UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”

INSTITUTO DE CIÊNCIA E TECNOLOGIA DE SOROCABA (ICTS)

RELATÓRIO PROJETO INTEGRADOR

Atividade 3 - Comunicação bidirecional segura e autenticação

Amanda Segura Mendes de Oliveira

Ana Clara Godoy Ensides

Beatriz Martuscelli da Silva Prado

Felipe Pellegrini Kumagae

Guilherme Yuiti de Queiroz Barbosa

Hebert de Oliveira Brito

Maysa Gabriela Lucas Izaias

Rafael Utsunomya Machado

SOROCABA-SP

2025

SUMÁRIO

1. Introdução

Na terceira parte do projeto em que tem-se o objetivo de controlar a planta de tanques acoplados (T1–T3), a comunicação entre o microcontrolador ESP32 e o servidor web tem papel fundamental na supervisão e controle do processo desses tanques acoplados. Enquanto na etapa anterior o foco estava restrito ao recebimento de dados de sensores pelo servidor, a presente atividade amplia o escopo para uma comunicação bidirecional, permitindo o envio de comandos e parâmetros do sistema web para o microcontrolador, consolidando um canal de controle e monitoramento remoto em tempo real utilizando o Node-Red.

A comunicação bidirecional proposta permite que o servidor não apenas receba leituras de variáveis de processo (como o nível do tanque intermediário T1 e a vazão da bomba de entrada P1), mas também envie instruções de controle ao ESP32, ajustando o setpoint aplicado à bomba conforme as demandas do sistema. Essa capacidade de interação mútua é essencial para a implementação de estratégias de controle em malha fechada, nas quais o nível de T3 é regulado automaticamente de acordo com o setpoint definido, mesmo diante de perturbações externas. Dessa forma, o sistema web atua como um supervisor digital que comunica, processa e registra todas as interações de controle entre o ambiente físico e a infraestrutura de software.

Além da funcionalidade operacional, esta etapa enfatiza a importância da segurança e da autenticação na comunicação entre os dispositivos conectados. Foram implementados mecanismos de autenticação de usuários (admnistrador e usuário comum) e de validação de dispositivos, assegurando que apenas entidades autorizadas possam acessar ou modificar os dados do processo. O armazenamento e a transmissão das informações foram reforçados por meio de técnicas de criptografia, e cada requisição (seja proveniente do ESP32 ou do cliente web) é registrada em log com carimbo de tempo, IP e status da operação. Esses recursos garantem a rastreabilidade, a confidencialidade e a integridade dos dados, elementos indispensáveis para a confiabilidade do sistema.

Para a implementação e validação do sistema, foram utilizados diferentes recursos de software que facilitaram o desenvolvimento da comunicação segura entre o ESP32 e o servidor. O Node-RED foi empregado como plataforma de orquestração de fluxos e integração entre o microcontrolador, back-end e front-end, permitindo testar rotas, processar dados e simular comandos. Para o armazenamento estruturado das informações, utilizou-se o XAMPP, reunindo o servidor Apache e o banco de dados

MySQL, onde foram criadas as tabelas destinadas aos usuários, leituras, comandos e logs. Além disso, scripts em Python foram desenvolvidos para realizar testes independentes de requisições, validação das rotas REST e verificação do fluxo de autenticação e criptografia. A Figura 1 mostra o Node-RED completo desenvolvido que será mais detalhado nas seções posteriores.

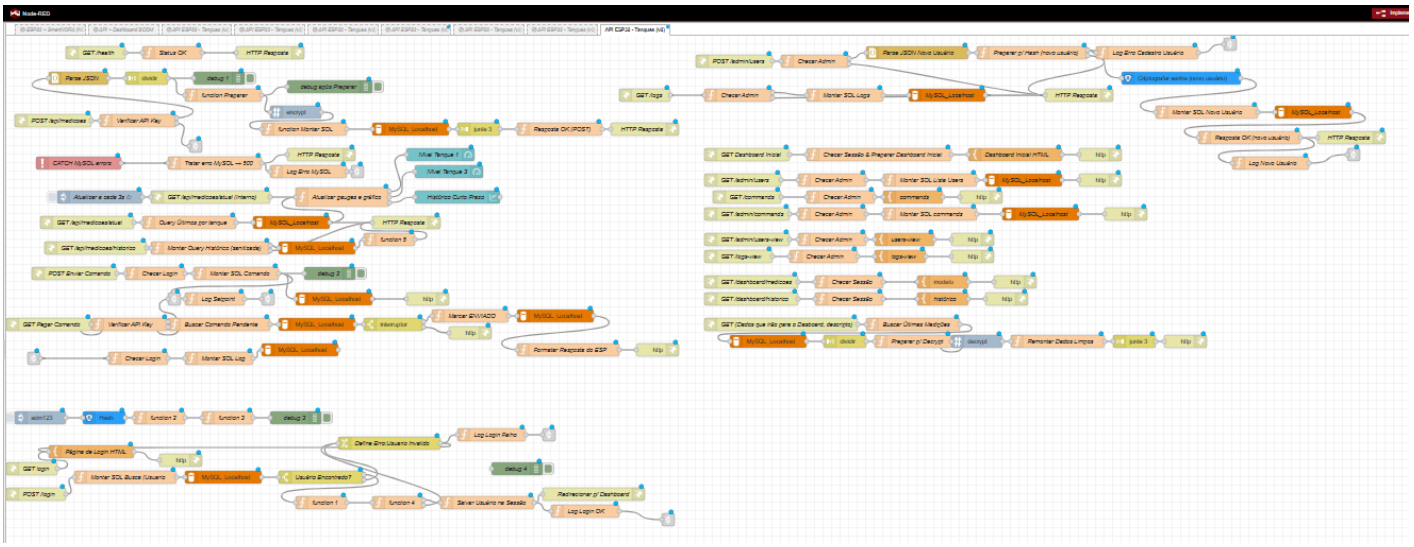


Figura 1 - Fluxos no Node-RED.

Por fim, a consolidação do back-end com rotas seguras, o gerenciamento de usuários com diferentes níveis de acesso e a implementação de logs de auditoria completam a estrutura de comunicação bidirecional segura. Essa arquitetura fornece a base para o desenvolvimento posterior do dashboard e das interfaces de visualização, que possibilitarão o acompanhamento intuitivo do processo e a análise dos dados de desempenho.

