armazenamento de conhecimento, atualizações dinâmicas, custos computacionais, desempenho em tarefas específicas, escalabilidade e risco de alucinação.

Tabela 2.1: Comparação entre RAG e FT (Y. Gao et al., 2023)

Critério	RAG	Fine-tuning
Armazenamento de conhecimento	Externo (corpus de documentos)	Interno (parâmetros do modelo)
Conhecimento dinâmico	Atualizável facilmente (atualização do corpus)	Difícil de atualizar; requer retreino
Custo computacional na inferência	Elevado (necessidade de recuperação)	Baixo (geração direta)
Desempenho em tarefas específicas	Depende da qualidade do recuperador	Alto se o modelo for especializado
Escalabilidade para múltiplos domínios	Alta; basta adicionar documentos	Baixa; cada domínio pode exigir novo modelo
Risco de alucinações	Reduzido se a recuperação for precisa	Mais elevado se o modelo memorizar informação incorreta

Como ilustrado na Figura 2.2, o RAG permite uma abordagem mais flexível e adaptável, enquanto o FT é mais indicado para tarefas altamente especializadas onde se pretende otimizar o desempenho do modelo para um domínio específico.

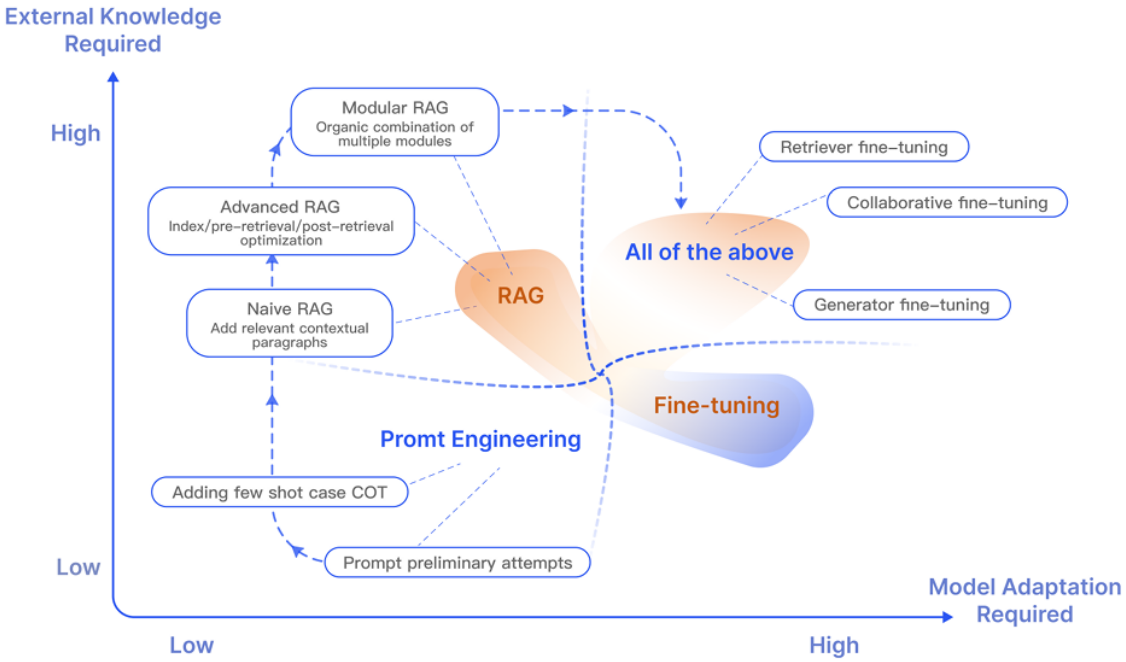


Figura 2.2: Comparação entre FT e RAG a nível de adaptação do modelo e dependência de conhecimento externo (Y. Gao et al., 2023)

Capítulo 3

Estado da Arte

Este capítulo apresenta, de forma sintética, o estado da arte que fundamenta a conceção, implementação e avaliação da solução proposta, seguindo a abordagem do DSR. Inicia com a revisão de literatura, orientada por Research Questions (RQ) e critérios explícitos de elegibilidade, para mapear práticas, lacunas e evidência relevante em RAG.

São analisadas as principais opções de armazenamento de conhecimento (vetorial, relacional, documentos e grafos), bem como práticas de indexação e estratégias de recuperação. São ainda sistematizados desafios técnicos típicos em suporte técnico e respetivas medidas de mitigação.

Procede-se à comparação de tecnologias *frontend* e *backend*, e à análise de frameworks RAG. Aborda-se ainda a integração com os sistemas externos Confluence e Control-M. Por fim, apresentam-se trabalhos relacionados, que validam escolhas arquiteturais e evidenciam ganhos de RAG em cenários reais de suporte técnico.

3.1 Revisão de Literatura

A revisão de literatura constitui um passo essencial no âmbito do DSR, uma vez que permite mapear o estado da arte, identificar lacunas de conhecimento e fundamentar a conceção e avaliação de artefactos (Peffers et al., 2007). Para além de apoiar a definição das questões de investigação, esta etapa contribui para assegurar que o desenvolvimento da solução proposta está relacionado com evidências científicas e práticas consolidadas.

Esta secção centra-se na apresentação detalhada da metodologia de investigação adotada e posterior análise dos resultados.

3.1.1 Metodologia de Investigação

A metodologia baseou-se na formulação das questões de investigação, as quais serviram de guia para o processo de conceção e avaliação do artefacto, bem como para a definição do âmbito da investigação e dos critérios de elegibilidade.

3.1.1.1 Questões de Investigação

As questões de investigação elaboradas encontram-se na Tabela 3.1.

Tabela 3.1: Questões de investigação

ID	Questão
RQ1	Quais são as principais diferenças, vantagens e limitações entre os tipos de armazenamento de conhecimento no contexto de RAG?
RQ2	Quais são as práticas atuais de indexação e recuperação de conhecimento em sistemas RAG e como se comparam?
RQ3	Quais são as abordagens de *augmentation* mais eficazes na melhoria da relevância e precisão das respostas em sistemas RAG?
RQ4	Quais são os principais desafios técnicos envolvidos na conceção e implementação de uma solução RAG aplicada ao suporte técnico e como os contornar?

A RQ1 centra-se na análise de diferentes abordagens e estruturas de armazenamento utilizadas na coleção de dados que compõem um sistema RAG. Esta questão será respondida através de uma análise crítica, resultando na justificação da abordagem mais adequada para o presente projeto, criando assim uma ponte natural para a exploração das questões seguintes.

Na RQ2, serão analisadas as diferentes práticas de indexação e recuperação de conhecimento, dado que estas etapas constituem elementos cruciais para garantir a eficiência e a precisão na recuperação da informação.

A RQ3 incidirá sobre a comparação das estratégias de *augmentation* aplicadas durante a fase precedente à geração, avaliando de que forma estas contribuem para aumentar a relevância e a precisão das respostas produzidas.

Por fim, a RQ4 analisa os principais desafios técnicos na implementação de soluções RAG, como a complexidade da documentação, limitações na recuperação semântica e risco de respostas incorretas. Serão também exploradas estratégias de mitigação, otimização de *chunking* e *embeddings*, uso de metadados e calibragem da *pipeline*, de forma a reforçar a robustez e fiabilidade do sistema.

3.1.1.2 Âmbito da Investigação

Foi definida uma *query* de investigação (3.1) com o objetivo de orientar a pesquisa bibliográfica de forma estruturada. A pesquisa foi complementada por uma abordagem exploratória e iterativa, ajustando os termos de pesquisa consoante os resultados obtidos e as necessidades específicas do trabalho. Note-se que a *query* foi redigida em inglês para maximizar os resultados.

```
("AI" OR "artificial intelligence")
AND ("retrieval-augmented generation" OR "RAG")
AND  (
("vector database" OR "relational database" OR "NoSQL database" OR "document
   database", "document store", "graph database")
OR ("retrieval" OR "indexing" OR "augmentation" OR  "prompt engineering")
OR ("challenge" OR "limitation" OR "scalability" OR "evaluation")
)
```

Código 3.1: *Query* de investigação

Esta *query* serviu como ponto de partida para identificar fontes relevantes sobre os conceitos fundamentais, desafios técnicos e práticas associadas à aplicação de RAG.

Os repositórios de pesquisa utilizados estão referenciados na Tabela 3.2. Estas fontes permitiram recolher estudos com informação fiável e de elevada qualidade sobre a área em análise.

Tabela 3.2: Bases de dados de investigação

Base de Dados (BD)	**Uniform Resource Locator (URL)**
ACM Digital Library	https://dl.acm.org/
B-On	https://www.b-on.pt/
Google Scholar	https://scholar.google.com/

Outras fontes de investigação também foram utilizadas para complementar o estudo, como documentação oficial e *websites* fidedignos.

3.1.1.3 Critérios de Elegibilidade

Os critérios de elegibilidade de artigos e *websites* encontram-se representados nas Tabelas 3.3 e 3.4, respetivamente. Tendo este projeto adotado uma *stack* tecnológica relativamente recente e muita matéria de estudo relevante estar constantemente a ser produzida, artigos publicados antes de 2021 foram excluídos. Artigos que não fornecem detalhes técnicos sobre a aplicação de RAG foram também excluídos, bem como aqueles que não foram revistos por pares ou não relacionados com o ramo de engenharia de *software*. Relativamente a *websites*, achou-se por bem a sua inclusão, mas apenas aqueles que incluam informação fidedigna, como documentação oficial de *frameworks* de desenvolvimento ou artigos publicados por entidades responsáveis pelas ferramentas.

Tabela 3.3: Critérios de inclusão e exclusão para artigos

Inclusão
Analisa tecnologias de armazenamento em termos de caraterísticas, funcionalidades e limitações
Discute a aplicação de RAG em contextos diversos
Analisa desafios técnicos relacionados com a indexação, recuperação de informação em sistemas RAG
Exclusão
Não relacionado com engenharia de *software*
Não fornece detalhes técnicos, arquiteturais ou metodológicos
Publicado antes de janeiro de 2021
Não revisto por pares

Tabela 3.4: Critérios de inclusão e exclusão para *websites*

Inclusão
Website oficial de *frameworks*, bibliotecas ou ferramentas relevantes Documentação técnica publicada por organizações ou empresas responsáveis pelas ferramentas
Blogs técnicos ou artigos publicados por equipas de desenvolvimento das ferramentas em questão
Páginas que descrevem APIs (Application Programming Interface), configurações, tutoriais ou guias técnicos detalhados relacionados com RAG ou integração de sistemas
Exclusão
Websites sem ligação oficial com as ferramentas analisadas
Fóruns, comentários ou respostas não verificadas
Conteúdos genéricos ou superficiais sem valor técnico concreto
Conteúdo desatualizado ou sem data de publicação explícita

A seleção dos artigos foi realizada de forma iterativa. Numa primeira fase, analisaram-se os títulos para aferir a sua relevância. Quando o título não fornecia informação suficiente, procedeu-se à leitura dos resumos. Nos casos em que persistiam dúvidas quanto ao enquadramento nos critérios de inclusão, recorreu-se à análise integral do artigo.

3.1.2 Resultados e Análise

De seguida serão apresentados resultados obtidos a partir da revisão da literatura, bem como a análise crítica dos mesmos face às questões de investigação definidas.

3.1.2.1 RQ1: Quais são as principais diferenças, vantagens e limitações entre os tipos de armazenamento de conhecimento no contexto de RAG?

A escolha do armazenamento de conhecimento é um fator crítico em sistemas RAG, pois influencia diretamente a eficiência na indexação, pesquisa e recuperação de informação relevante. Diferentes tipos de Sistemas de Gestão de Bases de Dados (SGBD) apresentam caraterísticas próprias que os tornam mais ou menos adequados para cenários RAG.

No contexto desta questão de investigação, consideraram-se quatro categorias de armazenamento de conhecimento: bases de dados vetoriais, bases de dados relacionais Structured Query Language (SQL), bases de dados de documentos Not Only SQL (NoSQL) e bases de dados orientadas a grafos. A análise efetuada fornece os elementos necessários para fundamentar a escolha do SGBD mais adequado ao projeto, concluindo com uma comparação crítica das tecnologias avaliadas.

Bases de Dados Vetoriais

As bases de dados vetoriais são soluções modernas que permitem o armazenamento de dados associados a vetores. No contexto de RAG, um vetor é uma sequência numérica de várias dimensões que codifica o significado semântico de um objeto, podendo este ser um texto,

imagem, áudio, etc. Dados semelhantes terão representações próximas no espaço vetorial (Rau et al., 2024; Špeletić et al., 2024).

O cálculo da distância entre dois vetores é feito através de métricas de similaridade:

- Similaridade por cosseno: avalia o valor do cosseno do ângulo compreendido entre dois vetores, amplamente utilizado em PLN;
- Distância Euclidiana: mede a distância direta entre dois pontos num espaço *n*-dimensional;
- Produto escalar: mede a magnitude da projeção de um vetor sobre outro.

Este tipo de cálculos permite encontrar conteúdos semanticamente semelhantes mesmo quando não há correspondência exata de palavras-chave.

Soluções modernas baseadas em IA designadas de modelos de *embedding* são amplamente utilizadas para converter dados não estruturados em vetores. Estes modelos, geralmente baseados em LLMs ou arquiteturas *transformer*, convertem o conteúdo em representações numéricas de alta dimensão (Rau et al., 2024; Špeletić et al., 2024). A efetividade de cada modelo de *embedding* varia conforme o idioma e o domínio do conteúdo. O modelo OpenAI Text-embedding-ada-002 apresenta desempenho competitivo em diversas línguas, com resultados especialmente fortes em inglês, mas também com boa generalização em idiomas amplamente falados, como espanhol, francês e português. Por outro lado, modelos como o Cohere Multilingual-v3 e o Aleph Alpha Luminous são projetados para lidar com tarefas multilingues, incluindo contextos com menos recursos linguísticos, embora seu desempenho varie conforme o *benchmark* e o idioma considerado (Kamalloo et al., 2023).

Um estudo conduzido por X. Wang et al. (2024) avaliou diferentes SGBDs vetoriais *open-source*: Weaviate, Faiss, Chroma, Qdrant e Milvus. A avaliação teve por base quatro critérios:

- *Multiple Index Type* – flexibilidade na escolha de estruturas de índice adaptadas às caraterísticas dos vetores;
- *Billion-scale* – capacidade de lidar com coleções de dados em escala massiva;
- *Hybrid Search* – combinação de pesquisa vetorial semântica com pesquisa tradicional por palavras-chave;
- *Cloud-native* – suporte a gestão, escalabilidade e resiliência em ambientes de *cloud computing*.

O Milvus e o Qdrant destacaram-se como sendo as soluções mais completas, embora cada um possua caraterísticas que o tornam mais adequado a contextos específicos. O Qdrant é ideal para projetos de média dimensão que valorizam uma configuração simples, baixa latência e fácil integração com *frameworks* de desenvolvimento, sendo uma escolha vantajosa quando o custo e a facilidade operacional são prioridades (Qdrant Team, 2024; X. Wang et al., 2024). Por outro lado, o Milvus sobressai em cenários que envolvem grandes volumes de dados, oferecendo escalabilidade avançada, alto desempenho, suporte a Graphics Processing Unit (GPU) e uma comunidade *open-source* ativa. Esta combinação faz do Milvus uma solução robusta e flexível, particularmente adequada para aplicações empresariais que exigem confiabilidade e capacidade de expansão. Ambas estão projetadas para serem sistemas distribuídos, existindo recursos nativos que suportam a escalabilidade horizontal, permitindo a adição de nós para aumentar a capacidade de armazenamento e processamento conforme necessário.

Além das opções *open-source*, várias soluções são oferecidas em modelo SaaS, como Pinecone, Weaviate Cloud e Qdrant Cloud (Guo et al., 2022). Estas plataformas disponibilizam funcionalidades semelhantes, mas com gestão totalmente abstraída do utilizador. Algumas vantagens na utilização de soluções SaaS incluem a escalabilidade automática com o ajuste dinâmico de recursos conforme o volume e complexidade das consultas e a garantia de replicação e tolerância a falhas por Service Level Agreement (SLA). Desvantagens incluem questões de privacidade e conformidade em cenários com dados confidenciais e custos mais elevados a longo prazo pelo pagamento recorrente do serviço.

Resumindo, seguem-se as principais vantagens e limitações no uso de uma BD vetorial numa ótica de RAG:

Vantagens:

- Pesquisa semântica eficiente baseada em vetores;
- Escaláveis para grandes volumes de dados;
- Possibilidade de pesquisa híbrida (semântica + palavras-chave).

Limitações:

- Custos e complexidade de implementação e manutenção;
- Recursos computacionais elevados em cenários de grande escala;
- Necessidade de integração com bases SQL/NoSQL para uma gestão de metadados mais eficiente.

Bases de Dados Relacionais (SQL)

As bases de dados relacionais constituem uma das tecnologias mais consolidadas no armazenamento e gestão de informação estruturada. Neste tipo de sistema os dados são armazenados em tabelas com colunas e linhas, cada coluna representando um tipo de dado e cada linha representando uma entrada específica. As relações entre as tabelas são definidas através de chaves primárias e chaves estrangeiras que facilitam consultas complexas com auxílio do SQL. Suportadas pelas propriedades do Atomicidade, Consistência, Isolamento e Durabilidade (ACID), estas bases de dados configuram uma tecnologia bastante testada e consolidada, garantindo transações confiáveis mesmo em cenários de falhas ou acesso simultâneo. Destacam-se alguns SGBDs como MySQL, Microsoft SQL Server e PostgreSQL, amplamente utilizados no mercado (IBM, 2023c).

Contudo, apresentam algumas limitações no que toca pesquisa semântica, ideal em sistemas RAG, uma vez que não são otimizadas de forma nativa para este tipo de operações.

Por esse motivo, surgiram soluções como PGVector, uma extensão do PostgreSQL que permite o armazenamento e a pesquisa de vetores diretamente numa base de dados relacional. Esta extensão adiciona um novo tipo de dado, denominado *vector*, possibilitando o armazenamento de vetores e permitindo a realização de operações de similaridade, como a similaridade por cosseno e distância euclidiana, diretamente através de consultas SQL. Esta solução apresenta algumas limitações relacionadas com a escalabilidade e desempenho com grandes volumes de dados. Além disso, o tamanho dos vetores é limitado a 2000 dimensões, o que impede a utilização de modelos de *embedding* maiores e mais robustos (Harjono, 2025; PGVector, 2025).

O uso de bases de dados relacionais numa vertente de armazenamento de vetores pode ser uma boa opção em sistemas RAG quando o volume de dados é moderado.

Um ponto forte das bases de dados relacionais é o armazenamento eficiente de metadados com informações sobre documentos, podendo complementar soluções de armazenamento vetorial, mantendo os vetores em sistemas especializados enquanto os metadados permanecem em tabelas relacionais, permitindo consultas combinadas e gestão mais organizada do conhecimento.

Vantagens:

- Utilizar uma base de dados relacional pode simplificar a infraestrutura, evitando a necessidade de gerir uma base de dados vetorial separada;
- Fácil integração com sistemas existentes que já utilizam bases de dados relacionais;
- Garantias das propriedades ACID;
- Excelente para armazenamento de metadados dos documentos.

Limitações:

- Escalabilidade limitada para grandes volumes de dados;
- Limitações no tamanho dos vetores suportados, que pode impedir a utilização de modelos de *embedding* mais robustos.

Bases de Dados de Documentos (NoSQL)

As bases de dados de documentos são um tipo de base de dados NoSQL que armazenam informação em formato semiestruturado, tipicamente em JavaScript Object Notation (JSON) ou Extensible Markup Language (XML). Armazenam informação dos documentos através de objetos compostos por pares chave-valor, o que permite representar dados complexos e com esquema personalizado. Estas bases de dados são úteis em cenários onde os dados são heterogéneos, mutáveis ou não possuem uma estrutura fixa (IBM, 2023b).

Exemplos populares como MongoDB, Couchbase e Elasticsearch incluem funcionalidades nativas de Full-Text Search (FTS), que consiste numa técnica de pesquisa que permite pesquisar palavras ou frases dentro de textos armazenados numa base de dados, em vez de se limitar a pesquisas restritas por chave ou metadado específico, o que constitui uma boa abordagem para recuperação de informação em sistemas RAG.

O FTS tanto no MongoDB como no Couchbase suporta uma pesquisa eficiente por palavras-chave, possibilitando tokenização, *stemming* e *stop words*, mas não realiza correspondência semântica, o que significa que termos sinónimos não são automaticamente reconhecidos. Já o Elasticsearch, por outro lado, suporta um FTS mais robusto, pois para além de suportar pesquisa por palavras-chave, oferece funcionalidades avançadas de análise de texto e ranking de relevância. Embora também não realize análise semântica nativa, a partir da versão 7, o Elasticsearch permite implementar pesquisas semânticas através de vetores e *embeddings* (Yang et al., 2025).

Vantagens:

- Permitem armazenar dados heterogéneos e sem estrutura rígida, adaptando-se facilmente a diferentes tipos de documentos e metadados;
- Suporte nativo a FTS;

- Fácil escalabilidade horizontal;
- Possibilidade de integração com vetores e *embeddings* (no caso do Elasticsearch 7+), o que possibilita combinar pesquisa tradicional e semântica;
- Desempenho superior em consultas complexas sobre documentos quando comparado com bases de dados relacionais.

Limitações:

- Pesquisas semânticas não são nativas, estando limitadas a pesquisas tradicionais por palavra-chave;
- Indexação e manutenção de grandes coleções de documentos podem consumir muitos recursos, afetando o desempenho;
- Poderão surgir problemas de consistência em cenários de transações complexas devido ao suporte limitado sobre as propriedades ACID.

Bases de Dados Orientadas a Grafos

As bases de dados orientadas a grafos armazenam informação como nós e arestas, sendo cada nó uma entidade e cada aresta uma relação entre entidades. Tanto nós como arestas podem possuir propriedades adicionais que descrevem atributos específicos. Este modelo é particularmente útil para representar dados com relações complexas e interconectadas, como redes sociais e recomendações, permitindo consultas sofisticadas sobre conexões entre entidades.

A consulta a bases de dados de grafos é realizada através de linguagens específicas, como Cypher (Neo4j) ou Gremlin (Apache TinkerPop), que permitem percorrer relacionamentos (*traversals*), identificar padrões e calcular métricas de centralidade ou similaridade entre nós. Esta abordagem é especialmente relevante em sistemas RAG, pois permite representar e explorar relações semânticas entre documentos ou entidades, complementando o armazenamento vetorial de conteúdo (Kuok, H. H. Liu e Lo, 2025).

Alguns exemplos de sistemas de gestão de bases de dados orientadas a grafos incluem:

- **Neo4j** – líder de mercado, forte em consultas complexas e visualização de grafos;
- **Amazon Neptune** – serviço gerido em nuvem que suporta grafos Resource Description Framework (RDF) e Property Graph;
- **OrientDB** – combina funcionalidades de grafos e NoSQL;
- **TigerGraph** – otimizado para grafos de grande escala e análises em tempo real.

Vantagens:

- Modelagem natural de relações complexas e interdependentes;
- Consultas de relacionamento altamente eficientes, mesmo em grafos grandes;
- Adequado para análises de redes e grafos de conhecimento em sistemas RAG;
- Flexibilidade na evolução do esquema, sem necessidade de reestruturar dados existentes.

Limitações:

- Curva de aprendizagem mais elevada devido a linguagens e paradigmas específicos de grafos;
- Menor maturidade em cenários de grande volume de dados comparativamente às bases de dados vetoriais ou relacionais;
- Escalabilidade pode ser custosa em grafos massivos;
- Integração com sistemas existentes (SQL, NoSQL) pode ser mais complexa.

No contexto de RAG, as bases de dados orientadas a grafos podem complementar soluções vetoriais, permitindo não apenas recuperar conteúdos semanticamente semelhantes, mas também explorar relações e contextos semânticos entre documentos ou entidades, enriquecendo a recuperação de conhecimento e suportando inferências mais complexas.

Comparação de tecnologias

Na Tabela 3.5 encontra-se a comparação resumida dos diferentes tipos de bases de dados analisadas nas secções anteriores. São apresentadas as principais caraterísticas, vantagens e limitações numa ótica de RAG e alguns exemplos de SGBDs existentes.

As bases de dados vetoriais destacam-se pela eficiência na pesquisa semântica e escalabilidade, embora sejam complexas de implementar e exigentes em recursos. As relacionais (SQL) são simples de integrar e adequadas para dados estruturados, mas têm limitações em escalabilidade e no suporte a vetores de alta dimensão. Já as orientadas a documentos (NoSQL) oferecem flexibilidade e boa escalabilidade horizontal, mas não suportam nativamente pesquisas semânticas e podem implicar custos elevados de indexação. Por fim, as orientadas a grafos são ideais para modelar relações complexas, embora apresentem maiores desafios de escalabilidade, integração e aprendizagem.

Tabela 3.5: Comparação entre tipos de bases de dados no contexto de RAG

Tipo de BD	Caraterísticas	Vantagens	Limitações	SGBDs
Vetoriais	Armazenam dados associados a vetores; suportam pesquisa semântica por similaridade	Pesquisa semântica eficiente; escaláveis; pesquisa híbrida (palavras-chave + semântica)	Implementação e manutenção complexa; alto consumo de recursos	Milvus, Qdrant, Weaviate, Faiss, Chroma, Pinecone, Weaviate Cloud, Qdrant Cloud
Relacionais (SQL)	Dados estruturados em tabelas; relações entre tabelas via chaves primárias/estrangeiras; suporte ACID	Infraestrutura simplificada; fácil integração com sistemas existentes; armazenamento eficiente de metadados; propriedades ACID	Escalabilidade limitada; suporte limitado a vetores de alta dimensão; desempenho reduzido com grandes volumes de dados	MySQL, PostgreSQL (com extensão PGVector), Microsoft SQL Server
Documentos (NoSQL)	Armazenam dados semi-estruturados (JSON, XML); flexíveis quanto a esquema	Armazenamento flexível de dados heterogéneos; suporte a FTS; boa escalabilidade horizontal; possível integração com *embeddings*	Pesquisas semânticas não nativas; indexação de grandes coleções pode ser custosa; consistência limitada em transações complexas	MongoDB, Couchbase, Elasticsearch
Orientadas a Grafos	Armazenam dados como nós e arestas; foco em relações complexas entre entidades; consultas por *traversal*	Modelagem natural de relações complexas; consultas de relacionamento eficientes; adequado para grafos de conhecimento; flexível quanto a evolução do esquema	Curva de aprendizagem elevada; escalabilidade custosa em grafos massivos; menor maturidade para grandes volumes; integração complexa com sistemas existentes	Neo4j, Amazon Neptune, OrientDB, TigerGraph

Para dar continuidade ao projeto, optou-se pela utilização da base de dados vetorial Qdrant, uma vez que reúne um conjunto de caraterísticas que a tornam particularmente adequada ao contexto em causa. A escolha desta tecnologia justifica-se por ser uma solução *open-source*, com baixo custo e fácil operação, oferecendo baixa latência e suporte eficiente à pesquisa

semântica. Além disso, apresenta-se como uma opção equilibrada e robusta para projetos de média dimensão, conciliando desempenho, escalabilidade e simplicidade de manutenção.

Tendo em consideração a opção por uma base de dados vetorial, as questões de investigação seguintes serão analisadas sob a perspetiva deste tipo de armazenamento.

3.1.2.2 RQ2: Quais são as práticas atuais de indexação e recuperação de conhecimento em sistemas RAG e como se comparam?

De forma a dar resposta a esta questão, serão descritas algumas práticas de indexação e recuperação, com foco em estruturas de armazenamento vetorial, tal como foi definido na RQ1. A análise será dividida em duas partes, primeiro a indexação, destacando abordagens de preparação e armazenamento dos dados, e depois a recuperação, abordando diferentes estratégias para localizar *chunks* relevantes. Por fim, em ambos os casos, será apresentada uma breve análise crítica, permitindo comparar as vantagens e limitações das abordagens existentes.

Indexação

A indexação consiste na preparação e armazenamento dos dados em estruturas adequadas. Algumas das práticas atuais são: *chunking*, indexação bruta e indexação aproximada.

Chunking

O chunking acontece numa fase de preparação dos dados e consiste na segregação dos documentos (Figura 3.1), os chamados *chunks*, e posteriormente a sua conversão em vetores. O chunking tem a vantagem de melhorar a precisão de recuperação, pois documentos inteiros tendem a gerar vetores genéricos que diluem o conteúdo relevante (Yepes et al., 2024).

A forma como um documento é dividido depende muito do seu tipo e da forma como está organizado. Em textos bem estruturados, como Markdown ou HyperText Markup Language (HTML), a segmentação tende a acompanhar a hierarquia do conteúdo (títulos, subtítulos, listas) garantindo que cada parte mantém a sua coerência. Já em documentos menos estruturados, como PDFs ou ficheiros Word, a divisão é feita sobretudo com base no tamanho, resultando em blocos de dimensão semelhante.

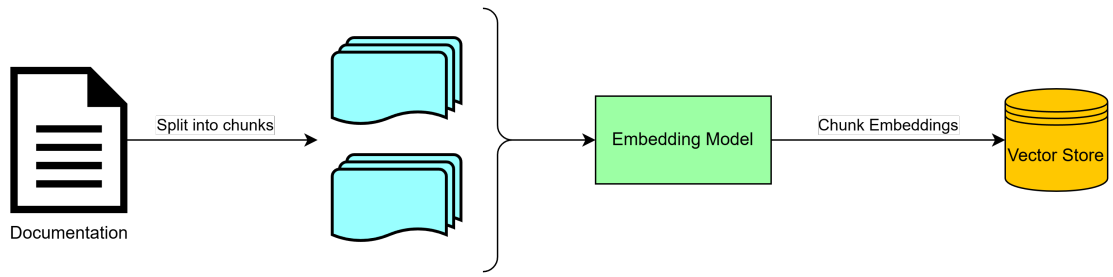


Figura 3.1: Processo de chunking (adaptado de Špeletić et al., 2024)

Indexação Bruta (Flat Index)

Na indexação bruta, todos os vetores são armazenados diretamente e a pesquisa é feita por comparação exata, dependendo do tipo de algoritmo utilizado (p.e. similaridade por cosseno). Tem a vantagem de a recuperação ser exata e sem aproximações mas é limitada em termos de escalabilidade para grandes volumes de dados.

Indexação Aproximada (Approximate Nearest Neighbor (ANN))

A indexação aproximada cria estruturas que aceleram a pesquisa de vetores mais próximos sem necessidade de comparar entre todos os pontos (Genesis e Keane, 2025).

Existem diversas técnicas de indexação aproximada, como por exemplo (Genesis e Keane, 2025; Mackenzie et al., 2025):

- Hierarchical Navigable Small World (HNSW) Graph: estrutura baseada em grafos navegáveis, excelente relação precisão-velocidade, popular em bases de dados Qdrant e Milvus ;
- Inverted File Index: vetores são agrupados em clusters e a pesquisa é feita apenas nos clusters mais relevantes sendo uma boa opção para coleções de grande dimensão;
- Product Quantization: comprime vetores para reduzir espaço sendo útil quando a memória é limitada.

