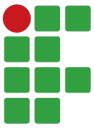

INSTITUTO FEDERAL
Rio Grande do Sul
Campus Rio Grande

Tecnologia em
ANÁLISE E
DESENVOLVIMENTO
DE SISTEMAS

MATHEUS MACIEL LEÃO

Iassistente de Consultas

Trabalho de Conclusão de Curso
apresentado como requisito parcial
para a obtenção do grau de Tecnólogo em
Análise e Desenvolvimento de Sistemas

Prof. Rafael Betito
Orientador

Rio Grande, setembro de 2024

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

1 INTRODUÇÃO

O presente trabalho tem como objetivo desenvolver uma solução tecnológica inovadora para otimizar o gerenciamento de consultas médicas. A saúde é uma área que demanda eficiência e precisão, e, com o avanço das tecnologias, torna-se cada vez mais importante integrar ferramentas que auxiliem os profissionais de saúde em suas atividades diárias. Este projeto se propõe a explorar essas possibilidades, utilizando tecnologias emergentes, como a Inteligência Artificial em modelos de transcrição de áudio para texto e análise automática de dados , para melhorar o atendimento aos pacientes e a organização das informações médicas.

A utilização dessas ferramentas visa melhorar a eficiência e a precisão no processo de documentação das consultas, permitindo que os profissionais de saúde se concentrem mais no atendimento ao paciente, enquanto o sistema automatiza tarefas administrativas e operacionais. Levando em consideração a escolha do usodessas ferramentas, o sistema ganhou o nome de Iassistente de Consultas.

Ao apresentar a ideia desse projeto a um profissional da área de psicologia, ele comentou que em muitos casos, pacientes, especialmente na rede de saúde pública, necessitam de acompanhamento por diferentes médicos durante o seu tratamento. Isso é particularmente comum em tratamentos psiquiátricos e psicológicos, onde a continuidade e a precisão no acompanhamento das consultas são cruciais. Com a crescente adoção da medicina online, surge uma necessidade ainda maior de métodos eficientes para documentar e compartilhar essas informações entre os profissionais envolvidos. Um sistema que centralize essas informações e permita o acesso fácil e organizado a dados de consultas anteriores pode fazer uma diferença significativa na qualidade do cuidado oferecido, garantindo que o paciente receba um tratamento mais coerente e bem documentado.

O público-alvo deste projeto inclui profissionais de saúde, como médicos, psicólogos e psiquiatras, que realizam consultas clínicas e necessitam de uma ferramenta eficiente para registrar as informações dos pacientes e acompanhar o progresso dos tratamentos.

Além dos profissionais de saúde, o sistema também é voltado para clínicas e instituições de saúde que buscam otimizar seus processos internos, garantindo maior agilidade e precisão no atendimento ao paciente.

Pacientes que utilizam serviços médicos também são beneficiados, uma vez que o sistema permite um acompanhamento mais preciso e contínuo de suas condições de saúde, resultando em um atendimento mais personalizado e eficaz.

1.1 Objetivos

Este trabalho tem como foco o desenvolvimento de um sistema de gerenciamento de consultas médicas que visa otimizar o registro e a organização das informações dos

pacientes. O sistema buscará integrar tecnologias de Inteligência Artificial (IA) para aprimorar a eficiência e precisão do atendimento, facilitando o trabalho dos profissionais de saúde e melhorando a experiência do paciente.

1.1.1 Objetivo Geral

Desenvolver um sistema de gerenciamento de consultas médicas que otimize o registro e a organização de informações dos pacientes, incorporando tecnologias emergentes de Inteligência Artificial para melhorar a eficiência e a precisão do atendimento médico.

1.1.2 Objetivos Específicos

Os objetivos específicos para que o sistema seja bem sucedido determinam as principais funcionalidades que ele deve englobar. Pode-se destacar:

- Criar um sistema de cadastro e gerenciamento de pacientes;
- Desenvolver funcionalidades para registro detalhado de consultas, incluindo tratamentos prescritos e anotações do profissional de saúde;
- Implementar tecnologias de transcrição de áudio para texto, permitindo a rápida documentação das consultas;
- Integrar um mecanismo de pré-consulta para fornecer um resumo das informações da consulta anterior, auxiliando na continuidade do tratamento.

2 ANÁLISE

Este capítulo apresenta a análise detalhada do sistema desenvolvido, com o objetivo de entender melhor os componentes, interações e funcionalidades que compõem a solução proposta. A análise foi conduzida em várias etapas, cada uma abordando um aspecto fundamental do sistema, desde a identificação dos atores envolvidos até a definição dos requisitos e a modelagem dos fluxos de trabalho.

Nos subcapítulo, são descritos os **atores**, que representam as entidades externas que interagem com o sistema, seja de forma direta ou indireta. Em seguida, a **elicitação de requisitos**, onde é descrita a identificação e definição dos requisitos funcionais e não funcionais, essenciais para o correto funcionamento da solução.

A análise prossegue com a construção do **diagrama de casos de uso**, que ilustra as principais interações entre os atores e o sistema, proporcionando uma visão clara das funcionalidades esperadas. Por fim, uma **visão geral do sistema** é apresentada, mostrando o fluxo de dados e as ferramentas de maneira resumida.

Essas etapas de análise são fundamentais para garantir que o sistema atenda às necessidades identificadas, proporcionando uma base sólida para as fases subsequentes de projeto e implementação.

2.1 Atores

Neste capítulo, apresentaremos os principais atores envolvidos em nosso sistema, uma API de ponta projetada para transcrição de consultas. Os seguintes papéis desempenham partes cruciais na garantia do processamento eficiente e preciso dos registros de consultas.

Profissional de Saúde Este ator é responsável por utilizar o sistema para realizar o cadastro de pacientes, registrar consultas, inserir tratamentos e prescrições, além de acessar e analisar as informações dos pacientes.

Paciente O paciente é o usuário principal do sistema, embora indiretamente. Ele fornece informações pessoais durante o cadastro e é o objeto das consultas registradas pelos profissionais de saúde. Embora não interaja diretamente com o sistema, suas informações e tratamentos são gerenciados por ele.

IA - ASR O reconhecimento de fala, também conhecido como ASR (Automatic Speech Recognition), reconhecimento de fala por computador ou fala para texto, é um modelo de inteligência artificial que processa a fala humana. Este ator será responsável pela transcrição dos áudios da consulta para texto no sistema.(JURAFSKY; MARTIN, 2023)

IA - LLM LLM (Large Language Models) são modelos de linguagem de grande escala, como o GPT(Generative Pré-trained Transformer), que são treinados em enormes quantidades de texto para aprender padrões e estruturas da linguagem.(JURAFSKY; MARTIN, 2023) Esses modelos são capazes de realizar uma variedade de tarefas de processamento de linguagem natural, como geração de texto, tradução automática, análise de sentimento, entre outras. No contexto do sistema, vai ser responsável por resumir a consulta após a transcrição ser gerada pelo ator IA-ASR.

2.2 Elicitação de Requisitos

Neste capítulo, apresentamos o processo de elicitação de requisitos que guiou a definição das funcionalidades do sistema. A elicitação de requisitos é uma etapa crucial no desenvolvimento de software, pois envolve a coleta e análise das necessidades dos usuários finais para garantir que o sistema a ser desenvolvido atenda às expectativas e requisitos do seu público-alvo.

Para a definição das funcionalidades do sistema, foram realizadas entrevistas abertas não estruturadas e reuniões com profissionais da área da saúde. Os encontros aconteceram nos dias 21/03/2024 e 25/03/2024, sendo com um profissional de psiquiatria e um de psicologia eduraram por volta de 25 minutos. Esses profissionais contribuíram com seu conhecimento e experiência prática, fornecendo informações valiosas sobre as necessidades específicas e desafios enfrentados no contexto de suas práticas clínicas. A partir dessas conversas, foram identificados os principais requisitos funcionais e não funcionais, que foram documentados e priorizados para orientar o desenvolvimento do sistema.

As funcionalidades do sistema foram delineadas para oferecer suporte abrangente ao processo de atendimento, desde o cadastro de pacientes até a transcrição de consultas e gerenciamento de tratamentos. A seguir, detalhamos cada um desses requisitos, especificando suas descrições, critérios de aceite e os atores envolvidos.

2.2.1 Requisitos Funcionais

Nesta seção, são apresentados os requisitos funcionais do sistema, que descrevem as funcionalidades e comportamentos essenciais que o sistema deve possuir para atender às necessidades dos usuários e dos atores envolvidos. Esses requisitos foram definidos com base na análise das interações esperadas entre os usuários e o sistema, bem como das funcionalidades necessárias para alcançar os objetivos do projeto.

Cada requisito funcional especificado a seguir é uma peça fundamental para garantir que o sistema cumpra adequadamente suas finalidades, oferecendo uma experiência de uso eficiente e atendendo às demandas identificadas durante a fase de elicitação de requisitos.

2.2.1.0.1 RF1 - Cadastro de Pacientes

- **Prioridade:** alta.
- **Descrição:**
  - Permitir o cadastro completo de informações dos pacientes, incluindo nome, idade, sexo, histórico médico, alergias, etc.
  - Possibilitar a edição e exclusão de informações de pacientes cadastrados.
- **Critérios de Aceite:**

1. Deve constar no banco de dados todos os dados informados no ato de cadastro pelo Profissional de Saúde.
2. Deve constar no banco de dados as edições e exclusões de informações no cadastro do Paciente pelo Profissional de Saúde.
3. O sistema deve retornar uma resposta HTTP 200 OK após o cadastro, edição ou exclusão bem-sucedidos de informações de pacientes.
4. Em caso de falha durante o cadastro, edição ou exclusão de informações de pacientes devido a erros de validação ou problemas de integridade dos dados, o sistema deve retornar uma resposta HTTP com o código de erro apropriado e uma mensagem explicativa.

- **Atores Envolvidos:**
  - Profissional de Saúde: Responsável por realizar o cadastro, edição e exclusão de informações de pacientes no sistema.

2.2.1.0.2 RF2 - Registro de Consultas

- **Prioridade:** alta.
- **Descrição:**
  - Permitir o registro detalhado de cada consulta, incluindo data, hora, profissional responsável, diagnóstico, tratamento prescrito e anotações relevantes.
  - Permitir a associação de consultas a pacientes cadastrados.
- **Critérios de Aceite:**
  1. O sistema deve armazenar todos os detalhes da consulta no banco de dados, incluindo data, hora, profissional responsável, diagnóstico, tratamento prescrito e anotações relevantes.
  2. Ao associar uma consulta a um paciente cadastrado, o sistema deve verificar a existência do paciente no banco de dados e realizar a associação corretamente.
  3. O sistema deve retornar uma resposta HTTP 200 OK após o registro bem-sucedido da consulta.
  4. Em caso de falha durante o registro da consulta devido a erros de validação ou problemas de integridade dos dados, o sistema deve retornar uma resposta HTTP com o código de erro apropriado e uma mensagem explicativa.
- **Atores Envolvidos:**
  - Profissional de Saúde: Responsável por iniciar e completar o registro da consulta no sistema.

2.2.1.0.3 RF3 - Tratamentos e Prescrições

- **Prioridade:** alta.
- **Descrição:**

  - Oferecer funcionalidades para inserção de tratamentos prescritos durante a consulta, incluindo medicamentos, dosagens, frequência de administração e duração do tratamento.
  - Garantir a possibilidade de edição e exclusão de tratamentos registrados.

