HENRIQUE RIBEIRO DA SILVA ALMEIDA

ARQUITETURA DE RECONHECIMENTO DE SENTIMENTOS EM AMBIENTES WEB: UM ESTUDO APLICADO À CONEXÃO ENTRE PACIENTES E PROFISSIONAIS DE SAÚDE MENTAL

Monografia do trabalho de conclusão de curso apresentado ao Instituto Federal Minas Gerais - Campus Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Bruno Ferreira

Formiga - MG

2026

Almeida, Henrique Ribeiro da Silva
A447u Um estudo aplicado à conexão entre pacientes e profissionais de saúde mental / Henrique Ribeiro da Silva Almeida - Formiga : IFMG, 2026.
172 p. :il. color.

Orientador: Bruno Ferreira

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. Saúde mental. 2. Plataforma web. 3. Arquitetura de software. 4. Inteligência artificial. 5. Visão computacional I. Ferreira, Bruno. II. Título.

CDD 362.2

Ficha catalográfica elaborada pela Bibliotecária Lívia Renata Santos- CRB/6-2561

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS
Campus Formiga
Diretoria de Ensino
Docência Área Acadêmica de Computação
Rua São Luiz Gonzaga, s/n - Bairro São Luiz - CEP 35570-000 - Formiga - MG
- www.ifmg.edu.br

FOLHA DE APROVAÇÃO

Henrique Ribeiro da Silva Almeida

Arquitetura de Reconhecimento de Sentimentos em Ambientes Web: Um Estudo Aplicado à Conexão entre Pacientes e Profissionais de Saúde Mental

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Minas Gerais - Campus Formiga, como Requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Aprovado em: 22 de outubro de 2026.

BANCA EXAMINADORA

Prof.º Dr. Bruno Ferreira

Prof.º Me. Everthon Valadão dos Santos

Prof.º Me. Roger Santos Ferreira

Formiga, 22 de outubro de 2026.

Documento assinado eletronicamente por **Bruno Ferreira**, **Professor**, em 18/11/2026, às 13:33, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Everthon Valadao dos Santos**, **Professor**, em 18/11/2026, às 13:55, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Roger Santos Ferreira**, **Professor**, em 18/11/2026, às 14:21, conforme Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade do documento pode ser conferida no site https://sei.ifmg.edu.br/consultadocs informando o código verificador **2913566** e o código CRC **F9E3A383**.

23211.001176/2025-65	2913566v1

Agradecimentos

Agradeço, primeiramente, a Deus, por me conceder força, saúde e perseverança durante toda a trajetória acadêmica e, principalmente, nos momentos de maior dificuldade ao longo do desenvolvimento deste trabalho.

Aos meus familiares, pelo apoio, incentivo e compreensão durante os períodos de dedicação aos estudos e ao desenvolvimento deste projeto. Agradeço por todo o suporte oferecido ao longo dessa caminhada.

Aos meus professores, pelos conhecimentos compartilhados, pela orientação e pelas contribuições que foram fundamentais para minha formação acadêmica e profissional.

Ao meu orientador, pela orientação, pelas sugestões e pelo acompanhamento durante o desenvolvimento deste trabalho, contribuindo para o aprimoramento das decisões tomadas ao longo do projeto.

Agradeço também às duas profissionais da área de Psicologia que participaram das demonstrações da plataforma e contribuíram com suas percepções e sugestões. Suas considerações foram importantes para avaliar as ideias propostas e aproximar o desenvolvimento da solução das necessidades observadas no contexto da saúde mental.

Por fim, agradeço a todos que, de alguma forma, contribuíram para a realização deste trabalho, seja por meio de apoio, conhecimento, incentivo ou simplesmente pela compreensão durante essa etapa da minha formação.

"Tudo vale a pena se a alma não é pequena." (Fernando Pessoa)

Resumo

A crescente demanda por serviços de saúde mental evidencia a necessidade de soluções tecnológicas que facilitem a conexão entre pacientes e profissionais da área e ofereçam recursos de apoio ao acompanhamento psicológico. Nesse contexto, este trabalho teve como objetivo desenvolver o *TherapistFriend*, uma plataforma *web* que integra funcionalidades de comunicação, gerenciamento de atendimentos e inteligência artificial. O desenvolvimento foi realizado com *React* no *frontend* e *Node.js*, *Express* e *TypeScript* no *backend*, utilizando *PostgreSQL* para persistência dos dados e uma arquitetura orientada à separação de responsabilidades, fundamentada em *Domain-Driven Design* (DDD) e nos princípios *SOLID*. Para a comunicação em tempo real, foram empregados *Socket.IO* e *WebRTC*, com suporte de um servidor *Coturn* para o estabelecimento de conexões de áudio e vídeo. A solução também incorpora dois microsserviços de inteligência artificial: um serviço textual, desenvolvido em *Flask* e integrado ao *Ollama*, destinado à análise de relatos e à geração de recomendações personalizadas; e um serviço de visão computacional, baseado em *YOLOv8* e treinado com dados do *AffectNet* adaptados ao formato *YOLO*, destinado à detecção facial e à classificação de expressões faciais durante sessões *online*. A comunicação entre o *backend* e o serviço de visão computacional utiliza *gRPC*. Adicionalmente, foram implementados mecanismos de autenticação e gerenciamento de sessões com *JSON Web Tokens* (JWT) e *Refresh Tokens*, armazenamento de tokens em cookies *HTTPOnly*, criptografia AES-256 das mensagens e validação de e-mail e telefone com *Nodemailer* e *Twilio*. Como resultado, foi desenvolvida uma plataforma que reúne funcionalidades de publicação de relatos anônimos, comunicação privada entre pacientes e profissionais, gerenciamento de agendas, sessões *online* e geração de relatórios e recomendações com apoio de inteligência artificial. O trabalho apresenta a implementação de uma solução de *software* para esse contexto, sem pretender comprovar sua eficácia clínica ou seu impacto no acompanhamento psicológico. A avaliação sistemática de usabilidade e a investigação dos benefícios da plataforma para seus usuários constituem possibilidades de trabalhos futuros.

Palavras-chave: saúde mental; plataforma web; arquitetura de software; inteligência artificial; visão computacional.

Abstract

The growing demand for mental health services highlights the need for technological solutions that facilitate the connection between patients and professionals in the field, while offering support resources for psychological monitoring. In this context, this work aimed to develop *TherapistFriend*, a *web* platform that integrates communication features, appointment management, and artificial intelligence. Development was carried out using *React* on the *frontend* and *Node.js*, *Express*, and *TypeScript* on the *backend*, utilizing *PostgreSQL* for data persistence and an architecture oriented toward the separation of concerns, grounded in *Domain-Driven Design* (DDD) and *SOLID* principles. For real-time communication, *Socket.IO* and *WebRTC* were employed, supported by a *Coturn* server to establish audio and video connections. The solution also incorporates two artificial intelligence microservices: a textual service, developed in *Flask* and integrated with *Ollama*, aimed at analyzing user reports and generating personalized recommendations; and a computer vision service, based on *YOLOv8* and trained with *AffectNet* data adapted to the *YOLO* format, designed for facial detection and expression classification during *online* sessions. Communication between the *backend* and the computer vision service utilizes *gRPC*. Additionally, authentication and session management mechanisms were implemented using *JSON Web Tokens* (JWT) and *Refresh Tokens*, token storage in *HTTPOnly* cookies, AES-256 message encryption, and email and phone validation using *Nodemailer* and *Twilio*. As a result, a platform was developed that brings together features for publishing anonymous reports, private communication between patients and professionals, schedule management, *online* sessions, and the generation of reports and recommendations with the aid of artificial intelligence. This work presents the implementation of a *software* solution for this context, without aiming to prove its clinical efficacy or its impact on psychological monitoring. Systematic usability evaluation and the investigation of the platform's benefits for its users constitute avenues for future work.

Keywords: mental health; web platform; software architecture; artificial intelligence; computer vision.

Lista de ilustrações

Figura 1 – Exemplo de imagens usadas no dataset 40
Figura 2 – Esquema de funcionamento do AES-256 44
Figura 3 – Diagrama de estrutura da arquitetura geral do sistema 56
Figura 4 – Diagrama de casos de uso para os atores paciente e profissional. 67
Figura 5 – Diagrama de casos de uso para o ator administrador. 68
Figura 6 – DER da estrutura de usuários e autenticação 71
Figura 7 – DER da estrutura de especialidades e agendamento 73
Figura 8 – DER da estrutura de relatos e registro de humor 75
Figura 9 – DER da estrutura de notificações e chats 76
Figura 10 – DER da estrutura de sessões e relatórios 78
Figura 11 – Diagrama do fluxo de login e autenticação 79
Figura 12 – Diagrama do fluxo de cadastro 80
Figura 13 – Diagrama do fluxo de refresh token 82
Figura 14 – Diagrama do fluxo de relatos 83
Figura 15 – Diagrama do fluxo de solicitação de conversa 84
Figura 16 – Diagrama do fluxo de registro de humor e geração de cards 86
Figura 17 – Diagrama do fluxo de agendamento 87
Figura 18 – Diagrama do fluxo de sessões online 89
Figura 19 – Diagrama do fluxo de geração de relatórios 91
Figura 20 – Diagrama do fluxo de análise de emoções durante as sessões 93
Figura 21 – Estrutura da application 96
Figura 22 – Estrutura da domain 98
Figura 23 – Estrutura da infraestrutura 100
Figura 24 – Estrutura da inteface 101
Figura 25 – Ponto de Composição 102
Figura 26 – Estrutura de diretórios dos componentes reutilizáveis. 104
Figura 27 – Estrutura de diretórios das páginas do sistema. 104
Figura 28 – Arquitetura de estabelecimento da conexão WebRTC utilizando servidor Coturn. 116
Figura 29 – Tela de cadastro. 136
Figura 30 – Modais códigos. 136
Figura 31 – Tela de login. 137
Figura 32 – Menus Laterais. 138
Figura 33 – Dashboard do paciente. 139
Figura 34 – Dashboard do profissional. 140
Figura 35 – Dashboard do administrador. 141

Figura 36 – Relatos para pacientes. 142
Figura 37 – Relatos para profissionais. 142
Figura 38 – Modal explicativo sobre a análise de relatos da inteligência artificial. . . 143
Figura 39 – Chat para desktop. 144
Figura 40 – Chat para mobile. 144
Figura 41 – Tela de notificações. 145
Figura 42 – Agenda vista pelo profissional. 146
Figura 43 – Agenda vista pelo paciente. 147
Figura 44 – Perfil privado do profissional. 148
Figura 45 – Perfil público do profissional. 149
Figura 46 – Listagem de profissionais. 150
Figura 47 – Perfil completo via modal. 150
Figura 48 – Lista de relatórios. 151
Figura 49 – Relatório. 152
Figura 50 – Sala de espera. 153
Figura 51 – Sala de sessão. 153
Figura 52 – Curvas de perda (*loss*) e evolução das métricas de validação ao longo do treinamento. 155
Figura 53 – Matriz de confusão resultante do modelo YOLOv8n otimizado com entrada de 96×96 pixels. 157
Figura 54 – Matriz de confusão resultante do modelo YOLOv8n otimizado com entrada de 224×224 pixels. 158
Figura 55 – Curva F1-Confidence mapeando o ponto ótimo de equilíbrio preditivo. 161

Lista de tabelas

Tabela 1 – Requisitos Funcionais. 53
Tabela 2 – Requisitos Não Funcionais. 54
Tabela 3 – Comparativo de desempenho do modelo YOLO entre as resoluções 96x96 e 224x224. 63
Tabela 4 – Verificação dos requisitos funcionais por meio de testes manuais. 134
Tabela 5 – Cores utilizadas nas notificações. 145
Tabela 6 – Cores utilizadas na agenda. 148
Tabela 7 – Métricas de desempenho por classe 160

Lista de Códigos

4.1 Código exemplo de criação de um *worker* para consumir os *jobs* 110
4.2 Código exemplo de um *job* 111
4.3 Gerenciador centralizado de WebSockets (*SocketManager.ts*) 112
4.4 Gerenciamento e conexão controlada do Socket no frontend (*SocketContext.tsx*) 114
4.5 Caso de Uso para Provisionamento de Parâmetros WebRTC (*GerarAcessoSessaoUseCase.ts*) 115
4.6 Gateway de Sinalização WebRTC (*SessaoSignalingGateway.ts*) 117
4.7 Fluxo de Negociação de Chamada e Loop de Análise (*useVideoSessionEngine.ts*) 119
4.8 Gerenciamento de Pipeline gRPC e Eventos Clínicos (*SessaoAnalysisGateway.ts*) 121
4.9 *Prompt da análise dos relatos* 123
4.10 *Exemplo real do retorno da classificação do relato* 124
4.11 *Prompt da geração dos cards* 125
4.12 *Exemplo real do retorno do registro de humor* 126
4.13 Trecho do código referente ao *Fallback* 127
4.14 Configuração e Hiperparâmetros de Treinamento da Rede Facial (*train.py*) . . 128
4.15 Definição de Interface de Comunicação gRPC (*analisador.proto*) 130
4.16 Servidor de Inferência e Decodificação OpenCV (*grpc_server.py*) 131

Lista de abreviaturas e siglas

OMS	*Organização Mundial da Saúde*
WebRTC	*Web Real-Time Communication*
HTML	*HyperText Markup Language*
CSS	*Cascading style Sheets*
APIs	*Application Programming Interfaces*
HTTP	*HyperText Transfer Protocol*
JSON	*JavaScript Object Notation*
CNNs	*Redes Neurais Convolucionais*
TCO	*Transmission Control Protocol*
DOM	*Document Objetc Model*
DDD	*Domain-Driven Design*
DIP	*Dependency Inversion Principle*
URL	*Uniform Resource Locator*
SQL	*Structured Query Language*
NATs	*Network Address Translation*
STURN	*Session Traversal Utilities for NAT*
TURN	*Traversal Using Relays around NAT*
SDP	*Session Description Protocol*
IP	*Internet Protocol*
GPUs	*Graphics Processing Units*
VRAM	*Video Random Access Memory*
ReLU	*Rectified Linear Unit*
R-CNN	*Region-based Convolutional Neural Networks*

COCO	*Common Objects in Context*
XML	*eXtensible Markup Language*
SMS	*Short Message Service*
FK	*Foreign Key*
DER	*Diagrama de Entidade e Relacionamento*
UUID	*Universally Unique Identifier*
RBAC	*Role-Based Access Control*
CRP	*Conselho Regional de Psicologia*
CPF	*Cadastro de Pessoas Físicas*
CORS	*Cross-Origin Resource Sharing*
IDL	*Interface Definition Language*
RAM	*Random Access Memory*
TDICs	*Tecnologia Digital da Informação e da Comunicação*
LGPD	*Lei Geral de Proteção de Dados Pessoais*
FER	*Facial Expression Recognition*

Sumário

1 INTRODUÇÃO 21
1.1 Justificativa 22
1.2 Objetivos 22
1.2.1 Objetivo Geral 22
1.2.2 Objetivos Específicos 23

2 FUNDAMENTAÇÃO TEÓRICA 24
2.1 Psicologia e Saúde Mental 24
2.2 Uso de Tecnologia na Psicologia 25
2.2.1 Telepsicologia e Atendimento Psicológico Remoto 26
2.2.2 Regulamentação da Telepsicologia no Brasil 26
2.3 Desenvolvimento de Sistemas Web 27
2.3.1 Frontend 27
2.3.1.1 REACT 28
2.4 Backend 28
2.4.0.1 NODE.JS e Express 29
2.5 Banco de dados PostgreSQL 29
2.6 Containerização e Virtualização com Docker 30
2.7 Comunicação em Sistemas Distribuídos e Tempo Real 30
2.7.1 WebSocket 30
2.7.1.1 O Handshake e a Conexão 31
2.7.2 WebRTC 31
2.7.2.1 Pilares Tecnológicos do WebRTC 32
2.7.2.2 O Processo de Sinalização e Travessia de Rede 32
2.7.3 gRPC 33
2.7.3.1 Protocol Buffers e HTTP/2 33
2.8 Inteligência Artificial 33
2.9 Inteligência Artificial Generativa 34
2.9.1 Princípios de Modelagem Generativa 34
2.9.2 Arquiteturas Fundamentais 34
2.9.3 Modelos de Linguagem de Grande Escala(LLM) 35
2.9.4 Arquitetura Baseada em Transformers 35
2.9.4.1 Mecanismo de Autoatenção (Self-Attention) 35
2.9.4.2 Componentes do Transformer (Encoder/Decoder) 35
2.9.5 Modelos de Linguagem Aplicados à Saúde Mental: Estado da Arte e Limitações 36

2.10 Ollama . . . 37
2.11 O Modelo de Linguagem Llama 3 . . . 37
2.11.0.1 Inovações e Performance . . . 38
2.12 Visão Computacional e Visão Neural . . . 38
2.12.1 Redes Neurais Convolucionais (CNNs) . . . 39
2.12.1.1 Camadas de Convolução e Filtros . . . 39
2.12.1.2 Camadas de Pooling e Funções de Ativação . . . 39
2.12.2 Paradigmas de Detecção: Do R-CNN ao YOLO . . . 39
2.12.3 YOLOv8 para Detecção Facial . . . 40
2.12.3.1 O Dataset AffectNet em Formato YOLO . . . 40
2.12.4 Estado da Arte no Reconhecimento de Expressões Faciais . . . 41
2.12.5 Limitações e Desafios na Inferência de Estados Emocionais . . . 42
2.13 Mensageria e Processamento Assíncrono . . . 42
2.14 Segurança da Informação . . . 43
2.14.1 Criptografia de Dados com AES-256 . . . 43
2.14.2 Hash de Senhas . . . 45
2.14.3 Autenticação e Controle de Acesso . . . 46
2.14.4 Privacidade e Proteção de Dados Pessoais na Saúde Digital . . . 46
2.15 Metodologias Ágeis de Desenvolvimento . . . 47
2.16 Arquitetura de Software e Boas Práticas . . . 47
2.17 Trabalhos Relacionados . . . 48

3 MATERIAIS E MÉTODO . . . 50
3.1 Equipamento . . . 50
3.2 Caracterização da Pesquisa . . . 50
3.3 Problema de Pesquisa e Objetivos . . . 51
3.4 Levantamento e Especificação dos Requisitos . . . 52
3.4.1 Requisitos Funcionais . . . 52
3.4.2 Requisitos Não Funcionais . . . 53
3.5 Procedimentos Metodológicos . . . 54
3.6 Arquitetura Geral do Sistema . . . 55
3.7 Backend Principal . . . 57
3.8 Banco de Dados . . . 58
3.8.1 Modelagem Relacional . . . 58
3.8.2 Estrutura das Entidades . . . 58
3.8.3 Relacionamentos . . . 59
3.8.4 Persistência e Integridade dos Dados . . . 59
3.9 Comunicação em Tempo Real e Processamento Assíncrono . . . 59
3.10 Frontend da Aplicação . . . 60
3.11 Microsserviço de Inteligência Artificial Textual . . . 61

3.12 Microsserviço de Visão Computacional . . . 62
3.12.1 Processo de Escolha do Modelo . . . 62
3.13 Segurança da Aplicação . . . 63
3.13.1 Verificação e Validação de Usuários . . . 64
3.14 Testes da Aplicação . . . 65
3.15 Containerização da Aplicação . . . 66

4 DESENVOLVIMENTO . . . 67
4.1 Casos de uso . . . 67
4.1.1 Casos de uso do paciente . . . 68
4.1.2 Casos de uso do profissional . . . 69
4.1.3 Casos de uso do administrador . . . 70
4.2 Banco de Dados . . . 70
4.2.1 Estrutura de Usuários e Autenticação . . . 70
4.2.2 Estrutura de Especialidades e Agendamento . . . 72
4.2.2.1 Estrutura de Relatos e Registro de Humor . . . 74
4.2.2.2 Estrutura de Notificações e Chats . . . 76
4.2.2.3 Estrutura de sessões e relatórios . . . 77
4.3 Fluxo de Cadastro e Autenticação . . . 79
4.4 Fluxo de Relatos . . . 82
4.5 Fluxo de solicitação de conversa . . . 84
4.6 Fluxo de Registro de Humor e Geração de Cards . . . 85
4.6.1 Fluxo de Agendamento . . . 87
4.7 Fluxo de Sessões Online . . . 88
4.8 Fluxo de Geração de Relatórios . . . 90
4.9 Fluxo de Análise de Emoções durante as sessões . . . 92
4.10 Desenvolvimento do Backend Principal . . . 94
4.10.1 Detalhes da implementação do DDD . . . 95
4.10.1.1 Camada de Aplicação (Application) . . . 95
4.10.1.2 Camada de Domínio (Domain) . . . 97
4.10.1.3 Camada de Infraestrutura (Infrastructure) . . . 98
4.10.1.4 Camada de Interface (Interface) . . . 100
4.10.1.5 Ponto de Composição (Main / Factories) . . . 101
4.11 Desenvolvimento do Frontend . . . 102
4.11.1 Princípios de Design e Gerenciamento de Estado . . . 102
4.11.2 Telas e Componentes . . . 103
4.11.2.1 Construção da Agenda . . . 107
4.11.3 Chat e suas funcionalidades . . . 107
4.12 Desenvolvimento da Mensageria . . . 109
4.12.1 Fluxo de Disparo a partir dos Casos de Uso . . . 111

4.13 Desenvolvimento do Sistema de Comunicação em Tempo Real . . . 111
4.13.1 Arquitetura e Gerenciamento de Conexões com Socket.io . . . 112
4.13.1.1 Orquestração e Segurança no Backend . . . 112
4.13.1.2 Sincronização e Ciclo de Vida Reativo no Frontend . . . 113
4.13.2 Mecanismo de Sinalização e Transmissão Multimídia com WebRTC . . . 115
4.13.2.1 Governança de Acesso e Provisionamento de Rede (STUN/TURN) . . . 115
4.13.2.2 Orquestração do Fluxo de Sinalização via Gateway . . . 117
4.13.2.3 Implementação do Cliente WebRTC e Consumo de Telemetria com IA . . . 118
4.13.2.4 Loop de Extração de Frames e Feedback de Telemetria . . . 119
4.13.2.5 Pipeline de Processamento de Telemetria e Resiliência no Backend . . . 120
4.14 Desenvolvimento do Microsserviço de Inteligência Artificial Textual 122
4.14.0.1 Análise de Engenharia de Prompts e Parametria de Hiperparâmetros . . . 123
4.14.0.2 Tratamento de Exceções e Estratégia de Fallback Estruturado . . . 127
4.15 Desenvolvimento do Microsserviço de Visão Computacional . . . 128
4.15.0.1 Pipeline de Treinamento e Otimização do Modelo YOLOv8 . . . 128
4.15.0.2 Arquitetura do Contrato gRPC e Camada de Serialização . . . 130
4.15.0.3 Construção do Servidor de Inicialização e Inferência . . . 130

5 RESULTADOS E ANÁLISE . . . 133
5.1 Validação dos Requisitos Funcionais . . . 133
5.2 Interfaces Desenvolvidas . . . 135
5.2.1 Cadastro . . . 135
5.2.2 Login . . . 137
5.2.3 Menu Lateral . . . 137
5.2.4 Dashboard do Paciente . . . 138
5.2.5 Dashboard do Profissional . . . 139
5.2.6 Dashboard Administrador . . . 140
5.2.7 Relatos . . . 141
5.2.8 Chats . . . 143
5.2.9 Notificações . . . 145
5.2.10 Agenda . . . 146
5.2.11 Perfil . . . 147
5.2.12 Explorar . . . 149
5.2.13 Relatórios . . . 151
5.2.14 Sessão . . . 152
5.3 Resultados do Treinamento do Modelo de Visão Computacional . . . 154
5.3.1 Análise Genérica de Convergência e Funções de Perda . . . 154
5.3.2 Matriz de Confusão e Avaliação de Desempenho . . . 156
5.3.2.1 Cálculo das Métricas por Classe . . . 159
5.3.3 Curvas de Sensibilidade Paramétrica e Ponto de Operação . . . 160

5.3.4 Discussão dos Resultados 161

6 CONSIDERAÇÕES FINAIS 163
6.1 Considerações sobre o Desenvolvimento da Plataforma 163
6.2 Considerações sobre as Tecnologias Selecionadas 164
6.3 Validação Conceitual da Plataforma 165
6.4 Limitações do Trabalho 166
6.5 Trabalhos Futuros 166

REFERÊNCIAS 168

1 Introdução

A saúde mental tem se tornado um tema cada vez mais relevante na sociedade contemporânea, especialmente diante do aumento de casos de ansiedade, depressão e outros transtornos psicológicos. De acordo com a Organização Mundial da Saúde (OMS), milhões de pessoas em todo o mundo sofrem com problemas relacionados à saúde mental, sendo que uma parcela significativa não recebe o tratamento adequado. Entre os principais fatores que dificultam o acesso ao acompanhamento psicológico estão o estigma social, a falta de informação e as barreiras de acesso aos serviços especializados (World Health Organization, 2025).

Nesse contexto, a tecnologia surge como uma aliada importante na ampliação do acesso à saúde, especialmente por meio de sistemas web que permitem a comunicação remota entre pacientes e profissionais. Plataformas digitais têm possibilitado novas formas de interação, promovendo maior flexibilidade, acessibilidade e alcance geográfico. No entanto, ainda existem desafios relacionados à confiança, privacidade e ao conforto emocional dos usuários ao compartilhar questões sensíveis (FIGUEROA, 2018).

Um dos principais obstáculos enfrentados por indivíduos que necessitam de apoio psicológico é o receio de julgamento ao expor seus sentimentos e experiências. Nesse sentido, o anonimato pode desempenhar um papel fundamental, proporcionando um ambiente mais seguro para que os usuários se expressem livremente. A possibilidade de relatar problemas de forma anônima pode reduzir barreiras iniciais e incentivar a busca por ajuda profissional (MARSHALL et al., 2024).

Diante desse cenário, este trabalho propõe o desenvolvimento de um sistema web voltado à conexão entre pacientes e profissionais da saúde mental, com foco na comunicação anônima inicial. A plataforma permite que usuários publiquem relatos de forma anônima, possibilitando que profissionais qualificados possam interagir e iniciar um atendimento, que pode evoluir para conversas privadas e, posteriormente, para sessões mais estruturadas.

Além disso, o sistema integra tecnologias modernas, como comunicação em tempo real por meio de *WebSockets*, videochamadas utilizando *WebRTC* e mecanismos de análise emocional baseados em inteligência artificial, com o objetivo de enriquecer a experiência do usuário e oferecer suporte mais eficiente. Essas funcionalidades buscam não apenas facilitar o contato entre paciente e profissional, mas também contribuir para uma melhor compreensão do estado emocional do usuário.

Portanto, este trabalho tem como objetivo desenvolver uma plataforma web para conectar pacientes e profissionais da saúde mental, integrando recursos de comunicação em tempo real, inteligência artificial e mecanismos de segurança. A proposta busca demonstrar

a aplicação dessas tecnologias no desenvolvimento de uma solução de *software* voltada ao contexto do acompanhamento psicológico remoto, explorando sua integração para implementar funcionalidades de comunicação, gerenciamento de atendimentos e apoio por inteligência artificial.

1.1 Justificativa

O aumento dos casos relacionados a transtornos mentais, como ansiedade e depressão, tem evidenciado a necessidade de soluções que ampliem o acesso ao cuidado psicológico. Apesar da crescente conscientização sobre a importância da saúde mental, ainda existem diversas barreiras que dificultam a busca por acompanhamento profissional, tais como o estigma social, a limitação de recursos e o receio de exposição pessoal (CLEMENT et al., 2015).

Nesse contexto, muitos indivíduos evitam procurar ajuda por medo de julgamento ou por desconforto em compartilhar questões íntimas de forma direta. Essa realidade demonstra a importância de estratégias que reduzam tais barreiras, tornando o acesso ao suporte psicológico mais acessível e acolhedor (CLEMENT et al., 2015).

A utilização de tecnologias digitais, especialmente sistemas *web*, apresenta-se como uma alternativa viável para minimizar essas dificuldades. Plataformas online permitem maior alcance, flexibilidade e praticidade no contato entre pacientes e profissionais, possibilitando atendimentos remotos e facilitando o primeiro contato. Além disso, o uso do anonimato pode contribuir significativamente para que os usuários se sintam mais seguros ao relatar suas experiências, promovendo maior engajamento inicial (PRETORIUS; CHAMBERS; COYLE, 2019).

Dessa forma, justifica-se o desenvolvimento de um sistema que permita a comunicação anônima entre pacientes e profissionais da saúde mental, proporcionando um ambiente seguro e acessível. A proposta busca não apenas facilitar o primeiro contato, mas também criar um fluxo contínuo de interação que possa evoluir para atendimentos mais estruturados, contribuindo para a democratização do acesso à saúde mental por meio da tecnologia.

1.2 Objetivos

1.2.1 Objetivo Geral

Desenvolver um sistema web que possibilite a conexão entre pacientes e profissionais da saúde mental, permitindo a comunicação inicial de forma anônima e evoluindo para interações privadas e atendimentos estruturados.

1.2.2 Objetivos Específicos

1. Desenvolver uma plataforma web acessível para interação entre usuários e profissionais.
2. Implementar a funcionalidade de publicação de relatos.
3. Permitir a comunicação em tempo real entre pacientes e profissionais.
4. Desenvolver um módulo de conversas privadas entre os usuários.
5. Implementar um módulo de agendamento de atendimentos.
6. Integrar recursos de videochamada para suporte remoto.
7. Aplicar técnicas de inteligência artificial para análise emocional dos relatos e geração de cards.
8. Aplicar reconhecimento facial de emoções em tempo real durante as sessões e posteriormente a geração de relatórios.
9. Proporcionar registro de humor.
10. Garantir a segurança e privacidade dos dados dos usuários.
11. Proporcionar uma interface intuitiva e de fácil utilização.

2 Fundamentação Teórica

Este capítulo apresenta os principais conceitos teóricos que fundamentam o desenvolvimento deste trabalho, abordando aspectos relacionados à psicologia, ao uso da tecnologia no contexto da saúde mental e aos conceitos técnicos envolvidos no desenvolvimento de sistemas web.

Inicialmente, são discutidos os conceitos básicos da psicologia e da saúde mental, destacando a importância do acompanhamento psicológico e os desafios enfrentados no acesso a esses serviços. Em seguida, aborda-se a utilização de tecnologias digitais como ferramentas de apoio no contexto psicológico, evidenciando suas contribuições e limitações.

Por fim, são apresentados os fundamentos relacionados ao desenvolvimento de sistemas web, incluindo arquitetura, comunicação entre cliente e servidor, além de tecnologias utilizadas para viabilizar funcionalidades como comunicação em tempo real e análise emocional por meio de inteligência artificial. Esses conceitos são essenciais para a compreensão da proposta desenvolvida neste trabalho.

2.1 Psicologia e Saúde Mental

A Psicologia é a ciência responsável pelo estudo do comportamento humano e dos processos mentais, abrangendo aspectos como emoções, pensamentos e interações sociais. Seu principal objetivo é compreender e auxiliar na promoção da saúde mental, contribuindo para o bem-estar dos indivíduos em diferentes contextos (MYERS; DEWALL, 2015)

Os transtornos mentais, por sua vez, caracterizam-se por alterações significativas no funcionamento emocional, cognitivo ou comportamental de um indivíduo, podendo impactar negativamente sua qualidade de vida. Entre os transtornos mais comuns estão a ansiedade e a depressão, que afetam milhões de pessoas em todo o mundo. De acordo com a Organização Mundial da Saúde (OMS), os transtornos mentais estão entre as principais causas de incapacidade global, representando um desafio significativo para os sistemas de saúde (World Health Organization, 2025).

O tratamento desses transtornos pode envolver diferentes abordagens, incluindo psicoterapia, acompanhamento psicológico contínuo e, em alguns casos, intervenções medicamentosas. A psicoterapia, em especial, destaca-se como uma das principais formas de tratamento, permitindo que o indivíduo compreenda melhor suas emoções, pensamentos e comportamentos, promovendo mudanças positivas ao longo do tempo.

Entretanto, apesar da importância do acompanhamento psicológico, ainda existem diversas barreiras que dificultam o acesso a esses serviços. Fatores como o estigma social

associado aos transtornos mentais, a falta de recursos financeiros e a limitação de acesso a profissionais qualificados contribuem para que muitos indivíduos não busquem ajuda adequada. Nesse cenário, torna-se necessário o desenvolvimento de alternativas que ampliem o acesso ao cuidado psicológico, tornando-o mais acessível, flexível e acolhedor.

2.2 Uso de Tecnologia na Psicologia

Com o avanço das tecnologias digitais, novas possibilidades surgiram para a área da saúde mental, especialmente no que se refere à oferta de serviços psicológicos por meio de plataformas online. Nesse contexto, a saúde digital, também denominada (*e-health*), abrange o uso de tecnologias da informação e comunicação em atividades relacionadas à saúde, incluindo a disponibilização de informações, a comunicação entre usuários e profissionais e a oferta de intervenções mediadas por recursos digitais.

Segundo Andersson e Titov (2014), intervenções psicológicas realizadas pela internet têm apresentado resultados favoráveis no tratamento de determinados transtornos mentais, especialmente aqueles relacionados à ansiedade e à depressão. Os autores destacam, entretanto, que os resultados dependem de fatores como a modalidade de intervenção, o acompanhamento profissional e as características dos pacientes. Dessa forma, as evidências sobre intervenções psicológicas on-line não devem ser generalizadas automaticamente para qualquer plataforma digital voltada à saúde mental.

O ambiente digital também possibilita diferentes formas de interação entre pacientes e profissionais, como comunicação síncrona por áudio e vídeo, troca assíncrona de mensagens e acesso remoto aos serviços. Esses recursos podem oferecer alternativas para pessoas que enfrentam dificuldades de deslocamento ou limitações geográficas. Entretanto, sua utilização depende de condições como acesso à internet, disponibilidade de dispositivos, acessibilidade das interfaces e adequação da modalidade de atendimento às necessidades de cada pessoa.

Outro aspecto relevante é a possibilidade de utilização de recursos que favoreçam a expressão de experiências pessoais, incluindo mecanismos de interação que permitam diferentes níveis de identificação dos usuários. Embora essas funcionalidades possam contribuir para a percepção de privacidade, o anonimato oferecido por uma plataforma não deve ser confundido com garantia absoluta de confidencialidade, uma vez que o tratamento de dados e as características técnicas do sistema também influenciam a proteção das informações.

Apesar das possibilidades oferecidas pelas tecnologias digitais, sua utilização no contexto psicológico exige atenção às responsabilidades éticas e profissionais, à qualidade dos serviços e à proteção das informações pessoais. No Brasil, a prestação de serviços psicológicos mediada por tecnologias digitais está sujeita às normas do Conselho Federal

de Psicologia e às demais disposições legais aplicáveis. Nesse sentido, o desenvolvimento de plataformas voltadas à saúde mental deve considerar não apenas os requisitos funcionais e tecnológicos, mas também as condições de privacidade, confidencialidade, acessibilidade e adequação dos recursos utilizados.

2.2.1 Telepsicologia e Atendimento Psicológico Remoto

A telepsicologia corresponde à prestação de serviços psicológicos mediada por tecnologias digitais de informação e comunicação. Essa modalidade permite que profissionais e pessoas atendidas interajam remotamente por diferentes meios, incluindo videoconferências, chamadas de áudio e ferramentas de comunicação textual. Sua utilização amplia as possibilidades de organização e prestação dos serviços psicológicos, mas exige que os recursos tecnológicos sejam compatíveis com as características da demanda e com as condições de atendimento.

Andersson e Titov (2014) destacam que intervenções psicológicas baseadas na internet podem apresentar resultados favoráveis em determinadas condições, embora existam desafios relacionados à adesão dos participantes, à avaliação clínica, à identificação de resultados desfavoráveis e à integração dessas intervenções com os serviços tradicionais de saúde. Os autores também ressaltam a importância do acompanhamento profissional em diferentes modalidades de intervenção.

No atendimento remoto, a escolha dos recursos tecnológicos deve considerar aspectos como a estabilidade da comunicação, a privacidade do ambiente, a proteção das informações compartilhadas e a capacidade de manejar situações que demandem encaminhamento ou atendimento presencial. Além disso, a modalidade remota não elimina a necessidade de avaliação profissional sobre a adequação do serviço às necessidades da pessoa atendida.

Dessa forma, a adoção de tecnologias para o atendimento psicológico remoto deve ser compreendida não apenas como uma escolha técnica, mas também como uma decisão que envolve responsabilidades profissionais, éticas e legais. O uso de plataformas digitais precisa estar alinhado às necessidades do serviço prestado e às condições necessárias para preservar os direitos e a segurança das pessoas atendidas.

2.2.2 Regulamentação da Telepsicologia no Brasil

No Brasil, o exercício profissional da Psicologia mediado por tecnologias digitais é regulamentado pela Resolução CFP nº 9/2024 (CONSELHO FEDERAL DE PSICOLOGIA, 2024), que revogou as Resoluções CFP nº 11/2018 (CONSELHO FEDERAL DE PSICOLOGIA, 2018) e nº 4/2020 (CONSELHO FEDERAL DE PSICOLOGIA, 2020). A norma reconhece a utilização das Tecnologias Digitais da Informação e da Comunicação (TDICs) na prestação de serviços psicológicos e atribui ao profissional a responsabilidade de

avaliar a adequação dos recursos tecnológicos empregados, considerando as características do serviço, as necessidades das pessoas atendidas e os aspectos éticos envolvidos.

Entre os aspectos relevantes para o atendimento remoto estão a proteção da privacidade e da confidencialidade, a identificação dos profissionais e usuários quando necessária, a adequação das formas de comunicação e a existência de estratégias para lidar com demandas de urgência e emergência. A resolução também prevê a necessidade de informar as pessoas atendidas sobre os recursos tecnológicos utilizados e as medidas adotadas para proteger o sigilo das informações.

No desenvolvimento de plataformas digitais voltadas à saúde mental, esses requisitos evidenciam a importância de considerar as responsabilidades dos profissionais e as características dos recursos tecnológicos disponibilizados. Mecanismos de autenticação, controle de acesso e proteção das mensagens podem contribuir para a segurança do sistema, mas sua implementação não comprova, isoladamente, a conformidade integral com as exigências legais e profissionais.

No TherapistFriend, a comunicação privada, as sessões on-line e os mecanismos de proteção das mensagens foram concebidos para integrar os recursos de interação entre pacientes e profissionais. Entretanto, a descrição dessas funcionalidades representa uma caracterização técnica da solução desenvolvida, não uma certificação de conformidade com a regulamentação da telepsicologia. A verificação dessa conformidade exige uma análise específica das práticas profissionais, dos fluxos de dados e das demais condições de utilização da plataforma.

2.3 Desenvolvimento de Sistemas Web

Os sistemas *web* são aplicações desenvolvidas para serem acessadas por meio de navegadores, utilizando a internet como meio de comunicação entre usuários e servidores. Esse tipo de sistema tem se tornado amplamente utilizado devido à sua acessibilidade, uma vez que não requer instalação local e pode ser acessado de diferentes dispositivos. Sendo geralmente dividido em duas camadas principais: frontend e backend, cada uma responsável por diferentes aspectos da aplicação. Essa separação permite maior organização, escalabilidade e manutenção do sistema.

2.3.1 Frontend

O frontend é responsável pela interface com o usuário, sendo o componente que interage diretamente com o cliente. Ele é desenvolvido com tecnologias voltadas à construção de interfaces dinâmicas e responsivas, permitindo uma melhor experiência de uso (TOTVS, 2021).

2.3.1.1 REACT

O React é uma biblioteca frontend JavaScript de código aberto com foco em criar interfaces de usuário em páginas web. Criado em 2011, Jordan Walke, um engenheiro de software do Facebook, feito com o objetivo de simplificar a criação de interfaces de usuário (UI) dinâmicas e de alta performance para aplicações web. Tendo sua criação motivada pela necessidade de resolver problemas de escalabilidade e manutenção no desenvolvimento front-end, especialmente em aplicações complexas como o feed de notícias do Facebook, onde a atualização constante da interface exigia uma abordagem mais eficiente do que as soluções disponíveis na época (MORAIS, 2021).

O React revolucionou o desenvolvimento frontend. Ele influenciou o surgimento de outras bibliotecas e *frameworks*, como *Vue.js* e *Svelte*, e continua sendo uma das ferramentas mais populares e amplamente utilizadas no desenvolvimento web e móvel. Suas vantagens são:

- **Componentização:** O *React* permite dividir a interface em componentes reutilizáveis, o que facilita a manutenção e a escalabilidade do código.
- **Virtual DOM (Document Object Model):** Em vez de manipular o DOM diretamente, o *React* cria uma representação virtual do DOM em memória, o que torna as atualizações de interface mais eficientes.
- **Declarativo:** O React permite descrever como a interface deve ser em um determinado estado, em vez de se preocupar com como chegar a esse estado (abordagem imperativa).

2.4 Backend

O backend é responsável pela lógica de negócio processamento de dados e comunicação com o banco de dados. Ele atua como intermediário entre o frontend e os recursos do sistema, garantindo o funcionamento adequado da aplicação. Tecnologias baseadas em *JavaScript*, como o *Node.js* com o *framework Express*, são amplamente utilizadas devido à sua eficiência e capacidade de lidar com múltiplas requisições de forma assíncrona (ALURA, 2024).

Além disso, linguagens como Python também são frequentemente empregadas no desenvolvimento de serviços especializados, especialmente em aplicações que envolvem processamento de dados e inteligência artificial. Frameworks como o Flask oferecem uma estrutura leve e flexível para a criação de APIs e microserviços, sendo amplamente adotados nesse tipo de cenário.

2.4.0.1 NODE.JS e Express

O Node.js é um ambiente de execução JavaScript criado por Ryan Dahl em 2009. Ele permite que o JavaScript, tradicionalmente uma linguagem voltada para o frontend, seja executado no lado do servidor (backend). Essa inovação foi possível graças ao motor V8, desenvolvido pelo Google para o Chrome, que compila JavaScript em código de máquina, garantindo alta performance. Sendo projetado para ser leve, eficiente e ideal para aplicações que exigem operações *I/O* (entrada e saída) assíncronas e em tempo real, como chats, *streaming* de dados e *APIS*. Possuindo um modelo inovador: sua arquitetura é totalmente *non-blocking thread* (não bloqueante). Se sua aplicação trabalha com processamento de arquivos e/ou realizada muito *I/O*, adotar esse tipo de arquitetura vai resultar em uma boa performance com relação ao consumo de memória, já que usa ao máximo e de forma eficiente o poder de processamento dos servidores – principalmente em sistemas que produzem uma alta carga de processamento (PEREIRA, 2014).

O Express é um *framework* de aplicativo web Node.js mínimo e flexível que fornece um conjunto robusto de recursos para aplicativos web e móveis (BROWN, 2019), lançado em 2010 por TJ Holowaychuk. Ele foi criado para simplificar o desenvolvimento de aplicações *web* e *APIs*, fornecendo uma camada de abstração sobre as funcionalidades básicas do *Node.js*. O *Express* é amplamente utilizado por sua simplicidade, flexibilidade e capacidade de integrar se com outras bibliotecas e ferramentas. As principais caraterísticas que impulsionaram sua adoção foram:

- **Sistemas de rotas:** O *Express* permite definir rotas de forma simples e intuitiva, associando *URLs* a funções de *callback*.
- **Middlewares:** O conceito de *middlewares* no *Express* permite adicionar funcionalidades extras (como autenticação, *logging* e tratamento de erros) de forma modular.

2.5 Banco de dados PostgreSQL

O *PostgreSQL* é um banco de dados relacional que usa a linguagem *SQL*(*Structured Query Language*) para gerenciar e manipular dados. Sendo, suas principais características:

- **ACID:** Garante atomicidade, consistência, isolamento e durabilidade nas transações.
- **Multiplataforma:** Funciona em Windows, Linux, macOS e outros sistemas.
- **Recursos avançados:** Suporta JSON, consultas geográficas, *full-text search*, replicação, e muito mais.
- **Extensível:** Permite criar tipos de dados personalizados, funções e extensões.
- **Conformidade com SQL:** Suporta a maioria dos padrões *SQL*.

2.6 Containerização e Virtualização com Docker

A containerização é uma abordagem que permite empacotar uma aplicação e suas dependências em um ambiente isolado, garantindo que ela possa ser executada de forma consistente em diferentes sistemas. Essa técnica tem se tornado amplamente utilizada no desenvolvimento de software moderno, especialmente em arquiteturas baseadas em microserviços (Google Cloud, s.d.).

O *Docker* é uma das principais ferramentas utilizadas para esse fim, permitindo a criação e execução de containers de forma eficiente. Segundo Merkel (2014), os containers *Docker* oferecem isolamento leve, inicialização rápida e portabilidade, facilitando o processo de desenvolvimento, testes e implantação de aplicações.

A utilização de containers contribui para a padronização do ambiente de execução, reduzindo problemas relacionados a incompatibilidades entre diferentes sistemas. Além disso, facilita a escalabilidade da aplicação, permitindo a replicação de serviços de forma simples e eficiente (Google Cloud, s.d.).

No contexto de sistemas distribuídos, como o proposto neste trabalho, a containerização desempenha um papel importante na organização e integração dos diferentes serviços, garantindo maior flexibilidade, confiabilidade e facilidade de manutenção.

2.7 Comunicação em Sistemas Distribuídos e Tempo Real

Em sistemas modernos, especialmente aqueles baseados em arquiteturas distribuídas, a comunicação entre diferentes componentes é um fator essencial para o funcionamento eficiente da aplicação. Diferentemente de sistemas monolíticos, onde os módulos estão centralizados, sistemas distribuídos exigem mecanismos capazes de garantir a troca de informações de forma confiável, escalável e, em muitos casos, em tempo real.

A comunicação em tempo real caracteriza-se pela capacidade de troca contínua de dados entre cliente e servidor, com baixa latência e atualização instantânea das informações. Esse tipo de comunicação é fundamental em aplicações que demandam interatividade, como chats, notificações e videoconferências, proporcionando uma experiência mais fluida e próxima da comunicação presencial.

2.7.1 WebSocket

Enquanto o protocolo *HTTP* tradicional opera em um modelo de requisição e resposta (*request-response*) — onde o cliente deve sempre iniciar a comunicação —, certas aplicações modernas exigem uma troca de dados constante e em tempo real. Nesse cenário, surge o protocolo *WebSocket*. Definido pela *RFC* 6455, ele permite a criação de um

canal de comunicação bidirecional (*full-duplex*) sobre uma única conexão *TCP* persistente (LOMBARD, 2015).

Segundo Fette e Melnikov (2011), essa abordagem reduz significativamente a latência e o *overhead* de comunicação, sendo amplamente utilizada em aplicações que exigem atualização constante de dados, como sistemas de mensagens instantâneas e notificações em tempo real.

2.7.1.1 O Handshake e a Conexão

O ciclo de vida de uma conexão *WebSocket* inicia-se obrigatoriamente com um processo conhecido como *Handshake* (aperto de mão). Este processo utiliza o protocolo *HTTP* para solicitar uma "atualização"da conexão (*Upgrade*). Uma vez que o servidor aceita a solicitação, a conexão *TCP* subjacente permanece aberta e o protocolo é alternado de *HTTP* para *WebSocket*.

- **Persistência:** A conexão permanece ativa enquanto não for explicitamente encerrada por uma das partes, eliminando a necessidade de renegociar o túnel de comunicação a cada nova mensagem.
- **Baixo *Overhead*:** Após o handshake inicial, as mensagens (chamadas de frames) possuem cabeçalhos extremamente reduzidos, variando entre 2 e 14 bytes, o que otimiza significativamente o tráfego de dados em comparação aos cabeçalhos *HTTP* que podem chegar a kilobytes.
- **Comunicação em Tempo Real:** Pela sua natureza bidirecional, o servidor pode realizar o push de dados para o cliente no exato momento em que um evento ocorre, sem que o cliente precise solicitar essa atualização.

2.7.2 WebRTC

O *Web Real-Time Communication (WebRTC)* surgiu da necessidade de transformar o navegador em uma plataforma de comunicação em tempo real e de código aberto. Proposto inicialmente pelo *Google* em 2011, após a aquisição da tecnologia *GIPS (Global IP Solutions)*, o projeto evoluiu para um padrão mantido pela *W3C* e pela *IETF*, eliminando a dependência de plugins proprietários, como o extinto *Adobe Flash Player*.

De acordo com Mahmoud e Abozariba (2024), o *WebRTC* tem sido amplamente adotado em aplicações de videoconferência e comunicação online, oferecendo suporte nativo em navegadores modernos e possibilitando a criação de soluções robustas para interação em tempo real, sendo a sua principal vantagem a arquitetura *peer-to-peer*(P2P).

2.7.2.1 Pilares Tecnológicos do WebRTC

Para viabilizar uma videochamada, o *WebRTC* utiliza três interfaces de programação (APIs) fundamentais, que operam de forma integrada:

- **MediaStream (getUserMedia):** Esta API é o ponto de entrada para a captura de mídia. Ela permite que o navegador solicite permissão para acessar o hardware do usuário, como câmeras e microfones, gerindo os fluxos de áudio e vídeo locais.
- **RTCPeerConnection:** Considerada o componente central da tecnologia, é responsável por estabelecer, manter e monitorar a conexão direta entre dois pares. Conforme descrito por Loreto e Romano (2014), esta *API* lida com a compressão de dados (*codecs*), criptografia e a travessia de *NATs* e *Firewalls* através do protocolo *ICE (Interactive Connectivity Establishment)*, utilizando servidores auxiliares *STUN* e *TURN*.
- **RTCDataChannel:** Além de mídia, o *WebRTC* permite a troca de dados genéricos com baixa latência. Em aplicações de videochamada, este canal pode ser utilizado para funcionalidades secundárias, como um chat de texto simultâneo ou o envio de arquivos entre os participantes, sem passar por um servidor intermediário.