2. Autenticação, autorização e segurança

Para atender aos requisitos de segurança e controle de acesso, foi implementado um sistema robusto de autenticação e autorização. Para isso, o ambiente Node-RED foi estendido com bibliotecas (nodes) voltadas à segurança do sistema. A **Figura 2 e 3** mostram as paletas instaladas, destacando-se o **node-red-contrib-bcrypt** para o hashing seguro de senhas, o **node-red-contrib-crypto-js** para a criptografia AES dos dados, e o **node-red-node-mysql** para a comunicação com o banco de dados.

Figura 2 - Biblioteca Node-RED utilizada para banco de dados.

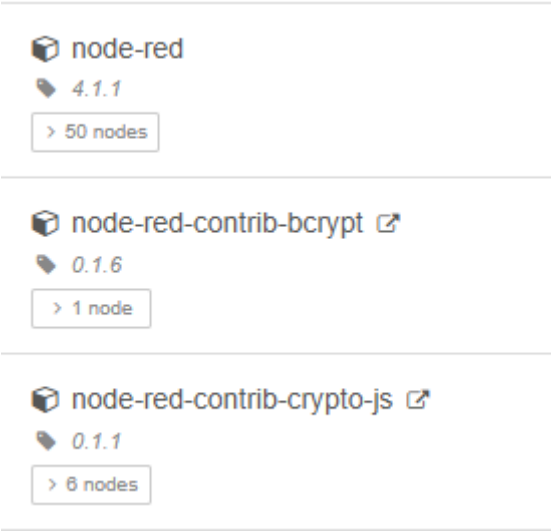


Figura 3 - Bibliotecas Node-RED utilizadas para segurança.

A instalação dessas dependências, como o **bcrypt**, é feita diretamente no ambiente do Node-RED via linha de comando, conforme ilustrado na **Figura 4**.

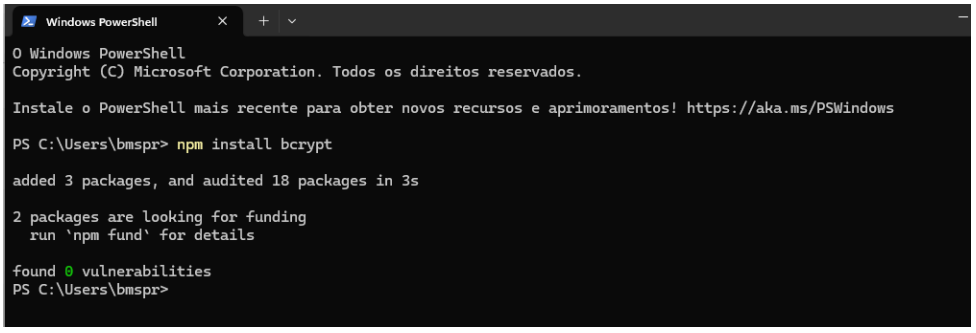


Figura 4 - Instalação da biblioteca bcrypt.

Contudo, para que bibliotecas como **bcrypt** e **crypto-js** pudessem ser acessadas de dentro dos nós **function**, e para habilitar o gerenciamento de sessões (via **express-session**), foi necessário modificar o arquivo de configuração **settings.js** do Node-RED. O bloco de código e a **Figura 5** detalham esse ajuste, onde o **express-session** é carregado como middleware e as bibliotecas de criptografia são injetadas no **functionGlobalContext**, tornando-as acessíveis para os fluxos.

```javascript
// topo do settings.js
const session = require('express-session');
const FileStore = require('session-file-store')(session);

// dentro do module.exports = {
httpNodeMiddleware: [
  session({
    store: new FileStore({ path: __dirname + '/sessions' }),
    secret:
'$2a$10$l2AhGYCdmnatorCZ/zki0.IMINS2azQoAzJqYSeS6k8GQtJHbHzEu',
    resave: false,
    saveUninitialized: false,
    cookie: { secure: false, maxAge: 24*60*60*1000 } // 1 dia
  })
],

// (para usar bcrypt em function nodes)
functionGlobalContext: {
  bcrypt: require('bcrypt'),
  cryptojs: require('crypto-js')
},
```

```
535     /** The following property can be used to set predefined values in Global Context.
536      * This allows extra node modules to be made available with in Function node.
537      * For example, the following:
538      *    functionGlobalContext: { os:require('os') }
539      * will allow the `os` module to be accessed in a Function node using:
540      *    global.get("os")
541      */
542     functionGlobalContext: {
543         bcrypt: require('bcrypt')
544         // os:require('os'),
545     },
```

Figura 5 - Detalhe do settings.js: Habilitação da biblioteca bcrypt no functionGlobalContext do Node-RED.

Essas bibliotecas possibilitaram a implementação dos blocos **bcrypt**, **encrypt** e **decrypt** no Node-RED. O bloco de **bcrypt** é utilizado para criptografar senha de novos usuários que estão sendo criados, por exemplo (Figura 6). Nas Figuras 7 e 8, pode-se notar que **encrypt** é utilizado para criptografar os dados para envio no banco de dados e **decrypt** é utilizado para descriptografar os dados do nível de água para exibição na tabela de medições.

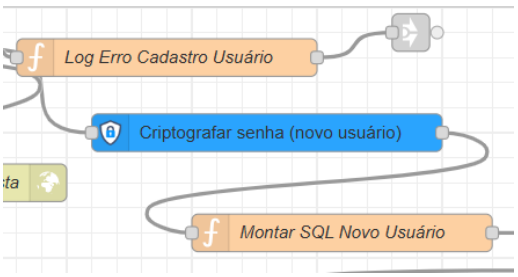


Figura 6 - Criptografar senha de novos usuários utilizando bcrypt.

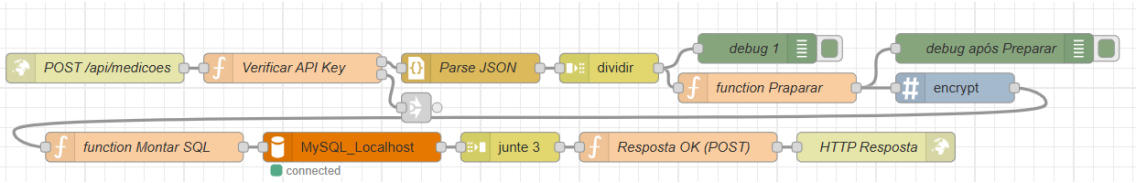


Figura 7 - Implementação de ecrypt.

