

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
GOIÁS CAMPUS ITUMBIARA
ENGENHARIA DE CONTROLE E AUTOMAÇÃO**



Nathan Nickolas Carneiro Dias

**DESENVOLVIMENTO DE UM SISTEMA DE CONTROLE DE AR-CONDICIONADO
VIA INTERNET COM ESP32 E PLATAFORMA WEB RESPONSIVA**

ITUMBIARA/GO

2025



INSTITUTO FEDERAL
Goiás

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
SISTEMA INTEGRADO DE BIBLIOTECAS

TERMO DE AUTORIZAÇÃO PARA DISPONIBILIZAÇÃO NO REPOSITÓRIO DIGITAL DO IFG - ReDi IFG

Com base no disposto na Lei Federal nº 9.610/98, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia de Goiás, a disponibilizar gratuitamente o documento no Repositório Digital (ReDi IFG), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, em formato digital para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IFG.

Identificação da Produção Técnico-Científica

- | | |
|--|---|
| ☐ Tese | ☐ Artigo Científico |
| ☐ Dissertação | ☐ Capítulo de Livro |
| ☐ Monografia – Especialização | ☐ Livro |
| ☒ TCC - Graduação | ☐ Trabalho Apresentado em Evento |
| ☐ Produto Técnico e Educacional - Tipo: _____ | |

Nome Completo do Autor: Nathan Nickolas Carneiro Dias

Matrícula: 20181040100016

Título do Trabalho: Desenvolvimento de um Sistema de Controle de Ar-Condicionado via Internet com ESP32 e Plataforma Web Responsiva

Autorização - Marque uma das opções

1. (X) Autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso aberto);
2. () Autorizo disponibilizar meu trabalho no Repositório Digital do IFG somente após a data ____/____/____ (Embargo);
3. () Não autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso restrito).

Ao indicar a opção **2 ou 3**, marque a justificativa:

- () O documento está sujeito a registro de patente.
() O documento pode vir a ser publicado como livro, capítulo de livro ou artigo.
() Outra justificativa: _____

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O/A referido/a autor/a declara que:

- i. o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;
- ii. obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autor/a, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia de Goiás os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;
- iii. cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia de Goiás.

Itumbiara-GO, 28/08/2025.

Local Data



Documento assinado digitalmente
NATHAN NICKOLAS CARNEIRO DIAS
Data: 28/08/2025 18:43:44-0300
Verifique em <https://validar.iti.gov.br>

Assinatura do Autor e/ou Detentor dos Direitos Autorais

Nathan Nickolas Carneiro Dias

**DESENVOLVIMENTO DE UM SISTEMA DE CONTROLE DE AR-CONDICIONADO
VIA INTERNET COM ESP32 E PLATAFORMA WEB RESPONSIVA**

Trabalho de conclusão de curso apresentado a banca avaliadora, parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Controle e Automação junto ao Instituto Federal de Educação, Ciência e Tecnologia de Goiás (IFG), Câmpus Itumbiara.

Orientador: Bruno Gabriel Gustavo Leonardo Zambolini Vicente

ITUMBIARA/GO

2025

Dados Internacionais de Catalogação na Publicação (CIP)

D541 Dias, Nathan Nickolas Carneiro

Desenvolvimento de um sistema de controle de ar-condicionado via internet com esp32 e plataforma web responsiva. -- Itumbiara, 2025.
62 p. : il.

Trabalho de conclusão de curso (Bacharel em Engenharia de Controle e Automação) - Instituto Federal de Educação, Ciência e Tecnologia de Goiás, Câmpus Itumbiara, 2025.

Orientadora: Prof. Dr. Bruno Gabriel Gustavo Leonardo Zambolini Vicente.

1.Controle de processo. 2.Armazenamento. 3.Automação residencial. I. Título. II. Dias, Nathan Nickolas Carneiro. III. Vicente, Bruno Gabriel Gustavo Leonardo Zambolini. IV. Instituto Federal de Educação, Ciência e Tecnologia de Goiás.

CDD 621.39

Instituto Federal de Educação, Ciência e Tecnologia de Goiás
Biblioteca Maria Gabriela Pacheco Pardey / Câmpus Itumbiara
Bibliotecária: Marla de Souza Correia CRB-1/3011



INSTITUTO FEDERAL
Goiás

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS
CÂMPUS ITUMBIARA

NATHAN NICKOLAS CARNEIRO DIAS

**DESENVOLVIMENTO DE UM SISTEMA DE CONTROLE DE AR-CONDICIONADO VIA INTERNET COM ESP32 E
PLATAFORMA WEB RESPONSIVA**

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Engenharia Elétrica do Instituto Federal de Educação, Ciência e Tecnologia de Goiás – Câmpus Itumbiara, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Controle e Automação.

Aprovado em 21 de Agosto de 2025.

(Assinado Eletronicamente).

Prof. Dr. Bruno Gabriel Gustavo Leonardo Zambolini Vicente
Instituto Federal de Goiás, Câmpus Itumbiara
Orientador

(Assinado Eletronicamente).

Prof. Dr. Diogo Soares Resende
Instituto Federal de Goiás, Câmpus Itumbiara
Avaliador

(Assinado Eletronicamente)

Prof. Dr. Hugo Xavier Rocha
Instituto Federal de Goiás, Câmpus Itumbiara
Avaliador

Documento assinado eletronicamente por:

- **Hugo Xavier Rocha**, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 25/08/2025 09:03:35.
- **Diogo Soares Resende**, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 21/08/2025 15:33:22.
- **Bruno Gabriel Gustavo Leonardo Zambolini Vicente**, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 21/08/2025 12:23:31.

Este documento foi emitido pelo SUAP em 21/08/2025. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifg.edu.br/autenticar-documento/> e forneça os dados abaixo:

Código Verificador: 682945

Código de Autenticação: 47a3e7e4ab



Instituto Federal de Educação, Ciência e Tecnologia de Goiás
Avenida Furnas, nº 55, 55, Bairro Village Imperial, ITUMBIARA / GO, CEP 75524-010
(64) 2103-5622 (ramal: 5622)

AGRADECIMENTOS

Meu primeiro e maior agradecimento é aos meus pais, Celismar Dias da Silva e Celeste Maria Carneiro Rodrigues Dias, pelo amor incondicional, incentivo constante e por acreditarem no meu potencial mesmo nos momentos mais difíceis.

A minha irmã, Ana Cecilia Carneiro Dias, por sempre acreditar que um dia o mundo fosse ser meu.

Ao meu orientador, Bruno Gabriel Gustavo Leonardo Zambolini Vicente, pela dedicação, paciência, ensinamentos e contribuições preciosas durante o desenvolvimento deste trabalho.

A todos os professores, por seus ensinamentos, e a todos os colaboradores e funcionários que com seu trabalho muitas vezes invisível contribuíram para que eu chegasse até aqui.

RESUMO

O presente trabalho de graduação dedica-se à criação de um sistema de controle de ar-condicionado via IoT, onde emprega-se um microcontrolador ESP32, aliado a uma página web adaptável, visando o manuseio à distância e o acompanhamento constante. O projeto engloba a montagem de uma estrutura cliente-servidor, com o ESP32 servindo como central de comando, ligada à rede *Wi-Fi*, recebendo ordens e mandando informações de operação via HTTP. A inteligência contida no ESP32 é encarregada de ativar os relés que controlam o sistema de climatização, bem como de ler dados de sensores do ambiente, como a temperatura e a umidade. Os dados reunidos são tratados e enviados a um aplicativo feito com JAVA, que gera o fluxo de dados e a apresentação ao usuário. O MySQL é usado para guardar os dados, assegurando a sua durabilidade e a possibilidade de análise ao longo do tempo. A exibição dos parâmetros do sistema foi criada usando a ferramenta *React*, que possibilita um acompanhamento ininterrupto do funcionamento do ar-condicionado, além da criação de painéis interativos com indicadores importantes, como os períodos de atividade, as temperaturas medidas e o estado dos comandos. A página web foi criada com ênfase na adaptabilidade, assegurando o acesso tanto por celulares quanto por computadores.

Palavras chave: Controle de processos, Armazenamento, Monitoramento *Online*, Internet das Coisas, ESP32, Automação Residencial, HTTP, JAVA, Mysql, Controle de Climatização, Computação em Nuvem, *React*.

ABSTRACT

This undergraduate thesis focuses on creating an IoT-enabled air conditioning control system, utilizing an ESP32 microcontroller combined with an adaptive web page for remote operation and constant monitoring. The project involves assembling a client-server structure, with the ESP32 serving as a command center, connected to a Wi-Fi network, receiving commands and sending operational information via HTTP.

The intelligence contained within the ESP32 is responsible for activating the relays that control the air conditioning system, as well as reading data from environmental sensors, such as temperature and humidity. The collected data is processed and sent to a Java application, which generates the data stream and displays it to the user. MySQL is used to store the data, ensuring its durability and the ability to be analyzed over time. The system parameters display was created using the React tool, which allows for continuous monitoring of the air conditioning system's operation, in addition to creating interactive dashboards with important indicators, such as operating times, measured temperatures, and control status. The website was designed with an emphasis on adaptability, ensuring access from both mobile and desktop devices.

Keywords: Process Control, Data Storage, Online Monitoring, Internet of Things, ESP32, Home Automation, HTTP, Mysql, Climate Control, Cloud Computing, React.

