**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**
**CAMPUS APUCARANA**

**SILAS TORRES ROCHA**

# CONTROLE DE ACESSO: SISTEMA DE CATRACA COM ESP32, DOCKER E IOT

Relatório técnico apresentado para documentação do projeto de retrofit e automação de controle de acesso do Campus Apucarana.

**APUCARANA - PR**
**Fevereiro de 2026**

# Sumário

**1 Introdução** **3**
1.1 Objetivos . . . 3
1.2 Escopo do Projeto . . . 3

**2 Arquitetura do Sistema** **4**

**3 Desenvolvimento de Hardware e Mecânica** **5**
3.1 Análise Preliminar e Justificativa Térmica . . . 5

**4 Desenvolvimento de Hardware e Mecânica** **7**
4.1 Adequação Mecânica e Lógica de Atuação . . . 7
4.2 Fluxo de Acesso e Sensores . . . 8
4.3 Interface Visual e Manutenção . . . 8
4.4 Projeto Eletrônico (PCB) . . . 8

**5 Desenvolvimento de Hardware e Mecânica** **10**
5.1 Integração Eletromecânica . . . 10
5.2 Caracterização Térmica e Segurança (LM7805) . . . 12
5.3 Lista de Materiais . . . 13

**6 Desenvolvimento de Firmware** **13**
6.1 Arquitetura Modular . . . 13
6.2 Máquina de Estados (FSM) . . . 14
6.3 Protocolos de Comunicação . . . 15

**7 Desenvolvimento do Backend e Integração** **15**
7.1 Servidor de Aplicação e Gateway de Validação . . . 16
7.2 Containerização e Orquestração . . . 16

**8 Desenvolvimento do Backend** **17**
8.1 Inicialização e Agendamento . . . 17
8.2 Documentação Interativa (Swagger) . . . 18
8.3 Segurança e Controle Administrativo . . . 18

**9 Resultados e Testes** **19**
9.1 Análise de Latência e Performance . . . 19
9.2 Implantação Final e Adaptações de Campo . . . 20
9.3 Diagnóstico e Auto-Recuperação . . . 21
9.4 Cabeamento e Infraestrutura . . . 22

**10 Considerações de Segurança e Rede** **23**
10.1 Requisitos de Infraestrutura (LAN Isolada) . . . 23
10.2 Prevenção de Burlas e Fluxo . . . 23

**11 Estrutura de Arquivos do Projeto** **23**
11.1 Estrutura do Backend (Docker) . . . 23
11.2 Estrutura do Firmware (ESP32) . . . 24

**12 Conclusão** **24**
12.1 Trabalhos Futuros . . . . . 24

**13 Documentação Técnica do Hardware** **24**

# 1 Introdução

O presente relatório descreve o desenvolvimento e a implementação de um sistema de modernização (*retrofit*) para catracas de controle de acesso do campus da UTFPR.

Originalmente, o equipamento encontrava-se inoperante e desconectado da rede. Uma análise preliminar revelou que a eletrônica original havia sofrido danos irreversíveis devido a fatores ambientais severos, incluindo oxidação avançada nos conectores de rede e corrosão na placa principal, agravada pela condensação de umidade e infestação de insetos no interior da estrutura.

Diante da inviabilidade econômica e técnica de reparar o hardware legado, que operava com baterias degradadas e lógica de segurança obsoleta, optou-se pelo desenvolvimento de uma nova arquitetura de controle. O novo projeto substitui integralmente a eletrônica antiga por uma solução baseada em Internet das Coisas (IoT), utilizando componentes de mercado, código aberto e protocolos de comunicação padronizados.

## 1.1 Objetivos

O objetivo geral deste projeto é restabelecer a segurança e o controle de fluxo de pessoas no campus através da reativação da catraca física, integrando-a aos sistemas de gestão acadêmica atuais.

Os objetivos específicos incluem:

- **Desenvolvimento de Hardware:** Projetar e fabricar uma nova Placa de Circuito Impresso (PCB) dedicada, utilizando o microcontrolador ESP32-S2 para gerenciamento de sensores e atuadores.
- **Adequação Mecânica:** Converter a lógica de travamento mecânico de "Normalmente Aberta"para "Normalmente Fechada"(*Fail-Secure*), aumentando a segurança física.
- **Engenharia de Software:** Implementar um servidor de validação (Backend) containerizado em Docker, capaz de processar requisições de acesso em tempo real e integrar-se com bases de dados legadas (LDAP/Sistemas Acadêmicos).
- **Robustez Ambiental:** Desenvolver um sistema de climatização ativa (ventilação forçada) e proteção (cases e impressas em 3D) para garantir a operação contínua do equipamento em ambiente externo.

## 1.2 Escopo do Projeto

O escopo do trabalho abrange três frentes multidisciplinares:

1. **Eletrônica e Firmware:** Design da PCB, programação do microcontrolador em C++ (Framework Arduino), implementação de drivers para leitura de QR Code/ Código de Barras e comunicação de rede (Ethernet W5500).
2. **Backend e Infraestrutura:** Desenvolvimento da API de controle em Python (FastAPI), configuração do *broker* de mensagens MQTT e orquestração do ambiente de produção via Docker Compose.

3. **Mecânica e Integração:** Manutenção dos mecanismos de giro, instalação de sensores indutivos e adaptação física do leitor em totem externo para mitigação de reflexos solares.

# 2 Arquitetura do Sistema

O sistema foi concebido sobre uma arquitetura distribuída de três camadas, priorizando o desacoplamento entre o controle físico e as regras de negócio. A comunicação entre as camadas ocorre via rede Ethernet (TCP/IP), utilizando uma abordagem híbrida de protocolos HTTP (síncrono) para validação crítica e MQTT (assíncrono) para telemetria e gestão.

**Camada 1: Hardware (Borda)** Composta pela catraca física e o controlador ESP32-S2. Esta camada opera como um "Cliente Leve", responsável apenas pela leitura dos sensores, acionamento do solenoide e interface com o usuário (LEDs). Ela não armazena dados de usuários localmente.

**Camada 2: Backend (Servidor de Aplicação)** Hospedado em containers Docker, é o cérebro do sistema. Processa as solicitações de acesso, consulta os scripts de validação institucional (`consulta.py`) e aplica as regras de horário e bloqueio.

**Camada 3: Dados e Integração** Compreende os arquivos de configuração persistentes (JSON) e a interface com o sistema legado da universidade (LDAP ou Banco de Dados Acadêmico), acessada via scripts auxiliares seguros.

O sistema foi modelado para atender diferentes perfis de usuários (alunos, servidores, visitantes) e administradores. O Diagrama de Caso de Uso abaixo (Figura 1) ilustra as interações principais, desde a validação de entrada até o reinício automático e bloqueio total.

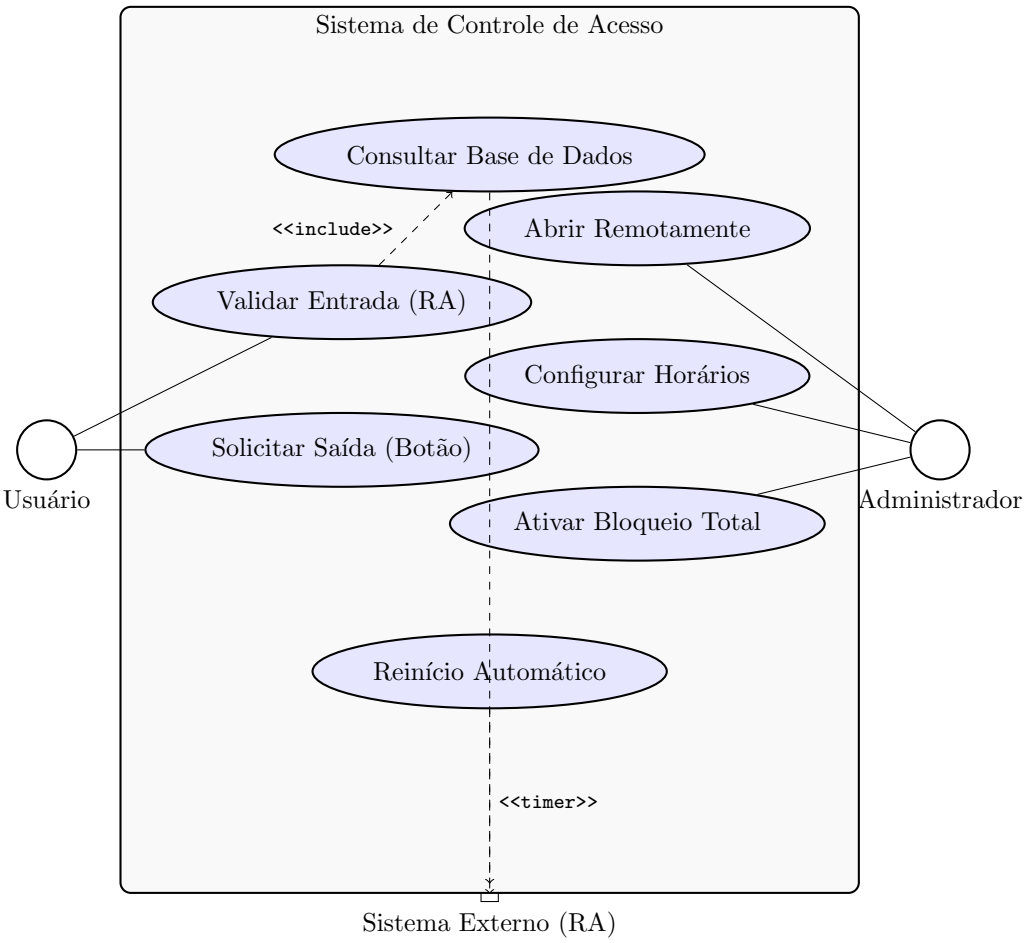


Figura 1: Diagrama de Caso de Uso: Interações entre Usuário, Administrador e Sistema.