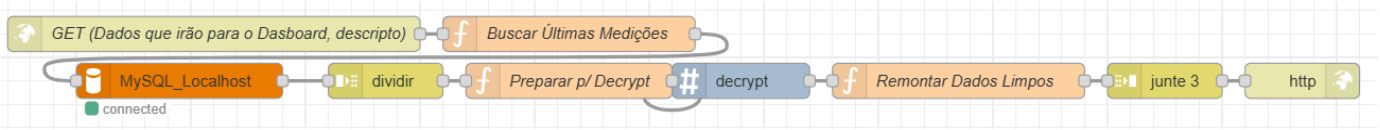


Figura 8 - Implementação de decrypt.

Como resultado da segurança, tem-se os dados criptografados no banco de dados, exibido na Figura 9. É possível observar que os dados de medições estão criptografados na colunas **value_encrypted** dentro da tabela **medicoes** no banco de dados **scom**. Na Figura 10 é possível visualizar as senhas criptografadas nesse mesmo banco de dados, mas agora na coluna **password_hash** na tabela **users**.

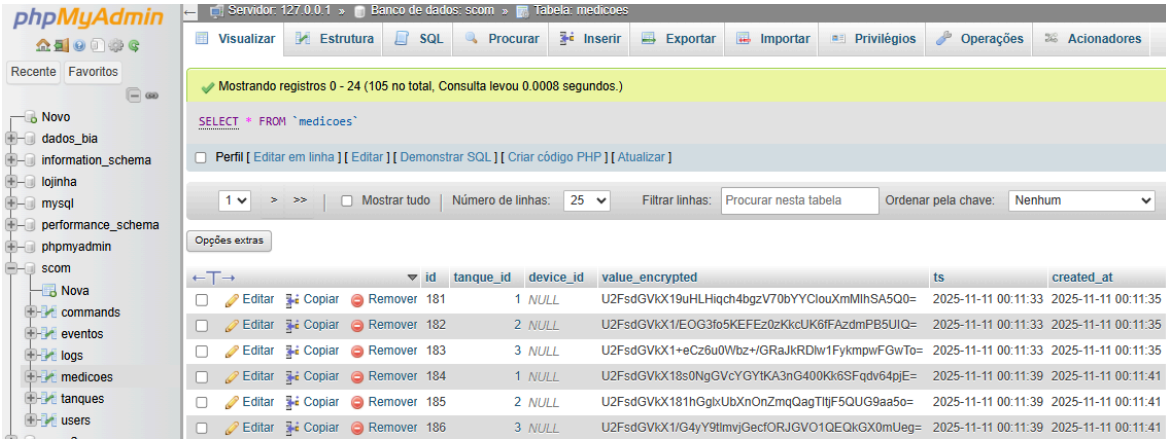


Figura 9 - Banco de dados com as medições criptografadas.

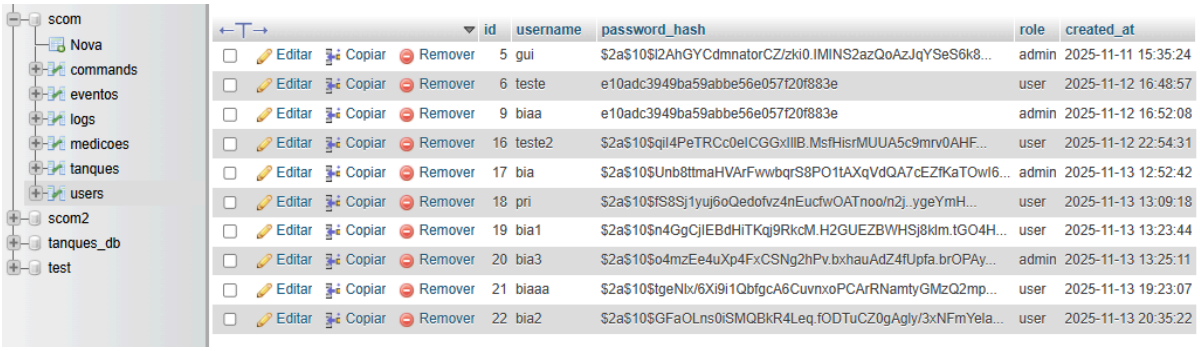


Figura 10 - Banco de dados com as senhas criptografadas.

Depois que os dados estão salvos de forma segura no banco de dados, a parte de visualização dos mesmo é feita a partir de autorização e autenticação do sistema. Criou-se um sistema que só pode ser acessado mediante autorização por login, sendo feita a distinção de usuário comum e de usuário administrador. Desse modo, para acessar a sessão e o dashboard, é preciso estar logado. A primeira página é justamente a do Login. A Figura 11 mostra a tentativa de login não bem sucedida.

Figura 11 - Acesso negado pois a autenticação falhou.

Assim, a proteção por sessão é feita em diversos momentos através dos blocos **Checar Sessão** ou **Checar Adm**. A Figura 12 exibe alguns fluxos que possuem verificação de login, como acesso ao dashboard, às medições, aos registros de atividade (logs-view), aos comandos de setpoint (commands) e visualização e cadastro de usuários. Assim, algumas páginas só podem ser acessadas por administradores como os fluxos GET /logs, GET /logs-view, GET /admin/users-view, GET /admin/users, GET /commands e GET /admin/commands.

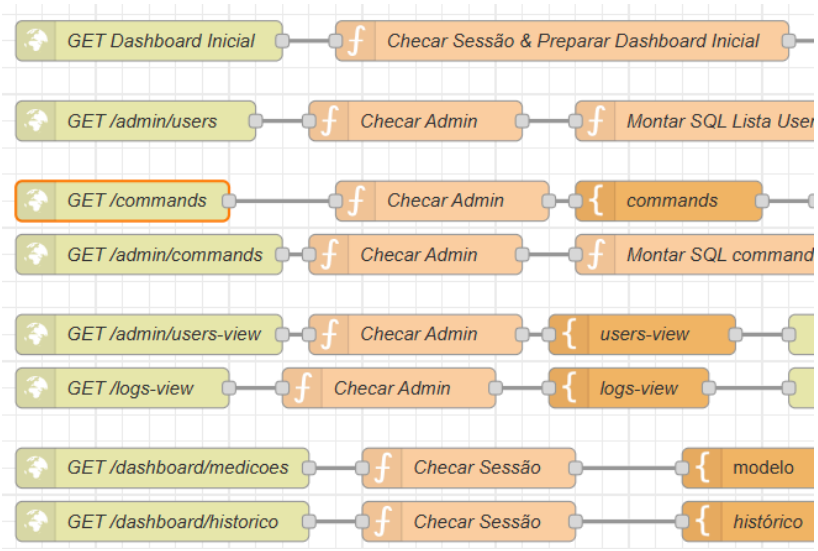


Figura 12 - Fluxos com proteção de sessão.