LISTA DE ILUSTRAÇÕES

Figura 1 - Esquema com funcionamento da malha aberta e fechada	13
Figura 2 - Representação de camadas da arquitetura IoT	17
Figura 3- Código spring-boot-starter-security	27
Figura 4- Código springdoc-openapi-starter-webmvc-ui.....	27
Figura 5-Código spring-boot-starter-web.....	28
Figura 6-Mysql-connector-j.....	28
Figura 7 -Código lombok	28
Figura 8 - Entidade de gerenciamento do ar	29
Figura 9- Código de uma query para exemplificar o conteúdo do repository	30
Figura 10- Transmissor infravermelho.....	30
Figura 11-ESP 32.....	30
Figura 12- Emissor infravermelho KY-005.....	31
Figura 13-Código do ESP32 para a conexão com WIFI;.....	31
Figura 14-Variáveis onde vão as credenciais da rede wifi.....	31
Figura 15- ESP-32 e receptor infravermelho	32
Figura 16 - Dashboard.....	34
Figura 17 - Layout visual	35
Figura 18 - Tela do gerenciamento de condicionadores de ar de uma sala.	36
Figura 19 - Tela de reserva.	36
Figura 20 - Planilha de reserva.	37
Figura 21 - Tela de configurações.....	37
Figura 22 - Botão para inserir planilha.....	38
Figura 23 - Modelo de planilha	38
Figura 24 - Botão verde responsável por fazer download do código.....	39
Figura 25 - Endpoints responsáveis pela renderização do dashboard.....	39
Figura 26 - Endpoints responsáveis pela exclusão de uma sala.....	41
Figura 27 - Endpoints responsáveis pelo controle dos ares.....	42
Figura 28 - Endpoint que cadastra os ares-condicionados.....	43
Figura 29 - Endpoints para gerenciar as reservas.....	44
Figura 30 - Agenda para desligar os condicionadores de ar.	46
Figura 31 - Agenda para ligar os ares-condicionados.	47

Figura 32 - Endpoints da aba de configurações.	47
Figura 33 - Código para capturar o infravermelho do controle.	51
Figura 34 - Código para usar o goodweather infravermelho.	52
Figura 35 - Estrutura para gerar o infravermelho do controle.....	52
Figura 36 - Função para gerar o infravermelho do controle.	53
Figura 37- Ar condicionado utilizado nos testes	58
Figura 38- Protótipo do hardware completo montado.	59

SUMÁRIO

CAPÍTULO I - INTRODUÇÃO	11
1.1 OBJETIVOS	12
CAPÍTULO II - REFERENCIAL TEÓRICO.....	13
1.2 SISTEMAS DE CONTROLE E AUTOMAÇÃO	13
1.3 SISTEMAS DE CONTROLE DE CLIMATIZAÇÃO	14
1.4 TECNOLOGIAS DE COMUNICAÇÃO	15
1.5 INTERNET DAS COISAS (IoT)	16
1.6 ARQUITETURA WEB MODERNA	18
1.7 FRONT-END: REACT E JAVASCRIPT	18
1.8 BACK-END: JAVA E APIS RESTFUL	19
1.9 MICROCONTROLADORES PARA IoT: ESP32.....	20
1.10 BANCO DE DADOS RELACIONAIS: MYSQL	21
1.11 SISTEMAS SUPERVISÓRIOS (SCADA E HMI WEB).....	22
1.12 FERRAMENTAS DE DESENVOLVIMENTO (IDES)	23
1.13 GERENCIAMENTO DE CÓDIGO-FONTE : GITHUB	25
CAPÍTULO III - METODOLOGIA	26
1.14 BACK-END	26
1.15 TRANSMISSOR IR+BACK-END.....	30
1.16 FRONT-END	34
1.17 FRONT-END + BACK-END.....	39
1.18 TESTES.....	49
3.5.1 METODOLOGIA DE TESTES.....	49
3.5.2 TESTES DA CAMADA DE INTEGRAÇÃO (API E FRONTEND)	49
3.5.3 TESTES DO CONTROLADOR ESP32 E SOLUÇÃO IR ADICIONAL	50
3.5.4 CONCLUSÃO DOS TESTES.....	53
CAPÍTULO IV – RESULTADOS E DISCUSSÃO.....	55
4.1 DESENVOLVIMENTO DO <i>FIRMWARE</i> PARA O MICROCONTROLADOR ESP32	55

4.2 CRIAÇÃO DA INTERFACE WEB RESPONSIVA	55
4.3 INTEGRAÇÃO DO ESP32 COM JAVA COM SPRING BOOT WEB.....	56
4.4 IMPLEMENTAÇÃO DO BANCO DE DADOS TEMPORAL (MYSQL)	57
4.5 CONFIGURAÇÃO DO FRONT COM REACT PARA VISUALIZAÇÃO EM TEMPO REAL	57
4.6 AVALIAÇÃO DO FUNCIONAMENTO DO SISTEMA EM AMBIENTE REAL 58	
CAPÍTULO VI – CONCLUSÃO.	61

CAPÍTULO I - INTRODUÇÃO

Manter a temperatura adequada em ambientes internos é essencial para garantir o conforto térmico, a saúde dos ocupantes e a eficiência na realização de atividades cotidianas, tanto em ambientes residenciais quanto comerciais. O controle térmico eficiente reduz desconfortos fisiológicos, como fadiga e desidratação, e melhora a concentração e o desempenho de tarefas, sendo especialmente relevante em locais de trabalho. Em regiões de clima predominantemente quente, como o Brasil, o uso de sistemas de climatização, sobretudo o ar-condicionado, tornou-se indispensável.

No entanto, a operação manual e desregulada desses equipamentos pode acarretar um consumo elevado de energia elétrica, contribuindo para o desperdício de recursos naturais e para o aumento dos custos operacionais. Além disso, o uso excessivo ou inadequado dos sistemas de climatização está associado a um desgaste prematuro dos componentes, reduzindo a vida útil dos aparelhos e aumentando a necessidade de manutenção corretiva (MEDEIROS et al., 2020).

Com o avanço das tecnologias relacionadas à Internet das Coisas (IoT) e à automação de ambientes, surgiram alternativas mais inteligentes e sustentáveis para o controle de climatização. A integração de sensores ambientais, microcontroladores programáveis e interfaces remotas permite o monitoramento contínuo de variáveis como temperatura e umidade, possibilitando o acionamento automatizado dos sistemas de climatização apenas quando necessário. Essa abordagem não apenas melhora o conforto térmico, como também otimiza o uso energético (SILVA et al., 2020).

A economia de energia se dá, principalmente, pela redução do tempo de funcionamento desnecessário dos aparelhos, pelo ajuste dinâmico da climatização conforme a ocupação e pelas respostas inteligentes às condições do ambiente. De acordo com Silva et al. (2020), sistemas automatizados de climatização podem reduzir o consumo energético em até 30%, quando comparados a sistemas operados manualmente. Essa economia impacta diretamente o orçamento familiar e corporativo, além de contribuir para a mitigação dos impactos ambientais causados pela geração de energia elétrica, que é em parte ainda baseada em fontes não renováveis (ANEEL, 2022).

Do ponto de vista ambiental, o uso consciente da energia reduz a emissão de gases de efeito estufa, contribuindo para o combate às mudanças climáticas (IPCC, 2021). Sob a perspectiva social, torna-se possível democratizar o acesso ao conforto térmico com responsabilidade e sustentabilidade, promovendo qualidade de vida sem comprometer os recursos das futuras gerações (OLIVEIRA e NASCIMENTO, 2019).

1.1 OBJETIVOS

O principal objetivo deste trabalho é desenvolver e avaliar um sistema integrado e inteligente para o controle remoto de ar-condicionado via web, utilizando tecnologias como ESP32, HTTP, MySQL, Java, *Spring boot web* e *React*, no contexto da Internet das Coisas (IoT) aplicada à automação residencial. A solução visa permitir o monitoramento, controle à distância e análise em tempo real dos parâmetros ambientais, com foco na otimização do uso energético e no conforto do usuário. Além disso, busca-se analisar a viabilidade, os custos, a escalabilidade e a experiência do usuário com a implementação do sistema em um ambiente real.

- Desenvolver o *firmware* para o microcontrolador ESP32, habilitando a comunicação via Wi-Fi e o envio de comandos infravermelhos para o ar-condicionado.
- Criar uma interface *web* responsiva, acessível via dispositivos móveis e *desktops*, para a interação e controle do sistema.
- Integrar o ESP32 com o Java com *Spring boot web* para gerenciar o fluxo de dados e orquestrar as comunicações entre os dispositivos e as aplicações *web*.
- Implementar um banco de dados temporal, utilizando MySQL, para o armazenamento e consulta histórica dos dados ambientais e de operação.
- Desenvolver um *front-end* com *React* para visualização em tempo real dos dados, oferecendo *dashboards* interativos e indicadores de desempenho do sistema.
- Avaliar o funcionamento do sistema em um ambiente real, medindo a estabilidade da comunicação, a usabilidade da interface e o impacto na economia de energia.

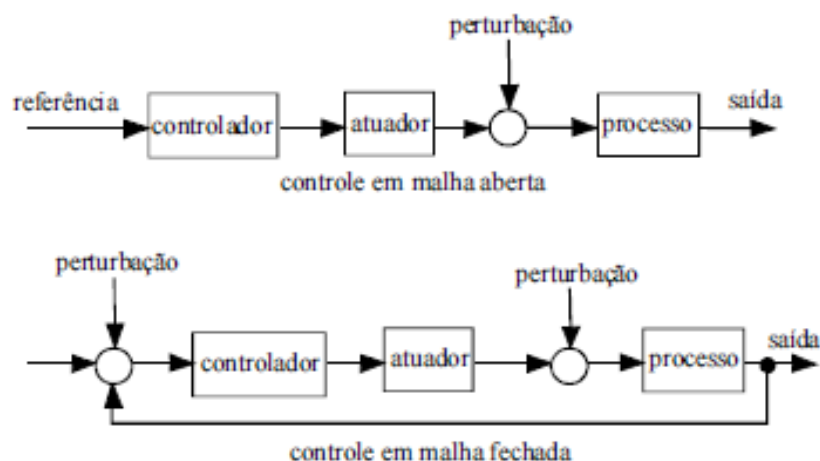
CAPÍTULO II - REFERENCIAL TEÓRICO

1.2 SISTEMAS DE CONTROLE E AUTOMAÇÃO

Sistemas de controle são a base da automação, definidos como arranjos de componentes físicos que atuam de maneira coordenada para gerenciar, direcionar ou regular o comportamento de outros sistemas ou processos, visando alcançar um objetivo predefinido (OGATA, 2010). A funcionalidade central de um sistema de controle reside na sua capacidade de medir a saída de um processo, compará-la com um valor de referência (*setpoint*) e, se houver diferença, gerar um sinal de controle para ajustar a entrada do processo, minimizando o erro.

Esses sistemas são classificados primariamente como de malha aberta, nos quais a saída não influencia a ação de controle e a ação de controle é predeterminada, sendo um exemplo simples um temporizador que desliga um aparelho de ar-condicionado após um tempo fixo, independentemente da temperatura alcançada no ambiente. Embora simples, a ausência de *feedback* os torna vulneráveis a perturbações e imprecisões. Em contraste, os sistemas de malha fechada (com *feedback*) são caracterizados pela medição da saída por um sensor, que é então comparada com o valor desejado. A diferença (erro) é utilizada para ajustar a entrada do controlador, que atua no processo para corrigir o desvio (BOLTON, 2015). Para o controle de ar-condicionados, um sistema de malha fechada seria o mais adequado, empregando sensores de temperatura para monitorar o ambiente e ajustar continuamente a operação do ar até que a temperatura desejada seja alcançada e mantida, conferindo maior robustez e precisão.

Figura 1 - Esquema com funcionamento da malha aberta e fechada



Groover (2014) destaca que a automação consiste no emprego de tecnologias para executar tarefas de forma autônoma ou semiautônoma, com mínima ou nenhuma intervenção humana, visando otimizar processos, aumentar a eficiência, melhorar a segurança e reduzir custos operacionais. No contexto de edifícios, a automação predial (*Building Automation Systems - BAS*) integra e gerencia diversos subsistemas, como climatização, iluminação, segurança, controle de acesso e detecção de incêndio. A automação da climatização é de suma importância para a eficiência energética, pois permite a programação de operação alinhada com a ocupação dos ambientes, o controle proporcional que ajusta a potência e o modo de operação com base na temperatura e no *setpoint*, a integração com sensores de presença para desligar aparelhos em salas vazias, e o monitoramento remoto para acompanhar o status e o consumo dos equipamentos a distância, possibilitando ações rápidas em caso de falhas ou ineficiências.

