Utilizando IA para Matching Aluno-Professor em Projetos de TCC

Trabalho de Conclusão de Curso

Emanuel Sena Gomes Santos

Sandro Santos Andrade
Orientador

Instituto Federal da Bahia - IFBA
Curso de Análise e Desenvolvimento de Sistemas
Campus Salvador

Salvador, Bahia, Brasil
15 de julho de 2026

Sumário

1 Visão geral

No cotidiano do ambiente acadêmico, a gestão e o desenvolvimento do Trabalho de Conclusão de Curso, o TCC, representam marcos críticos na formação dos discentes. Pensando em otimizar as interações institucionais, o aplicativo Emile foi concebido como uma interface móvel e unificada, permitindo que professores e estudantes gerenciem diversas atividades educacionais de forma fluida. Recentemente, a introdução da funcionalidade de registro de propostas de TCC no Emile abriu uma via de comunicação bidirecional: alunos podem cadastrar ideias de projetos em busca de orientação, e professores podem ofertar temas de pesquisa para atrair discentes interessados. Essa digitalização do processo estabeleceu o terreno necessário para a aplicação de soluções computacionais avançadas que possam guiar e otimizar ativamente a união entre os interesses dos alunos e a expertise do corpo docente.

1.1 Declaração do problema

Atualmente, a dinâmica do aplicativo Emile permite que alunos e professores manifestem interesse mútuo no desenvolvimento de propostas de TCC diretamente pela interface móvel. Contudo, o processo de descoberta e pareamento, também conhecido como *matching*, desses perfis ainda ocorre de maneira passiva e predominantemente manual. Ele baseia-se na navegação exploratória das partes ou no conhecimento prévio, muitas vezes restrito, que os alunos possuem sobre as linhas de pesquisa dos professores.

Esse cenário é problemático, pois consome tempo útil de gestão, está sujeito a vieses de visibilidade e frequentemente resulta em alocações subótimas, onde um docente com profundo conhecimento em um nicho específico, detalhado em seu currículo Lattes, não é acionado por mero desconhecimento do discente. Diante disso, este trabalho tem como objetivo implementar uma solução automatizada de Inteligência Artificial para realizar o *matching* ativo entre as propostas cadastradas e o currículo dos professores, democratizando o acesso à informação e elevando o rigor técnico das pesquisas da instituição.

1.2 Objetivos

1.2.1 Objetivo geral

O objetivo principal deste trabalho é desenvolver e integrar um módulo de Inteligência Artificial baseado na arquitetura *Advanced Retrieval-Augmented Generation* [1] ao aplicativo Emile, visando automatizar e otimizar o processo de pareamento ativo entre as propostas de Trabalho de Conclusão de Curso dos discentes e as linhas de pesquisa e expertise presentes nos currículos Lattes do corpo docente.

1.2.2 Objetivos específicos

Para o alcance do objetivo geral, foram delineados os seguintes objetivos específicos:

- Extrair, segmentar e estruturar os dados dos currículos do corpo docente na Plataforma Lattes por meio de rotinas automatizadas em lote.

- Implementar a vetorização semântica das seções curriculares extraídas, garantindo seu armazenamento e indexação em um banco de dados relacional com suporte vetorial.
- Construir o pipeline de recuperação de dados e inferência, aplicando técnicas de expansão de consulta, busca híbrida e reordenação estrutural para maximizar a precisão do cruzamento entre propostas e professores.
- Desenvolver a etapa de geração de texto utilizando Grandes Modelos de Linguagem de código aberto, para criar justificativas de recomendação explicáveis, imparciais e estritamente ancoradas nos documentos recuperados.
- Integrar o módulo de inteligência ao aplicativo Emile de maneira assíncrona, viabilizando o fluxo ponta a ponta desde o cadastro e edição da proposta até a notificação do usuário móvel.

1.3 Proposta de solução de software

A solução conceitual proposta consiste na integração de um módulo inteligente ao backend em Python do Emile, atuando de forma assíncrona na recomendação de orientadores. Para que essa automação ocorra com precisão, o sistema adota a arquitetura *Advanced Retrieval-Augmented Generation*, ou Advanced RAG [1], sustentada por técnicas de vetorização semântica da base de dados Lattes.

A dependência direta dessa base de dados estabelece, no entanto, uma delimitação fundamental no escopo deste trabalho. Embora a dinâmica do aplicativo Emile possibilite que tanto alunos quanto professores submetam propostas, o motor de *matching* desenvolvido atua exclusivamente no fluxo originado pelo **aluno**. O fluxo reverso, onde o sistema recomendaria alunos aptos para uma proposta submetida por um docente, ainda não é viável computacionalmente. Isso ocorre devido à inexistência atual de um repositório estruturado que mapeie de forma confiável as expertises, interesses e o histórico técnico dos discentes. Em contrapartida, o corpo docente possui suas competências amplamente detalhadas e publicamente acessíveis na Plataforma Lattes, fornecendo a matéria-prima ideal para a viabilização da arquitetura adotada.

É exatamente ao explorar essa rica base curricular que a solução se destaca de abordagens convencionais. Como resultado final desse processamento, o aplicativo móvel notifica o usuário apresentando não apenas uma lista de docentes compatíveis, mas uma justificativa textual gerada por Inteligência Artificial que explica de forma clara o motivo da adequação daquele perfil [2]. Para a plena compreensão da solução, é necessário detalhar os conceitos operacionais que fundamentam as etapas deste pipeline.

1.3.1 Processamento Offline e Online

Para viabilizar a recomendação, o sistema é estruturado em dois fluxos operacionais distintos: o processamento *offline* e o processamento *online*. A fase *offline* é responsável pela preparação da base de conhecimento, transformando currículos brutos em dados estrutu-

rados e vetoriais. Já a fase *online* compreende a interação direta com o usuário, desde o recebimento da proposta até a entrega da justificativa final.

A Figura 1 ilustra a divisão desses componentes e como o pipeline de dados flui através da infraestrutura de IA, conforme proposto no paradigma de sistemas baseados em recuperação.

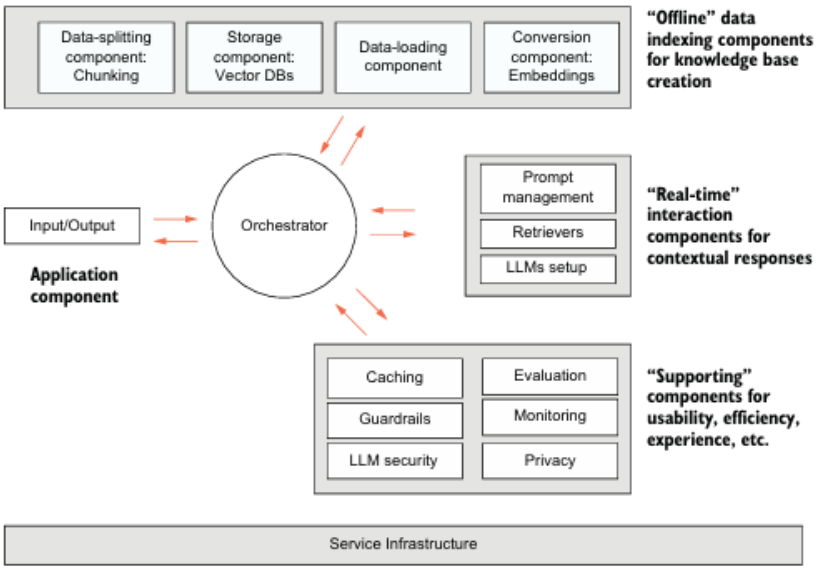


Figura 1: Representação das camadas Offline e Online da arquitetura do sistema.

Fonte: Extraído de [1].

1.3.2 Arquitetura RAG e Recomendação Explicável

O *Retrieval-Augmented Generation*, ou RAG [1], é um paradigma arquitetural que aprimora as capacidades de Grandes Modelos de Linguagem, os LLMs, ancorando a geração de texto em bases de dados externas e específicas. Em vez de depender exclusivamente das informações retidas durante o treinamento do modelo, chamada de memória paramétrica [3], o RAG busca documentos relevantes em tempo real para contextualizar a resposta, mitigando o risco de invenções algorítmicas, fenômeno conhecido como alucinações da Inteligência Artificial [4].

Em sua implementação mais rudimentar, conhecida como *Naïve RAG* ou RAG Ingênuo, o processo resume-se a um fluxo linear simples: os documentos são indexados, o sistema recupera os fragmentos mais similares à busca do usuário e os envia diretamente ao LLM para a geração da resposta. Contudo, em bases de dados complexas e repletas de jargões técnicos acadêmicos, o RAG Ingênuo frequentemente falha, pois resgata informações irrelevantes que geram ruído ou perde o contexto semântico real da intenção do aluno.

Para superar essas limitações, a solução deste projeto adota a arquitetura de Advanced RAG ou RAG Avançado [1]. Este modelo introduz otimizações cruciais e divide o fluxo de inteligência em etapas fundamentais, que abrangem desde a ingestão offline dos dados até a geração online da justificativa:

Extração e Processamento em Lote (Offline)
Os currículos Lattes são tratados como documentos internos estáticos durante o ciclo de orientação. A solução utiliza rotinas de *Web Scraping* para extrair esses dados publicamente disponíveis em formato de texto. Este processo ocorre em lote, técnica conhecida como *batch ingestion*, segmentando a informação estruturalmente em seções naturais como resumo, publicações e orientações. Essa fragmentação especializada, denominada *Adaptive Chunking* ou Segmentação Adaptativa [5], é vital para não corromper a integridade semântica do perfil do professor durante a vetorização.

Vetorização Semântica (Offline)
Em vez de depender de buscas exatas por palavras-chave, o texto extraído é convertido em vetores multidimensionais, conhecidos como *embeddings* [6], utilizando modelos pré-treinados de linguagem. Esses vetores capturam o significado e o contexto da informação, permitindo que a Inteligência Artificial compreenda, por exemplo, que as expressões ”Aprendizado de Máquina”e ”Machine Learning”representam a mesma área de expertise. O armazenamento e a busca dessas representações ocorrem em um banco de dados relacional híbrido munido de extensões vetoriais.

Para ilustrar de forma técnica as interações descritas, a Figura 2 apresenta o diagrama de sequência referente a toda a etapa *offline*. A modelagem destaca os principais métodos e componentes envolvidos, demonstrando o fluxo desde a requisição de extração dos currículos até a consolidação dos *embeddings* no banco de dados vetorial.

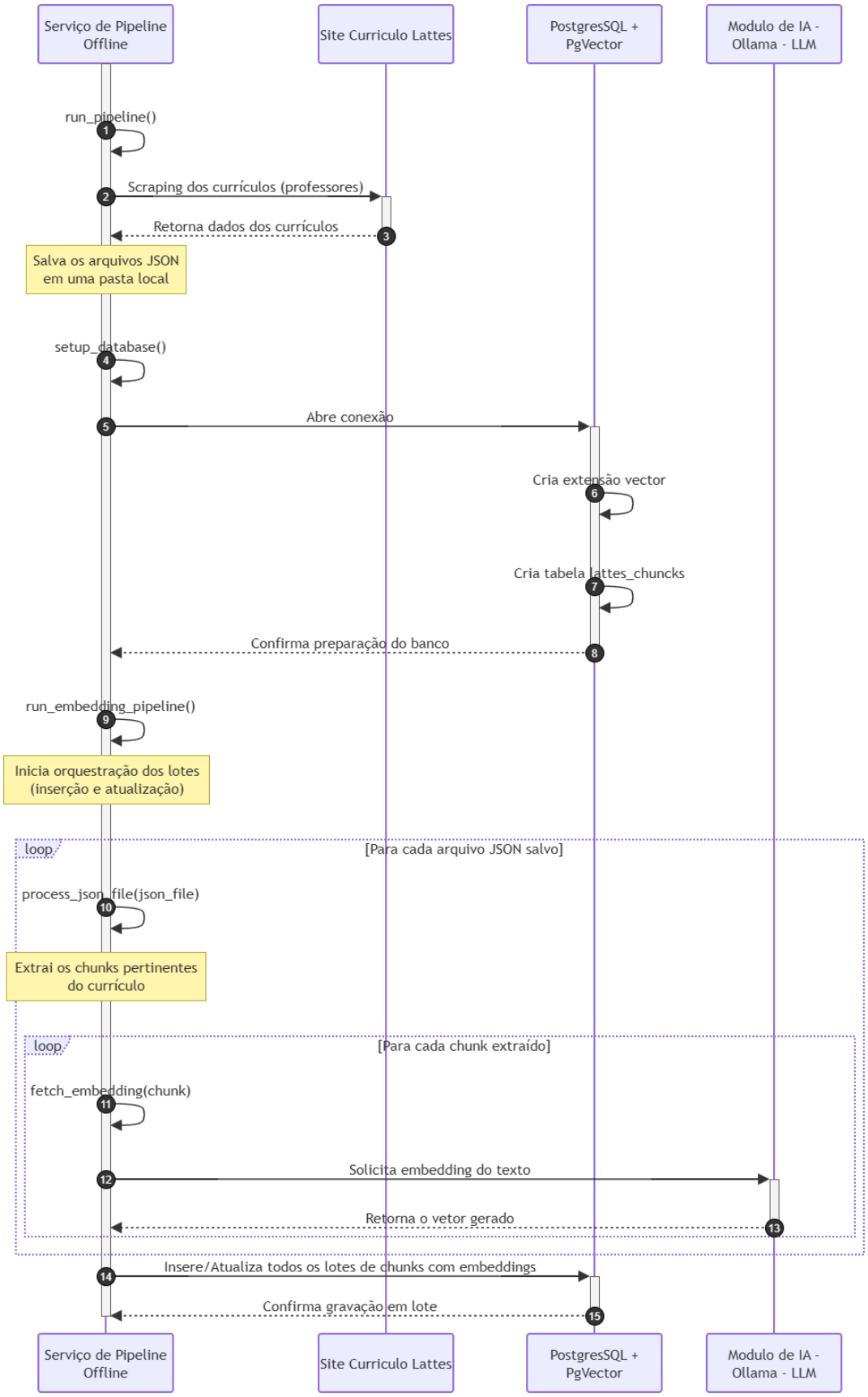


Figura 2: Diagrama de sequência ilustrando as interações e métodos do processamento *offline*.

