CENTRO PAULA SOUZA
ETEC PHILADELPHO GOUVÊA NETTO
Técnico em Eletroeletrônica

Claudinei Geraldo Ferreira Domingos
Fellype Etecheber de Almeida
Gabriel Silverio de Oliveira
José Rodrigo Lopes
Julio Cesar de Paula

ENDOSCOTECH

Automação de endoscopia veterinária

São José do Rio Preto
2026

Claudinei Geraldo Ferreira Domingos

Fellype Etecheber de Almeida

Gabriel Silverio de Oliveira

José Rodrigo Lopes

Julio Cesar de Paula

ENDOSCOTECH

Automação de endoscopia veterinária

Trabalho de Conclusão de Curso apresentado ao Curso Técnico em Eletroeletrônica da Etec Philadelpho Gouvêa Netto, orientado pelo prof. Mario Kenji Tamura, como requisito parcial para obtenção do título de técnico em Eletroeletrônica.

São José do Rio Preto

2026

Dedicatória

A todos os professores do Curso Técnico em Eletroeletrônica, que foram tão importantes na nossa vida acadêmica e no desenvolvimento deste trabalho de conclusão.

Agradecimentos

Somos gratos às nossas famílias pelo apoio que sempre nos deram durante toda a jornada do curso.

Deixamos um agradecimento especial ao Prof. Fernando Faitarone Brasilino que, a partir do problema de mercado por nós identificado, nos direcionou com a concepção e os caminhos iniciais para estruturar este projeto.

Ao nosso orientador, Prof. Mario Kenji Tamura, pelo incentivo, paciência e pela dedicação do seu tempo na condução deste trabalho de conclusão.

Agradecemos de forma especial ao Dr. David e a toda a equipe da Animed Clínica Veterinária, por terem aberto as portas da instituição, pela confiança depositada no nosso projeto e por disponibilizarem o ambiente operacional e a infraestrutura necessários para a validação piloto do sistema.

Também queremos agradecer à Escola Técnica Estadual Philadelpho Gouvêa Netto, a todos os professores e funcionários pela dedicação e empenho.

EPÍGRAFE

Viver é enfrentar um problema atrás do outro. O modo como você o encara é que faz a diferença.

— Benjamin Franklin

RESUMO

A tecnologia tem desempenhado um papel fundamental na evolução dos serviços de saúde, proporcionando maior eficiência, segurança e organização no gerenciamento de informações médicas. Na medicina veterinária, os exames de imagem são essenciais para o diagnóstico, intervenção e acompanhamento de diversas doenças em animais de pequeno e grande porte. A endoscopia veterinária consolida-se como um procedimento minimamente invasivo de altíssima relevância, que permite visualizar o interior do organismo por meio de um equipamento equipado com óptica avançada e iluminação de precisão. Apesar de sua importância indiscutível, muitas clínicas veterinárias ainda utilizam métodos arcaicos e pouco automatizados para capturar, armazenar e gerenciar as imagens obtidas durante os exames, gerando gargalos operacionais e riscos de perda de dados. Este trabalho apresenta o desenvolvimento técnico e estrutural de um sistema denominado Endoscotech, voltado para automação hospitalar em clínicas veterinárias, com foco na captura acionada por hardware (via pedal com microcontrolador ESP32) e no gerenciamento de imagens provenientes de exames endoscópicos por meio de um software dedicado. O sistema permite registrar dados do paciente, capturar imagens em tempo real de forma ergonômica, armazenar informações em banco de dados relacional, exportar laudos e comunicar-se com servidores PACS. Com a implementação do sistema, espera-se melhorar substancialmente a organização das informações clínicas, mitigar erros operacionais, elevar a assepsia do procedimento e facilitar o acesso ágil ao histórico médico dos pacientes.

Palavras-chave: Automação Hospitalar. Endoscopia Veterinária. Tecnologia Médica. Sistemas de Informação. ESP32. Python.

ABSTRACT

Technology has played a fundamental role in the evolution of healthcare services, providing greater efficiency, safety, and organization in the management of medical information. In veterinary medicine, imaging exams are essential for the diagnosis and monitoring of various diseases in animals. Veterinary endoscopy is a minimally invasive procedure that allows visualization of the inside of the organism using equipment equipped with a camera and lighting. Despite its importance, many veterinary clinics still use poorly automated methods to store and manage the images obtained during the exams. This work presents the development of a system called Endoscotech, aimed at hospital automation in veterinary clinics, focusing on the hardware-triggered capture (via an ESP32 microcontroller pedal) and management of images from endoscopic exams. The system allows recording patient data, capturing images in real time, storing information in a relational database, exporting reports, and communicating with PACS servers. With the implementation of the system, it is expected to substantially improve the organization of clinical information, reduce operational errors, elevate the asepsis of the procedure, and facilitate access to patients' medical history.

Keywords: Hospital Automation. Veterinary Endoscopy. Medical Technology. Information Systems. ESP32. Python.

SUMÁRIO

1. INTRODUÇÃO

A transformação digital tem impactado e reestruturado significativamente diversas áreas da sociedade contemporânea, incluindo de forma incisiva o setor da saúde. Sistemas informatizados e soluções de automação passaram a desempenhar um papel fundamental não apenas na organização administrativa, mas principalmente no gerenciamento crítico de informações clínicas, contribuindo de maneira direta para melhorar a qualidade dos serviços prestados, a precisão dos diagnósticos e a segurança do paciente. Na medicina veterinária, essa realidade não é diferente. Os exames de diagnóstico por imagem são amplamente utilizados para identificar patologias ocultas, planejar intervenções cirúrgicas, acompanhar tratamentos em tempo real e auxiliar o corpo clínico na tomada de decisões médicas complexas.

Entre esses exames de imagem, destaca-se a endoscopia veterinária, uma técnica que revolucionou a forma como os médicos interagem com a anatomia interna dos animais. A endoscopia permite a visualização direta e em alta definição do interior do organismo (trato gastrointestinal, respiratório, entre outros) sem a necessidade de procedimentos cirúrgicos abertos e invasivos. Entretanto, a despeito do alto nível tecnológico dos endoscópios modernos que contam com processadoras de vídeo potentes e fontes de luz avançadas, muitas clínicas veterinárias ainda enfrentam dificuldades operacionais graves na organização, captura e armazenamento das imagens geradas durante os exames.

A endoscopia veterinária é uma importante ferramenta de diagnóstico, permitindo a visualização interna do organismo animal de forma minimamente invasiva. Entretanto, muitas clínicas ainda enfrentam dificuldades no armazenamento e gerenciamento das imagens geradas durante os exames. Diante dessa necessidade, foi desenvolvido o sistema Endoscotech.

Em muitos casos práticos, essas imagens são capturadas de maneira improvisada, armazenadas manualmente em pastas de computadores sem criptografia ou dispositivos externos (pendrives), dificultando severamente a rastreabilidade, o acesso rápido ao histórico clínico e o compartilhamento de laudos com outros especialistas. Além disso, o ato de capturar a imagem frequentemente exige um funcionário extra apenas para apertar uma tecla, desviando uma pessoa que poderia

estar ajudando mais em outros setores, isso quando não pede que o médico veterinário desvie sua atenção do paciente e do endoscópio para manipular um teclado ou mouse de computador quando a demanda de funcionários é baixa, o que acaba quebrando a cadeia de assepsia e reduzindo a fluidez do exame.

Diante desse cenário de defasagem na infraestrutura de software e interface humana, surge a necessidade imperativa de desenvolver soluções tecnológicas que permitam automatizar e otimizar o processo de captura e armazenamento dessas imagens médicas. A automação hospitalar aplicada à medicina veterinária pode trazer benefícios imensuráveis, como maior segurança e integridade dos dados, extrema rapidez no acesso às informações prévias e, consequentemente, uma melhoria notável no atendimento aos pacientes. Este trabalho propõe o desenvolvimento completo (hardware e software) de um sistema de automação hospitalar voltado para clínicas veterinárias, capaz de capturar imagens de forma ergonômica via pedal eletrônico, armazenar em banco de dados estruturado e organizar o acervo proveniente de exames de endoscopia veterinária.

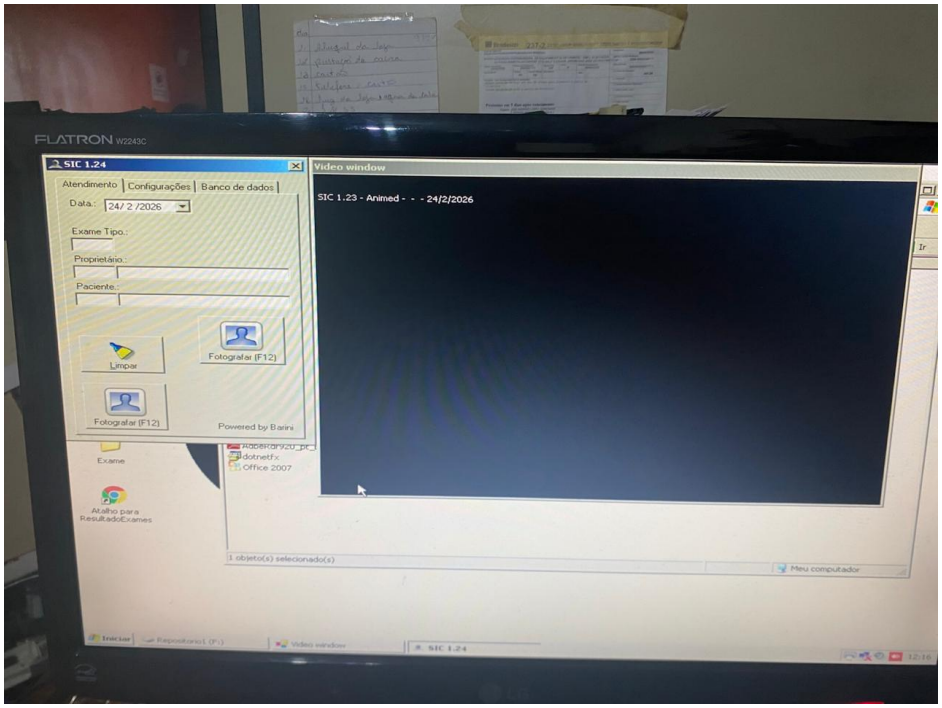
1.1. Justificativa

O constante avanço da engenharia mecatrônica e da tecnologia da informação na área da saúde tem permitido o desenvolvimento de sistemas capazes de melhorar significativamente a gestão de informações clínicas. No entanto, é notório que muitas clínicas e hospitais veterinários, especialmente os de pequeno e médio porte, ainda utilizam métodos tradicionais e softwares obsoletos para registrar exames, emitir laudos e armazenar imagens médicas de alta importância.

A ausência de sistemas automatizados modernos e integrados pode causar uma cascata de problemas administrativos e clínicos, tais como: a perda irrecuperável de dados do paciente, a dificuldade de acesso rápido ao histórico em situações de emergência, a lentidão na emissão de relatórios clínicos padronizados e o risco constante de falhas de segurança da informação. Esses fatores comprometem diretamente a eficiência do atendimento e dificultam o acompanhamento longitudinal da saúde do animal pelo médico veterinário. As Figuras 1 e 2 ilustram as telas dos sistemas antigos encontrados durante o levantamento de requisitos, que muitas vezes

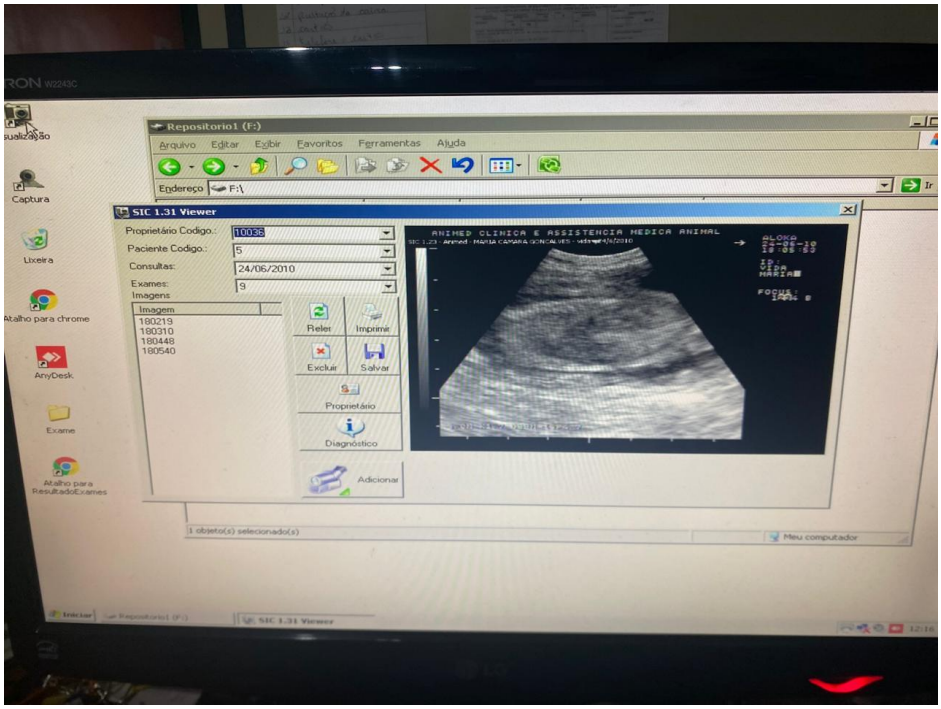
carecem de integração fluida, possuem interfaces confusas, não suportam padrões médicos modernos de comunicação e exigem múltiplos cliques para tarefas simples.

Figura 1: Exemplo da interface do sistema antigo de endoscopia, demonstrando a necessidade de modernização e a limitação de recursos visuais.



Fonte: Do próprio autor, 2026

Figura 2: Visualização de imagens no sistema antigo, evidenciando limitações graves de organização de banco de dados e layout obsoleto.



Fonte: Do próprio autor, 2026

A criação e implementação de um sistema de automação voltado para exames de endoscopia veterinária, aliando o desenvolvimento de software desktop de ponta à automação de hardware (Internet das Coisas - IoT), apresenta-se como uma solução definitiva para esses gargalos. Com um sistema informatizado como o Endoscotech, torna-se possível não apenas organizar melhor as informações clínicas e emitir laudos de forma ágil, mas também introduzir o conceito de acionamento hands-free (sem o uso das mãos) por meio de um pedal microcontrolado, garantindo que o operador mantenha o foco e a assepsia durante o procedimento. O desenvolvimento deste projeto reafirma o papel do técnico em Eletroeletrônica como um agente transformador, capaz de aplicar a tecnologia para resolver problemas reais e alinhar as práticas das clínicas veterinárias às exigências da Indústria 4.0 na saúde.

1.2. Objetivo Geral

Desenvolver, prototipar e validar um sistema de automação hospitalar integrado (envolvendo hardware e software), destinado à captura inteligente, armazenamento

seguro e gerenciamento eficiente de imagens e laudos provenientes de exames de endoscopia na medicina veterinária.

1.3. Objetivos Específicos

Para que o objetivo geral seja alcançado com rigor técnico e científico, delineiam-se os seguintes objetivos específicos:

- Desenvolver a interface e a lógica de um software desktop para o cadastro otimizado de pacientes veterinários e tutores.
- Criar um módulo relacional de banco de dados (SQLite) para o registro histórico e imutável de exames clínicos.
- Implementar um sistema de captura de imagens e vídeos em tempo real a partir da placa de captura do equipamento de endoscopia.
- Desenvolver um hardware de acionamento remoto (pedal) utilizando o microcontrolador ESP32, configurado como dispositivo USB HID (Human Interface Device), permitindo a captura de imagens sem contato manual com o computador.
- Estruturar um módulo para a geração e exportação automática de relatórios clínicos (laudos) em formato PDF.
- Integrar protocolos de comunicação médica (PACS/DICOM) para viabilizar o envio das imagens capturadas para servidores de armazenamento hospitalar remoto.

2. FUNDAMENTAÇÃO TEÓRICA

A elaboração de um sistema de automação médica requer um embasamento teórico sólido que perpasse a história do procedimento alvo, as exigências clínicas da sua aplicação na veterinária e as tecnologias de hardware e software necessárias para a sua modernização.

2.1. História e Evolução da Endoscopia

A evolução do conhecimento das patologias anatômicas do trato digestivo e respiratório ocorreu paralelamente aos avanços nas tecnologias de visualização médica e óptica física. Embora a ideia de visualizar cavidades internas remonte à antiguidade (com evidências de espéculos encontrados em ruínas do Império Romano), a história da endoscopia moderna tem seu marco inicial frequentemente

atribuído ao médico alemão Philipp Bozzini. Em 1806, Bozzini desenvolveu um instrumento rudimentar chamado "Lichtleiter" (condutor de luz), projetado para visualizar o interior do corpo humano utilizando um complexo sistema de espelhos e a luz gerada pela chama de uma vela.

Figura 3: Primeiro Endoscopio (condutor de luz) criado

Fonte: European Museum of Urology. Disponível em: https://history.uroweb.org/history-of-urology/diagnosis/looking-into-the-body/bozzini-and-the-lichtleiter/. Acesso em: 14 jun. 2026.

Embora as limitações técnicas e o desconforto térmico da época tenham impedido seu uso clínico em larga escala, o conceito de visualização interna minimamente invasiva estava definitivamente consolidado.

As inovações seguiram nas décadas posteriores. Em 1868, o médico alemão Adolph Kussmaul aprimorou o conceito ao introduzir o uso de um tubo reto metálico que, auxiliado por uma fonte de luz externa, permitiu as primeiras visualizações efetivas do estômago. O avanço tecnológico crucial e a verdadeira fundação da cistoscopia e

endoscopia operatória ocorreram no final do século XIX. Em 1879 (algumas fontes citam 1880), o médico alemão Maximilian Nitze, colaborando com fabricantes de instrumentos ópticos, desenvolveu o primeiro endoscópio equipado com um sistema de lentes de aumento e iluminação elétrica incandescente interna, permitindo uma visualização substancialmente mais clara e detalhada.

A verdadeira revolução óptica e a superação dos perigosos e desconfortáveis tubos rígidos, contudo, advieram em meados do século XX. Na década de 1950, o físico Harold Hopkins inventou um novo e engenhoso sistema de lentes cilíndricas de vidro (as "rod lenses") e consolidou a transmissão de luz fria através de feixes de fibras ópticas flexíveis. Essa tecnologia solucionou o problema do aquecimento interno, proporcionou imagens de alto brilho e abriu caminho para a criação dos primeiros fibroscópios totalmente flexíveis. Mais adiante, na década de 1980, com a introdução dos sensores digitais CCD (Charge-Coupled Device) na extremidade distal do aparelho, a visão ocular direta no endoscópio foi substituída pela transmissão eletrônica de imagens de alta resolução para monitores de vídeo, inaugurando a era moderna da vídeo-endoscopia.

2.2. A Endoscopia na Medicina Veterinária

A inserção e adaptação da endoscopia na medicina veterinária acompanhou de perto os saltos tecnológicos da medicina humana. Durante a década de 1970, médicos veterinários pioneiros começaram a adaptar os endoscópios de uso humano para examinar os tratos gastrointestinais e respiratórios de animais de companhia e de grande porte.

Figura 4: Fibroscópio de 1960, um dos primeiros modelos a ser adaptado para uso veterinário

Fonte: Case Western Reserve University. Disponível em: https://artsci.case.edu/dittrick/online-exhibits/explore-the-artifacts/hirschowitz-fiberoptic-endoscope-1960/. Acesso em 14 jun. 2026.

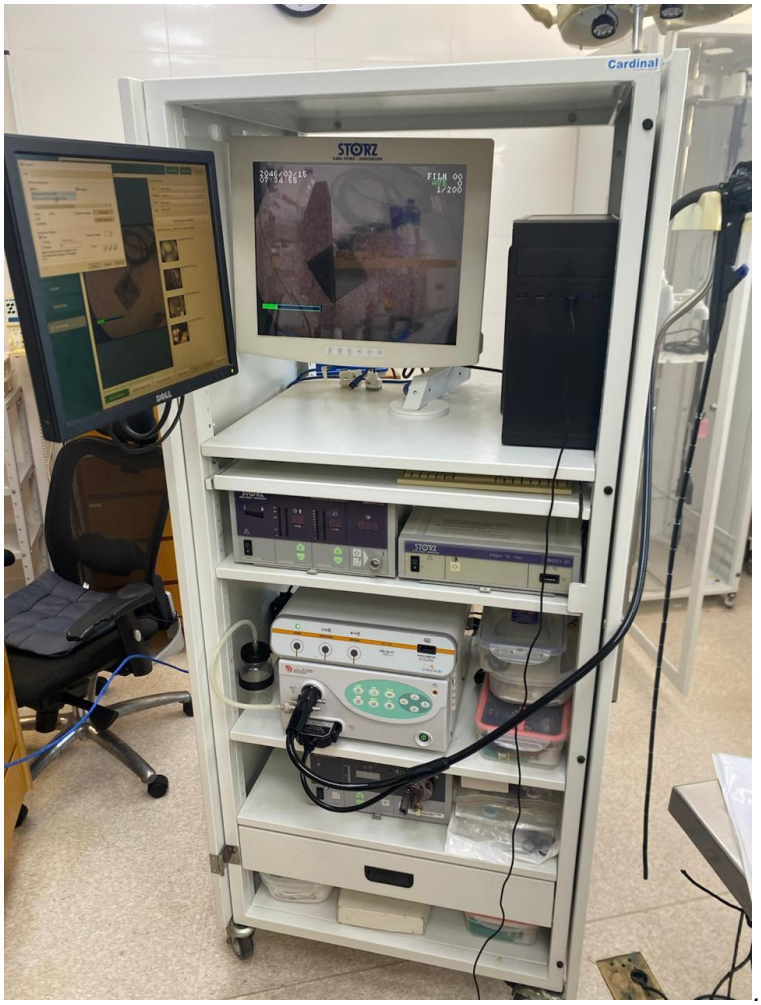
Com o avanço tecnológico e o barateamento da tecnologia de captura de vídeo digital na década de 1980, a técnica rapidamente se difundiu e se popularizou, tornando-se um serviço indispensável em hospitais e clínicas veterinárias globais.

Atualmente, a endoscopia veterinária destaca-se primariamente por ser uma abordagem diagnóstica e cirúrgica minimamente invasiva.[1] A não necessidade de realizar grandes incisões abdominais ou torácicas (laparotomias ou toracotomias exploratórias) resulta em um trauma significativamente menor aos tecidos do animal.[1] Esta abordagem cirúrgica conservadora reduz drasticamente a ocorrência de dor pós-operatória severa, diminui os riscos biológicos de infecção secundária e acelera de maneira considerável a recuperação clínica, permitindo, muitas vezes, que o paciente animal retorne ao domicílio no mesmo dia do procedimento.

Trata-se do método de eleição padrão-ouro para intervenções abdominais menos invasivas, coleta precisa de fragmentos de biópsia para análise histopatológica de mucosas e, crucialmente, para procedimentos terapêuticos de emergência.[1] Dentre as emergências mais comuns na clínica de pequenos animais, destaca-se a retirada de corpos estranhos (como brinquedos, ossos, pedaços de tecido) alojados no

esôfago ou no estômago de cães e gatos. Utilizando pinças de apreensão e laços passados pelo canal de trabalho do endoscópio, o médico veterinário pode remover o objeto obstrutivo de forma rápida e segura, evitando cirurgias abertas altamente complexas e de risco elevado. A Figura 3 exibe a torre de um equipamento endoscópico em operação padrão, composta pelos módulos essenciais para o exame.

Figura 5: Torre de equipamento endoscópico em operação clínica, composta por monitor, processadora de imagem digital, fonte de luz e computador de captura.



Fonte Do próprio autor, 2026

2.3. Automação Hospitalar e a Industria 4.0 na Saúde

A automação hospitalar consiste na utilização sinérgica de sistemas tecnológicos (hardware e software) para automatizar, monitorar e otimizar processos administrativos, logísticos e clínicos dentro de instituições de saúde. Esses sistemas são os pilares da saúde digital, responsáveis por gerenciar o ciclo de vida da

informação: desde o cadastro inicial do paciente, passando pelo registro de sinais vitais e exames complexos, até o diagnóstico, tratamento e faturamento.

A inserção dos preceitos da Indústria 4.0 (que engloba a Internet das Coisas - IoT, o Big Data, a Computação em Nuvem e a Inteligência Artificial) no ambiente médico busca transformar clínicas tradicionais em "Clínicas Inteligentes" (Smart Clinics). A utilização de sistemas informatizados integrados permite eliminar redundâncias, garantir a integridade dos prontuários médicos, reduzir drasticamente os erros humanos decorrentes de transcrições manuais e aumentar a eficiência operacional geral das instituições. No escopo deste projeto, a automação aplica-se diretamente na ponte entre a máquina (o endoscópio) e o banco de dados do paciente, eliminando a manipulação ineficiente de arquivos.

2.4. Endoscopia Veterinária e Procedimentos Diagnósticos

Como detalhado anteriormente, a endoscopia veterinária é um procedimento investigativo em tempo real que utiliza um instrumento mecatrônico longo e flexível chamado endoscópio, composto por uma microcâmera digital na extremidade distal, feixes de iluminação óptica e canais ocos de trabalho. Esse equipamento permite não apenas visualizar as estruturas internas, mas interagir com elas. Entre as principais aplicações da endoscopia veterinária estão o diagnóstico de doenças inflamatórias gastrointestinais crônicas, a identificação de úlceras, a avaliação do trato respiratório (broncoscopia para investigar colapsos de traqueia ou massas pulmonares), a remoção de corpos estranhos e a coleta de amostras de tecido suspeito para biópsia oncológica. Para que o diagnóstico seja assertivo, é fundamental que as imagens capturadas durante a navegação interna sejam de alta qualidade e devidamente atreladas ao prontuário do paciente correto.

2.5. Sistemas de Captura de Imagem e Integração PACS

Sistemas de captura de imagem são as pontes de software utilizadas para registrar, digitalizar e armazenar as imagens em movimento provenientes das saídas de vídeo analógicas ou digitais dos equipamentos médicos (como as processadoras de vídeo da torre de endoscopia). Esses sistemas convertem o fluxo de vídeo (frames) em

arquivos estáticos (como JPEG ou PNG) ou em clipes de vídeo (MP4) e organizam essa mídia visual dentro de um banco de dados estruturado, vinculando-a inequivocamente à identidade do paciente.

Na área da informática em saúde (Health IT), a padronização e o compartilhamento dessas imagens em rede são governados pelo padrão internacional DICOM (Digital Imaging and Communications in Medicine). Os sistemas modernos frequentemente integram-se aos servidores PACS (Picture Archiving and Communication System). Um sistema PACS é uma tecnologia de arquivamento de imagens médicas que provê armazenamento econômico e acesso conveniente a imagens de múltiplas modalidades (ultrassom, raio-X, endoscopia, ressonância magnética) de forma centralizada. Ao integrar a comunicação PACS no projeto Endoscotech, garante-se que a clínica veterinária não mantenha os exames presos em um único computador local, mas possa enviar (via protocolo de rede) os exames padronizados para o servidor central do hospital, permitindo que especialistas em diferentes salas acessem e avaliem as imagens instantaneamente.

3. METODOLOGIA

O presente Trabalho de Conclusão de Curso classifica-se quanto à sua natureza como uma pesquisa aplicada, de abordagem quali-quantitativa e com objetivos metodológicos exploratórios e experimentais, focada no desenvolvimento tecnológico de um sistema computacional ciber-físico (integração entre o hardware de chão de fábrica/clínica e o software de gestão).

3.1. Classificação da Pesquisa

A pesquisa aplicada caracteriza-se por gerar conhecimentos para a aplicação prática dirigida à solução de problemas específicos. Neste caso, o problema prático resolvido é a deficiência e a lentidão no arquivamento e captura de exames de endoscopia em clínicas veterinárias. O aspecto exploratório se deu pelo levantamento das necessidades reais dos médicos veterinários através de entrevistas informais e observação do fluxo de trabalho das clínicas, a fim de levantar os requisitos funcionais e não funcionais do software e do hardware.

3.2. Fases do Desenvolvimento

As etapas sequenciais do projeto foram delineadas para garantir um ciclo de vida de desenvolvimento de sistemas estruturado:

3.2.1. Levantamento e Análise de Requisitos: A partir de entrevistas e observações em campo junto à equipe clínica da unidade piloto, identificou-se a necessidade de um sistema integrado de captura que reduzisse o tempo de resposta e garantisse a ergonomia no manuseio dos equipamentos médico-veterinários. O entendimento do fluxo do exame endoscópico, os sistemas antigos (interface ruim, falta de atalhos rápidos) e definição dos requisitos do novo sistema.

3.2.2. Pesquisa Bibliográfica e Tecnológica: Estudo das linguagens de programação adequadas para interface gráfica de usuário (GUI) em desktop, processamento de imagem em tempo real, bancos de dados leves (SQLite) e arquitetura de microcontroladores de baixo custo com suporte a barramento USB.

3.2.3. Desenvolvimento do Hardware (IoT): Prototipagem da placa baseada no microcontrolador ESP32 para atuar como dispositivo de entrada humano (HID). Simulação, codificação em C++ e montagem física no invólucro do pedal.

3.2.4. Desenvolvimento do Software (Frontend e Backend): Programação do aplicativo desktop utilizando Python e as bibliotecas PySide6 para a construção das telas e OpenCV para o tratamento do stream de vídeo da placa de captura. Implementação das regras de negócio e CRUD (Create, Read, Update, Delete) do banco de dados.

3.2.5. Testes de Integração e Validação Funcional: Acoplamento do pedal ESP32 com o software rodando em ambiente de teste para validar a latência de captura, a estabilidade do banco de dados ao gerar laudos em PDF e o sucesso da transmissão via PACS para servidores simulados locais.

4. DESENVOLVIMENTO DO SISTEMA

4.1. Materiais, Mão de Obra e Orçamento

4.1.1. Montagem do S*istema de Proteção e Distribuição*

O desenvolvimento prático e os testes estruturais do projeto demandaram não apenas o esforço lógico de programação, mas também a criação física e o acondicionamento de um circuito de tomadas e dispositivos de segurança em caixa apropriada para suportar o ambiente clínico (resistência a impactos e umidade). As Figuras de 4 a 9 ilustram o processo prático de montagem, dimensionamento estrutural físico das tomadas dos dispositivos de proteção.

Figura 6: Etapa de medição técnica rigorosa, planejamento de layout e marcação dos furos no painel/caixa de tomadas para a acomodação e refrigeração do hardware de automação.