1.3 SISTEMAS DE CONTROLE DE CLIMATIZAÇÃO

Os sistemas de climatização têm como objetivo principal garantir o conforto térmico e a qualidade do ambiente interno, controlando parâmetros como temperatura, umidade, qualidade do ar e ventilação. A história desses sistemas remonta ao início do século XX, com o desenvolvimento do ar-condicionado moderno por Willis Carrier em 1902, que revolucionou ambientes comerciais e industriais ao permitir o controle preciso das condições ambientais (CARRIER, 2009).

Os sistemas de climatização são geralmente agrupados na categoria HVAC (*Heating, Ventilation and Air Conditioning*), que abrange três funções essenciais:

aquecimento, ventilação e condicionamento do ar. Cada uma dessas funções pode ser implementada por diferentes tecnologias e estratégias de controle, dependendo das necessidades do ambiente e do uso pretendido (ASHRAE, 2017). Sistemas HVAC modernos utilizam desde unidades compactas residenciais até instalações complexas para grandes edifícios comerciais e industriais.

A automação dos sistemas de climatização é fundamental para aprimorar sua eficiência energética, reduzir custos operacionais e melhorar o conforto dos usuários. Com a automação, é possível monitorar e controlar remotamente as variáveis ambientais, além de ajustar o funcionamento dos equipamentos conforme a demanda real, evitando o desperdício de energia (SILVA et al., 2020). Além disso, a integração com tecnologias de Internet das Coisas (IoT) e sistemas embarcados permite a implementação de estratégias avançadas, como o controle preditivo e a manutenção preditiva, ampliando a confiabilidade e a vida útil dos equipamentos (ZHANG; WANG; LI, 2019).

A importância da automação em sistemas HVAC também se reflete em aspectos ambientais, financeiros e sociais, já que contribui para a redução do consumo energético e das emissões de gases de efeito estufa, além de melhorar a qualidade de vida dos ocupantes dos ambientes controlados (OLIVEIRA; NASCIMENTO, 2019).

1.4 TECNOLOGIAS DE COMUNICAÇÃO

As tecnologias de comunicação são essenciais para sistemas de automação e Internet das Coisas (IoT), permitindo a troca eficiente de dados entre dispositivos e plataformas. Protocolos como HTTP e MQTT são amplamente utilizados: o HTTP funciona por requisição/resposta, sendo comum em aplicações *web*, enquanto o MQTT, baseado em *publish/subscribe*, é leve e ideal para dispositivos com recursos limitados, comuns em IoT (LIGHT, 2017; FIELDING et al., 1999).

Redes *Wi-Fi* e *Ethernet* são as principais tecnologias físicas para conectar dispositivos, oferecendo alta velocidade e confiabilidade. Outras tecnologias, como Zigbee e LoRaWAN, ganham destaque em cenários que exigem baixo consumo de energia e cobertura estendida (GONÇALVES; SILVA, 2020).

Além das redes *Wi-Fi* e *Ethernet*, a comunicação por infravermelho (IR) é amplamente empregada no controle remoto de dispositivos eletrônicos, incluindo sistemas de ar-condicionado. A comunicação IR é simples, de curto alcance e

direcionada, tornando-a eficiente para comandos pontuais em ambientes residenciais e comerciais (SANTOS; OLIVEIRA, 2018).

A escolha e integração adequadas dessas tecnologias garantem sistemas de automação eficientes, seguros e escaláveis, alinhados com as demandas da Indústria 4.0 e automação residencial inteligente (ATZORI; IERA; MORABITO, 2010).

1.5 INTERNET DAS COISAS (IoT)

A Internet das Coisas (IoT) representa a extensão da conectividade da internet para objetos físicos cotidianos, permitindo que esses objetos colem e troquem dados sem a necessidade de interação humana direta (ASHTON, 2009). Este paradigma transforma "coisas" comuns em dispositivos inteligentes e conectados, capazes de monitorar ambientes, interagir com o mundo físico e se comunicar entre si e com sistemas maiores através da rede (ATZORI; IERA; MORABITO, 2010).

Uma arquitetura comum para sistemas IoT geralmente compreende quatro camadas interdependentes. A Camada de Percepção é composta pelos dispositivos físicos, como sensores e atuadores, e microcontroladores (a exemplo do ESP32) que coletam dados do ambiente e executam ações. A Camada de Rede é responsável pela transmissão desses dados para a camada de processamento e vice-versa, englobando protocolos de comunicação como *Wi-Fi*, MQTT e HTTP, e toda a infraestrutura de rede. A Camada de Processamento é onde os dados brutos são tratados, armazenados e analisados, envolvendo servidores de *back-end*, bancos de dados e plataformas de nuvem (*Cloud Computing*) que oferecem serviços de análise e inteligência. Finalmente, a Camada de Aplicação é a interface final para o usuário, permitindo a interação com os dados e o controle dos dispositivos por meio de aplicações *web*, móveis ou *desktop*, sendo nela que o sistema supervisor *web* deste projeto se insere.

Figura 2 - Figura 2 - Representação de camadas da arquitetura IoT



Adaptado de Santana et al (2019)

No que tange aos protocolos de comunicação em IoT, o HTTP (*Hypertext Transfer Protocol*), amplamente utilizado na web, opera sobre TCP/IP no modelo de requisição-resposta. Embora robusto, pode ser considerado "pesado" para dispositivos com recursos limitados ou para comunicação em tempo real devido ao seu modelo *pull*, onde o cliente inicia a solicitação. Contudo, é perfeitamente adequado para a comunicação síncrona entre o ESP32 e o *back-end* em Java para o envio de comandos e atualização de status. Alternativamente, o MQTT (Message Queuing Telemetry Transport) é um protocolo de mensagens leve, baseado no padrão *publish/subscribe*, ideal para dispositivos com largura de banda limitada e alta latência (HOPA; MEKKI; AHMED; JURDI; CONTI, 2016). Embora não seja o principal protocolo da proposta inicial, sua flexibilidade para envio de telemetria e comandos em tempo real pode ser uma consideração valiosa para futuras expansões do sistema.

1.6 ARQUITETURA WEB MODERNA

A arquitetura web moderna se distancia dos modelos monolíticos tradicionais, favorecendo a modularidade e a distribuição, o que proporciona maior escalabilidade, flexibilidade e facilidade de manutenção. A clara separação entre *front-end* e *back-end* é um pilar dessa abordagem.

1.7 FRONT-END: REACT E JAVASCRIPT

O *Front-end*, ou lado do cliente, engloba tudo o que o usuário vê e com o qual interage diretamente em um aplicativo web. É a camada responsável pela interface do usuário (UI) e pela experiência do usuário (UX), construída fundamentalmente com HTML para estrutura, CSS para estilização e JavaScript para interatividade.

JavaScript, uma linguagem de programação de alto nível, interpretada, multiparadigma e padronizada pelo ECMAScript, é a espinha dorsal do desenvolvimento web no lado do cliente (FLANAGAN, 2011). Sua natureza assíncrona e a vasta gama de APIs do navegador a tornam essencial para a criação de funcionalidades dinâmicas e interativas, desde a manipulação do DOM e validação de formulários até a comunicação assíncrona (AJAX) com o *back-end*, sem recarregar a página, proporcionando experiências de usuário ricas e fluidas.

O *React*, uma biblioteca JavaScript desenvolvida pelo Facebook, é utilizada para construir interfaces de usuário (UIs) declarativas e baseadas em componentes. Seus princípios fundamentais incluem a divisão da UI em blocos independentes e reutilizáveis (componentes), que encapsulam sua própria lógica e renderização, promovendo modularidade e manutenibilidade. Para otimizar o desempenho, o *React* emprega um *Virtual DOM*, uma representação em memória da UI que, ao haver uma mudança de estado, compara-se com o estado anterior para calcular e aplicar apenas as diferenças mínimas no DOM real, resultando em interfaces mais rápidas e fluidas. Adota, tipicamente, um fluxo de dados unidirecional (de componentes pais para filhos), o que simplifica o rastreamento e a depuração de *bugs*. Além disso, o JSX, uma extensão de sintaxe para JavaScript, permite escrever estruturas de UI semelhantes a HTML diretamente no código JavaScript, facilitando a criação e visualização de componentes. A escolha do *React* para o sistema supervisorio web do IFG é justificada pela necessidade de uma interface de usuário dinâmica, que possa exibir

o status em tempo real dos ares-condicionados e reagir rapidamente aos comandos do usuário, oferecendo uma experiência de uso intuitiva e eficiente.

1.8 BACK-END: JAVA E APIS RESTFUL

O *Back-end*, ou lado do servidor, é o cérebro da aplicação, responsável por gerenciar a lógica de negócio, interagir com o banco de dados e fornecer dados para o *front-end* e outros serviços. A comunicação entre o *front-end* e o *back-end* geralmente ocorre por meio de APIs (Application Programming Interfaces).

Java, uma linguagem de programação orientada a objetos (POO), de alto desempenho, robusta e conhecida por seu princípio "*Write Once, Run Anywhere*" (WORA) através da *Java Virtual Machine* (JVM), é amplamente utilizada em sistemas de grande escala e aplicações corporativas (GOSLING et al., 2014). Sua maturidade e vasto ecossistema de bibliotecas e *frameworks* a tornam uma escolha poderosa para *back-ends* escaláveis. Entre esses, o Spring Boot se destaca como um *framework* popular do ecossistema Java que simplifica enormemente o desenvolvimento de aplicações *back-end*, especialmente microsserviços e APIs *RESTful* (SPRING, [s.d.]). Ele adota a "convenção sobre configuração", o que reduz significativamente a quantidade de código *boilerplate* e permite que os desenvolvedores se concentrem na lógica de negócio. Componentes como *Spring Data JPA* facilitam a interação com bancos de dados relacionais, enquanto *Spring Web* (anteriormente *Spring MVC*) fornece ferramentas robustas para construir *endpoints* REST. Complementarmente, o *Project Lombok* (LOMBOK, [s.d.]) é uma biblioteca Java que atua no processo de compilação, gerando automaticamente código repetitivo (como *getters*, *setters*, construtores, métodos *equals()* e *hashCode()*) a partir de anotações simples. Isso resulta em um código mais limpo, conciso e legível, eliminando a necessidade de escrever manualmente centenas de linhas de código padrão e reduzindo a chance de erros. A combinação do *Spring Boot* com Lombok é uma prática comum em projetos Java modernos, pois juntos eles otimizam drasticamente o processo de desenvolvimento, tornando-o mais rápido e menos propenso a falhas, ao mesmo tempo em que mantêm a robustez e a escalabilidade necessárias para o sistema de controle de ares-condicionados.