2.7.2.2 O Processo de Sinalização e Travessia de Rede

Diferente de outras tecnologias, o *WebRTC* não define um protocolo de sinalização obrigatório. Isso significa que, para que dois dispositivos se encontrem na internet, o desenvolvedor deve implementar um servidor de sinalização (geralmente utilizando *WebSockets*) para trocar informações de metadados, conhecidas como *Session Description Protocol (SDP)*.

Além disso, como a maioria dos dispositivos não possui um *IP* público direto, o *WebRTC* utiliza os mecanismos *STUN (Session Traversal Utilities for NAT)* para descobrir o endereço público do par, e *TURN (Traversal Using Relays around NAT)* como um servidor de contingência quando a conexão direta P2P é bloqueada por restrições de rede.

Na prática, os serviços *STUN* e *TURN* são disponibilizados por servidores especializados responsáveis por auxiliar o estabelecimento das conexões entre os pares. Dentre as implementações disponíveis, destaca-se o *Coturn*, uma solução de código aberto amplamente utilizada para disponibilização de servidores *TURN* e *STUN* em aplicações baseadas em WebRTC. Sua utilização contribui para aumentar a confiabilidade das conexões em cenários onde a comunicação direta entre os participantes é limitada por mecanismos de *NAT* ou políticas restritivas de *firewall* (MEETRIX, 2026).

2.7.3 gRPC

O *gRPC* (*Remote Procedure Call*) é um *framework* de código aberto e alto desempenho, desenvolvido originalmente pelo *Google*, que facilita a comunicação entre sistemas distribuídos. Ele permite que uma aplicação cliente chame diretamente métodos em uma aplicação servidora localizada em outra máquina como se fosse uma chamada de função local, abstraindo a complexidade da rede.

De acordo com Indrasiri e Kuruppu (2020), o *gRPC* baseia-se em dois pilares fundamentais para atingir sua alta eficiência: o uso do protocolo *HTTP/2* como transporte e o *Protocol Buffers (Protobuf)* como linguagem de interface e formato de serialização de dados.

2.7.3.1 Protocol Buffers e HTTP/2

Diferente de arquiteturas baseadas em *REST*, que geralmente utilizam *JSON* ou *XML* (formatos de texto legíveis por humanos, mas volumosos), o *gRPC* utiliza o *Protobuf*. Este é um mecanismo de serialização de dados binários, o que resulta em mensagens significativamente menores e em um processamento muito mais rápido de codificação e decodificação.

As principais vantagens técnicas do *gRPC* incluem:

- **Contrato Estritamente Tipado:** O uso de arquivos .proto define antecipadamente a estrutura dos dados e os serviços disponíveis, permitindo a geração automática de código (*stubs*) em diversas linguagens de programação.
- **Streaming Bidirecional:** Graças ao *HTTP/2*, o *gRPC* suporta fluxos de dados de forma nativa. Conforme destacam Indrasiri e Kuruppu (2020), é possível realizar streaming do cliente para o servidor, do servidor para o cliente, ou em ambas as direções simultaneamente.
- **Multiplexação:** O *HTTP/2* permite que múltiplas chamadas sejam enviadas através de uma única conexão *TCP* sem que uma bloqueie a outra, otimizando o uso dos recursos de infraestrutura.

2.8 Inteligência Artificial

A Inteligência Artificial (IA) define-se como o campo da ciência da computação dedicado ao desenvolvimento de sistemas capazes de simular capacidades cognitivas humanas, como o raciocínio, a aprendizagem e a percepção. Segundo Russell e Norvig (2020), a evolução da área permitiu a transição de sistemas baseados em regras rígidas

para modelos de aprendizado profundo (*Deep Learning*), que extraem padrões complexos diretamente de grandes volumes de dados.

2.9 Inteligência Artificial Generativa

A Inteligência Artificial Generativa (GenAI) representa uma subárea do aprendizado de máquina que se diferencia dos modelos tradicionais, denominados discriminativos. Enquanto os modelos discriminativos aprendem a mapear entradas para rótulos específicos (como classificar se uma imagem contém um rosto ou não), os modelos generativos aprendem a distribuição de probabilidade subjacente dos dados de treinamento para gerar novas instâncias de dados que poderiam realisticamente pertencer ao conjunto original (GOODFELLOW; BENGIO; COURVILLE, 2016a).

2.9.1 Princípios de Modelagem Generativa

O funcionamento da IA Generativa baseia-se na capacidade de compressão e reconstrução de informações. Através de espaços latentes de alta dimensão, o modelo identifica as características essenciais de um domínio — sejam as regras gramaticais de uma língua ou os traços anatômicos de uma face — e utiliza esse conhecimento para sintetizar novos conteúdos originais.

2.9.2 Arquiteturas Fundamentais

A evolução da GenAI é sustentada por três arquiteturas principais documentadas na literatura técnica:

- **Redes Neurais Generativas Adversariais (GANs):** Propostas por Ian Goodfellow em 2014, as *GANs* operam através de uma competição entre duas redes: um Gerador, que cria dados sintéticos, e um Discriminador, que tenta distinguir entre dados reais e falsos. Esse processo de "jogo de soma zero"força o gerador a produzir resultados cada vez mais indistinguíveis da realidade.

- **Modelos de Difusão:** Utilizados principalmente para geração de imagens e vídeos, esses modelos funcionam adicionando ruído gaussiano aos dados de forma iterativa e, em seguida, aprendendo o processo inverso para recuperar a estrutura original a partir do ruído puro (SPADINI, 2024).

- **Decodificadores de Transformers:** No domínio do texto, a IA Generativa utiliza exclusivamente a pilha de decodificadores da arquitetura Transformer (modelos *decoder-only*, como o *Llama 3*). Esses modelos preveem a próxima unidade lógica

(*token*) de forma autoregressiva, permitindo a criação de narrativas, diálogos e códigos de programação coerentes (VASWANI et al., 2017).

2.9.3 Modelos de Linguagem de Grande Escala(LLM)

Os *Large Language Models (LLMs)* constituem uma das aplicações mais bem-sucedidas da Inteligência Artificial Generativa. Eles são projetados para processar, interpretar e gerar linguagem natural em uma escala que mimetiza a fluência humana. O sucesso desses modelos reside na sua capacidade de prever a probabilidade estatística de sequências de palavras, permitindo a realização de tarefas complexas como tradução, sumarização e raciocínio lógico.

Conforme discutido anteriormente, a eficácia dos *LLMs* modernos não provém apenas da escala de dados, mas de uma mudança fundamental na forma como a computação lida com a linguagem: a adoção da arquitetura Transformer.

2.9.4 Arquitetura Baseada em Transformers

A arquitetura Transformer, introduzida por Vaswani et al. (2017), abandonou as abordagens recorrentes e convolucionais para o processamento de sequências, introduzindo uma estrutura puramente baseada em mecanismos de atenção. Essa mudança permitiu que os modelos fossem treinados em paralelo, aproveitando ao máximo o poder das *GPUs*.

2.9.4.1 Mecanismo de Autoatenção (Self-Attention)

O núcleo do Transformer é o mecanismo de *Self-Attention*. Matematicamente, para cada elemento da sequência, o modelo calcula uma pontuação de relevância em relação a todos os outros elementos. Isso é feito através da projeção da entrada em três espaços vetoriais distintos: Consultas (*Queries*), Chaves (*Keys*) e Valores (*Values*). A saída é uma soma ponderada dos valores, onde os pesos são determinados pelo produto escalar entre a consulta e a chave correspondente (VASWANI et al., 2017, p. 2).

2.9.4.2 Componentes do Transformer (Encoder/Decoder)

Estruturalmente, o Transformer original é composto por dois blocos principais:

- **Encoder:** Responsável por converter a sequência de entrada em uma representação numérica abstrata que preserva o contexto.
- **Decoder:** Responsável por utilizar essa representação para gerar a sequência de saída, um elemento por vez, de forma autorregressiva.

Enquanto o Transformer original utilizava ambos, a maioria dos *LLMs* generativos contemporâneos, como o *Llama 3*, utiliza uma arquitetura puramente de *Decoder*, otimizada especificamente para a geração de texto.

2.9.5 Modelos de Linguagem Aplicados à Saúde Mental: Estado da Arte e Limitações

O avanço dos modelos de linguagem de grande escala tem ampliado as possibilidades de aplicação da inteligência artificial na área da saúde mental. Esses modelos vêm sendo investigados em tarefas como identificação de sinais de sofrimento psicológico em textos, análise de conteúdo relacionado à saúde mental, disponibilização de agentes conversacionais e apoio a profissionais em atividades clínicas. Entretanto, o desempenho observado nessas tarefas varia conforme o modelo utilizado, a natureza dos dados e os critérios adotados para sua avaliação.

Em uma revisão sistemática, Guo et al. (2024) analisaram 40 estudos sobre aplicações de LLMs na saúde mental. Os trabalhos investigados incluíam a identificação de condições psicológicas e de ideação suicida por meio de análise textual, o desenvolvimento de agentes conversacionais e outras formas de utilização desses modelos. Embora os resultados indiquem possibilidades de aplicação, os autores também identificaram limitações relacionadas à inconsistência das respostas geradas, à ocorrência de informações incorretas ou inventadas, à interpretabilidade dos modelos e à ausência de estruturas de avaliação ética amplamente consolidadas. Essas limitações evidenciam que o desempenho satisfatório em determinadas tarefas não garante, por si só, a segurança do modelo em situações clínicas.

Em estudo de revisão publicado em 2025, Hua et al. (2025) examinaram aplicações generativas de LLMs na saúde mental, incluindo assistência clínica, aconselhamento e suporte emocional. Apesar dos resultados iniciais promissores, os autores observaram que os métodos de avaliação utilizados nos estudos eram frequentemente heterogêneos, dificultando a comparação entre os resultados e a determinação da efetividade das soluções. A revisão também apontou a necessidade de procedimentos de avaliação mais rigorosos antes da incorporação desses modelos como intervenções independentes no cuidado em saúde mental.

Entre os principais desafios relacionados à utilização de LLMs nesse domínio estão a possibilidade de geração de respostas inadequadas, a reprodução de vieses presentes nos dados de treinamento, a dificuldade de interpretar o processo que conduz a determinadas respostas e os riscos associados ao tratamento de informações pessoais sensíveis. Em situações que envolvem sofrimento psicológico intenso ou risco de suicídio, respostas incorretas ou interpretações inadequadas podem produzir consequências relevantes. Por

essa razão, esses modelos não devem ser considerados substitutos da avaliação realizada por profissionais qualificados, sendo necessário delimitar suas funções e avaliar cuidadosamente os riscos envolvidos em cada aplicação (GUO et al., 2024).

Nesse contexto, a utilização de modelos de linguagem em plataformas digitais de saúde mental deve considerar tanto suas capacidades de processamento da linguagem natural quanto suas limitações. No TherapistFriend, a inteligência artificial textual é empregada para analisar relatos publicados pelos usuários e gerar recomendações personalizadas a partir das informações processadas. Essa funcionalidade constitui um recurso auxiliar da plataforma e não deve ser interpretada como diagnóstico psicológico, identificação clínica definitiva ou substituição do acompanhamento profissional. A adoção desse recurso deve ser compreendida à luz das limitações descritas na literatura, especialmente quanto à confiabilidade das respostas, à proteção dos dados tratados e à necessidade de avaliação específica para o contexto de aplicação.

2.10 Ollama

O Ollama consolidou-se como uma das ferramentas mais eficientes para a execução de Grandes Modelos de Linguagem (LLMs) em ambientes locais. Ele atua como um gerenciador de modelos que simplifica a complexidade de configuração de ambientes de IA, permitindo que desenvolvedores executem modelos potentes sem a necessidade de infraestruturas em nuvem ou configurações complexas de bibliotecas de aprendizado profundo (BALARABE, 2025).

A grande vantagem do Ollama reside na sua capacidade de otimizar o uso de recursos de hardware local, como a memória de vídeo (*VRAM*). Em sistemas equipados com *GPUs* dedicadas, o *Ollama* automatiza o carregamento dos pesos do modelo, garantindo que a inferência ocorra com baixa latência, o que é crucial para aplicações que exigem respostas rápidas ou que lidam com dados sensíveis que não devem sair do ambiente local (ISAAC, 2025).

2.11 O Modelo de Linguagem Llama 3

O *Llama 3*, desenvolvido pela Meta, representa um salto significativo na evolução dos modelos de linguagem de código aberto. Treinado em um conjunto de dados massivo (superior a 15 trilhões de *tokens*), o modelo foi projetado para oferecer capacidades avançadas de raciocínio, geração de código e compreensão de contexto, competindo diretamente com modelos proprietários de escala industrial (META AI, 2024).

2.11.0.1 Inovações e Performance

A arquitetura do *Llama 3* introduz melhorias na eficiência do *tokenizer*(algoritmo que mapeia strings de texto para tokens discretos do vocabulário) e na atenção do modelo, permitindo que ele capture nuances linguísticas e técnicas com maior precisão. No contexto de desenvolvimento de software e análise de dados, destaca-se por sua habilidade em seguir instruções complexas e manter a coerência em diálogos longos (META AI, 2024).

Quando executado sobre o *framework Ollama*, o *Llama 3* torna-se uma ferramenta poderosa para:

- **Processamento de Linguagem Natural (PLN):** Resumo de textos técnicos, extração de entidades e tradução contextualizada.
- **Privacidade de Dados:** Processamento de informações privadas em servidor local, eliminando riscos de segurança associados a *APIs* externas.

2.12 Visão Computacional e Visão Neural

A Visão Computacional é a área da inteligência artificial que desenvolve sistemas capazes de extrair, analisar e compreender informações úteis de imagens ou sequências de vídeos. Historicamente, o campo baseava-se em técnicas de processamento digital de imagens e algoritmos geométricos rígidos. Contudo, a ascensão da Visão Neural Computacional — fundamentada no uso de redes neurais profundas — revolucionou o setor ao permitir que as máquinas aprendam características visuais de forma hierárquica, assemelhando-se ao funcionamento do sistema visual humano.

A Visão Computacional é definida como o campo que busca capacitar sistemas digitais a reconstruir e interpretar o mundo tridimensional a partir de dados visuais bidimensionais, como imagens e vídeos (SZELISKI, 2022). Historicamente, o processamento dessas informações dependia de algoritmos matemáticos rígidos para a extração manual de características. No entanto, a transição para o paradigma da Visão Neural, impulsionada pelo aprendizado profundo (Deep Learning), permitiu que o próprio sistema aprenda representações hierárquicas diretamente dos dados brutos (GOODFELLOW; BENGIO; COURVILLE, 2016a).

Essa evolução foi marcada pela substituição de filtros projetados manualmente por arquiteturas de Redes Neurais Convolucionais (*CNNs*), que mimetizam de forma simplificada a organização do córtex visual humano. Segundo Goodfellow, Bengio e Courville (2016b), essa abordagem permite a identificação automatizada de padrões complexos, desde bordas fundamentais em camadas iniciais até estruturas semânticas

detalhadas em camadas profundas, sendo a base tecnológica que viabiliza modelos de alta performance como o *YOLOv8*.

2.12.1 Redes Neurais Convolucionais (CNNs)

A base da visão neural moderna são as Redes Neurais Convolucionais (CNNs). Diferente das redes neurais tradicionais, as CNNs preservam a estrutura espacial dos dados através de operações de convolução, o que as torna ideais para o processamento de imagens (ULTRALYTICS, 2024).

2.12.1.1 Camadas de Convolução e Filtros

A camada de convolução utiliza um conjunto de filtros aprendíveis (núcleos ou *kernels*). À medida que o filtro desliza sobre a imagem de entrada, ele realiza o produto escalar entre seus pesos e a região da imagem, gerando um mapa de características (*feature map*). Nas camadas iniciais, esses filtros detectam padrões rudimentares, como bordas e variações de iluminação. Nas camadas subsequentes, a rede combina esses sinais para identificar texturas e formas complexas (ULTRALYTICS, 2024).

2.12.1.2 Camadas de Pooling e Funções de Ativação

Para reduzir a sensibilidade do modelo à localização exata de características e diminuir o custo computacional, as CNNs utilizam camadas de Pooling (subamostragem). O *Max Pooling*, por exemplo, seleciona o valor máximo de uma região, garantindo que as características mais relevantes sobrevivam ao processo de redução (EITCA Academy, 2023). Após a convolução, utiliza-se geralmente a função de ativação *ReLU (Rectified Linear Unit)*, que introduz não-linearidade ao modelo, permitindo o aprendizado de padrões visuais complexos (KANANI, 2020).

2.12.2 Paradigmas de Detecção: Do R-CNN ao YOLO

A evolução da visão neural levou ao desafio da Detecção de Objetos, que exige não apenas a classificação do que está na imagem, mas a localização exata de onde o objeto se encontra.

- **Modelos de Dois Estágios (Two-Stage):** Como o R-CNN(Region-based Convolutional Neural Networks), que primeiro propõe regiões de interesse e depois as classifica. Embora precisos, são computacionalmente lentos e inadequados para processamento em tempo real.
- **Modelos de Estágio Único (One-Shot):** Como o *YOLO (You Only Look Once)*, que será detalhado a seguir. Este paradigma trata a detecção como um problema de

regressão única, permitindo a análise de *frames* de vídeo em tempo real com alta eficiência.

2.12.3 YOLOv8 para Detecção Facial

O *YOLOv8*, lançado pela *Ultralytics*, representa o estado da arte em termos de velocidade e precisão para tarefas de visão computacional. Diferente de versões anteriores, ele utiliza uma arquitetura *anchor-free*, o que facilita a detecção de objetos com variadas proporções, como as faces humanas em diferentes ângulos e expressões (ULTRALYTICS, 2023).

Diferentemente de abordagens tradicionais que realizam múltiplas etapas para detecção de objetos, o YOLO utiliza uma única rede neural para prever simultaneamente as *bounding boxes* e as classes dos objetos presentes na imagem (YASEEN, 2024, p. 2). Essa característica torna o modelo especialmente adequado para aplicações que exigem processamento em tempo real, como sistemas de reconhecimento facial e análise emocional.

2.12.3.1 O Dataset AffectNet em Formato YOLO

O AffectNet é um dos maiores datasets de expressões faciais disponíveis, contendo imagens rotuladas com emoções como felicidade, tristeza, raiva, entre outras. Para a utilização no YOLOv8, o dataset deve ser convertido para o YOLO Format, que difere de formatos tradicionais como COCO (AMAZON WEB SERVICES, 2026b) ou XML (W3SCHOOLS, 2026). O dataset Affectnet no formato YOLO é um subconjunto reduzido e pré-processado do dataset original, possuindo ao todo cerca de 25.000 imagens.

No formato YOLO, cada imagem possui um arquivo .txt correspondente:

- Class ID: Um número inteiro representando a emoção (ex: 0 para 'Felicidade', 1 para 'Tristeza').
- Centro X e Centro Y: Coordenadas normalizadas (entre 0 e 1) do centro da face.
- Largura e Altura: Dimensões normalizadas da caixa delimitadora (*bounding box*).

Figura 1 – Exemplo de imagens usadas no dataset



Fonte: fatihkgg (2023)

2.12.4 Estado da Arte no Reconhecimento de Expressões Faciais

O reconhecimento de expressões faciais (Facial Expression Recognition — FER) constitui uma linha de pesquisa relevante em visão computacional, com aplicações em interação humano-computador, análise de comportamento e sistemas de apoio à saúde. Ao longo de sua evolução, as abordagens utilizadas nessa área passaram de métodos baseados na extração manual de características para modelos de aprendizado profundo, capazes de aprender representações visuais diretamente dos dados. As redes neurais convolucionais contribuíram significativamente para essa evolução, enquanto arquiteturas mais recentes passaram a explorar mecanismos de atenção e outras estratégias para representar relações entre diferentes regiões da face (KOPALIDIS et al., 2024).

O desenvolvimento e a comparação desses métodos dependem de bases de dados anotadas, utilizadas tanto no treinamento quanto na avaliação dos modelos. Entre os conjuntos empregados na literatura estão o FER-2013, o RAF-DB e o AffectNet, que apresentam diferenças quanto à quantidade de imagens, às condições de coleta e aos procedimentos de anotação. O AffectNet destaca-se por reunir imagens faciais coletadas em condições variadas, permitindo investigar o reconhecimento de expressões em cenários menos controlados. Entretanto, a diversidade dos dados também impõe desafios ao treinamento e à avaliação, especialmente quando existem diferenças entre as características das imagens utilizadas no treinamento e aquelas encontradas durante a aplicação do modelo (KOPALIDIS et al., 2024).

A avaliação de sistemas de reconhecimento de expressões faciais costuma utilizar métricas como acurácia, precisão, revocação e F1-score, que permitem analisar diferentes aspectos do desempenho de classificação. Entretanto, os resultados obtidos em uma base específica não devem ser interpretados isoladamente como evidência de desempenho equivalente em ambientes reais. Diferenças na distribuição das classes, nas condições de iluminação, na orientação da cabeça, na resolução das imagens e nas características dos indivíduos podem afetar a capacidade de generalização dos modelos. Por esse motivo, a comparação entre abordagens deve considerar não apenas os valores das métricas, mas também as condições experimentais e as características dos conjuntos de dados empregados (KOPALIDIS et al., 2024).

No contexto deste trabalho, esses aspectos são relevantes para compreender a utilização do YOLOv8 treinado com um subconjunto do AffectNet adaptado ao formato YOLO. O modelo foi empregado para detectar faces e associá-las às classes de expressões definidas nas anotações utilizadas no treinamento. Essa abordagem permite integrar a análise visual ao funcionamento da plataforma, mas seus resultados devem ser interpretados de acordo com as características dos dados utilizados e com as limitações inerentes à classificação de expressões faciais. A adequação do modelo ao contexto de uso depende, portanto, da avaliação de seu desempenho e não apenas da arquitetura escolhida.

2.12.5 Limitações e Desafios na Inferência de Estados Emocionais

Apesar dos avanços obtidos com o aprendizado profundo, o reconhecimento de expressões faciais apresenta limitações relacionadas tanto ao processamento das imagens quanto à interpretação dos resultados. Fatores como iluminação, oclusões, orientação da cabeça, resolução das imagens e diferenças entre os dados de treinamento e as condições de aplicação podem comprometer o desempenho dos modelos. Além disso, desequilíbrios entre as classes e inconsistências nas anotações podem influenciar o aprendizado e dificultar a comparação entre diferentes abordagens (CAI; LI; LI, 2023).

Outra limitação fundamental consiste na distinção entre reconhecer padrões de movimentos faciais e determinar o estado emocional de uma pessoa. Cai, Li e Li (2023), ao revisarem evidências provenientes de diferentes áreas de pesquisa, argumentam que as configurações faciais associadas a determinadas emoções não apresentam confiabilidade e especificidade suficientes para serem consideradas indicadores universais desses estados. A expressão observada pode variar de acordo com o indivíduo, a situação e o contexto cultural, de modo que uma mesma configuração facial pode estar associada a diferentes interpretações emocionais.

Essa distinção é especialmente importante em aplicações voltadas à saúde mental, nas quais classificações automatizadas podem ser interpretadas equivocadamente como avaliações psicológicas. Um modelo treinado para classificar expressões faciais aprende associações entre características visuais e rótulos presentes nos dados de treinamento, mas isso não comprova que seja capaz de identificar com precisão o estado emocional real de uma pessoa. Portanto, seus resultados devem ser tratados como estimativas relacionadas aos padrões visuais aprendidos, e não como diagnósticos ou evidências conclusivas sobre a condição psicológica do indivíduo.

No TherapistFriend, a análise por visão computacional deve ser compreendida como um recurso auxiliar, cujas saídas podem fornecer informações complementares ao profissional durante uma sessão online. A interpretação dessas informações deve considerar as limitações do modelo e o contexto do atendimento, sem substituir a avaliação profissional ou fundamentar conclusões clínicas exclusivamente nas classificações automatizadas.

2.13 Mensageria e Processamento Assíncrono

Em sistemas modernos, especialmente aqueles baseados em arquiteturas distribuídas, o processamento assíncrono desempenha um papel fundamental na garantia de desempenho, escalabilidade e resiliência. Diferentemente do processamento síncrono, no qual uma requisição depende da conclusão imediata de uma operação, o processamento assíncrono permite que tarefas sejam executadas em segundo plano, sem bloquear o fluxo principal da aplicação (BEVANS, 2025).

A mensageria é uma abordagem amplamente utilizada para viabilizar esse tipo de processamento, baseando-se na troca de mensagens entre diferentes componentes do sistema por meio de filas (*queues*). Nesse modelo, um produtor envia mensagens para uma fila, enquanto um ou mais consumidores processam essas mensagens de forma independente, permitindo maior desacoplamento entre os módulos da aplicação (AMAZON WEB SERVICES, 2026a).

Ferramentas como o *Redis* são frequentemente utilizadas como intermediários de mensageria devido à sua alta performance e capacidade de armazenamento em memória. O Redis é um sistema de armazenamento chave-valor que pode ser utilizado para diversas finalidades, incluindo cache, gerenciamento de sessões e implementação de filas de mensagens (IBM, s.d.b).

Além disso, bibliotecas como o *BullMQ* utilizam o *Redis* como base para gerenciamento de filas, oferecendo recursos como controle de concorrência, reprocessamento de tarefas, agendamento de *jobs* e monitoramento do estado das execuções. Essa abordagem permite o tratamento eficiente de tarefas que demandam maior tempo de processamento, como envio de notificações, processamento de dados e integração com serviços externos (RAFAEL, 2024).

2.14 Segurança da Informação

A segurança da informação é um aspecto fundamental no desenvolvimento de sistemas que lidam com dados sensíveis, especialmente no contexto da saúde, onde informações pessoais e emocionais dos usuários devem ser tratadas com estrito sigilo. Nesse cenário, é essencial garantir princípios como confidencialidade, integridade e disponibilidade dos dados (LNCC - Laboratório Nacional de Computação Científica, 2024).

A confidencialidade refere-se à proteção das informações contra acessos não autorizados, enquanto a integridade assegura que os dados não sejam alterados indevidamente. Já a disponibilidade garante que as informações estejam acessíveis sempre que necessário, evitando interrupções no funcionamento do sistema (LNCC - Laboratório Nacional de Computação Científica, 2024).

2.14.1 Criptografia de Dados com AES-256

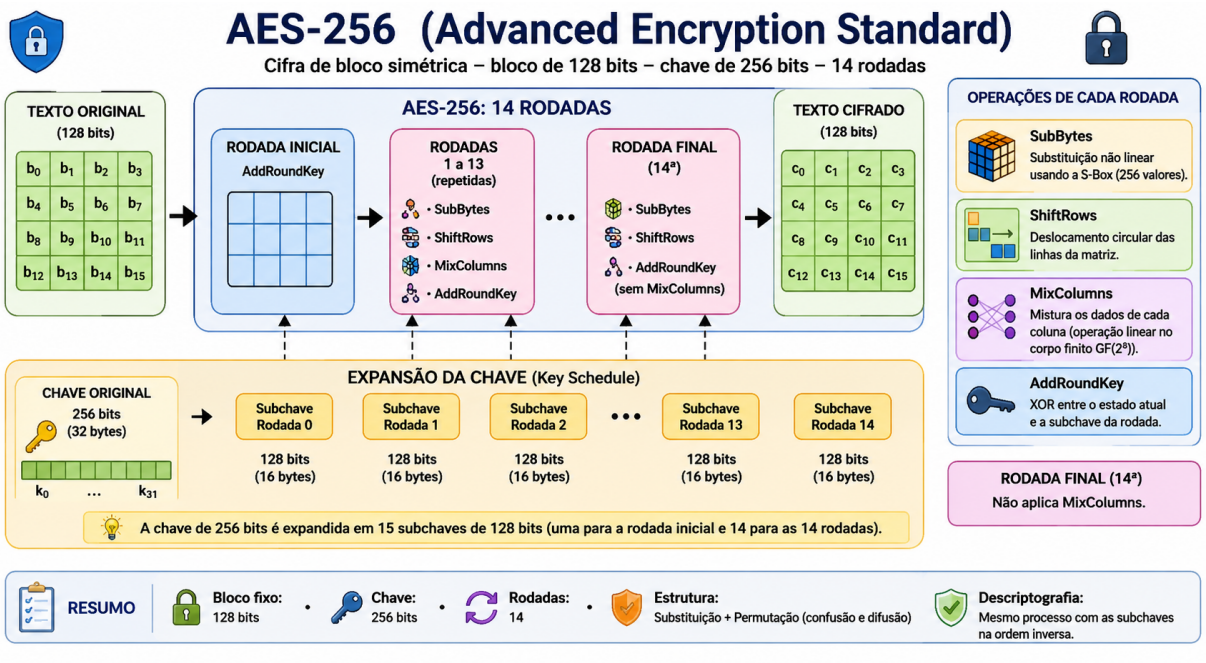
A criptografia é uma das principais técnicas utilizadas para proteger informações sensíveis, permitindo que os dados sejam convertidos em um formato ilegível para usuários não autorizados. Entre os algoritmos mais utilizados, destaca-se o *Advanced Encryption Standard (AES)*, amplamente adotado devido à sua segurança e eficiência (IBM, s.d.a).

Segundo Stallings (2016), o AES é um algoritmo de criptografia simétrica que

utiliza a mesma chave para criptografar e descriptografar dados, sendo considerado seguro e eficiente para aplicações modernas. Sua utilização é comum em sistemas que necessitam proteger dados armazenados ou transmitidos, como mensagens e informações pessoais. O *AES* trabalha com blocos fixos de 128 bits e pode utilizar chaves de 128, 192 ou 256 bits. Na versão *AES-256*, utilizada neste trabalho, o algoritmo emprega uma chave criptográfica de 256 bits e executa 14 rodadas de processamento sobre os dados originais.

A Figura 2 ilustra o funcionamento geral do algoritmo *AES-256*.

Figura 2 – Esquema de funcionamento do AES-256



Fonte: Próprio autor

Como descrito no livro *"Cryptography and Network Security"* (STALLINGS, 2016, p. 171-206), inicialmente, o texto original é dividido em blocos de 128 bits organizados em forma matricial. Em seguida, ocorre a expansão da chave criptográfica, processo conhecido como *Key Schedule*, no qual a chave principal de 256 bits é utilizada para gerar subchaves específicas aplicadas em cada rodada do algoritmo.

A expansão da chave é uma etapa importante do *AES*, pois contribui para aumentar a complexidade criptográfica do processo e fortalecer a proteção contra ataques.

O processo de criptografia inicia com uma rodada inicial denominada *AddRoundKey*, responsável por combinar os dados originais com a primeira subchave utilizando operações *XOR*.

Após essa etapa, o *AES-256* executa 13 rodadas intermediárias compostas pelas seguintes operações:

- **SubBytes:** Cada byte do bloco é substituído por outro byte de acordo com uma tabela de substituição (S-box), introduzindo não-linearidade ao processo.
- **ShiftRows:** As linhas da matriz são deslocadas ciclicamente, aumentando a difusão dos dados.
- **MixColumns:** As colunas da matriz são combinadas usando operações matemáticas, promovendo a mistura dos dados.
- **AddRoundKey:** Combina os dados processados com a subchave correspondente da rodada.

Essas transformações são responsáveis pelos princípios de confusão e difusão descritos por Shannon (1949), dificultando a identificação de padrões nos dados criptografados.

A última rodada do *AES-256* possui estrutura semelhante às rodadas intermediárias, porém sem execução da etapa *MixColumns*.

Ao final do processo, o algoritmo gera o texto cifrado, que somente poderá ser recuperado corretamente utilizando a chave criptográfica correspondente.

Segundo Paar e Pelzl (2010, p .116), o *AES-256* é considerado um dos algoritmos simétricos mais seguros da atualidade devido à elevada complexidade computacional necessária para quebra da criptografia por força bruta.

Além da segurança, o *AES* apresenta bom desempenho computacional, sendo amplamente utilizado em aplicações *web*, sistemas bancários, redes de comunicação segura e armazenamento de informações sensíveis.

2.14.2 Hash de Senhas

O armazenamento seguro de senhas é um aspecto crítico em sistemas que utilizam autenticação de usuários. Diferentemente da criptografia, que permite a reversão dos dados mediante o uso de uma chave, as funções de hash são projetadas para serem irreversíveis, garantindo que a senha original não possa ser recuperada a partir do valor armazenado (DAS, 2023).

Entre as técnicas mais utilizadas, destaca-se o uso do algoritmo bcrypt, que aplica um processo de *hash* adaptativo com *salt*, aumentando a resistência contra ataques de força bruta e tabelas de arco-íris (rainbow tables). Segundo Provos e MAZIÈRES (1999), o *bcrypt* foi desenvolvido com o objetivo de tornar o processo de quebra de senhas computacionalmente custoso, ajustando seu fator de complexidade ao longo do tempo.

Dessa forma, o uso de funções de *hash* seguras como o *bcrypt* contribui significativamente para a proteção das credenciais dos usuários, sendo considerado uma boa prática no desenvolvimento de sistemas web.

2.14.3 Autenticação e Controle de Acesso

A autenticação é o processo de verificação da identidade de um usuário, sendo essencial para restringir o acesso a recursos do sistema. Entre as abordagens mais utilizadas, destacam-se os mecanismos baseados em *tokens*, como o *JSON Web Token (JWT)*, que permitem a autenticação de forma segura e escalável (BRADLEY; SAKIMURA; JONES, 2015).

De acordo com Bradley, Sakimura e Jones (2015), o *JWT* é um padrão aberto que permite a transmissão segura de informações entre partes, sendo amplamente utilizado em aplicações web para controle de sessões e autenticação de usuários.

Além disso, o uso combinado de *cookies* e *tokens* de atualização (*refresh tokens*) permite manter sessões seguras por períodos prolongados, sem comprometer a segurança do sistema. Esse modelo reduz a necessidade de autenticação constante e melhora a experiência do usuário, ao mesmo tempo em que mantém mecanismos de proteção contra acessos indevidos (HARDT, 2012).

2.14.4 Privacidade e Proteção de Dados Pessoais na Saúde Digital

A privacidade e a proteção de dados pessoais são aspectos essenciais no desenvolvimento de sistemas digitais voltados à saúde mental, uma vez que essas aplicações podem armazenar informações relacionadas à vida pessoal, aos relatos emocionais e ao acompanhamento psicológico dos usuários. No Brasil, o tratamento dessas informações deve considerar a Lei Geral de Proteção de Dados Pessoais (LGPD), instituída pela Lei nº 13.709/2018, que estabelece regras para o tratamento de dados pessoais em meios físicos e digitais, visando à proteção dos direitos fundamentais de liberdade e privacidade (Brasil, 2018).

No contexto da saúde digital, a proteção da privacidade também envolve o controle de acesso às informações e a definição de quais usuários podem consultar cada tipo de dado. Em plataformas que conectam pacientes e profissionais, por exemplo, é importante distinguir informações cadastrais, relatos compartilhados publicamente, conversas privadas e registros relacionados aos atendimentos. A existência de funcionalidades de anonimato não elimina, por si só, os riscos à privacidade, pois o conteúdo publicado ou a combinação de informações disponíveis pode permitir a identificação indireta de uma pessoa.

O emprego de recursos de inteligência artificial acrescenta outras considerações relacionadas à privacidade, pois os dados utilizados na análise automatizada podem conter informações pessoais ou sensíveis. Por esse motivo, é importante delimitar a finalidade do processamento, restringir o acesso aos resultados e avaliar quais informações são efetivamente necessárias para cada funcionalidade. No caso de sistemas que analisam relatos textuais ou expressões faciais, os resultados devem ser tratados com cautela,

considerando as limitações dos modelos e evitando que suas inferências sejam interpretadas como diagnósticos ou avaliações definitivas sobre o estado psicológico de um indivíduo.

Portanto, a proteção de dados na saúde digital não depende exclusivamente da adoção de mecanismos como criptografia e autenticação. Ela também exige decisões sobre a coleta, o armazenamento, o compartilhamento, a retenção e o acesso às informações, em conformidade com as obrigações legais aplicáveis. Esses aspectos são particularmente relevantes para plataformas de acompanhamento psicológico, nas quais a confidencialidade das informações contribui para a proteção dos usuários e para a utilização responsável dos recursos tecnológicos.

2.15 Metodologias Ágeis de Desenvolvimento

As metodologias ágeis surgiram como uma alternativa aos modelos tradicionais de desenvolvimento de software, propondo maior flexibilidade, adaptação a mudanças e foco na entrega contínua de valor. De acordo com Beck et al. (2001), por meio do Manifesto Ágil, esse conjunto de práticas prioriza a colaboração entre equipes, a interação com o cliente e a capacidade de responder rapidamente a alterações nos requisitos.

Dentre as abordagens ágeis mais utilizadas, destacam-se o Scrum e o Kanban, que organizam o desenvolvimento em ciclos iterativos e incrementais. Essas metodologias permitem que o software seja desenvolvido de forma progressiva, com entregas frequentes e validação contínua, reduzindo riscos e aumentando a qualidade do produto final.

No contexto de sistemas complexos, como aplicações voltadas à saúde digital, a adoção de metodologias ágeis contribui para maior organização do processo de desenvolvimento, possibilitando ajustes constantes e melhor alinhamento com as necessidades dos usuários.

2.16 Arquitetura de Software e Boas Práticas

A arquitetura de *software* desempenha um papel fundamental no desenvolvimento de sistemas complexos, sendo responsável por definir a estrutura, organização e interação entre os componentes de uma aplicação. Uma arquitetura bem definida contribui para a escalabilidade, manutenção e evolução do sistema ao longo do tempo.

Entre as abordagens modernas de arquitetura, destaca-se o *Domain-Driven Design* (*DDD*), proposto por Evans (2003), que enfatiza a modelagem do software com base no domínio do problema. O *DDD* busca alinhar o desenvolvimento técnico com as regras de negócio, promovendo uma separação clara entre as camadas da aplicação, como domínio, aplicação e infraestrutura. Essa abordagem facilita a organização do código, melhora a

comunicação entre desenvolvedores e especialistas do domínio e contribui para a construção de sistemas mais coesos e compreensíveis.

Além disso, o uso de boas práticas de engenharia de software, como os princípios *SOLID*, é essencial para garantir a qualidade do código. De acordo com Martin (2000), os princípios *SOLID* consistem em um conjunto de diretrizes que visam tornar o software mais modular, flexível e de fácil manutenção. Entre esses princípios, destaca-se o Princípio da Inversão de Dependência (*DIP*), que recomenda que módulos de alto nível não dependam diretamente de módulos de baixo nível, mas sim de abstrações, promovendo baixo acoplamento entre os componentes do sistema.

A aplicação desses conceitos permite o desenvolvimento de sistemas mais organizados e adaptáveis a mudanças, o que é especialmente importante em aplicações que envolvem múltiplos módulos e funcionalidades complexas. Dessa forma, a adoção de arquiteturas bem definidas e boas práticas de desenvolvimento contribui significativamente para a qualidade e longevidade do software.

2.17 Trabalhos Relacionados

Dhaliwal et al. (2023) realizaram uma revisão de escopo sobre a psicoterapia síncrona oferecida remotamente por profissionais de saúde mental. O estudo analisou a qualidade desses serviços a partir de seis dimensões: aceitabilidade, acessibilidade, adequação, efetividade, eficiência e segurança. A revisão incluiu 48 artigos científicos e identificou que a efetividade era a dimensão mais frequentemente avaliada, enquanto aspectos como acessibilidade, adequação, eficiência e segurança recebiam menor atenção. Os autores também apontaram limitações relacionadas à diversidade geográfica, cultural e clínica das populações estudadas. Esse trabalho contribui para a compreensão dos desafios associados à oferta de serviços de saúde mental por meios digitais, evidenciando a importância de avaliar a qualidade dessas intervenções para além da disponibilização de recursos de comunicação remota.

Philippe et al. (2022) realizaram uma revisão abrangente da literatura sobre intervenções digitais voltadas ao cuidado em saúde mental. O estudo reuniu 304 trabalhos de revisão e contemplou diferentes modalidades de intervenção, incluindo comunicação síncrona e assíncrona entre pacientes e profissionais, programas terapêuticos on-line, aplicativos móveis, comunidades de apoio e tecnologias de monitoramento. Os resultados indicaram evidências favoráveis para determinadas intervenções digitais, embora a qualidade e a consistência dessas evidências variassem conforme a condição de saúde e a modalidade utilizada. Os autores também destacaram a necessidade de avaliações mais rigorosas e de pesquisas que considerem condições e populações ainda pouco estudadas. Diferentemente de Dhaliwal et al. (2023), que concentraram a análise na psicoterapia

síncrona remota, esse trabalho apresenta um panorama mais amplo das possibilidades de utilização de tecnologias digitais no cuidado em saúde mental.

PÉREZ-HERNÁNDEZ et al. (2021) desenvolveram um protótipo de plataforma web de telepsicologia, cujo objetivo é permitir que pacientes e terapeutas acompanhem tratamentos psicológicos remotamente e de forma assíncrona. A solução utiliza recursos de computação em nuvem e incorpora dispositivos biomédicos para coletar medidas relacionadas ao estresse e ao humor dos pacientes. Além disso, apresenta um componente baseado em *Deep Learning* para detectar pontos de referência faciais e estimar características emocionais. Os autores descrevem a arquitetura do protótipo e sua integração com esses recursos, discutindo sua aplicação no acompanhamento remoto de pacientes.

Esse último trabalho apresenta maior proximidade tecnológica com o TherapistFriend, especialmente pela integração entre uma plataforma de acompanhamento psicológico e recursos de inteligência artificial voltados à análise de características faciais. Entretanto, existem diferenças entre as abordagens: enquanto PÉREZ-HERNÁNDEZ et al. (2021). propõem o acompanhamento remoto e assíncrono de terapias, o TherapistFriend reúne funcionalidades de publicação de relatos, aproximação entre pacientes e profissionais por meio de solicitações de conversas privadas e realização de sessões on-line. Além disso, a plataforma desenvolvida neste trabalho incorpora análise textual de relatos por meio de um microsserviço de inteligência artificial e classificação de expressões faciais durante as sessões.

Em conjunto, os trabalhos apresentados abordam diferentes aspectos da utilização de tecnologias digitais na saúde mental. Enquanto os estudos de Dhaliwal et al. (2023) e Philippe et al. (2022) contribuem para a compreensão das intervenções digitais e dos desafios relacionados à sua qualidade, o protótipo de PÉREZ-HERNÁNDEZ et al. (2021) apresenta uma implementação tecnológica mais diretamente relacionada à proposta deste trabalho. O TherapistFriend diferencia-se pela integração de recursos de comunicação, acompanhamento e análise auxiliar por inteligência artificial em uma única plataforma. Ressalta-se, contudo, que a presença dessas funcionalidades não representa, isoladamente, evidência de efetividade clínica, sendo necessária uma avaliação específica para investigar seus possíveis benefícios no contexto da saúde mental.

3 Materiais e Método

Este capítulo apresenta os materiais, tecnologias, arquiteturas e métodos utilizados no desenvolvimento do sistema proposto neste trabalho. São descritas as ferramentas empregadas no *backend*, *frontend* e microsserviços, bem como as abordagens utilizadas para comunicação em tempo real, segurança, processamento assíncrono e inteligência artificial.

O desenvolvimento do sistema teve como objetivo a construção de uma plataforma web voltada à conexão entre pacientes e profissionais da saúde mental, permitindo comunicação anônima, interação em tempo real, análise emocional e gerenciamento de atendimentos.

3.1 Equipamento

O desenvolvimento e a execução foram realizadas primeiramente em um notebook equipado com um processador *Intel Core i5-7200u*, 8 *GB* de memória *RAM*, *SSD SATA 250 GB* e uma placa de vídeo *Intel HS Graphics 620*. Posteriormente houve mudança para um notebook equipado com um processador *Intel Core i7-13650HX (2.64 GHz)*, 32 GB de memória *RAM*, *SSD 500 GB* e uma placa de vídeo *NVIDIA GeForce RTX 3050 6 GB*.

3.2 Caracterização da Pesquisa

A presente pesquisa caracteriza-se como aplicada e exploratória. É considerada aplicada por envolver o desenvolvimento de uma solução tecnológica voltada a um problema prático: a disponibilização de recursos digitais que facilitem a comunicação e a interação entre pacientes e profissionais da saúde mental.

Também apresenta caráter exploratório, uma vez que envolve a investigação e a aplicação de diferentes tecnologias para integrar funcionalidades de comunicação em tempo real, acompanhamento remoto e ferramentas auxiliares de inteligência artificial no contexto da saúde mental.

Quanto aos procedimentos, o trabalho foi conduzido por meio do desenvolvimento incremental de uma aplicação web. As funcionalidades foram implementadas e integradas progressivamente, contemplando desde os mecanismos de cadastro e autenticação até os recursos de comunicação privada, agendamento, sessões on-line e análise automatizada de informações.

Durante o desenvolvimento, foram realizadas demonstrações da plataforma a duas

profissionais da psicologia, com o objetivo de apresentar a proposta e obter observações relacionadas à sua utilização no contexto da saúde mental. Embora essa participação não tenha seguido um protocolo estruturado de avaliação, com roteiro de perguntas ou instrumento padronizado de coleta de dados, as profissionais apresentaram sugestões relacionadas à identidade visual, à organização dos fluxos de utilização e à apresentação das informações disponibilizadas pelo sistema.

Entre as contribuições recebidas, destacou-se a sugestão de aprimorar a apresentação dos resultados da identificação de expressões faciais durante as sessões on-line, por meio de indicadores visuais com cores, complementando as informações textuais. Também foram apresentadas sugestões sobre a organização da interface e dos fluxos de utilização. Essas observações foram consideradas durante o desenvolvimento e resultaram em ajustes implementados na plataforma. Essa participação é caracterizada como uma contribuição consultiva de profissionais da área, não constituindo uma validação clínica ou uma avaliação sistemática da eficácia da solução.

3.3 Problema de Pesquisa e Objetivos

A proposta de desenvolvimento da plataforma TherapistFriend surgiu da necessidade de disponibilizar um ambiente digital que facilite a conexão e a comunicação entre pacientes e profissionais da saúde mental, reunindo recursos que permitam a publicação de relatos, a interação privada e o acompanhamento remoto. A partir dessa necessidade, estabeleceu-se como problema de pesquisa a seguinte questão: Como desenvolver uma plataforma web que integre recursos de comunicação entre pacientes e profissionais da saúde mental, acompanhamento remoto e ferramentas auxiliares de inteligência artificial?

Diante dessa questão, o objetivo geral do trabalho consiste em desenvolver uma plataforma web que disponibilize recursos para aproximar pacientes e profissionais da saúde mental, integrando funcionalidades de comunicação, gerenciamento de agendamentos, realização de sessões on-line e ferramentas auxiliares de inteligência artificial.

Para alcançar esse objetivo, foram definidos os seguintes objetivos específicos:

- Desenvolver mecanismos de cadastro, autenticação e gerenciamento de usuários, considerando os diferentes perfis de acesso;
- Implementar a publicação de relatos, permitindo que pacientes escolham entre a identificação ou o anonimato;
- Disponibilizar recursos de comunicação privada entre pacientes e profissionais, incluindo mensagens em tempo real;

- Implementar funcionalidades de agendamento e realização de sessões on-line com comunicação por áudio e vídeo;
- Desenvolver recursos de acompanhamento, incluindo o registro diário de humor e a disponibilização de conteúdos personalizados;
- Integrar um serviço de inteligência artificial para análise textual dos relatos;
- Desenvolver um serviço de visão computacional para identificação de categorias de expressões faciais durante as sessões on-line;
- Criar relatórios das sessões online baseado nas análises das expressões obtidas;
- Implementar mecanismos de segurança e proteção das informações dos usuários.

3.4 Levantamento e Especificação dos Requisitos

O levantamento dos requisitos ocorreu durante a concepção e o desenvolvimento da plataforma, considerando as necessidades associadas aos diferentes perfis de usuários e às funcionalidades previstas para o sistema. Embora não tenha sido elaborada uma especificação integral dos requisitos antes do início da implementação, parte dessas definições foi registrada por meio de anotações, diagramas e casos de uso ao longo do projeto.

Para fins de documentação, os requisitos foram consolidados nesta etapa e organizados em requisitos funcionais e não funcionais. Os requisitos funcionais descrevem os serviços e operações disponibilizados pela plataforma, enquanto os não funcionais representam características relacionadas à segurança, organização e funcionamento do sistema.

3.4.1 Requisitos Funcionais

Os requisitos funcionais identificam as principais funcionalidades disponibilizadas pelo TherapistFriend, considerando as operações realizadas por pacientes, profissionais da psicologia e administradores.

Tabela 1 – Requisitos Funcionais.

Identificação	**Requisito funcional**
RF01	Permitir o cadastro, a autenticação e o encerramento de sessão dos usuários.
RF02	Permitir o gerenciamento dos perfis de pacientes e profissionais.
RF03	Permitir que pacientes publiquem relatos de maneira anônima ou identificada.
RF04	Permitir a visualização dos relatos conforme sua modalidade de publicação, disponibilizando os relatos públicos aos visitantes e os demais aos profissionais autenticados.
RF05	Permitir que profissionais solicitem o início de uma conversa privada a partir de um relato.
RF06	Permitir que pacientes aceitem ou recusem solicitações de conversa.
RF07	Disponibilizar comunicação privada entre pacientes e profissionais por meio de mensagens.
RF08	Permitir o registro diário de humor pelos pacientes e a disponibilização de conteúdos personalizados relacionados a esses registros.
RF09	Permitir que profissionais configurem sua disponibilidade para agendamentos.
RF10	Permitir que pacientes solicitem agendamentos e que profissionais gerenciem as solicitações recebidas.
RF11	Permitir a realização de sessões on-line com comunicação por áudio e vídeo.
RF12	Disponibilizar recursos de análise textual dos relatos por meio de um serviço de inteligência artificial.
RF13	Disponibilizar a identificação de categorias de expressões faciais durante as sessões on-line.
RF14	Permitir que profissionais consultem relatórios gerados a partir das informações coletadas durante as sessões on-line.
RF15	Permitir que administradores analisem e aprovem os cadastros de profissionais.
RF16	Disponibilizar notificações relacionadas a mensagens, solicitações, agendamentos e outras interações realizadas na plataforma.