Fonte: Do próprio autor, 2026.

Figura 7: Verificação métrica do espaçamento e alinhamento das tomadas de automação no painel de comando, garantindo o acesso seguro e ergonômico para os periféricos.

Fonte: Do próprio autor, 2026.

Figura 8: Processo de furação da caixa elétrica para os parafusos de fixação das tomadas.

Fonte: Do próprio autor, 2026.

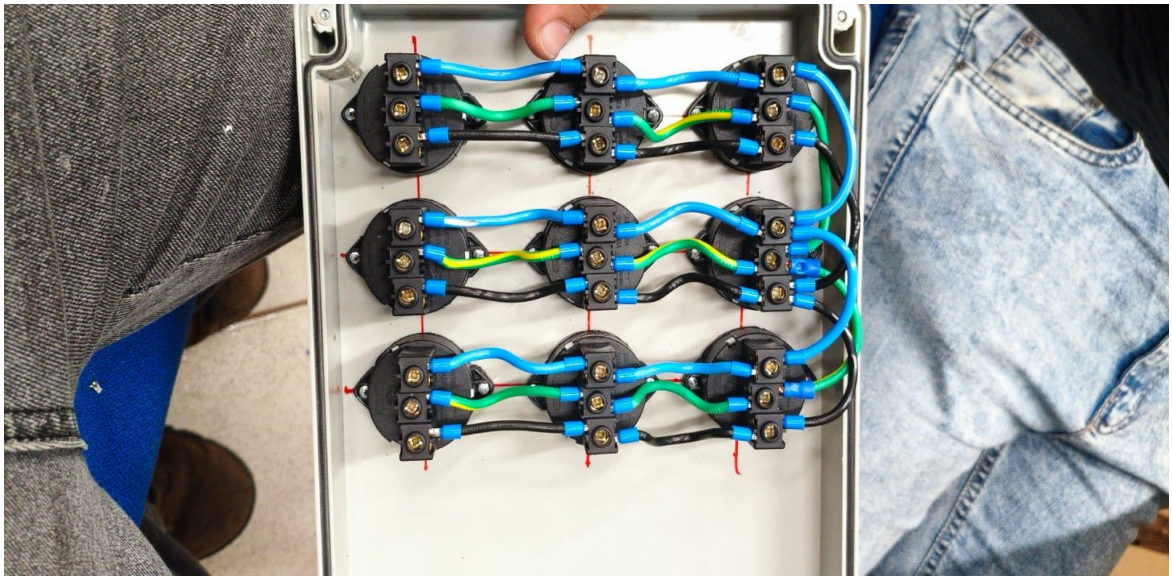
Figura 9: Etapa de fixação das tomadas

Fonte: Do próprio autor, 2026.

Figura 10: Visão frontal do sistema de proteção e distribuição de circuito com as tomadas instaladas e os dispositivos de proteção (IDR e DPS) embutidos nela.

Fonte: Do próprio autor, 2026.

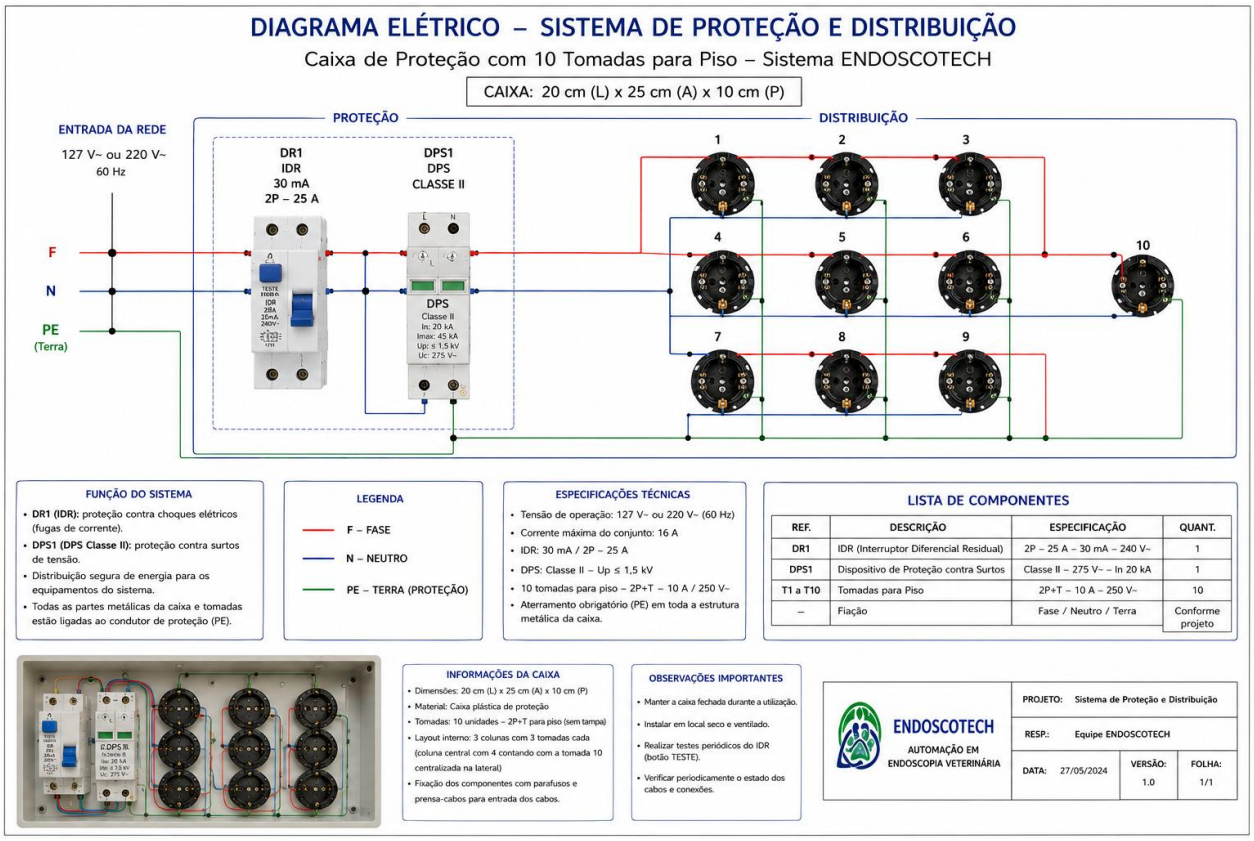
Figura 11: Visão interna das ligações das tomadas.



Fonte: Do próprio autor, 2026.

4.1.1.1. Diagrama Elétrico: Sistema de Proteção

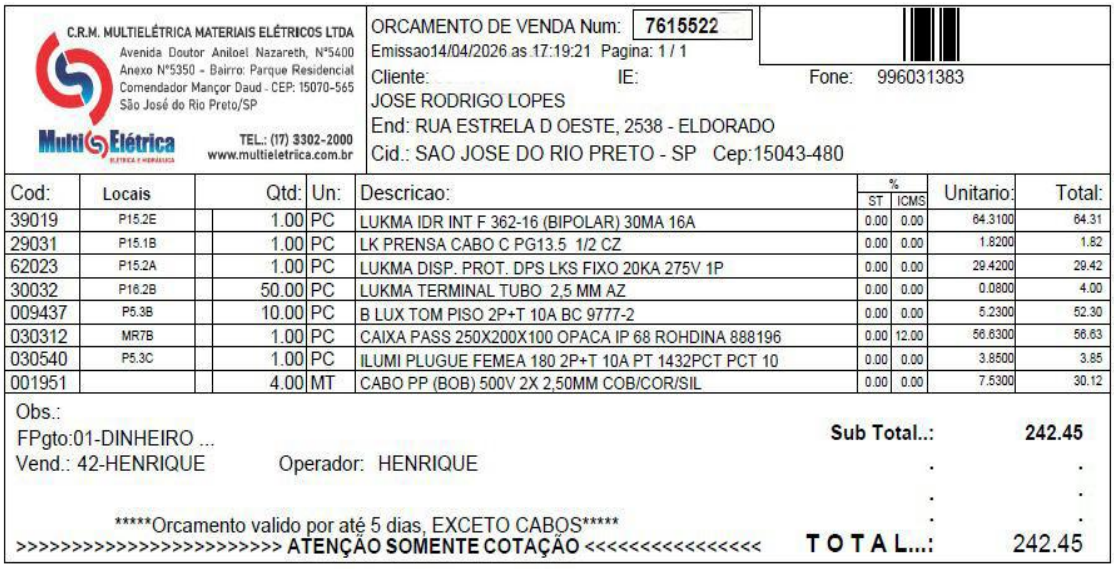
Figura 12: Diagrama Elétrico do sistema de proteção e distribuição



Fonte: Elaborado pelo autor utilizando a inteligência artificial Gemini, 2026.

4.1.2. Levantamento de custo

Figura 13: Orçamento de Materiais



C.R.M. MULTIELÉTRICA MATERIAIS ELÉTRICOS LTDA
Avenida Doutor Aniloel Nazareth, Nº5400
Anexo Nº5350 - Bairro: Parque Residencial
Comendador Mançor Daud - CEP: 15070-565
São José do Rio Preto/SP
TEL.: (17) 3302-2000
www.multieletrica.com.br

ORCAMENTO DE VENDA Num: 7615522
Emissao14/04/2026 as 17:19:21 Pagina: 1 / 1
Cliente: IE: Fone: 996031383
JOSE RODRIGO LOPES
End: RUA ESTRELA D OESTE, 2538 - ELDORADO
Cid.: SAO JOSE DO RIO PRETO - SP Cep:15043-480

Cod:	Locais	Qtd:	Un:	Descricao:	% ST	% ICMS	Unitario:	Total:
39019	P15.2E	1.00	PC	LUKMA IDR INT F 362-16 (BIPOLAR) 30MA 16A	0.00	0.00	64.3100	64.31
29031	P15.1B	1.00	PC	LK PRENSA CABO C PG13.5 1/2 CZ	0.00	0.00	1.8200	1.82
62023	P15.2A	1.00	PC	LUKMA DISP. PROT. DPS LKS FIXO 20KA 275V 1P	0.00	0.00	29.4200	29.42
30032	P16.2B	50.00	PC	LUKMA TERMINAL TUBO 2,5 MM AZ	0.00	0.00	0.0800	4.00
009437	P5.3B	10.00	PC	B LUX TOM PISO 2P+T 10A BC 9777-2	0.00	0.00	5.2300	52.30
030312	MR7B	1.00	PC	CAIXA PASS 250X200X100 OPACA IP 68 ROHDINA 888196	0.00	12.00	56.6300	56.63
030540	P5.3C	1.00	PC	ILUMI PLUGUE FEMEA 180 2P+T 10A PT 1432PCT PCT 10	0.00	0.00	3.8500	3.85
001951		4.00	MT	CABO PP (BOB) 500V 2X 2,50MM COB/COR/SIL	0.00	0.00	7.5300	30.12

Obs.:
FPgto:01-DINHEIRO ...
Vend.: 42-HENRIQUE Operador: HENRIQUE

Sub Total..: 242.45

*****Orcamento valido por até 5 dias, EXCETO CABOS*****
>>>>>>>>>>>>>>>>>>>>>>>> ATENÇÃO SOMENTE COTAÇÃO <<<<<<<<<<<<<<<<<

T O T A L...: 242.45

Fonte: Multielétrica, 2026.

Item	Valor
ESP32-S3	R$ 70,00
Pedal	R$ 70,00
Materiais elétricos	R$ 242,45
Total materiais	**R$ 382,45**
Desenvolvimento, programação e implantação:	**R$ 2.500,00**

O levantamento de custo total resultou em um valor de R$2.882,45 para implementação de um sistema Endoscotech

4.2. Estrutura do Sistema

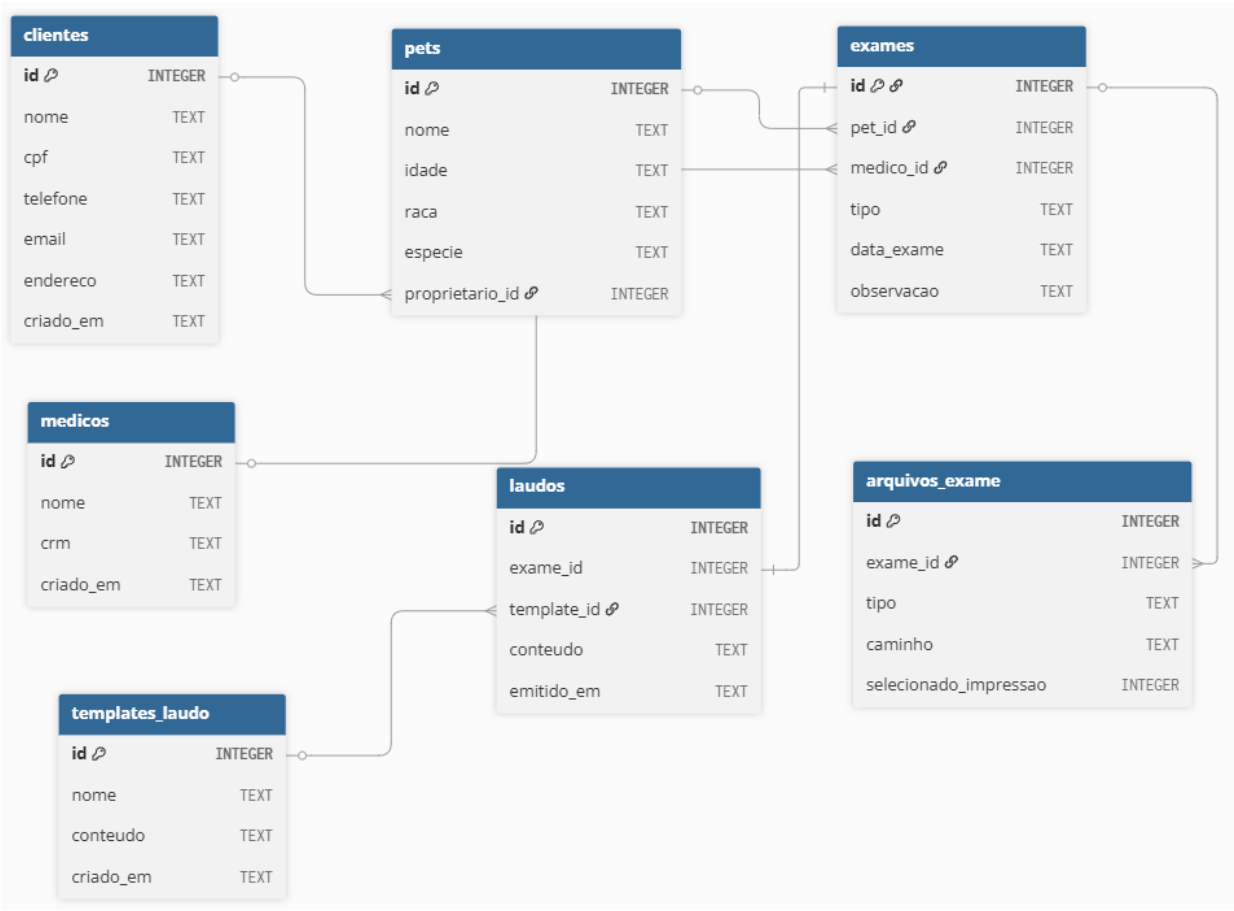
O ecossistema Endoscotech foi projetado sob a premissa do "design centrado no usuário", visando facilitar, acelerar e garantir a segurança do gerenciamento de exames endoscópicos nas clínicas veterinárias, minimizando a curva de aprendizado. As funcionalidades estruturais primárias do software incluem:

- **Módulo de Recepção:** Cadastro rápido e edição de perfis de pacientes e de seus respectivos médicos responsáveis.
- **Módulo de Exames:** Abertura e registro de novas sessões de exames, atreladas a um paciente específico e a uma data gerada pelo sistema.
- **Módulo de Captura em Tempo Real:** Motor de vídeo otimizado que exibe o stream da câmera, permitindo a captura instantânea de fotos (Frames) e a gravação de vídeos em alta definição.
- **Módulo de Laudos (Geração de Relatórios):** Ferramenta integrada de processamento de texto com máscaras/templates customizáveis. Permite ao médico veterinário descrever os achados clínicos, arrastar as imagens capturadas para o corpo do texto e exportar um laudo final em PDF, timbrado e formatado.
    - **Módulo de Exportação PACS:** Envio das imagens para o servidor central da clínica obedecendo as tags DICOM.

4.3. Modelagem do Banco de Dados

Para garantir que as informações médicas não sofram perdas ou corrupção, o sistema descarta planilhas ou pastas soltas e utiliza um banco de dados relacional robusto local (baseado em SQLite) para armazenar e cruzar as entidades do sistema. O esquema lógico armazena as seguintes informações estruturadas:

Figura 14: Imagem das tabelas do Banco de dados



Fonte: Elaborado pelo autor, com base em DBdiagram, 2026.

4.4. Interface do Sistema

A interface de usuário (UI) foi redesenhada do zero, abandonando a estética poluída dos sistemas legados. Desenvolvida com a biblioteca gráfica PySide6 (bindings oficiais do framework Qt para Python), a nova interface foca na usabilidade fluida, utilizando contrastes adequados para ambientes clínicos com baixa luminosidade (tema escuro), botões largos para telas sensíveis ao toque e painéis de metadados autoexplicativos. A Figura 15 apresenta o novo ambiente visual e operacional desenvolvido, demonstrando um design moderno e totalmente adaptado às necessidades de agilidade do operador clínico.

Figura 15: Nova interface do sistema, evidenciando de forma limpa o gerenciamento do paciente atual à direita e o amplo visor de vídeo da câmera do exame à esquerda, ladeado pelo acervo visual capturado

Fonte: Captura de tela do Sistema Endoscotech realizado pelo autor, 2026.

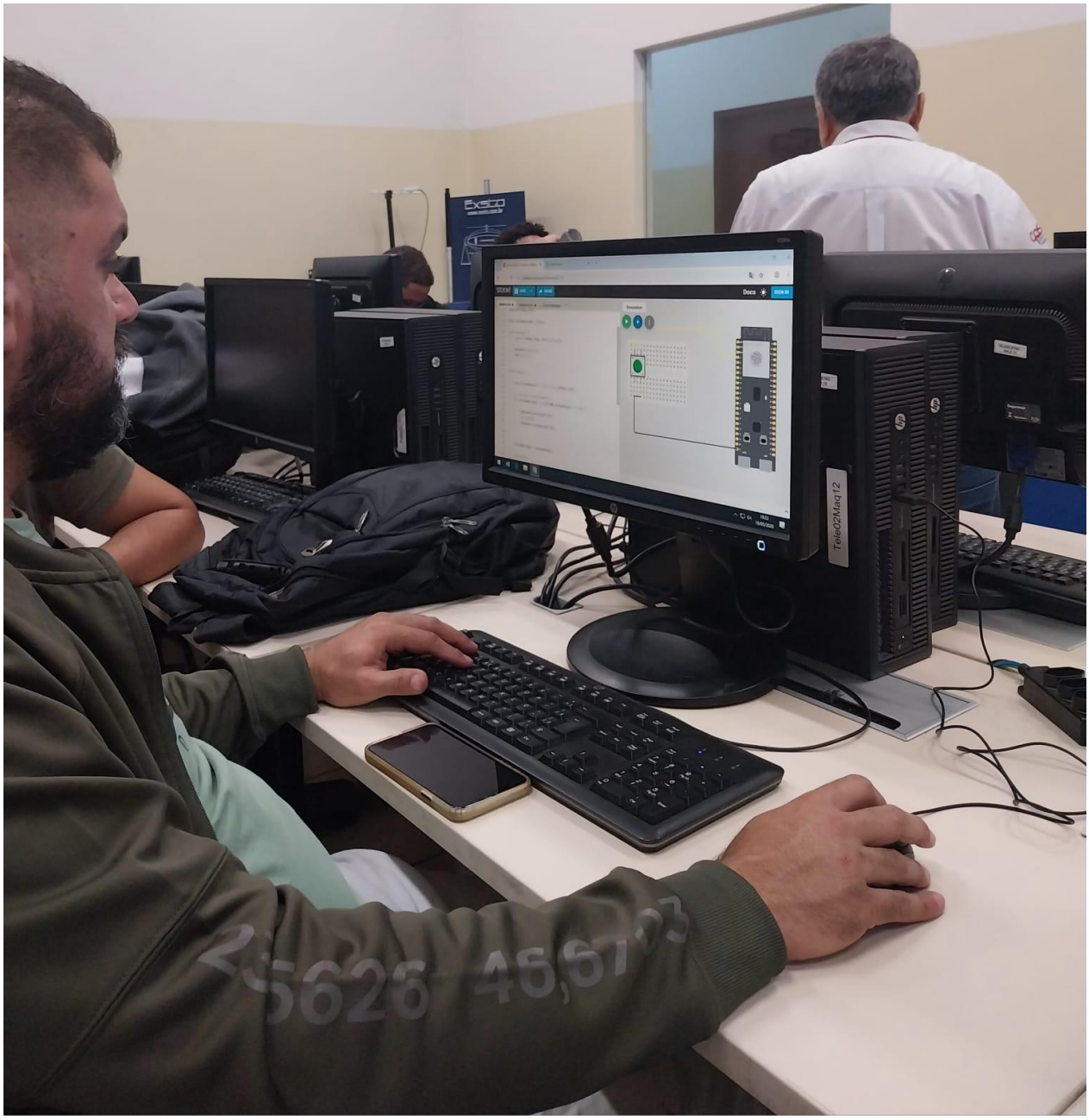
4.5. Arquitetura de Hardware e Funcionamento do ESP32

Para solucionar o problema da captura de imagem e o desvio de um funcionário extra apenas para o apertar de uma tecla, projetou-se um acionador remoto do tipo pedal. O núcleo de processamento deste dispositivo embarcado é o microcontrolador ESP32.

O ESP32, projetado e fabricado pela Espressif Systems, é um System on a Chip (SoC) de altíssima performance e baixo consumo de energia que tem se consolidado como o cérebro preferencial em projetos da Indústria 4.0, Internet das Coisas (IoT) e sistemas embarcados em todo o mundo. Diferente dos antigos microcontroladores de 8 bits, o ESP32 possui uma moderna arquitetura com processador Tensilica Xtensa Dual-Core de 32 bits, capaz de operar a frequências de clock de até 240 MHz. Ele é equipado com uma generosa memória SRAM interna de 520 KB e módulos de memória flash externa que frequentemente chegam a 4 MB ou mais, proporcionando espaço de sobra para o armazenamento do firmware complexo.

O grande diferencial do ESP32 é a sua conectividade sem fio nativa e completa (transceptores Wi-Fi 802.11 b/g/n e Bluetooth v4.2 BR/EDR e BLE integrados no mesmo chip de silício). Esta conectividade abre portas para futuras atualizações de telemetria sem fio do pedal. Ademais, o chip é ricamente dotado de interfaces periféricas de hardware, incluindo portas UART, SPI, I2C, canais de conversão Analógico-Digital (ADC) de alta resolução, conversores Digital-Analógico (DAC) e moduladores PWM. No contexto deste projeto, o ESP32 é programado para atuar como um dispositivo HID (Human Interface Device) sobre o barramento USB, interpretando o sinal elétrico (fechamento de contato seco) gerado pelo acionamento mecânico do pedal e convertendo-o instantaneamente em um comando de teclado reconhecido pelo sistema operacional do computador do hospital, sem a necessidade da instalação de drivers proprietários adicionais. A Figura 16 ilustra a fase de codificação embarcada.

Figura 16: Fase laboratorial de programação, compilação de código e simulação de circuitos lógicos no software embarcado do microcontrolador ESP32 através do site Wokwi



Fonte: Do próprio autor, 2026.

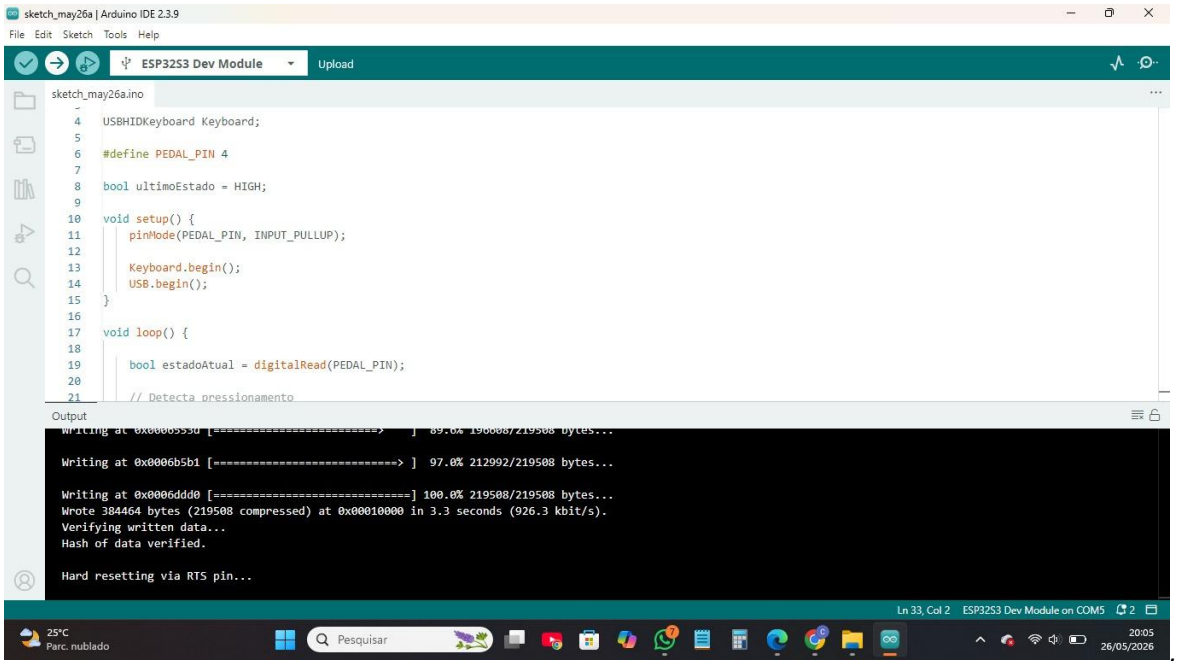
4.5.1 Código do Pedal

O código abaixo, desenvolvido na IDE do Arduino e compilado para a arquitetura Xtensa do ESP32, configura as portas USB do chip para se comportarem exatamente como um teclado convencional de computador. O algoritmo lê constantemente a porta digital PEDAL_PIN 4 vinculada fisicamente ao interruptor do pedal. Ao detectar a borda de descida (transição do pino de nível lógico HIGH para LOW, que indica que o médico pisou no pedal), a biblioteca USBHIDKeyboard simula o envio do *scancode* da tecla F8 para o computador e implementa rotinas de atraso (delay) para realizar o filtro *debounce*, prevenindo que ranhuras mecânicas no contato elétrico sejam lidas pelo sistema como cliques múltiplos acidentais. A tecla F8 foi escolhida como o atalho mestre (hotkey) mapeado no software em Python para realizar a captura da imagem ou iniciar a gravação do vídeo.

```
#include "USB.h"
#include "USBHIDKeyboard.h"

USBHIDKeyboard Keyboard;

#define PEDAL_PIN 4

bool ultimoEstado = HIGH;

void setup() {
  pinMode(PEDAL_PIN, INPUT_PULLUP);

  Keyboard.begin();
  USB.begin();
}
```

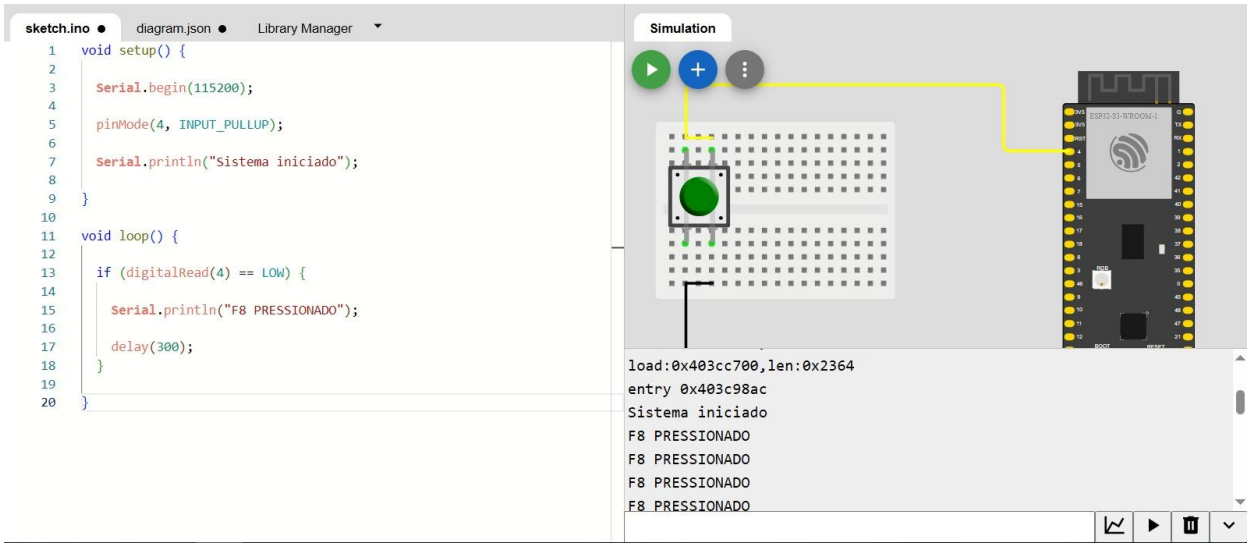
```
void loop() {
  bool estadoAtual = digitalRead(PEDAL_PIN);

  // Detecta o momento exato do clique (transição de HIGH para LOW)
  if (ultimoEstado == HIGH && estadoAtual == LOW) {

    // Pressiona a tecla F8
    Keyboard.press(KEY_F8);

    // Mantém pressionado por 50ms (tempo suficiente para o software registrar)
    delay(50);

    // Solta a tecla F8
    Keyboard.release(KEY_F8);

    // Pequeno delay extra de debounce para evitar duplo clique mecânico do pedal
    delay(50);
  }

  ultimoEstado = estadoAtual;

  delay(10);
}
```

Figura 17: 1ª Compilação bem-sucedida do Código e upload para o Esp32



Fonte: Arduino IDE, 26 mai. 2026.