- **Critérios de Aceite:**

  1. Os tratamentos prescritos durante uma consulta devem ser armazenados no banco de dados de forma consistente, incluindo informações como medicamentos, dosagens, frequência de administração e duração do tratamento.
  2. O sistema deve permitir a edição e exclusão de tratamentos registrados, atualizando os dados correspondentes no banco de dados.
  3. Após a inserção, edição ou exclusão de um tratamento, o sistema deve retornar uma resposta HTTP 200 OK para indicar o sucesso da operação.
  4. Em caso de falha ao inserir, editar ou excluir um tratamento devido a erros de validação ou problemas de integridade dos dados, o sistema deve retornar uma resposta HTTP com o código de erro apropriado e uma mensagem explicativa.

- **Atores Envolvidos:**

  - Profissional de Saúde: Responsável por inserir, editar e excluir informações sobre tratamentos e prescrições no sistema.

2.2.1.0.4 RF4 - Transcrição de Áudio para Texto

- **Prioridade:** alta.

- **Descrição:**

  - Integrar uma API de transcrição de áudio para texto para permitir a rápida documentação das consultas através da transcrição automática das gravações.

- **Critérios de Aceite:**

  1. A integração com a API de transcrição de áudio para texto deve ser realizada de forma eficiente, garantindo uma resposta rápida para as solicitações de transcrição.
  2. Após a transcrição de um áudio de consulta, o resumo do texto resultante deve ser armazenado no banco de dados para referência futura.
  3. O sistema deve lidar adequadamente com erros de conexão ou resposta da API de transcrição, garantindo a robustez da funcionalidade.
  4. Em caso de transcrição bem-sucedida, o sistema deve retornar uma resposta HTTP 200 OK contendo o texto da transcrição.
  5. Em caso de falha na transcrição devido a problemas na API ou no processamento do áudio, o sistema deve retornar uma resposta HTTP com o código de erro apropriado e uma mensagem explicativa.

- **Atores Envolvidos:**

  - Profissional de Saúde: Utiliza a funcionalidade de transcrição durante o registro de uma consulta no sistema.
  - IA-ASR: Gera o texto resultante da transcrição da gravação da consulta.
  - IA-LLM: Gera o resumo do texto gerado pela IA-ASR.

- **Restrições:**

  - As implementações atuais de ASR possuem um limite quanto ao tamanho de arquivo de áudio que pode ser mandado. No caso de consultas longas, esse arquivo terá de ser cortado e enviado em partes. No momento, o sistema não consulta a informação da onda de áudio do arquivo de gravação da transcrição, podendo cortar alguma palavra no processo de envio do áudio para a IA-ASR.

2.2.1.0.5 RF5 - Pré-Consulta

- **Prioridade:** alta.
- **Descrição:**

  - Desenvolver um mecanismo para apresentar um resumo das informações da consulta anterior durante o registro de uma nova consulta, auxiliando o profissional de saúde na continuidade do tratamento.

- **Critérios de Aceite:**

  1. O sistema deve consultar o banco de dados para recuperar as informações da consulta anterior do paciente durante o processo de pré-consulta.
  2. As informações recuperadas devem ser apresentadas de forma clara e legível para o profissional de saúde, auxiliando na continuidade do tratamento.
  3. Em caso de sucesso na recuperação das informações da consulta anterior, o sistema deve retornar uma resposta HTTP 200 OK contendo os dados da pré-consulta.
  4. Em caso de falha ao recuperar as informações da consulta anterior devido a problemas de integridade dos dados ou erros de conexão com o banco de dados, o sistema deve retornar uma resposta HTTP com o código de erro apropriado e uma mensagem explicativa.

- **Atores Envolvidos:**

  - Profissional de Saúde: Utiliza a funcionalidade de pré-consulta durante o registro de uma nova consulta no sistema.

2.2.2 Requisitos Não Funcionais

Nesta seção, são discutidos os requisitos não funcionais do sistema, que especificam critérios de qualidade, restrições e atributos que o sistema deve possuir, além das funcionalidades. Esses requisitos abordam aspectos como desempenho, segurança, escalabilidade, usabilidade e manutenção, entre outros.

Os requisitos não funcionais são essenciais para garantir que o sistema, além de funcionar corretamente, também seja robusto, seguro e eficiente, oferecendo uma experiência satisfatória aos usuários e facilitando a evolução futura da solução.

2.2.2.0.1 RNF1 - Segurança:

- Garantir a segurança dos dados dos pacientes, utilizando medidas adequadas de criptografia e controle de acesso para proteger as informações confidenciais.

2.2.2.0.2 RNF2 - Desempenho:

- Assegurar que o sistema seja responsivo e tenha um tempo de resposta rápido para evitar atrasos no registro das consultas e no acesso às informações dos pacientes.

2.2.2.0.3 RNF3 - Confiabilidade:

- Assegurar a integridade do sistema, minimizando a possibilidade de erros durante o registro das consultas e o armazenamento das informações dos pacientes.

2.2.2.0.4 RNF4 - Usabilidade/Documentação:

- Desenvolver uma solução com documentação completa e clara, para facilitar a implantação da API em sistemas existentes ou em novos sistemas.

2.3 Diagrama de Casos de Uso

Nessa sessão será apresentado o diagrama de casos de uso do Iassistente de Consultas , representado na figura 2.1.

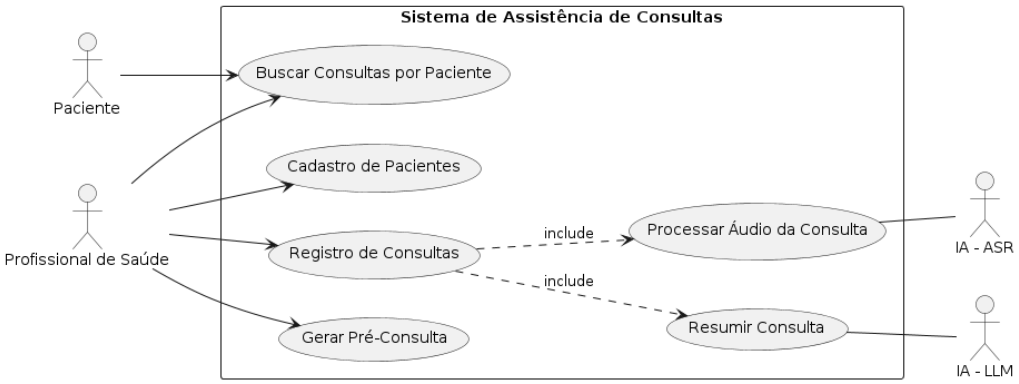

Figura 2.1: Diagrama de Casos de Uso

Na figura 2.1 o ator **Paciente** pode buscar suas consultas registradas no sistema. O ator **Profissional de Saúde** pode realizar o cadastro de pacientes, gerar pré-consultas, buscar consultas por paciente e registrar novas consultas. O ator **IA - ASR** está envolvido nas funcionalidades de transcrição de áudio da consulta para texto. O ator **IA - LLM** está envolvido nas funcionalidades de resumir a transcrição gerada.

2.3.1 Casos de Uso

Os detalhes dos casos de uso apresentados na figura 2.1 são os seguintes:

- **Gerar Pré-Consulta**: Este caso de uso permite que o Profissional de Saúde gere um resumo das informações da consulta anterior durante o registro de uma nova consulta. Ele implementa o requisito funcional RF3.

- **Buscar Consultas por Paciente**:Este caso de uso permite que o Paciente busque suas consultas registradas no sistema. Ele implementa o requisito funcional RF2.

- **Cadastro de Pacientes**: Este caso de uso permite que o Profissional de Saúde gerencie o cadastro dos pacientes. Ele implementa o requisito funcional RF1.

- **Registro de Consultas**: Este caso de uso permite que o Profissional de Saúde registre uma nova consulta no sistema. Ele implementa o requisito funcional RF4.

- **Processar Áudio da Consulta** : Este caso de uso inclui a transcrição do áudio da consulta para texto, realizada pela IA-ASR. Ele implementa o requisito funcional RF5.

- **Resumir Consulta**: Este caso de uso inclui a geração de um resumo da consulta após a transcrição do áudio para texto, realizado pela IA-LLM. Ele implementa o requisito funcional RF6.

2.4 Visão Geral do Sistema

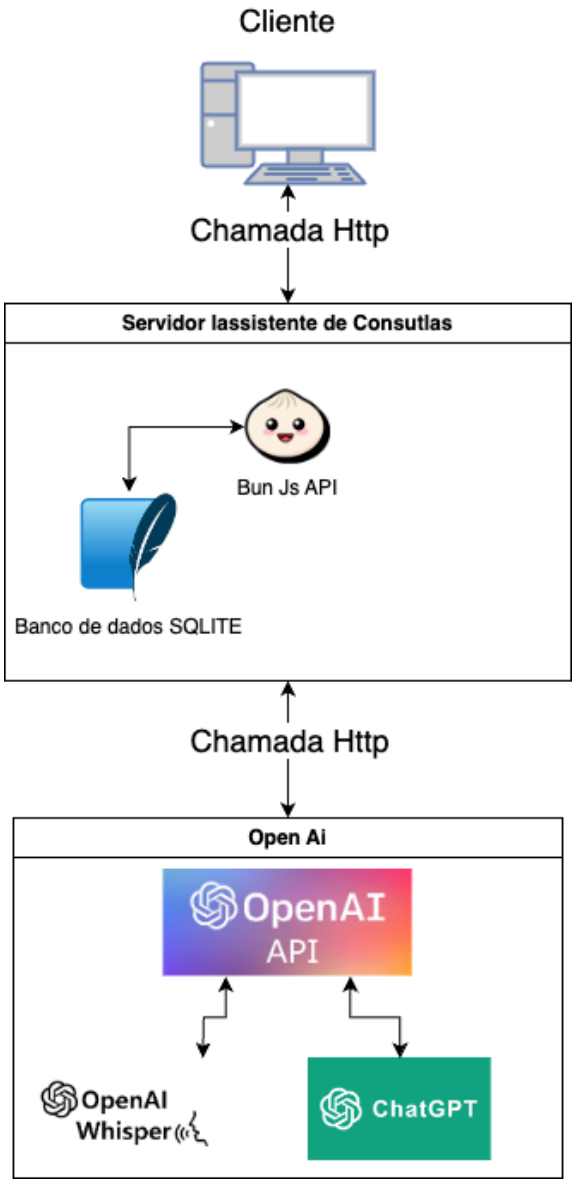

Figura 2.2: Visão geral do Sistema Iassistente de Consultas

Na Figura 2.2 podemos ver uma visão geral do sistema que ilustra a arquitetura e o fluxo básico de operações do Iassistente de Consultas, que utiliza a API da OpenAI, integrada em uma aplicação desenvolvida com Bun JS e uma API REST. Detalhes sobre a implementação e ferramentas serão detalhados nos capítulos de projeto 3 e desenvolvimento 4.

- **Chamada HTTP**: O processo inicia com uma chamada HTTP, que é enviada pelo cliente (pode ser um navegador web ou outra aplicação) para o servidor. Esta chamada pode incluir o upload de um arquivo de áudio para ser transcrito ou o envio de um texto para ser resumido.
- **Servidor**: O servidor é o componente que recebe a chamada HTTP e processa a solicitação. Este servidor está configurado para rodar uma aplicação desenvolvida

com Bun JS, que é uma plataforma para construir aplicações com alta performance em JavaScript.