# 3 Desenvolvimento de Hardware e Mecânica

## 3.1 Análise Preliminar e Justificativa Térmica

Antes de iniciar o projeto da nova eletrônica, foi realizado um monitoramento do ambiente interno da catraca utilizando um sensor DHT11 e um datalogger. O objetivo foi quantificar as condições agressivas que levaram à oxidação do sistema anterior.

A coleta de dados abrangeu o período entre Agosto e Outubro de 2025. Em vez de uma série temporal simples, optou-se por uma análise estatística dos extremos, conforme resumido na Tabela 1.

Tabela 1: Resumo das condições ambientais internas (Período: Ago-Out/2025).

| **Parâmetro** | **Valor Máximo/Crítico** | **Média Global** |
|---|---|---|
| Temperatura Interna | 47.1 °C | 23.0 °C |
| Umidade Relativa | 100.0 % | 68.7 % |
| Margem de Orvalho (Segurança) | $\approx 0.0\,^{\circ}$C (Crítico) | - |
| Tempo em Zona de Risco ($< 3^{\circ}$C) | 39.1 % do tempo | - |

A análise detalhada dos dados revela dois fenômenos destrutivos principais:

**1. Efeito Estufa e Acúmulo Térmico (Figura 2):** O gráfico de perfil diário (boxplot) demonstra que a estrutura metálica age como uma estufa. Observa-se uma amplitude térmica elevada, onde o calor se acumula rapidamente entre aproximadamente 10h e 14h. A "caixa"do gráfico (que representa 50% dos dados centrais) se alonga durante a tarde, indicando não apenas altas temperaturas, mas uma instabilidade térmica que fadiga as soldas e componentes eletrônicos por ciclos de expansão e contração.

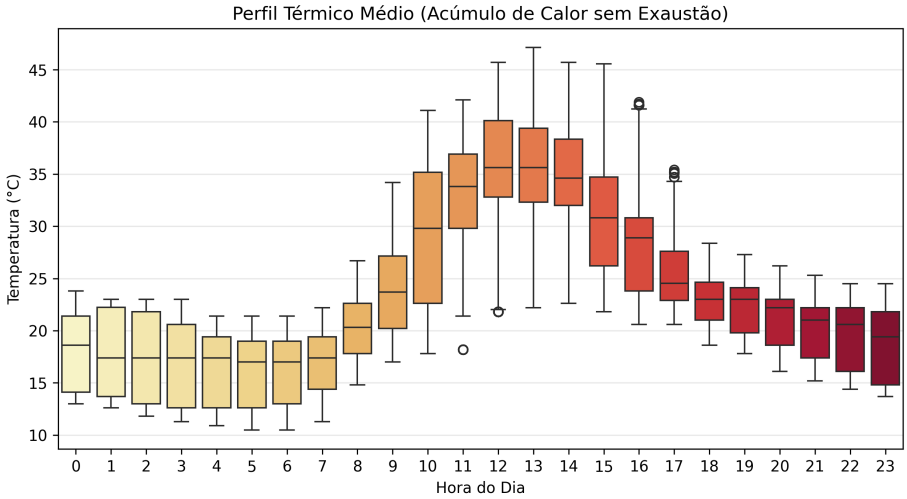


Figura 2: Perfil diário de temperatura. Note a elevação da mediana e o aumento da variabilidade (tamanho da caixa) no período vespertino.

**2. Saturação e Risco de Condensação (Figura 3):** A Figura 3 correlaciona temperatura e umidade. A área vermelha representa a zona de risco crítico. Observa-se que, nos momentos de resfriamento (geralmente à noite ou chuvas súbitas), a umidade relativa sobe drasticamente, aproximando-se de 100%.

Um dado interessante é a *Margem de Orvalho* (diferença entre a temperatura do ar e o ponto de orvalho). A análise mostrou que essa margem chega a cair para menos de 1°C. Isso significa que basta uma pequena variação de temperatura na superfície da PCB para que a água condense líquida sobre os componentes, causando a corrosão galvânica observada.

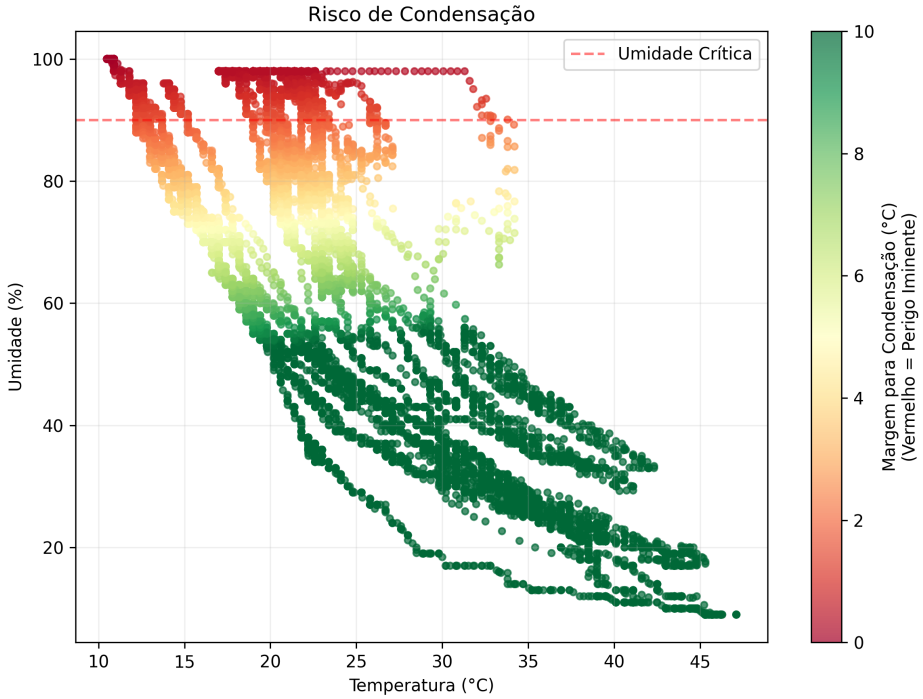


Figura 3: Mapeamento de risco. Pontos na região vermelha indicam margem de segurança $< 3°C$, onde a condensação é iminente, a linha é apenas um marcador visual para indicar que valores de umidade acima de 90% são considerados críticos.

Os dados mostraram que a ventilação natural é insuficiente para evitar tanto o acúmulo térmico quanto a saturação de umidade. A solução implementada utiliza ventilação forçada contínua: os coolers são acionados automaticamente junto com o sistema e permanecem ligados durante toda a operação. Essa estratégia garante a renovação constante do ar interno, reduzindo a probabilidade de que este atinja o ponto de orvalho e condense sobre os componentes eletrônicos.

Além da ventilação, foi adotada a proteção adicional de envernizamento da placa eletrônica. Essa camada isolante reduz a vulnerabilidade à oxidação galvânica e garante maior confiabilidade mesmo em cenários de condensação eventual.

# 4 Desenvolvimento de Hardware e Mecânica

## 4.1 Adequação Mecânica e Lógica de Atuação

O projeto mecânico original operava sob a lógica "Normalmente Aberta"(Fail-Safe), onde o solenoide permanecia destravado e só era acionado para bloquear o giro não autorizado. Este sistema dependia de uma bateria de backup que, devido à exposição térmica e ao tempo de inatividade, encontrava-se degradada, comprometendo a segurança física do acesso em caso de falha de energia.

Visando maior segurança patrimonial, o sistema foi convertido para a lógica "Normalmente Fechada"(Fail-Secure). Foi realizada uma adaptação mecânica no conjunto do solenoide, adicionando-se uma mola de compressão, parafuso e arruela para manter o pino de trava na posição de bloqueio por padrão.

Desta forma:

- **Em repouso:** A catraca permanece travada mecanicamente.
- **Na validação:** O solenoide é energizado apenas durante o intervalo de liberação (com *timeout* configurável via software), permitindo o giro.
- **Em falha de energia:** O sistema permanece travado, impedindo acessos não controlados.

Para garantir a operação contínua em cenários críticos, recomenda-se a instalação futura de um UPS (Nobreak) dedicado tanto para a catraca quanto para o servidor de validação, caso seja necessário opera sem energia prolongado.

## 4.2 Fluxo de Acesso e Sensores

O sistema foi reprojetado considerando a realidade do fluxo misto do campus, onde usuários (alunos, servidores e visitantes) podem ingressar por portões laterais ou estacionamento e utilizar a catraca apenas para saída.

- **Entrada Controlada:** A entrada exige validação rigorosa via crachá. O leitor original de código de barras foi substituído por um módulo multiformato (GM81S), capaz de ler códigos 1D e 2D (QR Code), permitindo expansões futuras para outros modelos.
- **Saída Livre:** Para atender ao fluxo de saída (incluindo visitantes sem crachá), foi eliminado o leitor de saída. O acionamento é feito via botão físico, simplificando a operação e reduzindo custos, garantindo que qualquer pessoa possa sair do campus sem barreiras.

## 4.3 Interface Visual e Manutenção

A interface visual original, limitada a um pequeno indicador bicolor de 20cm, foi substituída por um sistema de fitas de LED RGB, instaladas próximos as arestas do corpo da catraca. Isso permite feedbacks visuais intuitivos (ex: Branco para aguardando, Verde para liberado, Vermelho para negado/erro) visíveis a longa distância.