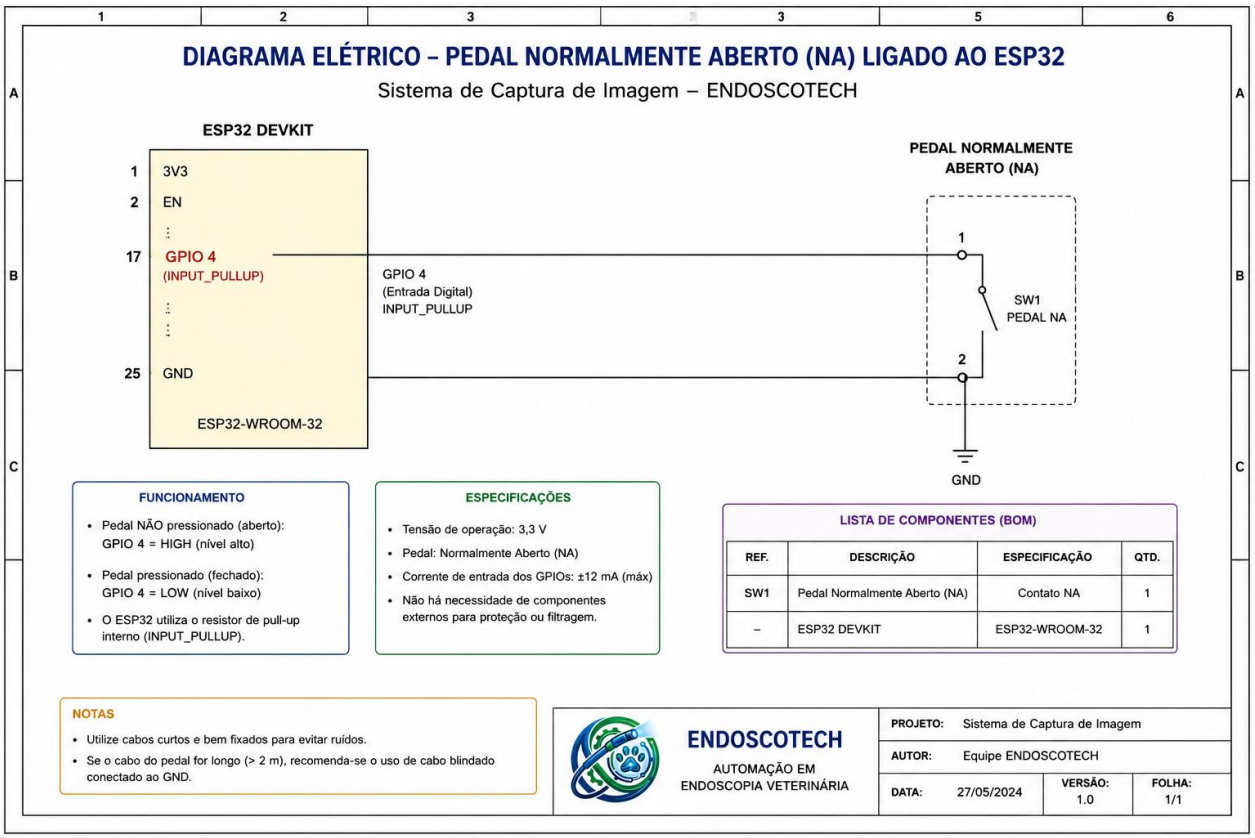
Figura 18: Foto do código para simulação no site Wokwi.



Fonte: Wokwi, 26 mai. 2026.

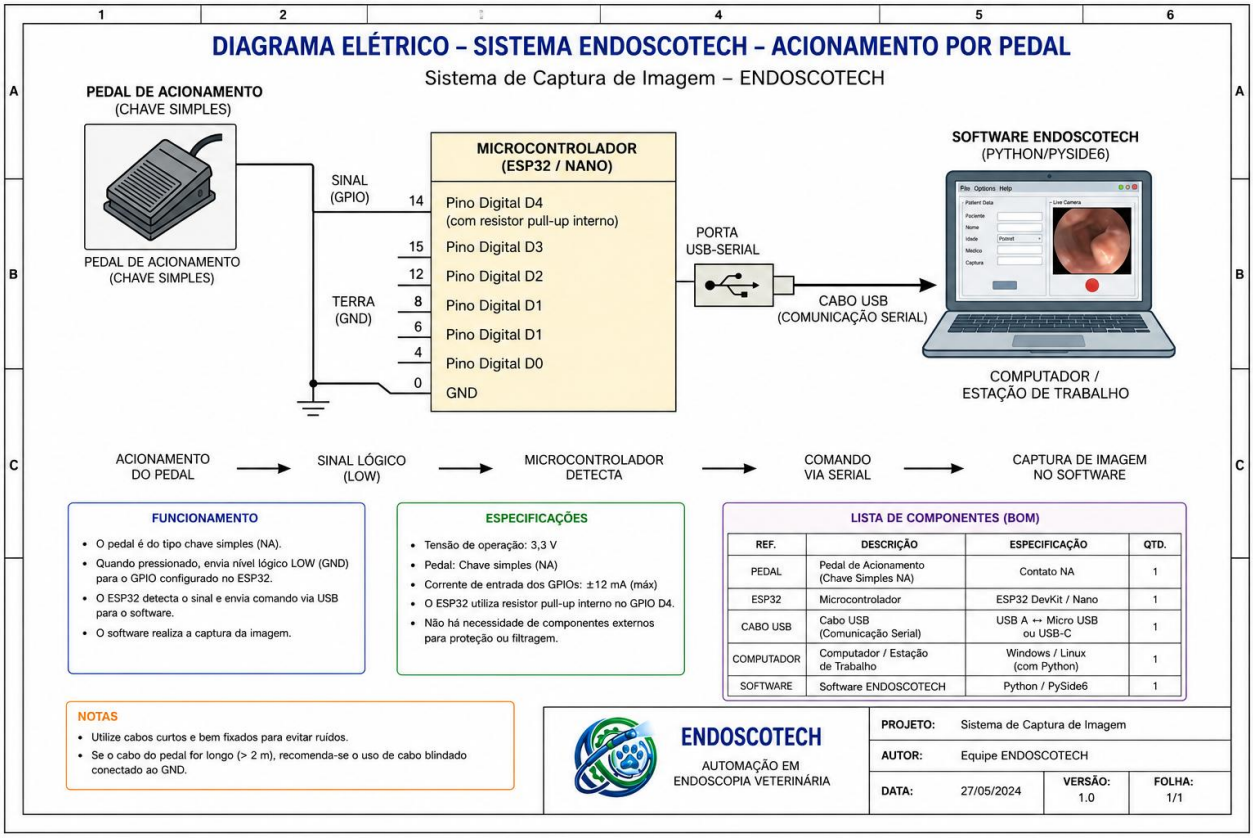
4.5.1.1. Diagrama Elétrico: Pedal

Figura 19: Diagrama elétrico de acionamento do pedal.



.Fonte: Elaborado pelo autor utilizando a inteligência artificial Gemini, 2026.

Figura 20: Esquema do fluxo de sinal e interface de hardware do sistema Endoscotech.



Fonte: Elaborado pelo autor utilizando a inteligência artificial Gemini, 2026.

4.5.2 Códigos Backend e Frontend do sistema

Código Backend e Frontend (Python Desktop)

O software de gestão e captura é programado em Python. O trecho abaixo demonstra partes cruciais da classe de visualização de Exames (TelaExames), que emprega o robusto framework PySide6 para construir os widgets e coordenar de forma assíncrona as chamadas às bibliotecas de visão computacional (OpenCV via cv2) e de acesso a banco de dados. Este código gerencia a interface que exibe a "linha do tempo" da gravação e a orquestração dos "QTimers" responsáveis por puxar os frames da câmera endoscópica, exibindo o diagnóstico ao vivo.