Antes de ações sensíveis, como no envio de comando, também há checagem de login. Por exemplo, depois de POST /api/send (Figura 14) e depois de “SALVAR LOG”, assegurando que há usuário na sessão antes de gravar registro (Figura 13).

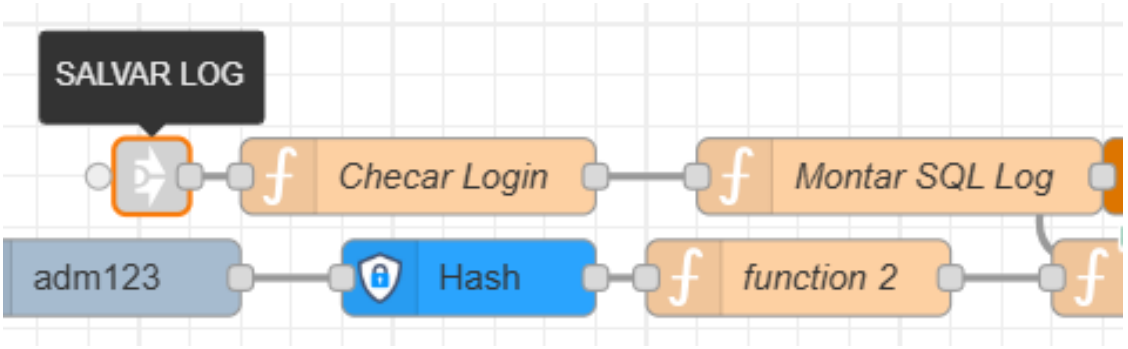


Figura 13 - Checagem de login.

3. Comunicação entre servidor, site e ESP32

O núcleo desta atividade foi a consolidação da comunicação bidirecional.Toda a lógica de back-end, bem como o front-end, foi implementada no Node-RED. A arquitetura de comunicação com o dispositivo ESP32 foi dividida em duas rotas de API principais, ambas protegidas por uma x-api-key.

A primeira via de comunicação é o recebimento de dados do ESP32, gerenciada pela rota POST /api/medicoes, como detalhado na **Figura 14**.

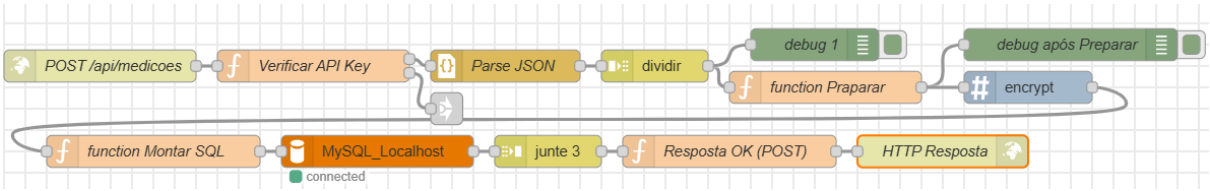


Figura 14 - Fluxo de recebimento e criptografia de dados do ESP32 (POST /api/medicoes).

Neste fluxo, a requisição do ESP32 é recebida e primeiramente passa pelo nó "Verificar API Key" para autenticar o dispositivo. Em seguida, o *payload* JSON é dividido (nó "dividir"), e o nó "**function Praparar**" isola o valor do **nivel**. Este valor é passado ao nó "encrypt", que utiliza o algoritmo AES para criptografá-lo antes que o nó "function Montar SQL" o prepare para ser inserido na coluna **value_encrypted** da tabela **medicoes**.

A segunda via de comunicação, do site para o ESP32, é mais complexa e envolve duas rotas que operam em conjunto, como visto na **Figura 15**.

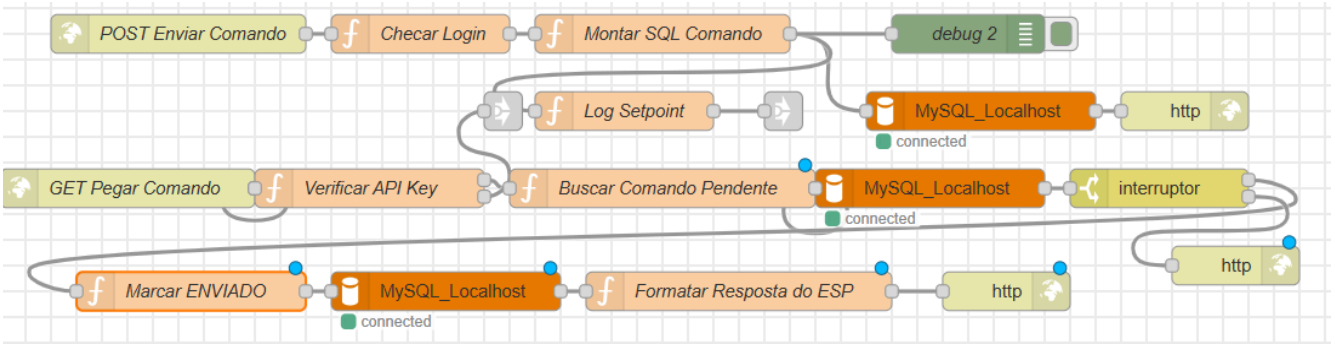


Figura 15 - Fluxo de envio de comandos (POST /api/send e GET /api/commands).

O processo de envio de um comando se inicia no front-end. Um usuário autenticado (seja 'admin' ou 'user') preenche o formulário de envio de parâmetros. Esse envio dispara uma requisição POST /api/send. No back-end, o Node-RED valida a sessão do usuário e executa o nó "Montar SQL Comando", que insere o comando na tabela commands do banco de dados com o status inicial 'pending' e o user_id do requisitante.

A **Figura 16** demonstra o resultado dessa primeira etapa. Um comando de 'setpoint' (ID 16) foi enviado pelo site e está registrado no sistema, aguardando que o microcontrolador o consulte.