Esta abordagem tem a vantagem de ser bastante escalável, mas pode causar perda de precisão na recuperação.

Indexação - Conclusões

Em síntese, o *chunking* melhora a precisão ao segmentar documentos em partes menores, enquanto a indexação bruta garante exatidão mas não é escalável. Já a indexação aproximada oferece maior eficiência em grandes volumes de dados, com uma ligeira perda de precisão.

No contexto deste projeto, justifica-se a utilização do *chunking*, dado que os documentos da equipa podem ter grande dimensão, e da abordagem HNSW, uma vez que o Qdrant a integra de forma nativa.

Recuperação

Destacam-se algumas das práticas atuais de recuperação: recuperação semântica, recuperação lexical e recuperação híbrida.

Recuperação Semântica

A recuperação semântica baseia-se exclusivamente em vetores. Inicialmente, a *query* de pesquisa é vetorizada pelo mesmo modelo de *embedding* utilizado na indexação, garantindo compatibilidade entre os vetores. Em seguida, realiza-se a pesquisa por similaridade entre o vetor da *query* e os vetores dos *chunks* previamente indexados (Figura 3.2). Esta abordagem permite identificar conteúdos relevantes mesmo quando não há correspondência literal de palavras. Após a recuperação dos *chunks* mais relevantes, estes são combinados com a *query* original para gerar o *prompt* final, que é enviado ao LLM, resultando na resposta elaborada. A principal vantagem desta abordagem é a capacidade de capturar relações semânticas, enquanto as limitações incluem maior custo computacional e necessidade de modelos de *embedding* robustos (Doan et al., 2024; Kamalloo et al., 2023).

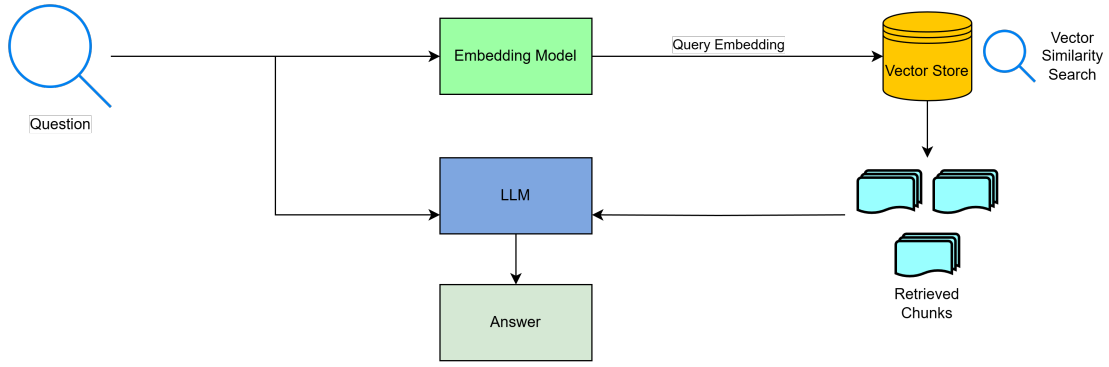


Figura 3.2: Processo de recuperação e geração exclusivamente com modelo de *embedding* (Adaptado de Špeletić et al., 2024)

Recuperação Lexical

A recuperação lexical atua diretamente sobre o texto legível dos *chunks*, sem recorrer a *embeddings*. As consultas são comparadas com os documentos usando correspondência de palavras-chave, destacando-se pelo baixo custo computacional e rapidez na recuperação. Um exemplo clássico é o Best Matching 25 (BM25), que identifica os *chunks* mais relevantes com base na frequência e na distribuição das palavras na coleção de documentos (Gupta, Ranjan e Singh, 2024; Kamalloo et al., 2023). Embora seja eficiente para consultas simples, este método não captura relações semânticas e pode falhar quando o significado desejado não coincide com os termos exatos presentes nos documentos.

Abordagem Híbrida

A abordagem híbrida combina recuperadores lexicais e semânticos. Inicialmente, o recuperador lexical filtra os *chunks* mais relevantes. De seguida processa-se a fase de *ranking*, onde através do modelo de *embedding*, os *chunks* e a *query* são vetorizados e avaliados a nível de similaridade, resultando no *top-k chunks* semanticamente mais adequados (Figura 3.3). Este método aproveita a rapidez da recuperação baseada em palavras-chave e a profundidade semântica dos *embeddings*, sendo adequado quando se procura equilíbrio entre a velocidade e qualidade das respostas (Kamalloo et al., 2023).

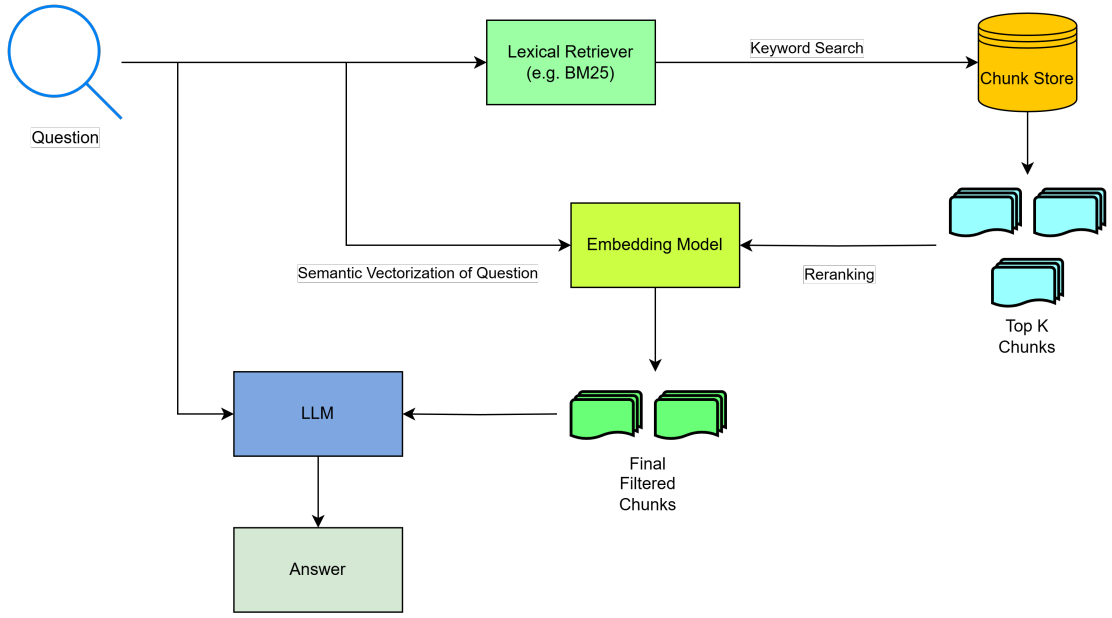


Figura 3.3: Processo de recuperação híbrida

Recuperação - Conclusões

A escolha entre recuperação semântica, lexical ou híbrida depende do contexto e dos requisitos do sistema. A recuperação semântica é mais eficaz para contextos complexos e técnicos, onde a compreensão do significado é crucial. A recuperação lexical pode ser suficiente para cenários mais simples ou quando a velocidade é uma prioridade. Já a abordagem híbrida combina as vantagens de ambas, permitindo uma recuperação inicial rápida por palavras-chave, seguida de uma filtragem semântica para garantir maior relevância e precisão das respostas, sendo especialmente indicada quando se pretende equilibrar desempenho e qualidade.

No contexto deste projeto, tendo por base a importância de captar semântica nas pesquisas, optou-se por adotar a abordagem de recuperação semântica.

3.1.2.3 RQ3: Quais são as abordagens de augmentation mais eficazes na melhoria da relevância e precisão das respostas em sistemas RAG?

A fase de *augmentation* consiste principalmente na construção do *prompt* final, mas pode também incluir transformações adicionais sobre os documentos recuperados, como sumarização ou reescrita, dependendo da abordagem adotada. Após a recuperação dos documentos ou *chunks* mais relevantes, estes são integrados de forma estruturada ao lado da *query* original, criando assim um *prompt* enriquecido permitindo ao modelo conhecer o contexto necessário para gerar uma resposta mais precisa. Prompt Engineering é o nome comum associado a esta prática e influencia diretamente a qualidade das respostas (Ibtasham, 2024).

A fase de *augmentation* pode envolver os seguintes fatores (Ibtasham, 2024; Son e S. Lee, 2025):

- Número e relevância dos documentos utilizados no prompt: caso se opte pelo uso da estratégia de reranking, esta será realizada na fase de *augmentation* com o objetivo de selecionar os *top-k chunks* finais mais relevantes;

- Formato e organização do *prompt* final: o *prompt* deve ser estruturado de forma a facilitar a compreensão do modelo, podendo incluir instruções claras, exemplos ou perguntas específicas;
- Tamanho total do *prompt* em relação ao limite do modelo: o limite de tamanho de entrada do modelo (*context window*) pode limitar a quantidade de informação que pode ser incluída no *prompt*. Estratégias de sumarização ou reescrita podem ser utilizadas a fim de encurtar este tamanho;
- Preservação da coerência: é importante que o prompt seja claro e coerente, evitando ambiguidades que possam levar a respostas incorretas ou irrelevantes.

Técnicas como *few-shot prompting* e *chain-of-thought prompting* são boas práticas de estruturação do *prompt* final. A primeira consiste em fornecer exemplos de perguntas e respostas semelhantes de forma a guiar o modelo a manter coerência e precisão sendo eficaz em domínios específicos com padrões repetitivos (Reynolds e McDonell, 2021). A segunda consiste em incluir instruções para que o modelo "pense passo a passo"de forma a levar a respostas mais completas e precisas (Wei et al., 2022).

Tendo por base o contexto deste projeto, a abordagem *chain-of-thought prompting* apresenta caraterísticas mais adequadas, uma vez que potencia a explicitação dos raciocínios intermédios do modelo, favorecendo a transparência do processo de geração de respostas e contribuindo para a obtenção de resultados mais robustos e fundamentados.

3.1.2.4 RQ4: Quais são os principais desafios técnicos envolvidos na conceção e implementação de uma solução RAG aplicada ao suporte técnico e como os contornar?

A eficiência do RAG depende da capacidade de recuperar documentação relevante. No contexto de suporte técnico, a recuperação necessita ser precisa para fornecer soluções corretas o que é desafiador devido à complexidade e especificidade da documentação (Isaza et al., 2024a).

No que toca recuperação por similaridades semânticas, Soman e Roychowdhury, 2024 refere que o uso de vetores para *chunks* de texto grandes (mais de 200 palavras) resulta em valores de similaridade artificialmente altos. Isso sugere que, mesmo quando as frases não são semanticamente parecidas, o modelo acha que são, apenas por serem longas. Contudo, em documentação com muitas abreviações ou parágrafos extensos por tópico, estas observações tornam-se mais relevantes. Além disso, concluiu-se que palavras-chave mais próximas do início de uma frase são recuperadas com maior precisão.

Adicionalmente, estudos recentes revelam que sistemas RAG apresentam dificuldades significativas quando aplicados em ambientes empresariais. Bruckhaus (2024) demonstram que, mesmo quando a resposta correta está presente no contexto, o sistema frequentemente falha em recuperá-la. Tal acontece, em parte, devido ao desajuste entre a estrutura dos documentos técnicos (como FAQs, procedimentos, *logs*) e as estratégias tradicionais de segmentação em *chunks*.

Esta falha é confirmada por Barnett et al. (2024), que identificaram múltiplos pontos críticos no funcionamento de sistemas RAG, incluindo:

- Falta de conteúdo: Quando a informação necessária não está no contexto, o sistema pode responder com conteúdos enganosos, sugerindo que sabe a resposta mesmo sem dados de apoio.
- Fraca classificação da documentação: Mesmo com a informação presente no contexto, ela pode não ser corretamente classificada e consequentemente não recuperada.
- Informação não extraída: Caso haja informações contraditórias no contexto, o recuperador pode apresentar falhas.
- Especificidade incorreta: O sistema pode gerar respostas muito vagas ou excessivamente específicas sem compreender com precisão o que foi solicitado.

Diversas técnicas de otimização podem ser utilizadas para contornar estas situações. A utilização de contextos maiores pode contribuir para respostas mais precisas, desde que a informação recuperada seja relevante e não exceda o limite de contexto (*context window*) do modelo LLM utilizado, evitando assim a diluição da relevância dos dados apresentados (Bruckhaus, 2024; Soman e Roychowdhury, 2024). A inclusão de metadados, como o nome do ficheiro e número do *chunk*, melhora a interpretação da informação recuperada. Modelos de *embeddings open-source* também se mostram eficazes, especialmente em textos curtos. Para garantir resultados robustos, é essencial calibrar cuidadosamente a *pipeline* RAG, incluindo *chunking*, *embeddings*, recuperação e consolidação — além de manter uma monitorização contínua na injeção de conhecimento, dado que o sistema pode lidar com entradas desconhecidas.

3.2 Tecnologias Frontend

O desenvolvimento *frontend* é responsável pela camada de interação entre o utilizador e a aplicação, englobando a construção da interface e a experiência de utilização. Atualmente, existem diversas bibliotecas e *frameworks* para desenvolvimento *frontend*, cada uma com diferentes filosofias, arquiteturas e conjuntos de funcionalidades. Entre as soluções mais populares encontram-se o React e o Angular, que apresentam abordagens distintas para a construção de *single-page applications* (SPAs).

Nas subseções seguintes são analisadas estas duas tecnologias, destacando as suas principais caraterísticas, vantagens e limitações, culminando numa comparação que permitirá fundamentar a escolha para o presente projeto.

3.2.1 React

Criado pela Meta, o React é uma biblioteca JavaScript *open-source* usada para construir SPAs (React Team, 2025).

Apresenta uma arquitetura baseada em componentes reutilizáveis, responsáveis por gerir a sua própria lógica e renderização. Estes podem ser do tipo funcionais, geridos através de *hooks*, ou de classe, que constituem uma abordagem mais antiga, baseada em métodos de ciclo de vida.

A informação flui através das *props*, que são dados transmitidos de componentes pai para filho, e através do *state*, que corresponde a dados internos que podem mudar ao longo do tempo dentro de um componente. Soluções como o Redux permitem a gestão global e centralizada do *state* (Schmalzried, 1981).

O React introduziu também o JavaScript XML (JSX), uma extensão de sintaxe que possibilita a escrita de código HTML dentro do JavaScript. Isto permite um desenvolvimento mais limpo e centralizado, uma vez que evita a separação do código *markup* e da lógica em ficheiros diferentes.

Document Object Model (DOM) é a representação dos objetos que compõem a estrutura e o conteúdo de um documento na web. O React mantém um DOM virtual, no qual as atualizações são aplicadas de forma eficiente, comparando-o com o DOM real e renderizando novamente apenas as partes que foram alteradas.

Apresenta, no entanto, uma curva de aprendizagem relativamente acentuada, pois o JSX, os *hooks* e a gestão do *state* podem ser confusos. Em grandes aplicações, podem surgir problemas de desempenho quando a atualização do *state* não é devidamente otimizada, originando renderizações desnecessárias (Schmalzried, 1981).

3.2.2 Angular

O Angular é uma *framework open-source* baseada em TypeScript, desenvolvida e mantida pelo Google, destinada à construção de SPAs. Trata-se de uma solução que já inclui diversas funcionalidades integradas de forma nativa, como roteamento, gestão do *state* e manipulação de formulários, oferecendo um conjunto de ferramentas estruturado desde o início, com objetivo de simplificar o fluxo de trabalho e gestão de dependências (Google Angular Team, 2025; Seshadri, 2018).

Apresenta uma arquitetura baseada no Model–View–Controller (MVC), mas sobretudo orientada a componentes. Integra o conceito de módulos, que contêm lógica de componentes relacionados de forma agrupada. Cada componente apresenta a classe (ou modelo), que inclui a lógica escrita em TypeScript, o *template* HTML e os estilos em Cascading Style Sheets (CSS).

Implementa o conceito de *data binding* bidirecional que permite sincronizar automaticamente dados entre o modelo e o *template*. Utiliza também o padrão Dependency Injection (DI) para facilitar a reutilização de serviços, promovendo baixo acoplamento. Além disso, o Angular faz uso extensivo da biblioteca Reactive Extensions for JavaScript (RxJS), adotando a programação reativa para lidar com fluxos de dados assíncronos, tais como chamadas HyperText Transfer Protocol (HTTP) e eventos.

À semelhança do React, apresenta uma curva de aprendizagem acentuada devido ao ecossistema extenso e pode apresentar desafios na manipulação do *state*. Apresenta também uma estrutura inicial pesada pois exige mais código *boilerplate* em comparação com outras tecnologias.

3.2.3 Tabela de comparação

Abaixo segue-se a Tabela 3.6 de comparação de tecnologias *frontend*, de forma a dar suporte à seleção da solução mais adequada para este projeto. A comparação aborda critérios como tipo, filosofia, arquitetura, gestão do *state*, *template*, desempenho e curva de aprendizagem.

Tabela 3.6: Comparação entre tecnologias frontend

Critério	React	Angular
Tipo	Biblioteca JavaScript	*Framework* completa baseada em TypeScript
Filosofia	Aborda uma filosofia de flexibilidade, permitindo a escolha livre de dependência externas para *routing*, gestão do *state*, etc	Aborda uma filosofia de "convenção acima de configuração"sendo uma solução integrada e mais restrita
Arquitetura	Baseada em componentes reutilizáveis; uso de *hooks* ou classes	Baseada em componentes e módulos; segue padrões MVC e DI
Gestão do State	Local (*state* e *props*); soluções externas como Redux	Inclui mecanismos nativos
Template	JSX (mistura HTML com JavaScript)	HTML + TypeScript + CSS em ficheiros separados
Desempenho	DOM virtual para otimizar renderizações	Uso eficiente de *change detection*, mas estrutura inicial mais pesada
Curva de Aprendizagem	Alta: *hooks*, JSX e gestão de *state* podem ser confusos	Alta: ecossistema extenso e mais código *boilerplate*

O React e o Angular são ambas tecnologias sólidas e amplamente utilizadas no desenvolvimento *web*. A escolha entre as duas depende dos requisitos específicos do projeto e da experiência da equipa. React oferece maior flexibilidade e é ideal para projetos que exigem personalização, enquanto Angular proporciona uma solução mais estruturada e integrada, adequada para aplicações empresariais complexas.

3.3 Tecnologias Backend

O *backend* corresponde à camada responsável pela lógica de negócio, processamento de dados e comunicação com a base de dados ou serviços externos.

A escolha da tecnologia de *backend* tem impacto direto na capacidade de manutenção, na velocidade de desenvolvimento e no suporte a requisitos de fiabilidade e interoperabilidade. Por esse motivo, torna-se relevante analisar diferentes linguagens e ecossistemas, de forma a selecionar a solução mais adequada ao contexto do projeto.

Entre as linguagens mais utilizadas encontram-se o Java e o Python, ambas amplamente consolidadas na indústria e suportadas por ecossistemas ricos em bibliotecas e *frameworks*. Nas subseções seguintes são apresentadas as principais caraterísticas de cada tecnologia, seguidas de uma tabela comparativa que sintetiza vantagens, limitações e cenários de utilização.

3.3.1 Java

O Java, criado em 1995 pela Sun Microsystems, é uma das linguagens de programação orientadas a objetos mais utilizadas devido à sua versatilidade e robustez. A filosofia *"write once, run anywhere"* caracteriza a linguagem, permitindo que uma aplicação seja desenvolvida uma vez e executada em qualquer sistema compatível (Bloch, 2008).

Esta portabilidade é garantida pelo compilador do Java Development Kit (JDK), que converte o código em *bytecode*. O *bytecode* é interpretado pela Java Virtual Machine (JVM), integrada no Java Runtime Environment (JRE), disponível em sistemas como Windows, macOS e Linux.

O JDK inclui uma biblioteca padrão extensa, que oferece funcionalidades essenciais como estruturas de dados, manipulação de datas e geração de números aleatórios, reduzindo a dependência de bibliotecas externas e acelerando o desenvolvimento.

Graças à sua escalabilidade, interoperabilidade e estabilidade, o Java é amplamente adotado em contextos empresariais. O seu ecossistema integra *frameworks* como o Spring, amplamente utilizado no desenvolvimento de *APIs* e de aplicações baseadas no padrão Spring MVC, e o Hibernate, uma *framework* de mapeamento objeto-relação que simplifica a persistência de dados (Mythily, Raj e Joseph, 2022). Estas ferramentas facilitam a criação de aplicações seguras, modulares e preparadas para ambientes *cloud*.

A linguagem beneficia ainda de documentação abrangente e de uma comunidade ativa, fatores que contribuem para a sua aprendizagem e consolidação como tecnologia de referência.

3.3.2 Python

O Python, criado em 1991 por Guido van Rossum, é uma linguagem de programação orientada a objetos, de alto nível e com semântica dinâmica. Por ser uma linguagem interpretada, o código é executado linha a linha, dispensando compilação prévia (Kholmatov, 2024).

A linguagem adota tipagem dinâmica, não exigindo a declaração explícita de tipos em variáveis ou funções, o que aumenta a flexibilidade e reduz a extensão do código. A definição de blocos é feita obrigatoriamente por indentação, reforçando a legibilidade e a organização.

Com uma sintaxe simples e próxima da língua inglesa, o Python apresenta baixa curva de aprendizagem. A sua ampla comunidade garante documentação extensa e suporte contínuo, consolidando-o como uma linguagem de referência em diversos domínios.

Frameworks como Flask permitem o desenvolvimento rápido de *APIs* e aplicações *web* robustas, facilitando a criação de serviços escaláveis e mantíveis (Grinberg, 2018).

3.3.3 Tabela de comparação

Abaixo segue-se a Tabela 3.7 onde compara diversos critérios das tecnologias *backend* analisadas nas secções anteriores.

Tabela 3.7: Comparação entre tecnologias backend

Critério	Java	Python
Curva de aprendizagem	Média a alta: sintaxe rigorosa e forte tipagem	Baixa: sintaxe simples e intuitiva
Compilação/Interpretação	Compilado para bytecode e executado na JVM	Interpretado, executado linha a linha
Tipagem	Estática: tipos declarados explicitamente	Dinâmica: tipos inferidos em tempo de execução
Comunidade	Grande, madura e consolidada	Muito grande, ativa e em crescimento
Documentação	Abrangente e detalhada	Abrangente e detalhada
Bibliotecas nativas	Extensa, robusta	Extensa, versátil
Uso típico	Aplicações corporativas, sistemas *web*, *software* empresarial	Sistemas *web*, análise de dados, automação

O Java e o Python são tecnologias bastante robustas e amplamente utilizadas em contextos empresariais, incluindo a integração em sistemas *web*, servindo de base para a aplicação das regras de negócio. O Java destaca-se pela sua tipagem estática e mais rígida, enquanto o Python se distingue pela sua tipagem dinâmica e flexível, em que os tipos são inferidos em tempo de execução. Ambas contam com o suporte de uma grande comunidade e com documentação abrangente.

3.4 Frameworks RAG

Nesta secção serão analisadas em detalhe algumas das *frameworks* de desenvolvimento RAG disponíveis no mercado atualmente, terminando com uma breve comparação e análise crítica.

3.4.1 LangChain

Fundada por Harrison Chase em 2022, LangChain é uma *framework open-source* para o desenvolvimento de aplicações que integram LLMs com fontes externas de dados e ferramentas de *software*. Está disponível em Python e JavaScript, oferecendo um ambiente centralizado e altamente extensível para construir soluções com LLMs de forma modular e reutilizável (IBM, 2024).

A arquitetura do LangChain é baseada no conceito de *chains*, que representam sequências de chamadas a LLMs e a outras ferramentas auxiliares. Estas *chains* podem ser compostas de forma modular, permitindo construir fluxos de execução complexos com facilidade (IBM, 2024; LangChain, 2024b).

Com a introdução do LangChain Expression Language (LCEL), passou a ser possível definir estas sequências de forma declarativa. O LCEL oferece diversos construtores prontos para uso, como (LangChain, 2024b; Silva, 2025):

- *create_stuff_documents_chain*: Formata uma lista de documentos num *prompt* para o LLM.
- *create_sql_query_chain*: Gera consultas SQL a partir linguagem natural.
- *create_history_aware_retrieve*: Utiliza o histórico de conversas para gerar consultas de pesquisa.
- *create_retrieval_chain*: Integra recuperação de documentos relevantes com geração de respostas por LLM.

O LangChain destaca-se pela sua flexibilidade e modularidade, permitindo construir soluções adaptadas a diferentes domínios, desde sistemas de *chat* com memória contextual até agentes capazes de executar raciocínios multi-etapas e integrar-se com APIs externas. A *framework* suporta LLMs de vários fornecedores, incluindo OpenAI, Cohere e modelos *open-source* como LLaMa, o que facilita a adaptação a diferentes necessidades e infraestruturas (LangChain, 2024b).

A arquitetura modular do LangChain permite combinar e substituir componentes de forma livre, tornando-o especialmente versátil para responder a requisitos específicos de negócio. Esta abordagem, no entanto, implica uma curva de aprendizagem algo acentuada, sobretudo devido à diversidade de conceitos e opções disponíveis. Apesar disso, a documentação oficial é bastante completa, com exemplos práticos, tutoriais e guias orientados para os casos de uso mais comuns (LangChain, 2024b).

No que respeita à integração com sistemas de armazenamento, o LangChain oferece suporte nativo a diversas soluções, tanto tradicionais como vetoriais, como Elasticsearch e Weaviate. Estas tecnologias possibilitam a pesquisa vetorial, o armazenamento de *embeddings* e a indexação eficiente de documentos. A *framework* disponibiliza abstrações como *VectorStoreRetriever* e *DocumentLoaders*, que facilitam a configuração e adaptação dos processos de indexação e recuperação de informação, ajustando-os às necessidades de cada aplicação (LangChain, 2024b).

A comunidade LangChain é bastante ativa e tem registado um crescimento significativo, refletido no dinamismo do repositório GitHub, que conta com mais de cem mil estrelas. A equipa responsável pelo projeto tem promovido o desenvolvimento contínuo da *framework*, bem como o lançamento de ferramentas complementares, como o LangSmith, dedicado à monitorização e depuração de aplicações baseadas em LLMs. Esta vitalidade comunitária traduz-se numa documentação frequentemente atualizada, partilha regular de boas práticas e integração de contributos externos, o que reforça a posição do LangChain como uma das principais referências no desenvolvimento de soluções RAG (LangChain, 2024a).

3.4.2 Haystack

Haystack é uma *framework* Python *open-source* desenvolvida pela empresa alemã Deepset, com o objetivo de facilitar a construção de *pipelines* baseadas em LLMs, especialmente para casos de uso como RAG, *question answering*, classificação, extração de informação e pesquisa semântica em documentos. A primeira versão surgiu em 2020, tendo recentemente evoluído para a versão 2.0, com uma reformulação completa da sua arquitetura (deepset, 2024b).

A principal inovação do Haystack 2.0 é a sua arquitetura orientada a componentes. Cada componente representa uma unidade funcional independente, com responsabilidade bem

definida dentro de uma *pipeline*. Estes componentes são combinados de forma declarativa para formar *pipelines* personalizadas, robustas e escaláveis. Entre os tipos de componentes disponíveis encontram-se (deepset, 2024a):

- **Document Stores**: responsáveis por armazenar documentos e suportar tanto índices tradicionais quanto vetoriais.
- **Retrievers**: mecanismos de recuperação de informação, com suporte a métodos tradicionais como BM25 e a recuperação semântica com *embeddings*.
- **Rankers**: permitem reordenar os documentos recuperados com base em critérios de relevância mais refinados.
- **Generators**: utilizam LLMs para gerar respostas com base na informação recolhida.
- **Prompt Nodes**: enviam prompts configuráveis para modelos locais ou remotos.
- **Routers**: introduzem lógica condicional nas *pipelines*, facilitando a criação de fluxos adaptativos.

As *pipelines* podem ser definidas em ficheiros YAML ou diretamente em Python, promovendo flexibilidade tanto para utilizadores técnicos como não técnicos.

A curva de aprendizagem do Haystack pode ser moderada, especialmente para utilizadores que não estejam familiarizados com conceitos como *pipelines* declarativas. No entanto, a documentação oficial é bastante completa e a existência de *pipelines* pré-configuradas — como a *PredefinedPipeline.INDEXING* e a *PredefinedPipeline.RAG* — facilita bastante o início do desenvolvimento. Estas permitem, respetivamente, indexar documentos e realizar tarefas de RAG com configuração mínima (deepset, 2024a).

Haystack oferece suporte nativo a várias tecnologias de armazenamento de documentos, como Elasticsearch e Weaviate. Em ambiente de desenvolvimento, o Haystack oferece o *InMemoryDocumentStore*, uma opção *lightweight* que armazena os documentos diretamente em memória, sendo ideal para testes locais (deepset, 2024a).

Haystack possui uma comunidade ativa. O repositório oficial no GitHub conta com mais de vinte mil estrelas (Haystack, 2024) onde as dúvidas de suporte são frequentemente respondidas e existe entregas regulares. A documentação é extensa e bem estruturada (deepset, 2024a).

3.4.3 Spring AI

Spring AI é uma *framework* recente, disponibilizada publicamente no início de 2024, com o objetivo de simplificar o desenvolvimento de aplicações baseadas em IA ecossistema Java. Inspirado por outros projetos como LangChain, o Spring AI surge como uma extensão natural do ecossistema Spring, promovendo a integração de LLMs em aplicações corporativas de forma modular (Spring Team, 2024).

A arquitetura do Spring AI é orientada a componentes e organizada em torno do conceito de *advisors*, responsáveis por encapsular diferentes estratégias de interação com LLMs e bases de dados vetoriais. A *framework* disponibiliza dois tipos principais de *advisors* para fluxos RAG (Spring Team, 2024):

- **QuestionAnswerAdvisor**: foca-se na recuperação exata de informação a partir de dados externos, construindo respostas exclusivamente com base nos documentos recuperados da base vetorial.
- **RetrievalAugmentationAdvisor**: combina a informação externa com o conhecimento prévio embutido no modelo base, permitindo uma resposta enriquecida e mais flexível.

Apesar destes mecanismos facilitarem a implementação, é igualmente possível interagir diretamente com os modelos através de *prompting* personalizado, sem recorrer a *advisors*, o que garante maior controlo sobre a orquestração das chamadas e a adaptação a cenários específicos.

A comunicação com estes componentes é configurada programaticamente através da Application Programming Interface (API) do Spring AI, possibilitando a definição de parâmetros como o limiar de similaridade na pesquisa.

Adicionalmente, a *framework* estrutura os fluxos de processamento em módulos reutilizáveis, entre os quais se destacam (Spring Team, 2024):

- *Document Reader Modules* - para leitura e preparação de documentos;
- *Embedding Modules* - para geração de vetores a partir de texto;
- *Retriever Modules* - para pesquisa em bases vetoriais;
- *LLM Modules* - para interação com o modelo de linguagem.

Esta abordagem modular facilita a composição e manutenção de *pipelines* RAG personalizadas.