A comunicação entre o *front-end*, o *back-end* e, em certos casos, o ESP32 (para o *back-end* enviar comandos ao dispositivo), será baseada em APIs *REST* (*Representational State Transfer*). *REST* é um estilo arquitetural para sistemas distribuídos, que define um conjunto de restrições para a comunicação entre componentes (FIELDING; TAYLOR, 2002). No *REST*, tudo é tratado como um recurso, identificado por uma URL (*Uniform Resource Locator*) única, como */api/ares-condicionados* para listar todos os aparelhos ou */api/ares-condicionados/{id}* para um aparelho específico. As operações sobre esses recursos são realizadas utilizando os métodos padrão do protocolo HTTP, ou end-points: *GET* para buscar dados, *POST* para criar novos recursos, adicionar um aparelho ou enviar um comando), *PUT* para substituição completa e *PATCH* para atualização parcial de recursos existentes, e *DELETE* para remover. Cada requisição do cliente ao servidor deve ser stateless, contendo todas as informações necessárias para o processamento, sem depender de sessões anteriores, o que melhora a escalabilidade do *back-end*. Os recursos são trocados em formatos como JSON (*JavaScript Object Notation*) ou XML, facilitando a interoperabilidade. O *back-end* em Java, com Spring Boot e o auxílio do Lombok, atuará como o orquestrador central, processando requisições do *front-end*, interagindo com o banco de dados MySQL e enviando comandos relevantes aos dispositivos ESP32.

1.9 MICROCONTROLADORES PARA IOT: ESP32

Microcontroladores são dispositivos semicondutores que integram em um único chip um processador (CPU), memória (*RAM* e *ROM/Flash*) e periféricos de entrada/saída (GPIOs, interfaces de comunicação como UART, SPI, I2C, ADC, DAC), projetados para sistemas embarcados e controle de hardware (MAZIDI; MAZIDI; MCKINLAY, 2007). Eles são o "cérebro" de muitos dispositivos IoT, executando software específico para interagir com sensores e atuadores.

O ESP32, um *System-on-Chip* (SoC) desenvolvido pela Espressif Systems, destaca-se por sua conectividade *Wi-Fi* (IEEE 802.11 b/g/n) e *Bluetooth* (BLE e *Classic*), além de apresentar baixo custo e baixo consumo de energia. É uma escolha robusta para projetos de IoT e automação (ESPRESSIF, [s.d.]). Suas principais características incluem um processador *dual-core* Tensilica Xtensa LX6, oferecendo

poder de processamento para tarefas complexas como processamento de dados de sensores, gerenciamento de conectividade e execução de lógica de controle. Possui memória *RAM* e *Flash* internas para armazenamento do *firmware* e dados, e um grande número de GPIOs configuráveis para diversas funções, como PWM e interfaces de comunicação serial. Crucialmente para este projeto, o ESP32 tem a capacidade de gerar e detectar sinais infravermelhos (IR), permitindo a emulação de códigos de controles remotos existentes para o controle de ares-condicionados legados. A conectividade Wi-Fi do ESP32 é o pilar da comunicação no seu projeto, possibilitando que o dispositivo se conecte à rede local do IFG no modo *station* e se comunique com o Back-end Java via HTTP ou outros protocolos. Pode ser programado utilizando a IDE do Arduino (com bibliotecas ESP32) ou o ESP-IDF, que oferece maior controle para aplicações complexas em C/C++. A integração do ESP32 é vital, atuando como a "ponte" entre o mundo físico (ares-condicionados) e o mundo digital (plataforma web), recebendo comandos do Back-end, traduzindo-os em ações físicas e, eventualmente, enviando dados de status ou de sensores de volta ao *back-end*.

1.10 BANCO DE DADOS RELACIONAIS: MYSQL

Sistemas de Gerenciamento de Banco de Dados (SGBD) são *softwares* que permitem a criação, manipulação e gerenciamento de bancos de dados. Entre eles, os Bancos de Dados Relacionais (BDs Relacionais) organizam os dados em estruturas tabulares, onde cada tabela representa uma entidade e é composta por linhas (registros ou tuplas) e colunas (atributos ou campos) (DATE, 2003). A interconexão entre tabelas é feita através de chaves primárias (identificadores únicos de um registro) e chaves estrangeiras (colunas que referenciam chaves primárias em outras tabelas), garantindo a integridade referencial dos dados.

O MySQL é um dos SGBDRs de código aberto mais populares do mundo, reconhecido por sua alta performance, robustez, escalabilidade e facilidade de administração (DUBOIS, 2014). Sua ampla adoção em aplicações web deve-se à sua arquitetura cliente-servidor eficiente e ao suporte a uma vasta gama de linguagens de programação. Para o sistema de controle de ares-condicionados, o MySQL será fundamental para o armazenamento de metadados dos aparelhos (ID único,

localização, tipo, fabricante e ID do ESP32 associado), o histórico de comandos e eventos (quem enviou, quando, qual comando, status de execução), o gerenciamento de usuários (incluindo credenciais e níveis de permissão) e, opcionalmente, mas recomendado, dados de telemetria de sensores para monitoramento e análise. Se o sistema incluir agendamento de tarefas (ligar/desligar), esses dados também serão persistidos no banco. A interação entre o *back-end* Java e o MySQL será realizada através do JDBC (*Java Database Connectivity*), uma API padrão do Java para acesso a SGBDRs, sendo que *frameworks* como Spring Data JPA simplificam essa interação, mapeando objetos Java para tabelas de banco de dados e reduzindo a complexidade do código de acesso a dados. Para auxiliar na gestão e interação direta com o banco de dados, o DBeaver emerge como uma ferramenta cliente universal. O DBeaver é um software de gerenciamento de banco de dados SQL multiplataforma e de código aberto que suporta uma ampla gama de sistemas de banco de dados, incluindo MySQL (DBeaver, [s.d.]). Ele oferece uma interface gráfica unificada para desenvolvedores, administradores de banco de dados e analistas de dados, permitindo a execução de consultas SQL, gerenciamento de objetos de banco de dados (tabelas, índices, *views*, procedimentos armazenados), importação/exportação de dados, e monitoramento de desempenho. Sua versatilidade e conjunto de recursos o tornam uma ferramenta valiosa para o desenvolvimento e manutenção do banco de dados deste projeto.

1.11 SISTEMAS SUPERVISÓRIOS (SCADA E HMI WEB)

Sistemas supervisórios, frequentemente categorizados como SCADA (*Supervisory Control e Data Acquisition*), são arquiteturas de controle industrial projetadas para monitorar e controlar processos remotos ou distribuídos. Eles coletam dados de campo através de RTUs (*Remote Terminal Units*) ou PLCs (*Programmable Logic Controllers*), transmitem-nos para uma estação central, onde são processados, visualizados em interfaces gráficas (IHM - Interface Homem-Máquina) e, com base nesses dados, permitem que operadores emitam comandos de controle (GELMAN, 2002).

No contexto da automação predial e, especificamente, para o controle de climatização, a implementação de um sistema supervisório via plataforma web oferece

vantagens significativas em comparação aos sistemas *scada* tradicionais. A principal delas é a acessibilidade universal, pois, diferentemente de iHMs dedicadas, uma interface web pode ser acessada de qualquer dispositivo (computador, *tablet*, *smartphone*) com um navegador e conexão à internet, eliminando a necessidade de instalações de *software* cliente e reduzindo custos. Além disso, a facilidade de manutenção e atualização é um ponto forte, uma vez que as atualizações são centralizadas no servidor e imediatamente acessíveis a todos os usuários. No que tange à experiência do usuário (UX¹), a escolha de tecnologias *front-end* modernas, como o *react*, é crucial. Com elas, torna-se plenamente possível desenvolver interfaces altamente interativas, que respondem dinamicamente às ações do usuário e se adaptam a diferentes tamanhos de tela. Tais interfaces não apenas facilitam a visualização em tempo real do status dos ar-condicionados, mas também simplificam o envio de comandos e a análise intuitiva de dados históricos, contribuindo para uma interação eficiente e agradável com o sistema. Finalmente, um sistema web bem arquitetado demonstra escalabilidade, podendo ser facilmente expandido para gerenciar um número crescente de dispositivos e usuários. A interface gráfica do sistema supervisor web permitirá que os usuários visualizem o *layout* das salas do ifg, o status operacional de cada ar-condicionado (ligado/desligado, temperatura atual, modo de operação), e enviem comandos com um clique.

1.12 FERRAMENTAS DE DESENVOLVIMENTO (IDES)

Para o desenvolvimento de software, os Ambientes de Desenvolvimento Integrados (IDES – *Integrated Development Environment*) desempenham um papel fundamental, oferecendo um conjunto de ferramentas que otimizam o processo de codificação, depuração e gerenciamento de projetos. Uma IDE geralmente integra um editor de código-fonte, ferramentas de automação de *build* e um depurador. A escolha da IDE correta pode impactar significativamente a produtividade do desenvolvedor, a qualidade do código e a facilidade de colaboração. Neste projeto, a programação do *back-end*, *front-end* e *firmware* embarcado se beneficiou de IDEs específicas, cada uma com suas particularidades e pontos fortes.

¹ UX: *User eXperience*

O IntelliJ IDEA, desenvolvido pela *JetBrains*, é uma IDE comercialmente líder e altamente conceituada para desenvolvimento Java, conhecida por sua inteligência de código superior, análise estática de código e recursos de refatoração (JETBRAINS, [s.d.]). Lançado inicialmente em 2001, o *IntelliJ* evoluiu para oferecer suporte robusto a uma vasta gama de linguagens e *frameworks*, incluindo o ecossistema Spring (com excelente integração para Spring Boot), Maven/Gradle, Git e bancos de dados. Sua capacidade de autocompletar código, detectar erros em tempo real e fornecer sugestões contextuais acelera o desenvolvimento Java. Para este projeto, o IntelliJ IDEA foi a ferramenta principal para a construção do *back-end* em Java com Spring Boot e Lombok, otimizando a escrita de APIs RESTful e a interação com o banco de dados MySQL, graças às suas poderosas ferramentas de depuração e ao suporte nativo para persistência de dados.

O Visual Studio Code (VS Code), da Microsoft, é um editor de código-fonte leve, porém extremamente poderoso, que se tornou uma das ferramentas mais populares entre desenvolvedores de diversas linguagens (MICROSOFT, [s.d.]). Lançado em 2015, o VS Code é de código aberto, multiplataforma e extensível, permitindo que a comunidade contribua com uma vasta gama de extensões que adicionam suporte a novas linguagens, *debuggers* e ferramentas. Sua flexibilidade o torna ideal para o desenvolvimento *front-end* com *React* e JavaScript, oferecendo recursos como autocompletar inteligente (IntelliSense), depuração integrada para JavaScript, integração com *Git* e um terminal embutido. Além disso, com extensões adequadas, o VS Code pode ser configurado para auxiliar no desenvolvimento de *firmware* para microcontroladores como o ESP32, permitindo a escrita de código em C/C++ e até mesmo a interface com as ferramentas de *build* do Arduino IDE ou ESP-IDF, embora a compilação e o *upload* para o dispositivo pudessem ser realizados externamente ou através de integrações específicas. Sua capacidade de gerenciar múltiplos projetos e arquivos de diferentes tecnologias em um único ambiente o torna versátil para o projeto.