Fonte: Autoria própria.

Pipeline de Consulta e Inferência (Online)
Após a estruturação da base de dados, o ciclo de inferência online garante a precisão da correlação e a formulação de uma justificativa rigorosamente embasada através das seguintes fases:

- **Pré-recuperação ou *Pre-retrieval*:** Focada na otimização da intenção de busca. Comumente, o aluno descreve sua proposta com termos generalistas, como Inteligência Artificial, enquanto o currículo do professor utiliza jargões estritos, como Redes Neurais e Aprendizado Profundo. O sistema utiliza um LLM para realizar a reescrita e a expansão da consulta [7], enriquecendo a proposta original com sinônimos e vocabulário técnico para ampliar as chances de pareamento.

- **Recuperação ou *Retrieval*:** Etapa de extração de dados do banco de dados vetorial. O sistema adota uma estratégia de busca híbrida. A busca densa localiza a similaridade semântica de conceitos [8], enquanto a busca esparsa assegura a correspondência exata de tecnologias e palavras-chave específicas. Utiliza-se um índice plano para calcular a distância matemática exata de todos os currículos, retornando os melhores perfis simultaneamente.

- **Pós-recuperação ou *Post-retrieval*:** Focada no refinamento crítico dos resultados encontrados. Os documentos recuperados passam por uma camada de reordenação estrutural, também referida como *reranking* [9]. Nesta etapa, aplica-se uma filtragem por departamento ou disponibilidade e reordena-se a lista com precisão algorítmica avançada, assegurando que o perfil mais aderente figure no topo das recomendações. É também nesta fase que o sistema aplica um limiar de corte de 0.3 para o *score* de similaridade dos *chunks* recuperados. Apenas os professores cujos fragmentos atinjam ou superem esse valor avançam para a etapa de inferência. Caso nenhum fragmento alcance essa métrica, o fluxo é interrompido precocemente e o sistema retorna diretamente uma mensagem padrão justificando que não foram encontrados docentes com perfil compatível para a proposta, poupando recursos computacionais e evitando alocações forçadas.

- **Geração ou *Generation*:** Acionada exclusivamente caso existam perfis aprovados pelo limiar da etapa anterior, esta fase final traduz os dados técnicos em uma recomendação explicável para o usuário [2]. Os dados do currículo Lattes e a proposta do aluno são injetados em moldes estruturados, garantindo que o modelo não confunda competências de diferentes docentes. O modelo é instruído via cadeia de pensamentos a analisar passo a passo os pontos de interseção entre a proposta e o professor. A justificativa é elaborada em uma única passagem de processamento, e submetida a uma moderação de saída, assegurando um tom formal, imparcial e puramente ancorado na base de conhecimento ou memória não-paramétrica.

Semelhante à etapa de preparação dos dados, o fluxo de execução das requisições em tempo real também foi mapeado. A Figura 3 exibe o diagrama de sequência que sintetiza o processamento *online*. O diagrama elucida os métodos e componentes acionados desde o

momento em que a proposta do aluno entra no sistema, passando pelo motor de inferência, até a devolução estruturada da recomendação e de sua justificativa.

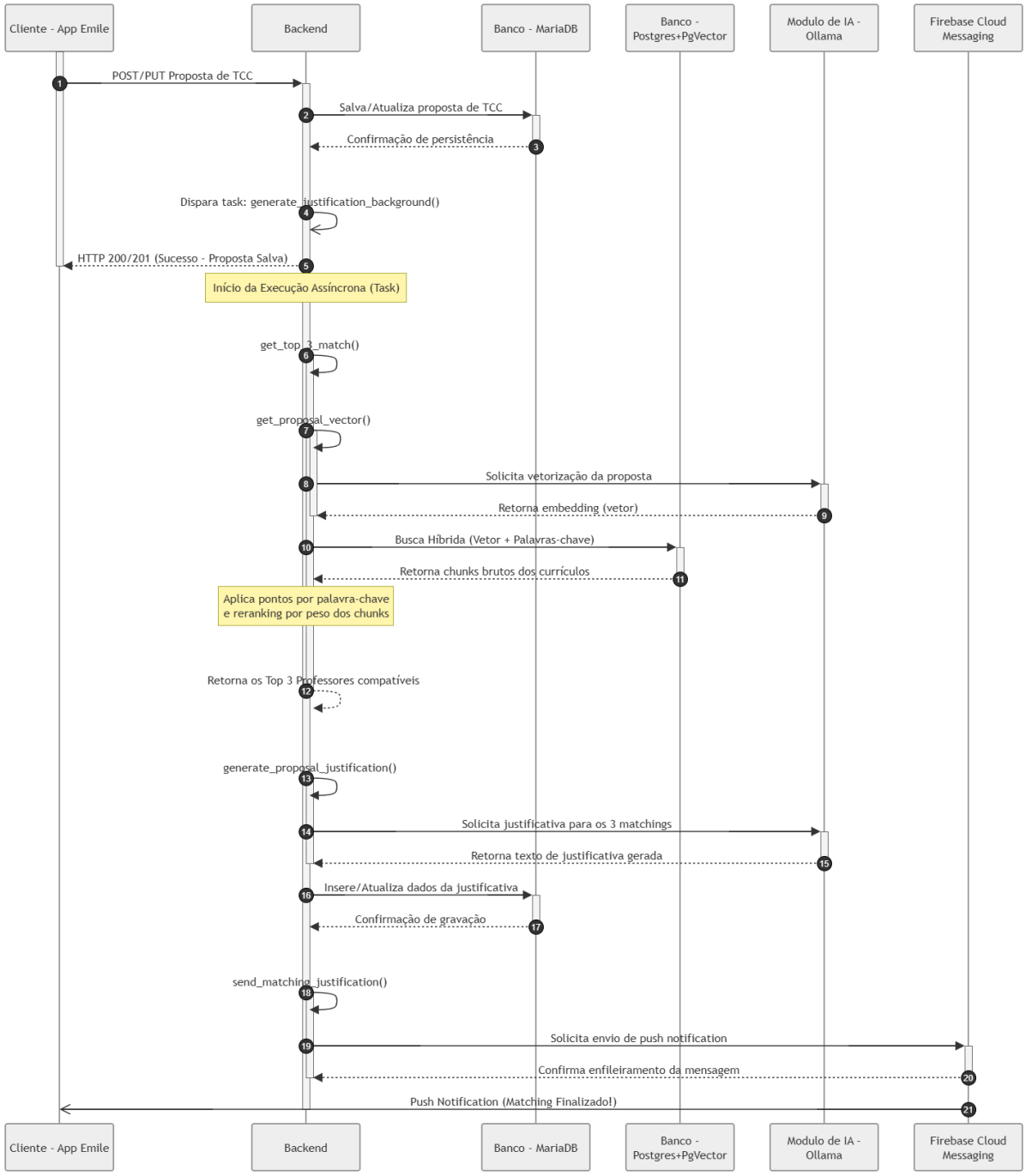


Figura 3: Diagrama de sequência ilustrando as interações e métodos do processamento *on-line*.

Fonte: Autoria própria.

1.4 Tecnologias adotadas

O desenvolvimento deste projeto fundamenta-se na integração de ferramentas de alto desempenho, bibliotecas de código aberto e arquiteturas escaláveis para o processamento de linguagem natural e gestão de dados:

- **Qt/QML:** Trata-se de um framework de desenvolvimento de interfaces gráficas baseado em uma linguagem declarativa (QML) e lógica em C++. Suas principais características incluem o alto desempenho de renderização, a separação clara entre a interface e a lógica de negócios, e a capacidade de compilação nativa para múltiplas plataformas, como Android, iOS e Desktop.

- **Docker:** O Docker é uma plataforma de virtualização leve baseada em contêineres, utilizada para empacotar aplicações e todas as suas dependências em ambientes isolados e portáveis.

- **Python:** É uma linguagem de programação de alto nível, multiparadigma e de sintaxe legível. Sua predominância no campo da Inteligência Artificial deve-se ao seu ecossistema robusto de bibliotecas para ciência de dados, integração de sistemas e orquestração de modelos de aprendizado de máquina, permitindo o desenvolvimento ágil de backends complexos.

- **MariaDB:** Trata-se de um sistema de gerenciamento de banco de dados relacional de código aberto, derivado do ecossistema MySQL e amplamente reconhecido por sua confiabilidade e alto desempenho.

- **PostgreSQL com pgvector:** O PostgreSQL é um sistema de gerenciamento de banco de dados relacional robusto e extensível. Com a extensão *pgvector*, o banco adquire a capacidade híbrida de armazenar e consultar vetores de alta dimensão ao lado de dados relacionais tradicionais, permitindo buscas por similaridade semântica através de índices especializados.

- **Ollama e Qwen3:** O Ollama é uma ferramenta de orquestração que permite a execução local de Modelos de Linguagem de Grande Escala de código aberto. Suas características principais são a privacidade dos dados e a independência de APIs proprietárias. Para a etapa de geração de texto e raciocínio lógico, adotou-se o modelo **Qwen3 de 30 bilhões de parâmetros (**`qwen3:30b`**)**, escolhido por sua alta capacidade de compreensão de contexto e precisão ao formular as justificativas do *matching*, permitindo que o processamento ocorra inteiramente em infraestrutura controlada.

- **Ollama Embeddings e Qwen3-Embedding:** Além da geração de texto, o ecossistema Ollama provê suporte local para modelos de vetorização. Neste projeto, utilizou-se o modelo **Qwen3-Embedding de 4 bilhões de parâmetros (**`qwen3-embedding:4b`**)**. Ele atua transformando os textos estruturados ou não estruturados da plataforma Lattes em representações numéricas densas de **2560 dimensões**, sendo altamente otimizado para capturar nuances semânticas complexas do domínio acadêmico com baixa latência para o banco de dados vetorial.

- **Postman:** É uma plataforma dedicada ao desenvolvimento, documentação e teste de APIs.
- **Selenium:** O Selenium é o framework padrão para a automação de navegadores web, permitindo a extração de dados de páginas dinâmicas que dependem da execução de JavaScript.
- **Firebase Cloud Messaging:** É uma solução de mensageria na nuvem que permite o envio de mensagens e notificações de forma confiável e multiplataforma. Destaca-se pela baixa latência na entrega, escalabilidade automática e pela capacidade de despertar aplicações em segundo plano para alertar o usuário sobre processos concluídos no servidor.

1.5 Trabalhos Relacionados

A literatura recente apresenta diversas abordagens para o problema de recomendação e alocação de orientadores no meio acadêmico, buscando preencher as lacunas das interações manuais. historicamente, sistemas pioneiros como o *Officehours* [10] focaram em otimizar o pareamento entre estudantes e supervisores por meio de algoritmos de aprendizado por reforço, refinando as sugestões com base na interação contínua e nos feedbacks do usuário.

Na mesma vertente metodológica, Zhang. [11] estabeleceu um *framework* analítico especificamente focado na seleção de orientadores acadêmicos, estruturando as recomendações com base em métricas tridimensionais que avaliam a relevância da pesquisa, a conectividade e a qualidade global do perfil. Explorando uma maior descentralização e o viés do controle pelo próprio discente, o sistema Grapevine [12] permitiu que os estudantes navegassem de forma exploratória e interativa pelas redes de pesquisa dos docentes, garantindo maior transparência e controle na tomada de decisão.

Por fim, adotando uma abordagem estritamente voltada à similaridade de conteúdo semântico, Wijanto, Rachmadiany e Karnalim [13] desenvolveram um motor baseado em técnicas clássicas de recuperação de informação para recomendar supervisores, utilizando como base comparativa os documentos representativos produzidos pelos alunos, como propostas ou resumos de teses. Em contraste direto com essas abordagens que dependem frequentemente do histórico de interações ou de mecanismos lexicais rígidos, o aplicativo Emile inova ao combinar a vetorização profunda de perfis curriculares na base Lattes com o fluxo da Geração Aumentada por Recuperação, viabilizando recomendações semânticas acompanhadas de justificativas textuais explícitas e tecnicamente verificáveis [2].

1.6 Estratégias de Matching Analisadas

Além dos trabalhos supracitados, a automação de recomendações no ambiente educacional pode ser construída sobre diferentes paradigmas tecnológicos. Nesta seção, detalhamos as mecânicas conceituais predominantes, contrapondo suas limitações à solução proposta neste trabalho.

1.6.1 Sistemas baseados em Frequência de Termos

Muitos trabalhos iniciais de recomendação acadêmica baseiam-se na contagem exata de palavras, utilizando modelos de Recuperação de Informação Clássica, como o TF-IDF [14]. O sistema extrai palavras da proposta do aluno e procura professores que repetem essas mesmas palavras em seus resumos. Embora simples e computacionalmente baratos, falham catastroficamente ao lidar com sinônimos e contextos complexos, exigindo que o aluno conheça o jargão técnico exato da área.

