UNIVERSIDADE FEDERAL DO MARANHÃO - UFMA
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA - CCET
ENGENHARIA DA COMPUTAÇÃO

Daniel Campos Galdez Monteiro

Protótipo IoT com Computação de Borda e Reconhecimento Facial para Registro Automatizado de Frequência Acadêmica

São Luís
2026

Daniel Campos Galdez Monteiro

Protótipo IoT com Computação de Borda e Reconhecimento Facial para Registro Automatizado de Frequência Acadêmica

Trabalho de Conclusão de Curso II apresentado ao Curso de Engenharia da Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação. Centro de Ciência Exatas e Tecnológicas da Universidade Federal do Maranhão.

Orientador: Prof. Dr. Luiz Henrique Neves Rodrigues

São Luís

2026

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

Galdez Monteiro, Daniel Campos.
Protótipo iot com computação de borda e reconhecimento facial para registro automatizado de frequência acadêmica / Daniel Campos Galdez Monteiro. - 2026.
85 p.

Orientador(a): Luiz Henrique Neves Rodrigues.
Curso de Engenharia da Computação, Universidade Federal do Maranhão, São Luís, Maranhão, 2026.

1. Frequência. 2. Biometria. 3. Borda. 4. Iot. 5. Protótipo. I. Rodrigues, Luiz Henrique Neves. II. Título.

Daniel Campos Galdez Monteiro

Protótipo IoT com Computação de Borda e Reconhecimento Facial para Registro Automatizado de Frequência Acadêmica

Trabalho de Conclusão de Curso II apresentado ao Curso de Engenharia da Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação. Centro de Ciência Exatas e Tecnológicas da Universidade Federal do Maranhão.

Trabalho **aprovado** em São Luís, 9 de julho de 2026:

Prof. Dr. Luiz Henrique Neves Rodrigues
Orientador

Prof. Me. Marcio Mendes Cerqueira
Examinador

Prof. Dr. Sergio Souza Costa
Examinador

São Luís
2026

Resumo

O registro de frequência acadêmica é uma atividade recorrente em aulas presenciais e pode demandar tempo, organização posterior dos dados e conferência manual por parte dos docentes. Diante desse contexto, este trabalho teve como objetivo desenvolver e verificar tecnicamente, em ambiente controlado, um protótipo distribuído para apoio ao registro automatizado de frequência acadêmica por reconhecimento facial. A solução, denominada AutoPonto, foi composta por um dispositivo embarcado baseado em ESP32-CAM, um nó de computação de borda para processamento local das capturas, um servidor central para regras acadêmicas e persistência dos dados, uma interface *web* para consulta e administração, além de integração com a plataforma InterSCity para telemetria. A metodologia adotada caracterizou-se como pesquisa aplicada, de desenvolvimento tecnológico e objetivo exploratório, com verificação funcional e coleta de métricas técnicas em simulação controlada. Os resultados indicaram que o protótipo executou o fluxo previsto, desde o cadastro acadêmico e biométrico até a captura, reconhecimento, registro de presença, retorno ao dispositivo e consulta posterior na aplicação *web*. Também foram observadas medidas de proteção dos dados biométricos, como criptografia e revogação imediata pelos seus titulares. Conclui-se que o AutoPonto demonstrou viabilidade técnica inicial no cenário testado, embora ainda sejam necessárias validações em ambiente real, com maior número de participantes, avaliação de uso contínuo e análise ampliada dos aspectos éticos, operacionais e de segurança.

Palavras-chave: Frequência; Biometria; Borda; IoT; Protótipo.

Abstract

Academic attendance recording is a recurring activity in face-to-face classes and may require time, later data organization, and manual checking by teachers. In this context, this work aimed to develop and technically verify, in a controlled environment, a distributed prototype to support automated academic attendance recording through facial recognition. The solution, named AutoPonto, was composed of an embedded device based on ESP32-CAM, an edge computing node for local image processing, a central server for academic rules and data persistence, a web interface for consultation and administration, and an integration with the InterSCity platform for telemetry. The adopted methodology was characterized as applied research, with technological development and an exploratory nature, based on functional verification and technical metric collection in a controlled simulation. The results indicated that the prototype executed the expected flow, from academic and biometric registration to image capture, recognition, attendance recording, device feedback, and later consultation through the web application. Measures to protect biometric data were also adopted, such as encryption and immediate revocation by the data subjects. It is concluded that AutoPonto demonstrated initial technical feasibility in the tested scenario, although further validation in real environments is still required, with a larger number of participants, continuous-use evaluation, and broader analysis of ethical, operational, and security aspects.

Keywords: Attendance; Biometrics; Edge; IoT; Prototype.

Lista de Figuras

Lista de quadros

Lista de Tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
CC	*Ciência da Computação*
DRF	*Django REST Framework*
FreeRTOS	*Free Real-Time Operating System*
GND	*Ground*
GPIO	*General Purpose Input/Output*
GPU	*Graphics Processing Unit*
HTTP	*Hypertext Transfer Protocol*
IoT	*Internet of Things*
IPS	*In-Plane Switching*
ISO	*International Organization for Standardization*
JPEG	*Joint Photographic Experts Group*
JWT	*JSON Web Token*
LED	*Light-Emitting Diode*
LGPD	*Lei Geral de Proteção de Dados Pessoais*
MQTT	*Message Queuing Telemetry Transport*
OpenCV	*Open Source Computer Vision Library*
ORM	*Object-Relational Mapping*
PCB	*Printed Circuit Board*
PIR	*Passive Infrared*
PLA	*Polylactic Acid*
PSRAM	*Pseudo Static Random Access Memory*
QR	*Quick Response*

RAM	*Random Access Memory*
REST	*Representational State Transfer*
RFID	*Radio Frequency Identification*
RSSI	*Received Signal Strength Indicator*
SCL	*Serial Clock*
SDA	*Serial Data*
SPI	*Serial Peripheral Interface*
SSD	*Solid-State Drive*
TFT	*Thin-Film Transistor*
UFMA	*Universidade Federal do Maranhão*
URL	*Uniform Resource Locator*
USB	*Universal Serial Bus*
VCC	*Tensão de alimentação positiva*

Sumário

1 Introdução

O registro de frequência é uma atividade recorrente em aulas presenciais e ainda é realizado, em muitos contextos, por chamada oral, assinatura em lista ou conferência manual posterior. Por mais que os procedimentos sejam simples, eles são capazes de gerar fraudes, erros de registro, retrabalho e consumo excessivo de tempo de aula (HAKIM; FITRIANI; MA'WE, 2025, p. 183).

Com a ampliação de tecnologias computacionais no ambiente educacional, diferentes soluções têm sido testadas para possibilitar a automatização da frequência acadêmica. Entre elas, destacam-se abordagens com identificação por radiofrequência (*Radio Frequency Identification*, RFID), biometria, visão computacional, dispositivos embarcados e Internet das Coisas, as quais buscam reduzir a intervenção manual e tornar os registros mais acessíveis em uma interface intuitiva (WYNE et al., 2015). No caso do reconhecimento facial, a identificação pode ocorrer sem contato físico direto com sensores, mas envolve desafios relacionados à qualidade das imagens, ao desempenho do reconhecimento e à proteção de dados biométricos (PIRES et al., 2025).

Apesar dessas possibilidades, parte das soluções existentes concentra-se em etapas isoladas do processo, como a captura da imagem ou a comparação facial, sem tratar de forma conjunta o dispositivo de captura, o local de processamento, a sincronização com um sistema central com as regras de negócio e o monitoramento técnico da operação. Nesse contexto, a computação de borda torna-se relevante por permitir que parte do processamento ocorra próximo à origem dos dados, reduzindo a dependência de comunicação contínua com o servidor central (KHAN et al., 2019).

Justifica-se, portanto, a realização deste trabalho pela necessidade de avaliar, de forma integrada, etapas que geralmente aparecem separadas em soluções de frequência automatizada: captura embarcada, processamento em borda, sincronização com um sistema central e acompanhamento por interface *web*. Para isso, adota-se como objeto de estudo o AutoPonto, um protótipo distribuído para chamada acadêmica automatizada composto por dispositivo ESP32-CAM, nó de computação de borda, servidor central, interface *web* e integração com a plataforma InterSCity para telemetria. O sistema não é tratado como solução institucional pronta, mas como protótipo técnico a ser analisado em ambiente de simulação controlada.

Diante desse cenário, a questão central deste trabalho é: em que medida o protótipo AutoPonto permite avaliar a viabilidade e o funcionamento do registro automatizado de frequência acadêmica por reconhecimento facial em uma arquitetura distribuída?

1.1 Objetivo geral

O objetivo geral deste trabalho é analisar a viabilidade e o funcionamento de uma arquitetura distribuída de computação de borda para registro automatizado de frequência acadêmica por reconhecimento facial, por meio do protótipo AutoPonto em um ambiente de simulação controlada.

1.2 Objetivos específicos

- Projetar e documentar o dispositivo embarcado de chamada, incluindo circuito eletrônico, placa de circuito impresso (*Printed Circuit Board*, PCB), caixa de montagem, periféricos de interação e alimentação.
- Desenvolver o *firmware* do dispositivo, contemplando tarefas *Free Real-Time Operating System* (FreeRTOS), máquina de estados, comunicação *Hypertext Transfer Protocol* (HTTP) e *Message Queuing Telemetry Transport* (MQTT), métricas técnicas e modos de economia de energia.
- Implementar o nó de computação de borda, incluindo comunicação com os dispositivos, filas de processamento, *cache* local, reconhecimento facial e geração de eventos de presença.
- Desenvolver o servidor central AutoPonto e a interface *web*, abrangendo modelo acadêmico, cadastro biométrico, sincronização com a borda, relatórios, *dashboards* e integração com a InterSCity.
- Verificar o funcionamento do protótipo em simulação controlada, analisando os fluxos de cadastro, sincronização, captura, reconhecimento, retorno ao dispositivo, consulta de presenças e métricas técnicas coletadas.

2 Fundamentação Teórica

2.1 Automatização do registro de frequência acadêmica

O registro de frequência acadêmica é uma prática administrativa que permite identificar a presença do discente em determinada aula, apoiar o controle da turma e produzir dados para consulta institucional. Na literatura, esse registro também é tratado como um indicador relacionado ao desempenho, ao engajamento e ao acompanhamento da trajetória dos estudantes (PIRES et al., 2025, p. 373).

Nos métodos manuais, como chamada oral e listas de presença, o registro depende da intervenção direta do docente e de posterior organização dos dados. Estudos apontam que métodos manuais podem consumir tempo de aula, estar sujeitos a erros de registro, e dificultar o monitoramento em tempo real (SARI; HARAPAN; INDRAWATI, 2025, p. 844). Neste trabalho, tais limitações são tratadas como fragilidades gerais do procedimento manual, sem assumir percentuais de erro ou economia de tempo que não tenham sido medidos no ambiente real de aplicação do protótipo.

Nesse sentido, a literatura recente apresenta diferentes formas de automatizar o registro de frequência, dependendo do mecanismo de identificação, da infraestrutura necessária e dos cuidados associados ao tratamento dos dados. No Quadro 1, reúnem-se essas abordagens em síntese comparativa.

Quadro 1 – Comparação entre formas de automação do registro de frequência acadêmica

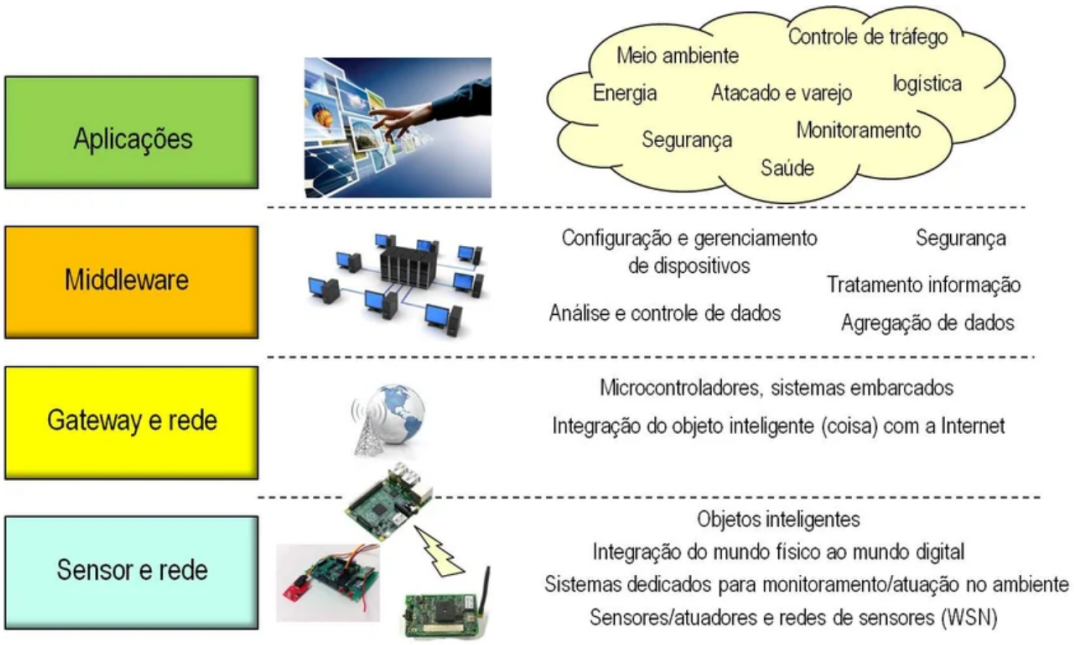
Autor	**Solução**	**Contribuição**	**Limitações**
Sari, Harapan e Indrawati (2025)	Código de barras	Automatiza o registro da presença e apoia o armazenamento e a geração de relatórios.	Depende de dispositivos de leitura e foi avaliada em contexto escolar específico.
Hakim, Fitriani e Ma'we (2025)	código de resposta rápida (*Quick Response Code*, *QR Code*)	Reduz preenchimentos manuais, armazena dados em banco e permite consultas, relatórios e notificações.	Exige controle do código, estabilidade da aplicação e compatibilidade entre dispositivos.
Adesoba e Joseph (2025)	Impressão digital	Autentica o estudante por biometria, reduzindo erros, fraudes e presença por terceiros.	Requer sensor, digitais disponíveis e cuidado com dados biométricos sensíveis.
Adesoba e Joseph (2025)	RFID	Identifica o estudante por etiqueta ou cartão associado a um identificador único.	Pode permitir marcação por terceiros caso o cartão ou etiqueta seja usado por outra pessoa.
Pires et al. (2025)	Reconhecimento facial	Registra presença por imagens da sala, integrando visão computacional, banco de dados e relatórios.	Depende de qualidade da imagem, consentimento, controle de acesso e adequação à Lei Geral de Proteção de Dados Pessoais (LGPD).

Fonte: Compilado pelo Autor (2026).

2.2 Internet das Coisas

A Internet das Coisas, ou *Internet of Things* (IoT), pode ser entendida como a interconexão de dispositivos físicos capazes de coletar e compartilhar informações por redes com fio ou sem fio. Essa estrutura combina elementos de *hardware*, como sensores, computadores e módulos de comunicação, com elementos de *software*, responsáveis pelo tratamento e disponibilização dos dados coletados (CHOUDHARY et al., 2025, p. 2). Na Figura 1, evidenciam-se as camadas simplificadas desse tipo de solução.

Figura 1 – Camadas funcionais de uma solução IoT



Fonte: Carissimi (2016).

Essa organização permite compreender soluções IoT como sistemas formados por camadas complementares, nas quais o dispositivo físico coleta dados do ambiente e os encaminha para etapas posteriores de processamento, armazenamento e consulta. Em sistemas de frequência automatizada, essa lógica ajuda a distinguir a camada de captura dos serviços responsáveis por validação, registro e apresentação das informações.

2.3 Sistemas embarcados e *hardware*

Em soluções IoT, sistemas embarcados são responsáveis pela interação direta com o ambiente físico. Esses dispositivos executam tarefas específicas, como leitura de sensores, acionamento de periféricos, coleta de dados e comunicação com outros dispositivos ou servidores (CHOUDHARY et al., 2025, p. 10). Em aplicações com câmera e detector de presença, Bagaskara e Susanto (2025, p. 14) apresentam uma organização em que o sensor infravermelho passivo (*Passive Infrared*, PIR) atua na detecção de movimento, enquanto a ESP32-CAM realiza a captura de imagens e a integração com uma aplicação remota. Esse tipo de composição ilustra o papel do sistema embarcado como ponto de contato entre o ambiente físico e as demais camadas computacionais.

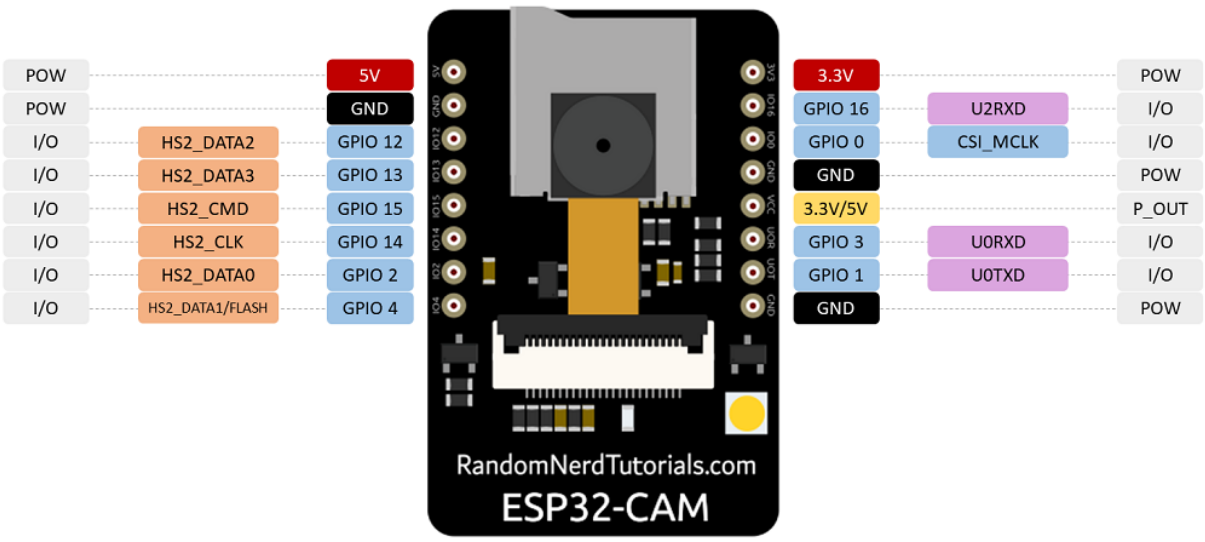
2.3.1 ESP32-CAM

A ESP32-CAM é uma placa de desenvolvimento compacta baseada no chip ESP32, voltada a aplicações embarcadas que exigem captura de imagem e comunicação sem fio.

No modelo AI-Thinker, o módulo integra câmera OV2640, conector para cartão microSD e pinos de entrada e saída de uso geral, do inglês *General Purpose Input/Output* (GPIO), para alimentação, comunicação serial e conexão com periféricos (SANTOS, 2020, p. 1). Por reunir câmera, conectividade Wi-Fi e entradas/saídas digitais em uma mesma placa, esse tipo de módulo é recorrente em protótipos IoT que precisam capturar dados visuais e encaminhá-los para outra camada do sistema.

Na Figura 2, ilustra-se o diagrama de pinos da ESP32-CAM AI-Thinker, indicando conexões de alimentação, GPIOs, interface serial, câmera e cartão microSD.

Figura 2 – Diagrama de pinos da ESP32-CAM AI-Thinker



Fonte: Santos (2020, p. 1).

A organização dos pinos mostra que a ESP32-CAM não deve ser tratada apenas como uma câmera, mas como uma unidade embarcada com restrições próprias. Santos (2020) destaca que a placa possui pinos específicos de alimentação, GPIOs compartilhados com o cartão microSD e pinos seriais usados para comunicação e gravação de código, pois o módulo não possui programador *Universal Serial Bus* (USB) integrado. Essas características afetam a forma como sensores e periféricos podem ser conectados ao módulo.

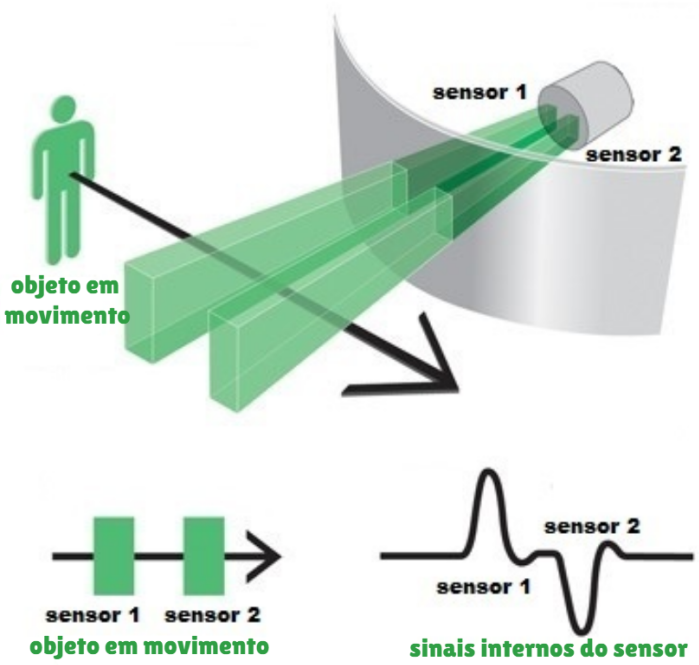
Em arquiteturas com captura de imagem, a ESP32-CAM tende a atuar como unidade de aquisição e comunicação, enquanto tarefas computacionalmente mais custosas podem ser encaminhadas a outros nós ou servidores. Kok et al. (2025, p. 20–21) descrevem uso semelhante em um sistema IoT no qual a ESP32-CAM captura imagens, recebe sinais de sensores e transmite os dados por Wi-Fi para processamento externo. Essa separação ajuda a compatibilizar captura local, limitações de processamento e integração com serviços de validação ou análise.

2.3.2 Sensor PIR de presença

O sensor *Passive Infrared* (PIR), ou infravermelho passivo, é um sensor usado para detectar movimento a partir de variações de radiação infravermelha no ambiente. Ghosh (2024) descreve esse tipo de sensor como baseado no efeito piroelétrico, no qual mudanças de temperatura provocam variações mensuráveis de tensão. Diferentemente de sensores ativos, o PIR não emite ondas para analisar o ambiente, mas reage a alterações térmicas, como a passagem de uma pessoa em seu campo de detecção.

Na Figura 3, explicita-se o princípio de detecção de movimento em um sensor PIR. Nesse tipo de sensor, a lente de Fresnel amplia o campo de detecção ao dividi-lo em regiões. Quando um corpo mais quente que o fundo atravessa essas regiões, os elementos internos percebem variações na radiação infravermelha e geram pulsos elétricos correspondentes (GHOSH, 2024, p. 4).

Figura 3 – Princípio de detecção de movimento em sensor PIR



Fonte: Traduzido de Ghosh (2024).

Em sistemas embarcados com câmera, o PIR pode atuar como elemento de acionamento da captura. Bagaskara e Susanto (2025, p. 14) apresentam uma aplicação em que o sensor PIR detecta movimento na área monitorada, enquanto a ESP32-CAM registra imagens associadas ao evento. Essa distinção é relevante para sistemas de frequência com reconhecimento facial porque o sinal produzido pelo PIR indica uma alteração térmica no campo de detecção, sem fornecer informação sobre a identidade do objeto ou pessoa detectada.

2.3.3 *Display* de transistor de filme fino (TFT) e interface periférica serial (SPI)

O *display* utilizado em sistemas embarcados atua como interface em campo de retorno de mensagens ao usuário. O modelo GC9A01 é um *display* circular de 1,28 polegada, com painel de comutação no plano (*In-Plane Switching*, IPS) e resolução de 240×240 pixels, características que o tornam adequado para interfaces compactas em dispositivos embarcados (RAWAT, 2026).