Finalmente, a IDE do Arduino é o ambiente de desenvolvimento padrão e oficial para a plataforma Arduino, projetada para ser simples e fácil de usar, especialmente para iniciantes em programação de microcontroladores (ARDUINO, [s.d.]). Lançada em 2005, ela permite a escrita de código em uma variação de C/C++ e facilita a compilação, o upload para a placa e o monitoramento serial. Embora mais básica em

comparação com o IntelliJ ou VS Code, a IDE do Arduino é fundamental para o desenvolvimento e depuração do *firmware* do ESP32. Ela simplifica a interação com as bibliotecas específicas do ESP32 para conectividade Wi-Fi e controle de infravermelho, sendo a principal ferramenta para programar e testar a lógica de controle diretamente no hardware do microcontrolador. Sua interface intuitiva e a vasta comunidade de suporte são vantagens significativas para o desenvolvimento embarcado deste projeto.

A combinação dessas três IDEs garante um fluxo de trabalho otimizado e eficiente, aproveitando as especializações de cada ferramenta para as respectivas partes do sistema: IntelliJ para o *back-end* robusto em Java, VS Code para a interface *front-end* moderna e responsiva, e a IDE do Arduino para a programação de baixo nível e interação direta com o hardware embarcado do ESP32.

1.13 GERENCIAMENTO DE CÓDIGO-FONTE : GITHUB

O gerenciamento de código-fonte é uma prática essencial no desenvolvimento de software, especialmente em projetos que envolvem múltiplas tecnologias e, potencialmente, colaboração. Ferramentas de controle de versão permitem rastrear alterações no código, reverter para versões anteriores, mesclar contribuições de diferentes desenvolvedores e manter um histórico completo do projeto.

O Git é um sistema de controle de versão distribuído, criado por Linus Torvalds em 2005, que se tornou o padrão da indústria para o rastreamento de mudanças em qualquer conjunto de arquivos (GIT, [s.d.]). Sua natureza distribuída significa que cada desenvolvedor tem uma cópia completa do histórico do repositório, permitindo que trabalhem offline e sincronizem suas alterações quando conveniente. O Git facilita a colaboração através de funcionalidades como *branching* (ramificação), que permite que diferentes funcionalidades ou correções de *bugs* sejam desenvolvidas isoladamente e, em seguida, integradas ao código principal (*merge*).

O GitHub é a maior plataforma de hospedagem de código-fonte e colaboração do mundo, baseada no Git (GITHUB, [s.d.]). Lançado em 2008, ele oferece uma interface web para gerenciamento de repositórios Git, facilitando a criação de projetos, o compartilhamento de código, a revisão de código (*pull requests*), o rastreamento de

issues (problemas e tarefas) e a automação de fluxos de trabalho (GitHub Actions). Para este projeto de TCC, o GitHub é uma ferramenta indispensável para:

- Armazenamento centralizado: Todos os códigos-fonte do projeto (Back-end Java, Front-end React/JavaScript e *firmware* ESP32) podem ser armazenados em um ou mais repositórios no GitHub, garantindo que o código esteja sempre seguro e acessível de qualquer lugar.
- Controle de versão: Cada alteração feita no código é registrada, permitindo que o histórico de desenvolvimento seja rastreado, que se volte a versões anteriores se necessário e que se identifique quando e por que certas modificações foram implementadas.
- Colaboração: Embora este possa ser um projeto individual, o GitHub oferece a infraestrutura para futuras colaborações, caso o projeto evolua ou envolva outros membros. A funcionalidade de *pull requests*, por exemplo, permite que o orientador ou outros revisores sugiram e revisem o código de forma organizada.
- Gerenciamento de projeto: As *issues* do GitHub podem ser utilizadas para listar tarefas, acompanhar o progresso do desenvolvimento, registrar *bugs* encontrados e planejar futuras implementações.
- Integração contínua e entrega contínua (CI/CD): O *GitHub Actions*, uma funcionalidade integrada ao GitHub, permite a automação de testes, compilações e até mesmo a implantação de partes do sistema, o que pode ser explorado para garantir que o código esteja sempre em um estado funcional.

Em suma, o GitHub não é apenas um local para "guardar" o código; ele é uma plataforma completa que otimiza todo o ciclo de vida do desenvolvimento de *software*, garantindo a organização, a segurança e a rastreabilidade.

CAPÍTULO III - METODOLOGIA

1.14 BACK-END

Inicialmente foi utilizado o Spring Initializer, uma ferramenta online que facilita a criação de projetos Spring Boot permitindo ao usuário escolher dependências, linguagem, versão do Java e outras configurações, gerando automaticamente a estrutura básica do projeto pronta para ser importada em uma IDE, para produzir a

base do pacote Java onde as informações de programação do *back-end* foram escritas. Nele pode-se criar um pacote que através de informações iniciais: o tipo de projeto, nesse caso o Maven, linguagem de programação, Java como escolhida e as dependências.

O Maven é uma ferramenta de automação e gerenciamento de projetos Java que facilita o controle de dependências, a compilação do código, a execução de testes e a geração de *builds*, usando arquivo central chamado pom.xml para configurar tudo de forma padronizada e reproduzível.

As dependências são dados e informações disponíveis em nuvem que auxiliam a programação, como por exemplo, o Spring web que faz possível conexões entre programas e a internet.

Nesse projeto, foram utilizadas como principais dependências o spring-boot-starter-security que é utilizado no gerenciamento de acesso dos *endpoints*, impedindo que pessoas sem permissão acessem as funcionalidades do sistema.

Figura 3- Código spring-boot-starter-security

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

O springdoc-openapi-starter-webmvc-ui que permite o acesso ao Swagger. O Swagger é uma plataforma para teste das funcionalidades da aplicação, ou programa.

Figura 4- Código springdoc-openapi-starter-webmvc-ui

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.8.4</version>
</dependency>
```

O spring-boot-starter-web que é responsável pelos *endpoints* e conexões web gerenciando os links de cada requisição. Os principais tipos de requisições utilizadas são *GET*, *POST*, *PUT*, *PATCH*, *DELETE*.

Figura 5-Código spring-boot-starter-web

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

O mysql-connector-j é essencial na conexão e gerenciamento do banco de dados do MySQL.

Figura 6-Mysql-connector-j

```
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <scope>runtime</scope>
</dependency>
```

O lombok é uma biblioteca que gera automaticamente código *boilerplate* (como métodos *getters*, *setters*, construtores, *equals*, *hashCode*, *toString*) em tempo de compilação.

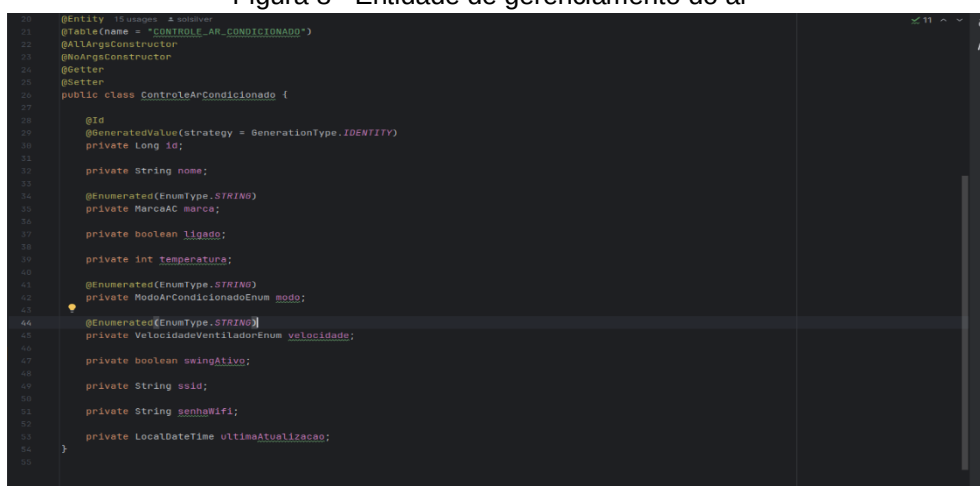
Figura 7 -Código lombok

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.28</version>
  <scope>provided</scope>
</dependency>
```

Após isso, cria-se os pacotes, ou pastas, onde as “responsabilidades” são armazenadas e separadas, consequentemente, estruturando o programa. O pacote “model” é o primeiro pacote a ser criado e é onde se armazenam as entidades e as DTOS. As DTOS (*Data Transfer Objects*) são objetos que servem como “contratos” para a transferência de dados entre camadas, garantindo que apenas as informações relevantes e seguras sejam expostas ou recebidas, protegendo a integridade e a evolução do seu modelo de domínio interno.

A entidade, uma classe publica dentro do Java, é feita para gerenciar as informações que vem ou que vão para o bando de dados e tem como obrigação a anotação `@Entity` e marcada com `@Table` para garantir o rastreamento da tabela relacionada. A entidade responsável pelo gerenciamento do estado do ar-condicionado é nomeada “ControleArCondicionado” relacionada a tabela “CONTROLE_AR_CONDIONADO”. Quando há requisição de informações na hora do funcionamento do programa a entidade transporta as informações para o banco de dados ou recebe as informações.

Figura 8 - Entidade de gerenciamento do ar



```

20 @Entity
21 @Table(name = "CONTROLE_AR_CONDIONADO")
22 @AllArgsConstructor
23 @NoArgsConstructor
24 @Getter
25 @Setter
26 public class ControleArCondicionado {
27
28     @Id
29     @GeneratedValue(strategy = GenerationType.IDENTITY)
30     private Long id;
31
32     private String nome;
33
34     @Enumerated(EnumType.STRING)
35     private MarcaAC marca;
36
37     private boolean ligado;
38
39     private int temperatura;
40
41     @Enumerated(EnumType.STRING)
42     private ModoArCondicionadoEnum modo;
43
44     @Enumerated(EnumType.STRING)
45     private VelocidadeVentiladorEnum velocidade;
46
47     private boolean swingAtivo;
48
49     private String ssid;
50
51     private String senhaWifi;
52
53     private LocalDateTime ultimaAtualizacao;
54 }
55

```

Outros pacotes essenciais foram criados, em ordem, primeiro a “Controller”, que vem com arquivos *controllers* marcados com “`@RestController`” e “`@RequestMapping`”. Tem papel de armazenamento de links de requisições e tipos de requisições, o que explica a marcação *rest controller*, e *request mapping* é o link principal para cada uma das requisições.

Em segundo, é estruturado o pacote “Service” que armazena os métodos, estrutura lógica do sistema. Para os arquivos denominados *servicer* é adicionado uma anotação chamada `@Service` que fara o sistema spring boot identificá-la e gerenciá-la como *service*.

O último pacote principal é a *Repository*, tem como responsabilidade controlar e gerenciar as chamadas ao banco de dados que denominamos como *Querys* códigos que buscam, mudam, colocam e apagam dados do banco.