Diferencial do Emile: O sistema abandona a dependência estrita de palavras-chave ao utilizar *Vetores* pré-treinados, que realizam a busca no espaço semântico. Além disso, o uso da técnica de expansão de consulta [7] adapta organicamente o linguajar do aluno para os termos técnicos da plataforma Lattes.

1.6.2 Filtros Colaborativos e Análise de histórico

Mecanismos clássicos de recomendação, baseados em Filtragem Colaborativa [15], analisam o histórico de orientações passadas do corpo docente. Eles cruzam o perfil do aluno com estudantes anteriores daquele mesmo professor para inferir matematicamente a adequação e sugerir correspondências.

Diferencial do Emile: Sistemas baseados puramente em histórico sofrem severamente do problema de início frio [15], penalizando novos professores que acabaram de ingressar na instituição e ainda não possuem orientações registradas. Ao focar os dados exclusivamente no conteúdo técnico do currículo Lattes por meio da extração textual, o Emile garante que a avaliação seja justa, técnica e imune à falta de métricas históricas de popularidade no campus.

1.6.3 Assistentes Virtuais Baseados em IA Generativa

Com o advento dos Grandes Modelos de Linguagem, surgiram plataformas baseadas em conversação que tentam auxiliar alunos na escolha da linha de pesquisa, utilizando unicamente a memória paramétrica [3] derivada do treinamento interno do próprio modelo para cruzar áreas de conhecimento.

Diferencial do Emile: Modelos generativos tradicionais tendem a alucinar ou inventar dados quando questionados sobre o currículo específico de indivíduos [4]. A solução proposta inibe essa falha ao aplicar o RAG Avançado. A memória do sistema é estritamente não-paramétrica, ou seja, limitada aos documentos injetados via *templates* seguros. A IA funciona apenas como um motor de raciocínio sobre verdades absolutas mapeadas no banco de dados, e não como fonte primária das informações.

A Tabela 1 sintetiza as diferenças tecnológicas entre os paradigmas de recomendação discutidos e a solução aplicada neste TCC, fundamentando as avaliações de acordo com os limites estruturais de cada tecnologia.

Tabela 1: Comparação justificada entre as estratégias de *matching*

Critério	Algoritmo TF-IDF [14]	IA Generativa Simples	RAG Avançado + Lattes [5]
Compreensão Semântica	**Baixa.** Limita-se à correspondência lexical exata, falhando sistemicamente ao lidar com sinônimos e ambiguidades.	**Alta.** O modelo processa ambiguidades e compreende nativamente as intenções complexas do usuário.	**Alta.** A vetorização densa [6] projeta textos em um espaço semântico robusto, capturando nuances de jargões.
Confiabilidade dos Dados	**Alta.** Resultados mapeiam diretamente a frequência real das palavras nos documentos indexados, sem inserções fictícias.	**Baixa.** Possui altíssima propensão a alucinações [4], gerando falsas áreas de atuação para docentes reais.	**Altíssima.** Geração de texto estritamente restrita aos currículos extraídos.
Transparência e Explicabilidade	**Baixa.** O algoritmo apresenta apenas um cálculo numérico de relevância, sem oferecer uma interpretação ou feedback ao usuário.	**Média.** O modelo redige explicações fluidas para as indicações, porém, frequentemente justificadas em premissas falsas.	**Alta.** Formula justificativas [2] que cruzam, linha a linha, as competências explícitas do Lattes com a demanda da proposta.
Manutenção da Base	**Simples.** Atualização estática e direta dos textos indexados, sem necessidade de recalibração severa.	**Complexa.** Exige retreinamento periódico e oneroso da memória paramétrica do modelo [3] para absorver novos docentes.	**Automatizada.** Atualização em lote com re-vetorização assíncrona dos novos fragmentos de texto do currículo.

2 Requisitos

Para garantir a priorização assertiva das entregas do software, os requisitos levantados foram categorizados com base no método MoSCoW (*Must have, Should have, Could have, Won't have*), detalhando as obrigatoriedades técnicas e os desejos arquiteturais do projeto.

2.1 Requisitos funcionais

Os requisitos funcionais descritos abaixo delimitam as capacidades e os comportamentos esperados do sistema, tanto na interação com os usuários quanto em suas operações de bastidores.

- **[Must] RF1** - O backend do sistema deve capturar e atualizar o texto dos currículos do corpo docente na Plataforma Lattes por meio de rotinas automatizadas de *Web Scraping* em lote.
- **[Must] RF2** - O sistema deve segmentar e converter as seções do currículo Lattes em representações vetoriais densas, armazenando-as de forma indexada no banco de dados PostgreSQL.
- **[Should] RF3** - O backend deve processar a recuperação dos perfis de forma assíncrona, não bloqueando a interface do aplicativo móvel enquanto os cálculos vetoriais e lógicos ocorrem.
- **[Should] RF4** - O sistema deve, antes da etapa de busca, reescrever a proposta do usuário utilizando modelos de linguagem para incluir sinônimos acadêmicos por meio da técnica de expansão de consulta [7], elevando a precisão do cruzamento de dados.
- **[Should] RF5** - O sistema deve aplicar algoritmos de reordenação estrutural (*reranking* [9]) nos documentos recuperados para refinar e selecionar os professores estatisticamente mais aptos.
- **[Could] RF6** - O aplicativo deve notificar o dispositivo móvel do usuário por meio do serviço Firebase assim que a análise computacional for concluída.
- **[Must] RF7** - O sistema deve exibir, na tela de resultados da proposta, uma lista ordenada das recomendações acompanhada de parágrafos curtos gerados de forma inteligente, justificando técnica e textualmente a aderência de cada professor [2].

2.2 Requisitos não-funcionais

Os requisitos a seguir determinam as premissas arquiteturais, restrições de infraestrutura e padrões de qualidade que o software deve seguir para operar de forma robusta e confiável.

- **[Must] RNF1** - O aplicativo cliente deve ser multiplataforma, garantindo execução consistente e fluida em diferentes sistemas operacionais.

- **[Must] RNF2** - O particionamento de texto dos currículos Lattes (*chunking*) [5] deve preservar a coesão e o contexto lógico das seções originais.
- **[Should] RNF3** - O mecanismo de busca deve priorizar a exatidão da recomendação e a varredura completa da base em detrimento do tempo de resposta inicial.
- **[Should] RNF4** - A recuperação de dados deve combinar similaridade semântica densa [6] e correspondência exata de palavras-chave para garantir alta assertividade.
- **[Must] RNF5** - A geração de justificativas pela Inteligência Artificial deve ser estritamente ancorada nos currículos recuperados, mitigando categoricamente alucinações cognitivas do modelo [4].
- **[Must] RNF6** - Os textos gerados devem passar por moderação rígida para assegurar um tom acadêmico, profissional e imparcial.
- **[Must] RNF7** - As justificativas produzidas devem ser rastreáveis e verificáveis contra os documentos originais, permitindo validação automatizada e humana do embasamento sugerido.
- **[Must] RNF8** - O sistema de recuperação deve aplicar um limiar mínimo de similaridade para garantir a qualidade do *matching*, abstendo-se de recomendar docentes caso nenhum alcance esta métrica.

3 Design

3.1 Dimensões de Design do Sistema RAG

A Tabela 2 apresenta as dimensões de design consideradas para a arquitetura do sistema de recomendação, organizadas pelas fases do pipeline RAG.

Tabela 2: Taxonomia de Opções de Design para a Implementação do RAG.

Dimensão / Etapa	**Opções Disponíveis**
1. Arquitetura e Processamento de Dados (Data Pipeline)	
Pipeline Structure (Arquitetura)	Indexing + Generation, Modular RAG, Naïve / Advanced RAG
Data Source (Fonte de Dados)	Open Internet, Internal documents, APIs/real-time, Multi-source
Data Loading (Ingestão)	Batch ingestion, Streaming ingestion
Parsing & Extraction (Extração)	Text extraction, Structured parsing
Chunking Strategy (Cortes)	Fixed-size, Semantic, Recursive, Sliding window, Specialized/Adaptive
Chunk Size & Overlap (Tamanho)	Small chunks, Large chunks
2. Representação e Armazenamento (Storage)	
Embedding Model (Vetores)	Pretrained, Domain-specific, Multimodal
Storage (Armazenamento / DB)	Vector DB, hybrid DB, Graph DB
Indexing Strategy (Indexação)	Flat index, ANN (approximate), hierarchical index
3. Estratégias de Busca (Retrieval)	
Retriever Type (Tipo de Busca)	Dense, Sparse, hybrid, Neural
Retrieval Strategy (Estratégia)	Top-k, Iterative, Recursive, Adaptive
Pre-retrieval Optimization (Pré-Busca)	Query rewriting, Index optimization
Post-retrieval Processing (Pós-Busca)	Re-ranking, Compression, Filtering
4. Geração e Modelagem (Generation & LLM)	
Augmentation Strategy (Contexto)	Simple concatenation, Prompt templates, Context selection
Prompt Engineering (Comandos)	Instruction, Few-shot, Chain-of-thought (CoT)
LLM Selection (Cérebro da IA)	General-purpose LLM, Domain-specific, Local vs API
Generation Strategy (Geração)	Single-pass, Multi-step, Tool-augmented
Memory Type (Memória)	Parametric (LLM), Non-parametric (KB)
5. Avaliação e Variantes (Evaluation)	
Evaluation Metrics (Métricas)	Accuracy, Relevance, Faithfulness, Retrieval metrics
Evaluation Method (Avaliação)	Automated metrics, human evaluation
RAG Variants (Variantes)	Multimodal, Graph RAG, Agentic RAG, Self-reflective
6. Infraestrutura e Operações (RAGOps)	
RAGOps Stack (Infraestrutura)	Critical, Essential, Enhancement layers
Caching Strategy (Otimização)	Query caching, Embedding caching
Security & Guardrails (Segurança)	Access control, Prompt filtering, Output moderation
Latency vs Quality (Tradeoff)	Fast retrieval, Deep retrieval
Scalability Strategy (Escala)	Distributed vector DB, Sharding, Load balancing

Justificativa das Decisões de Design

Com base na taxonomia apresentada, a implementação do módulo de Inteligência Artificial do aplicativo Emile foi guiada pelas seguintes escolhas arquiteturais, justificadas técnica e operacionalmente pelas restrições e objetivos do ambiente acadêmico:

- **1. Arquitetura e Processamento de Dados**
  Escolhas: Advanced RAG, Open Internet, Batch Ingestion, Text extraction e Adaptive Chunking.
  Justificativa: A escolha pela arquitetura Advanced RAG [1] justifica-se pela necessidade de superar o RAG ingênuo, aprimorando o processamento de textos densos. Os dados são originados da internet aberta, especificamente da base pública da Plataforma Lattes. A ingestão ocorre em lotes através de extração de texto, uma vez que os currículos são extensos e não sofrem alterações em tempo real, eliminando a necessidade e os custos de processamento via *streaming*. A segmentação adota a estratégia adaptativa [5], uma decisão vital para preservar a integridade estrutural das seções do Lattes (como resumos, linhas de pesquisa e publicações) e garantir que o contexto semântico não seja corrompido durante o particionamento.

- **2. Representação e Armazenamento**
  Escolhas: Pretrained (Qwen3-Embedding:4b), Hybrid DB e Flat index.
  Justificativa: O sistema utiliza o modelo de *embedding* pré-treinado **Qwen3-Embedding de 4 bilhões de parâmetros** [6], provido pelo ecossistema Ollama. Essa escolha justifica-se por ele possuir um vocabulário denso altamente capacitado para interpretar o jargão técnico-científico, gerando representações numéricas robustas de **2560 dimensões**. Essa alta dimensionalidade garante uma retenção profunda das características semânticas sem a necessidade de treinamentos proprietários. O armazenamento ocorre em um banco de dados híbrido, utilizando o PostgreSQL com *pgvector*, escolhido propositalmente para centralizar e aliar a robustez relacional à indexação vetorial. A estratégia de indexação adotada foi o índice plano. O motivo dessa escolha é que, ao calcular a distância matemática exata contra todos os vetores da base, o sistema prioriza a exatidão absoluta do pareamento em detrimento da velocidade.

- **3. Estratégias de Busca**
  Escolhas: Hybrid, Top-k, Query rewriting e Re-ranking.
  Justificativa: O motor de recuperação híbrido justifica-se por combinar a compreensão semântica com a correspondência exata de tecnologias e palavras-chave. Para mitigar a forte assimetria de vocabulário entre os alunos e os professores, aplica-se a reescrita de consulta na fase de pré-recuperação [7]. Em seguida, os resultados passam por uma reordenação estrutural na pós-recuperação [9]. O motivo dessa reordenação é a atribuição de pesos distintos para as diferentes seções do currículo Lattes fragmentado; o algoritmo recalcula a pontuação para dar maior ênfase a blocos de texto que reflitam com maior fidedignidade e peso empírico a área de atuação direta do professor.

- **4. Geração e Modelagem**
  Escolhas: General e API (Qwen3:30b), Templates, Context selection, Instruction, Chain-

of-Thought (CoT) e Non-parametric memory.
Justificativa: A interface de geração interage com o Grande Modelo de Linguagem **Qwen3 de 30 bilhões de parâmetros**, servido via API. O uso deste modelo específico justifica-se pela sua alta capacidade de raciocínio avançado, combinada à necessidade de garantir a privacidade absoluta dos dados das propostas dos discentes e a total independência de custos com processamento em nuvem. A construção do contexto baseia-se em moldes rígidos e seleção estrita para não confundir a atenção da IA com informações misturadas de múltiplos professores. O modelo é induzido a detalhar os motivos da adequação passo a passo, para que a justificativa entregue ao aluno seja pedagogicamente transparente e possua rastreabilidade lógica [2]. A geração utiliza estritamente a memória não-paramétrica [3], bloqueando o conhecimento prévio do modelo com o intuito expresso de anular alucinações sobre competências irreais [4].