- **Bun Js API**: Esta camada representa a aplicação construída em Bun JS, que expõe uma API REST. A aplicação processa as solicitações recebidas, que incluem a gestão dos arquivos de áudio, a chamada para a API da OpenAI, e a integração com o banco de dados.
- **Banco de Dados SQLITE**: Após o processamento inicial, os dados podem ser armazenados em um banco de dados SQLITE, que pode guardar informações como o resultado das transcrições, resumos gerados e os metadados dos arquivos processados.
- **API da OpenAI**: A aplicação Bun JS faz chamadas à API da OpenAI para realizar tarefas específicas:
  - **Modelo ASR (Whisper)**: O áudio enviado é processado pelo modelo de Reconhecimento Automático de Fala (ASR), o Whisper, que transcreve o áudio em texto.
  - **Modelo LLM (ChatGPT)**: O texto resultante pode ser enviado ao modelo LLM (Large Language Model), como o ChatGPT, para gerar um resumo ou realizar outras operações de processamento de linguagem natural.
- **Resposta ao Cliente**: Finalmente, os resultados, como a transcrição do áudio ou o resumo do texto, são enviados de volta ao cliente como uma resposta HTTP. O cliente pode então visualizar ou processar esses resultados conforme necessário.

Essa visão geral do sistema reflete a arquitetura modular e escalável da aplicação, onde diferentes componentes (servidor, banco de dados, e APIs externas) trabalham em conjunto para fornecer uma solução eficiente e integrada.

3 PROJETO

3.1 Diagramas de Classes

Os diagramas de classes apresentados nessa sessão representam a estrutura e os relacionamentos entre as diversas classes envolvidas no Iassistente de Consultas. Ele ilustra a organização das entidades e os serviços que operam sobre elas. Foram separados em dois módulos: resumo de consultas e consultas e pacientes.

3.1.1 Módulo de Resumo de Consultas: Descrição das Classes e Relacionamentos

Nesta subseção, será apresentada a estrutura de classes que compõe o módulo de resumo de consultas. O diagrama de classes na Figura 3.1 ilustra os principais componentes e suas interações dentro deste módulo para o processo de geração de resumos a partir de consultas médicas.

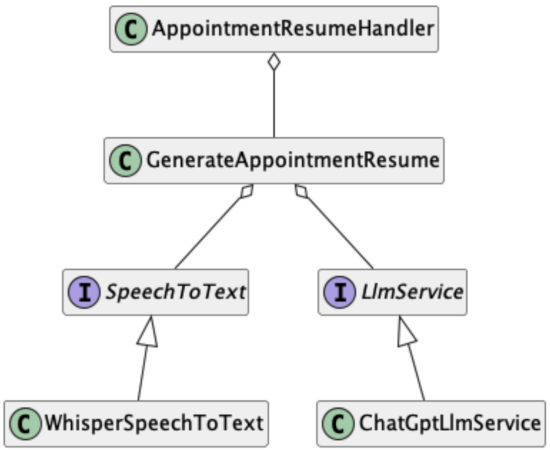

Figura 3.1: Diagrama de classes do módulo de resumo de consultas

3.1.1.1 Classes de Serviço

A Figura 3.2 apresenta um diagrama que detalha as classes de serviço que são fundamentais para o funcionamento do módulo de resumo de consultas.

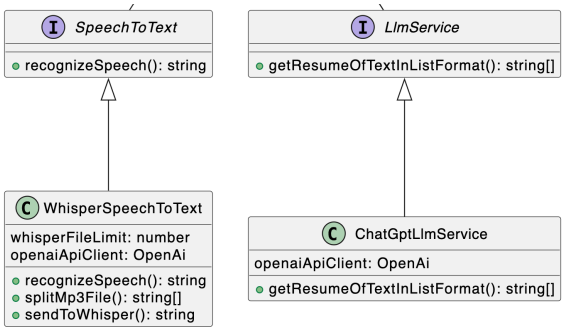

Figura 3.2: Diagrama de classe do handler e orquestração

No diagrama da Figura 3.2, observamos um conjunto de serviços que utilizam o padrão **Adapter** para integrar com APIs externas. O padrão Adapter permite que uma classe trabalhe com uma interface diferente da que foi projetada inicialmente, proporcionando uma forma de adaptar a comunicação entre diferentes sistemasfacilitando a integração e a substituição dessas dependências sem alterar a lógica do sistema.(GAMMA et al., 1994)

- **LlmService:** Interface que define o método para gerar um resumo de texto em formato de lista.
- **ChatGptLlmService:** Implementação da interface *LlmService* utilizando a API do OpenAI. Contém um cliente da API OpenAI (*openaiApiClient*).
- **SpeechToText:** Interface que define o método para reconhecer fala a partir de um arquivo de áudio.
- **WhisperSpeechToText:** Implementação da interface *SpeechToText* utilizando a API do OpenAI.

3.1.1.2 Classes de Orquestração e Manipulação

A Figura 3.3 apresenta o diagrama de classes que detalha as interações das classes responsáveis pela orquestração das operações de Inteligência Artificial e manipulação das rotas da API dentro do módulo de resumo de consultas.

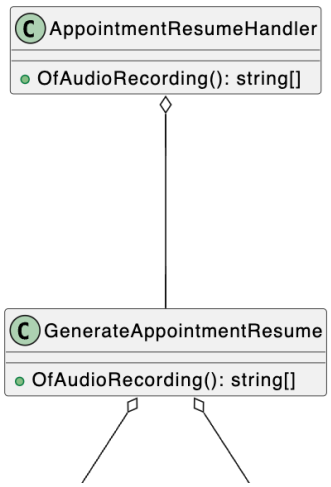

Figura 3.3: Diagrama de classe do handler e orquestração

A classe *GenerateAppointmentResume* é responsável por orquestrar as operações de IA para gerar um resumo de uma consulta médica a partir de uma gravação de áudio. Ela utiliza os serviços *LlmService* e *SpeechToText*.

Por outro lado, a classe *AppointmentResumeHandler* gerencia as regras relacionadas à rota da API para gerar resumos de consultas médicas, interagindo diretamente com a classe *GenerateAppointmentResume* para realizar suas operações.

3.1.2 Módulo de Consultas e Pacientes: Descrição das Classes e Relacionamentos

Nesta subseção, será apresentada a estrutura de classes que compõe o módulo de consultas e pacientes. O diagrama de classes na Figura 3.4 ilustra os componentes e suas interações dentro deste módulo essencial para persistir as informações e suas atualizações no banco de dados.

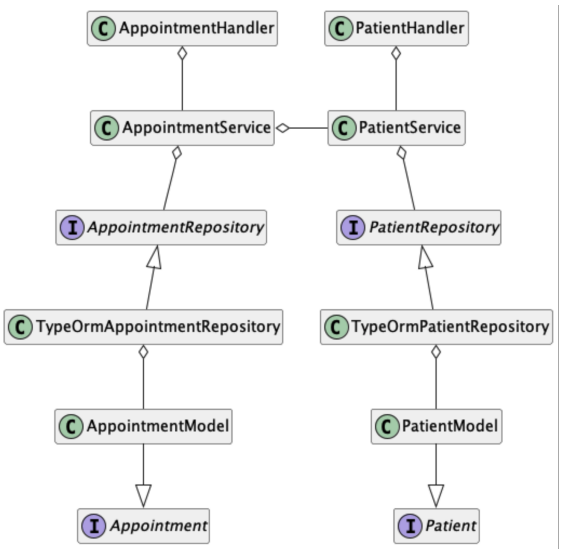

Figura 3.4: Diagrama de classes do módulo de consultas e pacientes

3.1.2.1 Interfaces e Implementações de Repositórios

O padrão Repository é um padrão de design que se utiliza para encapsular o acesso a dados, separando a lógica de acesso a dados da lógica de negócio. O objetivo principal desse padrão é fornecer uma abstração que facilite a comunicação entre a camada de domínio e a camada de persistência. Com isso, permite-se que as operações de CRUD (Create, Read, Update, Delete) sejam realizadas de forma desacoplada, promovendo uma maior flexibilidade e manutenibilidade do código. O padrão Repository ajuda a manter a integridade do modelo de domínio e permite que a lógica de persistência seja alterada sem afetar o restante do sistema (FOWLER, 2003a).

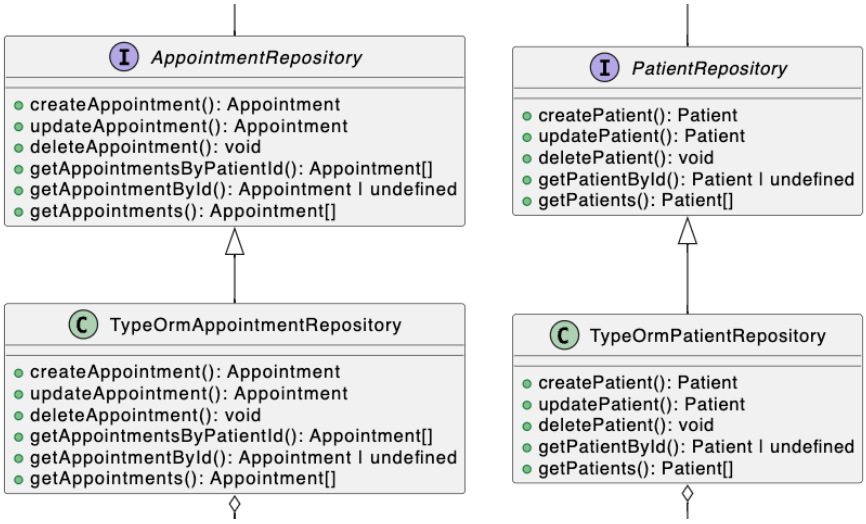

Figura 3.5: Diagrama de classe dos Repositórios

Como mostra a figura 3.5, no sistema esse padrão de projeto parece por meio das interfaces **AppointmentRepository** e **PatientRepository**, bem como nas suas implementações **TypeOrmAppointmentRepository** e **TypeOrmPatientRepository**.

- **AppointmentRepository:** Interface que define os métodos para a criação, atualização, exclusão e consulta de consultas médicas.

- **TypeOrmAppointmentRepository:** Implementação da interface AppointmentRepository utilizando o ORM TypeORM. Implementa todos os métodos definidos na interface *AppointmentRepository*.

- **PatientRepository:** Interface que define os métodos para a criação, atualização, exclusão e consulta de pacientes.

- **TypeOrmPatientRepository:** Implementação da interface PatientRepository utilizando o ORM TypeORM. Implementa todos os métodos definidos na interface *PatientRepository*.

3.1.2.2 Interfaces de Entidades e Modelos de Dados

Neste subcapítulo, são apresentadas as interfaces que definem a estrutura das entidades principais do sistema, bem como os modelos de dados que as implementam como mostra a figura 3.6. As interfaces estabelecem contratos claros para os objetos, garantindo que suas implementações sigam um padrão definido. Os modelos de dados, por sua vez, são as concretizações dessas interfaces, representando as estruturas que serão persistidas no banco de dados.

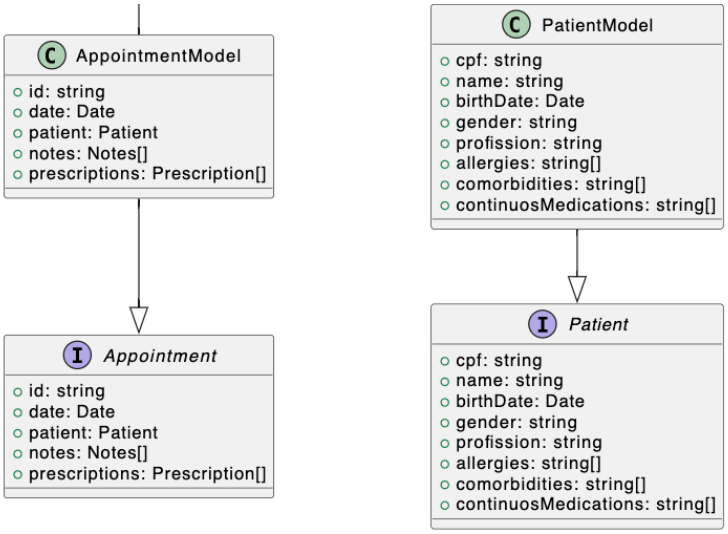

Figura 3.6: Diagrama de classe das Interfaces de Entidades e Modelos de Dados

- **Appointment:** Interface que define a estrutura de uma consulta médica.
- **AppointmentModel:** Classe que representa o modelo de dados para uma consulta médica. Implementa a interface *Appointment*
- **Patient:** Interface que define a estrutura de um paciente.
- **PatientModel:** Classe que representa o modelo de dados para um paciente. Implementa a interface *Patient*

3.1.2.3 Serviços e Manipuladores

Neste subcapítulo, será abordada a arquitetura interna das classes de serviços e manipuladores (*handlers*) do sistema. A Figura 3.7 apresenta o diagrama de classes que detalha a relação entre essas classes, ilustrando como os serviços encapsulam a lógica de negócio e como os manipuladores interagem com esses serviços para atender às solicitações dos usuários via API.