Apesar de ser uma *framework* emergente, o Spring AI apresenta uma estrutura suficientemente flexível para acomodar diversos casos de uso. A separação clara entre módulos permite que cada fase da *pipeline* seja configurada ou substituída conforme os requisitos específicos da aplicação.

Por estar profundamente integrado com o ecossistema Spring, a *framework* beneficia de recursos como injeção de dependências, configuração centralizada, gestão de contexto e integração com outras soluções do universo Spring Boot, o que a torna especialmente atrativa para ambientes corporativos baseados em Java (Spring Team, 2024).

O Spring AI oferece integração nativa com várias bases de dados vetoriais, através da abstração *VectorStore*. Atualmente, estão disponíveis adaptadores oficiais para soluções amplamente utilizadas no mercado, como Qdrant, Milvus e Pinecone. Estes armazenamentos vetoriais podem ser utilizados diretamente nos módulos de recuperação, com suporte para parâmetros como o número de documentos a recuperar e o grau de similaridade exigido (Spring Team, 2024).

Dado o seu lançamento recente, a comunidade do Spring AI ainda se encontra em crescimento. No entanto, beneficia do forte ecossistema da Spring Framework e da extensa base de utilizadores da comunidade Java. A documentação é bem estruturada e dá exemplos práticos para cobrir os principais casos de uso atuais (Spring Team, 2024).

3.4.4 Tabela de comparação

Para resumir, na Tabela 3.8 estão descritos os principais pontos de comparação entre as diferentes tecnologias analisadas.

Tabela 3.8: Comparação entre frameworks RAG

Critério	LangChain	Haystack	Spring AI
Linguagens	Python e JavaScript	Python	Java
Arquitetura	Orquestração modular e declarativa	Arquitetura orientada a componentes	Arquitetura modular: integração com o ecossistema Spring
Flexibilidade	Elevada	Elevada	Moderada a elevada
Curva de aprendizagem	Média	Baixa a média	Baixa, especialmente para programadores Java
Compatibilidade com bases de dados vetoriais	FAISS, In Memory, Qdrant, PGVector, Weaviate, Pinecone	In Memory, Qdrant, PGVector, Weaviate, Pinecone	Milvus, Qdrant, PGVector, Weaviate, Pinecone
Suporte e comunidade	Muito elevados: comunidade ampla e ativa	Moderados a elevados: com foco empresarial	Moderados: em crescimento

Em termos de arquitetura as tecnologias são semelhantes em alguns principios gerais, mas diferem na forma como organizam os componentes. A LangChain usa o conceito de *chains* que permite encadear múltiplos passos de forma declarativa. Já a Haystack possui um arquitetura orientada a componentes reutilizáveis. No Spring AI a arquitetura é modular e integrada no ecossistema Spring Boot com foco para desenvolvedores Java, usando abstrações típicas do Spring.

No que diz respeito à flexibilidade, LangChain e Haystack oferecem maior capacidade de personalização, com suporte a múltiplas configurações e integrações. O Spring AI, apesar de mais recente, apresenta uma modularidade sólida.

Em termos de curva de aprendizagem, o Spring AI tende a ser mais simples para programadores familiarizados com o ecossistema Spring e Java. A Haystack combina flexibilidade com *pipelines* pré-configuradas, acelerando o processo de aprendizagem. A LangChain, contudo, exige um maior domínio técnico.

Todas as ferramentas analisadas oferecem integração com sistemas de armazenamento vetorial da atualidade. O suporte é robusto e adaptável em todas, com destaque para a abstração VectorStore no Spring AI e a diversidade de conectores nativos no Haystack.

No que respeita a suporte e comunidade, LangChain e Haystack apresentam ecossistemas consolidados, com documentação extensa, comunidades ativas e evolução contínua. O Spring AI, embora ainda em fase de maturação, conta com documentação estruturada e o respaldo da comunidade Spring, o que lhe confere um nível moderado, indicando um bom ponto de partida com potencial de crescimento acelerado.

No contexto deste projeto, optou-se pela utilização do Spring AI, uma vez que esta *framework* se adequa melhor ao conhecimento técnico da equipa B2C, por esta ser baseada em Java e, consequentemente, facilitar a integração em projetos já existentes.

3.5 Sistemas Externos

Nesta secção serão analisados dois sistemas externos relevantes ao presente projeto: o Confluence e o Control-M. A avaliação destes sistemas centra-se na capacidade de aceder, estruturar e enriquecer a informação disponível, permitindo a sua integração com a solução RAG a conceber. Nas subseções seguintes apresentam-se as principais caraterísticas, métodos de acesso e estratégias de integração de cada sistema, de forma a fundamentar a conceção da solução.

3.5.1 Confluence

Desenvolvido pela Atlassian, o Confluence é uma plataforma amplamente utilizada para gestão de conhecimento e documentação empresarial. Permite criar, organizar e partilhar páginas de forma colaborativa, funcionando como uma Wiki (Atlassian, 2024b).

Atualmente esta ferramenta disponibiliza a Confluence Cloud Representational State Transfer (REST) API, que permite aceder a páginas e seus respetivos conteúdos através de uma interface RESTful (Atlassian, 2024a).

Para sistemas RAG, existem soluções mais sofisticadas, como o ConfluenceLoader da LangChain, que fornece uma abstração de acesso à API do Confluence, facilitando a integração com as *pipelines* de indexação da informação (Horvat, s.d.). Para soluções como Spring AI, este nível de integração nativa ainda não existe, o que implica a necessidade de desenvolvimento de um módulo personalizado para acesso à API e extração da informação necessária.

3.5.2 Control-M

O Control-M é uma ferramenta de automação de fluxos e gestão de *jobs*, amplamente utilizada em ambientes empresariais para agendar, monitorizar e gerir tarefas (BMC Software, 2024b). Atualmente existem diversas formas de aceder a informação histórica de execução de *jobs* no Control-M, como (BMC Software, 2024a,b):

- Control-M Manager: talvez a forma mais comum de acesso a informação histórica de execução dos *jobs*. Permite fácil visualização por operadores e não requer muitos conhecimentos técnicos. Contudo, não é uma interface programática, o que impossibilita a sua integração direta com sistemas RAG.
- Control-M Automation API: interface RESTful que permite consultar execução dos *jobs*, *logs* e estados. Permite acesso programático sendo muito fácil a sua integração com sistemas RAG.
- Control-M Command Line Interface (CLI): permite acesso a informação histórica através de comandos no terminal. É ideal para automatização via scripts e fácil de integrar com Continuous Integration / Continuous Delivery (CI/CD).
- Base de dados interna do Control-M: disponibiliza acesso completo aos dados brutos históricos e oferece alta flexibilidade para consultas complexas por se basear em SQL.

A escolha entre estas opções depende dos requisitos específicos do sistema e também, claro, da infraestrutura disponível e autorizada pela organização. Neste caso em específico, a Purple opta por utilizar a base de dados interna do Control-M por oferecer uma

grande flexibilidade, sendo esta solução vastamente utilizada por outros sistemas internos da empresa.

Na fase de indexação, recomenda-se a estruturação e enriquecimento dos dados antes do armazenamento, em detrimento de uma abordagem de dados brutos. Para sistemas que realizam pesquisa vetorial semântica com o objetivo de obter respostas mais relevantes, a estruturação dos dados em linguagem natural constitui uma vantagem significativa. De acordo com Ali, 2025, relatórios e documentos estruturados facilitam a divisão em *chunks* e a geração de *embeddings* de maior qualidade, assegurando coerência semântica. Adicionalmente, dados estruturados permitem auditoria e versionamento, aspetos essenciais em ambientes empresariais. O uso de relatórios em linguagem natural contribui ainda para a redução dos custos de processamento e para a melhoria dos tempos de execução (Ali, 2025; Murtaza et al., 2025).

3.6 Métricas de Avaliação

A avaliação de tarefas de geração PLN, exige a comparação entre uma resposta gerada pelo sistema (*candidate*) e uma resposta de referência considerada correta (*reference*). Neste contexto, as métricas procuram quantificar o grau de similaridade entre os dois textos, seja ao nível lexical ou semântico.

Grande parte das métricas partilha a lógica de cálculo baseada em precisão, recall e F1:

- **Precisão**: proporção de unidades do texto candidato que coincidem com a referência.
- **Recall**: proporção de unidades da referência que são recuperadas pelo candidato.
- **F1**: média harmónica entre precisão e recall, equilibrando ambos os fatores.

Estas medidas servem de base para métricas mais sofisticadas, como o Recall-Oriented Understudy for Gisting Evaluation (ROUGE) ou BERTScore. Mais recentemente, surgiram também abordagens LLM-based, mas que não utilizam diretamente estas medidas.

3.6.1 ROUGE

O ROUGE (Recall-Oriented Understudy for Gisting Evaluation) representa uma série de métricas usadas para a avaliação de sumarização automática e PLN. Estas medem a sobreposição lexical de n-gramas entre o texto candidato e o texto de referência. Têm a vantagem de ser simples e consumir poucos recursos, sendo indicadas quando a coincidência literal de palavras é importante. Por outro lado, ignoram a semelhança semântica.

Os principais tipos de ROUGE incluem:

- **ROUGE-N**: mede a sobreposição de n-gramas, tipicamente unigrama (ROUGE-1) ou bigrama (ROUGE-2).
- **ROUGE-L**: avalia a maior subsequência comum, captando a ordem das palavras.
- **ROUGE-W**: uma variação ponderada do ROUGE-L, que penaliza interrupções na subsequência comum.
- **ROUGE-S**: considera pares de palavras que mantêm a mesma ordem, mesmo que não estejam adjacentes.

3.6.2 BERTScore

O BERTScore tem semelhanças com o ROUGE, mas em vez de usar comparação lexical, usa comparação entre *embeddings*. Tem a vantagem de ser mais robusto por captar a semântica das palavras. Por outro lado, é mais pesado computacionalmente e não avalia diretamente a consistência factual, que pode resultar em pontuações elevadas a alucinações.

3.6.3 LLM-based

O método LLM-based consiste na utilização de um LLM de *chat* como avaliador. Através da elaboração de um *prompt* com instruções claras, o LLM devolve uma classificação baseada numa escala previamente definida. Esta abordagem permite avaliar de forma eficiente a coerência e a factualidade entre a referência e o candidato. Além disso, é bastante flexível, uma vez que se adapta aos critérios de estilo e de factualidade pretendidos, o que o torna mais próximo do julgamento humano.

Por outro lado, apresenta desvantagens, como o maior custo computacional e alguma inconsistência nos resultados, que podem variar consoante o LLM escolhido.

3.7 Trabalhos Relacionados

Os trabalhos relacionados apresentados nesta secção evidenciam a aplicação de arquiteturas RAG em diferentes cenários de suporte técnico. Estes estudos permitem compreender como a combinação de recuperação de dados com LLMs é benéfica em contextos reais, destacando as decisões arquiteturais mais eficazes e os benefícios de adaptar os modelos ao domínio específico.

3.7.1 RAG Chatbot para a OptiMicro Technologies

A OptiMicro, empresa de *software* Canadiana, decidiu apostar no desenvolvimento de um chatbot utilizando RAG para melhorar o seu serviço de apoio técnico da empresa, mais especificamente no seu serviço DentalWare (H.-C. Lee et al., 2024).

O *software* foi desenvolvido através de uma plataforma *low-code* denominada Flowise AI e a sua arquitetura consistiu em três fases principais: indexação, recuperação e geração. Na fase de indexação, foram utilizados registos de suporte técnico e manuais do *software* DentalWare, segmentados em *chunks* de texto. Os dados foram convertidos em vetores através do modelo all-MiniLM-L6-v2, acedido via API da HuggingFace, e armazenados numa base de dados vetorial Pinecone, escolhida pelo seu elevado desempenho. Na etapa de recuperação, a *query* do utilizador é vetorizada com o mesmo modelo e comparada com os vetores indexados, sendo selecionados os segmentos mais relevantes. Por fim, a resposta é gerada com o modelo Large Language Model Meta AI (LLaMA) 2 (7B), executado localmente via Ollama.

Esta arquitetura encontra-se descrita no diagrama da Figura 3.4, onde é possível verificar de que forma os componentes se relacionam. É de notar que foram utilizados dois documentos como fonte de informação externa definidos de forma estática, tornando o *software* um pouco restrito.

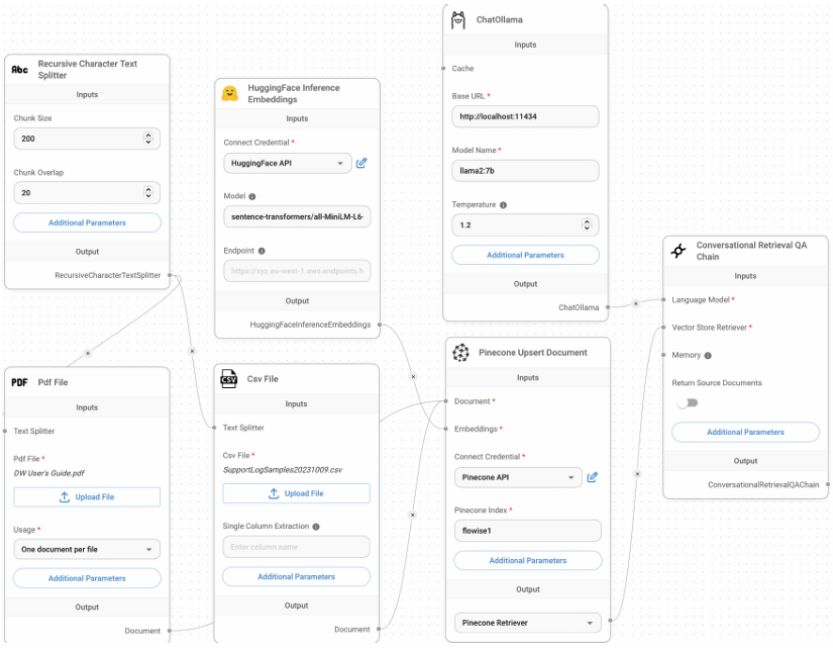


Figura 3.4: Arquitetura do chatbot RAG da OptiMicro (H.-C. Lee et al., 2024)

Para avaliar o desempenho do chatbot, foram utilizadas 75 questões reais relacionadas com suporte técnico ao *software* DentalWare. As respostas geradas por este chatbot foram comparadas com as de um chatbot LLM genérico, tendo ambas sido avaliadas com base nos critérios de ROUGE e por humanos.

Os resultados demonstraram que o chatbot RAG obteve melhores pontuações nas métricas ROUGE-1, ROUGE-2 e ROUGE-L com aumentos de 38%, 188% e 40%, respetivamente, face ao modelo genérico. A avaliação humana também destacou a clara vantagem do RAG.

Estes resultados confirmam a superioridade do modelo RAG no contexto de apoio técnico, pela sua capacidade de gerar respostas mais relevantes e alinhadas com os dados específicos da empresa.

Contudo, é de notar a limitação do sistema da DentalWare na fase de indexação, uma vez que não permite a integração dinâmica de nova documentação, ao contrário do que se prevê no presente trabalho.

3.7.2 Sistema RAG de Recomendação para Suporte Técnico

Isaza et al., 2024b propuseram um sistema de recomendação para resolução de incidentes em suporte técnico baseado em RAG, com o objetivo de sugerir soluções fundamentadas em documentação.

A arquitetura do sistema encontra-se representada na Figura 3.5, sendo composta por cinco etapas principais. A primeira é o pré-processamento, onde certas caraterísticas do incidente são normalizadas e comparadas com siglas ou nomes alternativos, de forma a melhorar a *query* de recuperação. Segue-se a classificação de *tickets*, através do classificador transformer IBM Slate 125m, onde o assunto e a descrição do incidente são analisados para

determinar se a sua resolução é viável apenas com a informação fornecida — casos denominados *single-turn*. Caso o *ticket* não seja classificado como *single-turn*, a *pipeline* termina imediatamente. As etapas seguintes consistem na geração de uma *query* concisa a partir do conteúdo do *ticket*, seguida da recuperação com *reranking*, onde são selecionados os três documentos mais relevantes a partir de uma base de dados vetorial Milvus. A resposta final é gerada com base nesses documentos e é acompanhada de caminhos para as fontes consultadas.

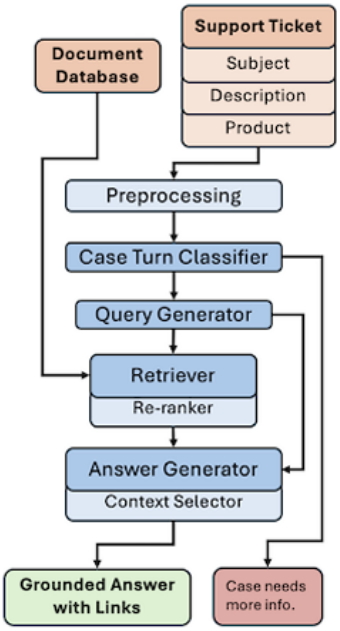


Figura 3.5: Arquitetura do sistema de recomendação de resolução de incidentes proposto por Isaza et al., 2024b

O sistema foi testado em diversos aspetos, quer na geração de *queries* ou quer na geração da respostas, tendo sido utilizados diferentes LLMs para cada vertente. O modelo Mixtral 8x7B Instruct, que é um LLM *fine-tuned* para interações dialogadas como *chatbots*, obteve um BERTScore F1 de 0,91 na tarefa de normalizar incidentes em *queries* concisas, semelhante a LLMs maiores como Falcon-40B, porém com menor custo computacional. Na geração de respostas, este mesmo modelo, combinado com RAG, superou o GPT-4o em métricas como ROUGE-L F1 (0,41 vs. 0,34) e BERTScore F1 (0,87 vs. 0,86), destacando a eficiência de modelos menores com contexto recuperado. Os autores destacaram também que modelos *fine-tuned* com infusão de conhecimento técnico, como o modelo InstructLab-IT, podem superar modelos de grande porte para casos gerais.

3.7.3 Conclusões

Com base na análise dos trabalhos relacionados, é possível concluir que arquiteturas RAG se mostram especialmente eficazes em contextos de suporte técnico, por permitirem respostas mais relevantes e alinhadas com dados específicos da organização. Observa-se também que a escolha adequada dos componentes (modelo de *embedding*, base vetorial, LLM) e o pré-processamento das *queries* influenciam diretamente o desempenho. Além disso, modelos de menor porte, quando ajustados ao domínio, podem superar LLMs genéricos maiores, oferecendo melhores resultados com menor custo computacional.

Capítulo 4

Análise

A fase de análise assume um papel central no ciclo de desenvolvimento de sistemas, uma vez que estabelece as bases conceptuais e técnicas que orientarão as etapas subsequentes de desenho, implementação e validação. É neste momento que se procura compreender, de forma estruturada, tanto o problema em estudo como o contexto em que a solução irá operar, identificando necessidades, constrangimentos e oportunidades de melhoria.

No âmbito deste projeto, a análise incide sobre a definição da solução a implementar, a caracterização do seu modelo de domínio e a sistematização dos requisitos funcionais e não funcionais. Para além disso, são descritos os principais casos de uso que traduzem os cenários de interação previstos com o sistema. Estes elementos permitem alinhar a visão tecnológica com as necessidades reais da equipa B2C, assegurando que a solução proposta não só responde aos problemas identificados, como também acrescenta valor às operações de suporte.

Desta forma, este capítulo visa estabelecer um entendimento partilhado entre os diferentes *stakeholders*, promovendo clareza, consistência e rastreabilidade ao longo do projeto. Ao mesmo tempo, procura-se adotar uma abordagem equilibrada entre rigor técnico e aplicabilidade prática, garantindo que a solução concebida é exequível, sustentável e capaz de se integrar no ecossistema tecnológico existente.

4.1 Solução

Este projeto visa desenvolver uma solução que otimize e agilize as atividades de suporte, focando-se, nesta fase, em duas ferramentas essenciais para a equipa: Confluence e Control-M. O projeto contempla o desenvolvimento dos seguintes artefactos:

- Sistema RAG: Aplicação web protótipo que permita à equipa B2C realizar consultas assistidas por inteligência artificial à documentação do Confluence. O sistema deverá possibilitar a configuração de parâmetros de pesquisa, monitorização das páginas relevantes e sincronização automática face a alterações.
- Processo de extração de dados históricos do Control-M: Leitura dos dados do Control-M, geração de relatórios estruturados e integração desses relatórios no Confluence, centralizando a documentação e tornando-a acessível a consultas assistidas por IA.

Esta solução não só apoiará as atividades de suporte, como também permitirá que os restantes *stakeholders* internos da B2C acedam à documentação de forma mais eficiente e centralizada.

4.2 Modelo de Domínio

Na Figura 4.1 apresenta-se o modelo de domínio da solução.

O sistema assenta na manipulação de páginas do Confluence (classe *Pagina*), que se podem especializar em dois tipos:

- **Relatório Control-M**: páginas criadas automaticamente pelo sistema de extração, contendo relatórios com dados históricos;
- **Documentação de Suporte**: páginas criadas ou geridas por utilizadores, destinadas a documentação de suporte geral.

Além das páginas do Confluence, o modelo contempla a *Página RAG*, que representa a versão indexada de uma página no sistema RAG. Esta entidade está diretamente associada ao ciclo de vida da informação indexada, possuindo sempre um *Estado*, que pode ser *INDEXADA* ou *EXCLUIDA*.

A criação, alteração ou remoção de páginas no Confluence origina automaticamente a geração de um *Evento*. Este evento é responsável por acionar a atualização da respetiva *Página RAG*, garantindo que o sistema RAG mantém a informação sincronizada e atualizada.

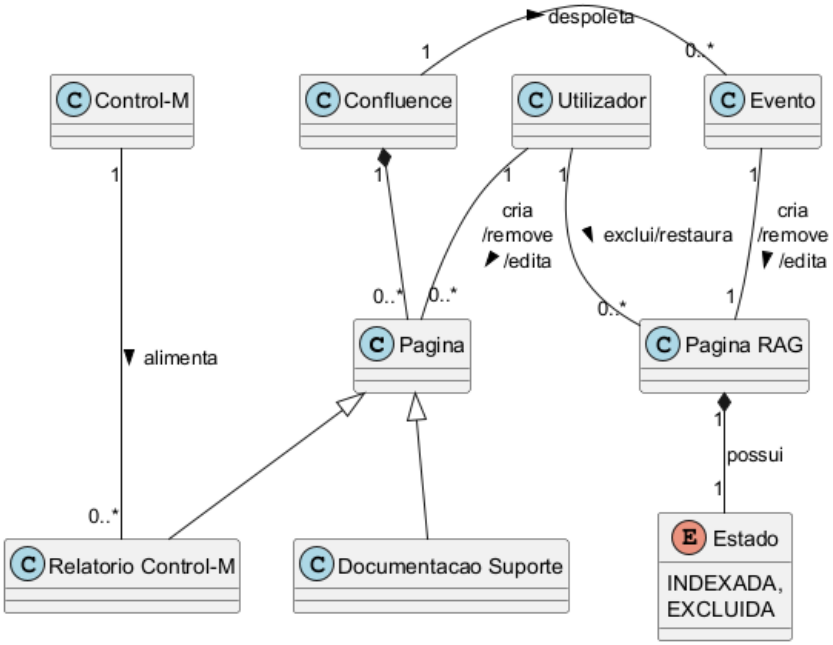


Figura 4.1: Modelo de domínio

4.3 Engenharia de Requisitos

O processo de engenharia de requisitos consiste na identificação, definição, documentação e gestão dos requisitos associados ao desenvolvimento de um sistema de *software* ou projeto. Este processo permite compreender de forma clara as necessidades do cliente, analisar possíveis soluções e, quando necessário, negociar alternativas viáveis. Adicionalmente, os requisitos definidos servem de base para a posterior validação do sistema (De Lucia, Qusef et al., 2010).

Os requisitos podem ser classificados em duas categorias principais: funcionais e não funcionais. Os requisitos funcionais descrevem o comportamento do sistema, especificando as funcionalidades que este deve oferecer, bem como as condições de entrada e os resultados esperados. Por outro lado, os requisitos não funcionais referem-se a restrições relativas à qualidade do sistema, abrangendo aspetos como desempenho, usabilidade, segurança, fiabilidade e manutenibilidade. Para a especificação dos requisitos não funcionais no âmbito deste projeto, foi adotado o modelo FURPS+, que organiza essas dimensões de forma sistemática (De Lucia, Qusef et al., 2010; Suman e Rohtak, 2014).

O levantamento dos requisitos foi realizado com base em interações diretas com os *stakeholders* do projeto, tendo por base as necessidades e processos atuais presentes na equipa. A secção seguinte apresenta a descrição detalhada dos requisitos identificados.

4.3.1 Requisitos Funcionais (RF)

Procede-se à apresentação dos requisitos funcionais definidos juntamente com os *stakeholders* da Purple. Estes requísitios foram subdivididos em diferentes categorias de funcionalidade: sincronização com o Confluence (Tabela 4.1), recuperação assistida (Tabela 4.2), gestão do conhecimento (Tabela 4.3) e integração de dados históricos do Control-M (Tabela 4.4). Utilizou-se a técnica de prioritização MoSCoW associada a cada requisito (Clegg e Barker, 1994).

Tabela 4.1: Requisitos funcionais da sincronização com o Confluence

ID	Título	Descrição	Prioridade
1.1	Mecanismo de sincronização / fase de indexação	Deverá existir um mecanismo de sincronização automática entre a informação disponível no Confluence e a base de conhecimento do sistema.	Must have
1.2	Criação de página	O sistema deve captar eventos de criação de páginas. Quando uma página é criada no Confluence, deverá ser automaticamente indexada no sistema RAG. A partir desse momento ficará elegível para consultas.	Must have
1.3	Edição de página	O sistema deve captar eventos de edição de páginas. Quando uma página é editada no Confluence, deverá ser automaticamente reindexada no sistema RAG. A partir desse momento ficará elegível para consultas com o seu conteúdo atualizado.	Must have
1.4	Remoção de página	O sistema deve captar eventos de remoção de páginas. Quando uma página é removida no Confluence, a mesma deverá também ser removida do sistema por completo. A partir desse momento deixará de estar elegível para consultas	Must have

Tabela 4.2: Requisitos funcionais da recuperação assistida por IA

ID	Título	Descrição	Prioridade
2.1	Motor de pesquisa	O sistema deverá implementar um mecanismo de recuperação assistida por inteligência artificial para consultas de documentação do Confluence. Deverá existir uma interface onde o utilizador consiga especificar uma query de input em linguagem natural e o sistema deverá recuperar informação relevante do conhecimento para fornecer uma resposta adequada.	Must have
2.2	Configuração do motor de pesquisa	O sistema deverá permitir ao utilizador configurar: grau de similaridade, categorias documentais (p.e. procedimentos de suporte, funcionalidades) e *top-k chunks*	Should have

Tabela 4.3: Requisitos funcionais do sistema de gestão do conhecimento

ID	Título	Descrição	Prioridade
3.1	Visualização de páginas	O sistema deverá implementar uma interface gráfica que permita ao utilizador visualizar informação das páginas do sistema e alguns dos seus metadados mais relevantes, como id, título, categoria, data de criação, data de edição e estado.	Could have
3.2	Pesquisa e filtro de páginas	O sistema deverá implementar um mecanismo de pesquisa de páginas e aplicação de filtros de forma a otimizar o acesso à informação	Could have
3.3	Exclusão de páginas	O sistema deverá permitir a exclusão de um página. A exclusão apenas afetará o sistema RAG e não o Confluence. Quando uma página é excluída, deixa de estar elegível para consultas.	Could have
3.4	Restauração de páginas	O sistema deverá permitir a restauração de uma página excluída. Quando uma página é restaurada, passa a estar elegível para consultas.	Could have

Tabela 4.4: Requísitos funcionais para a integração de dados históricos do Control-M

ID	Título	Descrição	Prioridade
4.1	Extração dos dados	Deverá existir um processo agendado de extração de dados históricos do Control-M. Esta extração incidirá sobre *jobs* falhados, devendo ser extraídos metadados relevantes como: pasta, nome, id, dia e intervalo de tempo de execução. Deve ocorrer semanalmente e refletir os dados dos últimos 7 dias.	Should have
4.2	Escrita dos dados	Os dados extraídos devem ser registados numa página do Confluence, dentro de uma secção dedicada a relatórios. O formato de apresentação deve seguir um padrão em tabelas com colunas fixas.	Should have

4.3.2 Requisitos Não Funcionais (RNF)

Seguem-se os requisítos não funcionais definidos baseados no modelo FURPS+ (Suman e Rohtak, 2014). Para este projeto apenas foram definidos requisitos não funcionais para funcionalidade, confiabilidade, desempenho e restrições de desenho, tal como representado na Tabela **??**.

Tabela 4.5: Requisitos não funcionais

ID	DESCRIÇÃO
	Funcionalidade
1.1	O tempo entre a ocorrência de um evento de criação, edição ou remoção de página no Confluence e a respetiva atualização na base de conhecimento do sistema não deverá exceder 60 segundos
1.2	O sistema RAG deve registar operações críticas (exclusão/restauração de documentos, sincronizações) para fins de auditoria
1.3	O sistema RAG deverá respeitar os limites do espaço do Confluence que a equipa B2C tem acesso
	Confiabilidade
2.1	O sistema RAG deve estar disponível 99,5% do tempo durante o horário laboral. Em caso de falha, a recuperação automática deve ocorrer em menos de 10 minutos
	Desempenho
3.1	O sistema RAG deve suportar, no mínimo, 20 utilizadores em simultâneo sem degradação perceptível de desempenho
	Restrições de Desenho
4.1	O sistema RAG deve integrar-se com o Confluence, bases de dados relacional, base de dados vetorial e serviço de IA.
4.2	O sistema de extração de dados do Control-M deve integrar-se com o Confluence e base de dados relacional interna do Control-M

4.3.3 Casos de Uso

Com base nos requisitos funcionais acima descritos, foram elaborados alguns casos de uso, descritos no diagrama de casos de uso da Figura 4.2.

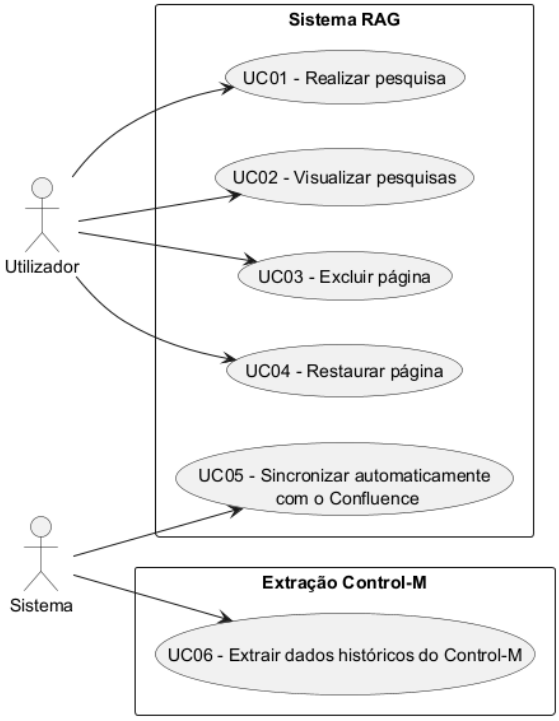


Figura 4.2: Diagrama de casos de uso

As seguintes tabelas apresentam cada Caso de Uso (UC) com mais detalhe, descrevendo pré-condições, pós-condições, fluxos principais e alternativos, entre outras caraterísticas.