- Telas de Exames:

```
# -*- coding: utf-8 -*-
"""Tela de exames: captura de imagens (JPG) e vídeos (MP4), seleção para impressão."""

import sys
import random
from pathlib import Path
from datetime import datetime

sys.path.insert(0, str(Path(__file__).resolve().parent.parent))

import cv2
from PySide6.QtWidgets import (
    QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QLabel, QComboBox, QDateEdit,
    QListWidget, QListWidgetItem, QCheckBox, QMessageBox, QFileDialog, QGroupBox,
    QSplitter, QFrame, QScrollArea, QGridLayout, QDialog, QDialogButtonBox,
    QInputDialog, QLineEdit, QStackedWidget,
)
from PySide6.QtCore import Qt, QTimer, QDate, QSize, QEvent
from PySide6.QtGui import QImage, QPixmap, QKeySequence, QShortcut, QPainter, QPen, QColor, QIcon

from app.database import (
    listar_pacientes, listar_exames, inserir_exame,
    listar_arquivos_exame, inserir_arquivo_exame, definir_selecao_impressao,
    excluir_arquivo_exame, buscar_exame, buscar_paciente, PASTA_EXPORTACOES_PDF,
)
from app.captura import (
    listar_dispositivos_captura, capturar_frame, salvar_imagem,
```

```
    iniciar_gravacao_video, gravar_frame_video, parar_gravacao_video,
    obter_nome_dispositivo,
)
from app.config import (
    get_dispositivo_captura, set_dispositivo_captura, load_config, get_tecla_pedal,
    get_pacs_host, get_pacs_port, get_pacs_ae_title, get_pacs_ae_title_local,
)
from app.edicao_imagem import recortar_fundo_preto
from app.laudos_impressao import gerar_pdf_imagens, imprimir_imagens_qt
from app.pacs import enviar_imagem_pacs
from ui.edicao_imagem_dialog import EdicaoImagemDialog
from ui.config_dialog import ConfigDialog
from ui.combo_estilo import aplicar_estilo_selecao_visivel
from ui.estilo_botoes import aplicar_estilo_botao_primario,
aplicar_estilo_botao_secundario
from ui.tema import estilo_cabecalho, estilo_titulo_cabecalho, estilo_preview,
estilo_thumb, ACCENT
from app.log_sistema import registrar
from app.paths import DATA_DIR


class TimelineWidget(QWidget):
    """Barra de linha do tempo que cresce enquanto a gravação está ativa."""

    def __init__(self, parent=None):
        super().__init__(parent)
        self.setMinimumHeight(36)
        self._segundos = 0
        self._gravando = False

    def set_gravando(self, gravando: bool, segundos: int = 0):
        self._gravando = gravando
        self._segundos = segundos
```

```
        self.update()

    def paintEvent(self, event):
        painter = QPainter(self)
        painter.setRenderHint(QPainter.RenderHint.Antialiasing)
        w, h = self.width(), self.height()
        painter.fillRect(0, 0, w, h, QColor("#0a1322"))

        # Trilha de fundo
        track_y = h // 2
        track_h = 4
        painter.fillRect(0, track_y - track_h // 2, w, track_h, QColor("#1e2d42"))

        # Marcadores de tempo a cada 30s
        duracao_max = max(300, self._segundos + 60)
        pen_mark = QPen(QColor("#2a3d5c"), 1)
        painter.setPen(pen_mark)
        painter.setFont(painter.font())
        for s in range(0, duracao_max + 1, 30):
            x = int(s / duracao_max * w)
            painter.drawLine(x, track_y - 8, x, track_y + 8)
            if s > 0 and x < w - 20:
                painter.setPen(QColor("#3a5070"))
                painter.drawText(x + 2, track_y - 10, f"{s}s")
                painter.setPen(pen_mark)

        if self._gravando and self._segundos > 0:
            # Barra de progresso vermelha
            prog_w = int(self._segundos / duracao_max * w)
            painter.fillRect(0, track_y - track_h // 2, prog_w, track_h,
QColor("#e74c3c"))
            # Cabeça de leitura
            cx = prog_w
```

```
            painter.setBrush(QColor("#e74c3c"))
            painter.setPen(Qt.NoPen)
            painter.drawEllipse(cx - 6, track_y - 6, 12, 12)
        elif not self._gravando:
            # Linha central neutra
            painter.fillRect(0, track_y - 1, w, 2, QColor("#1e2d42"))

        painter.end()


class TelaExames(QWidget):
    def __init__(self, main_window):
        super().__init__()
        self.main_window = main_window
        self.exame_id = None
        self.pasta_exame = None
        self.cap = None
        self.gravacao_video = None
        self.nome_dispositivo = "Desconhecido"
        self._timer_preview = QTimer(self)
        self._timer_preview.timeout.connect(self.atualizar_preview)
        self._preview_falhas_consecutivas = 0
        self._modo_leitura = False
        self._preview_intervalo_normal = 40
        self._preview_intervalo_lento = 150
        self._rec_started_at: datetime | None = None
        self._status_timer = QTimer(self)
        self._status_timer.timeout.connect(self._atualizar_status_toolbar)
        self._status_timer.start(1000)

        layout = QVBoxLayout(self)
        layout.setContentsMargins(0, 0, 0, 0)
        layout.setSpacing(0)
```

```
        # ── TOOLBAR SUPERIOR ──────────────────────────────────────────────────────────────────────
        toolbar = QFrame()
        self._tool_frame = toolbar
        toolbar.setFixedHeight(50)
        toolbar.setFrameShape(QFrame.NoFrame)
        toolbar.setStyleSheet("QFrame { background:#0d1b2e; border-bottom:1px solid #1e2d42; }")
        tl = QHBoxLayout(toolbar)
        tl.setContentsMargins(12, 0, 12, 0)
        tl.setSpacing(6)

        self.btn_top_gravar = QPushButton("● Gravar")
        self.btn_top_gravar.setCheckable(True)
        self.btn_top_gravar.setMinimumHeight(34)
        self.btn_top_gravar.setStyleSheet(
            "QPushButton { background:#c0392b; color:#fff; border:none; border-radius:7px;"
            " padding:6px 14px; font-weight:700; }"
            "QPushButton:checked { background:#e74c3c; }"
            "QPushButton:hover { background:#e74c3c; }"
        )
        self.btn_top_gravar.clicked.connect(self.toggle_gravacao)

        _btn_style = (
            "QPushButton { background:#1c283d; color:#dce6f5; border:1px solid #3a4d6e;"
            " border-radius:7px; padding:6px 14px; }"
            "QPushButton:hover { background:#24344d; }"
            "QPushButton:disabled { color:#4a5a6a; border-color:#2a3545; }"
        )
        self.btn_top_captura = QPushButton("Capturar")
```

```
        camera_icon_path = DATA_DIR / "icons" / "camera.svg"
        if camera_icon_path.exists():
            self.btn_top_captura.setIcon(QIcon(str(camera_icon_path)))
            self.btn_top_captura.setIconSize(QSize(16, 16))
        self.btn_top_captura.setStyleSheet(_btn_style)
        self.btn_top_captura.setMinimumHeight(34)
        self.btn_top_captura.clicked.connect(self.capturar_imagem)

        self.btn_top_preview = QPushButton("👁 Iniciar Visualização")
        self.btn_top_preview.setStyleSheet(_btn_style)
        self.btn_top_preview.setMinimumHeight(34)
        self.btn_top_preview.clicked.connect(self.toggle_preview)

        self.btn_top_anotar = QPushButton("✒ Anotar")
        self.btn_top_anotar.setStyleSheet(_btn_style)
        self.btn_top_anotar.setMinimumHeight(34)
        self.btn_top_anotar.clicked.connect(
            lambda: QMessageBox.information(self, "Anotar", "Anotação disponível no
modo de exame avançado.")
        )

        self.btn_top_zoom = QPushButton("Zoom (1.0x)")
        zoom_icon_path = DATA_DIR / "icons" / "search.svg"
        if zoom_icon_path.exists():
            self.btn_top_zoom.setIcon(QIcon(str(zoom_icon_path)))
            self.btn_top_zoom.setIconSize(QSize(16, 16))
        self.btn_top_zoom.setStyleSheet(_btn_style)
        self.btn_top_zoom.setMinimumHeight(34)
        self.btn_top_zoom.clicked.connect(
            lambda: QMessageBox.information(self, "Zoom", "Zoom disponível no
modo de exame avançado.")
        )
```

```
        sep1 = QFrame()
        sep1.setFrameShape(QFrame.Shape.NoFrame)
        sep1.setFixedWidth(1)
        sep1.setStyleSheet("background: #2a3d5c;")

        self.btn_limpar = QPushButton("🗑 Limpar")
        self.btn_limpar.setStyleSheet(_btn_style)
        self.btn_limpar.setMinimumHeight(34)
        self.btn_limpar.clicked.connect(self._limpar_selecao)

        self.btn_finalizar_top = QPushButton("Finalizar exame")
        save_icon_path = DATA_DIR / "icons" / "save.svg"
        if save_icon_path.exists():
            self.btn_finalizar_top.setIcon(QIcon(str(save_icon_path)))
            self.btn_finalizar_top.setIconSize(QSize(16, 16))
        self.btn_finalizar_top.setMinimumHeight(34)
        self.btn_finalizar_top.setStyleSheet(
            "QPushButton { background:#27ae60; color:#fff; border:1px solid #7af0a0;"
            " border-radius:7px; padding:6px 18px; font-weight:700; }"
            "QPushButton:hover { background:#2ecc71; }"
        )
        self.btn_finalizar_top.clicked.connect(self._voltar_para_pacientes)

        for b in (self.btn_top_gravar, self.btn_top_preview, self.btn_top_captura,
self.btn_top_anotar, self.btn_top_zoom):
            tl.addWidget(b)
        tl.addWidget(sep1)
        tl.addWidget(self.btn_limpar)
        tl.addStretch(1)
        tl.addWidget(self.btn_finalizar_top)
        layout.addWidget(toolbar)
```

```
# ── CONTEÚDO PRINCIPAL ──────────────────────────────────────────
content_w = QWidget()
content_l = QHBoxLayout(content_w)
content_l.setContentsMargins(8, 8, 8, 4)
content_l.setSpacing(8)

# ── ESQUERDA: label + meta + viewport + ECG ──────────────────────
left = QWidget()
left_l = QVBoxLayout(left)
left_l.setContentsMargins(0, 0, 0, 0)
left_l.setSpacing(4)

lbl_vis = QLabel("VISUALIZAÇÃO PRINCIPAL")
lbl_vis.setStyleSheet(
    "color:#4a6a8a; font-size:10px; font-weight:700; letter-spacing:1px;"
)
left_l.addWidget(lbl_vis)

# Barra de metadados
meta = QFrame()
self._meta_frame = meta
meta.setFixedHeight(38)
meta.setStyleSheet(
    "QFrame { background:#0f1a2b; border:none; border-radius:8px; }"
)
meta_h = QHBoxLayout(meta)
meta_h.setContentsMargins(10, 4, 10, 4)
meta_h.setSpacing(14)
lbl_style = "color:#dce6f5; font-size:12px; font-weight:600;"

# Ícone paciente com texto
```

```
        self.lbl_meta_nome = QLabel("—")
        self.lbl_meta_id = QLabel("ID —")
        self.lbl_meta_data = QLabel(datetime.now().strftime("%d/%m/%Y"))
        self.lbl_meta_hora = QLabel(datetime.now().strftime("%H:%M:%S"))
        for w in (self.lbl_meta_nome, self.lbl_meta_id, self.lbl_meta_data,
self.lbl_meta_hora):
            w.setStyleSheet(lbl_style)
            meta_h.addWidget(w)
        meta_h.addStretch(1)
        self.lbl_meta_rec = QLabel("● REC")
        self.lbl_meta_rec.setStyleSheet("color:#ff4d4d; font-weight:800; font-
size:12px;")
        self.lbl_meta_rec.setVisible(False)
        self.lbl_meta_hd = QLabel("—")
        self.lbl_meta_hd.setStyleSheet(
            "background:#b7691a; color:#fff; border-radius:4px;"
            " padding:2px 8px; font-size:11px; font-weight:800;"
        )
        meta_h.addWidget(self.lbl_meta_rec)
        meta_h.addWidget(self.lbl_meta_hd)
        left_l.addWidget(meta)

        # Feedback de estado
        self.lbl_estado = QLabel(f"⚠ Sem sinal | — | {self.nome_dispositivo}")
        self.lbl_estado.setStyleSheet(
            "color:#7a9bbf; font-size:11px; font-weight:600; background:transparent;
padding:4px 0;"
        )
        left_l.addWidget(self.lbl_estado)

        # Viewport
        self.lbl_preview = QLabel()
        self.lbl_preview.setMinimumSize(640, 360)
```

```
        self.lbl_preview.setAlignment(Qt.AlignCenter)
        self.lbl_preview.setStyleSheet(
            "background:#0a1322; color:#9fb3d4;"
            " border:1px solid #2a3d5c; border-radius:10px;"
        )
        self.lbl_preview.setText("Selecionar paciente → Salvar → Iniciar → Capturar")
        left_l.addWidget(self.lbl_preview, 1)

        # Linha do tempo de gravação
        timeline_frame = QFrame()
        timeline_frame.setFixedHeight(52)
        timeline_frame.setFrameShape(QFrame.NoFrame)
        timeline_frame.setStyleSheet("QFrame { background:#0a1322; border:none; border-radius:6px; }")
        tl_hl = QHBoxLayout(timeline_frame)
        tl_hl.setContentsMargins(10, 6, 10, 6)
        tl_hl.setSpacing(10)
        self._lbl_timeline = QLabel("● Gravação")
        self._lbl_timeline.setStyleSheet(
            "color:#4a6a8a; font-size:11px; font-weight:700; background:transparent;"
        )
        self._timeline = TimelineWidget()
        tl_hl.addWidget(self._lbl_timeline)
        tl_hl.addWidget(self._timeline, 1)
        left_l.addWidget(timeline_frame)

        content_l.addWidget(left, 1)

        # ── DIREITA: GALERIA ──────────────────────────────────────────────────

        right = QWidget()
        right.setFixedWidth(238)
```

```
        right_l = QVBoxLayout(right)
        right_l.setContentsMargins(0, 0, 0, 0)
        right_l.setSpacing(4)

        # ── FORMULÁRIO (sempre visível no topo) ──────────────────────────
        _sm_style = (
            "QPushButton { background:#1c283d; color:#dce6f5; border:1px solid #3a4d6e;"
            " border-radius:5px; padding:3px 9px; font-size:10px; }"
            "QPushButton:hover { background:#24344d; }"
        )

        # ── combos ocultos (usados internamente por salvar_exame) ──
        self._updating_combo = False  # Flag para evitar chamadas recursivas
        self.combo_paciente = QComboBox()
        self.combo_paciente.setVisible(False)

self.combo_paciente.currentIndexChanged.connect(self._on_paciente_changed)
        self.data_exame = QDateEdit()
        self.data_exame.setCalendarPopup(True)
        self.data_exame.setDate(QDate.currentDate())
        self.data_exame.setVisible(False)
        self.combo_tipo = QComboBox()
        self.combo_tipo.setVisible(False)
        self.combo_tipo.addItems(["Endoscopia", "Ultrassom", "Colonoscopia", "Outro"])
        self.combo_tipo.setVisible(False)

        # ── INFORMAÇÕES DO EXAME ─────────────────────────────────────────

        info_exame = QFrame()
        info_exame.setFrameShape(QFrame.NoFrame)
```

```
        info_exame.setStyleSheet("QFrame { background:#0d1b2e; border:none;
border-radius:8px; }")
        info_exame_l = QVBoxLayout(info_exame)
        info_exame_l.setContentsMargins(16, 16, 16, 16)
        info_exame_l.setSpacing(12)

        # ── PACIENTE ──────────────────────────────────────────────────────
        lbl_paciente_titulo = QLabel("PACIENTE")
        lbl_paciente_titulo.setStyleSheet("color:#7a9bbf; font-size:10px; font-
weight:600; background:transparent;")
        info_exame_l.addWidget(lbl_paciente_titulo)

        self.lbl_bot_nome = QLabel("—")
        self.lbl_bot_nome.setStyleSheet("color:#e8eef8; font-size:14px; font-
weight:700; background:transparent;")
        info_exame_l.addWidget(self.lbl_bot_nome)

        self.lbl_bot_id = QLabel("ID —")
        self.lbl_bot_id.setStyleSheet("color:#9fb3d4; font-size:11px;
background:transparent;")
        info_exame_l.addWidget(self.lbl_bot_id)

        # ── MÉDICO ────────────────────────────────────────────────────────
        lbl_medico_titulo = QLabel("MÉDICO")
        lbl_medico_titulo.setStyleSheet("color:#7a9bbf; font-size:10px; font-
weight:600; background:transparent;")
        info_exame_l.addWidget(lbl_medico_titulo)

        self.lbl_bot_medico = QLabel("—")
        self.lbl_bot_medico.setStyleSheet("color:#e8eef8; font-size:14px; font-
weight:700; background:transparent;")
```

```
        info_exame_l.addWidget(self.lbl_bot_medico)

        # ── TIPO ──────────────────────────────────────────────────────
        lbl_tipo_titulo = QLabel("TIPO")
        lbl_tipo_titulo.setStyleSheet("color:#7a9bbf; font-size:10px; font-weight:600;
background:transparent;")
        info_exame_l.addWidget(lbl_tipo_titulo)

        self.combo_bot_tipo = QComboBox()
        self.combo_bot_tipo.addItems(["Endoscopia", "Ultrassom", "Colonoscopia",
"Outro"])
        self.combo_bot_tipo.setMinimumHeight(32)
        aplicar_estilo_selecao_visivel(self.combo_bot_tipo)
        self.combo_bot_tipo.currentTextChanged.connect(self._on_tipo_changed)
        info_exame_l.addWidget(self.combo_bot_tipo)

        right_l.addWidget(info_exame)

        # "Salvar exame" — pequeno, para ajuste manual se necessário
        self.btn_salvar_exame = QPushButton("Salvar")
        save_icon_path = DATA_DIR / "icons" / "save.svg"
        if save_icon_path.exists():
            self.btn_salvar_exame.setIcon(QIcon(str(save_icon_path)))
            self.btn_salvar_exame.setIconSize(QSize(16, 16))
        self.btn_salvar_exame.setMinimumHeight(28)
        self.btn_salvar_exame.setStyleSheet(
            _sm_style +
            "QPushButton { background:#1e293b; color:#e2e8f0; font-weight:700;
border:1px solid #334155; }"
            "QPushButton:hover { background:#273549; }"
        )
        self.btn_salvar_exame.clicked.connect(self.salvar_exame)
```

```
        right_l.addWidget(self.btn_salvar_exame)

        btn_cfg = QPushButton("⚙  Configurações")
        btn_cfg.setStyleSheet(_sm_style)
        btn_cfg.clicked.connect(self.abrir_config)
        right_l.addWidget(btn_cfg)

        # ── SEPARADOR + GALERIA ──────────────────────────────────────
        sep_gal = QFrame()
        sep_gal.setFrameShape(QFrame.Shape.NoFrame)
        sep_gal.setStyleSheet("color:#1e2d42;")
        right_l.addWidget(sep_gal)

        rh = QHBoxLayout()
        titulo_galeria = QLabel("GALERIA")
        titulo_galeria.setStyleSheet(
            "color:#9fb3d4; font-weight:800; font-size:12px; letter-spacing:1px;"
        )
        rh.addWidget(titulo_galeria)
        rh.addStretch()
        btn_abrir_s = QPushButton("Abrir exame")
        btn_abrir_s.setToolTip("Abrir exame existente")
        btn_abrir_s.setStyleSheet(_sm_style)
        btn_abrir_s.clicked.connect(self.on_abrir_exame_clicked)
        rh.addWidget(btn_abrir_s)
        right_l.addLayout(rh)

        self.lista_imagens = QListWidget()
        self.lista_imagens.setSpacing(4)
        self.lista_imagens.setMovement(QListWidget.Movement.Static)

self.lista_imagens.setVerticalScrollMode(QListWidget.ScrollMode.ScrollPerPixel)
```

```
        self.lista_imagens.setStyleSheet(
            "QListWidget { background:#0b1220; border:none; }"
            "QListWidget::item { border:none; padding:2px; }"
        )
        right_l.addWidget(self.lista_imagens, 1)

        self._lbl_galeria_vazia = QLabel("Nenhum exame aberto.\nSalve um exame
para capturar.")
        self._lbl_galeria_vazia.setAlignment(Qt.AlignCenter)
        self._lbl_galeria_vazia.setStyleSheet(
            "color:#3a5070; font-size:11px; padding:16px; background:transparent;"
        )
        self._lbl_galeria_vazia.setWordWrap(True)
        right_l.addWidget(self._lbl_galeria_vazia)

        self.btn_editar_img = QPushButton("Editar imagem")
        self.btn_editar_img.setMinimumHeight(30)
        aplicar_estilo_botao_secundario(self.btn_editar_img)
        self.btn_editar_img.clicked.connect(self.editar_imagem_selecionada)
        right_l.addWidget(self.btn_editar_img)

        act_row = QHBoxLayout()
        act_row.setSpacing(4)
        self.btn_imprimir = QPushButton("Imprimir")
        self.btn_imprimir.setMinimumHeight(28)
        aplicar_estilo_botao_secundario(self.btn_imprimir)
        self.btn_imprimir.clicked.connect(self.imprimir_imagens)
        self.btn_exportar_pdf = QPushButton("PDF")
        self.btn_exportar_pdf.setMinimumHeight(28)
        aplicar_estilo_botao_secundario(self.btn_exportar_pdf)
        self.btn_exportar_pdf.clicked.connect(self.exportar_pdf_imagens)
        self.btn_enviar_pacs = QPushButton("PACS")
        self.btn_enviar_pacs.setMinimumHeight(28)
```

```
        aplicar_estilo_botao_secundario(self.btn_enviar_pacs)
        self.btn_enviar_pacs.clicked.connect(self.enviar_para_pacs)
        act_row.addWidget(self.btn_imprimir)
        act_row.addWidget(self.btn_exportar_pdf)
        act_row.addWidget(self.btn_enviar_pacs)
        right_l.addLayout(act_row)

        # Dummy para compatibilidade com código que referenciava _right_stack
        self._right_stack = QStackedWidget()

        content_l.addWidget(right)
        layout.addWidget(content_w, 1)

        # ── BARRA DE STATUS INFERIOR ──────────────────────────────────────
        bottom = QFrame()
        bottom.setFixedHeight(50)
        bottom.setFrameShape(QFrame.NoFrame)
        bottom.setStyleSheet("QFrame { background:#0d1a2c; border:none; }")
        bl = QHBoxLayout(bottom)
        bl.setContentsMargins(8, 0, 8, 0)
        bl.setSpacing(6)

        self.btn_voltar = QPushButton("Voltar")
        voltar_icon_path = DATA_DIR / "icons" / "arrow-left.svg"
        if voltar_icon_path.exists():
            self.btn_voltar.setIcon(QIcon(str(voltar_icon_path)))
            self.btn_voltar.setIconSize(QSize(16, 16))
        self.btn_voltar.setMinimumHeight(34)
        self.btn_voltar.setStyleSheet(
            "QPushButton { background:#1c283d; color:#dce6f5; border:1px solid
#3a4d6e;"
            " border-radius:6px; padding:6px 12px; }"
```

```
            "QPushButton:hover { background:#24344d; }"
        )
        self.btn_voltar.clicked.connect(self._voltar_para_pacientes)
        bl.addWidget(self.btn_voltar)

        sep2 = QFrame()
        sep2.setFrameShape(QFrame.Shape.NoFrame)
        sep2.setFixedWidth(1)
        sep2.setStyleSheet("background:#2a3d5c;")
        bl.addWidget(sep2)

        # ── STATUS ──────────────────────────────────────────────────────────

        bl.addSpacing(8)
        self.lbl_bot_hd = QLabel("—")
        self.lbl_bot_hd.setStyleSheet(
            "background:#b7691a; color:#fff; border-radius:4px;"
            " padding:2px 7px; font-size:10px; font-weight:800;"
        )
        self.lbl_bot_fmt = QLabel("PNG")
        self.lbl_bot_fmt.setStyleSheet(
            "background:#2a7cff; color:#fff; border-radius:4px;"
            " padding:2px 7px; font-size:10px; font-weight:800;"
        )
        self.lbl_bot_rec = QLabel("● REC")
        self.lbl_bot_rec.setStyleSheet("color:#ff4d4d; font-weight:800; font-
size:11px;")
        self.lbl_bot_rec.setVisible(False)
        for lbl in (self.lbl_bot_hd, self.lbl_bot_fmt, self.lbl_bot_rec):
            bl.addWidget(lbl)

        bl.addStretch(1)
```

```
self.btn_pronto = QPushButton("Finalizar exame")
save_icon_path = DATA_DIR / "icons" / "save.svg"
if save_icon_path.exists():
    self.btn_pronto.setIcon(QIcon(str(save_icon_path)))
    self.btn_pronto.setIconSize(QSize(16, 16))
self.btn_pronto.setMinimumHeight(34)
self.btn_pronto.setStyleSheet(
    "QPushButton { background:#27ae60; color:#fff; border:1px solid #7af0a0;"
    " border-radius:6px; padding:6px 16px; font-weight:700; }"
    "QPushButton:hover { background:#2ecc71; }"
)
self.btn_pronto.clicked.connect(self._voltar_para_pacientes)
bl.addWidget(self.btn_pronto)

layout.addWidget(bottom)

# ── ALIASES para compatibilidade com métodos existentes ──────
self.btn_gravar = self.btn_top_gravar
self.btn_capturar = self.btn_top_captura
self.btn_iniciar_preview = self.btn_top_preview
self.lbl_rec = QLabel()
self.lbl_rec_tempo = QLabel()
self.lbl_res = QLabel()
self.lbl_pedal = QLabel()
self._grupo_imagens = None

self.setStyleSheet(
    "QGroupBox { font-weight:700; color:#dce6f5; border:1px solid #2a3d5c;"
    " border-radius:8px; margin-top:8px; padding-top:8px; }"
    "QGroupBox::title { subcontrol-origin:margin; left:10px; padding:0 6px; }"
    "QLabel { color:#dce6f5; }"
)
```

```
        self._carregar_preview_placeholder()
        self.aplicar_tema(True)
        self.carregar_combos()
        self._registrar_atalho_pedal()
        self.atualizar_dados()

    # ── TEMA ──────────────────────────────────────────────────────────────────

    def aplicar_tema(self, escuro: bool):
        """Sempre tema escuro."""
        self._tool_frame.setStyleSheet(
            "QFrame { background:#0d1b2e; border-bottom:1px solid #1e2d42; }"
        )
        self._meta_frame.setStyleSheet(
            "QFrame { background:#0f1a2b; border:none; border-radius:8px; }"
        )
        self.lbl_preview.setStyleSheet(
            "background:#0a1322; color:#9fb3d4;"
            " border:1px solid #2a3d5c; border-radius:10px;"
        )
        self.setStyleSheet(
            "QGroupBox { font-weight:700; color:#dce6f5; border:1px solid #2a3d5c;"
            " border-radius:8px; margin-top:8px; padding-top:8px; }"
            "QGroupBox::title { subcontrol-origin:margin; left:10px; padding:0 6px; }"
            "QListWidget { background:#0f1a2b; color:#e8eef8; border:1px solid #2a3d5c; border-radius:8px; }"
            "QListWidget::item { border:none; padding:2px; }"
            "QLabel { color:#dce6f5; }"
        )

    # ── HELPERS ───────────────────────────────────────────────────────────────
```

```
    def _atualizar_label_pedal(self):
        tecla = get_tecla_pedal()
        self.lbl_pedal.setText(
            f"🦶 Pedal: tecla [{tecla}] = Capturar imagem (mantenha a janela em foco)"
        )

    def _limpar_selecao(self):
        """Desmarca todas as imagens da seleção de impressão."""
        if not self.exame_id:
            return
        resp = QMessageBox.question(
            self, "Limpar seleção",
            "Desmarcar todas as imagens da galeria?",
            QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No,
            QMessageBox.StandardButton.No,
        )
        if resp == QMessageBox.StandardButton.Yes:
            for i in range(self.lista_imagens.count()):
                item = self.lista_imagens.item(i)
                w = self.lista_imagens.itemWidget(item)
                if w:
                    for cb in w.findChildren(QCheckBox):
                        cb.setChecked(False)

    def _carregar_preview_placeholder(self):
        try:
            base = DATA_DIR / "exames"
            if not base.exists():
                return
            candidatos = sorted(base.rglob("*.jpg"), key=lambda p: p.stat().st_mtime,
reverse=True)
            if not candidatos:
```

```
            candidatos = sorted(base.rglob("*.png"), key=lambda p:
p.stat().st_mtime, reverse=True)
            if not candidatos:
                return
            pm = QPixmap(str(candidatos[0]))
            if pm.isNull():
                return
            self.lbl_preview.setPixmap(
                pm.scaled(
                    self.lbl_preview.width() or 640,
                    self.lbl_preview.height() or 360,
                    Qt.AspectRatioMode.KeepAspectRatioByExpanding,
                    Qt.TransformationMode.SmoothTransformation,
                )
            )
        except Exception:
            pass

    def _toggle_freeze_preview(self):
        if self.cap is None:
            return
        if self._timer_preview.isActive():
            self._timer_preview.stop()
        else:
            self._timer_preview.start()

    def _voltar_para_pacientes(self):
        """Validações antes de finalizar o exame - Sistema médico seguro."""
        # Em modo leitura, volta sem validações
        if hasattr(self, '_modo_leitura') and self._modo_leitura:
            return

        # Verificar se há paciente selecionado
```

```
paciente_id = self.combo_paciente.currentData()
if not paciente_id:
    QMessageBox.warning(
        self,
        "Checagem de Dados",
        "❌ Nenhum paciente selecionado!\n\n"
        "É obrigatório selecionar um paciente antes de finalizar o exame.\n"
        "Por favor, escolha um paciente na lista."
    )
    return

# Salvar exame automaticamente se não foi salvo ainda
if not hasattr(self, 'exame_id') or not self.exame_id:
    self.salvar_exame(silencioso=True)
    if not hasattr(self, 'exame_id') or not self.exame_id:
        QMessageBox.warning(
            self,
            "Checagem de Dados",
            "❌ Exame não foi salvo!\n\n"
            "É obrigatório salvar o exame antes de finalizar.\n"
            "Por favor, clique em 'Salvar exame' primeiro."
        )
        return

# Verificar se há pelo menos uma captura (imagem ou vídeo)
try:
    from app.database import listar_arquivos_exame
    arquivos = listar_arquivos_exame(self.exame_id)
    if not arquivos:
        resposta = QMessageBox.question(
            self,
            "Checagem de Dados",
        "⚠️ Nenhuma captura registrada!\n\n"
```

```
                "Este exame não possui imagens ou vídeos capturados.\n"
                "Deseja finalizar mesmo assim?\n\n"
                "Recomendação: Capture pelo menos uma imagem antes de finalizar.",
                QMessageBox.Yes | QMessageBox.No,
                QMessageBox.No
            )
            if resposta == QMessageBox.No:
                return
        except Exception:
            pass  # Se não conseguir verificar, permite continuar

        # Se passou em todas as validações, permite finalizar
        self._resetar_estado_exame()
        return

    def _atualizar_status_toolbar(self):
        agora = datetime.now().strftime("%H:%M:%S")
        self.lbl_meta_hora.setText(agora)
        if self.gravacao_video:
            if self._rec_started_at is None:
                self._rec_started_at = datetime.now()
            secs = int((datetime.now() - self._rec_started_at).total_seconds())
            mm, ss = secs // 60, secs % 60
            tempo = f"{mm:02d}:{ss:02d}"
            self.btn_top_gravar.setText(f"● Gravar  {tempo}")
            self.btn_top_gravar.setChecked(True)
            self.btn_top_gravar.setText(f"● REC  {tempo}")
            self.btn_top_gravar.setChecked(True)
            self.lbl_meta_rec.setVisible(True)
            self.lbl_bot_rec.setVisible(True)
            self.lbl_rec.setVisible(True)
            self.lbl_rec_tempo.setText(f"{tempo} min")
```

```
            self._lbl_timeline.setText(f"● Gravando  {tempo}")
            self._lbl_timeline.setStyleSheet(
                "color:#e74c3c;  font-size:11px;  font-weight:700;
background:transparent;"
            )
            self._timeline.set_gravando(True, secs)
        else:
            self._rec_started_at = None
            self.btn_top_gravar.setText("● Gravar")
            self.btn_top_gravar.setChecked(False)
            self.btn_top_gravar.setText("● Gravar")
            self.btn_top_gravar.setChecked(False)
            self.lbl_meta_rec.setVisible(False)
            self.lbl_bot_rec.setVisible(False)
            self.lbl_rec.setVisible(False)
            self.lbl_rec_tempo.setText("00:00 min")
            self._lbl_timeline.setText("● Gravação")
            self._lbl_timeline.setStyleSheet(
                "color:#4a6a8a;  font-size:11px;  font-weight:700;
background:transparent;"
            )
            self._timeline.set_gravando(False, 0)

    def _registrar_atalho_pedal(self):
        tecla = get_tecla_pedal()
        seq = QKeySequence(tecla)
        if seq.isEmpty():
            seq = QKeySequence("F8")
        self._shortcut_pedal = QShortcut(seq, self)
        self._shortcut_pedal.setContext(Qt.ApplicationShortcut)
        self._shortcut_pedal.activated.connect(self.capturar_imagem)
```

```
    # ── COMBOS ──────────────────────────────────────────────────────────────

    def carregar_combos(self):
        self._updating_combo = True
        self.combo_paciente.clear()
        for p in listar_pacientes():
            id_clinica = (p.get("id_clinica") or "").strip()
            nome = p.get("nome", "")
            if id_clinica:
                texto = f"{id_clinica} - {nome}"
            else:
                texto = f"{nome} — #{p['id']}" if nome else f"#{p['id']}"
            self.combo_paciente.addItem(texto, p["id"])
        self.combo_paciente.setCurrentIndex(-1)
        self._updating_combo = False

    def _on_paciente_changed(self, index):
        """Atualiza os labels do menu lateral quando o paciente é alterado no combo."""
        if self._updating_combo:
            return
        paciente_id = self.combo_paciente.currentData()
        if paciente_id:
            try:
                from app.database import buscar_paciente
                paciente = buscar_paciente(paciente_id)
                if paciente:
                    nome = paciente.get("nome", "")
                    id_cli = (paciente.get("id_clinica") or "").strip()
                    medico_nome = (paciente.get("medico_nome") or "").strip()
                    self.lbl_bot_nome.setText(nome if nome else "—")
                    self.lbl_bot_id.setText("ID " + id_cli if id_cli else "ID —")
```

```
                self.lbl_bot_medico.setText(medico_nome if medico_nome else "—")
            except Exception as e:
                print("Erro ao atualizar paciente:", e)

    def _atualizar_info_card(self):
        """Sincroniza os labels do cartão com os dados do paciente/exame."""
        p = getattr(self, "_paciente_atual", None)
        if p:
            nome = p.get("nome") or "—"
            id_ = p.get("id_clinica") or ""
            medico = p.get("medico_nome") or "—"

            # LATERAL
            self.lbl_bot_nome.setText(nome)
            self.lbl_bot_id.setText(f"ID {id_}" if id_ else "ID —")
            self.lbl_bot_medico.setText(medico)

            # TOPO (se existir)
            if hasattr(self, "lbl_top_nome"):
                self.lbl_top_nome.setText(nome)
            if hasattr(self, "lbl_top_id"):
                self.lbl_top_id.setText(f"ID {id_}" if id_ else "ID —")
            if hasattr(self, "lbl_top_medico"):
                self.lbl_top_medico.setText(medico)

            # VISUALIZAÇÃO PRINCIPAL
            self.lbl_meta_nome.setText(nome)
            self.lbl_meta_id.setText(f"ID {id_}" if id_ else "ID —")
        else:
            # Limpa tudo
            self.lbl_bot_nome.setText("—")
            self.lbl_bot_id.setText("ID —")
            self.lbl_bot_medico.setText("—")
```

```
            if hasattr(self, "lbl_top_nome"):
                self.lbl_top_nome.setText("—")

            # VISUALIZAÇÃO PRINCIPAL
            self.lbl_meta_nome.setText("—")
            self.lbl_meta_id.setText("ID —")

    def _on_tipo_changed(self, tipo):
        """Atualiza o combo_tipo quando o combo_bot_tipo é alterado."""
        if self._updating_combo:
            return
        self._updating_combo = True
        self.combo_tipo.setCurrentText(tipo)
        self._updating_combo = False
        # Atualizar status se estiver em "Exame finalizado"
        if self.lbl_estado.text() == "✅ Exame finalizado":
            if self.cap is not None:
                self.lbl_estado.setText(f"🟢 Visualização ativa | {self.lbl_bot_hd.text()} |
{self.nome_dispositivo}")
            else:
                self.lbl_estado.setText(f"⚠️ Sem sinal | — | {self.nome_dispositivo}")

    def _desativar_captura(self):
        self._timer_preview.stop()

        if self.cap:
            self.cap.release()
            self.cap = None

        self.btn_top_gravar.setEnabled(False)
        self.btn_top_captura.setEnabled(False)
        self.btn_top_preview.setEnabled(False)
```

```
        self.btn_pronto.setEnabled(False)

    def _resetar_estado_exame(self):
        self.exame_id = None
        self.pasta_exame = None

        # parar preview
        if self._timer_preview.isActive():
            self._timer_preview.stop()

        # parar gravação se existir
        if self.gravacao_video:
            from app.captura import parar_gravacao_video
            parar_gravacao_video(self.gravacao_video)
            self.gravacao_video = None

        # reset visual
        self.lbl_estado.setText(f"⚠️ Sem sinal | — | {self.nome_dispositivo}")
        self.lbl_preview.setText("Selecionar paciente → Salvar → Iniciar → Capturar")

    def focar_paciente(self, paciente_id):
        self._resetar_estado_exame()
        self._paciente_atual = None

        # Recarregar combo para garantir que o paciente recém-cadastrado esteja disponível
        self.carregar_combos()

        self._updating_combo = True
        idx = self.combo_paciente.findData(paciente_id)
        if idx >= 0:
            self.combo_paciente.setCurrentIndex(idx)
```

```
        self._updating_combo = False

        try:
            p = buscar_paciente(paciente_id)
            if p:
                nome = p.get("nome", "")
                id_cli = (p.get("id_clinica") or "").strip()
                medico_nome = (p.get("medico_nome") or "").strip()

                self._paciente_atual = {
                    "id": paciente_id,
                    "nome": nome,
                    "id_clinica": id_cli,
                    "medico_nome": medico_nome
                }

        except Exception as e:
            print("Erro ao focar paciente:", e)

        self.data_exame.setDate(QDate.currentDate())
        self._atualizar_info_card()
        self.combo_bot_tipo.setCurrentText("Endoscopia")

    # ── EXAME ──────────────────────────────────────────────────────────
    ──────────────────────────────────────────────


    def novo_exame(self):
        self._modo_leitura = False
        self._resetar_estado_exame()
        self._paciente_atual = None
        self.carregar_combos()
        self.combo_paciente.setCurrentIndex(-1)
        self.combo_tipo.setCurrentText("Endoscopia")
```

```
        self.lista_imagens.clear()
        self._lbl_galeria_vazia.setVisible(True)
        self.btn_capturar.setEnabled(False)
        self.btn_gravar.setEnabled(False)
        # Reativar botões
        self.btn_top_gravar.setEnabled(True)
        self.btn_top_captura.setEnabled(True)
        self.btn_top_preview.setEnabled(True)
        self.btn_pronto.setEnabled(True)
        # Limpar labels do menu lateral
        self.lbl_bot_nome.setText("—")
        self.lbl_bot_id.setText("ID —")
        self.lbl_bot_medico.setText("—")
        self.combo_bot_tipo.setCurrentText("Endoscopia")
        # Limpar labels da visualização principal
        self.lbl_meta_nome.setText("—")
        self.lbl_meta_id.setText("ID —")

    def get_paciente_selecionado(self):
        """Retorna o paciente selecionado ou None."""
        paciente_id = self.combo_paciente.currentData()
        if not paciente_id:
            return None
        return {"id": paciente_id}

    def on_abrir_exame_clicked(self):
        paciente_id = self.combo_paciente.currentData()

        print("DEBUG BOTÃO - paciente_id:", paciente_id)

        if not paciente_id:
            QMessageBox.warning(self, "Aviso", "Selecione um paciente")
            return
```

```
        from ui.lista_exames_paciente import TelaListaExames as
TelaHistoricoExames

        dialog = TelaHistoricoExames(self, paciente_id)
        dialog.exec()

    def carregar_exame_por_id(self, exame_id, modo_leitura=False):
        e = buscar_exame(exame_id)
        if not e:
            return
        self._carregar_exame(e, modo_leitura=modo_leitura)

    def _carregar_exame(self, e, modo_leitura=False):
        self.exame_id = e["id"]
        self._modo_leitura = modo_leitura
        self.pasta_exame = e.get("pasta_arquivos") or ""
        nome = str(e.get("paciente_nome") or "Paciente")
        id_cli = str(e.get("paciente_id_clinica") or e.get("paciente_id") or "—")
        medico_nome = (e.get("medico_nome") or "").strip()
        tipo_exame = e.get("tipo", "Endoscopia")

        self.lbl_meta_nome.setText(nome)
        self.lbl_meta_id.setText("ID " + id_cli)
        self.lbl_meta_data.setText(datetime.now().strftime("%d/%m/%Y"))

        self.lbl_bot_nome.setText(nome)
        self.lbl_bot_id.setText("ID " + id_cli)
        self.lbl_bot_medico.setText(medico_nome if medico_nome else "—")
        self.combo_bot_tipo.setCurrentText(tipo_exame if tipo_exame else
"Endoscopia")
        self.carregar_combos()
        self._updating_combo = True
```

```
self.combo_paciente.setCurrentIndex(self.combo_paciente.findData(e["paciente_id"]))
        self._updating_combo = False
        self.combo_tipo.setCurrentText(e.get("tipo", "Endoscopia"))
        try:
            dt = e.get("data_exame", "")
            if dt:
                self.data_exame.setDate(QDate.fromString(dt, "yyyy-MM-dd"))
        except Exception:
            pass
        # Desabilitar botões em modo leitura
        if self._modo_leitura:
            self._desativar_captura()
            self.lbl_estado.setText("📄 Modo leitura | Exame finalizado")
            self.lbl_preview.setText("Exame em modo leitura. Não é possível adicionar imagens ou vídeos.")
        else:
            self.btn_capturar.setEnabled(True)
            self.btn_gravar.setEnabled(True)
            self.btn_top_preview.setEnabled(True)
            self.btn_pronto.setEnabled(True)
            self.lbl_preview.setText("Exame carregado. Inicie a visualização para capturar imagens ou vídeos.")
        self.atualizar_lista_imagens()

    def salvar_exame(self, silencioso=False):
        paciente_id = self.combo_paciente.currentData()
        if not paciente_id:
            if not silencioso:
                QMessageBox.warning(self, "Aviso", "Selecione um paciente.")
            return
        tipo = self.combo_tipo.currentText()
```

```
        data_ex = self.data_exame.date().toString("yyyy-MM-dd") + " " + datetime.now().strftime("%H:%M:%S")
        pasta = self.pasta_exame
        if not pasta and self.exame_id:
            ex = buscar_exame(self.exame_id)
            pasta = ex.get("pasta_arquivos") if ex else None
        if not pasta:
            base = DATA_DIR / "exames"
            base.mkdir(parents=True, exist_ok=True)
            pasta = str(base / f"exame_{datetime.now().strftime('%Y%m%d_%H%M%S')}")

        # Obter medico_nome do paciente
        medico_nome = ""
        try:
            p = buscar_paciente(paciente_id)
            if p:
                medico_nome = (p.get("medico_nome") or "").strip()
        except Exception:
            pass

        if self.exame_id:
            from app.database import atualizar_exame
            atualizar_exame(
                self.exame_id,
                paciente_id=paciente_id,
                tipo=tipo, data_exame=data_ex, pasta_arquivos=pasta, medico_nome=medico_nome,
            )
        else:
            self.exame_id = inserir_exame(
                paciente_id=paciente_id,
```

```
            tipo=tipo,          data_exame=data_ex,          pasta_arquivos=pasta,
medico_nome=medico_nome,
        )
        self.pasta_exame = pasta
    self._atualizar_info_card()

    # Atualizar labels do menu lateral após salvar
    try:
        p = buscar_paciente(paciente_id)
        if p:
            nome_paciente = p.get("nome", "")
            id_cli = (p.get("id_clinica") or "").strip()
            medico_nome = (p.get("medico_nome") or "").strip()
            self.lbl_bot_nome.setText(nome_paciente if nome_paciente else "—")
            self.lbl_bot_id.setText("ID " + id_cli if id_cli else "ID —")
            self.lbl_bot_medico.setText(medico_nome if medico_nome else "—")
            self.combo_bot_tipo.setCurrentText(tipo if tipo else "Endoscopia")
    except Exception:
        pass

    self.btn_capturar.setEnabled(True)
    self.btn_gravar.setEnabled(True)
    self._lbl_galeria_vazia.setVisible(False)
    self.btn_capturar.setEnabled(True)
    self.btn_gravar.setEnabled(True)
    self.atualizar_lista_imagens()

# ── CÂMERA / PREVIEW ──────────────────────────────────────────────
──────────────────────────────────────

def toggle_preview(self):
    if self.cap is not None:
        self.parar_preview()
```

```
            return
        # Atualizar status imediatamente para remover "Exame finalizado"
        self.lbl_estado.setText(f"🔵 Conectando | — | {self.nome_dispositivo}")
        disp = get_dispositivo_captura()
        self.nome_dispositivo = obter_nome_dispositivo(disp)
        self.cap = cv2.VideoCapture(disp, cv2.CAP_DSHOW)
        if not self.cap.isOpened():
            registrar(f"Captura: falha ao abrir dispositivo de captura (índice {disp})", "ERRO")
            QMessageBox.warning(self, "Erro", "Não foi possível abrir o dispositivo de captura. Verifique em Configurações.")
            self.cap = None
            self.lbl_estado.setText(f"⚠️ Sem sinal | — | {self.nome_dispositivo}")
            return
        # Obter resolução real do dispositivo
        w = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        h = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        self.lbl_bot_hd.setText(f"{w}x{h}")
        registrar(f"Captura: visualização iniciada (dispositivo {disp}, resolução {w}x{h})", "INFO")
        self._preview_falhas_consecutivas = 0
        self._timer_preview.setInterval(self._preview_intervalo_normal)
        self._timer_preview.start()
        # Atualizar estado inicial
        if self.exame_id and self.pasta_exame:
            self.lbl_estado.setText(f"🟡 Aguardando captura | {w}x{h} | {self.nome_dispositivo}")
        else:
            self.lbl_estado.setText(f"🟢 Visualização ativa | {w}x{h} | {self.nome_dispositivo}")

    def parar_preview(self):
        self._timer_preview.stop()
```

```
        if self.cap is not None:
            self.cap.release()
            self.cap = None
        registrar("Captura: visualização encerrada", "INFO")
        self.lbl_preview.setText("Visualização encerrada.")
        self.lbl_bot_hd.setText("—")
        if self.gravacao_video:
            self.parar_gravacao()

    def atualizar_preview(self):
        if self.cap is None or not self.cap.isOpened():
            return
        ret, frame = self.cap.read()
        if not ret or frame is None:
            self._preview_falhas_consecutivas += 1
            if self._preview_falhas_consecutivas >= 20 and
self._timer_preview.interval() != self._preview_intervalo_lento:
                self._timer_preview.setInterval(self._preview_intervalo_lento)
            if self._preview_falhas_consecutivas == 30:
                self.lbl_preview.setText("Sem sinal válido do dispositivo.\nVerifique o
cabo e a placa de captura.")
                self.lbl_estado.setText(f"⚠️ Sem sinal | HD 1080p |
{self.nome_dispositivo}")
            return
        try:
            h, w = frame.shape[:2]
        except (IndexError, AttributeError):
            self._preview_falhas_consecutivas += 1
            return
        if h < 64 or w < 64 or h > 4096 or w > 4096:
            self._preview_falhas_consecutivas += 1
            return
```

```
        if frame.dtype.name != "uint8" or len(frame.shape) != 3 or frame.shape[2] !=
3:
            self._preview_falhas_consecutivas += 1
            return
        self._preview_falhas_consecutivas = 0

        # Atualizar resolução dinamicamente
        self.lbl_bot_hd.setText(f"{w}x{h}")

        # Atualizar label de estado com resolução atual
        if self.gravacao_video:
            if self._rec_started_at is None:
                self._rec_started_at = datetime.now()
            secs = int((datetime.now() - self._rec_started_at).total_seconds())
            mm, ss = secs // 60, secs % 60
            tempo = f"{mm:02d}:{ss:02d}"
            self.lbl_estado.setText(f"🔴 Gravando {tempo} | {w}x{h} |
{self.nome_dispositivo}")
        elif self.exame_id and self.pasta_exame:
            self.lbl_estado.setText(f"🟡 Aguardando captura | {w}x{h} |
{self.nome_dispositivo}")
        else:
            self.lbl_estado.setText(f"🟢 Visualização ativa | {w}x{h} |
{self.nome_dispositivo}")
        if self._timer_preview.interval() == self._preview_intervalo_lento:
            self._timer_preview.setInterval(self._preview_intervalo_normal)
        if self.gravacao_video:
            gravar_frame_video(self.gravacao_video, frame)
        try:
            mean_full, std_full = cv2.meanStdDev(frame)
            if float(max(std_full)) > 72:
                return
            top_rows = max(1, int(h * 0.15))
```

```
            mean_top, std_top = cv2.meanStdDev(frame[:top_rows])
            mean_rest, _ = cv2.meanStdDev(frame[top_rows:])
            if float(max(std_top)) > 55 and float(mean_rest.mean()) < 35:
                return
        except Exception:
            pass
        preview_largura_max = 1024
        if w > preview_largura_max:
            r = preview_largura_max / w
            frame = cv2.resize(frame, (preview_largura_max, max(64, int(h * r))),
interpolation=cv2.INTER_AREA)
            h, w = frame.shape[:2]
        else:
            frame = frame.copy()
        qimg = QImage(frame.data, w, h, 3 * w, QImage.Format.Format_BGR888)
        if qimg.isNull():
            return
        pix = QPixmap.fromImage(qimg)
        lw, lh = self.lbl_preview.width(), self.lbl_preview.height()
        if pix.width() <= lw and pix.height() <= lh:
            scaled_pix = pix
        else:
            scaled_pix = pix.scaled(lw, lh, Qt.KeepAspectRatio,
Qt.SmoothTransformation)
        self.lbl_preview.setPixmap(scaled_pix)

    def capturar_imagem(self):
        # Criar exame automaticamente se não existir
        if self.exame_id is None or not self.pasta_exame:
            self.salvar_exame(silencioso=True)
            if self.exame_id is None or not self.pasta_exame:
                QMessageBox.warning(self, "Aviso", "Selecione um paciente e clique
em 'Salvar exame' antes de capturar.")
```

```
            return
        if self.cap is None or not self.cap.isOpened():
            ret, frame = capturar_frame()
        else:
            ret, frame = self.cap.read()
        if not ret or frame is None:
            registrar("Captura: falha ao capturar frame (sem sinal?)", "ERRO")
            QMessageBox.warning(self, "Erro", "Não foi possível capturar o frame.")
            return
        caminho = salvar_imagem(frame, self.pasta_exame)
        if caminho:
            inserir_arquivo_exame(self.exame_id, "imagem", caminho,
selecionado_impressao=1)
            self.atualizar_lista_imagens()
        else:
            QMessageBox.warning(self, "Erro", "Não foi possível salvar a imagem.")

    def toggle_gravacao(self):
        if self.gravacao_video:
            caminho = parar_gravacao_video(self.gravacao_video)
            self.gravacao_video = None
            self.btn_top_gravar.setChecked(False)
            if caminho and self.exame_id:
                inserir_arquivo_exame(self.exame_id, "video", caminho)
                self.atualizar_lista_imagens()
            return
        # Criar exame automaticamente se não existir
        if self.exame_id is None or not self.pasta_exame:
            self.salvar_exame(silencioso=True)
            if self.exame_id is None or not self.pasta_exame:
                QMessageBox.warning(self, "Aviso", "Selecione um paciente e clique
em 'Salvar exame' antes de gravar.")
                return
```

```
        if self.cap is None:
            QMessageBox.warning(self, "Aviso", "Inicie a visualização antes de gravar
vídeo.")
            return
        w_p = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        h_p = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        if w_p < 1 or h_p < 1:
            w_p, h_p = 1920, 1080
        cfg = load_config()
        fps = int(cfg.get("fps_video", 30))
        self.gravacao_video = iniciar_gravacao_video(
            self.pasta_exame, largura=w_p, altura=h_p, fps=fps,
            tamanho_preview=(w_p, h_p),
        )
        if not self.gravacao_video or self.gravacao_video[1] is None:
            self.gravacao_video = None
            QMessageBox.warning(self, "Erro", "Não foi possível iniciar a gravação.")
            return
        self.btn_top_gravar.setChecked(True)

    def parar_gravacao(self):
        if self.gravacao_video:
            parar_gravacao_video(self.gravacao_video)
            self.gravacao_video = None
            self.btn_top_gravar.setChecked(False)

    # ── GALERIA ──────────────────────────────────────────────────────
──────────────────────────────────────────

    def _thumbnail_imagem(self, caminho, largura=132, altura=74):
        if not Path(caminho).exists():
            return QPixmap(largura, altura)
        img = QImage(caminho)
```

```
        if img.isNull():
            return QPixmap(largura, altura)
        return QPixmap.fromImage(img.scaled(largura, altura, Qt.KeepAspectRatio,
Qt.SmoothTransformation))

    def atualizar_lista_imagens(self):
        self.lista_imagens.clear()
        if hasattr(self, "_lista_imagens_widget_to_item"):
            self._lista_imagens_widget_to_item.clear()
        if not self.exame_id:
            self._lbl_galeria_vazia.setVisible(True)
            return
        self._lbl_galeria_vazia.setVisible(False)
        thumb_w, thumb_h = 132, 74
        for arq in listar_arquivos_exame(self.exame_id):
            item = QListWidgetItem()
            item.setData(Qt.UserRole, arq)
            item.setSizeHint(QSize(0, thumb_h + 26))
            w = QWidget()
            h = QHBoxLayout(w)
            h.setContentsMargins(8, 8, 8, 8)
            h.setSpacing(10)
            w.setStyleSheet(
                "QWidget { background:#0b1628; border:1px solid #2a3d5c; border-
radius:10px; }"
            )
            lbl_thumb = QLabel()
            lbl_thumb.setFixedSize(thumb_w, thumb_h)
            lbl_thumb.setStyleSheet(estilo_thumb())
            lbl_thumb.setAlignment(Qt.AlignCenter)
            if arq.get("tipo") == "imagem" and Path(arq["caminho"]).exists():
                lbl_thumb.setPixmap(self._thumbnail_imagem(arq["caminho"],
thumb_w, thumb_h))
```

```
            else:
                lbl_thumb.setText("🎬 Vídeo" if arq.get("tipo") == "video" else "?")
            h.addWidget(lbl_thumb)
            col = QVBoxLayout()
            col.setSpacing(4)
            dt = (

datetime.fromtimestamp(Path(arq["caminho"]).stat().st_mtime).strftime("%H:%M:
%S")
                if Path(arq["caminho"]).exists() else "--:--:--"
            )
            topo = QHBoxLayout()
            lbl_hora = QLabel(dt)
            lbl_hora.setStyleSheet("color:#e8eef8; font-weight:700;")
            topo.addWidget(lbl_hora)
            topo.addStretch(1)
            if arq.get("tipo") == "video":
                lbl_fmt = QLabel("MP4")
                lbl_fmt.setStyleSheet(
                    "background:#2a7cff; color:#fff; border-radius:4px;"
                    " padding:1px 6px; font-size:10px; font-weight:700;"
                )
            else:
                lbl_fmt = QLabel("PNG")
                lbl_fmt.setStyleSheet(
                    "background:#2a7cff; color:#fff; border-radius:4px;"
                    " padding:1px 6px; font-size:10px; font-weight:700;"
                )
            topo.addWidget(lbl_fmt)
            col.addLayout(topo)
            nome = Path(arq["caminho"]).name
            if arq.get("tipo") == "video":
                nome += " (vídeo)"
```

```
            lbl_nome = QLabel(nome)
            lbl_nome.setStyleSheet("font-size:12px; color:#dce6f5;")
            lbl_nome.setWordWrap(True)
            lbl_nome.setMaximumWidth(160)
            col.addWidget(lbl_nome)
            linha_cb = QHBoxLayout()
            cb = QCheckBox("incluir")
            cb.setStyleSheet("color:#9fb3d4; font-size:10px;")
            cb.setChecked(bool(arq.get("selecionado_impressao", 1)))
            cb.stateChanged.connect(
                lambda state, a=arq: definir_selecao_impressao(a["id"], state ==
Qt.CheckState.Checked)
            )
            linha_cb.addWidget(cb)
            linha_cb.addStretch(1)
            col.addLayout(linha_cb)
            col.addStretch()
            h.addLayout(col)
            self.lista_imagens.addItem(item)
            self.lista_imagens.setItemWidget(item, w)
            if not hasattr(self, "_lista_imagens_widget_to_item"):
                self._lista_imagens_widget_to_item = {}
            self._lista_imagens_widget_to_item[w] = item
            for child in (lbl_thumb, lbl_nome, cb):
                child.installEventFilter(self)
            w.installEventFilter(self)

    def atualizar_dados(self):
        if self.exame_id:
            self.atualizar_lista_imagens()
        self.carregar_combos()

    def eventFilter(self, obj, event):
```

```
        if event.type() == QEvent.Type.MouseButtonPress and hasattr(self,
"_lista_imagens_widget_to_item"):
            target = obj
            while target:
                item = self._lista_imagens_widget_to_item.get(target)
                if item is not None:
                    self.lista_imagens.setCurrentItem(item)
                    break
                target = target.parent() if hasattr(target, "parent") else None
        return super().eventFilter(obj, event)

    def editar_imagem_selecionada(self):
        item = self.lista_imagens.currentItem()
        if not item:
            QMessageBox.information(self, "Aviso", "Selecione uma imagem na lista.")
            return
        arq = item.data(Qt.UserRole)
        if arq.get("tipo") != "imagem":
            QMessageBox.information(self, "Aviso", "Apenas imagens podem ser
editadas.")
            return
        d = EdicaoImagemDialog(arq["caminho"], self)
        if d.exec():
            self.atualizar_lista_imagens()

    # ── IMPRESSÃO / PDF / PACS ──────────────────────────────────────────
──────────────────────────────────

    def _imagens_selecionadas_impressao(self):
        if not self.exame_id:
            return []
        pasta = self.pasta_exame
        if not pasta:
```

```
            ex = buscar_exame(self.exame_id)
            if ex:
                pasta = ex.get("pasta_arquivos") or ""
        paths = []
        for i in range(self.lista_imagens.count()):
            item = self.lista_imagens.item(i)
            w = self.lista_imagens.itemWidget(item)
            if not w:
                continue
            cbs = w.findChildren(QCheckBox)
            if not cbs or not cbs[0].isChecked():
                continue
            arq = item.data(Qt.UserRole)
            if not arq or arq.get("tipo") != "imagem":
                continue
            c = arq.get("caminho")
            if not c:
                continue
            p = Path(c)
            if not p.is_absolute() and pasta:
                p = (Path(pasta) / c).resolve()
            else:
                p = p.resolve()
            if p.exists():
                paths.append(str(p))
        return paths

    def _linha_dados_paciente_impressao(self):
        if not self.exame_id:
            return None
        ex = buscar_exame(self.exame_id)
        if not ex:
            return None
```

```
        paciente = buscar_paciente(ex.get("paciente_id")) if ex.get("paciente_id") else None
        partes = []
        if paciente:
            if paciente.get("id_clinica"):
                partes.append(f"Pront.: {paciente.get('id_clinica', '')}")
            partes.append(f"Paciente: {paciente.get('nome', '')}")
            if paciente.get("cpf"):
                partes.append(f"CPF: {paciente.get('cpf', '')}")
            if paciente.get("data_nascimento"):
                partes.append(f"Nasc.: {paciente.get('data_nascimento', '')}")
            if paciente.get("telefone"):
                partes.append(f"Tel: {paciente.get('telefone', '')}")
            if paciente.get("email"):
                partes.append(f"Email: {paciente.get('email', '')}")
        else:
            partes.append(f"Paciente: {ex.get('paciente_nome', '')}")
        partes.append(f"Data exame: {ex.get('data_exame', '')}")
        partes.append(f"Tipo: {ex.get('tipo', '')}")
        if ex.get("medico_nome"):
            partes.append(f"Médico: {ex.get('medico_nome', '')}")
        return " | ".join(p for p in partes if p).strip(" | ")

    def imprimir_imagens(self):
        paths = self._imagens_selecionadas_impressao()
        if not paths:
            QMessageBox.information(self, "Aviso", "Marque as imagens para incluir.")
            return
        linha_paciente = self._linha_dados_paciente_impressao()
        if imprimir_imagens_qt(paths, self, linha_dados_paciente=linha_paciente):
            QMessageBox.information(self, "OK", "Impressão enviada para a impressora.")
        else:
```

```
            QMessageBox.information(
                self, "Dica",
                "Não foi possível enviar para a impressora. Use \"Exportar PDF\" e imprima o arquivo."
            )

    def exportar_pdf_imagens(self):
        paths = self._imagens_selecionadas_impressao()
        if not paths:
            QMessageBox.information(self, "Aviso", "Marque as imagens para incluir.")
            return
        PASTA_EXPORTACOES_PDF.mkdir(parents=True, exist_ok=True)
        ex = buscar_exame(self.exame_id) if self.exame_id else None
        paciente_nome = (ex.get("paciente_nome") or "exame").replace(" ", "_") if ex else "exame"
        data_ex = (ex.get("data_exame") or "").replace("-", "") if ex else ""
        nome_sugerido = f"{paciente_nome}_{data_ex}.pdf" if data_ex else "imagens_exame.pdf"
        path_pdf, _ = QFileDialog.getSaveFileName(
            self, "Salvar PDF", str(PASTA_EXPORTACOES_PDF / nome_sugerido), "PDF (*.pdf)"
        )
        if path_pdf:
            linha_paciente = self._linha_dados_paciente_impressao()
            gerar_pdf_imagens(paths, path_pdf, linha_dados_paciente=linha_paciente)
            QMessageBox.information(self, "OK", f"PDF salvo: {path_pdf}")

    def enviar_para_pacs(self):
        paths = self._imagens_selecionadas_impressao()
        if not paths:
            QMessageBox.information(self, "Aviso", "Selecione as imagens desejadas para enviar ao PACS.")
```

```
            return
        host = get_pacs_host()
        if not host:
            QMessageBox.warning(
                self, "PACS",
                "Configure o servidor PACS em Configurações (host, porta e AE Titles) e teste a conexão."
            )
            return
        ex = buscar_exame(self.exame_id) if self.exame_id else None
        patient_id = str(ex.get("paciente_id") or self.exame_id or "") if ex else ""
        patient_name = (ex.get("paciente_nome") or "UNKNOWN") if ex else "UNKNOWN"
        def _uid():
            return f"1.2.840.10008.{''.join(random.choices('0123456789', k=24))}"
        study_uid = _uid()
        series_uid = _uid()
        ok_count = 0
        erro_msg = None
        for path in paths:
            ok, msg = enviar_imagem_pacs(
                path, host, get_pacs_port(), get_pacs_ae_title(), get_pacs_ae_title_local(),
                patient_id=patient_id, patient_name=patient_name,
                study_uid=study_uid, series_uid=series_uid,
            )
            if ok:
                ok_count += 1
            else:
                erro_msg = msg
        if ok_count == len(paths):
            QMessageBox.information(self, "PACS", f"{ok_count} imagem(ns) enviada(s) ao PACS com sucesso.")
```

```
        elif ok_count > 0:
            QMessageBox.warning(
                self, "PACS",
                f"{ok_count} de {len(paths)} enviada(s). Falha: {erro_msg or 'desconhecida'}."
            )
        else:
            QMessageBox.warning(self, "PACS", erro_msg or "Nenhuma imagem foi enviada.")

    def abrir_config(self):
        try:
            dlg = ConfigDialog(None)
            dlg.setWindowFlags(dlg.windowFlags() | Qt.WindowType.WindowStaysOnTopHint)
            dlg.setWindowModality(Qt.WindowModality.ApplicationModal)
            dlg.raise_()
            dlg.activateWindow()
            dlg.exec()
        except Exception as e:
            QMessageBox.critical(self, "Erro", f"Não foi possível abrir as configurações:\n{e}")
```