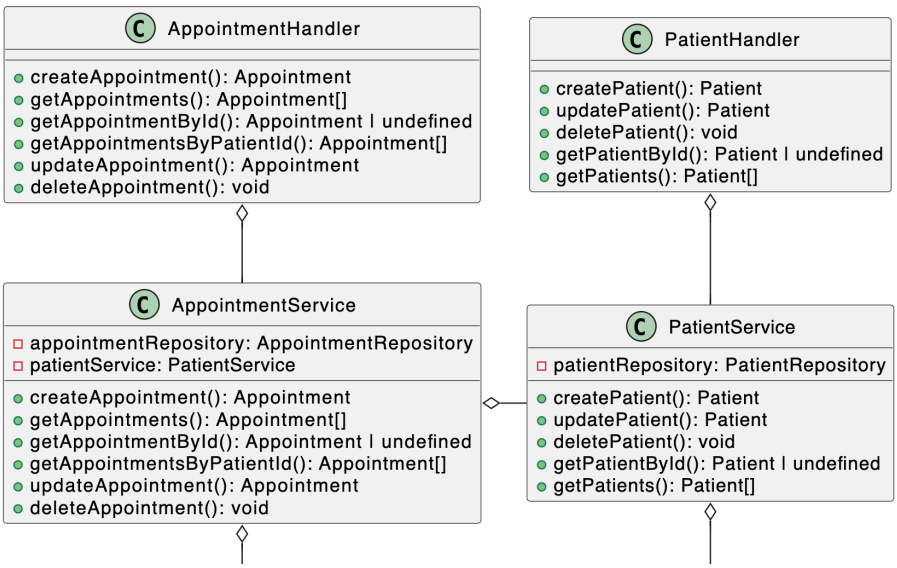

Figura 3.7: Diagrama de Classes Services e Handlers

Serviços são classes responsáveis pela lógica de negócio e operações relacionadas a um domínio específico. Eles agem como intermediários entre a camada de controle e a camada de persistência.

- **AppointmentService:** Classe de serviço que gerencia as operações relacionadas a consultas médicas. Depende da interface *AppointmentRepository* para acessar dados persistidos e da classe *PatientService* para integrar informações de pacientes com consultas.
- **PatientService:** Classe de serviço que gerencia as operações relacionadas a pacientes, incluindo a criação, atualização, e obtenção de informações de pacientes.

Handlers são responsáveis por gerenciar as regras de negócio específicas relacionadas às rotas da API. Eles interagem diretamente com os serviços para realizar as operações solicitadas pelos usuários. Neste sistema, temos os seguintes *handlers*:

- **AppointmentHandler:** Classe que gerencia as regras relacionadas à rota da API para operações de consultas médicas. Depende da classe *AppointmentService* para realizar suas operações.
- **PatientHandler:** Classe que gerencia as regras relacionadas à rota da API para operações de pacientes. Depende da classe *PatientService* para realizar suas operações.

O padrão de **Injeção de Dependência** é utilizado para fornecer as dependências necessárias a essas classes sem acoplá-las diretamente (FOWLER, 2024). No exemplo da classe *AppointmentService*, ela é injetada com as dependências *AppointmentRepository* e *PatientService*, permitindo uma maior flexibilidade e testabilidade. Isso facilita a substituição de implementações e a realização de testes unitários, promovendo um design mais modular e sustentável.

3.2 Diagramas de Sequencia

Nessa sessão será apresentado um diagrama de sequencia que dá enfase no que seria o uso de dia a dia do sistema. Ele apresenta o processo onde o profissional de saúde insere a gravação de consulta, o sistema processa e gera o resumo, para que o profissional use-o junto dos outros dados para persistir a consulta no banco de dados.

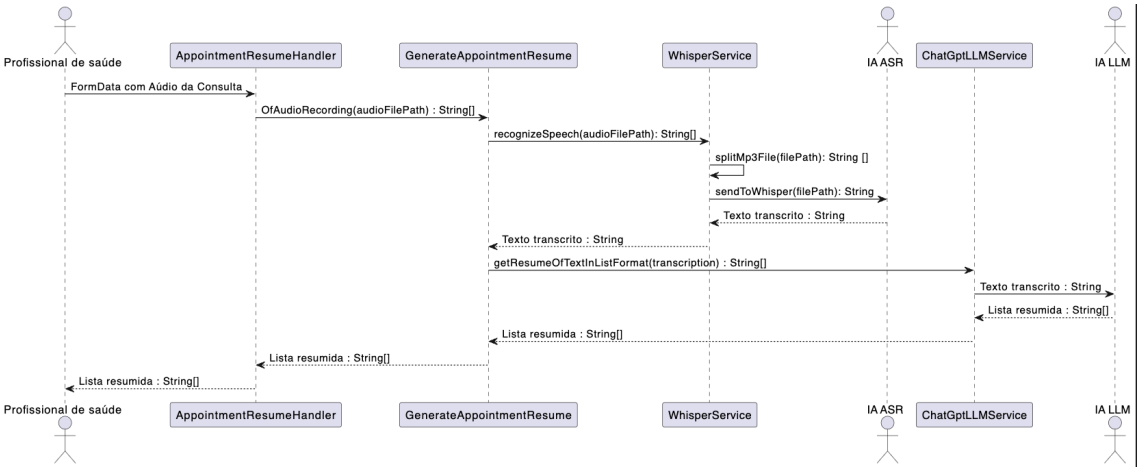

Figura 3.8: Diagrama de Sequencia Parte 1

O Diagrama de Sequencia representado pela figura 3.8 ilustra a primeira parte do fluxo que trata da transcrição do áudio da consulta e a geração de um resumo em formato de lista. O fluxo inicia com o *Profissional de saúde* enviando um formulário com o áudio da consulta para a classe `AppointmentResumeHandler`. Esta classe, por sua vez, delega a tarefa de processamento do áudio à classe `GenerateAppointmentResume`, que coordena os serviços necessários para a transcrição e o resumo.

O primeiro serviço chamado é o `WhisperService`, que é responsável por dividir o arquivo de áudio em partes menores, se necessário, e enviar essas partes para a IA ASR. A IA ASR retorna o texto transcrito, que é consolidado e enviado de volta para a classe `GenerateAppointmentResume`.

Com o texto transcrito, a classe `GenerateAppointmentResume` chama o serviço `ChatGptLLMService`, que utiliza a IA LLM para gerar um resumo em formato de lista. Este resumo é então retornado ao Profissional de saúde através da `AppointmentResumeHandler`, que pode ajustá-lo conforme suas preferências antes de prosseguir para o agendamento da consulta.

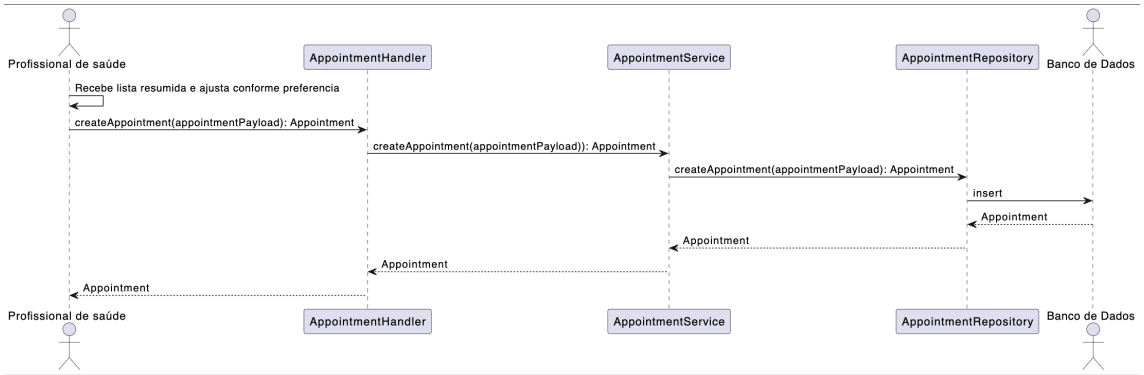

Figura 3.9: Diagrama de Sequencia Parte 2

Dando sequencia, o Diagrama Ilustrado pela figura3.4 começa o fluxo Após o *Profissional de saúde* ajustar a lista resumida, ele inicia o processo de criação de uma nova consulta através da classe `AppointmentHandler`. A Requisição contendo as informações da consulta é passado para a classe `AppointmentService`, que então se comunica com a `AppointmentRepository` para inserir os dados no banco de dados (SGBD).

O `AppointmentRepository` realiza a inserção no banco de dados e retorna o objeto `Appointment` criado para a `AppointmentService`, que, por sua vez, repassa a consulta criada de volta à `AppointmentHandler`. Finalmente, a consulta é retornada ao Profissional de saúde, completando assim o fluxo de registro da consulta.

Este fluxo é crítico para garantir que as consultas sejam registradas de maneira correta e eficiente no sistema, com todos os dados necessários armazenados e disponíveis para futuras referências.

3.3 Modelo de Entidade-Relacionamento

O Diagrama de Entidade-Relacionamento (ERD) a seguir representa as entidades e seus relacionamentos no sistema. Foram modeladas as seguintes classes: *PatientModel*, *AppointmentModel*, *NotesModel* e *PrescriptionModel*.

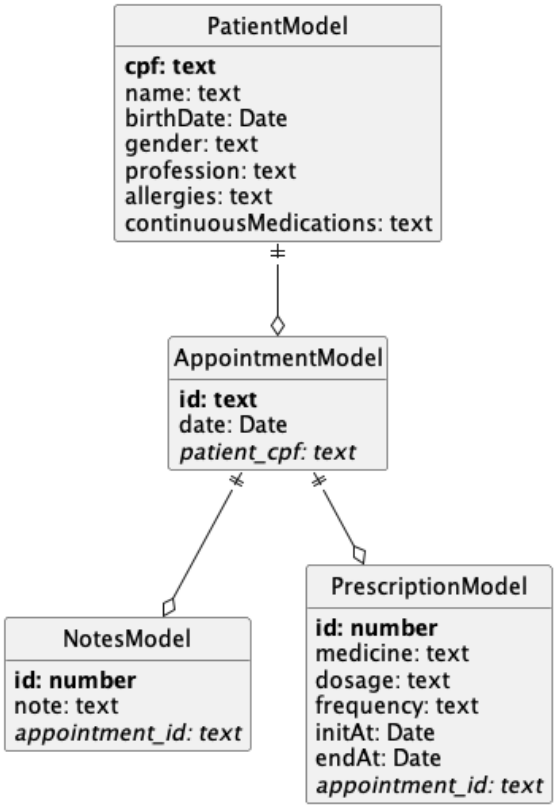

Figura 3.10: Diagrama de Entidade-Relacionamento

Este diagrama de classes representa a estrutura e os relacionamentos entre as diversas classes envolvidas no sistema de gerenciamento de consultas e pacientes. Ele ilustra a organização das entidades e os serviços que operam sobre elas.