Tabela 4.6: UC01 – Realizar pesquisa

ID	UC01
Nome	Realizar pesquisa
Ator	Utilizador da Equipa de Suporte
Objetivo	Permitir ao utilizador pesquisar informações relevantes na base de conhecimento com apoio de IA
Pré-condições	1. Base de conhecimento sincronizada.
Fluxo Principal	1. Utilizador acede à interface de pesquisa; 2. Utilizador introduz *query* em linguagem natural; 3. Utilizador opcionalmente define parâmetros de pesquisa (similaridade, categorias de documentos, *top-k*); 4. Sistema executa recuperação e exibe respostas com fontes.
Fluxos Alternativos	3a. Se não definir parâmetros, usa valores por defeito; 4a. Se não houver resultados, é apresentada mensagem informativa.
Pós-condições	Resposta com links e documentos relevantes é apresentada
RF ligados	2.1, 2.2

Tabela 4.7: UC02 – Visualizar páginas

ID	UC02
Nome	Visualizar páginas
Ator	Utilizador da Equipa de Suporte
Objetivo	Permitir ao utilizador visualizar numa interface gráfica as páginas do sistema
Pré-condições	N/A
Fluxo Principal	1. Utilizador acede à interface de visualização; 2. O sistema apresenta todas as páginas indexadas e excluídas; 3. Utilizador pode pesquisar e/ou aplicar filtros (por nome, data, etc.); 4. Sistema atualiza a listagem com base nos critérios definidos.
Fluxos Alternativos	2a/4a. Se não forem encontradas páginas o sistema deve comunicar ao utilizador.
Pós-condições	Lista de páginas apresentada ao utilizador com sucesso
RF ligados	3.1, 3.2

Tabela 4.8: UC03 – Excluir página

ID	UC03
Nome	Excluir página
Ator	Utilizador da Equipa de Suporte
Objetivo	Permitir ao utilizador excluir uma página indexada
Pré-condições	1. A página deve estar indexada; 2. A página deve existir no Confluence.
Fluxo Principal	1. O utilizador acede à interface de visualização das páginas indexadas; 2. O utilizador despoleta a ação de exclusão de uma página; 3. O sistema pede confirmação; 4. O utilizador confirma a exclusão; 5. O sistema remove a indexação da página e move-a para a categoria de excluídas.
Fluxos Alternativos	3a. O utilizador não confirma a exclusão: caso de uso termina.
Pós-condições	1. A página é excluída de forma *soft-delete*, ficando disponível para restauração no futuro; 2. A página deixa de ser considerada pelo mecanismo de recuperação.
RF ligados	3.3

Tabela 4.9: UC04 – Restaurar página

ID	UC04
Nome	Restaurar página
Ator	Utilizador da Equipa de Suporte
Objetivo	Permitir ao utilizador restaurar uma página excluída
Pré-condições	1. A página deve estar excluída; 2. A página deve existir no Confluence.
Fluxo Principal	1. O utilizador acede à interface de visualização das páginas excluídas; 2. O utilizador despoleta a ação de restaurar uma página; 3. O sistema faz a reindexação da página e move-a para a categoria de indexadas.
Fluxos Alternativos	N/A
Pós-condições	1. A página é reindexada e movida para a categoria de indexadas; 2. A página passa a ser tida em conta pelo mecanismo de recuperação.
RF ligados	3.4

Tabela 4.10: UC05 – Sincronizar automaticamente com o Confluence

ID	UC05
Nome	Sincronizar automaticamente com o Confluence
Ator	Sistema
Objetivo	Atualizar automaticamente a base de conhecimento a partir do espaço definido no Confluence
Pré-condições	1. Conexão com a API do Confluence.
Fluxo Principal	1. Uma página do Confluence é criada, editada ou removida, despoletando um evento; 2. O sistema inicia o processo de sincronização da página; 3. Base de conhecimento é atualizada.
Fluxos Alternativos	2a. Falha na receção do evento: o sistema audita o erro; 2b. Credenciais da API expirados ou API indisponível: o sistema audita o erro; 2c. Receção de evento de documento editado e o mesmo pertencer à categoria de documentos excluídos: o sistema ignora e cancela o processo de sincronização.
Pós-condições	Base de conhecimento sincronizada com o Confluence
RF ligados	1.1, 1.2, 1.3, 1.4

Tabela 4.11: UC06 – Extração de dados históricos do Control-M

ID	UC06
Nome	Extrair dados históricos do Control-M
Ator	Sistema
Objetivo	Extrair automaticamente dados históricos de *jobs* falhados do Control-M para fins de registo e recuperação posterior pelo sistema RAG
Pré-condições	1. Conexão com a base de dados do Control-M; 2. Conexão com a API do Confluence.
Fluxo Principal	1. O sistema inicia automaticamente o processo semanal de extração; 2. O sistema identifica os *jobs* falhados dos últimos 7 dias e extrai os metadados relevantes; 3. O sistema formata os dados em tabelas de colunas fixas e regista-os numa página do Confluence.
Fluxos Alternativos	2a. Não existem jobs falhados: o sistema indica no relatório que não há dados.
Pós-condições	Página com o relatório criada no Confluence
RF ligados	4.1, 4.2

Capítulo 5

Desenho

O presente capítulo tem como objetivo detalhar o desenho da solução proposta, articulando de forma clara e estruturada a arquitetura do sistema com base nos requisitos previamente definidos. Esta fase constitui um ponto de convergência entre a análise de necessidades e a implementação técnica, permitindo não só uma visão global do sistema, mas também a compreensão dos seus componentes, interações e fluxos de dados.

Ao longo deste capítulo, serão apresentados diagramas e representações que ilustram diferentes níveis de detalhe da arquitetura, desde a visão de contexto até aos componentes internos e à implantação física dos sistemas. Serão igualmente discutidas as principais decisões de desenho, incluindo escolhas tecnológicas, padrões arquiteturais adotados e mecanismos de comunicação entre sistemas.

A abordagem escolhida visa fornecer uma referência consistente para a implementação, assegurando que todos os elementos da solução estão alinhados com os objetivos funcionais e não funcionais definidos, promovendo a manutenção futura, a escalabilidade e a eficiência operacional.

5.1 Modelo Arquitetural

Para garantir uma representação clara e consistente, foi adotado o modelo C4 como base para a documentação arquitetural (Brown, 2013). Este modelo permite a descrição do sistema em diferentes níveis de abstração, facilitando a compreensão por parte de diferentes stakeholders.

O modelo C4 define 4 níveis de granularidade: contexto, containers, componentes e código. Os diagramas de contexto apresentam a menor granularidade e constituem um bom ponto de partida para obter uma visão global do sistema, da interação dos utilizadores e dos sistemas externos. Já os diagramas de containers detalham a organização interna de cada sistema, mostrando os principais containers (aplicações, bases de dados) e como estes comunicam entre si. Os diagramas de componentes aprofundam ainda mais o detalhe, representando a decomposição interna de cada container. Neles são identificados os principais componentes de *software*, as respetivas responsabilidades e as relações entre si, permitindo compreender como a lógica de negócio e os serviços se organizam internamente. Por fim, os diagramas de código correspondem ao nível de maior granularidade e descrevem a implementação real, tipicamente em termos de classes, métodos e outros elementos do código-fonte. Este nível é considerado opcional no modelo C4 e, neste documento, será apenas detalhado no Anexo D, uma vez que é mais frequentemente gerado de forma automática pelos Integrated Development Environments (IDE) .

Nos diagramas C4 apresentados neste capítulo, consideraram-se externos os sistemas sobre os quais a B2C não tem influência direta.

Para complementar aos diagramas C4, serão apresentadas duas vistas fora do âmbito deste modelo: diagramas de implantação e diagramas de sequência. Os primeiros são úteis para ilustrar a arquitetura física do sistema, já os segundos servem para tornar mais clara a interação entres os diferentes componentes para cada funcionalidade.

5.2 Escolhas Tecnológicas

A seleção das tecnologias utilizadas neste projeto resultou de uma análise cuidada do estado da arte, bem como da consideração prática do conhecimento técnico existente na equipa e da necessidade de integração com os sistemas já implementados na Purple. Tecnologias emergentes, como no caso da base de dados vetorial, foram incorporadas pela sua relevância para os objetivos da solução, enquanto as opções relativas ao *frontend*, *backend* e bases de dados relacionais refletem simultaneamente boas práticas de mercado e a experiência da equipa de desenvolvimento, assegurando fácil manutenção futura.

- *Frontend* (SPA): React;
- *Backend* (API): Spring AI;
- Base de dados vetorial: Qdrant;
- Base de dados relacional: Microsoft SQL Server;
- Extração de dados do Control-M: Java e Oracle Database.

5.3 Contexto C4

A solução contempla o desenvolvimento de funcionalidades em dois sistemas distintos: o *B2C Core*, neste caso responsável pela extração automatizada de dados históricos do Control-M, e o sistema RAG protótipo (Figura 5.1). Ambos interagem através do Confluence, onde o sistema RAG tem acesso aos conteúdos documentais publicados em torno dos projetos da equipa e o *B2C Core* que é responsável por publicar os relatórios do Control-M, que ficam posteriormente elegíveis para recuperação.

A leitura dos dados históricos é feita através da base de dados interna Oracle deste sistema, sendo o próprio Control-M responsável por iniciar o processo de extração no horário agendado. Considerou-se adequada a atribuição da responsabilidade de extração a este sistema, uma vez que a sua funcionalidade principal é a gestão de fluxos e, portanto, está mais apto a gerir processos recorrentes.

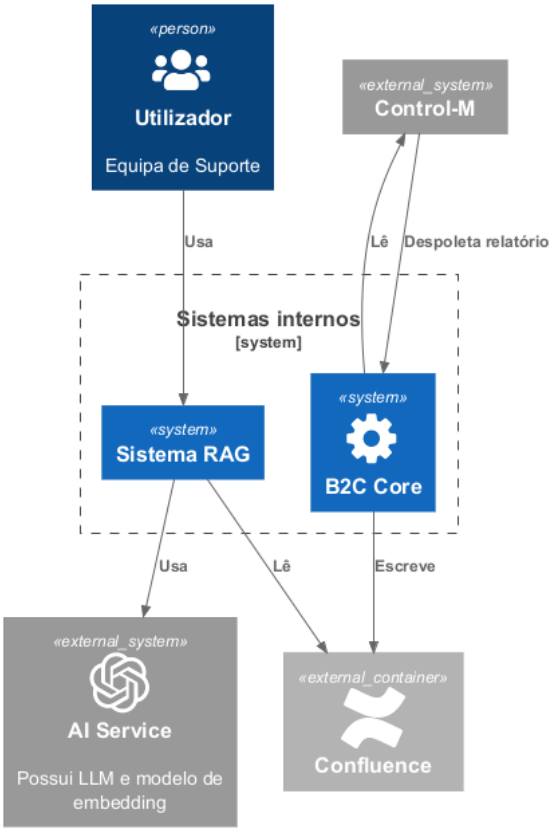


Figura 5.1: Diagrama de contexto C4

Por fim, o sistema RAG é responsável por aceder a um sistema de inteligência artificial, que integra um modelo LLM de *chat* e um modelo de *embedding*, recorrendo a soluções da família GPT através de uma chave de API dedicada.

5.4 Containers C4

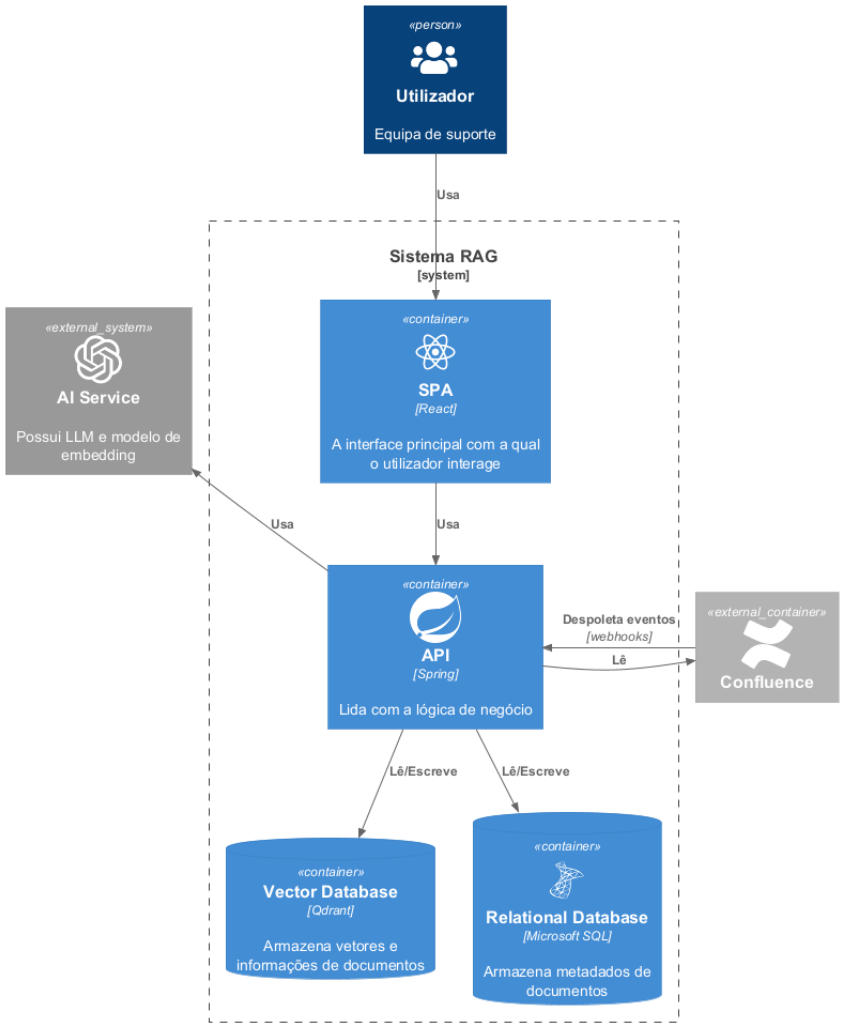


Figura 5.2: Diagrama de containers C4 do sistema RAG

Na Figura 5.2 encontra-se o diagrama de containers do sistema RAG. Este inclui uma Single-Page Application (SPA) em React, que serve de interface para interação funcional por parte dos utilizadores da equipa de suporte. O SPA depende de uma API Spring Boot, responsável pela gestão das regras de negócio e pelas principais interações com sistemas externos e de armazenamento.

Para este sistema justificou-se a adoção de dois tipos de base de dados, uma do tipo relacional e outra do tipo vetorial. A base de dados relacional tem a função de armazenar os metadados e estado dos documentos, sendo totalmente indicada para este tipo de operações. Já a base de dados vetorial tem a função de armazenar o conteúdo dos documentos e os seus vetores semânticos associados.

5.5 Componentes C4

Seguem-se os diagramas de componentes onde se irá detalhar o funcionamento lógico dos containers: SPA, API e Report.

5.5.1 Sistema RAG - SPA

Na Figura 5.3 consta o diagrama de componentes da SPA. Optou-se por utilizar uma arquitetura React que integra o componente *Router*, que é o ponto de entrada na aplicação através do URL da respetiva página, selecionando qual conteúdo renderizar na *View*. Esta última representa cada página da SPA e tem como dependência a camada *Component*, que representa componentes React reutilizáveis próprios ou provenientes de bibliotecas externas, a camada *Reducer*, que gere o *state* da aplicação e a sua atualização com base em ações, e o *Saga*, que é a camada que interage diretamente com a API e dispara ações de sucesso ou erro para o *Reducer*, atualizando o state conforme os resultados.

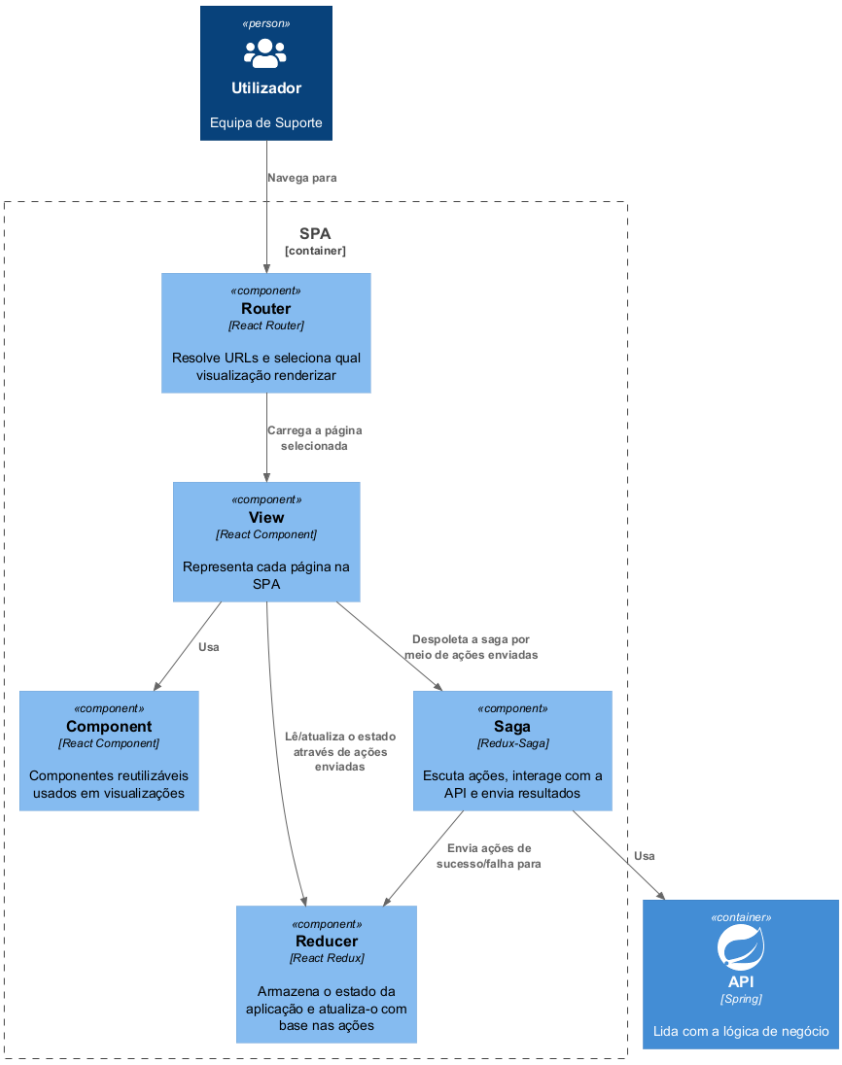


Figura 5.3: Diagrama de componentes C4 da SPA

5.5.2 Sistema RAG - API

Nesta secção serão detalhados os componentes da API do sistema RAG. Começa-se por introduzir os padrões e boas práticas adotados, seguido de uma comparação de alternativas de desenho relacionadas com a sincronização com o Confluence e terminando com a descrição do desenho final.

5.5.2.1 Padrões e boas práticas adotadas

Este desenho foi idealizado com base nos princípios da arquitetura Onion, com o objetivo de otimizar a testabilidade, manutenibilidade e confiabilidade do sistema. A arquitetura assenta na interação de várias camadas concêntricas em torno do núcleo representado pelo domínio.

Também foi aplicado o padrão Data Transfer Object (DTO) para transportar dados entre camadas sem expor diretamente a lógica de negócio e estabelecer contratos de resposta com a SPA.

Adicionalmente, seguiu-se o padrão Model–View–Controller (MVC), garantindo separação de responsabilidades entre lógica de negócio, apresentação e processamento de pedidos.

Relativamente aos serviços RAG, nomeadamente o armazenamento vetorial e a IA, são utilizadas abstrações do Spring AI implementadas segundo o padrão Adapter, para facilitar a integração esses serviços.

Por último, recorreu-se ao princípio de DI, promovendo baixo acoplamento entre componentes e aumentando a flexibilidade e a testabilidade do sistema.

5.5.2.2 Sincronização entre o sistema RAG e Confluence

Para o sistema de sincronização com dados do Confluence foram consideradas duas alternativas de desenho: *polling* e *webhooks*.

Alternativa 1

Na abordagem com *polling* (Figura 5.4), o cliente efetua pedidos periódicos ao sistema com o objetivo de detetar alterações (Bhandari, 2024). Para o nosso contexto, o sistema RAG verificaria, num intervalo de tempo configurado, a existência de atualizações nas páginas do Confluence, procedendo à sincronização dos conteúdos de acordo com o tipo de evento detetado.

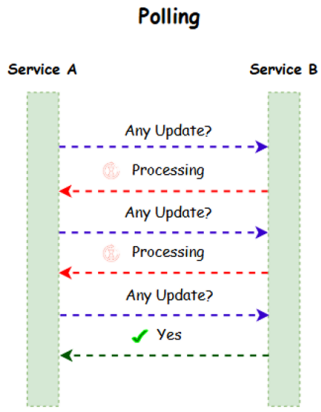


Figura 5.4: Abordagem com *polling* (Bhandari, 2024)

Alternativa 2

A abordagem com *webhooks* (Figura 5.5) permite que uma aplicação externa envie pedidos em tempo real para uma aplicação interna sempre que ocorre um evento específico (Bhandari, 2024).

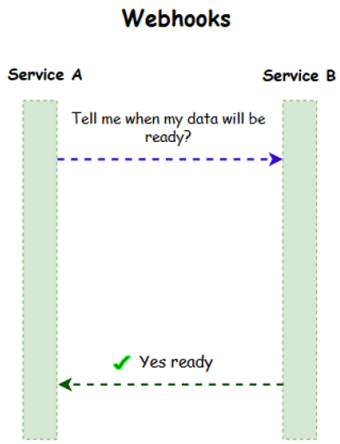


Figura 5.5: Abordagem com Webhooks

Comparação de Alternativas

As vantagens da utilização do *polling* são a grande compatibilidade com sistemas externos que não suportam *Webhooks* e a redução de complexidade por não ser necessário a exposição de um *endpoint* dedicado aos sistemas externos. Em contrapartida, este método implica um maior consumo de recursos, resultante de pedidos recorrentes que frequentemente não produzem alterações, e não assegura sincronização em tempo real (Bhandari, 2024).

A utilização de *Webhooks*, por sua vez, garante maior eficiência, evitando chamadas desnecessárias, e possibilita a atualizações em tempo real. Contudo, este modelo exige a disponibilização de um *endpoint* dedicado no cliente interno, bem como a configuração de mecanismos de autenticação e autorização. Acresce ainda a necessidade de medidas complementares para recuperação de eventos, em cenários de indisponibilidade do servidor externo (Bhandari, 2024).

Face à boa compatibilidade oferecida pelo Confluence e por garantir uma sincronização em tempo real, a abordagem com *webhooks* foi selecionada como solução a implementar no projeto.

5.5.2.3 Desenho Final

Na Figura 5.6 está representada a arquitetura de componentes final da API, que constitui o core do sistema RAG a ser desenvolvido.

O núcleo da arquitetura é formado pelo componente Domain, que engloba os modelos de negócio, serviços de domínio e interfaces de domínio, como é o caso do *Repository Interface*. À volta deste núcleo, encontram-se as camadas responsáveis pela interação com a infraestrutura para comunicação com serviços externos (Infrastructure), pelo repositório relacional (*JPA Repository*) e pelos serviços aplicacionais (*Application Service*), onde está implementada a lógica dos casos de uso.

O componente DTOs é responsável por transportar dados entre as camadas sem expor a lógica interna e garantindo o contrato de comunicação com a SPA.

O *Controller* é o ponto de entrada da API, seguindo a lógica MVC, e coordena a execução dos serviços aplicacionais.

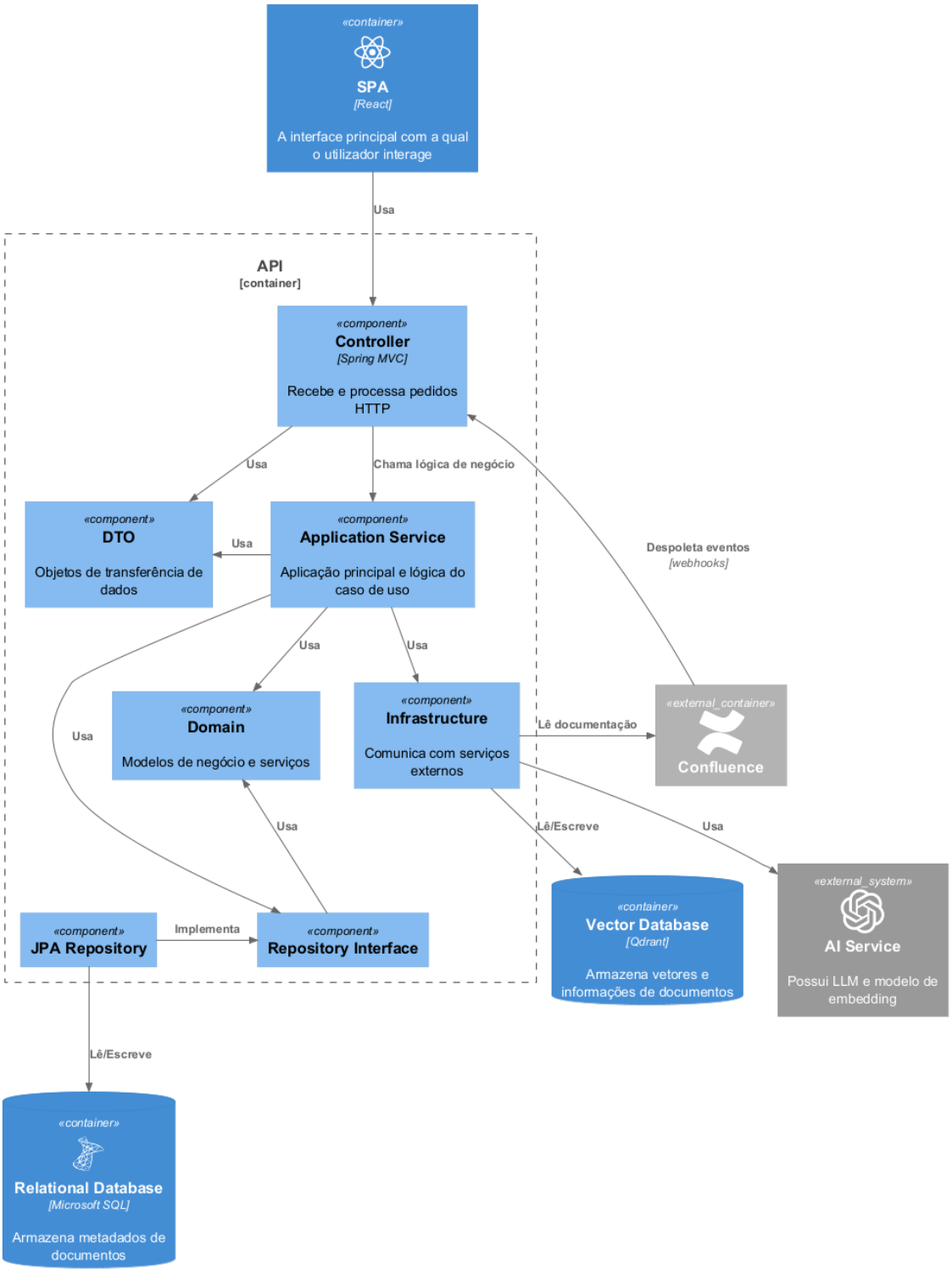


Figura 5.6: Diagrama de componentes C4 da API

5.5.3 B2C Core - Report

Na Figura 5.7 encontra-se o diagrama de componentes do container Report, que é o módulo exclusivo do *B2C Core* a ser desenvolvido no âmbito deste projeto.

Este representa uma aplicação Java responsável por suportar todos os relatórios e regras de negócio associadas no contexto da equipa B2C. O processo começa pelo componentes *Processor*, que é responsável por interpretar e validar os parâmetros de entrada e controlar o processo de extração. Este é composto em primeira instância pelo *Generator*, responsável por ler os dados relevantes do Control-M e gerar o relatório. Seguidamente, o componente *Writer* é responsável por formatar o relatório e publicá-lo no Confluence. Ambos dependem do componente *Infrastructure*, que abstrai a comunicação com os sistemas externos.

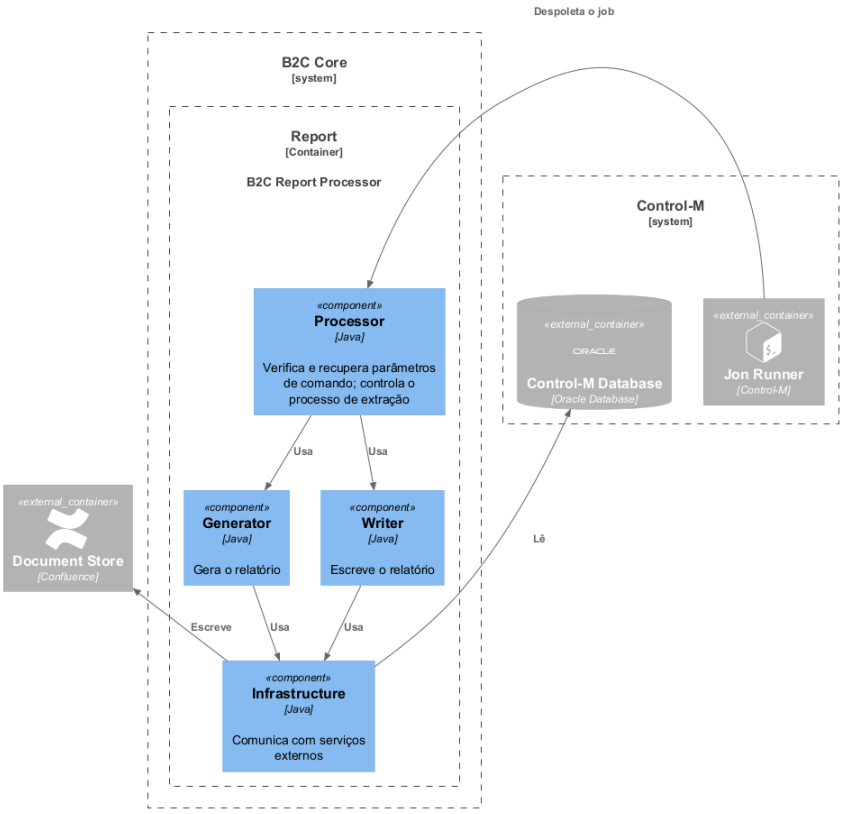


Figura 5.7: Diagrama de componentes C4 do módulo de extração de dados do Control-M

5.6 Diagramas de Implantação

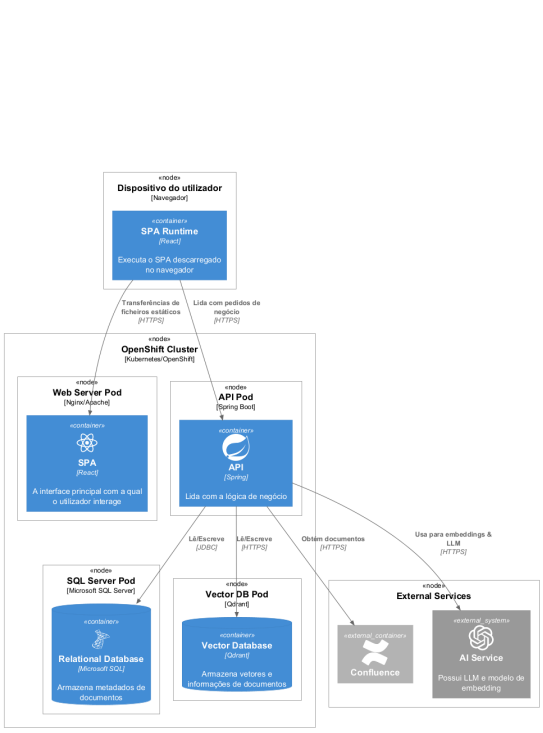
Nesta secção são apresentados os diagramas de implantação, que descrevem a distribuição física dos componentes do sistema, as suas interações e os ambientes onde são executados. Estes diagramas permitem compreender como os diferentes módulos são alocados em servidores e infraestruturas locais, facilitando a análise de requisitos não funcionais como escalabilidade e desempenho.