- Tela de Laudos:

```
# -*- coding: utf-8 -*-
"""Tela de emissão de laudos com máscaras (templates) pré-definidos."""

import sys
import json
from pathlib import Path
sys.path.insert(0, str(Path(__file__).resolve().parent.parent))

from PySide6.QtWidgets import (
    QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QLabel, QComboBox,
```

```
    QTextEdit, QPlainTextEdit, QListWidget, QListWidgetItem, QMessageBox,
    QFileDialog, QDialog, QLineEdit, QDialogButtonBox, QFormLayout, QFrame,
    QGroupBox, QScrollArea, QGridLayout, QHBoxLayout, QStyle, QCompleter,
)
from PySide6.QtCore import Qt, QSize, QStringListModel
from PySide6.QtGui import QPixmap, QIcon, QColor, QPainter

from app.database import (
    listar_exames, buscar_exame, buscar_laudo_por_exame, salvar_laudo,
    listar_templates_laudo, buscar_template_laudo,
    listar_arquivos_exame, PASTA_LAUDOS_PDF,
    inserir_template_laudo, atualizar_template_laudo, excluir_template_laudo,
)
from app.paths import DATA_DIR
from app.laudos_impressao import gerar_pdf_laudo
from ui.combo_estilo import aplicar_estilo_selecao_visivel
from ui.dialog_title_bar import adicionar_barra_janela
from ui.estilo_botoes import aplicar_estilo_botao_primario
from ui.tema import estilo_cabecalho, estilo_titulo_cabecalho


class DialogTemplate(QDialog):
    """Criar/editar template (máscara) de laudo."""
    def __init__(self, parent=None, template_id=None):
        super().__init__(parent)
        self.template_id = template_id
        titulo = "Editar template" if template_id else "Novo template"
        self.setWindowTitle(titulo)
        self.setMinimumSize(500, 400)
        layout = QVBoxLayout(self)
        conteudo = adicionar_barra_janela(self, layout, titulo)
        form = QFormLayout()
        self.nome = QLineEdit()
```

```
        self.nome.setFrame(False)
        self.nome.setPlaceholderText("Ex: Laudo padrão Endoscopia")
        self.conteudo = QTextEdit()
        self.conteudo.setPlaceholderText("Digite o texto modelo. Use [PACIENTE],
[DATA], [TIPO] como marcadores se quiser.")
        form.addRow("Nome do template", self.nome)
        form.addRow("Conteúdo", self.conteudo)
        btns = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btns.accepted.connect(self.accept)
        btns.rejected.connect(self.reject)
        form.addRow(btns)
        conteudo.addLayout(form)
        if template_id:
            t = buscar_template_laudo(template_id)
            if t:
                self.nome.setText(t.get("nome", ""))
                self.conteudo.setPlainText(t.get("conteudo", ""))


class TelaLaudos(QWidget):
    def __init__(self, main_window):
        super().__init__()
        self.main_window = main_window
        self.exame_atual = None
        self.exame_id = None
        self._todos_pacientes = []  # Armazenar lista completa para referência
        self._paciente_map = {}  # Mapear texto -> ID do paciente
        self.setStyleSheet("background: transparent;")

        outer = QVBoxLayout(self)
        outer.setContentsMargins(0, 0, 0, 0)
        outer.setSpacing(0)
```

```
        # ── CABEÇALHO ──────────────────────────────────────────────
──────────────────────────────────────

        header_layout = QHBoxLayout()
        header_layout.setContentsMargins(16, 8, 16, 8)
        header_layout.setSpacing(12)

        titulo = QLabel("LAUDO DO EXAME")
        titulo.setStyleSheet("""
    font-size: 18px;
    font-weight: bold;
    color: #E5E9F0;
""")

        header_layout.addWidget(titulo)
        header_layout.addStretch()

        # seletor de exame no cabeçalho
        lbl_ex = QLabel("Buscar paciente:")
        lbl_ex.setStyleSheet("color:#9fb3d4;                                font-size:12px;
background:transparent;")
        self.inp_busca_paciente = QLineEdit()
        self.inp_busca_paciente.setPlaceholderText("Nome ou ID do paciente...")
        self.inp_busca_paciente.setMinimumWidth(300)
        self.inp_busca_paciente.setMinimumHeight(32)
        self.inp_busca_paciente.setStyleSheet(
            "QLineEdit { background:#0f1a2b; color:#dce6f5; border:none; border-
radius:6px; padding:6px 10px; font-size:13px; }"
            "QLineEdit:focus { outline:none; }"
        )
        self.inp_busca_paciente.textChanged.connect(self._filtrar_pacientes)
        # Configurar completer para lista de sugestões
        self.completer_pacientes = QCompleter()
```

```
self.completer_pacientes.setCaseSensitivity(Qt.CaseSensitivity.CaseInsensitive)
        self.completer_pacientes.setFilterMode(Qt.MatchFlag.MatchContains)
        self.inp_busca_paciente.setCompleter(self.completer_pacientes)
        self.completer_pacientes.activated.connect(self._on_paciente_selecionado)
        header_layout.addWidget(lbl_ex)
        header_layout.addWidget(self.inp_busca_paciente)

        # seletor de template
        lbl_tpl = QLabel("Template:")
        lbl_tpl.setStyleSheet("color:#9fb3d4;                    font-size:12px;
background:transparent;")
        self.combo_template = QComboBox()
        self.combo_template.setMinimumWidth(180)
        self.combo_template.setMinimumHeight(32)
        aplicar_estilo_selecao_visivel(self.combo_template)
        self.combo_template.currentIndexChanged.connect(self._aplicar_template)

        _btn_sm = (
            "QPushButton { background:#1c283d; color:#dce6f5; border:1px solid
#3a4d6e;"
            " border-radius:5px; padding:4px 10px; font-size:11px; }"
            "QPushButton:hover { background:#24344d; }"
        )
        self.btn_novo_template = QPushButton("+ Template")
        self.btn_novo_template.setToolTip("Novo template de laudo")
        self.btn_novo_template.setStyleSheet(_btn_sm)
        self.btn_novo_template.setMinimumHeight(30)
        self.btn_novo_template.clicked.connect(self.novo_template)

        header_layout.addWidget(lbl_tpl)
        header_layout.addWidget(self.combo_template)
        header_layout.addWidget(self.btn_novo_template)
```

```
outer.addLayout(header_layout)

# ── CARTÃO DO PACIENTE ────────────────────────────────
────────────────────

info = QFrame()
info.setObjectName("PatientCard")
info.setFrameShape(QFrame.NoFrame)
info.setStyleSheet("background: transparent;")
ig = QGridLayout(info)
ig.setContentsMargins(16, 10, 16, 10)
ig.setHorizontalSpacing(20)
ig.setVerticalSpacing(4)

def lbl_key(t):
    x = QLabel(t)
    x.setStyleSheet("color:#7a9bbf; font-size:11px; background:transparent;")
    return x

def lbl_val():
    x = QLabel("—")
    x.setStyleSheet(
        "color:#dce6f5; font-size:12px; font-weight:600;
background:transparent;"
    )
    return x

ig.addWidget(lbl_key("Paciente"), 0, 0)
self._lbl_paciente = lbl_val()
ig.addWidget(self._lbl_paciente, 0, 1)

ig.addWidget(lbl_key("ID"), 1, 0)
self._lbl_id = lbl_val()
ig.addWidget(self._lbl_id, 1, 1)
```

```
        ig.addWidget(lbl_key("Data"), 0, 2)
        self._lbl_data = lbl_val()
        ig.addWidget(self._lbl_data, 0, 3)

        ig.addWidget(lbl_key("Médico solicitante"), 1, 2)
        self._lbl_medico = lbl_val()
        ig.addWidget(self._lbl_medico, 1, 3)

        ig.setColumnStretch(1, 2)
        ig.setColumnStretch(3, 2)
        outer.addWidget(info)

        # ── ÁREA COM SCROLL ──────────────────────────────────────────
──────────────────────────────

        scroll = QScrollArea()
        scroll.setWidgetResizable(True)
        scroll.setFrameShape(QFrame.Shape.NoFrame)
        scroll.setHorizontalScrollBarPolicy(Qt.ScrollBarPolicy.ScrollBarAlwaysOff)
        scroll.setStyleSheet(
            "QScrollArea { background:#0a1628; border:none; }"
            "QScrollBar:vertical { background:#0f1a2b; width:8px; border-radius:4px; }"
            "QScrollBar::handle:vertical { background:#2a3d5c; border-radius:4px; }"
        )

        body = QWidget()
        body.setStyleSheet("QWidget { background:#0a1628; }")
        bl = QVBoxLayout(body)
        bl.setContentsMargins(16, 14, 16, 14)
        bl.setSpacing(12)

        _gb_style = (
            "QGroupBox { color:#9fb3d4; font-size:11px; font-weight:700;"
```

```
        " border:1px solid #2a3d5c; border-radius:8px; margin-top:8px; padding-top:8px; }"
        "QGroupBox::title { subcontrol-origin:margin; left:10px; padding:0 6px;"
        " color:#9fb3d4; background:#0a1628; }"
    )
    _te_style = (
        "QPlainTextEdit, QTextEdit { background:#0d1b2e; color:#dce6f5;"
        " border:none; border-radius:6px; padding:8px; font-size:13px; }"
    )

    def plain_block(titulo, placeholder, min_h=90, max_h=None):
        gb = QGroupBox(titulo)
        gb.setStyleSheet(_gb_style)
        gl = QVBoxLayout(gb)
        gl.setContentsMargins(8, 8, 8, 8)
        te = QPlainTextEdit()
        te.setPlaceholderText(placeholder)
        te.setMinimumHeight(min_h)
        # Fonte padrão na criação
        from PySide6.QtGui import QFont
        te.setFont(QFont("Segoe UI", 10))
        if max_h is not None:
            te.setMaximumHeight(max_h)
        te.setStyleSheet(_te_style)
        gl.addWidget(te)
        bl.addWidget(gb)
        return te

    self._txt_achados = plain_block(
        "Descrição / Achados",
        "Descreva sistematicamente as estruturas e lesões observadas…",
        min_h=200,
    )
```

```
        self._txt_recomendacoes = plain_block(
            "Recomendações (opcional)",
            "Conduta, controles, exames complementares…",
            min_h=80,
            max_h=120,
        )

        # Imagens anexadas - Layout com dois painéis lado a lado + preview abaixo
        img_gb = QGroupBox("Imagens anexadas ao laudo")
        img_gb.setStyleSheet(_gb_style)
        img_main_layout = QVBoxLayout(img_gb)
        img_main_layout.setContentsMargins(16, 12, 16, 12)
        img_main_layout.setSpacing(12)

        # Layout horizontal para os dois painéis superiores
        top_panels = QHBoxLayout()
        top_panels.setSpacing(12)

        # Painel da esquerda - Seleção de imagens
        left_panel = QFrame()
        left_panel.setFrameShape(QFrame.NoFrame)
        left_panel.setStyleSheet("QFrame { background:#0d1b2e; border:none;
border-radius:8px; }")
        left_layout = QVBoxLayout(left_panel)
        left_layout.setAlignment(Qt.AlignCenter)
        left_layout.setSpacing(12)
        left_layout.setContentsMargins(20, 20, 20, 20)

        # Ícone de imagem centralizado em branco
        icon_label = QLabel()
        icon_label.setAlignment(Qt.AlignCenter)
        icon_label.setStyleSheet("background:transparent;")
        _icon_img_path = DATA_DIR / "icons" / "image.svg"
```

```
        if _icon_img_path.exists():
            icon_img = QIcon(str(_icon_img_path))
            pixmap = icon_img.pixmap(48, 48)
            colored_pixmap = QPixmap(pixmap.size())
            colored_pixmap.fill(Qt.transparent)
            painter = QPainter(colored_pixmap)
            painter.setRenderHint(QPainter.RenderHint.Antialiasing)
            painter.drawPixmap(0, 0, pixmap)

painter.setCompositionMode(QPainter.CompositionMode.CompositionMode_Sou
rceIn)
            painter.fillRect(colored_pixmap.rect(), QColor("white"))
            painter.end()
            icon_label.setPixmap(colored_pixmap)
        left_layout.addWidget(icon_label)

        # Botão selecionar
        self._btn_add_img = QPushButton("  Selecionar imagens do exame")
        _icon_img = DATA_DIR / "icons" / "image.svg"
        if _icon_img.exists():
            self._btn_add_img.setIcon(QIcon(str(_icon_img)))
            self._btn_add_img.setIconSize(QSize(18, 18))
        self._btn_add_img.setStyleSheet(
            "QPushButton { background:#1c283d; color:#4cc9f0; border:1px solid
#3a4d6e;"
            " border-radius:8px; padding:10px 20px; font-size:12px; font-weight:500; }"
            "QPushButton:hover { background:#24344d; border-color:#4cc9f0; }"
        )
        self._btn_add_img.clicked.connect(self._pick_images_for_laudo)
        left_layout.addWidget(self._btn_add_img, alignment=Qt.AlignCenter)

        # Texto explicativo
        img_hint = QLabel("Escolha entre as capturas já realizadas neste exame")
```

```
        img_hint.setAlignment(Qt.AlignCenter)
        img_hint.setStyleSheet("color:#7a9bbf;                    font-size:11px;
background:transparent;")
        left_layout.addWidget(img_hint)

        top_panels.addWidget(left_panel, 3)

        # Painel da direita - Status/Preview
        right_panel = QFrame()
        right_panel.setFrameShape(QFrame.NoFrame)
        right_panel.setStyleSheet("QFrame  {  background:#0d1b2e;  border:none;
border-radius:8px; }")
        right_layout = QVBoxLayout(right_panel)
        right_layout.setAlignment(Qt.AlignCenter)
        right_layout.setSpacing(12)
        right_layout.setContentsMargins(16, 16, 16, 16)

        # Ícone de pasta/arquivo em branco (padrão do sistema)
        folder_icon = QLabel()
        folder_icon.setAlignment(Qt.AlignCenter)
        folder_pixmap = QPixmap(48, 48)
        folder_pixmap.fill(Qt.transparent)
        folder_painter = QPainter(folder_pixmap)
        folder_painter.setRenderHint(QPainter.RenderHint.Antialiasing)
        folder_painter.setPen(QColor("#9fb3d4"))
        folder_painter.setBrush(QColor("#9fb3d4"))
        # Desenhar pasta estilizada
        folder_painter.drawRoundedRect(8, 16, 32, 20, 2, 2)
        folder_painter.drawRoundedRect(12, 10, 16, 8, 2, 2)
        folder_painter.end()
        folder_icon.setPixmap(folder_pixmap)
        right_layout.addWidget(folder_icon)
```

```
        # Texto "Nenhuma imagem anexada"
        self._lbl_no_images = QLabel("Nenhuma imagem anexada")
        self._lbl_no_images.setAlignment(Qt.AlignCenter)
        self._lbl_no_images.setStyleSheet("color:#9fb3d4;      font-size:13px;      font-
weight:500; background:transparent;")
        right_layout.addWidget(self._lbl_no_images)

        # Subtexto
        sub_lbl = QLabel("As imagens anexadas aparecerão no laudo final")
        sub_lbl.setAlignment(Qt.AlignCenter)
        sub_lbl.setStyleSheet("color:#7a9bbf;                    font-size:10px;
background:transparent;")
        sub_lbl.setWordWrap(True)
        right_layout.addWidget(sub_lbl)

        top_panels.addWidget(right_panel, 2)
        img_main_layout.addLayout(top_panels)

        # Container para as imagens selecionadas (fica abaixo dos dois painéis)
        self._img_container = QWidget()
        self._img_container.setStyleSheet("background:transparent;")
        self._img_layout = QGridLayout(self._img_container)
        self._img_layout.setContentsMargins(0, 8, 0, 0)
        self._img_layout.setSpacing(8)
        img_main_layout.addWidget(self._img_container)

        bl.addWidget(img_gb)
        bl.addStretch(1)

        scroll.setWidget(body)
        outer.addWidget(scroll, 1)
```

```
# ── BARRA INFERIOR ──────────────────────────────────────────────────
bottom = QFrame()
bottom.setFixedHeight(54)
bottom.setFrameShape(QFrame.NoFrame)
bottom.setStyleSheet("QFrame { background:#0d1a2c; border:none; }")
brow = QHBoxLayout(bottom)
brow.setContentsMargins(16, 0, 16, 0)
brow.setSpacing(8)

_btn_tool = (
    "QPushButton { background:#1c283d; color:#dce6f5; border:1px solid #3a4d6e;"
    " border-radius:6px; padding:6px 16px; font-size:12px; }"
    "QPushButton:hover { background:#24344d; }"
)
self.btn_voltar = QPushButton("  Voltar ao exame")
# Ícone de seta para voltar
_icon_voltar = DATA_DIR / "icons" / "arrow-left.svg"
if _icon_voltar.exists():
    self.btn_voltar.setIcon(QIcon(str(_icon_voltar)))
self.btn_voltar.setIconSize(QSize(16, 16))
self.btn_voltar.setStyleSheet(_btn_tool)
self.btn_voltar.setMinimumHeight(36)
self.btn_voltar.clicked.connect(
    lambda: main_window.pilha.setCurrentIndex(2)
)

self.btn_exportar = QPushButton("  Exportar texto…")
# Ícone clipboard do sistema
_icon_clipboard = DATA_DIR / "icons" / "clipboard.svg"
if _icon_clipboard.exists():
    self.btn_exportar.setIcon(QIcon(str(_icon_clipboard)))
```

```
self.btn_exportar.setIconSize(QSize(16, 16))
self.btn_exportar.setStyleSheet(_btn_tool)
self.btn_exportar.setMinimumHeight(36)
self.btn_exportar.clicked.connect(self._exportar_txt)

self.btn_exportar_pdf = QPushButton("  Exportar PDF…")
# Ícone de arquivo/documento para PDF
_icon_pdf = DATA_DIR / "icons" / "file-text.svg"
if _icon_pdf.exists():
    self.btn_exportar_pdf.setIcon(QIcon(str(_icon_pdf)))
self.btn_exportar_pdf.setIconSize(QSize(16, 16))
self.btn_exportar_pdf.setStyleSheet(_btn_tool)
self.btn_exportar_pdf.setMinimumHeight(36)
self.btn_exportar_pdf.clicked.connect(self.exportar_pdf)

self.btn_imprimir = QPushButton("  Imprimir…")
# Ícone de impressora
_icon_print = DATA_DIR / "icons" / "printer.svg"
if _icon_print.exists():
    self.btn_imprimir.setIcon(QIcon(str(_icon_print)))
self.btn_imprimir.setIconSize(QSize(16, 16))
self.btn_imprimir.setStyleSheet(_btn_tool)
self.btn_imprimir.setMinimumHeight(36)
self.btn_imprimir.clicked.connect(self._imprimir_laudo)

self.btn_salvar = QPushButton("  Salvar laudo")
# Ícone de salvar (disco/check)
_icon_salvar = DATA_DIR / "icons" / "save.svg"
if _icon_salvar.exists():
    self.btn_salvar.setIcon(QIcon(str(_icon_salvar)))
self.btn_salvar.setIconSize(QSize(16, 16))
self.btn_salvar.setStyleSheet(
```

```
            "QPushButton { background:#1a7a3a; color:#fff; border:1px solid #30c060;"
            " border-radius:6px; padding:6px 24px; font-weight:700; font-size:12px; }"
            "QPushButton:hover { background:#22964a; }"
        )
        self.btn_salvar.setMinimumHeight(36)
        self.btn_salvar.setMinimumWidth(140)
        self.btn_salvar.clicked.connect(self._salvar)

        brow.addWidget(self.btn_voltar)
        brow.addStretch(1)
        brow.addWidget(self.btn_exportar)
        brow.addWidget(self.btn_exportar_pdf)
        brow.addWidget(self.btn_imprimir)
        brow.addWidget(self.btn_salvar)
        outer.addWidget(bottom)

        self._image_entries: list[dict] = []
        self.atualizar_dados()

    # ── DADOS ──────────────────────────────────────────────────────────────

    def atualizar_dados(self):
        from app.database import listar_pacientes
        pacientes = listar_pacientes()
        self._todos_pacientes = pacientes  # Armazenar lista completa para referência

        # Popular completer com lista de pacientes
        lista_textos = []
        self._paciente_map = {}
        for p in pacientes:
```

```
        # Usar id_clinica se existir, caso contrário usar id do paciente
        id_paciente = p.get("id_clinica") or p.get("id")

        # Buscar último exame do paciente para obter data e médico
        exames = listar_exames(paciente_id=p["id"])
        data_exame = ""
        medico = ""

        if exames:
            ultimo_exame = max(exames, key=lambda x: x.get("data_exame", ""))
            data_exame = ultimo_exame.get("data_exame", "")
            medico_nome = ultimo_exame.get("medico_nome", "")
            medico = medico_nome if medico_nome else ""

        # Se não houver médico no exame, usar médico do paciente
        if not medico:
            medico = p.get("medico_nome", "") or ""

        # Formato: ID, Nome, Data do exame, médico (só exibe médico se houver)
        if medico and data_exame:
            texto = f"{id_paciente}, {p.get('nome', '')}, {data_exame}, {medico}"
        elif medico:
            texto = f"{id_paciente}, {p.get('nome', '')}, {medico}"
        elif data_exame:
            texto = f"{id_paciente}, {p.get('nome', '')}, {data_exame}"
        else:
            texto = f"{id_paciente}, {p.get('nome', '')}"

        lista_textos.append(texto)
        self._paciente_map[texto] = p["id"]

    model = QStringListModel(lista_textos)
    self.completer_pacientes.setModel(model)
```

```
        self.combo_template.blockSignals(True)
        self.combo_template.clear()
        self.combo_template.addItem("— Sem template —", None)
        for t in listar_templates_laudo():
            self.combo_template.addItem(t["nome"], t["id"])
        self.combo_template.blockSignals(False)

        self._ao_trocar_exame()

    def _filtrar_pacientes(self, texto):
        """Filtra pacientes baseado no texto digitado e carrega automaticamente o último exame."""
        texto = texto.strip().lower()
        if not texto:
            self._paciente_selecionado = None
            self.exame_id = None
            self._ao_trocar_exame()
            return

        # Buscar pacientes correspondentes
        from app.database import buscar_pacientes
        pacientes = buscar_pacientes(texto)

        if len(pacientes) == 1:
            # Se apenas um paciente encontrado, carregar o último exame automaticamente
            self._paciente_selecionado = pacientes[0]
            exames = listar_exames(paciente_id=pacientes[0]["id"])
            if exames:
                # Carregar o último exame (mais recente)
                ultimo_exame = max(exames, key=lambda x: x.get("data_exame", ""))
                self.exame_id = ultimo_exame["id"]
```

```
                self._ao_trocar_exame()
            else:
                self.exame_id = None
                self._ao_trocar_exame()
        else:
            # Múltiplos pacientes ou nenhum encontrado
            self._paciente_selecionado = None
            self.exame_id = None
            self._ao_trocar_exame()

    def _on_paciente_selecionado(self, texto):
        """Chamado quando um paciente é selecionado na lista de sugestões."""
        paciente_id = self._paciente_map.get(texto)
        if paciente_id:
            self._paciente_selecionado = {"id": paciente_id}
            exames = listar_exames(paciente_id=paciente_id)
            if exames:
                # Carregar o último exame (mais recente)
                ultimo_exame = max(exames, key=lambda x: x.get("data_exame", ""))
                self.exame_id = ultimo_exame["id"]
                self._ao_trocar_exame()
            else:
                # Se não houver exames, carregar dados do paciente mesmo assim
                self.exame_id = None
                self._carregar_dados_paciente(paciente_id)

    def _carregar_dados_paciente(self, paciente_id):
        """Carrega dados do paciente no cartão de paciente mesmo sem exames."""
        from app.database import buscar_paciente
        paciente = buscar_paciente(paciente_id)
        if paciente:
            nome = paciente.get("nome") or "—"
            pid = paciente.get("id_clinica") or paciente.get("id") or "—"
```

```
            data = "—"
            medico = paciente.get("medico_nome", "") or ""

            self._lbl_paciente.setText(nome)
            self._lbl_id.setText(str(pid))
            self._lbl_data.setText(str(data))
            self._lbl_medico.setText(medico if medico else "—")
        else:
            for lbl in (self._lbl_paciente, self._lbl_id, self._lbl_data, self._lbl_medico):
                lbl.setText("—")

    def _ao_trocar_exame(self):
        self.exame_atual = buscar_exame(self.exame_id) if self.exame_id else None

        if self.exame_atual:
            nome = self.exame_atual.get("paciente_nome") or "—"
            pid = self.exame_atual.get("paciente_id_clinica") or
self.exame_atual.get("paciente_id") or "—"
            data = self.exame_atual.get("data_exame") or "—"
            medico = self.exame_atual.get("medico_nome") or ""

            # Se não houver médico no exame, buscar médico do paciente
            if not medico and self.exame_atual.get("paciente_id"):
                from app.database import buscar_paciente
                paciente = buscar_paciente(self.exame_atual.get("paciente_id"))
                if paciente:
                    medico = paciente.get("medico_nome", "") or ""

            self._lbl_paciente.setText(nome)
            self._lbl_id.setText(str(pid))
            self._lbl_data.setText(str(data))
            self._lbl_medico.setText(medico if medico else "—")
        else:
```

```
            for lbl in (self._lbl_paciente, self._lbl_id, self._lbl_data, self._lbl_medico):
                lbl.setText("—")

        self._txt_achados.clear()
        self._txt_recomendacoes.clear()
        self._limpar_imagens()

        if not self.exame_id:
            return

        laudo = buscar_laudo_por_exame(self.exame_id)
        if laudo:
            conteudo = laudo.get("conteudo", "") or ""
            try:
                data = json.loads(conteudo)
                self._txt_achados.setPlainText(data.get("achados", ""))
                self._txt_recomendacoes.setPlainText(data.get("recomendacoes", ""))
            except (json.JSONDecodeError, TypeError):
                # compatibilidade com formato antigo (texto simples)
                self._txt_achados.setPlainText(conteudo)

    def _aplicar_template(self):
        tid = self.combo_template.currentData()
        if not tid:
            return
        t = buscar_template_laudo(tid)
        if not t:
            return
        texto = t.get("conteudo", "")
        if self.exame_atual:
            texto = texto.replace("[PACIENTE]",
self.exame_atual.get("paciente_nome", ""))
            texto = texto.replace("[DATA]", self.exame_atual.get("data_exame", ""))
```

```
            texto = texto.replace("[TIPO]", self.exame_atual.get("tipo", ""))
        self._txt_achados.setPlainText(texto)

    def novo_template(self):
        d = DialogTemplate(self)
        if d.exec() == QDialog.DialogCode.Accepted:
            nome = d.nome.text().strip() or "Sem nome"
            inserir_template_laudo(nome, d.conteudo.toPlainText())
            self.atualizar_dados()

    def editar_template(self):
        tid = self.combo_template.currentData()
        if not tid:
            QMessageBox.information(self, "Aviso", "Selecione um template para
editar.")
            return
        d = DialogTemplate(self, tid)
        if d.exec() == QDialog.DialogCode.Accepted:
            atualizar_template_laudo(tid, d.nome.text().strip() or "Sem nome",
d.conteudo.toPlainText())
            self.atualizar_dados()

    def remover_template(self):
        tid = self.combo_template.currentData()
        if not tid:
            QMessageBox.information(self, "Aviso", "Selecione na lista o template que
deseja remover.")
            return
        nome = self.combo_template.currentText()
        resp = QMessageBox.question(
            self, "Confirmar",
            f'Remover o template "{nome}"?',
            QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No,
```

```
            QMessageBox.StandardButton.No,
        )
        if resp == QMessageBox.StandardButton.Yes:
            excluir_template_laudo(tid)
            self.atualizar_dados()

    # ── IMAGENS ──────────────────────────────────────────────────────
    ─────────────────────────────────────

    def _limpar_imagens(self):
        for e in list(self._image_entries):
            w = e.get("widget")
            if w:
                w.setParent(None)
                w.deleteLater()
        self._image_entries.clear()

    def _pick_images_for_laudo(self):
        if not self.exame_atual:
            QMessageBox.information(self, "Aviso", "Selecione um exame primeiro.")
            return
        exame_id = self.exame_atual.get("id")
        arquivos = listar_arquivos_exame(exame_id, "imagem")
        if not arquivos:
            QMessageBox.information(self, "Sem imagens", "Nenhuma imagem de
    captura disponível para este exame.")
            return
        base_dir = Path(__file__).resolve().parent.parent
        for a in arquivos:
            caminho = a.get("caminho") or ""
            if not caminho:
                continue
            p = Path(caminho)
```

```
            if not p.is_absolute():
                p = (base_dir / caminho).resolve()
            if p.exists() and not any(e.get("path") == str(p) for e in self._image_entries):
                self._add_image_row(str(p))

    def _add_image_row(self, path: str):
        # Frame da imagem com preview maior
        img_frame = QFrame()
        img_frame.setFrameShape(QFrame.NoFrame)
        img_frame.setStyleSheet("QFrame { background:#0f1a2b; border:none;
border-radius:8px; }")
        vl = QVBoxLayout(img_frame)
        vl.setContentsMargins(8, 8, 8, 8)
        vl.setSpacing(6)

        # Thumbnail maior
        thumb = QLabel()
        thumb.setFixedSize(140, 100)
        thumb.setAlignment(Qt.AlignmentFlag.AlignCenter)
        thumb.setStyleSheet("background:#0a1628; border-radius:4px;")
        pm = QPixmap(path)
        if not pm.isNull():
            scaled_pm = pm.scaled(140, 100, Qt.AspectRatioMode.KeepAspectRatio,
                                  Qt.TransformationMode.SmoothTransformation)
            thumb.setPixmap(scaled_pm)
        else:
            thumb.setText("🖼")
            thumb.setStyleSheet("color:#9fb3d4; font-size:24px; background:#0a1628;
border-radius:4px;")
        vl.addWidget(thumb, alignment=Qt.AlignCenter)

        # Nome do arquivo (truncado)
```

```
        lbl_path = QLabel(Path(path).name[:20] + "..." if len(Path(path).name) > 20
else Path(path).name)
        lbl_path.setAlignment(Qt.AlignCenter)
        lbl_path.setStyleSheet("color:#9fb3d4;                    font-size:10px;
background:transparent;")
        vl.addWidget(lbl_path)

        # Botão remover
        btn_rm = QPushButton("✕")
        btn_rm.setFixedSize(24, 24)
        btn_rm.setStyleSheet(
            "QPushButton { background:#7a1010; color:#fff; border:none; border-
radius:12px;"
            " font-size:10px; font-weight:bold; } QPushButton:hover {
background:#a01818; }"
        )
        entry = {"widget": img_frame, "path": path}
        btn_rm.clicked.connect(lambda _, e=entry: self._remove_image_row(e))
        vl.addWidget(btn_rm, alignment=Qt.AlignCenter)

        # Adicionar ao grid (4 colunas)
        row_idx = len(self._image_entries) // 4
        col_idx = len(self._image_entries) % 4
        self._img_layout.addWidget(img_frame, row_idx, col_idx)
        self._image_entries.append(entry)

        # Esconder mensagem de "Nenhuma imagem"
        if hasattr(self, '_lbl_no_images'):
            self._lbl_no_images.setText(f"{len(self._image_entries)}
imagem(s) anexada(s)")
            self._lbl_no_images.setStyleSheet("color:#4cc9f0; font-size:13px; font-
weight:500; background:transparent;")
```

```
    def _remove_image_row(self, entry: dict):
        w = entry.get("widget")
        if w:
            w.setParent(None)
            w.deleteLater()
        if entry in self._image_entries:
            self._image_entries.remove(entry)

    # ── AÇÕES ──────────────────────────────────────────────────────────
──────────────────────────────────────

    def _salvar(self):
        if not self.exame_id:
            QMessageBox.warning(self, "Aviso", "Selecione um exame.")
            return
        exame_id = self.exame_id
        conteudo = json.dumps({
            "achados": self._txt_achados.toPlainText(),
            "recomendacoes": self._txt_recomendacoes.toPlainText(),
        })
        template_id = self.combo_template.currentData()
        salvar_laudo(exame_id, conteudo, template_id)
        QMessageBox.information(self, "OK", "Laudo salvo.")

    def _exportar_txt(self):
        if not self.exame_id:
            QMessageBox.warning(self, "Aviso", "Selecione um exame.")
            return
        exame_id = self.exame_id
        path, _ = QFileDialog.getSaveFileName(self, "Exportar texto do laudo", "",
"Texto (*.txt)")
        if path:
            texto = (
```

```
            self._txt_achados.toPlainText()
            + "\n\nRecomendações:\n"
            + self._txt_recomendacoes.toPlainText()
        )
        Path(path).write_text(texto, encoding="utf-8")
        QMessageBox.information(self, "OK", f"Texto exportado: {path}")

    def _imprimir_laudo(self):
        if not self.exame_id:
            QMessageBox.warning(self, "Aviso", "Selecione um exame.")
            return
        exame_id = self.exame_id
        try:
            from PySide6.QtPrintSupport import QPrinter, QPrintDialog
            from PySide6.QtGui import QTextDocument
        except ImportError:
            QMessageBox.critical(self, "Imprimir", "Suporte a impressão não
disponível.")
            return

        printer = QPrinter(QPrinter.PrinterMode.HighResolution)
        # força retrato sem importar QPageLayout (evita erro de DLL)
        try:
            _layout = printer.pageLayout()
            _Portrait = type(_layout).Orientation.Portrait
            _layout.setOrientation(_Portrait)
            printer.setPageLayout(_layout)
        except Exception:
            pass
        dlg = QPrintDialog(printer, self)
        if dlg.exec() != QPrintDialog.DialogCode.Accepted:
            return
        # força retrato novamente após o diálogo
```

```
        try:
            _layout = printer.pageLayout()
            _layout.setOrientation(type(_layout).Orientation.Portrait)
            printer.setPageLayout(_layout)
        except Exception:
            pass

        achados = self._txt_achados.toPlainText()
        recom = self._txt_recomendacoes.toPlainText()
        paciente = self._lbl_paciente.text()
        data = self._lbl_data.text()

        # Adicionar logo no HTML da impressão
        logo_html = ""
        try:
            from app.config import get_caminho_logo
            from pathlib import Path
            logo_path = get_caminho_logo()
            if logo_path and Path(logo_path).exists():
                # Converter caminho para formato base64 para embutir no HTML
                import base64
                with open(logo_path, 'rb') as f:
                    logo_data = base64.b64encode(f.read()).decode()
                    logo_ext = Path(logo_path).suffix.lower().replace('.', '')
                    logo_html = f'<img src="{logo_data}"
style="max-height:30px; max-width:130px; margin-bottom:5px; display:block;
margin-left:auto; margin-right:auto;">'
        except Exception:
            pass

        html = (
            "<html><head><style>"
            "body { font-family: Arial; font-size: 12pt; }"
```

```
            "h2 { color: #1a3a6a; } h3 { color: #2a5a9a; margin-top:16px; }"
            "p { margin: 4px 0; }"
            "</style></head><body>"
            f"{logo_html}"
            f"<h2>Laudo do Exame</h2>"
            f"<p><b>Paciente:</b> {paciente}    <b>Data:</b> {data}</p>"
            f"<hr/><h3>Descrição / Achados</h3><p>{achados.replace(chr(10),
'<br/>')}</p>"
        )
        if recom.strip():
            html += f"<h3>Recomendações</h3><p>{recom.replace(chr(10),
'<br/>')}</p>"
        html += "</body></html>"

        doc = QTextDocument()
        doc.setHtml(html)
        try:
            _fn = getattr(doc, 'print_', None) or getattr(doc, 'print', None)
            if _fn:
                _fn(printer)
            else:
                raise AttributeError("Método de impressão não encontrado")
        except Exception as e:
            QMessageBox.critical(self, "Imprimir", f"Não foi possível imprimir:\n{e}")

    def exportar_pdf(self):
        if not self.exame_id:
            QMessageBox.warning(self, "Aviso", "Selecione um exame.")
            return
        dados = buscar_exame(self.exame_id)
        if not dados:
            return
        lista_imagens = []
```

```
        base_dir = Path(__file__).resolve().parent.parent
        for e in self._image_entries:
            p = Path(e.get("path", ""))
            if p.exists():
                lista_imagens.append(str(p))
        pasta_pdf = PASTA_LAUDOS_PDF
        pasta_pdf.mkdir(parents=True, exist_ok=True)
        paciente_nome = (dados.get("paciente_nome") or "exame").replace(" ", "_")
        data_ex = (dados.get("data_exame") or "").replace("-", "")
        nome_sugerido = f"laudo_{paciente_nome}_{data_ex}.pdf" if data_ex else
"laudo.pdf"
        path_pdf, _ = QFileDialog.getSaveFileName(
            self, "Salvar laudo em PDF", str(pasta_pdf / nome_sugerido), "PDF (*.pdf)"
        )
        if path_pdf:
            conteudo_txt = (
                self._txt_achados.toPlainText()
                + "\n\nRecomendações:\n"
                + self._txt_recomendacoes.toPlainText()
            )
            gerar_pdf_laudo(
                conteudo_txt, dados, path_pdf,
                incluir_cabecalho=True,
                lista_imagens=lista_imagens if lista_imagens else None,
            )
            QMessageBox.information(self, "OK", f"PDF salvo: {path_pdf}")

    # alias para compatibilidade
    def ao_trocar_exame(self):
        self._ao_trocar_exame()

    def aplicar_template(self):
        self._aplicar_template()
```

```
    def salvar_laudo_tela(self):
        self._salvar()

    def aplicar_tema(self, escuro: bool):
        """Sempre tema escuro."""
        header = self.findChild(QFrame, "LaudoHeader")
        if header:
            header.setStyleSheet(
                "QFrame#LaudoHeader { background:#222736; border-bottom:1px solid
#2a3352; }"
            )
        self.setStyleSheet(
            "QLabel { color:#e8eef8; }"
            "QTextEdit { color:#e8eef8; background:#0a1628; border:1px solid
#2a3d5c; }"
            "QComboBox { color:#e8eef8; background:#0a1628; border:1px solid
#2a3d5c; }"
        )
```