3.3.1 Relacionamentos

- **PatientModel** para **AppointmentModel**: Um paciente pode ter múltiplas consultas (relação um-para-muitos).
- **AppointmentModel** para **NotesModel**: Uma consulta pode ter múltiplas notas (relação um-para-muitos).
- **AppointmentModel** para **PrescriptionModel**: Uma consulta pode ter múltiplas prescrições (relação um-para-muitos).

O diagrama de ERD ilustra claramente como as entidades estão relacionadas entre si no sistema, proporcionando uma visão abrangente da estrutura de dados.

3.4 Arquitetura

A arquitetura do sistema foi projetada seguindo os princípios da Arquitetura Limpa, conforme descrito por Robert C. Martin (MARTIN, 2017). A organização das pastas e módulos do projeto foi cuidadosamente planejada para garantir a separação de responsabilidades, facilitando a manutenção, a escalabilidade e a testabilidade do sistema. Esta seção descreve as principais camadas do sistema: core e infra.

3.4.0.1 Core

A camada *core*, representada na figura 3.11 é o núcleo do sistema, onde se encontram as regras de negócio e o comportamento essencial da aplicação. Esta camada está dividida em duas subcamadas: *domain* e *application*.

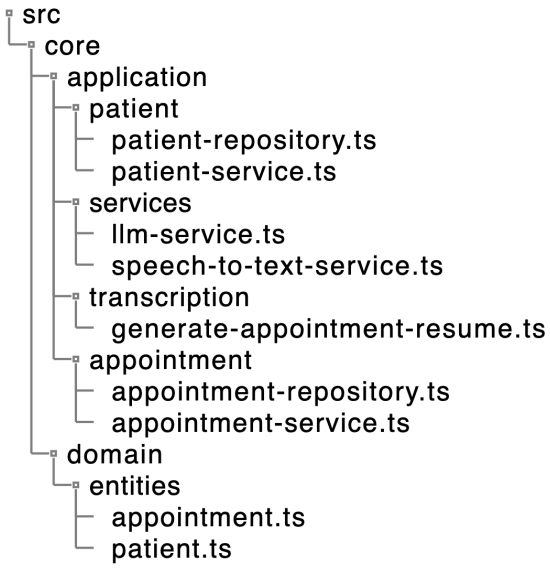

Figura 3.11: Estrutura de pastas: Camada Core

3.4.0.1.1 Domain

A subcamada *domain* contém as entidades e objetos de valor que representam os conceitos centrais do domínio da aplicação. Estes componentes encapsulam as regras de negócio e garantem a integridade das operações realizadas no sistema. No projeto, essa subcamada contem as entidades referentes aos Pacientes e as Consultas.

3.4.0.1.2 Application

A subcamada *application* é responsável por orquestrar as operações de negócio, interagindo com as entidades do *domain* e gerenciando os fluxos de trabalho do sistema. Esta subcamada não deve conter lógica de negócio complexa, mas sim coordenar as interações entre diferentes componentes (MARTIN, 2017). A estrutura da subcamada application, conforme a figura 3.11 no projeto é a seguinte:

appointment: Esta subpasta contém arquivos relacionados à gestão de consultas médicas. O arquivo *appointment-repository.ts* define a interface para a manipulação dos dados de consultas, enquanto o *appointment-service.ts* implementa a lógica de negócios para criar, atualizar, deletar e buscar consultas, aplicando as regras de negócio definidas na camada de domínio.

patient: Similar à subpasta de *appointment*, esta subpasta contém arquivos que tratam da gestão de pacientes. O *patient-repository.ts* define a interface para manipulação dos dados de pacientes, e o *patient-service.ts* implementa a lógica necessária para gerenciar esses dados, garantindo que as operações realizadas estejam de acordo com as regras de negócio.

services: Esta subpasta é dedicada a serviços mais genéricos que são utilizados pelo sistema, mas que não se enquadram diretamente em uma entidade de negócio específica.

Por exemplo, o *llm-service.ts* trata das interações com modelos de linguagem de grande escala, como o ChatGPT, e o *speech-to-text-service.ts* lida com a transcrição de áudio para texto, usando serviços como o Whisper. Esses serviços são invocados por outras partes do sistema, principalmente durante o processo de geração de resumos de consultas a partir de gravações de áudio.

transcription: Focada na funcionalidade específica de gerar resumos de consultas, esta subpasta contém o arquivo *generate-appointment-resume.ts*, que coordena o processo de transcrição de áudio e geração de resumos textuais, utilizando os serviços definidos na subpasta *services*.

3.4.0.2 Infra

A camada *infra* lida com a implementação dos detalhes externos que o sistema precisa para funcionar, como interações com bancos de dados, APIs de terceiros, e outros serviços externos. Esta camada contém adaptações e implementações específicas que permitem que o *core* 3.11 funcione de maneira agnóstica a essas dependências externas. A organização da camada *infra* no projeto está representada na figura 3.12.

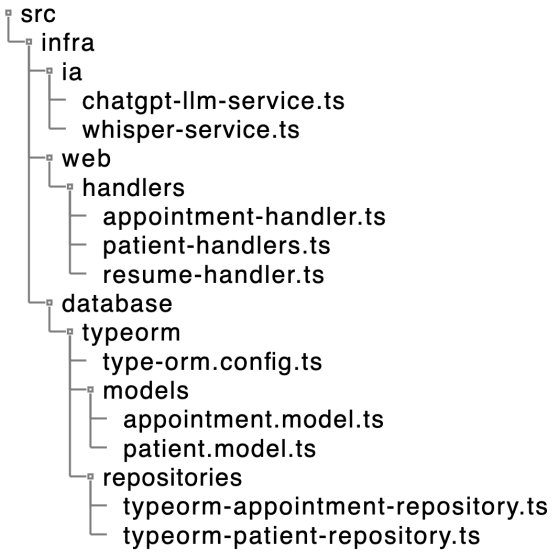

Figura 3.12: Estrutura de pastas: Camada Infra

Na figura 3.12 existem as subpastas ia, web e database.

ia: Esta subpasta é dedicada aos serviços de Inteligência Artificial utilizados no sistema. Por exemplo, o arquivo *chatgpt-llm-service.ts* implementa a comunicação com o serviço de linguagem ChatGPT(OPENAI, 2020) para geração de resumos textuais, enquanto o *whisper-service.ts* cuida da transcrição de áudio para texto usando o Whisper. Esses serviços são fundamentais para funcionalidades como a transcrição de consultas médicas e a geração automática de resumos.

web: A subpasta *web* contém os manipuladores que servem como ponto de entrada para as interações com o sistema via web. Arquivos como *appointment-handler.ts*, *patient-handlers.ts* e *resume-handler.ts* gerenciam as requisições HTTP, validam dados de entrada, e encaminham as solicitações para as camadas apropriadas da aplicação, como os serviços na camada *application*.

database: Esta subpasta contém tudo relacionado à persistência de dados. O arquivo *type-orm.config.ts* configura o TypeORM para trabalhar com o banco de dados utilizado.

Já os arquivos na subpasta *models*, como *appointment.model.ts* e *patient.model.ts*, definem os modelos de dados correspondentes às entidades de negócio. Os repositórios, como *typeorm-appointment-repository.ts* e *typeorm-patient-repository.ts*, implementam as interfaces definidas na camada de aplicação, permitindo a interação com o banco de dados por meio do TypeORM.

3.4.1 Conclusão

A arquitetura do sistema, fundamentada na Arquitetura Limpa, assegura uma estrutura robusta e flexível, capaz de evoluir e se adaptar às necessidades futuras, ao mesmo tempo que mantém a clareza e a separação de responsabilidades entre as diferentes partes do sistema.

3.5 Ferramentas

3.5.1 Bun JS

A escolha do JavaScript para o backend se deu pelo robusto ecossistema e bibliotecas nativas para interação com os modelos GPT e Whisper da OpenAI. O Bun é um runtime de JavaScript construído do zero para atender ao ecossistema moderno de JavaScript. Ele tem três principais objetivos: rapidez, APIs elegantes e integração coesa. Para isso, o Bun começa rapidamente e roda rápido, estendendo o motor de JavaScript construído para o Safari. Além disso, oferece uma API minimizada e otimizada para tarefas comuns, como iniciar um servidor HTTP e escrever arquivos.(BUN, 2024a) Isso torna o Bun uma escolha atraente para desenvolver APIs backend, pois é projetado como um substituto para o Node.js, implementando naturalmente centenas de APIs de Node.js e Web, incluindo fs, path, Buffer e mais. A escolha do Bun ao invés do Node JS se justifica ainda mais quando levamos em conta a nescessidade de interagir muito com o IO para realizar a quebra dos arquivos de audio antes de mandar para transcrição.(BUN, 2024b)

3.5.2 OpenAI API

A abordagem utilizada por APIs da OpenAI, como GPT e Whisper, permite a integração desses modelos de linguagem avançados em aplicações sem a necessidade de recursos computacionais específicos. Isso torna mais fácil implementar e gerenciar sistemas que dependem dessas tecnologias, permitindo uma maior escalabilidade e flexibilidade. No momento da análise, as versões dos modelos utilizados foram o Chat GPT-3.5 Turbo e o Whisper-1, que eram as opções com o melhor custo-benefício e documentação disponível. Além disso, a execução remota dos modelos reduz a carga de trabalho da máquina local, o que é especialmente importante em aplicações que demandam processamento intensivo, como inteligência artificial e aprendizado automático.(OPENAI, 2024)(OPENAI, 2020)(OPENAI, 2022)

No entanto, é importante mencionar que a dependência de APIs externas pode gerar custos ao longo prazo, pois as empresas que oferecem essas APIs podem aumentar os preços ou alterar suas políticas de uso. Contudo, o sistema atualmente em desenvolvimento utiliza o padrão Ports and Adapters para transcrição e resumo. Logo, se houver uma máquina com os requisitos necessários para executar modelos de IA LLM e Whisper localmente, essas APIs podem ser substituídas para reduzir o custo ao longo prazo. Isso permitiria uma maior autonomia e controle sobre as informações processadas, tornando-se um modelo mais sustentável e eficaz no futuro.

3.5.3 FFMPEG

O FFMPEG(DEVELOPERS, 2000) é uma ferramenta de software livre extremamente poderosa e versátil para a manipulação de arquivos de áudio e vídeo. Amplamente utilizada em diversas aplicações, o FFMPEG permite a conversão, gravação, edição e transmissão de arquivos multimídia em uma grande variedade de formatos. Sua flexibilidade e robustez tornam-no uma escolha popular para tarefas que exigem processamento avançado de mídia.

Neste projeto, o FFMPEG é utilizado para otimizar o processo de transcrição de áudio em consultas médicas. Especificamente, ele é empregado para dividir grandes arquivos de áudio em partes menores, facilitando o processamento desses segmentos por APIs de transcrição, como o Whisper, que possuem limites no tamanho dos arquivos que podem ser processados de uma só vez.

O uso do FFMPEG oferece várias vantagens no contexto deste projeto. Primeiramente, ao dividir o áudio em segmentos menores, o sistema consegue enviar essas partes para a API de transcrição de forma paralela ou sequencial, acelerando o processo geral de transcrição. Isso é particularmente útil quando se trata de áudios longos, como gravações de consultas médicas detalhadas, que poderiam exceder os limites de tamanho impostos pela API.

Além disso, ao dividir o áudio em segmentos, o FFMPEG permite que o sistema contorne os limites da API, evitando erros que poderiam ocorrer ao tentar processar arquivos grandes demais de uma só vez. Com isso, o sistema pode processar áudios de qualquer duração, garantindo que todos os trechos da consulta sejam transcritos com precisão, sem perder informações importantes.