Na Figura 4, apresenta-se o módulo GC9A01, destacando sua aplicação em interfaces gráficas compactas.

Figura 4 – *Display* circular GC9A01

Fonte: Rawat (2026).

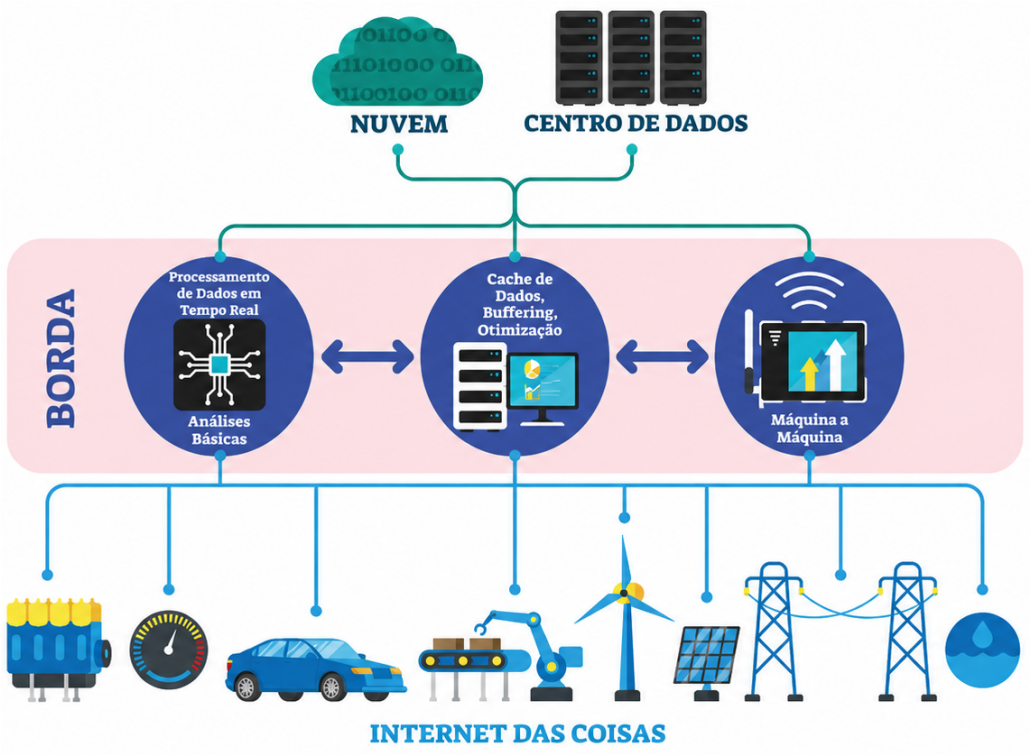
A comunicação entre o microcontrolador e esse tipo de *display* é normalmente realizada por SPI, uma interface serial usada para transferir dados entre um controlador e periféricos. No caso do GC9A01, Rawat (2026, p. 4) descreve o uso das linhas *Serial Clock* (SCL) e *Serial Data* (SDA) para a comunicação SPI, além de pinos de controle para comando/dado, seleção do dispositivo e reinicialização. A integração desse módulo, portanto, ocupa cinco GPIOs de controle e comunicação, além dos pinos de alimentação, aspecto relevante em placas com quantidade limitada de conexões disponíveis, como a ESP32-CAM.

2.4 Computação de borda

A computação de borda, ou *edge computing*, desloca parte do processamento e do armazenamento para dispositivos ou nós próximos da origem dos dados, reduzindo

a dependência de uma nuvem ou servidor central. Khan et al. (2019) caracterizam essa abordagem como uma extensão dos modelos centralizados, aproximando recursos computacionais dos usuários. Na Figura 5, representa-se essa organização ao posicionar a borda como camada intermediária entre dispositivos IoT e servidores.

Figura 5 – Computação de borda em uma arquitetura IoT



Fonte: Traduzido de Mouser Electronics (2024).

A adoção da borda não elimina a necessidade de servidores centrais. Ela apenas redistribui parte das tarefas, permitindo que algumas operações ocorram perto do local onde o dado é produzido. Essa divisão pode reduzir latência, tráfego de rede e circulação de dados brutos, mas também introduz limites relacionados a processamento, memória, armazenamento, energia e manutenção dos nós locais (PIOVESAN et al., 2025). Por isso, a borda deve ser entendida como uma estratégia de arquitetura, e não como substituição direta da nuvem.

Kerner (2026) observa que a borda tende a ser mais adequada quando a aplicação depende de baixa latência, disponibilidade local, conectividade limitada ou restrição no envio de dados sensíveis, enquanto regras de negócio continuam mais compatíveis com infraestrutura em nuvem. Em sistemas de frequência baseados em imagem, essa discussão ajuda a diferenciar a captura local, o processamento próximo ao ambiente de uso e a persistência central dos registros.

2.5 Comunicação entre componentes

Em uma arquitetura distribuída, a comunicação entre componentes pode ocorrer por diferentes protocolos. O HTTP, sigla de *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto), segue o modelo de requisição e resposta, no qual um cliente envia uma solicitação a um servidor e espera uma resposta indicando o resultado da operação (KUROSE; ROSS, 2021, p. 72). Essa forma de comunicação é adequada quando há uma operação bem delimitada entre cliente e servidor.

O MQTT, sigla de *Message Queuing Telemetry Transport* (Transporte de Telemetria por Enfileiramento de Mensagens), é um protocolo leve de publicação e assinatura, projetado para dispositivos com restrições de processamento, memória, energia ou largura de banda (YUAN, 2017). Nesse modelo, os clientes publicam mensagens em tópicos ou assinam tópicos de interesse, enquanto um *broker* recebe as publicações e as encaminha aos assinantes correspondentes.

O *broker* é, portanto, o componente que intermedeia a comunicação no modelo de publicação e assinatura, recebendo mensagens dos publicadores e encaminhando-as aos assinantes dos tópicos correspondentes. Paul et al. (2026, p. 2–3) descrevem o Mosquitto como um *broker* MQTT leve, implementado em C e voltado a ambientes com restrição de recursos, embora sua arquitetura de execução em uma única *thread* possa limitar a escalabilidade sob cargas elevadas. Essa característica é compatível com sistemas de borda de menor porte, nos quais o *broker* atua como intermediário local para estados, eventos e comandos entre poucas dezenas de dispositivos e aplicações.

Em uma rede física de dispositivos, o modem e o ponto de acesso cumprem papéis distintos. O modem realiza a ligação entre a rede interna e a rede de acesso do provedor, permitindo a comunicação com a Internet, enquanto o ponto de acesso fornece conectividade sem fio aos dispositivos da rede local (KUROSE; ROSS, 2021, p. 391). Em sistemas IoT com computação de borda, essa distinção ajuda a compreender a rede local como parte do ambiente de execução, pois a troca de dados entre dispositivo embarcado, nó de borda e demais camadas da aplicação depende dessa infraestrutura de comunicação.

2.6 Reconhecimento facial

2.6.1 Detecção facial com Yunet

A detecção facial é a etapa responsável por localizar regiões de face em uma imagem. Essa etapa geralmente antecede o reconhecimento facial, pois define qual região da imagem será encaminhada para alinhamento, extração de características e comparação. Ante o exposto, Wu, Peng e Yu (2023) descrevem o YuNet como um detector facial a nível de milissegundos, projetado especificamente para dispositivos de borda. Segundo os autores, o

modelo busca equilibrar acurácia e eficiência computacional por meio de uma arquitetura compacta, composta por etapas simplificadas de extração e fusão de características.

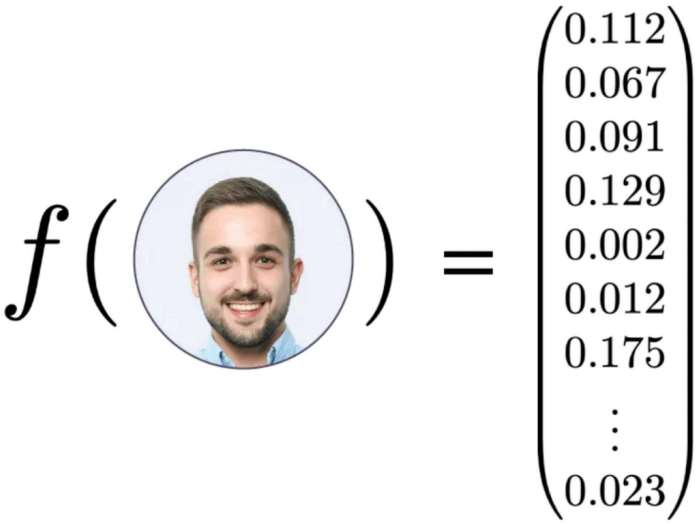
A escolha de detectores leves é relevante em cenários que dispositivos de Internet das Coisas operam com limitações de memória, armazenamento e capacidade computacional. Wu, Peng e Yu (2023) destacam que modelos faciais muito robustos podem atingir alta precisão, mas tendem a exigir maior custo computacional, o que dificulta sua implantação em ambientes de borda. Dessa forma, o YuNet se aproxima das necessidades de protótipos que precisam detectar faces em tempo real antes da etapa de reconhecimento.

2.6.2 Reconhecimento facial com SFace e *embeddings*

Reconhecimento facial é um processo computacional voltado à identificação ou verificação de faces em imagens. Dulcic (2021) apresenta esse processo como uma sequência formada por detecção da face, extração de características e classificação. Dessa forma, o reconhecimento não se baseia na comparação direta entre imagens brutas, mas na extração de informações da região facial para produzir uma representação adequada à comparação computacional.

Dulcic (2021) complementa que essa representação numérica é chamada de *embedding*. Em termos gerais, um *embedding* facial é um vetor que concentra características relevantes da face. Na Figura 6, ilustra-se essa conversão da imagem facial para uma representação vetorial.

Figura 6 – Representação de uma face como vetor de *embedding*



Fonte: Dulcic (2021).

Depois de gerado o *embedding*, a comparação ocorre entre o vetor extraído da captura analisada e os vetores já associados a identidades conhecidas. Dulcic (2021)

descreve essa etapa como o cálculo de distâncias entre representações faciais: quanto menor a distância entre dois vetores, maior a semelhança entre as faces comparadas. A decisão final depende de um limiar de aceitação, isto é, um valor usado para definir se a similaridade observada é suficiente para aceitar ou rejeitar uma correspondência.

O SFace pode ser situado nesse conjunto de modelos profundos de reconhecimento facial. Boutros et al. (2022, p. 3) descrevem o modelo como uma rede que extrai características faciais por meio de uma camada de *embedding*, produzindo representações adequadas à comparação por similaridade de cosseno.

Essa lógica permite compreender a identificação como um processo dependente da qualidade da captura. Iluminação, pose, distância, resolução e enquadramento podem afetar a representação obtida, enquanto o limiar de decisão influencia a possibilidade de falso aceite ou falsa rejeição. Por isso, o reconhecimento facial deve ser tratado como uma verificação técnica sujeita a limites, e não como identificação infalível.

2.6.3 Dados biométricos e cuidados com privacidade

Em sistemas de reconhecimento facial, imagens da face e representações derivadas, como *embeddings*, exigem atenção especial quando são usadas para identificar ou autenticar uma pessoa. A LGPD define como dado pessoal sensível o dado biométrico vinculado a uma pessoa natural (BRASIL, 2018, art. 5º, II). Assim, ainda que um sistema não armazene a imagem bruta da face, o vetor facial associado a um estudante continua exigindo proteção, pois pode ser utilizado como referência para verificação de identidade.

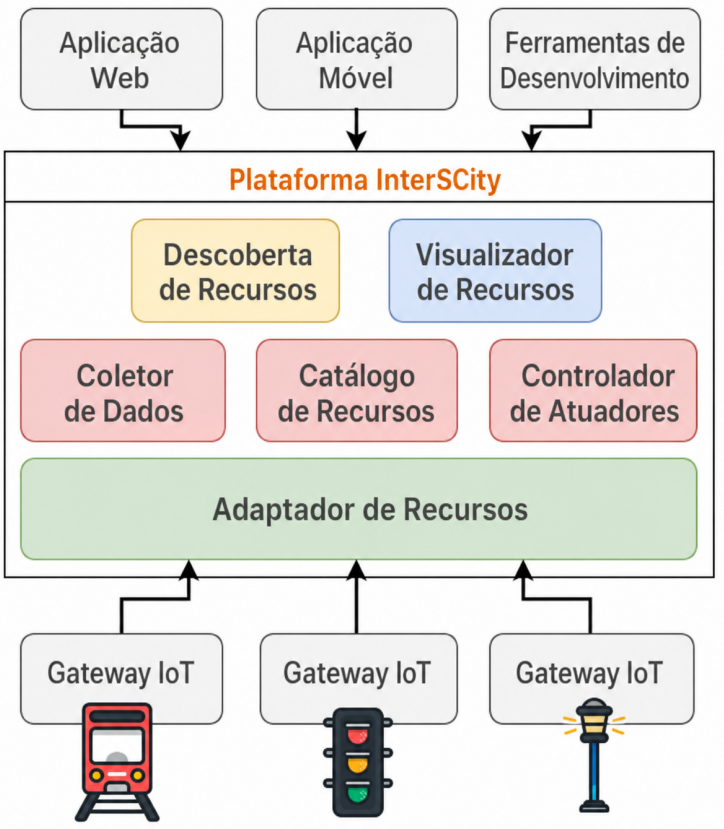
Nessa conjuntura, a criptografia dos vetores faciais atua como uma medida técnica de proteção, embora não elimine a natureza sensível do dado biométrico. O Fernet é um mecanismo de criptografia simétrica disponibilizado pela biblioteca *cryptography* do Python, no qual a mesma chave é usada para criptografar e descriptografar os dados (MARIMUTHU, 2025). Aplicado a *embeddings* faciais, esse recurso impede que os vetores armazenados sejam lidos diretamente sem a chave correta, mas deve ser combinado com controle de acesso e guarda segura da chave.

2.7 Telemetria com InterSCity

Del Esposte et al. (2017, p. 2–3) apresentam a InterSCity como uma plataforma aberta, baseada em microsserviços, desenvolvida para apoiar aplicações de cidades inteligentes. A proposta da plataforma é fornecer serviços de alto nível para o gerenciamento de recursos IoT heterogêneos, armazenamento e processamento de dados, além de descoberta de recursos com base em informações de contexto. Essa organização busca reduzir a dependência de soluções isoladas e favorecer a integração entre dispositivos, serviços e aplicações em ambientes urbanos.

Na Figura 7, explicita-se a arquitetura geral da plataforma InterSCity, organizada em microsserviços que se comunicam com dispositivos IoT e aplicações externas.

Figura 7 – Arquitetura da plataforma InterSCity



Fonte: Traduzido de Del Esposte et al. (2017, p. 5).

Entre os microsserviços da plataforma, o *Resource Adaptor* funciona como uma camada de entrada para integração com os dispositivos. Del Esposte et al. (2017, p. 5) descrevem esse microsserviço como um intermediador usado para registrar e atualizar recursos, enviar dados de sensores e assinar eventos relacionados a comandos de atuação. O *Data Collector*, por sua vez, armazena, filtra e disponibiliza os dados coletados pelos dispositivos, incluindo dados de contexto e eventos observados em um instante específico (ESPOSTE et al., 2017, p. 5).

A relação entre InterSCity e telemetria está no recebimento e acompanhamento de dados operacionais produzidos por dispositivos. Silva e Santos (2024, p. 269–271) tratam a telemetria como uma ferramenta IoT voltada ao monitoramento contínuo de variáveis em tempo real que permite acompanhar condições de operação e apoiar decisões de gestão. Embora o estudo desses autores esteja voltado ao transporte, a ideia de monitoramento contínuo também se aplica a dispositivos IoT em outros domínios.

No caso de um sistema de frequência com dispositivo de borda, a InterSCity pode ser compreendida como uma plataforma auxiliar para telemetria e monitoramento, e

não como a camada responsável pela decisão principal do reconhecimento ou do registro acadêmico. Seu papel conceitual está relacionado ao recebimento e à organização de dados operacionais de dispositivos.

2.8 Aplicações web e persistência de dados

2.8.1 Dados relacionais e não relacionais

Em aplicações web, a persistência de dados depende do tipo de informação manipulada. Silva (2025, p. 14) apresenta bancos de dados como sistemas voltados ao armazenamento, recuperação e atualização de informações. Nesse conjunto, o modelo relacional organiza dados em tabelas e relações, enquanto modelos não relacionais podem priorizar flexibilidade, desempenho em acessos específicos ou estruturas simples de chave e valor.

O PostgreSQL se enquadra no modelo relacional e é descrito por Silva (2025, p. 10) como um sistema associado à confiabilidade, consistência e manipulação de dados estruturados complexos. Essa característica favorece aplicações que precisam manter registros duráveis e relações bem definidas entre entidades. Guarnieri (2025, p. 3) também observa que a escolha entre modelos relacionais e não relacionais deve considerar perfil de consulta, consistência e objetivos de escalabilidade.

O Redis, por sua vez, é um banco chave-valor em memória, voltado ao acesso rápido aos dados. Silva (2025, p. 23) observa que o Redis mantém dados em memória principal e oferece baixa latência em operações de leitura. Por isso, seu uso se aproxima de dados operacionais, temporários ou acessados com frequência, e não de registros que exigem persistência durável e relações complexas.

No contexto deste trabalho, essa distinção justifica a separação entre armazenamento central e armazenamento operacional. Dados acadêmicos ajustam-se melhor a um banco relacional, pois dependem de consistência e relações entre entidades. Já dados usados na borda, como cópias de consulta rápida, podem ser mantidos em *cache* para reduzir consultas constantes ao servidor central. Assim, PostgreSQL e Redis cumprem papéis complementares: o primeiro voltado à persistência principal e o segundo ao apoio operacional de baixa latência na borda, sem substituir a base relacional.

2.8.2 Interface de Programação de Aplicações (API) *Representational State Transfer* (REST) com *Django REST Framework*

Em uma aplicação web, o *backend* concentra a lógica executada no servidor. Shyam e Mukesh (2020, p. 139) apresentam o Django como um *framework* web em Python voltado ao desenvolvimento rápido de aplicações orientadas a banco de dados. Essa camada não

se limita à apresentação visual dos dados, pois sustenta as operações que permitem o funcionamento da aplicação.

A comunicação entre o *backend* e outros componentes pode ser feita por uma API REST. Patil (2025, p. 1) define API como uma interface que permite a troca de dados entre aplicações sem que uma precise conhecer detalhes internos da outra. Em APIs REST, essa comunicação é organizada em torno de recursos acessados por localizadores uniformes de recursos (*Uniform Resource Locators*, URLs) e manipulados por métodos HTTP, como *GET, POST, PUT* e *DELETE* (PATIL, 2025, p. 3–6). Com isso, interfaces *web*, aplicações móveis ou dispositivos externos podem consumir serviços do servidor por um contrato de comunicação padronizado.

O *Django REST Framework* (DRF) amplia o uso do Django para a criação de APIs REST. Patil (2025, p. 9) descreve o DRF como uma ferramenta voltada à construção de APIs escaláveis, seguras e manuteníveis, abrangendo recursos como serialização, autenticação, permissões, *cache* e versionamento. Em um sistema de frequência acadêmica com dispositivo de borda e interface *web*, a API REST atua como ponto de integração entre as várias camadas.

2.8.3 Interface *web* com React

A interface *web*, ou *frontend*, corresponde à camada de interação entre o usuário e os serviços disponibilizados pelo servidor. Essa parte não concentra a regra principal de negócio, mas apresenta dados, formulários e mensagens para o usuário. Le (2019, p. 7) descreve o React como uma biblioteca *JavaScript* declarativa e baseada em componentes, voltada à construção de interfaces interativas e reutilizáveis.

Em aplicações com diferentes perfis de usuário, a interface também organiza o acesso às páginas conforme o papel de cada pessoa no sistema. No trabalho de Le (2019, p. 6), o uso de uma aplicação *web* com React é associado à construção de um painel de gerenciamento com autenticação e interfaces distintas para usuários. Essa ideia é compatível com sistemas acadêmicos, nos quais alunos, professores e administradores podem visualizar informações diferentes.

Os *dashboards* complementam essa camada ao sintetizar dados em uma forma de elementos gráficos consultáveis. Filipe (2023, p. 7) observa que ferramentas de visualização, como *dashboards*, apoiam a tomada de decisão a curto, médio e longo prazo, especialmente quando há grande volume de dados coletados. No contexto deste trabalho, essa função se relaciona à apresentação de relatórios acadêmicos, pois esse recurso permite que os dados registrados pelo sistema sejam acompanhados de modo intuitivo.

3 Metodologia

Este trabalho caracteriza-se como uma pesquisa aplicada, de desenvolvimento tecnológico e objetivo exploratório. É aplicada por partir de um problema prático do contexto acadêmico: apoiar o registro de frequência por meio de um fluxo automatizado e integrado a sistemas digitais. Possui caráter tecnológico porque tem como principal resultado um protótipo funcional composto por dispositivo embarcado, reconhecimento facial em computação de borda, servidor central, interface *web* e integração com a plataforma InterSCity.

O objetivo exploratório justifica-se pelo estágio do sistema, que não foi implantado em turma real nem avaliado em operação institucional. O trabalho concentrou-se na implementação do protótipo e no cumprimento dos seus requisitos em um ambiente controlado.

Quanto à análise, a pesquisa foi baseada em dados técnicos quantitativos e registros descritivos produzidos durante a simulação controlada. A avaliação técnica foi realizada a partir de métricas programadas nos *softwares* dos principais componentes do sistema e da inspeção dos registros gerados durante os fluxos de cadastro, captura, reconhecimento, sincronização e consulta pela interface *web*.

3.1 Visão geral da metodologia

A primeira etapa consistiu na definição dos requisitos do protótipo e na escolha de uma arquitetura distribuída, composta por dispositivo embarcado, nó de borda e servidor central.

Na sequência, foi desenvolvido o dispositivo físico de chamada, baseado em *ESP32-CAM*. Essa etapa envolveu a seleção dos componentes eletrônicos, a elaboração do esquema elétrico, o desenvolvimento da placa de circuito impresso e a construção da caixa de montagem. O artefato resultante deve ser responsável por interagir com o usuário, enviar imagens e apresentar retorno local sobre os estados do sistema.

A terceira etapa correspondeu ao desenvolvimento do *firmware*. O código foi dividido em uma tarefa *FreeRTOS* para cada funcionalidade principal do microcontrolador. Também foi implementada uma máquina de estados para representar situações como inicialização, ausência de rede, operação normal, espera por resposta do servidor e economia de energia.

Na quarta etapa, foi implementado o nó de borda, responsável por intermediar a comunicação entre os dispositivos embarcados e o servidor central. Projetado para execução em computador de placa única, o nó realiza recepção, armazenamento temporário,

processamento de dados, publicação de mensagens e sincronização com a aplicação principal. Nessa camada, as imagens enviadas pela *ESP32-CAM* são processadas pelo serviço de reconhecimento facial e convertidas em eventos de presença quando uma face cadastrada é identificada.

A quinta etapa consistiu no desenvolvimento do servidor central AutoPonto, que concentra a lógica acadêmica do sistema, sendo ela composta por cadastros, eventos de presença, coleta, geração e proteção de biometria facial, sincronização com os nós de borda, relatórios e interface com usuários. O servidor também foi integrado à plataforma InterSCity, utilizada como camada complementar para métricas e monitoramento dos dispositivos através do envio de telemetria operacional.