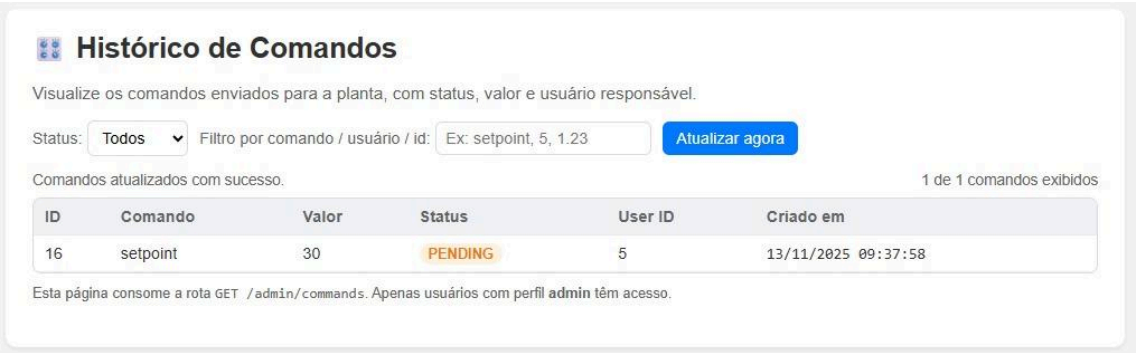


Figura 16 - Dashboard "Histórico de Comandos" exibindo o comando (ID 16) com o status "PENDING" após o envio pelo usuário.

Em seguida, o microcontrolador ESP32, simulado pelo script **teste.py**, entra no fluxo. O script executa uma requisição **GET /api/commands** de forma periódica, autenticando-se com a **x-api-key** em cada chamada. Do lado do servidor, essa rota (protegida pela API Key) consulta no banco de dados o comando 'pending' mais antigo.

A **Figura 17** captura o momento exato dessa transação. O console do **teste.py** mostra que o simulador do ESP32, em seu primeiro ciclo, chamou a API e recebeu com sucesso os dados do comando ID 16.

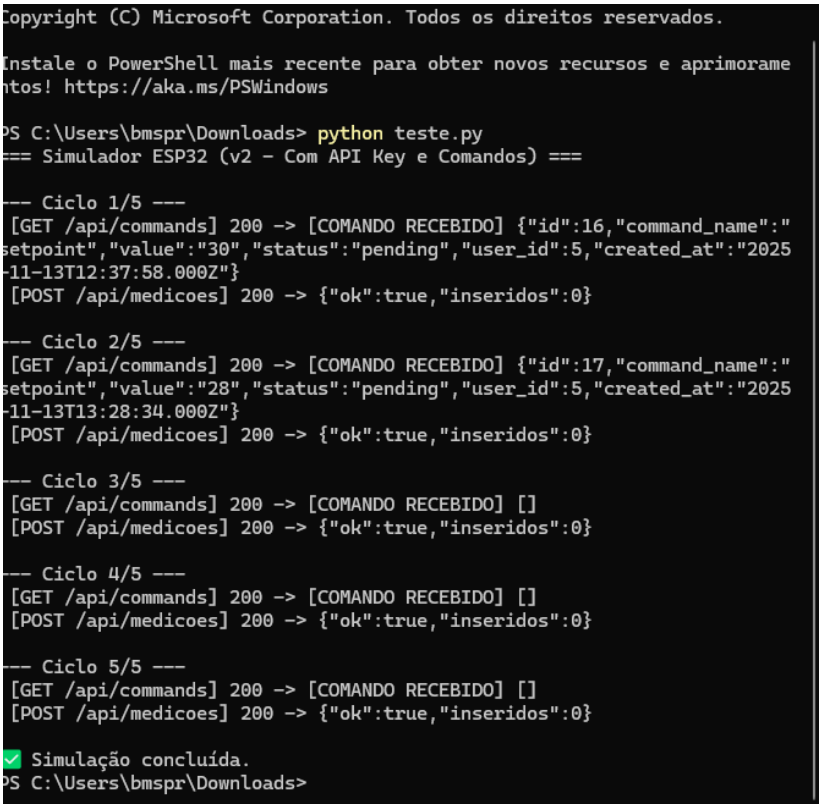


Figura 17 - Console do simulador teste.py exibindo a resposta "[COMANDO RECEBIDO]" para o comando de ID 16, confirmando o recebimento dos dados.

Assim que o servidor entrega o comando ao ESP32, o nó "Marcar ENVIADO" é executado, preparando e executando uma consulta UPDATE para alterar o status do comando no banco de dados para 'sent'.

A **Figura 18** mostra o estado final do sistema. A mesma página "Histórico de Comandos", ao ser atualizada, agora reflete o status "SENT" para o comando ID 16. Essa sequência de imagens valida todo o ciclo de comunicação do site para o ESP32, mediado pelo servidor e pelo banco de dados.

Figura 18 - O dashboard “Histórico de Comandos” após a consulta do ESP32, mostrando o status do comando (ID 16) atualizado para “SENT”.

O ciclo se fecha com o envio dos dados do ESP32 para o servidor. O próprio console da Figura 17 mostra o ESP32 realizando o **POST/api/medicoes,** que, por sua vez, é recebido, validado pela x-api-key, criptografado e salvo no banco de dados, completando a comunicação bidirecional.

4. Banco de dados relacional e estrutura de armazenamento

Para suportar as novas funcionalidades de autenticação, comandos e auditoria, a estrutura do banco de dados **scom** foi expandida**.**