Sistema RAG

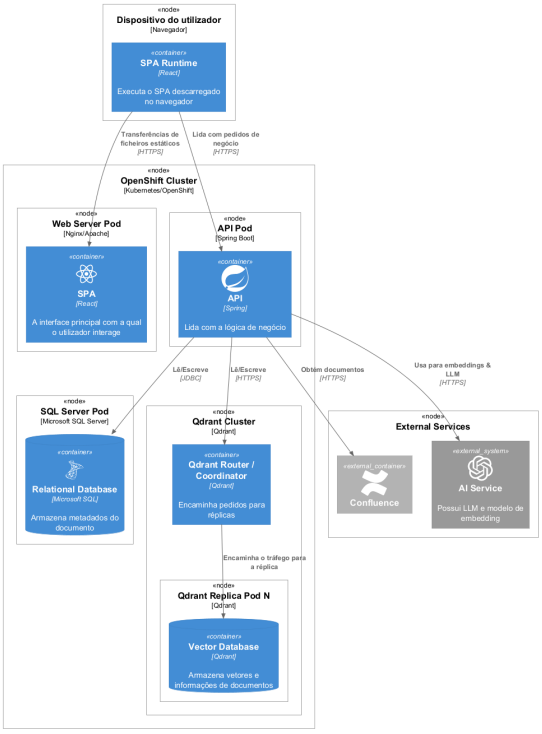
O sistema RAG será concebido para suportar um número máximo de aproximadamente 20 utilizadores em simultâneo. Tendo em conta este nível de utilização, optou-se por não implementar escalabilidade horizontal, concentrando-se apenas na correta alocação de recursos necessários tanto no *frontend* como no *backend*.

No que se refere ao armazenamento vetorial utilizando o Qdrant, consideraram-se duas alternativas de implementação:

- **Alternativa 1 – Nó único:** Consiste na utilização de um único nó de base de dados (Figura 5.8a). Esta abordagem é mais simples de implementar e permite tempos de resposta mais rápidos, sendo adequada para sistemas de pequena escala ou quando a documentação gerida não é volumosa.
- **Alternativa 2 – Nós distribuídos:** Envolve a utilização de múltiplos nós, aproveitando a capacidade nativa do Qdrant para escalabilidade horizontal (Figura 5.8b). Esta solução apenas se justifica em cenários onde a quantidade de documentação seja muito extensa ou esteja em crescimento contínuo, garantindo maior redundância, disponibilidade e capacidade de processamento distribuído.



(a) Diagrama de implantação do sistema RAG - Alternativa 1



(b) Diagrama de implantação do sistema RAG - Alternativa 2

Figura 5.8: Alternativas de implantação do sistema RAG

Report

No caso do módulo de Report, manteve-se a arquitetura já consolidada na B2C, em que toda a lógica de exportação de relatórios é desenvolvida em Java, compilada num ficheiro *.jar* e implementada como um serviço autónomo no servidor da organização (Figura 5.9).

Esta abordagem garante integração transparente com a infraestrutura existente, facilita a manutenção e permite a monitorização centralizada das execuções e eventuais falhas.

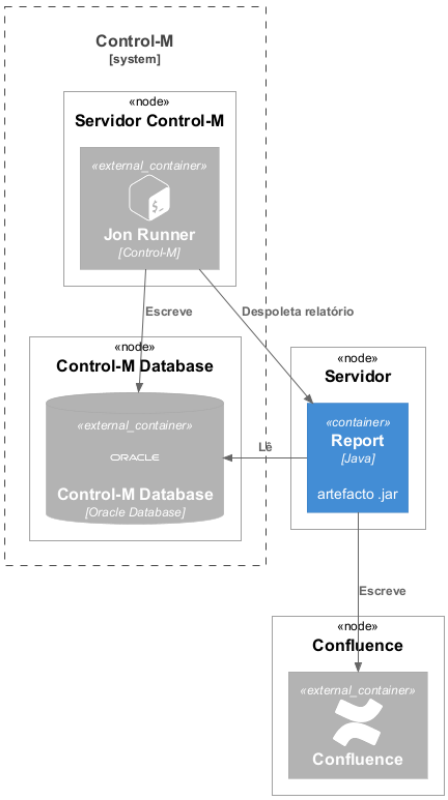


Figura 5.9: Diagrama de implantação do módulo de extração de dados do Control-M

5.7 Diagramas de Sequência

Optou-se por apresentar diagramas de sequência apenas para os casos de uso mais relevantes para o caso de estudo, nomeadamente aqueles que influenciam a *pipeline* RAG e o processo de extração e publicação dos dados históricos do Control-M. Estes diagramas ilustram de forma clara o fluxo de interação entre os principais componentes, facilitando a compreensão das operações críticas e servindo de referência para a implementação.

A interação entre componentes da SPA foi abstraída e focou-se na interação dos componentes da API por ser mais relevante para a compreensão da lógica de negócio.

Segue-se a apresentação dos diagramas, juntamente com uma breve descrição do seu fluxo.

5.7.1 Pipeline RAG

A *pipeline* RAG deste projeto em primeira instância consiste na indexação dos dados numa base de dados vetorial seguido do processo de recuperação assistida por IA.

Indexação de dados

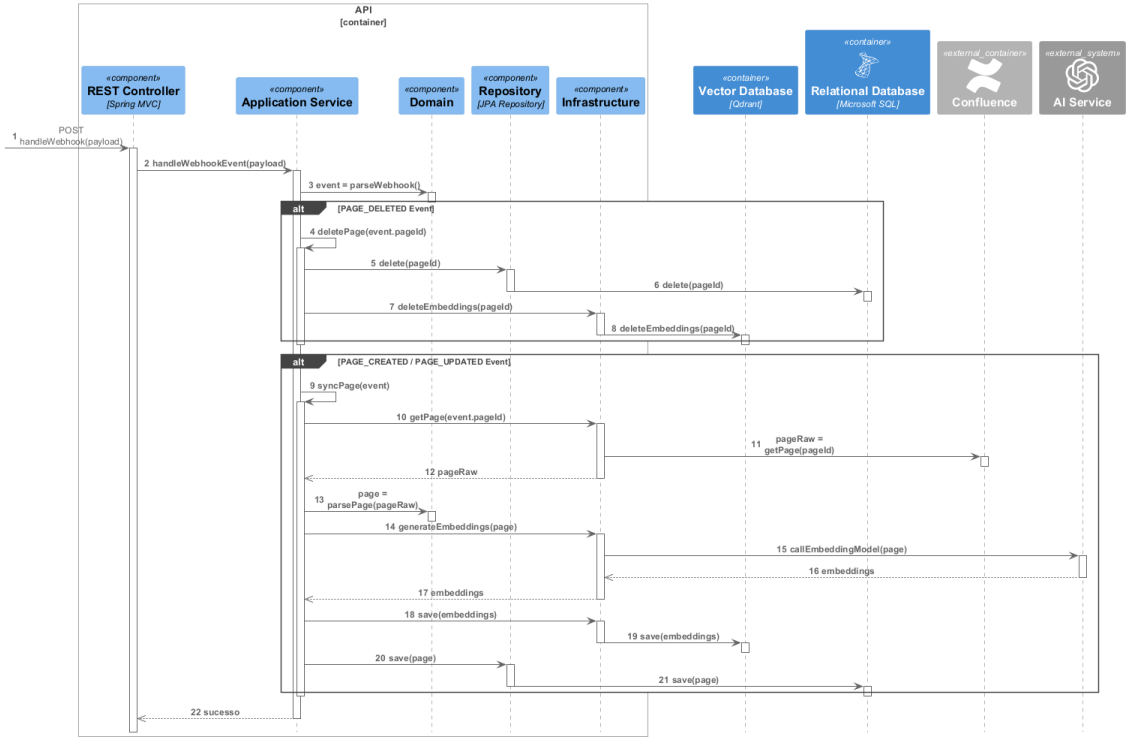


Figura 5.10: Diagrama de sequência do mecanismo de sincronização com o Confluence - UC05

A indexação acontece de forma automática através de um mecanismo de sincronização que é despoletado através da receção de *webhooks* do Confluence. O respetivo diagrama de sequência encontra-se na Figura 5.10.

- Primeiramente o *webhook* referente a um evento de criação, edição ou remoção de página é recebido pela API (1);
- O *Controller* encaminha o pedido para a camada de *Service* onde ocorre a interpretação dos seus dados (2-3);
- Caso se trate de um evento de página removida, ocorre um processo de remoção tanto na base de dados relacional como na base de dados vetorial (4-8);
- Quando o evento se trata de edição ou criação de uma página, primeiramente o seu conteúdo é lido do Confluence e interpretado (9-13), seguido da geração dos vetores com recurso ao modelo de *embedding* (14-17) e finalmente a página é armazenada/-sobreposta nas bases de dados (18-21).

Recuperação assistida por IA

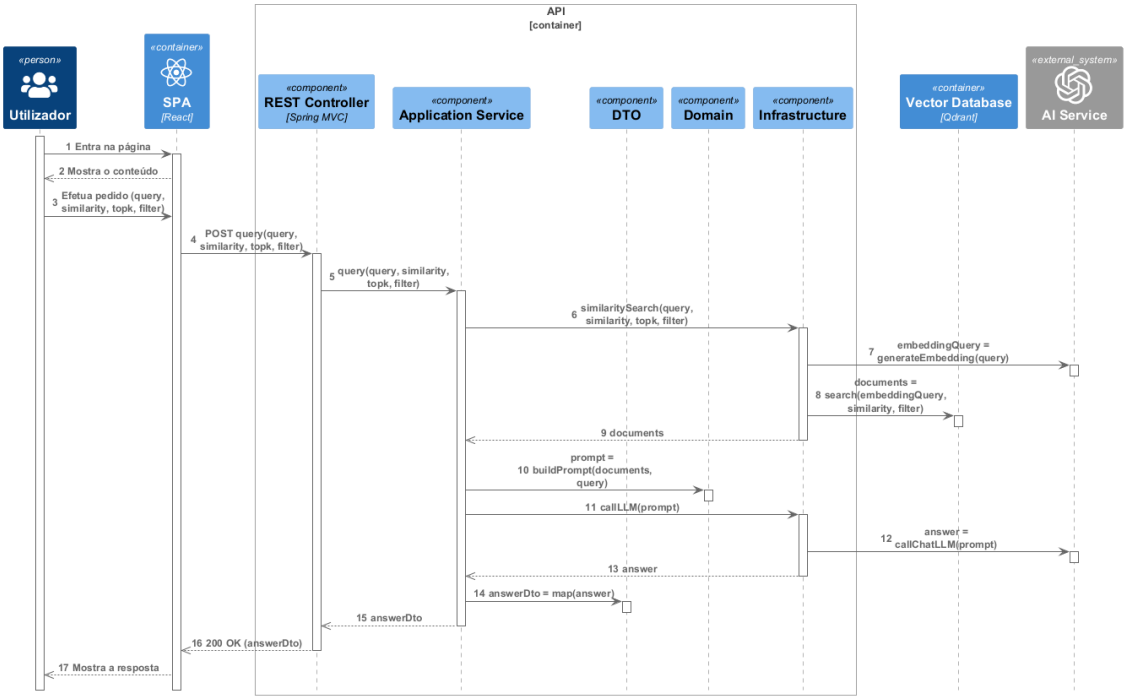


Figura 5.11: Diagrama de sequência da recuperação assistida por IA - UC01

A recuperação assistida por IA é despoletada por um utilizador e tem por objetivo reunir documentação relevante para dar resposta a determinada *query*. O respetivo diagrama de sequência encontra-se na Figura 5.11.

- O utilizador acede à respetiva página na SPA (1-2);
- O utilizador elabora uma *query* de pesquisa, podendo configurar o grau de similaridade e as categorias de filtro; caso não especifique, são aplicados valores por defeito (3);
- O pedido passa entre as camadas *Controller* e *Service* da API (4-5);
- Primeiramente o serviço realiza uma pesquisa vetorial de todos os *chunks* relevantes baseado na *query* do utilizador, sendo essa query vetorizada com recurso ao modelo de *embedding* de forma a ser possível comparar com os vetores já existentes (6-9);
- Baseado nos *chunks* recuperados, o *prompt* é construído (10);
- O *prompt* é encaminhado ao LLM, gerando a resposta final (11-16);
- A resposta é apresentada ao utilizador na interface (17).

5.7.2 Extração de dados do Control-M

A extração de dados do Control-M é despoletada pelo próprio sistema, que inicia o processo de extração no horário agendado. O respetivo diagrama de sequência encontra-se na Figura 5.12.

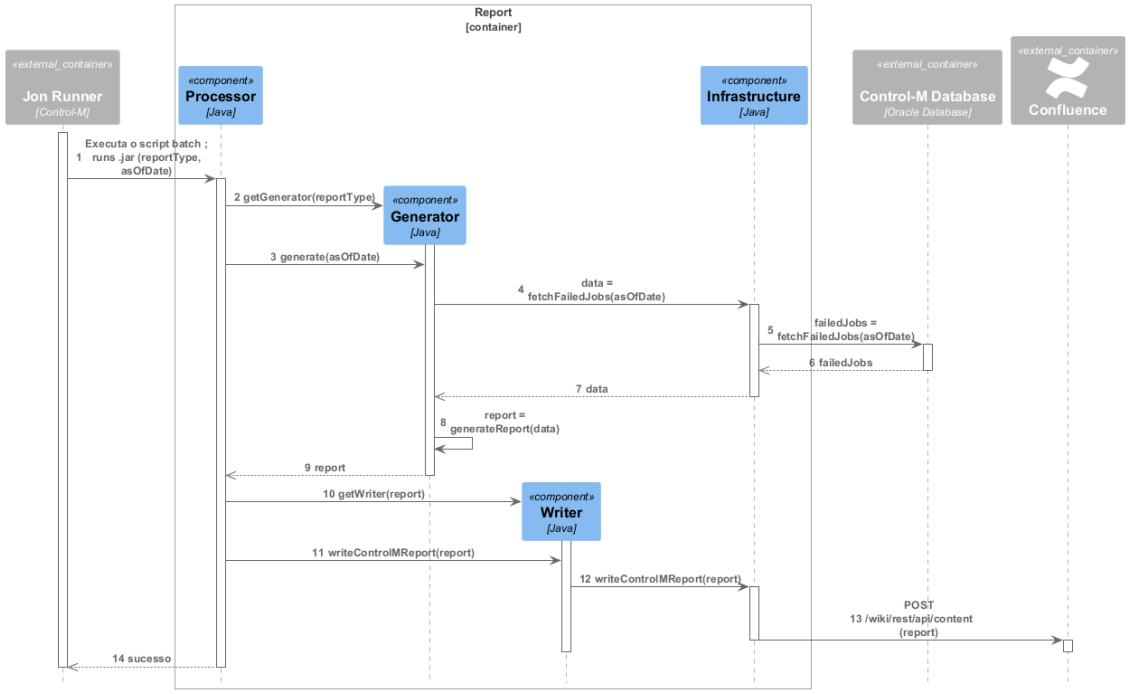


Figura 5.12: Diagrama de sequência da extração de dados do Control-M - UC06

- O módulo de extração é acionado pelo *job* do Control-M (1);
- A instância de *Generator* é criada com base no tipo de relatório (2);
- Os dados são obtidos da base de dados do Control-M e o relatório é construído (3-9);
- A instância de *Writer* é criada com base no tipo de relatório (10);
- O relatório é formatado e escrito no Confluence (11-14).

Capítulo 6

Implementação

Este capítulo apresenta a implementação dos principais artefactos desenvolvidos no âmbito do projeto, detalhando a estrutura, componentes e tecnologias utilizadas em cada módulo. Serão abordados os aspetos técnicos relevantes, incluindo diagramas, exemplos de código e explicações sobre a integração entre os diferentes sistemas, de forma a ilustrar o funcionamento global da solução proposta.

6.1 Estrutura

Segue-se a apresentação da estrutura geral de cada artefacto desenvolvido. Serão apresentados os respetivos diagramas de implementação e detalhes técnicos de cada componente.

6.1.1 Módulo SPA

Na Figura 6.1 encontra-se o diagrama de implementação da SPA.

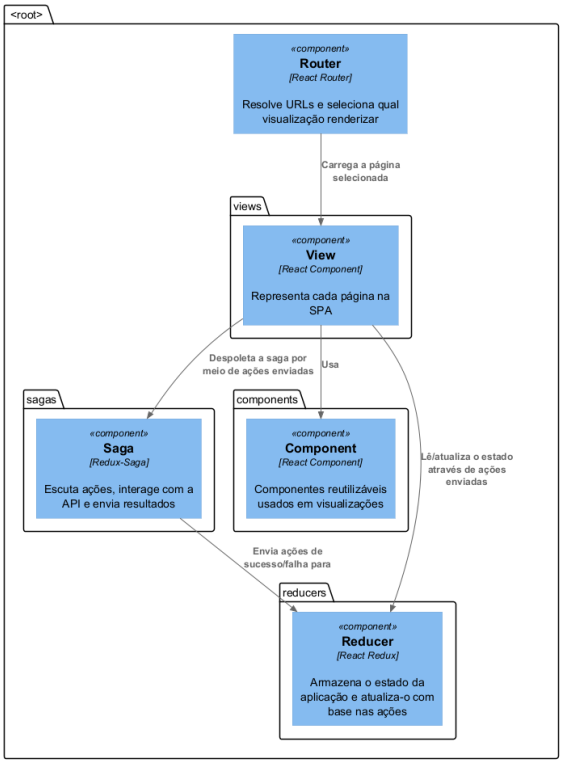


Figura 6.1: Diagrama de implementação da SPA

O *Router* encontra-se configurado no ficheiro App.js, presente na *root* do projeto, e foi implementado com o auxílio da biblioteca *react-router-dom*, sendo considerada a solução padrão para gestão de rotas em aplicações React.

Este componente define rotas para duas interfaces desenvolvidas (6.1): página de recuperação com o path "/"e página de gestão de documentos no path "/manage".

```
...
<Router>
    <TopNavbar/>
     <Routes>
        <Route path="/" element={<Retriever/>}/>
        <Route path="/manage" element={<DocumentManager/>}/>
    </Routes>
</Router>
...

```

Código 6.1: Composição do Router

De seguida, destaca-se o *Component*, responsável pela camada de componentes reutilizáveis em toda a aplicação. Foram adotados componentes do ecossistema *Material UI*, que serviram de base para o desenvolvimento da interface do sistema. Sobre estes, foram construídos componentes personalizados, como a barra de navegação partilhada em todas as páginas, *TopNavbar* (6.2), que recorre aos seguintes elementos do *Material UI*: *AppBar*, utilizado para criar barras de navegação fixas no topo da interface; *Toolbar*, que organiza e alinha os conteúdos dentro da *AppBar*; e *Typography*, dedicado à apresentação de texto, garantindo consistência visual em toda a aplicação.

```
...
function TopNavbar() {
    return (
        <AppBar position="static" sx={(theme) => topNavbar(theme)}>
            <Toolbar sx={(theme) => toolbarStyles(theme)}>
                <Typography sx={(theme) => logoStyle(theme)}>
                    B2C RAG
                </Typography>

...
```

Código 6.2: Barra de navegação

Desenvolvido com auxílio da biblioteca *react-redux*, o *Reducer* permite aos componentes React acedam aos dados armazenados no Redux Store e executem ações para atualizar esses dados, garantindo a gestão eficiente do *state* da aplicação de forma centralizada. No Código 6.3 encontra-se um exemplo de *Reducer* para a página de recuperação. É possível notar que o *initialState* define valores por defeito para as configurações opcionais de grau de similaridade e *top-k chunks*. Mais abaixo é possível visualizar a atualização do *state* despoletado por ações *Redux*.

```
...
const initialState = {
    messages: [],
    input: "",
    topChunks: 3,
    similarityThreshold: 75,
    treeNodes: [],
    treeNodeChecked: [],
};

const retrieverReducer = (state = initialState, action) => {
    switch (action.type) {
        case SET_RETRIEVER_MESSAGES:
            return {
                ...state,
                messages: action.payload,
            };
...

```

Código 6.3: Exemplo de *Reducer* da página de recuperação

O *Saga*, implementado com *redux-saga*, é responsável por gerir operações assíncronas da aplicação, como chamadas a serviços externos, garantindo que estas não sejam tratadas diretamente no *Reducer*. Neste contexto, o Saga gere as interações com a API.

No exemplo 6.4, a função *retrieverSaga* utiliza o efeito *takeLatest*, garantindo que apenas a ação mais recente seja processada, evitando sobrecarga e resultados desatualizados.

```
...
function* retrieverSaga() {
    yield takeLatest(QUERY_RETRIEVER_AI, callRetrieverAi);
    yield takeLatest(FETCH_TREE_NODES, fetchTreeNodes);
}
...

```

Código 6.4: Exemplo de saga de recuperação

Por fim, a *View* constitui a camada responsável pela interação entre o utilizador e a lógica da aplicação, integrando todas as camadas previamente abordadas. Cada componente da *View* representa uma página distinta da SPA, neste caso duas páginas, conforme descrito no *Router* (Código 6.1). Esta camada combina componentes reutilizáveis da camada *Component* com elementos do *Material UI*, assegurando consistência visual e usabilidade. A lógica de interação está diretamente conectada ao Redux Store através dos métodos *mapStateToProps* e *mapDispatchToProps*, permitindo o mapeamento eficiente das variáveis do *state* e o disparo de ações diretamente pelas *props* dos componentes da *View*.

6.1.2 Módulo API

Na Figura 6.2 encontra-se o diagrama de implementação da API.

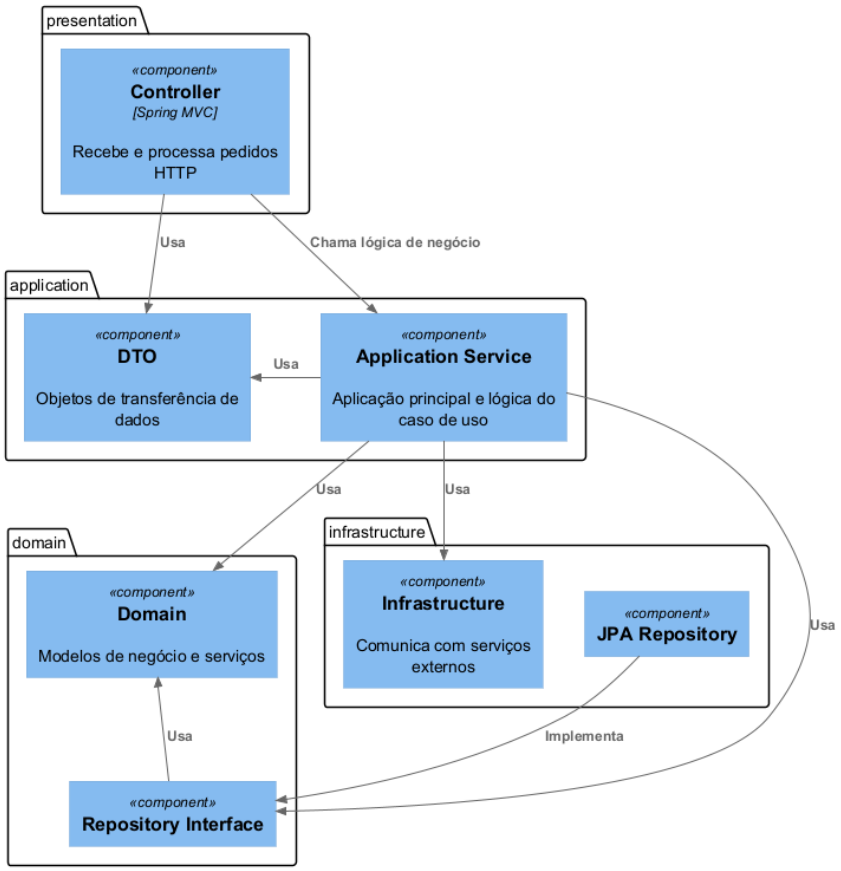


Figura 6.2: Diagrama de implementação da API

A camada de domínio integra os componentes *Domain* e *Repository Interfaces*. O primeiro contempla os modelos de domínio associados aos conceitos de negócio, como *Page*, *Webhook* e *Prompt*, bem como serviços de domínio que disponibilizam mecanismos para manipulação e construção de entidades. Por sua vez, o componente *Repository Interfaces* estende a interface *JpaRepository*, permitindo a definição de métodos personalizados para consultas à base de dados relacional.

A camada *Infrastructure* é fundamental pois integra a lógica para a comunicação com sistemas externos e bases de dados.

Para comunicação com o Confluence, elaborou-se um *client* próprio que define métodos para obtenção de informação e conteúdo das páginas. Este *client* estabelece uma conexão REST com o Confluence, fazendo uso das variáveis (Código 6.5) *BASE_URL*, que é o ponto de entrada na *wiki*; *SPACE_ID*, que define o espaço da equipa B2C e onde serão encontradas as páginas relevantes; *USER* e *API_TOKEN*, que são as credenciais de autenticação da *API REST*.

```
...
@Service
public class ConfluenceApiClient {
    @Value("${confluence.base_url}")
    private String BASE_URL;
    @Value("${confluence.space.id}")
    private String SPACE_ID;
    @Value("${confluence.api_token}")
    private String API_TOKEN;
    @Value("${confluence.user}")
    private String USER;
...

```

Código 6.5: Definição do *ConfluenceApiClient*

Relativamente aos serviços de armazenamento vetorial e IA, foram utilizadas abstrações do Spring AI.

Começando pelo armazenamento, Qdrant foi a tecnologia selecionada por ter compatibilidade nativa com Spring AI e por ter capacidade de armazenar grandes quantidades de vetores sem perder desempenho. No *application.properties* do projeto Spring foram configuradas as seguintes propriedades fundamentais para estabelecer conexão: *host*, *port* e *collection-name*. Desta forma, é possível usar a abstração *VectorStore*, do Spring AI, por DI, nos serviços aplicacionais (Figura 6.3).

```
...
spring.ai.vectorstore.qdrant.host=${
    QDRANT_HOST}
spring.ai.vectorstore.qdrant.port=6334
spring.ai.vectorstore.qdrant.collection-
    name=b2c-rag
...

```

Código (6.6) Propriedades do Qdrant

```
...
@Service
@Slf4j
public class QueryService {

    ...
    private final VectorStore
    vectorStore;
    ...

    public QueryService(..., VectorStore
     vectorStore, ...) {
        this.vectorStore = vectorStore;
        ...
    }
...

```

Código (6.7) Vector Store por DI

Figura 6.3: Uso da abstração Vector Store

Relativamente ao serviço de IA, foram utilizados recursos do Spring AI para facilitar a integração com o serviço da OpenAI. No Código 6.8 estão apresentadas as propriedades necessárias para configurar a ligação, incluindo a *api-key*, o modelo de *chat* e o modelo de *embedding*. Esta configuração permite utilizar a abstração *ChatClient* para interagir com o modelo de *chat* e a *VectorStore* para realizar pesquisas vetoriais por similaridade, recorrendo ao modelo de *embedding* definido.

```
...
spring.ai.openai.api-key=${OPENAI_API_KEY}
spring.ai.openai.chat.options.model=gpt-4o-mini
spring.ai.openai.embedding.options.model=text-embedding-3-small
...

```

Código 6.8: Configuração dos modelos de IA

As camadas de *Presentation* e *Application* incluem os componentes responsáveis pela interação REST com a API e pela implementação da lógica dos casos de uso. O *Controller* utiliza a anotação *@RestController* do Spring Framework para definir os *endpoints* e as interfaces de acesso às funcionalidades, conforme exemplificado no Código 6.9. Já o *Application Service* segue uma abordagem semelhante, recorrendo à anotação *@Service* para coordenar as interações entre as camadas *Infrastructure* e *Domain*.

```
...
@PostMapping("/query")
public ResponseEntity<ChatResponseDto> query(@RequestBody UserQueryDto queryDto) {
    return ResponseEntity.ok(ChatResponseDto.builder().reply(queryService.query(
    queryDto)).build());
}
...

```

Código 6.9: Exemplo de *endpoint* no *Controller*

6.1.3 Módulo Report - Extração Control-M

O diagrama de implementação do módulo *Report* encontra-se representado na Figura 6.4, evidenciando os seus principais componentes e a forma como interagem para fornecer relatórios históricos sobre os *jobs* do Control-M.

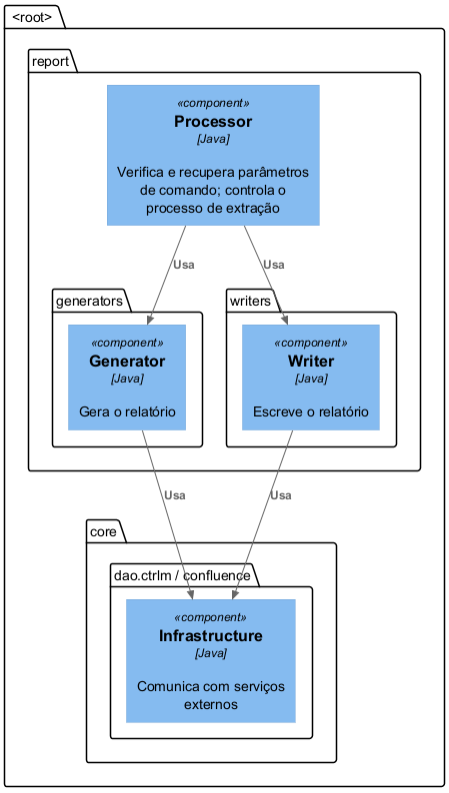


Figura 6.4: Diagrama de implementação do Report

Destaca-se o componente *Infrastructure*, nomeadamente os dois elementos cruciais que permitem a extração e publicação da informação.