- **5. Avaliação**
  Escolhas: Accuracy, Relevance, Faithfulness, Retrieval metrics, Automated metrics e Human evaluation.
  Justificativa: A qualidade do sistema é medida combinando métricas clássicas de recuperação com a fidelidade do texto gerado. A homologação dos resultados integra testes automatizados [16] e a validação humana. Essa dupla chancela é justificada porque não basta o algoritmo indicar alta similaridade matemática; é impreterível que a recomendação gerada faça sentido prático sob a ótica real de orientação acadêmica.

- **6. Infraestrutura e Operações**
  Escolhas: Deep retrieval, Moderation e Guardrails.
  Justificativa: O *trade-off* de latência versus qualidade foi deliberadamente inclinado para o aprofundamento da pesquisa. Essa decisão só foi possível graças à arquitetura assíncrona do Emile: como o usuário é notificado posteriormente via mensagem *push*, o servidor adquire o tempo necessário para executar buscas vetoriais minuciosas e gerações completas sem causar travamentos na interface do cliente. Além disso, as saídas do modelo passam por camadas de moderação de conteúdo, garantindo que o texto lido pelo aluno mantenha, obrigatoriamente, um tom institucional e ético.

3.2 Projeto UML

3.2.1 Diagrama de Classes

O Diagrama destaca o núcleo do pareamento: a relação entre o usuário proponente, a proposta submetida e a justificativa gerada para os matchings realizados.

A modelagem evidencia a dependência forte entre a proposta de TCC e sua respectiva justificativa, demonstrando que o texto explicativo gerado pelo pipeline de RAG pertence umbilicalmente ao ciclo de vida da proposta avaliada. A Figura 4 ilustra os atributos de cada classe e suas interações sistêmicas.

Figura 4: Diagrama de Classes das entidades principais de pareamento do sistema.

Fonte: Elaborado pelo autor (2026).

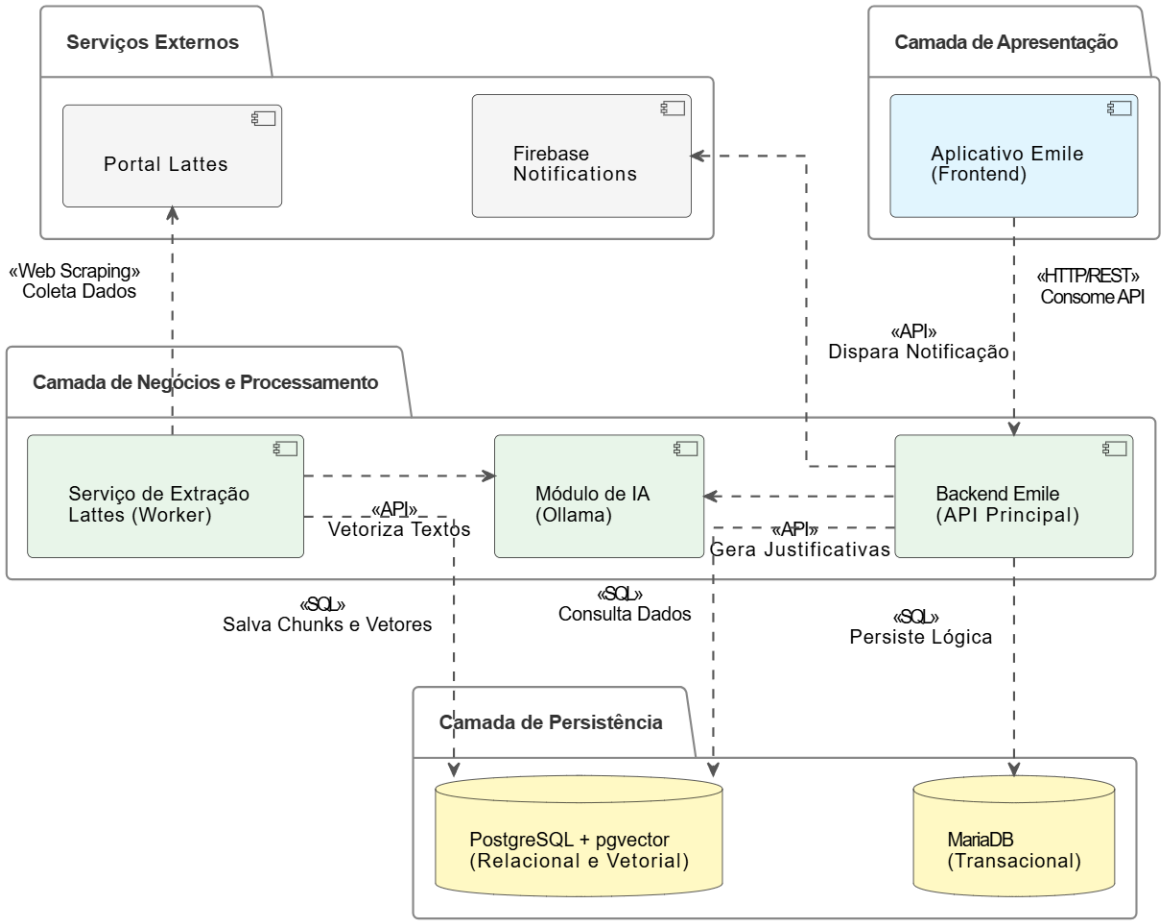
3.2.2 Diagrama de Componentes

Enquanto o diagrama de classes foca nas entidades internas do código, o Diagrama de Componentes é crucial para ilustrar a arquitetura física, modular e distribuída da solução. Sua importância neste trabalho reside na necessidade de justificar a orquestração de múltiplas tecnologias independentes, evidenciando o isolamento de responsabilidades e a tolerância a falhas.

O diagrama demonstra a separação clara entre a camada de apresentação, a camada de negócios e os serviços externos. Destaca-se, de forma particular, a estratégia de persistência poliglota: o uso do MariaDB para dados transacionais e gerenciamento de estado das propostas, operando em paralelo ao PostgreSQL munido da extensão *pgvector*, este último dedicado com exclusividade ao armazenamento analítico e processamento matemático de alta dimen-

sionalidade demandado pela Inteligência Artificial e arquitetura RAG. A Figura 5 detalha essas conexões e protocolos de comunicação.

Figura 5: Diagrama de Componentes e interações sistêmicas da arquitetura RAG.



Fonte: Elaborado pelo autor (2026).

3.3 Visão arquitetural

O sistema adota uma arquitetura distribuída e modular, fundamentada primariamente no estilo Cliente-Servidor, mas expandida para incorporar Processamento Assíncrono, Pipelines de Dados Isolados e Persistência híbrida. Essa composição arquitetural foi projetada para suportar a alta demanda computacional inerente às operações de Inteligência Artificial sem comprometer a responsividade da interface com o usuário.

A comunicação inicial segue o padrão Cliente-Servidor clássico: o aplicativo móvel Emile (Cliente), desenvolvido com o *framework* Qt/QML, interage com a API principal em Python (Servidor) por meio de requisições hTTP e protocolo REST. O servidor atua como o orquestrador das regras de negócio e autenticação.

Para garantir o desempenho e a escalabilidade, a arquitetura divide o processamento do sistema de recomendação em dois ciclos operacionais distintos e completamente desacoplados: o pipeline *offline* e o pipeline *online*.

O pipeline *offline* é responsável exclusivamente pela preparação da base de conhecimento. O Serviço de Extração Lattes atua como um *worker* de retaguarda isolado do fluxo do usuário. Seu papel é realizar a raspagem de dados, consumir o módulo de IA para gerar as representações vetoriais dos currículos e popular o banco de dados. Esse processo ocorre de forma autônoma e programada, atualizando o índice semântico sem causar qualquer impacto na latência do aplicativo móvel.

Por outro lado, o pipeline *online* lida com a interação direta do usuário e é orquestrado de maneira assíncrona. Assim que o estudante cadastra ou atualiza uma proposta de TCC, o backend registra a transação e libera a requisição HTTP imediatamente, garantindo uma interface fluida. Em segundo plano, o sistema executa o fluxo RAG: ele converte a proposta do aluno em vetor, calcula a similaridade no banco de dados previamente populado, consulta o modelo de linguagem para redigir a justificativa para os professores retornados e persiste as informações consolidadas. Ao finalizar essa operação intensiva, o servidor aciona o Firebase *Cloud Messaging* para emitir uma notificação *push* ao dispositivo do usuário, informando a conclusão do *matching*.

Sustentando ambas as operações, o sistema faz uso da Persistência híbrida, com o domínio de dados segmentado:

- Camada Transacional (MariaDB): Dedicada às operações rápidas, mantendo a consistência e a integridade do estado lógico do sistema.
- Camada Analítica/Vetorial (PostgreSQL + pgvector): Dedicada exclusivamente ao armazenamento multidimensional gerado no fluxo *offline* e à execução dos cálculos matemáticos de similaridade de cosseno demandados no fluxo *online*.

Essa separação cristalina, ilustrada no diagrama de componentes na Figura 5, garante alta disponibilidade e previsibilidade de latência, blindando as operações cotidianas do aplicativo contra os gargalos computacionais inerentes ao processamento de Inteligência Artificial.

3.4 Modelo de Banco de Dados

Para garantir a integridade, a performance e o armazenamento adequado das informações manipuladas pelo sistema Emile e pelo módulo de Inteligência Artificial, a modelagem de dados adota a abordagem de Persistência híbrida. Em vez de centralizar todas as operações em um único Sistema Gerenciador de Banco de Dados, a arquitetura particiona as entidades de acordo com a natureza de suas cargas de trabalho.

O primeiro domínio é gerenciado pelo **MariaDB**, um banco de dados relacional clássico otimizado para transações rápidas de leitura e escrita. Ele atua como a fonte de verdade do aplicativo móvel, mantendo a consistência do estado lógico das propostas, dos perfis de

usuários e das justificativas geradas. Este isolamento assegura que as requisições cotidianas dos alunos não sofram concorrência de recursos.

O segundo domínio opera no **PostgreSQL**, equipado com a extensão *pgvector*. Este atua como um banco de dados relacional e vetorial voltado à Inteligência Artificial. A decisão de segregar os fragmentos de currículos Lattes neste banco decorre da necessidade exclusiva de armazenar vetores multidimensionais e aplicar cálculos de similaridade de cosseno diretamente no nível do banco para o pipeline de RAG, operação inviável no banco transacional primário.

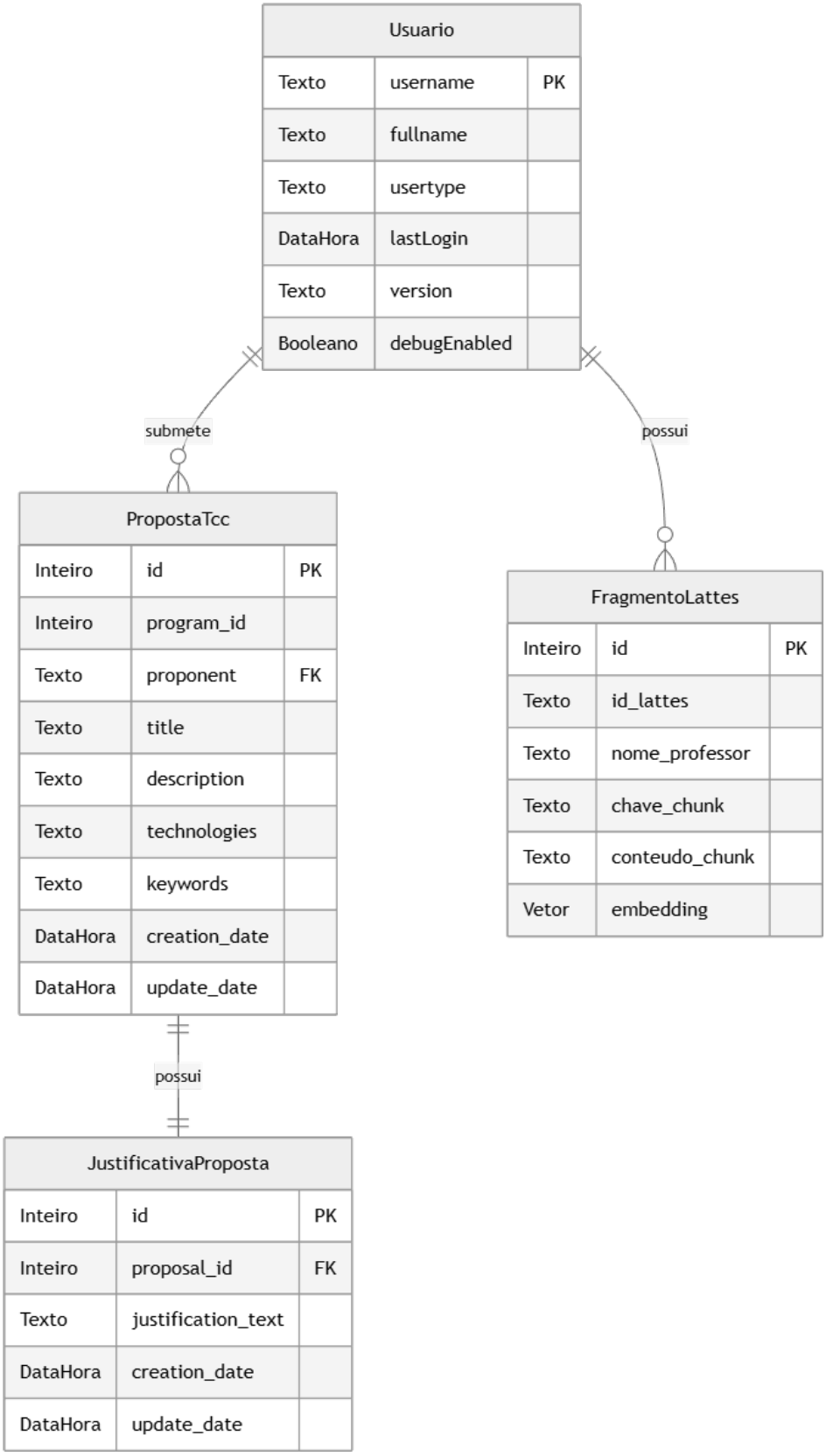
3.4.1 Modelo Lógico de Dados

No nível lógico, o ecossistema de dados é composto pelas seguintes entidades principais:

- **Usuário:** representa o estudante proponente ou o professor orientador que interage com o aplicativo Emile, contendo informações de perfil, credenciais e histórico de acesso;
- **Proposta de TCC:** entidade conceitual que representa o projeto acadêmico submetido por um aluno em busca de orientação, armazenando o escopo, palavras-chave e tecnologias requeridas;
- **Justificativa da Proposta:** representa o parecer gerado automaticamente pelo módulo de Inteligência Artificial, contendo o texto explicativo que embasa os motivos de adequação entre o currículo do professor e a proposta;
- **Fragmento Lattes:** representa a unidade de conhecimento extraída publicamente do currículo do corpo docente, armazenando tanto o conteúdo textual bruto quanto a sua representação semântica em formato de vetor matemático.