Por fim, foi definida a avaliação preliminar, organizada em duas frentes:

- **Verificação funcional**: conferência do comportamento físico e sistêmico do protótipo, incluindo montagem eletrônica, *firmware*, telas, comunicação entre componentes e atendimento aos requisitos funcionais;
- **Métricas técnicas**: coleta e análise de tempos de processamento, uso de recursos computacionais, taxas de sucesso, falhas observadas e registros produzidos durante a simulação controlada.

3.2 Requisitos do sistema

A definição dos requisitos tomou como referência a dinâmica comum das aulas presenciais do curso de Engenharia da Computação da Universidade Federal do Maranhão. Nesse cenário, uma aula possui turma, professor, alunos matriculados, horário, sala e um procedimento de registro de frequência. A proposta do sistema parte dessa lógica já existente, substituindo a chamada oral por um fluxo automatizado baseado em reconhecimento facial.

3.2.1 Requisitos funcionais

Os requisitos funcionais descrevem as funções mínimas, ou de alta prioridade, que o sistema deve executar. No Quadro 2, sistematizam-se os requisitos funcionais considerados para o protótipo.

Quadro 2 – Requisitos funcionais do protótipo

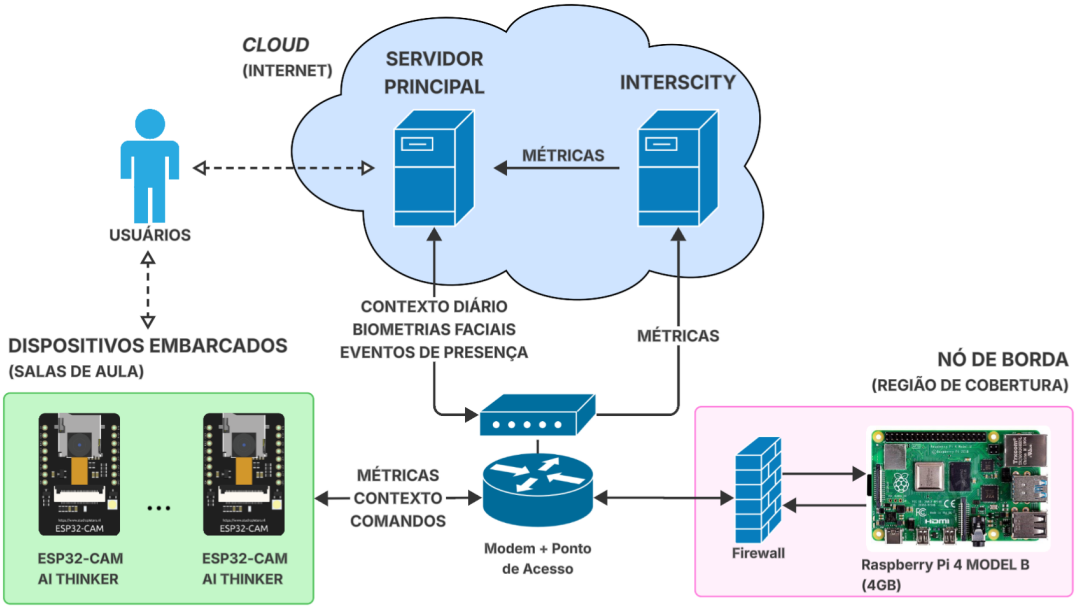
Código	**Requisito**	**Evidência de verificação**	**Origem**
RF01	Permitir o cadastro e a associação de alunos, professores, turmas, aulas, salas e dispositivos de chamada.	Registros administrativos, telas do sistema e consultas da API.	Servidor
RF02	Permitir que o aluno cadastre sua biometria facial a partir de capturas enviadas pelo sistema.	Registro biométrico criado, geração de vetor facial e mensagens de erro em capturas inválidas.	Servidor
RF03	Impedir o cadastro ativo de uma biometria facial já associada a outro aluno.	Registro de tentativa recusada ou validação de similaridade facial.	Servidor
RF04	Enviar ao nó de borda os dados necessários para a chamada do dia, incluindo aulas, alunos elegíveis e biometrias cadastradas.	Execução da sincronização, quantidade de aulas, alunos e biometrias enviadas.	Servidor e Borda
RF05	Permitir que o dispositivo consulte se há aula vigente ou próxima aula associada à sua sala.	Requisições *GET /context*, resposta recebida e comportamento do dispositivo.	Dispositivo e Borda
RF06	Permitir que o dispositivo capture e envie imagens ao nó de borda durante uma chamada ativa.	Requisições *POST /frame*, resposta recebida e comportamento da borda.	Dispositivo e Borda
RF07	Processar as imagens recebidas no nó de borda para detectar face e comparar com as biometrias dos alunos da aula.	Resultados dos reconhecimentos faciais e registros do programa.	Borda
RF08	Registrar presença quando a face reconhecida corresponder a aluno matriculado na aula em andamento.	Evento de presença gerado e aluno identificado.	Borda e Servidor
RF09	Enviar retorno ao dispositivo após reconhecimento, falha ou mudança de contexto da aula.	Mensagens MQTT, retorno no *display* e sinalização local.	Borda e Dispositivo
RF10	Disponibilizar as presenças registradas em telas e relatórios para alunos e professores.	Telas de frequência, relatório por turma/aula e consultas de presença.	Servidor

Fonte: Autoria própria (2026).

3.3 Arquitetura geral do sistema distribuído

Na Figura 8, evidencia-se a arquitetura geral do AutoPonto. O sistema foi organizado em quatro partes principais: dispositivo embarcado, nó de computação de borda, servidor central e a plataforma InterSCity.

Figura 8 – Arquitetura geral do AutoPonto



Fonte: Autoria própria (2026).

Os dispositivos baseados em *ESP32-CAM* e o nó de borda se comunicam por meio de um roteador configurado como ponto de acesso, em uma rede local, o que permite a troca de dados sem depender diretamente da Internet. A conexão externa continua necessária para sincronizar dados com o servidor central e para publicar informações operacionais dos dispositivos embarcados na plataforma InterSCity.

O dispositivo embarcado, localizado diretamente nas salas de aula, atua como ponto físico de chamada. Sua função é obter o contexto da aula atual ou próxima, capturar imagens da câmera, enviá-las ao nó de borda e apresentar retorno local ao usuário por meio de texto, luzes e som. O dispositivo, portanto, apenas realiza a interação local e encaminha os dados necessários ao processamento.

O nó de borda atua como intermediário entre as salas de aula e o servidor central. Ele recebe as imagens enviadas pelas *ESP32-CAM*, mantém uma cópia local dos dados necessários às chamadas do dia, atualizando-os, quando possível, após cada início de horário, e executa o reconhecimento facial. Essa cópia local reúne informações sobre dispositivos, salas, aulas, alunos vinculados e vetores biométricos, o que permite o registro de presença mesmo quando a conexão externa estiver indisponível no momento, sendo registrado, de fato, em uma próxima sincronização.

A plataforma InterSCity aparece como uma camada complementar da arquitetura. Sua função é representar recursos IoT e receber dados de telemetria operacional dos dispositivos por meio do nó de borda. Essa integração não substitui o servidor central nem participa diretamente da decisão de presença. Por isso, uma falha na publicação desses dados não impede o fluxo principal da aplicação.

O servidor central AutoPonto concentra a lógica acadêmica da aplicação. Nele são mantidos os cadastros de alunos, professores, turmas, aulas, salas, dispositivos, biometrias e registros de presença. Essa camada também disponibiliza as telas *web* utilizadas por alunos, professores e administradores, além das consultas e relatórios de frequência. De modo complementar, o servidor central recupera dados operacionais publicados na InterSCity e os apresenta em um mapa e em painéis interativos de monitoramento.

3.4 Desenvolvimento do dispositivo embarcado

O dispositivo embarcado foi projetado a partir da junção entre circuitos de alimentação, microcontrolador, sensor de presença, *display* e circuito indicador. O esquema elétrico e o desenho da placa de circuito impresso foram elaborados no KiCad, uma suíte de código aberto para projetos eletrônicos, utilizada para desenvolvimento de esquemáticos elétricos e leiaute de PCB (KiCad, s.d.).

3.4.1 Circuito de alimentação

Foi utilizada uma fonte Samsung EP-TA50BW com saída de 5 Vcc e corrente máxima de 1,55 A para alimentação geral do dispositivo. Essa tensão foi aplicada a um conversor MP1584 ajustado para 3,3 V, responsável por alimentar o restante do circuito. O conversor foi escolhido por aceitar entrada entre 4,5 V e 28 V e fornecer corrente de saída de até 3 A (Monolithic Power Systems, 2011).

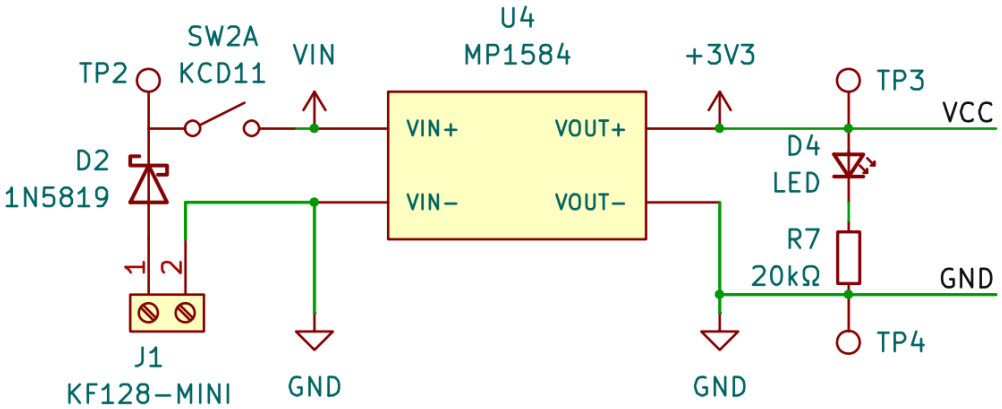
A escolha da fonte levou em conta principalmente o consumo da *ESP32-CAM*, pois esse módulo reúne microcontrolador, câmera, memória pseudoestática de acesso aleatório (*Pseudo Static Random Access Memory*, PSRAM), Wi-Fi e diodo emissor de luz (*Light-Emitting Diode*, LED) de *flash*. O documento técnico da placa informa consumo de 180 mA em 5 V com o *flash* desligado e 310 mA com o *flash* em brilho máximo (DFRobot, s.d.). Mesmo o protótipo utilizando alimentação em 3,3 V, esses valores serviram como referência para considerar a *ESP32-CAM* como a principal carga do circuito.

Além da *ESP32-CAM*, foram considerados os periféricos ligados à mesma alimentação. O *display* TFT apresenta corrente máxima de 10,5 mA na alimentação lógica e 60 mA na iluminação de fundo (Zhongjingyuan Technology Inc., s.d.). O *buzzer* ativo possui corrente máxima de 30 mA (Jiangsu Huawha Electronics Co., Ltd., 2019), enquanto o sensor SR602 apresenta corrente em repouso de 20 µA (UELECTRONICS, s.d.). Os módulos periféricos somados possuem consumo menor que a placa de desenvolvimento, mantendo a fonte e o conversor com uma margem segura.

Na Figura 9, detalham-se os terminais da fonte conectados à placa por meio de um borne. A linha positiva passa por um diodo Schottky 1N5819, utilizado como proteção contra inversão de polaridade. Esse diodo suporta correntes de até 1 A e apresenta queda

de tensão máxima de 0,60 V (Vishay General Semiconductor, 2020). Após essa proteção, as tensões de entrada e saída do MP1584 foram verificadas pelos pontos de prova TP2, TP3 e TP4.

Figura 9 – Circuito de alimentação do dispositivo



Fonte: Autoria própria (2026).

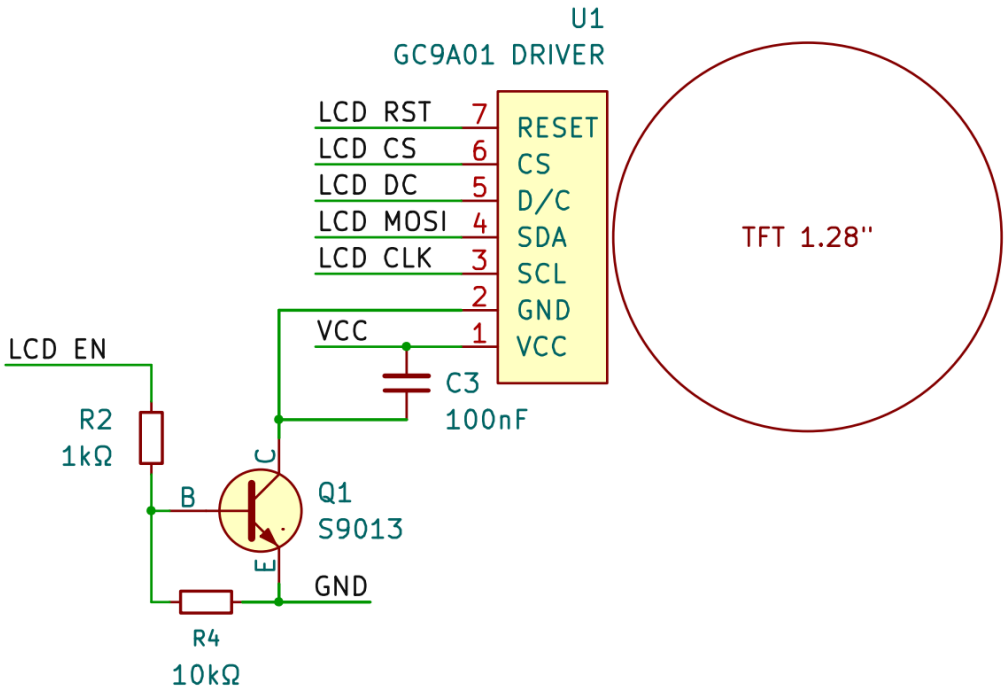
Também foi incluído um interruptor do tipo gôndola para ligar e desligar o circuito sem desconectar a fonte. A escolha se justifica porque o modelo KCD11 possui especificação de 250 Vca/3 A ou 125 Vca/6 A, superior até mesmo à capacidade máxima da fonte utilizada (NCR, s.d.). Na saída de 3,3 V, foi inserido um LED azul com resistor de 20 kΩ como indicação visual de alimentação.

3.4.2 Circuito do *display*

O módulo TFT GC9A01 foi utilizado para apresentar mensagens de estado ao usuário, como ausência de aula, espera por reconhecimento e retorno após a tentativa de chamada. Sua alimentação foi conectada à linha de 3,3 V, compatível com a tensão máxima de operação lógica indicada para o módulo (Zhongjingyuan Technology Inc., s.d.). A comunicação com a *ESP32-CAM* foi realizada por interface SPI, com linhas de *clock*, dados, seleção, reinicialização e controle de dados/comando.

Na Figura 10, apresenta-se o circuito do *display*. O retorno ao terra foi chaveado pelo transistor Q1, modelo S9013, capaz de suportar uma corrente de coletor de até 500 mA, valor superior à corrente máxima prevista para o módulo TFT e sua iluminação (Fairchild Semiconductor, 2002). Um sinal do microcontrolador na base desliga o módulo quando o dispositivo permanece inativo. O resistor de 1 kΩ limita a corrente de base, enquanto o resistor de 10 kΩ mantém a base em nível baixo quando o pino de controle não está ativo.

Figura 10 – Circuito de acionamento do *display* TFT



Fonte: Autoria própria (2026).

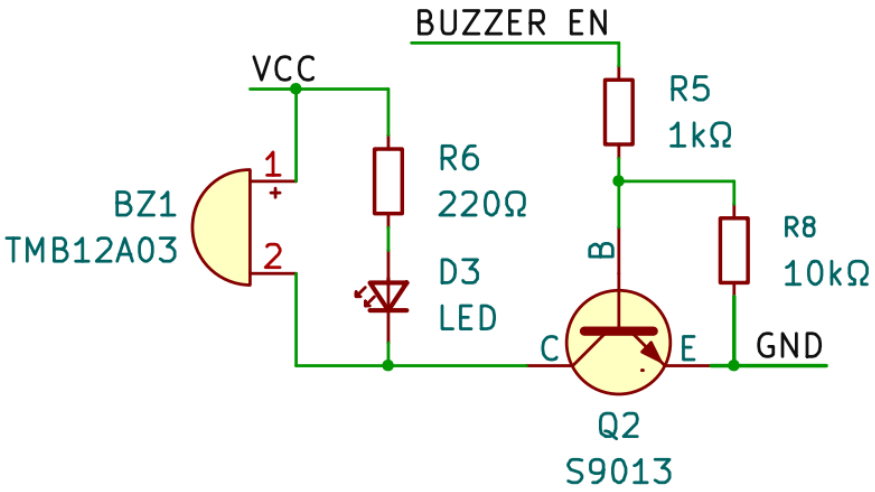
Por fim, foi adicionado um capacitor cerâmico de 100 nF próximo ao módulo. Esse capacitor atuou como desacoplamento local, reduzindo variações rápidas de tensão provocadas pelo conversor de alimentação do circuito e pela atividade do *display*.

3.4.3 Circuito indicador

O circuito indicador é formado por um LED verde e por um *buzzer* ativo TMB12A03. Seu uso foi reservado para respostas curtas associadas ao registro positivo de presença do usuário. O *buzzer* foi alimentado pela linha de 3,3 V, em sua tensão nominal, e possui corrente máxima especificada de 30 mA (Jiangsu Huawha Electronics Co., Ltd., 2019).

Na Figura 11, mostra-se o circuito de acionamento do indicador. O transistor Q2 foi utilizado como chave de baixo lado. Quando um sinal é enviado pelo microcontrolador, o transistor permite a passagem de corrente pelo LED e pelo *buzzer*. O LED verde foi ligado em série com resistor de 220 Ω, escolhido para limitar a corrente do indicador visual sem grande perda do brilho.

Figura 11 – Circuito indicador com LED e *buzzer*



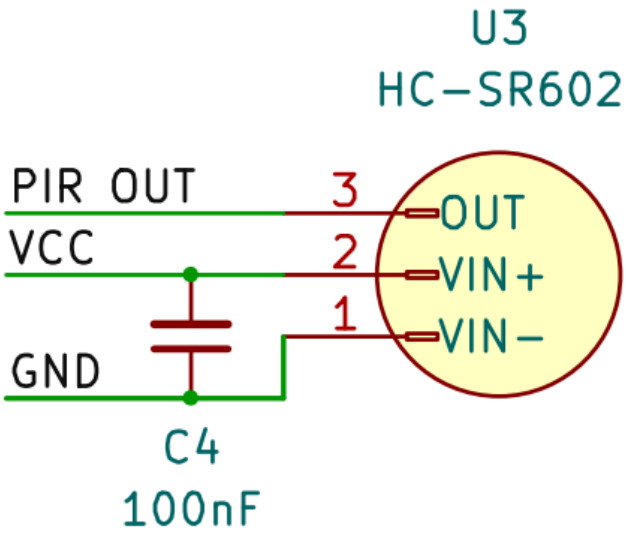
Fonte: Autoria própria (2026).

3.4.4 Sensor de presença

O sensor de presença utilizado foi o SR602, baseado em detecção de movimento por infravermelho. No protótipo, sua função foi indicar a aproximação de um usuário ao dispositivo, permitindo que o *firmware* alternasse entre estados de atividade, espera e economia de energia. O módulo opera com alimentação entre 3,3 V e 15 V e fornece saída digital em nível alto de 3,3 V (UELECTRONICS, s.d.). Essas características permitiram sua ligação direta à alimentação de 3,3 V e à entrada digital da *ESP32-CAM*.

Na Figura 12, registra-se a ligação do sensor. Seguindo as orientações do *datasheet* e de modo semelhante ao *display*, foi adicionado um capacitor cerâmico de 100 nF entre alimentação e terra para desacoplamento.

Figura 12 – Circuito do sensor de presença



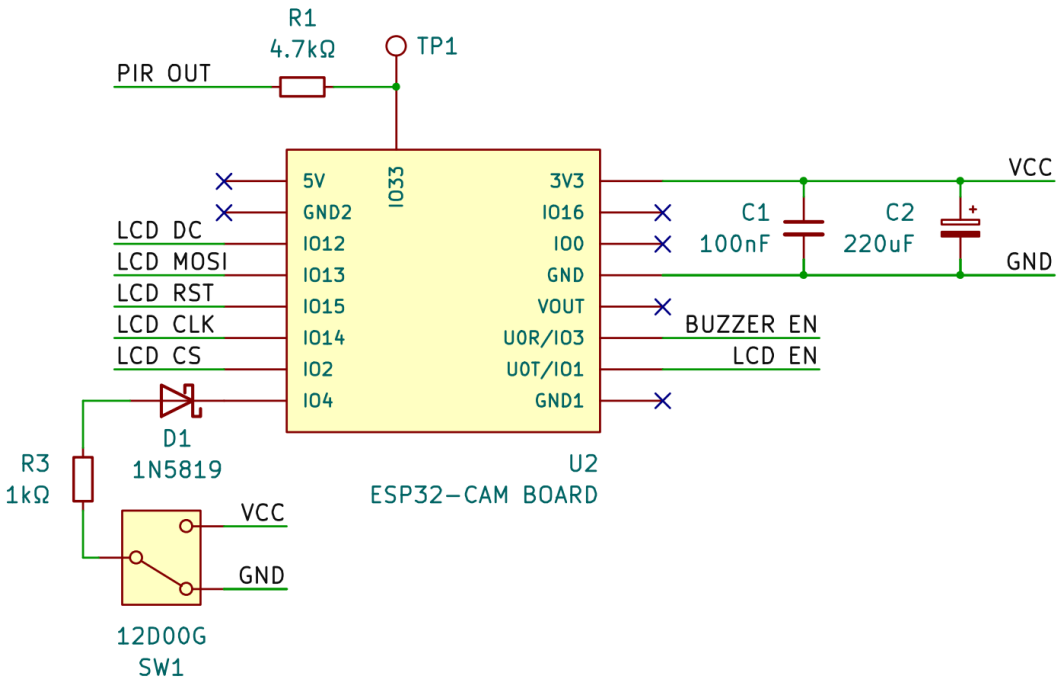
Fonte: Autoria própria (2026).

3.4.5 Microcontrolador e conexões principais

A unidade de controle do dispositivo foi uma placa de desenvolvimento *ESP32-CAM* AI-Thinker. Esse módulo foi escolhido por integrar microcontrolador, Wi-Fi e câmera em uma mesma placa. A placa é acompanhada por uma câmera OV2640, suporta múltiplos modos de economia de energia e execução com FreeRTOS (DFRobot, s.d.).

Na Figura 13, explicitam-se as conexões principais do microcontrolador. Cinco pinos foram reservados para a comunicação SPI com o *display*. O sinal de saída do sensor de presença foi conectado ao pino de entrada número 33, por meio de resistor de 4,7 kΩ, permitindo ao *firmware* identificar a presença de usuário próximo ao dispositivo.

Figura 13 – Conexões da placa de desenvolvimento *ESP32-CAM*



Fonte: Autoria própria (2026).

Os pinos U0R e U0T foram utilizados como saídas digitais para ativação do *buzzer* e do *display*. Essa decisão aproveitou os poucos pinos disponíveis no módulo, mas exigiu atenção durante gravação do *firmware* e desativação de qualquer monitoramento serial, pois esses sinais compartilham funções com a interface serial da placa.

A alimentação da *ESP32-CAM* foi feita pela linha de 3,3 V proveniente do conversor MP1584. Foram adicionados um capacitor eletrolítico de 220 μF e um capacitor cerâmico de 100 nF entre tensão de alimentação positiva (VCC) e terra (GND). O capacitor eletrolítico foi posicionado próximo ao microcontrolador e utilizado como reservatório de energia rápido em picos de consumo de câmera, de *flash* e de Wi-Fi, enquanto o capacitor cerâmico atuou no desacoplamento de ruídos de alta frequência.