- Tela Principal ou Abertura:

```
# -*- coding: utf-8 -*-
"""
Sistema de Captura de Imagens Médicas - Endoscopia e US
Ponto de entrada da aplicação.
"""

import sys
import os
from pathlib import Path
```

```
# Configurar engine de fonte antes de qualquer import do Qt
os.environ["QT_QPA_PLATFORM"] = "windows:fontengine=freetype"

# Garantir que o projeto está no path
ROOT = Path(__file__).resolve().parent
if str(ROOT) not in sys.path:
    sys.path.insert(0, str(ROOT))

# Quando compilado (.exe), BASE_DIR é a pasta do executável
from app.paths import BASE_DIR, DATA_DIR, BUNDLED_DATA_DIR

# Ao executar como .exe (PyInstaller ou Nuitka): criar pasta data e subpastas na primeira execução
_is_compiled = getattr(sys, "frozen", False) or hasattr(sys, "__compiled__")
if _is_compiled:
    try:
        import shutil
        DATA_DIR.mkdir(parents=True, exist_ok=True)
        (DATA_DIR / "logs").mkdir(parents=True, exist_ok=True)
        (DATA_DIR / "laudos_pdf").mkdir(parents=True, exist_ok=True)
        (DATA_DIR / "exportacoes_pdf").mkdir(parents=True, exist_ok=True)
        (DATA_DIR / "exames").mkdir(parents=True, exist_ok=True)
        # Se ícones/logo vêm do empacotado (ex.: Nuitka onefile) e ainda não estão em DATA_DIR, copiar
        if BUNDLED_DATA_DIR != DATA_DIR and BUNDLED_DATA_DIR.exists():
            for sub in ("icons", "logo"):
```

```
            src = BUNDLED_DATA_DIR / sub
            dst = DATA_DIR / sub
            if src.exists() and src.is_dir() and (not dst.exists() or not list(dst.iterdir())):
                try:
                    shutil.copytree(src, dst, dirs_exist_ok=True)
                except Exception:
                    pass
        # Garantir config.json (em AppData quando .exe) com senha técnica padrão ("tecnico")
        from app.config import load_config
        load_config()
    except Exception:
        try:
            DATA_DIR.mkdir(parents=True, exist_ok=True)
        except Exception:
            pass

from PySide6.QtWidgets import QApplication, QTextEdit, QPlainTextEdit
from PySide6.QtCore import Qt, QElapsedTimer, QCoreApplication
from PySide6.QtGui import QFont, QIcon, QFontDatabase
from ui.main_window import MainWindow
from ui.splash_screen import criar_splash, SPLASH_LARGURA, SPLASH_ALTURA
from app.log_sistema import registrar

def main():
    import traceback as _tb
```

```
    try:
        # Garantir pasta data ao lado do exe (reforço ao iniciar; Nuitka onefile)
        if getattr(sys, "frozen", False) or hasattr(sys, "__compiled__"):
            try:
                DATA_DIR.mkdir(parents=True, exist_ok=True)
                for sub in ("logs", "laudos_pdf", "exportacoes_pdf", "exames"):
                    (DATA_DIR / sub).mkdir(parents=True, exist_ok=True)
                (DATA_DIR / ".Endoscotech").write_text("ok", encoding="utf-8")
            except Exception:
                pass
        # Configurações de renderização de fonte ANTES de criar QApplication
        # Elimina contorno/franja colorida (subpixel artifacts)

QApplication.setHighDpiScaleFactorRoundingPolicy(Qt.HighDpiScaleFactorRoundin
gPolicy.PassThrough)

        # NOTA: AA_UseHighDpiPixmaps e AA_EnableHighDpiScaling são
DEPRECADOS no Qt6
        # High DPI é habilitado automaticamente no Qt6/PySide6

        app = QApplication(sys.argv)
        app.setApplicationName("Endoscopia e US - Captura de Imagens Médicas")
        app.setStyle("Windows")
        # Fonte padrão do Windows com configuração otimizada
        font = QFont("Segoe UI", 10)
        font.setStyleStrategy(QFont.PreferAntialias)
        font.setHintingPreference(QFont.HintingPreference.PreferNoHinting)
```

```
app.setFont(font)

# QSS - remove bordas e estiliza inputs
app.setStyleSheet("""
    /* REMOVE QUALQUER BORDA GLOBAL */
    * {
        border: none;
        outline: none;
    }

    /* INPUTS SEM LINHA */
    QLineEdit, QTextEdit, QPlainTextEdit, QComboBox {
        border: none;
        background-color: #0f1f35;
        padding: 6px;
    }

    /* REMOVE LINHAS DE CONTAINERS */
    QFrame {
        border: none;
    }

    QGroupBox {
        border: 0px;
    }
```

```
    QGroupBox::title {
        border: 0px;
    }

    /* REMOVE GRID DE TABELA */
    QTableWidget {
        border: none;
        gridline-color: transparent;
    }

    /* HEADER SEM LINHA */
    QHeaderView::section {
        border: none;
        background-color: #0f1f35;
    }

    /* REMOVE LINHAS DE SCROLL */
    QScrollArea {
        border: none;
    }
""")

# Splash antes de abrir o sistema
splash = criar_splash()
splash.show()
screen = app.primaryScreen().geometry()
```

```
x = (screen.width() - SPLASH_LARGURA) // 2
y = (screen.height() - SPLASH_ALTURA) // 2
splash.move(x, y)
app.processEvents()

# Progresso real durante carregamento
splash.update_progress(5, "Inicializando sistema...")
app.processEvents()

# Inicializar banco de dados
splash.update_progress(15, "Carregando banco de dados...")
app.processEvents()
from app.database import init_db, remover_exames_orfaos
from app.paths import DATA_DIR
init_db()

# Remover exames órfãos (exames que apontam para pacientes inexistentes)
remover_exames_orfaos()

# Garantir que a coluna sexo exista no banco de dados
import sqlite3
from pathlib import Path
db_path = DATA_DIR / "endoscopia.db"
conn = sqlite3.connect(db_path)
cursor = conn.cursor()
cursor.execute("PRAGMA table_info(pacientes)")
```

```
colunas = cursor.fetchall()
nomes_colunas = [col[1] for col in colunas]
if "sexo" not in nomes_colunas:
    try:
        cursor.execute("ALTER TABLE pacientes ADD COLUMN sexo TEXT")
        conn.commit()
    except sqlite3.OperationalError:
        pass
conn.close()

# Carregar configurações
splash.update_progress(25, "Carregando configurações...")
app.processEvents()
from app.config import load_config
load_config()

# Inicializar componentes principais
splash.update_progress(40, "Carregando interface principal...")
app.processEvents()
registrar("Aplicação iniciada", "INFO")
win = MainWindow()

# 🔥 CORREÇÃO DEFINITIVA DO FIXEDSYS
for w in app.allWidgets():
    f = w.font()
    # Se estiver pedindo fonte monoespaçada → corrige
```

```
            if f.styleHint() == QFont.TypeWriter:
                print("PROBLEMA EM:", w)
                f.setStyleHint(QFont.SansSerif)
                f.setFamily("Segoe UI")
                w.setFont(f)

        # Ícone global
        splash.update_progress(60, "Configurando ícones...")
        app.processEvents()
        for icon_name in ("app.ico", "app.png", "icon.ico", "icon.png"):
            for folder in (BASE_DIR, BUNDLED_DATA_DIR / "icons", DATA_DIR / "icons"):
                icon_path = folder / icon_name
                if icon_path.exists():
                    app.setWindowIcon(QIcon(str(icon_path)))
                    break
            else:
                continue
            break

        # Finalizando carregamento
        splash.update_progress(80, "Finalizando carregamento...")
        app.processEvents()

        # Pequena pausa para visualização
        import time
```

```
        time.sleep(0.5)

        splash.update_progress(100, "Sistema pronto!")
        app.processEvents()
        time.sleep(0.3)

        win.showMaximized()
        splash.finish(win)
        result = app.exec()
        sys.exit(result)
    except Exception as e:
        import traceback
        try:
            registrar(f"Erro ao iniciar aplicação: {e}", "ERRO")
        except Exception:
            pass
        traceback.print_exc()
        try:
            from PySide6.QtWidgets import QApplication as QApp, QMessageBox
            if not QApp.instance():
                _app = QApplication(sys.argv)
            QMessageBox.critical(None, "Erro ao iniciar", f"Falha ao iniciar o
sistema:\n{e}")
        except Exception:
            print(f"Erro ao iniciar: {e}")
        sys.exit(1)
```

```
if __name__ == "__main__":
    try:
        main()
    except Exception as e:
        with open("CRASH.log", "w", encoding="utf-8") as f:
            f.write(f"CRASH: {e}\n")
            import traceback
            f.write(traceback.format_exc())
        input(f"ERRO: {e}. Pressione Enter...")
```