O Código 6.10 apresenta a consulta utilizada para identificar *jobs* que falharam, obtendo informações relevantes como o grupo, a tabela de agendamento, o estado de conclusão, a hora de início e a descrição do *job*.

```
...
    var sql = String.format("""
            SELECT rh.GROUP_NAME, rh.SCHED_TABLE, JOB_MEM_NAME, START_TIME,
    ENDED_STATUS, dtj.DESCRIPTION
            FROM BW2_REPORTP.RUNINFO_HISTORY rh JOIN
            (SELECT* FROM (SELECT RANK() OVER (PARTITION BY SCHED_TABLE ORDER BY
    LAST_UPLOAD DESC) as RK, JOB_NAME, DESCRIPTION
            FROM BW2_REPORTP.DEF_TABLES_AND_JOBS
            WHERE DATA_CENTER = '%s' AND APPLICATION = '%s'
            AND NOT TASK_TYPE ='Dummy') WHERE RK =1) dtj
            ON rh.JOB_MEM_NAME = dtj.JOB_NAME
            WHERE rh.DATA_CENTER ='%s' AND rh.APPLICATION = '%s'
            AND rh.GROUP_NAME LIKE ?
            AND NOT SCHED_TABLE = JOB_MEM_NAME
            AND START_TIME BETWEEN TO_TIMESTAMP(?, 'yyyy-mm-dd hh24:mi:ss')
            AND TO_TIMESTAMP(?, 'yyyy-mm-dd hh24:mi:ss')
            AND ENDED_STATUS = 32""", ctmDataCenter, ctmApplication, ctmDataCenter,
    ctmApplication);
...

```

Código 6.10: *Query* SQL para a recolha de *jobs* falhados da base de dados do Oracle do Control-M

Após a recolha e processamento dos dados, o módulo utiliza o *endpoint* ilustrado no Código 6.11 para gerar páginas no Confluence de forma automática. Este mecanismo permite disponibilizar os relatórios de forma organizada e acessível aos utilizadores.

```
...
{
    "type": "page",
    "title": "Control-M History Report - Week from 27/07 to 02/08",
    "ancestors": [
        {
        "id": <ID>
        }
    ],
    "space": {
        "key": "<KEY>"
    },
    "body": {
        "storage": {
            "value": "<HTML_CONTENT>",
            "representation": "storage"
        }
    }
}
...
```

Código 6.11: *Endpoint* para criação de página no Confluence - POST /wiki/rest/api/content

6.2 Casos de Uso

Esta secção detalha alguns dos casos de uso implementados no âmbito do projeto, detalhando os fluxos funcionais e as interações entre os diferentes módulos do sistema.

6.2.1 Pipeline RAG - Indexação/Sincronização com o Confluence (UC5)

Os pedidos apenas chegam à API mediante a subscrição prévia de um *webhook* do Confluence, que despoleta eventos quando há criação, edição ou remoção de páginas.

A subscrição processa-se através da chamada do seguinte *endpoint* na API do Confluence:

- POST /wiki/rest/webhooks/1.0/webhook

O conteúdo do pedido é composto pela informação presente no Código 6.12. Este inclui o nome do *webhook*, eventos subscritos (*page_created*, *page_updated*, *page_trashed*), URL interno do sistema RAG onde os eventos são processados, e informação do espaço elegível.

```
{
  "name": "Page Created/Updated/Deleted Hook",
  "events": [
    "page_created",
    "page_updated",
    "page_trashed"
  ],
  "url": "<BASE_URL>/webhook",
  "active": true,
  "filters": {
    "spaceId": 131075
  }
}

```

Código 6.12: Subscrição de *webhooks*

No Código 6.13 encontra-se um exemplo do conteúdo de um evento de edição de página. Este contém informações como o ID da página e tipo de evento, cruciais para despoletar as ações internas de sincronização.

```
{
  "page": {
    "idAsString": "18382858",
    "creatorAccountId": "5dc54bb7ad36280c5279fd35",
    "spaceKey": "~5dc54bb7ad36280c5279fd35",
    "spaceId": 131075,
    "modificationDate": 1750007555806,
    "lastModifierAccountId": "5dc54bb7ad36280c5279fd35",
    "self": "<PAGE_URL>",
    "id": "18382858",
    "title": "<TITLE>",
    "creationDate": 1749940298032,
    "contentType": "page",
    "version": 3
  },
  "accountType": "customer",
  "timestamp": 1750007555828,
  "userAccountId": "5dc54bb7ad36280c5279fd35",
  "updateTrigger": "edit_page",
  "suppressNotifications": false
}

```

Código 6.13: Exemplo de evento de edição de página

O pedido chega à API interna pelo *endpoint*:

- *@PostMapping("/webhook") public ResponseEntity<Void> handleWebhook(@RequestBody byte[] bodyBytes)*

Segue-se a identificação do tipo de evento (Código 6.14): quando o evento é do tipo *CONFLUENCE_CREATE_PAGE* ou *CONFLUENCE_UPDATE_PAGE* o método *contextService.indexDocument* é chamado; quando o evento é do tipo *CONFLUENCE_DELETE_PAGE* o método *contextService.deleteDocument* é chamado.

```
...
@Override
protected void process(JSONObject jsonObject, WebhookEventType eventType) throws
    ConfluencePageNotFound {
    String pageId = jsonObject.getJSONObject("page").getString("idAsString");
    switch (eventType) {
        case CONFLUENCE_CREATE_PAGE, CONFLUENCE_UPDATE_PAGE ->
        contextService.indexDocument(pageId, IndexType.AUTOMATIC);
        case CONFLUENCE_DELETE_PAGE -> contextService.deleteDocument(pageId);
    }
}
...
```

Código 6.14: Tratamento dos *webhooks*

No que toca à indexação, primeiramente verifica-se se a página do evento pertence à categoria de excluídos. Em caso afirmativo, ignora-se o evento.

A partir do *pageId*, são efetuadas duas chamadas à API do Confluence:

- *GET /api/v2/pages/<pageId>* : obtenção dos metadados da página
- *GET /rest/api/content/<pageId>?expand=body.storage* : obtenção do conteúdo da página em formato HTML

Caso a página já tenha sido indexada no passado, procede-se à atualização dos seus metadados e vetores. Caso contrário, os dados são criados de início. A atualização dos metadados é feita diretamente na entidade *Page* e de seguida persistida. Já no caso dos vetores procede-se à sua reescrita.

A reescrita de vetores consiste na remoção dos antigos e escrita dos novos. A remoção é feita com o *QdrantClient* e o método desenvolvido encontra-se no Código 6.15

```
...
private void deleteConfluencePageVector(String pageId) {
    var filter = Points.Filter.newBuilder()
    .addMust(Points.Condition.newBuilder()
        .setField(Points.FieldCondition.newBuilder()
            .setKey(CONFLUENCE_PAGE_ID)
                .setMatch(Points.Match.newBuilder()
                    .setText(pageId)
                .build())
            .build())
        .build())
    .build();
    client.deleteAsync(QDRANT_COLLECTION_NAME, filter);
}
...
```

Código 6.15: Método de remoção de vetores

A fase de escrita de vetores procede-se da seguinte forma (Código 6.16):

- Divisão da página em vários chunks: isto é importante para uniformizar o tamanho dos chunks na base de dados vetorial. Foi feito com auxílio de uma biblioteca nativa do Spring AI, *MarkdownDocumentReader*, que lê o conteúdo markdown e faz uma divisão ponderada. Visto isto, foi necessária a conversão da página do Confluence em HTML para Markdown.

- Preparação dos Documents: Depois de obtida a *List<Document>* procede-se à adição de alguns metadados relevantes para ser persistidos juntamente com os vetores. Estes incluem, *CONFLUENCE_PAGE_ID*, *FOLDER_TREE*, *CREATE_DATE* e *MODIFY_DATE*. A partir destes metadados, torna-se possível, no futuro, aplicar filtros às pesquisas vetoriais.
- Armazenamento dos vetores: com a chamada *vectorStore.accept(documents)* os documentos são persistidos na base de dados vetorial. Este método chama implicitamente o modelo de *embedding* configurado, criando um vetor para cada *chunk*.

```
...
    private void indexConfluencePage(ConfluencePage confluencePage) {
        ConfluenceDocumentReader confReader = new ConfluenceDocumentReader();
        Resource markdownPage = confReader.getConfluenceDocumentMarkdown(
    confluencePage);
        var reader = new MarkdownDocumentReader(markdownPage,
                MarkdownDocumentReaderConfig.builder().build());

        List<Document> documents = reader.get();

        documents = documents
                .stream()
                .map(doc -> {
                    String contentWithTitle = "Chunk parent page title:" +
    confluencePage.getTitle()
                            + "\n\n"
                            + doc.getFormattedContent();
                    var newDoc = new Document(contentWithTitle);
                    newDoc.getMetadata().put(CONFLUENCE_PAGE_ID, confluencePage.
    getId());
                    newDoc.getMetadata().put(FOLDER_TREE, confluencePage.
    getFolderTree());
                    newDoc.getMetadata().put(CREATE_DATE, confluencePage.
    getCreatedAt());
                    newDoc.getMetadata().put(MODIFY_DATE, confluencePage.
    getUpdatedAt());
                    return newDoc;
                })
                .toList();

        vectorStore.accept(documents);
    }
...
```

Código 6.16: Método de indexação de vetores

6.2.2 Pipeline RAG - Recuperação assistida (UC1)

A funcionalidade de recuperação começa na SPA com a chamada REST à API. O *body* é composto pela *query* do utilizador, *similarityThreshold*, *top-k chunks* e categoria documental no *eligibleFolderTree* (Código 6.17).

```
...
const response = await fetch(`${API_HOST}/query`, {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
    },
    body: JSON.stringify({
        query: action.payload.query,
        similarityThreshold: action.payload.similarityThreshold / 100,
        topK: action.payload.topChunks,
        eligibleFolderTree: action.payload.treeNodeChecked,
    }),
});
...
```

Código 6.17: Chamada do recuperador na SPA

Quando o pedido chega à API procede-se da seguinte forma (Código 6.19):

- Pesquisa vetorial: pesquisa na base de dados vetorial para obtenção de uma *List<Document>* com uso do método *vectorStore.similaritySearch* presente no *VectorStore*. Nesta pesquisa configurou-se o *similarityThreshold*, *topK* e *filterExpression*. Este último permite o filtro por metadados armazenados juntos com os vetores (Código 6.18 e Código 6.19).
- Construção do *prompt*: baseado na *List<Document>* obtida anteriormente, procede-se à construção do *prompt* final que resulta da junção da informação dos documents com a *query* inicial. A estratégia utilizada foi a *chain-of-thought prompting*, estando o *prompt* ilustrado no Código A.1.
- Chamada ao modelo de *chat*: procede-se à chamada ao modelo de *chat* com o *fullPrompt*.

```
...
        return vectorStore.similaritySearch(SearchRequest.builder()
                .query(queryDto.query)
                .similarityThreshold(queryDto.similarityThreshold)
                .topK(queryDto.topK)
                .filterExpression(filter)
                .build());
...

```

Código 6.18: Pesquisa vetorial

```
...
public String query(UserQueryDto queryDto) {
    List<Document> docs = searchDocuments(queryDto);

    if (docs.isEmpty()) {
        return "No relevant documents were found in the eligible" +
        " folder tree to answer your question.";
    }
    String context = promptBuilder.buildContext(docs);
    String fullPrompt = promptBuilder.buildDefaultPrompt(
    new Prompt(context, queryDto.query)
    );
    return chatClient.prompt().user(fullPrompt).call().content();
}
...
```

Código 6.19: Serviço de recuperação na API

6.2.3 Extração de dados do Control-M (UC6)

Conforme descrito na análise, este caso de uso visa a extração automática de dados históricos relativos a *jobs* falhados no Control-M, o subsequente processamento desses dados e a sua integração no Confluence.

Após a extração dos dados da base de dados Oracle, conforme detalhado na Secção 6.1.3, procede-se ao processamento, que consiste na transformação dos dados recolhidos numa tabela em formato HTML. Para tal, foi desenvolvida uma solução de mapeamento de representações HTML, responsável pela construção dinâmica das linhas da tabela, conforme exemplificado no Código 6.20.

```
...

private Row getRow(FailedJobCtrlmRAG d) {
    return Row.build()
            .addCell(Cell.of(d.getFolder()))
            .addCell(Cell.of(d.getJob()))
            .addCell(Cell.of(d.getJobDescription()))
            .addCell(Cell.of(d.getOrderId()))
            .addCell(Cell.of(d.getOrderDate()))
            .addCell(Cell.of(d.getDay().toString()))
            .addCell(Cell.of(d.getDayOfWeek().toString()))
            .addCell(Cell.of(d.getStartTime().toString()))
            .addCell(Cell.of(d.getEndTime().toString()))
            .addCell(Cell.of(""));
}
...

```

Código 6.20: Construção de linhas da tabela dos relatórios do Control-M

Através do Control-M Manager foi também definido o horário de execução periódica da extração para todos os domingos às 21 horas (Figura 6.5).

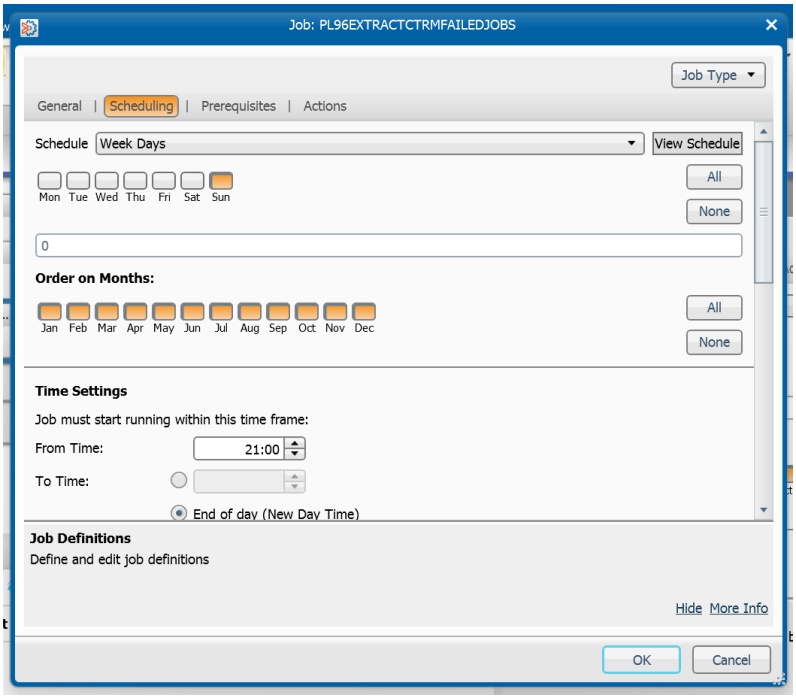


Figura 6.5: Definição do horário de execução

6.3 Interface Gráfica

Esta secção apresenta as interfaces gráficas desenvolvidas para o sistema RAG, evidenciando a forma como os utilizadores interagem com as funcionalidades implementadas.

6.3.1 Página de Recuperação

A Figura 6.6 ilustra a interface de recuperação assistida.

Nesta página, o utilizador pode submeter perguntas ao sistema, que serão processadas pela *pipeline* RAG desenvolvida.

Os principais elementos da interface incluem:

- **Campo de input de pergunta:** permite inserir a *query*, que é submetida à API assim que o botão de envio for acionado;
- **Área de respostas:** apresenta as respostas fornecidas pelo sistema e o histórico;
- **Filtros de pesquisa:** permitem ajustar alguns parâmetros como *top-k chunks*, grau de similaridade e filtros de documentação.

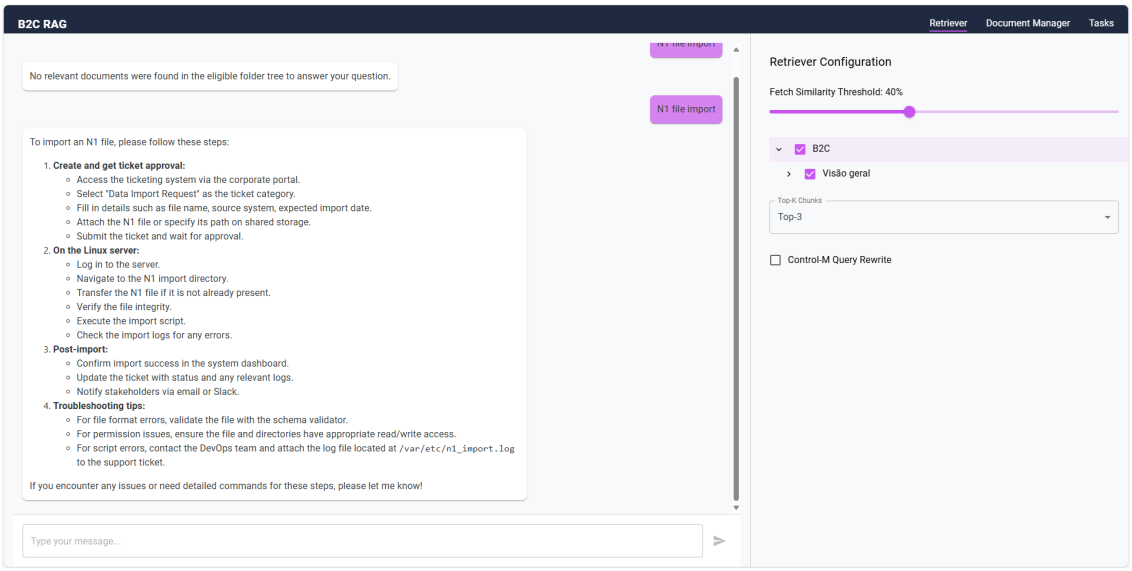


Figura 6.6: Interface de recuperação

6.3.2 Página de Gestão de Documentos

A Figura 6.7 ilustra a interface de gestão de documentos. Esta página permite ao utilizador visualizar e realizar pesquisas de conteúdos indexados no Confluence, bem como gerir as exclusões.

Os elementos visuais relevantes incluem:

- **Tabelas de indexação e exclusão**: tabelas contendo informação geral sobre as páginas indexadas como ID, título, categoria e datas de criação, atualização e indexação;
- **Filtros e pesquisa**: barra de pesquisa e possibilidade de aplicação de filtros individuais sobre as colunas;
- **Ações**: estão disponíveis ações para indexação manual — no caso de a automática falhar — exclusão de páginas e restauração de páginas.

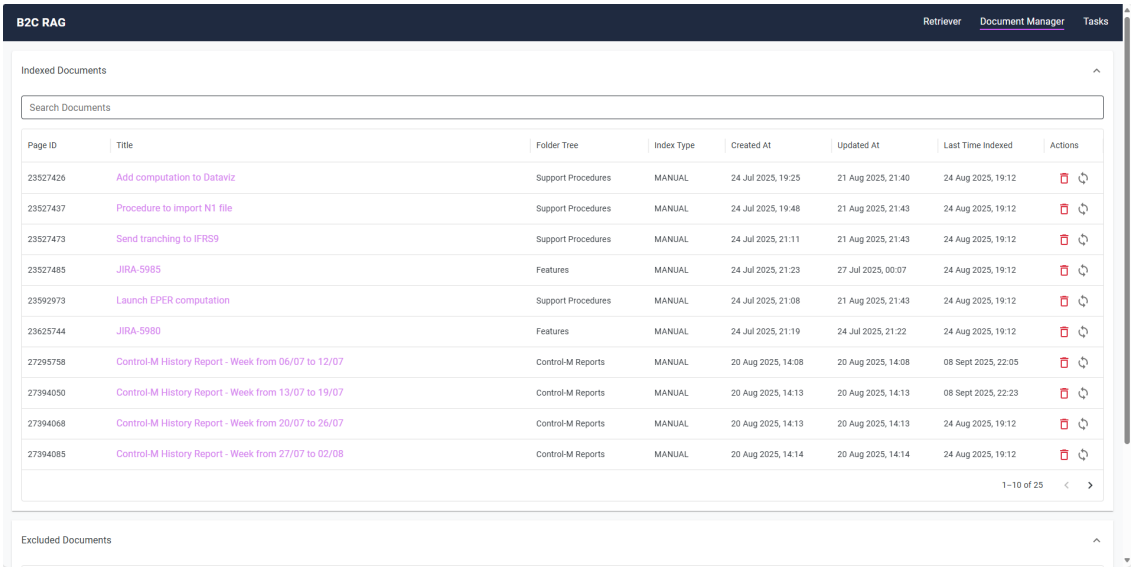


Figura 6.7: Interface de visualização e gestão de páginas

Capítulo 7

Avaliação da Solução

O presente capítulo dedica-se à avaliação sistemática da solução desenvolvida, tendo por base uma metodologia assente em métricas quantitativas e qualitativas. Serão analisadas a eficácia e a consistência da geração de respostas pelo sistema RAG, bem como a qualidade técnica do seu código, incluindo a cobertura de testes, complexidade e manutenibilidade. Complementarmente, inclui-se a recolha de *feedback* dos utilizadores, permitindo aferir a perceção e aceitação da proposta. Esta abordagem integrada visa fornecer uma visão crítica e fundamentada sobre o desempenho e a robustez da solução, em consonância com os objetivos definidos inicialmente.

7.1 Metodologia

A metodologia adotada para a avaliação da solução seguiu a abordagem Goal Question Metric (GQM) (Caldiera e Rombach, 1994). O processo iniciou-se com a formulação clara dos objetivos de avaliação, orientados para a verificação da eficácia e eficiência da solução proposta. Em seguida, esses objetivos foram decompostos em questões específicas, concebidas para orientar a recolha de evidências empíricas de forma sistemática. Por fim, foram definidas métricas objetivas que permitiram quantificar as respostas a essas questões, assegurando a comparabilidade e a reprodutibilidade dos resultados. A Figura 7.1 representa a estrutura do modelo GQM seguido.

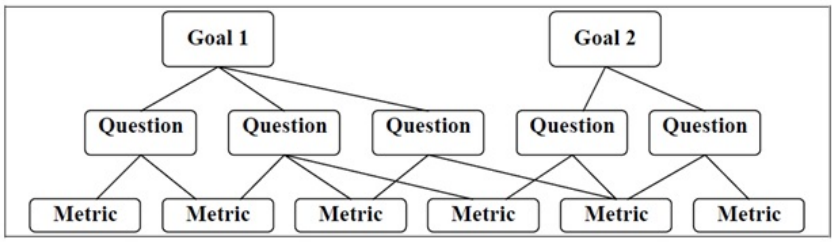


Figura 7.1: Estrutura do modelo GQM

Na seguinte secção, procede-se à fundamentação da abordagem de avaliação do sistema RAG, que serve de justificação para a definição das métricas utilizadas, e apresenta-se a estrutura GQM definida para a presente avaliação.

7.1.1 Abordagem de Avaliação da Qualidade do Sistema RAG

Na Secção 3.6 foram apresentadas as métricas de avaliação ROUGE, BERTScore e LLM-based, ferramentas utilizadas para medir a qualidade das respostas geradas por um sistema RAG.

Após uma análise cuidada, com o objetivo de escolher a métrica mais adequada para prosseguir com os testes, as métricas ROUGE e BERTScore foram descartadas por apresentarem algumas limitações. A métrica ROUGE baseia-se na comparação superficial, o que pode não capturar a verdadeira semântica das respostas. Já a métrica BERTScore, embora utilize representações semânticas, não avalia diretamente a consistência factual, tendo produzido resultados artificialmente elevados em alguns testes efetuados.

Optou-se pela utilização de LLMs, uma vez que esta abordagem permite realizar uma avaliação mais avançada e qualitativa das respostas geradas.

Processo

O processo de avaliação ocorreu de forma iterativa para cada par questão–resposta de referência (Figura 7.2).

Primeiramente, o sistema de testes recebe os seguintes parâmetros: par questão–resposta, grau de similaridade pretendido, *top-k chunks* e filtros (1). A configuração dos LLMs é realizada diretamente no sistema RAG, através do ficheiro *application.properties*. Em seguida, a resposta candidata é obtida pela chamada ao *endpoint POST /query* (2).

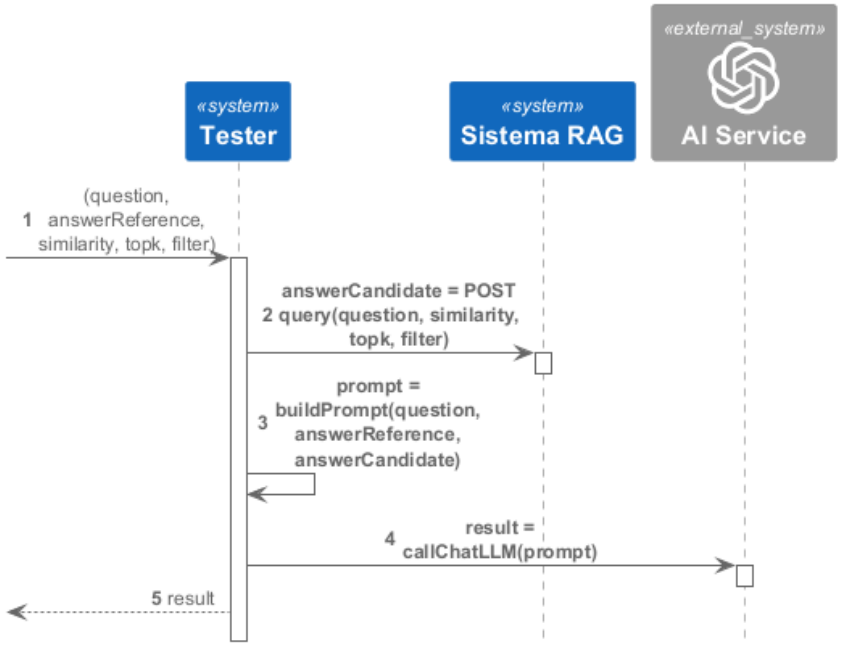


Figura 7.2: Processo de avaliação

Por fim, o *prompt* de avaliação é construído e submetido ao modelo LLM de *chat*, que gera um resultado numérico entre 1 e 5, com base na descrição presente na legenda de pontuações da Tabela 7.1 (3-5). De forma a verificar a qualidade do sistema, definiu-se que uma pontuação igual ou superior a 4 está dentro dos padrões de qualidade desejados.

O *prompt* e excerto do algoritmo de avaliação encontram-se na Figura A.2 e Código B.1, respetivamente. O modelo LLM utilizado para a avaliação foi o *gpt-4o-mini*.

Tabela 7.1: Legenda de pontuações

Pontuação	**Descrição**
1	Completamente irrelevante
2	Ligeiramente relacionado, mas maioritariamente incorreto ou irrelevante
3	Parcialmente relacionado, parcialmente correto, mas com elementos importantes em falta
4	Maioritariamente correto, com pequenas diferenças ou informação extra irrelevante
5	Perfeitamente correto, corresponde totalmente ao significado da referência

7.1.2 Estrutura GQM

Primeiramente, procedeu-se à definição de métricas partilhadas entre diferentes questões do GQM (Tabela 7.2). As métricas M1, M2 e M3 centram-se nos valores de pontuação referidos anteriormente na Tabela 7.1 e são utilizadas para avaliar a qualidade das repostas geradas. As restantes métricas complementam a avaliação anterior, abrangendo dimensões de satisfação subjetiva, qualidade do código e manutenibilidade, permitindo uma análise mais completa e objetiva do sistema em estudo.

Tabela 7.2: Métricas do GQM

Código	**Métrica**	**Alvo de Referência**
M1	Pontuação média (1-5)	≥ 4.5
M2	Percentagem de respostas com pontuação ≥ 4	$\geq 90\%$
M3	Desvio padrão da pontuação	≤ 1
M4	Escala Likert (1-5)	≥ 3
M5	Percentagem de cobertura de código	$\geq 80\%$
M6	Complexidade ciclomática média	≤ 5
M7	Índice de manutenibilidade	$\geq 80\%$

A estrutura GQM definida encontra-se na Tabela 7.3.

Tabela 7.3: Estrutura GQM

Código	Objetivo(G)/Questão(Q)	**Métrica** (Tabela 7.2)
G1	**Analisar o sistema RAG para avaliar a eficácia das respostas, com respeito à correção, relevância e fundamentação, do ponto de vista da equipa de suporte, no contexto de documentação geral**	
Q1.1	Como variam qualidade (média), consistência (desvio padrão) e proporção de respostas aceitáveis (≥ 4) por modelo de *chat*?	M1, M2, M3
Q1.2	Como variam qualidade, consistência e proporção de respostas aceitáveis por modelo de *embedding*?	M1, M2, M3
Q1.3	Qual o impacto de *similarity threshold* e *top-k* na qualidade, consistência e proporção de respostas aceitáveis?	M1, M2, M3
G2	**Analisar o sistema RAG para avaliar a eficácia das respostas no contexto de Control-M, incluindo um módulo de reescrita para avaliar a capacidade de mapear linguagem natural para o esquema dos relatórios (semanas/intervalos)**	
Q2.1	Como variam qualidade, consistência e proporção de respostas aceitáveis com e sem reescrita no contexto de Control-M	M1, M2, M3
G3	**Analisar a proposta do sistema RAG para avaliar a clareza da apresentação e a utilidade percebida, do ponto de vista da equipa de suporte, no contexto de uma sessão interna**	
Q3.1	A apresentação foi clara?	M4
Q3.2	A proposta é relevante para a equipa?	M4
Q3.3	Qual a viabilidade e aceitação da proposta?	M4
G4	**Analisar o código do sistema RAG para avaliar robustez e manutenibilidade, do ponto de vista da engenharia de software**	
Q4.1	Qual a cobertura de testes automatizados (unitários e integração)?	M5
Q4.2	Qual é o nível de complexidade do código?	M6
Q4.3	O código é fácil de manter e evoluir?	M7

É de notar que o âmbito da avaliação realizada não abrange testes não funcionais de forma detalhada, focando-se apenas na vertente funcional do sistema RAG e qualidade do código.

7.2 Análise dos Resultados

Procede-se, de seguida, à análise dos resultados que dão reposta aos objetivos e questões apresentados na estrutura GQM, associando as suas respetivas métricas.

7.2.1 G1 e G2 - Análise da Qualidade do Sistema RAG

Os testes realizados incidiram sobre duas vertentes principais: a recuperação de documentação geral de suporte e a recuperação de dados históricos do Control-M. Esta abordagem permitiu explorar o isolamento na recuperação, possibilitado pela definição dos filtros de categoria de documentos implementados.

Para simplificar a nomenclatura utilizada nas secções seguintes, foi elaborada uma legenda com os critérios de configuração dos testes, apresentada na Tabela 7.4.

Tabela 7.4: Legenda de critérios de configuração dos testes

Sigla	Descrição
E	Modelo de *embedding*
C	Modelo de *chat*
ST	Similarity Threshold
TK	Top-K *chunks*
N	Número de perguntas efetuadas
ND	Número de documentos indexados

Conjunto de dados

O conjunto de dados utilizado é constituído por 138 *queries*, das quais 108 se encontram associadas à documentação geral de suporte e 30 dizem respeito a relatórios do Control-M. Foram igualmente considerados 33 documentos de referência, dos quais 25 correspondem a documentação geral e 8 a relatórios do Control-M. Este conjunto de dados foi construído a partir de cenários reais de utilização pela B2C, apresentando, assim, uma dimensão adequada e uma diversidade suficiente para a validação da solução proposta.

