UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE GRADUAÇÃO EM CIÊNCIA
DA COMPUTAÇÃO

Cynara Valéria de Oliveira Costa do Amaral
Cavalcanti

Retrieval-Augmented Generation e LLM: Um Estudo de Caso em Aplicativos Mobile e Interfaces de Chatbot

Trabalho de Graduação

Recife
2025

Cynara Valéria de Oliveira Costa do Amaral Cavalcanti

Retrieval-Augmented Generation e LLM: Um Estudo de Caso em Aplicativos Mobile e Interfaces de Chatbot

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação

Orientador (a): Ricardo Prudêncio

Recife
2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Cavalcanti, Cynara Valéria de Oliveira Costa do Amaral.
Retrieval-Augmented Generation e LLM: Um Estudo de Caso em Aplicativos Mobile e Interfaces de Chatbot / Cynara Valéria de Oliveira Costa do Amaral Cavalcanti. - Recife, 2025.
35 p. : il.

Orientador(a): Ricardo Prudêncio
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2025.
9.
Inclui referências, apêndices, anexos.

1. RAG. 2. Mobile. 3. Server-Driven UI. 4. Chatbot. 5. Flutter. 6. LangChain. I. Prudêncio, Ricardo. (Orientação). II. Título.

000 CDD (22.ed.)

CYNARA VALÉRIA DE OLIVEIRA COSTA DO AMARAL CAVALCANTI

Retrieval-Augmented Generation e LLM: Um Estudo de Caso em Aplicativos Mobile e Interfaces de Chatbot

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Aprovado em: 26/03/2025

BANCA EXAMINADORA

_____

Profa. Dr. Ricardo Prudêncio (Orientador)

Universidade Federal de Pernambuco

_____

Prof. Dr. Filipe Calegário (Examinador Interno)

Universidade Federal de Pernambuco

Agradecimentos

Não poderia deixar de começar a agradecer a minha mãe, Cybelly, que desde a escolha do curso sempre me apoiou em todas as minhas decisões dentro da graduação. Sem a qual eu definitivamente não estaria aqui hoje, que sempre se fez presente desde marmitas a todo o seu investimento na minha formação acadêmica desde o colégio.

Gostaria de agradecer também ao meu pai, Euclides, que mesmo não percebendo, me ajudou bastante.

A minha avó, “vovi”, que sempre acreditou na neta favorita e sempre me encorajou na formação, também desde o colégio.

Aos meus amigos e familiares, que sempre estiveram dispostas a me ajudar e me ouvir quando eu mais precisava falar.

Aos meus amigos da graduação, aos quais dividimos todos os perrengues da vida universitária.

As oportunidades que encontrei e me encontraram no caminho, CITi, Apple Academy e XP, que sempre acreditaram no meu potencial e sem eles eu não iria conseguir finalizar este trabalho.

Ao meu namorado, Leonardo, que foi uma grande inspiração para esse trabalho, pessoa com a qual conto em todos os aspectos da minha vida, quem me apoia e acredita em mim.

A todos os professores que me guiaram até aqui, em especial Ricardo Prudêncio, meu orientador, que desde antes do início do trabalho já me apoiava e me direcionava para os melhores caminhos.

E por fim, agradeço a Cynara do passado, que sempre acreditou que esse caminho seria possível.

“The idea that technology alone can solve social and political problems is a dangerous delusion. But the idea that they can be solved without technology is also wrongheaded.”

— Mustafa Suleyman, The Coming Wave: Technology, Power, and the Twenty-first Century's Greatest Dilemma

Resumo

Com o avanço das tecnologias de Large Language Models (LLMs) e sua crescente adoção em diversos domínios, surge a necessidade de otimizar o acesso a informações em ambientes complexos e com uma quantidade de dados elevada, como o de prontuários eletrônicos médicos. A busca por informações precisas e rápidas é essencial para os profissionais de saúde, especialmente em cenários críticos e para situações em que o contato e atenção para com o paciente é muito importante. Diante desse contexto, a aplicação de interfaces server-driven e técnicas de Recuperação Aumentada por Geração (RAG) pode proporcionar uma experiência mais eficiente e intuitiva para os usuários.

Este trabalho propõe o desenvolvimento de um sistema server-driven para aplicações móveis, utilizando tecnologias modernas, como Flutter, Python, Flask, Gemini, MongoDB e LangChain, com foco em criar uma interface que possibilite a interação em um chat estilo ChatGPT. O objetivo é avaliar como essas soluções podem melhorar o fluxo de informações para os médicos, ao mesmo tempo em que se analisa a integração de dados não estruturados em sistemas de recomendação e recuperação.

Palavras-chaves: LLMs, Server-driven UI, RAG, Flutter, Prontuários Eletrônicos, Chatbot, MongoDB, Gemini, LangChain.

Abstract

With the advancement of Large Language Model (LLM) technologies and their increasing adoption across various domains, there arises a need to optimize access to information in complex environments with large amounts of data, such as medical electronic health records. The search for precise and quick information is essential for healthcare professionals, particularly in critical scenarios where patient interaction and attention are of utmost importance. In this context, the application of server-driven interfaces and Retrieval-Augmented Generation (RAG) techniques can provide users with a more efficient and intuitive experience.

This study proposes the development of a server-driven system for mobile applications using modern technologies such as Flutter, Python, Flask, Gemini, MongoDB, and LangChain. It focuses on creating an interface that enables interaction through a ChatGPT-style chat. The goal is to evaluate how these solutions can enhance the information flow for medical professionals while analyzing the integration of unstructured data into recommendation and retrieval systems.

Keywords: LLMs, Server-driven UI, RAG, Flutter, Electronic Health Records, Chatbot, MongoDB, Gemini, LangChain.

Lista de Figuras

Lista de Abreviaturas e Siglas

RAG	Retrieval-Augmented Generation
LLM	Large Language Model
UI	User Interface
REST	Representational State Transfer
MVP	Minimum Viable Product
KAG	Knowledge-Augmented Generation

Sumário

CAPÍTULO 1

Introdução

1.1 Descrição do Problema

Com o aumento exponencial do volume de dados e, consequentemente, a complexidade crescente dos prontuários eletrônicos, além de uma dificuldade maior de interação com os pacientes, os profissionais da saúde enfrentam desafios significativos ao acessar informações críticas de forma rápida e eficiente. Em muitos casos, a busca por dados relevantes torna-se demorada e desgastante, especialmente em situações emergenciais, onde o tempo é essencial. O modelo tradicional de navegação em prontuários não consegue atender adequadamente a necessidade de acessibilidade, contextualização e velocidade exigidas em ambientes médicos.

Essa lacuna pode resultar em perda de produtividade, aumento do estresse dos profissionais e, em casos extremos, impactar a qualidade do atendimento ao paciente.

1.2 Objetivo e Motivação

O objetivo deste trabalho é desenvolver uma solução mobile que facilite a consulta de informações em prontuários eletrônicos médicos por meio de uma interface interativa no estilo de chat. Utilizando tecnologias modernas, como Flutter, Python, Flask, Gemini, MongoDB e LangChain, o sistema será projetado para otimizar o acesso a dados de maneira rápida e intuitiva, especialmente em situações onde o tempo é crucial.