- Cadastro de paciente:

```
# -*- coding: utf-8 -*-
"""Tela de gestão de pacientes — lista com edição e exclusão."""

from pathlib import Path
import sys
sys.path.insert(0, str(Path(__file__).resolve().parent.parent))

from PySide6.QtWidgets import (
    QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QTableWidget, QTableWidgetItem,
    QHeaderView, QFormLayout, QLineEdit, QMessageBox, QLabel, QAbstractItemView,
    QFrame, QDateEdit, QTextEdit, QScrollArea, QStackedWidget, QComboBox,
)
```

```
from PySide6.QtCore import Qt, QDate, QTimer, QSize
from PySide6.QtGui import QIcon

from app.database import listar_pacientes, buscar_paciente, inserir_paciente, atualizar_paciente, excluir_paciente
from app.paths import DATA_DIR


class TelaGestaoPacientes(QWidget):
    def __init__(self, main_window):
        global DATA_DIR
        super().__init__()
        self.main_window = main_window
        self._editando_id = None

        layout = QVBoxLayout(self)
        layout.setContentsMargins(0, 0, 0, 0)
        layout.setSpacing(0)

        action_bar = QFrame()
        action_bar.setFixedHeight(54)
        action_bar.setFrameShape(QFrame.NoFrame)
        action_bar.setStyleSheet("QFrame { background:#0d1a2c; border:none; }")
        al = QHBoxLayout(action_bar)
        al.setContentsMargins(12, 0, 12, 0)
        al.setSpacing(8)
```

```
btn_novo = QPushButton("+ Novo Paciente")
btn_novo.setToolTip("Cadastrar um novo paciente")
btn_novo.setMinimumHeight(34)
btn_novo.setStyleSheet(
    "QPushButton { background:#27ae60; color:#fff; border:1px solid #7af0a0;"
    " border-radius:7px; padding:6px 16px; font-weight:700; }"
    "QPushButton:hover { background:#2ecc71; }"
)
btn_novo.clicked.connect(self.novo)
al.addWidget(btn_novo)
al.addStretch(1)

self.inp_busca = QLineEdit()
self.inp_busca.setFrame(False)
self.inp_busca.setPlaceholderText("🔍  Buscar por nome ou ID…")
self.inp_busca.setMinimumWidth(260)
self.inp_busca.setMinimumHeight(34)
self.inp_busca.setStyleSheet(
    "QLineEdit { background:#0f1a2b; color:#dce6f5; border:1px solid #2a3d5c;"
    " border-radius:7px; padding:4px 12px; font-size:12px; }"
    "QLineEdit:focus { border:1px solid #5a8fd4; }"
)
self.inp_busca.textChanged.connect(self._filtrar)
al.addWidget(self.inp_busca)
```

```
        layout.addWidget(action_bar)


        self.tabela = QTableWidget()
        self.tabela.setColumnCount(7)
        from PySide6.QtGui import QFont
        self.tabela.setFont(QFont("Segoe UI", 10))
        self.tabela.horizontalHeader().setFont(QFont("Segoe UI", 10))
        self.tabela.setHorizontalHeaderLabels(
            ["Nº", "ID", "Nome", "Nascimento", "Observação", "Capturas", "Ações"]
        )
        self.tabela.horizontalHeader().setSectionResizeMode(0,
QHeaderView.ResizeToContents)
        self.tabela.horizontalHeader().setSectionResizeMode(1,
QHeaderView.ResizeToContents)
        self.tabela.horizontalHeader().setSectionResizeMode(2, QHeaderView.Stretch)
        self.tabela.horizontalHeader().setSectionResizeMode(3,
QHeaderView.ResizeToContents)
        self.tabela.horizontalHeader().setSectionResizeMode(4, QHeaderView.Stretch)
        self.tabela.horizontalHeader().setSectionResizeMode(5,
QHeaderView.ResizeToContents)
        self.tabela.horizontalHeader().setSectionResizeMode(6,
QHeaderView.ResizeToContents)
        self.tabela.verticalHeader().setDefaultSectionSize(46)
        self.tabela.setSelectionBehavior(QAbstractItemView.SelectRows)
        self.tabela.setEditTriggers(QAbstractItemView.NoEditTriggers)
        self.tabela.setAlternatingRowColors(True)
        self.tabela.verticalHeader().setVisible(False)
```

```
        self.tabela.setShowGrid(False)
        self.tabela.setFocusPolicy(Qt.NoFocus)

        tabela_container = QWidget()
        tabela_container.setStyleSheet("background:transparent;")
        tc_layout = QVBoxLayout(tabela_container)
        tc_layout.setContentsMargins(12, 8, 12, 12)
        tc_layout.setSpacing(0)
        tc_layout.addWidget(self.tabela)
        layout.addWidget(tabela_container, 1)

        self.lbl_vazio = QLabel(
            "Nenhum paciente cadastrado.\nClique em \"+ Novo Paciente\" para começar."
        )
        self.lbl_vazio.setAlignment(Qt.AlignmentFlag.AlignCenter)
        layout.addWidget(self.lbl_vazio)

        # ── FORMULÁRIO (em modal) ──────────────────────────────────────────

        self._stack = QStackedWidget()
        layout.addWidget(self._stack, 1)

        pagina_lista = QWidget()
        pl = QVBoxLayout(pagina_lista)
        pl.setContentsMargins(0, 0, 0, 0)
        pl.setSpacing(0)
```

```
pl.addWidget(action_bar)
pl.addWidget(tabela_container, 1)
pl.addWidget(self.lbl_vazio)
self._stack.addWidget(pagina_lista)

pagina_form = QWidget()
pf = QVBoxLayout(pagina_form)
pf.setContentsMargins(24, 24, 24, 24)
pf.setSpacing(0)
pf.setAlignment(Qt.AlignTop)

header = QFrame()
header.setFrameShape(QFrame.NoFrame)
header.setStyleSheet("background:transparent; border:none;")
hl = QHBoxLayout(header)
hl.setContentsMargins(0, 0, 0, 0)
hl.setSpacing(0)

breadcrumb = QLabel("Paciente")
breadcrumb.setStyleSheet(
    "color:#e8eef8; font-size:24px; font-weight:700; background:transparent;"
)
hl.addWidget(breadcrumb)
hl.addStretch(1)
pf.addWidget(header)
pf.addSpacing(24)
```

```
        subtitle = QLabel("Cadastre as informações básicas para iniciar um novo exame.")
        subtitle.setStyleSheet("color:#7a9bbf; font-size:14px; background:transparent;")
        pf.addWidget(subtitle)
        pf.addSpacing(24)

        card = QFrame()
        card.setFrameShape(QFrame.NoFrame)
        card.setStyleSheet("QFrame { background:#0d1b2e; border:none; border-radius:12px; }")
        card_layout = QVBoxLayout(card)
        card_layout.setContentsMargins(32, 32, 32, 32)
        card_layout.setSpacing(20)

        _label_base = "color:#9fb3d4; font-size:12px; font-weight:600; background:transparent;"
        _input_style = (
            "QLineEdit, QTextEdit, QDateEdit, QComboBox {"
            " background:#0a1628; color:#dce6f5;"
            " border:1px solid #2a3d5c; border-radius:8px;"
            " padding:12px 14px; font-size:13px; }"
            "QLineEdit:focus, QTextEdit:focus, QDateEdit:focus, QComboBox:focus {"
            " border:1px solid #4cc9f0; }"
            "QComboBox::drop-down { border:none; width:30px; }"
            "QComboBox QAbstractItemView { background:#0a1628; color:#dce6f5; selection-background-color:#1a4a8a; }"
```

```
        )

        def mk_field_label(text, optional=False, required=False):
            if optional:
                lbl = QLabel(f"{text} <span style='color:#7a9bbf; font-
weight:400;'>Opcional</span>")
                lbl.setStyleSheet(_label_base)
                return lbl
            elif required:
                lbl = QLabel(f"{text} <span style='color:#ff6b6b;'>*</span>")
                lbl.setStyleSheet(_label_base)
                return lbl
            else:
                lbl = QLabel(text)
                lbl.setStyleSheet(_label_base)
                return lbl

        row1 = QHBoxLayout()
        row1.setSpacing(20)

        id_col = QVBoxLayout()
        id_col.setSpacing(6)
        id_col.addWidget(mk_field_label("ID / Prontuário", optional=True))
        self._inp_id_clinica = QLineEdit()
        self._inp_id_clinica.setFrame(False)
        self._inp_id_clinica.setPlaceholderText("Ex: 30032026")
```

```
self._inp_id_clinica.setFixedHeight(44)
self._inp_id_clinica.setStyleSheet(_input_style)
id_col.addWidget(self._inp_id_clinica)
row1.addLayout(id_col, 1)

nome_col = QVBoxLayout()
nome_col.setSpacing(6)
nome_col.addWidget(mk_field_label("Nome completo", required=True))
self._inp_nome = QLineEdit()
self._inp_nome.setFrame(False)
self._inp_nome.setPlaceholderText("Ex: José Rodrigo Lopes")
self._inp_nome.setFixedHeight(44)
self._inp_nome.setStyleSheet(_input_style)
nome_col.addWidget(self._inp_nome)
row1.addLayout(nome_col, 2)
card_layout.addLayout(row1)

row2 = QHBoxLayout()
row2.setSpacing(20)

nasc_col = QVBoxLayout()
nasc_col.setSpacing(6)
nasc_col.addWidget(mk_field_label("Data de nascimento"))

nasc_container = QFrame()
nasc_container.setFrameShape(QFrame.NoFrame)
```

```
        nasc_container.setStyleSheet("QFrame { background:#0a1628; border:none;
border-radius:8px; }")
        nasc_hl = QHBoxLayout(nasc_container)
        nasc_hl.setContentsMargins(12, 0, 12, 0)
        nasc_hl.setSpacing(8)

        cal_icon_path = DATA_DIR / "icons" / "file-text.svg"
        if cal_icon_path.exists():
            cal_icon = QLabel()
            cal_icon.setPixmap(QIcon(str(cal_icon_path)).pixmap(16, 16))
        else:
            cal_icon = QLabel("🗓")
            cal_icon.setStyleSheet("font-size:16px; background:transparent;")
        nasc_hl.addWidget(cal_icon)

        self._inp_nascimento = QDateEdit()
        self._inp_nascimento.setCalendarPopup(True)
        self._inp_nascimento.setSpecialValueText("DD / MM / AAAA")
        self._inp_nascimento.setMinimumDate(QDate(1900, 1, 1))
        self._inp_nascimento.setDate(QDate(1900, 1, 1))
        self._inp_nascimento.setFixedHeight(42)
        self._inp_nascimento.setFixedWidth(140)
        self._inp_nascimento.setStyleSheet(
            "QDateEdit { background:transparent; color:#7a9bbf; border:none; font-
size:13px; }"
            "QDateEdit:focus { outline:none; }"
```

```
        )
        nasc_hl.addWidget(self._inp_nascimento)
        nasc_col.addWidget(nasc_container)
        row2.addLayout(nasc_col)

        sexo_col = QVBoxLayout()
        sexo_col.setSpacing(6)
        sexo_col.addWidget(mk_field_label("Sexo"))

        sexo_container = QFrame()
        sexo_container.setFrameShape(QFrame.NoFrame)
        sexo_container.setStyleSheet("QFrame { background:#0a1628; border:none;
border-radius:8px; }")
        sexo_hl = QHBoxLayout(sexo_container)
        sexo_hl.setContentsMargins(12, 0, 12, 0)
        sexo_hl.setSpacing(8)

        sexo_icon_path = DATA_DIR / "icons" / "file-text.svg"
        if sexo_icon_path.exists():
            sexo_icon = QLabel()
            sexo_icon.setPixmap(QIcon(str(sexo_icon_path)).pixmap(16, 16))
        else:
            sexo_icon = QLabel(" 👤 ")
            sexo_icon.setStyleSheet("font-size:16px; background:transparent;")
        sexo_hl.addWidget(sexo_icon)
```

```
        self._inp_sexo = QComboBox()
        self._inp_sexo.addItems(["Masculino", "Feminino"])
        self._inp_sexo.setFixedHeight(42)
        self._inp_sexo.setFixedWidth(147)
        self._inp_sexo.setStyleSheet(
            "QComboBox { background:transparent; color:#7a9bbf; border:none; font-
size:13px; }"
            "QComboBox:focus { outline:none; }"
        )
        sexo_hl.addWidget(self._inp_sexo)
        sexo_col.addWidget(sexo_container)
        row2.addLayout(sexo_col)

        med_col = QVBoxLayout()
        med_col.setSpacing(6)
        med_col.addWidget(mk_field_label("Médico responsável", optional=True))

        self._inp_medico = QLineEdit()
        self._inp_medico.setPlaceholderText("Digite o nome do médico responsável")
        self._inp_medico.setFixedHeight(44)
        self._inp_medico.setStyleSheet(_input_style)
        med_col.addWidget(self._inp_medico)
        row2.addLayout(med_col, 1)
        card_layout.addLayout(row2)

        obs_col = QVBoxLayout()
```

```
        obs_col.setSpacing(6)
        obs_col.addWidget(mk_field_label("Observações", optional=True))

        self._inp_observacoes = QTextEdit()
        self._inp_observacoes.setPlaceholderText("Observações gerais sobre o paciente...")
        self._inp_observacoes.setFixedHeight(100)
        self._inp_observacoes.setStyleSheet(_input_style)
        self._inp_observacoes.setFont(QFont("Segoe UI", 10))
        obs_col.addWidget(self._inp_observacoes)

        self._lbl_contador = QLabel("0 / 500")
        self._lbl_contador.setAlignment(Qt.AlignRight)
        self._lbl_contador.setStyleSheet("color:#5a7a9a; font-size:11px; background:transparent;")
        obs_col.addWidget(self._lbl_contador)

        self._inp_observacoes.textChanged.connect(lambda:
self._lbl_contador.setText(f"{len(self._inp_observacoes.toPlainText())} / 500"))
        card_layout.addLayout(obs_col)

        buttons = QHBoxLayout()
        buttons.setSpacing(12)
        buttons.addStretch(1)

        btn_cancelar = QPushButton("  ×  Cancelar")
```

```
        btn_cancelar.setFixedHeight(44)
        btn_cancelar.setFixedWidth(140)
        btn_cancelar.setStyleSheet(
            "QPushButton { background:#1c283d; color:#dce6f5; border:1px solid #3a4d6e;"
            " border-radius:8px; font-size:13px; font-weight:500; }"
            "QPushButton:hover { background:#24344d; }"
        )
        btn_cancelar.clicked.connect(self._cancelar_form)
        self._btn_cancelar = btn_cancelar
        buttons.addWidget(btn_cancelar)

        self._btn_salvar = QPushButton("Salvar paciente")
        self._btn_salvar.setFixedHeight(44)
        save_icon_path = DATA_DIR / "icons" / "save.svg"
        if save_icon_path.exists():
            self._btn_salvar.setIcon(QIcon(str(save_icon_path)))
            self._btn_salvar.setIconSize(QSize(18, 18))
        self._btn_salvar.setStyleSheet(
            "QPushButton { background:#1a7a3a; color:#fff; border:1px solid #30c060;"
            " border-radius:8px; font-size:13px; font-weight:600; padding:0 20px; }"
            "QPushButton:hover { background:#22964a; }"
        )
        self._btn_salvar.clicked.connect(self._salvar_form)
        buttons.addWidget(self._btn_salvar)
        card_layout.addLayout(buttons)
```

```
        security = QLabel("🔒  Seus dados estão protegidos e são usados apenas durante o exame.")
        security.setAlignment(Qt.AlignCenter)
        security.setStyleSheet("color:#5a7a9a; font-size:12px; background:transparent; margin-top:16px;")
        card_layout.addWidget(security)

        pf.addWidget(card)
        pf.addStretch(1)

        self._stack.addWidget(pagina_form)

        self.atualizar_dados()
        self.aplicar_tema(True)

        self._stack.setCurrentIndex(0)

    def aplicar_tema(self, escuro: bool):
        """Sempre tema escuro."""
        self.setStyleSheet(
            """
            QWidget { color:#e8eef8; background:transparent; }
            QHeaderView::section {
                background:#222736; color:#9fb3d4;
                border:none; border-bottom:1px solid #2a3d5c;
```

```
            padding:8px 12px; font-size:11px; font-weight:700;
        }
        QTableWidget {
            background:#0d1b2e; color:#dce6f5;
            alternate-background-color:#0f2035;
            gridline-color:#1a2d48;
            border:1px solid #2a3d5c; border-radius:10px;
            selection-background-color:#1a4a8a;
            selection-color:#ffffff; font-size:13px;
        }
        QTableWidget::item { padding:6px 12px; border:none; }
        QTableWidget::item:hover { background:#162840; }
        QTableCornerButton::section { background:#222736; border:none; }
        """
    )
    self.lbl_vazio.setStyleSheet(
        "color:#6b7280; padding:32px; font-size:13px; background:transparent;"
    )

def _filtrar(self, texto: str):
    texto = texto.strip().lower()
    for row in range(self.tabela.rowCount()):
        nome = self.tabela.item(row, 2)
        id_ = self.tabela.item(row, 1)
        visivel = (not texto) or (
            (nome is not None and texto in nome.text().lower()) or
```

```
            (id_ is not None and texto in id_.text().lower())
        )
        self.tabela.setRowHidden(row, not visivel)

def atualizar_dados(self):
    self.tabela.setRowCount(0)
    self.tabela.clearContents()

    pacientes = listar_pacientes()
    self.lbl_vazio.setVisible(len(pacientes) == 0)
    for row, p in enumerate(pacientes):
        self.tabela.insertRow(row)
        item_num = QTableWidgetItem(str(row + 1))
        item_num.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
        self.tabela.setItem(row, 0, item_num)

        item_id = QTableWidgetItem(p.get("id_clinica") or "")
        item_id.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
        self.tabela.setItem(row, 1, item_id)

        self.tabela.setItem(row, 2, QTableWidgetItem(p.get("nome", "")))

        nasc = p.get("data_nascimento") or ""
        if nasc and "-" in nasc:
            try:
                parts = nasc.split("-")
```

```
                nasc = f"{parts[2]}/{parts[1]}/{parts[0]}"
            except Exception:
                pass
        item_nasc = QTableWidgetItem(nasc)
        item_nasc.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
        self.tabela.setItem(row, 3, item_nasc)

        obs = (p.get("observacoes") or "").strip()
        if len(obs) > 60:
            obs = obs[:57] + "..."
        item_obs = QTableWidgetItem(obs)
        item_obs.setToolTip(p.get("observacoes") or "")
        self.tabela.setItem(row, 4, item_obs)

        total_cap = p.get("total_capturas", 0) or 0
        item_cap = QTableWidgetItem(str(total_cap) if total_cap else "—")
        item_cap.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
        self.tabela.setItem(row, 5, item_cap)

        btn_editar = QPushButton("Editar")
        btn_excluir = QPushButton("Excluir")
        btn_editar.setFixedSize(80, 28)
        btn_excluir.setFixedSize(80, 28)
        btn_editar.setStyleSheet(
            "QPushButton { background:#1c283d; color:#dce6f5; border:1px solid
#3a4d6e;"
```

```
            " border-radius:5px; font-size:11px; }"
            "QPushButton:hover { background:#24344d; }"
        )
        btn_excluir.setStyleSheet(
            "QPushButton { background:#7a1010; color:#fff; border:1px solid #a03030;"
            " border-radius:5px; font-size:11px; }"
            "QPushButton:hover { background:#a01818; }"
        )
        btn_editar.clicked.connect(lambda checked, id_=p["id"]: self.editar(id_))
        btn_excluir.clicked.connect(lambda checked, id_=p["id"]: self.excluir(id_))

        old = self.tabela.cellWidget(row, 6)
        if old:
            old.deleteLater()

        cell = QWidget()
        cell.setStyleSheet("background:transparent;")
        h = QHBoxLayout(cell)
        h.setContentsMargins(4, 2, 16, 2)
        h.setSpacing(6)
        h.addWidget(btn_editar)
        h.addWidget(btn_excluir)
        h.addStretch()
        self.tabela.setCellWidget(row, 6, cell)

    def _abrir_form(self, paciente_id=None):
```

```
self._editando_id = paciente_id
if paciente_id:
    p = buscar_paciente(paciente_id)
    if p:
        self._inp_id_clinica.setText(p.get("id_clinica") or "")
        self._inp_nome.setText(p.get("nome") or "")
        self._inp_observacoes.setPlainText(p.get("observacoes") or "")
        dn = (p.get("data_nascimento") or "").strip()
        if dn:
            try:
                self._inp_nascimento.setDate(QDate.fromString(dn, "yyyy-MM-dd"))
            except Exception:
                self._inp_nascimento.setDate(QDate(1900, 1, 1))
        else:
            self._inp_nascimento.setDate(QDate(1900, 1, 1))
        self._inp_medico.setText(p.get("medico_nome") or "")
        sexo = p.get("sexo") or ""
        if sexo == "M":
            self._inp_sexo.setCurrentIndex(0)
        elif sexo == "F":
            self._inp_sexo.setCurrentIndex(1)
        else:
            self._inp_sexo.setCurrentIndex(0)
else:
    self._inp_id_clinica.clear()
    self._inp_nome.clear()
```

```
            self._inp_observacoes.clear()
            self._lbl_contador.setText("0 / 500")
            self._inp_nascimento.setDate(QDate(1900, 1, 1))
            self._inp_medico.clear()
            self._inp_sexo.setCurrentIndex(0)
        self._stack.setCurrentIndex(1)
        self._inp_nome.setFocus()

    def _cancelar_form(self):
        self._stack.setCurrentIndex(0)

    def _salvar_form(self):
        nome = self._inp_nome.text().strip()
        if not nome:
            QMessageBox.warning(self, "Aviso", "O campo Nome é obrigatório.")
            self._inp_nome.setFocus()
            return
        id_clinica = self._inp_id_clinica.text().strip()
        observacoes = self._inp_observacoes.toPlainText().strip()
        dn = self._inp_nascimento.date().toString("yyyy-MM-dd") \
            if self._inp_nascimento.date() > QDate(1900, 1, 1) else ""
        medico_nome = self._inp_medico.text().strip()
        sexo = "M" if self._inp_sexo.currentIndex() == 0 else "F"

        if self._editando_id:
            atualizar_paciente(
```

```
                self._editando_id,
                nome=nome, id_clinica=id_clinica,
                cpf="", data_nascimento=dn,
                telefone="", email="",
                observacoes=observacoes,
                medico_nome=medico_nome,
                sexo=sexo,
            )
            paciente_id = self._editando_id
        else:
            paciente_id = inserir_paciente(
                nome=nome, id_clinica=id_clinica,
                cpf="", data_nascimento=dn,
                telefone="", email="",
                observacoes=observacoes,
                medico_nome=medico_nome,
                sexo=sexo,
            )
        self._cancelar_form()
        self.atualizar_dados()

    def novo(self):
        self._abrir_form()

    def editar(self, id_):
        self._abrir_form(id_)
```

```
    def excluir(self, id_):
        resp = QMessageBox.question(
            self, "Confirmar",
            "Excluir este paciente? Esta ação não pode ser desfeita.",
            QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No,
            QMessageBox.StandardButton.No,
        )
        if resp == QMessageBox.StandardButton.Yes:
            excluir_paciente(id_)
            self.atualizar_dados()
```