Por fim, o pino IO4 foi associado a uma chave tátil por meio de resistor e diodo Schottky 1N5819. Na placa *ESP32-CAM*, esse pino serve para controlar o LED de *flash* integrado ao módulo. A inclusão do diodo evitou uma ligação direta entre a chave e o pino, reduzindo o risco de conflito elétrico quando o IO4 estivesse configurado como saída ou em estados intermediários de inicialização.

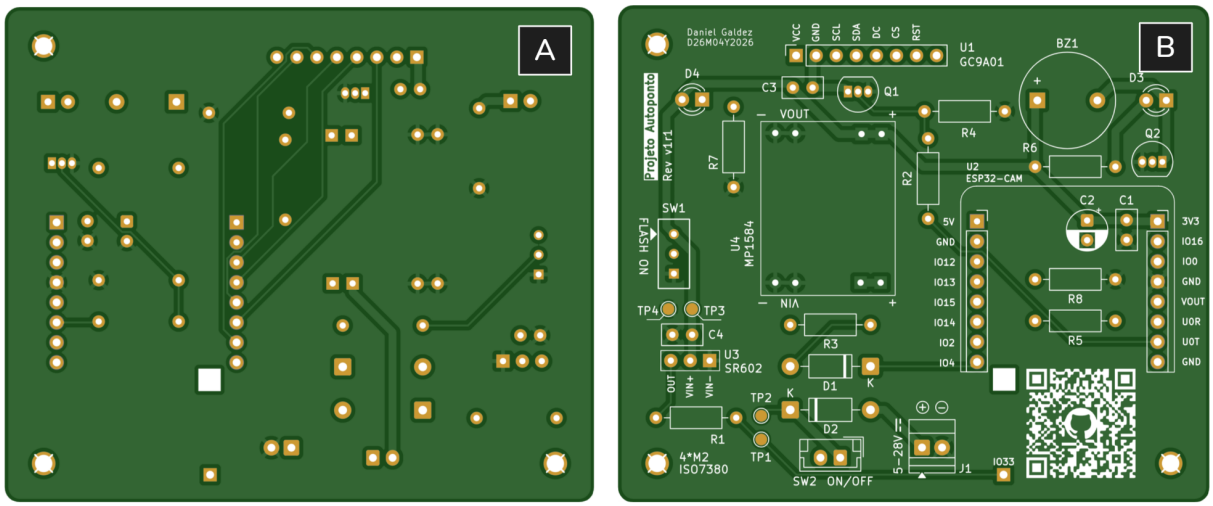
3.4.6 Projeto da PCB

Após a definição do esquema elétrico, foi elaborado o leiaute da placa de circuito impresso. A placa foi organizada em duas faces, para melhor distribuição das trilhas de cobre e evitar o cruzamento entre elas.

No roteamento, foram adotadas espessuras de 1 mm para as linhas de alimentação e trilhas de 0,25 mm para os sinais digitais. Essa separação foi usada porque a alimentação atende os módulos de maior consumo, enquanto os sinais SPI, controle do *display*, acionamento do indicador e leitura do sensor conduzem correntes menores. Também foi utilizado plano de terra para manter uma referência comum entre circuitos e evitar trilhas isoladas de GND.

Na Figura 14, evidenciam-se as duas faces da PCB desenvolvida. A visualização da frente permite observar a disposição principal dos módulos e conectores, enquanto a visualização do verso mostra a continuidade do roteamento e o aproveitamento da segunda face da placa.

Figura 14 – Visualização da PCB. A) Face frontal; B) Face traseira



Fonte: Autoria própria (2026).

3.4.7 Lista de componentes e preços

Considerando os componentes representados nos esquemas elétricos, o custo total estimado do dispositivo foi de R$ 296,81. Esse valor considera apenas os itens eletrônicos empregados na montagem do circuito e o lote mínimo de cinco placas impressas encomendadas através da fabricante JLCPCB, sem incluir ferramentas, frete, material de soldagem, impressão 3D ou eventuais peças de reposição. A lista completa de materiais, com quantidade, especificação, preço unitário e subtotal, encontra-se no Apêndice A.

3.4.8 Caixa de montagem impressa em 3D

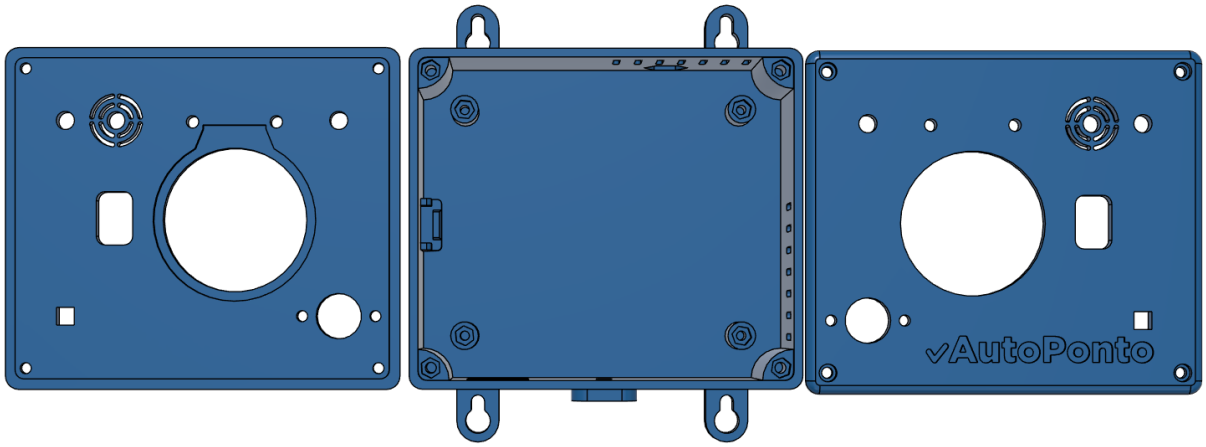
A caixa de montagem foi modelada no Autodesk Fusion 360, *software* utilizado para modelagem tridimensional de peças e preparação de projetos mecânicos (Autodesk, s.d.). O modelo foi desenvolvido a partir das dimensões da PCB, da disposição dos componentes e do uso de parafusos M2 com cabeça Allen, compatíveis com o padrão da Organização

Internacional de Normalização (*International Organization for Standardization*, ISO) 7380. As medidas dos recortes e encaixes foram conferidas pela medição dos componentes com um paquímetro.

A estrutura foi dividida em base e tampa. A base recebeu encaixes para a chave tátil de controle do *flash*, o interruptor gôndola e a antena externa da *ESP32-CAM*. Também foram incluídas aberturas de ventilação, suportes para fixação do dispositivo em parede e quatro colunas internas para fixação da PCB. Nessas colunas, foram previstos alojamentos hexagonais para porcas sextavadas.

A tampa foi projetada com recortes para os elementos que interagem com o usuário: câmera, *display*, sensor PIR, LEDs indicadores, saída sonora do *buzzer* e LED de *flash*. Na Figura 15, explicita-se o *design* final do modelo 3D.

Figura 15 – Modelo 3D da caixa de montagem



Fonte: Autoria própria (2026).

A impressão foi realizada em uma Bambu Lab A1 mini, com filamento de ácido polilático (*Polylactic Acid*, PLA) azul. O PLA foi escolhido por ser adequado à prototipagem de projetos eletrônicos e apresentar boa disponibilidade (Bambu Lab, s.d.).

3.5 Firmware do dispositivo ESP32

O *firmware* do dispositivo foi desenvolvido utilizando o *framework* Arduino sobre o ESP32. Sua função no sistema foi controlar a câmera, o *display*, a comunicação com o nó de borda, o recebimento de comandos MQTT, o sensor PIR e os modos de economia de energia.

3.5.1 Organização em tarefas FreeRTOS

O código do dispositivo foi dividido em tarefas FreeRTOS para separar responsabilidades executadas em paralelo. As tarefas de câmera e *display* foram fixadas no núcleo de aplicação, enquanto as tarefas de rede e MQTT foram fixadas no núcleo de protocolo. O laço principal do Arduino permaneceu responsável por rotinas de coordenação, como expiração do contexto da aula, tratamento do sensor PIR, controle de *idle*, acionamento de *deep sleep* e retorno positivo por buzzer e LED.

No Quadro 3, organiza-se a distribuição das tarefas, seus núcleos e prioridades. As prioridades foram definidas após testes empíricos do tempo de processamento das rotinas, com maior prioridade atribuída à captura de imagem e separação das tarefas mais custosas entre os núcleos disponíveis.

Quadro 3 – Organização das tarefas do firmware

Tarefa	**Núcleo**	**Responsabilidade no protótipo**	**Prioridade**
Display	Aplicação	Renderizar mensagens e pré-visualização em tempo real da câmera no *display* TFT.	2
Câmera	Aplicação	Capturar imagens no formato *Joint Photographic Experts Group* (JPEG) 240×240, alimentar as filas de pré-visualização e de publicação de quadros.	3
Rede	Protocolo	Gerenciar Wi-Fi, buscar contexto por HTTP e enviar imagens JPEG ao nó de borda.	2
MQTT	Protocolo	Conectar ao *broker* local, receber comandos e publicar estados, métricas e eventos do PIR.	2
Loop	Aplicação	Coordenar expiração de contexto, PIR, *idle*, *deep sleep* e retorno positivo por LED/*buzzer*.	1

Fonte: Autoria própria (2026).

3.5.2 Modo *idle*

Após a organização das tarefas FreeRTOS, foi implementado um modo *idle* para reduzir atividades desnecessárias quando não havia usuário próximo ao dispositivo. Esse modo foi controlado a partir do sensor PIR, configurado como uma interrupção do sistema. Quando o sensor detectava movimento, a interrupção sinalizava ao *firmware* que havia presença próxima, mas quando não havia nova detecção por um intervalo definido, o dispositivo era marcado como parado e o comportamento das tarefas em execução era alterado.

No modo *idle*, a tarefa da câmera deixava de produzir capturas para pré-visualização e envio, evitando processamento contínuo de imagens. A tarefa de rede também passava a evitar o envio de quadros ao nó de borda, mantendo apenas as verificações necessárias ao contexto da aula. A tarefa do *display* reduzia a atualização visual e podia desligar a tela.

A comunicação MQTT, por sua vez, continuava disponível para publicar estados e receber comandos, pois essa comunicação era necessária para manter o dispositivo integrado.

Esse modo funcionou como uma etapa intermediária antes do *deep sleep*. Enquanto o *idle* reduzia o trabalho das tarefas sem encerrar completamente o funcionamento do dispositivo, o *deep sleep* era acionado apenas quando a ausência de movimento persistia por mais tempo. Com isso, o protótipo evitava capturas e atualizações contínuas quando não havia usuário em frente ao equipamento, mas ainda preservava a capacidade de retomar a operação quando o sensor PIR detectava nova presença.

3.5.3 Máquina de estados

A implementação das tarefas foi baseada em uma máquina de estados simples, usada para separar as principais condições de operação do protótipo. O diagrama completo dessa lógica é apresentado no Apêndice B. No Quadro 4, resumem-se os estados implementados e a função de cada um no fluxo do *firmware*.

Quadro 4 – Estados do firmware do dispositivo embarcado

Estado	**Função no protótipo**	**Transição principal**
BOOTING	Preparação dos pinos, câmera, *display*, identificador do dispositivo e tarefas principais.	Inicialização concluída.
NET OFF	Indicação de ausência de conexão Wi-Fi e tentativa de reconexão.	Wi-Fi conectado.
MQTT OFF	Indicação de falha de comunicação com o *broker* MQTT local.	MQTT conectado.
FETCHING	Consulta ao nó de borda para obter o contexto da aula vigente ou próxima.	Contexto válido recebido.
WORKING	Operação normal do dispositivo, com captura de imagem quando há chamada ativa.	Imagem enviada ao nó de borda.
WAITING SERVER	Espera pelo retorno do nó de borda após o envio da imagem.	Resposta recebida ou tempo limite atingido.
SLEEPING	Encerramento das tarefas e desligamento do *display* antes da economia de energia.	Acionamento do *deep sleep*.

Fonte: Autoria própria (2026).

3.6 Nó de computação de borda

O nó de computação de borda foi implementado para operar entre os dispositivos ESP32, o servidor central AutoPonto e a plataforma InterSCity. A solução foi preparada para execução em uma Raspberry Pi 4 Model B com 4 GB de memória, utilizando contêineres Docker. Esse modelo foi escolhido por reunir processamento em quatro

núcleos, memória suficiente para os serviços locais, conectividade de rede e baixo consumo (Raspberry Pi, s.d.).

O programa que reúne os serviços locais foi chamado de *EdgeNode* e foi projetado para manter o fluxo de chamada próximo à sala de aula, reduzindo a dependência de requisições contínuas ao servidor central. Quando possuísse um *snapshot* válido, composto por uma parte relevante dos dados acadêmicos e biométricos recuperados do servidor central, o nó podia continuar reconhecendo presenças durante o dia. A conexão externa permaneceu necessária apenas para sincronizar os dados, diariamente e antes de cada horário de aula, além de publicar telemetria dos dispositivos fisicamente próximos na InterSCity. Ainda assim, o reconhecimento local dependeu principalmente dos dados baixados previamente.

3.6.1 Componentes do *EdgeNode*

O *EdgeNode* foi organizado em quatro componentes principais. O serviço *edge-app* expôs a API HTTP consumida pelos ESP32, escutou *logs* MQTT, publicou comandos aos dispositivos, consumiu eventos de presença e executou a sincronização com o servidor central. O serviço *face-worker* processou os quadros recebidos, executando detecção e reconhecimento facial. O Redis armazenou o *snapshot* do dia, filas e presenças pendentes. O Mosquitto atuou como *broker* MQTT local.

3.6.2 Fluxo de reconhecimento facial

O fluxo de reconhecimento facial foi iniciado no dispositivo embarcado, que consultou o serviço local do *EdgeNode* por meio da rota *GET /context*. Nessa etapa, o nó validou o identificador do dispositivo e o segredo compartilhado enviados pelo cabeçalho da requisição. Também foi necessário confirmar a data do *snapshot* armazenado no Redis e a associação do dispositivo a uma sala. Quando havia aula em andamento ou próxima aula, o serviço retornou o contexto necessário para orientar o funcionamento do *firmware*.

Durante uma chamada ativa, o dispositivo enviou uma imagem JPEG ao *EdgeNode* por meio da rota *POST /frame*. O *edge-app* validou a requisição, verificou se havia aula aberta para a sala e conferiu o limite da fila de processamento. Quando a fila ainda possuía capacidade, a captura de câmera foi armazenada no Redis com os dados do dispositivo, da sala, da aula e do instante de recebimento. Esse uso de fila separou o ingresso de imagens do processamento facial, evitando que a resposta ao *POST* da ESP32 dependesse do tempo de reconhecimento.

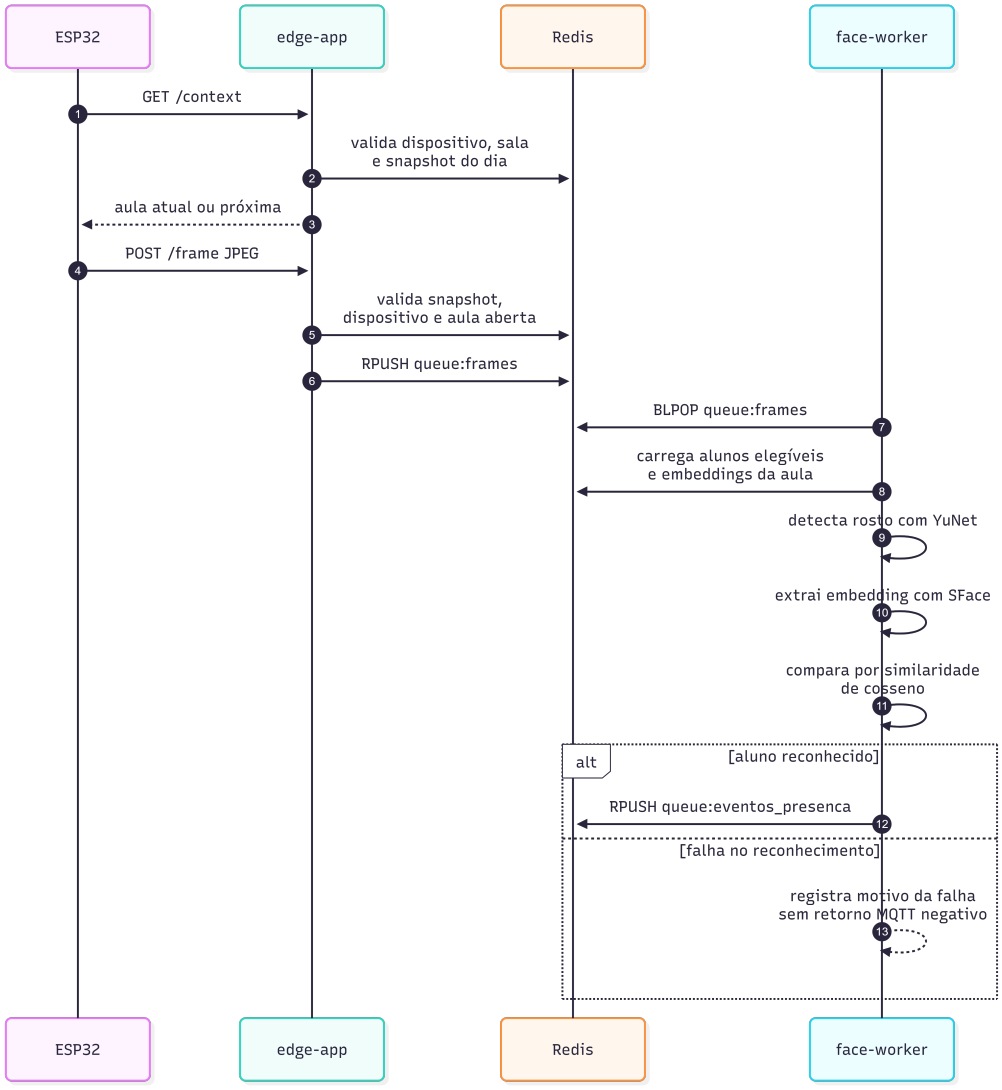
Dessa forma, o processamento das imagens foi concentrado no *face-worker*. Para cada imagem consumida da fila, o serviço decodificou o JPEG, aplicou o YuNet para

detecção facial e utilizou o SFace para gerar o *embedding* da face detectada. Em seguida, foram carregados do Redis apenas os *embeddings* dos alunos elegíveis para a aula em andamento, restringindo a comparação ao contexto acadêmico correto.

A decisão de reconhecimento foi feita por similaridade de cosseno. O melhor candidato foi selecionado pelo maior *score*, e o evento de presença foi gerado somente quando esse valor atingiu o limiar configurado no ambiente do *EdgeNode*. Situações como erro de decodificação, ausência de face, falha na extração do *embedding* ou face não reconhecida foram registradas como falhas do processamento e tratadas com o descarte da imagem, sem envio de retorno negativo ao dispositivo.

Na Figura 16, ilustra-se o fluxo local de reconhecimento facial. O diagrama inclui os comandos Redis usados para manipulação das filas em *cache*.

Figura 16 – Diagrama de sequência do fluxo local de reconhecimento facial no *EdgeNode*



Fonte: Autoria própria (2026).

3.6.3 Sincronização com servidor principal

A sincronização do *EdgeNode* com o servidor principal foi feita por uma rotina executada no contêiner do *edge-app*. Essa rotina foi disponibilizada para execução manual ou por agendamento do sistema operacional. Em cada *pull* de dados, o *EdgeNode* enviou sua identificação e um *token* específico, usados pelo servidor central para autenticar requisições de nós de borda.

A resposta compreendeu a data do *snapshot*, o instante de sincronização e os dados necessários para o *cache* Redis local, como dispositivos por código, aulas por sala, alunos por aula, alunos por identificador e *embeddings* faciais criptografados. Antes de substituir o *cache* local, o *EdgeNode* descriptografou os *embeddings* com uma chave Fernet compartilhada e validou o conteúdo recebido. Quando algum *embedding* falhou, o *snapshot* anterior permaneceu em uso.

Os eventos de presença gerados no nó de borda foram enviados ao servidor principal por um *endpoint* separado, acionado no momento da criação do evento. Quando o envio fosse concluído com sucesso, a presença era marcada como sincronizada localmente. Em caso de falha, o evento permaneceu pendente no Redis para próxima sincronização. Por fim, após salvar os novos dados, o *EdgeNode* publicou um comando de atualização de contexto para todos os dispositivos associados.

3.6.4 MQTT e InterSCity

O *edge-app* assinou o conjunto de tópicos de *logs* dos dispositivos para receber publicações MQTT. As mensagens recebidas foram tratadas conforme o tipo informado no conteúdo da publicação:

- **Estado operacional**: registrou a condição atual do dispositivo, como inicialização, ausência de rede ou busca de contexto;
- **Evento do PIR**: indicou presença detectada pelo sensor de movimento;
- **Métricas técnicas**: reuniu informações de métricas calculadas no *firmware* do dispositivo publicante.

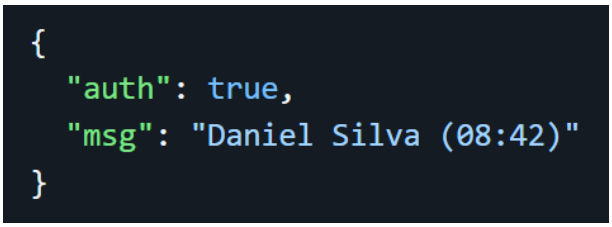
Quando o dispositivo indicou estado *idle*, as métricas foram ignoradas para evitar registrar o equipamento em uma condição que não faz parte do escopo da análise.

Os comandos MQTT foram enviados em tópicos individuais, para que cada dispositivo recebesse apenas mensagens destinadas ao seu identificador. O principal comando foi o retorno de presença positiva, contendo uma mensagem abreviada com nome e horário. Ao receber esse comando, o dispositivo atualiza o *display* e aciona o LED

verde e o *buzzer* por curto intervalo. Também foi utilizado o comando de renovação de contexto, que força dispositivos consultarem novamente o nó de borda.

Na Figura 17, registra-se um exemplo de *payload* enviado no comando de retorno positivo.

Figura 17 – *Payload* MQTT para ativar retorno positivo em dispositivos embarcados



Fonte: Autoria própria (2026).

Ademais, a integração com a InterSCity foi tratada como camada complementar de observabilidade. O *EdgeNode* publicou estado, eventos PIR e métricas técnicas no *Resource Adaptor* apenas quando o dispositivo possuía identificador correspondente dessa plataforma no *snapshot*. Falhas de publicação não foram projetadas para impedir o reconhecimento facial nem o registro de presenças.