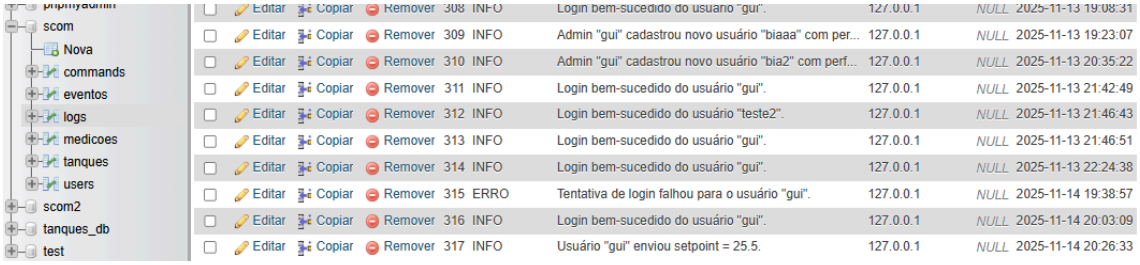


Figura 19 -

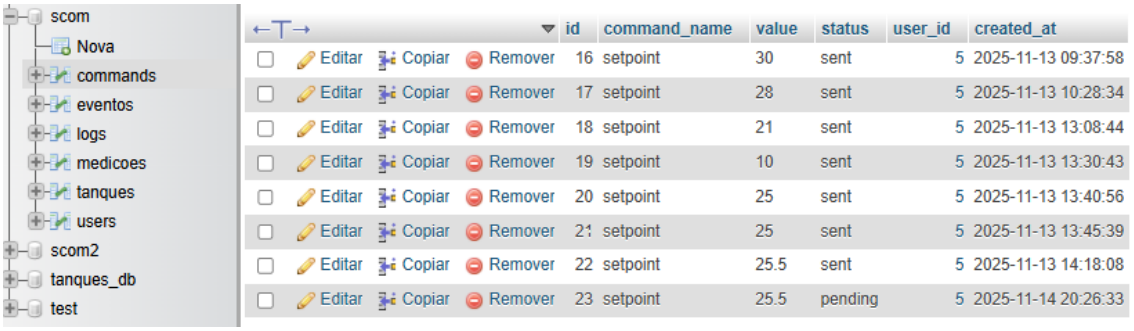


Figura 20 -

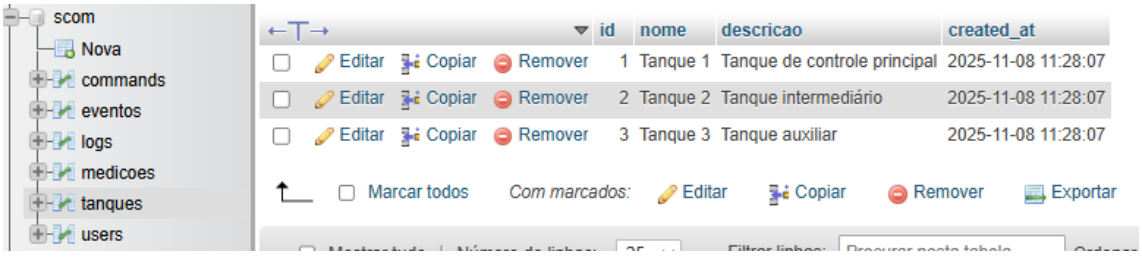


Figura 21 -

Tabelas **users** e **medicoes** estão exibidas na seção 1 de segurança, autenticação e autorização deste relatório nas Figuras 9 e 10. Tabela **eventos** ainda não está sendo utilizada.

5. Registro e auditoria (logs)

Para garantir a rastreabilidade e a segurança do sistema foi implementado um mecanismo de auditoria, centralizado na tabela **logs** do banco de dados. Esta tabela foi projetada para capturar todos os eventos relevantes da aplicação, armazenando o tipo de evento (**event_type**), uma descrição detalhada (**description**), o endereço de IP da requisição (**ip_address**) e o **user_id** (quando aplicável).

A implementação no Node-RED foi desenhada para ser modular. Um exemplo claro é o fluxo de autenticação, detalhado na **Figura 22**.

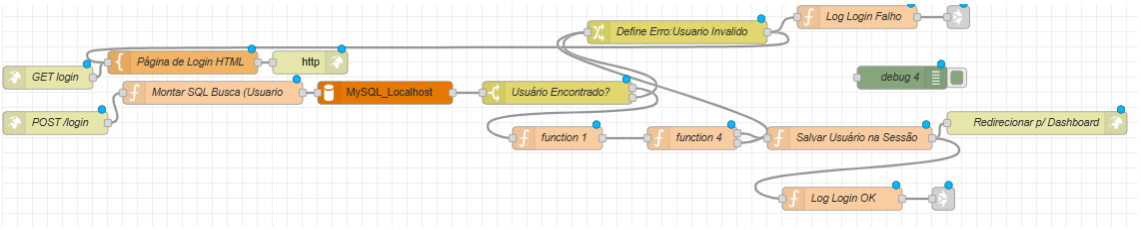


Figura 22 - Detalhe do fluxo de login (POST /login), mostrando os gatilhos de log para sucesso ("Log Login OK") e falha ("Log Login Falho").

Como a **Figura 22** demonstra, após o processo de verificação de senha (nó "function 4", que executa o bcrypt.compare), o fluxo se ramifica. Se a autenticação for bem-sucedida, ele aciona o nó "Log Login OK" antes de salvar a sessão. Se falhar, ele aciona o nó "Log Login Falho". Esses nós não gravam no banco de dados diretamente; em vez disso, eles preparam uma mensagem e a enviam (via link out) para um fluxo central de registro. Esse fluxo central, "SALVAR LOG", é mostrado na **Figura 23.**

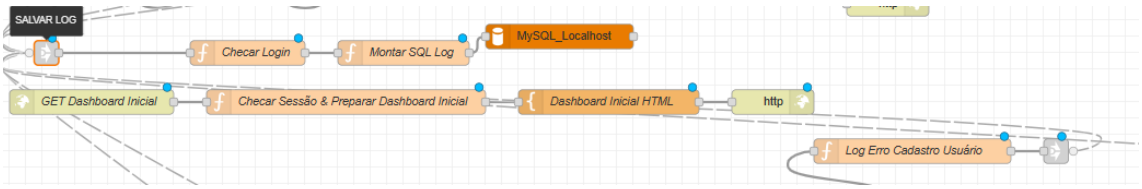
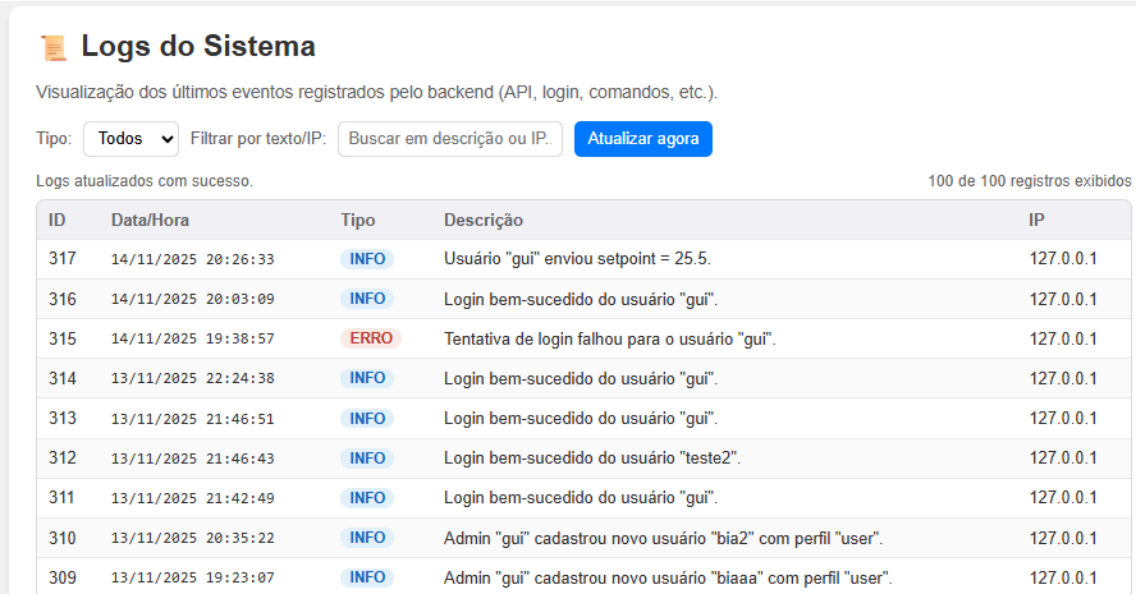