7.2.1.1 Comparação entre LLMs de Chat

Critérios do cenário de teste:

- C = *gpt-3.5-Turbo, gpt-4o-mini, gpt-4.1, gpt-4o, gpt-4.1-mini*
- E = *text-embedding-3-large*
- ST = 40%
- TK = 5
- N = 108
- ND = 25

Este teste serviu para analisar como se comporta o sistema com a variação do modelo LLM de *chat* (C). Tomou por base uma amostra composta por 25 documentos e 108 perguntas, relacionados com documentação geral de suporte. Optou-se por utilizar um valor de ST baixo e TK alto de forma a se recuperar um contexto maior. Desta forma conseguimos avaliar a capacidade do modelo de chat de interpretar grandes quantidades de informação contextual. Os resultados encontram-se ilustrados na Tabela 7.5 e na Figura 7.3.

Tabela 7.5: Comparação dos modelos de *chat* em termos de pontuação média e desvio padrão.

Modelo de Chat	Pontuação Média	Desvio Padrão
gpt-3.5-Turbo	4,37	1,14
gpt-4o-mini	4,57	0,99
gpt-4.1	4,34	1,36
gpt-4o	4,34	1,30
gpt-4.1-mini	4,63	0,96

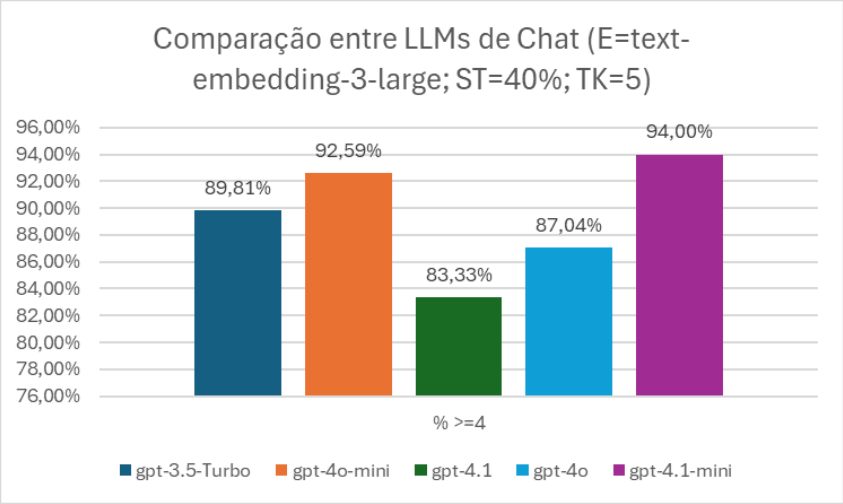


Figura 7.3: Desempenho de modelos de *Chat*

Pela análise dos resultados, é notável a superioridade de modelos mais pequenos como, *gpt-4.1-mini* e *gpt-4o-mini*, destacando-se o primeiro por apresentar uma pontuação média de 4,63 e um desvio padrão reduzido, assegurando boa consistência.

7.2.1.2 Comparação entre LLMs de Embedding

Critérios do cenário de teste:

- C = *gpt-4.1-mini*
- E = *text-embedding-3-large, text-embedding-3-small, text-embedding-ada-002*
- ST = 40%
- TK = 3
- N = 108
- ND = 25

Este teste serviu para analisar como se comporta o sistema com a variação do modelo LLM de *embedding* (E). Tomou por base uma amostra composta por 25 documentos e 108 perguntas, relacionados com documentação geral de suporte. Optou-se por utilizar um valor

de TK baixo de forma a se obter um contexto mais reduzido. Desta forma, conseguimos avaliar a capacidade do modelo de *embedding* de incluir representações semânticas mais próximas da realidade. O *gpt-4.1-mini* foi utilizado como modelo de *chat*, sendo este o mais eficaz com base no teste anterior. Os resultados encontram-se ilustrados na Tabela 7.6 e na Figura 7.4.

Tabela 7.6: Comparação dos modelos de *embedding* em termos de pontuação média e desvio padrão.

Modelo de Embedding	Pontuação Média	Desvio Padrão
text-embedding-3-large	4,59	0,99
text-embedding-3-small	4,37	1,24
text-embedding-ada-002	4,16	1,46

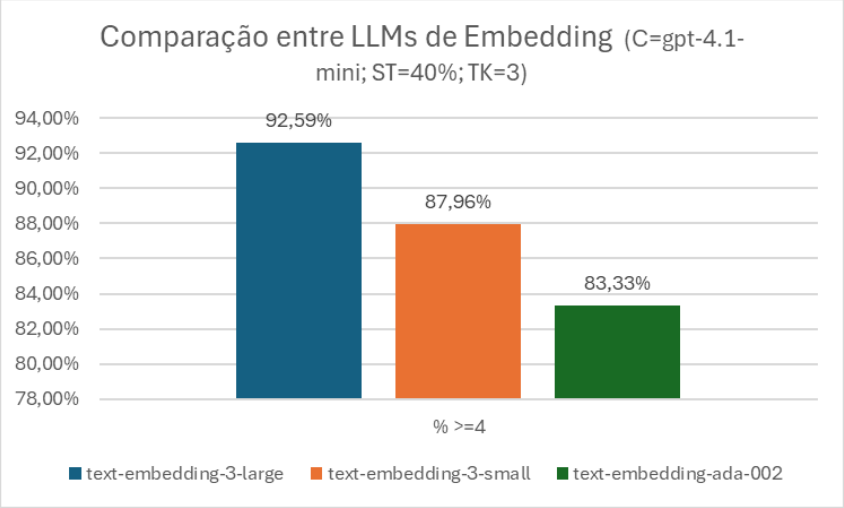


Figura 7.4: Desempenho de modelos de *Embedding*

Através da análise dos resultados, é possível concluir que o modelo *text-embedding-3-large* apresenta-se superior por ter uma percentagem de notas maior ou igual a 4 de 92.59%. Este modelo apresenta 3072 dimensões enquanto que os outros apenas 1536. Isto leva-nos a concluir que quantas mais dimensões semânticas o modelo tiver, melhor são os resultados no sistema RAG.

7.2.1.3 Relação entre ST e TK

Critérios do cenário de teste:

- C = *gpt-4.1-mini*
- E = *text-embedding-3-large*
- ST = 20%, 40%, 60%
- TK = 3, 5, 8
- N = 108

- ND = 25

Este teste serviu para analisar como se comporta o sistema com a variação ST e TK. Tomou por base uma amostra composta por 25 documentos e 108 perguntas, relacionados com documentação geral de suporte. O LLM de *chat* foi fixado a *gpt-4.1-mini* e o modelo de *embedding* a *text-embedding-3-large*. Fez-se variar três valores de ST e três valores de TK, constituindo uma combinação de 9 testes. Os resultados encontram-se ilustrados na Figura 7.5 e na Tabela 7.7.

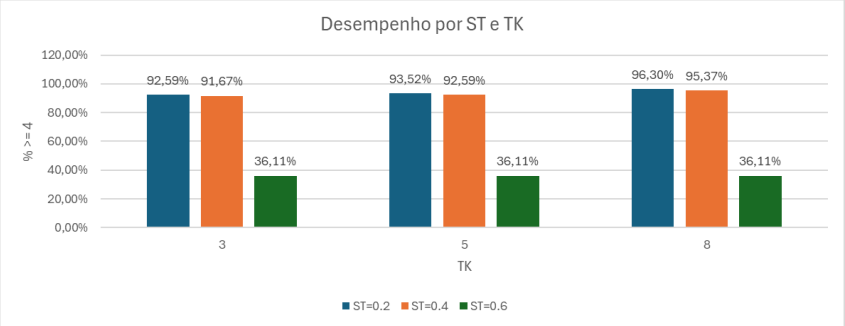


Figura 7.5: Desempenho por ST e TK

Tabela 7.7: Comparação de resultados face à variação de ST e TK

TK	ST	Pontuação Média	Desvio Padrão
3	20%	4,59	0,94
3	40%	4,52	1,07
3	60%	2,40	1,87
5	20%	4,64	0,85
5	40%	4,60	0,97
5	60%	2,37	1,84
8	20%	4,74	0,67
8	40%	4,67	0,88
8	60%	2,40	1,87

Com base na análise dos resultados, observa-se que o desempenho é significativamente menor para todos os TK quando o ST é de 60%. Por outro lado, a variação do TK com ST de 20% ou 40% mostra resultados bastante positivos, destacando-se o caso de TK = 8, que apresenta o melhor desempenho. Além disso, quanto maior o número de *chunks* recuperados, mais corretas tendem a ser as respostas. Já valores de ST acima de 40% demonstram tendência a gerar respostas irrelevantes.

7.2.1.4 Recuperação de dados históricos do Control-M

Critérios do cenário de teste:

- C = *gpt-4.1-mini*
- E = *text-embedding-3-large*
- ST = 40%

- TK = 1
- N = 30
- ND = 8

Este teste serviu para analisar se o sistema é capaz de gerar respostas corretas no contexto de dados históricos do Control-M. Tomou por base uma amostra composta por 8 relatórios e 30 perguntas, ambos relacionados exclusivamente com os dados históricos de execução dos *jobs*. O LLM de *chat* foi fixado a *gpt-4.1-mini* e o modelo de *embedding* a *text-embedding-3-large*.

Os relatórios do Control-M estão organizados no Confluence em estruturas de dados fixas. Com base nessas estruturas e no tipo de questões habitualmente colocadas sobre estes dados, foram utilizadas duas abordagens de recuperação: com e sem reescrita de *query*.

A reescrita de *query* consiste na inclusão de elementos adicionais à *query* inicial, permitindo ao recuperador semântico captar melhor certas palavras-chave presentes nos relatórios. Por exemplo:

- *Query* inicial: Que jobs falharam na segunda semana de julho?
- *Query* reescrita: Procura os jobs falhados no "Control-M History Report - Week from 06/07 to 12/07"

Neste exemplo, assumindo que o termo "segunda semana de julho"não está presente na base de conhecimento, o recuperador não teria forma de encontrar os dados pretendidos. Com a reformulação da *query*, torna-se possível interpretar este tipo de pesquisa, otimizando-a para casos de uso reais.

O *prompt* formulado para a reescrita da *query* encontra-se no Código A.3.

Devido ao facto de a amostra ser reduzida, o TK foi fixado a 1. Desta forma, centramos a avaliação na recuperação vetorial e conseguimos analisar com mais rigor as diferenças entre as duas abordagens.

Os resultados encontram-se ilustrados na Figura 7.6 e na Tabela 7.8.

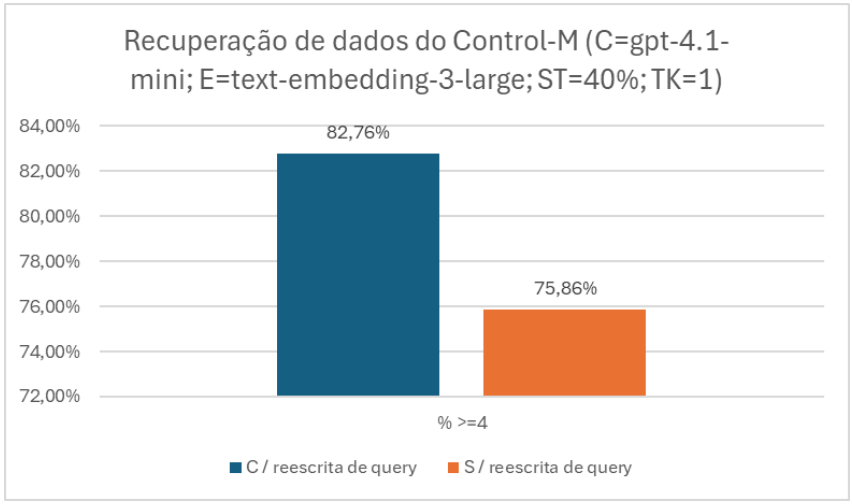


Figura 7.6: Desempenho com e sem reescrita de *query*

Tabela 7.8: Avaliação da recuperação de dados do Control-M com ou sem reescrita de *query*

Reescrita de query	Pontuação Média	Desvio Padrão
Sim	4,21	1,27
Não	4,0	1,31

Pela análise dos resultados, podemos concluir que a utilização de reescrita de *query* tem um impacto significativo a nível percentual de pontuações iguais ou superiores a 4, provando assim a eficácia desta abordagem no que diz respeito à recuperação de dados estruturados.

7.2.1.5 Conclusões

Com base na experimentação realizada, é possível retirar algumas conclusões relevantes sobre o desempenho do sistema. A análise evidencia que modelos de menor dimensão, como o *gpt-4.1-mini* e o *gpt-4o-mini*, revelam resultados bastante consistentes, superando modelos maiores em termos de relação de qualidade. Do lado dos *embeddings*, confirma-se a superioridade do *text-embedding-3-large*, o que reforça a importância da dimensionalidade na representação semântica.

No que respeita à parametrização, observa-se que valores de ST acima de 40% prejudicam significativamente a recuperação no contexto deste projeto, enquanto valores mais baixos, aliados a um TK superior, potenciam respostas mais completas e precisas. Já no caso do *Control-M*, a inclusão da reescrita de *queries* demonstrou ganhos claros, ainda que moderados, evidenciando o seu papel como técnica útil para lidar com dados estruturados e pesquisas ambíguas.

Concluindo, o sistema protótipo desenvolvido apresenta resultados satisfatórios no que diz respeito à recuperação de documentação, com especial destaque para a documentação geral de suporte. A recuperação de dados do Control-M, contudo, revelou-se aquém do desejado, apresentando ainda bastante espaço para melhorias com vista à sua aplicação em contextos reais. Estas melhorias poderão passar pela integração de diversas técnicas alternativas analisadas no estado da arte, como a utilização de recuperação híbrida, pela combinação de modelos lexicais com modelos semânticos, ou até mesmo pela adoção de uma abordagem de armazenamento orientada a grafos, capaz de estruturar de forma mais eficiente os metadados dos *jobs* e potenciar consultas mais complexas e contextualmente relevantes.

7.2.2 G3 - Análise de Feedback dos Utilizadores

De forma a complementar a avaliação do sistema com evidências qualitativas, foi realizada uma sessão interna com elementos da equipa de suporte com o objetivo de apresentar a solução protótipo e recolher opiniões.

A sessão foi organizada pela seguinte estrutura cronológica:

1. **Contextualização da tecnologia**: iniciou-se com um breve enquadramento conceptual do RAG, destacando-se os seus princípios de funcionamento e arquitetura;
2. **Apresentação do desenho da solução e principais fluxos**: em seguida, foi detalhada a arquitetura proposta, com descrição dos componentes do sistema e casos de uso para integração com o Confluence e Control-M;

3. **Demonstração prática**: por fim, procedeu-se à execução de um protótipo funcional, ilustrando casos de utilização representativos, e simulando situações concretas do quotidiano da equipa.

No fim da sessão, foi encaminhado aos elementos da equipa um pequeno questionário retrospetivo. A estrutura do questionário assenta numa escala de Likert de cinco pontos (1 = Discordo totalmente; 5 = Concordo totalmente), permitindo avaliar três dimensões centrais:

- **Clareza da apresentação**: compreensão, organização e exemplos apresentados;
- **Relevância da proposta**: adequação às necessidades da equipa e utilidade prática;
- **Viabilidade e aceitação**: perceção de exequibilidade técnica e interesse em adoção futura.

O questionário contou com respostas de seis elementos da equipa e os resultados estão apresentados na Tabela 7.9.

Tabela 7.9: Questionário retrospetivo da sessão

Questão	**1**	**2**	**3**	**4**	**5**
A apresentação do sistema RAG foi clara e fácil de compreender.	—	—	—	33,30%	66,70%
A sessão estava bem estruturada e organizada.	—	—	—	16,70%	83,30%
Os exemplos e explicações utilizados ajudaram a entender a proposta.	—	—	—	16,70%	83,30%
A proposta apresentada responde às necessidades da equipa.	—	—	—	16,70%	83,30%
O sistema RAG é relevante para o meu trabalho/atividades diárias.	—	—	—	16,70%	83,30%
O projeto poderá contribuir para melhorar a eficiência da equipa.	—	—	—	16,70%	83,30%
Considero a solução tecnicamente viável a curto e médio prazo.	—	—	16,70%	50%	33,30%
Vejo potencial de adoção do sistema pela equipa.	—	—	—	16,70%	83,30%
Estou interessado em participar em futuras fases de testes ou pilotos.	—	—	—	16,70%	83,30%

De forma geral, os resultados do questionário evidenciam uma perceção bastante positiva da sessão e da solução apresentada. A clareza da exposição, a organização da sessão e a relevância da proposta obtiveram avaliações muito favoráveis, maioritariamente situadas nos níveis 4 e 5 da escala de Likert. Verifica-se igualmente uma forte concordância quanto ao potencial contributo do sistema para a eficiência da equipa e ao interesse em participar em futuras fases de teste. A dimensão onde se registou maior prudência prende-se com a viabilidade técnica a curto e médio prazo, onde se observaram respostas mais distribuídas, refletindo uma postura de cautela relativamente à concretização imediata do projeto. No seu conjunto, os dados recolhidos sugerem uma aceitação elevada e um interesse efetivo na evolução da solução.

7.2.3 G4 - Análise da Qualidade do Código

A análise da qualidade do código centrou-se apenas no sistema RAG desenvolvido em Spring Boot, devido ao facto de ser o artefacto principal, e por implementar a lógica de negócio relevante para a investigação.

As seguintes secções apresentam os testes automatizados realizados, bem como testes de complexidade e manutenibilidade.

7.2.3.1 Testes Automatizados

Os testes automatizados realizados no projeto Spring Boot, em Java, foram desenvolvidos com o auxilio de duas ferramentas: JUnit e Mockito (Kaczanowski, 2013). A principal finalidade do JUnit consiste em automatizar a validação de pequenos blocos de código, garantindo que o seu comportamento corresponde ao esperado. A adoção desta ferramenta promove a deteção precoce de erros, facilita a manutenção do *software* e incentiva a prática de desenvolvimento orientado a testes. O Mockito, por sua vez, é uma biblioteca que complementa o JUnit ao possibilitar a criação de objetos simulados (*mocks*). Estes objetos permitem isolar unidades de código durante o processo de teste, substituindo dependências externas e recriando cenários específicos de execução. Desta forma, o Mockito contribui para aumentar a precisão e a abrangência dos testes, assegurando maior fiabilidade na validação de funcionalidades complexas.

Segue-se uma breve contextualização dos testes efetuados.

Testes Unitários

De forma a testar os componentes e métodos do código de forma isolada, foram elaborados testes unitários. Para isolar cada método, foram utilizados *mocks* que simulam o comportamento de lógica externa.

O Código 7.1 apresenta um exemplo de teste unitário de um cenário positivo, de recuperação assistida por IA, onde existem documentos presentes no resultado. Para este cenário foram criados *mocks* do *vectorStore*, para simular a recuperação de vetores, *promptBuilder*, para a construção do *prompt* final, e *chatClient* para simular uma chamada ao LLM. Com o *verify* é possível verificar quantas vezes os métodos foram chamados e qual o conteúdo de entrada.

```
...
@Test
void query_shouldReturnAnswer_whenDocumentsFound() {
    // Arrange
    UserQueryDto dto = new UserQueryDto();
    dto.query = "What is AI?";
    dto.eligibleFolderTree = List.of("root");
    dto.similarityThreshold = 0.8;
    dto.topK = 3;
    Document doc = new Document("Hello World!");
    when(vectorStore.similaritySearch(any(SearchRequest.class))).thenReturn(List.of(
    doc));
    when(promptBuilder.buildContext(anyList())).thenReturn("Context");
    when(promptBuilder.buildDefaultPrompt(any())).thenReturn("Full Prompt");
    // Act
    String result = queryService.query(dto);
    // Assert
    assertEquals("Mocked Response", result);
    verify(vectorStore, times(1)).similaritySearch(any(SearchRequest.class));
    verify(promptBuilder, times(1)).buildContext(List.of(doc));
    verify(promptBuilder, times(1)).buildDefaultPrompt(any());
    verify(chatClient, times(1)).prompt();
}
...

```

Código 7.1: Cenário positivo - Recuperação com documentos presentes

O Código 7.2 apresenta um exemplo de teste unitário de um cenário negativo, de recuperação de uma página no Confluence, onde é expectável um problema de conexão. Desta forma, conseguimos verificar que o sistema cumpre a sua função de lançar exceções em cenários de erro.

```
...
@Test
void testGetPageContentThrowsConfluenceExceptionOnUnirestException() throws
    Exception {
    // Arrange
    String pageId = "123";
    try (MockedStatic<Unirest> unirestMockedStatic = mockStatic(Unirest.class)) {
        unirestMockedStatic.when(() -> Unirest.get(anyString()))
        .thenAnswer(i -> {
            throw new RuntimeException(new UnirestException("Network error"));
        });
        var method = ConfluenceApiClient.class.getDeclaredMethod("getPageContent",
    String.class);
        method.setAccessible(true);
        // Act / Assert
        assertThrows(Exception.class, () -> method.invoke(client, pageId));
    }
}
...

```

Código 7.2: Cenário negativo - Recuperação de conteúdo de página do Confluence com problema de conexão

Testes de Integração

Os testes de integração têm como objetivo validar o comportamento da aplicação de ponta a ponta, assegurando que os diferentes componentes interagem corretamente entre si. No presente caso, o ponto de entrada da aplicação situa-se no *Controller*, onde estão definidos os *endpoints* utilizados por aplicações externas para comunicação.

Para a execução destes testes, recorreu-se ao *MockMvc*, uma ferramenta disponibilizada pelo Spring Framework, que permite simular chamadas HTTP de forma controlada, sem a necessidade de iniciar um servidor real. Com esta abordagem, é possível verificar o correto funcionamento das rotas, parâmetros e respostas da aplicação, garantindo que a lógica de negócio é respeitada em todo o fluxo (Código 7.3).

De modo a isolar a lógica interna da aplicação, e evitar dependências externas durante os testes, foram criados *mocks* apenas para os serviços externos: *vectorStore*, *qdrantClient*, *documentRepository* e *confluenceApiClient*.

```
...
@Autowired
private MockMvc mockMvc;
@MockitoBean
private VectorStore vectorStore;
@MockitoBean
private QdrantClient qdrantClient;
@MockitoBean
private DocumentRepository documentRepository;
@MockitoBean
private ConfluenceApiClient confluenceApiClient;
...

```

Código 7.3: *Beans* do Spring Simulados

O Código 7.4 apresenta um exemplo de teste de integração efetuado, neste caso para testar um cenário de indexação de uma nova página através de um *webhook* vindo do Confluence (*mockMvc.perform(post("/webhook"))*). Com este tipo de testes, validamos comportamentos de entrada e saída de forma a garantir um fluxo correto de ponta a ponta.

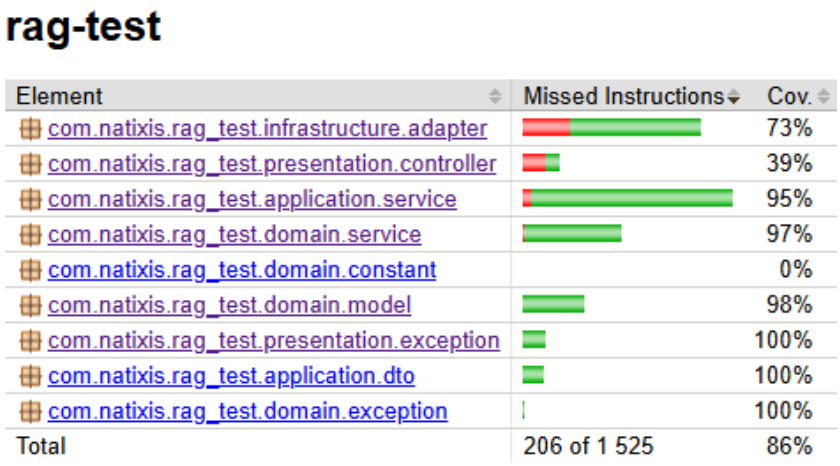
```
...
@Test
void handleWebhook_confluenceCreatePage_indexesPageAndVectors() throws Exception {
    // Arrange
    when(documentRepository.findById(pageId)).thenReturn(Optional.empty());
    when(documentRepository.save(any(Page.class))).thenAnswer(inv -> inv.getArgument
    (0));
    when(confluenceApiClient.getPageContent(pageId)).thenReturn("ok");
    when(confluenceApiClient.getConfluencePage(pageId)).thenReturn(confluencePage);
    // Act
    mockMvc.perform(post("/webhook")
    .contentType(MediaType.APPLICATION_JSON)
    .content(webhookJson.getBytes(StandardCharsets.UTF_8)))
    .andExpect(status().isOk());
    // Assert
    verify(confluenceApiClient, times(1)).getPageContent(pageId); // used by
    isPageDeleted()
    verify(confluenceApiClient, times(1)).getConfluencePage(pageId); // used by
    indexDocument()
    verify(documentRepository, times(1)).findById(pageId);
    verify(documentRepository, times(1)).save(any(Page.class));
    verify(qdrantClient, never()).deleteAsync(anyString(), any(Points.Filter.class))
    ;
    ArgumentCaptor<List<Document>> docsCaptor = ArgumentCaptor.forClass(List.class);
    verify(vectorStore, times(1)).accept(docsCaptor.capture());
    List<Document> docs = docsCaptor.getValue();
    assertThat(docs).isNotNull();
    Document first = docs.get(0);
    assertThat(first.getFormattedContent()).isNotBlank();
    assertThat(first.getMetadata().get(CONFLUENCE_PAGE_ID)).isEqualTo(pageId);
    assertThat(first.getMetadata().get(PAGE_TITLE)).isEqualTo(pageTitle);
}
...

```

Código 7.4: *Webhook* de criação de página

Cobertura

Com o objetivo de assegurar a fiabilidade e a qualidade da aplicação, foi definida uma cobertura mínima de testes de 80%, recorrendo ao *plugin* JaCoCo. Esta prática está alinhada com as recomendações mais comuns, contribuindo para a deteção precoce de erros e para a manutenção de um código robusto e sustentável.

Os testes foram concebidos para abranger tanto cenários positivos como negativos, validando o comportamento dos componentes perante diferentes condições de entrada. Para demonstrar os resultados obtidos, foi gerado um relatório de cobertura, através do JaCoCo, sendo apresentado na Figura 7.7 um exemplo relativo à cobertura global do projeto.



rag-test

Element	Missed Instructions	Cov.
com.natixis.rag_test.infrastructure.adapter		73%
com.natixis.rag_test.presentation.controller		39%
com.natixis.rag_test.application.service		95%
com.natixis.rag_test.domain.service		97%
com.natixis.rag_test.domain.constant		0%
com.natixis.rag_test.domain.model		98%
com.natixis.rag_test.presentation.exception		100%
com.natixis.rag_test.application.dto		100%
com.natixis.rag_test.domain.exception		100%
Total	206 of 1 525	86%

Figura 7.7: Cobertura de testes

7.2.3.2 Complexidade e Manutenibilidade

A avaliação da complexidade e da manutenibilidade do código do sistema RAG foi realizada com recurso à ferramenta *SonarGraph*.

No que respeita à complexidade, foi utilizada uma métrica de complexidade ciclomática média, sendo o valor obtido de 1,67, o que indica uma baixa complexidade das funções implementadas. Este resultado significa que os métodos apresentam uma estrutura de controlo simples, com reduzido número de ramificações e caminhos alternativos de execução. Consequentemente, o esforço necessário para compreender, testar e modificar cada função é mínimo, o que favorece a clareza e a previsibilidade do comportamento do sistema (Figura 7.8).

Figura 7.8: Complexidade ciclomática

Relativamente ao índice de manutenibilidade, o sistema atingiu o valor de 80, sendo 100 o valor máximo. Em termos práticos, este índice traduz a facilidade em realizar alterações, corrigir erros e adicionar novas funcionalidades, reduzindo a probabilidade de introdução de falhas durante a evolução do *software* (Figura 7.9).

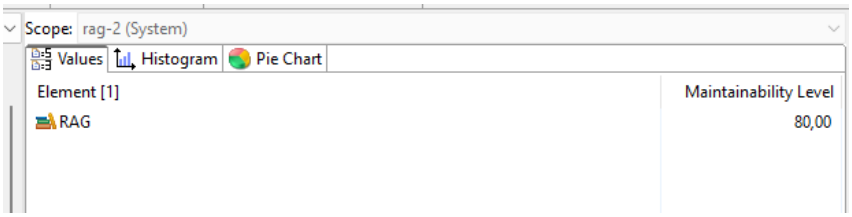


Figura 7.9: Índice de manutenibilidade

Em síntese, os resultados obtidos revelam que o sistema apresenta simultaneamente baixa complexidade e elevada manutenibilidade.

7.2.3.3 Conclusões

Os testes de código realizados garantem a fiabilidade do sistema RAG, abrangendo cenários positivos e negativos com uma cobertura desejada. A baixa complexidade ciclomática e o elevado índice de manutenibilidade confirmam que o código é claro, fácil de compreender e apto a evoluir. Assim, o sistema revela boa qualidade, facilitando manutenção e extensão futuras.

Não obstante, o foco exclusívo desta análise no módulo Spring Boot e a dependência de *mocks* podem ocultar problemas de integração real com sistemas externos. Como trabalho futuro, seria recomendado a introdução de testes de mutações para aferir a eficácia dos testes, a implementação de testes de contrato com serviços externos, bem como testes de desempenho.

Capítulo 8

Conclusão

O presente capítulo sintetiza os principais contributos do trabalho desenvolvido, apresentando de forma integrada os resultados alcançados, as limitações identificadas e as perspetivas de continuidade futura.

8.1 Trabalho Realizado

Este trabalho percorreu, de forma progressiva e coerente, todas as etapas do DSR necessárias à conceção de um sistema RAG para apoio a atividades de suporte técnico, articulando fundamentação teórica, opções de desenho e validação experimental.

Inicialmente, procedeu-se à contextualização do problema no âmbito da equipa B2C, relacionado com a rotatividade da equipa de suporte, a extensa documentação e o conhecimento tático disperso, que reforçam a necessidade de soluções que agilizem o acesso à informação. Este problema motivou a definição clara dos objetivos do trabalho.

De seguida, estabeleceram-se os fundamentos teóricos essenciais, cobrindo IA, ML, DL, LLMs e FT, e discutindo em detalhe a *framework* RAG e o racional para a sua adoção face a alternativas. Esta base foi determinante para orientar decisões técnicas subsequentes. Foi realizada a revisão do estado da arte, incluindo a análise comparativa de abordagens de armazenamento (vetorial, relacional, documentos e grafos), práticas de indexação e estratégias de recuperação (semântica, lexical e híbrida), bem como de técnicas de *augmentation* e Prompt Engineering. Compararam-se tecnologias de *frontend*, *backend* e *frameworks* RAG, e analisou-se a integração com o Confluence e o Control-M. Esta análise inicial justificou as escolhas tecnológicas.