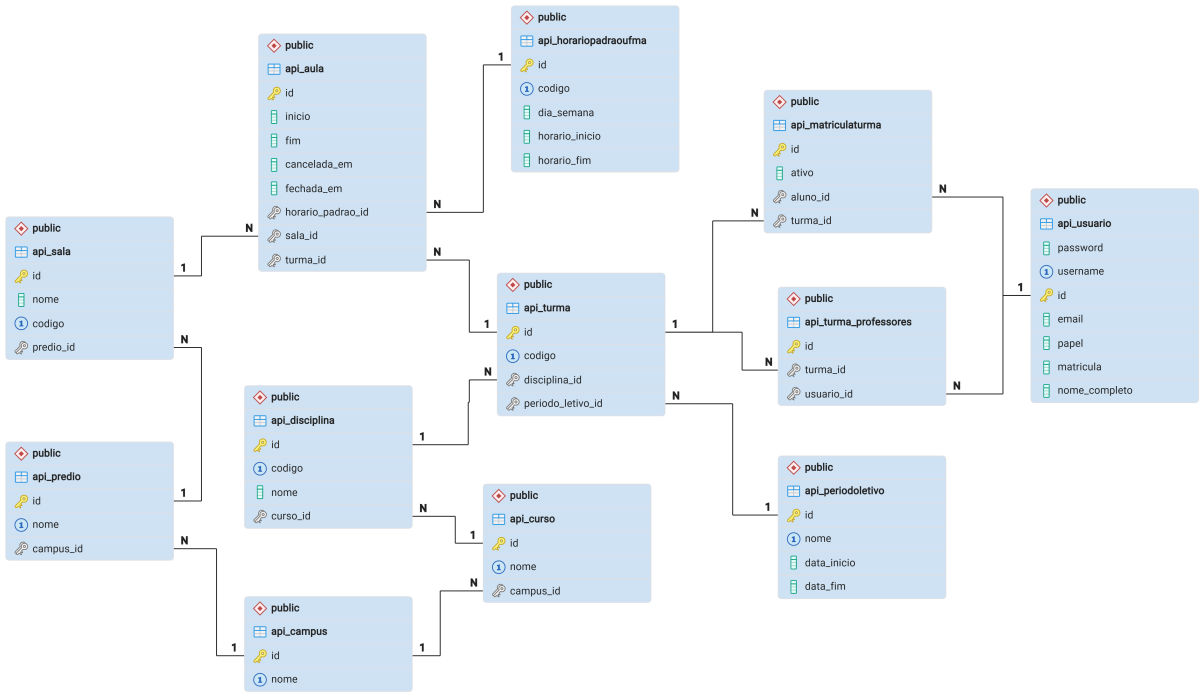
3.7 Servidor central AutoPonto

O servidor central AutoPonto foi desenvolvido como a camada mais importante do sistema distribuído. Nele foram criados e modificados os dados reais sobre o ambiente acadêmico e dispositivos IoT. Diferente do *EdgeNode*, que executa o processamento de imagens próximo do usuário final, o servidor central concentra regras acadêmicas, autenticação, relatórios e interface *web*.

3.7.1 Modelagem e implementação do *backend*

A modelagem do servidor central foi realizada antes da consolidação dos fluxos da API, com o objetivo de organizar as entidades persistentes e as relações necessárias ao funcionamento do sistema. Essa etapa delimitou usuários, papéis, cursos, disciplinas, turmas, matrículas, aulas, salas, dispositivos, nós de borda, biometrias faciais, registros de presença e eventos de reconhecimento. Na Figura 18, recorta-se o diagrama de entidade-relacionamento referente às entidades acadêmicas do servidor central.

Figura 18 – Diagrama entidade-relacionamento das entidades acadêmicas



Fonte: Autoria própria (2026).

A implementação do *backend* foi feita em Django e *Django REST Framework*. O PostgreSQL foi adotado como banco persistente do servidor central, responsável por armazenar dados acadêmicos e *embeddings* faciais criptografados. O modelo de dados foi definido no próprio código, por meio das classes de modelo e do mapeamento objeto-relacional (*Object-Relational Mapping*, ORM) do Django. A partir dessas classes, as tabelas, relacionamentos e restrições do banco de dados foram gerados automaticamente.

As regras acadêmicas também ficaram concentradas no servidor central. As aulas foram geradas a partir dos horários das turmas, mas seus estados foram calculados conforme a data, a sala e o intervalo previsto de início e fim. Para o registro automatizado de presença, foram considerados os seguintes critérios:

- **Cancelada**: quando a data de cancelamento estava preenchida;
- **Fechada**: quando a data de fechamento estava preenchida ou quando o horário de fim da aula já havia sido atingido;
- **Aberta**: quando o horário atual estava entre o início e o fim da aula;
- **Planejada**: quando a aula ainda não havia iniciado e não estava cancelada ou fechada.

Os *endpoints* administrativos foram implementados por *ViewSets*. Também foram criadas rotas específicas para autenticação *JSON Web Token* (JWT), dados do usuário

autenticado, calendário, *dashboards*, relatórios, biometria, mapa público de dispositivos e integração com os nós de borda. Os eventos de presença enviados pelo *EdgeNode* foram recebidos por um contrato separado, autenticado por *token* de nó.

3.7.2 Cadastro biométrico e segurança

A comunicação do *EdgeNode* com o servidor utilizou *tokens* específicos gerados individualmente por nó de borda. Já os demais atores do sistema utilizaram JWT, com *refresh token* em *cookie* protegido e *access token* em armazenamento local.

Uma vez autenticado, o cadastro biométrico foi realizado pelo próprio aluno na interface *web*, a partir de qualquer local. O *backend* recebeu a captura enviada pelo *website* em base64, validou o formato e o tamanho da imagem e utilizou a biblioteca *Open Source Computer Vision Library* (OpenCV) com YuNet/SFace para detectar o rosto e gerar o *embedding*.

Todos os *embeddings* gerados foram armazenados criptografados de forma síncrona com Fernet, utilizando-se uma chave fornecida por variável de ambiente. O servidor enviou ao *EdgeNode* apenas o vetor criptografado, e este o descriptografou localmente usando a mesma chave secreta.

Durante o cadastro biométrico, o novo *embedding* precisava ser comparado com todas as biometrias ativas já cadastradas para evitar duplicidade entre alunos. Em uma instituição com muitos discentes, consultar e descriptografar todos esses vetores diretamente do banco de dados aumentaria o custo da operação de modo considerável. Por esse motivo, o Redis foi utilizado como *cache* em memória para biometrias ativas descriptografadas.

Essa escolha não alterou o nível de proteção desses dados, pois um agente com acesso à memória do processo ou do ambiente de execução também poderia acessar a chave de descriptografia usada pela aplicação. Portanto, a criptografia foi mantida apenas como proteção do armazenamento persistente e durante a troca de dados com o nó de borda.

Quando uma biometria foi aceita, qualquer *embedding* ativo anterior do mesmo aluno foi revogado. A revogação removeu o vetor sensível e marcou o registro como inativo.

3.7.3 *Frontend* e telas da aplicação

O *frontend* foi implementado em React. A interface foi organizada por permissões de acesso, de acordo com o papel do usuário. O aluno acessa painel com aulas, próximas chamadas, frequência, calendário, cadastro biométrico, perfil e histórico de presenças. O professor acompanha turmas, chamadas, calendário, detalhes de aula e relatórios de frequência. O administrador manipula usuários, dados acadêmicos e de dispositivos através de formulários.

Também foi incluído um mapa IoT público, baseado nos nós de borda e dispositivos cadastrados no sistema. Esse mapa exibiu a localização dos nós de borda e suas últimas sincronizações. Ao selecionar um nó, foi possível acessar os dispositivos associados e consultar gráficos com suas métricas recentes ou em períodos fixos de duas ou 24 horas. Quando disponíveis, essas métricas foram recuperadas pela API principal a partir do *Collector* da InterSCity, onde os dispositivos embarcados publicaram seus dados operacionais.

3.8 Procedimentos de avaliação

3.8.1 Simulação controlada

A simulação controlada foi planejada para reproduzir chamadas acadêmicas em uma turma com salas, horários e alunos fictícios. O roteiro considerou a preparação de dados acadêmicos, cadastro biométrico, sincronização do *EdgeNode*, consulta de contexto pelo ESP32, envio de capturas da câmera, reconhecimento facial, retorno MQTT, registro de presença no servidor central e consulta pela interface *web*.

O servidor central foi executado localmente em um computador com processador Intel Core i7, 16 GB de memória de acesso aleatório (*Random Access Memory*, RAM), armazenamento de estado sólido (*Solid-State Drive*, SSD) e unidade de processamento gráfico (*Graphics Processing Unit*, GPU) dedicada NVIDIA. Nessa máquina, foram executados os serviços necessários à API principal e ao banco de dados durante a simulação. Esse ambiente foi utilizado apenas para os testes controlados, sem representar uma implantação definitiva em produção.

A integração com a InterSCity utilizou uma instância pública hospedada em servidor do Laboratório de Sistemas Distribuídos da Universidade Federal do Maranhão. Essa instância recebeu as métricas publicadas pelos dispositivos embarcados e permitiu consulta de dados pelo servidor central.

Entre os itens controlados estavam a quantidade de dispositivos usados, a máquina ou placa executando o *EdgeNode*, a massa de dados fictícia, a aula usada na simulação, os rostos autorizados, as tentativas de reconhecimento e as condições de iluminação.

3.8.2 Verificação funcional

A verificação funcional foi definida como a etapa responsável por conferir se o protótipo executava os comportamentos previstos para cada componente do sistema distribuído. Essa verificação considerou os requisitos funcionais apresentados no Quadro 2, a proteção dos dados por criptografia e a telemetria com o servidor InterSCity.

Além disso, foi observado se componentes físicos do protótipo se comportavam de maneira previsível, isto é, sem falhas técnicas inesperadas como componentes defeituosos, alimentação inadequada ou montagem incorreta. Por fim, foram definidas as condições para o registro de métricas do sistema, apresentadas no Quadro 5.

Quadro 5 – Condições da simulação controlada

Categoria	**Condição**
Local da simulação	Área social de uma residência convencional.
Período de execução	8 de junho de 2026 a 15 de junho de 2026. Sete rodadas de testes divididas em sete dias distintos.
Voluntários	Oito.
Dados persistidos no servidor principal	Um período letivo, um campus, um curso, um prédio, uma disciplina, uma sala, uma turma, sete aulas, um horário padrão de 24 horas, oito alunos, um professor, um administrador.
Tentativas de reconhecimento	Sete tentativas diárias por voluntário, com 392 totais.
Posicionamento do dispositivo	30 cm de distância do usuário, na altura dos olhos, ângulo normal de captura da câmera.
Iluminação	Iluminação natural da tarde.
Condição de rede	Rede local, dispositivo a cinco metros do ponto de acesso.
Serviços externos utilizados	API pública da plataforma InterSCity.

Fonte: Autoria própria (2026).

3.8.3 Coleta de métricas técnicas

A coleta de métricas técnicas foi organizada a partir das saídas nos três componentes principais do sistema distribuído: *firmware* do dispositivo embarcado, *EdgeNode* e servidor central AutoPonto. Essa separação permitiu observar, respectivamente, o comportamento do ESP32-CAM, o processamento local de borda e os fluxos acadêmicos no servidor.

No *firmware*, a coleta foi baseada nas mensagens enviadas ao tópico MQTT *log/{deviceId}* e posteriormente publicadas na plataforma InterSCity. No Quadro 6, apresentam-se os campos emitidos pelo dispositivo e a finalidade de cada grupo de métricas.

Quadro 6 – Métricas coletadas no *firmware* do ESP32-CAM

Grupo	**Campos**	**Unidade ou tipo**	**Finalidade na análise**
Estado operacional	*kind=status*, *status*	Categoria	Acompanhar estados como operação, busca de contexto, sono e indisponibilidade.
Memória interna	*heap_free*, *heap_min*, *heap_max*	Bytes	Verificar disponibilidade de memória dinâmica durante a simulação.
Memória PSRAM	*psram_free*, *psram_min*, *psram_max*	Bytes	Observar a margem de memória usada para captura e clonagem dos quadros JPEG.
Rede e envio HTTP	*rssi*, *post_max_ms*	dBm e ms	Relacionar qualidade do Wi-Fi e maior tempo de envio de imagem ao nó de borda.
Tarefas FreeRTOS	*avg_us.loop*, *avg_us.mqtt*, *avg_us.network*, *avg_us.camera*, *avg_us.display*	µs	Medir o custo médio de execução das rotinas principais do *firmware*.
Amostras das tarefas	*avg_count.loop*, *avg_count.mqtt*, *avg_count.network*, *avg_count.camera*, *avg_count.display*	Contagem	Ponderar as médias de execução recebidas em cada publicação MQTT.
Presença no sensor PIR	*kind=pir*, *presenca*	Evento booleano	Registrar detecção de movimento.

Fonte: Autoria própria (2026).

No *EdgeNode*, as métricas foram registradas em arquivos locais quando a coleta estava habilitada. A planilha armazenou as amostras individuais, enquanto um documento de texto consolidou quantidade e média por métrica. No Quadro 7, resumem-se essas métricas.

Quadro 7 – Métricas coletadas no *EdgeNode*

Grupo	**Métricas**	**Origem**	**Finalidade na análise**
API local do nó	Latência de */context* e */frame*	*edge-app*	Medir latência e sucesso das consultas de contexto e dos envios de quadros pela ESP32.
Fila de processamento	Espera do quadro antes do processamento	*face-worker*	Avaliar atraso entre o recebimento do quadro e o início do reconhecimento facial.
Reconhecimento facial	Detecção, extração de *embedding*, comparação e tempo total	*face-worker*	Separar os custos de detecção, extração do vetor facial, comparação e processamento total.
Sincronização com servidor	Tempo de *pull* do *snapshot* e eventos de falha	*edge-app*	Medir a obtenção do *snapshot* e contabilizar falhas de *pull*, aplicação do *snapshot*, envio de presenças e comando MQTT pós-sincronização.
Publicação InterSCity	Tempo de publicação e eventos de descarte/falha	*edge-app*	Medir latência, sucesso, falha e descarte por fila cheia nas publicações de telemetria.

Fonte: Autoria própria (2026).

No servidor central, a coleta foi implementada por um coletor de evidências técnicas semelhante ao usado no *EdgeNode*. No Quadro 8, apresentam-se os grupos escolhidos para o servidor principal.

Quadro 8 – Métricas coletadas no servidor central AutoPonto

Grupo	**Métricas**	**Ponto de coleta**	**Finalidade na análise**
Cadastro biométrico	Tempo total, tempo de geração do *embedding*, capturas recebidas, capturas decodificadas, faces detectadas e falhas de detecção	Serviço de biometria e *view* de cadastro	Medir o fluxo de cadastro facial, volume de capturas e falhas de detecção.
Proteção dos *embeddings*	Tempo de criptografia, descriptografia e listagem do *cache* biométrico	Camadas de criptografia e *cache*	Avaliar custo da criptografia, descriptografia e leitura do *cache* biométrico.
Sincronização com borda	Tempo total do *snapshot* e totais de dispositivos, aulas e *embeddings* enviados	Serviço de sincronização de borda	Caracterizar tempo e tamanho do *snapshot* enviado ao nó de borda.
Recepção de presenças	Tempo total de ingestão e quantidade de eventos recebidos	Sincronização e contrato de borda	Medir ingestão dos eventos reconhecidos localmente e tamanho dos lotes recebidos.
Consultas da interface	Tempo de resposta de perfil, *dashboards*, calendário, relatórios e mapa público	*Views* de interface e mapa público	Medir tempo de resposta das telas usadas na verificação funcional.
Consulta ao InterSCity	Latência da requisição, total de falhas e estado retornado	Serviço de integração InterSCity	Registrar latência, falhas e estados das chamadas ao *Collector*.

Fonte: Autoria própria (2026).

A análise dos dados técnicos foi planejada com estatística descritiva e inspeção dos registros da simulação. Para métricas numéricas, foram consideradas medidas de tendência simples, como média e mediana.

4 Análise dos Resultados

Este capítulo apresenta a análise dos resultados obtidos com o protótipo em uma simulação controlada. A avaliação concentrou-se na verificação técnica preliminar do sistema e não pretende demonstrar superioridade definitiva sobre métodos de frequência acadêmica tradicionais, mas indicar a viabilidade técnica inicial de um fluxo automatizado baseado em IoT, computação de borda e aplicação *web*.

4.1 Verificação funcional

A verificação funcional foi analisada a partir de evidências visuais, registros de servidores e telas da aplicação. Essa organização permitiu relacionar cada evidência ao comportamento esperado no Quadro 2.

4.1.1 Evidências do dispositivo embarcado

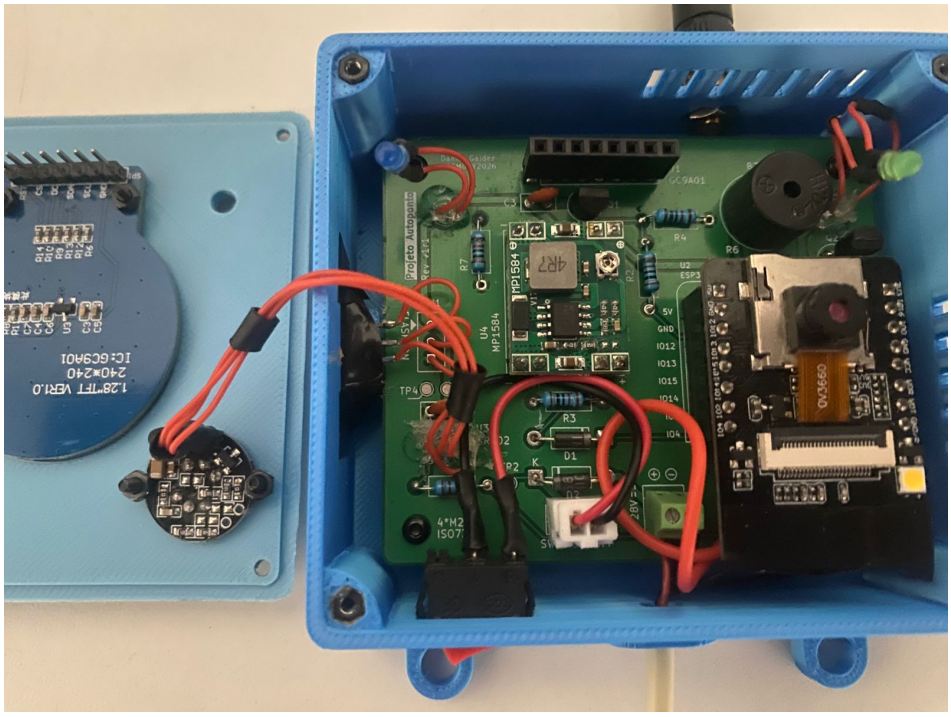
A montagem final reuniu caixa de proteção, câmera, *display*, sensor PIR, LED, *buzzer* e demais componentes eletrônicos. Na Figura 19, evidencia-se essa montagem e observa-se que o protótipo assumiu a função de ponto local de interação com o usuário, corroborando com a visão de Choudhary et al. (2025) que dispositivos embarcados são elementos físicos voltados à coleta de dados.

Figura 19 – Dispositivo AutoPonto montado na caixa de proteção

Fonte: Autoria própria (2026).

Na Figura 20, observa-se a disposição interna dos componentes, utilizada para conferir a coerência entre a montagem física e o projeto eletrônico.

Figura 20 – Montagem interna do protótipo e disposição dos componentes



Fonte: Autoria própria (2026).

A inspeção visual da montagem foi relevante para identificar possíveis causas de falhas durante os ensaios. Nessa etapa, foram identificados transistores, um sensor PIR e fios defeituosos, os quais foram substituídos antes da continuidade dos testes. Após essas correções, o conjunto passou a operar de forma estável, sem reincidência dos problemas observados inicialmente. Esse resultado indica que as instabilidades não estavam associadas ao projeto elétrico do sistema, mas a falhas pontuais de componentes e conexões físicas.

Durante seu funcionamento, o dispositivo apresentou ao usuário estados operacionais do *firmware*, como busca por conexão, busca por contexto, *feedback* da câmera, ausência de aula e mensagem positiva após reconhecimento facial. Na Figura 21, evidencia-se esse comportamento. Com isso, verificou-se o atendimento do RF05, pois o equipamento alterou sua operação a partir do contexto de aula recebido da borda, e do RF09, pois houve retorno local ao usuário após a resposta do sistema.

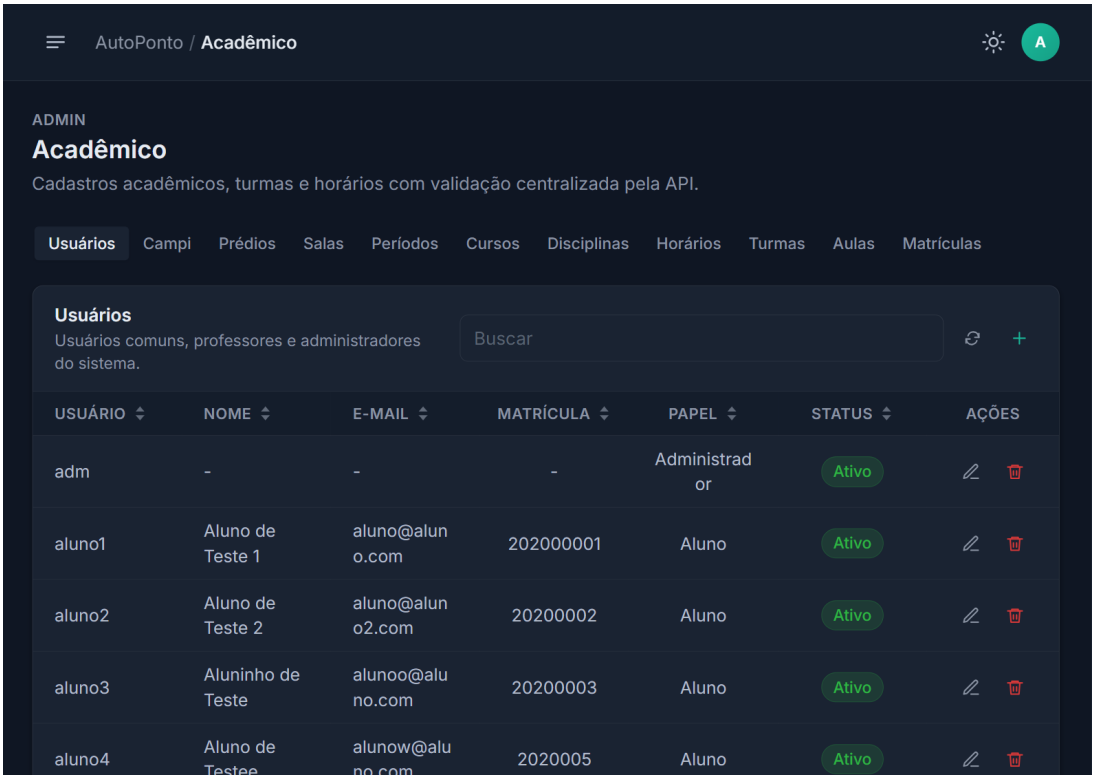
Figura 21 – Protótipo em funcionamento durante a simulação de chamada

Fonte: Autoria própria (2026).

4.1.2 Cadastros acadêmicos e biométricos

A primeira etapa sistêmica verificada foi a criação dos cadastros necessários ao fluxo de chamada. Na aplicação *web*, foram registrados e associados alunos, professor, turma, aula, sala, dispositivo e nó de borda. Na Figura 22, apresenta-se a tela administrativa utilizada para esses cadastros.

Figura 22 – Cadastros acadêmicos e operacionais utilizados na simulação

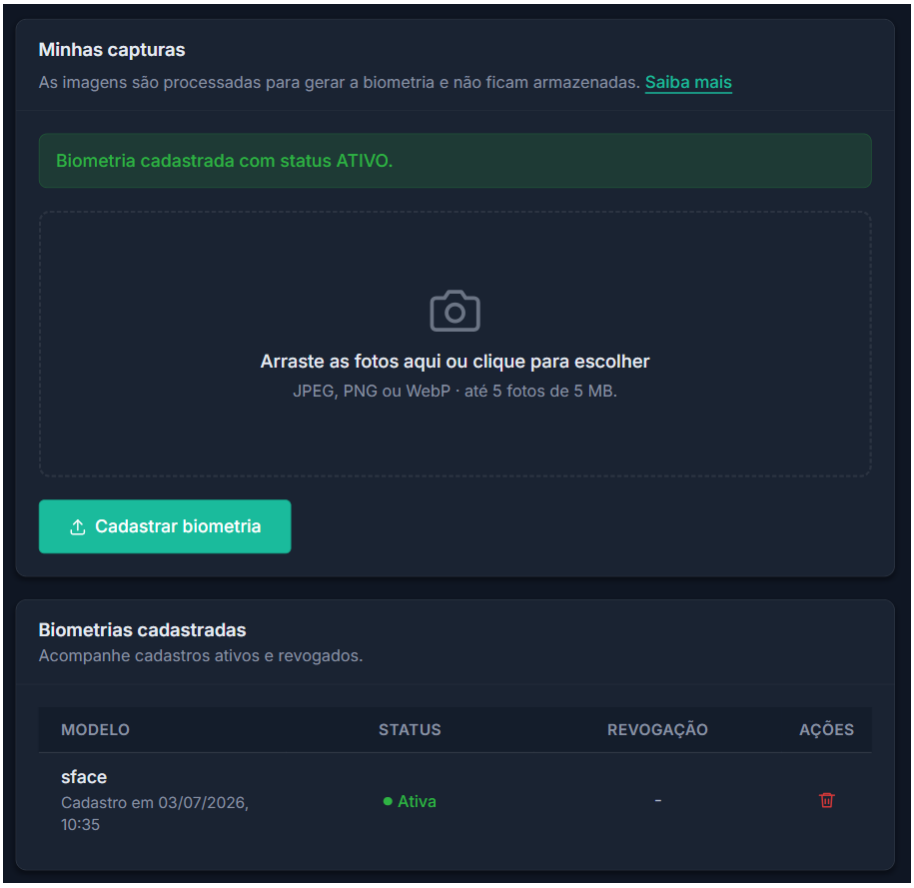


Fonte: Autoria própria (2026).