Figura 23 - O fluxo central "SALVAR LOG", que recebe eventos (via link in), valida a sessão e os insere no banco de dados.

Este fluxo "SALVAR LOG" é o pilar da auditoria. Ele é acionado por link in nodes de múltiplos pontos da aplicação (como os do fluxo de login, do envio de setpoints e da validação de API keys). Ao ser acionado, ele checa o login (se houver), monta o INSERT final e o executa no MySQL_Localhost, centralizando toda a lógica de auditoria em um único ponto reusável.

O resultado de toda essa coleta de dados é apresentado ao administrador na interface web, como mostra a **Figura 24**. Esta página, acessível apenas ao 'admin', consome a rota /logs para consultar e exibe os 100 eventos mais recentes do banco de

dados, permitindo ao administrador monitorar a saúde e a segurança da aplicação em tempo real. É possível ver registros de envio se sepoints, de login bem-sucedido, erro ao logar e cadastro de novos usuários.



Logs do Sistema

Visualização dos últimos eventos registrados pelo backend (API, login, comandos, etc.).

Tipo: Todos | Filtrar por texto/IP: Buscar em descrição ou IP. | Atualizar agora

Logs atualizados com sucesso. 100 de 100 registros exibidos

ID	Data/Hora	Tipo	Descrição	IP
317	14/11/2025 20:26:33	INFO	Usuário "gui" enviou setpoint = 25.5.	127.0.0.1
316	14/11/2025 20:03:09	INFO	Login bem-sucedido do usuário "gui".	127.0.0.1
315	14/11/2025 19:38:57	ERRO	Tentativa de login falhou para o usuário "gui".	127.0.0.1
314	13/11/2025 22:24:38	INFO	Login bem-sucedido do usuário "gui".	127.0.0.1
313	13/11/2025 21:46:51	INFO	Login bem-sucedido do usuário "gui".	127.0.0.1
312	13/11/2025 21:46:43	INFO	Login bem-sucedido do usuário "teste2".	127.0.0.1
311	13/11/2025 21:42:49	INFO	Login bem-sucedido do usuário "gui".	127.0.0.1
310	13/11/2025 20:35:22	INFO	Admin "gui" cadastrou novo usuário "bia2" com perfil "user".	127.0.0.1
309	13/11/2025 19:23:07	INFO	Admin "gui" cadastrou novo usuário "biaaa" com perfil "user".	127.0.0.1

Figura 24 - Interface de visualização de logs, disponível para administradores, exibindo eventos de envio de setpoint pelo usuário "gui".

6. Interface web (front-end)

O front-end do sistema foi inteiramente construído utilizando os nós http in e template (HTML/Mustache) do Node-RED. A interface foi projetada para ser o principal ponto de interação do usuário com o sistema, sendo seu acesso totalmente condicionado à autenticação e autorização.

Após o login bem-sucedido na rota /login, o usuário é redirecionado para o dashboard principal, servido pela rota GET /dashboard (identificada no fluxo como "GET Dashboard Inicial"), como mostra a **Figura 25**.

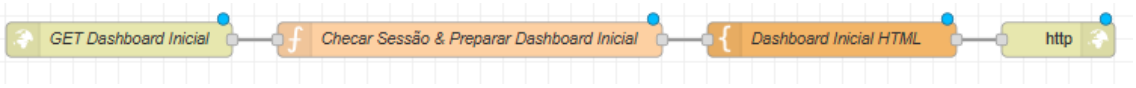


Figura 25 - Fluxo da rota principal do dashboard (GET /dashboard), mostrando o nó de checagem de sessão.

O nó "Checar Sessão & Preparar Dashboard Inicial" é crucial: ele valida se o usuário está logado e, em caso positivo, injeta seus dados da sessão (incluindo o role) no msg. O template "Dashboard Inicial HTML" utiliza essa informação de role (especificamente a variável msg.isAdmin) para renderizar condicionalmente a seção "Administração" usando a lógica {{#isAdmin}}...{{/isAdmin}}.

6.1 Acesso do Usuário Comum

Para um usuário com role 'user', a variável **msg.isAdmin** é falsa. Isso faz com que o template mustache oculte a seção "Administração", renderizando a interface de privilégios mínimos, como demonstrado na **Figura 26**.

Figura 26 - Dashboard principal para um "Usuário Comum", exibindo apenas a seção de Monitoramento, comprovando a restrição de acesso.

O usuário comum tem acesso total à seção "Monitoramento". Os fluxos de back-end para essas páginas, vistos na **Figura 27**, confirmam a lógica de autorização.

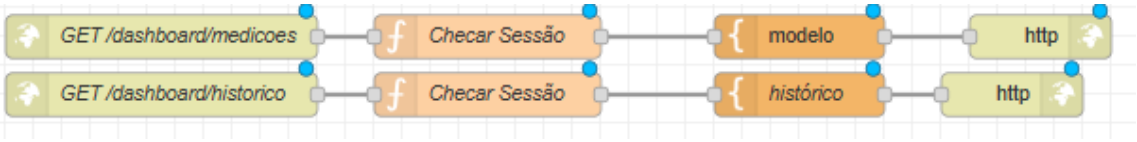


Figura 27 - Fluxos das rotas de "Monitoramento" (/dashboard/medicoes e /dashboard/historico), protegidas pelo nó "Checar Sessão".