Por fim, o FFMPEG é configurável para ajustar o tamanho dos segmentos de acordo com as necessidades específicas do sistema e os limites da API de transcrição. Essa flexibilidade torna o FFMPEG uma ferramenta essencial para garantir a eficiência e a robustez do processo de transcrição de áudio, permitindo que o sistema ofereça um suporte confiável aos profissionais de saúde na documentação das consultas médicas.

3.5.4 SQLite

Para o desenvolvimento inicial deste projeto, o SQLite foi escolhido como o banco de dados padrão da API. O SQLite é uma biblioteca de banco de dados relacional leve e auto-contida que não requer uma configuração complexa de servidor, o que o torna uma excelente opção para desenvolvimento, testes e pequenos projetos. Sua simplicidade e portabilidade permitem que o sistema seja utilizado imediatamente, sem a necessidade de configurar e manter a infraestrutura de um banco de dados relacional tradicional, como MySQL ou PostgreSQL(HIPP, 2000)(CORPORATION, 1995)(GROUP, 1996).

Uma das principais vantagens do SQLite é que ele armazena todos os dados em um único arquivo, facilitando a configuração e a mobilidade do ambiente de desenvolvimento. Isso é especialmente útil durante as fases iniciais do projeto, quando a prioridade é a agilidade no desenvolvimento e a facilidade de testes. Com o SQLite, os desenvolvedores podem se concentrar no desenvolvimento das funcionalidades principais da API sem se preocupar com a administração de um servidor de banco de dados.

Além disso, a escolha do SQLite não compromete a escalabilidade do sistema. Como o projeto utiliza um ORM (Object-Relational Mapping), a migração para um banco de dados relacional mais robusto e tradicional, como MySQL ou PostgreSQL, pode ser realizada de maneira simples e tranquila. O ORM abstrai a camada de persistência de dados,

permitindo que a lógica da aplicação permaneça inalterada enquanto o backend de armazenamento de dados pode ser trocado conforme as necessidades do sistema evoluem.

Quando o sistema precisar escalar e suportar um volume maior de dados e de transações, a transição do SQLite para outro banco de dados será facilitada pelas ferramentas de migração oferecidas pelo ORM. Dessa forma, o SQLite serve como uma solução eficaz para o estágio inicial do projeto, ao mesmo tempo em que permite uma evolução suave para uma infraestrutura mais robusta conforme as demandas do sistema crescem.

4 DESENVOLVIMENTO

Neste capítulo, serão discutidos os aspectos técnicos do desenvolvimento do Iassistente de Consultas, focando em trechos de código relevantes tanto da primeira quanto da última versão do algoritmo de transcrição de consultas. A evolução do código será analisada, demonstrando as melhorias implementadas ao longo do tempo.

Além disso, será apresentada uma seção específica para o contrato da API do sistema, que foi documentado utilizando a especificação OpenAPI 3.0. A documentação completa da API pode ser acessada por meio de um arquivo YAML, disponível no repositório do projeto no GitHub. A integração desse contrato visa garantir a clareza e a padronização das interações entre os diferentes módulos do sistema.

O código completo e o contrato da API podem ser acessados no repositório do projeto, disponível em `https://github.com/matheusmacielleao/iassistente-consultas`.

4.1 Recorte do primeiro Algoritmo de Transcrição

O objetivo principal do Primeiro Algoritmo, representado pelo recorte de código 4.1 foi validar a eficácia das técnicas de transcrição e resumo aplicadas a um arquivo de áudio.

```
1         async function processOnGpt(message: string): Promise<string[]>
      {
2   const result = await openai.chat.completions.create({
3     messages: [
4       {
5         role: "user",
6         content:
7           "gere um resumo em pontos da seguinte trancricao de audio: "
     +
8           message,
9       },
10    ],
11    model: "gpt-3.5-turbo",
12  });
13  const processedMessage = result.choices[0].message.content;
14  return processedMessage.split("\n");
15 }
16
17 async function mp3ToText(filePath: string): Promise<string> {
18   const transcription = await openai.audio.transcriptions.create({
19     file: fs.createReadStream(filePath),
20     model: "whisper-1",
21     response_format: "verbose_json",
22     timestamp_granularities: ["word"],
23   });
```

```
24
25   return transcription.text;
26 }
27
28 const execute = async () => {
29   const filePath = "./audio.mp3";
30   const resolvedText = await mp3ToText(filePath);
31   const processedText = await processOnGpt(resolvedText);
32   console.log(processedText);
33 };
34 execute();
35
```

Código 4.1: Primeiro algoritmo

O código 4.1 descreve o fluxo de processamento de áudio para texto e subsequente resumo utilizando os serviços de IA. A função `mp3ToText` (linhas 10-18) é responsável pela conversão de um arquivo MP3 em texto escrito. Nessa função, a chamada ao método `openai.audio.transcriptions.create` na linha 13 realiza a transcrição do áudio, utilizando o modelo `whisper-1` (linha 14), conhecido por sua alta precisão na tarefa de reconhecimento de fala. O áudio é lido a partir do arquivo utilizando `fs.createReadStream` na linha 13.

Após obter a transcrição do áudio, o texto resultante é processado pela função `processOnGpt`(linhas 1-9), que utiliza o modelo de linguagem `gpt-3.5-turbo`. Dentro dessa função, na linha 3, ocorre a chamada ao método `openai.chat.completions.create`, que envia uma mensagem ao GPT-3.5, solicitando a geração de um resumo em formato de pontos da transcrição do áudio (linha 6). O resultado é capturado na variável `processedMessage` (linha 8) e, em seguida, dividido em linhas utilizando `split("\n")` para criar uma lista de itens.

A função `execute` (linhas 20-26) coordena o fluxo do programa. Primeiramente, define o caminho do arquivo MP3 (linha 21) e, em seguida, chama `mp3ToText` (linha 22) para obter a transcrição do áudio. Com o texto transcrito, a função `processOnGpt` (linha 23) é invocada para gerar o resumo. Finalmente, o resumo gerado é exibido no terminal usando `console.log(processedText)` (linha 24), facilitando a revisão e análise do conteúdo transcrito.

Esse Primeiro Algoritmo inicial estabeleceu a base para o desenvolvimento subsequente e após ser submetidos aos testes descritos nas sessões 5.1.1 e 5.1.2 foi refinado para criar as três classes descritas a seguir na Seção 4.2.

4.2 Recorte do algoritmo final responsável pela Transcrição e Resumo de Consultas com Inteligência Artificial

Nessa sessão serão apresentados recortes de códigos da implementação final do módulo do sistema que transcreve e resume consultas médicas. Este sistema utiliza serviços de reconhecimento de fala e modelos de linguagem natural avançados para transformar gravações em texto e gerar resumos estruturados. Abaixo, detalhamos cada componente e suas funções:

4.2.1 Serviço de Reconhecimento de Fala: *WhisperService*

A classe WhisperService possui três principais funções que trabalham em conjunto para realizar a transcrição de arquivos de áudio utilizando o modelo Whisper da Ope-

nAI. A função principal, `recognizeSpeech`, orquestra as outras duas funções. Ela é responsável por dividir o arquivo de áudio original em partes menores, respeitando o limite de tamanho do Whisper, através da função `splitMp3File`. Em seguida, a função `sendToWhisper` realiza a chamada ao modelo Whisper para transcrever cada uma dessas partes, gerando a transcrição final.

```
1     async splitMp3File(filePath: string): Promise<SlitedFilePaths> {
2     const mbChuncksSize = this.whisperFileLimit;
3
4     const parser: IAudioMetadata = await mm.parseFile(filePath, {
5       duration: true,
6     });
7     const file = Bun.file(filePath);
8
9     const fileSizeMb = file.size / 1000000;
10
11    if (!parser.format.duration) {
12      throw new Error("Error parsing file duration");
13    }
14
15    const duration = parser.format.duration | 0;
16    const rest = fileSizeMb % mbChuncksSize;
17
18    const cuts =
19      rest !== 0
20        ? ((fileSizeMb / mbChuncksSize) | 0) + 1
21        : (fileSizeMb / mbChuncksSize) | 0;
22
23    const sectionsSize = ((duration / cuts) | 0) + 1;
24
25    const date = new Date(Date.UTC(2000, 0, 0, 0, 0, 0, 0));
26    const sections: string[] = [];
27
28    for (let i = 0; i < cuts; i++) {
29      const start = \`\${date.getMinutes()}:\${date.getSeconds()}\`;
30      date.setTime(date.getTime() + sectionsSize * 1000);
31      const end = \`\${date.getMinutes()}:\${date.getSeconds()}\`;
32      sections.push(\`[\${start} - \${end}] part-\${i + 1}\`);
33    }
34
35    const split = new MediaSplit({
36      input: filePath,
37      sections,
38    } as any);
39
40    const splitedFilesPaths: Array<string> = (await split.parse()).map(
41      (section: { name: string }) => {
42        return section.name;
43      }
44    );
45    return splitedFilesPaths;
46  }
```

Código 4.2: Método splitMp3File

O método representado pelo código 4.2, divide o arquivo de áudio original em partes menores, caso ele exceda o limite de tamanho especificado. Ele utiliza a biblioteca *music-metadata* para analisar a duração do arquivo de áudio. Logo após calcula quantas partes

serão necessárias para cumprir o limite de tamanho, e divide o áudio em seções de acordo com a duração total. E por fim, utiliza a classe *MediaSplit* para realizar a divisão, gerando uma lista de caminhos para os arquivos resultantes.

```
1            async sendToWhisper(audioFilePath: string): Promise<string>
     {
2         const transcription = await this.openai.audio.transcriptions.
     create({
3           file: fs.createReadStream(audioFilePath),
4           model: "whisper-1",
5           response_format: "verbose_json",
6           timestamp_granularities: ["word"],
7         });
8
9         return transcription.text;
10          }
11
```

Código 4.3: Método sendToWhisper

O método *sendToWhisper* representado pelo código 4.3, envia um arquivo de áudio para o serviço de transcrição do modelo Whisper. Ele faz uma chamada à API do OpenAI, especificando o modelo *whisper-1* e o formato de resposta em JSON detalhado. O método retorna o texto transcrito.

```
1               async recognizeSpeech(audioFilePath: string): Promise<
     string> {
2                 const splitedFilesPaths = await this.splitMp3File(
     audioFilePath);
3
4                 const transcriptions = await Promise.all(
5                   splitedFilesPaths.map((filePath) => this.
     sendToWhisper(filePath))
6                 );
7                 return transcriptions.join(" ");
8               }
9
```

Código 4.4: Método recognizeSpeech

Este método é responsável por orquestrar o processo de transcrição. Ele recebe o caminho de um arquivo de áudio (*audioFilePath*) como parâmetro e executa uma sequencia de passos. Primeiramente, o arquivo de áudio é dividido em partes menores, se necessário, utilizando o método *splitMp3File*. Logo após cada parte do áudio é então enviada ao modelo Whisper através do método *sendToWhisper*. Finalmente, as transcrições retornadas são unificadas em uma única string, que é então retornada como resultado final.