Figura 9- Código de uma query para exemplificar o conteúdo do repository

```
SQL

SELECT marca, modelo FROM ar_condicionado WHERE temperatura > 25;
```

1.15 TRANSMISSOR IR + BACK-END

Para a criação do transmissor do infravermelho (Figura 8) foi utilizado um ESP 32 (Figura 9) com *WIFI* embutido e um emissor infravermelho KY-005 (Figura 10) que interage com o back-end através do *endpoint* que busca o ID do aparelho cadastrado. Para executar a comunicação é necessário configurar o código do ESP32 para estabelecer uma conexão a internet que é modulada por um método inicial de *void setup* e que, por consequência, aciona o *endpoint* responsável por buscar os dados do ar-condicionado cadastrado, em caso de falha da conexão o *void setup* retorna uma mensagem de erro.

Figura 10- Transmissor infravermelho

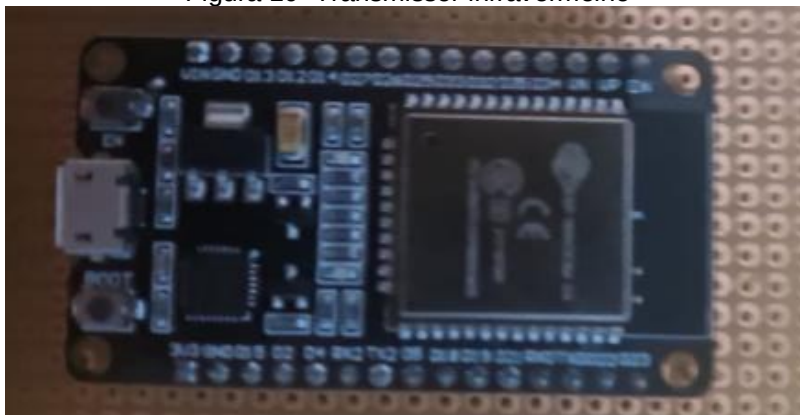


Figura 11-ESP 32

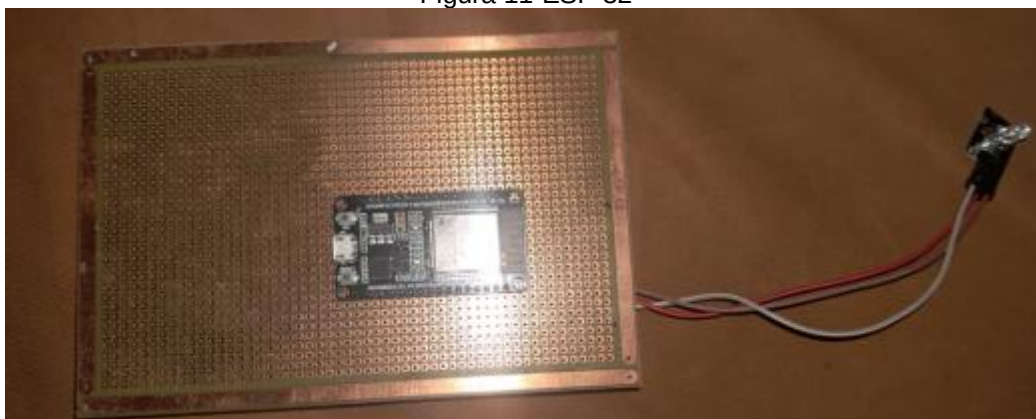


Figura 12- Emissor infravermelho KY-005



Figura 13-Código do ESP32 para a conexão com WIFI;

```
void setup() {
  Serial.begin(115200);
  irsend.begin();

  WiFi.begin(ssid, password);
  Serial.print("Conectando ao WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConectado! IP: " + WiFi.localIP().toString());
}
```

Há três bibliotecas principais que sustentam o funcionamento do aparelho, a primeira delas, nomeada, “<Wifih>” é a biblioteca responsável pela conexão à internet através de dados fornecidos pelo usuário: O SSID, que é o nome da rede conectada, a senha da rede.

Figura 14-Variáveis onde vão as credenciais da rede wifi

```
const char* ssid = "Galaxy A25 5G 3FBB";
const char* password = "na42ce36";
```

As duas outras bibliotecas interagem entre si, uma enviando a requisição pra API do *back-end*, nomeada “<HTTPClient.h>” e a outra processa os dados enviados de volta após a requisição, essa sendo a <ArduinoJson.h>.

O *void loop*, responsável por manter a atualização constante do envio de infravermelho do ar monitorando ininterruptamente o estado de funcionamento do ar baseado nos dados do banco. Dentro dele, é gerada uma requisição HTTP para o *endpoint* constituído pela URL base, o complemento relacionado a “*controller*” e o ID

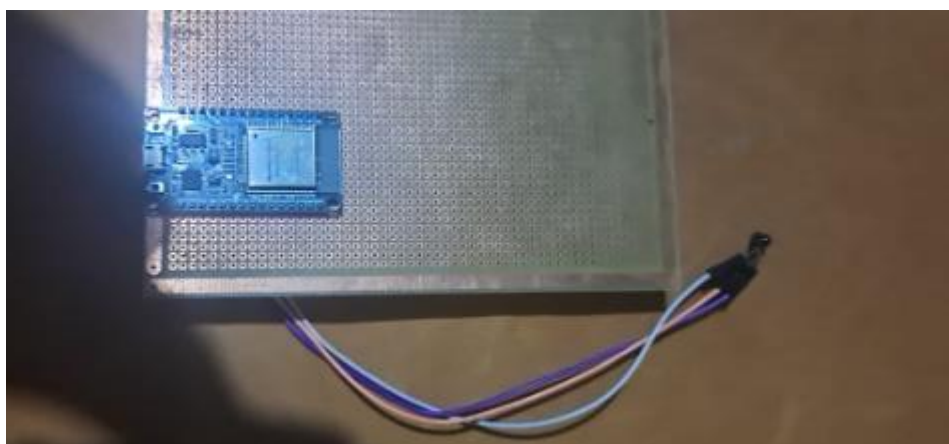
do aparelho. Após isso o *back-end* é acionado pela requisição HTTP onde ele passa por processos de pesquisa e montagem da resposta em formato Json.

Quando a resposta é positiva, ou status 200, os dados são sequenciados no void loop para que se adequem as variáveis que estão sendo enviadas e iniciam o processo dentro do método “enviarComando”.

Dentro do método dependendo do tipo de marca relacionada pelo Enum de “MarcasAC”, sendo elas: *Lg, Samsung, Midea, Daikin, Panasonic, Fujitsu, Gree, Toshiba, Hitachi, Komeco, Goodweather*, e desconhecida gera-se o código “IR” que será transmitido.

Cada *enum* possui uma biblioteca correspondente para o gerenciamento, com exceção da *Goodweather*, para a qual foi necessário criar um método separado para administrar o funcionamento. O método da *Goodweather* foi desenvolvido utilizando códigos mapeados manualmente diretamente do controle do ar-condicionado da sala de testes. Com uma grande amostra de dados coletados, foi possível estruturar um padrão lógico por meio do uso de I.A., que foi empregada na criação dos códigos hexadecimais correspondentes aos comandos a serem enviados para o ar-condicionado. Foi criado um circuito com ESP-32 e um receptor infravermelho, representado na imagem 13, que opera por meio de um código responsável por mapear e converter o sinal em código hexadecimal.

Figura 15- ESP-32 e receptor infravermelho



O código IR gerado para controlar um ar-condicionado nesse caso segue uma estrutura fixa de 48 bits (12 caracteres hexadecimais), cujo padrão é: 0xD5[XX]7A030000, onde D5 é um prefixo fixo comum a todos os comandos capturados (relacionado ao protocolo do controle remoto), e 7A030000 é um sufixo também fixo que define o tipo de comando como sendo uma combinação de modo e temperatura.

A única parte que varia no meio do código é o byte [XX], que representa uma combinação entre o modo de operação (como *COOL*, *HEAT*, *FAN*, *DRY* ou *AUTO*) e a temperatura configurada. Para isso, o serviço define um valor base para cada modo: *COOL* → 0x20; *HEAT* → 0x80; *FAN* → 0x60; *DRY* → 0x40; *AUTO* → 0x00

Esse valor base é somado à diferença entre a temperatura desejada e 16 (temperatura mínima suportada). Por exemplo, se o modo for *HEAT* e a temperatura for 20 °C, o valor calculado será $0x80 + (20 - 16) = 0x84$, gerando o *byte* intermediário como 84. Com isso, o código final será montado como 0xD5847A030000. Caso o parâmetro ligado seja *false*, significa que o ar-condicionado deve ser desligado, e nesse caso o serviço retorna diretamente o código fixo de desligamento: 0xD56214000000.

Adicionalmente, se o parâmetro *swing* for *true*, ou seja, se o comando deve ativar a oscilação do ar, o serviço também adiciona à lista de retorno o código fixo 0xD56216040000, que representa o comando de *swing*.

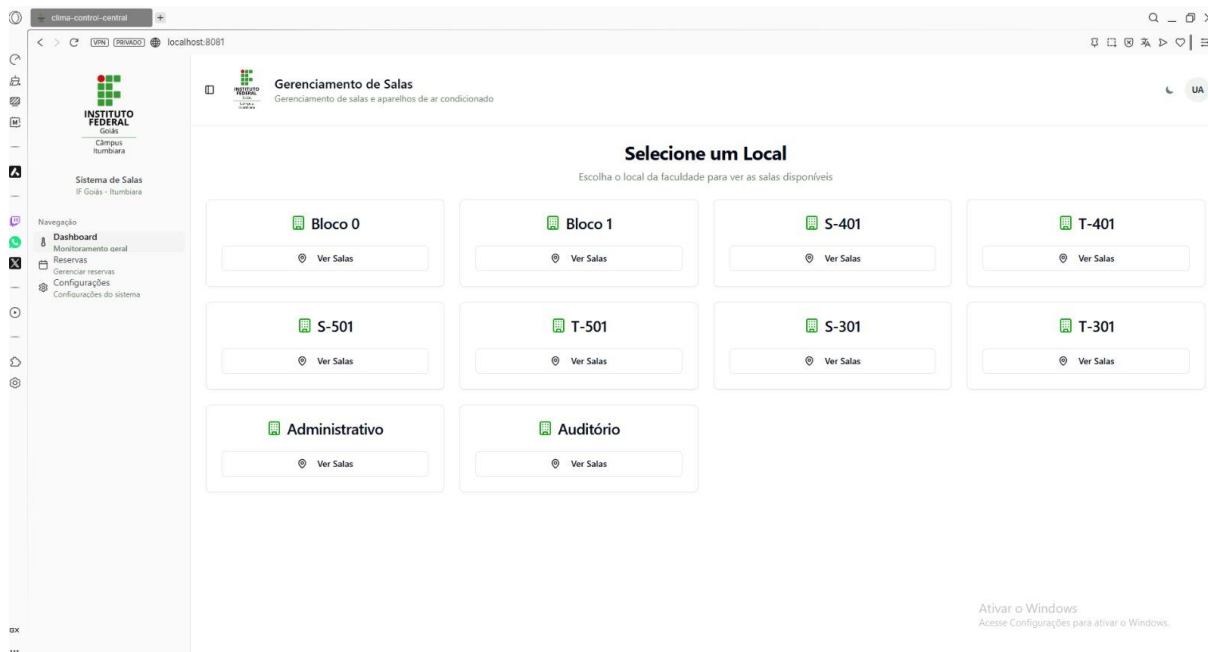
Como resultado, o serviço retorna uma lista de um ou dois códigos IR, dependendo das opções selecionadas. Essa lista pode ser usada em sequência (enviando os sinais IR com um pequeno intervalo entre eles) para controlar o aparelho com precisão. Após transmitido o infravermelho o ar-condicionado capta essa onda e reproduz o comando mudando parâmetros de temperatura, velocidade, ligar e *desligar*, *modo* e *swing*.