A fase de análise da solução definiu o modelo de domínio, requisitos funcionais e não funcionais, e casos de uso que alinham as necessidades operacionais com critérios de qualidade e de testabilidade.

Foi detalhado o desenho arquitetural segundo o modelo C4 (contexto, containers e componentes) e foram comparadas alternativas de sincronização com o Confluence (*polling* vs. *webhooks*), tendo-se adotado webhooks para atualizações em tempo real. Apresentaram-se ainda as alternativas de implantação e os diagramas de sequência da *pipeline* RAG e da extração de dados do Control-M.

Posteriormente, foi descrita a implementação dos módulos SPA (React), API (Spring Boot) e Report (Java) para extração e publicação no Confluence. Foram esclarecidos os fluxos de indexação, pesquisa e publicação, bem como os principais pontos de integração.

Por fim, apresentou-se a avaliação segundo o modelo GQM. Recorreu-se a métricas baseadas em LLMs para avaliar a qualidade das respostas e estudou-se o impacto de diferentes modelos de *chat* e *embeddings*, bem como dos parâmetros de recuperação (ST e TK). No contexto do Control-M, testou-se a reescrita de *queries* para mapear linguagem natural para o esquema dos relatórios. Complementarmente, avaliaram-se a qualidade técnica do código (cobertura, complexidade e manutenibilidade) e a perceção dos utilizadores em relação à solução desenvolvida. Os resultados evidenciam um desempenho robusto na documentação geral e ganhos com *embeddings* de maior dimensionalidade.

Tabela 8.1: Grau de realização dos objetivos definidos

Objetivo	**Grau de Realização**
O1 - Sistema RAG protótipo	Cumprido integralmente
O2 - Control-M	Cumprido a nível técnico

Em termos de realização dos objetivos (Tabela 8.1), o desenvolvimento do sistema RAG protótipo (O1) foi cumprido integralmente, com validação técnica e funcional, e cumprimento das métricas de qualidade definidas. O objetivo relativo à integração e exploração de dados do Control-M (O2) foi concretizado a nível técnico. Contudo, a recuperação não atingiu a métrica de precisão de referência ($\geq 90\%$), apesar de esforços de otimização mediante reescrita de *queries*.

8.2 Limitações

A principal limitação do trabalho desenvolvido prende-se com a impossibilidade de realizar testes operacionais completos em ambiente produtivo, decorrente de restrições da infraestrutura interna da organização. Esta limitação impediu a recolha de métricas diretas de desempenho em contexto real de utilização pela equipa de suporte.

Contudo, esta restrição foi mitigada através da adoção de uma estratégia de avaliação alternativa, assente na utilização de um conjunto de dados de teste suficientemente representativo e abrangente, capaz de integrar os diferentes cenários expectáveis de utilização prática. Esta abordagem permitiu assegurar a validade estatística dos resultados e conferiu robustez à análise experimental, ainda que não substitua integralmente a evidência empírica proveniente de um teste piloto em ambiente operacional.

8.3 Trabalho Futuro

O trabalho desenvolvido expõe oportunidades claras de evolução técnica e de operacionalização. Propõem-se as seguintes linhas de continuidade:

- Condução de um teste piloto operacional com a equipa de suporte, recolhendo tempos de resolução e *feedback* qualitativo, para dar prioridade a melhorias com base na utilização.
- Avaliação da qualidade do *software* e realização de testes não funcionais:
  - Introdução de testes por mutações para aferir a eficácia do conjunto de testes, elevando a deteção de falhas subtis na lógica de orquestração RAG;

  - Implementação de testes de contrato com serviços externos críticos (Confluence, serviço de IA, Qdrant) para estabilizar integrações e detetar quebras;
  - Execução de testes de desempenho e carga (latência de recuperação, *throughput* da API, tempo de indexação) para garantir objetivos de nível de serviço definidos nos requisitos não funcionais.
- Otimização a integração de dados do Control-M:
  - Revisão da estruturação dos relatórios e considerar índices dedicados para dados tabulares;
  - Especialização da recuperação para tabelas e expansão da reescrita de *queries* com deteção de intenções temporais (semanas/intervalos) e filtros por entidades (*jobs*, pastas);
  - Exploração da recuperação híbrida (BM25 + *reranking* semântico) para melhorar cobertura em *queries* lexicalmente específicas.
- Introdução de *prompt templates* parametrizados para tarefas específicas do domínio, como por exemplo:
  - Descrição detalhada de funcionalidades a partir de um identificador com URLs para fontes.
  - Redação de *emails* procedimentais. Respostas padrão com referências a evidências e passos.
  - Geração de listas de verificação operacionais a partir de incidentes típicos.
- Expansão de fontes de conhecimento para além do Confluence:
  - *Logs* e alertas operacionais, permitindo perguntas orientadas a eventos;
  - Repositórios de código, mensagens de *commit*, *READMEs* e *pipelines*.

8.4 Apreciação Final

O trabalho realizado permitiu demonstrar o potencial de aplicação de sistemas RAG no apoio ao suporte técnico, oferecendo à organização uma solução protótipo com ganhos promissores claros na eficiência de acesso à informação. Embora em fase experimental, a solução abre caminho para futuras evoluções e integração em contexto produtivo, alinhando-se com as necessidades concretas da equipa B2C.

Para o autor, a dissertação representou uma oportunidade de aprofundar conhecimentos em inteligência artificial, engenharia de *software* e integração de sistemas empresariais, consolidando competências técnicas e metodológicas aplicadas a um problema de relevância prática.

No plano académico e científico, este trabalho contribuiu para a reflexão sobre a adoção de *frameworks* RAG em ambientes corporativos, explorando tanto aspetos técnicos como de integração com fluxos já existentes. Os resultados obtidos reforçam a relevância desta abordagem e constituem um ponto de partida sólido para investigações futuras que visem a sua generalização e maturidade operacional.

Bibliografia

Ali, Owais (2025). «Retrieval Augmented Generation for Intelligent Querying of Databases and Documents». Em.

Atlassian (2024a). *Confluence Cloud REST API v2 Documentation*. Acedido a 13 jul. 2025. url: https://developer.atlassian.com/cloud/confluence/rest/v2/intro/#about.

– (2024b). *Get Started with Confluence: Overview Guide*. Acedido a 13 jul. 2025. url: https://www.atlassian.com/software/confluence/resources/guides/get-started/overview#about-confluence.

Barnett, Scott et al. (2024). «Seven failure points when engineering a retrieval augmented generation system». Em: *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*, pp. 194–199.

Bhandari, Prakash (2024). *Polling vs Webhooks*. https://www.prakashbhandari.com.np/posts/polling-vs-webhooks/. Acedido a 13 ago. 2025.

Bloch, Joshua (2008). *Effective java*. Addison-Wesley Professional.

BMC Software (2024a). *Control-M Automation API: Monthly Workload Automation Service*. Acedido a 13 jul. 2025. url: https://docs.bmc.com/xwiki/bin/view/Control-M-Orchestration/Control-M/workloadautomation/Control-M-Automation-API/ctmapimonthly/.

– (2024b). *Control-M: Application Workflow Orchestration*. Acedido a 13 jul. 2025. url: https://www.bmc.com/it-solutions/control-m.html.

Brown, Simon (2013). «Software architecture for developers». Em: *Coding the Architecture*.

Bruckhaus, Tilmann (2024). «Rag does not work for enterprises». Em: *arXiv preprint arXiv:2406.04369*.

Caldiera, Victor R Basili1 Gianluigi e H Dieter Rombach (1994). «The goal question metric approach». Em: *Encyclopedia of software engineering*, pp. 528–532.

Clegg, Dai e Richard Barker (1994). *Case method fast-track: a RAD approach*. Addison-Wesley Longman Publishing Co., Inc.

De Lucia, Andrea, Abdallah Qusef et al. (2010). «Requirements engineering in agile software development». Em: *Journal of emerging technologies in web intelligence* 2.3, pp. 212–220.

deepset (2024a). *Haystack Documentation*. Acedido a 13 jul. 2025. url: https://docs.haystack.deepset.ai/docs.

– (2024b). *Haystack Overview: Introduction*. Acedido a 13 jul. 2025. url: https://haystack.deepset.ai/overview/intro.

Doan, Nguyen Nam et al. (2024). «A Hybrid Retrieval Approach for Advancing Retrieval-Augmented Generation Systems». Em: *Proceedings of the 7th International Conference on Natural Language and Speech Processing (ICNLSP 2024)*, pp. 397–409.

Eibich, Matouš, Shivay Nagpal e Alexander Fred-Ojala (2024). *ARAGOG: Advanced RAG Output Grading*. arXiv: 2404.01037 [cs.CL]. url: https://arxiv.org/abs/2404.01037.

Gao, Yunfan et al. (2023). «Retrieval-augmented generation for large language models: A survey». Em: *arXiv preprint arXiv:2312.10997* 2.1.

Genesis, Jeanie e Frazier Keane (2025). «Integrating Knowledge Retrieval with Generation: A Comprehensive Survey of RAG Models in NLP». Em.

Google Angular Team (2025). *Angular - Documentation*. `https://angular.io/docs`. Acedido a 28 set. 2025.

Gotterbarn, Don, Keith Miller e Simon Rogerson (1997). «Software engineering code of ethics». Em: *Communications of the ACM* 40.11, pp. 110–118.

Grinberg, Miguel (2018). *Flask web development*. "O'Reilly Media, Inc."

Guo, Rentong et al. (2022). «Manu: a cloud native vector database management system». Em: *arXiv preprint arXiv:2206.13843*.

Gupta, Shailja, Rajesh Ranjan e Surya Narayan Singh (2024). «A comprehensive survey of retrieval-augmented generation (rag): Evolution, current landscape and future directions». Em: *arXiv preprint arXiv:2410.12837*.

Harjono, Karel Joshua (2025). «From Embeddings to Entities: A Comparative Analysis of RAG Architectures in Academic Domains». Em.

Haystack (2024). `https://github.com/deepset-ai/haystack`. Acedido a 13 jul. 2025.

Horvat, Dora (s.d.). «TRANSFORMATION OF BUSINESSSYSTEMSUSING LARGE LANGUAGE MODELS AND THE RAG METHOD». Em: ().

IBM (2023a). *AI vs. machine learning vs. deep learning vs. neural networks: What's the difference?* Acedido a 28 jul. 2025. url: `https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks`.

– (2023b). *NoSQL Databases*. Acedido a 28 jul. 2025. url: `https://www.ibm.com/think/topics/nosql-databases`.

– (2023c). *Relational Databases*. Acedido a 28 jul. 2025. url: `https://www.ibm.com/think/topics/relational-databases`.

– (2024). *LangChain: Como os agentes de linguagem estão mudando os negócios*. Acedido a 13 mai. 2025. url: `https://www.ibm.com/think/topics/langchain`.

Ibtasham, Md Saleh (2024). «Towards contextually aware large language models for software requirements engineering: A retrieval augmented generation framework». Em.

Isaza, Paulina Toro et al. (2024a). «Retrieval Augmented Generation-Based Incident Resolution Recommendation System for IT Support». Em: url: `https://arxiv.org/abs/2409.13707`.

– (2024b). «Retrieval Augmented Generation-Based Incident Resolution Recommendation System for IT Support». Em: *arXiv preprint arXiv:2409.13707*.

Kaczanowski, Tomek (2013). *Practical unit testing with JUnit and Mockito*. Tomasz Kaczanowski.

Kamalloo, Ehsan et al. (2023). «Evaluating embedding APIs for information retrieval». Em: *arXiv preprint arXiv:2305.06300*.

Kholmatov, Abrorjon (2024). «PYTHON FOR WEB DEVELOPMENT: FROM FRAMEWORKS TO REAL-WORLD APPLICATIONS». Em: *Engineering problems and innovations* 2.Spes. 2 DI.

Kuok, Ka Lok, Hao Hui Liu e Wai Weng Lo (2025). «CrimeKGQA: A Crime Investigation System Based on Knowledge Graph RAG». Em: *Intelligent Systems Conference*. Springer, pp. 184–195.

LangChain (2024a). `https://github.com/langchain-ai/langchain`. Acedido a 13 jul. 2025.

– (2024b). *LangChain Python Docs: Chains Module*. Acedido a 13 mai. 2025. url: `https://python.langchain.com/v0.1/docs/`.

Lee, Ho-Chit et al. (2024). «Development of an RAG-Based LLM Chatbot for Enhancing Technical Support Service». Em: *TENCON 2024-2024 IEEE Region 10 Conference (TENCON)*. IEEE, pp. 1080–1083.

Lewis, Patrick et al. (2020). «Retrieval-augmented generation for knowledge-intensive nlp tasks». Em: *Advances in neural information processing systems* 33, pp. 9459–9474.

Machinery, Computing (1950). «Computing machinery and intelligence-AM Turing». Em: *Mind* 59.236, p. 433.

Mackenzie, Joel et al. (2025). «Efficient In-Memory Inverted Indexes: Theory and Practice». Em: *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 4102–4105.

Murtaza, Syed Shariyar et al. (2025). «Implementing Retrieval Augmented Generation Technique on Unstructured and Structured Data Sources in a Call Center of a Large Financial Institution». Em: *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pp. 598–606.

Mythily, M, A Samson Arun Raj e Iwin Thanakumar Joseph (2022). «An analysis of the significance of spring boot in the market». Em: *2022 international conference on inventive computation technologies (ICICT)*. IEEE, pp. 1277–1281.

Peffers, Ken et al. (2007). «A design science research methodology for information systems research». Em: *Journal of management information systems* 24.3, pp. 45–77.

PGVector (2025). *pgvector: Open-source vector similarity search for Postgres*. `https://github.com/pgvector/pgvector`. Acedido a 9 set. 2025.

Qdrant Team (2024). *Qdrant Documentation: Framework Integrations*. Acedido a 29 jun. 2025. url: `https://qdrant.tech/documentation/frameworks/`.

Rau, David et al. (2024). «Context embeddings for efficient answer generation in rag». Em: *arXiv preprint arXiv:2407.09252*.

React Team (2025). *React Reference*. Acedido a 30 ago. 2025. url: `https://react.dev/reference/react`.

Reynolds, Laria e Kyle McDonell (2021). «Prompt programming for large language models: Beyond the few-shot paradigm». Em: *Extended abstracts of the 2021 CHI conference on human factors in computing systems*, pp. 1–7.

Rho, Donghwan et al. (2024). «Encryption-friendly LLM architecture». Em: *arXiv preprint arXiv:2410.02486*.

Roumeliotis, Konstantinos I e Nikolaos D Tselikas (2023). «Chatgpt and open-ai models: A preliminary review». Em: *Future Internet* 15.6, p. 192.

Salıcı, Mustafa e Üyesi Ercan Ölçer (2024). «Impact of Transformer-Based Models in NLP: An In-Depth Study on BERT and GPT». Em: *2024 8th International Artificial Intelligence and Data Processing Symposium (IDAP)*. IEEE, pp. 1–6.

Schmalzried, Hermann (1981). *Solid state reactions*. Weinheim Germany.

Seshadri, Shyam (2018). *Angular: Up and running: Learning angular, step by step*. "O'Reilly Media, Inc."

Siami-Irdemoosa, Elnaz, Saeid R Dindarloo e Mostafa Sharifzadeh (2015). «Work breakdown structure (WBS) development for underground construction». Em: *Automation in construction* 58, pp. 85–94.

Silva, Thayná Camargo da (2025). «Extracting Knowledge Graphs from User Stories using LangChain». Em: *arXiv preprint arXiv:2506.11020*.

Soman, Sumit e Sujoy Roychowdhury (2024). «Observations on Building RAG Systems for Technical Documents». Em: Published as a Tiny Paper at ICLR 2024. url: `https://arxiv.org/pdf/2404.00657`.

Son, Minjun e Sungjin Lee (2025). «Advancing Multimodal Large Language Models: Optimizing Prompt Engineering Strategies for Enhanced Performance». Em: *Applied Sciences* 15.7, p. 3992.

Špeletić, Matija et al. (2024). «EXPLORING RAG IN MEDICAL QUESTION ANSWERING: INTEGRATING LLMS AND VECTOR DATABASES». Em.

Spring Team (2024). *Spring AI Reference Documentation*. Acedido a 13 jul. 2025. url: `https://docs.spring.io/spring-ai/reference/`.

Suman, Manoj Wadhwa e MDU Rohtak (2014). «A comparative study of software quality models». Em: *International Journal of Computer Science and Information Technologies* 5.4, pp. 5634–5638.

Wang, Xiaohua et al. (2024). «Searching for best practices in retrieval-augmented generation». Em: *arXiv preprint arXiv:2407.01219*.

Wei, Jason et al. (2022). «Chain-of-Thought Prompting Elicits Reasoning in Large Language Models». Em: *Advances in Neural Information Processing Systems*. Ed. por S. Koyejo et al. Vol. 35. Curran Associates, Inc., pp. 24824–24837. url: `https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf`.

Wu, Xiao-Kun et al. (2025). «Llm fine-tuning: Concepts, opportunities, and challenges». Em: *Big Data and Cognitive Computing* 9.4, p. 87.

Yang, Jeff et al. (2025). «SuperRAG: Beyond RAG with Layout-Aware Graph Modeling». Em: *arXiv preprint arXiv:2503.04790*.

Yepes, Antonio Jimeno et al. (2024). «Financial report chunking for effective retrieval augmented generation». Em: *arXiv preprint arXiv:2402.05131*.

Anexo A

Prompts

```
1                 """
2                 You are a technical support assistant. Your task is to help the user
     by answering their question using the information provided in the context below
    .
3
4                 Instructions:
5                 - If the answer is unclear or incomplete in the context, politely
    state that and suggest what information the user could provide next.
6                 - Give step-by-step troubleshooting guidance if possible.
7                 - Keep the tone professional, clear, and supportive.
8                 - The context will be provided in a markdown format.
9
10                Context:
11                =========
12                %s
13                =========
14
15                User query: %s
16                """
17
```

Código A.1: *Prompt* RAG elaborado - *chain-of-thought prompting*

```
"""
You are an impartial evaluator. Your task is to assess how well a candidate answer
    matches the reference answer.s

                Question:
                "%s"

                Reference Answer:
                "%s"

                Candidate Answer:
                "%s"

                Evaluation Instructions:
                1. Compare the candidate answer to the reference answer in terms of
    semantic similarity and relevance.
                2. Do not reward unnecessary extra information if it is irrelevant
    to the reference.
                3. Be strict: if the candidate does not convey the same meaning as
    the reference, it should get a low score.
                4. Score the candidate answer on a scale from 1 to 5:
                   - 1 = Completely unrelated
                   - 2 = Slightly related, but mostly incorrect or irrelevant
                   - 3 = Somewhat related, partially correct, but missing important
    elements
                   - 4 = Mostly correct, with minor differences or extra irrelevant
    information
                   - 5 = Perfectly correct, fully matches the reference in meaning

                Output your answer in the following format only: <number from 1 to
    5>
"""

```

Código A.2: *Prompt* de avaliação das respostas

```
"""
                You are a query rewriting assistant for a RAG system. When a user asks a question about failed jobs, rewrite the query to include:
                A specific title matching the report format: "Control-M History Report - Week from DD/MM to DD/MM"

                Relevant time range inferred from the query (e.g., "second week of July" results in "Week from 06/07 to 12/07")

                Clarify whether the user is asking about daily or monthly failed jobs

                Preserve the user's intent and any filters (e.g., job name, folder, time of day)

                The weeks start on Sunday and end on Saturday, example: from 13/07 to 19/07

                ========================

                Examples:

                User Query: "What jobs failed in the second week of July?" ; Rewritten Query: "Retrieve failed jobs from Control-M History Report - Week from 06/07 to 12/07"
                User Query: "Which jobs failed on July 15th?" ; Rewritten Query: "Retrieve failed jobs from Control-M History Report - Week from 13/07 to 19/07 for 2025-07-15 "
                User Query: "Job PL96DWTCGRAMBXEEPE failed on july?" ; Rewritten Query: "Retrieve failed job PL96DWTCGRAMBXEEPE information from "Control-M History Report - Week from 29/06 to 05/07" ; "Control-M History Report - Week from 06/07 to 12/07" ; "Control-M History Report - Week from 13/07 to 19/07" ; "Control-M History Report - Week from 20/07 to 26/07" ; "Control-M History Report - Week from 27/07 to 02/08" "

                ========================

                Real User Query: %s

                """

```

Código A.3: *Prompt* reescrita de *query* para Control-M

Anexo B

Algoritmos

```
            ChatClient chatClient = getChatClient();
            Map<String, String> questionsAnswers = readFile();


            List<Integer> results = new ArrayList<>();

            for (Map.Entry<String, String> entry : questionsAnswers.entrySet()) {
                String question = entry.getKey();
                String answerGold = entry.getValue();

                String answerRag = callRagSystem(question);

                String response = chatClient.prompt()
                        .user(evaluationPrompt(question, answerGold, answerRag))
                        .call()
                        .content();

                results.add(Integer.parseInt(response));
            }


            pt.println("=====================================================");
            pt.println("Test: " + test.getKey() + " " + test.getValue());
            pt.println("Results: " + String.join(",", results.stream().map(Object::
    toString).toList()));
            pt.println("Mean: " + mean(results));
            pt.println("Median: " + median(results));
            pt.println("% >= 4: " + percentageGreaterOrEqualThanFour(results) + "%")
    ;
            pt.println("Standard Deviation: " + standardDeviation(results));

```

Código B.1: Algoritmo de experimentação automatizada

Anexo C

Sessão de Apresentação Interna

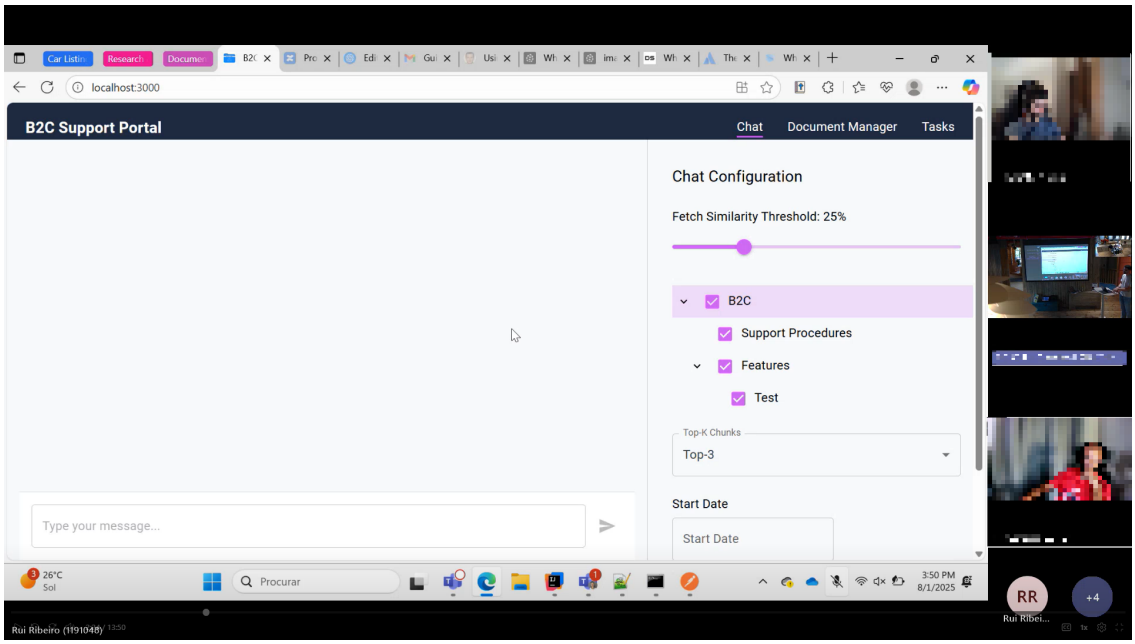


Figura C.1: Evidência de sessão interna de apresentação da solução

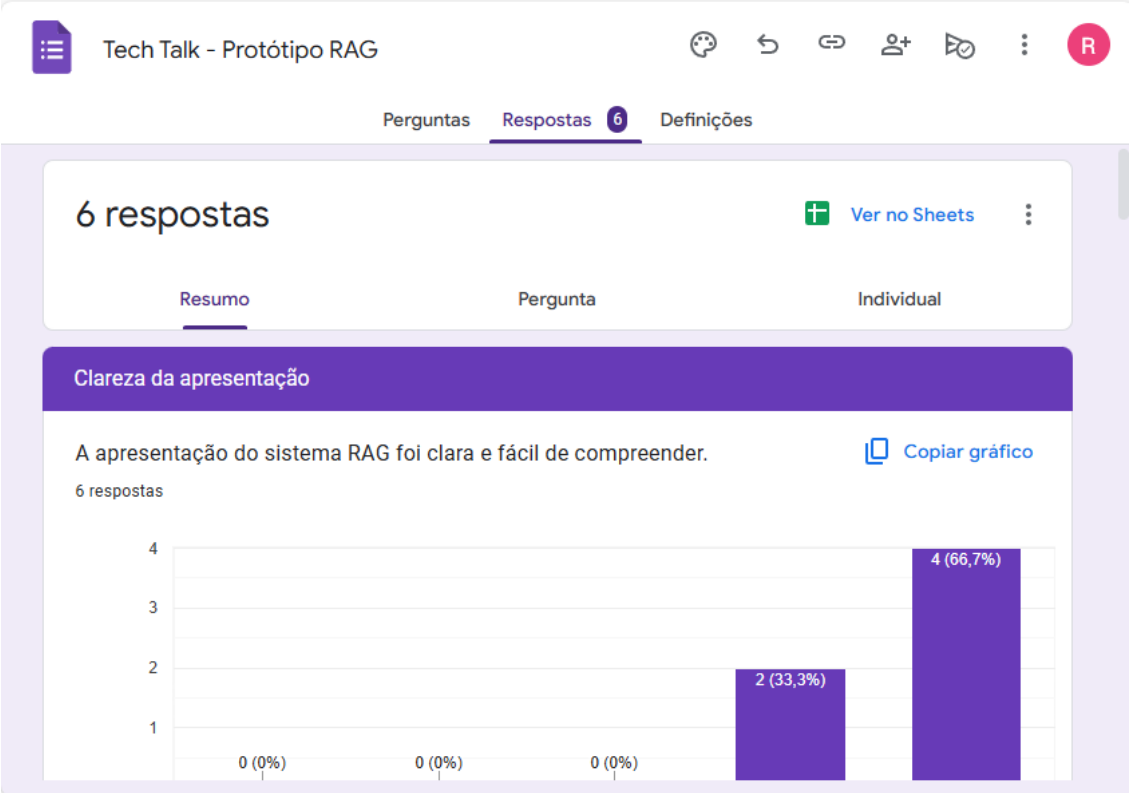


Figura C.2: Evidência de questionário retrospetivo realizado após a sessão interna

Anexo D

Modelação e Dados

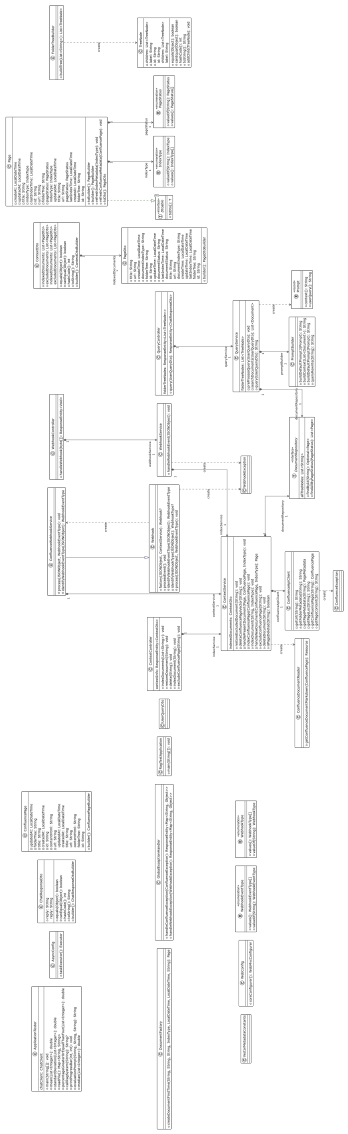


Figura D.1: Diagrama de classes Unified Modeling Language (UML) - Sistema RAG

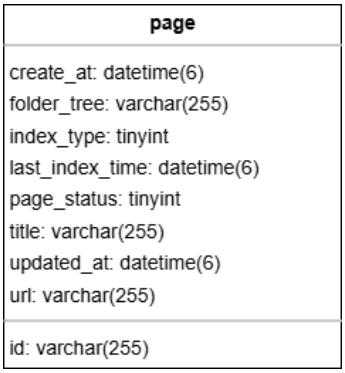


Figura D.2: Diagrama entidade-relação da base de dados relacional composto exclusivamente pela *page*

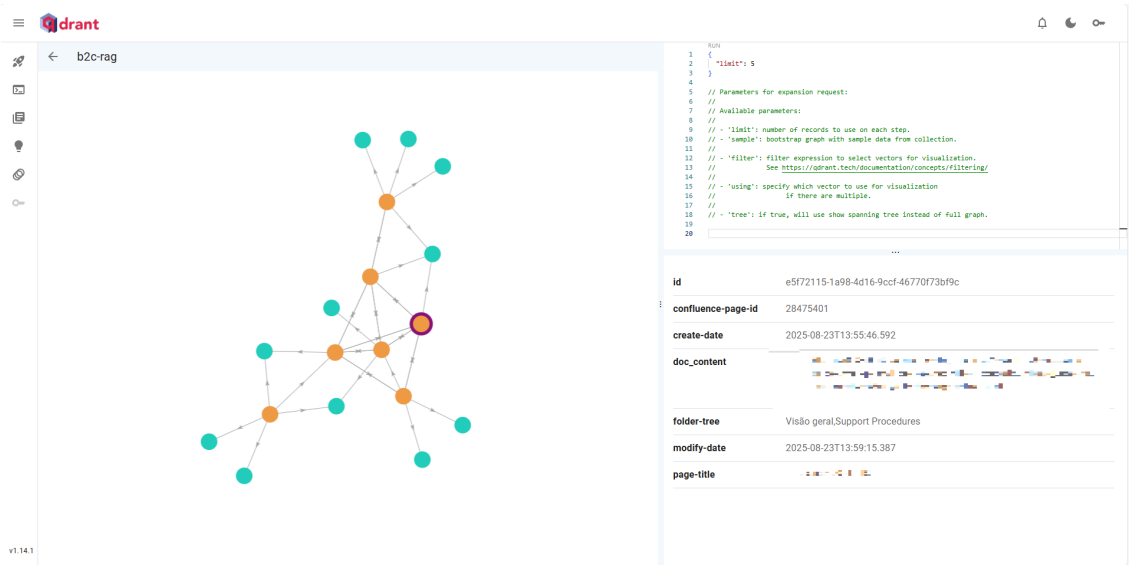


Figura D.3: Qdrant - Associação de metadados a cada vetor armazenado