A motivação principal para este projeto é a necessidade de oferecer aos profissionais da saúde uma ferramenta prática e eficiente que reduza o esforço na busca por informações, permitindo que eles concentrem mais atenção no atendimento aos pacientes. Além disso, o uso de tecnologias como os LLMs apresenta um grande potencial para transformar a forma como dados complexos são acessados e interpretados, tornando o tema relevante e alinhado às demandas atuais.

1.3 Trabalhos Relacionados

Na literatura atual, há bastante discussão sobre a aplicabilidade de RAG em diversos contexto e tecnologias atuais, em Zhao [1] foram realizados vários estudos com LLMs

diferentes e sua capacidade de recomendação através do uso dessa tecnologia. Ainda no contexto de RAG, o estudo de Kulkarni [2] traz consigo a temática de chatbot, muito em alta no mercado atual.

Já referente à literatura sobre Server-Drive UI, temos Benis [3] especialmente no contexto mobile, comentando sobre as vantagens do uso da tecnologia.

1.4 Organização do Trabalho

Este trabalho está estruturado em seis capítulos, organizados de forma a facilitar o entendimento do problema, as soluções propostas e os resultados obtidos. No Capítulo 2, são abordados os conceitos básicos que fundamentam o trabalho, como Recuperação Aumentada por Geração (RAG), bancos de dados vetoriais, buscas vetoriais (Vector Search) e interfaces server-driven.

O Capítulo 3 descreve a metodologia adotada para o desenvolvimento do projeto, incluindo a seleção de frameworks, bibliotecas e tecnologias utilizadas. No Capítulo 4, é apresentada a arquitetura do sistema, com detalhes sobre as tecnologias escolhidas, a modelagem do sistema e as considerações feitas durante a construção da solução. Já o Capítulo 5 foca no fluxo de uso do aplicativo, explicando as funções gerais e as interações principais do usuário com a aplicação. Por fim, o Capítulo 6 apresenta a conclusão do trabalho, discutindo os resultados alcançados, as limitações encontradas e as oportunidades para trabalhos futuros.

CAPÍTULO 2

Conceitos Básicos

Neste capítulo serão definidos os conceitos necessários para o entendimento dos capítulos que se seguem. Na primeira seção serão discutidos conceitos relacionados ao RAG em si, contemplando conceitos como Vector Database e Vector Search. A segunda seção define o conceito e a prática do Server Driven UI.

2.1 RAG (Retrieval-Augmented Generation)

RAG (Retrieval-Augmented Generation)[4] é uma técnica que combina modelos de linguagem **(LLMs)** com mecanismos de recuperação de dados para gerar respostas mais precisas e contextualizadas. Ao contrário de modelos que dependem apenas do treinamento em dados pré-existentes, o RAG utiliza informações atualizadas e específicas recuperadas de bases de dados externas para enriquecer a geração de texto. Isso com o objetivo de gerar respostas da LLM com base em informações que ela não treinada com.

O funcionamento básico do RAG envolve:
Consulta à base de dados: Quando uma consulta é realizada, o sistema utiliza mecanismos de busca para localizar os dados mais relevantes em uma base externa, como documentos ou bancos de dados vetoriais.
Geração de resposta: Após recuperar as informações, o modelo de linguagem integra esses dados à resposta, gerando um conteúdo que combina conhecimento prévio com informações recuperadas em tempo real.

2.1.1 Embeddings

Embeddings [9] são representações numéricas em formato vetorial que traduzem dados complexos, como palavras, frases, imagens ou até mesmo documentos inteiros, em números que podem ser processados por algoritmos de aprendizado de máquina. A ideia principal é representar informações em um espaço de alta dimensão onde a proximidade entre vetores reflete a similaridade semântica entre os dados originais.

Como os embeddings funcionam:
Codificação Semântica: Modelos de linguagem, como BERT ou GPT, mapeiam palavras ou frases para vetores em um espaço matemático, de forma que itens semanticamente relacionados estejam mais próximos uns dos outros.

- Exemplo: As palavras “médico” e “hospital” terão vetores próximos, enquanto “médico” e “carro” estarão distantes.

Espaço Vetorial: Cada embedding é um ponto em um espaço multidimensional (geralmente com centenas ou milhares de dimensões). A distância entre esses pontos (como distância cosseno ou euclidiana) é usada para medir a similaridade entre os dados.

Importância dos embeddings no RAG:
Busca Semântica: Ao usar embeddings, o RAG pode encontrar informações relevantes mesmo que a consulta do usuário não coincida exatamente com os dados armazenados. Por exemplo, uma consulta com “clínica médica” pode recuperar documentos que falam sobre “hospitais”.

Redução de Dimensionalidade: Representar dados como vetores torna o processo de busca mais eficiente e escalável, especialmente em bancos de dados grandes.

Como embeddings são gerados:
Modelos Pré-Treinados: Modelos como BERT, GPT ou Sentence Transformers são comumente usados para gerar embeddings de texto.
Treinamento Personalizado: Empresas ou projetos específicos podem treinar modelos para gerar embeddings ajustados a seus domínios, como saúde, jurídico ou educação.

2.1.2 Vector Database

Um Vector Database é um tipo de banco de dados otimizado para armazenar e processar representações vetoriais de dados. Em um contexto de RAG, os dados (como textos ou imagens) são convertidos em vetores por meio de técnicas de aprendizado de máquina, permitindo uma busca eficiente com base em similaridades matemáticas.

Os vector databases são amplamente usados para:
Indexação semântica: Localizar conteúdos similares com base no significado, e não apenas em palavras-chave.
Integração com embeddings: Utilizar representações de textos e dados gerados por modelos como BERT ou OpenAI para buscas avançadas.

2.1.3 Vector Search

O Vector Search é o processo de consulta dentro de um Vector Database, onde são usadas métricas de distância (como a distância cosseno ou euclidiana) para encontrar os vetores mais semelhantes ao vetor de consulta. Isso permite:

Busca semântica: Retornar informações relevantes mesmo que as palavras exatas não coincidam.
Personalização: Aplicar filtros para melhorar os resultados com base no contexto ou nas preferências do usuário.

Essa técnica é essencial para melhorar a recuperação de informações em sistemas RAG, garantindo respostas rápidas e relevantes.