4.2.2 Serviço de Modelo de Linguagem: *ChatGptLLMService*

```
1 export class ChatGptLLMService implements LlmService {
2   constructor(private readonly openai: OpenAI) {}
3
4   async getResumeOfTextInListFormat(prompt: string): Promise<string[]>
     {
5     const result = await this.openai.chat.completions.create({
6       messages: [
7         {
8           role: "user",
```

```
9           content: "gere um resumo em pontos da seguinte trancrição de
    áudio, sabendo que podem conter erros de transcrição: " + prompt,
10        },
11      ],
12      model: "gpt-3.5-turbo",
13    });
14
15    const processedMessage = result.choices[0].message.content;
16    if (!processedMessage) {
17      throw new Error("Error processing message on GPT");
18    }
19
20    return processedMessage.split("\n");
21  }
22 }
```

Código 4.5: Classe ChatGptLLMService

A Classe ChatGptLLMService representada pelo código 4.5 implementa o serviço de modelo de linguagem utilizando a API da OpenAI. E possui a seguinte estrutura:

- **Construtor**: Recebe uma instância do cliente OpenAI.
- **Método *getResumeOfTextInListFormat***:
  1. O método faz uma chamada assíncrona à API do OpenAI utilizando o modelo *gpt-3.5-turbo*. A chamada inclui uma mensagem do tipo *user*, que instrui o modelo a "gerar um resumo em pontos da seguinte transcrição de áudio, sabendo que podem conter erros de transcrição", seguida pelo conteúdo do *prompt*.
  2. A resposta da API é armazenada na variável *result*. O conteúdo da primeira escolha (*choices[0]*) na resposta é extraído e armazenado na variável *processedMessage*.
  3. Se a resposta da API não contiver um conteúdo válido, o método lança uma exceção com a mensagem *"Error processing message on GPT"*.
  4. Caso contrário, o conteúdo da mensagem é dividido em linhas usando o método *split("\n")*, e o resultado é retornado como uma lista de strings. Cada string da lista representa um item do resumo gerado.

  Este método aproveita a capacidade do modelo GPT-3.5 de processar e resumir textos complexos, retornando um resumo estruturado da transcrição da consulta.

4.2.3 Classe GenerateAppointmentResume

```
1 export class GenerateAppointmentResume {
2   constructor(
3     private readonly llmService: LlmService,
4     private readonly speechToText: SpeechToText
5   ) {}
6
7   async OfAudioRecording(audioFilePath: string): Promise<string[]> {
8     const transcription = await this.speechToText.recognizeSpeech(
9       audioFilePath
10    );
```

```
11      return this.llmService.getResumeOfTextInListFormat(transcription);
12    }
13  }
```

Código 4.6: Classe GenerateAppointmentResume

A *GenerateAppointmentResume* representada pelo código 4.6 é responsável por orquestrar a interação entre dois serviços distintos: o serviço de reconhecimento de fala (*SpeechToText*) e o serviço de linguagem natural (*LlmService*). A seguir, são descritas as principais funções da classe:

- **Método *OfAudioRecording*:** Este método é responsável por realizar a transcrição do áudio e gerar um resumo. O processo é dividido em duas etapas principais:
  1. **Reconhecimento de Fala:** A primeira etapa envolve a chamada ao método *recognizeSpeech* do serviço *SpeechToText*. Este método converte o áudio, localizado no caminho especificado por *audioFilePath*, em texto.
  2. **Geração do Resumo:** Com o texto obtido, o método *getResumeOfTextInListFormat* do serviço *LlmService* é chamado para gerar um resumo em formato de lista. Este resumo é retornado como um array de strings.

A *GenerateAppointmentResume* implementa o padrão de projeto **Facade** ao simplificar a interação entre os serviços complexos de reconhecimento de fala e processamento de linguagem natural. O padrão Facade proporciona uma interface simplificada para os consumidores desses serviços, escondendo a complexidade envolvida em suas operações (FOWLER, 2003b). A classe oferece uma forma única e coesa de acessar as funcionalidades dos serviços de reconhecimento de fala e linguagem natural, sem que o cliente precise lidar diretamente com a complexidade de cada serviço individualmente.

Este design permite que o usuário da *GenerateAppointmentResume* execute a transcrição e resumo de áudio com uma única chamada ao método *OfAudioRecording*, tornando a integração com esses serviços mais intuitiva e menos propensa a erros.

4.2.4 Resumo

Este Recorte de codigo demonstra como o sistema integra serviços de reconhecimento de fala e modelos de linguagem avançados para transcrever e resumir consultas médicas. A classe *GenerateAppointmentResume* é o núcleo deste processo, coordenando as transcrições e resumos utilizando os serviços *WhisperService* e *ChatGptLLMService*. Este mecanismo automatiza a criação de resumos de consultas, melhorando a eficiência e a precisão na documentação médica.

4.3 Documentação do Contrato da API

A documentação do contrato da API foi realizada utilizando o OpenAPI 3.0, um padrão amplamente adotado para descrever e documentar APIs RESTful . OpenAPI 3.0 permite que desenvolvedores definam todos os aspectos de uma API, como seus caminhos, métodos HTTP, parâmetros, respostas e outros detalhes essenciais (OPENAPI SPECIFICATION VERSION 3.0.3, 2020). Essa documentação serve tanto como uma referência para desenvolvedores que consomem a API quanto como uma base para a geração automática de código cliente e servidor.

O uso do OpenAPI 3.0 facilita a comunicação entre diferentes equipes e partes interessadas no projeto, além de garantir que a API esteja bem documentada e seja facilmente compreendida e integrada por desenvolvedores externos. O contrato da API desenvolvido neste projeto foi documentado utilizando um arquivo YAML, que pode ser acessado pelo link `https://matheusmacielleao.github.io/iassistente-consultas/`. Essa página foi gerada a partir de um arquivo YAML que contém a especificação completa da API, descrevendo detalhadamente cada endpoint e suas funcionalidades.

5 TESTES

Neste capítulo, será descrito a escolha de dois materiais para testes. Esses testes foram aplicados e nas duas versões do algoritmo de transcrição e resumo, disponíveis nas seções 4.1 e 4.2. Além disso será apresentada a avaliação de um profissional de saúde que analisou os resultados e uma comparação de performance e custo. Vale ressaltar que logo após o desenvolvimento da primeira versão do algoritmo 4.1, os dois materiais de teste já foram escolhidos e usados para validá-la. O motivo dessa decisão foi para pegar de antemão possíveis problemas de performance que seriam posteriormente levados em consideração ao implementar a versão final 4.2. Sendo assim, apresentaremos primeiro os materiais de teste, logo após mostraremos os resultados desses testes na primeira versão 4.1 e o logo após o mesmo na versão final 4.2, gerando uma comparação justa e mostrando como as evoluções afetaram a performance.

5.1 Materiais de teste

A seguir serão apresentados dois materiais usados para teste: um vídeo de uma consulta simulada feito em um trabalho de universidade e uma gravação de uma consulta particular, com autorização prévia do profissional de saúde, que também se dispôs a analisar os resultados.

5.1.1 Teste 1: Video de Consulta Simulada

O vídeo utilizado para o teste foi extraído de uma gravação intitulada *"Uma Primeira Consulta Médica Simulada na Iniciação ao Exame Clínico"*(SOUSA MUNOZ, 2021), disponível no YouTube. O vídeo possui 11:24 minutos e tem a seguinte descrição: "Neste vídeo, os estudantes de graduação em medicina da UFPB, Gabriel Fernando Vasconcelos Teles e Iasmin Nunes Duarte, da turma 111, realizaram a primeira simulação de uma consulta ambulatorial na sua segunda semana na disciplina de Semiologia Médica, após uma aula teórica sobre consulta médica e comunicação com o paciente. Paciente simulada por Iasmin foi uma mulher de 49 anos com queixa de dor torácica. Na simulação, orientei Gabriel e Iasmin a não incluírem o exame físico. A gravação foi realizada por Thomaz Feijó de Albuquerque, da mesma turma, e com o público dos demais colegas da turma 111, em uma sala de aula do Centro de Ciências Médicas da UFPB."

O arquivo MP3 gerado a partir deste vídeo foi utilizado para testes pois é uma simulação fiel feita por um profissional da saúde e seus alunos de uma universidade federal. Além disso possui qualidade de áudio satisfatória e uma duração curta, sendo perfeito para os primeiros testes do algoritmo de transcrição e suas integrações com as APIs da OpenAi.

5.1.2 Teste 2: Gravação de Consulta Real

Para um teste com parametros diferentes, uma gravação de uma consulta particular real foi feita com autorização do profissional de saúde. Para gravar o áudio foi utilizado um Iphone 13 Mini, usando app Recorder configurado em qualidade máxima, posicionado na mesa de centro da sala, que ficava entre a cadeira e a do profissional. A gravação terminou com um arquivo de 33:15 minutos. A duração muito maior, o equipamento simples e a distancia entre as pessoas e o equipamento foram fatores para escolher esse teste, simulando condições menos favoráveis do que a do teste anterior 5.1.1.

5.2 Testes Executados no Primeiro Algoritmo

Para validar o desempenho do primeiro algoritmo, após a execução dos testes 5.1.1 e 5.1.2 foram extraídos e analisados o tamanho do arquivo, duração do áudio, custo da API OpenAI e tempo de execução. A Tabela 5.1 apresenta os resultados desses testes, detalhando cada uma das métricas avaliadas.

O primeiro teste 5.1.1 foi conduzido com um arquivo de áudio de 11 minutos e 24 segundos, ocupando 13,4 MB de espaço. O custo associado ao uso da API OpenAI para transcrição foi de 0,07 centavos de dólar, e o tempo total de execução do processo foi de 57,58 segundos.

O resumo gerado pelo retorno foi:

- Gabriel, médico especializado em física médica, recebe Patrícia, de 41 anos, com dor no peito.
- Patrícia relata que a dor começou durante uma caminhada e persistiu ao voltar para casa.
- Já teve episódios semelhantes no passado, mas agora a dor é mais intensa.
- Patrícia relata histórico familiar de problemas cardíacos, como o pai que teve um ataque cardíaco.
- A dor é descrita como uma pressão constante no peito, atingindo um nível de intensidade 8 em uma escala de 1 a 10.
- A dor piora ao realizar esforços e melhora quando descansa.
- Patrícia não apresenta outros sintomas, como febre, falta de ar ou inchaço.
- Alimentação saudável, exceto por consumo excessivo de sal.
- Vai e volta do trabalho caminhando, cerca de um quilômetro.
- Separada há dois anos, sem contato com o ex-marido.
- Gabriel recomenda exames cardíacos, como eletrocardiograma e exames laboratoriais para investigar a causa da dor.
- Patrícia concorda em realizar os exames na clínica de Gabriel, agradece e sai.

Tabela 5.1: Testes Executados no Primeiro Algoritmo

Teste	Duração do Áudio	Tamanho do Arquivo	Custo Total da OpenAI	Tempo de execução
Teste1 5.1.1	11:24m	13.4mb	0.07 centavos de dólar	57.58 segundos
Teste2 5.1.1	33:37m	32mb	n/a	n/a

No segundo teste 5.1.2, foi utilizado um arquivo de áudio de 33 minutos e 37 segundos, com tamanho de 32 MB. No entanto, esse teste falhou porque a API Whisper tem um limite de 25 MB por chamada, tornando impossível aferir resultados para este arquivo.

Após a análise completa do vídeo pelo profissional de saúde, constatou-se que tanto a transcrição quanto o resumo gerados foram satisfatórios, refletindo com precisão o conteúdo do áudio. A validação do Primeiro Algoritmo desempenhou um papel crucial, pois além de confirmar a viabilidade da proposta do sistema, também revelou um ponto crítico relacionado à performance, onde problemas de velocidade e restrições apresentadas pelos resultados do segundo teste foram identificados. Esse insight foi essencial para direcionar os esforços futuros no desenvolvimento do sistema. A partir dessas descobertas, foi possível focar na otimização da velocidade e da eficiência ao desenvolver o algoritmo final 4.2, garantindo que a solução final, ofereça uma experiência mais ágil e satisfatória para o usuário.

5.3 Testes Executados no Algoritmo Final

Para validar o desempenho do algoritmo final, foram realizados testes detalhados para avaliar o impacto das melhorias implementadas. Esses testes também foram conduzidos para avaliar o tamanho do arquivo, a duração do áudio, o custo da API OpenAI e o tempo de execução. A Tabela 5.2 apresenta os resultados desses testes, detalhando cada uma das métricas avaliadas.