1.16 FRONT-END

As informações lidas pela interface do *front-end* são enviadas para o *back-end* onde são processadas, salvas, deletadas e alteradas, ou seja, o *back-end* administra as informações enquanto o *front-end* as demonstra ao usuário.

A interface foi projetada para ser a mais intuitiva possível permitindo ao usuário utilizar a plataforma de modo fácil, rápido e sem complicações.

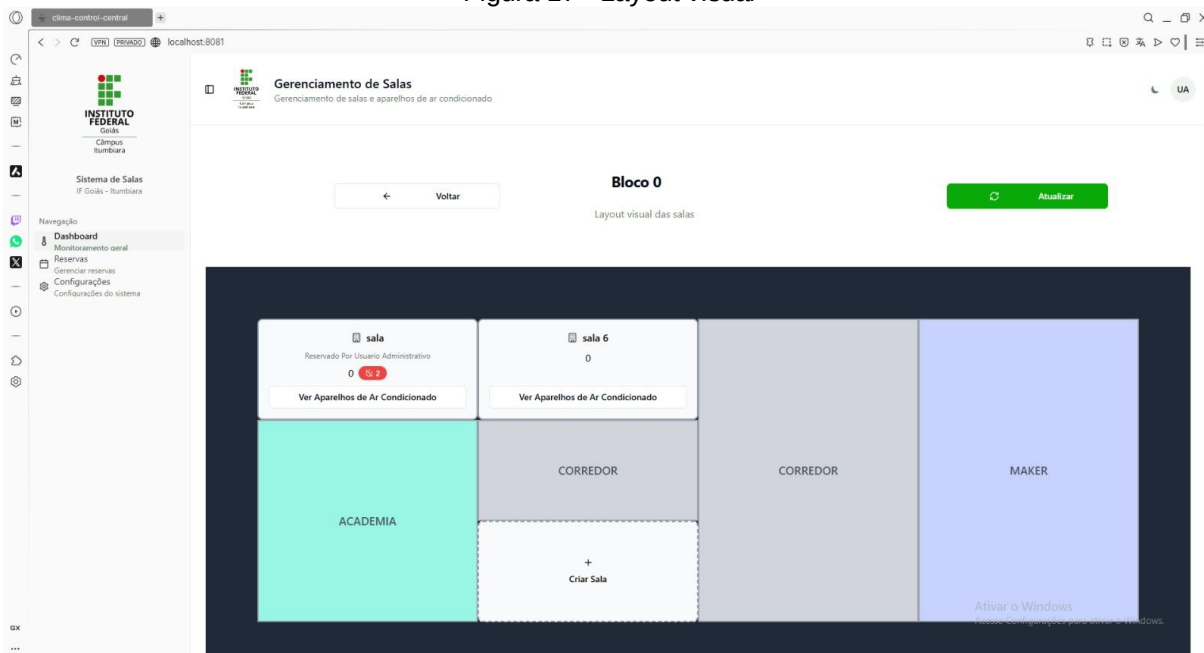
Figura 16 - Dashboard.



Contendo algumas telas que separam informações e torna essa interação usuário-programa funcional, e contribuindo para o funcionamento correto do *back-end*. Essas telas foram divididas em *Dashboard*, *Reservas* e *Configurações*.

Há também a possibilidade de acessar um *layout* visual das salas que permite ao usuário cadastrar salas e acessar a lista de condicionadores de ar com base em um mapa industrial que mostra os setores, salas e ambientes presentes no prédio, facilitando e melhorando a experiência do usuário.

Figura 17 - Layout visual



O *Dashboard* contém os cards de sala que possui as informações dos ares cadastrados e a opção de exclusão da sala. A tela de "Gerenciamento de Salas" permite aos usuários visualizarem uma lista paginada de salas, filtrá-las por nome, verificar status de reserva em tempo real. A interface é responsiva, adaptando-se a diferentes estados (como o tema escuro) e exibindo informações de acordo com o usuário logado.

Figura 18 - Tela do gerenciamento de condicionadores de ar de uma sala.

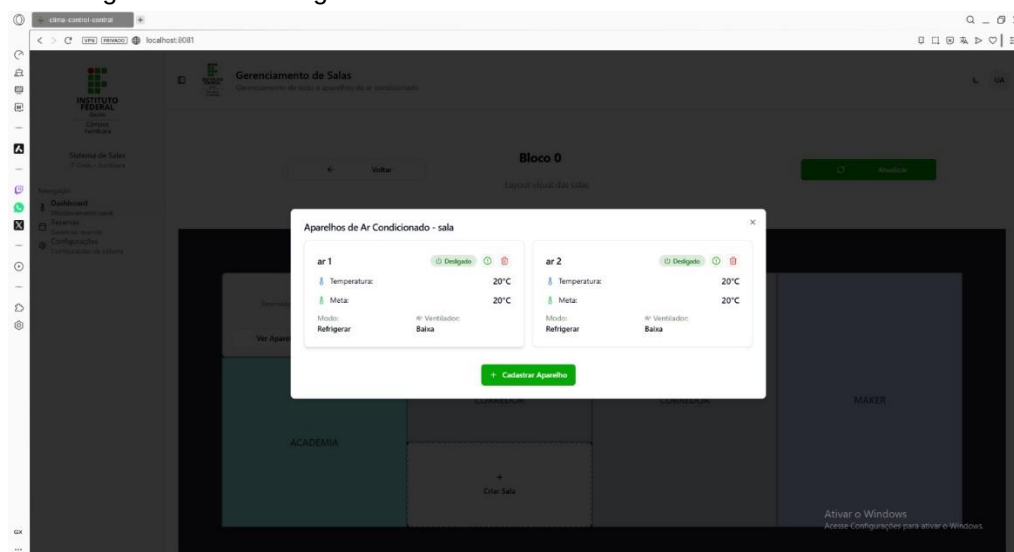
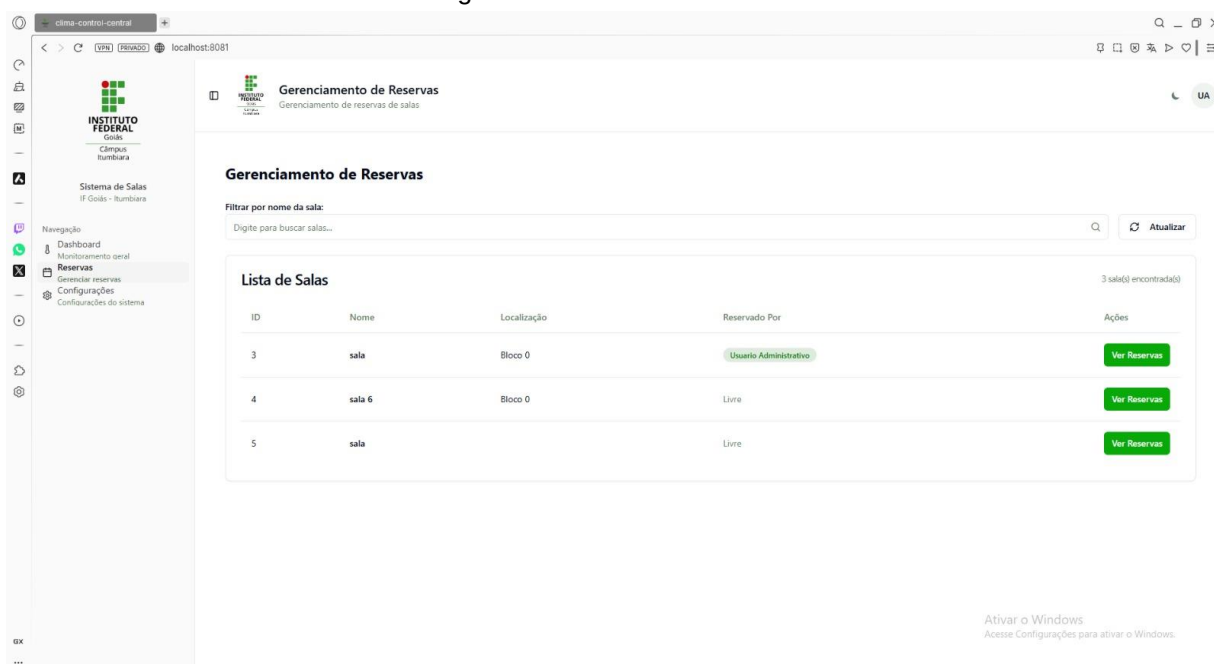


Figura 19 - Tela de reserva.



O gerenciamento de salas, como observado anteriormente, sofre alterações de atividade conforme o usuário logado, mas em um geral essa tela permite a interação direta da maioria dos usuários com as configurações do ar-condicionado e informações de reserva. Nela é possível, selecionar o ar-condicionado específico dentro daquela sala e alterar por meio dele temperatura, modo, velocidade e ligar e desligar.

Figura 20 - Planilha de reserva.

