DESENVOLVIMENTO DE UM PROTÓTIPO WEB PARA ACESSO E CONTROLE REMOTO DE COMPUTADORES

SALLYTSA DIAZ SANTANA MAGALHÃES BENTO, BRUNO MICHEL PERA[1].

[1]Instituto Educacional Ribeiro Goulart, Jardim Oriente, 37, Rua Isaac Newton - 12236-240 - São José dos Campos-SP, Brasil, sallytsadiaz@gmail.com, brunomichel00@gmail.com.

Resumo

O acesso remoto a computadores possibilita suporte técnico, administração e acompanhamento de estações sem presença física no local. Este trabalho apresenta o desenvolvimento e a análise do MiniAnyDesk 2.0, um protótipo de controle remoto composto por um agente desktop em Python, um servidor intermediário em Node.js e uma interface web. O agente captura a tela, comprime os quadros em JPEG e os transmite em Base64 por eventos Socket.IO, enquanto o navegador envia comandos de mouse e teclado à máquina selecionada. A metodologia envolveu inspeção do código-fonte, descrição da arquitetura e teste exploratório de inicialização das rotas HTTP. O servidor iniciou corretamente, respondeu ao endpoint de estado e disponibilizou a página do controlador. A análise confirmou a implementação do fluxo de registro, transmissão e entrada remota, mas identificou limitações de autenticação, autorização, persistência e distribuição dos quadros. Conclui-se que o protótipo demonstra a viabilidade didática da arquitetura, porém necessita de controles de segurança e otimizações antes de uso fora de ambiente autorizado e supervisionado.

Palavras-chave: Acesso remoto. WebSocket. Socket.IO. Automação de interface. Segurança de software.

Área do Conhecimento: Ciências Exatas e da Terra - Ciência da Computação.

Introdução

Ferramentas de acesso remoto permitem visualizar e controlar uma interface gráfica executada em outro computador, reduzindo a necessidade de deslocamento em tarefas de suporte, manutenção e administração. O protocolo Remote Framebuffer, utilizado pelo VNC, consolidou o princípio de transmitir a representação visual da máquina remota e devolver eventos de entrada do usuário, independentemente do sistema de janelas empregado (Richardson; Levine, 2011). Em aplicações web, o protocolo WebSocket oferece comunicação bidirecional após uma negociação inicial e mantém um canal adequado à troca contínua de mensagens entre navegador e servidor (Fette; Melnikov, 2011).

Sobre essa base, bibliotecas orientadas a eventos simplificam o tratamento de conexões persistentes. O Socket.IO disponibiliza comunicação bidirecional, de baixa latência e baseada em eventos, com alternativas de transporte como WebSocket e HTTP long-polling (Socket.IO, 2026). No ecossistema Python, a implementação python-socketio permite que clientes escritos nessa linguagem se comuniquem com a implementação JavaScript compatível, mantendo o mesmo modelo de eventos (Python-SocketIO, 2026). Essa integração favorece protótipos em que a captura e a automação local são feitas em Python, enquanto o servidor e a interface utilizam tecnologias web.

Apesar da utilidade, o controle remoto envolve informações sensíveis e capacidade de executar ações no computador de destino. Por isso, identificação da máquina, autenticação do usuário, autorização por sessão, proteção do transporte, registro de atividades e consentimento local são requisitos centrais. A arquitetura de confiança zero recomenda que usuários, dispositivos e recursos não recebam confiança implícita apenas por sua localização de rede e que autenticação e autorização sejam verificadas antes do acesso (Rose et al., 2020). Em um protótipo acadêmico, analisar essas lacunas é tão importante quanto demonstrar o fluxo funcional.