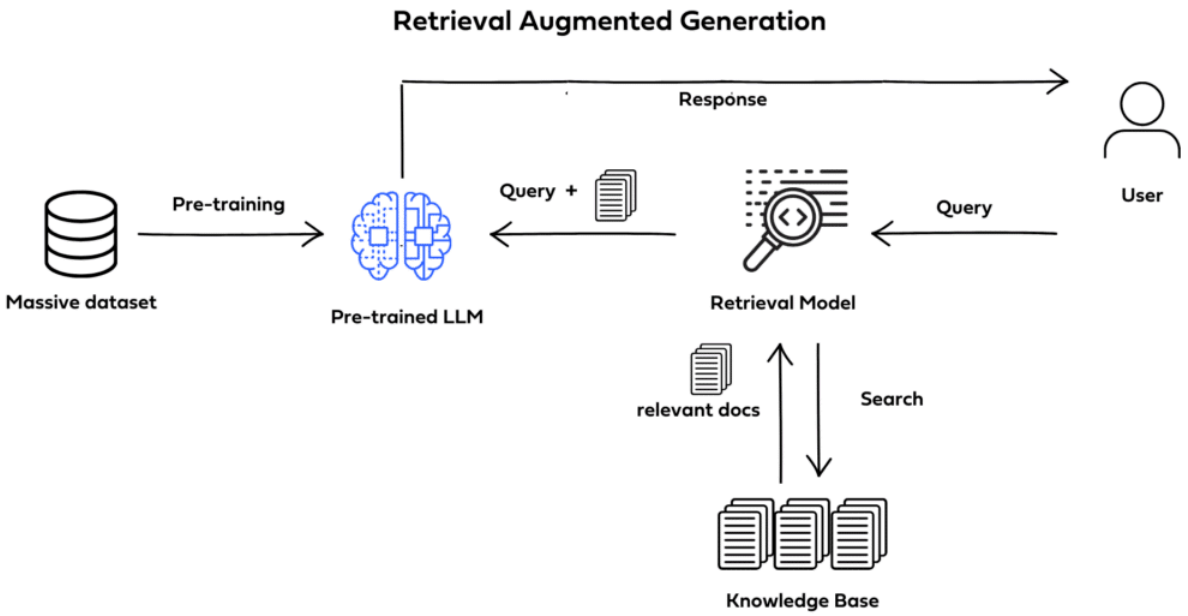

Figura 1: Arquitetura RAG (Fonte: Tarun Singh [4])

2.1.4 Funcionamento Geral

O funcionamento geral do RAG (Retrieval-Augmented Generation) pode ser dividido em etapas que integram as fases de recuperação e geração de informações, utilizando os conceitos de embeddings, Vector Database e Vector Search:

Pré-processamento dos Dados
Antes de iniciar o processo de recuperação, é necessário preparar os dados que estarão disponíveis no sistema. Isso envolve:

- Conversão dos dados em embeddings: Os documentos ou informações da base de conhecimento são convertidos em vetores por meio de modelos pré-treinados.
- Armazenamento no Vector Database: Os embeddings gerados são armazenados em um banco de dados vetorial, onde serão utilizados para consultas futuras.
- Organização e indexação: Os vetores são organizados de forma que a busca por similaridade seja otimizada, garantindo respostas rápidas e precisas.

Consulta do Usuário
Quando um usuário realiza uma consulta, como uma pergunta ou solicitação de informação:

- O texto da consulta é processado e convertido em um embedding por um modelo de linguagem pré-treinado.
- Esse embedding é usado como referência para buscar documentos relevantes no Vector Database.

Recuperação de Dados
A consulta do usuário é comparada aos embeddings armazenados no Vector Database usando técnicas de Vector Search, como:

- Distância Cosseno: Mede o ângulo entre os vetores para identificar similaridades semânticas.
- Distância Euclidiana: Mede a proximidade entre os vetores no espaço vetorial.

Os documentos ou dados mais relevantes são então recuperados com base nos resultados dessa comparação.

Geração da Resposta
Após a recuperação dos documentos relevantes:

- Os dados recuperados são fornecidos ao modelo de linguagem (LLM), que utiliza essas informações como contexto adicional para gerar uma resposta.
- O modelo combina seu conhecimento prévio com os dados recuperados, criando uma resposta mais precisa e contextualizada.

Apresentação ao Usuário
A resposta gerada pelo sistema é retornada ao usuário, contendo:

- Informações atualizadas e específicas retiradas da base de conhecimento.
- Texto fluido e contextualizado, semelhante ao que seria gerado em uma conversa humana.

Exemplo Prático

- **Consulta:** Um médico pergunta: “Qual o tratamento mais indicado para pacientes com diabetes tipo 2?”
- **Recuperação:** O sistema consulta os dados em um Vector Database que contém artigos médicos e diretrizes, encontrando documentos relacionados ao tratamento da diabetes.
- **Geração:** O modelo de linguagem utiliza esses documentos para gerar uma resposta detalhada, citando tratamentos, medicamentos e considerações importantes.
- **Resposta:** O usuário recebe uma resposta clara e contextualizada baseada nos dados recuperados.

O RAG funciona como uma ponte entre bases de dados específicas e modelos de linguagem, unindo a precisão de dados estruturados com a fluidez e adaptabilidade de modelos pré-treinados. Esse fluxo garante que os usuários obtenham respostas relevantes e úteis, mesmo em consultas complexas ou com informações desatualizadas no modelo.

2.1.5 Otimização da Consulta

A forma como uma pergunta de entrada é tratada pode impactar significativamente a qualidade das respostas geradas. Existem diversas técnicas [10] para aprimorar esse processo, como as elencadas a seguir:

Expansão de Consulta (Query Expansion)

Para melhorar a recuperação de informações, a consulta do usuário pode ser reformulada ou expandida antes da busca. Algumas abordagens incluem:

- **Sub-Queries:** Perguntas complexas podem ser divididas em subperguntas menores e mais específicas.
- **Reescrita de Pergunta (Query Rewrite):** Reformular a consulta de forma mais clara e estruturada para melhorar a correspondência semântica com os documentos armazenados.
- **Geração de Pergunta Hipotética (HyDE - Hypothetical Document Embedding):** Criar uma resposta hipotética baseada na pergunta do usuário e buscar documentos que sejam semanticamente próximos a essa resposta.

Roteamento de Consulta (Query Routing)

Dependendo da natureza da pergunta, diferentes fontes de dados podem ser utilizadas. Isso inclui roteamento para bases de conhecimento estruturadas (como Knowledge Graphs) ou bases de documentos não estruturados.

2.1.6 Estratégias de Geração de Respostas

Após a recuperação dos documentos, diferentes técnicas podem ser usadas para garantir que a resposta seja relevante e precisa:

Filtragem e Reclassificação (Re-ranking)

Nem todos os documentos recuperados são igualmente úteis. Métodos de re-ranking podem melhorar a relevância dos documentos, removendo aqueles irrelevantes ou redundantes antes de passá-los ao modelo de geração.

Compactação do Contexto

Muitas vezes, o volume de informação recuperado é grande demais para ser processado de forma eficiente. Para otimizar isso:

- **Condensação da Informação:** Resumir os documentos mais relevantes antes de enviá-los ao modelo de linguagem.
- **Filtragem Automática:** O próprio modelo pode decidir quais trechos dos documentos são essenciais para a resposta.

Recuperação Iterativa (Iterative Retrieval)

Em vez de realizar uma única busca e gerar uma resposta, o RAG pode refazer consultas adicionais enquanto gera o texto, refinando continuamente a resposta.

Modular RAG

Em arquiteturas mais avançadas, diferentes módulos podem ser incorporados para aprimorar o processo de busca dos vetores a partir dos embeddings, como:

- **Generate-Read:** O modelo gera um esboço da resposta antes de buscar informações para validar ou complementar seu conteúdo.
- **Self-RAG:** O próprio modelo decide se precisa buscar mais informações antes de gerar a resposta final.