Além das modificações eletrônicas, foi realizada a manutenção mecânica completa:

1. Remoção de improvisos anteriores (ripas de madeira utilizadas para travamento manual da catraca inoperante).
2. Limpeza profunda e remoção de ninhos de insetos da estrutura interna.
3. Lubrificação do eixo central e do sistema de rolamentos do torniquete, restaurando a suavidade do giro.

## 4.4 Projeto Eletrônico (PCB)

O circuito impresso foi desenvolvido em face simples, tendo como núcleo de processamento o microcontrolador ESP32-S2. A placa gerencia três níveis de tensão distintos (12V, 5V e 3.3V) para acomodar a diversidade de periféricos e a lógica de controle.

**Gerenciamento de Energia:** Inicialmente, protótipos utilizaram reguladores lineares (AMS1117), porém estes apresentaram dissipação térmica excessiva. A versão final adota um conversor DC-DC *Buck* (LM2596), que garante eficiência na conversão da fonte externa de 12V para os níveis lógicos. Para a alimentação geral, optou-se pela sustentabilidade do projeto através do reaproveitamento de uma fonte de 12V/8A, recuperada do sistema legado.

**Condicionamento de Sinais e Atuadores:**

- **Interface de Potência (ULN2003):** Um *array* de transistores Darlington é utilizado para o acionamento das fitas de LED RGB e do solenoide de travamento. Como o solenoide consome aproximadamente 550mA em operação e cada canal do ULN2003 suporta 500mA [3], foi implementada uma configuração de canais paralelos para dividir a corrente de carga, garantindo a longevidade do componente.
- **Imunidade a Ruído (Nível Lógico):** Os sensores indutivos de giro e o botão de saída operam em 12V. Esta escolha projetual visa mitigar a interferência eletromagnética (EMI) induzida pela estrutura metálica da catraca. O sinal é convertido para 3.3V (nível lógico do ESP32) através de transistores BC547 configurados como chaves saturadas e pull-up / pull-down.
- **Sensor de Presença (PIR):** Um sensor piroelétrico foi adicionado para detecção de aproximação. Embora não componha a lógica crítica de segurança, ele atua na experiência do usuário (UX), alterando o padrão de iluminação dos LEDs para indicar que o sistema está pronto para receber o crachá.

O sensor PIR atua na experiência do usuário (UX), alterando o padrão de iluminação dos LEDs de Cores Institucionais (Figura 4) para um Padrão de Alerta (Figura 5) assim que detecta a aproximação de um usuário, indicando intuitivamente para apresentar o crachá.

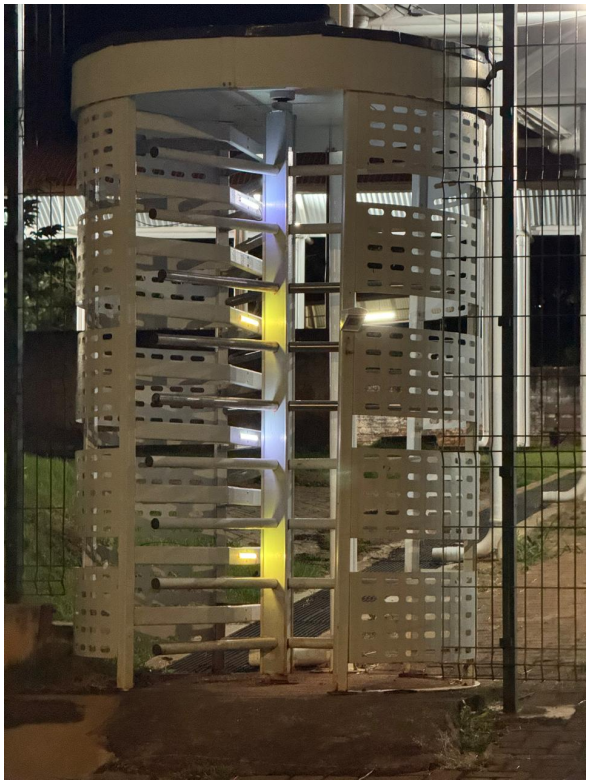

Figura 4: Modo Standby: Iluminação nas cores institucionais (Branco/Amarelo) enquanto ocioso.



Figura 5: Modo Detecção: Feedback visual (Vermelho) acionado pelo PIR ao detectar aproximação.

# 5 Desenvolvimento de Hardware e Mecânica

## 5.1 Integração Eletromecânica

A eletrônica foi acomodada na case original, os suportes foi impressa em 3D (Figura 6), projetada para proteger a placa principal contra contato direto e poeira, mantendo aberturas estratégicas para o fluxo de ar.

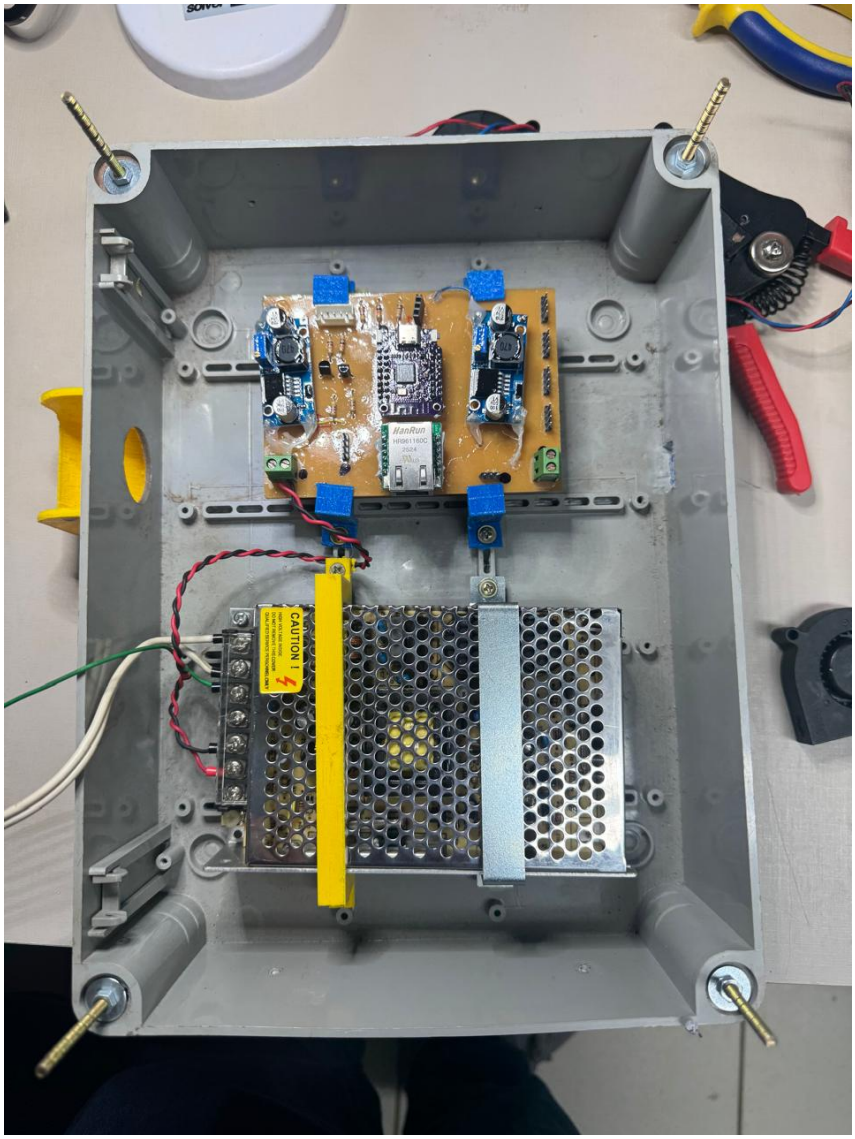


Figura 6: Case de proteção da eletrônica (Vista interna sem os coolers instalados).

**Vedação e Acabamento:** Para garantir redundância, utilizou-se **Selante de Poliuretano (PU)** em todas as junções da case e na fixação do acrílico. A escolha pelo PU em detrimento do silicone convencional deve-se à sua maior resistência mecânica, flexibilidade sob variação térmica e resistência aos raios UV. Externamente, o cabeamento foi organizado utilizando canaletas, garantindo proteção física aos condutores e melhor estética visual ao conjunto.

**Proteção do Sistema de Ventilação:** Para proteger as entradas e saídas de ar (admissão e exaustão) contra intempéries, foi desenvolvido um "escudo"impresso em 3D. Este design cria um caminho que permite o fluxo de ar dos coolers, mas impede que a água da chuva ou respingos escorram para o interior da case eletrônica.

**Otimização de Recursos (PWM e Cores):** O microcontrolador ESP32-S2 possui uma limitação de hardware de 8 canais de PWM (LEDC). Como o projeto utiliza 4 segmentos de fita LED RGB independentes, seriam necessários 12 canais de controle (4

segmentos × 3 cores). Para contornar essa limitação sem onerar a CPU com PWM via software (o que poderia causar latência no sistema de segurança), adotou-se uma estratégia híbrida:

- **Canais Azul e Verde:** Controle via PWM (Hardware), permitindo ajuste de brilho e mistura de cores.
- **Canal Vermelho:** Controle Digital (ON/OFF).

Essa decisão de engenharia prioriza a estabilidade do processamento central. A combinação permite a reprodução das cores mais críticas (Branco = R:1, G:255, B:255; Amarelo = R:1, G:255, B:0), mantendo o feedback visual rico e o sistema seguro.