Essa evidência demonstra o cumprimento do RF01, pois o sistema permitiu registrar as entidades mínimas e relacioná-las ao equipamento físico de chamada. A organização desses dados em uma aplicação com API e interface é coerente com o papel de sistemas *web* em centralizar cadastros, autenticação, formulários e consultas para diferentes perfis de usuário (PATIL, 2025, p. 1–9).

O cadastro biométrico foi verificado pela tela do aluno, na qual capturas faciais foram enviadas ao servidor para validação, detecção da face e geração do vetor facial. Na Figura 23, evidencia-se essa verificação e confirma-se o atendimento do RF02, uma vez que o sistema permitiu ao aluno cadastrar a própria biometria facial. Dessa forma, o cadastro biométrico funcionou como etapa preparatória ao reconhecimento posterior no nó de borda.

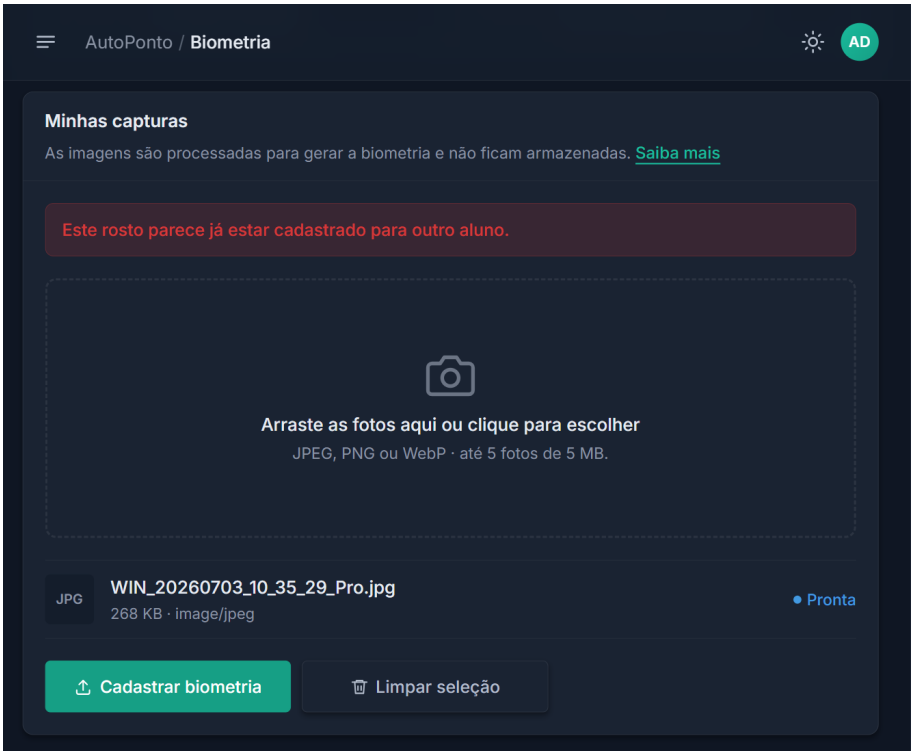
Figura 23 – Cadastro biométrico facial realizado na interface *web*



Fonte: Autoria própria (2026).

Também foi verificada a regra de impedimento de biometria ativa duplicada entre alunos distintos. Na Figura 24, registra-se essa validação. Com isso, o RF03 foi considerado cumprido no contexto da simulação, pois o servidor recusou a manutenção de um vetor facial ativo já associado a outro estudante.

Figura 24 – Validação contra duplicidade de biometria facial ativa



Fonte: Autoria própria (2026).

4.1.3 Sincronização da borda

Após os cadastros no servidor central, a sincronização do *EdgeNode* foi verificada pela obtenção do *snapshot* pelo nó de borda. Na Figura 25A, registra-se a resposta HTTP obtida durante a sincronização, enquanto, na Figura 25B, mostra-se o armazenamento dos dados no *cache* Redis local. Esses registros confirmam o cumprimento do RF04, pois demonstram que as informações necessárias ao reconhecimento local foram enviadas do servidor central para o *EdgeNode*. Esse resultado se relaciona à proposta de computação de borda, que desloca parte do armazenamento e do processamento para próximo da origem dos dados (KHAN et al., 2019).

Figura 25 – A) Resposta HTTP à sincronização; B) *Cache* Redis local

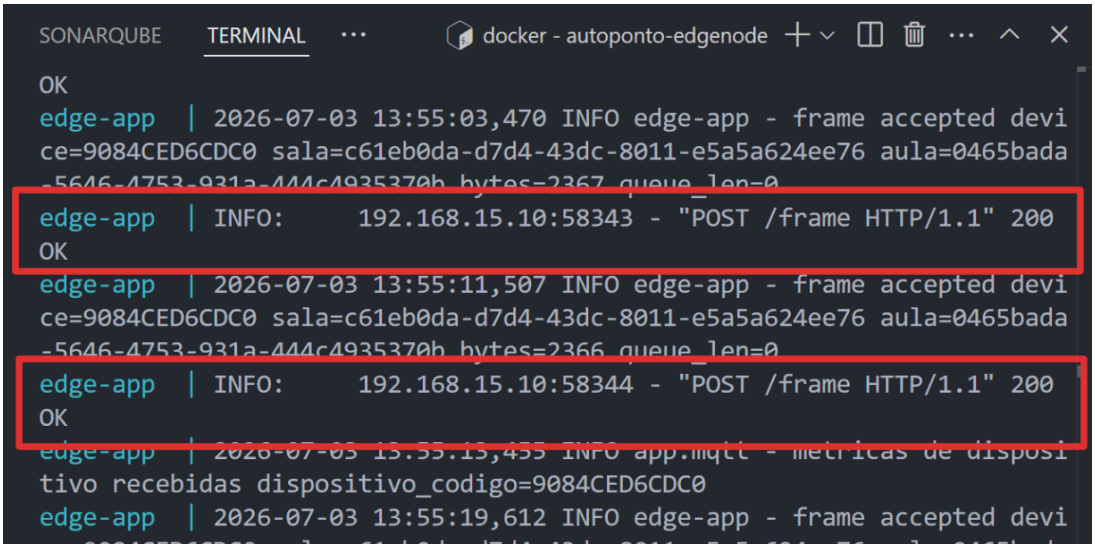
Fonte: Autoria própria (2026).

Com o *snapshot* disponível, o *EdgeNode* publicou comandos MQTT para os dispositivos associados, solicitando a renovação do contexto de aula. A partir dessa notificação assíncrona, o dispositivo embarcado consultou imediatamente a API REST local do nó de borda para verificar se havia aula vigente. Esse achado vai ao encontro do que apontam Kurose e Ross (2021, p. 72) e Yuan (2017), ao diferenciarem o uso de requisições HTTP, adequadas a interações diretas de requisição e resposta, do MQTT, mais apropriado à comunicação assíncrona baseada em publicação e assinatura.

4.1.4 Captura, reconhecimento facial e retorno local

Durante uma chamada ativa, o dispositivo capturou imagens e as enviou ao nó de borda por meio da rota de recebimento de quadros. Na Figura 26, evidencia-se esse envio. Com isso, o RF06 foi considerado cumprido, pois o dispositivo conseguiu produzir capturas e encaminhá-las ao componente responsável pelo processamento.

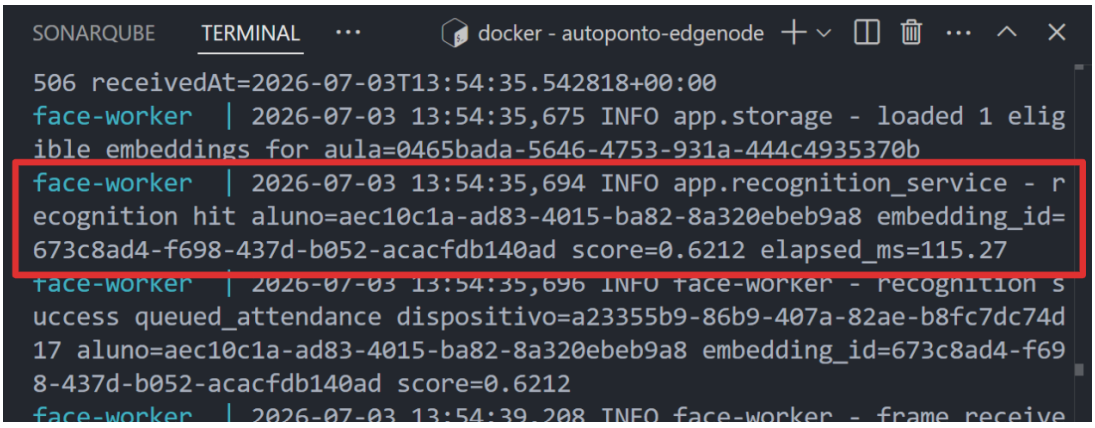
Figura 26 – Envio de imagem capturada pelo dispositivo ao *EdgeNode*



Fonte: Autoria própria (2026).

No *face-worker*, as imagens recebidas foram decodificadas, submetidas à detecção facial, convertidas em *embedding* e comparadas com os vetores dos alunos elegíveis para a aula, conforme apontam os *logs* do servidor na Figura 27. Esse comportamento confirma o RF07, uma vez que o nó de borda processou as imagens recebidas e executou a comparação facial.

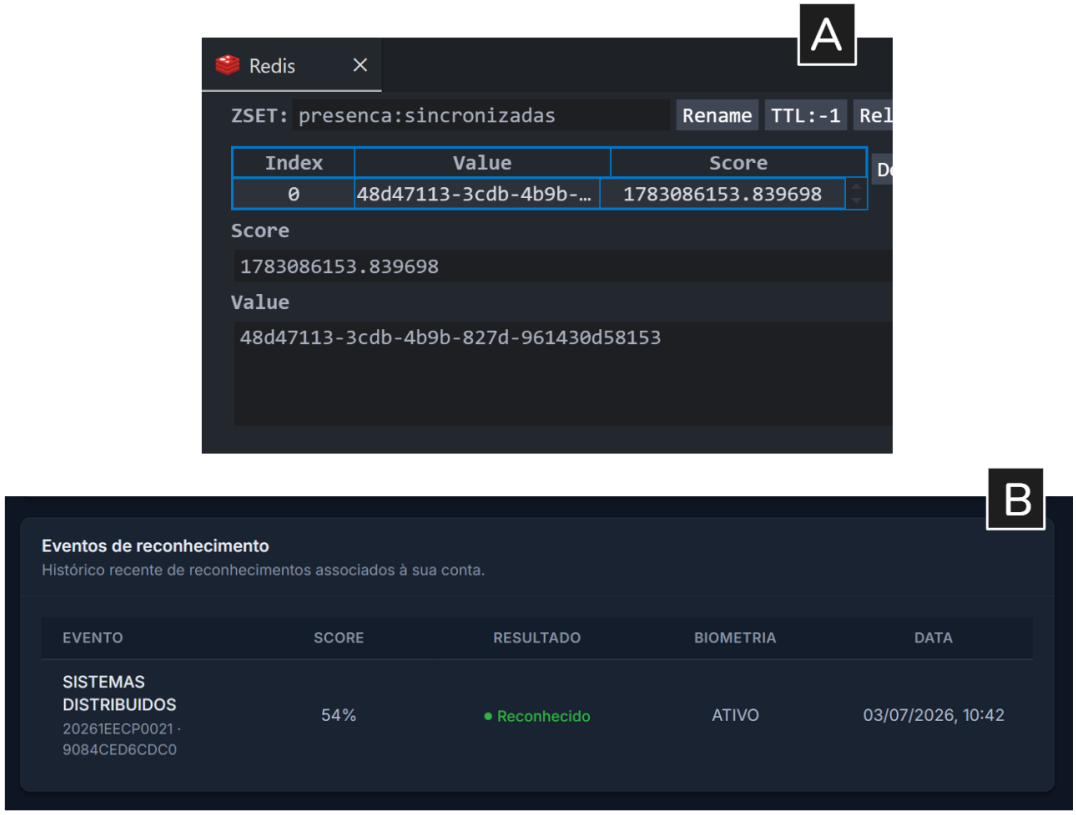
Figura 27 – Processamento facial no *face-worker* do nó de borda



Fonte: Autoria própria (2026).

Quando a similaridade atingiu o limiar configurado, o *EdgeNode* gerou, registrou e sincronizou um evento de presença associado ao aluno e à aula em andamento. Na Figura 28A, apresenta-se a presença sincronizada no *cache* local do nó de borda, enquanto, na Figura 28B, mostra-se a presença já disponível na interface *web*. Dessa forma, o RF08 foi cumprido, pois uma face reconhecida de aluno matriculado resultou em presença vinculada ao contexto acadêmico correto.

Figura 28 – A) Presença sincronizada em *cache*; B) Presença sincronizada no *website*



Fonte: Autoria própria (2026).

Após o reconhecimento positivo, o nó de borda enviou retorno ao dispositivo por MQTT, atualizando o *display* e acionando LED e *buzzer*. Na Figura 29, registra-se essa evidência. Esse comportamento confirma o RF09, pois o dispositivo recebeu e apresentou feedback local após o processamento. Os retornos visual e sonoro foram importantes para o fluxo, pois o aluno precisa perceber que sua tentativa de chamada foi concluída.

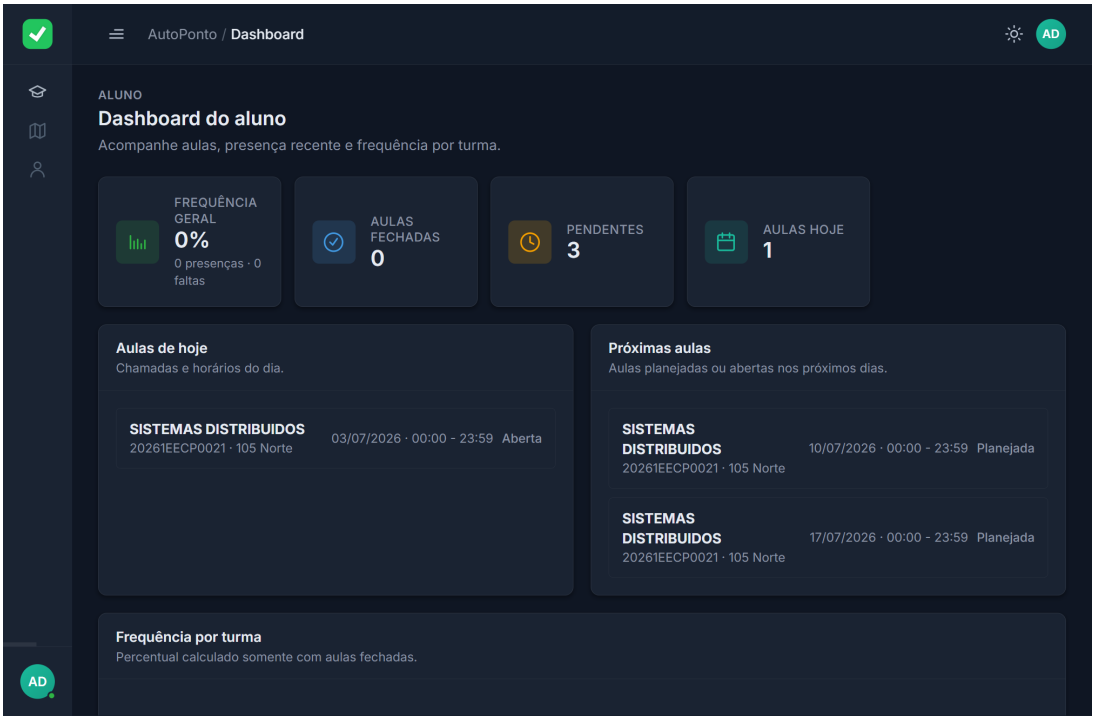
Figura 29 – Retorno positivo enviado ao dispositivo por MQTT

Fonte: Autoria própria (2026).

4.1.5 Consulta das presenças e telas da aplicação *web*

Na Figura 30, apresenta-se o *dashboard* do aluno, primeira tela da perspectiva discente. Essa tela sintetizou informações acadêmicas associadas ao usuário autenticado, como aulas, frequência e próximos compromissos.

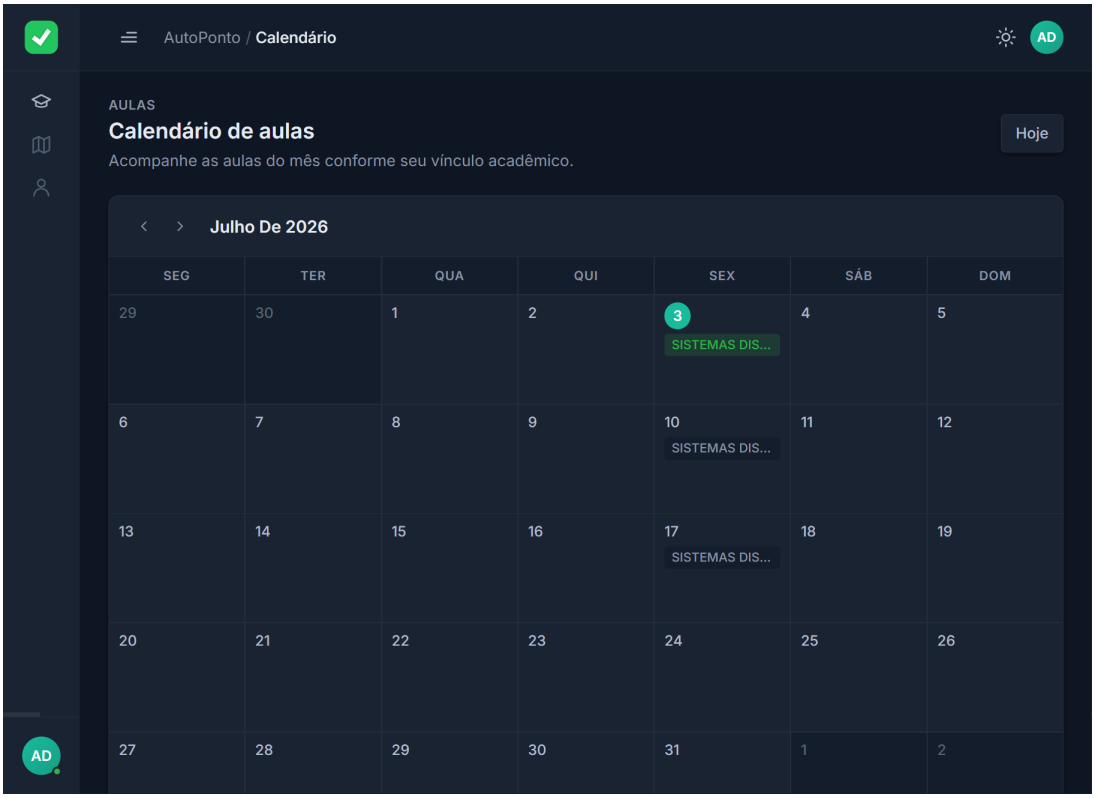
Figura 30 – *Dashboard* do aluno na aplicação *web*



Fonte: Autoria própria (2026).

Na Figura 31, ilustra-se o calendário acadêmico do aluno, que complementou a verificação ao apresentar as aulas em uma organização temporal. De modo correspondente, o professor também conseguiu visualizar, na interface *web*, as aulas que ministra, o que indicou a correta associação entre usuário, turma e componente curricular. Essa evidência reforça o cumprimento do RF01, pois depende da existência de cadastros consistentes de turmas, aulas, horários, professores e vínculos acadêmicos no servidor central.

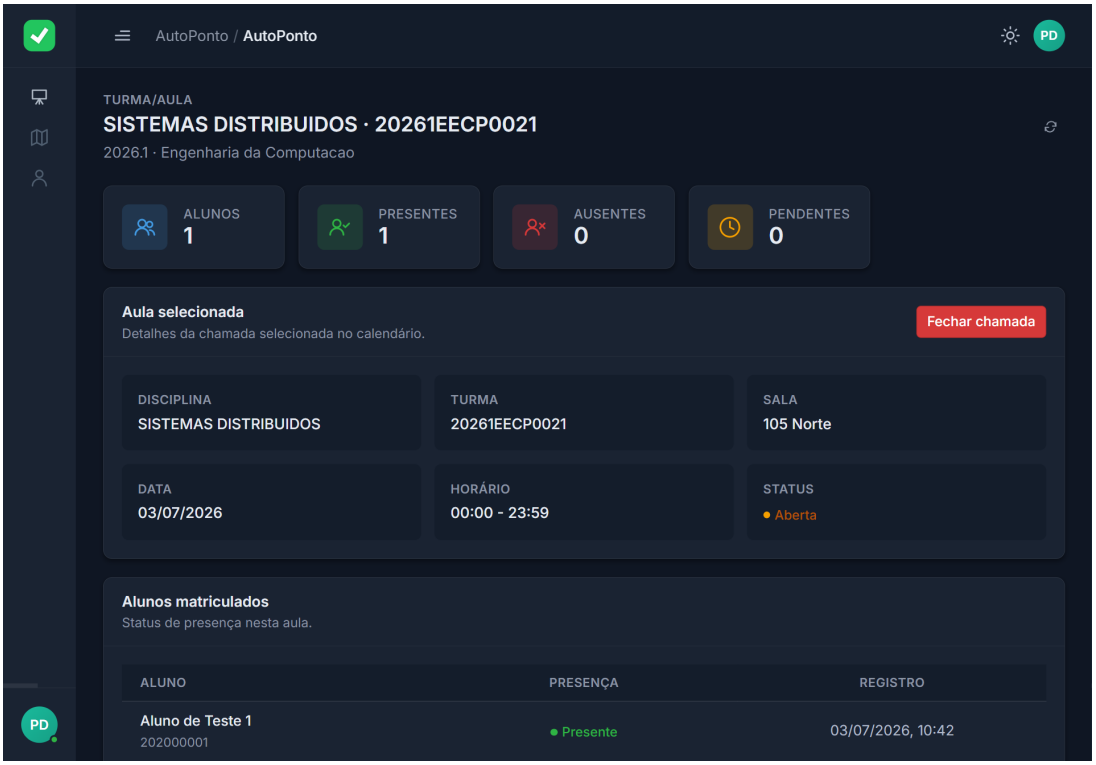
Figura 31 – Calendário acadêmico disponível ao aluno



Fonte: Autoria própria (2026).

Na Figura 32, apresenta-se a tela de detalhes da aula, que confirmou novamente o cumprimento do RF10, agora pela perspectiva do professor responsável pela turma. Nessa tela, foi possível verificar os registros de alunos presentes e ausentes, acompanhar a frequência da aula e realizar o fechamento manual da chamada quando necessário.