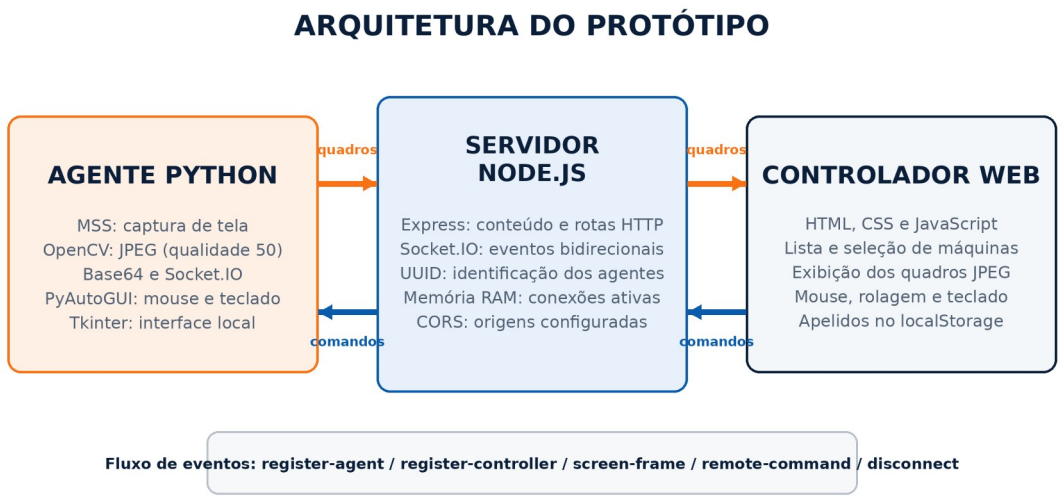
O objetivo deste trabalho foi descrever, verificar e discutir o MiniAnyDesk 2.0, um protótipo de acesso remoto formado por agente Python, servidor Node.js e controlador web. Buscou-se identificar os componentes implementados, registrar evidências mínimas de funcionamento do servidor, analisar o percurso dos quadros e comandos e apontar melhorias necessárias para evolução segura e escalável do sistema.

Metodologia

A pesquisa possui natureza aplicada e caráter descritivo, organizada como resultado de projeto. O objeto analisado foi o conjunto de arquivos fornecido: agent.py, server.js, public/index.html, package.json e dependências. O procedimento foi dividido em quatro etapas: inspeção estática dos componentes; reconstrução do fluxo de comunicação; execução exploratória do servidor em ambiente local; e comparação das decisões do protótipo com princípios técnicos e de segurança presentes na documentação das tecnologias utilizadas.

Arquitetura do protótipo. O sistema foi dividido em três componentes, conforme a Figura 1. O agente desktop foi implementado em Python e apresenta uma interface Tkinter para informar o endereço do servidor, conectar a estação e exibir o identificador recebido. A captura do monitor principal é realizada pela biblioteca MSS; cada quadro é convertido do formato BGRA para BGR e codificado em JPEG pelo OpenCV com qualidade 50. O conteúdo binário é transformado em Base64 e enviado em um evento screen-frame. A frequência configurada no código é de 12 quadros por segundo.

Figura 1 - Arquitetura e fluxo de dados do MiniAnyDesk 2.0.



Fonte: elaborado pelos autores (2026).

O servidor utiliza Node.js, Express e Socket.IO. O Express fornece a interface estática e as rotas HTTP /status e /agents, padrão compatível com a construção de aplicações e serviços HTTP descrita na documentação do framework (Express.js, 2026). Quando um agente emite register-agent, o servidor cria um UUID, associa esse identificador ao socket e divulga a lista atualizada. Os controladores são registrados separadamente. Os quadros recebidos são encaminhados aos controladores conectados e os comandos são direcionados ao socket correspondente ao agentId selecionado. As estruturas agents e controllers permanecem apenas na memória RAM.

A interface do controlador foi escrita em HTML, CSS e JavaScript. Ela apresenta uma barra lateral com as máquinas disponíveis, permite selecionar uma estação e associar um apelido armazenado no localStorage do navegador. O quadro recebido é aplicado ao atributo src de uma imagem. O movimento do mouse é normalizado entre zero e um de acordo com as dimensões exibidas; no agente, os valores são multiplicados pela resolução da tela real. Também foram mapeados clique, duplo clique, rolagem, tecla pressionada e pausa do compartilhamento.

Procedimento de verificação. O teste exploratório foi restrito à inicialização segura do servidor e às rotas que não executam controle remoto. O arquivo server.js foi iniciado em ambiente local, com a porta padrão 3000. Em seguida, foram realizadas requisições HTTP para /status, /agents e para a página inicial. A cadeia de captura e automação foi verificada por inspeção do código, sem controlar máquinas de terceiros. Foram considerados como critérios a presença dos eventos, o encaminhamento por identificador, o tratamento de desconexão e a correspondência entre os comandos gerados no navegador e as ações executadas pelo agente.