Para a exaustão térmica, foram especificados coolers do tipo *Blower* (Série B5015 [11]). A escolha por este modelo deve-se à sua capacidade de direcionar o fluxo de ar com maior pressão estática, ideal para o espaço confinado da case. A vida útil estimada destes componentes é de aproximadamente 30.000 horas de operação contínua.

## 5.2 Caracterização Térmica e Segurança (LM7805)

Devido à indisponibilidade momentânea de reguladores chaveados (Buck) durante a montagem, optou-se pelo uso do regulador linear LM7805. Embora este componente ofereça excelente imunidade a ruídos eletromagnéticos, ele dissipa o excedente de tensão na forma de calor.

Para validar a segurança, foi realizado um teste de "pior caso"(sem ventilação forçada). O gráfico na Figura 7 demonstra que o componente atinge o equilíbrio térmico próximo a 90°C.

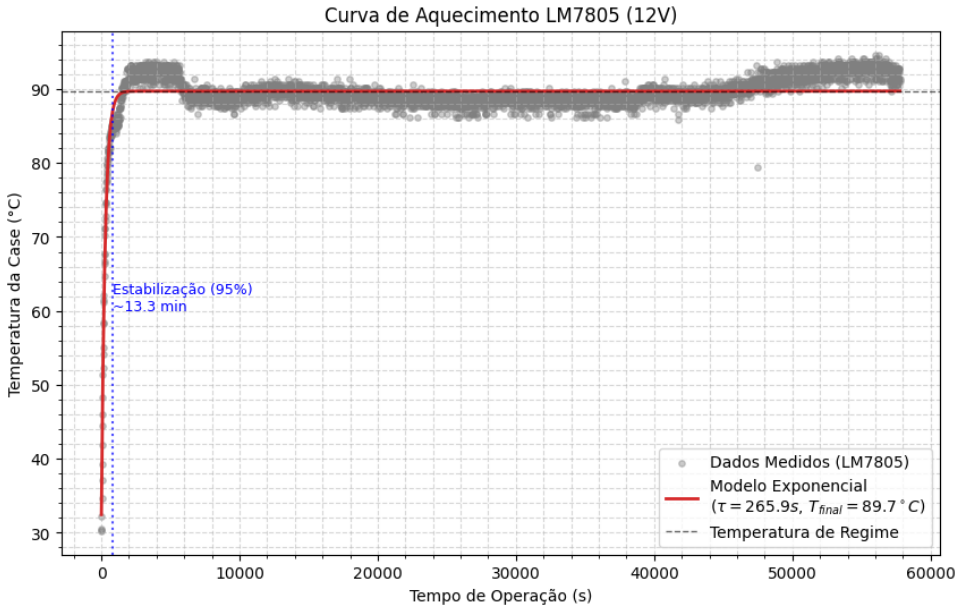


Figura 7: Curva de aquecimento do regulador LM7805 (Cenário sem ventilação).

**Notas de Manutenção :**

- **Segurança:** O dissipador opera em temperatura elevada. Recomenda-se aguardar 15 minutos após o desligamento antes de realizar manutenções internas.

## 5.3 Lista de Materiais

A Tabela 2 detalha os componentes utilizados na montagem da versão final.

Tabela 2: Lista de Materiais

| **Componente / Material** | **Qtd.** | **Observação** |
|---|---|---|
| Microcontrolador ESP32-S2 | 1 | Controle Central |
| Módulo Ethernet W5500 | 1 | Conectividade Cabeada |
| Leitor GM81S (Código de Barras/QR) | 1 | Leitura Multiformato |
| Regulador LM2596 (Buck Converter) | 2 | 12V $\rightarrow$ 5V/3.3V |
| Regulador LM7805 | 2 | Dedicado aos Coolers |
| Driver ULN2003 | 2 | Controle de Potência (LEDs/Solenoide) |
| Transistor BC547 | 3 | Conversão de Nível Lógico |
| Capacitores Eletrolíticos | 4 | Filtragem de Fonte |
| Fonte 12V 2A | 1 | Alimentação do sistema |
| Solenoide de Travamento 12V | 1 | Atuador Físico |
| Sensores Indutivos | 2 | Detecção de Giro |
| Sensor PIR (HC-SR501)* | 1 | *Opcional (UX) |
| Botão de Saída | 1 | Mecânico |
| Fita LED RGB 5050 (Segmentos) | 4 | Feedback Visual |
| Coolers Blower 5015 5V | 2 | Ventilação Forçada |
| Conectores Borne (KRE) | 2 | Alimentação e Solenoide |
| Resistores Diversos | 7 | Pull-up/Divisores |
| Parafusos M3/M4 | 12 | Fixação |
| Porcas e Arruelas | 8 | Fixação |
| Chapa de Acrílico 3mm | 1 | Janela do Leitor |
| Selante PU | N/A | Vedação |
| Filamento (Impressão 3D) | N/A | Case e Suportes |

# 6 Desenvolvimento de Firmware

O firmware encontra-se em sua terceira iteração. O desenvolvimento iniciou-se com FreeRTOS nativo (ESP-IDF), mas migrou-se para o *Framework Arduino* (C++) para agilizar a integração, devido a problemas com o stack de rede W5500.

O código opera em arquitetura *non-blocking* (sem bloqueios). O ciclo principal (*loop*) utiliza agendamento temporal baseado na função *millis()*, permitindo que tarefas de manutenção, como verificação de rede e *heartbeat* MQTT, ocorram sem interromper a leitura dos sensores de quadratura, garantindo latência mínima na liberação da catraca. Para robustez em produção, o sistema conta com *Watchdog Timer* (WDT) via software e um reinício preventivo programado a cada 24 horas.

## 6.1 Arquitetura Modular

O código foi estruturado visando manutenibilidade e desacoplamento de hardware. As responsabilidades foram isoladas em diretórios e arquivos de cabeçalho (.h):

- **config/:** Centraliza definições de hardware (`MapeamentoPinos.h`) e credenciais (`Configuracoes.h`), facilitando a portabilidade.

- **modulos/sensores/:** Abstrai o controle de baixo nível. O arquivo `ControleCatraca.h` gerencia as interrupções dos sensores e o acionamento do solenoide.
- **modulos/comunicacao/:** Separa a lógica de rede.
  - `ClienteHTTP.h`: Dedicado exclusivamente à validação síncrona de RAs.
  - `ClienteMQTT.h`: Gerencia telemetria assíncrona e comandos remotos.
- **modulos/leitor_GM81S/:** Implementa o driver UART para o leitor de QR Code, incluindo limpeza de *buffer* e tratamento de strings.
- **AutoDiagnóstico:** Um módulo de supervisão verifica continuamente o estado dos periféricos (Rede, MQTT, Sensores) e reporta falhas visualmente via LEDs e telemétricamente ao servidor.

Visando a alta disponibilidade, o firmware foi projetado com múltiplos níveis de tolerância a falhas (fallbacks). O reinício preventivo diário é, primariamente, orquestrado de forma remota pelo servidor durante a madrugada (adotando um ruído temporal). Caso haja falha de comunicação e o servidor não envie o comando, o próprio firmware atua como fallback, forçando um auto-reinício caso o tempo de atividade (uptime) exceda 24 horas somado a uma tolerância.

É imperativo ressaltar que a catraca é plenamente capaz de operar de forma autônoma e contínua sem este recurso. A reinicialização atua estritamente como uma redundância preventiva para forçar a desfragmentação da memória. Esta medida mitiga o desgaste natural de alocação dinâmica causado pelo uso da biblioteca de Strings.

Além disso, o sistema de supervisão por Watchdog por software não opera com um tempo estático global. Ele adota valores dinâmicos e parametrizáveis de acordo com a rotina em execução: existem timeouts estritos para solicitações de rede e detecção de giro mecânico, e tolerâncias maiores (na escala de horas) para análises do módulo de Autodiagnóstico. Essa diferenciação evita que anomalias em sensores de UX (não-críticos como o PIR) gerem falsos positivos.

## 6.2 Máquina de Estados (FSM)

Para garantir a consistência lógica do acesso, foi implementada uma Máquina de Estados Finitos usando interrupções (IRAM_ATTR) para ler os sensores indutivos. Esta abordagem impede condições de corrida, como liberação dupla. Os estados principais (`EstadoCatraca_t`) são:

1. **AGUARDANDO:** Estado de repouso. Solenoide travado (Fail-Secure).
2. **LIBERADA:** Após validação do RA. O solenoide destrava e define o sentido autorizado. Um *timer* de *timeout* é iniciado.
3. **EM_GIRO:** Detectado via interrupção nos sensores. O firmware valida a sequência de pulsos (A e B) para confirmar o sentido correto.
4. **COMPLETO:** O giro atinge o número de passos definidos (`STEPS_POR_PASSAGEM`). O acesso é computado e a catraca trava imediatamente.
5. **ERRO/BLOQUEIO:** Acionado em caso de tentativa de giro reverso ou "carona". O sistema emite alerta visual vermelho.