A Figura 6 ilustra o Diagrama Entidade-Relacionamento conceitual, mapeando as interações e cardinalidades essenciais entre essas entidades.

Figura 6: Modelo Lógico ilustrando as entidades abstratas e regras de negócio do sistema.



Fonte: Elaborado pelo autor (2026).

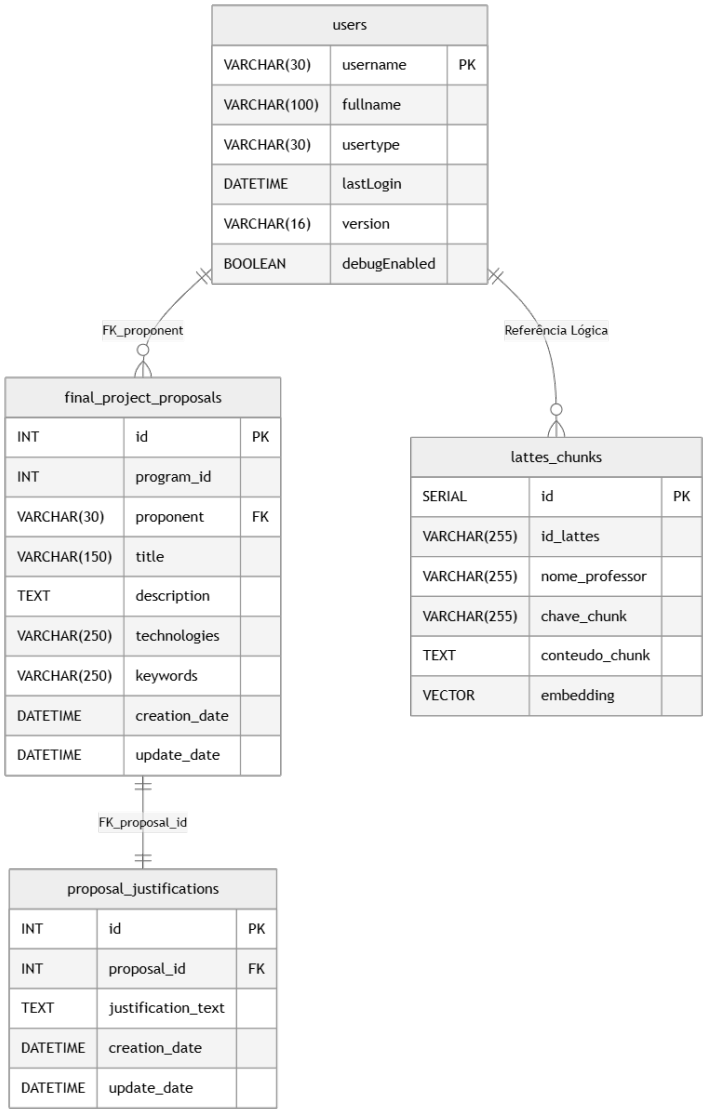
3.4.2 Modelo Físico de Dados

Conforme a arquitetura híbrida estabelecida, no nível físico, o sistema é estruturado nas seguintes tabelas principais:

- **users:** alocada no **MariaDB**, armazena os dados transacionais de autenticação e controle de versão utilizando tipagens padrão como VARChAR e DATETIME;
- **final_project_proposals:** alocada no **MariaDB**, consolida os dados das propostas e utiliza chaves estrangeiras físicas para se relacionar de forma estrita com a tabela de usuários correspondente;
- **proposal_justifications:** alocada no **MariaDB**, armazena as justificativas dos matchings realizados e vincula-se diretamente ao ID da proposta atrelada;
- **lattes_chunks:** isolada no **PostgreSQL**, estrutura os metadados textuais extraídos em grandes proporções e utiliza a tipagem especializada `VECTOR`, provida pela extensão *pgvector*, para viabilizar a indexação espacial e o pipeline do RAG Avançado.

A Figura 7 ilustra o esquema físico final e o particionamento do ecossistema.

Figura 7: Modelo Físico de Dados particionado entre o MariaDB e PostgreSQL.



Fonte: Elaborado pelo autor (2026).

Cabe destacar uma particularidade arquitetural mapeada na modelagem física: a relação entre a tabela `users` e a tabela `lattes_chunks`, denotada no diagrama como uma *Referência Lógica*. Como as tabelas residem em infraestruturas distintas, é tecnicamente inviável a declaração de uma restrição física de chave estrangeira entre elas no nível do banco de dados.

Para contornar essa limitação intrínseca da Persistência híbrida, a integridade referencial é delegada à camada de negócios da aplicação. O backend do Emile, atua como o orquestrador exclusivo desse relacionamento, responsabilizando-se por cruzar os identificadores lógicos cruzados entre a base transacional e a base vetorial. Essa abordagem garante o rigor relacional dos dados sem criar um acoplamento indesejado entre os servidores de banco de dados.

4 Testes de software

Para a validação das funcionalidades do aplicativo Emile e de seu respectivo módulo de Inteligência Artificial dedicado ao sistema de recomendação de orientadores, adotou-se uma estratégia focada em testes funcionais e de integração. A opção por priorizar esse modelo em detrimento de testes unitários tradicionais justifica-se pela arquitetura altamente distribuída e assíncrona do sistema. Neste contexto, a integridade do ecossistema e a qualidade das respostas dependem diretamente do correto fluxo de dados entre os componentes e da orquestração adequada com os Modelos de Linguagem de Grande Escala.

As validações foram centralizadas na ferramenta Postman, por meio da qual simulou-se o comportamento do aplicativo cliente ao disparar requisições contra os *endpoints* do *backend*. A execução dos testes dividiu-se em duas macroetapas metodológicas:

1. **Validação Transacional:** Focada no ciclo de vida padrão da aplicação, garantindo o correto recebimento, persistência no banco de dados e tráfego das propostas de projeto submetidas pelos alunos.

2. **Homologação do *Pipeline* Analítico:** Etapa de avaliação isolada do mecanismo de busca, recuperação e geração de texto. Utilizando *payloads* baseados em propostas acadêmicas reais contendo título, descrição detalhada, pilhas tecnológicas e palavras-chave, os testes visaram mensurar o desempenho do algoritmo ao correlacionar as demandas técnicas do projeto do aluno com o histórico de publicações e áreas de atuação dos professores cadastrados.

4.1 Testes Comparativos e Análise de Estratégias de *Matching*

Para justificar empiricamente a adoção de uma arquitetura de alta complexidade computacional, implementou-se um endpoint de teste (`POST /test-match-professors`). O objetivo desta rota não é o uso em produção, mas a condução de simulações comparativas para medir a eficácia da solução proposta contra as abordagens tradicionais que serviram de *baseline* neste estudo.

Ao receber uma única proposta, este endpoint paraleliza a avaliação do texto submetido e executa o pareamento através de três vertentes distintas:

- **Busca por Palavras-Chave (TF-IDF):** O sistema realiza uma contagem crua de termos coincidentes entre a proposta e os currículos, com base no algoritmo TF-IDF, enviando o volume resultante para o modelo de linguagem.

- **RAG Ingênuo (*Naïve RAG*):** Representa o fluxo basal no qual o texto é vetorizado e a recuperação de dados baseia-se estritamente na similaridade matemática, descartando a adoção de buscas híbridas. Os fragmentos retornados são enviados diretamente ao modelo de linguagem para a geração da resposta, carecendo de etapas de otimização

refinadas, como a expansão de consulta (*query rewriting*) pré-busca ou a reordenação estrutural (*reranking*) pós-busca.

- **RAG Avançado (*Advanced RAG*):** O pipeline completo desenvolvido para este trabalho, englobando a recuperação híbrida e o refinamento contextual.

Requisição Comparativa (Payload de Entrada):


```
{
  "program_id": 10,
  "proponent": "Emanuel Sena",
  "title": "Desenvolvimento de um Sistema Distribuído para Monitoramento e
            Gerenciamento de Recursos em Sistemas Operacionais",
  "description": "Projeto e implementação de uma plataforma distribuída para
                  monitoramento em tempo real de recursos de sistemas operacionais,
                  incluindo utilização de CPU, memória, disco e processos em
                  múltiplas máquinas. A solução utilizará uma arquitetura
                  distribuída para coleta e agregação de métricas, permitindo a
                  análise do comportamento dos nós da rede. A interface gráfica
                  será desenvolvida com Qt/QML, oferecendo dashboards interativos
                  e visualização de dados em tempo real. O trabalho abordará
                  conceitos de sistemas operacionais, como processos, threads,
                  concorrência e comunicação entre processos, além de temas
                  relacionados a sistemas distribuídos, como tolerância a falhas,
                  escalabilidade e comunicação em rede.",
  "technologies": "C++, Qt, QML, Qt Quick, Qt Network, gRPC, SQLite, Docker,
                   Linux, CMake",
  "keywords": "Sistemas Operacionais, Sistemas Distribuídos, Monitoramento de
               Recursos, Concorrência, Threads, Processos, IPC, Tolerância a
               Falhas, Escalabilidade, Comunicação em Rede, Qt, QML"
}
```