3.4.2 Requisitos Não Funcionais

Os requisitos não funcionais estabelecem características relacionadas à segurança, à organização arquitetural, à comunicação e ao funcionamento da plataforma. Esses requisitos foram consolidados a partir das decisões técnicas adotadas durante o desenvolvimento, considerando as necessidades de proteção das informações e integração entre os componentes do sistema.

Tabela 2 – Requisitos Não Funcionais.

Identificação	**Requisito não funcional**
RNF01	Utilizar mecanismos de autenticação e gerenciamento de sessão para controlar o acesso à plataforma.
RNF02	Restringir o acesso às funcionalidades e informações de acordo com o perfil e as permissões de cada usuário.
RNF03	Proteger as credenciais dos usuários por meio de mecanismos de hash criptográfico.
RNF04	Proteger o conteúdo das mensagens privadas por meio de criptografia AES-256.
RNF05	Utilizar cookies com atributos de segurança, incluindo HttpOnly, para o armazenamento dos tokens de autenticação.
RNF06	Implementar mecanismos de validação de dados de entrada para reduzir o processamento de informações inválidas.
RNF07	Disponibilizar mecanismos de comunicação em tempo real para mensagens, notificações e sinalização das sessões on-line.
RNF08	Permitir o processamento assíncrono de tarefas por meio de mecanismos de filas, evitando concentrar todas as operações no fluxo principal das requisições.
RNF09	Organizar a aplicação em módulos com responsabilidades definidas, favorecendo a manutenção e a evolução do sistema.
RNF10	Permitir a comunicação entre os componentes da arquitetura por meio de interfaces e protocolos definidos.
RNF11	Disponibilizar uma interface adaptável a diferentes tamanhos de tela e dispositivos.

A consolidação desses requisitos estabelece as principais características técnicas consideradas na construção da plataforma, servindo como referência para a apresentação das decisões arquiteturais e dos mecanismos implementados nas seções seguintes

3.5 Procedimentos Metodológicos

O desenvolvimento da plataforma TherapistFriend foi conduzido de maneira incremental, dividido em etapas que permitiram implementar e integrar progressivamente os diferentes componentes da aplicação.

Inicialmente, foram identificadas as necessidades que orientaram a proposta do sistema, considerando os diferentes perfis de usuários e as funcionalidades necessárias para possibilitar a interação entre pacientes e profissionais da saúde mental. A partir desse levantamento, foram definidos os principais recursos da plataforma e elaborados diagramas para representar os casos de uso, os fluxos de utilização e a organização dos componentes.

Na etapa seguinte, foi estabelecida a arquitetura da aplicação, contemplando a separação entre *frontend*, *backend* principal, banco de dados e serviços auxiliares de

inteligência artificial. Também foram definidas as tecnologias utilizadas para a comunicação entre os componentes e o processamento das tarefas executadas pela plataforma.

Posteriormente, iniciou-se a implementação dos módulos funcionais, abrangendo os mecanismos de cadastro e autenticação, gerenciamento de usuários, publicação de relatos, comunicação privada, notificações, agendamentos e sessões on-line. Os recursos foram desenvolvidos progressivamente e integrados ao *backend* principal, permitindo a evolução da aplicação conforme as funcionalidades eram implementadas.

Em paralelo, foram desenvolvidos os serviços auxiliares de inteligência artificial. O primeiro foi destinado à análise textual dos relatos e à geração de conteúdos personalizados relacionados aos registros de humor. O segundo foi desenvolvido para identificar categorias de expressões faciais durante as sessões on-line, utilizando um modelo de visão computacional treinado com dados do conjunto *AffectNet*.

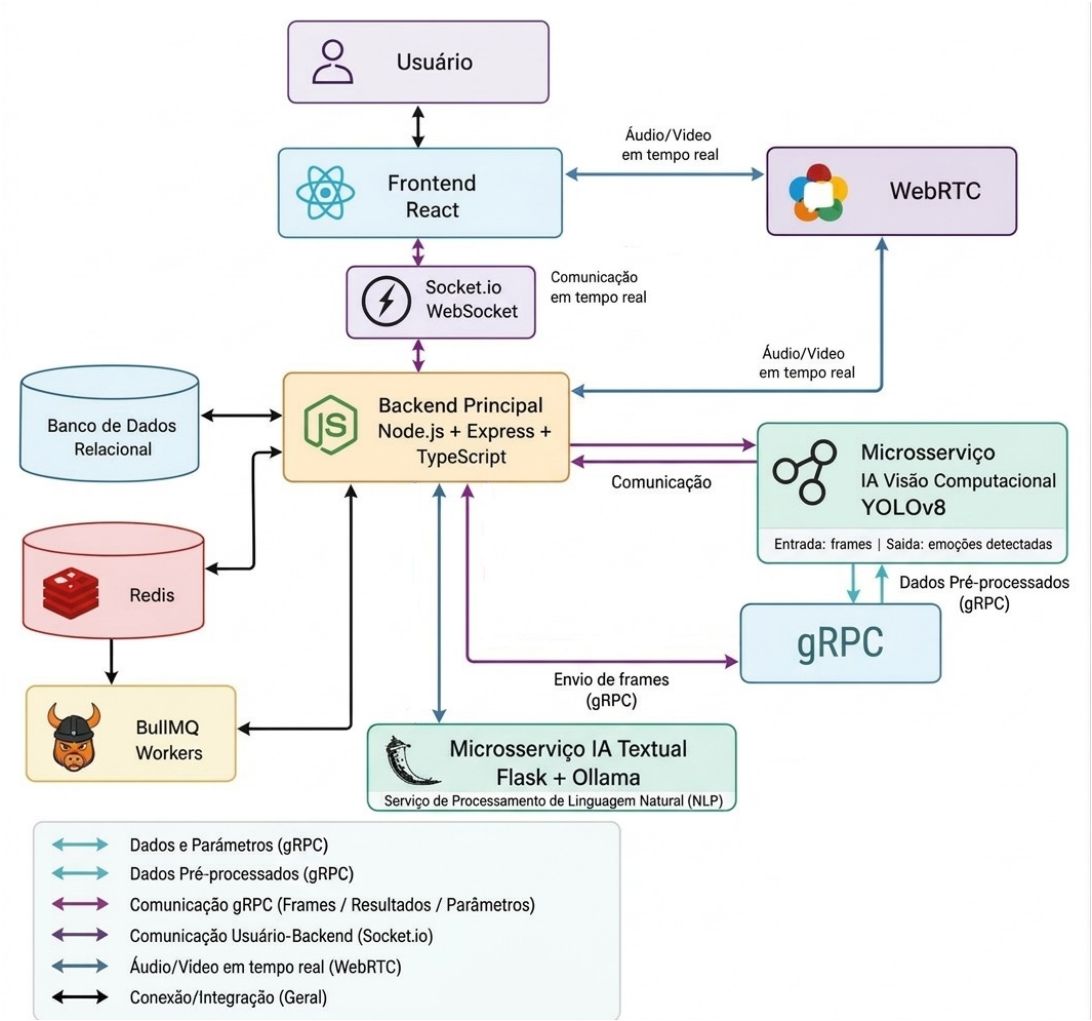
Após a implementação dos componentes, foram realizadas atividades de integração e testes manuais, abrangendo os fluxos funcionais, os cenários de erro e a comunicação entre os diferentes componentes da arquitetura. Também foram realizadas demonstrações da plataforma a duas profissionais da psicologia, cujas observações contribuíram para ajustes na interface, na organização dos fluxos e na apresentação dos resultados da identificação de expressões faciais.

Por fim, foram analisados os resultados obtidos com o treinamento do modelo de visão computacional, considerando as métricas disponibilizadas durante os experimentos. Os procedimentos e resultados dessa avaliação são apresentados nas seções específicas deste trabalho.

3.6 Arquitetura Geral do Sistema

A arquitetura do sistema foi desenvolvida utilizando uma abordagem distribuída, modular e orientada a serviços, com o escopo de promover escalabilidade, desacoplamento rigoroso entre componentes e uma clara segregação de responsabilidades arquiteturais. A topologia estrutural da plataforma segmenta-se em três macrocamadas principais: o *frontend* cliente, o *backend* monolítico principal e os microsserviços analíticos especializados. Adicionalmente, a infraestrutura mobiliza mecanismos de barramento de mensagens e armazenamento volátil para viabilizar comunicações duplex em tempo real e processamentos assíncronos de alta performance. O arranjo topológico e o mapeamento dos fluxos de dados entre estes componentes integrados encontram-se ilustrados na Figura 3.

Figura 3 – Diagrama de estrutura da arquitetura geral do sistema



Fonte: Próprio autor

Na periferia do ecossistema, a camada de *frontend* rege a interface do usuário e a experiência de interatividade, conectando-se ao *backend* principal — o núcleo central das regras de negócio — por meio de requisições *HTTP* REST e canais bidirecionais via *WebSocket*. Este servidor central centraliza fluxos críticos como autenticação, controle de sessões e gerenciamento de usuários. Para a persistência de longo prazo de dados estruturados e relacionais (tais como registros de profissionais, pacientes, agendamentos e prontuários), o ecossistema utiliza um banco de dados relacional. Em paralelo, a infraestrutura adota o *Redis* como camada de dados volátil e temporária; essa tecnologia atua em conjunto com a biblioteca *BullMQ* para gerenciar filas de processamento assíncrono e executar tarefas pesadas em segundo plano, mitigando gargalos de I/O no fluxo principal da aplicação.

Ramificando-se do núcleo da plataforma, o ecossistema estende sua capacidade computacional através de dois microsserviços autônomos e especializados em inteligência artificial. O primeiro deles, dedica-se ao processamento de linguagem natural (*NLP*). Ele

analisa textualmente os relatos clínicos de desabafo e gera estímulos cognitivos personalizados (*cards* de suporte) acionando modelos de linguagem de grande porte (*LLMs*) locais controlados pelo ambiente *Ollama*. O segundo microsserviço foca-se em visão computacional e inferência emocional em tempo real, empregando uma rede neural convolucional profunda baseada na arquitetura *YOLOv8* (variante *Nano*). A integração técnica entre o *backend* principal e este módulo de análise facial é viabilizada pelo protocolo *gRPC* (*gRPC Remote Procedure Calls*), cuja escolha se justifica pela necessidade de transporte binário de alta eficiência e latência ultra-reduzida.

Esta topologia distribuída converge de forma síncrona durante a realização das teleconsultas na plataforma. Enquanto a transmissão bidirecional de áudio e vídeo entre o paciente e o psicólogo ocorre de maneira direta ponto a ponto utilizando o protocolo *WebRTC*, o navegador do cliente atua como um coletor contínuo de telemetria. Periodicamente, as matrizes de *frames* visuais são amostradas e despachadas do *frontend* para o *backend* via conexões *WebSocket*. Ao receber essa carga de dados multimídia, o servidor de aplicação atua como um *gateway* de roteamento, encaminhando o *buffer* de imagem instantaneamente através do canal *gRPC* para o microsserviço de visão computacional, completando o pipeline de retorno analítico e *feedback* emocional sem interferir na estabilidade da conferência médica.

3.7 Backend Principal

O núcleo analítico e operacional da plataforma está centralizado no *backend* principal, desenvolvido sob o ecossistema *Node.js* utilizando o *framework Express* e a linguagem *TypeScript*. A convergência dessas tecnologias viabilizou a construção de uma camada servidora de alta performance, onde a tipagem estática e explícita do *TypeScript* atuou como um mecanismo preditivo de mitigação de inconsistências em tempo de compilação, elevando a manutenibilidade global do código-fonte. O *Express*, por sua vez, operou de forma minimalista na abstração e roteamento de primitivas *HTTP*, servindo como alicerce maleável para o ecossistema estrutural da aplicação.

Para mitigar a complexidade das interações do sistema e evitar o acoplamento entre componentes, a arquitetura interna foi concebida sob as diretrizes do Domain-Driven Design (DDD) e da arquitetura em camadas (Domínio, Aplicação, Infraestrutura e Interface). O ecossistema foi segmentado em contextos delimitados voltados ao negócio — abrangendo as frentes de gestão clínica de pacientes e profissionais, catalogação de especialidades, agendamentos, monitoramento comportamental e comunicação interativa — e apoiado por subdomínios genéricos e transversais para administração, autenticação e notificações. A integração entre esses módulos foi viabilizada de forma assíncrona e desacoplada por meio do padrão de Eventos de Domínio (*Domain Events*), utilizando um despachante de

eventos (*EventDispatcher*) na camada de aplicação para comunicar ocorrências do domínio à infraestrutura, que executa operações persistentes e de comunicação em tempo real via *WebSockets*.

Esta modularização estratégica é suportada empiricamente pela aplicação sistemática dos princípios *SOLID*. Destaca-se a implementação do Princípio da Inversão de Dependência (*Dependency Inversion Principle - DIP*), o qual dita que os módulos de alto nível (regras de negócio) nunca devem depender diretamente de implementações de baixo nível (como drivers de bancos de dados ou clientes de rede), mas sim de abstrações de interfaces. Essa blindagem arquitetural assegura que o *backend* atue como um intermediador coeso e resiliente, centralizando as políticas de segurança, criptografia, autenticação e autorização do ecossistema ao coordenar as trocas de estado com o *frontend*, o banco de dados relacional, a camada de *cache* com o *Redis* e os microsserviços de inteligência artificial.

3.8 Banco de Dados

O sistema utiliza banco de dados relacional *PostgreSQL* para armazenamento e gerenciamento das informações da aplicação. A modelagem foi estruturada de forma a garantir organização, integridade e consistência dos dados, permitindo o relacionamento entre os diferentes módulos.

A estrutura do banco foi desenvolvida considerando as necessidades da plataforma, incluindo gerenciamento de usuários, pacientes, profissionais, relatos, mensagens, sessões, notificações, agendamentos e registro emocional.

3.8.1 Modelagem Relacional

A modelagem relacional foi adotada devido à necessidade de organização estruturada dos dados e ao relacionamento entre as entidades da aplicação. Essa abordagem permite maior consistência das informações e facilita operações de consulta, atualização e controle de integridade.

As tabelas foram estruturadas utilizando chaves primárias e estrangeiras para garantir o relacionamento adequado entre os registros, evitando redundância de dados e inconsistências.

3.8.2 Estrutura das Entidades

A arquitetura do banco de dados foi organizada de forma modular, separando as entidades de acordo com suas responsabilidades dentro do sistema.

A entidade de usuários é responsável pelo armazenamento das informações principais de autenticação e identificação dos usuários da plataforma. A partir dela, derivam entidades específicas relacionadas aos pacientes e profissionais, permitindo separação das informações conforme o perfil do usuário.

Além disso, o banco possui entidades voltadas ao gerenciamento de relatos, mensagens privadas, sessões de atendimento, notificações, especialidades, agendamentos e autenticação, permitindo integração entre os diferentes módulos do sistema.

O DER (Diagrama entidade relacionamento) será apresentado na Seção 4.2

3.8.3 Relacionamentos

Os relacionamentos entre as entidades foram estruturados para permitir integração entre pacientes e profissionais durante o fluxo de utilização da plataforma.

Essa estrutura relacional contribui para organização dos dados e facilitou a implementação das regras de negócio da aplicação.

3.8.4 Persistência e Integridade dos Dados

Para garantir a integridade das informações armazenadas, foram utilizadas restrições relacionais e validações durante as operações realizadas pela aplicação.

Além disso, o sistema implementa mecanismos de autenticação e controle de acesso para proteção dos dados armazenados, considerando a natureza sensível das informações relacionadas à saúde mental dos usuários.

A persistência estruturada dos dados contribui para maior confiabilidade da aplicação e possibilita o gerenciamento adequado das informações geradas pelos diferentes módulos do sistema.

3.9 Comunicação em Tempo Real e Processamento Assíncrono

Para garantir maior responsividade, mitigar o acoplamento entre componentes e otimizar o gerenciamento de rotinas concorrentes, a arquitetura da plataforma integra mecanismos de comunicação em tempo real e processamento assíncrono. A transmissão instantânea de dados entre o *frontend* e o servidor principal é viabilizada por meio do protocolo *WebSocket*, implementado via *Socket.io*. Essa abordagem mitiga a necessidade de requisições intermitentes ao servidor (*polling*), viabilizando a atualização dinâmica da interface sem a necessidade de recarregamento da página. Na prática, esse fluxo de comunicação bidirecional sustenta funcionalidades críticas do sistema, tais como a troca de mensagens em *chats*, o envio de notificações imediatas, o gerenciamento de solicitações

de conversa, a atualização de estados em sessões clínicas, a transmissão de inferências emocionais e o retorno visual de relatórios gerados dinamicamente.

Paralelamente, a execução de tarefas computacionalmente onerosas em segundo plano é gerenciada por uma infraestrutura de mensageria baseada em filas, utilizando o *Redis* como armazenamento em memória em conjunto com a biblioteca *BullMQ* para a orquestração de *jobs*. Essa topologia impede o bloqueio do fluxo de execução principal da aplicação, delegando rotinas complexas para operarem de forma assíncrona. Dentre os processos secundários submetidos a esse pipeline, destacam-se a análise interpretativa de relatos textuais, a geração automatizada de relatórios estatísticos, a confecção de *cards* personalizados para o usuário e o disparo assíncrono do subsistema de notificações.

Complementando essa dinâmica, a plataforma adota uma arquitetura orientada a eventos por meio de Eventos de Domínio (*Domain Events*) (MICROSOFT, 2026). Sob essa óptica, ações primárias disparadas pelos usuários na camada de aplicação publicam ocorrências de negócio de forma agnóstica. Esses eventos são interceptados por manipuladores secundários que desencadeiam os fluxos assíncronos descritos sem reter ou impactar o tempo de resposta da requisição original, promovendo uma clara segregação de responsabilidades e elevando a escalabilidade global do ecossistema.

3.10 Frontend da Aplicação

A camada de interface e interação direta com o usuário foi materializada através da biblioteca *React*, uma tecnologia baseada no ecossistema *JavaScript* voltada à construção de interfaces SPA (*Single Page Applications*) altamente dinâmicas. A escolha do *React* fundamenta-se em sua arquitetura declarativa e orientada a componentes modulares e reutilizáveis, modelo que propicia uma segregação estrita entre lógica de estado e renderização visual. No escopo da plataforma, essa componentização garantiu que elementos complexos fossem fracionados em estruturas menores e de responsabilidade única, simplificando os ciclos de manutenção corretiva e evolução de recursos. Para a estilização e identidade visual das telas, optou-se pelo uso de *CSS* puro de forma customizada, assegurando controle absoluto sobre as folhas de estilo sem a sobrecarga de dependências de *frameworks* utilitários terceiros. Esse desenho visual foi projetado sob os pilares da responsividade, viabilizando uma adaptação fluida do *layout* a diferentes resoluções e dispositivos.

O fluxo informacional e a sincronia entre componentes distantes na árvore de renderização são orquestrados nativamente através da *Context API* do *React* (REACT, 2026). Esta abordagem substituiu de forma eficiente o padrão de acoplamento vertical conhecido como *prop drilling*, centralizando o estado global em provedores (*Providers*) especializados. No ecossistema cliente, essa malha de dados gerencia ciclos vitais transversais, englobando o ciclo de autenticação, o perfil e metadados do usuário logado, o histórico de mensagens

ativas do *chat*, o armazenamento volátil de *cards* cognitivos personalizados emitidos pelo microsserviço de LLM e o barramento de recepção de *Notificações* assíncronas em tempo real.

A camada de interface integra um conjunto de tecnologias periféricas e APIs nativas voltadas ao suporte das funcionalidades de teleconsulta, comunicação e gestão clínica. A comunicação audiovisual direta e descentralizada entre o paciente e o profissional de saúde é viabilizada pela tecnologia *WebRTC* (*Web Real-Time Communication*), utilizada para a transmissão estável de fluxos de áudio e vídeo em topologia *peer-to-peer*. Para a troca contínua de eventos assíncronos e atualizações dinâmicas da interface com o servidor, a camada cliente incorpora a instância do *Socket.io*, estabelecendo o canal estável de comunicação bidirecional. No escopo dos componentes visuais de gerenciamento, o sistema adota a biblioteca *React Big Calendar* para prover o controle interativo de agendas, horários disponíveis e organização de consultas. Complementarmente, a exibição de alertas informativos e respostas imediatas do sistema a ações do usuário é gerenciada por um subsistema de notificações visuais temporárias do tipo *Toast*, assegurando uma experiência fluida e responsiva.

3.11 Microsserviço de Inteligência Artificial Textual

O suporte cognitivo e a capacidade de processamento de linguagem natural (*Natural Language Processing - NLP*) da plataforma são providos por um microsserviço especializado, desenvolvido de forma autônoma e desacoplada do servidor principal. Esta camada foi estruturada na linguagem *Python* utilizando o *framework Flask*, cuja escolha técnica fundamenta-se em sua flexibilidade arquitetural, caráter minimalista e na maturidade do ecossistema para a integração de ferramentas de aprendizado de máquina. O núcleo de inferência baseia-se na execução local do modelo fundacional de grande porte *Llama 3* (variante de 8 bilhões de parâmetros, *Llama 3 8b*) (OLLAMA, s.d.), gerenciado por meio do ambiente *Ollama*. A adoção dessa topologia de implantação local destina-se a eliminar a dependência de serviços externos e APIs proprietárias, assegurando a soberania, a governança e o sigilo no tratamento de dados de saúde sensíveis.

Funcionalmente, o microsserviço cumpre o propósito de viabilizar a triagem clínica preditiva e a personalização de estímulos informativos. No âmbito da triagem, a tecnologia é empregada para realizar a análise de sentimentos e a classificação de risco em relatos textuais, gerando diagnósticos estruturados que atuam como indicadores clínicos preliminares para auxiliar os profissionais na identificação de estados de urgência. No escopo da personalização, o microsserviço é utilizado para sintetizar conteúdos informativos contextualizados — compostos por orientações práticas e mensagens motivacionais — mapeados a partir dos registros emocionais dos usuários.

3.12 Microsserviço de Visão Computacional

O processamento de visão computacional e o reconhecimento de expressões faciais são executados por um microsserviço especializado, desenvolvido na linguagem *Python* de forma independente e desacoplada do servidor principal para isolar as operações de alto custo computacional. A comunicação interserviços entre o *backend* principal e este componente é viabilizada pelo protocolo *gRPC* (*Google Remote Procedure Call*), cuja adoção destina-se a garantir baixa latência e alta eficiência na transmissão de dados.

No escopo da inteligência artificial, o microsserviço emprega o modelo preditivo *YOLOv8n* com a finalidade de realizar a detecção facial e a subsequente inferência emocional em tempo real. A seleção técnica desta solução decorreu de testes comparativos prévios envolvendo diferentes arquiteturas de redes neurais e conjuntos de dados. O modelo final utiliza como base de conhecimento pesos decorrentes do treinamento com o *dataset* público *AffectNet* adaptado para o formato *YOLO*, provendo o mapeamento de características faciais distribuídas em oito categorias emocionais distintas: raiva, desprezo, nojo, medo, felicidade, neutralidade, tristeza e surpresa.

3.12.1 Processo de Escolha do Modelo

A definição do modelo de visão computacional utilizado na plataforma ocorreu a partir de uma etapa exploratória de experimentação, na qual foram avaliadas diferentes arquiteturas de redes neurais e conjuntos de dados voltados ao reconhecimento de expressões faciais. Inicialmente, foram realizados experimentos com as arquiteturas *ResNet-50* e *EfficientNet-B0*, utilizando o conjunto de dados *FER-2013*.

Nessa primeira etapa, a *ResNet-50* obteve uma acurácia de 62,16%, enquanto a *EfficientNet-B0* alcançou 69,82%. Apesar dos resultados obtidos, foram identificadas limitações relacionadas às características das imagens presentes no *FER-2013*, que apresenta registros com diferentes níveis de ruído e qualidade visual. Além disso, o tempo necessário para realizar os treinamentos dessas arquiteturas representava uma dificuldade diante da capacidade computacional disponível para o desenvolvimento do projeto.

Diante dessas condições, optou-se por investigar uma arquitetura mais leve, que pudesse atender à necessidade de processamento contínuo de imagens durante as sessões on-line. Assim, foi selecionado o *YOLOv8n*, uma variante de menor porte da família *YOLOv8*, em conjunto com o dataset *AffectNet*, adaptado para o formato *YOLO*. Essa mudança também permitiu utilizar uma abordagem de detecção de objetos para localizar e classificar as expressões faciais nas imagens.

Após essa escolha, foram realizados dois experimentos para avaliar a influência da resolução das imagens de entrada no desempenho do modelo. Em ambos, foram mantidos os mesmos parâmetros de treinamento, alterando-se apenas a resolução, de 96 x 96 para

224 x 224 pixels. Os treinamentos foram configurados para 100 épocas, com tamanho de lote (batch size) igual a 64 e utilização do otimizador *AdamW*.

Os resultados foram analisados por meio das métricas de precisão (*Precision*), revocação (*Recall*), mAP@50 e mAP@50-95, obtidas durante a validação. A Tabela 3. apresenta os maiores valores registrados em cada métrica ao longo dos dois treinamentos.

Tabela 3 – Comparativo de desempenho do modelo YOLO entre as resoluções 96x96 e 224x224.

Métrica	**96 x 96**	**224 x 224**
Precisão	68,00%	73,40%
Revocação	73,00%	78,00%
mAP@50	78,00%	83,11%
mAP@50-95	77,80%	83,75%

Os resultados demonstraram que o treinamento com resolução de 224 x 224 pixels apresentou valores superiores em todas as métricas analisadas. Em relação à resolução de 96 x 96 pixels, foram observados aumentos de 5,4 pontos percentuais na precisão, 5,0 pontos percentuais na revocação, 5,11 pontos percentuais no mAP@50 e 5,95 pontos percentuais no mAP@50-95.

Com base nessa comparação, foi adotada a resolução de 224 x 224 pixels na configuração final do modelo, por apresentar os resultados mais satisfatórios nas métricas de validação utilizadas. Essa configuração passou a ser utilizada no microsserviço responsável pela identificação de categorias de expressões faciais durante as sessões on-line.

3.13 Segurança da Aplicação

Dada a natureza sensível e confidencial das informações clínicas manipuladas, a plataforma integra múltiplos mecanismos de segurança projetados para garantir o controle de acesso, a proteção de dados e a integridade das comunicações. O processo de autenticação baseia-se no padrão *JSON Web Token (JWT)* (IBM, 2024) em conjunto com *refresh tokens*, viabilizando o gerenciamento seguro e a renovação de sessões sem exigir reautenticações frequentes do usuário. Para mitigar riscos decorrentes do comprometimento de credenciais, as senhas passam por um processo de derivação e são armazenadas sob a forma de *hashes* criptográficos gerados pelo algoritmo *bcrypt* (KERLYN, 2019). Adicionalmente, a persistência desses *tokens* na camada cliente ocorre por meio de *cookies HTTP Only* (MDN WEB DOCS, 2026), uma diretriz que isola os dados de autenticação de *scripts* do navegador para atenuar vulnerabilidades a ataques de injeção de código.

A validação de requisições e o controle de fluxo no *backend* foram fundamentados no padrão de projeto *Intercepting Filter* (MALINDA, 2025), materializado por meio de

uma arquitetura baseada em *middlewares* que operam de forma centralizada tanto na validação lógica quanto na proteção de recursos do sistema. Esse ecossistema interceptador subdivide-se em cinco componentes funcionais específicos:

- **Filtro de Autenticação (*auth*)**: Interceptador encarregado da decodificação e validação estrutural da assinatura dos *tokens JWT*.
- **Filtro de Autorização (*authorize*)**: Mecanismo de controle de acesso baseado em papéis (*Role-Based Access Control - RBAC*) para restringir o consumo de recursos conforme o perfil do usuário.
- **Filtro de Validação (*Joi*)**: Camada destinada à higienização e validação sintática dos esquemas de dados recebidos nas requisições HTTP.
- **Filtro de Verificação de Sockets (*verifySocketToken*)**: Validador responsável pela inspeção de segurança e autenticação durante o *handshake* de conexões via *WebSockets*.
- **Filtro Global de Exceções (*errorHandler*)**: Mecanismo de interceptação de falhas de execução voltado à padronização de respostas de erro e prevenção de vazamento de informações da infraestrutura.

Por fim, a garantia de sigilo nas comunicações síncronas e assíncronas baseia-se em duas camadas de segurança. O tráfego de mensagens privadas entre pacientes e profissionais é protegido por protocolos de comunicação segura na camada de transporte. Complementarmente, o servidor principal emprega o algoritmo de criptografia simétrica *AES-256* para cifrar os conteúdos textuais antes de sua gravação. Essa abordagem mitiga riscos de exposição dos dados mesmo perante eventuais acessos não autorizados à camada de persistência.

3.13.1 Verificação e Validação de Usuários

A plataforma incorpora mecanismos estritos de verificação cadastral destinados a mitigar a proliferação de contas inválidas, atenuar o risco de acessos indevidos e elevar a confiabilidade do ecossistema. Durante a etapa de registro de usuários gerais, a autenticidade dos canais de contato fornecidos é atestada por meio do envio de códigos de confirmação temporários. A validação dos endereços eletrônicos é viabilizada pelo *backend* principal com o suporte da biblioteca *Nodemailer* (REINMAN, 2026), ao passo que a verificação dos números telefônicos ocorre de forma análoga via mensagens de *SMS* processadas pela plataforma de comunicação *Twilio* (TWILIO, 2026). A conclusão bem-sucedida dessas validações em duas etapas atua como um pré-requisito técnico para a liberação de recursos de autenticação e segurança da conta.

Adicionalmente, o sistema implementa um fluxo de governança administrativa dedicado exclusivamente à habilitação de profissionais de saúde mental. Sob essa diretriz, as contas de novos profissionais permanecem sob um estado de pendência operacional até que os dados cadastrais, credenciais de identificação civil e qualificações de especialidades médicas declaradas sejam submetidos à auditoria minuciosa por um administrador do sistema. O acesso às funcionalidades de atendimento psicológico é condicionado de forma restrita a essa aprovação institucional prévia. Essa barreira de controle atua diretamente na redução de riscos decorrentes da atuação de agentes não credenciados, garantindo a credibilidade ética da plataforma e salvaguardando a segurança clínica dos pacientes.

3.14 Testes da Aplicação

Durante o desenvolvimento do TherapistFriend, foram realizados testes automatizados e manuais com o objetivo de verificar o funcionamento das funcionalidades implementadas e identificar possíveis falhas na aplicação.

Os testes automatizados foram desenvolvidos utilizando o *framework Jest*, com foco principal nos casos de uso (Use Cases) do *backend*, responsáveis por concentrar as regras de negócio da aplicação. Foram contemplados os principais módulos do sistema, incluindo autenticação, usuários, pacientes, profissionais, agenda, relatos, chat, notificações, sessões e administração. Também foram implementados testes para outros componentes, como entidades de domínio, eventos e funções auxiliares.

Para avaliar a abrangência dos testes automatizados, foi utilizada a funcionalidade de cobertura de código do Jest, por meio da opção –coverage. Os resultados indicaram uma cobertura geral de 82,57% das instruções, 83,24% das linhas, 67,73% das ramificações e 74,94% das funções. Esses indicadores permitiram identificar quais partes da implementação estavam contempladas pelos testes e quais ainda apresentavam trechos não cobertos.

Além dos testes automatizados, foram realizados testes manuais por meio da utilização direta da plataforma, verificando os principais fluxos de interação entre pacientes, profissionais e administradores. Foram avaliadas funcionalidades como cadastro, autenticação, publicação e visualização de relatos, solicitações de conversa, troca de mensagens, agendamentos e realização de sessões on-line. Também foram verificadas situações de erro e restrições de acesso relacionadas aos diferentes perfis de usuário.

Dessa forma, a combinação dos testes automatizados e manuais permitiu verificar tanto o comportamento individual das regras de negócio quanto o funcionamento das funcionalidades integradas da plataforma.

3.15 Containerização da Aplicação

A padronização do ambiente de execução e a orquestração dos serviços da plataforma foram viabilizadas por meio da tecnologia de containerização *Docker*. A adoção dessa ferramenta permitiu o isolamento completo dos componentes do sistema, simplificando a configuração do ambiente de desenvolvimento e otimizando a interoperabilidade entre o *backend* principal, o *frontend* e os microsserviços especializados. Adicionalmente, essa abordagem conferiu alta portabilidade à aplicação, garantindo que o ciclo de implantação e a execução do ecossistema ocorram de maneira consistente e reprodutível em diferentes ambientes de infraestrutura, independentemente das configurações subjacentes do sistema operacional hospedeiro.

4 Desenvolvimento

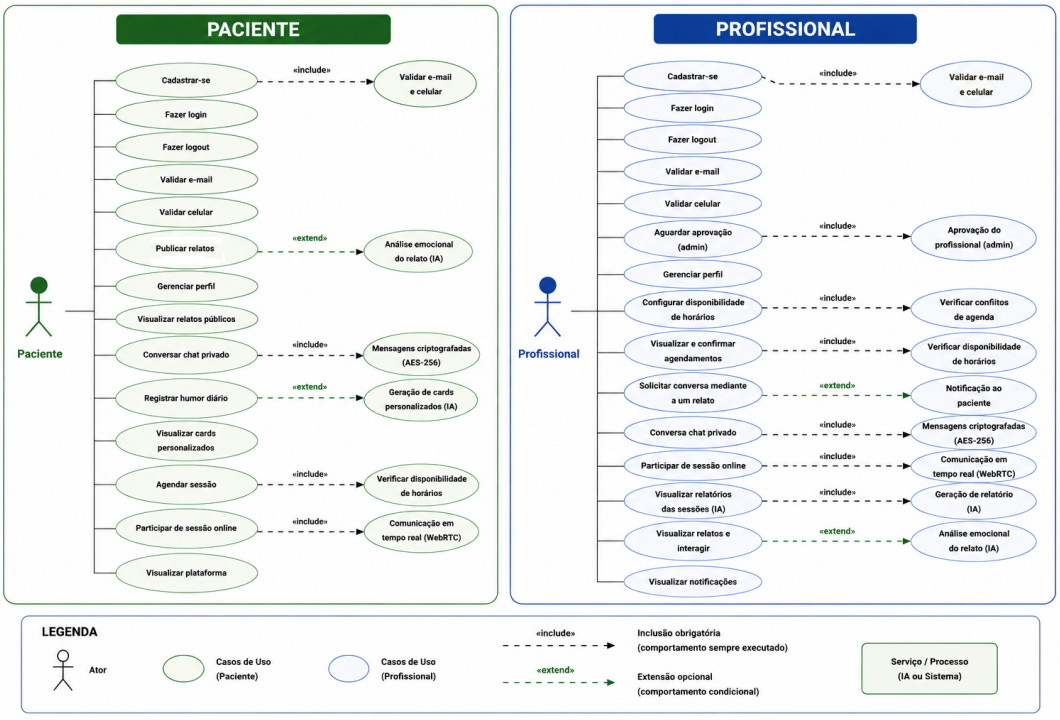
Este capítulo apresenta o desenvolvimento prático da plataforma proposta, descrevendo a implementação das principais funcionalidades do sistema, os fluxos de comunicação entre os componentes da arquitetura e a integração das tecnologias utilizadas durante a construção da aplicação.

4.1 Casos de uso

Os casos de uso foram definidos com base nos requisitos levantados durante a fase de análise, considerando as necessidades dos usuários e os objetivos propostos pela aplicação. A plataforma foi projetada para atender a três perfis principais de usuários: paciente, profissional da saúde mental e administrador, cada um possuindo permissões e funcionalidades específicas.

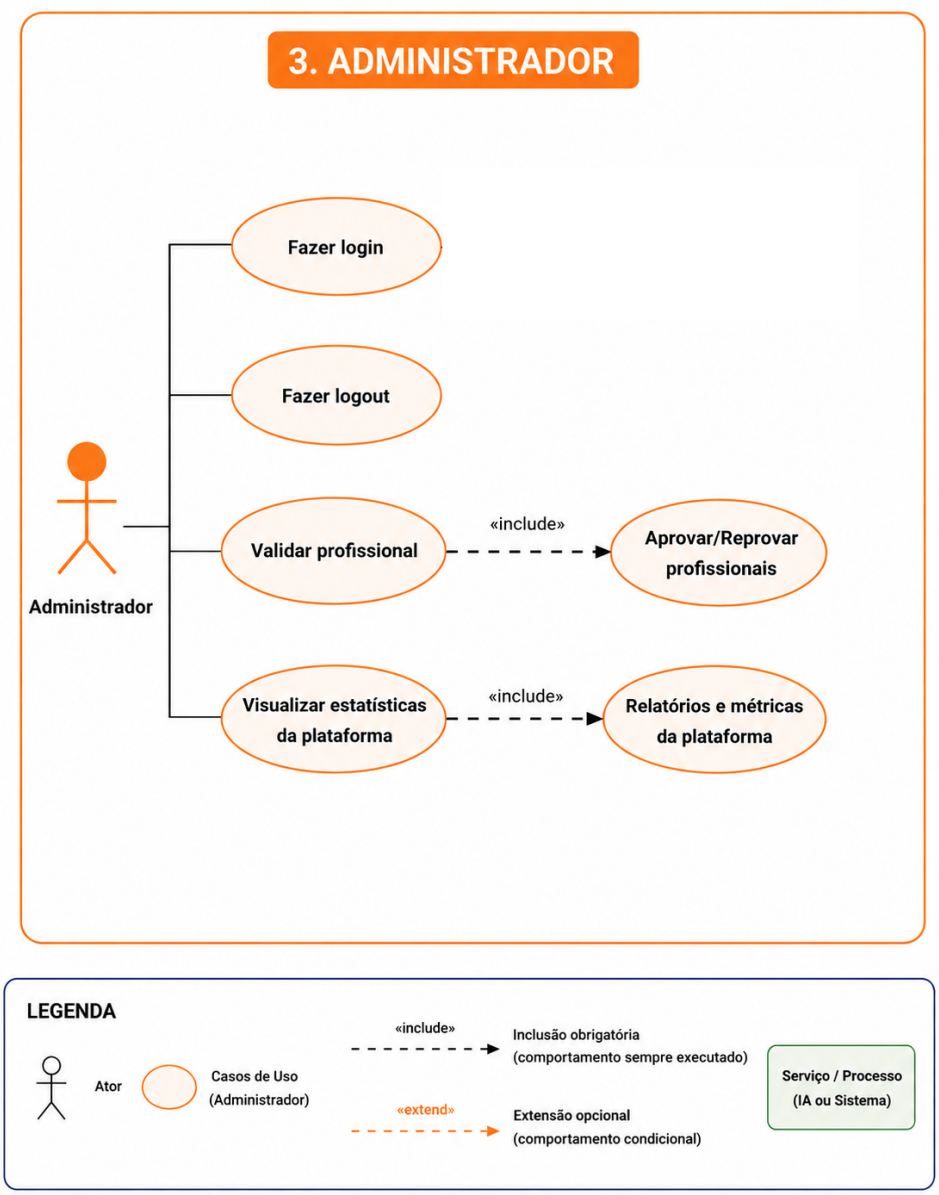
As Figuras 4 e 5 apresentam os diagramas de casos de uso da aplicação, destacando as principais funcionalidades disponibilizadas para cada ator e suas respectivas interações com o sistema.

Figura 4 – Diagrama de casos de uso para os atores paciente e profissional.



Fonte: Próprio autor

Figura 5 – Diagrama de casos de uso para o ator administrador.



Fonte: Próprio autor

4.1.1 Casos de uso do paciente

O paciente possui acesso aos recursos relacionados à utilização da plataforma, interação com relatos, comunicação com profissionais e acompanhamento de suas atividades. O processo de cadastro inclui a validação do endereço de e-mail e do número de telefone, permitindo confirmar os dados informados durante a criação da conta.

Entre os recursos disponíveis, destaca-se a publicação de relatos, na qual os relatos publicados podem ser disponibilizados publicamente para visualização por outros usuários da plataforma. Dessa forma, o paciente também pode visualizar relatos públicos e interagir com o conteúdo disponibilizado, de acordo com as permissões estabelecidas pelo sistema.

A plataforma também permite o registro do humor diário pelo paciente. A partir dessas informações, o sistema pode realizar o processamento dos dados por meio do microserviço de inteligência artificial e disponibilizar cards personalizados como recurso complementar de acompanhamento.

Outro recurso específico do paciente está relacionado à comunicação com profissionais. Quando um profissional solicita o início de uma conversa a partir de um relato publicado, o paciente pode aceitar ou recusar a solicitação. Em caso de aceite, a comunicação privada pode ser estabelecida por meio do sistema de chat.

O paciente também pode realizar o agendamento de sessões com profissionais disponíveis e participar das sessões online disponibilizadas pela plataforma. Além disso, possui acesso às notificações e às demais funcionalidades relacionadas ao gerenciamento de seu perfil.

4.1.2 Casos de uso do profissional

O profissional possui funcionalidades voltadas principalmente ao gerenciamento de sua atuação na plataforma e à interação com os pacientes. Após realizar o cadastro e validar seus dados de contato, o profissional permanece sujeito à aprovação administrativa antes de obter acesso às funcionalidades destinadas ao atendimento.

Após sua aprovação, o profissional pode gerenciar suas informações de perfil e configurar sua disponibilidade de horários, permitindo que os pacientes visualizem os períodos disponíveis para agendamento.

O profissional também pode visualizar os agendamentos realizados e realizar sua confirmação, além de participar das sessões online previamente estabelecidas com os pacientes.

A interação com pacientes pode ser iniciada a partir dos relatos publicados na plataforma. Nesse contexto, o profissional pode solicitar uma conversa com o autor de um relato, ficando o estabelecimento do chat condicionado à aceitação do paciente.

Durante sua utilização da plataforma, o profissional também pode visualizar e interagir com os relatos publicados pelos pacientes, além de acessar relatórios gerados a partir das informações coletadas durante as sessões. Esses relatórios utilizam recursos de inteligência artificial como ferramenta complementar de apoio ao profissional.

Por fim, o profissional possui acesso às notificações relacionadas às atividades da plataforma e aos recursos necessários para gerenciamento de seu perfil e atendimentos.

4.1.3 Casos de uso do administrador

O administrador possui um conjunto reduzido de funcionalidades, concentradas no gerenciamento e na supervisão da plataforma.

Sua principal responsabilidade está relacionada à validação dos profissionais cadastrados. Após o cadastro de um profissional, suas informações permanecem pendentes até que sejam analisadas pelo administrador, que pode aprovar ou rejeitar o cadastro conforme os dados apresentados.

Além da validação dos profissionais, o administrador possui acesso a estatísticas da plataforma, permitindo acompanhar métricas relacionadas à utilização do sistema e fornecer uma visão geral sobre seu funcionamento.

4.2 Banco de Dados

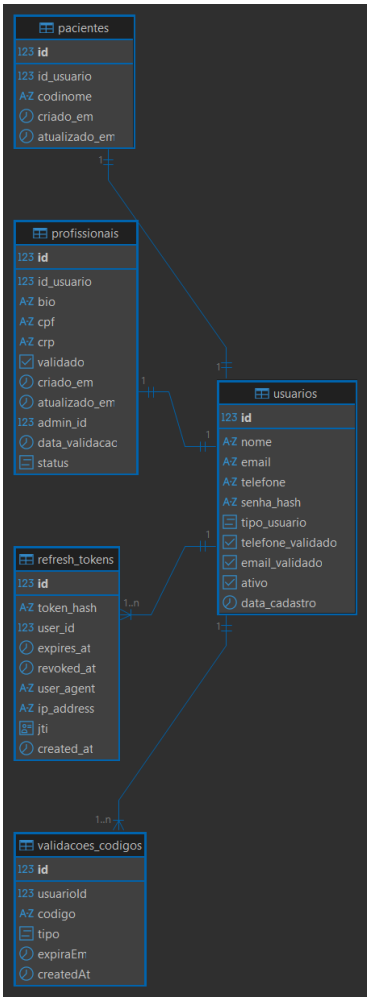
Dando continuidade ao que foi descrito no capítulo de materiais e métodos, essa seção apresenta a estrutura do banco de dados relacional mostrando o diagrama entidade-relacionamento (DER) utilizado para modelar as tabelas e os relacionamentos entre elas. Devido a alta complexidade do sistema, organizou-se o DER em grupos de tabelas relacionadas, facilitando a compreensão da estrutura.

4.2.1 Estrutura de Usuários e Autenticação

A estrutura de usuários é o núcleo central do banco de dados, permitindo o gerenciamento dos diferentes perfis de usuários (pacientes, profissionais e administradores) e suas respectivas funcionalidades. É a partir dela que são estabelecidos os relacionamentos com outras entidades. A estrutura de autenticação ela fica responsável por armazenar os dados relacionados ao controle de acesso.

A Figura 6 apresenta o DER relacionado à estrutura de usuários e autenticação.

Figura 6 – DER da estrutura de usuários e autenticação



Fonte: Próprio autor

A tabela de usuarios é a entidade principal, contendo informações básicas. A partir dela, se definem as tabelas de pacientes e profissionais. Tais tabelas são responsáveis por armazenar informações específicas de cada tipo de usuário como se observa na Figura 6. Já a tabela de autenticacao (*refresh_tokens*) é responsável por armazenar os dados relacionados ao controle de acesso. Abaixo é apresentado os campos das tabelas:

- **usuarios**: id, nome, email, telefone, senha_hash, tipo_usuario, telefone_validado, email_validado, data_cadastro e ativo. O campo tipo_usuario é utilizado para diferenciar os perfis de usuários (paciente, profissional ou administrador), enquanto os campos telefone_validado e email_validado indicam se as informações de contato foram confirmadas pelos usuários;
- **pacientes**: id, usuario_id (*FK*), codinome, criado_em, atualizado_em. O campo codinome é utilizado para permitir que os pacientes publiquem relatos de forma anônima, protegendo sua identidade dentro da plataforma;

- **profissionais**: id, id_usuario (*FK*), bio, cpf, crp, validado, criado_em, atualizado_em, admin_id (*FK*), data_validacao e status. O campo validado indica se o profissional foi aprovado por um administrador, enquanto o campo status pode ser utilizado para indicar se o profissional está ativo ou inativo na plataforma. O campo admin_id é uma chave estrangeira que referencia o administrador responsável pela validação do profissional, permitindo rastrear o processo de aprovação e garantir a supervisão adequada dos profissionais cadastrados. O campo crp é utilizado para armazenar o número do Conselho Regional de Psicologia, garantindo que apenas profissionais devidamente registrados possam atuar na plataforma;

- **refresh_tokens**: id, token_hash, user_id (*FK*), expired_at, revoked_at, user_agent, ip_address, jti, created_at. O campo token_hash é utilizado para armazenar uma versão hash do token de atualização, garantindo maior segurança no armazenamento dos tokens. Os campos expired_at e revoked_at permitem controlar a validade dos tokens, enquanto os campos user_agent e ip_address registram informações sobre o dispositivo e local de onde o token foi gerado, contribuindo para a detecção de atividades suspeitas. O campo jti (*JWT ID*) é utilizado para identificar de forma única cada token, facilitando o gerenciamento e revogação dos tokens quando necessário;

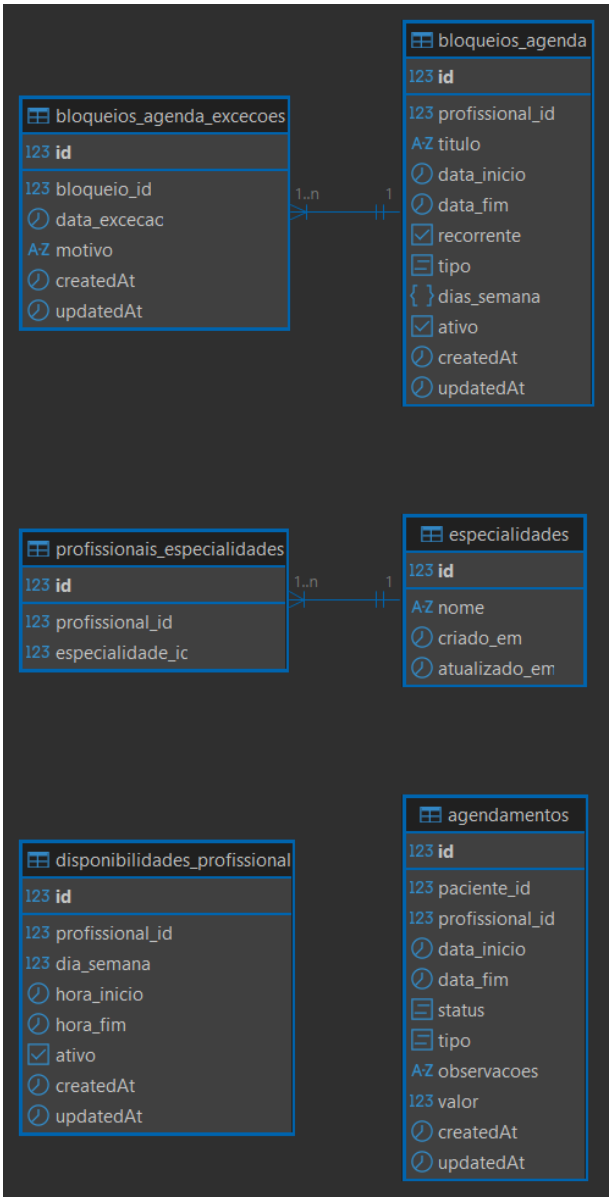
- **validacoes_codigos**: id, usuarioId, codigo, tipo, expiraEm. A tabela de validações de códigos armazena os códigos de verificação enviados aos usuários para confirmação de suas informações de contato. O campo usuarioId é uma chave estrangeira que referencia o usuário ao qual o código de validação pertence, permitindo associar os códigos de validação diretamente aos usuários. O campo tipo é utilizado para diferenciar os tipos de validação (e-mail ou telefone), enquanto o campo expiraEm indica a data e hora de expiração, garantindo que os códigos de validação sejam temporários e seguros.