2.1.7 Avaliação do RAG

Para medir a eficácia do RAG, diversas métricas podem ser utilizadas:

- **Precisão da Recuperação:** Mede se os documentos mais relevantes foram selecionados corretamente (métricas como Recall@K e NDCG).
- **Fidelidade da Resposta:** Avalia se o modelo gerou uma resposta baseada apenas nos

documentos recuperados, evitando alucinações.
- **Rejeição de Perguntas Negativas:** Mede a capacidade do modelo de identificar e não responder perguntas para as quais não há resposta na base de dados.

A combinação dessas técnicas e métricas permite aprimorar continuamente os sistemas RAG, garantindo respostas mais precisas, contextualizadas e confiáveis.

2.2 Server-Driven UI

A Server-Driven UI é uma abordagem arquitetural que transfere a responsabilidade pela definição e controle da interface do usuário (UI) do lado do cliente para o lado do servidor. Em vez de o aplicativo definir localmente a estrutura, o layout e os componentes visuais da interface, o servidor envia todas as informações necessárias para que o cliente construa e renderize a interface de maneira dinâmica. Geralmente todas essas informações advindas do backend, estão no formato json, o que facilita a comunicação entre o back e o front, já que através das requisições REST é possível e relativamente fácil a comunicação entre as duas interfaces.

Essa abordagem é especialmente útil em cenários onde é necessária maior flexibilidade na atualização da interface ou personalização para diferentes contextos e usuários, reduzindo significativamente a necessidade de atualizações frequentes do aplicativo.

O fluxo típico de uma Server-Driven UI pode ser descrito em etapas:
- **Solicitação do Cliente:** O cliente (aplicativo móvel ou web) faz uma solicitação ao servidor para obter as informações da interface.
- **Definição pelo Servidor:** O servidor processa a solicitação e retorna uma estrutura detalhada contendo:
    - O layout da interface (organização dos componentes).
    - Os tipos de componentes (botões, listas, textos, etc.).
    - Estilos e comportamentos (cores, tamanhos, interações).
- **Renderização no Cliente:** O cliente interpreta a estrutura recebida e renderiza a interface para o usuário.

Essa abordagem permite que a interface seja alterada pelo servidor em tempo real, sem a necessidade de atualizar ou recompilar o aplicativo. De forma que conseguimos lidar com possíveis imprevistos de forma mais fácil, sem ter que gerar uma nova compilação do app para as lojas, por exemplo, além de ter uma fácil manutenção.

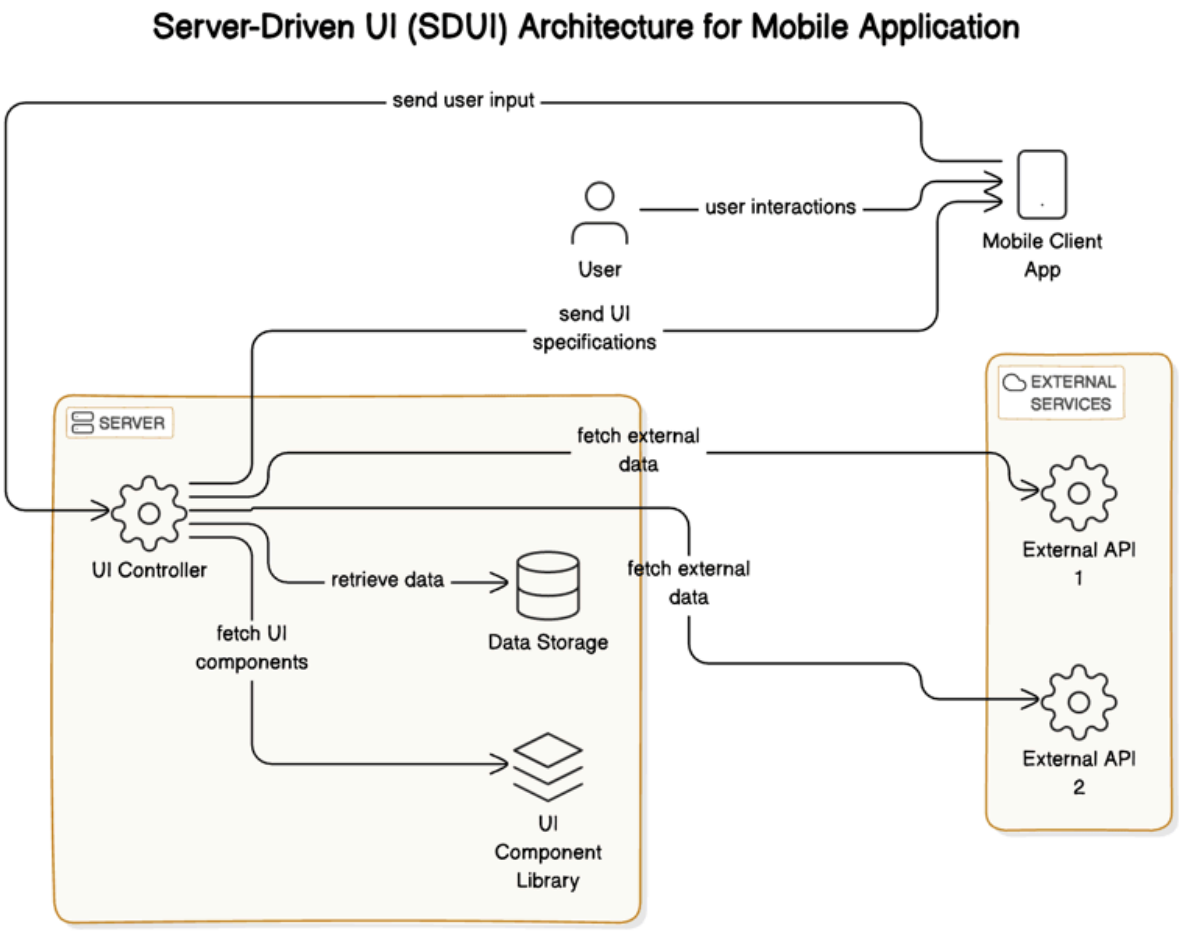

Figura 2: Arquitetura Server-Drive-UI (Fonte: Pooja Raj [5]

CAPÍTULO 3

Arquitetura

Este capítulo apresenta a arquitetura do sistema desenvolvido para este trabalho, descrevendo os componentes e a interação entre eles, bem como a modelagem completa do fluxo de dados e funcionalidades, assim como as tecnologias escolhidas e suas motivações.

A proposta baseia-se em um sistema dinâmico que utiliza técnicas de Recuperação Aumentada por Geração (RAG), integrando um backend, um banco de dados vetorial e um modelo de linguagem de grande escala (LLM) para oferecer respostas otimizadas e contextualizadas aos usuários.