Resposta Comparativa (JSON de Saída Estruturado):


```
{
    "status": "sucesso",
    "titulo_proposta": "Desenvolvimento de um Sistema Distribuído para Monitoramento...",
    "metricas_globais": {
        "Tempo de Geração do Vetor da Proposta (s)": 0.25
    },
    "estrategias": {
        "TF-IDF": {
            "metricas": {
                "Tempo de Busca dos Chuncks (s)": 0.25,
                "Tempo de Geração LLM (s)": 838.79,
```

```
    "Tempo Total da Estratégia (s)": 839.03,
    "Score Médio (Top 3)": 0.97
},
"justificativa": "Com base no alinhamento entre a proposta e o histórico
acadêmico, os seguintes professores são recomendados:\n\n1. Allan Edgard
Silva Freitas\nO professor Allan Edgard Silva Freitas possui experiência
direta em sistemas distribuídos com foco em tolerância a falhas, como
evidenciado na avaliação do vCubeChain em sistemas assíncronos sujeitos a
falhas, e em projetos de monitoramento, exemplificados pelo Sistema de
Monitoramento de Parâmetros Físico-Químicos em Piscicultura. Essa combinação
de expertise em arquiteturas distribuídas resilientes e sistemas de
monitoramento alinha-se perfeitamente com o objetivo do projeto do aluno,
que visa desenvolver uma solução para monitoramento em tempo real de
recursos de sistemas operacionais, garantindo robustez e escalabilidade.
Sua experiência prática em abordagens distribuídas e monitoramento torna-se
essencial para orientar os desafios técnicos específicos do trabalho
proposto.\n\n2. Manoel Carvalho Marques Neto\nO professor possui experiência
direta com arquiteturas distribuídas escaláveis, como evidenciado no trabalho
\"Um Modelo de Arquitetura Distribuída Configurável para Construção de
Sistemas Escaláveis\", além de atuação em soluções de mineração de dados
paralela e distribuída, conforme registrado em \"SMART MINING - Uma solução
de mineração de dados paralela e distribuída baseada em serviços\". Essa
expertise alinha-se perfeitamente com o foco do projeto do aluno em
desenvolver um sistema distribuído para monitoramento de recursos do sistema
operacional, utilizando tecnologias como gRPC e Qt.\n\n3. Sandro Santos
Andrade\nO professor Sandro Santos Andrade possui experiência direta no
desenvolvimento de sistemas distribuídos utilizando as mesmas tecnologias
propostas pelo aluno, como Qt, C++ e QML, conforme evidenciado no projeto
\"Meg: Arquitetura de Software, Frameworks e Geradores de Código para
Aplicativos Móveis\", que aborda comunicação em rede e concorrência em
arquiteturas distribuídas. Sua expertise em frameworks baseados em Qt para
monitoramento e gestão de recursos, aliada à implementação de interfaces
gráficas interativas, alinha-se perfeitamente com os objetivos do projeto
do aluno, permitindo orientação técnica especializada na execução da
plataforma de monitoramento em tempo real.",
"resultados": [
    {
        "professor": "Allan Edgard Silva Freitas",
        "score_final": 1.00,
        "chave_chunk": "bancas"
    },
    {
        "professor": "Manoel Carvalho Marques Neto",
        "score_final": 0.95,
        "chave_chunk": "bancas"
```

```
        },
        {
            "professor": "Sandro Santos Andrade",
            "score_final": 0.95,
            "chave_chunk": "projetos_pesquisa"
        }
    ]
},
"Naive RAG": {
    "metricas": {
        "Tempo de Busca dos Chuncks (s)": 0.06,
        "Tempo de Geração LLM (s)": 291.71,
        "Tempo Total da Estratégia (s)": 291.77,
        "Score Médio (Top 3)": 0.50
    },
    "justificativa": "Com base no alinhamento entre a proposta e o histórico
    acadêmico, os seguintes professores são recomendados:\n\n1. Sandro Santos
    Andrade\nO professor Sandro Santos Andrade possui experiência direta em
    sistemas distribuídos baseados em componentes, como demonstrado nos projetos
    ATOME (configuração e implantação de sistemas distribuídos) e IRIDIUM
    (framework para sistemas adaptativos distribuídos). Sua atuação no
    desenvolvimento da infraestrutura de comunicação distribuída (QCM) e em
    frameworks Qt, como QtScraper, alinha-se perfeitamente com a necessidade
    de implementar uma plataforma distribuída utilizando gRPC e Qt/QML para
    monitoramento em tempo real de recursos de sistemas operacionais.\n\n2.
    Allan Edgard Silva Freitas\nO professor Allan Edgard Silva Freitas possui
    experiência direta na área de sistemas distribuídos, com um trabalho em
    andamento sobre \"Plataforma de Gestão de Estado de Hosts para Ambientes
    de Sistemas Distribuídos Escaláveis Horizontalmente\", alinhando-se
    perfeitamente ao foco do projeto do aluno em monitoramento e gerenciamento
    de recursos em ambientes distribuídos. Além disso, ele concluiu um Trabalho
    de Conclusão de Curso (\"Escalonador de Tarefas de Tempo Real Para um
    Simulador de Sistemas Distribuídos\"), demonstrando expertise em conceitos-
    chave como escalabilidade e comunicação em rede, essenciais para a
    implementação proposta com gRPC e Qt. Essa combinação de experiência prática
    e teórica garante suporte técnico preciso para o desenvolvimento da
    plataforma descrita.\n\n3. Manoel Carvalho Marques Neto\nO professor Manoel
    Carvalho Marques Neto possui experiência direta em sistemas de monitoramento,
    com destaque para a implementação de \"Providers WBEM para a Arquitetura
    Linux\" em seu Trabalho de Conclusão de Curso, que aborda gerenciamento e
    coleta de parâmetros de sistemas operacionais. Sua trajetória inclui
    projetos como \"Sistema de monitoramento de parâmetros físico-químicos em
    tanques de piscicultura\", demonstrando expertise em desenvolver plataformas
    distribuídas para coleta e análise de dados em tempo real. Essa experiência
    alinha-se perfeitamente com o objetivo do aluno de criar um sistema para
```

```
    monitoramento de recursos de sistemas operacionais, utilizando arquitetura
    distribuída e comunicação em rede.",
    "resultados": [
        {
            "professor": "Sandro Santos Andrade",
            "score_final": 0.52,
            "chave_chunk": "orientacoes"
        },
        {
            "professor": "Allan Edgard Silva Freitas",
            "score_final": 0.51,
            "chave_chunk": "orientacoes"
        },
        {
            "professor": "Manoel Carvalho Marques Neto",
            "score_final": 0.48,
            "chave_chunk": "orientacoes"
        }
    ]
},
"Advanced RAG": {
    "metricas": {
        "Tempo de Busca dos Chuncks (s)": 0.08,
        "Tempo de Geração LLM (s)": 380.49,
        "Tempo Total da Estratégia (s)": 380.56,
        "Score Médio (Top 3)": 0.47
    },
    "justificativa": "Com base no alinhamento entre a proposta e o histórico
    acadêmico, os seguintes professores são recomendados:\n\n1. Sandro Santos
    Andrade\nO professor Sandro Santos Andrade possui experiência direta no
    desenvolvimento de sistemas distribuídos utilizando Qt, QML e C++, conforme
    demonstrado no projeto Meg, que aborda explicitamente comunicação em rede e
    concorrência|conceitos centrais ao projeto proposto. Sua liderança no
    projeto Qt Component Model (QCM), focado em arquiteturas distribuídas
    baseadas em componentes, alinha-se perfeitamente com os requisitos técnicos
    de monitoramento em tempo real e gestão de recursos em múltiplas máquinas.
    Essa experiência prática garante uma orientação técnica adequada para a
    implementação da plataforma descrita, utilizando as tecnologias
    especificadas.\n\n2. Allan Edgard Silva Freitas\nO professor Allan Edgard
    Silva Freitas possui experiência direta em sistemas distribuídos, com
    destaque para a dissertação \"Plataforma de Gestão de Estado de Hosts para
    Ambientes de Sistemas Distribuídos Escaláveis Horizontalmente\", que aborda
    monitoramento e gestão de recursos em ambientes distribuídos, alinhando-se
    perfeitamente ao foco do projeto do aluno em monitoramento de recursos de
    sistemas operacionais. Sua trajetória inclui também trabalhos relacionados
```

```
            a escalabilidade, comunicação em rede e arquiteturas distribuídas,
            fundamentais para a implementação de uma plataforma de monitoramento em
            tempo real com as tecnologias propostas (gRPC, Docker, C++). Essa
            experiência prática e temática garante suporte técnico e conceitual
            adequado ao desenvolvimento do sistema proposto.\n\n3. Flávia Maristela
            Santos Nascimento\nO professor Flávia Maristela Santos Nascimento possui
            expertise direta em sistemas de tempo real distribuídos com tolerância a
            falhas, alinhando-se perfeitamente ao foco do projeto do aluno em um sistema
            distribuído para monitoramento de recursos operacionais. Sua linha de
            pesquisa em \"Sistemas de Tempo Real\" e \"Tolerância a Falhas\" aborda
            justamente os desafios de escalabilidade, comunicação em rede e recuperação
            de falhas, fundamentais para a implementação do sistema proposto. A
            experiência do professor em protocolos de comunicação e análise de
            escalonamento também complementa as necessidades técnicas do projeto, como
            a coleta e agregação em tempo real de métricas em múltiplas máquinas.",
            "resultados": [
                {
                    "professor": "Sandro Santos Andrade",
                    "score_final": 0.52,
                    "chave_chunk": "projetos_pesquisa"
                },
                {
                    "professor": "Allan Edgard Silva Freitas",
                    "score_final": 0.45,
                    "chave_chunk": "orientacoes"
                },
                {
                    "professor": "Flávia Maristela Santos Nascimento",
                    "score_final": 0.45,
                    "chave_chunk": "linhas_de_pesquisa"
                }
            ]
        }
    }
}
```

Os resultados empíricos retornados por este endpoint expõem de forma clara as nuances operacionais e o *trade-off* entre latência e precisão semântica inerente a cada método. A partir da amostragem do *JSON* de resposta (cujas métricas de tempo foram convertidas para segundos para fins de legibilidade), três comportamentos arquiteturais distintos são evidenciados:

Primeiramente, a estratégia **TF-IDF** apresentou um tempo de execução muito alto para um sistema interativo, aproximadamente 839 segundos. Por depender exclusivamente da frequência de termos e carecer de um filtro semântico eficiente na etapa de recuperação, o algoritmo injeta um volume massivo e ruidoso de fragmentos na janela de contexto do LLM. Essa sobrecarga de *tokens* é a responsável direta pela explosão no tempo de inferência e geração da resposta.

Em contrapartida, o ***Naïve RAG*** demonstrou ser a abordagem mais enxuta computacionalmente, cerca de 291 segundos. Ao utilizar a busca vetorial simples, ele resolve o gargalo de contexto do TF-IDF filtrando os currículos matematicamente antes da geração. Contudo, embora consiga identificar perfis aderentes às tecnologias explícitas como por exemplo, a proficiência em C++ e Qt do Prof. Sandro, sua recuperação carece de profundidade, limitando-se a uma similaridade de superfície.

Por fim, o ***Advanced RAG***, com tempo total de aprox. 380 segundos, comprova empiricamente a necessidade de sua adoção. O acréscimo de cerca de 90 segundos em relação ao modelo ingênuo é o custo computacional exigido pelas etapas de expansão de consulta (*query rewriting*) e reordenação inteligente (*reranking*). Esse refinamento permite que a IA transcenda o mapeamento superficial de ferramentas e compreenda o cerne arquitetural do projeto. Isso fica evidente nos resultados: o RAG Avançado foi o único capaz de realizar um isolamento semântico profundo a ponto de incluir a Prof. Flávia Maristela nas recomendações. Enquanto os métodos basais focaram apenas nas linguagens de programação, o *pipeline* avançado identificou que as necessidades abstratas da proposta como ”tolerância a falhas”e ”tempo real”possuíam aderência exata com as linhas de pesquisa teóricas da docente, garantindo um pareamento qualitativamente superior.

5 Implantação

A implantação do sistema Emile e de seu respectivo módulo de Inteligência Artificial foi projetada para garantir escalabilidade, segurança e previsibilidade na comunicação entre os serviços. Para fins de documentação, esta etapa divide-se no mapeamento da arquitetura de implantação (distribuição física e lógica dos componentes) e na estratégia de publicação para o usuário final.

5.1 Arquitetura de Implantação

A arquitetura física do sistema adota um modelo distribuído, mapeando os componentes lógicos de software para seus respectivos nós de execução e hardware. O Diagrama de

Implantação (Figura 8) ilustra como esses artefatos estão alocados e como se comunicam de forma assíncrona e síncrona no ambiente de produção.

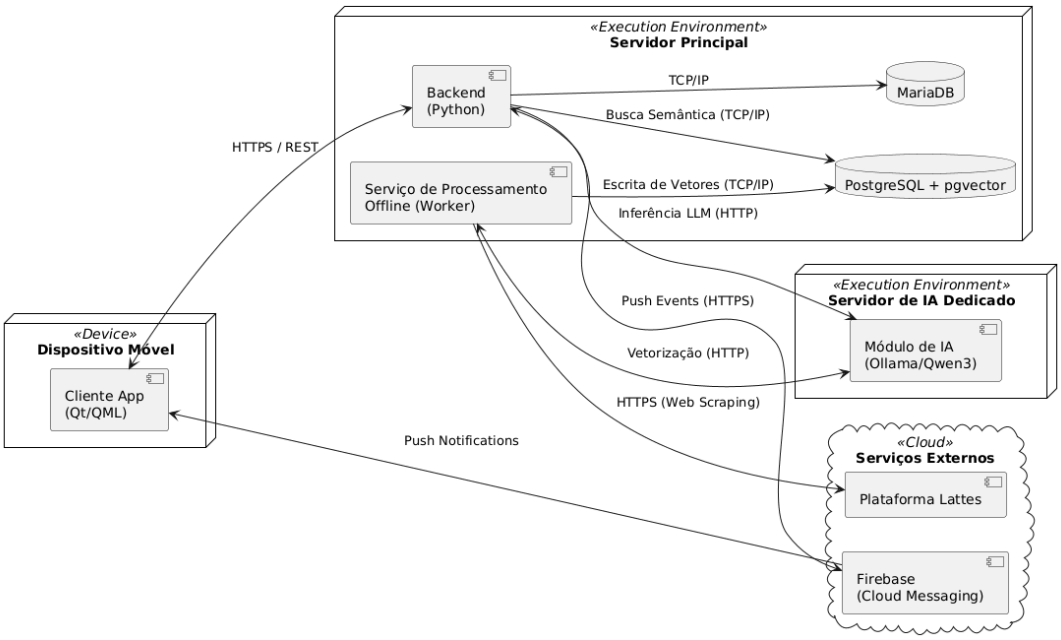


Figura 8: Diagrama de Implantação ilustrando a distribuição física dos nós e componentes do sistema.

Fonte: Autoria própria (2026).

A infraestrutura é estruturada em torno de nós computacionais que abrigam os componentes essenciais para o funcionamento ponta a ponta do RAG Avançado:

- **Cliente App:** Desenvolvido nativamente em Qt/QML, atua como o ponto de interação com o usuário. Este componente é implantado nos dispositivos móveis (Android e iOS), operando como um nó cliente que consome os serviços da aplicação via requisições hTTP seguras e recebe alertas de background.

- **Backend:** Componente central desenvolvido em Python e alocado no servidor em nuvem. Ele atua como o orquestrador do fluxo *online*, responsável por processar as regras de negócio da aplicação, gerenciar as requisições dos usuários e orquestrar as avaliações de propostas de TCC em tempo real.

- **Serviço de Processamento Offline:** Componente isolado operando como um *worker* de retaguarda. Ele gerencia exclusivamente a preparação da base de conhecimento do ecossistema.

- **MariaDB:** Componente de persistência relacional hospedado no servidor. É responsável exclusivamente por gerenciar o estado transacional do sistema, garantindo a integridade dos cadastros de usuários e propostas submetidas, operando de forma isolada das cargas de trabalho da Inteligência Artificial.