4.2.2 Estrutura de Especialidades e Agendamento

A estrutura de especialidades e agendamento é fundamental para o funcionamento da plataforma, permitindo que os pacientes possam agendar sessões com profissionais de acordo com suas necessidades e preferências. Essa estrutura envolve a definição das especialidades oferecidas pelos profissionais, a configuração de horários disponíveis para atendimento e o gerenciamento dos agendamentos realizados pelos pacientes.

A Figura 7 apresenta o DER relacionado à estrutura de especialidades e agendamento.

Figura 7 – DER da estrutura de especialidades e agendamento



Fonte: Próprio autor

A estrutura de especialidades foi desacoplada dos profissionais para garantir flexibilidade e eliminar redundâncias. Dessa forma, viabiliza-se o relacionamento de muitos para muitos: um profissional pode ter várias especialidades, e uma especialidade pode atender a múltiplos profissionais. Abaixo, é apresentado a descrição de cada campo das tabelas relacionadas:

- **especialidades**: id, nome, criado_em, atualizado_em. A tabela de especialidades armazena as diferentes áreas de atuação dos profissionais, permitindo que os pacientes possam escolher profissionais com base em suas necessidades específicas;
- **profissionais_especialidades**: id, profissional_id (*FK*) e especialidade_id (*FK*).

Esta tabela de relacionamento estabelece a conexão entre os profissionais e suas respectivas especialidades, viabilizando a flexibilidade necessária para acomodar a diversidade de perfis profissionais na plataforma;

- **disponibilidades_profissional**: id, profissional_id (*FK*), dia_semana, hora_inicio, hora_fim e ativo. A tabela de disponibilidades permite que os profissionais configurem sua disponibilidade para atendimento, facilitando o processo de agendamento por parte dos pacientes;

- **bloqueios_agenda**: id, profissional_id (*FK*), titulo, data_inicio, data_fim, recorrente, tipo, dias_semana e ativo. A tabela de bloqueios de agenda é utilizada para registrar períodos em que os profissionais não estarão disponíveis para atendimento, seja por motivos pessoais, férias ou outras circunstâncias, garantindo que os pacientes não possam agendar sessões durante esses períodos;

- **bloqueios_agenda_excecoes**: id, bloqueio_id (*FK*), data_excecao e motivo. Esta tabela de exceções de bloqueio de agenda permite que os profissionais registrem datas específicas em que um bloqueio de agenda recorrente não se aplicará, oferecendo maior flexibilidade na gestão de suas disponibilidades;

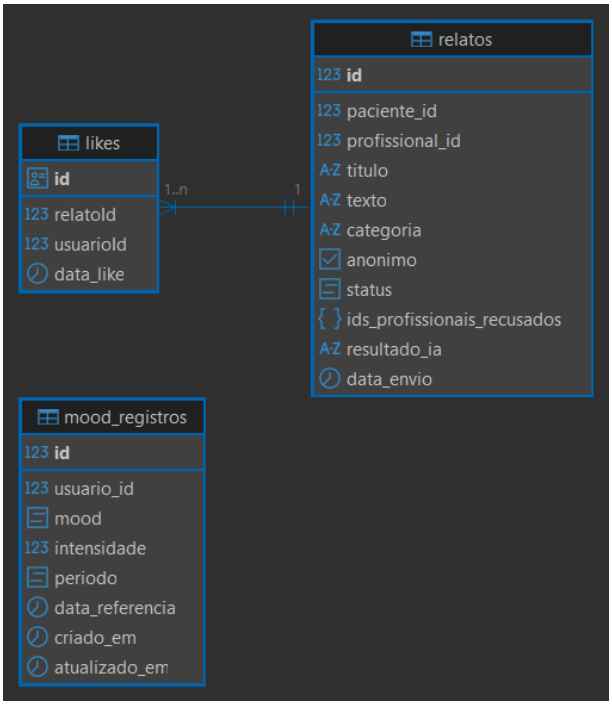
- **agendamentos**: id, paciente_id (*FK*), profissional_id (*FK*), data_inicio, data_fim, status, tipo, observacoes e valor. A tabela de agendamentos gerencia as sessões marcadas entre pacientes e profissionais, registrando informações sobre a data do agendamento e o status da sessão (confirmada, pendente ou cancelada). O campo valor foi adicionado porém, não foi utilizado durante o desenvolvimento da aplicação, mas pode ser utilizado futuramente para implementar funcionalidades relacionadas a pagamentos e faturamento dentro da plataforma.

4.2.2.1 Estrutura de Relatos e Registro de Humor

A estrutura de relatos e registro de humor é essencial para a proposta da plataforma, permitindo que os pacientes compartilhem suas experiências, sentimentos e desafios relacionados à saúde mental. Essa estrutura envolve a criação de entidades para armazenar os relatos publicados pelos pacientes, bem como o registro diário do humor, possibilitando um acompanhamento mais detalhado do estado emocional dos usuários ao longo do tempo.

A Figura 8 apresenta o DER relacionado à estrutura de relatos e registro de humor.

Figura 8 – DER da estrutura de relatos e registro de humor



Fonte: Próprio autor

A estrutura de relatos e registro de humor foi composta por três tabelas sendo uma para registro dos relatos outra para registrar os likes e uma outra separada para o registro do humor diário dos pacientes. Abaixo, é apresentado a descrição de cada campo das tabelas relacionadas:

- **relatos**: id, paciente_id (*FK*), profissional_id (*FK*), titulo, texto, anonimo, status, ids_profissionais_recusados e resultados_ia e data_envio. A tabela de relatos armazena as publicações feitas pelos pacientes, permitindo que compartilhem suas experiências e sentimentos relacionados à saúde mental. O campo anonimo indica se o relato foi publicado de forma anônima ou identificado, o campo profissional_id é utilizados para registrar o profissional que assumiu o relato, o campo status indica o status (pendente, aguardando aprovação, em conversa ou arquivado), o campo ids_profissionais_recusados armazena uma lista de profissionais que foram recusados pelo paciente para assumir o relato, e o campo resultados_ia é utilizado para armazenar os resultados gerados pela inteligência artificial, oferecendo insights para os profissionais;
- **likes**: id (UUID), relato_id (*FK*), usuario_id (*FK*) e data_like. A tabela de likes registra as interações dos usuários com os relatos publicados, permitindo que os pacientes recebam feedback e apoio da comunidade por meio de curtidas;

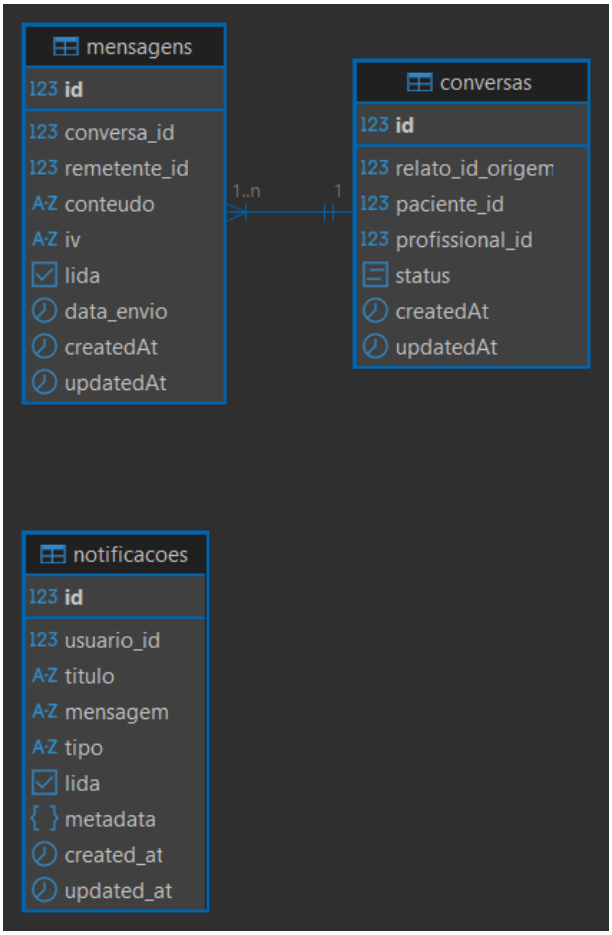
- **registro_humor**: id, usuario_id (*FK*), mood, intensidade, periodo, data_referencia, criado_em, atualizado_em. A tabela registra o humor do paciente colocando no campo mood (ansiedade, tristeza, raiva, medo, felicidade e neutro), a intensidade é uma escala de 1 a 5 e o campo periodo é utilizado para indicar o período do dia (manhã, tarde ou noite) em que o humor foi registrado, permitindo um acompanhamento mais detalhado do estado emocional dos pacientes ao longo do tempo.

4.2.2.2 Estrutura de Notificações e Chats

A estrutura de notificações e chats é fundamental para a comunicação eficiente entre os usuários da plataforma, permitindo que os pacientes e profissionais possam interagir de forma rápida e segura. Essa estrutura envolve a criação de entidades para gerenciar as notificações enviadas aos usuários, bem como o registro das conversas realizadas por meio do sistema de chat integrado à plataforma.

A Figura 9 apresenta o DER relacionado à estrutura de notificações e chats.

Figura 9 – DER da estrutura de notificações e chats



Fonte: Próprio autor

A estrutura de notificações e chats foi composta por três tabelas sendo uma para

registro das notificações e outras para o registro das conversas realizadas por meio do sistema de chat. Abaixo, é apresentado a descrição de cada campo das tabelas relacionadas:

- **notificacoes**: id, usuario_id (*FK*), tipo, titulo, mensagem, metadata, lida e created_at. A tabela de notificações gerencia as mensagens enviadas aos usuários, permitindo que sejam informados sobre eventos importantes dentro da plataforma. O campo tipo é utilizado para categorizar as notificações (agendamento, mensagem, sistema, etc.), enquanto os campos titulo e mensagem armazenam o conteúdo da notificação. O campo metadata é utilizado para armazenar informações adicionais, como links ou dados específicos do evento que gerou a notificação. O campo lida indica se a notificação foi visualizada pelo usuário, permitindo um melhor gerenciamento das comunicações dentro da plataforma;

- **conversas**: id, relato_id_origem, paciente_id (*FK*), profissional_id (*FK*), status e created_at. A tabela de conversas registra as interações entre pacientes e profissionais por meio do sistema de chat integrado à plataforma. O campo relato_id_origem é utilizado para associar a conversa ao relato que deu origem. O campo status indica o status da conversa (ativa, arquivada, etc.);

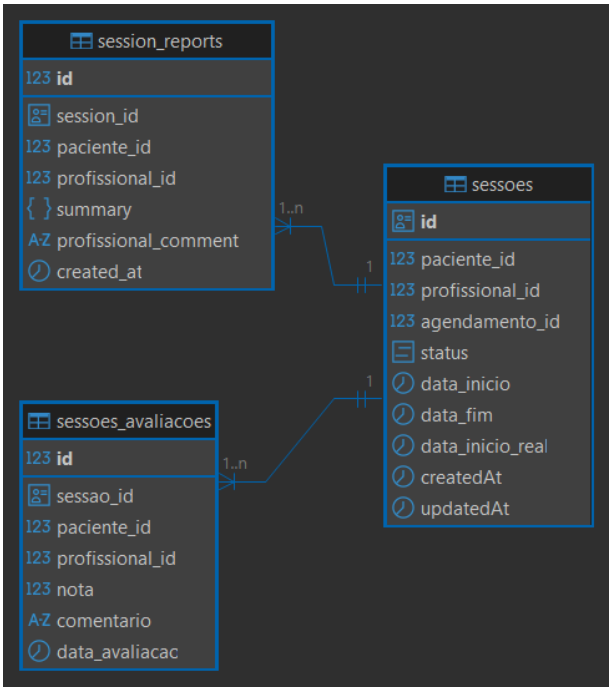
- **mensagens**: id, conversa_id (*FK*), remetente_id (*FK*), conteudo, iv, lida e data_envio. A tabela de mensagens armazena o conteúdo das mensagens trocadas entre pacientes e profissionais durante as conversas realizadas por meio do sistema de chat, garantindo que as comunicações sejam registradas e possam ser acessadas posteriormente. O campo remetente_id é utilizado para identificar o usuário que enviou a mensagem, enquanto o campo iv (*initialization vector*) é utilizado para armazenar o vetor de inicialização necessário para a criptografia das mensagens, garantindo a segurança e confidencialidade das comunicações.

4.2.2.3 Estrutura de sessões e relatórios

A estrutura de sessões e relatórios é essencial para o acompanhamento e avaliação dos atendimentos realizados na plataforma, permitindo que os profissionais possam registrar informações sobre as sessões realizadas com os pacientes, bem como gerar relatórios que auxiliem no monitoramento do progresso dos pacientes ao longo do tempo.

A Figura 10 apresenta o DER relacionado à estrutura de sessões e relatórios.

Figura 10 – DER da estrutura de sessões e relatórios



Fonte: Próprio autor

A estrutura de sessões e relatórios é composto por três tabelas sendo uma para registro das sessões, outra para o registro dos relatórios gerados e outra para o registro dos feedbacks dos pacientes. Abaixo, é apresentado a descrição de cada campo das tabelas relacionadas:

- **sessoes**: id (UUID), agendamento_id (*FK*), paciente_id (*FK*), profissional_id (*FK*), data_inicio, data_fim, data_inicio_real e status. A tabela de sessões gerencia os atendimentos realizados entre pacientes e profissionais, registrando informações sobre a data e hora do início e término da sessão, e o status da sessão (agendada, ativa, finalizada e cancelada). O campo data_inicio_real é utilizado para registrar a data e hora real de início da sessão, permitindo a geração de relatórios mais precisos;
- **session_reports**: id, session_id (*FK*), profissional_id (*FK*), paciente_id (*FK*), summary, profissional_comment, e created_at. A tabela de relatórios armazena os documentos gerados de forma atomática após as sessões. O campo summary é utilizado para armazenar um resumo do relatório, que pode incluir informações sobre o estado emocional do paciente ao longo do tempo. O campo profissional_comment permite que os profissionais adicionem comentários adicionais ao relatório, oferecendo insights e observações relevantes sobre o atendimento realizado;
- **sessoes_avaliacoes**: id, sessao_id (*FK*), paciente_id (*FK*), profissional_id (*FK*), nota, comentario e data_avaliacao. A tabela de feedbacks registra as avaliações

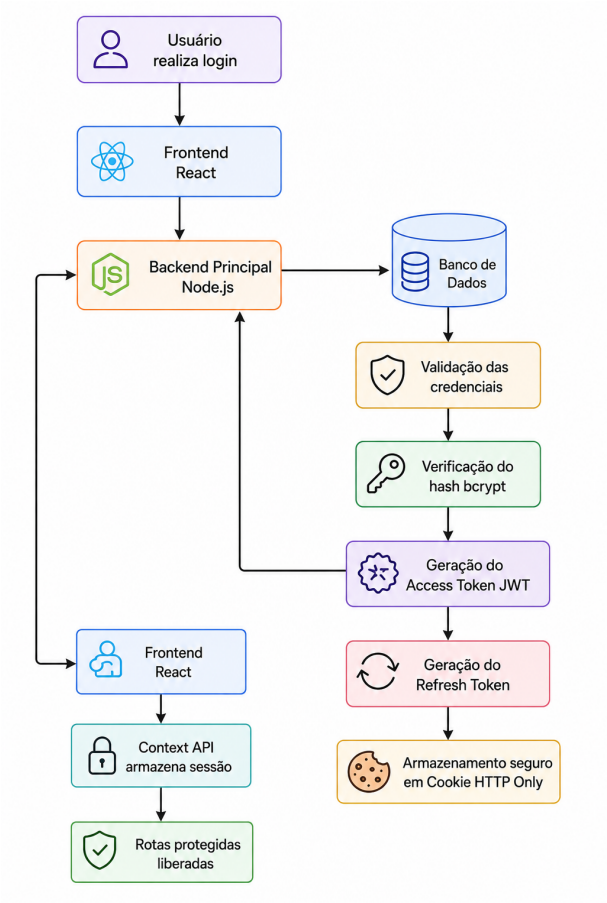
feitas pelos pacientes sobre as sessões realizadas com os profissionais, permitindo que os pacientes forneçam feedback sobre a qualidade do atendimento recebido. O campo nota é utilizado para armazenar uma classificação dada pelo paciente à sessão, enquanto o campo comentario permite que os pacientes deixem comentários adicionais sobre sua experiência.

4.3 Fluxo de Cadastro e Autenticação

O fluxo de cadastro e autenticação é um processo fundamental para garantir a segurança e a integridade dos dados dos usuários na plataforma, permitindo que os pacientes, profissionais e administradores possam acessar suas contas de forma segura e protegida. O processo de cadastro envolve a coleta de informações básicas dos usuários, como nome, e-mail, telefone e senha, bem como a validação dessas informações para garantir que sejam legítimas e estejam em conformidade com os requisitos da plataforma.

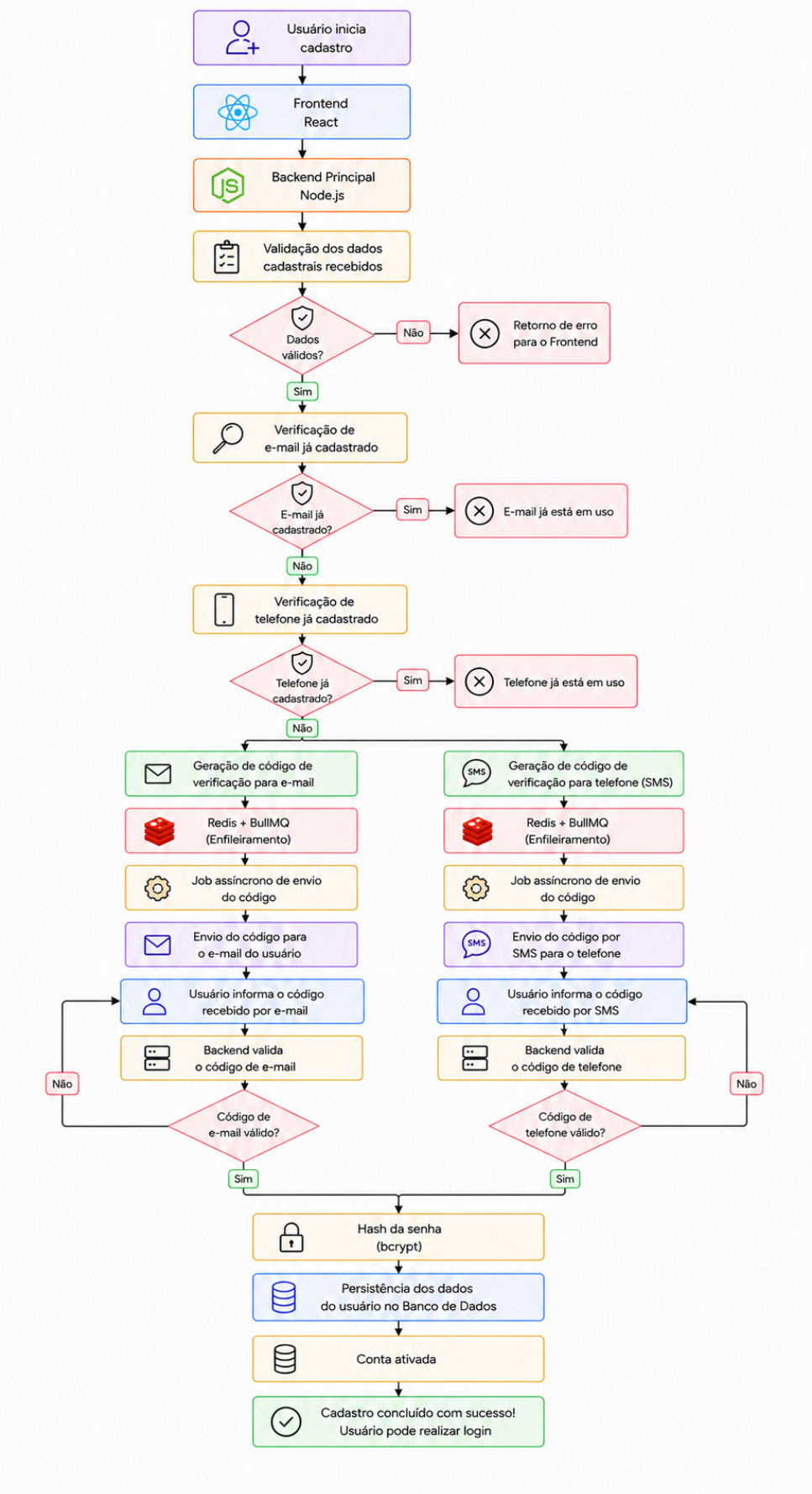
A Figura 11 apresenta um diagrama do fluxo de autenticação e a Figura 12 apresenta o fluxo de cadastro.

Figura 11 – Diagrama do fluxo de login e autenticação



Fonte: Próprio autor

Figura 12 – Diagrama do fluxo de cadastro



Fonte: Próprio autor

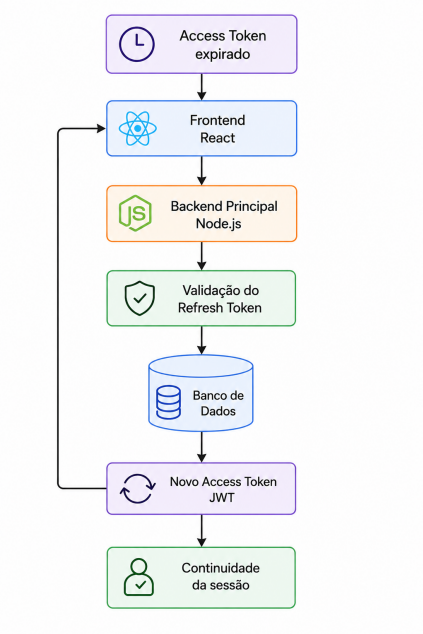
O processo de cadastro é feito seguindo etapas sequenciais, onde o usuário preenche um formulário com suas informações básicas, como nome, e-mail, telefone e senha. Após o envio do formulário, a plataforma realiza a validação dos dados fornecidos, verificando se o e-mail e telefone são válidos. Para isso, a plataforma envia códigos de validação, que devem ser inseridos na plataforma para confirmar a validade das informações de contato. Somente após a validação bem-sucedida o usuário tem sua conta criada e pode acessar a plataforma. Vale ressaltar que o perfil administrador é criado por meio de um *script* de *seed* no banco de dados, e não possui um processo de cadastro público.

Já o processo de autenticação é feito seguindo o fluxo apresentado na Figura 11, onde o usuário insere suas credenciais (e-mail e senha) para acessar a plataforma. A plataforma verifica as credenciais fornecidas, e se forem válidas, gera *tokens* de acesso e atualização para gerenciar a sessão do usuário. O *token* de acesso é utilizado para autenticar as requisições feitas pelo usuário durante sua interação com a plataforma, enquanto o *token* de atualização é utilizado para renovar as credenciais expiradas, garantindo a permanência do usuário no sistema sem a necessidade de uma nova autenticação manual. Concluída essa estapa, o *frontend* faz uma requisição para o *backend* solicitando as informações do perfil do usuário e as gerencia por meio da *Context API*, o que assegura o controle global dos dados de navegação.

No primeiro *login* o usuário é direcionado para uma tela de configuração de perfil onde deve fornecer informações adicionais. No caso do paciente, ele pode escolher um codinome que é usado para o anonimato em relatos, marcação de sessões e outras interações dentro da plataforma. Já o profissional deve fornecer informações relacionadas à sua especialidade, biografia, crp e cpf, que são utilizadas para validar seu cadastro e permitir que ele atue na plataforma.

O fluxo de renovação de credenciais está apresentado na Figura 13, demonstrando como o usuário permanece autenticado na plataforma de forma transparente, sem exigir um novo *login*.

Figura 13 – Diagrama do fluxo de refresh token



Fonte: Próprio autor

O processo de refresh token é composto por várias etapas, incluindo a verificação, geração e atualização. A seguir, é descrito cada etapa do processo:

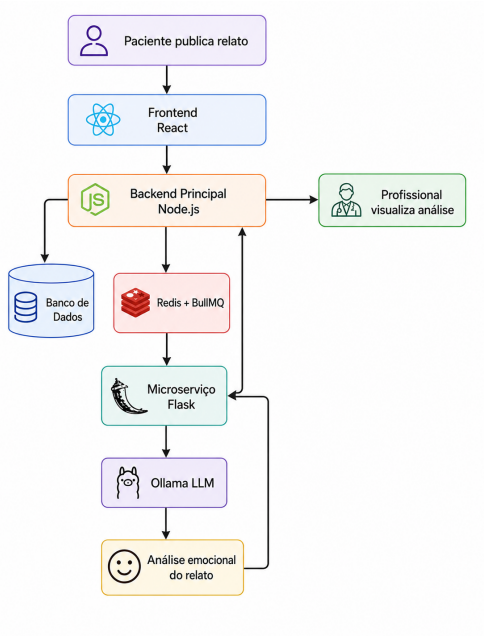
- **Verificação do token de atualização**: Quando o *token* de acesso expira o *backend* retorna um erro 401 (*Unauthorized*) para o *frontend*, que captura o erro via *axios* e então faz uma requisição para o endpoint de *refresh token*. O *backend* verifica a validade do *token* de atualização, garantindo que ele não tenha expirado ou sido revogado;
- **Geração de um novo token de acesso**: Se o *token* de atualização for válido, o *backend* gera um novo *token* de acesso para o usuário, permitindo que ele continue autenticado e possa acessar os recursos da plataforma sem interrupção;
- **Renovação do token de atualização**: O backend também gera um novo *token* de atualização e revoga o *token* antigo.

4.4 Fluxo de Relatos

O fluxo de relatos é um processo central para a proposta da plataforma, permitindo que os pacientes compartilhem suas experiências, sentimentos e desafios relacionados à saúde mental. O processo envolve a criação, publicação e interação, proporcionando um espaço seguro e acolhedor para que os pacientes possam expressar suas emoções sem medo de julgamento através do anonimato.

A Figura 14 apresenta um diagrama do fluxo de relatos.

Figura 14 – Diagrama do fluxo de relatos



Fonte: Próprio autor

O processo de relatos é composto por várias etapas, incluindo a criação, análise por inteligência artificial e interação com a comunidade. A seguir, é descrito cada etapa do processo:

- **Criação do relato**: Os pacientes podem criar relatos compartilhando suas experiências, sentimentos e desafios relacionados à saúde mental. Durante a criação do relato, os pacientes têm a opção de publicar de forma anônima ou não, permitindo que escolham o nível de privacidade que desejam para suas publicações. O anonimato neste caso faz com que somente o profissional consiga visualizar;
- **Análise por inteligência artificial**: Após a criação, o conteúdo é enviado para um microserviço de inteligência artificial que realiza uma análise detalhada do texto, e o classifica em três niveis de gravidade (leve, moderado e grave). Os resultados dessa análise são armazenados no banco de dados e podem ser utilizados para gerar insights para os profissionais. Todo o processo é realizado de forma assíncrona, garantindo que o paciente não fique aguardando muito para ter seu relato publicado, visto que a análise do microserviço fica disponível apenas para os profissionais, e não para os pacientes;
- **Interação com a comunidade**: Os relatos publicados permitem que outros usuários interajam por meio de curtidas, oferecendo apoio e feedback aos pacientes. Essa

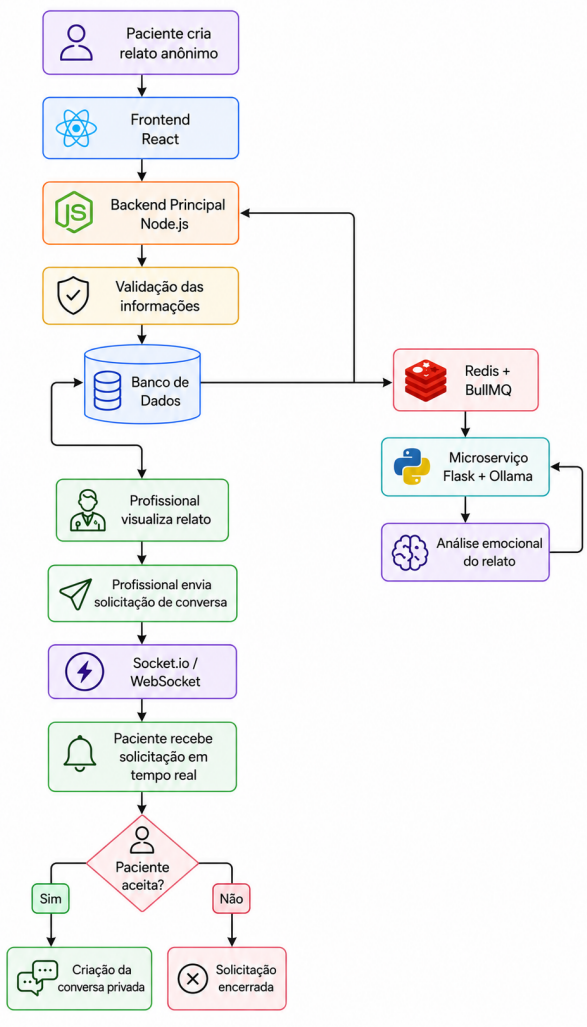
interação contribui para a construção de uma comunidade acolhedora e solidária dentro da plataforma.

4.5 Fluxo de solicitação de conversa

O fluxo de solicitação de conversa é um processo importante para a plataforma, permitindo que os profissionais posssam se conectar com os pacientes de forma proativa, oferecendo suporte e acompanhamento.

A Figura 15 apresenta um diagrama do fluxo de solicitação de conversa.

Figura 15 – Diagrama do fluxo de solicitação de conversa



Fonte: Próprio autor

O processo de solicitação de conversa é composto por várias etapas, incluindo a análise dos relatos, envio da solicitação e resposta do paciente. A seguir, é descrito cada etapa do processo:

- **Análise dos relatos**: Os profissionais podem analisar os relatos publicados pelos pacientes, utilizando o resultado gerado pela inteligência artificial como apoio. A classificação de gravidade fornecida pela inteligência artificial pode ajudar os profissionais a identificar relatos que possam indicar um estado emocional mais delicado, permitindo que eles priorizem suas interações e ofereçam suporte de forma mais eficaz;

- **Envio da solicitação**: Caso o profissional identifique um relato que sinta que possa ajudar, ele pode enviar uma solicitação de conversa ao assumir para si o relato. Essa solicitação é enviada ao paciente, informando sobre o interesse do profissional em estabelecer uma comunicação;

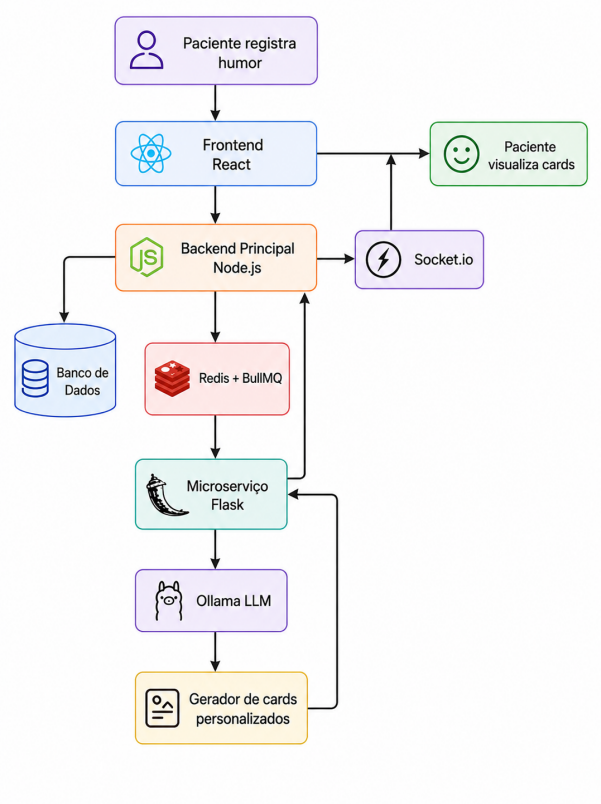
- **Resposta do paciente**: O paciente recebe a solicitação de conversa e tem a opção de aceitar ou recusar a proposta do profissional. Caso aceite, uma conversa é iniciada entre o paciente e o profissional, permitindo que eles possam interagir de forma mais direta. Em caso de recusa, o profissional é informado sobre a decisão do paciente, e o relato permanece disponível para que outros profissionais possam analisá-lo e, se desejarem, enviar suas próprias solicitações de conversa. Os profissionais que tiverem seus pedidos recusados, terão seu id registrado no campo ids_profissionais_recusados da tabela de relatos, garantindo que não possam enviar novas solicitações para aquele relato específico, respeitando a decisão do paciente e evitando abordagens indesejadas.

4.6 Fluxo de Registro de Humor e Geração de Cards

O fluxo de registro de humor e geração de cards serve como um processo de apoio aos pacientes. Ele permite que os pacientes registrem seu humor diariamente, podendo ser feitos até três registros por dia (manhã, tarde e noite). A partir desses registros, a plataforma gera cards personalizados, oferecendo informações e dicas relacionadas ao estado emocional do pacientes, contribuindo para o autoconhecimento e o autocuidado.

A Figura 16 apresenta um diagrama do fluxo de registro de humor e geração de cards.

Figura 16 – Diagrama do fluxo de registro de humor e geração de cards



Fonte: Próprio autor

O processo de registro de humor e geração de cards é realizado por meio de várias etapas, que serão descritas a seguir:

- **Registro de humor**: Os pacientes podem registrar seu humor diariamente, escolhendo entre diferentes opções de emoções (triste, neutro, feliz, ansioso e feliz) e indicando a intensidade em uma escala de um a cinco. O registro também inclui o período do dia (manhã, tarde ou noite), permitindo um acompanhamento mais detalhado do estado emocional dos pacientes ao longo do tempo;
- **Armazenamento dos registros**: Os registros são armazenados no banco de dados, permitindo um controle sobre o histórico emocional. Além disso, podem ser utilizados em uma atualização futura da plataforma, onde os profissionais terão acesso ao histórico emocional dos pacientes e utilizar essas informações para oferecer um suporte mais personalizado durante as sessões;
- **Geração de cards personalizados**: A partir dos registros de humor, a plataforma gera cards personalizados para os pacientes. Esses cards são de três tipos: Dica do dia, estatísticas e motivação. Cada tipo de card são exibidos de forma diferente e oferecem informações e dicas relacionadas ao estado emocional do paciente. O processo ocorre de forma assíncrona, evitando do paciente ficar aguardando. Quando

os cards ficam prontos, o proprio *frontend* recebe via *WebSocket* e atualiza a interface para exibir;

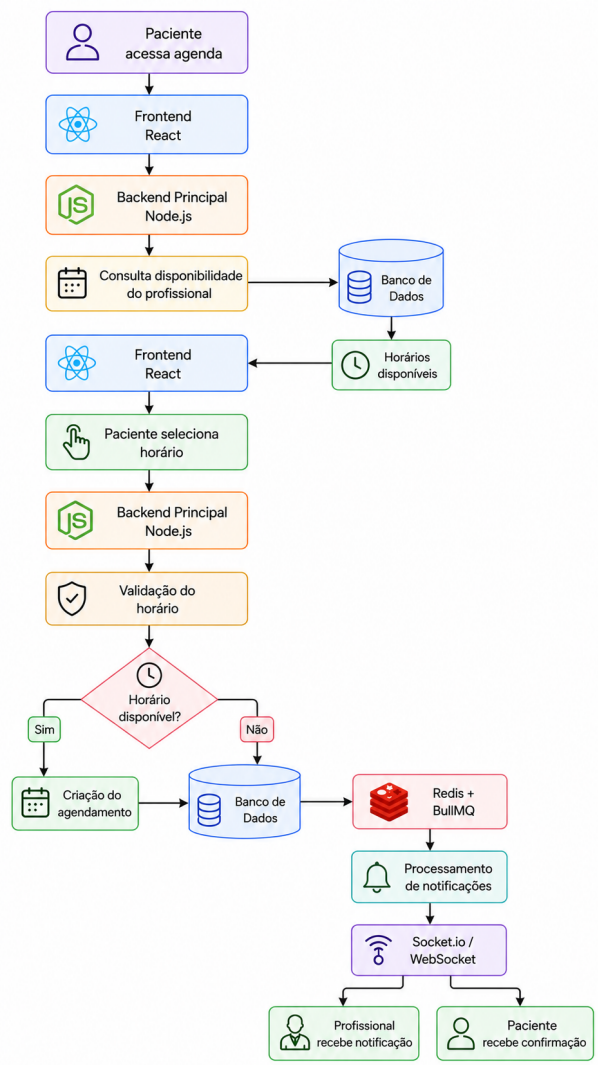
- **Acesso aos cards**: Os pacientes podem acessar os cards gerados a qualquer momento indo na página inicial (*Dashboard*).

4.6.1 Fluxo de Agendamento

O fluxo de agendamentos é o que permite um paciente consiga marcar uma sessão dentro da plataforma. Ele envolve desde a escolha do profissional, passando pela escolha do horário, até a confirmação do agendamento.

A Figura 17 apresenta um diagrama simplificado do fluxo de agendamento.

Figura 17 – Diagrama do fluxo de agendamento



Fonte: Próprio autor

Para que o processo de agendamento funcione, é necessário que os profissionais

tenham disponibilidades configuradas. O processo de agendamento é composto por várias etapas, que serão descritas a seguir:

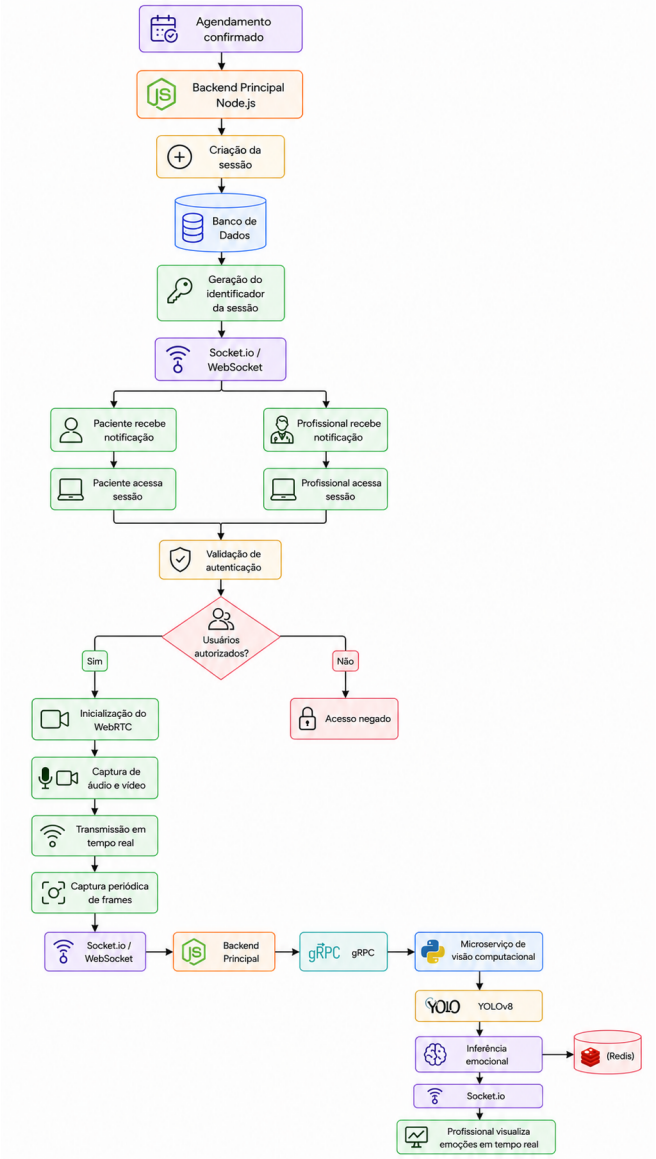
- **Escolha do profissional**: O paciente pode escolher um profissional com base em suas especialidades, biografia e avaliações de outros pacientes. A plataforma oferece filtros para facilitar a busca por profissionais que atendam às necessidades específicas dos pacientes. Vale ressaltar que a ideia de escolha normalmente se dá pelo contato direto que o paciente teve ao conversar com o profissional por meio do chat;
- **Escolha do horário**: Após escolher o profissional, o paciente pode visualizar as disponibilidades configuradas pelo profissional e escolher um horário que seja conveniente para ambos. A plataforma verifica se o horário escolhido está disponível e não conflita com outros agendamentos ou bloqueios de agenda do profissional;
- **Confirmação do agendamento**: Depois de escolher o horário, o paciente confirma o agendamento. A plataforma registra no banco de dados e envia uma notificação para o profissional poder confirmar ou recusar, após confirmar a sessão é marcada e um processo assincrono é iniciado para fazer o envio do lembrete com o link da sessão 10 minutos antes de começar.

4.7 Fluxo de Sessões Online

O fluxo de sessões online é o processo que permite que os pacientes e profissionais possam realizar atendimentos de forma remota por meio da plataforma. Esse processo envolve a criação de sessões, a comunicação entre os participantes e o acompanhamento emocional dos pacientes ao longo do tempo.

A Figura 18 apresenta um diagrama do fluxo de sessões online.

Figura 18 – Diagrama do fluxo de sessões online



Fonte: Próprio autor

O processo de sessões online é composto por várias etapas, que serão descritas a seguir:

- **Criação da sessão**: As sessões são criadas a partir dos agendamentos confirmados entre pacientes e profissionais. A plataforma gera um link único usando UUID para cada sessão, que é enviado para os participantes por meio de notificações, garantindo que ambos tenham acesso ao link;
- **Realização da sessão**: No horário agendado, os pacientes e profissionais acessam o link da sessão para realizar o atendimento de forma remota. Primeiro os participantes entram em uma sala de espera, onde são instruidos a permitir o acesso à câmera e microfone ao passo que o sistema verifica se o usuário tem autorizção para acessar a

sessão ou se o link ainda está válido. Após a verficação o usuário pode entrar na sala da sessão e após o profissional dar início que a sessão se torna ativa;

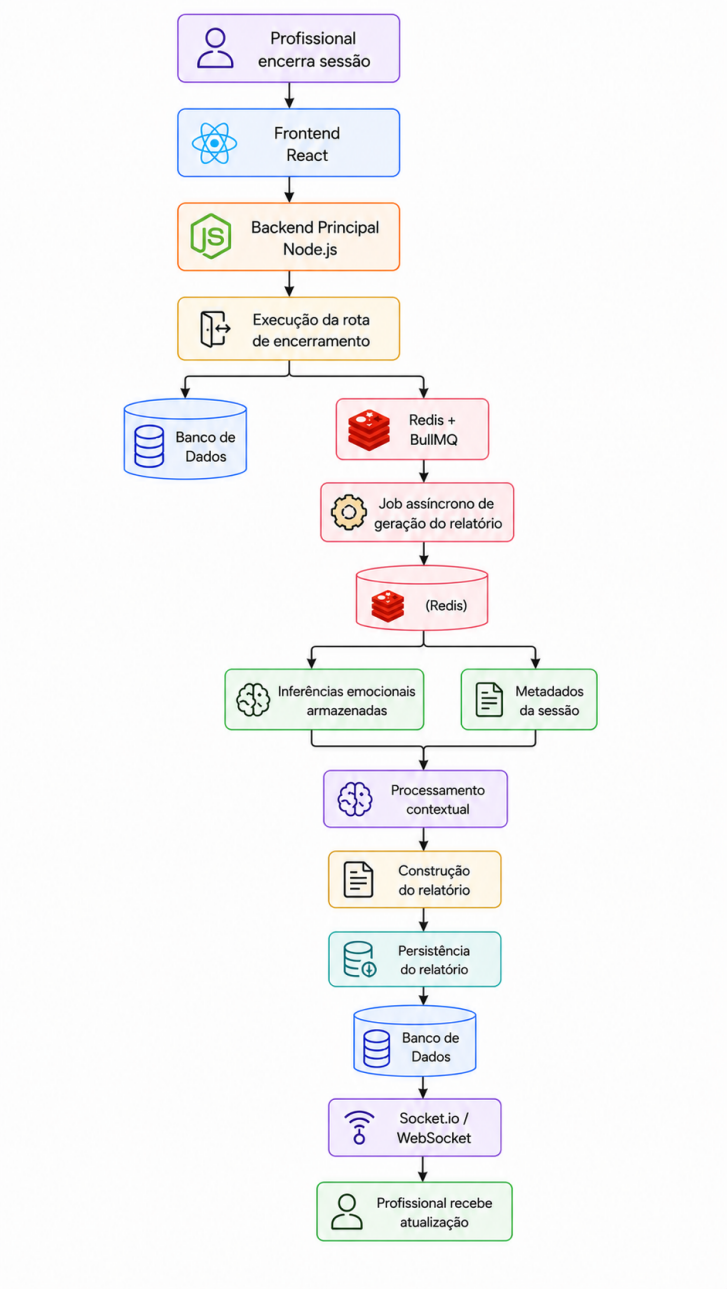
- **Acompanhamento emocional**: Periodicamente, durante a sessão frames do paciente são capturados e enviados para um microserviço para realizar uma inferência da emoção e o resultado é exibido para o profissional em tempo real.

4.8 Fluxo de Geração de Relatórios

O fluxo de geração de relatórios é um processo que permite que os profissionais possam visualizar informações detalhadas sobre as análises emocionais dos pacientes, logo após o encerramento da sessão. As inferências realizadas durante o atendimento são compiladas e gerado um relatório detalhado sobre o estado emocional do paciente ao longo do tempo. Esses relatórios são armazenados no banco de dados e podem ser acessados pelos profissionais a qualquer momento, oferecendo insights valiosos para o acompanhamento e suporte aos pacientes.

A Figura 19 apresenta um diagrama do fluxo de geração de relatórios.

Figura 19 – Diagrama do fluxo de geração de relatórios



Fonte: Próprio autor

O processo de geração de relatórios é composto por várias etapas, que serão descritas a seguir:

- **Encerramento da sessão**: Após o profissional encerrar a sessão, um processo assíncrono é iniciado para compilar as inferências realizadas durante o atendimento e gerar um relatório detalhado sobre o estado emocional do paciente;
- **Compilação dos resultados**: As inferências realizadas que foram armazenadas no *Redis* ao longo da sessão são recuperadas pelo *id* da sessão e compiladas para criar

um panorama completo do estado emocional do paciente. Depois disso, as inferências são excluídas do *Redis* para liberar espaço;

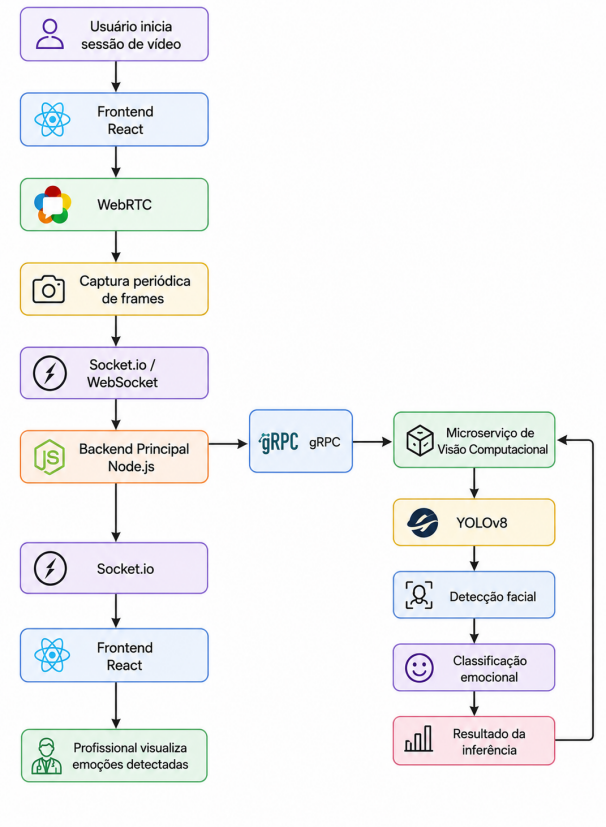
- **Geração do relatório**: Com base na compilação dos resultados, a plataforma gera um relatório detalhado sobre o estado emocional do paciente durante a sessão. O relatório inclui informações sobre as emoções predominantes e variações emocionais ao longo do tempo;
- **Armazenamento e acesso ao relatório**: Os relatórios gerados são armazenados no banco de dados e enviados via *WebSockets* para os profissionais.

4.9 Fluxo de Análise de Emoções durante as sessões

O fluxo de análise de emoções durante as sessões é um processo que permite que os profissionais possam acompanhar em tempo real o estado emocional dos pacientes durante os atendimentos online. Esse processo envolve a captura de frames do paciente durante a sessão, o envio desses frames para um microserviço de análise de emoções e a exibição dos resultados para os profissionais.

A Figura 20 apresenta um diagrama do fluxo de análise de emoções durante as sessões.