3.1 Arquitetura/Modelagem do sistema completo

A arquitetura do sistema pode ser dividida em diversos componentes que trabalham em conjunto para processar as requisições dos usuários, buscar dados relevantes e gerar respostas enriquecidas. O diagrama apresentado ilustra o fluxo completo do sistema:

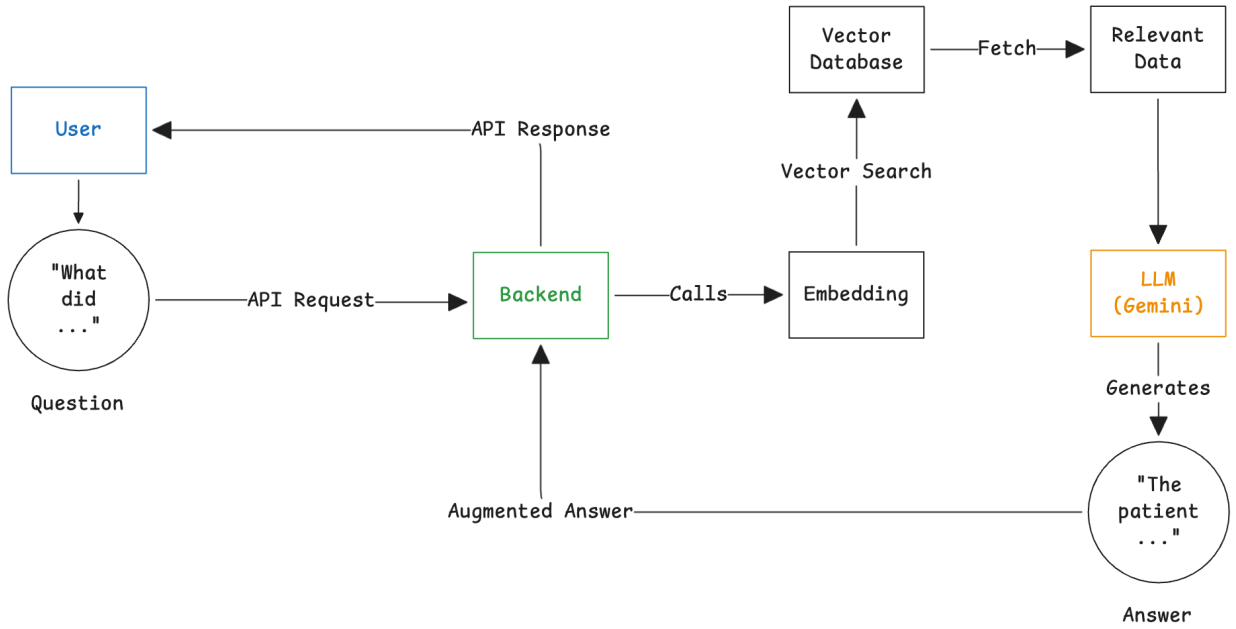

Figura 3: Arquitetura Do Sistema (Fonte: Elaborada pela autora)

Componentes Principais

- **Usuário:**
  - O ponto de entrada do sistema, onde o usuário faz perguntas ou solicita informações via a interface do aplicativo móvel.
  - Essas perguntas são enviadas como uma requisição de API para o backend.

- **Backend:**
  - Serve como intermediário entre o cliente e os outros componentes do sistema.

    - Funções principais:
        - Receber as requisições do cliente.
        - Processar as perguntas, chamando o componente de embeddings para criar uma representação vetorial da consulta.
        - Orquestrar a comunicação com o banco de dados vetorial e o modelo de linguagem.

- **Embedding:**
    - Transforma a consulta textual do usuário em uma representação vetorial.
    - Essa representação é usada para buscar similaridades no banco de dados vetorial.
    - No caso deste projeto, estamos utilizando o **all-MiniLM-L6-v2** do Hugging Face.

- **Vector Database:**
    - Um banco de dados otimizado para armazenar e processar embeddings, que no caso do MongoDB utilizamos um Vetor Search Index para direcionar onde ele irá fazer essa comparação nas "colunas" do banco de dados.
    - Funções principais:
        - Busca Vetorial (Vector Search): Identifica os dados relevantes com base na similaridade entre os vetores armazenados e a consulta do usuário.
        - Retorna as informações mais relevantes ao backend.

- **LLM (Gemini):**
    - Após os dados relevantes serem recuperados, o LLM é chamado para processar esses dados e gerar uma resposta contextualizada.
    - Combina o conhecimento pré-treinado do modelo com as informações recuperadas do banco de dados vetorial para criar uma resposta precisa e enriquecida.

- **Resposta ao Usuário:**
    - O backend recebe a resposta gerada pelo LLM e a formata para ser exibida na interface do usuário.
    - A resposta, a nível de **MVP**, é apenas textual representando o retorno do backend.

3.2 Seleção de frameworks/bibliotecas

3.2.1 Flutter

O Flutter foi escolhido como framework principal para o desenvolvimento do aplicativo móvel devido à sua capacidade de criar interfaces nativas para iOS e Android com uma base de código única. Essa característica reduz o esforço de desenvolvimento e manutenção, permitindo maior agilidade na entrega do sistema. Além disso, o Flutter oferece uma performance próxima à de aplicativos nativos, sendo ideal para aplicações que exigem alta responsividade.

3.2.2 Python e Flask

O Python foi selecionado pela sua versatilidade e ampla utilização no desenvolvimento de

aplicações backend e em soluções baseadas em inteligência artificial. O Flask, um framework minimalista e eficiente, foi escolhido para construir as APIs RESTful responsáveis por conectar o cliente ao servidor. A simplicidade do Flask facilita a integração com bibliotecas avançadas, como LangChain e Gemini, essenciais para o funcionamento do sistema, de forma que seria mais intuitivo utilizar Flask, por ser um framework do Python, para centralizar o uso de funções em um mesmo backend. No cenário de uma aplicação de maior escalabilidade, isso seria uma abordagem ruim, mas em cenários de testes, é uma boa escolha.

3.2.3 LangChain

O LangChain é uma biblioteca especializada em pipelines que combinam modelos de linguagem com técnicas de recuperação de informações. Sua flexibilidade permite a implementação eficiente da Recuperação Aumentada por Geração (RAG), possibilitando que dados contextuais sejam integrados às respostas geradas. O LangChain também suporta diversas integrações com bancos de dados vetoriais, tornando-o uma escolha estratégica para o projeto. No caso deste projeto, a escolha foi óbvia devido a integração com o MongoDB e com o Gemini.

3.2.4 Gemini

O Gemini é uma ferramenta poderosa para gerenciar e interagir com modelos de linguagem de grande escala (LLMs). Ele foi escolhido pela sua eficiência em processar consultas e gerar embeddings, elementos centrais para a busca semântica no sistema. Além disso, o Gemini é altamente compatível com as tecnologias selecionadas, facilitando a implementação da RAG. A motivação para a não utilização do ChatGPT pela OpenAI, foi puramente por logística. Hoje sabemos que a OpenAI tem os melhores LLMs do mercado, além de excelentes gerações de embeddings, porém todas as requisições custam alguns dólares, coisa que ao longo da construção do projeto iria gerar um custo considerável.

3.1.5 MongoDB

O MongoDB foi escolhido como banco de dados para o sistema devido à sua flexibilidade no armazenamento de dados não estruturados e sua compatibilidade com a gestão de embeddings. Sua estrutura orientada a documentos (JSON) facilita a manipulação e recuperação de dados complexos, sendo ideal para armazenar as representações vetoriais necessárias para o funcionamento do sistema. Além de que, pelo fato de estarmos utilizando a estratégia de Server-Drive UI, a escolha de um banco de dados NoSQL foi a melhor, devido a forma de manipulação dos dados ser facilitada pelo uso do JSON.