- **PostgreSQL + pgvector:** Componente de persistência híbrida e analítica. Diferente do banco transacional, este componente abriga a extensão vetorial, sendo dedicado ao armazenamento matemático de alta dimensionalidade e aos cálculos espaciais de similaridade entre os *chunks* dos currículos e as propostas.
- **Módulo de Inteligência Artificial:** Componente de processamento neural alocado em um servidor dedicado, estruturalmente isolado da aplicação principal. Ele integra o orquestrador Ollama e os modelos de linguagem fundacionais (`qwen3:30b` e `qwen3-embedding:4b`). Ao segregar esse componente em uma infraestrutura própria, o sistema garante que a alta demanda de processamento da IA não concorra com os recursos do *backend* ou do banco de dados.
- **Firebase:** Componente de mensageria operado externamente na infraestrutura em nuvem do Google. Ele é acionado pelo *backend* assim que o Módulo de IA conclui a etapa de pareamento, assumindo a responsabilidade de rotear e entregar as notificações *push* de forma assíncrona ao Cliente App.

5.2 Publicação e Distribuição

O processo de disponibilização do Cliente App segue as diretrizes oficiais de empacotamento da Google Play Store (para dispositivos Android) e da Apple App Store (para iOS). A compilação cruzada do código-fonte, viabilizada pelo *framework* Qt, resulta na geração dos pacotes binários nativos exigidos por cada ecossistema.

Como o sistema utiliza processamento assíncrono, o aplicativo demanda acesso contínuo à rede e autorização nativa do sistema operacional para a recepção de notificações pelo Firebase em plano de fundo. Para garantir a aprovação nos processos de revisão das lojas de aplicativos, o uso dessas permissões é justificado de forma explícita na política de privacidade institucional.

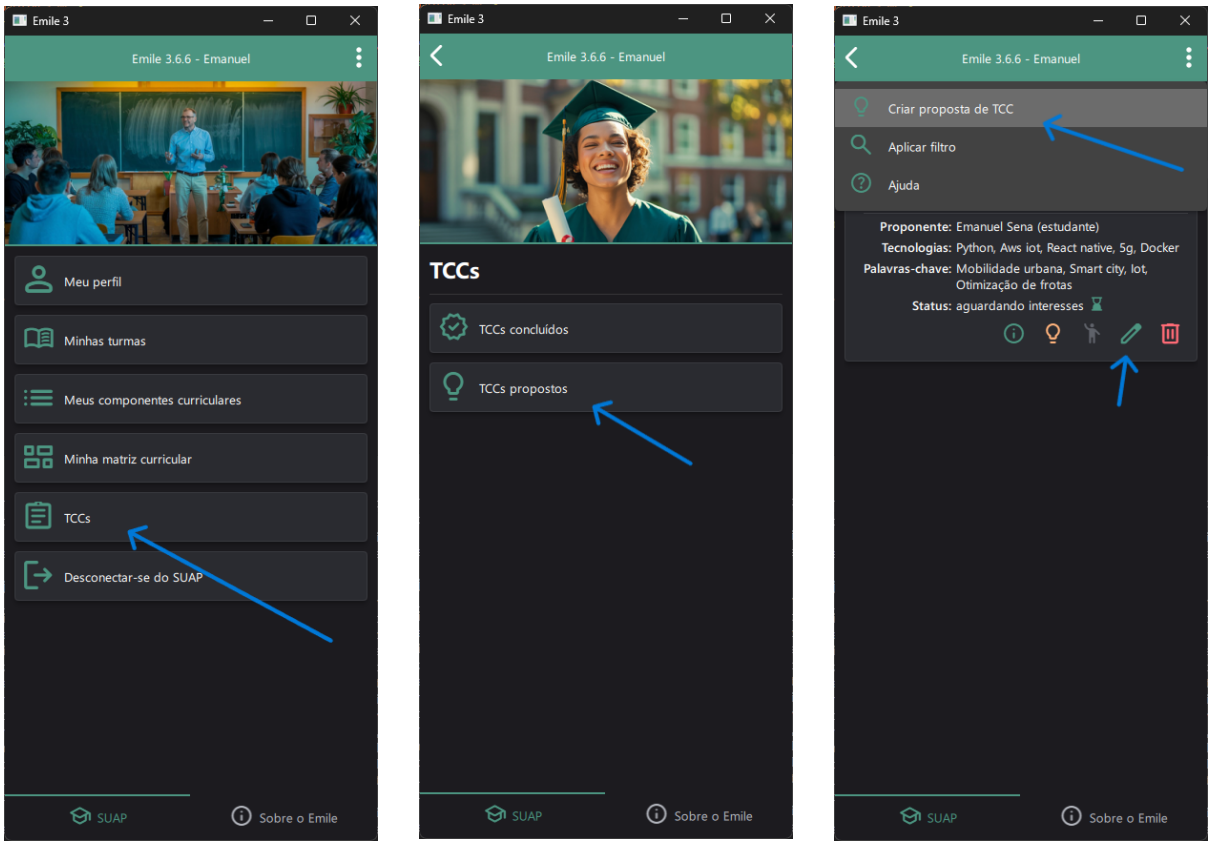
Toda a complexidade computacional inerente ao armazenamento vetorial e à inferência dos Grandes Modelos de Linguagem é delegada aos componentes alocados no servidor, mantendo o Cliente App como uma camada estritamente leve, altamente responsiva e otimizada para minimizar o consumo de bateria e processamento dos dispositivos dos discentes.

6 Manual do Usuário

O fluxo foi projetado para ser intuitivo, transparente e estar perfeitamente integrado às operações rotineiras de gerenciamento de TCC.

6.1 Navegação e Cadastro de Propostas

Para iniciar o processo, o usuário deve realizar a navegação a partir da tela inicial do aplicativo Emile, acessando o módulo específico de TCC e, em seguida, selecionando a seção de **TCCs Propostos**. A Figura 9 ilustra esse fluxo passo a passo, evidenciando o caminho percorrido pelo estudante na interface móvel até a listagem das propostas.

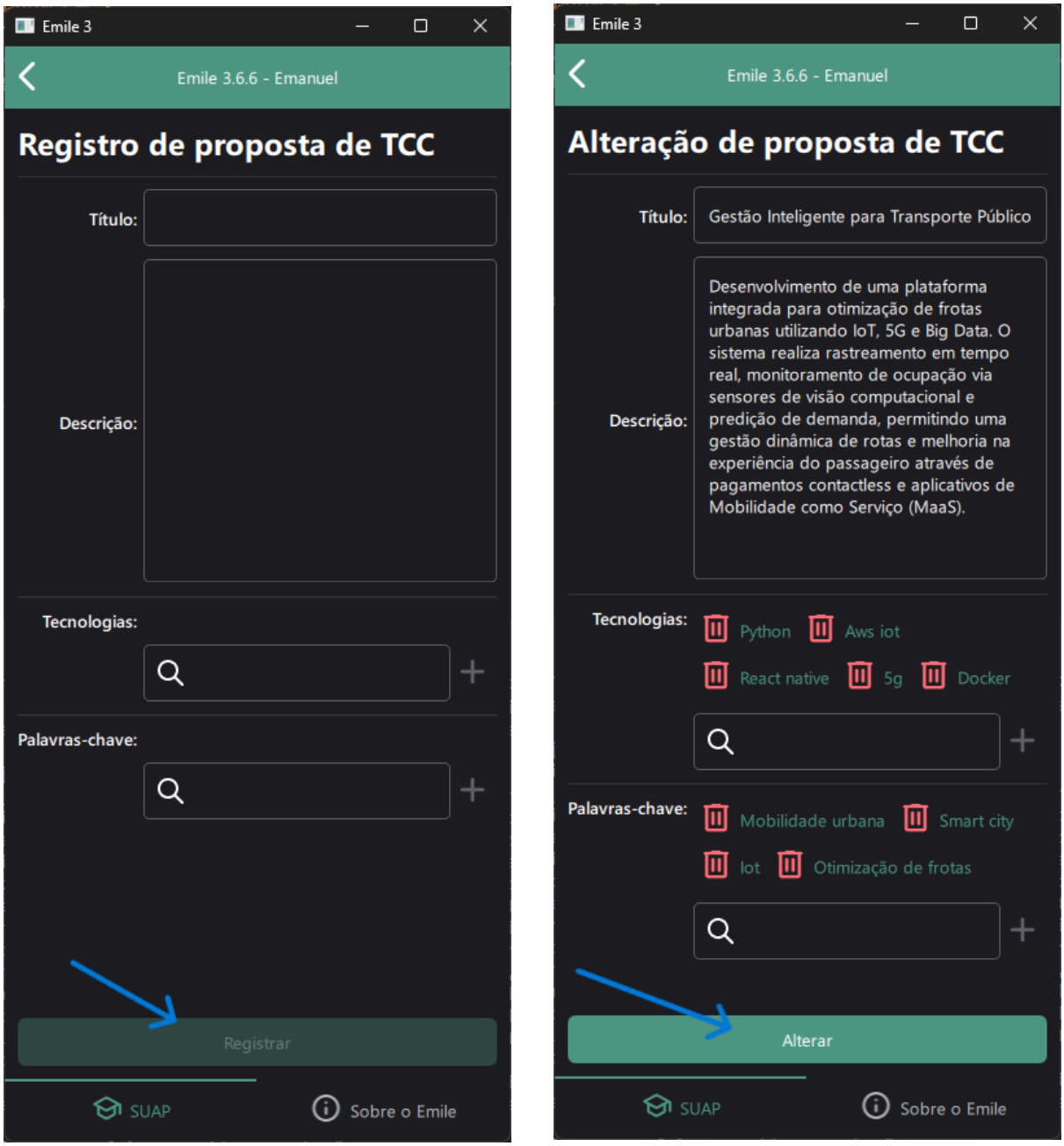


Módulo de TCC | TCCs Propostos | Criar um TCC

Figura 9: Fluxo de navegação desde a tela inicial até a seção de TCCs Propostos, onde é possível ver a listagem dos TCCs e acessar a tela de cadastro.

Ao alcançar esta interface, o usuário possui a liberdade de submeter uma nova ideia de projeto ou alterar uma submissão existente. Ao **cadastrar** uma nova proposta, o aplicativo registra os dados textuais e dispara, de maneira assíncrona, o motor de Inteligência Artificial no *backend* para encontrar os docentes mais compatíveis. De forma análoga, caso o aluno decida **editar** o escopo ou as tecnologias de uma proposta previamente cadastrada, o sistema detecta a mudança e reinicia o processamento analítico para atualizar o *matching* com base nas novas informações.

A Figura 10 ilustra as telas de gerenciamento do aplicativo, apresentando as interfaces de cadastro e edição lado a lado.



Tela de Cadastro

Tela de Edição

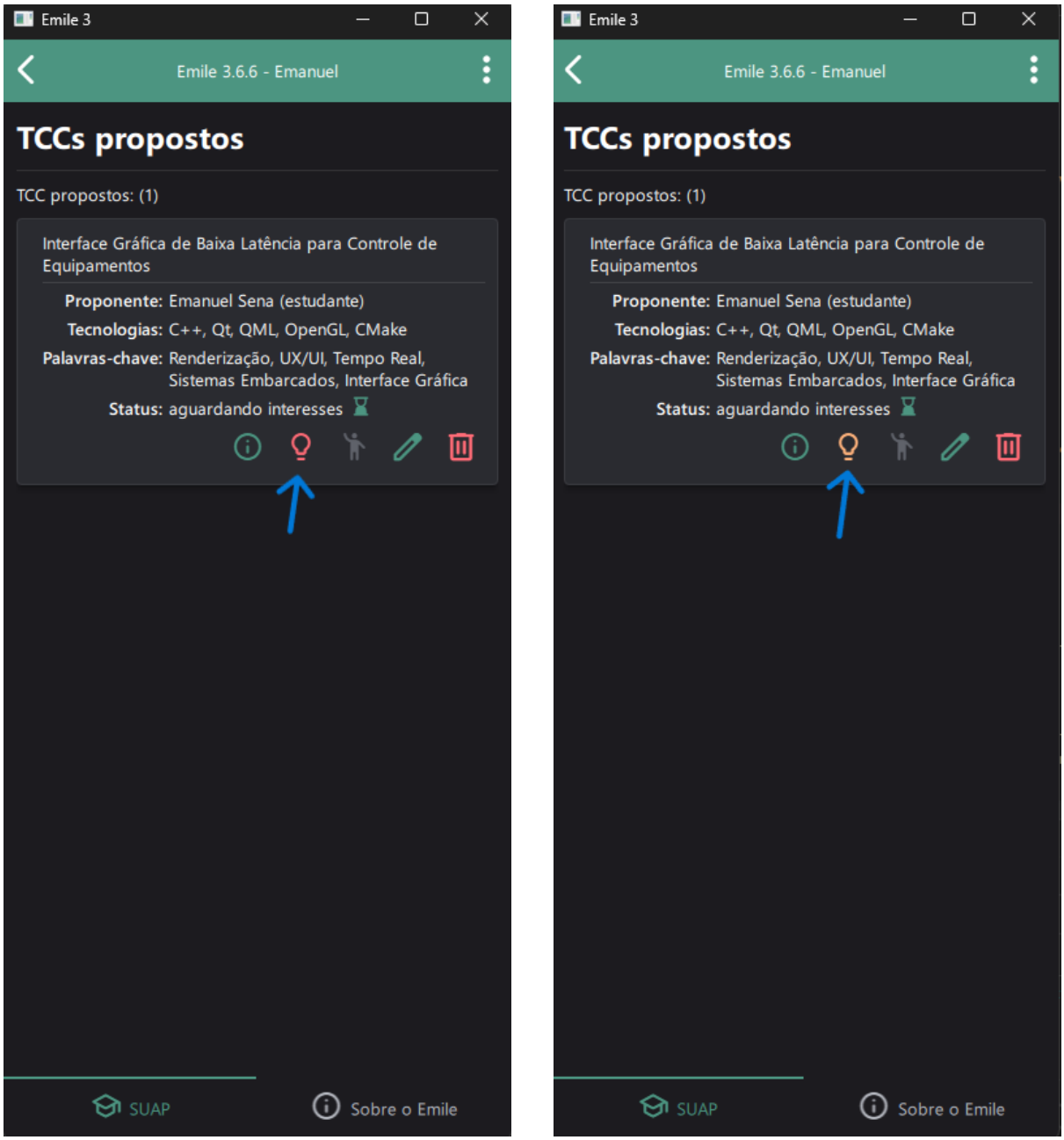
Figura 10: Telas de submissão e atualização de propostas de TCC no aplicativo Emile.