- Tela principal de abertura:

```
import subprocess
import time
import psutil
import os
import sys
import logging
import keyboard  # pip install keyboard


# ===== CONFIG =====
# Pasta onde está este script/exe (ao instalar, mesma pasta do Endoscotech.exe)
if getattr(sys, "frozen", False):
    _BASE_DIR = os.path.dirname(os.path.abspath(sys.executable))
else:
    _BASE_DIR = os.path.dirname(os.path.abspath(__file__))
```

```
# Garantir pasta de trabalho = pasta do exe (importante quando inicia pelo atalho do Windows)
try:
    os.chdir(_BASE_DIR)
except Exception:
    pass

APP_PATH = os.path.join(_BASE_DIR, "Endoscotech.exe")
PROCESS_NAME = "Endoscotech.exe"
CHECK_INTERVAL = 3
STARTUP_DELAY = 1
# App e log em %LOCALAPPDATA%\EndoscotechApp (sempre gravável; evita falha em Program Files)
_appdata = os.environ.get("LOCALAPPDATA") or os.environ.get("APPDATA") or os.path.expanduser("~")
_CONFIG_DIR = os.path.join(_appdata, "EndoscotechApp")
PARAR_FILE = os.path.join(_CONFIG_DIR, "watchdog_parar.flag")
EXIT_HOTKEY = "ctrl+shift+alt+f12"
LOG_FILE = os.path.join(_CONFIG_DIR, "watchdog.log")

# ===== LOG =====
try:
    os.makedirs(_CONFIG_DIR, exist_ok=True)
except Exception:
    pass
logging.basicConfig(
```

```
    filename=LOG_FILE,
    level=logging.INFO,
    format="%(asctime)s - %(message)s"
)

def is_process_running(name):
    for proc in psutil.process_iter(['name']):
        if proc.info['name'] == name:
            return True
    return False

def start_app():
    try:
        subprocess.Popen([APP_PATH], cwd=_BASE_DIR)
        logging.info("Aplicativo iniciado.")
    except Exception as e:
        logging.error(f"Erro ao iniciar app: {e}")

def sair_watchdog():
    """Chamado quando o usuário pressiona o atalho de saída técnica."""
    try:
        logging.info("Saída técnica acionada.")
        os.startfile("explorer.exe")
    except Exception as e:
        logging.error(f"Erro ao abrir Explorer: {e}")
    sys.exit(0)
```

```
def main():
    logging.info("Watchdog iniciado.")
    # Registra o atalho global. No Windows, execute como Administrador para funcionar.
    try:
        keyboard.add_hotkey(EXIT_HOTKEY, sair_watchdog, suppress=False)
        logging.info(f"Atalho de saída registrado: {EXIT_HOTKEY}")
    except Exception as e:
        logging.error(f"Atalho não registrado: {e}. Execute como Administrador.")

    time.sleep(STARTUP_DELAY)

    while True:
        # Se o app criou o arquivo de parar (menu técnico > Encerrar watchdog), sair
        if os.path.isfile(PARAR_FILE):
            try:
                os.remove(PARAR_FILE)
            except Exception:
                pass
            logging.info("Saída solicitada pelo app (watchdog_parar.flag).")
            try:
                os.startfile("explorer.exe")
            except Exception:
                pass
            sys.exit(0)
```

```
        if not is_process_running(PROCESS_NAME):
            logging.warning("App não encontrado. Reiniciando...")
            start_app()
        time.sleep(CHECK_INTERVAL)

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

4.6. Integração com o Padrão PACS/DICOM

O desenvolvimento do software do projeto Endoscotech engloba uma funcionalidade de altíssima relevância e complexidade no cenário de Tecnologia da Informação em Saúde (Health IT): a comunicação com sistemas PACS (Picture Archiving and Communication System) utilizando o padrão universal DICOM (Digital Imaging and Communications in Medicine).

Como observado no método “enviar_para_pacs(self)” implementado no código Python, o software possibilita que, ao final do exame endoscópico, o veterinário não mantenha as imagens capturadas presas isoladamente no disco rígido local da estação de trabalho. O código utiliza bibliotecas Python de manipulação de dados e redes DICOM (como pydicom e pynetdicom) para envelopar os arquivos das imagens em pacotes formatados com metadados obrigatórios do paciente, do médico e da instituição de origem. Esses pacotes são então transmitidos com segurança sobre redes TCP/IP por meio dos chamados *Application Entities* (AE Titles), portas e IPs configurados no software.

Essa implementação resolve um dos maiores problemas de continuidade do fluxo hospitalar. Em hospitais veterinários complexos, enquanto a endoscopia ocorre na sala de cirurgia, radiologistas ou especialistas localizados em outras alas ou até remotamente podem puxar ("Retrieve") ou receber ("Store") as imagens em seus próprios visualizadores PACS (como ClearCanvas ou sistemas web) imediatamente.

Isso agrega valor imensurável ao projeto, elevando a aplicação do status de um mero visualizador de câmera para um sofisticado nó de uma rede hospitalar 4.0.

5. DOCUMENTAÇÃO TÉCNICA

Sistema de Endoscopia Endoscotech

Empresa: Endoscotech

Produto: Endoscotech

Status do Produto: Em operação

Ambiente de uso: Clínica Veterinária (expansível para médica)

5.1. Visão Geral do Sistema

O Endoscotech é um sistema proprietário destinado à captura, visualização e armazenamento de imagens endoscópicas em tempo real. A solução foi projetada para operação contínua em ambiente clínico veterinário, com arquitetura tipo appliance e interface restrita ao usuário final.

O sistema utiliza hardware padrão de mercado combinado com software proprietário da Endoscotech, priorizando estabilidade operacional, rastreabilidade de eventos e futura interoperabilidade com servidores PACS por meio do padrão DICOM.

Observação de interoperabilidade: O Endoscotech encontra-se tecnicamente preparado para integração com servidores PACS padrão de mercado, estando a implementação final sujeita a validação conjunta com parceiro tecnológico.

5.2. Finalidade Pretendida:

- Aquisição de imagens endoscópicas
- Visualização em tempo real do exame
- Registro de imagens por acionamento externo (pedal USB)
- Armazenamento local estruturado
- Preparação para interoperabilidade com PACS

Classificação regulatória: Em avaliação (uso veterinário no estágio atual).

5.3. Arquitetura Técnica

5.3.1. Plataforma

- Sistema operacional: Windows 10 Pro / Windows 11 Pro
- Linguagem principal: Python (empacotado com PyInstaller)
- Framework de vídeo: Baseado em drivers UVC/DirectShow do Windows
- Tipo de captura: Compatível com placas USB padrão UVC (AV e HDMI)

5.3.2. Hardware de Referência

- CPU: Intel Core i3 7ª geração ou superior
- Memória RAM: 8 GB
- Armazenamento: SSD
- Interface: USB padrão
- Dispositivo de captura: Placas USB UVC compatíveis com mercado

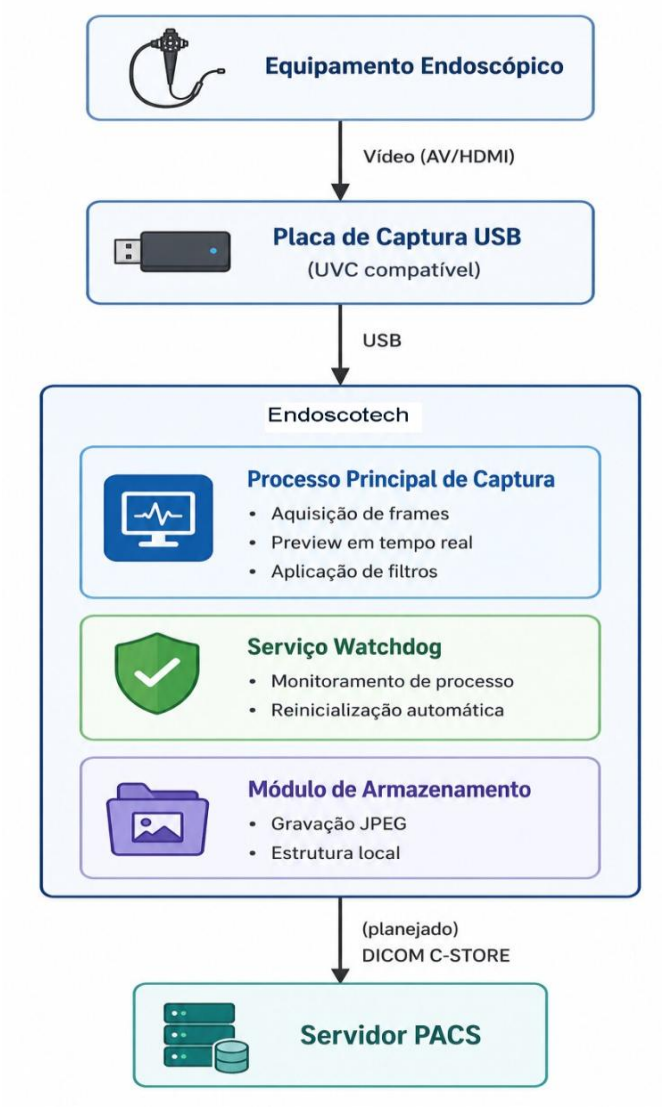
5.4. Estrutura de Processos

5.4.1. Arquitetura projetada com foco em resiliência operacional:

- Processo principal de captura
- Serviço watchdog independente
- Inicialização automática com o sistema operacional
- Reinicialização automática em caso de falha
- Operação em modo kiosk restrito

5.4.2. Diagrama de Arquitetura

Figura 21: Diagrama de arquitetura do sistema

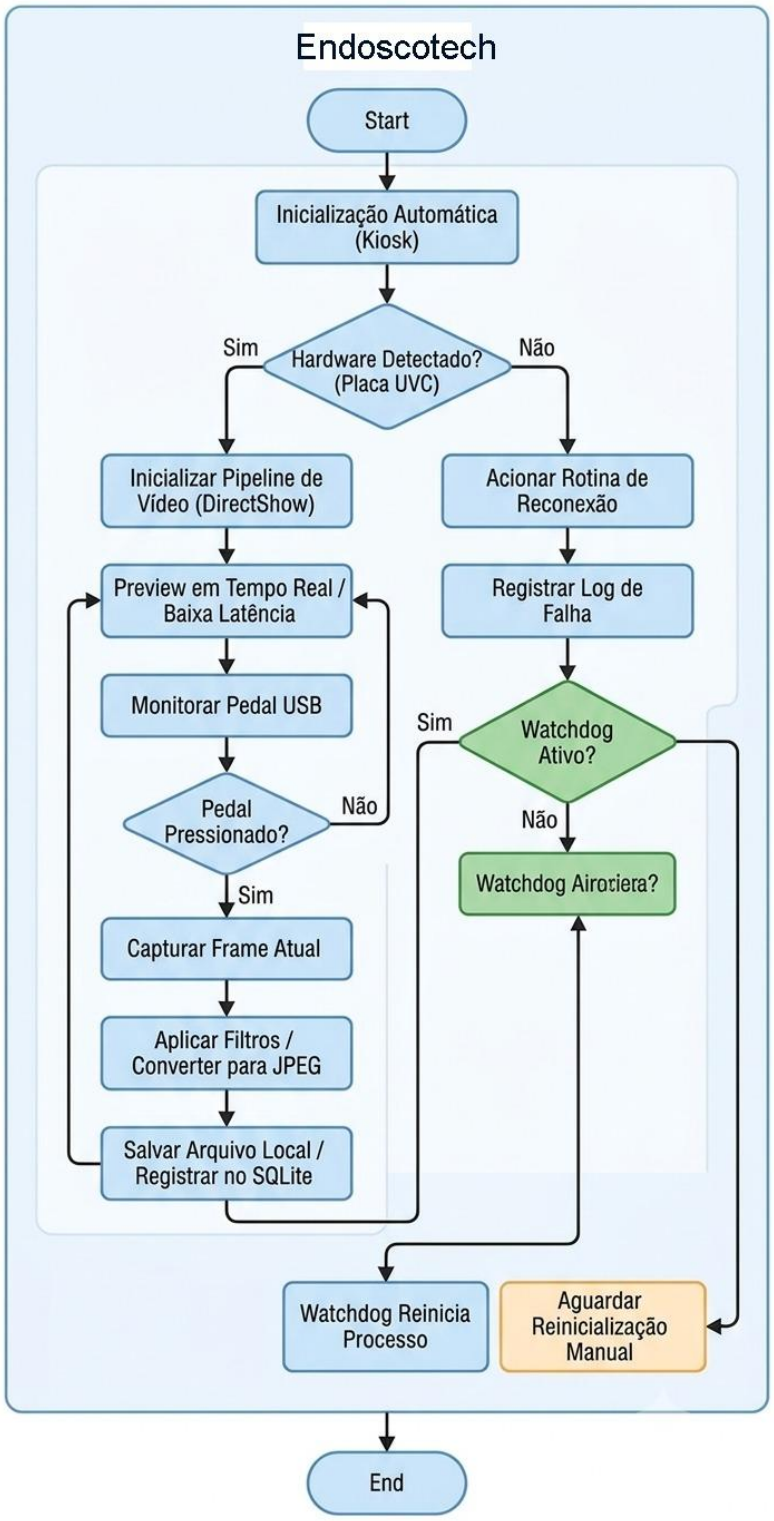


Fonte: Elaborado pelo autor, com base em DBdiagram, 2026.

5.4.3. Fluxo de Aquisição de Imagem

5.4.3.1. Detecção da Placa de Captura

Figura 22 – Fluxograma lógico do processo de aquisição de imagens e resiliência do Endoscotech.



.

Fonte: Elaborado pelo autor, com base em DBdiagram, 2026.

Durante a inicialização, o Endoscotech detecta automaticamente dispositivos de captura compatíveis conectados, via USB.

- Inicialização do Stream

O pipeline de vídeo é iniciado automaticamente após a detecção válida do dispositivo.

- Preview em Tempo Real

Visualização contínua do exame com baixa latência.

- Captura de Imagem

Acionamento por pedal USB configurável.

- Armazenamento Local

Imagens armazenadas localmente em estrutura organizada.

- Integração PACS

Funcionalidade em desenvolvimento e sujeita a validaação conjunta com parceiro PACS.

Especificações de Imagem

- Formato atual: JPEG
- Resolução máxima: 1920 x 1080
- Resolução mínima: até 640 x 480
- Modo de captura: imagem estática via pedal
- Profundidade de cor: conforme dispositivo
- Integração DICOM
- Status: Em desenvolvimento
- Estratégia prevista:
    - Conversão JPEG → DICOM
    - Envio via C-STORE
    - Configuração de AE Title
    - Parametrização de destino PACS

Planejado: MWL, MPPS e Storage Commitment.

Nota: A interoperabilidade final será validada em conjunto com fornecedor PACS compatível.

5.5 Confiabilidade e Segurança operacional

- Recursos implementados:
- Watchdog ativo
- Inicialização automática
- Modo kiosk
- Recuperação automática de falhas
- Proteção contra encerramento acidental
- Sistema abrangente de logs

Queda de Energia

Após religamento do computador, o sistema retorna à condição operacional normal.

FMEA — Análise de Modos de Falha (IEC 62304 / ISO 14971 — preliminar)

ID	Função	Modo de falha	Efeito
F1	Captura de vídeo	Travamento do processo	Interrupção do exame
F2	Interface USB	Desconexão da placa	Perda de imagem
F3	Armazenamento	Falha de gravação	Perda de registro
F4	Operação do usuário	Encerramento indevido	Interrupção
F5	Energia	Queda elétrica	Indisponibilidade

Parte da Tabela continua aqui

ID	Severidade	Ocorrência	Detecção	Mitigação
F1	Média	Baixa	Alta	Watchdog.com reinício
F2	Média	Média	Média	Rotina de reconexão
F3	Alta	Baixa	Média	Logs+validação de escrita
F4	Média	Baixa	Alta	Modo kiosk
F5	Média	Média	Alta	Auto start no boot

Status: análise preliminar em evolução conforme ciclo de desenvolvimento.

5.6 Controle de Versão do Software

- O Endoscotech adota práticas de rastreabilidade de software alinhadas a boas práticas de ciclo de vida.
- Controles implementados:
- Identificação de versão no build
- Empacotamento controlado (PyInstaller)
- Registro de alterações (changelog)
- Controle interno de releases pela Endoscotech
- Objetivo: garantir repetibilidade, rastreabilidade e suporte estruturado.

5.7 Requisitos de Hardware

- Mínimo recomendado:
- Intel Core 4ª geração ou superior
- 8 GB RAM
- SSD
- Porta USB padrão
- Requisitos de Rede (Integração PACS)
- Protocolo: TCP/IP
- Portas DICOM: conforme servidor
- Rede local: não obrigatória para operação básica
- Envio remoto: previsto

5.8 Logs e Diagnóstico

Logs armazenados em diretório dedicado com opção de exportação.

Eventos registrados:

- Rede
- Hardware
- Sistema operacional
- Software Endoscotech

Logs podem ser fornecidos ao parceiro PACS para troubleshooting.

5.9 Backup e Suporte

Rotinas de backup implementadas

Suporte remoto disponível

Estrutura preparada para manutenção assistida

5.10 Roadmap tecnológico

- Integração DICOM completa
- Worklist
- Envio automático ao PACS
- Interoperabilidade avançada

Contato técnico:

Endoscotech — Desenvolvimento Endoscotech

Documento técnico preliminar para avaliação de parceria PACS.

Anexo A — ROBUSTEZ TÉCNICA E INTEROPERABILIDADE

A.1 Status de Integração DICOM

O Endoscotech encontra-se em fase de preparação para interoperabilidade DICOM. A arquitetura já contempla a cadeia de conversão JPEG → DICOM e envio via C-STORE.

Situação atual:

- Pipeline de captura validado
- Estrutura preparada para encapsulamento DICOM
- Parametrização de destino PACS prevista

Próximos marcos técnicos:

- Implementação operacional de C-STORE
- Elaboração do DICOM Conformance Statement

- Validação de interoperabilidade com parceiro PACS

A.2 Ensaios de Verificação e Validação Executados

Foram conduzidos testes funcionais e de robustez com foco em operação clínica contínua.

Testes realizados:

- Teste de inicialização automática após reboot
- Teste de operação contínua prolongada
- Teste de encerramento forçado do processo
- Teste de desconexão e reconexão de dispositivo USB
- Teste de integridade de gravação de imagens

Resultado geral: comportamento estável dentro do estágio atual de desenvolvimento.

A.3 Estratégia de Recuperação de Dispositivo USB

O sistema implementa rotina de detecção e recuperação de falhas de captura.

Comportamento previsto:

- Monitoramento contínuo do dispositivo de vídeo
- Tentativa automática de reconexão
- Reinicialização do pipeline de captura quando necessário
- Atuação do watchdog em caso de falha persistente

Este mecanismo visa reduzir intervenção do operador e manter continuidade operacional.

A.4 Gestão Contínua de Risco

A análise de risco do Endoscotech é um processo contínuo ao longo do ciclo de desenvolvimento.

Práticas adotadas:

- Revisão periódica da FMEA
- Monitoramento via logs de campo
- Atualização de controles de mitigação quando aplicável

Fim do Anexo A

6. AMBIENTE DE IMPLANTAÇÃO E LOCALIZAÇÃO DO SISTEMA

O sistema **Endoscotech** foi implantado e validado em ambiente operacional real, configurado como um *appliance* médico-veterinário de uso dedicado.

Localização Física e Institucional

A unidade piloto de validação do software está localizada e operacional no seguinte endereço:

- **Instituição:** Animed Medicina Veterinária Avançada
- **Endereço:** Rua Saldanha Marinho, 2361, Parque Industrial
- **Município:** São José do Rio Preto – SP
- **Coordenadas Geográficas de Referência:** Região Noroeste Paulista

Figura 23 – Localização geográfica da unidade piloto de implantação do sistema Endoscotech.

Fonte: Elaborado pelo autor, com base em Google Maps (2026).

O ambiente clínico foi selecionado por apresentar a densidade de exames necessária para os testes de estresse, robustez de hardware e verificação do tempo de resposta do pedal USB em procedimentos reais.

7. RESULTADOS E DISCUSSÕES

Através das metodologias implementadas, do sólido levantamento bibliográfico e do intenso planejamento do ciclo de desenvolvimento ciber-físico, o sistema "Endoscotech" superou a fase de prova de conceito e demonstrou estabilidade de operação clínica invejável. Os módulos de captura via interruptor mecânico de pedal acoplados ao microcontrolador inteligente (ESP32) alcançaram com sucesso a integração sem falhas com o software Python, devido à eficiente emulação da classe USB HID. Esta abordagem tecnológica aumentou a produtividade clínica, permitindo ao médico tirar as fotos no momento exato que precisa além de deixar o funcionário que ficaria apenas apertando a tecla, livre para ajudar em outros setores, o que se traduz em maior lucratividade em um mundo onde tempo é dinheiro.

Paralelamente, a robusta arquitetura do banco de dados relacional (SQLite) interconectado com a interface do PySide6 assegurou que os registros médicos se mantenham salvaguardados sem redundâncias prejudiciais, permitindo buscas, cruzamentos e consultas históricas eficazes dos laudos exportados (PDF) e das mídias digitais extraídas dos pacientes. A funcionalidade avançada de exportação via PACS demonstrou que sistemas desenvolvidos como projetos tecnológicos em escolas técnicas podem apresentar características escaláveis e intercambiáveis perfeitamente comparáveis aos dispendiosos softwares proprietários de grandes multinacionais. A evolução, atestada pelo design sofisticado e mais efetivo, praticidade do pedal e segurança gerada pelo sistema da caixa de tomadas (com o IDR e o DPS), atestou de forma inconteste as especificações de orçamento reduzido e atestou a irrefutável viabilidade técnica da concepção de matrizes elétricas aplicadas ao ambiente das disciplinas do Trabalho de Conclusão de Curso (TCC).

7.1. Estudo Corporativo de Mercado

O rigoroso estudo da viabilidade comercial e econômica de um projeto de engenharia biomédica voltado à modernização tecnológica de ecossistemas de saúde exige,

como premissa absoluta, a compreensão estrutural da precificação e das ofertas sistêmicas realizadas pelos oligopólios operantes no mercado nacional. O presente projeto fundamenta sua exequibilidade ao propor o desenvolvimento e a implantação de um sistema de captura de imagens médicas altamente avançado e intuitivo, cujo Valor Geral de Vendas (VGV) final foi posicionado no limiar competitivo de R$ 3.500,00 por unidade implantada. Para fundamentar a viabilidade intrínseca desta meta e validar matematicamente as margens do modelo de negócios, conduziu-se um extensivo levantamento focado nos componentes de custo primário e na política de preços das empresas concorrentes predominantes: a OCRAM Sistemas e a Zscan Software.

A elaboração física da solução repousou no princípio do *Lean Engineering*, focando na obtenção de materiais de infraestrutura de alta conformidade e baixo custo global. A planilha orçamentária do projeto inicial demonstrou um custeio total de materiais (BOM) fixado em R$ 382,45. Este montante suporta com folga a infraestrutura eletroeletrônica, abrangendo desde a fundamental interface de controle microprocessada via placa ESP32-S3 (orçada em R$ 70,00) que conferirá agilidade sem fio ao sistema, até o acionador mecânico de chão em si (pedal a R$ 70,00). Além do núcleo de processamento, o valor total de materiais já incorpora R$ 242,45 alocados exclusivamente para robustecer a segurança física e elétrica do equipamento perante os padrões da ABNT, financiando a integração de Dispositivos de Proteção contra Surtos (DPS), Interruptores Diferenciais Residuais (IDR), caixas de proteção IP 68 e insumos de blindagem adquiridos em revendedores industriais. Quando somados aos custos intelectuais, calculados em R$ 2.500,00 referentes às horas técnicas de desenvolvimento, arquitetura de programação do software proprietário e o serviço braçal de implantação, o custo final da base de entrega do produto é formatado em R$ 2.882,45. Contrastando esse número com a meta de venda estipulada, o projeto gera uma robusta margem de lucro operacional unitária de R$ 617,55, ratificando indubitavelmente sua saúde financeira estrutural primária.

Contudo, a verdadeira força disruptiva da precificação de R$ 3.500,00 revela-se em sua total plenitude ao confrontar o cenário hegemônico de corporações como a Zscan Software e a OCRAM. A OCRAM Sistemas, pioneira no interior de São Paulo, atua com sistemas focados em captura de laudos com braços até na medicina veterinária, evidenciando licitações governamentais pontuais que apontam vendas sistêmicas

próximas a R$ 1.380,00, muitas vezes apenas para licenciamentos primários. No entanto, a balança comercial e o Custo Total de Propriedade (TCO) das clínicas brasileiras são severamente governados pelas tabelas corporativas da Zscan.

A inteligência de mercado demonstra que o modelo de faturamento da Zscan se consolida por meio da hiper-modularização dos seus produtos periféricos, onerando pesadamente o setor médico. Um simples olhar sobre as cotações oficiais aponta que, na configuração tradicional analógica com fio hoje considerada básica e intrusiva no centro operatório a Zscan comercializa apenas as suas placas de interface (Zscan EVO) por R$ 1.050,00, atrelando a venda de pesados e caros pedais mecânicos de chão por R$ 965,00. A mera aquisição de cabos vitais de reposição de vídeo S-Video flutua no custo opressivo de R$ 325,00.

O diagnóstico comparativo torna-se dramático quando clínicas demandam as conveniências ergonômicas inerentes ao controle de infecção e prevenção de riscos através do paradigma sem fio. A Zscan capitaliza essa necessidade vendendo interfaces de rádio frequência (RF), cobrando abusivos R$ 1.449,00 pelo adaptador de botão para endoscópios das marcas Pentax ou Fujinon, e idênticos R$ 1.449,00 para o pedal wireless. Se uma instituição busca a atualização final para capturas em Alta Definição (HD/SDI), somente o componente de conversão externa custará ao hospital inacreditáveis R$ 6.690,00.

Vale enfatizar que todos estes custos referem-se à base puramente física. O direito de usar o sistema exige a compra de licenças de software isoladas. Consultas recentes ao Portal Nacional de Contratações Públicas demonstram vultosos repasses governamentais para a renovação de licenças de software dessa marca, com prefeituras firmando contratos por inexigibilidade que superam cifras de R$ 5.388,00, indicando um monopólio silencioso do fornecimento da gestão digital das imagens dos pacientes.

Sob a luz desta profunda disfunção de preços instaurada pelas marcas vigentes, o desenvolvimento do presente equipamento a R$ 3.500,00, que embutirá o hardware microprocessado, o dispositivo mecânico e a implantação sistêmica do software unificado em um pacote conciso e definitivo, é economicamente transformador. Ao calcular o abismo competitivo, conclui-se que o presente projeto possibilita a uma instituição de saúde adquirir e instalar uma infraestrutura tecnológica com níveis de comunicação sem fio via o avançado chip ESP32 por um valor que mal ultrapassa o

custo de um único pedal com botões analógicos das concorrentes (R$ 2.499,00 em um sistema misto Zscan). O posicionamento proposto erradica a imprevisibilidade de custos e as pesadas taxas de licenciamento passivas, garantindo flexibilidade operacional e atestando que a proposta não apenas cumpre os requisitos rigorosos de viabilidade técnica da engenharia biomédica nacional, mas demonstra possuir uma escalabilidade de negócios singular e extremamente promissora no mercado de consultórios independentes e policlínicas populares.

7.1.1. Engenharia de Precificação: O Abismo Competitivo

Para substanciar com clareza cristalina a viabilidade estonteante de um projeto orçado e vendido a R$ 3.500,00, é mandatório desconstruir, componente por componente, a tabela de preços inflacionada praticada pela Zscan. O mercado de equipamentos médicos tradicionalmente camufla margens de lucro exorbitantes sob a justificativa de custos inerentes a certificações e "tecnologia de ponta médica". Contudo, quando a tabela de preços é confrontada com a realidade eletrônica, o abismo competitivo revela-se insustentável.

Os dados mercadológicos oficiais, obtidos de distribuidores da Zscan, revelam a seguinte estrutura de comercialização para equipamentos que, funcionalmente, realizarão a mesma tarefa do projeto acadêmico de R$ 3.500,00:

Hardware Periférico e Soluções (Zscan)	Especificação e Aplicabilidade	Preço Varejo (R$)
Cabo de Vídeo Simples (BNC/RCA)	Transmissão analógica básica de 5 Metros	325,00
Cabo de Vídeo S-Video	Transmissão de vídeo de 5 Metros	325,00
Botão de Captura Físico com Fio	Adaptador para botão de Endoscópio Olympus / Pentax / Fujinon	925,00 a 965,00
Pedal de Chão com Fio	Acionador de captura padrão de chão	965,00
Placa de Captura Analógica (Zscan EVO)	Interface USB 2.0 com saídas S-Video, RCA, BNC	1.050,00

Botão de Captura Sem Fio	Adaptador RF para endoscópios específicos	1.449,00
Pedal de Chão Sem Fio	Acionador RF alimentado por baterias comerciais	1.449,00
Placa de Captura USB HD	Conversor para alta resolução (conexões HDMI/DVI)	6.690,00
Placa de Captura USB SDI	Padrão Broadcast SDI para vídeo médico	6.690,00

Nota Analítica: A estratégia comercial dos distribuidores da Zscan embute uma tática de ancoragem de preço, oferecendo descontos marginais de 8% para pagamentos integrais à vista ou facilidades ilusórias de parcelamento em 3 vezes sem juros, o que psicologicamente atenua o impacto do custo, mas não altera a gravidade do desembolso.

A tabela acima evidencia o custo avassalador que uma clínica deve suportar apenas para montar a infraestrutura física secundária (sem o computador e sem o aparelho de endoscopia propriamente dito). Se um administrador hospitalar for montar uma solução Zscan básica com fio para uma sala de exames (comprando a Placa Analógica de R$ 1.050,00 e o Pedal com Fio de R$ 965,00), ele já compromete R$ 2.015,00.

Se a demanda clínica exigir a modernização para a ergonomia sem fio, buscando evitar o risco biológico e físico de cabos no chão do centro cirúrgico, o custo dos componentes básicos (Placa Analógica e Pedal Sem Fio) salta para R$ 2.499,00. Se o hospital necessitar da placa HDMI de alta definição somada ao pedal sem fio, o custo de hardware puro explode para espantosos R$ 8.139,00.

Ressalta-se um ponto crucial na análise financeira: nenhum desses valores inclui o licenciamento do software proprietário da Zscan, o qual é obrigatório para que as imagens passem pela placa e se transformem em laudos médicos.

Quando justapomos o custo do pedal da Zscan (R$ 965,00 a R$ 1.449,00) ao custo estrutural do pedal mecânico orçado na MultiElétrica para o projeto (R$ 70,00), percebe-se um sobrepreço imposto pelo mercado de mais de 1.300%. O projeto do TCC prova, por meio da engenharia de componentes diretos, que a interface física pode ser resolvida por uma fração ínfima do valor ditado pela Zscan. O mesmo ocorre na comunicação sem fio, que será detalhada mais adiante, onde o chip ESP32-S3 de

R$ 70,00 dizima a necessidade da tecnologia proprietária do botão sem fio da Zscan de R$ 1.449,00.

Diante desses dados empíricos, a viabilidade de comercializar o sistema completo (Hardware, Software e Instalação) desenvolvido no escopo deste trabalho por R$ 3.500,00 não é apenas uma estimativa prudente; é uma declaração de disrupção de mercado. O projeto oferece a uma clínica a previsibilidade orçamentária absoluta. O administrador sabe que, por R$ 3.500,00, a solução está implementada e funcional, erradicando o medo de custos surpresa ocultos em aditivos de software ou quebras de cabos superfaturados.

7.1.2. Quadro Comparativo de Mercado

Para sintetizar a discrepância tarifária e evidenciar a viabilidade comercial perante a banca avaliadora, a tabela a seguir consolida os custos de comercialização do projeto proposto frente aos parâmetros cobrados pelos líderes de mercado. A comparação evidencia que o ecossistema proposto substitui a fragmentação de custos por um valor de venda unificado.

Critério de Comparação	Projeto Proposto	Zscan Software	OCRAM Sistemas
Modelo de Comercialização	Solução Integrada (Hardware e Software)	Modular (Venda separada de licenças e periféricos)	Predominantemente Software de Laudos
Custo de Hardware (Interface de Captura e Pedal)	R$ 140,00 (ESP32-S3 e Pedal)	R$ 2.015,00 (Analógico) a R$ 8.139,00 (HD sem fio)	Não aplicável (Hardware de terceiros)
Licenciamento de Software / Sistema Integrado	R$ 2.500,00 (Desenvolvimento e Implantação)	R$ 9.990,00 (Exemplo: ZSCAN EVO HD)	R$ 1.380,00 a R$ 3.470,00 (Apenas Licença)
Valor Final para o Cliente (VGV)	R$ 3.500,00	A partir de R$ 9.990,00	Variável (Soma-se o custo de hardware extra)

A análise objetiva deste quadro comparativo atesta o potencial de penetração do projeto. Enquanto a aquisição de um sistema HD completo na concorrência, como o Zscan EVO HD, atinge a cifra de R$ 9.990,00, exigindo ainda compras paralelas de periféricos custosos, e a OCRAM cobra entre R$ 1.380,00 e R$ 3.470,00 puramente pela cessão de licenças de laudo, o projeto acadêmico empacota a infraestrutura completa de comunicação (hardware) e controle (software) de forma orgânica. Assim, a estipulação do valor de venda em R$ 3.500,00 consolida uma oportunidade rara de mercado: um custo de implantação radicalmente inferior ao da liderança e uma rentabilidade (ROI) justificada e otimizada pela aplicação inteligente de novos microcontroladores acessíveis.

8. CONSIDERAÇÕES FINAIS

A aplicação final, minuciosa e estrutural do projeto tecnológico Endoscotech no contexto clínico-hospitalar veterinário valida de maneira concreta o quão premente e indispensável é a presença tecnológica integrada e a automatização em fluxos laboratoriais e cirúrgicos modernos. Partindo da análise crítica de ferramentas e softwares obsoletos, com interfaces defasadas que comprovadamente retardavam o fluxo de análise vital e multiplicavam de forma exponencial as chances de extravio ou corrupção de dados dos pacientes (conforme levantamento metodológico e exaustiva identificação dos problemas inerentes aos sistemas legados nas clínicas), a concepção engenhosa do novo ecossistema não apenas organiza e padroniza a arquitetura dos laudos impressos, mas também otimiza integral e definitivamente a sobrecarga de trabalho do corpo clínico hospitalar.

A formidável integração técnica e interdisciplinar demonstrada por este trabalho técnico aliando, de um lado, os mais puros fundamentos da eletroeletrônica e programação em microcontroladores de 32-bits (criação de firmware sólido e com proteção anti-ruído para o chip ESP32 atuando sobre barramento USB) com o objetivo de suprimir e contornar graves limitações mecânicas da interface humana, o sistema de proteção para o médico, paciente e para o sistema desempenhado pelo sistema de proteção e distribuição de circuito (onde o IDR atua protegendo contra choques elétricos e o DPS protegendo o sistema contra surtos na rede) e somada, de outro lado, à excelência em engenharia de software baseada em metodologias ágeis e

bibliotecas gráficas pesadas (emprego maciço de Python associado ao poderoso PySide6 para construção do frontend reativo, visão computacional com OpenCV e manipulação assíncrona de bancos de dados via SQLite) reflete asseveramente o nível e a maturidade dos conhecimentos técnicos absolutos adquiridos nas fileiras e oficinas dos cursos mantidos pelo Centro Paula Souza. Ao automatizar severamente tarefas redundantes de arquivamento textual, edição manual, exportação médica (via PACS DICOM) e captura simultânea paralela à visualização do exame ao vivo de endoscopia, a máquina cede espaço e permite que o talento do médico veterinário se dedique com foco estrito, integral e exclusivo ao conforto clínico, à terapêutica de risco e ao diagnóstico precoce do paciente animal em caráter emergencial ou de rotina. A reestruturação de processos e a modernização profunda proposta através deste exímio Trabalho de Conclusão asseveram e consolidam, por consequência, um salto avassalador rumo ao futuro pragmático e comercialmente sustentável em toda a intrincada cadeia de prestação e asseguração dos serviços de excelência em saúde animal e automação preditiva global.

9. REFERÊNCIAS

- ABNT – Associação Brasileira de Normas Técnicas. NBR 14724: Informação e documentação – Trabalhos acadêmicos.
- ANVISA. *Resolução RDC nº 15, de 15 de março de 2012*. Dispõe sobre os requisitos de boas práticas para o processamento de produtos para a saúde.
- ANVISA. *Resolução RDC nº 6, de 1º de março de 2013*. Dispõe sobre os requisitos de Boas Práticas de Funcionamento para os serviços de endoscopia.
- BOZZINI, P. *O desenvolvimento do Lichtleiter e as origens da endoscopia*.
- BRASIL. Lei nº 13.709/2018 – Lei Geral de Proteção de Dados (LGPD).
- CASE WESTERN RESERVE UNIVERSITY. Hirschowitz fiberoptic Endoscope, 1960.
- DIGITAL IMAGING AND COMMUNICATIONS IN MEDICINE (DICOM). *Standard for handling, storing, printing, and transmitting information in medical imaging*.
- ESPRESSIF SYSTEMS. *ESP32 Technical Reference Manual and Architecture*.[2]
- EUROPEAN MUSEUM OF UROLOGY. Bozzini and the Lichtleiter.

- OLIVEIRA, Marcos. Sistemas de informação na área da saúde. Rio de Janeiro: Editora Ciência Moderna, 2019.

- REVISTA DE MEDICINA VETERINÁRIA. *A Evolução dos Endoscópios Veterinários: Uma Perspectiva Histórica*.

- SILVA, João. Tecnologia aplicada à medicina veterinária. São Paulo: Editora Saúde, 2020.

- SOCIEDADE BRASILEIRA DE ENDOSCOPIA DIGESTIVA (SOBED). *História e Evolução da Endoscopia*. Tratado Ilustrado de Endoscopia Digestiva.

- BARREIRO, Adelson Diogo; CARVALHO, Carolina do Nascimento; SANTOS, Ingrid Machado dos; SOUZA, Julyana Chaday Alves de. Do diploma ao negócio: a importância do empreendedorismo na gestão de clínicas veterinárias. REPOSITÓRIO UNIVERSITÁRIO DA ÂNIMA (RUNA), 12, 2024. Disponível em: https://repositorio.animaeducacao.com.br/items/db5dd030-b243-4e30-96b1-ed7750f32f55
- OCRAM SISTEMAS INFORMATIZADOS. Disponível em: https://ocramsistemas.com.br/
- OCRAM SISTEMAS INFORMATIZADOS - Captura de Imagens Médicas e Laudos . Disponível em: https://www.ocramsistemas.com.br/ocram/produtos/captura/index.php

- OCRAM SISTEMAS INFORMATIZADOS - Captura de Imagens Médicas e Laudos (Medicina Veterinária). Disponível em: http://www.ocramsistemas.com.br/ocram/produtos/captura_vet/index.php

- OCRAMCLIMA® - Mestria de Ar Puro. Disponível em: https://ocram-clima.com/

- OCRAMCLIMA® - Loja. Disponível em: https://ocram-clima.com/loja/

- PREFEITURA MUNICIPAL DE CAMETÁ – PA, Comissão permanente de licitação. Disponível em: https://prefeituradecameta.pa.gov.br/wp-content/uploads/2022/04/CONTRATO-ADMINISTRATIVO-No-6.PE_.019-2022-PMCSMS.pdf

- ZSCAN SOFTWARE IMPORTACAO E EXPORTACAO LTDA – Licinexus. Contratos públicos. Disponível em: https://www.licinexus.com.br/licitacoes/fornecedor/zscan-software-importacao-e-exportacao-ltda/

- PLACA DE CAPTURA ANALÓGICA ZSCAN USB em 3x no cartão ... Disponível em: https://www.massafmedical.com.br/loja/produto/placa-de-captura-zscan-usb/

- Pedais de captura, acessórios exclusivos #zscansoftware – YouTube. Disponível em: https://www.youtube.com/watch?v=tJv3b_gx5a8