Figura 20 – Diagrama do fluxo de análise de emoções durante as sessões



Fonte: Próprio autor

O processo foi todo pensado em garantir a privacidade e rapidez. Desta forma todo o ciclo ocorre sobre o controle da plataforma, sem a necessidade de terceiros, o que evita possíveis riscos de segurança e garante que os dados dos pacientes sejam tratados de forma confidencial. A seguir, é descrito cada etapa do processo:

- **Captura de frames**: Durante a sessão, frames do paciente são capturados periodicamente. A captura é realizada de forma discreta e respeitosa, garantindo que os pacientes se sintam confortáveis durante o atendimento;
- **Envio para análise**: Os quadros (frames) capturados são convertidos para o formato de objeto binário (*Blob*) e transmitidos ao *backend* principal via *WebSocket*, que realiza o encaminhamento ao microsserviço por meio do protocolo *gRPC*. A utilização do *Blob* permite o envio dos dados em binário bruto de 8 bits, preservando as dimensões originais do arquivo e otimizando o consumo de largura de banda de rede tanto do cliente quanto do servidor. Ademais, o encaminhamento via *gRPC* assegura uma comunicação de baixa latência, visto que utiliza o HTTP/2 como protocolo de transporte (permitindo multiplexação de requisições na mesma conexão) e adota o Protocol Buffers (Protobuf) como formato de serialização binária, que é drasticamente mais leve e rápido de processar do que o tradicional JSON;

- **Exibição dos resultados**: Após a análise, o resultado é transmitido de volta ao *backend* principal para ser persistido no *Redis*. Em seguida, o dado é encaminhado via *WebSocket* ao *frontend*, sendo exibido exclusivamente ao profissional.

4.10 Desenvolvimento do Backend Principal

O *backend* principal foi desenvolvido adotando uma arquitetura baseada nos princípios do *Domain-Driven Design* (DDD), com o objetivo de organizar o sistema em módulos independentes e estritamente alinhados aos domínios de negócio da aplicação. Essa abordagem permitiu a separação clara de responsabilidades e a redução do acoplamento entre as funcionalidades implementadas. No escopo desta arquitetura, cada camada desempenha um papel fundamental, descritos a seguir:

- **Application (Aplicação)**: Camada responsável por coordenar a execução das tarefas do sistema e direcionar o fluxo de dados. Ela não contém regras de negócio complexas, funcionando como uma mediadora que recebe as requisições, aciona os casos de uso (*use cases*) e delega as ações necessárias para a camada de domínio.
- **Domain (Domínio)**: O núcleo (*core*) da aplicação. Esta camada é isolada de tecnologias externas e contém a lógica de negócio principal, as regras, validações e os modelos conceituais (como Entidades e Objetos de Valor). É a tradução mais fiel das regras de negócio para o código.
- **Infrastructure (Infraestrutura)**: Camada de suporte que fornece capacidades técnicas para as demais camadas. É onde se implementam os detalhes de tecnologia e recursos externos, como a persistência de dados (bancos de dados, repositórios), comunicação com APIs, serviços de mensageria, *WebSockets* e configurações do framework.
- **Interfaces / Presentation (Interfaces)**: Responsável por expor as funcionalidades do sistema para o mundo exterior. É nesta camada que se definem os pontos de entrada da aplicação, como controladores HTTP (APIs REST), rotas e serialização/deserialização de dados (DTOs).

Para promover o baixo acoplamento entre as camadas do sistema, aplicaram-se os princípios SOLID, com destaque para o Princípio da Inversão de Dependência (*Dependency Inversion Principle* — DIP). A adoção dessa abordagem garantiu que o núcleo de regras de negócio dependesse estritamente de abstrações (interfaces) e não de implementações concretas, favorecendo a modularidade, a testabilidade unitária e a extensibilidade do *software*. Adicionalmente, concebeu-se o uso do padrão *Singleton* para componentes compartilhados, assegurando a existência de uma instância única para

serviços globais essenciais. Para viabilizar essa estrutura, implementou-se o padrão *Factory*, centralizando a responsabilidade de instanciar e injetar as dependências dos módulos da aplicação, garantindo que a complexidade de inicialização dos serviços fique isolada do restante do sistema. Complementando a arquitetura, implementou-se o padrão de *Domain Events* (Eventos de Domínio) para intermediar a comunicação assíncrona e desacoplada entre diferentes componentes do modelo de negócios.

4.10.1 Detalhes da implementação do DDD

Esta seção descreve detalhadamente a organização de cada camada da aplicação, concentrando-se nos componentes fundamentais para o funcionamento do sistema. Estruturas secundárias ou auxiliares foram desconsideradas nesta análise para otimizar o escopo do texto. O detalhamento de cada nível arquitetural é apresentado nas subseções subsequentes:

4.10.1.1 Camada de Aplicação (Application)

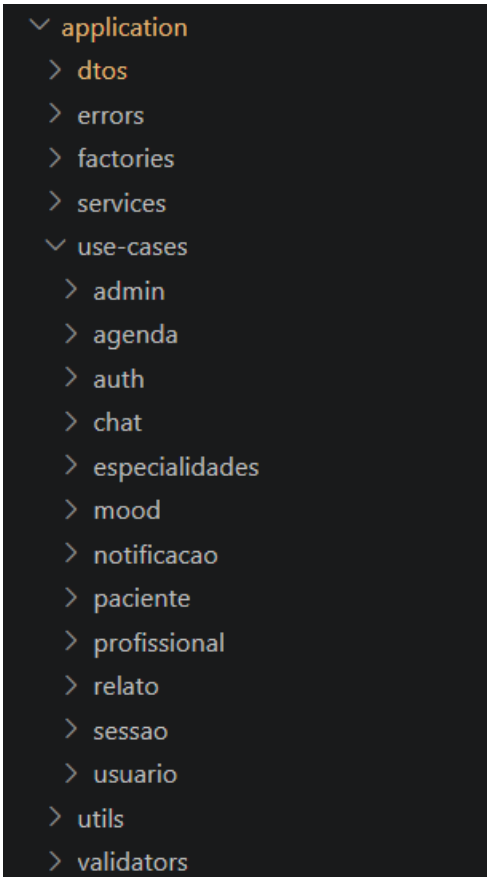
A camada de *Application* atua como uma integradora e coordenadora dos fluxos do sistema, sendo responsável por receber as requisições externas e direcioná-las aos fluxos de negócio. Conforme ilustrado na Figura 21, este nível arquitetural adota uma segmentação modular e bem delimitada para garantir a manutenibilidade e o isolamento de responsabilidades:

- **dtos**: Agrupa os *Data Transfer Objects* (Objetos de Transferência de Dados), estruturas de dados puras utilizadas para mapear e tipar a entrada e a saída de informações que trafegam entre as subcamadas.
- **errors**: Concentra a definição de classes de erro customizadas e específicas da aplicação, padronizando o tratamento de exceções antes que estas sejam propagadas para as camadas mais externas.
- **factories**: Fábrica responsável por encapsular a lógica de criação de usuário, serve para criar a casca na tabela corresponde ao tipo de usuário (paciente ou profissional) no banco de dados.
- **services**: Serviços de aplicação que contêm lógicas que transcendem um único caso de uso, mas que ainda não pertencem ao núcleo de domínio, servindo como utilitários compartilhados no escopo global deste nível.
- **use-cases**: Representa o núcleo executor das funcionalidades de negócio, organizado de forma granular por módulos da aplicação, como agenda, paciente, profissional, entre outros. Cada fluxo operacional possui uma lógica específica, implementada em

um caso de uso bem definido. Para verificar o comportamento dessas implementações, foram desenvolvidos testes automatizados utilizando o *framework Jest*. Esses testes são organizados em arquivos com a extensão *.spec.ts*, associados aos respectivos casos de uso, permitindo verificar suas regras de negócio e identificar possíveis falhas durante o desenvolvimento e a manutenção da aplicação.

- **utils**: Módulo reservado para funções utilitárias genéricas, empregadas no gerenciamento de operações temporais de datas e horas. Assim como os casos de uso, todas as ferramentas presentes nesta pasta contam com suporte de testes para mitigar regressões de *software*.
- **validators**: Diretório encarregado da validação sintática e de integridade dos dados de entrada antes que estes atinjam os casos de uso. Para assegurar regras rígidas de validação, utilizou-se a biblioteca *Joi*, mitigando a persistência ou processamento de payloads corrompidos ou mal-formados.

Figura 21 – Estrutura da application



Fonte: Próprio autor

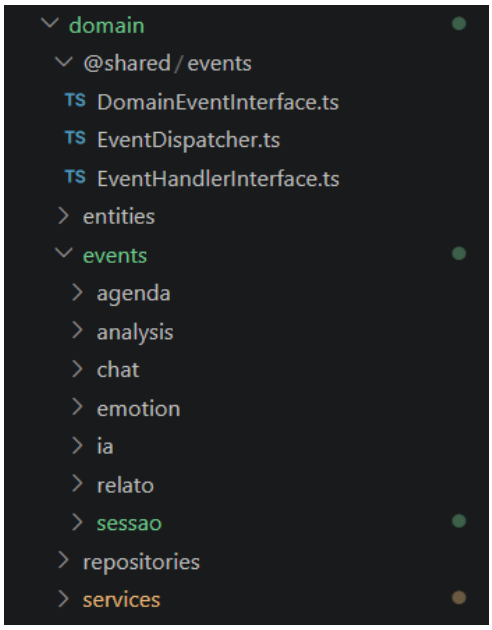
4.10.1.2 Camada de Domínio (Domain)

A camada de *Domain* representa o coração do sistema, contendo as regras de negócio mais puras, entidades independentes e as definições arquiteturais que guiam o comportamento do software. Livre de dependências de frameworks ou tecnologias externas, este nível adota uma organização orientada ao domínio (DDD) para isolar e proteger o núcleo da aplicação. Conforme ilustrado na Figura 22, esta camada segmenta suas responsabilidades da seguinte forma:

- **@shared / events**: Diretório reservado para os componentes globais e reutilizáveis de mensageria da camada. Agrupa interfaces fundamentais como *DomainEventInterface.ts* e *EventHandlerInterface.ts*, além do *EventDispatcher.ts*, que atua como o mediador responsável por registrar, gerenciar e disparar os eventos de domínio de forma desacoplada para toda a aplicação.
- **entities**: Concentra as entidades de negócio centrais da aplicação. Estes objetos possuem identidade própria e encapsulam o estado e os comportamentos essenciais do domínio, garantindo a consistência das regras de negócio em sua origem.
- **events**: Organiza os eventos de domínio específicos gerados pelo sistema, estruturados de forma modular e granular por subdomínios operacionais da plataforma. Cada pasta reflete uma funcionalidade de negócio:
  - *agenda*: Eventos relacionados a marcações, cancelamentos e atualizações de horários.
  - *analysis* e *emotion*: Eventos resultantes dos processos de detecção de emoções e geração de relatórios.
  - *chat*: Mensageria assíncrona gerada pela troca de mensagens internas.
  - *ia*: Eventos disparados por processamentos e respostas do módulo de inteligência artificial.
  - *relato*: Notificações de domínio atreladas ao envio ou modificação de relatos na plataforma.
  - *sessao*: Eventos que delimitam o ciclo de vida, início e encerramento das sessões de atendimento.
- **repositories**: Contém os contratos e interfaces dos repositórios. Estes arquivos definem os métodos de persistência e consulta necessários para o domínio, sem expor os detalhes de implementação do banco de dados.
- **services**: Diretório encarregado de concentrar as interfaces e contratos dos serviços globais e de infraestrutura utilizados pelo domínio. Esta pasta não contém

implementações concretas, mas sim as definições das abstrações necessárias para a comunicação com agentes externos e utilitários do sistema.

Figura 22 – Estrutura da domain



Fonte: Próprio autor

4.10.1.3 Camada de Infraestrutura (Infrastructure)

A camada de *Infrastructure* é responsável por fornecer suporte operacional e tecnológico às demais camadas do sistema, implementando de forma concreta as interfaces definidas no domínio e lidando diretamente com recursos externos, bibliotecas de terceiros e frameworks. Conforme ilustrado na Figura 23, esta camada organiza os seus módulos operacionais para garantir o desacoplamento arquitetural e a escalabilidade da plataforma:

- **config**: Possui um arquivo único de configuração do *Redis*
- **container**: Agrupa os arquivos de instânciação de classes, divididos por responsabilidades técnicas:
  - *eventContainer.ts*: Instancia os *handlers* (YOSHI, 2024) de evento de domínio.
  - *grpcContainer.ts*: Instancia o protocolo *gRPC*.
  - *repositoryContainer.ts*: Instancia os repositórios.
- **database**: Diretório focado na persistência e manipulação da base de dados relacional através do ORM *Sequelize*. Apresenta uma organização segmentada:
  - *mappers*: Classes responsáveis por mapear os modelos de dados da base de dados para as entidades de negócio puras do domínio e vice-versa.

  - *models*: Modelos e esquemas estruturais que refletem a tabela da base de dados no ecossistema do código.
  - *repositories*: Implementações concretas dos contratos de repositório do domínio, executando operações reais de consulta e escrita (utilizando *Sequelize* e *PostgreSQL*).
  - *seeders*: Scripts automatizados para popular a base de dados com registros iniciais de configuração ou testes.

- **events**: Contém as implementações concretas e os subscritores (*listeners*) que ouvem e respondem aos eventos disparados pelo *EventDispatcher* (SYMFONY DOCS, s.d.).

- **gateways**: Classes responsáveis por encapsular a comunicação e a integração com serviços externos ou intermediar protocolos de comunicação específicos. Nesta arquitetura, o módulo subdivide-se em dois componentes especializados:
  - *SessaoSignalingGateway*: Encarregado de gerenciar toda a infraestrutura de sinalização para as conexões *WebRTC*, permitindo o estabelecimento e a coordenação de fluxos de mídia ou dados em tempo real.
  - *SessaoAnalysisGateway*: Responsável por intermediar a comunicação com o serviço de análise de emoções, abstraindo o envio de dados e o recebimento dos resultados processados.

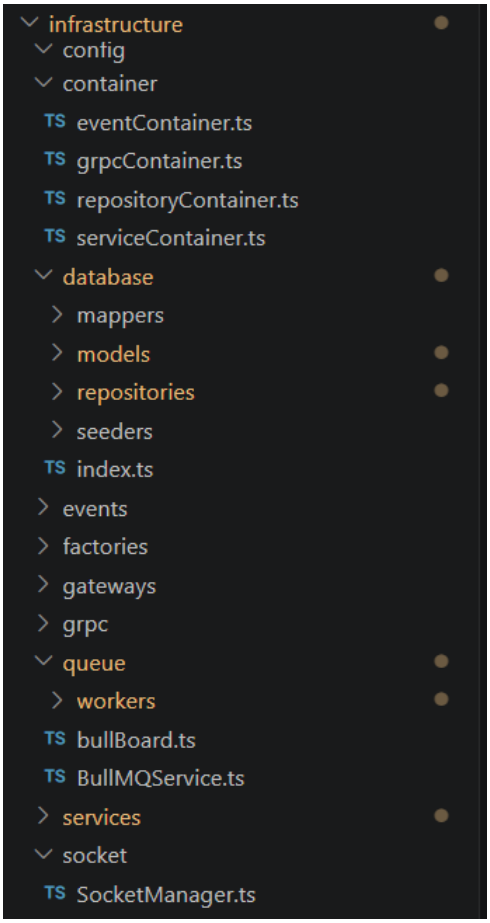
- **grpc**: Camada de transporte que aloja a implementação do servidor e dos clientes gRPC, servindo de ponte de comunicação direta de alta performance com microsserviços.

- **queue**: Subsistema responsável pela gestão de processamento assíncrono e tarefas em segundo plano baseadas em filas. Inclui a pasta *workers* para os executores de tarefas, o *bullBoard.ts* para a disponibilização de um painel visual de monitoramento e o *BullMQService.ts* para a integração robusta com o framework *BullMQ* (gerido via *Redis*).

- **services**: Aloja a codificação concreta dos serviços técnicos cujas interfaces foram estipuladas no domínio, tais como implementações de criptografia, envio de e-mails, SMS e serviços complementares.

- **socket**: Diretório encarregado da infraestrutura de comunicação em tempo real bidirecional. Contém o *SocketManager.ts*, componente focado no ciclo de vida das conexões baseadas em *WebSockets*, gerindo canais e o fluxo instantâneo de dados (como o sistema de chats).

Figura 23 – Estrutura da infraestrutura



Fonte: Próprio autor

4.10.1.4 Camada de Interface (Interface)

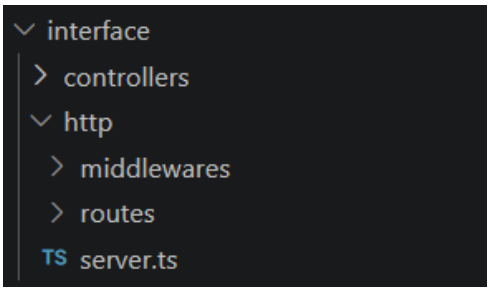
A camada de *Interface* (também conhecida como camada de apresentação ou adaptadores de entrada) é a porta de entrada da aplicação. Ela é responsável por expor os pontos de acesso do sistema para o mundo exterior, recebendo as requisições dos usuários ou clientes, traduzindo esses estímulos externos para o formato aceito pelos casos de uso e devolvendo a resposta apropriada. Conforme ilustrado na Figura 24 de diretórios, este nível organiza-se da seguinte forma:

- **controllers**: Agrupa os controladores da aplicação, componentes encarregados de receber os dados brutos das requisições, delegar a execução dos fluxos operacionais para os casos de uso correspondentes na camada de aplicação e formatar a resposta final (HTTP Status e Payload) que será enviada de volta ao cliente.
- **http**: Módulo dedicado especificamente ao protocolo de comunicação HTTP, isolando as configurações de rede e rotas do restante da lógica de apresentação:

  - *middlewares*: Contém os interceptadores de requisições, responsáveis por executar lógicas transversais antes que a requisição chegue ao controlador, tais como validação de tokens de autenticação, controle de permissões de acesso e tratamento global de erros.
  - *routes*: Concentra a definição das rotas e caminhos da API, mapeando os endpoints HTTP (métodos GET, POST, PUT, DELETE, entre outros) e associando-os aos seus respectivos *middlewares* e controladores.

- *server.ts*: Arquivo encarregado de instanciar e configurar de forma dedicada a aplicação *Express*. Suas atribuições incluem a injeção de middlewares globais (como a desserialização de payloads em *JSON* e o gerenciamento de *cookies*), a parametrização das regras de *CORS* para controle de origens permitidas, o acoplamento do ecossistema de rotas e o tratamento centralizado de exceções através do *errorHandler*. Por fim, exporta a instância configurada do *app* e a conexão com a base de dados (*db*) para serem inicializadas pelo ponto de entrada principal da aplicação.

Figura 24 – Estrutura da inteface



Fonte: Próprio autor

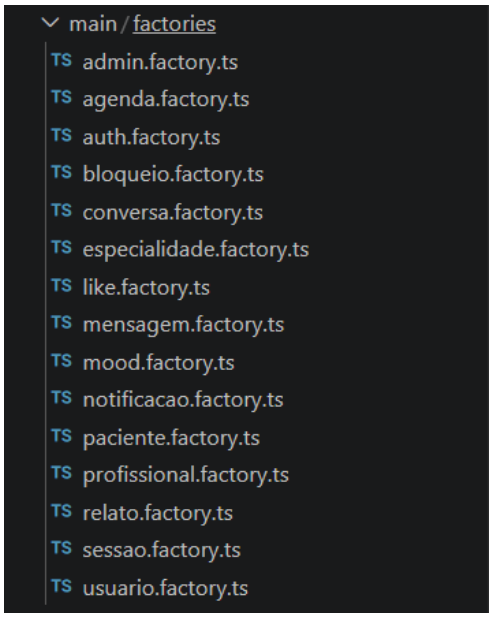
4.10.1.5 Ponto de Composição (Main / Factories)

A pasta *factories*, atua isolada das camadas conceituais puras do Domain-Driven Design (DDD). Este isolamento é uma decisão arquitetural estratégica que implementa o padrão *Composition Root* (Raiz de Composição). A sua função exclusiva é servir como o ponto centralizado onde a aplicação é montada, acoplada e injetada, evitando que as regras de negócio conheçam detalhes de frameworks ou de persistência. Conforme ilustrado na Figura 25, o diretório segmenta-se em fábricas especializadas por subdomínios operacionais:

- **Orquestração de Casos de Uso**: Cada arquivo (como *agenda.factory.ts*, *paciente.factory.ts*, *sessao.factory.ts* e *usuario.factory.ts*) é responsável por instanciar os casos de uso correspondentes da camada de *Application*, injetando as implementações concretas de repositórios e serviços técnicos desenvolvidos na camada de *Infrastructure*.

- **Centralização de Instâncias (Singleton/Containers)**: Ao concentrar a lógica de instanciação em um único ponto isolado, o sistema garante que as dependências globais (como a conexão do banco de dados, clientes gRPC ou despachantes de eventos) sejam propagadas de forma consistente e reaproveitadas por toda a aplicação.
- **Preservação da Inversão de Dependência**: Ao assumir a responsabilidade de "amarrar"o sistema, o módulo de *factories* impede que camadas internas tenham que instanciar as suas próprias dependências, salvaguardando a pureza e a testabilidade do *Domain* e da *Application*.

Figura 25 – Ponto de Composição



Fonte: Próprio autor

4.11 Desenvolvimento do Frontend

O ecossistema cliente da aplicação foi desenvolvido utilizando a biblioteca *React*, adotando uma abordagem arquitetural moderna orientada a componentes modulares e reutilizáveis. O foco principal do desenvolvimento desta camada concentrou-se na separação rígida de responsabilidades, otimização de renderizações, legibilidade do código e na garantia de uma experiência de usuário fluida e responsiva, operando de forma totalmente desacoplada das regras de negócio do servidor Backend.

4.11.1 Princípios de Design e Gerenciamento de Estado

Para sustentar as regras de apresentação da interface de forma escalável, o frontend baseia-se em pilares estruturais que eliminam a necessidade de bibliotecas externas complexas de estilização ou de estado global:

- **Componentização Reutilizável**: A interface foi fragmentada de forma granular. Elementos comuns de interface (*UI Components*) foram isolados em componentes puros. Esses módulos recebem dados e comportamentos via propriedades (*props*), permitindo a sua reutilização em diferentes fluxos e telas da aplicação, reduzindo a duplicidade de código.

- **Gerenciamento de Estado com Context API**: O estado global da aplicação e o compartilhamento de informações transversais — como os dados do usuário autenticado, fluxos de chat em tempo real e notificações — são gerenciados de forma nativa através da *Context API* do React. Isso viabiliza um fluxo de dados limpo e centralizado, mitigando o problema de acoplamento excessivo por propagação de propriedades (*prop drilling*).

- **Encapsulamento de Lógica com Hooks Customizados e Axios**: Toda a lógica de efeitos colaterais, consumo de APIs e estados locais complexos foi extraída dos componentes visuais e isolada em funções especializadas denominadas *Hooks* customizados. A aplicação estrutura duas abstrações principais: o gerenciamento centralizado do ciclo de vida de notificações do sistema e o controle dos fluxos de vídeo via *WebRTC*, integrando a captura de mídia em tempo real a análise de emoções. Para a comunicação assíncrona com o ecossistema *backend*, utilizou-se a biblioteca *Axios* através de uma instância cliente configurada globalmente com interceptadores para anexação automatizada de credenciais e tratamento padronizado de erros. Essa separação garante que os componentes fiquem encarregados majoritariamente da renderização estrutural, elevando a testabilidade e o reaproveitamento do código.

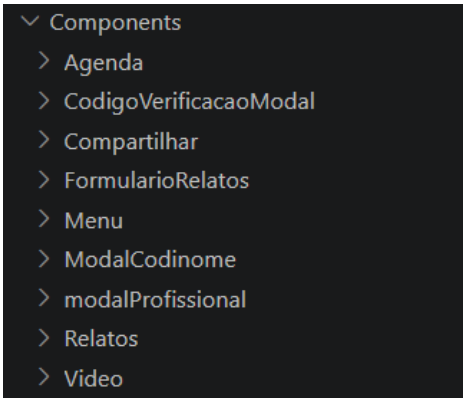
- **Estilização com CSS Nativo e Responsividade**: A identidade visual da plataforma foi desenvolvida utilizando exclusivamente *CSS* nativo, eliminando a dependência e o impacto na performance (*overhead*) de frameworks externos ou bibliotecas de *CSS-in-JS*. A interface foi projetada para ser totalmente responsiva por meio da combinação de *Flexbox*, *CSS Grid* e regras de *media queries*. Essa abordagem garante que o *layout* se adapte de forma fluida e orgânica a qualquer resolução de tela, assegurando uma experiência de usuário consistente tanto em dispositivos móveis quanto em desktops.

4.11.2 Telas e Componentes

Para viabilizar o desenvolvimento da interface, estruturou-se uma separação clara e modular de responsabilidades em pastas e arquivos. A divisão de diretórios reflete o desacoplamento entre componentes visuais atômicos (reutilizáveis) e as páginas estruturadas (telas completas), as quais combinam esses blocos menores para consolidar a experiência do usuário nos três perfis distintos da plataforma.

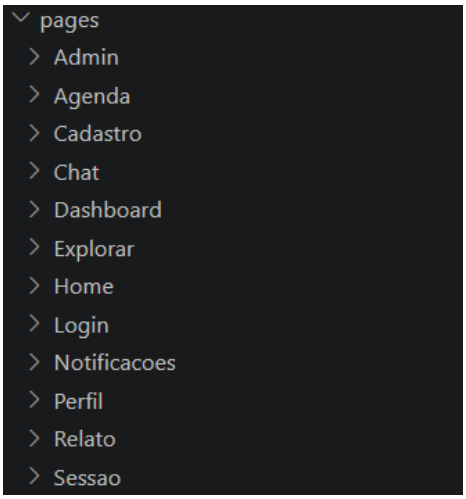
A organização física e hierárquica desses elementos é apresentada nas Figuras 26 e 27, que ilustram, respectivamente, a árvore de diretórios dos componentes de interface e das páginas de visualização:

Figura 26 – Estrutura de diretórios dos componentes reutilizáveis.



Fonte: Próprio autor

Figura 27 – Estrutura de diretórios das páginas do sistema.



Fonte: Próprio autor

A distribuição das responsabilidades técnicas, lógicas e visuais contidas nas pastas do sistema subdivide-se detalhadamente nos seguintes tópicos:

- **components (Componentes Reutilizáveis)**: Diretório focado no encapsulamento de elementos de interface modulares. Cada componente possui sua própria lógica de apresentação acoplada a um arquivo de estilização *CSS* isolado, operando de forma independente:
  - *Agenda*: Módulo encarregado de centralizar as interfaces de caixas de diálogo e modais vinculados ao fluxo cronológico. Engloba as janelas para a marcação de

consultas, gerenciamento de bloqueios de horários, cancelamentos de sessões e da disponibilidade do especialista.

- *CodigoVerificacaoModal*: Componente especializado na renderização de um modal para a validação de códigos de segurança, sendo reaproveitado tanto nos fluxos de verificação de e-mail quanto de números de telefone.
- *Compartilhar*: Componente responsável pela gestão do sistema de compartilhamento de agendas, permitindo que o profissional de saúde gere e disponibilize o acesso aos seus horários diretamente para os pacientes.
- *FormularioRelatos*: Estrutura de formulário padrão e fortemente tipada utilizada para a criação, escrita e submissão de relatos emocionais na plataforma.
- *Menu*: Barra de navegação lateral da aplicação. Apresenta uma arquitetura condicional que renderiza um menu específico para administradores e outro adaptável para pacientes e profissionais. O menu comum exibe opções dinamicamente com base nas permissões concedidas ao tipo de usuário autenticado.
- *ModalCodinome* e *ModalProfissional*: Interfaces em formato de caixa de diálogo projetadas para o preenchimento e a configuração inicial do perfil de pacientes e profissionais.
- *Relatos*: Componente focado na exibição, formatação e interação com os relatos armazenados, implementando regras de visualização distintas conforme o ator (paciente ou profissional) que o está acessando.
- *Video*: Centraliza os componentes responsáveis pelo fluxo multimídia da aplicação, encarregando-se da renderização da transmissão de vídeo e da exibição síncrona das emoções detectadas ao longo das sessões clínicas.

- **pages / views (Telas do Sistema)**: Módulo que concentra as páginas completas da aplicação, organizadas de forma lógica e segmentadas conforme o nível de privilégio e o contexto de negócio. Neste nível, os componentes reutilizáveis descritos anteriormente são importados, injetados e compostos para materializar os fluxos da plataforma:
  - *Admin*: Páginas de acesso restrito ao perfil de administrador. Contém um *dashboard* gerencial com indicadores consolidados do sistema, uma tela central de gerenciamento de usuários e duas interfaces específicas para a auditoria: uma dedicada à validação de credenciais profissionais e outra voltada exclusivamente à visualização de registros de pacientes.
  - *Agenda*: Contempla duas páginas distintas focadas no cronograma do sistema. A primeira consiste na área de trabalho do profissional, onde este parametriza sua grade de horários de atendimento, enquanto a segunda é disponibilizada ao paciente para a realização de agendamentos.

- *Cadastro*: Tela destinada ao fluxo de inserção e criação de novas contas de usuários na plataforma.
- *Chat*: Página dedicada ao ecossistema de mensageria instantânea e assíncrona da aplicação, intermediando a conversa entre os pares.
- *Dashboard*: Fornece painéis de controle customizados e divididos por papéis. O painel do profissional exibe dados analíticos relevantes à sua rotina (como o volume de consultas semanais, agendamentos mensais ativos e conteúdos informativos da área). Em contrapartida, o painel do paciente adota uma abordagem de suporte e acolhimento clínico, integrando visualizações de relatos recentes, cartões informativos, registro de estados emocionais e um módulo interativo de respiração guiada para mitigação de crises de ansiedade.
- *Explorar*: Interface de busca focada na listagem de profissionais disponíveis na plataforma, fornecendo mecanismos de paginação e filtragem avançada por especialidades.
- *Home*: Página pública inicial do sistema, possuindo caráter institucional e explicativo sobre os recursos da plataforma.
- *login*: Interface simples para a autenticação e inserção de credenciais de acesso ao sistema.
- *Notificações*: Centralizadora de alertas do usuário. Realiza a separação visual dos avisos por meio de esquemas de cores e incorpora funcionalidades contextuais dinâmicas dependendo da natureza do alerta.
- *Perfil*: Agrupa três páginas de configuração de dados. Duas são destinadas à visualização e edição interna de informações pessoais do profissional e do paciente, enquanto a terceira compreende a página pública do profissional, exibindo seu currículo, competências e avaliações recebidas.
- *Relato*: Tela específica para o gerenciamento estruturado de relatos textuais, equipada com componentes integrados de filtragem e paginação de dados.
- *Sessao*: Módulo composto por seis páginas que orquestram o ciclo de vida dos atendimentos. A primeira atua como sala de espera virtual, efetuando testes de hardware e coletando o consentimento para uso de câmera e microfone. A segunda abriga a sala de atendimento síncrono, onde ocorrem a videoconferência e o monitoramento emocional em tempo real. A terceira e quarta telas compreendem o pós-atendimento, direcionando o paciente para a avaliação do serviço e informando o profissional sobre o processamento assíncrono do relatório clínico. Por fim, a quinta e a sexta páginas destinam-se à gestão documental, fornecendo uma listagem histórica e uma tela de visualização detalhada dos relatórios gerados pela inteligência artificial.

4.11.2.1 Construção da Agenda

A interface gráfica da agenda foi desenvolvida utilizando a biblioteca *React Big Calendar*, que fornece uma estrutura robusta e pré-configurada para a exibição de calendários em visualizações de mês, semana, dia e agenda de compromissos. A escolha desta ferramenta justificou-se pela sua alta capacidade de customização e suporte nativo a eventos dinâmicos, permitindo alinhar a experiência visual às necessidades clínicas da plataforma.

Diferente de implementações estáticas, a agenda opera de forma totalmente reativa, sendo alimentada em tempo real com dados provenientes do ecossistema *backend*. O fluxo de funcionamento e renderização deste módulo baseia-se nos seguintes aspectos técnicos:

- **Consumo Assíncrono de Dados**: Ao renderizar a página, os ganchos de ciclo de vida (*useEffect*) disparam requisições HTTP via *Axios* para os *endpoints* do servidor, solicitando a grade de horários disponíveis, bloqueios ou consultas já agendadas para o intervalo de datas visível.
- **Tratamento e Mapeamento de Objetos**: Como a biblioteca *React Big Calendar* exige uma estrutura de dados estrita para renderizar os eventos no mapa visual, o frontend executa uma função de mapeamento (*data parsing*). Esse processo converte as strings de data/hora no padrão ISO vindas do banco de dados *PostgreSQL* em objetos nativos *Date* do JavaScript, associando propriedades obrigatórias como *title*, *start*, *end* e metadados de identificação.
- **Interatividade e Acoplamento de Modais**: Em vez de utilizar as caixas de diálogo nativas do navegador, as ações de clique na agenda foram customizadas. O disparo dos eventos *onSelectSlot* (clicar em um espaço vazio) ou *onSelectEvent* (clicar em um compromisso existente) intercepta o comportamento padrão da biblioteca para abrir dinamicamente os componentes especializados, tais como os modais de agendamento, bloqueio ou cancelamento de horários.
- **Diferenciação Visual por Estado**: Através da manipulação de propriedades de estilização (*eventPropGetter*), a agenda aplica classes *CSS* e esquemas de cores distintos para cada tipo de compromisso (por exemplo, diferenciando visualmente um horário disponível, uma consulta confirmada ou um período de bloqueio configurado pelo profissional), otimizando a legibilidade e a navegação do usuário.

4.11.3 Chat e suas funcionalidades

O chat foi desenvolvido focado na simplicidade e segurança. Ele é integrado à plataforma e permite apenas que os integrantes da conversa possam enviar e receber

mensagens de texto. Apesar da simplicidade, o chat atende aos objetivos propostos pela plataforma. Abaixo, é descrito as funcionalidades do chat:

- **Envio e recebimento de mensagens**: O chat permite que pacientes e profissionais enviem e recebam mensagens de texto. O recurso não oferece suporte ao envio de conteúdo multimídia, como imagens ou arquivos, mantendo a comunicação restrita a mensagens textuais;
- **Criptografia das mensagens**: As mensagens trocadas no chat são criptografadas utilizando o algoritmo de criptografia simétrica AES-256, com o objetivo de proteger o conteúdo armazenado e tratado pela aplicação. Essa criptografia atua sobre o conteúdo das mensagens, não substituindo a proteção do canal de comunicação por HTTPS (TLS). Portanto, na configuração utilizada, embora o conteúdo das mensagens seja criptografado pela aplicação, a transmissão é feita usando HTTP e não oferece a proteção de transporte proporcionada pelo HTTPS;
- **Registro das conversas**: Todas as mensagens são registradas no banco de dados, permitindo que os usuários possam acessar o histórico de suas interações;
- **Notificações de novas mensagens**: Os usuários recebem notificações quando novas mensagens são recebidas. Todo o mecanismo de notificações foi desenvolvido utilizando o *WebSocket* para garantir uma comunicação em tempo real entre os usuários, além do uso de um *Context API* que engloba toda o front e permite que as mensagens recebidas sejam gerenciadas de forma global, garantindo que as notificações sejam exibidas corretamente independentemente da tela em que o usuário esteja navegando;
- **Detalhes de experiências**: O chat permite que usuarios possam deletar ou editar mensagens enviadas com um limite de tempo de 5 minutos após o envio, garantindo que os usuários possam corrigir possíveis erros ou retirar as enviadas por engano, sem comprometer a integridade das conversas. Além disso, os usuários podem visualizar detalhes sobre as mensagens, como data e hora de envio, e se a mensagem foi lida pelo destinatário. Um mecanismo de rolagem automática ao receber novas mensagens foi implementado, garantindo que os usuários possam acompanhar as conversas em tempo real sem precisar rolar manualmente para visualizar as mensagens mais recentes;
- **Responsividade**: O chat é responsivo e se adapta a diferentes tamanhos de tela, garantindo uma experiência de usuário consistente e agradável tanto em dispositivos móveis quanto em *desktops*;
- **Carregamento eficiente**: O carregamento das mensagens é otimizado para garantir uma experiência fluida, mesmo em conversas com um grande número de mensagens.

O chat utiliza técnica de paginação *cursor*, fazendo com que as mensagens sejam carregadas em blocos à medida que o usuário rola a conversa, evitando sobrecarregar a interface e o navegador com um grande volume de mensagens de uma só vez;

- **Mecanismo de segurança**: O chat é protegido por um mecanismo de autenticação e autorização, garantindo que apenas os usuários autorizados possam acessar as conversas e enviar mensagens, protegendo a privacidade e segurança dos usuários.

4.12 Desenvolvimento da Mensageria

O processo de criação do processamento assíncrono se deu utlizando *Redis* e *BullMQ*, com o objetivo de desacoplar operações de longa duração do fluxo principal da aplicação. A adoção dessa arquitetura permitiu que tarefas computacionalmente mais custosas fossem executadas em segundo plano, evitanto bloqueios nas requisições realizadas pelos usuários e melhorando a experiência de utilização da plataforma.

A arquitetura de mensageria assíncrona foi projetada para garantir que o servidor principal permaneça altamente disponível, delegando tarefas pesadas — como o processamento de relatórios por inteligência artificial e o disparo de notificações — para uma fila de processamento gerenciada em segundo plano.

A materialização desse ecossistema distribui-se de forma modular entre configurações, contratos e implementações concretas, estruturadas da seguinte forma:

- **Configuração da Instância do Redis (*infrastructure/config/Redis.ts*)**: Este arquivo centraliza os parâmetros de conexão e credenciais necessárias para estabelecer a comunicação com o banco de dados em memória *Redis*. Ele instancia o cliente de conexão de forma otimizada, garantindo o uso de *pools* de conexões reutilizáveis tanto para a publicação de novas tarefas quanto para a escuta ativa realizada pelos consumidores.

- **Contrato e Tipagem da Fila (*application/services/IQueueService.ts*)**: Alinhado aos princípios de inversão de dependência, este arquivo estipula as regras do motor de filas no nível de domínio. Ele define a estrutura *QueueOptions*, mapeando as configurações de atraso (*delay*), tentativas de reexecução (*attempts*) e estratégias de *backoff*. Adicionalmente, estabelece a interface *IQueueService*, que dita a assinatura dos métodos abstratos para a adição e gerenciamento de *jobs* no sistema.

- **Implementação Concreta com BullMQ (*infrastructure/queue/BullMQService.ts*)**: Consiste na classe que implementa fisicamente a interface *IQueueService*. Utilizando a biblioteca *BullMQ*, este serviço encapsula a lógica de criação das filas (*Queues*) do

Redis, traduzindo as chamadas de métodos abstratos em inserções reais de dados no *Redis*.

- **Processadores Assíncronos (*infrastructure/queue/workers/*)**: Os consumidores da fila, denominados *workers*, foram isolados nesta pasta e organizados de forma granular por categorias de negócio (como envio de e-mails, geração de relatórios, entre outros). Cada arquivo de *worker* executa estritamente uma única responsabilidade de computação demorada. Para unificar o ciclo de vida desses processos, a pasta conta com um arquivo *index.ts*, responsável pela inicialização única e em lote de todos os escutadores.

O Código 4.1 apresenta um exemplo de implementação de um *worker* especializado na execução de tarefas relacionadas à inteligência artificial, como a análise de gravidade de relatos e a geração de cartões informativos (*cards*). Este *worker* é configurado para operar com uma concorrência máxima de cinco tarefas simultâneas, garantindo eficiência sem sobrecarregar os recursos do servidor.

```
export function setupIAWorker(iaAnalisarGravidade: IIAnalisarGravidade, iaCardsService: IIACardsService) {
    const worker = new Worker('ia-analise-geracao', async (job: Job) => {
        switch (job.name) {
            case 'analisar-gravidade' :
                return  await iaAnalisarGravidade.gravidade(job.data);

            case 'gerar-cards' :
                return await iaCardsService.gerarCardsMood(job.data)
        }
    }, {
        connection: RedisConnection,
        concurrency: 5 // Permite processar até 5 jobs simultaneamente
    });

    worker.on('failed', (job, err) => {
        if (job?.name === 'analisar-gravidade') {
            console.error(`[IAWorker] Análise de gravidade para relato ${job?.data.relatoId} falhou permanentemente: ${err.message}`);
        } else if (job?.name === 'gerar-cards') {
            console.error(`[IAWorker] Geração de cards para usuário ${job?.data.usuarioId} falhou permanentemente: ${err.message}`);
        } else {
            console.error(`[IAWorker] Job ${job?.name} falhou permanentemente: ${err.message}`);
        }
    });
```

```
}
```

Código 4.1 – Código exemplo de criação de um *worker* para consumir os *jobs*

4.12.1 Fluxo de Disparo a partir dos Casos de Uso

Para preservar o isolamento das camadas e evitar o acoplamento da infraestrutura web com o ciclo de processamento pesado, o gatilho para a criação de um novo *job* ocorre diretamente dentro da camada de *Application*, especificamente nos arquivos de *use-case*.

Quando uma ação de negócio exige um processamento demorado (como a finalização de uma sessão que demanda um relatório analítico da IA), o caso de uso invoca o método da interface *IQueueService* recebida via injeção de dependência. O caso de uso apenas envia os parâmetros identificadores necessários e finaliza imediatamente a sua execução, retornando uma resposta HTTP de sucesso ao cliente em poucos milissegundos. Paralelamente, o *BullMQService* publica a tarefa no *Redis*, fazendo com que um *worker* especializado a capture de forma assíncrona para concluir o processamento pesado em segundo plano.

O Código 4.2 ilustra um exemplo de disparo de um *job* para a geração de relatórios analíticos ao final de uma sessão clínica.

```
1       await this.analiseQueue.addJob('analise-sessao', 'gerar-
    relatorio',{
2           sessaoId: sessao.id,
3           pacienteId: sessao.paciente_id,
4           profissionalId: sessao.profissional_id
5       },);
```

Código 4.2 – Código exemplo de um *job*

4.13 Desenvolvimento do Sistema de Comunicação em Tempo Real

Para atender à necessidade de interações instantâneas, desenvolveu-se um ecossistema de comunicação em tempo real estruturado em duas frentes tecnológicas complementares: o tráfego de dados orientados a eventos (mensageria via chat e notificações do sistema) e a sinalização de videoconferências para a realização das sessões. A engenharia de software tradicional baseada no modelo de requisição-resposta do protocolo HTTP convencional (*Short/Long Polling*) mostrou-se ineficiente para este cenário, dado o alto consumo de banda e o atraso latente na entrega das mensagens.

Para contornar essas limitações, a arquitetura do sistema adotou o protocolo *WebSockets*, viabilizado por meio da biblioteca *Socket.io*, para estabelecer canais bidirecionais e persistentes de texto e eventos. Complementarmente, para a transmissão estável de fluxos de áudio e vídeo de alta qualidade com baixa latência, integrou-se o protocolo *WebRTC*

(*Web Real-Time Communication*), permitindo o tráfego de mídia diretamente entre os clientes conectados.

4.13.1 Arquitetura e Gerenciamento de Conexões com Socket.io

O *Socket.io* atua como o motor de mensageria da plataforma. A escolha por esta tecnologia, em detrimento do uso da API nativa de *WebSockets*, justifica-se pelos seus mecanismos integrados de resiliência, que incluem a reconexão automática em caso de instabilidade, suporte a *fallbacks* de transporte e a facilidade no isolamento de fluxos através de salas (*rooms*).

A implementação do sistema de comunicação bidirecional em tempo real foi centralizada no ecossistema *backend*, utilizando a classe *SocketManager* (conforme apresentado no Código 4.3). Este componente abstrai o acoplamento direto com o *framework* de servidor, isolando o tráfego de dados por meio de *namespaces* e salas específicas para otimizar o uso da memória do servidor.

4.13.1.1 Orquestração e Segurança no Backend

A inicialização do servidor híbrido ocorre no arquivo *src/server.ts*, onde a instância do *Socket.io* é acoplada ao servidor HTTP nativo do *Node.js*, aplicando regras estritas de *CORS* baseadas em funções de *callback*. Uma vez estabelecido o canal, a classe *SocketManager* assume o controle do ciclo de vida, conforme detalhado no código a seguir:

```
export class SocketManager {
    constructor(
        private io: Server,
        private tokenService: ITokenService,
        private grpcClient: IAnalisadorGrpcClient,
        private conversaRepository: IConversaRepository,
        private eventDispatcher: EventDispatcherInterface
    ) {}

    public init(): void {
        // 1. Middleware de Autenticacao Global (Namespace "/")
        this.io.use(verifySocketToken(this.tokenService));

        // 2. Configura os Namespaces especificos (Isolamento de trafego)
        this.setupSignalingNamespace();
        this.setupAnalysisNamespace();

        // 3. Configura o Namespace Padrao (Chat e Notificacoes)
        this.setupDefaultNamespace();
        console.log('Socket.io: Todos os Namespaces e Gateways inicializados.');
```

```
21      }
22
23      private setupDefaultNamespace() {
24          this.io.on('connection', (socket: AuthenticatedSocket) => {
25              const usuario = socket.usuario;
26              if (usuario?.id) {
27                  socket.join(`usuario_${usuario.id}`);
28              }
29
30              socket.on('join_conversa', async (conversaId: number) => {
31                  const conversa = await this.conversaRepository.
    buscarPorId(conversaId);
32                  if (conversa && (conversa.pacienteId === usuario?.id ||
    conversa.profissionalId === usuario?.id)) {
33                      socket.join(`conversa_${conversaId}`);
34                  } else {
35                      socket.emit('error_chat', 'Acesso negado a conversa'
    );
36                  }
37              });
38          });
39      }
40  }
```

Código 4.3 – Gerenciador centralizado de WebSockets (*SocketManager.ts*)

Para impedir conexões anônimas ou maliciosas, desenvolveu-se o middleware interceptador *verifySocketToken.ts*. Ele extrai de forma segura o *accessToken* armazenado nos *cookies* HTTP da requisição de aperto de mão (*handshake*). Caso o token seja validado com sucesso pelo *TokenService*, as propriedades de identificação e tipo do usuário são injetadas diretamente na tipagem estendida do *AuthenticatedSocket*, disponibilizando os dados de domínio para as regras de negócio subsequentes.

4.13.1.2 Sincronização e Ciclo de Vida Reativo no Frontend

No lado do cliente (*frontend*), o ciclo de vida das conexões persistentes e o gerenciamento do estado em tempo real foram estruturados por meio do arquivo *SocketContext.tsx*. Esse módulo implementa um provedor global (*Provider*) encarregado de instanciar a biblioteca *socket.io-client*, garantindo que uma única instância do cliente seja compartilhada por toda a aplicação.

O fluxo de gerenciamento de rede e conexão executado por essa camada baseia-se nos seguintes pilares de implementação:

- **Resolução Dinâmica de Endereço (URL)**: Para suportar o desenvolvimento em ambiente local multiproduto e redes heterogêneas, o arquivo implementa a rotina

getSocketUrl(). Através de expressões regulares, a função analisa o *hostname* da janela do navegador; caso identifique que o cliente está rodando em *localhost* ou dentro de faixas de IPs de redes locais (como 192.168.x.x), aponta a conexão para a porta dedicada do backend (*8000*), caso contrário, adota o *origin* de produção de forma transparente.

- **Prevenção de Múltiplas Instâncias com useMemo**: O cliente do socket é inicializado de forma tardia e encapsulado por um hook *useMemo* com um *array* de dependências vazio. Isso blinda o sistema contra os ciclos constantes de renderização do React, assegurando que o handshake inicial de conexão e os registradores de de ciclo de vida (*connect*, *disconnect*, *connect_error*) permaneçam associados a um único objeto de transporte estável.
- **Conexão Condicional e Sincronismo de Domínio**: Em vez de conectar o socket imediatamente, o provedor utiliza uma estratégia orientada ao estado de autenticação do usuário, conforme demonstrado a seguir:

```
// Conectar apenas após o usuário ser devidamente carregado
useEffect(() => {
  if (loadingUsuario) return;

  if (!usuario) {
    console.log("Sem usuario - socket permanece desconectado");
    return;
  }
  console.log("Conectando socket...");
  socket.connect();
}, [loadingUsuario, usuario, socket]);
```

Código 4.4 – Gerenciamento e conexão controlada do Socket no frontend (*SocketContext.tsx*)

Como observado no código, a conexão física é postergada até que o *UserContext* finalize o processamento dos dados do usuário (*loadingUsuario*). No momento em que o método *socket.connect()* é acionado, a biblioteca anexa automaticamente os *cookies* de sessão à requisição de *handshake*. Esse comportamento permite que o *SocketManager* no *backend* intercepte o *token*, valide a identidade do cliente e o aloque nas respectivas salas de mensagens e notificações de maneira totalmente transparente e assíncrona.

Por fim, os dados reativos de mensagens e notificações são gerenciados de forma complementar através do *ChatContext.jsx* e *NotificationContext.jsx*. Eles consomem o socket injetado pelo provedor, registrando e limpando dinamicamente escutadores ativos (como os eventos *'nova_mensagem_chat'*, *'mensagens_lidas'* e *'nova_notificacao'*), atualizando

as contagens globais de notificações da interface em tempo real à medida que as interações ocorrem.

4.13.2 Mecanismo de Sinalização e Transmissão Multimídia com WebRTC

A infraestrutura voltada à transmissão síncrona de áudio e vídeo foi materializada sob o ecossistema *WebRTC*. Como o protocolo opera de forma descentralizada através de conexões ponto a ponto (*Peer-to-Peer*, ou *P2P*), o desenvolvimento dividiu-se em duas frentes fundamentais no *backend*: um caso de uso para governança de segurança e ao provisionamento de rede, e um *Gateway* de sinalização dedicado executado sobre um *namespace* isolado do *Socket.io*.