3.2.6 Hugging Face

O modelo Hugging Face **all-MiniLM-L6-v2** foi selecionado para a geração de embeddings, sendo uma escolha estratégica por sua leveza e alta eficiência. Esse modelo é amplamente utilizado para transformar textos em representações vetoriais, permitindo realizar buscas semânticas de forma precisa e rápida.
O **all-MiniLM-L6-v2** é otimizado para aplicações em tempo real, oferecendo uma boa relação entre desempenho e precisão. Por ser um modelo compacto, ele é ideal para cenários em que a performance é um requisito crítico, como no processamento de consultas de usuários. Além de ser open source é bastante utilizado no âmbito de RAG no mercado hoje em dia.

CAPÍTULO 4

Desenvolvimento

Este capítulo detalha as etapas da construção do sistema de recomendação de filmes, desde a concepção da ideia até a implementação do modelo RAG com LLMs. A abordagem adotada foi iterativa, permitindo ajustes ao longo do desenvolvimento.

4.1 Concepção da ideia

A concepção do projeto teve início em 2024, durante uma consulta médica. Durante a conversa, o médico mencionou que utilizava frequentemente o ChatGPT para auxiliar em diagnósticos e tratamentos, validando informações e otimizando sua tomada de decisão. Essa observação despertou meu interesse em explorar uma solução que não apenas aprimorasse o suporte técnico na medicina, mas também melhorasse a experiência humana no atendimento. O objetivo seria facilitar o acesso a informações do paciente de forma ágil, permitindo que médicos e pacientes tivessem interações mais qualificadas, reduzindo o tempo gasto na busca por dados.

Embora a implementação do RAG tenha sido planejada para as etapas finais do desenvolvimento, essa foi a primeira questão que explorei. Isso porque era essencial validar a viabilidade técnica da solução, considerando a integração entre um aplicativo mobile e uma arquitetura baseada em frameworks Python.

Além dos desafios arquiteturais, minha compreensão sobre o funcionamento do RAG ainda era limitada. Existia a noção do potencial da tecnologia, mas não do processo necessário para implementá-la. Assim, antes de formalizar a proposta, investi tempo pesquisando sobre os princípios do RAG e suas aplicações, levando em consideração minhas restrições de tempo e orçamento. Muitos bancos de dados vetorizados disponíveis no mercado são pagos, assim como diversas embedding functions de alta qualidade, que geralmente não são open-source.

De maneira geral, a concepção do projeto foi muito mais do que apenas idealizar e iniciar a prototipação. Foi necessário validar a viabilidade técnica e prática da solução antes mesmo de estruturar a proposta formalmente.

4.2 Prototipação

Assim como em outros projetos, considero essencial mapear os fluxos da aplicação antes de iniciar o desenvolvimento. Para isso, utilizo o Figma na criação de protótipos, permitindo uma visualização clara da solução antes da implementação. A figura abaixo ilustra esse processo:

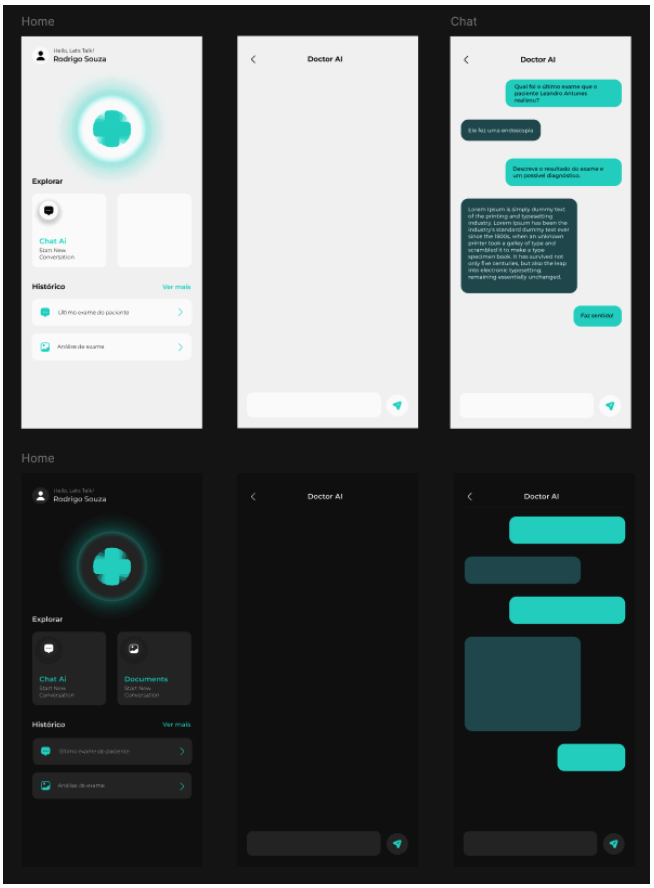

Figura 4: Protótipo (Fonte: Elaborada pela autora)

4.3 Desenvolvimento Mobile

Com a finalização do protótipo, o próximo passo natural foi iniciar o desenvolvimento do código mobile. Nesta etapa, com a ideia e a arquitetura do projeto bem definidas, ficou claro que a solução exigiria o uso de uma REST API para gerenciar as chamadas entre o aplicativo e o sistema RAG.

Para estruturar a interface do aplicativo de maneira eficiente, desenvolvi um Design Pattern que define a construção da parte visual com base nas respostas do backend, responsável pela comunicação com o RAG.

Esse padrão segue uma abordagem semelhante ao **Chain of Responsability** [6], onde cada componente lida autonomamente com sua requisição e mantém um contrato definido com o backend. O aplicativo mobile recebe um retorno em JSON, contendo parâmetros

pré-estabelecidos, como um campo "title", utilizado para a construção dinâmica dos componentes na interface.

Com essa abordagem, a implementação se tornou mais modular e escalável. Cada componente possui um handler específico, garantindo que toda a estrutura do projeto siga uma lógica consistente, bastando apenas desenvolver novos handlers conforme necessário para diferentes elementos da interface. Para maior conhecimento sobre o desenvolvimento do código da parte do mobile, basta acessar o Repositório Mobile [7].

4.4 Desenvolvimento Backend

O backend foi projetado para oferecer uma API REST estruturada, seguindo uma abordagem modular para facilitar a manutenção e escalabilidade do sistema. As principais funcionalidades implementadas incluem:

- Gerenciamento de Conversas: criação e recuperação de conversas do usuário.
- Histórico de Mensagens: armazenamento e recuperação de interações passadas.
- RAG e Respostas do LLM: integração com o modelo para geração de recomendações baseadas na consulta do usuário.
- Interface Server-Driven UI: resposta dinâmica para a construção da interface no frontend.

Falando um pouco mais sobre os endpoints e a integração com as funcionalidades citadas acima:

Rota Inicial (/)
Responsável por verificar se o servidor está ativo, retornando uma resposta simples.

Widget de Exploração e Histórico (/v1/home)
Retorna a estrutura JSON dos widgets da interface, utilizados pelo frontend para exibição dinâmica de informações.

- Os widgets incluem:
    - Explorer Carousel: sugestão de funcionalidades para o usuário.
    - History Cards: exibição do histórico de conversas do usuário.

Recuperação de Conversas (/v1/get-conversation)
Permite recuperar mensagens de uma conversa específica com base no conversationId.
As mensagens são formatadas para diferenciar se foram enviadas pelo usuário ou pelo chatbot. Reconhecendo através do parâmetro “isUser”.