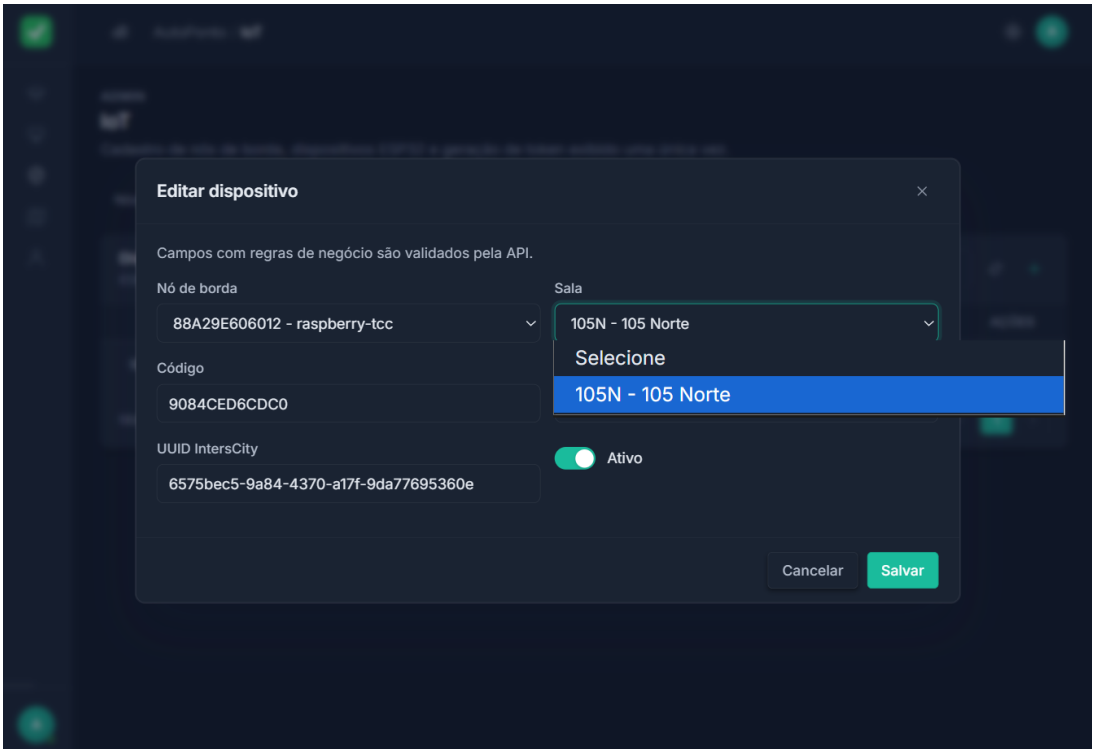
Figura 32 – Detalhes da aula e registros de frequência disponíveis ao professor



Fonte: Autoria própria (2026).

Na Figura 33, explicita-se a associação entre sala e dispositivo de chamada, que complementou a verificação do RF01 e deu suporte ao RF05. Essa relação é necessária para que o dispositivo consulte o contexto correto de aula, pois a chamada automatizada depende da vinculação entre equipamento físico, sala e horários cadastrados.

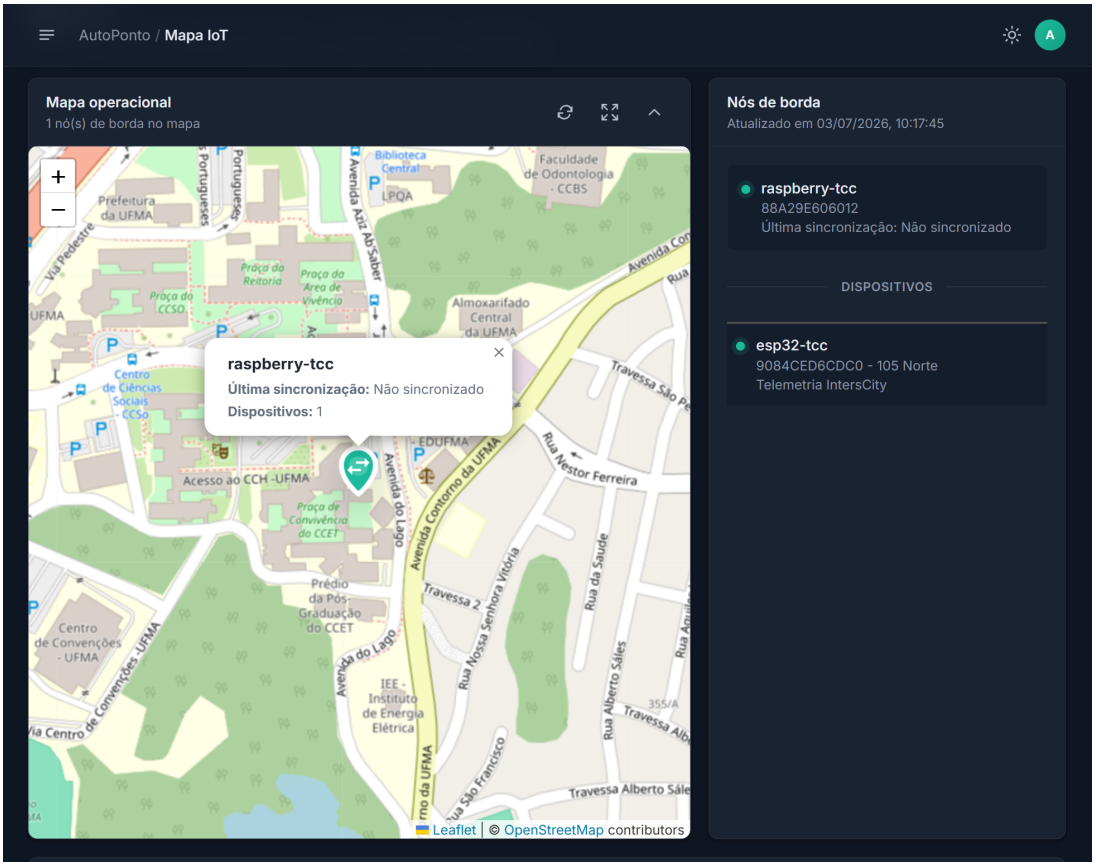
Figura 33 – Associação entre sala e dispositivo de chamada



Fonte: Autoria própria (2026).

Na Figura 34, apresenta-se a camada de observabilidade por meio do mapa IoT. Embora essa tela não participe diretamente da decisão de presença, ela permitiu acompanhar a localização lógica dos nós e dispositivos cadastrados, funcionando como apoio ao monitoramento operacional do protótipo.

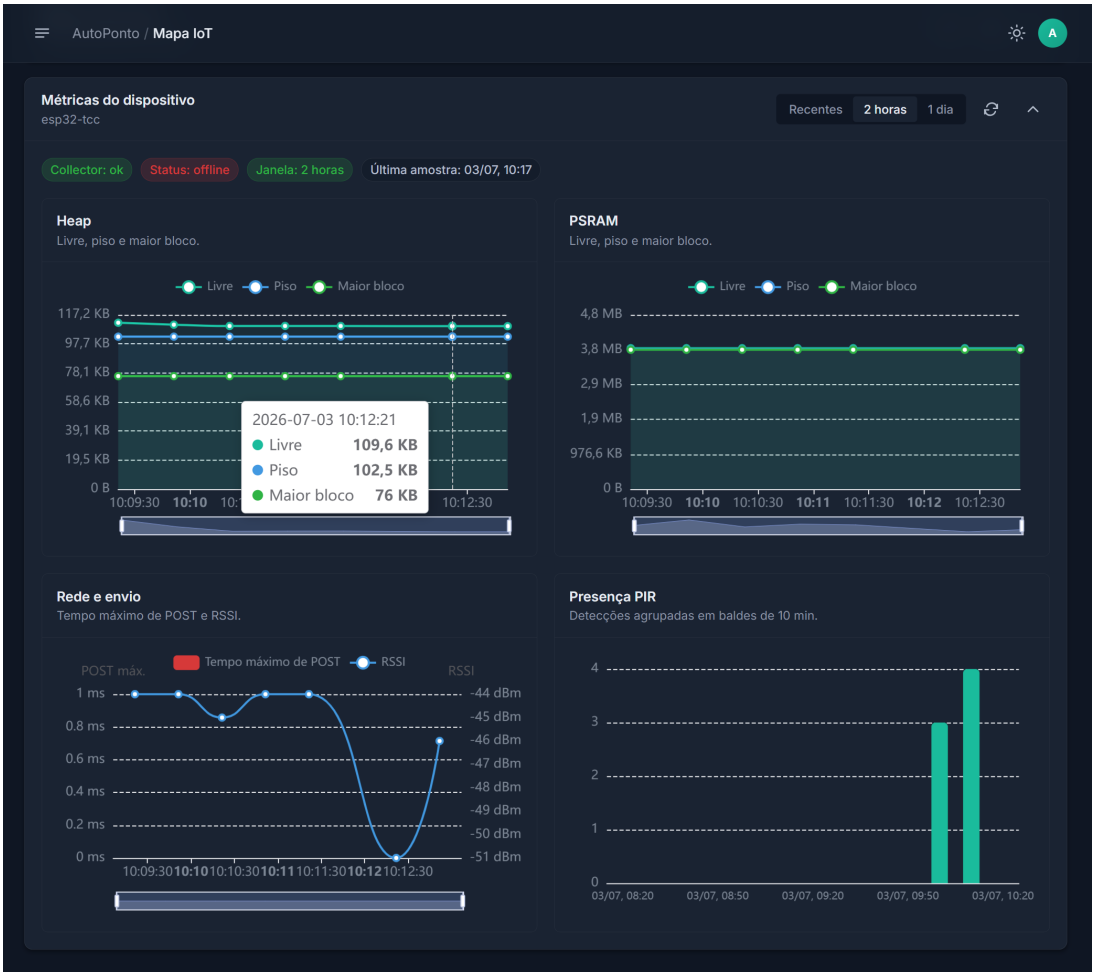
Figura 34 – Mapa IoT com nós de borda e dispositivos cadastrados



Fonte: Autoria própria (2026).

Na Figura 35, ilustra-se a consulta de telemetria dos dispositivos, que demonstrou a recuperação de dados operacionais, como estado, sinal de rede, memória e eventos do sensor PIR. Esse resultado vai ao encontro do uso da InterSCity como plataforma aberta para gerenciamento de recursos IoT e armazenamento de dados coletados, conforme discutido anteriormente no referencial teórico com base em Esposte et al. (2017, p. 2–5).

Figura 35 – Gráficos de telemetria dos dispositivos embarcados



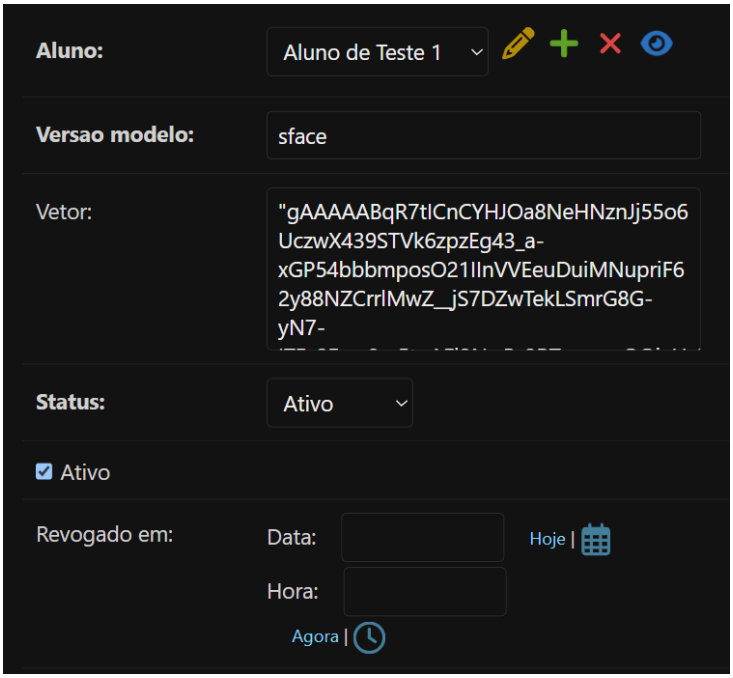
Fonte: Autoria própria (2026).

Em síntese, as evidências analisadas indicam que os requisitos funcionais RF01 a RF10 foram cumpridos no cenário controlado. O cumprimento foi observado pelo fluxo completo: cadastros acadêmicos e operacionais, cadastro e validação biométrica, sincronização para a borda, consulta de contexto, captura de imagem, reconhecimento local, geração de presença, retorno ao dispositivo e consulta posterior na aplicação *web*. Ainda assim, essa verificação permanece limitada ao ambiente de simulação, não substituindo testes em cenários reais.

4.1.6 Criptografia e revogação dos *embeddings* faciais

Consoante o mecanismo de proteção descrito por Marimuthu (2025), verificou-se que o vetor facial não ficou salvo em formato numérico diretamente legível. Em vez disso, o campo correspondente ao *embedding* foi armazenado como uma sequência criptografada. Na Figura 36, evidencia-se esse armazenamento.

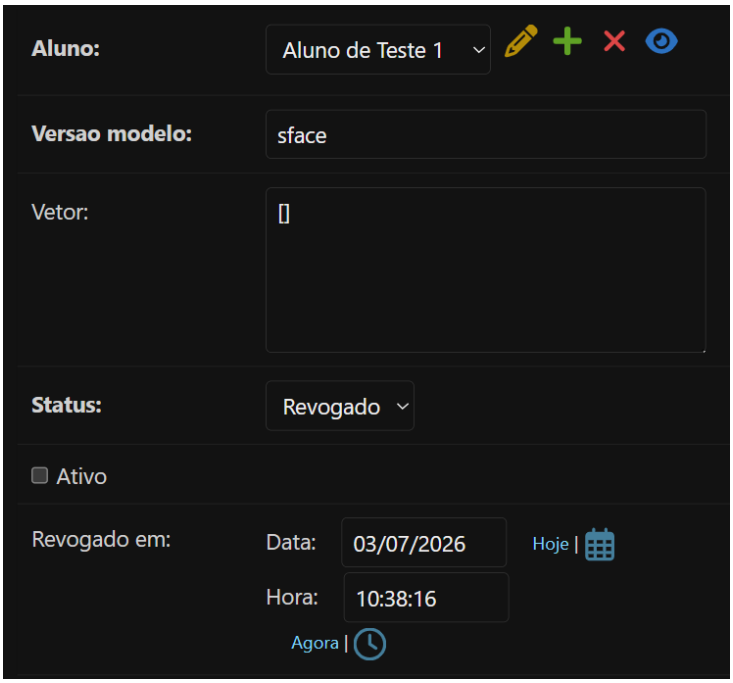
Figura 36 – *Embedding* facial armazenado em formato criptografado



Fonte: Autoria própria (2026).

Além do armazenamento criptografado, foi verificada a revogação manual da biometria facial. Ao revogar o cadastro biométrico pela aplicação, o registro deixou de ser considerado ativo e o vetor facial anteriormente armazenado foi apagado. Na Figura 37, registra-se essa revogação.

Figura 37 – Biometria facial com remoção do vetor armazenado



Fonte: Autoria própria (2026).

O resultado mostra que a proteção do dado biométrico ocorreu em duas etapas: o *embedding* permaneceu criptografado enquanto estava ativo e foi removido após a revogação manual. Essa combinação é coerente com o tratamento cauteloso exigido pela LGPD para dados biométricos vinculados a pessoa natural (BRASIL, 2018).

4.2 Métricas técnicas do dispositivo embarcado

As métricas do dispositivo embarcado foram analisadas para verificar estabilidade básica do *firmware*, margem de memória, qualidade de conexão, comportamento das tarefas FreeRTOS e ativação do sensor PIR. Essa leitura foi necessária porque a *ESP32-CAM* reúne câmera, Wi-Fi e GPIOs em uma placa compacta, mas possui limitações de alimentação, memória e disponibilidade de pinos que afetam o desenho do protótipo (SANTOS, 2020, p. 1).

Tabela 1 – Métricas do dispositivo ESP32-CAM durante a simulação

Indicador	**Valor observado**	**Análise**
Estados operacionais e PIR	203 publicações de estado; 384 eventos PIR	Os registros indicam alternância entre *offline*, *fetching*, *working* e *sleep*, além de detecções de presença pelo sensor PIR durante a simulação.
Memória interna	média de 113.059 bytes livres; mínimo de 96.540 bytes	A memória dinâmica manteve margem operacional, sem indicação de esgotamento durante as rotinas de captura, comunicação e atualização visual.
Memória PSRAM	média de 4.036.554 bytes livres	A PSRAM permaneceu próxima de 4 MB livres, com margem para captura, pré-visualização e clonagem dos quadros JPEG usados pelo dispositivo.
Rede Wi-Fi e envio HTTP	indicador de intensidade do sinal recebido (*Received Signal Strength Indicator*, RSSI) médio de -43,9 dBm; envio médio de 392,1 ms; máximo de 749 ms	O sinal Wi-Fi foi forte no ambiente de teste, e os envios efetivos de imagem ao nó de borda permaneceram abaixo de 1 s no pior caso observado.
Tarefas FreeRTOS	*display*: 45,40 ms; rede: 6,61 ms; câmera: 3,63 ms; MQTT: 1,64 ms; *loop*: 0,93 ms	A tarefa de *display* apresentou o maior custo médio, enquanto as demais rotinas permaneceram em faixas menores de execução.
Amostras das tarefas	352.443 execuções agregadas	A contagem agregada permitiu ponderar os tempos médios das tarefas e observar a recorrência das rotinas principais do *firmware*.

Fonte: Autoria própria (2026).

Na Tabela 1, indica-se que o dispositivo embarcado manteve funcionamento compatível com o fluxo previsto para a simulação. A memória interna e a PSRAM permaneceram com margem suficiente para as rotinas do *firmware*, enquanto o RSSI médio sugere que a rede local não foi o principal limitador no cenário testado. Os envios HTTP de imagem ficaram abaixo de 1 s no pior caso observado, o que foi compatível com o fluxo interativo da chamada. Entre as tarefas FreeRTOS, o maior custo médio ocorreu na atualização do *display*; ainda assim, as demais rotinas permaneceram com tempos menores.

4.3 Métricas técnicas do nó de computação de borda

O nó de borda concentrou as métricas mais diretamente relacionadas ao desempenho do reconhecimento facial. Na Tabela 2, sintetizam-se essas métricas.

Tabela 2 – Métricas do *EdgeNode* durante a simulação

Indicador	**Valor observado**	**Amostras**	**Análise**
Consulta de contexto	média de 33,64 ms	126	As requisições ao *endpoint /context* foram concluídas com sucesso e baixa latência, permitindo ao dispositivo obter rapidamente a aula vigente ou próxima aula.
Envio de quadros	média de 87,39 ms	952	O *endpoint /frame* recebeu os quadros sem falhas registradas. Esse tempo corresponde à resposta inicial ao dispositivo, antes do processamento facial completo.
Espera em fila	média de 9,89 ms	952	A fila apresentou baixo tempo de espera, sem indício de acúmulo relevante de imagens durante a simulação.
Reconhecimento facial	detecção: 20,32 ms; extração: 88,32 ms; comparação: 1,81 ms	952	A extração do *embedding* concentrou o maior custo interno. A comparação teve baixo impacto, indicando que o volume de biometrias no *cache* não gerou atraso relevante.
Processamento total	média de 48,72 ms	952	O tempo total do *face-worker* permaneceu baixo. As tentativas sem reconhecimento positivo estiveram associadas a quadros sem face válida ou sem correspondência suficiente.
Sincronização do *snapshot*	média de 6.780,87 ms	126	A sincronização foi a etapa mais custosa e instável, mas ocorre como preparação do nó de borda, não em cada tentativa individual de reconhecimento.
Publicação na InterSCity	média de 530,29 ms	924	As publicações de telemetria apresentaram funcionamento majoritariamente estável, confirmando o papel da camada de observabilidade.
Falhas de sincronização	42 eventos	42	As falhas ficaram associadas à sincronização com o servidor central, reforçando a necessidade de retentativas e registros de erro no nó de borda.

Fonte: Autoria própria (2026).

Na Tabela 2, indica-se que o *EdgeNode* respondeu com baixa latência às interações diretas do dispositivo embarcado. As consultas de contexto, o envio de quadros e a espera em fila apresentaram tempos reduzidos, sem falhas registradas nas requisições locais. No reconhecimento facial, a extração do *embedding* concentrou o maior custo interno, enquanto a comparação com os vetores elegíveis teve impacto pequeno no tempo total. A sincronização do *snapshot* foi a etapa mais custosa, mas deve ser interpretada como preparação da borda, e não como parte de cada tentativa de chamada. As publicações na InterSCity funcionaram de forma majoritariamente estável.

4.4 Métricas técnicas do servidor central e da interface *web*

No servidor central, as métricas foram analisadas considerando cadastro biométrico, proteção dos *embeddings*, geração de *snapshot*, ingestão de presenças, consultas da interface e recuperação de telemetria. Na Tabela 3, avalia-se se o servidor sustentou as operações necessárias ao fluxo de chamadas.

Tabela 3 – Métricas do servidor central AutoPonto durante a simulação

Indicador	**Valor observado**	**Amostras**	**Análise**
Cadastro biométrico	média de 688,57 ms	56	O fluxo completo ficou abaixo de 1 s em média, mas teve 2 falhas, indicando dependência da qualidade da captura e da validação facial.
Geração do *embedding*	média de 533,72 ms	56	Foi a etapa mais custosa do cadastro biométrico, concentrando a maior parte do tempo de processamento facial no servidor.
Proteção dos *embeddings*	criptografia: 4,77 ms; descriptografia: 0,27 ms	28	A criptografia e a descriptografia tiveram baixo custo, sem representar gargalo relevante no fluxo biométrico.
Cache biométrico	média de 31,72 ms	56	A listagem dos *embeddings* ativos ocorreu em tempo reduzido, permitindo preparar dados para *cache* e sincronização.
Snapshot da borda	média de 27,42 ms	42	A montagem do *snapshot* foi rápida, com envio dos dados acadêmicos e biométricos necessários ao *EdgeNode*.
Recepção de presença	média de 83,59 ms	14	A ingestão do evento reconhecido localmente ocorreu em menos de 100 ms, indicando baixo custo de persistência no servidor central.
Interface *web*	entre 15,79 ms e 73,49 ms	322	As consultas de perfil, calendário, *dashboards* e mapa apresentaram tempos compatíveis com uso interativo.
Consulta à InterSCity	média de 364,53 ms	28	As chamadas ao *Collector* tiveram maior latência que as rotinas internas, mas foram concluídas sem falhas.

Fonte: Autoria própria (2026).

Na Tabela 3, indica-se que o servidor central sustentou as operações principais do fluxo em tempos compatíveis com a simulação controlada. O cadastro biométrico concentrou o maior custo médio, principalmente na geração do *embedding*, enquanto criptografia, leitura do *cache*, montagem do *snapshot*, ingestão de presença e consultas da interface permaneceram em faixas reduzidas. A consulta à InterSCity apresentou latência maior que as rotinas internas, mas sem falhas registradas.

4.5 Síntese da análise

Os resultados da simulação controlada indicam que o AutoPonto alcançou uma integração técnica preliminar entre os principais componentes propostos. O dispositivo embarcado executou a interação local, o nó de borda recebeu e processou capturas, o servidor central manteve os dados acadêmicos e biométricos, e a interface *web* disponibilizou

consultas relacionadas às presenças. A telemetria complementou essa análise ao registrar métricas operacionais e permitir acompanhar o comportamento do dispositivo sem interferir diretamente na decisão de presença.