4.13.2.1 Governança de Acesso e Provisionamento de Rede (STUN/TURN)

Antes de estabelecer a conexão de mídia, a aplicação exige uma validação estrita de segurança. Esse fluxo é orquestrado pelo caso de uso *GerarAcessoSessaoUseCase*, pertencente à camada de *Application*. Sua responsabilidade é verificar se a sessão solicitada existe, se ela encontra-se em estado elegível (não cancelada ou finalizada) e se o usuário requisitante é, de fato, um participante legítimo do atendimento (paciente ou profissional de saúde vinculado), mitigando vulnerabilidades de acesso indevido.

Uma vez concluídas as verificações de autorização, o caso de uso é responsável por disponibilizar aos clientes a configuração da infraestrutura *ICE* utilizada durante o estabelecimento da conexão WebRTC. Para isso, foram configurados servidores *STUN* e *TURN* utilizando a solução de código aberto *Coturn*, permitindo que os navegadores obtenham os candidatos *ICE* necessários para negociação da conexão entre os participantes da sessão.

```
1  // Configuração dos Servidores de Rede (ICE Servers)
2  private buildIceServers( username: string, credential: string ):
       IceServer[] {
3
4          return [
5
6              {
7                  urls: `stun:${this.config.host}:${this.config.port}`
8              },
9
10             {
11                 urls: `turn:${this.config.host}:${this.config.port}`,
       username, credential
12             }
13         ];
```

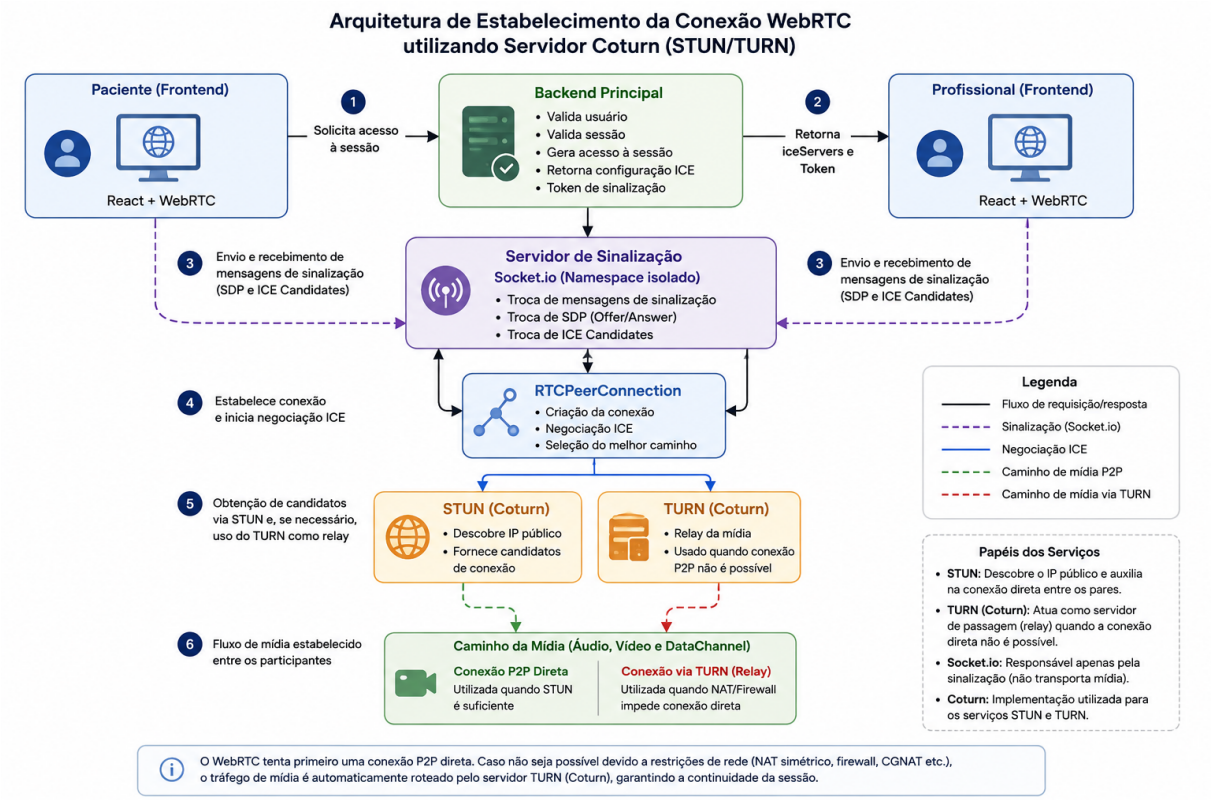
```
14      }
```

Código 4.5 – Caso de Uso para Provisionamento de Parâmetros WebRTC (*GerarAcessoSessaoUseCase.ts*)

O *payload* retornado fornece a lista de servidores *ICE*, composta por um servidor *STUN* público — responsável por descobrir o endereço de IP público do dispositivo do usuário — e um servidor *TURN* parametrizado por variáveis de ambiente, encarregado de atuar como intermediário (*relay*) de mídia caso os *firewalls* dos usuários impeçam a conexão direta *P2P*.

Nesta implementação, tanto o serviço *STUN* quanto o serviço *TURN* foram disponibilizados por meio do servidor *Coturn*, executado como parte da infraestrutura da aplicação. A utilização de um servidor *TURN* próprio proporcionou maior controle sobre o ambiente de comunicação e eliminou a dependência exclusiva de servidores públicos, possibilitando maior confiabilidade durante o estabelecimento das sessões de videoconferência.

Figura 28 – Arquitetura de estabelecimento da conexão WebRTC utilizando servidor Coturn.



Fonte: Próprio autor

4.13.2.2 Orquestração do Fluxo de Sinalização via Gateway

Com os parâmetros de rede provisionados, a negociação técnica entre os navegadores inicia-se através da classe *SessaoSignalingGateway*. Esse módulo opera dentro do namespace isolado */sessao-signaling* configurado no *SocketManager*, atuando como o intermediador dos metadados de rede sem que o servidor precise processar ou decodificar os fluxos de mídia, conforme descrito no Código 4.6.

```
export class SessaoSignalingGateway {
    constructor(private io: Namespace, private sessaoRepository: ISessaoRepository) {
        this.setupEventListeners();
    }

    private setupEventListeners() {
        this.io.on('connection', (socket: Socket) => {
            socket.on('join-session', (sessaoId: string) => {
                socket.join(sessaoId);
                socket.to(sessaoId).emit('user-joined', { userId: socket.id });
            });

            socket.on('offer', async (data: { offer: any, sessaoId: string }) => {
                const resultado = await this.sessaoRepository.buscarPorId(data.sessaoId);
                if(resultado && !resultado.data_inicio_real){
                    resultado.iniciarSessao();
                    await this.sessaoRepository.atualizar(resultado);
                }
                 socket.to(data.sessaoId).emit('offer', { from: socket.id, offer: data.offer });
            });

            socket.on('answer', (data: { answer: any, sessaoId: string }) => {
                socket.to(data.sessaoId).emit('answer', { from: socket.id, answer: data.answer });
            });

            socket.on('ice-candidate', (data: { candidate: any, sessaoId: string }) => {
                socket.to(data.sessaoId).emit('ice-candidate', { from: socket.id, candidate: data.candidate });
            });
        });
    }
```

```
31 }
```

Código 4.6 – Gateway de Sinalização WebRTC (*SessaoSignalingGateway.ts*)

O funcionamento do ciclo de sinalização do *Gateway* obedece a uma lógica sequencial e reativa:

1. **join-session**: O usuário conecta-se ao canal lógico da sessão. O servidor aloca-o em uma sala restrita do *socket* e emite o evento *user-joined* para alertar o cliente remoto de que o par está pronto para a negociação.

2. **offer**: O par iniciador envia sua oferta SDP (especificando codecs e capacidades de mídia). No momento exato em que a oferta transita pelo servidor, o sistema captura o evento para executar uma regra de negócio de domínio: invoca o método *iniciarSessao()* na entidade e atualiza a propriedade *data_inicio_real* no banco de dados, marcando de forma automatizada o início cronológico oficial da consulta médica.

3. **answer** e **ice-candidate**: O *Gateway* encaminha de forma assíncrona a resposta SDP do par receptor e os candidatos *ICE* encontrados por ambos os dispositivos, garantindo que os navegadores fechem o enlace técnico e iniciem o fluxo de vídeo *P2P*.

4.13.2.3 Implementação do Cliente WebRTC e Consumo de Telemetria com IA

No ecossistema *frontend*, a gerência dos fluxos de mídia e sinalização síncrona foi modularizada através do padrão *Custom Hook* com o componente *useVideoSessionEngine.ts*, exposto globalmente por meio do *VideoSessionProvider*. Esta arquitetura separa as preocupações de interface de usuário das rotinas de baixo nível da API de *PeerConnection* do navegador.

O ciclo de estabelecimento da videoconferência compreende três fases sequenciais operadas no lado do cliente:

1. **Estágio de Pré-configuração e Sala de Espera**: Implementado pelo componente *SalaEspera.tsx*, o sistema intercepta o usuário antes do ingresso na conferência ativa. Esta interface consome o *GerarAcessoSessaoUseCase* do *backend* para obter os servidores *ICE* e invoca de forma assíncrona o método *navigator.mediaDevices.getUserMedia()*. Este procedimento realiza a validação prévia do hardware (câmera e microfone) e injeta o *MediaStream* nativo como um *preview* local instanciado por um *callback ref* reativo, mitigando falhas de permissão em tempo de execução.

2. **Negociação de Mídia Assimétrica (SDP Peer-to-Peer)**: Ao acionar a consulta, o gancho *useVideoSessionEngine* inicializa a instância do *RTCPeerConnection* aplicando as regras assimétricas baseadas nos papéis lógicos (*roles*) dos utilizadores do sistema, conforme detalhado no Código 4.7.

```
// Inicialização assimétrica baseada no papel de domínio
const startCall = useCallback(async () => {
    let currentStream = localStream;
    if (!currentStream) {
        currentStream = await navigator.mediaDevices.getUserMedia({
   video: true, audio: true });
        setLocalStream(currentStream);
    }

    const peer = await initWebRTC(currentStream);

    if (isProfissional) {
        const offer = await peer.createOffer();
        await peer.setLocalDescription(offer);
        socketSignaling.current?.emit('offer', { offer, sessaoId });
    } else {
        startAnalysisLoop(currentStream);
    }
    setIsCallStarted(true);
}, [localStream, initWebRTC, isProfissional, sessaoId, startAnalysisLoop
   ]);
```

Código 4.7 – Fluxo de Negociação de Chamada e Loop de Análise (*useVideoSessionEngine.ts*)

Como evidenciado no fragmento de código acima, o usuário *profissional* atua como o elemento provocador da conexão, gerando a oferta técnica (*Session Description Protocol - SDP*), configurando sua descrição local e emitindo o evento *'offer'* via WebSocket. O navegador do *paciente*, por sua vez, opera em modo de escuta passiva através do evento *socketSignaling.on('offer')*, respondendo automaticamente com uma descrição remota, gerando uma *answer* e estabelecendo o enlace de mídia sem demandar interações manuais do paciente.

4.13.2.4 Loop de Extração de Frames e Feedback de Telemetria

Para viabilizar a análise computacional de emoções em tempo real sem sobrecarregar o tráfego da rede com fluxos contínuos de vídeo pesado, implementou-se uma rotina de amostragem discreta de imagens (*startAnalysisLoop*), restrita exclusivamente ao dispositivo do paciente para preservar sua privacidade de escopo.

A rotina instancia um elemento *HTMLVideoElement* em memória e anexa a ele o fluxo de mídia local ativo. Sob um intervalo cronometrado de **2000 milissegundos** (2 segundos), o sistema realiza o *downsampling* geométrico da imagem capturada utilizando um elemento *Canvas*, reduzindo a resolução para uma matriz estável de 160x120 pixels. O resultado é convertido de forma assíncrona em um objeto binário de tamanho reduzido (*Blob JPEG*) com compressão de 50% e despachado via *WebSocket* por meio do evento *'video-frame'* no namespace */sessao-analysis*.

O *frontend* consome de forma assíncrona os retornos gerados pelo pipeline de IA. O hook expõe o estado reativo *currentEmotion*, que renderiza dinamicamente o componente *EmotionBadge* sobre o vídeo remoto do paciente, além de apresentar uma barra de progresso com a confiança preditiva gerada pelo modelo.

O sistema possui uma rotina de monitoramento de instabilidade (*analysis-error*) caso o processamento sofra atrasos ou falhas, a interface exibe um alerta toast de *"IA Instável"* contendo uma rotina de auto-descarte (*auto-dismiss*) de 4 segundos controlada por temporizadores lógicos, prevenindo a poluição visual do painel clínico durante a consulta.

4.13.2.5 Pipeline de Processamento de Telemetria e Resiliência no Backend

A recepção, validação e roteamento das matrizes de vídeo destinadas à inferência computacional são gerenciadas no ecossistema de *backend* pela classe *SessaoAnalysisGateway*. Este componente expõe o *namespace* isolado */sessao-analysis* e atua como uma camada intermediária de baixa latência, efetuando a ponte entre o cliente WebSockets e o microsserviço de inteligência artificial através de chamadas de procedimento remoto gRPC (*gRPC Remote Procedure Calls*).

Para suportar a carga de requisições de alta frequência de múltiplos clientes simultâneos sem degradar a estabilidade da aplicação principal, a infraestrutura adota três padrões arquiteturais de resiliência e concorrência:

- **Bloqueio de Concorrência por Estado (Debounce de Infraestrutura)**: Para evitar o empilhamento de requisições de um mesmo cliente caso o pipeline de IA sofra gargalos, o *Gateway* mantém um conjunto de estados em memória denominado *processingSockets*. Ao receber o evento *'video-frame'*, o sistema verifica se o identificador único do *socket* já está em processamento; em caso positivo, o *frame* atual é descartado imediatamente de forma silenciosa, blindando o servidor contra condições de corrida e exaustão de memória.
- **Padrão de Corrida Competitiva (Promise Race)**: A comunicação com o microsserviço de IA é envelopada por um mecanismo de *Timeout* estrito de **5000 milissegundos**. Caso a rede ou o modelo YOLO/gRPC apresentem degradação

de performance, uma promessa concorrente rejeita a operação. Isso impede que *threads* do NodeJS fiquem retidas indefinidamente aguardando respostas de fluxos multimídia.

- **Desacoplamento por Disparo de Eventos de Domínio**: Uma vez processada a inferência com sucesso, o *Gateway* adota uma arquitetura orientada a eventos (*Event-Driven*), notificando o ecossistema através de um *EventDispatcher*, conforme exibido a seguir:

```
try {
    this.processingSockets.add(socket.id);
// Execucao concorrente com limite de tempo (gRPC vs Timeout)
const analise = await Promise.race([
    this.grpcClient.analisarFrame({
        sessaoId: data.sessaoId,
        imagem: data.image
    }),
    new Promise((_, reject) =>
        setTimeout(() => reject(new Error('TIMEOUT_GRPC')), this.GRPC_
    TIMEOUT)
    )
]) as any;

// 1. Emissao em tempo real para o painel do profissional
this.nsp.to(data.sessaoId).emit('analysis-feedback', analise);

// 2. Persistencia assincrona via Event Dispatcher de Dominio
this.eventDispatcher.notify(
    new FrameAnalisadoIA({
        sessaoId: data.sessaoId,
        emocaoPredominante: analise.emocao,
        confianca: analise.confianca,
        timestamp: Date.now()
    })
);

} catch (error: any) {
    if (error.message === 'TIMEOUT_GRPC') {
        console.warn(`[IA-Gateway] IA demorou demais na sessão ${data.
    sessaoId}. Frame descartado.`);
        // Avisar o front que a IA está lenta
        socket.emit('analysis-error', { message: 'IA Latency High' });
    } else {
        console.error(`[IA-Gateway] Erro gRPC na sessão ${data.sessaoId
    }:`, error);
        }
```

```
35 } finally {
36     this.processingSockets.delete(socket.id);
37 }
```

Código 4.8 – Gerenciamento de Pipeline gRPC e Eventos Clínicos (*SessaoAnalysisGateway.ts*)

Como pode ser observado no Código 4.8, o *Gateway* realiza uma dupla distribuição dos dados resultantes de forma assíncrona. Primeiramente, despacha via *websocket* direcionado o evento *'analysis-feedback'* contendo a emoção e a acurácia para atualizar a interface do profissional em tempo real.

Em paralelo, o método *eventDispatcher.notify()* instancia o evento de domínio *FrameAnalisadoIA*. Este padrão garante um acoplamento fraco: o *gateway* de rede não possui conhecimento sobre regras de persistência ou geração de relatórios; ele apenas sinaliza o sistema, permitindo que manipuladores secundários (*Handlers*) capturem a telemetria em segundo plano e salvem os históricos métricos no *Redis* para posterior consolidação do prontuário médico.

Por fim, o bloco de fechamento *finally* assegura a resiliência operacional da rotina através do comando *this.processingSockets.delete(socket.id)*. Independentemente do desfecho da execução — seja uma inferência bem-sucedida, uma falha crítica de rede ou o estouro do tempo limite de processamento (*TIMEOUT_GRPC*) —, o canal de escuta do cliente é invariavelmente liberado, restabelecendo a prontidão do *Gateway* para receber e avaliar a subsequente amostragem de vídeo enviada pelo navegador.

4.14 Desenvolvimento do Microsserviço de Inteligência Artificial Textual

Com o objetivo de incorporar recursos de inteligência artificial à plataforma, foi desenvolvido um microsserviço independente responsável pelo processamento de informações textuais relacionadas aos registros dos pacientes. Sua implementação permite separar essas operações do *backend principal*, mantendo as funcionalidades de inteligência artificial em um componente próprio.

A arquitetura foi desenvolvida utilizando o *framework Flask*, em ambiente *Python 3*, integrado a uma instância do *Ollama* responsável pela execução do modelo *Llama 3*. A comunicação entre o *backend* principal e o microsserviço ocorre por meio de requisições *HTTP*, utilizando endpoints específicos para cada funcionalidade.

O microsserviço contempla duas operações principais. A primeira realiza a análise textual dos relatos publicados pelos pacientes, atribuindo uma classificação de gravidade que fica disponível como informação auxiliar para os profissionais. A segunda utiliza os

registros diários de humor para gerar cards personalizados, apresentados no dashboard do paciente. Embora ambas utilizem o mesmo modelo, cada operação possui instruções e parâmetros próprios, conforme apresentado na seção seguinte.

4.14.0.1 Análise de Engenharia de Prompts e Parametria de Hiperparâmetros

A interação com o modelo é realizada por meio de prompts elaborados especificamente para cada funcionalidade. Esses prompts definem o contexto da solicitação, as informações que devem ser consideradas durante o processamento e o formato esperado para a resposta. Além disso, foram estabelecidos parâmetros distintos para controlar a geração dos conteúdos, considerando as características de cada operação.

A primeira operação é disponibilizada pelo *endpoint /classificar-gravidade*, responsável por receber o texto de um relato e encaminhá-lo ao modelo para análise. O prompt utilizado estabelece três critérios: identificação do sentimento predominante, verificação de possíveis sinais de risco e avaliação da urgência apresentada no texto.

A partir desses critérios, são definidas três classificações possíveis:

- **Leve:** contempla situações de estresse cotidiano, sentimentos positivos ou desabafos sem sinais evidentes de ruptura emocional.
- **Mediano:** abrange relatos com tristeza profunda, ansiedade nítida ou conflitos interpessoais relevantes, mas sem indicação de risco à vida.
- **Grave:** contempla relatos que mencionem ideação suicida, automutilação, crises de pânico agudas ou desesperança extrema.

O prompt efetivamente utilizado na implementação é apresentado no Código 4.9

```
prompt = f"""Você é um assistente de triagem psicológica altamente treinado.
Analise o relato do usuário abaixo para identificar o nível de suporte necessário.

RELATO: "{relato}"

CRITÉRIOS DE ANÁLISE:
1. Sentimento: Qual a emoção predominante?
2. Risco: Existe menção a autolesão, ideação suicida ou violência? (Isso torna o caso GRAVE imediatamente)
3. Urgência: O usuário parece estar em crise aguda ou é um desabafo de cansaço acumulado?

CLASSIFICAÇÕES POSSÍVEIS:
```

```
- LEVE: Estresse cotidiano, sentimentos positivos ou desabafos sem
    sinais de ruptura emocional.
- MEDIANO: Tristeza profunda, ansiedade nítida, conflitos interpessoais
    sérios, mas sem risco de vida.
- GRAVE: Ideação suicida, automutilação, crises de pânico agudas ou
    desesperança extrema ("não aguento mais" sem contexto).

Responda APENAS em formato JSON:
{
    "analise_curta": "breve justificativa da sua decisão",
    "gravidade": "leve/mediano/grave",
    "pontuacao_risco": 1-10
}
"""
```

Código 4.9 – *Prompt da análise dos relatos*

A título de exemplo, considere o relato de entrada: "Estou em prantos. Todo dia me sinto mais afastado de mim.". Após o processamento das instruções, o modelo retorna a estrutura exibida no Código 4.10.

```
{
    "analise_curta": "O usuário relata choro frequente e sensação
    acentuada de despersonalização ou perda de identidade, indicando um
    quadro de sofrimento emocional significativo, mas sem menção explí
    cita a ideação suicida ou autolesão.",
    "gravidade": "mediano",
    "pontuacao_risco": 5
}
```

Código 4.10 – *Exemplo real do retorno da classificação do relato*

Para essa operação, foi definida a temperatura em 0,0. Esse parâmetro reduz a aleatoriedade na seleção dos *tokens* durante a geração da resposta, favorecendo uma menor variação entre respostas produzidas em condições semelhantes. Entretanto, esse ajuste não garante que a classificação esteja correta, não elimina possíveis alucinações e não impede interpretações equivocadas do relato.

Após receber a resposta do modelo, a aplicação realiza o tratamento do conteúdo retornado, procurando identificar as palavras correspondentes às categorias de gravidade. Caso encontre o termo grave, essa classificação é priorizada; caso contrário, verifica a presença de mediano. Se nenhum dos dois termos for encontrado, o código retorna leve. Quando ocorre uma exceção durante a comunicação ou o processamento, o retorno é indeterminado.

É importante observar que, embora o prompt solicite uma justificativa e uma pontuação de risco, a implementação atual utiliza somente a categoria de gravidade identificada

na resposta. A justificativa e a pontuação não são extraídas nem disponibilizadas pelo endpoint. Além disso, a aplicação não exige que a resposta do modelo seja um *JSON* válido, pois a interpretação é realizada por meio da busca textual pelos termos das classificações.

A segunda operação é disponibilizada pelo *endpoint /cards-dashboard*, que recebe três informações provenientes do registro diário de humor do paciente: o estado emocional informado, sua intensidade em uma escala de 1 a 5 e o período do dia correspondente.

O prompt utiliza essas informações para contextualizar a solicitação e orientar o modelo na elaboração de três cards destinados ao *dashboard*:

- **Dica do Dia:** deve apresentar uma sugestão prática relacionada ao momento informado.
- **Estatística:** deve apresentar uma informação relacionada à temática de saúde mental.
- **Motivação:** defve oferecer uma mensagem de acolhimento.

O trecho principal do prompt utilizado é apresentado no Código 4.11

```
contexto_usuario = f"O usuário relatou estar se sentindo '{mood}' (intensidade {intensidade}/5) durante o período da '{periodo}'."

prompt = f"""
{contexto_usuario}

Com base nisso, gere 3 cards para um dashboard de saúde mental em português.
A 'Dica do Dia' deve ser prática para o momento, a 'Estatística' deve ser algo interessante, e a 'Motivação' deve ser um acolhimento.

Retorne APENAS um objeto JSON puro, sem textos extras, seguindo rigorosamente este modelo:
{
  "periodo": "{periodo}",
  "cards": [
    {"title": "Dica do Dia", "content": "..."},
    {"title": "Estatística", "content": "..."},
    {"title": "Motivação", "content": "..."}
  ]
}
"""
```

Código 4.11 – *Prompt da geração dos cards*

Como exemplo, considerando um registro de ansiedade, com intensidade 4 e período da noite, as informações são incorporadas ao contexto do prompt. A resposta obtida é mostrado no Código 4.12:

```
{
    "periodo": "noite",
    "cards": [
        {
            "title": "Dica do Dia",
            "content": "Reserve alguns minutos para realizar uma
    atividade tranquila antes de dormir."
        },
        {
            "title": "Estatística",
            "content": "A qualidade do sono está relacionada ao bem-
    estar emocional."
        },
        {
            "title": "Motivação",
            "content": "Respeite seu próprio ritmo. Pequenos passos tamb
    ém representam progresso."
        }
    ]
}
```

Código 4.12 – *Exemplo real do retorno do registro de humor*

Diferentemente da classificação dos relatos, essa operação utiliza a temperatura de 0,8, permitindo que conteúdo gerado possua variação léxica e não se torne repetitivo. Este valor tolera uma maior entropia na amostragem de *tokens*, gerando acolhimentos personalizads e dinâmicos para o paciente.

Apesar da definição de critérios nos prompts e da utilização de parâmetros específicos para cada operação, a classificação dos relatos não possui uma validação independente que confirme a precisão dos resultados. A temperatura de 0,0 reduz a variação da amostragem, mas não constitui um mecanismo de prevenção de alucinações ou de identificação de classificações incorretas.

A implementação também não dispõe de um procedimento específico para revisar possíveis falsos negativos, situações em que um relato com sinais relevantes de risco poderia receber uma classificação inferior à necessária. Essa limitação merece atenção porque, no tratamento atual da resposta, caso o modelo retorne um conteúdo que não contenha os termos grave ou mediano, a aplicação assume a classificação leve. Portanto, uma resposta inesperada ou fora do formato solicitado pode resultar em uma classificação inadequada.

Em relação à segurança dos conteúdos gerados, não foi realizado um procedimento sistemático para avaliar a adequação das respostas, verificar a ocorrência de informações

incorretas ou analisar possíveis conteúdos prejudiciais. Da mesma forma, as demonstrações da plataforma realizadas com duas profissionais da psicologia não contemplaram a avaliação dos prompts ou dos resultados produzidos pelo modelo. As contribuições dessas profissionais foram restritas à interface e aos fluxos de utilização do sistema, não constituindo validação clínica das funcionalidades de inteligência artificial.

Dessa forma, a classificação dos relatos e os conteúdos personalizados devem ser compreendidos como recursos auxiliares da plataforma. A classificação não representa diagnóstico, avaliação clínica ou garantia de identificação de situações de risco, cabendo ao profissional realizar sua própria análise do relato.

4.14.0.2 Tratamento de Exceções e Estratégia de Fallback Estruturado

Em cenários de degradação de hardware ou indisponibilidade da sub-rede local onde o servidor Ollama está hospedado, o microsserviço emprega uma tática de tolerância a falhas. No método de gerar *cards*, a captura genérica de exceções no bloco *catch* intercepta a quebra da requisição HTTP (controlada por um *timeout* preventivo de 40 segundos).

Em vez de propagar o erro e interromper a renderização na interface do usuário, a rotina intercepta a falha e retorna instantaneamente um objeto JSON estático, porém perfeitamente compatível com a assinatura esperada pelo *frontend*. Essa abordagem assegura que o painel do usuário permaneça visualmente íntegro e funcional, mantendo uma experiência de uso contínua mesmo sob condições de falha severa na infraestrutura de Inteligência Artificial.

O trecho de código a seguir (Código 4.13) evidencia a implementação do mecanismo de *Fallback* estruturado:

```
    try:
        response = requests.post(OLLAMA_API_URL, json=payload, timeout
    =40)
        response.raise_for_status()

        resposta_ollama = response.json()
        logger.info(f"Resposta bruta do Ollama: {resposta_ollama}")  #
    Debug

        resposta_texto = response.json().get('response', '').strip()
        return json.loads(resposta_texto)

    except Exception as e:
        logger.error(f"Erro ao gerar cards personalizados: {e}")

        return {
            "periodo": periodo,
            "cards": [
```

```
17                 {"title": "Dica do Dia", "content": "Tire 5 minutos para
    uma respiração consciente agora."},
18                 {"title": "Lembrete", "content": "Sua saúde mental é uma
    prioridade, não um luxo."},
19                 {"title": "Apoio", "content": "Estamos aqui para
   acompanhar sua jornada, passo a passo."}
20             ]
21         }
```

Código 4.13 – Trecho do código referente ao *Fallback*

4.15 Desenvolvimento do Microsserviço de Visão Computacional

Com o intuito de conferir autonomia operacional e alta performance técnica ao ecossistema, a camada de análise comportamental foi construída integralmente de forma nativa e interna por meio de um microsserviço dedicado de Visão Computacional. Diferente das abordagens baseadas em APIs proprietárias de terceiros, o pipeline técnico compreende o treinamento customizado de uma rede neural convolucional profunda e o desenvolvimento de um servidor de inferência de baixíssima latência que se comunica com o ecossistema *backend* através do protocolo gRPC (*gRPC Remote Procedure Calls*).

O ciclo de desenvolvimento do componente seguiu três etapas de engenharia computacional: o treinamento iterativo com o *dataset* de expressões faciais, a compilação do contrato de interface IDL (*Interface Definition Language*) via buffers de protocolo e a consolidação do script de escuta do servidor.

4.15.0.1 Pipeline de Treinamento e Otimização do Modelo YOLOv8

Para o treinamento do modelo, foi utilizado um conjunto de dados baseado no *AffectNet*, convertido para o formato *YOLO*. O conjunto contempla oito categorias de expressões faciais: raiva, desprezo, nojo, medo, felicidade, neutro, tristeza e surpresa.

O treinamento partiu dos pesos pré-treinados yolov8n.pt, utilizando a técnica de *transfer learning*. Dessa maneira, o processo iniciou a partir de parâmetros previamente aprendidos pela arquitetura, que foram ajustados durante o treinamento com as imagens do conjunto de dados utilizado no projeto.

O script que rege a execução do treinamento da rede convolucional é apresentado no Código a seguir:

```
1 from ultralytics import YOLO #Versão 8.1.0
2
3
4 def train_psico_yolo_224px():
5     # Carregar modelo Nano pre-treinado para transfer learning
```

```
    model = YOLO("yolov8n.pt")

    model.train(
        data="data/YOLO_format/data.yaml",
        epochs=100, # Número total de vezes que o modelo vai passar por todo o conjunto de dados durante o treino.
        imgsz=224, # Redimensiona todas as imagens para o tamanho de 224x224 pixels antes de enviar para a rede.
        batch=64,  # Número de imagens processadas por vez antes de atualizar os pesos do modelo.
        device=0, # Rodar na GPU
        project="psico_ai_02",
        name="affectnet_224px_otimizado_02",
        save=True,
        # --- Ajustes para combater o Falso Positivo (Background) ---
        augment=True, #Ativa o aumento de dados automático (rotação, espelhamento, etc.) evita overfitting
        close_mosaic=20, # Desliga o mosaico no final para limpar os ruídos das bordas
        optimizer="AdamW", # Define o algoritmo de otimização dos pesos
        lr0=0.001, # Taxa de aprendizado inicial. Controla o tamanho do "passo" que o otimizador dá ao corrigir os erros do modelo.
        seed=0,             # Garante reprodutibilidade  (ordem dos dados e inicialização)
        deterministic=True, # Força algoritmos determinísticos na GPU

    )

if __name__ == "__main__":
    train_psico_yolo_224px()
```

Código 4.14 – Configuração e Hiperparâmetros de Treinamento da Rede Facial (*train.py*)

O processo de convergência do modelo aplicou técnicas específicas de regularização para problemas de classificação fina. Embora o *dataset* de origem possua dimensões de 96 x 96 pixels, a variável *imgsz* foi configurada em 224 pixels para compatibilizar a escala escalar de extração de características internas da rede YOLOv8. Essa técnica ajuda a reduzir a ocorrência de falsos positivos induzidos pelo plano de fundo (*background noise*).

Destaca-se o uso do argumento *close_mosaic=20*. Esta técnica determina que o aumento de dados por mosaico (fusão de quatro imagens aleatórias) seja completamente desativado durante as últimas 20 épocas do treinamento. Esse procedimento purifica os gradientes finais, limpando artefatos ruidosos localizados nas bordas das caixas delimitadoras (*bounding boxes*) e forçando o modelo a convergir com base estritamente na geometria centrofacial (olhos, boca e sobrancelha). O otimizador *AdamW* foi introduzido com uma taxa de aprendizado inicial ($\eta = 0.001$) aplicando decaimento de peso adaptativo para

prevenir problemas de sobreajuste (*overfitting*).

4.15.0.2 Arquitetura do Contrato gRPC e Camada de Serialização

Para viabilizar o envio a cada dois segundos de *frames* binários capturados pela webcam do paciente até o modelo de inteligência artificial, o protocolo HTTP tradicional foi preterido em favor do gRPC sobre a camada de transporte HTTP/2. A definição do contrato de comunicação baseia-se em um arquivo de especificação técnica (*analisador.proto*), o qual dita as assinaturas estruturais de chamadas de procedimento remoto.

O arquivo de interface é apresentado a seguir, detalhando os métodos de inferência e os tipos de mensagens serializadas:

```
syntax = "proto3";

package analisador;

service AnalisadorService {
  rpc DetectEmotion (FrameRequest) returns (FrameResponse);
}

message FrameRequest {
  string sessaoId = 1;
  bytes imagem = 2;
}

message FrameResponse {
  string emocao = 1;
  float confianca = 2;
}
```

Código 4.15 – Definição de Interface de Comunicação gRPC (*analisador.proto*)

Por meio do compilador de *Protocol Buffers* (*protoc*), o arquivo de interface gera os módulos de acoplamento *analisador_pb2.py* (contendo as estruturas de dados e métodos para serialização binária) e *analisador_pb2_grpc.py* (contendo as classes abstratas de infraestrutura servidora e pelos procuradores de cliente, também denominados *stubs* de comunicação, responsáveis por intermediar as chamadas de rede). Esse processo garante que a imagem transmitida transite como uma cadeia pura de bytes brutos (*raw bytes*), eliminando o *overhead* de codificação em formatos textuais pesados como o Base64.

4.15.0.3 Construção do Servidor de Inicialização e Inferência

A execução em tempo real do microsserviço é centralizada no componente *grpc_server.py*. Ele implementa a classe abstrata gerada e anexa o checkpoint com os pesos neurais (*best.pt*)

obtidos ao término do processo de treinamento, operando em modo contínuo de escuta na porta padrão *50051*. O código-fonte estrutural que rege o servidor é detalhado a seguir:

```
    class AnalisadorService(analisador_pb2_grpc.
AnalisadorServiceServicer):
    def __init__(self):
        # Carrega os pesos otimizados resultantes do treinamento interno
        self.model = YOLO("./runs/detect/psico_ai/affectnet_96px5/
weights/best.pt")

    def DetectEmotion(self, request, context):
        try:
            # Converte bytes brutos do Protocol Buffer para matriz
OpenCV
            nparr = np.frombuffer(request.imagem, np.uint8)
            frame = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

            if frame is None:
                return analisador_pb2.FrameResponse(emocao="Sem Frame",
confianca=0.0)

            # Execucao da predicao com threshold de confianca estrito
            results = self.model.predict(source=frame, imgsz=224, conf
=0.54, verbose=False)

            if len(results) > 0 and len(results[0].boxes) > 0:
                box = results[0].boxes[0]
                label = self.model.names[int(box.cls)]
                conf = float(box.conf)
                return analisador_pb2.FrameResponse(emocao=label,
confianca=conf)

            return analisador_pb2.FrameResponse(emocao="Neutro",
confianca=0.0)
        except Exception as e:
            print(f"Erro na inferencia gRPC: {e}")
            return analisador_pb2.FrameResponse(emocao="Erro", confianca
=0.0)

def serve():
    # Alocacao de ThreadPoolExecutor para gerenciar requisicoes
concorrentes
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    analisador_pb2_grpc.add_AnalisadorServiceServicer_to_server(
AnalisadorService(), server)
    server.add_insecure_port('[::]:50051')
    server.start()
    print("Servidor gRPC ativo na porta 50051")
```

```
36      server.wait_for_termination()
37
38  if __name__ == "__main__":
39      serve()
```

Código 4.16 – Servidor de Inferência e Decodificação OpenCV (*grpc_server.py*)

O fluxo interno do método *DetectEmotion* opera como um pipeline linear de transformação de dados:

1. **Decodificação Volátil**: O *array* de bytes brutos serializado pelo *Protocol Buffer* é mapeado em memória volátil através da função *np.frombuffer* e instantaneamente convertido em uma matriz de cores tridimensional BGR pelo *cv2.imdecode*, evitando gravações em disco rígido.

2. **Inferência e Thresholding**: O modelo YOLO executa o método *predict* parametrizado com uma restrição de confiança de **0.54** (*conf=0.54*). Esse limiar estrito atua como um filtro purificador: predições abaixo de 54% de probabilidade matemática são sumariamente descartadas, categorizando o estado do paciente como "Neutro" e eliminando variações térmicas micro-expressivas irrelevantes.

3. **Despacho de Metadados**: Uma vez identificada a *bounding box* de maior relevância, o identificador de classe (*box.cls*) é convertido no rótulo de texto correspondente (*label*) e retornado de forma imediata dentro da estrutura tipada *FrameResponse*.

Para sustentar a concorrência assíncrona exigida, o método de inicialização *serve()* aloca um gerenciador de concorrência *ThreadPoolExecutor* fixado em até **10 worker threads**. Essa configuração garante que múltiplas conexões de fluxos de sinalização simultâneos possam ser avaliadas concorrentemente sem que uma inferência tardia bloqueie o recebimento de novos pacotes na porta *50051*.

5 Resultados e Análise

Este capítulo apresenta os resultados obtidos durante o desenvolvimento e validação da plataforma proposta, bem como uma análise das funcionalidades implementadas e sua contribuição para os objetivos definidos neste trabalho.

A avaliação foi realizada considerando aspectos funcionais, arquiteturais e tecnológicos da aplicação, verificando o comportamento dos módulos desenvolvidos e a integração entre os diferentes componentes que compõem o sistema.

Os resultados obtidos demonstram a viabilidade da utilização de uma arquitetura distribuída integrada a recursos de inteligência artificial para apoio ao acompanhamento emocional e psicológico dos usuários da plataforma.

5.1 Validação dos Requisitos Funcionais

Nesta seção são apresentados os resultados da verificação dos requisitos funcionais definidos para a plataforma TherapistFriend. A análise considera as funcionalidades implementadas e os procedimentos de teste realizados durante o desenvolvimento, contemplando a execução manual dos principais fluxos de utilização do sistema.

Os testes manuais foram realizados por meio da utilização direta da plataforma, simulando as interações entre pacientes, profissionais e administradores. Foram verificados os processos de cadastro, autenticação, publicação de relatos, comunicação privada, agendamento e realização de sessões on-line, além das funcionalidades auxiliares disponibilizadas aos usuários.

A Tabela 4 apresenta um resumo dos principais requisitos funcionais desenvolvidos.

Tabela 4 – Verificação dos requisitos funcionais por meio de testes manuais.

Requisito funcional	**Cenário de verificação**	**Resultado observado**
Cadastro de usuários	Realização do cadastro de pacientes e profissionais, verificando o preenchimento dos dados e a criação das contas.	Funcionamento observado
Validação de e-mail	Solicitação e confirmação do código de verificação enviado ao endereço de e-mail informado.	Funcionamento observado
Validação de telefone	Solicitação e confirmação do código de verificação enviado ao número de telefone informado.	Funcionamento observado
Aprovação de profissionais	Cadastro de um profissional e verificação do processo de aprovação realizado pelo administrador.	Funcionamento observado
Relatos	Publicação e visualização de relatos, considerando as diferentes modalidades de visibilidade e identificação.	Funcionamento observado
Solicitação de conversa	Solicitação de contato por um profissional a partir de um relato e apresentação da solicitação ao paciente.	Funcionamento observado
Chat privado	Aceitação de uma solicitação de conversa e realização da troca de mensagens entre paciente e profissional.	Funcionamento observado
Registro de humor	Registro diário do estado emocional pelo paciente.	Funcionamento observado
Geração de cards personalizados	Verificação da disponibilização de conteúdos personalizados após o registro de humor.	Funcionamento observado
Agenda	Configuração de disponibilidades pelo profissional e solicitação e gerenciamento de agendamentos pelos usuários.	Funcionamento observado
Sessões on-line com análise de expressões faciais	Entrada dos participantes na sessão, comunicação por áudio e vídeo e apresentação dos resultados da análise de expressões faciais.	Funcionamento observado
Relatórios automáticos	Encerramento de uma sessão e verificação da geração e disponibilização do relatório ao profissional.	Funcionamento observado
Notificações em tempo real	Execução de interações que geram notificações e verificação de sua apresentação na plataforma.	Funcionamento observado

Os testes realizados permitiram observar o funcionamento dos principais fluxos da aplicação e a integração entre os módulos envolvidos. Os resultados apresentados indicam que os comportamentos esperados para os cenários executados foram observados durante a utilização da plataforma.

Além dos testes manuais, foram desenvolvidos testes automatizados utilizando o framework Jest, com foco principalmente nos casos de uso do backend e em outros componentes da aplicação. Esses testes complementaram as verificações manuais, permitindo avaliar o comportamento das regras de negócio de maneira isolada.

Ressalta-se que os cenários apresentados correspondem às verificações realizadas durante o desenvolvimento, não tendo sido estabelecidos como critérios de aceite formais previamente à implementação. Dessa forma, os resultados representam uma verificação funcional da solução desenvolvida, não uma avaliação exaustiva de todos os comportamentos possíveis do sistema.

5.2 Interfaces Desenvolvidas

Esta seção apresenta as principais interfaces desenvolvidas para a plataforma, demonstrando as variações conforme o perfil do usuário e a adaptabilidade para diferentes tamanhos de tela. Embora todo o sistema tenha sido projetado com foco na responsividade, apenas algumas telas serão exibidas em suas versões voltadas para dispositivos móveis, com o objetivo de evitar a redundância de informações e a poluição visual.

5.2.1 Cadastro

A interface foi projetada com foco na simplicidade do processo de criação de contas, disponibilizando apenas os campos essenciais para identificação e autenticação do usuário. Os campos de entrada contemplam nome, endereço de e-mail, número de telefone e senha, permitindo o registro inicial tanto de pacientes quanto de profissionais.

A Figura 29 apresenta a tela de cadastro desenvolvida para a plataforma.

Figura 29 – Tela de cadastro.



Fonte: Próprio autor

A Figura 30 apresenta o componente utilizado para validação dos dados informados durante o cadastro.

Figura 30 – Modais códigos.



Fonte: Próprio autor

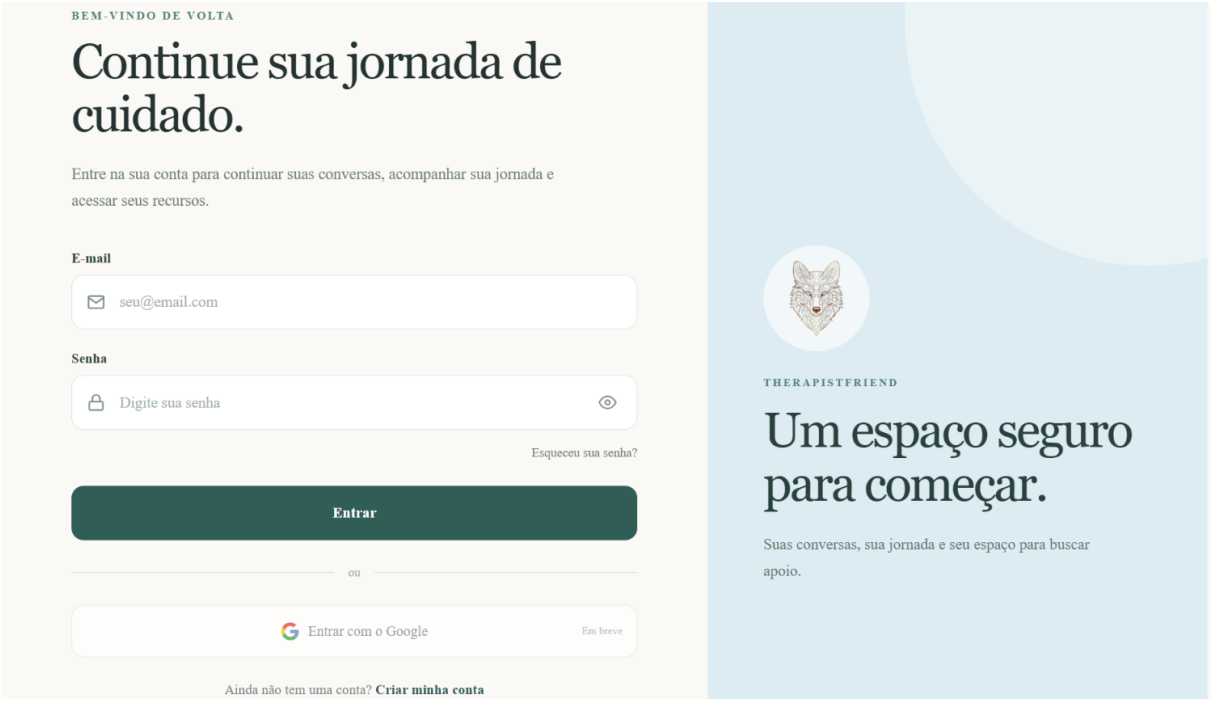
O modal foi desenvolvido como um componente reutilizável, sendo utilizado tanto para validação de e-mail quanto de telefone celular. Sua implementação permitiu padronização visual e redução da duplicação de código no frontend da aplicação. O componente

também apresenta controle temporal para validade dos códigos enviados, contribuindo para maior segurança durante o processo de verificação dos usuários.

5.2.2 Login

A autenticação dos usuários é realizada por meio da tela de login apresentada na Figura 31. Essa interface foi desenvolvida para fornecer acesso aos diferentes perfis existentes na plataforma, incluindo pacientes, profissionais e administradores.

Figura 31 – Tela de login.



Fonte: Próprio autor

A interface foi projetada seguindo a identidade visual adotada na aplicação, priorizando simplicidade e facilidade de utilização. Para autenticação, o usuário deve informar seu endereço de e-mail e senha previamente cadastrados no sistema.

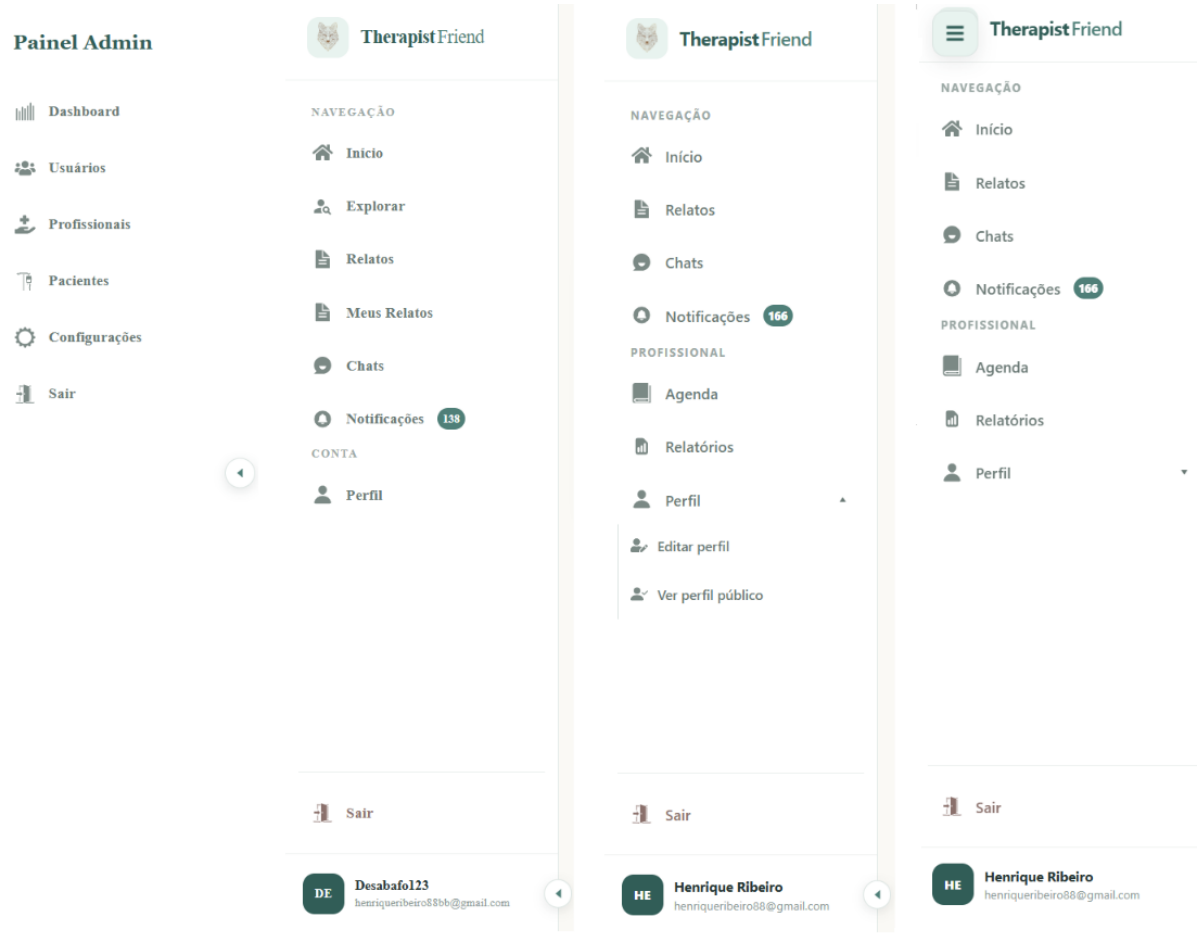
Além dos campos de autenticação, a tela disponibiliza mecanismos de navegação para as funcionalidades relacionadas ao cadastro de novos usuários e acesso a tela principal, permitindo que o usuário seja direcionado para os respectivos fluxos quando necessário.