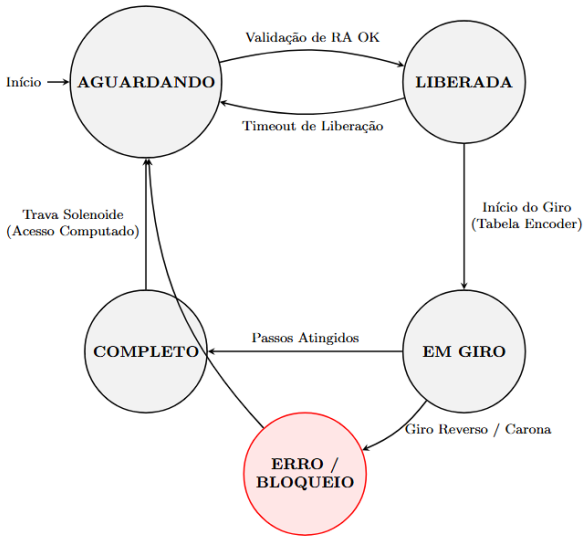


Figura 8: Diagrama de Transição de Estados (FSM)

## 6.3 Protocolos de Comunicação

O projeto utiliza uma topologia de rede local (LAN) isolada via cabo (Ethernet W5500) para segurança, adotando uma abordagem híbrida de protocolos:

**HTTP (Síncrono - Crítico):** Utilizado para autenticação. Ao ler um crachá, o ESP32 realiza um `POST` para o endpoint `/api/v1/validar_ra`. A escolha pelo HTTP deve-se à necessidade de uma resposta atômica e bloqueante ("Acesso Permitido/Negado") antes da atuação física.

**MQTT (Assíncrono - Gestão):** Utilizado para monitoramento e controle. O dispositivo atua como cliente Pub/Sub:

- **Publicação:** Envia eventos para o tópico `controle_acesso/evento` (ex: "GIRO_COMPLETO", "MOVIMENTO_PIR").
- **Assinatura:** Escuta o tópico `controle_acesso/comando`, permitindo que operadores enviem comandos remotos (ex: "LIBERAR_AGORA", "REINICIAR") sem interação física com o equipamento.

O código suporta reconexão automática ao broker, o que garante resiliência em falhas de rede.

# 7 Desenvolvimento do Backend e Integração

A arquitetura do sistema adota o paradigma de Cliente Leve (*Thin Client*). Nesta topologia, o firmware do ESP32 é intencionalmente isento de regras de negócio complexas, atuando apenas como uma interface de hardware que coleta o ID (RA) e aciona atuadores baseados em comandos explícitos. Toda a inteligência, validação e registro de dados residem no servidor (Backend).

O desenvolvimento da aplicação priorizou a performance e a facilidade de integração com sistemas institucionais, utilizando a linguagem Python 3.11.

## 7.1 Servidor de Aplicação e Gateway de Validação

Para o servidor web, optou-se pelo framework **FastAPI**. Diferente de frameworks síncronos tradicionais (como Django ou Flask), o FastAPI utiliza o padrão ASGI (*Asynchronous Server Gateway Interface*). Isso permite que a aplicação gerencie requisições HTTP de validação e conexões MQTT de telemetria simultaneamente, sem bloqueio de I/O, garantindo alta disponibilidade mesmo em horários de pico.

A arquitetura de dados não mantém um banco de dados proprietário de usuários. Em vez disso, o sistema atua como um *Gateway de Validação*. A rota principal, `/api/v1/validar_ra`, implementa um padrão de projeto *Adapter* que executa os seguintes passos:

1. **Verificação de Regras de Acesso:** Consulta um arquivo de configuração persistente (`config_acesso.json`) para verificar se o sistema está em modo de bloqueio total (*lockdown*) ou se o horário atual permite o fluxo.

2. **Abstração de Consulta Externa:** A validação do RA não é feita diretamente pelo código principal. O backend delega essa tarefa a um script externo isolado, denominado `consulta.py`. O servidor executa este script como um subprocesso do sistema operacional e captura o código de saída (*stdout*).

3. **Segurança e Desacoplamento:** O script `consulta.py` é tratado como uma "caixa preta"pela aplicação principal. É nesta camada que residem as credenciais sensíveis (senhas, acessso LDAP) e a lógica proprietária da instituição. Essa separação garante que o código-fonte da catraca possa ser auditado ou alterado sem expor dados críticos de segurança da universidade.

**Tratamento de Erros e Interface (UX):** Embora seja uma API de backend, foi implementado um tratamento de exceções personalizado (`custom_404_handler`). Caso um usuário ou administrador tente acessar uma rota inexistente via navegador, o sistema renderiza uma página HTML amigável em vez do erro padrão JSON. Isso melhora a usabilidade e facilita o diagnóstico visual de conectividade.

**Agendamento de Tarefas:** Um agendador interno (*background task*), baseado na biblioteca `asyncio`, monitora o horário e realiza reinícios preventivos do serviço. Foi adicionado um algoritmo de *Jitter* (ruído aleatório de tempo) para evitar que, em um cenário com múltiplas catracas, todas reiniciem exatamente no mesmo segundo, sobrecarregando a rede.

## 7.2 Containerização e Orquestração

Para mitigar a disparidade entre o ambiente de desenvolvimento (Windows) e o servidor de produção (Linux/Debian), todo o ecossistema foi containerizado utilizando **Docker**.

A orquestração é definida no arquivo `docker-compose.yml`, que gerencia dois serviços operando em uma rede interna isolada (*bridge*):

- **Serviço Backend:** Baseado na imagem `python:3.11-slim`, hospeda a API FastAPI e as regras de negócio.

- **Broker MQTT:** Utiliza a imagem oficial do Eclipse Mosquitto. Atua como intermediário na comunicação assíncrona, recebendo a telemetria do ESP32 e distribuindo comandos.

A integridade dos dados é garantida através de **Volumes do Docker**. Logs do Mosquitto e arquivos de configuração de acesso são mapeados para fora dos contêineres, assegurando que as definições de horário e o histórico de funcionamento persistam mesmo após atualizações ou recriação dos serviços.

# 8 Desenvolvimento do Backend

## 8.1 Inicialização e Agendamento

O ambiente Docker garante a reprodutibilidade da execução. As Figuras 9 e 10 demonstram o processo de *bootstrap* do servidor e o agendamento da tarefa de reinício automático, que inclui um "ruído"(jitter) aleatório para evitar picos de rede (ex: agendado para 01:42 AM).

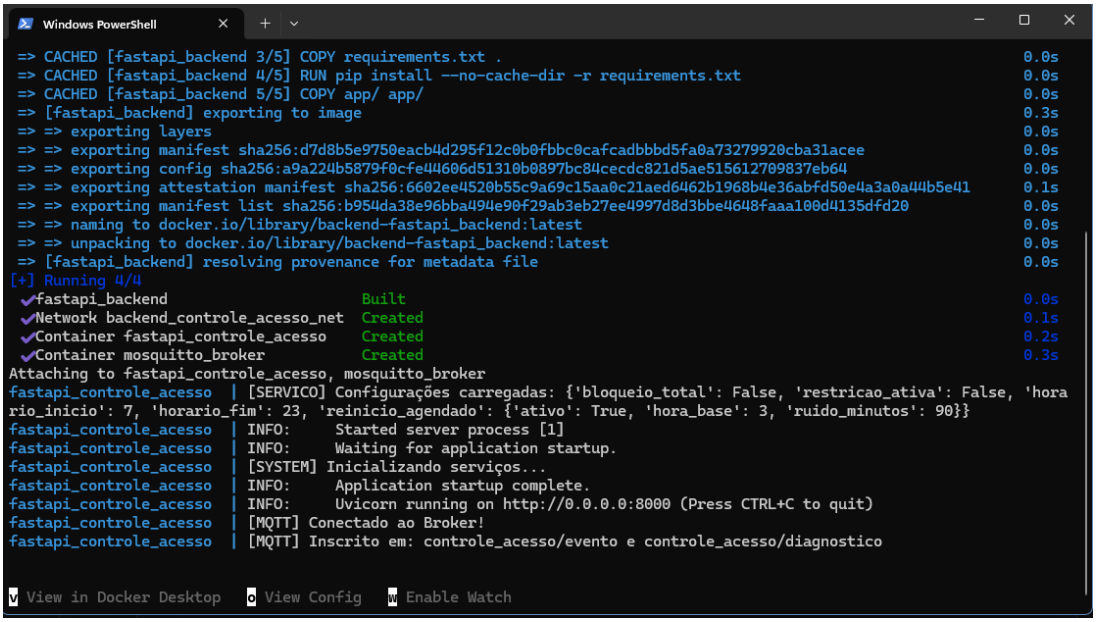


Figura 9: Terminal: Inicialização dos containers Docker.

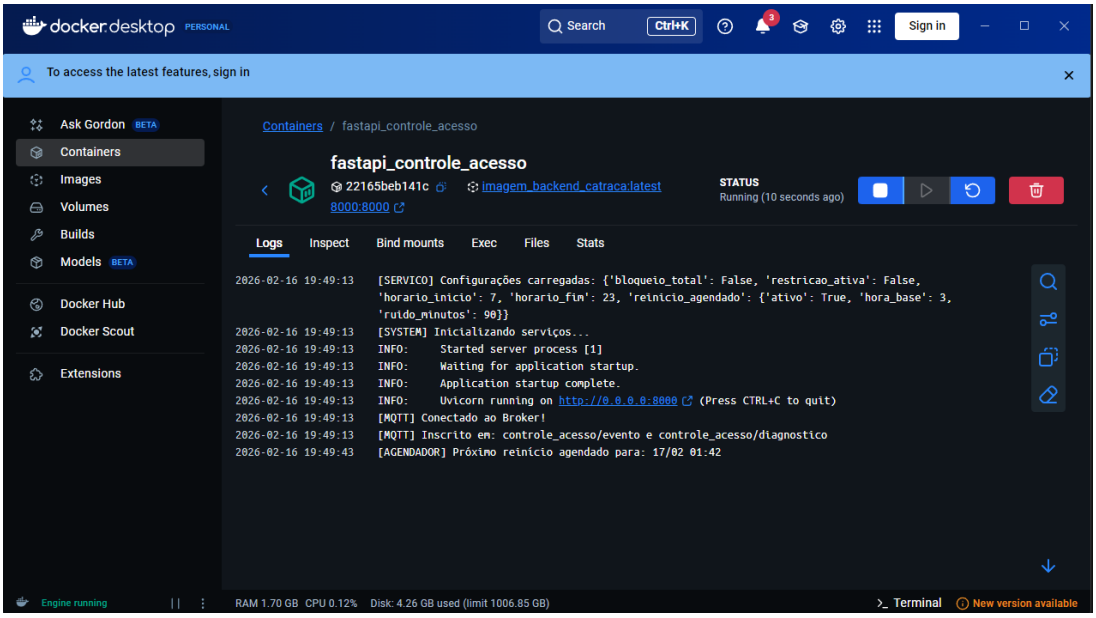


Figura 10: Log da Aplicação: Detalhe do agendamento de manutenção com horário aleatório.