O primeiro teste 5.1.1 no algoritmo final foi conduzido com um arquivo de áudio de 11 minutos e 24 segundos, ocupando 13,4 MB de espaço. O custo associado ao uso da API OpenAI para transcrição foi de 0,07 centavos de dólar, e o tempo total de execução foi reduzido para 14,58 segundos. Essa redução no tempo de execução em comparação com os testes anteriores reflete a eficácia das melhorias implementadas no serviço de reconhecimento de fala.

O resumo gerado pelo retorno foi:

- Gabriel, médico especializado em física médica, recebe Patrícia, de 41 anos, com dor no peito.
- Patrícia relata que a dor começou durante uma caminhada e persistiu ao voltar para casa.
- Já teve episódios semelhantes no passado, mas agora a dor é mais intensa.
- Patrícia relata histórico familiar de problemas cardíacos, como o pai que teve um ataque cardíaco.
- A dor é descrita como uma pressão constante no peito, atingindo um nível de intensidade 8 em uma escala de 1 a 10.
- A dor piora ao realizar esforços e melhora quando descansa.

- Patrícia não apresenta outros sintomas, como febre, falta de ar ou inchaço.
- Alimentação saudável, exceto por consumo excessivo de sal.
- Vai e volta do trabalho caminhando, cerca de um quilômetro.
- Separada há dois anos, sem contato com o ex-marido.
- Gabriel recomenda exames cardíacos, como eletrocardiograma e exames laboratoriais para investigar a causa da dor.
- Patrícia concorda em realizar os exames na clínica de Gabriel, agradece e sai.

O segundo teste 5.1.2 foi realizado com um arquivo de áudio de 33 minutos e 37 segundos, com tamanho de 32 MB. O custo total para a transcrição foi de 0,21 centavos de dólar, e o tempo de execução foi de 36,5 segundos. Embora este teste tenha exigido mais chamadas à API devido ao tamanho do arquivo, o custo total permaneceu consistente com o teste anterior. A eficiência do novo serviço em lidar com arquivos grandes ao dividi-los e processá-los assíncronamente foi uma melhoria significativa em comparação com os testes do algoritmo anterior. Por motivos de privacidade não foi incluído o resultado do resumo.

Além dos resultados quantitativos, o feedback qualitativo do profissional de saúde que participou dos testes foi extremamente positivo. Após realizar a consulta e autorizar a gravação do teste 2 5.1.2, o profissional comentou: "Parece uma boa ferramenta pra lembrar de alguns detalhes que posso esquecer de anotar e focar mais na consulta em si. Gostei muito e com certeza usaria no dia a dia". Este feedback confirma que a ferramenta não só atendeu aos requisitos técnicos, mas também proporcionou um valor real e prático para o usuário final, reforçando o sucesso da solução desenvolvida.

Tabela 5.2: Testes Executados no Algoritmo Final

Teste	Duração do Áudio	Tamanho do Arquivo	Custo Total da OpenAI	Tempo de execução
Teste1 5.1.1	11:24m	13.4mb	0.07 centavos de dólar	14.58 segundos
Teste2 5.1.2	33:37m	32mb	0.21 centavos de dólar	36.5 segundos

Após a análise dos resultados apresentados na Tabela 5.2, foi possível observar uma evolução significativa em relação aos testes do algoritmo anterior. A implementação da classe 4.2.1 *WhisperService*, e principalmente ao código 4.2 não só conseguiu lidar com arquivos maiores ao dividi-los em partes menores e processá-los de maneira assíncrona, como também melhorou o tempo de execução. Importante destacar que, apesar do aumento no número de chamadas à API,o custo total ao executar o Teste 5.1.1 permaneceu o mesmo, mostrando a eficiência do serviço em manter o custo baixo enquanto melhora o desempenho. Esses resultados confirmam a eficácia das mudanças implementadas e destacam a importância da otimização contínua para alcançar um desempenho ideal. A partir dessas descobertas, os esforços futuros podem ser direcionados para ajustes adicionais e aprimoramentos na solução final, garantindo uma experiência mais eficiente e satisfatória para o usuário.

5.4 Comparação dos Resultados dos Testes

Para uma avaliação mais completa das melhorias implementadas, a Tabela 5.3 compara os resultados dos testes do primeiro algoritmo com os resultados do algoritmo final.

Esta comparação é fundamental para destacar a eficácia das otimizações realizadas e como estas abordaram os problemas identificados anteriormente.

Tabela 5.3: Comparação dos Resultados dos Testes do Algoritmo Inicial e Final

Teste	Tamanho do Arquivo	Duração do Áudio	Custo Total API OpenAI	Tempo de Execução (Inicial)	Tempo de Execução (Final)	Redução no Tempo de Execução
Teste1	13.4 MB	11:24 m	0.07 centavos de dólar	57.58 segundos	14.58 segundos	74.7%
Teste2	32 MB	33:37 m	0.21 centavos de dólar	n/a	36.5 segundos	-

Na Tabela 5.3, são apresentadas as comparações entre os resultados obtidos nos testes do algoritmo inicial e os resultados do algoritmo final. A análise dos dados revela as seguintes conclusões:

- Teste 1: O primeiro teste com o algoritmo inicial teve um tempo de execução de 57,58 segundos para um arquivo de 13,4 MB. Com a implementação do algoritmo final, o tempo de execução foi reduzido para 14,58 segundos, representando uma diminuição significativa de 74,7%. Essa melhoria destaca a eficácia das otimizações feitas, que resultaram em um desempenho mais ágil e eficiente.

- Teste 2: O segundo teste no algoritmo inicial falhou devido ao limite de 25 MB por chamada da API Whisper, tornando impossível obter resultados. No algoritmo final, o teste foi bem-sucedido com um arquivo de 32 MB, com um custo total de 0,21 centavos de dólar e um tempo de execução de 36,5 segundos. Isso demonstra que o problema foi resolvido através da divisão dos arquivos e processamento assíncrono, permitindo lidar com arquivos maiores sem comprometer o custo.

6 CONSIDERAÇÕES FINAIS

O principal objetivo deste trabalho foi desenvolver uma solução inovadora para a transcrição e resumo de consultas médicas, utilizando tecnologias de inteligência artificial, com ênfase no modelo Whisper da OpenAI.

O processo de desenvolvimento do *Iassistente de Consultas* iniciou-se com a definição clara dos objetivos, seguida por um refinamento criterioso através da elicitação de requisitos com profissionais de saúde. A contribuição desses especialistas foi fundamental para garantir que todos os aspectos essenciais fossem contemplados. Com base nesses requisitos, o sistema foi modelado, utilizando diagramas de caso de uso, classes, sequência e entidade-relacionamento, além de pesquisas detalhadas para a seleção das ferramentas ideais.

As ferramentas selecionadas foram implementadas e testadas em uma primeira versão, que, após uma análise crítica, evoluiu para uma versão mais robusta. A nova versão foi submetida a testes adicionais, cujos resultados confirmaram a eficácia do sistema.

Com base nos resultados apresentados e nas análises realizadas, é possível afirmar que o objetivo principal foi atingido com sucesso. O sistema desenvolvido provou ser uma ferramenta útil e precisa para auxiliar os profissionais de saúde na documentação das consultas. No entanto, foram identificadas algumas limitações que abrem caminho para possíveis melhorias e expansões em trabalhos futuros.

Durante o desenvolvimento e testes, algumas limitações foram identificadas:

- **Qualidade do equipamento de captura de áudio e ambiente:** A qualidade do microfone e o ambiente em que a gravação é realizada podem impactar significativamente o resultado da transcrição. Ruídos de fundo, eco ou distância dos participantes ao microfone podem reduzir a precisão do reconhecimento de fala.

- **Custo de uso e dependência de APIs externas:** A dependência de APIs externas, como a da OpenAI, pode ser um obstáculo, especialmente em termos de custo, o que pode inviabilizar a implantação em sistemas públicos de saúde.

- **Corte de áudio em momentos inadequados:** A versão atual do sistema foi programada para aceitar apenas arquivos no formato MP3 e corta o áudio em intervalos predeterminados. Isso pode afetar o resultado da transcrição, especialmente se o corte ocorrer no meio de uma fala importante.

Com base nas limitações identificadas, algumas direções para trabalhos futuros foram propostas:

- **Compatibilidade com outros formatos de áudio:** Expandir a compatibilidade do sistema para aceitar diferentes formatos de arquivos de áudio além do MP3.

- **Padrão Adapter para classes de IA:** Implementar o padrão Adapter para as classes de IA, permitindo uma configuração mais flexível e o uso de modelos de IA locais, como as versões abertas do Whisper e LLaMA3, dependendo do ambiente de execução.(AI, 2024)
- **Melhoria na análise de áudio:** Aprimorar a análise de áudio para lidar melhor com ruídos, bem como implementar um algoritmo de corte que considere a onda sonora, cortando o áudio apenas em momentos de pausa na conversa.

6.1 Conclusão

Em conclusão, o objetivo proposto neste projeto foi plenamente atingido. A solução desenvolvida mostrou-se eficaz na transcrição e resumo de consultas médicas, oferecendo uma ferramenta que pode melhorar a prática clínica ao fornecer transcrições precisas e detalhadas. As limitações identificadas apontam para áreas de melhoria que podem ser exploradas em trabalhos futuros, reforçando o potencial da solução proposta para se tornar uma ferramenta robusta e amplamente aplicável.

REFERÊNCIAS

AI, M. **LLaMA 3**: large language model meta ai. Acessado em 02/09/2024, `https://ai.meta.com/llama/`.

BUN. **Bun - A fast all-in-one JavaScript runtime**. Disponível em https://bun.sh, acessado em 06/06/2024.

BUN. **Bun Benchmarks**. Disponível em https://github.com/oven-sh/bun/tree/main/bench, acessado em 10/07/2024.

CORPORATION, O. **MySQL**. [S.l.: s.n.], 1995. https://www.mysql.com.

DEVELOPERS, F. **FFmpeg**. [S.l.: s.n.], 2000. https://ffmpeg.org.

FOWLER, M. **Patterns of Enterprise Application Architecture**. [S.l.]: Addison-Wesley, 2003.

FOWLER, M. **Padrões de Arquitetura de Aplicações Corporativas**. [S.l.]: Addison-Wesley, 2003. Tradução em Português.

FOWLER, M. **Inversion of Control Containers and the Dependency Injection pattern**. Accessed: 2024-09-01, `https://martinfowler.com/articles/injection.html`.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns**: elements of reusable object-oriented software. [S.l.]: Addison-Wesley, 1994.

GROUP, T. P. G. D. **PostgreSQL**. [S.l.: s.n.], 1996. https://www.postgresql.org.

HIPP, D. R. **SQLite**. [S.l.: s.n.], 2000. https://www.sqlite.org.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing**: an introduction to natural language processing, computational linguistics, and speech recognition. 3rd.ed. London, UK: Pearson, 2023.

MARTIN, R. C. **Clean Architecture**: a craftsman's guide to software structure and design. Upper Saddle River, NJ: Prentice Hall, 2017.

OPENAI. **GPT (Generative Pre-trained Transformer)**. Accessed: 2024-01-07.

OPENAI. **Whisper (Automatic Speech Recognition)**. Accessed: 2024-01-07.

OPENAI. **OpenAI APIs**. Accessed: 2024-01-07.

OPENAPI Specification Version 3.0.3. [S.l.]: OpenAPI Initiative, 2020.

SOUSA MUNOZ, P. R. L. de. **Uma Primeira Consulta Médica Simulada na Iniciação ao Exame Clínico**. Acesso em: 05/05/2024.