5.2.3 Menu Lateral

O menu lateral é o componente essencial presente em quase todas as telas, sendo o principal responsável por disponibilizar as opções de navegação do sistema. A plataforma

apresenta estruturas distintas de menu para cada tipo de usuário, conforme ilustrado na Figura 32.

Figura 32 – Menus Laterais.



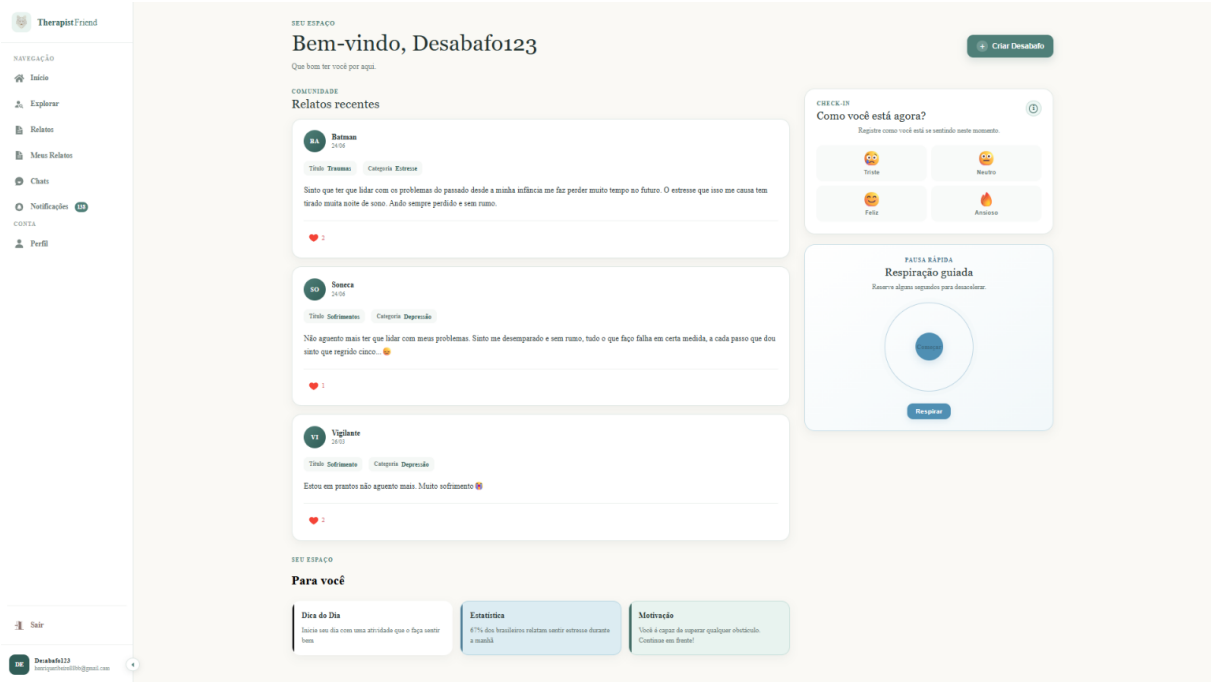
Fonte: Próprio autor

A Figura 32 apresenta, em ordem, as variações do menu lateral para cada perfil da plataforma: o primeiro corresponde ao administrador, o segundo ao paciente e o terceiro ao profissional. Por fim, a última imagem ilustra o comportamento do menu em dispositivos de telas menores, adotando o formato de menu hambúrguer.

5.2.4 Dashboard do Paciente

Após a autenticação, os usuários do tipo paciente são direcionados para o *dashboard* apresentado na Figura 33. Essa interface funciona como página inicial da área do paciente, reunindo as principais informações e funcionalidades disponíveis na plataforma.

Figura 33 – Dashboard do paciente.



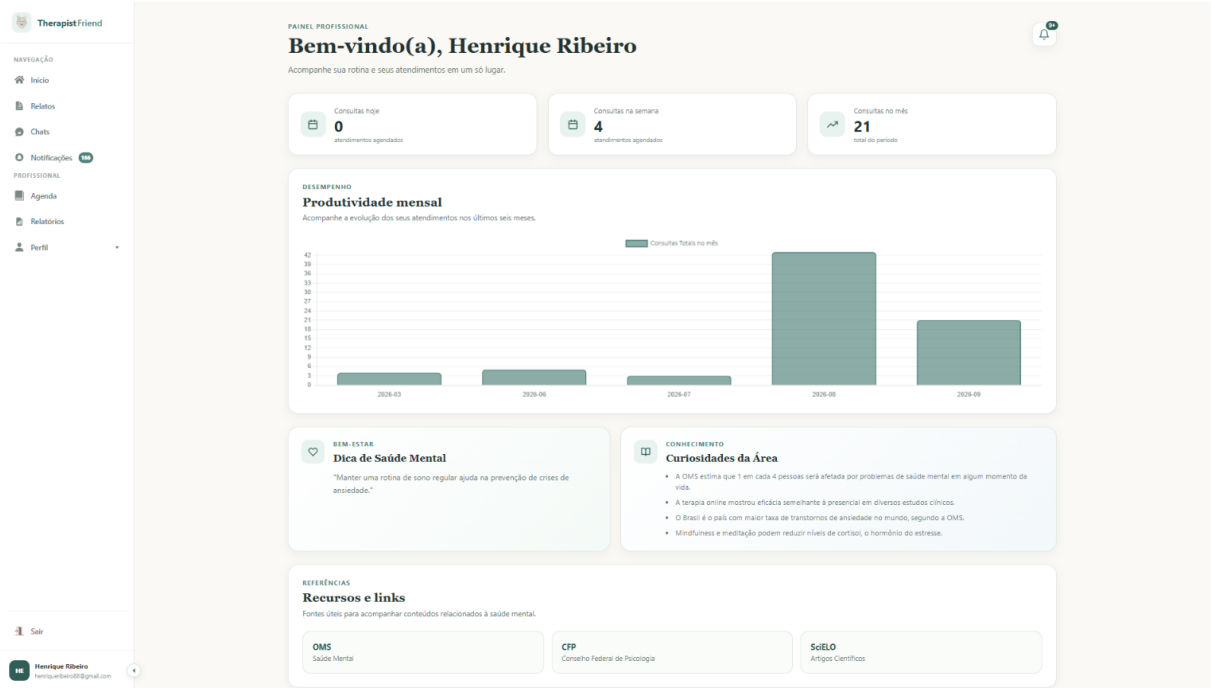
Fonte: Próprio autor

O *dashboard* do paciente foi desenvolvido com foco no acolhimento, centralizando o acesso a relatos recentes de outros usuários e a opção de abertura de um modal para novos depoimentos. Adicionalmente, a interface integra dois componentes dinâmicos: um registro de humor e uma ferramenta de respiração guiada, além de cards informativos que reagem dinamicamente ao estado emocional relatado pelo paciente.

5.2.5 Dashboard do Profissional

Após a autenticação, os usuários do tipo profissionais são direcionados para o *dashboard* apresentado na Figura 34. Essa interface funciona como página inicial da área do profissional, reunindo as principais informações e funcionalidades disponíveis na plataforma.

Figura 34 – Dashboard do profissional.



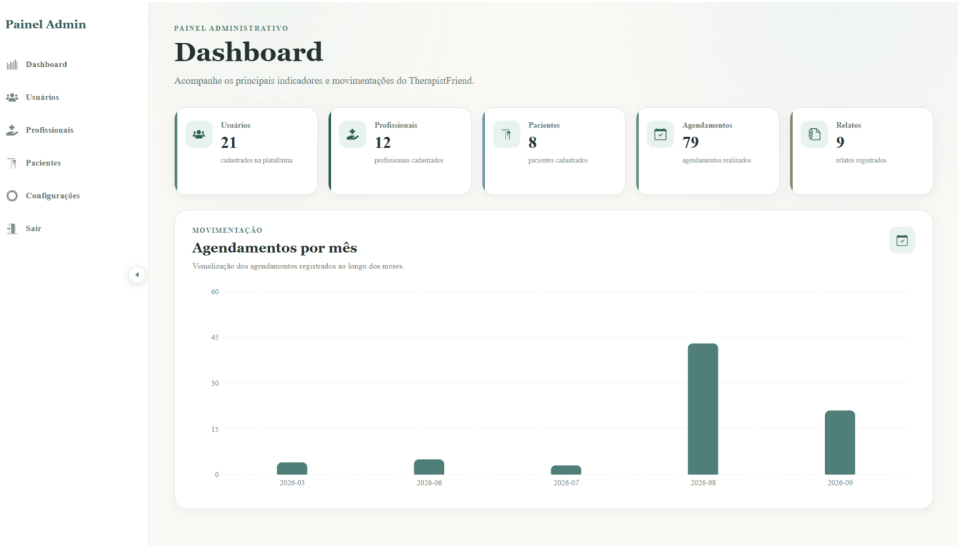
Fonte: Próprio autor

Projetado com foco na gestão de atendimentos, o painel do profissional exibe dados essenciais referentes às consultas agendadas. O componente dispõe ainda de três seções (cards) dedicadas a conteúdos sobre saúde mental, estruturadas atualmente de forma estática.

5.2.6 Dashboard Administrador

Após a autenticação, os usuários do tipo admin são direcionados para o *dashboard* apresentado na Figura 35. Essa interface funciona como página inicial da área do administrador, reunindo informações da plataforma.

Figura 35 – Dashboard do administrador.



Fonte: Próprio autor

O *dashboard* do administrador apresenta uma interface minimalista voltada à exibição de dados analíticos sobre a plataforma. Esse painel atua essencialmente como uma base de indicadores e métricas sobre a usabilidade e o engajamento dentro do sistema.

5.2.7 Relatos

A funcionalidade de relatos divide-se em duas telas: uma de uso comum para pacientes e profissionais, e outra exclusiva para a visualização dos relatos do próprio paciente. A tela comum renderiza os dados de forma dinâmica de acordo com o perfil de acesso. O comportamento para pacientes é exibido na Figura 36, e o formato para profissionais é detalhado na Figura 37.

Figura 36 – Relatos para pacientes.



Fonte: Próprio autor

Figura 37 – Relatos para profissionais.

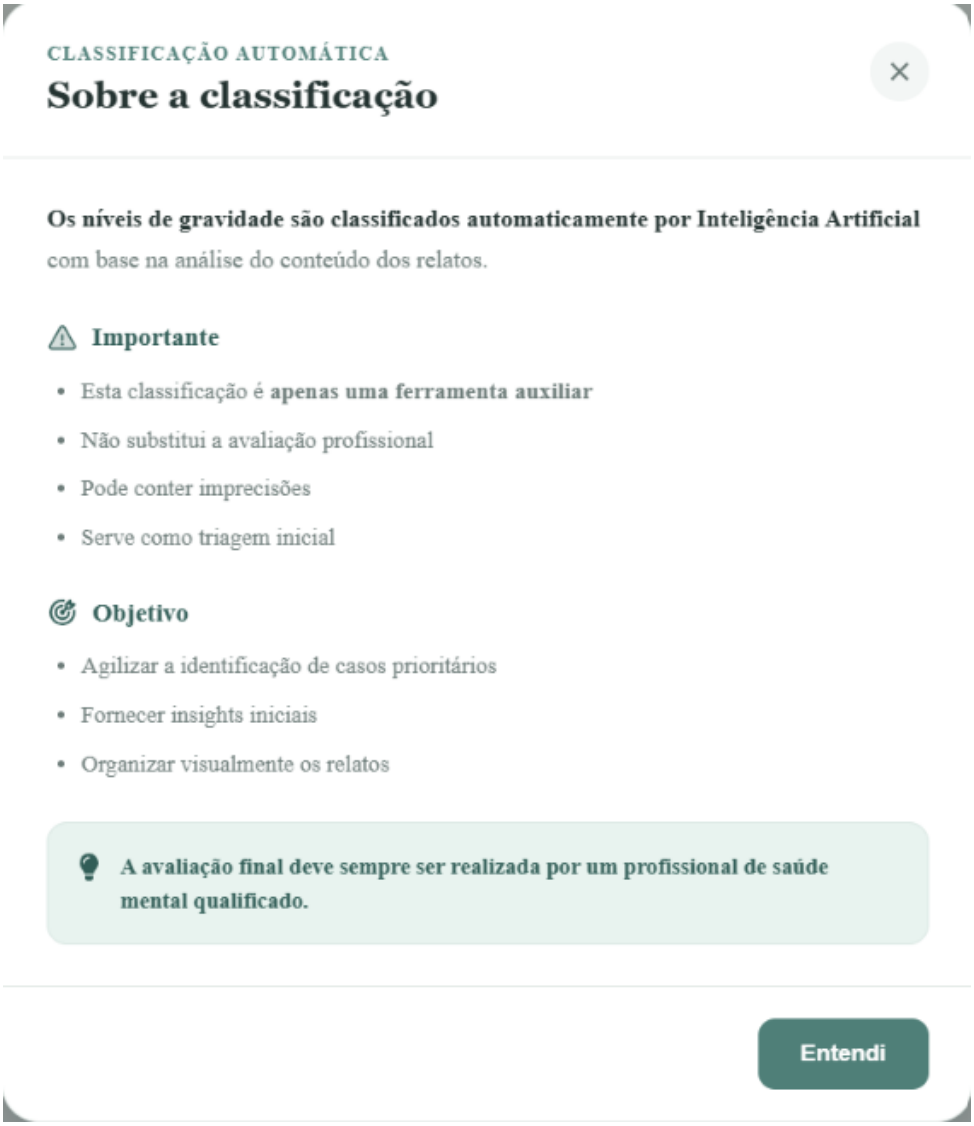


Fonte: Próprio autor

Como elemento comum a ambas as variações, nota-se a presença da paginação, do componente de renderização dos relatos e das tags de filtragem. A distinção reside na forma como os relatos são exibidos para os profissionais, os quais contam com uma análise de Inteligência Artificial (IA). Essa visualização inclui uma legenda explicativa em formato de grid para os níveis de classificação (leve, mediano, grave e indeterminado) e um botão que aciona o modal explicativo sobre a finalidade da IA, reforçando que a ferramenta não substitui o diagnóstico profissional, conforme ilustrado na Figura 38. Em contrapartida, a

interface do paciente não apresenta essa análise, exibindo, no lugar do grid, o botão para abertura do modal de criação de novos relatos. Vale ressaltar que os dados são estruturados e formatados pelo backend especificamente para cada perfil, eliminando a necessidade de manipulação ou tratamento de dados no frontend.

Figura 38 – Modal explicativo sobre a análise de relatos da inteligência artificial.

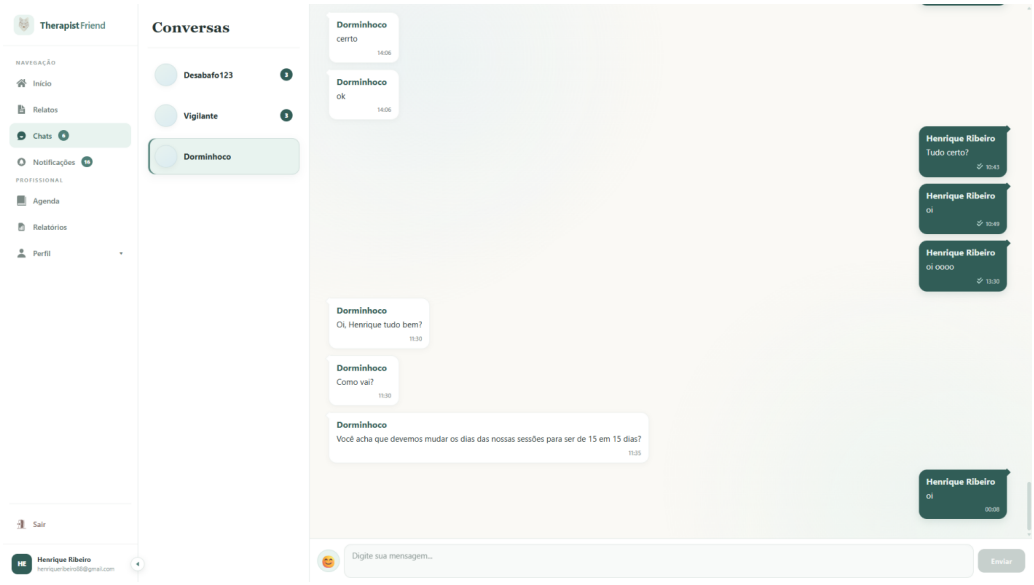


Fonte: Próprio autor

5.2.8 Chats

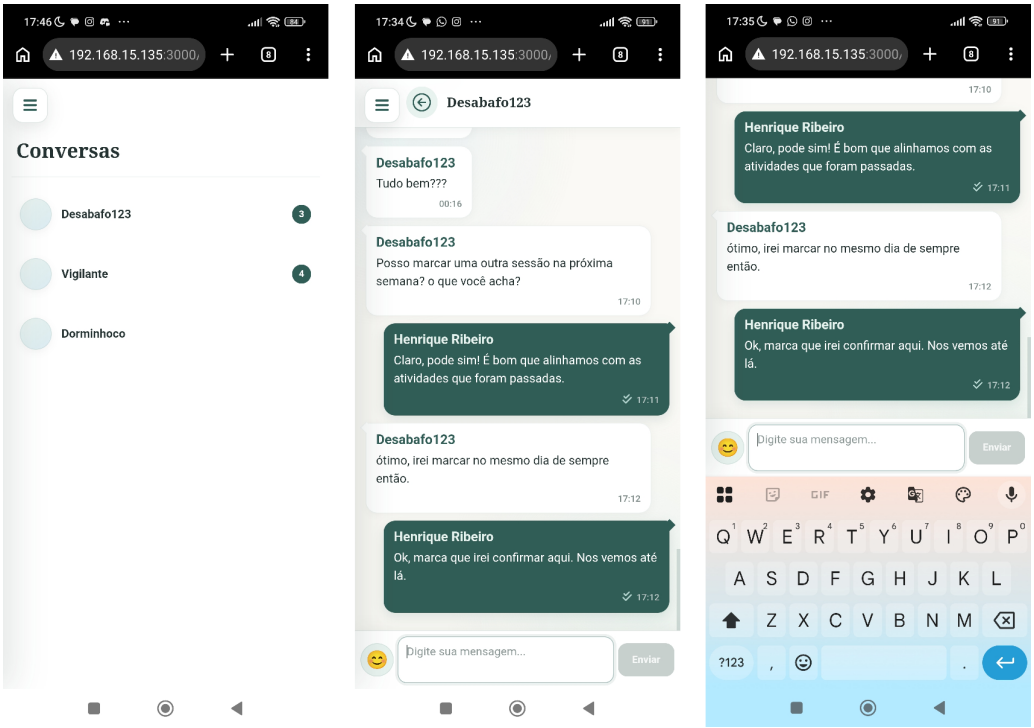
A tela de chat foi projetada com foco na usabilidade, garantindo uma experiência otimizada tanto em computadores (desktop) quanto em dispositivos móveis. Embora apresente variações visuais para se adequar a diferentes tamanhos de tela, a interface mantém estritamente as mesmas funcionalidades em ambas as plataformas. A Figura 39 ilustra o comportamento do chat em telas maiores, enquanto a Figura 40 apresenta sua disposição em ambiente móvel.

Figura 39 – Chat para desktop.



Fonte: Próprio autor

Figura 40 – Chat para mobile.



Fonte: Próprio autor

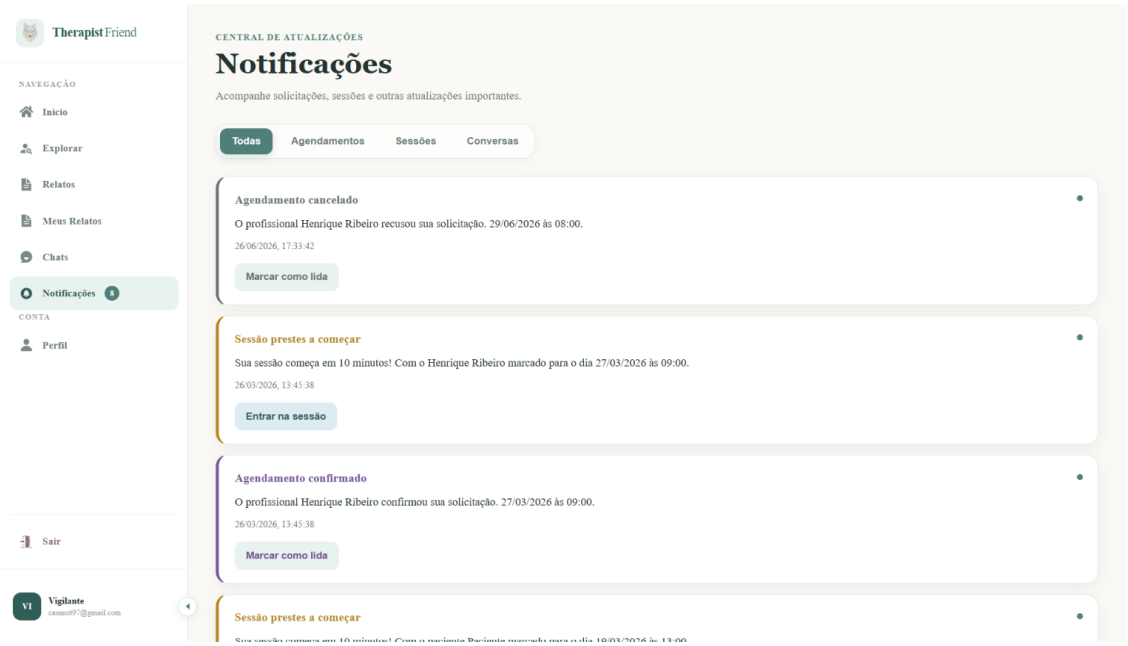
Como pode ser observado na Figura 39, o Context API específico para o gerenciamento do chat está em pleno funcionamento. No menu lateral, é exibida a contagem total de mensagens não lidas, enquanto a lista de conversas apresenta o total individualizado por usuário. Ao analisar a interface móvel, destacam-se as adaptações realizadas no *layout*

dinâmico, projetadas especificamente para ajustar os elementos de tela à subida do teclado virtual.

5.2.9 Notificações

O design da tela de notificações prioriza a simplicidade e a diferenciação visual entre as categorias de alertas, como ilustrado na Figura 41.

Figura 41 – Tela de notificações.



Fonte: Próprio autor

Para garantir essa distinção, o sistema associa uma cor exclusiva a cada categoria de notificação, conforme listado na Tabela 5.

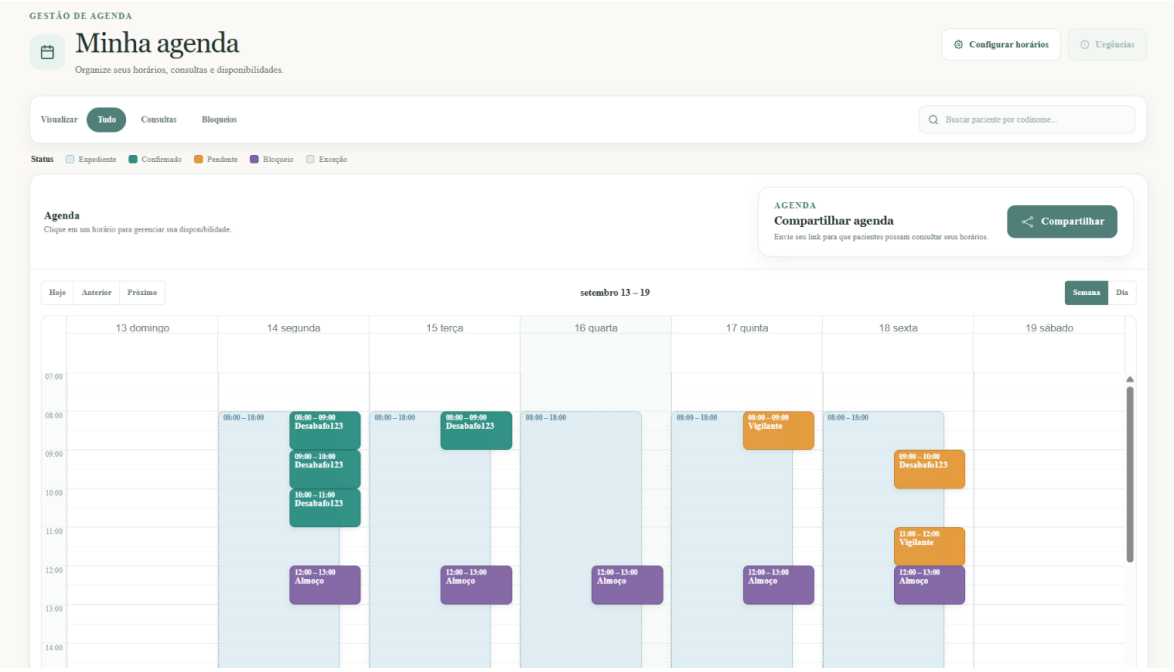
Tabela 5 – Cores utilizadas nas notificações.

Tipo da Notificação	**Cor Hexadecimal**
Nova Solicitação de Agendamento	#B34F73
Agendamento Confirmado	#76559A
Solicitação de Conversa Rejeitada	#D8892F
Nova Solicitação de Conversa	#3F7895
Sessão prestes a começar	#B78620
Solicitação de conversa	#40877E
Agendamento Cancelado	#687572
Relatório da sessão está disponível	#243331
Nova Conversa Criada	#2F8F83

5.2.10 Agenda

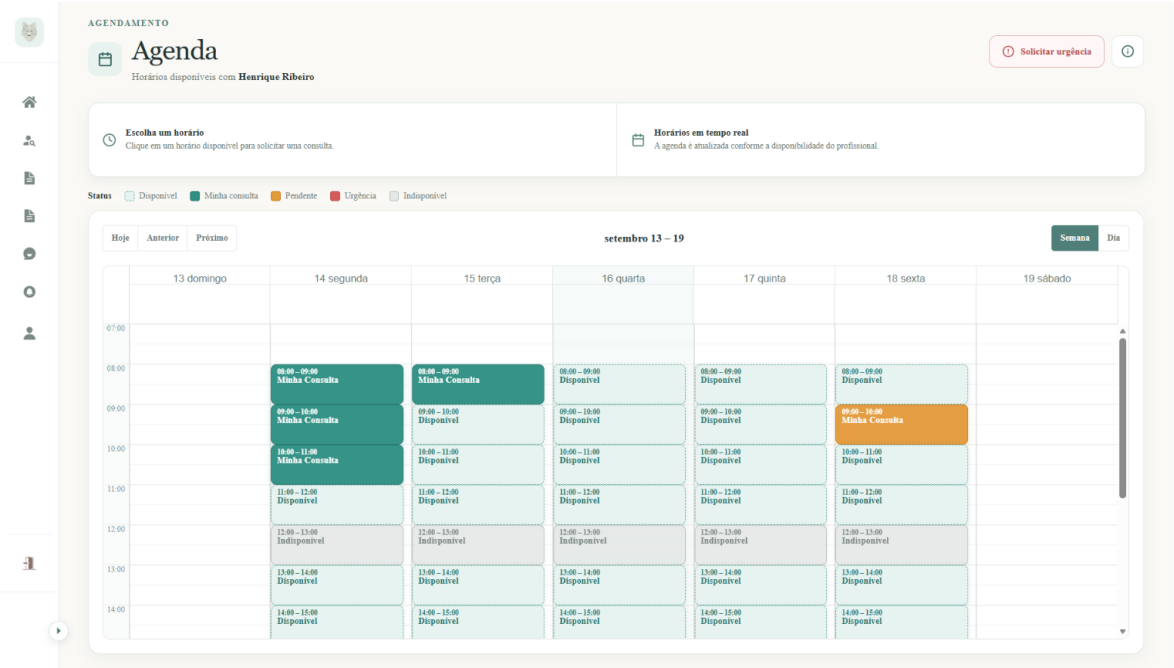
Desenvolvida com a biblioteca *React Big Calendar*, a agenda apresenta uma interface simples e familiar aos usuários. Diferenciando-se do padrão de mercado — que geralmente exibe apenas blocos com os horários selecionáveis —, esta solução oferece uma visualização mensal completa. O componente possui duas variantes de visualização: uma voltada para o profissional e outra para o paciente, conforme ilustrado nas Figuras 42 e 43, respectivamente.

Figura 42 – Agenda vista pelo profissional.



Fonte: Próprio autor

Figura 43 – Agenda vista pelo paciente.



Fonte: Próprio autor

As variações de interface baseiam-se estritamente nas necessidades de cada ator do sistema. Enquanto o profissional gerencia seus horários, o paciente necessita apenas visualizar as opções e efetuar os agendamentos. As especificidades de cada perfil são detalhadas a seguir:

- **Agenda do Profissional:** Apresenta recursos para configurar disponibilidades, além de criar ou remover bloqueios de horários. Permite aceitar, recusar ou cancelar consultas, dispondo também de filtros por tipo de evento (consultas, bloqueios ou exibição completa) e busca por codinome do paciente. Oferece visualizações por mês, semana, dia e eventos, além de um recurso de compartilhamento de link para facilitar o acesso do paciente.
- **Agenda do Paciente:** Exibe os horários disponíveis do profissional, apresenta o status atual de suas solicitações e conta com um componente informativo sobre o funcionamento do fluxo. Sua exibição restringe-se às perspectivas de semana e dia.

Ambas as interfaces adotam uma codificação visual por cores para otimizar a usabilidade, conforme discriminado na Tabela 6.

5.2.11 Perfil

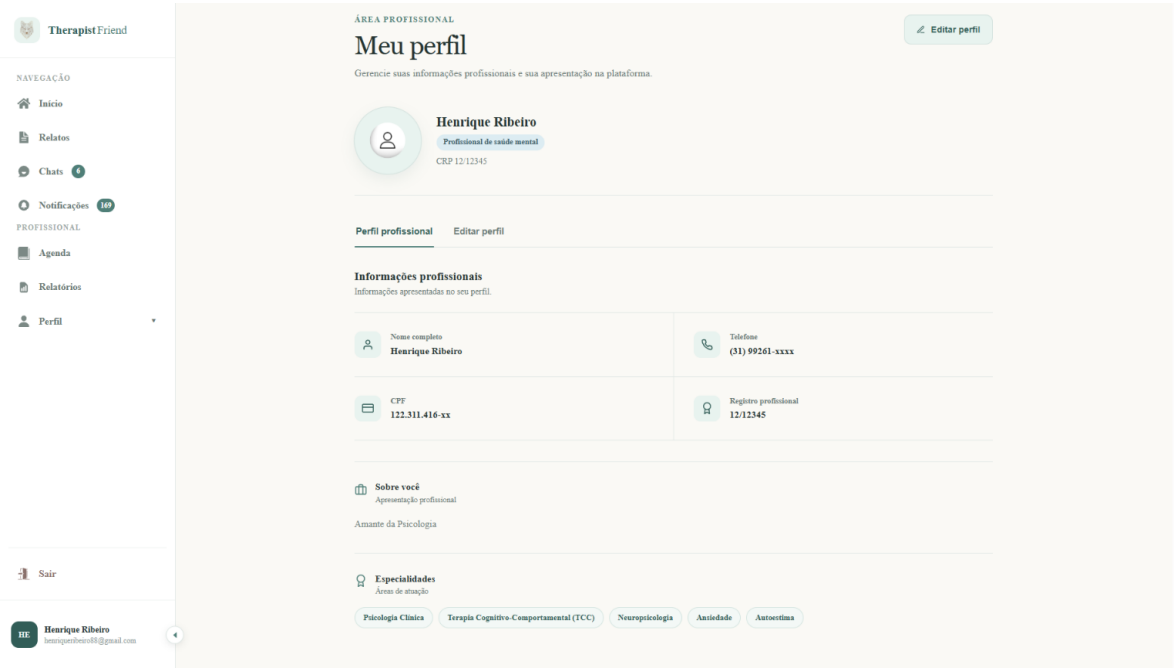
As telas de perfil foram desenvolvidas sob uma abordagem em duas vertentes: a primeira dedicada à configuração e visualização privada dos dados, disponível tanto para

Tabela 6 – Cores utilizadas na agenda.

Tipo do Evento	**Explicação**	**Cor Hexadecimal**
Expediente	Horário de trabalho	#DCECF2
Confirmado	Agendamento garantido	#2F8F83
Pendente	Aguardando resposta	#E39A3B
Bloqueio	Bloqueio pontual sobre o expediente	#8064A2
Exceções	Casos especiais de remoção de bloqueio	#E4E8E7

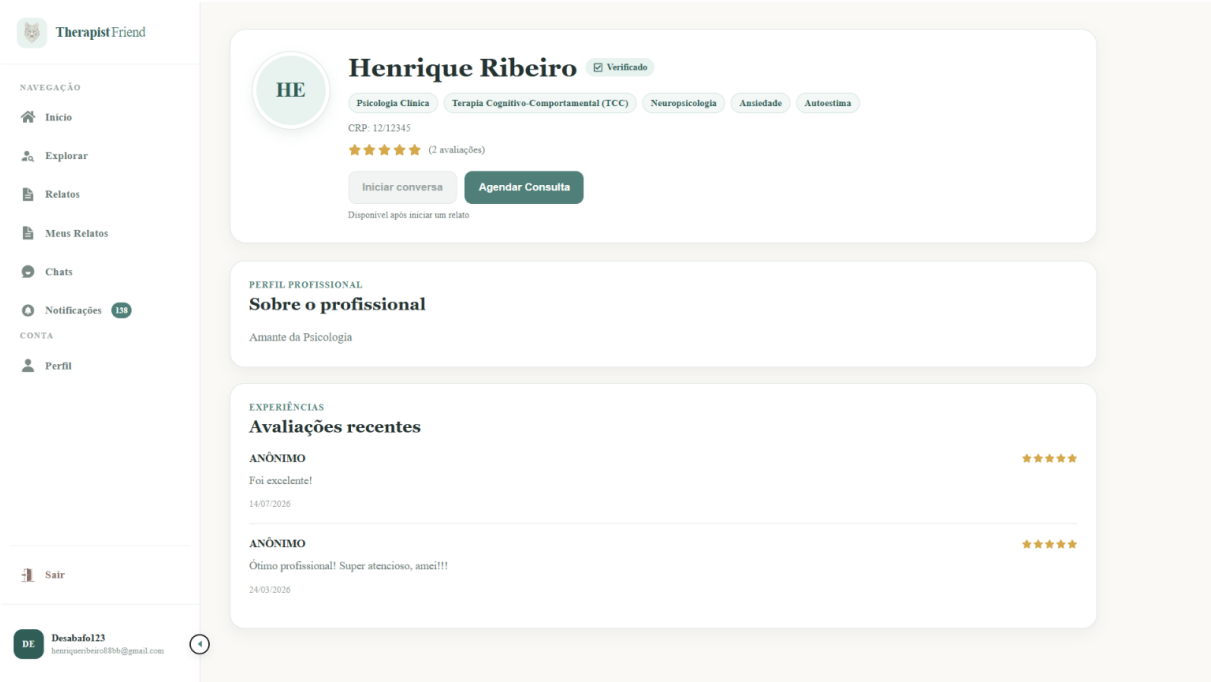
profissionais quanto para pacientes, e a segunda voltada à exibição pública do perfil do profissional. As interfaces de ajuste e visualização interna são apresentadas na Figura 44, enquanto a página pública correspondente ao profissional pode ser observada na Figura 45.

Figura 44 – Perfil privado do profissional.



Fonte: Próprio autor

Figura 45 – Perfil público do profissional.



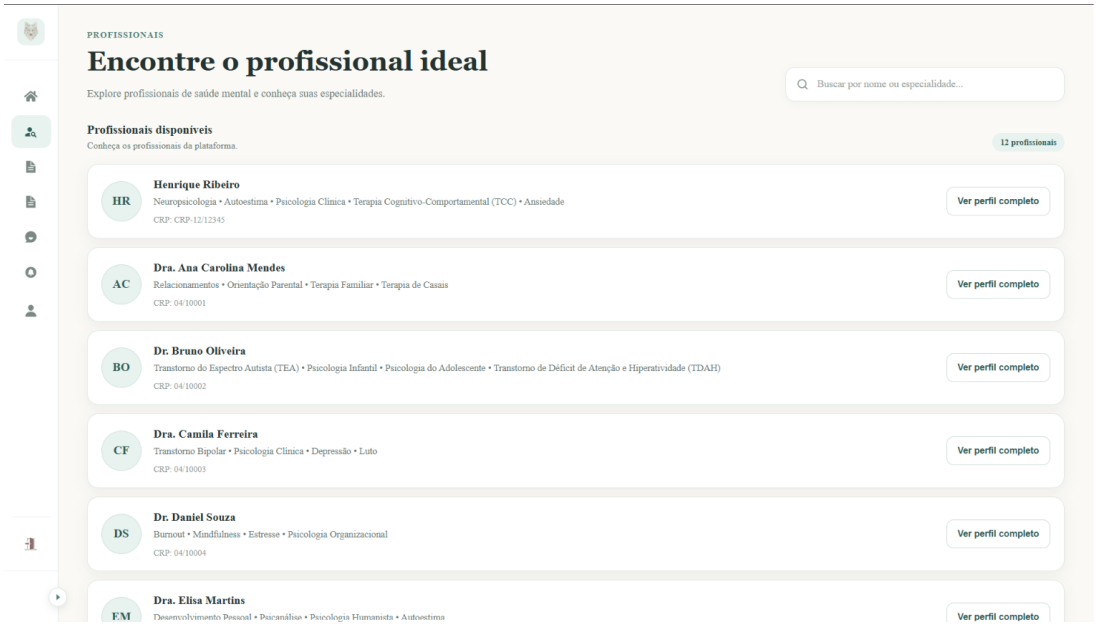
Fonte: Próprio autor

A interface privada de perfil dispõe de um card dinâmico que alterna entre os modos de visualização e edição. A Figura 44 ilustra a perspectiva do profissional, sendo que a área do paciente segue o mesmo padrão de layout, divergindo apenas quanto aos dados exibidos. Por sua vez, a tela de perfil público do profissional reúne suas informações básicas, o acesso à sua agenda de atendimentos e a exibição das avaliações submetidas pelos pacientes após as consultas online, conforme demonstrado na Figura 45.

5.2.12 Explorar

Tela reponsável pela listagem dos profissionais. Onde os pacientes podem procurar e ver mais sobre os profissionais antes de aceitarem conversar com eles. A Figura 46 mostra a tela.

Figura 46 – Listagem de profissionais.



Fonte: Próprio autor

A interface dispõe de um campo de busca que permite a filtragem por nome ou especialidade. Cada especialista é listado com informações resumidas, contudo, o sistema oferece a opção de abertura de um modal para a exibição do perfil completo, conforme ilustrado na Figura 47. Além de detalhar os dados do profissional, essa janela flutuante oferece atalhos para redirecionar o usuário tanto à página de perfil público — onde constam as avaliações — quanto diretamente à agenda de atendimentos. Adicionalmente, o componente de listagem integra um mecanismo de paginação.

Figura 47 – Perfil completo via modal.

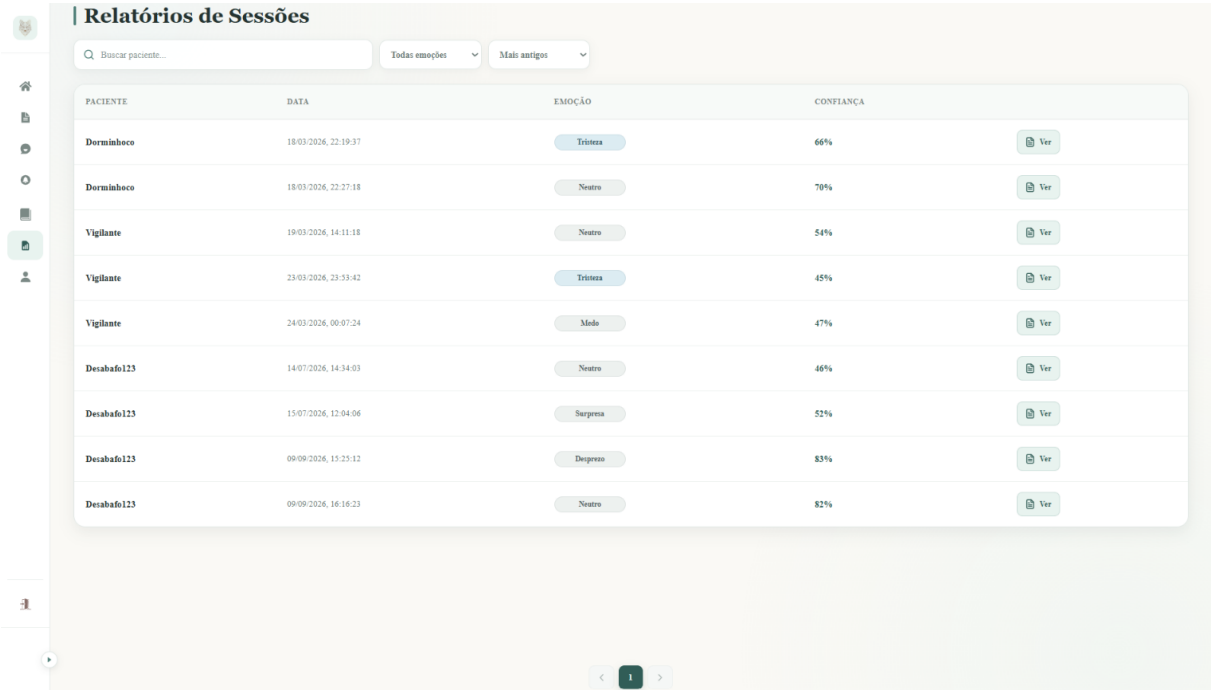


Fonte: Próprio autor

5.2.13 Relatórios

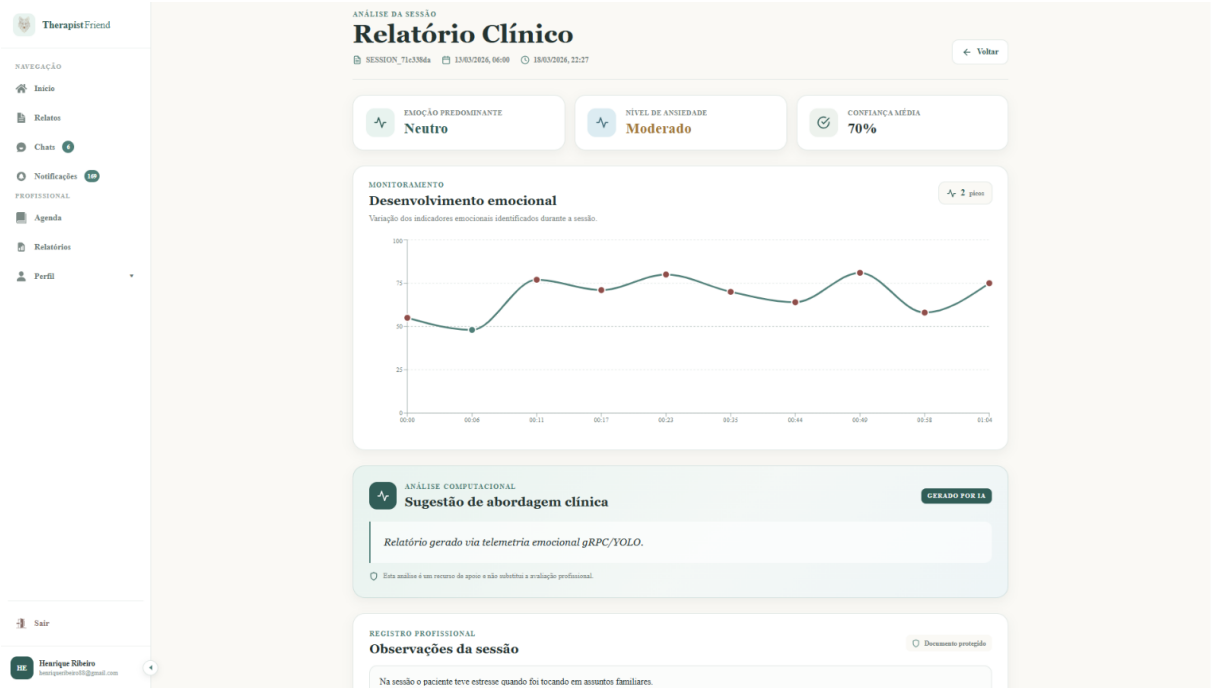
A funcionalidade de relatórios possui uma tela de listagem e uma página de detalhamento, priorizando um design limpo e intuitivo. A Figura 48 exibe a lista de relatórios do sistema, e a Figura 49 apresenta o documento completo com suas informações detalhadas.

Figura 48 – Lista de relatórios.



Fonte: Próprio autor

Figura 49 – Relatório.

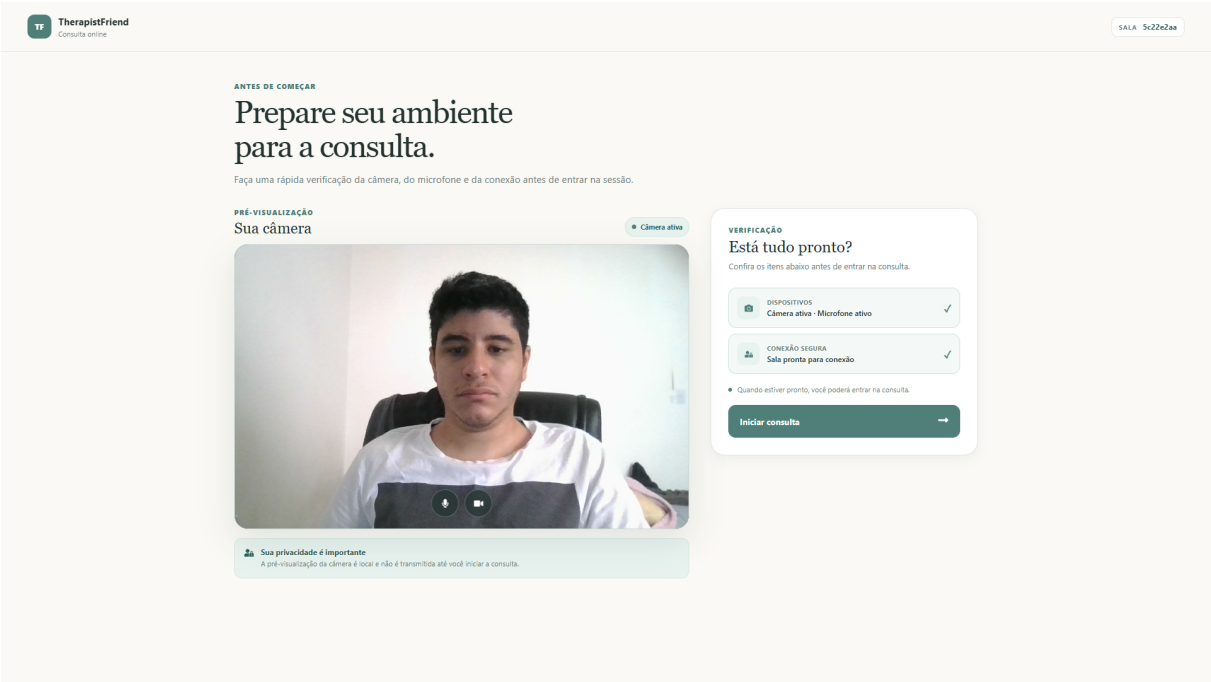


Fonte: Próprio autor

5.2.14 Sessão

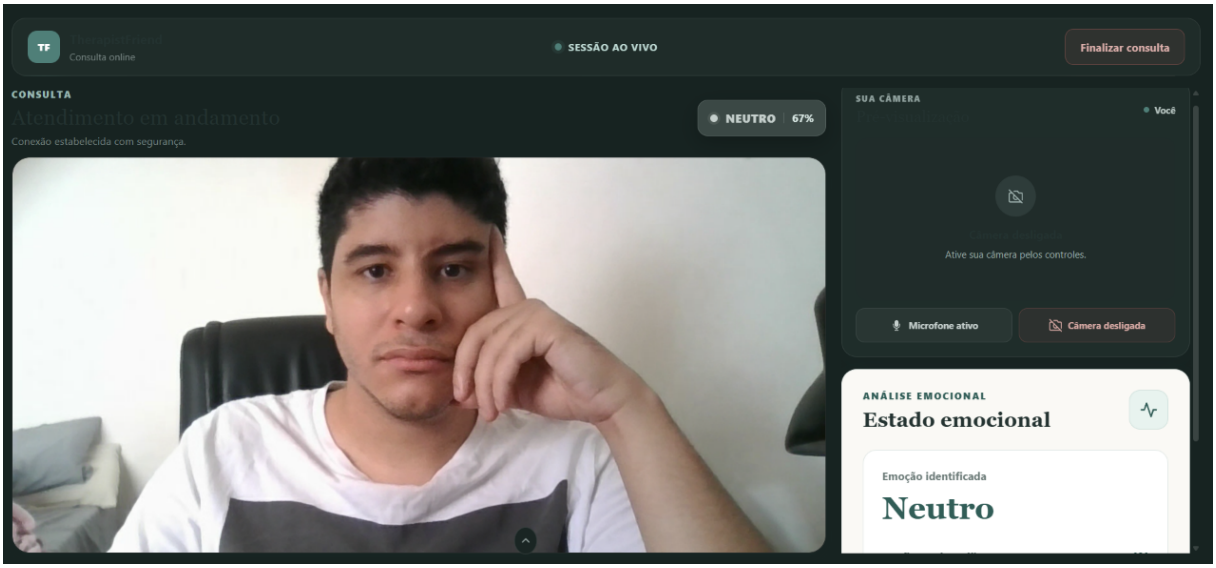
Para o funcionamento das sessões online, foi construído duas páginas distintas: uma é a sala de espera, onde os participantes precisam autorizar o uso de câmera e microfone, e a outra é a sala de sessão, onde ocorre a interação entre paciente e profissional. A Figura 50 apresenta a sala de espera, enquanto a Figura 51 ilustra a sala de sessão.