## 8.2 Documentação Interativa (Swagger)

Para facilitar o trabalho da equipe de TI e futuras integrações, o backend expõe automaticamente uma documentação interativa via Swagger UI (Figura 11), acessível na rota `/docs`, ou redirecionado pela paquina de erro 404 . Isso permite testar requisições (cURL) sem escrever código.(Figura 12)

## 8.3 Segurança e Controle Administrativo

Para operações críticas (abertura remota, alteração de horários, configuração de lockdown), a API exige uma **Senha Administrativa** no corpo da requisição. Isso impede que acessos não autorizados à rede interna resultem em controle físico da catraca.

Abaixo, um exemplo de chamada via cURL para o comando de abertura, conforme documentado na rota `/docs`:

```
curl -X 'POST' \
  'https://localhost:8000/api/v1/comando/abrir' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "senha": "SenhaExemplo"
}'
```

Listing 1: Exemplo de comando administrativo seguro via cURL (Senha exemplo)




Figura 11: Interface de Documentação Automática (FastAPI/Swagger).

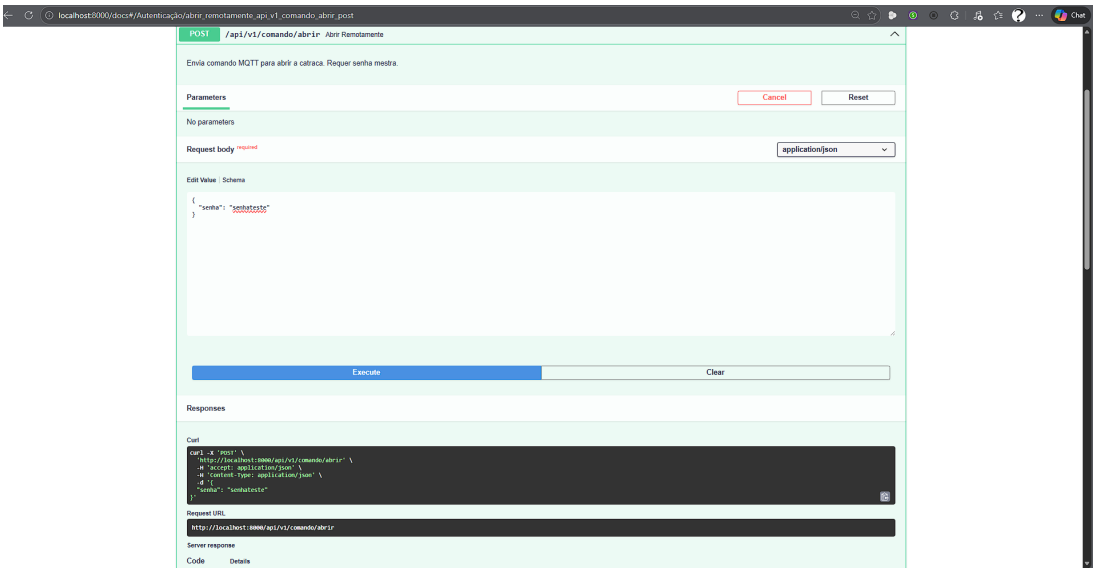


Figura 12: O comando cURL pode ser copiado para testes ou até mesmo utilizado em outras implementações.

# 9 Resultados e Testes

## 9.1 Análise de Latência e Performance

- **Cenário Ideal (Local):** O tempo de processamento interno do backend foi de aproximadamente **59ms**.
- **Cenário Simulado (Com Delay):** Observou-se nos logs um tempo total de operação próximo a **1,059s**. A análise do código indicou que a biblioteca de comunicação utilizada no teste possuía um parâmetro de *timeout* padrão de 1 segundo para manter a porta aberta. Este comportamento (overhead) foi corrigido via configuração na versão final, não impactando a lógica central de validação.

- **Resolução de Nomes (DNS):** Embora o microcontrolador ESP32 na catraca possa operar via DHCP (não exigindo IP fixo), é altamente recomendado que o servidor backend seja acessado através de um serviço de DNS interno . Isso facilita a substituição do servidor físico sem necessidade de recompilar e regravar o *firmware* de todas as catracas.

- **Resiliência:** Testes de desconexão forçada (remoção do cabo de rede) comprovaram que o sistema de verificação e a rotina de reconexão automática do Firmware funcionam conforme o esperado, restabelecendo a comunicação MQTT/HTTP sem intervenção humana assim que o link físico é recuperado.

- **Tratamento de Exceções (UX):** A API implementa tratamento customizado para rotas inexistentes (Erro 404). Acessos indevidos via navegador renderizam uma página HTML amigável que redireciona o administrador para a documentação (Swagger), em vez de exibir um erro JSON bruto.

## 9.2 Implantação Final e Adaptações de Campo

A instalação no ambiente real exigiu adaptações de engenharia para lidar com as condições climáticas e de iluminação do campus.

**Evolução do Alojamento do Leitor (Do 3D ao Totem):** O projeto original contemplava um invólucro (*case*) impresso em 3D utilizando **PETG** (Polietileno Tereftalato de Glicol), material escolhido pela sua resistência mecânica e estabilidade contra radiação UV. Para a vedação, utilizou-se **Selante de Poliuretano (PU)**, garantindo flexibilidade e durabilidade superior ao silicone.

Um diferencial técnico deste design era o **Controle de Umidade Passivo**: O projeto integrava um orifício de respiro coberto por uma membrana de **PTFE (Teflon)**. Devido às propriedades hidrofóbicas e inertes do PTFE, esta membrana permitia a saída de vapor (evitando condensação interna), mas bloqueava a entrada de água líquida.

Além disso, a case original foi projetada com uma **inclinação de 33°**, de modo a evitar a incidência direta de chuvas sobre a câmera do leitor e reduzir a exposição solar no sensor. O design incluía ainda um **sistema de pingadeira**, que direcionava a água da chuva para longe da lente, garantindo operação confiável mesmo em dias chuvosos. Contudo, os testes revelaram que em horários de pico solar, a inclinação causava reflexos não no leitor em si, mas em determinadas películas protetoras de smartphones, dificultando a leitura.

**Adaptação para o Totem:** Apesar da robustez mecânica da case 3D, optou-se pela adaptação de um **Totem de Álcool em Gel** desativado. A estrutura metálica vertical permitiu criar uma câmara escura para o leitor, eliminando reflexos e mantendo a proteção contra intempéries, além de reaproveitar um ativo da universidade.

O fluxo completo de validação, desde a apresentação do crachá até o acionamento do solenoide, pode ser visualizado no Diagrama de Sequência da Figura 13.