Envio de Perguntas e Geração de Respostas (/v1/send-question)
Endpoint que recebe a mensagem do usuário, armazena no histórico e processa a resposta via RAG.
Fluxo da requisição:

1. O usuário envia uma pergunta.
2. A API identifica a conversa correspondente ou cria uma nova.
3. A pergunta é armazenada no MongoDB.
4. O modelo RAG gera uma resposta.
5. A resposta do chatbot é salva na conversa.
6. A resposta gerada é retornada ao usuário.

O uso do MongoDB permite o armazenamento das conversas, garantindo que o histórico seja acessível para futuras interações. Além de a própria função do vector search também ser feita no MongoDB.

Para um entendimento mais geral do funcionamento com o RAG, a imagem abaixo representa a rota que a aplicação chama ao enviar uma mensagem ao chatbot. Para maior conhecimento sobre o desenvolvimento do código da parte do backend, basta acessar o Repositório Server [8].

4.5 Implementação RAG

O RAG é basicamente o componente essencial da solução, pois permite que o sistema recupere informações relevantes sobre os pacientes a partir do vector database MongoDB e utilize o Gemini para gerar respostas contextualizadas. A implementação do RAG combina diversas técnicas para melhorar a precisão das respostas e garantir que apenas informações verificadas sejam utilizadas.

A seguir, detalhamos os principais componentes da implementação do RAG no backend.

4.5.1 Carregamento e Armazenamento dos Dados

A primeira etapa da implementação do RAG é o **carregamento dos dados dos pacientes**. O código responsável por essa funcionalidade está na função load_data() como mostra na figura 7 logo abaixo, que percorre arquivos JSON armazenados no diretório /patients_mocks (que consta com pacientes fictícios a fim de povoar o banco de dados), processa as informações e as converte em **embeddings** para inserção no **MongoDB**.

Processamento dos arquivos JSON:

- O sistema lê os arquivos que contêm informações sobre os pacientes, como histórico médico, consultas, vacinas, etc.
- Cada documento é estruturado para garantir que os dados sejam armazenados corretamente.

Geração de embeddings:

- As informações de cada paciente são convertidas em um único **vetor de representação** usando o modelo **GoogleGenerativeAIEmbeddings**.
- Esse embedding é armazenado na coleção **patients** do MongoDB, permitindo que as consultas futuras sejam feitas por **similaridade semântica**.

Essa estrutura garante que as consultas possam buscar informações relevantes sem depender de palavras exatas, tornando a recuperação de dados mais eficiente.

4.5.2 Consulta Otimizada e Recuperação de Dados

A recuperação de informações ocorre na função query_data(). Essa função é responsável por:

Expansão da Consulta (Query Expansion)

- Antes de buscar informações, o sistema reescreve a consulta para otimizar a busca, garantindo que a pesquisa seja feita da maneira mais eficaz possível.
- A função rewrite_prompt() utiliza um LLM para reformular a pergunta, removendo

ambiguidades e melhorando a precisão da recuperação. Isso evita erros causados por perguntas mal formuladas ou muito genéricas.

Busca Vetorial no MongoDB

- O MongoDB é utilizado como um banco de dados vetorial, permitindo buscas semânticas usando a funcionalidade **MongoDBAtlasVectorSearch**.
- A consulta é transformada em um embedding e comparada com os documentos armazenados, retornando os mais relevantes.

Re-rank de Documentos

- Após recuperar os documentos relevantes, a função re_rank_documents() reordena os resultados de acordo com sua relevância. Isso garante que as informações mais úteis apareçam primeiro.

Iteração na Recuperação (Iterative Retrieval)

- Se o modelo LLM indicar que não há informações suficientes para responder à pergunta, o sistema realiza uma segunda rodada de busca no banco de dados.
- A pesquisa é expandida para incluir mais documentos, melhorando a precisão das respostas.

4.5.3 Geração de Respostas pelo LLM

Uma vez que os documentos relevantes são recuperados, o LLM é utilizado para gerar uma resposta baseada apenas nos dados disponíveis. Para isso, um prompt estruturado foi criado para garantir que a resposta siga diretrizes médicas, como:

- Utilizar **apenas informações verificadas**.
- **Não fazer suposições** caso os dados não estejam disponíveis.
- **Responder no mesmo idioma da pergunta**.
- **Manter um tom clínico e profissional**.

A estrutura do prompt é definida na função ChatPromptTemplate.from_messages() e garante que o LLM siga essas diretrizes ao gerar a resposta.

4.5.4 Avaliação do RAG

Para medir a **eficácia do sistema RAG**, foram implementadas métricas de avaliação utilizando a função evaluate_queries(). O objetivo é verificar se os documentos retornados pelo MongoDB realmente contêm as informações solicitadas pelo usuário.

Recall@K

- Mede a proporção de documentos relevantes que estão entre os **K primeiros resultados** retornados pelo RAG.

- Se a consulta pede informações sobre um paciente e o sistema recupera corretamente seus dados entre os primeiros K resultados, o Recall@K será alto.

NDCG@K (Normalized Discounted Cumulative Gain)

- Avalia a **ordenação** dos resultados retornados, garantindo que documentos mais relevantes sejam priorizados.
- Isso é importante para garantir que o modelo não apenas retorne documentos corretos, mas também os apresente na ordem mais útil.

A avaliação é feita chamando evaluate_queries(), que testa múltiplas perguntas e mede a qualidade da recuperação de informações.

Porém, um ponto em relação ao RAG evaluation, é que pelo fato de não envolver retorno de documentos de um paciente apenas, ou seja, termos um total de documentos que é diferente do total de pacientes, os resultados do Recall@ e do NDCG@ são sempre iguais. Isso pelo fato de que na chamada para o RAG, ele sempre está retornando o paciente esperado, em pelo menos 3 resultados da busca.

Para maior conhecimento sobre o desenvolvimento do código da parte do RAG, basta acessar o Repositório Server [8].

4.6 Resultado e Interface Desenvolvida

A seguir são apresentados exemplos visuais da interface desenvolvida, destacando a experiência do usuário e a interação com o chatbot através da aplicação móvel. As figuras ilustram o fluxo principal, desde o envio da consulta até a exibição da resposta gerada pelo sistema.