Como detalhado na **Figura 27**, essas rotas são protegidas pelo nó "Checar Sessão", que valida se *qualquer* usuário está logado, permitindo o acesso de ambos os perfis. A primeira rota, /dashboard/medicoes, leva à página da **Figura 28**.

Figura 28 - Página "Medições atuais", exibindo o formulário de envio de setpoint e a visualização textual dos últimos dados recebidos.

Esta página cumpre o requisito fundamental da tarefa de fornecer um "formulário simples de envio de parâmetros" (o campo "Novo Setpoint") e uma "área básica de visualização textual" (o bloco JSON "Últimos Dados Recebidos"). A visualização em JSON comprova que os dados estão sendo recebidos do ESP32 e descriptografados pela API /api/data/latest para exibição. A segunda rota de monitoramento, /dashboard/historico, leva à página da **Figura 29**.

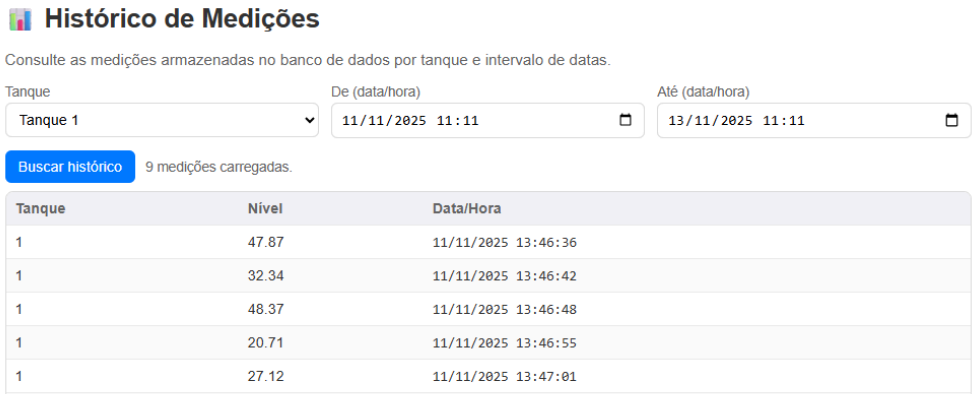


Figura 29 - Página “Histórico de Medições”, que consome a rota /api/medicoes/historico para exibir dados filtrados e descriptografados.

Essa interface permite ao usuário consultar o banco de dados e visualizar os dados históricos, que também são descriptografados no back-end antes de serem apresentados em formato de tabela.

6.2 Acesso do Administrador

Quando um usuário com **role** ‘admin’ acessa o dashboard, a variável **msg.isAdmin** é verdadeira, fazendo com que o template **mustache** renderize a seção “Administração”, como visto na Figura 30.

Figura 30 - Dashboard principal para um usuário com perfil "Administrador", exibindo as seções de Monitoramento e Administração.

O administrador pode acessar as mesmas páginas de monitoramento que o usuário comum, mas também obtém acesso aos links "Logs do sistema", "Cadastro de usuários" e "Histórico de comandos". A segurança, no entanto, não confia apenas em ocultar links. A diferença crucial está na camada de autorização do back-end, como demonstram as **Figuras 31** e **32**.

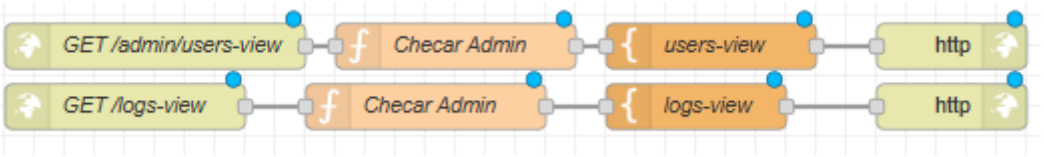


Figura 31 - Fluxos das páginas /admin/users-view e /logs-view, protegidas pelo nó "Checar Admin".

Figura 32 - Fluxo da página /commands, também protegido pelo nó "Checar Admin".

Como os fluxos demonstram, as rotas de administração (/admin/users-view, /logs-view e /commands) são protegidas por um nó "Checar Admin". Este nó realiza uma verificação mais estrita, validando se msg.req.session.user.role === 'admin', bloqueando qualquer tentativa de acesso direto de um usuário comum. Um exemplo do resultado de uma dessas rotas é a página de "Cadastro de Usuários", vista na **Figura 33**.

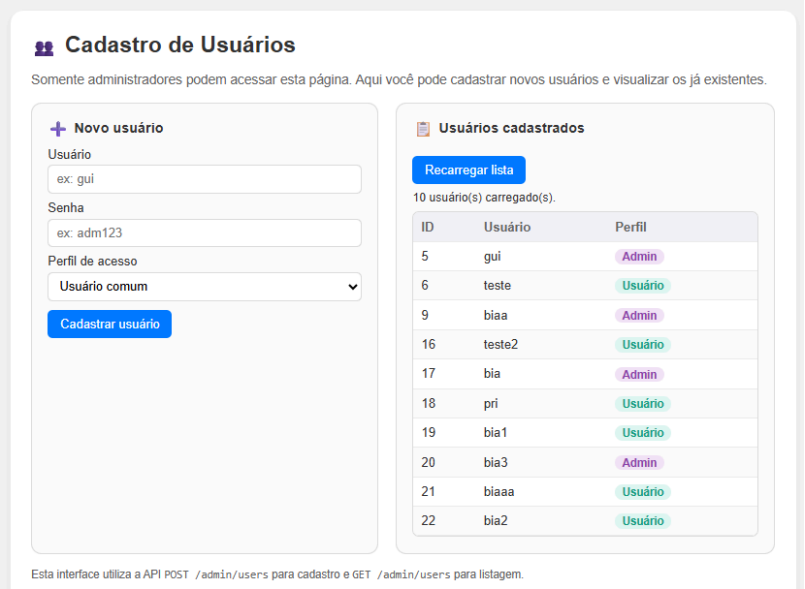


Figura 33 - Página de "Cadastro de Usuários", acessível apenas por administradores, que consome as rotas POST e GET /admin/users.

Esta interface comprova o funcionamento das rotas administrativas, permitindo ao admin criar novos usuários (chamando POST /admin/users) e listar os existentes (chamando GET /admin/users), funcionalidades que são corretamente negadas aos usuários comuns.

7. Validação final e documentação

Organização e documentação do código, incluindo README e descrição das rotas.