Resultados

O servidor iniciou na porta 3000 e disponibilizou a interface estática. A rota /status retornou um objeto JSON com status igual a OK, nome Mini AnyDesk 2.0 e contadores de agentes e controladores inicialmente zerados. A rota /agents retornou um vetor vazio antes do registro de estações. A página inicial apresentou o título do controlador, confirmando o carregamento do conteúdo público. A Tabela 1 sintetiza as verificações e separa os resultados executados das constatações obtidas por inspeção estática.

Tabela 1 - Verificações realizadas no protótipo.

Item	Procedimento	Evidência obtida
Inicialização	Execução de server.js	Serviço iniciado na porta 3000
Estado do serviço	GET /status	JSON com status OK e contadores zerados
Lista inicial	GET /agents	Vetor vazio antes da conexão de agentes
Interface	GET /	Página do controlador e recursos estáticos carregados
Eventos	Inspeção do código	Registro, quadros, comandos e desconexão mapeados
Entrada remota	Inspeção do agente	Movimento, clique, duplo clique, rolagem, tecla e pausa

Fonte: elaborado pelos autores (2026).

A inspeção mostrou coerência entre as três camadas. O agente envia agentId e image; o servidor retransmite os mesmos campos; e o controlador atualiza a imagem somente quando o identificador recebido coincide com a máquina selecionada. No sentido inverso, o controlador anexa agentId ao comando, o servidor localiza o socket do agente e o agente interpreta o campo type. Essa separação torna o protótipo compreensível e favorece seu uso didático para demonstrar comunicação em tempo real, eventos, normalização de coordenadas e integração entre linguagens.

Figura 2 - Pré-visualização estática da interface do controlador web.



Fonte: elaborado pelos autores a partir do código do protótipo (2026).

A interface organiza as máquinas em uma barra lateral, mantém a seleção visual e concentra a tela remota na área principal. Os apelidos são persistidos apenas no navegador, enquanto os UUIDs são recriados a cada registro. Portanto, após uma reconexão, o mesmo computador pode receber outro identificador e perder a associação com o nome salvo. O botão Parar envia stop-sharing e limpa a imagem local; embora o agente também reconheça start-sharing, o controlador fornecido não apresenta um botão para retomar a transmissão.

Discussão

Os resultados demonstram uma implementação mínima do ciclo típico de desktop remoto: captura da interface, envio de quadros, apresentação ao operador e devolução de eventos de entrada. A opção por JPEG com qualidade 50 reduz o volume de cada quadro, mas a codificação Base64 amplia o tamanho do conteúdo transmitido em relação aos bytes binários e exige conversões adicionais. Além disso, a frequência fixa de 12 quadros por segundo não considera alterações de cena, disponibilidade de banda ou atraso da conexão. Uma evolução pode empregar mensagens binárias, redução adaptativa de resolução e frequência, detecção de regiões modificadas ou codecs apropriados para vídeo.

O encaminhamento atual envia cada screen-frame a todos os controladores, mesmo que apenas um deles tenha selecionado o agente. O filtro ocorre no navegador depois do recebimento, o que consome banda e expõe quadros a clientes que não precisam deles. O próprio Socket.IO oferece mecanismos de direcionamento a subconjuntos de conexões, como salas e emissões para sockets específicos (Socket.IO, 2026). Assim, cada sessão deveria criar uma associação explícita entre operador e máquina, e o servidor deveria enviar a imagem somente ao controlador autorizado.

A principal limitação é a ausência de autenticação e autorização. Qualquer cliente capaz de alcançar o servidor pode tentar registrar-se como controlador e enviar comandos para um agentId conhecido. A lista de origens CORS reduz acessos de navegadores fora das origens configuradas, mas não substitui autenticação e não impede clientes que não aplicam a política do navegador. Também não há aceite local por sessão, expiração de autorização, registro de auditoria, limitação de tentativas ou validação rigorosa do conteúdo dos eventos. Em coerência com confiança zero, a evolução deve autenticar usuário e dispositivo, autorizar cada sessão e reavaliar continuamente o acesso ao recurso (Rose et al., 2020).