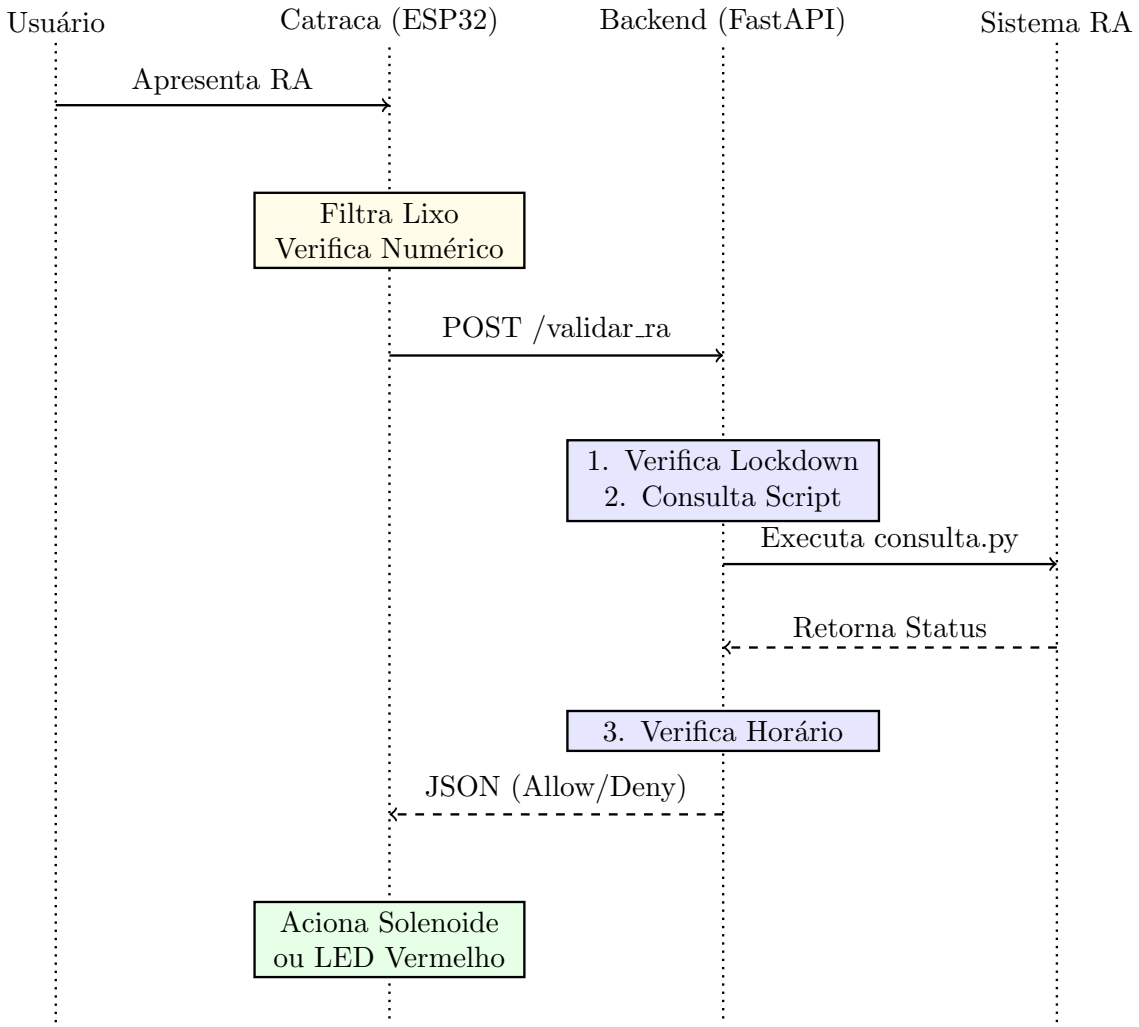


Figura 13: Diagrama de Sequência: Fluxo de validação de acesso.

## 9.3 Diagnóstico e Auto-Recuperação

O sistema de autodiagnóstico provou-se eficaz em testes. A Figura 14 apresenta um log real onde o sistema detectou uma falha no "Indutor 1"(travado) e, após o reparo físico, validou o retorno à operação normal automaticamente.

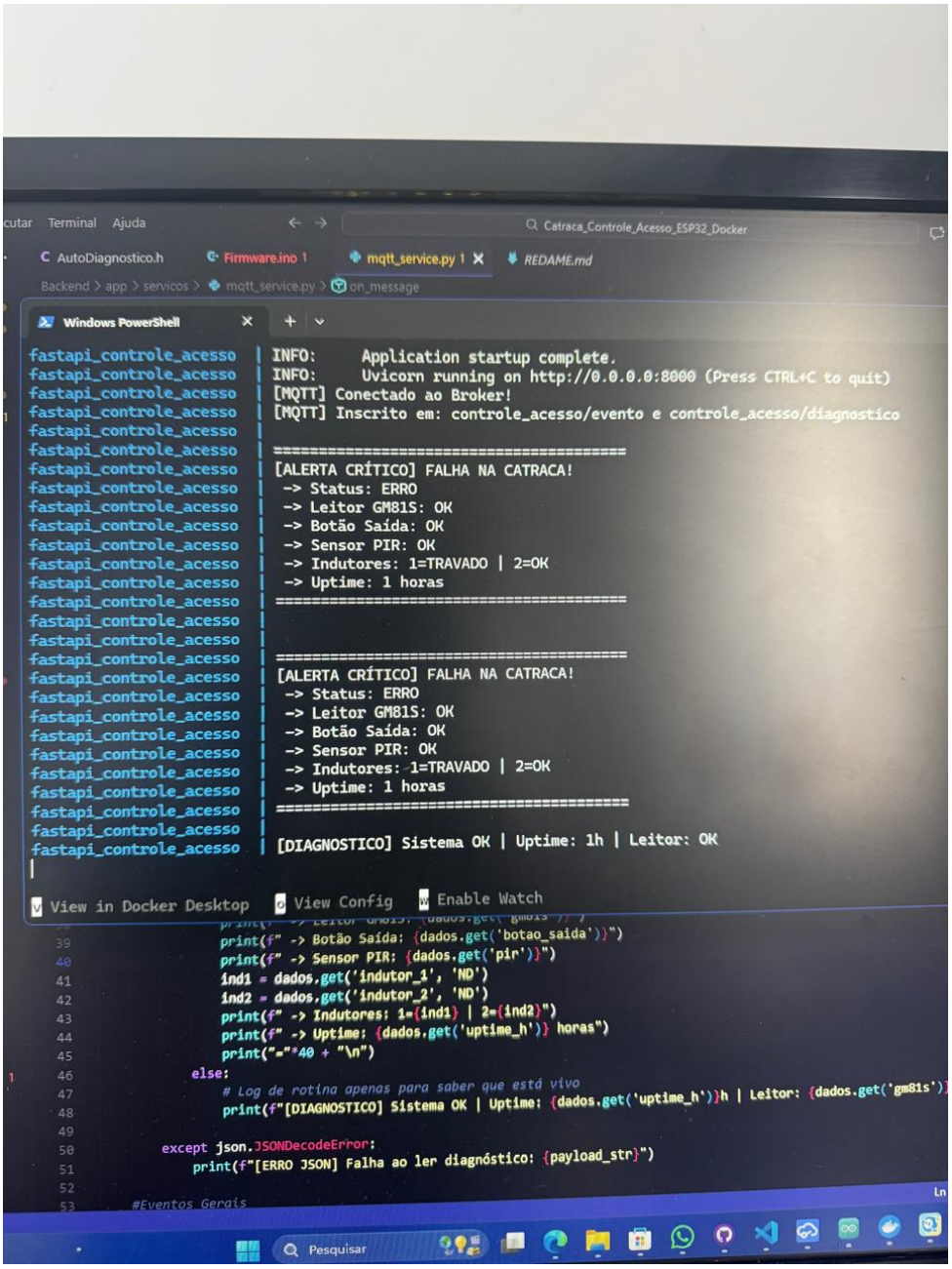


Figura 14: Log de Diagnóstico: Detecção de falha no sensor e recuperação do sistema.

Procedimentos detalhados de configuração de rede, diagramas elétricos completos e guias de reparo estarão disponíveis no **Manual de Manutenção** e no **Manual do Usuário** que acompanham a entrega técnica do equipamento.

## 9.4 Cabeamento e Infraestrutura

Para garantir a integridade do sinal em um ambiente ruidoso, toda a comunicação entre a catraca e o switch de rede utiliza cabo de par trançado blindado (STP), aterrado em uma das extremidades para drenar interferências eletromagnéticas.

# 10 Considerações de Segurança e Rede

Embora o sistema utilize protocolos padrão TCP/IP (HTTP/MQTT), a segurança da implementação pede segregação de rede.

## 10.1 Requisitos de Infraestrutura (LAN Isolada)

A catraca e o servidor Docker devem operar em uma **VLAN (Virtual LAN) isolada**, sem acesso direto da rede pública ou da rede de alunos (Wi-Fi aberto).

- **Risco de Interceptação:** Como o protocolo MQTT padrão trafega em texto claro (se não configurado com TLS), um atacante na mesma rede poderia realizar um ataque de *Sniffing* (interceptação de pacotes) para ler os RAs ou injetar comandos maliciosos.
- **Comandos Não Autenticados:** Na configuração atual, visando baixa latência, o broker MQTT pode aceitar conexões anônimas internas. Se exposto à rede pública, qualquer usuário poderia publicar no tópico `controle_acesso/comando` e destravar a catraca remotamente.

Portanto, a segurança física da porta do switch e a configuração de firewall da VLAN são requisitos de instalação, conforme detalhado no Manual de Manutenção anexo a este projeto.

## 10.2 Prevenção de Burlas e Fluxo

O firmware implementa lógica estrita de sentido único. Caso o usuário tente girar a catraca no sentido contrário ao autorizado (ou forçar a passagem), o solenoide trava imediatamente e o sistema entra em modo de **ERRO/VIOLAÇÃO**. Para resetar este estado e liberar nova tentativa, é necessário acionar o botão de saída/reset. Esta medida visa desencorajar tentativas de "carona"ou uso indevido.

# 11 Estrutura de Arquivos do Projeto

A organização final dos diretórios reflete a separação de responsabilidades entre Hardware e Software.

## 11.1 Estrutura do Backend (Docker)

```
/Backend
        docker-compose.yml       # Orquestração dos containers
        app/
            main.py              # Ponto de entrada (FastAPI)
            recursos/
                config_acesso.json # Regras de horário/Lockdown
                consulta.py        # Script "Caixa Preta" (LDAP/Banco)
            rotas/               # Endpoints da API
            servicos/            # Lógica de validação e MQTT
            templates/           # Páginas de erro (404.html)
        mosquitto/               # Configurações do Broker MQTT
```

## 11.2 Estrutura do Firmware (ESP32)

```
/Firmware
        Firmware.ino            # Loop principal (Non-blocking)
        config/                 # Mapeamento de pinos e IPs
        modulos/
            sensores/           # Controle de hardware (Solenoide/Indutivos)
            comunicacao/        # Clientes HTTP e MQTT
            leitor_GM81S/       # Driver do leitor
            utils/              # Autodiagnóstico e Watchdog
            Controle_Leds/      # Feedback visual (UX)
```

# 12 Conclusão

O projeto atingiu seu objetivo principal de modernizar o controle de acesso do campus, substituindo uma eletrônica obsoleta e danificada por uma solução modular, baseada em padrões abertos (IoT, Docker, Python).