Figura 50 – Sala de espera.



Fonte: Próprio autor

Figura 51 – Sala de sessão.



Fonte: Próprio autor

Como se pode observar, a sala de espera apresenta um design limpo e minimalista. A interface exibe a pré-visualização do vídeo do usuário assim que a permissão da câmera é concedida, além de um botão para ingressar na sala de sessão. Há também dois indicadores textuais que confirmam a detecção do dispositivo de captura e a validação da conexão.

A sala de sessão apresenta uma estrutura mais robusta, com variações de interface projetadas especificamente para cada perfil de usuário:

- **Visão do Profissional**: Dispõe de um botão para encerrar o atendimento e gerar o relatório clínico, acompanhado de um indicador de que a sessão está ativa. A análise em tempo real das emoções é exibida em dois pontos estrategicamente posicionados: um indicador baseado em cores na parte superior do vídeo do paciente e um painel analítico mais amplo no canto inferior direito da tela. Adicionalmente, a interface oculta controles flutuantes para ativação e desativação do microfone e da câmera.
- **Visão do Paciente**: Apresenta um layout minimalista, centralizando a transmissão de vídeo do profissional e os botões flutuantes para controle de áudio e vídeo. O painel inclui também a sinalização de sessão ativa e uma mensagem informativa sobre a execução da análise por Inteligência Artificial.

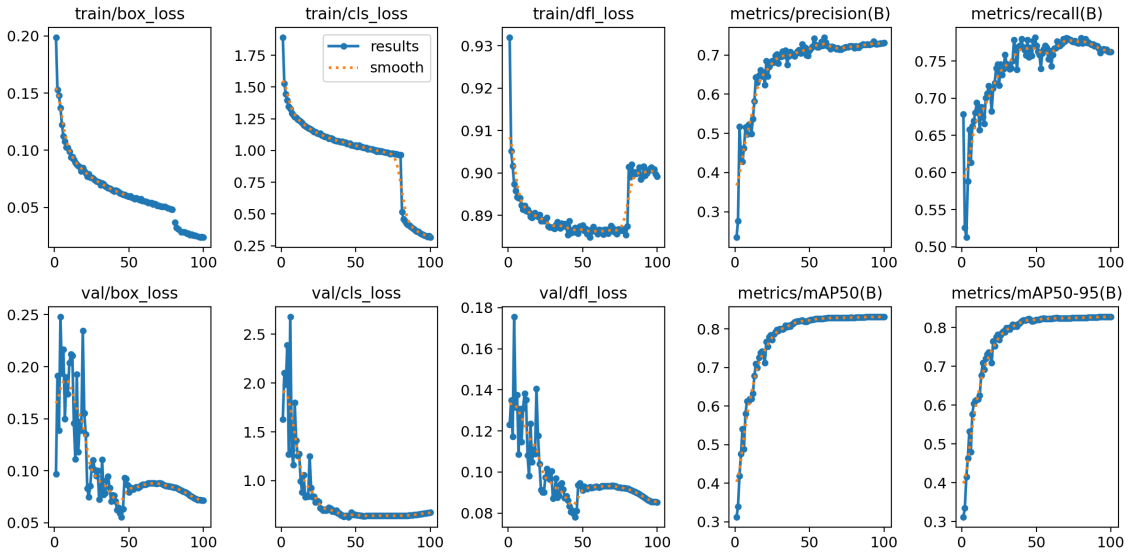
5.3 Resultados do Treinamento do Modelo de Visão Computacional

Esta seção apresenta os resultados experimentais obtidos durante o treinamento e a avaliação do modelo de visão computacional baseado na arquitetura *YOLOv8n*, desenvolvido para a identificação de categorias de expressões faciais. Os experimentos foram realizados utilizando o conjunto de dados *AffectNet* adaptado ao formato *YOLO*, considerando oito categorias de expressões. A avaliação contempla o comportamento das perdas durante o treinamento, as métricas gerais de detecção e a distribuição dos acertos e erros entre as classes, incluindo a ocorrência de falsos positivos associados ao *background*.

5.3.1 Análise Genérica de Convergência e Funções de Perda

Para avaliar o comportamento evolutivo da rede ao longo das 100 épocas de treinamento estipuladas, utilizou-se o arquivo de métricas consolidadas `results.png`, gerado nativamente pelo *framework* da *Ultralytics*. O comportamento histórico das funções de perda (*loss curves*) e das métricas gerais de validação está ilustrado na Figura 52.

Figura 52 – Curvas de perda (*loss*) e evolução das métricas de validação ao longo do treinamento.



Fonte: Próprio autor

O comportamento do treinamento do modelo foi acompanhado por meio das curvas de perda (*loss*) e das métricas de avaliação registradas ao longo das 100 épocas, conforme apresentado na Figura 52. Foram consideradas as perdas de localização das caixas delimitadoras (`box_loss`), classificação (`cls_loss`) e distribuição focal (`dfl_loss`), tanto para os dados de treinamento quanto para os dados de validação.

Ao longo do treinamento, observou-se uma redução progressiva da perda de classificação nos dados de treinamento (`train/cls_loss`), que passou de 1,892 na primeira época para 0,96699 na época 80. Esse comportamento indica que o modelo reduziu gradualmente os erros de classificação durante o processo de ajuste dos seus parâmetros.

Entretanto, a evolução da perda de classificação nos dados de validação (`val/cls_loss`) apresentou um comportamento diferente. Embora tenha ocorrido uma redução expressiva nas primeiras épocas, essa métrica atingiu seu menor valor na época 45, com 0,62764, e posteriormente apresentou pequenas oscilações e tendência de crescimento, alcançando 0,67323 na última época. A diferença entre as curvas de treinamento e validação indica que a redução dos erros nos dados utilizados para treinamento não foi acompanhada por uma melhora equivalente nos dados de validação. Esse comportamento evidencia uma diferença de desempenho entre os dois conjuntos e exige cautela na interpretação da capacidade de generalização do modelo.

Outro comportamento relevante foi observado entre as épocas 80 e 81. Conforme a configuração do treinamento, foi utilizado o parâmetro `close_mosaic=20`, que desativa a técnica de aumento de dados por mosaico nas 20 épocas finais. Após essa alteração, a `train/cls_loss` apresentou uma redução acentuada, passando de 0,96699 na época 80

para 0,51695 na época 81. Essa queda está relacionada à mudança na distribuição dos exemplos utilizados durante o treinamento, que deixam de apresentar a composição artificial de imagens produzida pelo mosaico. Entretanto, a perda de validação não apresentou uma redução correspondente, passando de 0,64007 para 0,64083 nesse intervalo. Portanto, essa alteração não deve ser interpretada isoladamente como uma melhora na capacidade de generalização do modelo.

Apesar da diferença observada entre as perdas de treinamento e validação, as métricas de detecção apresentaram evolução positiva ao longo do processo. O mAP@50 atingiu seu maior valor na época 98, com 83,14%, enquanto o mAP@50-95 alcançou 82,75% na época 96. Ao final das 100 épocas, os valores registrados foram de 83,13% e 82,73%, respectivamente. Esses resultados indicam que o modelo manteve desempenho próximo aos seus melhores valores de avaliação até o encerramento do treinamento.

Dessa forma, os resultados demonstram que o treinamento possibilitou a redução das perdas e a evolução das métricas de detecção. Contudo, a diferença entre as curvas de classificação dos conjuntos de treinamento e validação, especialmente nas épocas finais, representa uma limitação que deve ser considerada. Assim, os resultados indicam desempenho consistente nas métricas gerais de detecção, mas não permitem afirmar, isoladamente, que não houve sobreajuste (*overfitting*) durante o treinamento.

5.3.2 Matriz de Confusão e Avaliação de Desempenho

Inicialmente, foi realizado um treinamento utilizando imagens com resolução de entrada de 96×96 pixels. Posteriormente, um segundo treinamento foi executado com resolução de 224×224 pixels. Ambos utilizaram as mesmas configurações experimentais, diferenciando-se apenas pelo parâmetro `imgsz`. As matrizes de confusão resultantes são apresentadas nas Figuras 53 e 54, permitindo comparar a distribuição das classificações corretas e incorretas entre as duas resoluções.

Um dos aspectos observados nessa comparação foi a quantidade de falsos positivos associados ao *background*. No treinamento com resolução de 96×96 pixels, foram contabilizadas 4.867 detecções incorretas nessa categoria. Já no treinamento com resolução de 224×224 pixels, esse número foi reduzido para 2.301, representando uma diminuição de aproximadamente 52,72%.

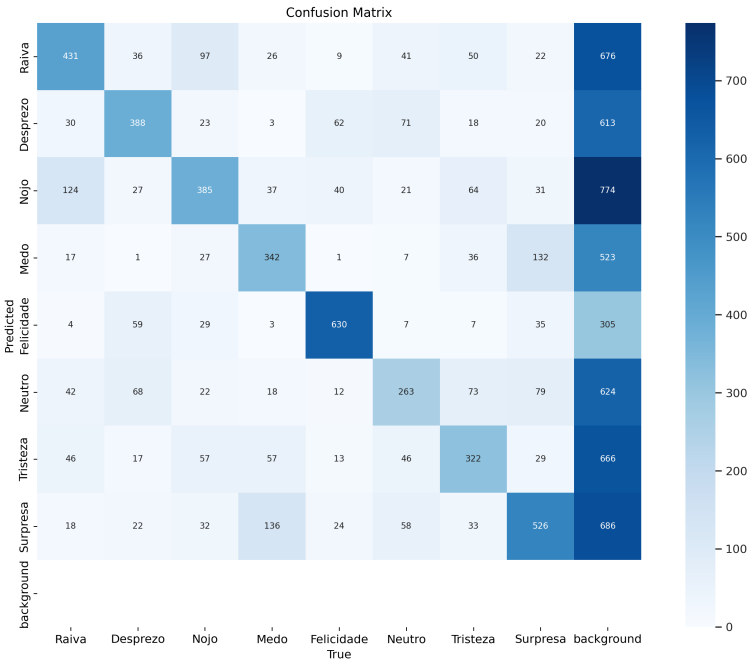
Essa redução foi calculada pela diferença entre as quantidades de falsos positivos dos dois treinamentos, dividida pelo valor inicial, conforme a Equação 5.1:

$$R = \frac{FP_{96} - FP_{224}}{FP_{96}} \times 100 \tag{5.1}$$

Em que FP_{96} e FP_{224} representam, respectivamente, as quantidades de falsos positivos associados ao *background* nos treinamentos com resoluções de 96×96 e 224×224 pixels.

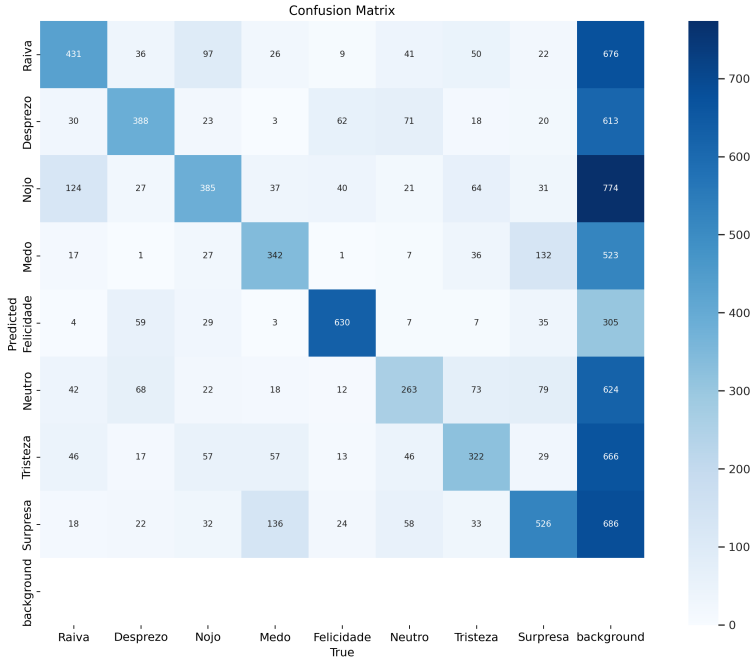
Além da redução observada nos falsos positivos, a comparação das métricas gerais de validação também apresentou resultados favoráveis ao treinamento com maior resolução. Considerando que as demais configurações foram mantidas, os resultados indicam que, nas condições experimentais utilizadas, a resolução de 224×224 pixels favoreceu o desempenho do modelo. Por esse motivo, essa configuração foi adotada na versão integrada à plataforma.

Figura 53 – Matriz de confusão resultante do modelo YOLOv8n otimizado com entrada de 96×96 pixels.



Fonte: Próprio autor

Figura 54 – Matriz de confusão resultante do modelo YOLOv8n otimizado com entrada de 224 × 224 pixels.



Fonte: Próprio autor

A matriz de confusão apresentada na Figura 54 permite analisar a distribuição dos acertos e dos erros de classificação entre as oito categorias de expressões faciais utilizadas no treinamento. A diagonal principal representa as detecções classificadas corretamente, enquanto os valores fora dessa diagonal indicam ocorrências em que uma categoria foi confundida com outra. A matriz também apresenta a categoria *background*, utilizada para representar regiões sem uma classe correspondente. A partir desses dados, são derivadas as métricas de desempenho por classe exibidas na Tabela 7, cujos fundamentos de cálculo são detalhados na Subseção 5.3.2.1.

Entre as categorias avaliadas, Felicidade apresentou o maior número de acertos, com 665 ocorrências corretamente classificadas. Essa classe também obteve os melhores resultados de precisão (72,20%), revocação (84,07%) e F1-score (77,69%), indicando maior consistência na identificação dessa categoria em comparação às demais.

Por outro lado, a categoria Neutro apresentou maior dificuldade de classificação, com 314 acertos e F1-score de 44,70%. Observa-se que diversas ocorrências foram classificadas incorretamente como Neutro, contribuindo para a redução de sua precisão. Também foram identificadas confusões entre outras categorias, como Medo e Surpresa, evidenciadas por 100 ocorrências verdadeiras de Medo classificadas como Surpresa e 77 ocorrências verdadeiras de Surpresa classificadas como Medo.

Outro aspecto relevante é a quantidade de falsos positivos associados ao *background*.

Foram contabilizadas 2.301 detecções de categorias de expressões em regiões classificadas como *background*. Esse resultado evidencia uma limitação do modelo na distinção entre regiões que apresentam características compatíveis com as classes treinadas e regiões que não deveriam gerar uma detecção.

Dessa forma, embora o modelo tenha apresentado resultados mais consistentes em categorias como Felicidade, os valores observados para Neutro e a quantidade de falsos positivos relacionados ao *background* demonstram que ainda existem limitações na capacidade de discriminação entre as classes. Esses resultados devem ser considerados na interpretação das informações apresentadas pela plataforma, uma vez que as classificações representam categorias visuais identificadas pelo modelo, não uma confirmação do estado emocional real do participante.

5.3.2.1 Cálculo das Métricas por Classe

A partir da matriz de confusão (Figura 54) gerada automaticamente pelo Ultralytics YOLO, foram calculadas as métricas de precisão (*Precision*), revocação (*Recall*) e F1-score para cada uma das oito categorias de expressões faciais. Para isso, foram utilizados os valores absolutos apresentados na matriz, considerando a diagonal principal como o número de detecções classificadas corretamente para cada categoria.

Como a matriz apresenta as classes previstas nas linhas e as classes reais nas colunas, a soma dos valores de cada linha corresponde ao total de previsões realizadas para determinada categoria, enquanto a soma dos valores de cada coluna representa o total de ocorrências reais daquela categoria. Dessa maneira, os valores necessários para o cálculo das métricas foram obtidos diretamente da distribuição dos acertos e erros apresentada na matriz.

A precisão (*Precision*) representa a proporção de previsões de determinada classe que foram classificadas corretamente. Seu cálculo é realizado pela divisão dos verdadeiros positivos (*True Positives* – TP) pela soma dos verdadeiros positivos e falsos positivos (*False Positives* – FP), conforme a Equação 5.2:

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

A revocação (*Recall*) representa a proporção de ocorrências reais de uma classe que foram identificadas corretamente pelo modelo. Seu cálculo considera a divisão dos verdadeiros positivos pela soma dos verdadeiros positivos e falsos negativos (*False Negatives* – FN), conforme a Equação 5.3:

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

Por sua vez, o F1-score corresponde à média harmônica entre precisão e revocação, permitindo avaliar conjuntamente essas duas métricas. Seu cálculo é apresentado na Equação 5.4:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5.4}$$

Os resultados numéricos obtidos para cada classe a partir dessas formulações são sintetizados na Tabela 7.

Tabela 7 – Métricas de desempenho por classe

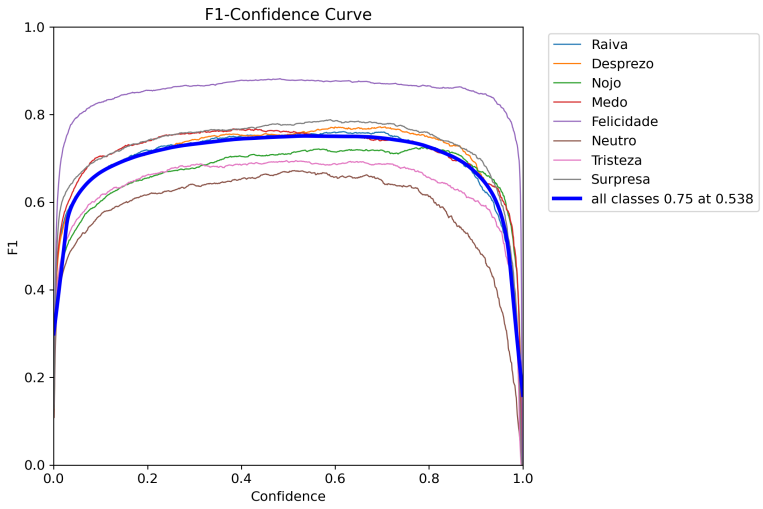
Classe	**Acertos (TP)**	**Precision**	**Recall**	**F1-score**
Raiva	501	48,88%	70,37%	57,69%
Desprezo	469	50,43%	75,89%	60,59%
Nojo	473	43,72%	70,39%	53,93%
Medo	416	53,27%	66,88%	59,30%
Felicidade	665	72,20%	84,07%	77,69%
Neutro	314	35,24%	61,09%	44,70%
Tristeza	395	45,51%	65,51%	53,70%
Surpresa	651	53,85%	74,49%	62,51%

Ressalta-se que as métricas foram calculadas individualmente para cada categoria a partir dos valores da matriz de confusão, considerando a inclusão do background na contabilização dos falsos positivos e falsos negativos. A acurácia convencional por classe não foi utilizada, pois a matriz de detecção de objetos não fornece os verdadeiros negativos necessários para esse cálculo.

5.3.3 Curvas de Sensibilidade Paramétrica e Ponto de Operação

Para determinar o limiar de corte lógico ideal (*confidence threshold*) a ser integrado na rotina do microserviço de produção, avaliou-se o comportamento das curvas de sensibilidade do modelo via arquivos `F1_curve.png`, `P_curve.png` e `R_curve.png`. A curva unificada *F1-Confidence* é apresentada na Figura 55.

Figura 55 – Curva F1-Confidence mapeando o ponto ótimo de equilíbrio preditivo.



Fonte: Próprio autor

A análise matemática do gráfico indica que thresholds de confiança excessivamente restritivos (próximos a 1.0) penalizam severamente a taxa de captura de expressões reais (*Recall*), enquanto limiares muito permissivos inserem ruídos espaciais na detecção. O ponto ótimo de equilíbrio estatístico ocorre na interseção expressa por ***all classes 0.75 at 0.538***.

Isso indica que ao adotar um limiar de corte de $0,54$ (54% de confiança), o modelo sustenta seu *F1-Score* máximo global de $0,75$, garantindo harmonia entre a precisão das predições emitidas e a sensibilidade de capturar as microexpressões do paciente ao longo da sessão.

Diante disso, fixou-se o parâmetro padrão na rotina de inferência do microsserviço (`model.predict(conf=0.54)`). Para mitigar as perdas decorrentes de subamostragem física no ecossistema de produção, o *frontend* foi configurado para processar e transmitir os frames da webcam em uma resolução padronizada de 480×480 pixels com compressão JPEG de 85%, garantindo baixa latência de tráfego de rede via *WebSockets*/*gRPC* sem degradação da acurácia clínica obtida no mapeamento de relevo facial.

5.3.4 Discussão dos Resultados

Embora o modelo tenha apresentado resultados satisfatórios, algumas limitações foram observadas em cenários específicos. Em situações de baixa iluminação, oclusão parcial da face e variações acentuadas no ângulo da câmera, houve uma redução na confiabilidade das respostas geradas.

Apesar dessas restrições, o desempenho obtido mostrou-se adequado para o objetivo deste trabalho. Isso ocorre porque o sistema de visão computacional atua estritamente

como uma ferramenta complementar ao trabalho do profissional de saúde mental, não sendo utilizado para diagnósticos ou tomadas de decisão clínica. Dessa forma, os resultados demonstram que o modelo é capaz de fornecer informações adicionais úteis para o acompanhamento emocional durante as sessões online.

6 Considerações Finais

O presente trabalho teve como objetivo desenvolver uma plataforma *web* para a conexão entre pacientes e profissionais da saúde mental, integrando tecnologias de engenharia de *software*, comunicação em tempo real, sistemas distribuídos e inteligência artificial em uma única solução. A proposta buscou oferecer um ambiente seguro para acolhimento, acompanhamento emocional e realização de atendimentos *online*, fornecendo recursos tanto para os pacientes quanto para os profissionais.

Ao longo do projeto, todas as funcionalidades inicialmente propostas foram concretizadas, permitindo a construção de um sistema capaz de unificar diferentes módulos de forma organizada e desacoplada. O ecossistema contempla autenticação segura, publicação de relatos anônimos, comunicação privada entre usuários, gerenciamento de agendamentos, sessões de videoconferência *online*, geração automática de relatórios e a aplicação de inteligência artificial como suporte ao acompanhamento emocional.

6.1 Considerações sobre o Desenvolvimento da Plataforma

O processo de engenharia evidenciou a importância de adotar uma arquitetura organizada para aplicações que concentram um grande volume de funcionalidades e regras de negócio complexas. A adoção do *Domain-Driven Design* (DDD) permitiu estruturar o sistema em módulos independentes, o que favoreceu a separação de responsabilidades e facilitou a manutenção e a evolução do código-fonte.

A organização modular do *backend* principal possibilitou o avanço independente dos diferentes domínios da aplicação — incluindo agenda, relatos, pacientes, profissionais, usuários, autenticação, sessões, *chat*, notificações, humor diário e especialidades —, reduzindo drasticamente o acoplamento entre os componentes.

A aplicação dos princípios SOLID, com especial destaque para o Princípio da Inversão de Dependência, contribuiu para tornar os módulos reutilizáveis. Adicionalmente, empregou-se o padrão *Singleton* em componentes compartilhados da aplicação, enquanto a adoção de *Domain Events* favoreceu a comunicação assíncrona e desacoplada entre diferentes contextos do sistema.

Outro aspecto relevante envolveu o gerenciamento de filas e mensageria por meio do *Redis* e da biblioteca *BullMQ*. Essa abordagem permitiu que tarefas de alto custo computacional fossem executadas em segundo plano (*background jobs*), mitigando o impacto sobre o fluxo principal de requisições do *backend* e otimizando o desempenho geral do sistema.

A implementação da comunicação bidirecional em tempo real via *Socket.io* viabilizou a sincronização instantânea de dados entre pacientes e profissionais, abrangendo recursos essenciais como notificações imediatas, *chat* privado e solicitações de conversa. Complementarmente, a realização das sessões *online* foi viabilizada pelo ecossistema *WebRTC*, tecnologia que assegurou o estabelecimento de conexões ponto a ponto (*P2P*) para a transmissão estável de fluxos de áudio e vídeo de alta qualidade com baixa latência, utilizando o próprio *Socket.io* como seu canal de sinalização.

Sob a ótica da segurança, foram concebidos mecanismos de autenticação baseados em *JSON Web Tokens* (JWT) combinados a *Refresh Tokens* armazenados em *cookies HTTPOnly*, além da aplicação de criptografia simétrica AES-256 nas mensagens trocadas entre os usuários. Incorporaram-se também rotinas de validação de e-mail com o *Nodemailer* e verificação telefônica por meio da *API* da *Twilio*, elevando a confiabilidade dos cadastros na base de dados.

Por fim, a cobertura de testes unitários durante o ciclo de desenvolvimento validou as regras de negócio implementadas, assegurando a estabilidade e a robustez do *backend* contra regressões de código.

6.2 Considerações sobre as Tecnologias Selecionadas

As tecnologias selecionadas mostraram-se adequadas para atender aos requisitos funcionais e não funcionais estabelecidos para o projeto. A adoção do *Node.js* com *Express* e *TypeScript* possibilitou o desenvolvimento de uma *API* robusta, modular e fortemente tipada, o que favoreceu a manutenção e reduziu a incidência de erros durante o ciclo de desenvolvimento.

No *frontend*, o emprego do *React* permitiu a construção de uma interface baseada em componentes reutilizáveis, facilitando a organização estrutural e promovendo a reusabilidade de código. A *Context API* simplificou o gerenciamento de estados globais, enquanto a implementação de leiautes responsivos viabilizou o acesso à plataforma em diferentes resoluções de tela.

A implementação de mecanismos de proteção de rotas garantiu que funcionalidades restritas permanecessem acessíveis estritamente aos usuários devidamente autenticados. Paralelamente, a integração com a biblioteca *React Big Calendar* permitiu o desenvolvimento de um ecossistema para o gerenciamento de disponibilidades, bloqueios de agenda e marcação de consultas entre pacientes e profissionais.

A arquitetura baseada em microsserviços também demonstrou resultados positivos. O microsserviço textual, desenvolvido em *Flask* com o suporte do *Ollama*, permitiu realizar análises dos relatos publicados pelos pacientes e a geração de cartões (*cards*) personalizados

com recomendações e mensagens de apoio.

Da mesma forma, o microsserviço de visão computacional possibilitou a integração entre o reconhecimento facial e a detecção de emoções, utilizando um modelo preditivo baseado na arquitetura *YOLO*, treinado especificamente para este trabalho com a base de dados *AffectNet*. A comunicação entre o *backend* principal e esse serviço especializado foi estabelecida por meio do protocolo *gRPC*, proporcionando baixa latência durante as inferências executadas ao longo das sessões *online*.

A integração dessas diferentes ferramentas demonstrou que abordagens baseadas em arquiteturas distribuídas podem ser empregadas de forma eficiente em ecossistemas voltados à área da saúde, permitindo a separação de responsabilidades e facilitando futuras expansões do sistema.

6.3 Validação Conceitual da Plataforma

Durante o desenvolvimento da plataforma TherapistFriend, foram realizadas demonstrações da solução para duas profissionais da Psicologia, com o objetivo de apresentar as funcionalidades implementadas e obter contribuições relacionadas à sua concepção e utilização no contexto do acompanhamento psicológico.

Nessas demonstrações, foram apresentados os principais recursos da plataforma, incluindo a publicação de relatos, a comunicação entre pacientes e profissionais, o gerenciamento de agendamentos, as sessões on-line e os recursos de apoio baseados em inteligência artificial. A partir da observação da aplicação, as profissionais forneceram sugestões de melhorias relacionadas à organização visual da interface, à escolha das cores e aos fluxos de navegação e interação entre os usuários.

Entre as contribuições recebidas, destacaram-se sugestões para aprimorar a identidade visual da plataforma e tornar a apresentação das informações durante as sessões on-line mais compreensível. Nesse último aspecto, foram consideradas melhorias na identificação de categorias de expressões faciais em tempo real, utilizando elementos visuais, como cores, acompanhados de informações textuais. As sugestões recebidas foram consideradas durante o desenvolvimento e incorporadas à aplicação.

Ressalta-se que a participação das profissionais ocorreu por meio de demonstrações e fornecimento de sugestões, não tendo sido conduzida como uma avaliação formal de usabilidade ou validação clínica. Não foram aplicados questionários, protocolos estruturados de avaliação ou procedimentos experimentais com pacientes. Dessa forma, essa participação teve caráter consultivo, contribuindo para o aperfeiçoamento da interface e dos fluxos da plataforma, sem constituir evidência de eficácia clínica ou de validação profissional dos resultados produzidos pelos recursos de inteligência artificial.

6.4 Limitações do Trabalho

Apesar dos resultados alcançados, mapearam-se restrições técnicas e metodológicas ao longo do desenvolvimento da plataforma.

Em primeiro lugar, o modelo de visão computacional exibe dependência direta da qualidade das imagens capturadas durante as sessões. Consequentemente, verifica-se uma redução na confiabilidade das inferências em cenários com baixa iluminação, oclusões faciais parciais ou posicionamento inadequado da câmera.

Ademais, os dados produzidos pelos modelos de inteligência artificial possuem caráter exclusivamente consultivo e complementar. Desse modo, tais métricas não devem ser interpretadas como instrumentos de diagnóstico clínico, tampouco como substitutos da avaliação soberana realizada pelos profissionais de saúde mental.

Por fim, aponta-se a ausência de validações experimentais com grupos expandidos de pacientes e terapeutas sob cenários reais de atendimento. A execução dessa etapa permitiria a coleta de métricas robustas acerca da usabilidade e da efetividade prática da solução desenvolvida.

6.5 Trabalhos Futuros

Como continuidade desta pesquisa, diversas vertentes de evolução tecnológica e científica podem ser exploradas.

No âmbito do desenvolvimento de *software*, destacam-se a construção de aplicações móveis nativas, a migração do ecossistema para ambientes de produção em nuvem (*cloud computing*), a implementação de mecanismos avançados de escalabilidade horizontal e a integração com um *gateway* de pagamento especializado para automatizar a cobrança e o repasse de valores das consultas de forma segura. Além disso, pretende-se realizar testes de simultaneidade para avaliar o comportamento da plataforma diante de múltiplos usuários acessando e utilizando seus recursos ao mesmo tempo, permitindo identificar possíveis gargalos de desempenho e oportunidades de otimização.

Em relação à segurança, propõe-se a implantação da comunicação por meio do protocolo HTTPS, com utilização de TLS, especialmente para a transmissão das mensagens privadas trocadas entre pacientes e profissionais. Essa melhoria permitirá adicionar uma camada de proteção aos dados durante o tráfego entre cliente e servidor, complementando a criptografia AES-256 já implementada no conteúdo das mensagens.

Sob a ótica da inteligência artificial, projeta-se a incorporação de modelos multimodais, capazes de correlacionar simultaneamente dados extraídos dos relatos textuais, das expressões faciais e de sinais de voz para a obtenção de análises emocionais mais

abrangentes.

Por fim, pretende-se conduzir ensaios de usabilidade e avaliações práticas estruturadas com usuários reais e profissionais da Psicologia. Esse direcionamento viabilizará avaliar a experiência de utilização da plataforma no contexto do acompanhamento psicológico remoto, além de identificar novos requisitos para o aprimoramento contínuo da solução.

Referências

ALURA. *Back-End: o que é e um guia para iniciar na área*. 2024. Disponível em: <https://www.alura.com.br/artigos/backend>. Acesso em: 29 abr. 2026. Citado na página 28.

AMAZON WEB SERVICES. *Filas de mensagens*. 2026. Disponível em: <https://aws.amazon.com/pt/message-queue/>. Acesso em: 22 abr. 2026. Citado na página 43.

AMAZON WEB SERVICES. *Formato de arquivo COCO*. 2026. Disponível em: <https://docs.aws.amazon.com/pt_br/rekognition/latest/customlabels-dg/md-coco-overview.html>. Acesso em: 22 mai. 2026. Citado na página 40.

ANDERSSON, G.; TITOV, N. Advantages and limitations of internet-based interventions for common mental disorders. *World Psychiatry*, v. 13, n. 1, p. 4–11, 2014. Disponível em: <https://doi.org/10.1002/wps.20083>. Citado 2 vezes nas páginas 25 e 26.

BALARABE, T. *What is Ollama: Running Large Language Models Locally*. 2025. Disponível em: <https://medium.com/@tahirbalarabe2/what-is-ollama-running-large-language-models-locally-e917ca40defe>. Acesso em: 16 abr. 2026. Citado na página 37.

BECK, K. et al. *Manifesto para o Desenvolvimento Ágil de Software*. 2001. Disponível em: <https://agilemanifesto.org/iso/ptbr/manifesto.html>. Acesso em: 4 abr. 2026. Citado na página 47.

BEVANS, D. *Asynchronous vs. Synchronous Programming: Key Similarities and Differences*. 2025. Disponível em: <https://www.mendix.com/blog/asynchronous-vs-synchronous-programming/>. Acesso em: 22 abr. 2026. Citado na página 42.

BRADLEY, J.; SAKIMURA, N.; JONES, M. B. *JSON Web Token (JWT)*. [S.l.], 2015. Disponível em: <https://datatracker.ietf.org/doc/html/rfc7519>. Citado na página 46.

Brasil. *Lei nº 13.709, de 14 de agosto de 2018: Lei Geral de Proteção de Dados Pessoais (LGPD)*. 2018. Dispõe sobre o tratamento de dados pessoais, inclusive nos meios digitais. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm>. Acesso em: 29 out. 2026. Citado na página 46.

BROWN, E. *Web Development with Node & Express: Leveraging the JavaScript Stack*. 2. ed. Sebastopol, CA: O'Reilly Media, Inc., 2019. ISBN 978-1492053484. Citado na página 29.

CAI, Y.; LI, X.; LI, J. Emotion recognition using different sensors, emotion models, methods and datasets: A comprehensive review. *Sensors*, MDPI, v. 23, n. 5, p. 1–33, 2023. Disponível em: <https://doi.org/10.3390/s23052455>. Citado na página 42.

CLEMENT, S. et al. What is the impact of mental health-related stigma on help-seeking? a systematic review of quantitative and qualitative studies. *Psychological Medicine*, v. 45, n. 1, p. 11–27, 2015. Citado na página 22.

CONSELHO FEDERAL DE PSICOLOGIA. *Resolução n° 11,de 11 de maio de 2018*. 2018. Disponível em: <https://leis.org/institucionais/br/cfp/lei/resolucao-do-exercicio-profissional/2018/11/resolucao-do-exercicio-profissional-n-11-2018-regulamenta-a-prestacao-de-servicos-psicologicos-realizados-por-meios-de-tecnologias-da-informacao-e-da-comunicacao-e-revoga-a-resolucao-cfp-n-112012>. Acesso em: 29 out. 2026. Citado na página 26.

CONSELHO FEDERAL DE PSICOLOGIA. *Resolução n° 4,de 26 de março de 2020*. 2020. Disponível em: <https://leis.org/institucionais/br/cfp/lei/resolucao-do-exercicio-profissional/2020/4/resolucao-do-exercicio-profissional-n-4-2020-dispoe-sobre-regulamentacao-de-servicos-psicologicos-prestados-por-meio-de-tecnologia-da-informacao-e-da-comunicacao-durante-a-pandemia-do-covid-19>. Acesso em: 29 out. 2026. Citado na página 26.

CONSELHO FEDERAL DE PSICOLOGIA. *Resolução n° 9, de 18 de julho de 2024*. 2024. Disponível em: <https://leis.org/cfp/lei/resolucao-do-exercicio-profissional/2024/9/resolucao-do-exercicio-profissional-n-9-2024-regulamenta-o-exercicio-profissional-da-psicologia-mediado-por-tecnologias-digitais-da-informacao-e-da-comunicacao-tdics-em-territorio-nacional-e-revoga-as-resolucao-cfp-n-11-de-11-de-maio-de-2018-e-resolucao-cfp-n-04-de-26-de-marco-de>. Acesso em: 29 out. 2026. Citado na página 26.

DAS, D. *Difference between AES and SHA256*. 2023. Disponível em: <https://diptendud.medium.com/difference-between-aes-and-sha256-706d6b2eb2ef>. Acesso em: 24 abr. 2026. Citado na página 45.

DHALIWAL, R. et al. Synchronous web-based psychotherapy for mental disorders from a health quality perspective: Scoping review. *Journal of Medical Internet Research*, v. 25, p. e40710, 2023. Disponível em: <https://www.jmir.org/2023/1/e40710>. Acesso em: 03 nov. 2026. Citado 2 vezes nas páginas 48 e 49.

EITCA Academy. *How do pooling layers help in reducing the dimensionality of the image while retaining important features?* 2023. Disponível em: <https://pt.eitca.org/artificial-intelligence/>. Acesso em: 13 abr. 2026. Citado na página 39.

EVANS, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Boston: Addison-Wesley Professional, 2003. Citado na página 47.

FATIHKGG. *AffectNet YOLO Format*. Kaggle, 2023. Disponível em: <https://www.kaggle.com/datasets/fatihkgg/affectnet-yolo-format>. Acesso em: 03 nov. 2026. Citado na página 40.

FETTE, I.; MELNIKOV, A. *The WebSocket Protocol*. [S.l.], 2011. Disponível em: <https://datatracker.ietf.org/doc/html/rfc6455>. Acesso em: 31 mar. 2026. Citado na página 31.

FIGUEROA, R. *Saiba o que é eHealth e como se aplica o conceito*. 2018. Disponível em: <https://portaltelemedicina.com.br/o-que-e-ehealth>. Acesso em: 10 set. 2026. Citado na página 21.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. Cambridge: MIT Press, 2016. Citado 2 vezes nas páginas 34 e 38.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. MIT Press, 2016. Disponível em: <http://www.deeplearningbook.org>. Citado na página 38.

Google Cloud. *O que é a conteinerização?* s.d. Disponível em: <https://cloud.google.com/discover/what-is-containerization?hl=pt-BR>. Acesso em: 29 abr. 2026. Citado na página 30.

GUO, Z. et al. Large language models for mental health applications: Systematic review. *JMIR Mental Health*, v. 11, p. e57400, 2024. Disponível em: <https://doi.org/10.2196/57400>. Acesso em: 03 nov. 2026. Citado 2 vezes nas páginas 36 e 37.

HARDT, D. *The OAuth 2.0 Authorization Framework*. [S.l.], 2012. Disponível em: <https://www.rfc-editor.org/rfc/rfc6749>. Citado na página 46.

HUA, Y. et al. A scoping review of large language models for generative tasks in mental health care. *npj Digital Medicine*, v. 8, n. 1, p. 230, 2025. Disponível em: <https://doi.org/10.1038/s41746-025-01611-4>. Acesso em: 03 nov. 2026. Citado na página 36.

IBM. *JSON Web Token (JWT)*. 2024. Disponível em: <https://www.ibm.com/docs/pt-br/cics-ts/6.x?topic=cics-json-web-token-jwt>. Acesso em: 19 mai. 2026. Citado na página 63.

IBM. *O que é criptografia?* s.d. Disponível em: <https://www.ibm.com/br-pt/think/topics/encryption>. Acesso em: 24 abr. 2026. Citado na página 43.

IBM. *O que é o Redis?* s.d. Acessado em: 22 de abr. de 2026. Disponível em: <https://www.ibm.com/br-pt/think/topics/redis>. Acesso em: 22 abr. 2026. Citado na página 43.

INDRASIRI, K.; KURUPPU, D. *gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes*. Sebastopol: O'Reilly Media, 2020. ISBN 978-1492058335. Citado na página 33.

ISAAC. 2025. Disponível em: <https://www.guiahardware.es/pt/Requisitos-de-hardware-para-usar-o-Ollama-com-modelos-locais/>. Acesso em: 17 abr. 2026. Citado na página 37.

KANANI, M. *Understanding the Rectified Linear Unit (ReLU): A Key Activation Function in Neural Networks*. 2020. Disponível em: <https://medium.com/@meetkp/28108fba8f07>. Acesso em: 13 abr. 2026. Citado na página 39.

KERLYN. *Uma breve introdução sobre BCrypt*. 2019. Disponível em: <https://medium.com/reprogramabr/uma-breve-introdu%C3%A7%C3%A3o-sobre-bcrypt-f2fad91a7420>. Acesso em: 19 mai. 2026. Citado na página 63.

KOPALIDIS, T. et al. Advances in facial expression recognition: A survey of methods, benchmarks, models, and datasets. *Information*, v. 15, n. 3, p. 135, 2024. Disponível em: <https://doi.org/10.3390/info15030135>. Acesso em: 03 nov. 2026. Citado na página 41.

LNCC - Laboratório Nacional de Computação Científica. *Os quatro pilares da segurança da informação – Confidencialidade, Disponibilidade, Integridade e Autenticidade*. 2024. Disponível em: <https://www.gov.br/lncc/pt-br>. Acesso em: 24 abr. 2026. Citado na página 43.

LOMBARD, A. *WebSocket: Lightweight Client-Server Communications*. Sebastopol: O'Reilly Media, 2015. ISBN 978-1449369279. Citado na página 31.

LORETO, S.; ROMANO, S. P. *Real-Time Communication with WebRTC: Peer-to-Peer in the Browser*. Sebastopol: O'Reilly Media, 2014. ISBN 978-1449371876. Citado na página 32.

MAHMOUD, H.; ABOZARIBA, R. A systematic review on WebRTC for potential applications and challenges beyond audio video streaming. *Multimedia Tools and Applications*, v. 84, n. 1, p. 2909–2946, 2024. Disponível em: <https://doi.org/10.1007/s11042-024-20448-9>. Citado na página 31.

MALINDA, G. *Catching Requests with the Intercepting Filter Pattern: Gatekeeper Vibes*. 2025. Disponível em: <https://medium.com/@pkgmalinda/catching-requests-with-the-intercepting-filter-pattern-gatekeeper-vibes-a1af0a5e08dc>. Acesso em: 23 mai. 2026. Citado na página 63.

MARSHALL, P. et al. Understanding the impacts of online mental health peer support forums: Realist synthesis. *JMIR Mental Health*, v. 11, n. 1, 2024. Disponível em: <https://mental.jmir.org/2024/1/e55750>. Acesso em: 10 set. 2026. Citado na página 21.

MARTIN, R. C. *Design Principles and Design Patterns*. 2000. Disponível via Wayback Machine. Disponível em: <http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf>. Acesso em: 4 abr. 2026. Citado na página 48.

MDN WEB DOCS. *Cookies HTTP*. 2026. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Guides/Cookies>. Acesso em: 1 jun. 2026. Citado na página 63.

MEETRIX. *Meetrix Coturn/Turn Server*. 2026. Disponível em: <https://aws.amazon.com/marketplace/pp/prodview-zrea7eq3c4jbe>. Acesso em: 05 ago. 2026. Citado na página 32.

MERKEL, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, Belltown Media, v. 2, n. 239, p. 2, 2014. ISSN 1075-3583. Disponível em: <https://www.seltzer.com/margo/teaching/CS508.19/papers/merkel14.pdf>. Citado na página 30.

META AI. *Introducing Meta Llama 3: The most capable openly available LLM to date*. 2024. Disponível em: <https://ai.meta.com/blog/meta-llama-3/>. Acesso em: 16 abr. 2026. Citado 2 vezes nas páginas 37 e 38.

MICROSOFT. *Design e implementação de eventos de domínio*. 2026. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/domain-events-design-implementation>. Acesso em: 21 mai. 2026. Citado na página 60.

MORAIS, P. *A História do React*. 2021. Disponível em: <https://medium.com/@pptern unes/a-hist%C3%B3ria-do-react-ba346c416fe1>. Acesso em: 29 abr. 2026. Citado na página 28.

MYERS, D. G.; DEWALL, C. N. *Psychology*. 11. ed. [S.l.]: WORTH, 2015. Citado na página 24.

OLLAMA. *Llama 3: 8B – Meta Llama 3 is the most capable openly available LLM to date*. s.d. Disponível em: <https://ollama.com/library/llama3:8b>. Acesso em: 21 mai. 2026. Citado na página 61.

PAAR, C.; PELZL, J. *Understanding Cryptography: A Textbook for Students and Practitioners*. Berlin, Heidelberg: Springer Science & Business Media, 2010. 116 p. ISBN 978-3-642-04100-6. Disponível em: <http://ndl.ethernet.edu.et/bitstream/123456789/89369/1/Understanding%20cryptography%20a%20textbook%20for%20students%20and%20practitioners%20by%20Christof%20Paar%2C%20Jan%20Pelzl.pdf>. Citado na página 45.

PEREIRA, C. R. *Node.js: Aplicações Web real-time com Node.js*. 1. ed. São Paulo: Casa do Código, 2014. ISBN 978-8566250930. Acesso em: 16 fev. 2025. Citado na página 29.

PHILIPPE, T. J. et al. Digital health interventions for delivery of mental health care: Systematic and comprehensive meta-review. *JMIR Mental Health*, v. 9, n. 5, p. e35159, 2022. Disponível em: <https://mental.jmir.org/2022/5/e35159>. Acesso em: 03 nov. 2026. Citado 2 vezes nas páginas 48 e 49.

PRETORIUS, C.; CHAMBERS, D.; COYLE, D. Young people's online help-seeking and mental health difficulties: Systematic narrative review. *Journal of Medical Internet Research*, v. 21, n. 11, p. e13873, 2019. Disponível em: <https://www.jmir.org/2019/11/e13873/>. Citado na página 22.

PROVOS, N.; MAZIÈRES, D. A future-adaptive password scheme. In: *Proceedings of the 1999 USENIX Annual Technical Conference*. Monterey, CA: USENIX Association, 1999. Disponível em: <https://www.usenix.org/legacy/event/usenix99/provos/provos.pdf>. Citado na página 45.

PÉREZ-HERNÁNDEZ, D. et al. A web-based telepsychology platform prototype using cloud computing and deep learning tools. In: *Progress in Artificial Intelligence*. [S.l.]: Springer, 2021. (Lecture Notes in Computer Science, v. 12981), p. 97–108. Citado na página 49.

RAFAEL. *Resolvendo problemas de performance com Redis e Bull*. 2024. Disponível em: <https://dev.to/3x_pay/resolvendo-problemas-de-performance-com-redis-e-bull-8a3>. Acesso em: 22 abr. 2026. Citado na página 43.

REACT. *Context*. 2026. Disponível em: <https://pt-br.legacy.reactjs.org/docs/context.html>. Acesso em: 9 jun. 2026. Citado na página 60.

REINMAN, A. *Nodemailer – Send emails from Node.js*. 2026. Disponível em: <https://nodemailer.com/>. Acesso em: 26 mai. 2026. Citado na página 64.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 4. ed. Hoboken: Pearson, 2020. Citado na página 33.

SHANNON, C. E. Communication theory of secrecy systems. *Bell System Technical Journal*, Wiley Online Library, v. 28, n. 4, p. 656–715, 1949. Disponível em: <https://doi.org/10.1002/j.1538-7305.1949.tb00928.x>. Citado na página 45.

SPADINI, A. S. *Deep Learning: o guia completo para iniciantes*. 2024. Alura. Disponível em: <https://www.alura.com.br/artigos/deep-learning>. Acesso em: 9 abr. 2026. Citado na página 34.

STALLINGS, W. *Cryptography and Network Security: Principles and Practice*. 7. ed. Boston: Pearson, 2016. Global Edition. ISBN 978-1292158587. Citado 2 vezes nas páginas 43 e 44.

SYMFONY DOCS. *The EventDispatcher Component*. s.d. Disponível em: <https://symfony.com/doc/current/components/event_dispatcher.html>. Acesso em: 8 jun. 2026. Citado na página 99.

SZELISKI, R. *Computer Vision: Algorithms and Applications*. 2. ed. Washington: Springer Nature, 2022. Citado na página 38.

TOTVS. *Front end: O que é, como funciona e qual a importância*. 2021. Disponível em: <https://www.totvs.com/blog/developers/front-end/>. Acesso em: 29 abr. 2026. Citado na página 27.

TWILIO. *Twilio – Customer Engagement Platform*. 2026. Disponível em: <https://www.twilio.com/en-us>. Acesso em: 26 mai. 2026. Citado na página 64.

ULTRALYTICS. *Apresentação do Ultralytics YOLOv8*. 2023. Disponível em: <https://www.ultralytics.com/pt/blog/introducing-ultralytics-yolov8#yolov5-vs-yolov8>. Acesso em: 15 abr. 2026. Citado na página 40.

ULTRALYTICS. *Rede Neural Convolucional (CNN)*. 2024. Disponível em: <https://www.ultralytics.com/pt/glossary/convolutional-neural-network-cnn>. Acesso em: 12 abr. 2026. Citado na página 39.

VASWANI, A. et al. Attention is all you need. *Advances in neural information processing systems*, v. 30, p. 1–11, 2017. Disponível em: <https://scispace.com/pdf/attention-is-all-you-need-1hodz0wcqb.pdf>. Citado na página 35.

W3SCHOOLS. *Introduction to XML*. 2026. Disponível em: <https://www.w3schools.com/xml/xml_whatis.asp>. Acesso em: 22 mai. 2026. Citado na página 40.

World Health Organization. *Mental disorders*. 2025. Disponível em: <https://www.who.int/news-room/fact-sheets/detail/mental-disorders>. Acesso em: 10 set. 2026. Citado 2 vezes nas páginas 21 e 24.

YASEEN, M. What is YOLOv8: An in-depth exploration of the internal features of the next-generation object detector. *arXiv*, 2024. Disponível em: <https://doi.org/10.48550/arXiv.2408.15857>. Citado na página 40.

YOSHI. Exploring handlers in programming: A friendly dive. *DEV Community*, 2024. Disponível em: <https://dev.to/yo-shi/exploring-handlers-in-programming-a-friendly-dive-40d>. Acesso em: 8 jun. 2026. Citado na página 98.