No agente, PyAutoGUI transforma os eventos em ações reais de mouse e teclado; sua documentação confirma as operações de movimento, clique, rolagem e teclas empregadas no protótipo (PyAutoGUI, 2026). Entretanto, o código desativa o mecanismo de segurança FAILSAFE e elimina a pausa entre ações. Para um sistema de controle remoto, recomenda-se preservar um método local de interrupção, exibir indicação visível de sessão ativa, solicitar consentimento e restringir comandos potencialmente perigosos. Também é necessário alinhar a URL de implantação com as origens permitidas, proteger todo o tráfego com TLS e armazenar segredos fora do código-fonte.

A memória volátil simplifica o protótipo, mas limita a continuidade do serviço e a execução com múltiplas instâncias. Reiniciar o servidor remove agentes e controladores, e um balanceador pode encaminhar eventos para processos diferentes. Uma versão ampliada pode usar banco de dados para usuários, dispositivos e auditoria, além de um adaptador de mensagens para distribuir eventos. Testes futuros devem medir latência, taxa de quadros, uso de CPU, consumo de banda e comportamento sob perda de pacotes, sempre em laboratório e com autorização das máquinas envolvidas.

Conclusão

O MiniAnyDesk 2.0 alcançou o propósito de demonstrar, em um protótipo compacto, a integração entre captura de tela em Python, intermediação em Node.js/Socket.IO e controle por navegador. O teste exploratório confirmou a inicialização do servidor, as rotas básicas e a entrega da interface; a inspeção do código confirmou o mapeamento de registro, transmissão, seleção de agente e comandos de entrada. A arquitetura é adequada como objeto didático para estudar sistemas distribuídos e comunicação em tempo real.

Contudo, o projeto ainda não deve ser tratado como solução pronta para produção. As prioridades são autenticação e autorização por sessão, consentimento local, criptografia de transporte, direcionamento seletivo dos quadros, validação de comandos, persistência, auditoria e restauração de mecanismos de interrupção. Com essas melhorias e uma avaliação quantitativa de desempenho, o protótipo poderá evoluir para uma ferramenta de suporte remoto mais segura, eficiente e confiável.

Referências

EXPRESS.JS. 4.x API Reference. [S. l.], 2026. Disponível em: https://expressjs.com/en/4x/api/. Acesso em: 13 ago. 2026.

FETTE, I.; MELNIKOV, A. The WebSocket Protocol. RFC 6455. Internet Engineering Task Force, 2011. DOI: 10.17487/RFC6455. Disponível em: https://www.rfc-editor.org/info/rfc6455. Acesso em: 13 ago. 2026.

PYAUTOGUI. Cheat Sheet. [S. l.], 2026. Disponível em: https://pyautogui.readthedocs.io/en/latest/quickstart.html. Acesso em: 13 ago. 2026.

PYTHON-MSS. Examples. [S. l.], 2026. Disponível em: https://python-mss.readthedocs.io/latest/examples.html. Acesso em: 13 ago. 2026.

PYTHON-SOCKETIO. Getting Started. [S. l.], 2026. Disponível em: https://python-socketio.readthedocs.io/en/latest/intro.html. Acesso em: 13 ago. 2026.

RICHARDSON, T.; LEVINE, J. The Remote Framebuffer Protocol. RFC 6143. Internet Engineering Task Force, 2011. DOI: 10.17487/RFC6143. Disponível em: https://www.rfc-editor.org/info/rfc6143. Acesso em: 13 ago. 2026.

ROSE, S. et al. Zero Trust Architecture. Gaithersburg: National Institute of Standards and Technology, 2020. (NIST Special Publication 800-207). DOI: 10.6028/NIST.SP.800-207. Disponível em: https://csrc.nist.gov/pubs/sp/800/207/final. Acesso em: 13 ago. 2026.

SOCKET.IO. Introduction. Version 4. [S. l.], 2026. Disponível em: https://socket.io/docs/v4/. Acesso em: 13 ago. 2026.