A análise preliminar de dados ambientais foi crucial para a engenharia da nova solução, justificando o investimento em ventilação forçada e a mudança da lógica de travamento para *Fail-Secure*. Embora a integração com o sistema legado tenha introduzido uma latência de 4 segundos, este tempo é aceitável considerando a segurança da validação em tempo real.

Como melhoria futura, sugere-se a implementação de cache local no Backend (Redis) para RAs frequentes, visando reduzir o tempo de validação.

## 12.1 Trabalhos Futuros

Para aprimoramentos subsequentes (Versão 2.0), sugere-se:

- Instalação de sistemas No-Break (UPS) dedicados na ponta (catraca) para manter o acesso e a comunicação durante falhas severas de energia.
- Expansão das regras do *Backend* para permitir criação de perfis temporários de acesso (QR Codes descartáveis para visitantes).
- Para futuras revisões de hardware, recomenda-se a substituição do regulador linear por um conversor DC-DC Step-Down (ex: LM2596), que possui eficiência superior a 90%, eliminando a geração excessiva de calor.
- **Interface de Saída Sem Desgaste:** Observou-se que o botão mecânico de saída pode ser um ponto de falha devido ao alto ciclo de uso. Sugere-se a substituição futura por um sensor capacitivo (ex: módulo TTP223), que elimina o desgaste mecânico e permite a ativação por aproximação, aumentando a durabilidade do conjunto. O botão utilizado possui vida útil típica de 300.000 até 1.000.000 de ciclos, conforme especificado em seu datasheet [12].

# 13 Documentação Técnica do Hardware

Todos os artefatos de engenharia desenvolvidos para este projeto, incluindo modelos 3D (STL), arquivos de fabricação da PCB (Gerber), código-fonte e bibliotecas customizadas, estão disponíveis publicamente no repositório oficial: `https://github.com/SilasTorresRocha/Catraca_Controle_Acesso_ESP32_Docker`

As Figuras 15 e 16 apresentam, respectivamente, o diagrama esquemático final e o layout da placa de circuito impresso desenvolvida. Vale notar que o sistema de ventilação possui regulação de tensão dedicada (não mostrada no esquemático principal) para isolar o ruído dos motores da lógica digital.

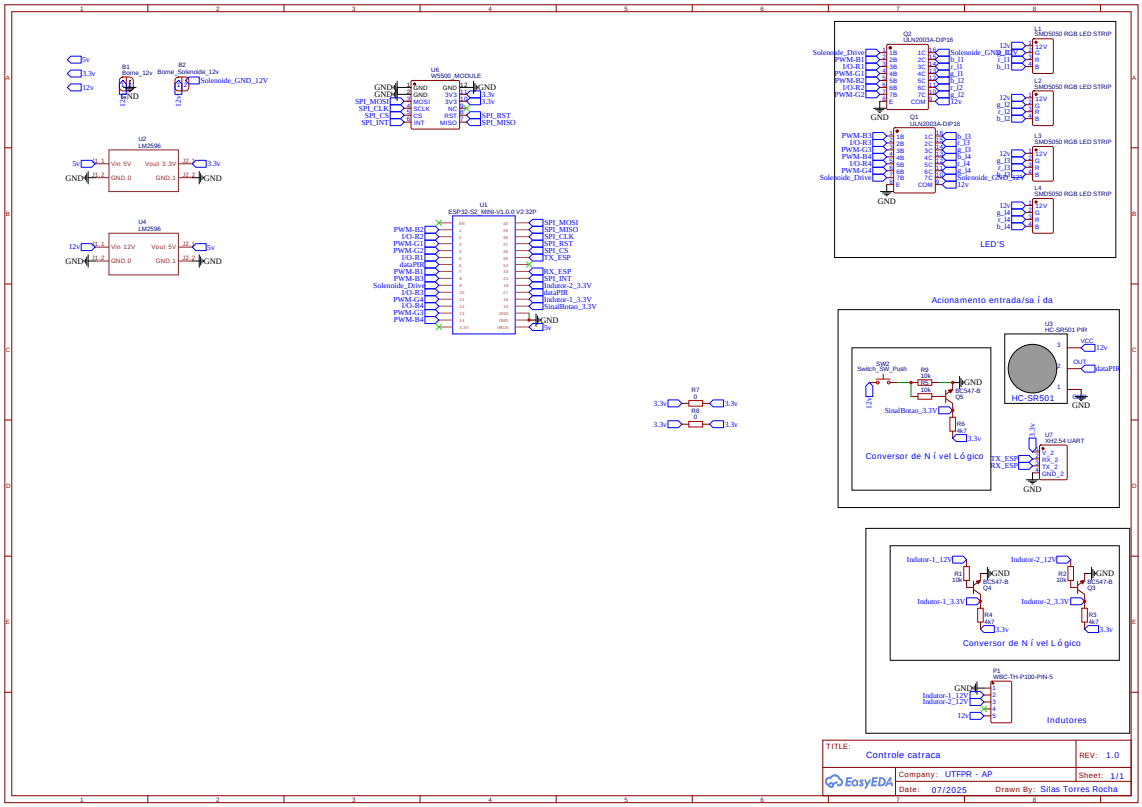


Figura 15: Diagrama Esquemático da Eletrônica de Controle.

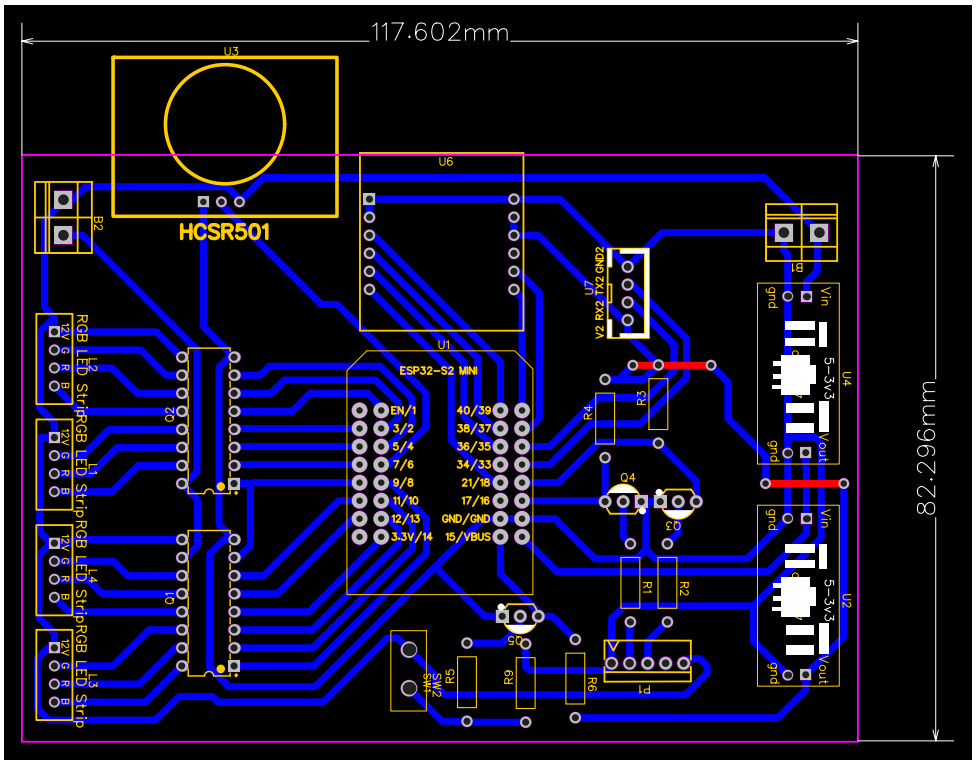


Figura 16: Layout da PCB (Versão 2.1.3).

# Referências


[1] Espressif Systems. *ESP32-S2 Technical Reference Manual*. 2025.

[2] Docker Inc. *Docker Documentation*. Disponível em: `https://docs.docker.com`.

[3] Texas Instruments. *ULN2003A Datasheet*. Disponível em: `https://www.ti.com/lit/ds/symlink/uln2003a.pdf`.

[4] FastAPI. *FastAPI Documentation*. Disponível em: `https://fastapi.tiangolo.com`.

[5] Eclipse Foundation. *Eclipse Mosquitto Documentation*. Disponível em: `https://mosquitto.org/documentation/`.

[6] WIZnet. *W5500 Datasheet*. Disponível em: `https://docs.wiznet.io/Product/ioModule/WIZ610io`.

[7] ON Semiconductor. *BC547 Datasheet*. Disponível em: `https://www.onsemi.com/pdf/datasheet/bc547-d.pdf`.

[8] Texas Instruments. *LM2596 Datasheet*. Disponível em: `https://www.ti.com/lit/ds/symlink/lm2596.pdf`.

[9] Adafruit. *HC-SR501 PIR Motion Sensor Datasheet*. Disponível em: `https://cdn-learn.adafruit.com/assets/assets/000/010/584/original/PIR_Motion_Sensor_v1.0.pdf`.

[10] Hubtronics. *GM81S Manual*. Disponível em: `https://hubtronics.in/docs/GM861S.pdf`.

[11] Mechatronics Fan Group. *B5015 Series Blower Fan Datasheet*. Disponível em: `https://www.digikey.com/en/products/detail/mechatronics`.

[12] Mouser Electronics. *Tactile Switch Datasheet*. Disponível em: `https://www.mouser.com/catalog/specsheets/grhls00979-1.pdf`.