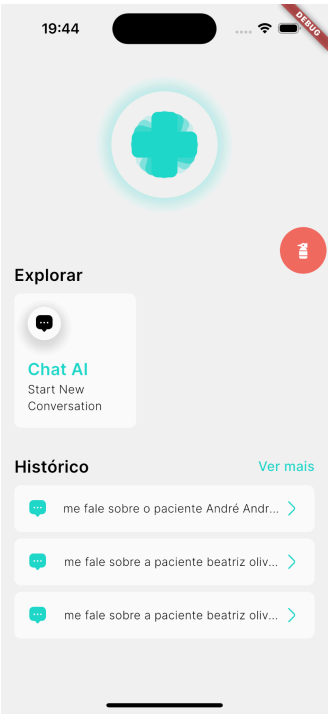

Figura 5: Interface inicial com sugestões de uso e histórico de consultas.
(Fonte: Elaborada pela autora)

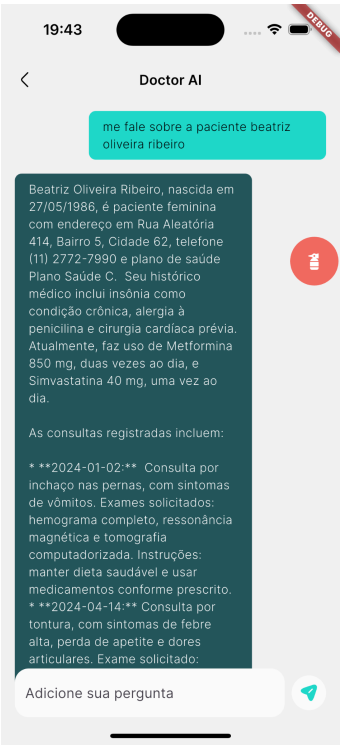

Figura 6: Exemplo de interação extensa com o chatbot, ilustrando uma consulta realizada pelo usuário. (Fonte: Elaborada pela autora)

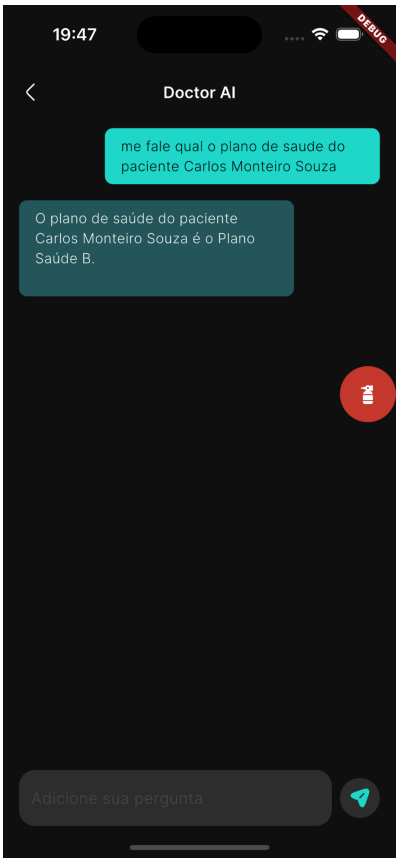

Figura 7: Exemplo de interação curta com o chatbot, ilustrando uma consulta realizada pelo usuário. (Fonte: Elaborada pela autora)

CAPÍTULO 5

Conclusão e Oportunidades

A implementação de uma solução baseada em RAG combinada com LLMs e interfaces server-driven mostrou-se promissora para otimizar o acesso a informações em ambientes críticos, especialmente na área de saúde, onde a eficiência e precisão são cruciais. O desenvolvimento do sistema apresentado neste trabalho comprovou a viabilidade técnica e destacou vantagens significativas, tais como maior rapidez no acesso às informações e uma experiência mais intuitiva e contextualizada para os usuários.

Durante o desenvolvimento do projeto, foi possível validar a integração eficaz entre tecnologias modernas, como Flutter, Python, Flask, Gemini, MongoDB e LangChain. Além disso, o uso de embeddings e Vector Database permitiu uma recuperação de dados precisa e rápida, otimizando significativamente as consultas semânticas.

Entretanto, apesar dos resultados positivos, algumas limitações foram identificadas. Uma delas foi a ausência de uma análise mais profunda sobre a escalabilidade e desempenho do sistema em cenários com alta concorrência e grande volume de requisições simultâneas. Embora o estudo tenha validado a viabilidade técnica da integração das tecnologias, não foi explorado detalhadamente como o sistema se comportaria sob carga real, especialmente considerando múltiplos usuários acessando o serviço simultaneamente.

Como oportunidades para trabalhos futuros, destaca-se:

- Explorar técnicas avançadas de otimização e expansão de consultas para melhorar ainda mais a precisão das respostas geradas;
- Avaliar o desempenho do sistema em ambientes de produção com bases de dados maiores e em tempo real;
- Investigar a aplicação do sistema em outros contextos médicos específicos ou em áreas diferentes, validando sua generalização e escalabilidade;
- Integrar mecanismos adicionais de segurança e privacidade, especialmente relevantes devido à natureza sensível dos dados médicos manipulados;
- Realizar estudos com usuários reais para analisar o impacto na produtividade e satisfação dos profissionais da saúde.
- Sistema de salvar as respostas geradas e perguntas feitas pelo usuário para manter a comunicação dentro de uma mesma conversa.
- Adicionar a possibilidade de leitura de documentos em uma conversa.

Uma outra oportunidade futura relevante seria explorar técnicas mais avançadas como o **Knowledge-Augmented Generation (KAG)**, que utiliza grafos de conhecimento estruturados para melhorar ainda mais a precisão e a relevância das respostas geradas pelos modelos, especialmente em contextos complexos como o da saúde.

Este trabalho demonstrou o potencial significativo das tecnologias de RAG e LLMs aplicadas ao contexto da saúde, oferecendo um caminho sólido para futuras pesquisas e aplicações práticas que possam continuar aprimorando o atendimento médico por meio da inovação tecnológica.

Referências

[1] Zihuai Zhao, Wenqi Fan, Jiatong Li, Yunqing Liu, Xiaowei Mei, Yiqi Wang, Zhen Wen, Fei Wang, Xiangyu Zhao, Jiliang Tang, and Qing Li. Recommender Systems in the Era of Large Language Models (LLMs). https://arxiv.org/pdf/2307.02046. 29 Abr 2024

[2] Mandar Kulkarni, Praveen Tangarajan, Kyung Kim, Anusua Trivedi. Reinforcement Learning for Optimizing RAG for Domain Chatbots. https://arxiv.org/pdf/2401.06800. 10 Jan 2024

[3] Explorando a Arquitetura de Server Driven UI. https://repositorio.ufms.br/retrieve/fbbd9ca8-369e-4b03-8bbb-21463693bec7/972.pdf

[4] Tarun Singh. Introduction to Retrieval-Augmented Generation (RAG) and its Transformative Role in AI https://medium.com/@krtarunsingh/introduction-to-retrieval-augmented-generation-rag-and-its-transformative-role-in-ai-c07e35da7f01 3 Fev 2024

[5] Pooja Raj. Unlocking the Power of Server-Driven UI Architecture in Mobile App Development https://dev.to/poojankv/unlocking-the-power-of-server-driven-ui-architecture-in-mobile-app-development-4g23 28 Mai 2024

[6] Refactoring Guru. Chain of Responsability https://refactoring.guru/design-patterns/chain-of-responsibility

[7] Repositório Mobile https://github.com/CynaraCosta/tcc-mobile

[8] Repositório Server https://github.com/CynaraCosta/tcc-server

[9] Jagadeesan Ganesh. A Comprehensive Guide to Retrieval-Augmented Generation (RAG): What It Is and How to Use It https://medium.com/@shravankoninti/mastering-rag-a-deep-dive-into-embeddings-b78782aa1259 13 Nov 2024

[10] Novita AI. O que é RAG: Uma introdução abrangente à geração aumentada de recuperação
https://blogs.novita.ai/pt/what-is-rag-a-comprehensive-introduction-to-retrieval-augmented-generation/?utm_source=chatgpt.com **26 April 2024**