6.2 Acompanhamento e Visualização do *Matching*

Após a submissão, o aplicativo não bloqueia a navegação do estudante. O usuário pode acompanhar o status do processamento diretamente pelo **ícone de lâmpada** localizado no bloco da sua proposta. O sistema utiliza uma sinalização semântica de cores para indicar a disponibilidade da recomendação:

- **Lâmpada Vermelha:** Indica que o *matching* ainda não está disponível. O sistema está executando os cálculos de similaridade vetorial e a geração do matching em segundo plano.
- **Lâmpada Amarela:** Indica que o fluxo do RAG Avançado foi concluído com sucesso e a IA finalizou suas recomendações.

A Figura 11 apresenta o estado visual do ícone de lâmpada durante e após o processamento, contrastando os dois cenários.

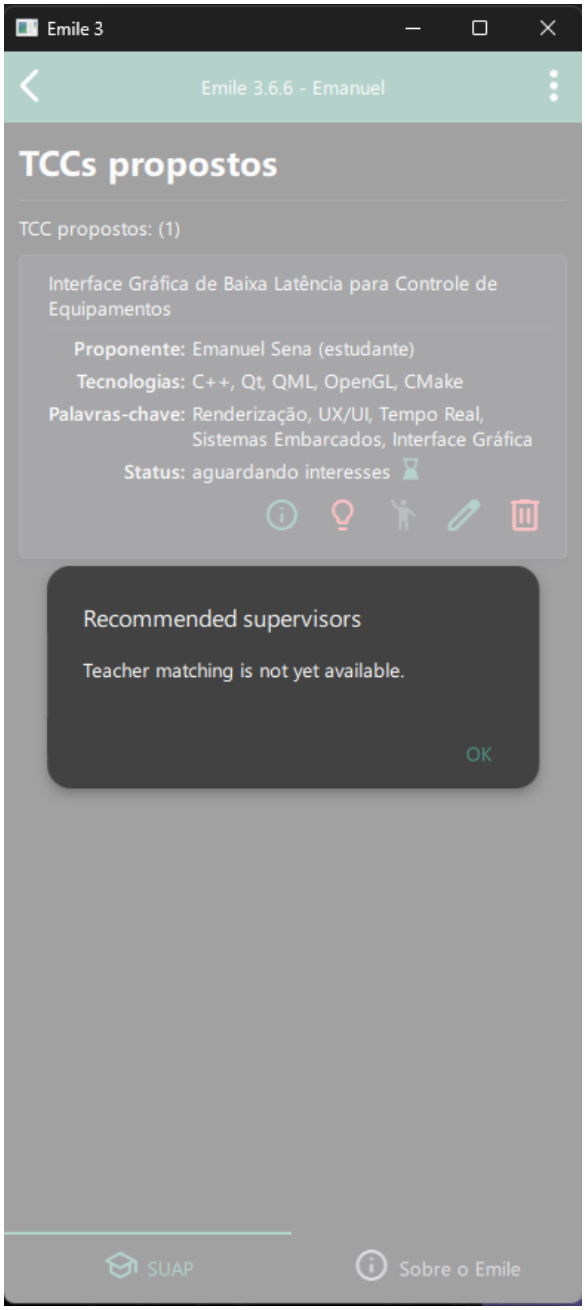


Lâmpada Vermelha: Em processamento

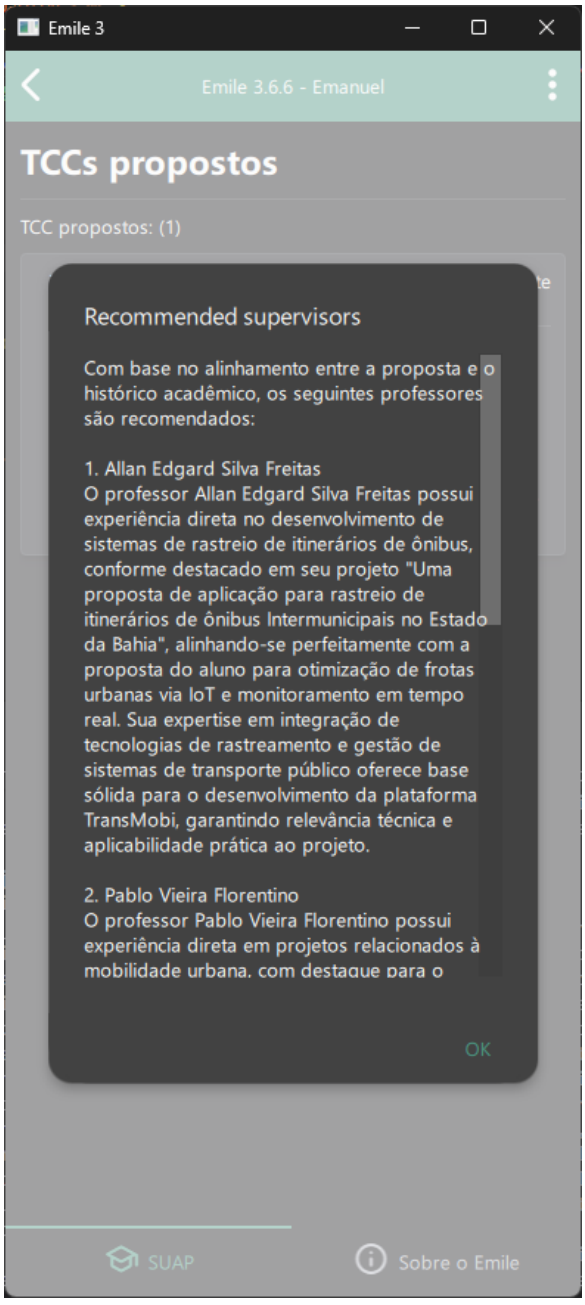
Lâmpada Amarela: Matching concluído

Figura 11: Indicadores visuais de status do pareamento no card da proposta.

A interação com o ícone produz resultados distintos dependendo do status atual do pareamento (Figura 12). Caso o estudante clique sobre a lâmpada enquanto ela estiver na coloração vermelha, o sistema exibirá um aviso informando que o pareamento ainda não está disponível. Por outro lado, ao clicar no ícone amarelo, o sistema exibirá a lista ranqueada dos docentes recomendados, acompanhada das justificativas técnicas formuladas detalhadamente pelo modelo generativo.



Interação com a Lâmpada Vermelha



Interação com a Lâmpada Amarela

Figura 12: Resultados da interação do usuário com os diferentes estados do ícone de lâmpada.

Referências

1 KIMOTHI, A. **A Simple Guide to Retrieval Augmented Generation**. [S.l.: s.n.], 2024. Referência fundamental para o entendimento das arquiteturas e fluxos do RAG.

2 ZHANG, Y.; CHEN, X. Explainable recommendation: A survey and new perspectives. **Foundations and Trends in Information Retrieval**, v. 14, n. 1, p. 1–101, 2020.

3 LEWIS, P. et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2020. v. 33.

4 JI, Z. et al. Survey of hallucination in natural language generation. **ACM Computing Surveys**, v. 55, n. 12, p. 1–38, 2023.

5 GAO, Y. et al. Retrieval-augmented generation for large language models: A survey. **arXiv preprint arXiv:2312.10997**, 2023.

6 REIMERS, N.; GUREVYCH, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In: **EMNLP-IJCNLP**. [S.l.: s.n.], 2019. p. 3982–3992.

7 WANG, L.; YANG, N.; WEI, F. Query2doc: Query expansion with large language models. In: **EMNLP**. [S.l.: s.n.], 2023. p. 9414–9423.

8 KARPUKHIN, V. et al. Dense passage retrieval for open-domain question answering. In: **EMNLP**. [S.l.: s.n.], 2020. p. 6769–6781.

9 NOGUEIRA, R. et al. Document ranking with a pretrained sequence-to-sequence model. In: **Findings of EMNLP**. [S.l.: s.n.], 2020. p. 708–718.

10 GAO, Y.; ILVES, K.; GLOWACKA, D. Officehours: A system for student supervisor matching through reinforcement learning. In: **IUI Companion**. [S.l.: s.n.], 2015. p. 29–32.

11 ZHANG, M. et al. A research analytics framework-supported recommendation approach for supervisor selection. **British Journal of Educational Technology**, v. 47, n. 2, p. 403–420, 2016.

12 RAHDARI, B.; BRUSILOVSKY, P.; SABET, A. J. Connecting students with research advisors through user-controlled recommendation. In: **ACM RecSys**. [S.l.: s.n.], 2021.

13 WIJANTO, M. C.; RACHMADIANY, R.; KARNALIM, O. Thesis supervisor recommendation with representative content and information retrieval. **Journal of Information Systems Engineering and Business Intelligence**, v. 6, n. 2, p. 143–150, 2020.

14 MANNING, C. D.; RAGHAVAN, P.; SCH”uTZE, H. **Introduction to Information Retrieval**. [S.l.]: Cambridge University Press, 2008.

15 RICCI, F.; ROKACH, L.; SHAPIRA, B. **Recommender Systems Handbook**. 3. ed. [S.l.]: Springer, 2022.

16 ES, S. et al. Ragas: Automated evaluation of retrieval augmented generation. In: **EACL System Demonstrations**. [S.l.: s.n.], 2024. p. 150–158.

Glossário, Siglas e Abreviações

API: *Application Programming Interface* - Interface de Programação de Aplicações. Conjunto de rotinas e padrões estabelecidos por um software para a utilização de suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação.

Backend: Camada de retaguarda de um software, invisível ao usuário final, responsável por processar a lógica de negócios, acessar bancos de dados e garantir a segurança das operações do sistema.

Chunk: Segmento ou fragmento. No contexto de processamento de linguagem natural e arquitetura RAG, refere-se a um bloco de texto extraído de um documento maior e estruturado para ser vetorizado.

CoT: *Chain-of-Thought* - Cadeia de Pensamentos. Técnica de engenharia de *prompt* que induz um modelo de linguagem a detalhar seu raciocínio passo a passo antes de chegar a uma resposta final.

CRUD: *Create, Read, Update, Delete* - Criar, Ler, Atualizar, Excluir. Acrônimo que representa as quatro operações básicas de persistência de dados em sistemas computacionais.

Embedding: Representação vetorial densa de palavras, frases ou documentos em um espaço contínuo, capturando o significado semântico e contextual da informação.

Endpoint: Ponto de extremidade em uma API. URL específica onde um serviço web pode ser acessado por uma aplicação cliente para executar uma função ou acessar recursos específicos.

hTTP: *hypertext Transfer Protocol* - Protocolo de Transferência de hipertexto. Protocolo base de comunicação da *World Wide Web*, utilizado para a transferência de dados e requisições entre clientes e servidores.

IA: Inteligência Artificial. Ramo da ciência da computação que se propõe a elaborar sistemas que simulem a capacidade humana de raciocinar, tomar decisões e resolver problemas.

JSON: *JavaScript Object Notation* - Notação de Objetos JavaScript. Formato leve e independente de linguagem utilizado para estruturar e trocar dados entre sistemas cliente-servidor.

Latência: Tempo de resposta. Refere-se ao atraso temporal medido (geralmente em milissegundos) entre o envio de uma requisição e a recepção completa da resposta correspondente por parte do sistema.

LLM: *Large Language Model* - Grande Modelo de Linguagem. Modelo de Inteligência Artificial treinado em grandes quantidades de texto para compreender e gerar linguagem natural.

Matching: Pareamento ou correspondência. No contexto deste trabalho, refere-se ao processo computacional de cruzar e encontrar afinidades técnicas entre as propostas dos alunos e as competências dos professores.

Payload: Carga útil. Refere-se ao conjunto de dados essenciais transmitidos em uma requisição ou resposta de rede, excluindo cabeçalhos e metadados de protocolo.

RAG: *Retrieval-Augmented Generation* - Geração Aumentada por Recuperação. Arquitetura que melhora as respostas de um LLM integrando-o a um sistema de busca em uma base de dados ou documentos externos.

Reranking: Reordenação. Etapa de pós-processamento em sistemas de busca onde uma lista inicial de documentos recuperados tem sua ordem matematicamente reavaliada e refinada sob novos critérios estruturais.

REST: *Representational State Transfer* - Transferência de Estado Representacional. Estilo arquitetural para sistemas distribuídos que define um conjunto de restrições para a criação de *Web Services*.

TCC: Trabalho de Conclusão de Curso. Avaliação final exigida em cursos de graduação no Brasil para a obtenção do diploma.

TF-IDF: *Term Frequency - Inverse Document Frequency* - Frequência do Termo - Inverso da Frequência nos Documentos. Medida estatística que indica a importância de uma palavra em um documento em relação a uma coleção de documentos.

Token: Unidade básica de processamento em modelos de linguagem. Pode representar uma palavra, parte de uma palavra ou até um único caractere, servindo como base para os cálculos internos da inteligência artificial.

UML: *Unified Modeling Language* - Linguagem de Modelagem Unificada. Linguagem visual padrão utilizada para especificação, documentação e construção de artefatos em engenharia de software.

Web Scraping: Raspagem de rede. Técnica de extração automatizada de dados que coleta informações diretamente do código estruturado de páginas da internet.