O principal resultado do protótipo não é a comprovação de acurácia em larga escala, mas a demonstração de um fluxo funcional distribuído. Em termos acadêmicos, isso significa que a proposta avançou do desenho metodológico para um artefato verificável, com evidências físicas, registros de execução e dados técnicos por camada. Em termos técnicos, a análise mostrou que o uso de computação de borda é adequado ao objetivo de manter o reconhecimento próximo à sala, enquanto o servidor central permanece responsável por regras acadêmicas, autenticação, persistência, relatórios e interface.

A avaliação foi realizada em ambiente controlado, com poucos voluntários, dados fictícios e condições conhecidas de rede, distância e iluminação. Não foram avaliados desempenho em turma real, percepção de alunos e professores, taxa de falsas aceitações em população ampla, resistência a tentativas de fraude, operação simultânea de vários dispositivos ou estabilidade em uso prolongado. Portanto, os resultados sustentam a viabilidade técnica inicial do protótipo, mas não autorizam afirmar prontidão para implantação institucional sem novas etapas de validação.

5 Considerações finais

A partir do desenvolvimento e da análise dos resultados, conclui-se que o objetivo geral foi atendido dentro do escopo definido para este trabalho. O AutoPonto foi implementado e verificado tecnicamente em uma simulação controlada, na qual o fluxo de chamada automatizada pôde ser executado de forma integrada. Esse fluxo partiu do cadastro acadêmico e biométrico no servidor central, seguiu para a sincronização com o nó de borda, passou pela consulta de contexto no dispositivo embarcado e resultou no reconhecimento facial local, no registro da presença e na consulta posterior das informações pela interface *web*.

Os objetivos específicos também foram contemplados. O trabalho resultou em um dispositivo embarcado documentado, com circuito eletrônico, PCB, caixa de montagem, periféricos de interação e *firmware*. O nó de computação de borda foi implementado para intermediar a comunicação com os dispositivos, manter dados locais, processar as capturas faciais e gerar eventos de presença. No servidor central, foram desenvolvidos o modelo acadêmico, o cadastro biométrico, os fluxos de sincronização, os relatórios, os *dashboards* e a interface *web*. A integração com a InterSCity acrescentou uma camada de telemetria e monitoramento, enquanto a simulação controlada permitiu verificar os principais fluxos funcionais e analisar métricas técnicas do protótipo.

Como contribuição, o trabalho apresenta uma arquitetura distribuída aplicada ao registro de frequência acadêmica, articulando dispositivo físico, processamento de borda, aplicação *web*, proteção de dados biométricos e observabilidade. A proposta demonstrou que é possível organizar a chamada automatizada em camadas, com o reconhecimento facial executado próximo ao ambiente de uso e com o servidor central concentrando as regras acadêmicas, os registros persistentes e as consultas dos usuários.

Os resultados, contudo, devem ser entendidos como uma verificação técnica preliminar. A avaliação não foi realizada em turma real, não mediu aceitação de alunos e professores, não acompanhou o uso do sistema ao longo de um período letivo e não avaliou múltiplos dispositivos em operação simultânea. Também permanecem pontos em aberto quanto à robustez biométrica, à variação de iluminação, à qualidade da rede, à escala institucional e à governança de dados sensíveis.

Dessa forma, o protótipo demonstrou viabilidade técnica inicial nas condições testadas, mas ainda não deve ser tratado como solução pronta para implantação institucional. Os resultados finais indicam que o trabalho pode ser aprofundado nas seguintes frentes:

- validar o sistema em aulas reais, com autorização institucional e termo de consentimento adequado ao tratamento de dados biométricos;
- ampliar a quantidade de participantes, turmas, salas e dispositivos, para observar o comportamento do sistema em cenários mais próximos do uso institucional;
- investigar técnicas de detecção de vivacidade, a fim de reduzir riscos de fraude no reconhecimento facial;
- aprimorar o dispositivo físico, incluindo PCB, caixa de montagem, fixação, consumo energético e estabilidade dos componentes;
- avaliar a operação simultânea de múltiplos nós de borda e dispositivos, com atenção à sincronização, filas, rede local e persistência dos eventos de presença.

Referências Bibliográficas

ADESOBA, O. C.; JOSEPH, I. M. Aa fingerprint-based attendance system for improved efficiency. **ITEGAM-JETIA**, v. 11, n. 51, p. 9–19, 2025. Disponível em: <https://itegam-jetia.org/journal/index.php/jetia/pt_BR/article/view/1305>. Acesso em: 2 jul. 2026. Citado na página 18.

Autodesk. **Autodesk Fusion**. s.d. Disponível em: <https://www.autodesk.com/in/products/fusion-360/overview>. Acesso em: 30 jun. 2026. Citado na página 40.

BAGASKARA, A. R.; SUSANTO, A. Design of an internet of things (iot) based security system using esp32 cam and passive infrared receiver (pir) motion sensor in the home environment. **International Journal of Engineering Computing Advanced Research**, v. 2, n. 1, p. 11–18, 2025. Citado 2 vezes nas páginas 19 e 21.

Bambu Lab. **Bambu Lab A1 mini: Especificações técnicas**. s.d. Disponível em: <https://bambulab.com/pt-br/a1-mini/tech-specs>. Acesso em: 30 jun. 2026. Citado na página 41.

BOUTROS, F.; HUBER, M.; SIEBKE, P.; RIEBER, T.; DAMER, N. **SFace: Privacy-friendly and Accurate Face Recognition using Synthetic Data**. 2022. Disponível em: <https://arxiv.org/abs/2206.10520>. Acesso em: 2 jul. 2026. Citado na página 26.

BRASIL. **Lei nº 13.709, de 14 de agosto de 2018: Lei Geral de Proteção de Dados Pessoais (LGPD)**. 2018. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm>. Acesso em: 29 jun. 2026. Citado 2 vezes nas páginas 26 e 72.

CARISSIMI, A. **Internet das Coisas: Middlewares e outras coisas**. 2016. Disponível em: <https://www.researchgate.net/publication/301298394_Internnet_das_Coisas_Middlewares_e_outras_coisas>. Acesso em: 2 jul. 2026. Citado na página 19.

CHOUDHARY, V.; GUHA, P.; PAU, G.; MISHRA, S. An overview of smart agriculture using internet of things (iot) and web services. **Environmental and Sustainability Indicators**, Elsevier, v. 26, p. 100607, 2025. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2665972725000285>. Acesso em: 2 jul. 2026. Citado 3 vezes nas páginas 18, 19 e 55.

DFRobot. **ESP32-CAM Development Board: SKU DFR0602**. [S.l.], s.d. Disponível em: <https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0602_Web.pdf>. Acesso em: 30 jun. 2026. Citado 2 vezes nas páginas 34 e 38.

DULčIć, L. **Face Recognition with FaceNet and MTCNN**. 2021. Disponível em: <https://medium.com/@culuma/face-recognition-with-facenet-and-mtcnn-11e77240adb6>. Acesso em: 2 jul. 2026. Citado na página 25.

ESPOSTE, A. M. D.; KON, F.; COSTA, F. M.; LAGO, N. Interscity: A scalable microservice-based open source platform for smart cities. In: **Proceedings of the 6th**

International Conference on Smart Cities and Green ICT Systems. Porto, Portugal: SCITEPRESS, 2017. p. 35–46. ISBN 978-989-758-241-7. Disponível em: <https://interscity.org/publications/>. Acesso em: 2 jul. 2026. Citado 3 vezes nas páginas 26, 27 e 69.

Fairchild Semiconductor. **SS9013: NPN Epitaxial Silicon Transistor**. [S.l.], 2002. Disponível em: <https://www.mouser.com/datasheet/2/149/SS9013-1012678.pdf>. Acesso em: 30 jun. 2026. Citado na página 35.

FILIPE, D. J. **Real-time Events Dashboard for Hospitals**. Dissertação (Master's dissertation) — University of Coimbra, 2023. Disponível em: <https://www.proquest.com/openview/2621d480bf3ed3514d8eb68e7580d2a7/1.pdf>. Acesso em: 2 jul. 2026. Citado na página 29.

GHOSH, A. **What is a PIR Sensor, How it Works**. 2024. Disponível em: <https://thecustomizewindows.com/2024/03/what-is-a-pir-sensor-works/>. Acesso em: 2 jul. 2026. Citado na página 21.

GUARNIERI, P. L. Estudo de caso comparando o mongodb e o postgresql na realização de consultas agregadas de dados. Universidade Federal de Uberlândia, 2025. Citado na página 28.

HAKIM, M.; FITRIANI, F.; MA'WE, H. Development of student attendance application with qr code scan. **Krisnadana Journal**, v. 4, n. 3, p. 182–192, 2025. Disponível em: <https://ejournal.sidyanusa.org/index.php/jkdn/article/view/874>. Acesso em: 2 jul. 2026. Citado 2 vezes nas páginas 15 e 18.

Jiangsu Huawha Electronics Co., Ltd. **TMB12A03 Product Specification**. [S.l.], 2019. Disponível em: <https://www.alldatasheet.com/datasheet-pdf/pdf/1775624/HUAWHA/TMB12A03.html>. Acesso em: 30 jun. 2026. Citado 2 vezes nas páginas 34 e 36.

KERNER, S. M. **A computação de borda pode tornar a IA mais sustentável?** 2026. Disponível em: <https://www.computerweekly.com/br/reportagen/A-computacao-de-borda-pode-tornar-a-IA-mais-sustentavel>. Acesso em: 2 jul. 2026. Citado na página 23.

KHAN, W. Z.; AHMED, E.; HAKAK, S.; YAQOOB, I.; AHMED, A. Edge computing: A survey. **Future Generation Computer Systems**, v. 97, p. 219–235, 2019. Citado 3 vezes nas páginas 15, 23 e 60.

KiCad. **About KiCad**. s.d. Disponível em: <https://www.kicad.org/about/kicad/>. Acesso em: 30 jun. 2026. Citado na página 34.

KOK, C. L.; HENG, J. B.; KOH, Y. Y.; TEO, T. H. Energy-, cost-, and resource-efficient iot hazard detection system with adaptive monitoring. **Sensors**, MDPI, v. 25, n. 6, p. 1761, 2025. Disponível em: <https://www.mdpi.com/1424-8220/25/6/1761>. Acesso em: 2 jul. 2026. Citado na página 20.

KUROSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet: uma abordagem top-down**. 8. ed. São Paulo: Pearson, 2021. ISBN 978-6555160536. Citado 2 vezes nas páginas 24 e 61.

LE, D. **Developing a User Management Dashboard with Fullstack Javascript**. Monografia (Bachelor's thesis) — Turku University of Applied Sciences, 2019. Disponível em: <https://www.theseus.fi/handle/10024/260696>. Acesso em: 2 jul. 2026. Citado na página 29.

MARIMUTHU, P. **Securely Encrypting Sensitive Data in Python with Fernet**. 2025. Disponível em: <https://parthibanmarimuthu.medium.com/securely-encrypting-sensitive-data-in-python-with-fernet-dd50638bde0f>. Acesso em: 2 jul. 2026. Citado 2 vezes nas páginas 26 e 70.

Monolithic Power Systems. **MP1584: 3A, 1.5MHz, 28V Step-Down Converter**. [S.l.], 2011. Disponível em: <https://www.monolithicpower.com/en/documentview/productdocument/index/version/2/document_type/Datasheet/lang/en/sku/MP1584EN-LF-Z/document_id/204/>. Acesso em: 30 jun. 2026. Citado na página 34.

Mouser Electronics. **Computação na Borda (Edge Computing): O Futuro do Processamento de Dados Distribuídos**. 2024. Traduzido pela Equipe Embarcados. Disponível em: <https://embarcados.com.br/computacao-na-borda-edge-computing-o-futuro-do-processamento-de-dados-distribuidos/>. Acesso em: 2 jul. 2026. Citado na página 23.

NCR. **RS-001G (KCD11) Series Rocker Switch**. [S.l.], s.d. Datasheet técnico do interruptor KCD11. Disponível em: <https://www.ncr.hk/uploads/Switches/Rocker_Switch/KCD1-101F.pdf>. Acesso em: 30 jun. 2026. Citado na página 35.

PATIL, G. **Django REST APIs Demystified: Simplifying API Development with Django**. New York: Apress, 2025. ISBN 979-8-8688-1850-9. Disponível em: <https://link.springer.com/book/10.1007/979-8-8688-1850-9>. Acesso em: 2 jul. 2026. Citado 2 vezes nas páginas 29 e 58.

PAUL, T. C.; LERTPONGRUJIKORN, P.; NGUYEN, H. D.; SALEHI, M. A. **Benchmarking Message Brokers for IoT Edge Computing: A Comprehensive Performance Study**. 2026. Disponível em: <https://arxiv.org/abs/2603.21600>. Acesso em: 2 jul. 2026. Citado na página 24.

PIOVESAN, D.; MACIEL, J. N.; ZALEWSKI, W.; LEDESMA, J. J. G.; CAVALLARI, M. R.; JUNIOR, O. H. A. Edge computing: Performance assessment in the hybrid prediction method on a low-cost raspberry pi platform. **Eng**, MDPI, v. 6, n. 10, p. 255, 2025. Disponível em: <https://www.mdpi.com/2673-4117/6/10/255>. Acesso em: 2 jul. 2026. Citado na página 23.

PIRES, J. D. D. C.; OLIVEIRA, M. C.; NETO, B. F. dos S.; RIBEIRO, M. de M.; FRAGOSO, R. S. de M. End-to-end automated student attendance recording system using surveillance camera. **Revista Brasileira de Informática na Educação**, v. 33, p. 371–393, 2025. Disponível em: <https://journals-sol.sbc.org.br/index.php/rbie/article/view/5125>. Acesso em: 2 jul. 2026. Citado 3 vezes nas páginas 15, 17 e 18.

Raspberry Pi. **Raspberry Pi 4 Model B specifications**. s.d. Disponível em: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>. Acesso em: 1 jul. 2026. Citado na página 44.

RAWAT, A. **Interface GC9A01 Round Display with STM32 via SPI & LVGL**. 2026. Disponível em: <https://controllerstech.com/how-to-interface-gc9a01-round-display-with-stm32-using-spi-lvgl-integration/>. Acesso em: 2 jul. 2026. Citado na página 22.

SANTOS, S. **ESP32-CAM AI-Thinker Pinout Guide: GPIOs Usage Explained**. 2020. <https://randomnerdtutorials.com/esp32-cam-ai-thinker-pinout/>. Acesso em: 2 jul. 2026. Citado 2 vezes nas páginas 20 e 72.

SARI, N.; HARAPAN, E.; INDRAWATI, S. W. The development of a digital-based attendance system to enhance the discipline of fifth grade students at sd negeri 62 palembang. **JMKSP (Jurnal Manajemen, Kepemimpinan, dan Supervisi Pendidikan)**, v. 10, n. 1, p. 843–855, 2025. Citado 2 vezes nas páginas 17 e 18.

SHYAM, A. A django based educational resource sharing website: Shreic. **Journal of Scientific Research, Institute of Science, Banaras Hindu University, Varanasi, India**, 2020. Disponível em: <https://www.bhu.ac.in/research_pub/jsr/Volumes/JSR_64_01_2020/34.pdf>. Acesso em: 2 jul. 2026. Citado na página 28.

SILVA, G. M. H.; SANTOS, R. A. O. Telemetria como ferramenta iot na gestão da segurança de frotas veiculares. **Editora Impacto Científico**, p. 269–285, 2024. Disponível em: <https://periodicos.newsciencepubl.com/editoraimpacto/article/view/1363>. Citado na página 27.

SILVA, L. R. d. Bancos de dados relacionais e bancos de dados chave-valor: comparação de desempenho em aplicações de troca de mensagens em tempo real. Universidade Federal do Rio Grande do Sul, 2025. Citado na página 28.

UELECTRONICS. **SR-602 Pyroelectric Human Infrared Sensor Module Instructions**. [S.l.], s.d. Disponível em: <https://uelectronics.com/wp-content/uploads/2026/01/AR4812-SR602-Mini-Sensor-PIR-Datasheet.pdf>. Acesso em: 30 jun. 2026. Citado 2 vezes nas páginas 34 e 37.

Vishay General Semiconductor. **1N5817, 1N5818, 1N5819: Schottky Barrier Plastic Rectifier**. [S.l.], 2020. Disponível em: <https://www.vishay.com/docs/88525/1n5817.pdf>. Acesso em: 30 jun. 2026. Citado na página 35.

WU, W.; PENG, H.; YU, S. Yunet: A tiny millisecond-level face detector. **Machine Intelligence Research**, v. 20, n. 5, p. 656–665, 2023. ISSN 2731-5398. Disponível em: <https://link.springer.com/article/10.1007/s11633-023-1423-y>. Citado 2 vezes nas páginas 24 e 25.

WYNE, M. F.; REEVES, J.; MONTES, F. X.; GURBACH, T. J. Business intelligence dashboard for academic program management. In: **2015 ASEE Annual Conference & Exposition**. [s.n.], 2015. p. 26–312. Disponível em: <https://www.academia.edu/22314242/Business_Intelligence_Dashboard_for_Academic_Program_Management>. Acesso em: 29 jun. 2026. Citado na página 15.

YUAN, M. **Conhecendo o MQTT**. 2017. Disponível em: <https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/>. Acesso em: 2 jul. 2026. Citado 2 vezes nas páginas 24 e 61.

Zhongjingyuan Technology Inc. **Round TFT LCD Module: 1.28 inch 240RGB x 240 dots, ZJY128R-IG01**. [S.l.], s.d. Disponível em: <https://www.elecrow.com/download/1.28_inch_round_LCD_datasheet.pdf>. Acesso em: 30 jun. 2026. Citado 2 vezes nas páginas 34 e 35.

APÊNDICE A – Lista de materiais do dispositivo embarcado

Na Tabela 4, sintetiza-se a lista de materiais utilizada na montagem do dispositivo embarcado, com impostos inclusos sobre o custo.

Tabela 4 – Lista de materiais do dispositivo embarcado (Outubro/2025)

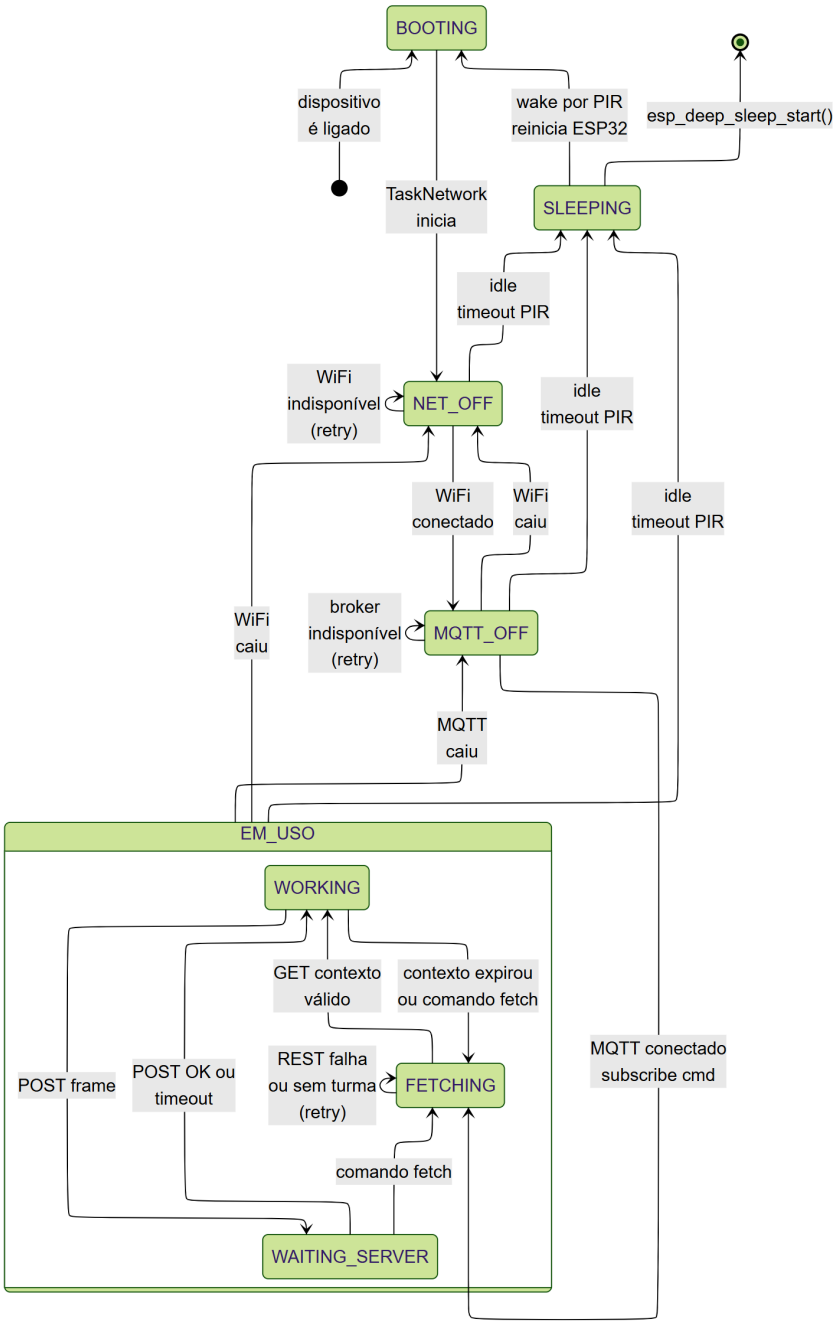
Item	**Componente**	**Especificação**	**Qtd.**	**Preço unit.**	**Subtotal**
1	Fonte de alimentação	Samsung EP-TA50BW, saída 5 Vcc e corrente máxima de 1,55 A	1	R$ 35,00	R$ 35,00
2	Conversor de tensão	MP1584, conversor *buck*, saída ajustada para 3,3 V	1	R$ 4,40	R$ 4,40
3	Microcontrolador com câmera	*ESP32-CAM* AI-Thinker	1	R$ 89,50	R$ 89,50
4	Módulo de *display*	TFT circular 1,28”, controlador GC9A01, interface SPI	1	R$ 28,00	R$ 28,00
5	Sensor de presença	HC-SR602, sensor PIR de movimento	1	R$ 7,90	R$ 7,90
6	*Buzzer* ativo	TMB12A03, alimentação nominal de 3 V	1	R$ 0,75	R$ 0,75
7	Transistor NPN	S9013, usado no chaveamento do *display* e do indicador	2	R$ 0,10	R$ 0,20
8	Diodo Schottky	1N5819, usado na proteção de alimentação e no circuito do IO4	2	R$ 0,11	R$ 0,22
9	LED verde	Indicador visual de retorno positivo	1	R$ 0,05	R$ 0,05
10	LED azul	Indicador visual de alimentação	1	R$ 0,05	R$ 0,05
11	Resistor	220 Ω	1	R$ 0,03	R$ 0,03
12	Resistor	1 kΩ	3	R$ 0,03	R$ 0,09
13	Resistor	4,7 kΩ	1	R$ 0,03	R$ 0,03
14	Resistor	10 kΩ	2	R$ 0,03	R$ 0,06
15	Resistor	20 kΩ	1	R$ 0,03	R$ 0,03
16	Capacitor cerâmico	100 nF	3	R$ 0,10	R$ 0,30
17	Capacitor eletrolítico	220 μF	1	R$ 0,30	R$ 0,30
18	Borne de entrada	KF128-MINI, 2 vias	1	R$ 1,10	R$ 1,10
19	Interruptor gôndola	KCD11, liga/desliga da alimentação	1	R$ 0,67	R$ 0,67
20	Chave tátil	12D00G ou equivalente	1	R$ 0,25	R$ 0,25
21	Placa de circuito impresso	PCB do dispositivo embarcado	5	R$ 23,60	R$ 117,98
				Total	**R$ 286,91**

Fonte: Autoria própria (2026).

APÊNDICE B – Máquina de estados do firmware

Na Figura 38, apresenta-se a máquina de estados utilizada no *firmware* do dispositivo embarcado.

Figura 38 – Fluxo de estados do firmware do ESP32



Fonte: Autoria própria (2026).