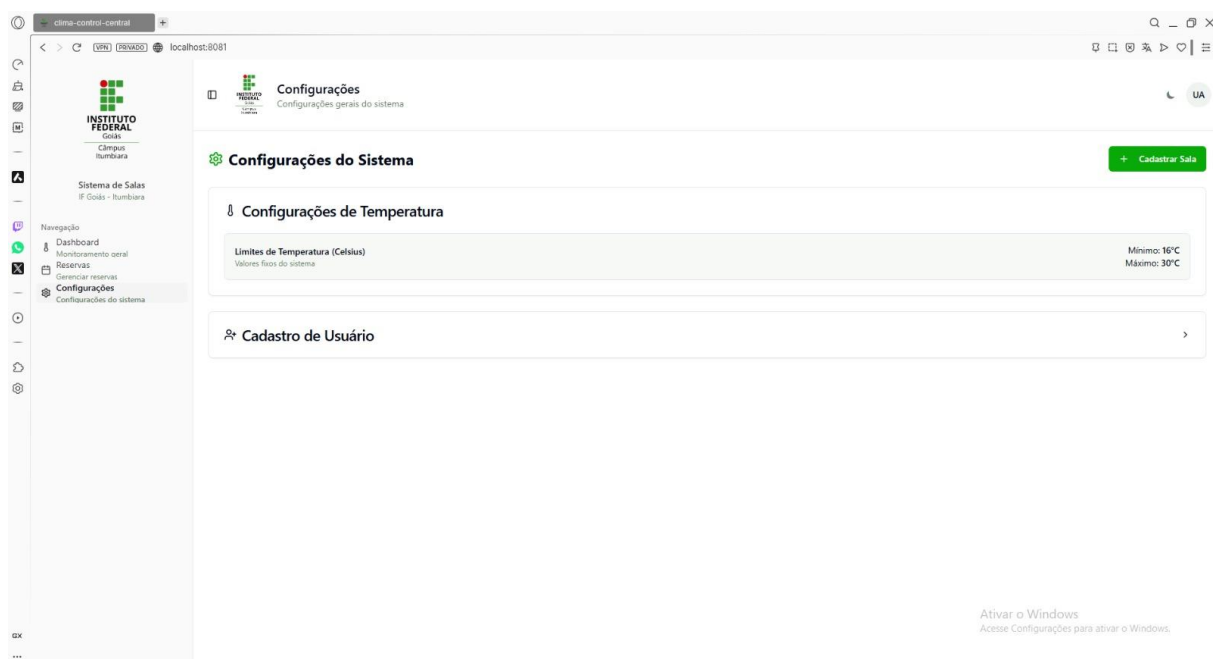
Gerenciamento de Reservas

Quadro de Horários - sala

Horário	Segunda-feira	Terça-feira	Quarta-feira	Quinta-feira	Sexta-feira	Sábado	Domingo
00:00 - 01:00	Livre	Livre	Livre	Livre	Livre	Livre	Livre
01:00 - 02:00	Livre	Livre	Livre	Livre	Livre	Livre	Livre
02:00 - 03:00	Livre	Livre	Livre	Livre	Livre	Livre	Livre
03:00 - 04:00	Livre	Livre	Livre	Livre	Livre	Livre	Livre
04:00 - 05:00	Livre	Livre	Livre	Livre	Livre	Livre	Livre
05:00 - 06:00	Livre	Livre	Livre	Livre	Livre	Livre	Livre
06:00 - 07:00	Livre	Livre	Livre	Livre	Livre	Livre	Livre
07:00 - 08:30	Livre	Livre	Livre	Livre	Livre	Livre	Livre
08:30 - 08:45	Livre	Livre	Livre	Livre	Livre	Livre	Livre
08:45 - 10:15	Livre	Livre	Livre	Livre	Livre	Livre	Livre
10:15 - 10:30	Livre	Livre	Livre	Livre	Livre	Livre	Livre
10:30 - 12:00	Livre	Livre	Livre	Livre	Livre	Livre	Livre
12:00 - 13:00	Livre	Livre	Livre	Livre	Livre	Livre	Livre
13:00 - 14:30	Livre	Livre	Livre	Livre	Livre	Livre	Livre

As reservas é uma tela que permite a usuários específicos, administradores e professores, reservar salas impedindo que alunos e outros professores, com exceção do usuário que reservou a sala e o administrativo, tenham permissão para alterar configurações do ar-condicionado no período da reserva. A interface é projetada em formato de planilha onde são divididas as reservas por horário e dia da semana.

Figura 21 - Tela de configurações.



As configurações do sistema são exclusivas para administradores e possuem opções de alterações de escalas termométricas como de *Celsius* para *Fahrenheit* e a opção de cadastro de usuários.

As telas e interfaces são alteradas conforme o usuário, sendo eles alunos, professores e administradores. Os alunos são a opção de usuário mais básica tendo permissões simples e que se limitam a alterações de configurações dos ares quando não há reservas. Os professores são a segunda opção de usuário e permite a criação de reservas e a alteração dos ares condicionados, mas diferente da terceira opção de usuário, os administradores, não podem excluir, gerar código para o *hardware* ou alterar reservas, fazer reservas para terceiros, criar salas e alterar configurações mais complexas do programa.

O cadastramento de usuários pode ser feito de forma manual ou utilizando planilhas com as informações do usuário, com nome, senha e o tipo do usuário esse cadastramento precisa ser feito em um modelo de planilha pré-programado que é lido pela interface e faz o cadastramento automático facilitando que grandes quantidades de pessoas sejam adicionadas ao banco de dados rapidamente.

Figura 22 - Botão para inserir planilha



Figura 23 - Modelo de planilha

	1	B	3	4
1	Nome	Matrícula	Senha	Tipo
2	João Adm	100001	senha123	ADMINISTRATIVO
3	Maria Prof	100002	senha456	PROFESSOR
4	Ana Aluna	100005	nova123	ALUNO
5				
6				

Além disso, há a possibilidade de por meio de um ícone presente na tela de identificação dos ares-condicionados baixar as informações do código de processamento de cada ar em específico.

Figura 24 - Botão verde responsável por fazer download do código



1.17 FRONT-END + BACK-END.

A interação *front-end* e *back-end* reside em um ciclo contínuo de requisições HTTP do *front-end* e respostas JSON do *back-end*. Para a tela inicial, ou dashboard, utiliza-se um *endpoint* que orquestra a obtenção dos dados da sala (imagem 16) e outro para busca de salas que se utiliza no filtro.

A interação prática entre o *front-end* e o *back-end* é vividamente exemplificada nas funcionalidades de "Gerenciamento de Salas", subsequente controle de "Aparelhos de Ar Condicionado", no "Gerenciamento de Reservas" e, crucialmente, na aba de "Configurações" da aplicação *ClimateControl*. Quando um usuário navega para a tela principal de Gerenciamento de Salas, o componente *front-end* responsável, construído em tecnologias como *React* ou *Angular*, assume proativamente a responsabilidade de iniciar a exibição dos dados. Sem possuir as informações das salas localmente, o *front-end* "sabe" que precisa delas e, automaticamente, formula e dispara uma requisição HTTP GET para o *back-end*.

Figura 25 - Endpoints responsáveis pela renderização do dashboard.

```
@GetMapping("/salas") no usages 1 solsilver
public Page<SalaDTO> buscarSalas(
    @RequestParam(required = false) String nome,
    @RequestParam(defaultValue = "0") int page,
    @RequestParam(defaultValue = "10") int size) {
    Pageable pageable = PageRequest.of(page, size);
    return salaService.buscarSalas(nome, pageable);
}

@GetMapping("/salas/nomes") no usages 1 solsilver
public List<String> buscarNomesSalas() {
    return salaService.buscarNomesSalas();
}
```

Para o carregamento inicial das salas, essa requisição tipicamente incluirá parâmetros de paginação padrão, como GET `/salas?page=0&size=10`. Paralelamente, ou logo após o login, o *front-end* obtém o perfil do usuário logado, como "Administrador", uma informação crucial que guiará a renderização condicional de elementos na interface. Ao receber a requisição GET `/salas`, o servidor *back-end*,

utilizando um framework como *Spring Boot*, roteia essa solicitação para o método `buscarSalas` do controlador apropriado. Neste ponto, o *back-end* reativamente extrai os parâmetros `nome`, `page` e `size` da URL, construindo um objeto *Pageable* para gerenciar a paginação. A lógica de negócio, encapsulada no `salaService`, é acionada, invocando o `salaRepository` para consultar o banco de dados e buscar as salas de acordo com o filtro de nome e a paginação solicitados. Um passo crucial de enriquecimento de dados ocorre: para cada sala encontrada, o *back-end* executa uma lógica adicional, consultando o `reservaSalaRepository` para verificar se há uma reserva ativa para aquela sala no momento atual. Se uma reserva ativa for identificada, o nome do usuário responsável pela reserva é anexado ao objeto `SalaDTO` correspondente. Essa agregação e processamento de dados em tempo real é uma responsabilidade exclusiva do *back-end*, garantindo que o *front-end* receba informações completas e já processadas. Após todo o processamento e enriquecimento, o *back-end* serializa o objeto `Page<SalaDTO>`, que engloba tanto a lista de salas quanto metadados de paginação (como o total de páginas e itens), em um formato JSON. Esta resposta JSON é então enviada de volta ao *front-end* com um status HTTP 200 OK, indicando o sucesso da operação. Uma vez que o navegador do usuário recebe essa resposta JSON, o código JavaScript do *front-end* assume o papel ativo de renderização dinâmica da interface. Ele processa os dados recebidos e, para cada `SalaDTO`, cria e exibe um "card" na interface, apresentando dinamicamente o nome da sala, sua localização e, se a informação estiver presente, o aviso "Reservado por: [Nome do Usuário]".

O controle de ações e acessos é um ponto chave aqui: utilizando o perfil do usuário logado, o *front-end* toma a decisão prática de renderizar condicionalmente o botão "Excluir Sala". Se o usuário não possuir privilégios administrativos, o botão simplesmente não é exibido, protegendo a interface de tentativas de ações não permitidas. O gerenciamento da paginação também é *handled* pelo *front-end*; os metadados recebidos são utilizados para renderizar os controles de navegação e o texto de status, como "Mostrando: X de Y salas". Cada clique em um controle de paginação pelo usuário dispara uma nova requisição *GET* com o parâmetro `page` ajustado, reiniciando o ciclo de interação para obter o próximo conjunto de dados.

Além da exibição paginada, a tela de Gerenciamento de Salas oferece a funcionalidade de filtro por nome. Ao interagir com o *select* de filtro, o *front-end* não

apenas dispara uma nova requisição para /salas com o parâmetro nome apropriado para a filtragem principal, mas antes disso, para popular as opções de filtro disponíveis, ele realiza uma requisição GET para o *endpoint* /salas/nomes. Este *endpoint* do *back-end*, por sua vez, consulta o repositório de salas para extrair todos os nomes de salas existentes e os retorna em uma lista de *strings*. O front-end recebe essa lista e a utiliza para preencher as opções do *select* de filtro, otimizando a experiência do usuário ao fornecer sugestões precisas de pesquisa.

Figura 26 - Endpoints responsáveis pela exclusão de uma sala.

```
@DeleteMapping("/sala") no usages  solsilver *
public ResponseEntity<Void> excluirSala(@RequestParam Long id) {
    salaService.deletarSala(id);
    return ResponseEntity.ok().build();
}
```

A interação continua quando um usuário administrativo deseja excluir uma sala. Após a decisão do usuário, o *front-end* envia uma requisição DELETE para o *endpoint* /sala, passando o id da sala a ser excluída como um RequestParam. O *back-end* recebe essa solicitação, e o salaService invoca o salaRepository.deleteById(id), removendo o registro da sala do banco de dados. Uma vez concluída a operação, o *back-end* responde com um ResponseEntity.ok().build(), sinalizando sucesso ao

front-end, que então atualiza a interface para refletir a remoção da sala, geralmente removendo o card correspondente da visualização.

Figura 27 - Endpoints responsáveis pelo controle dos ares.

```

@PutMapping("/{id}") no usages new *
public ResponseEntity<?> atualizarControler(
    @PathVariable Long id,
    @RequestBody ControlerArCondicionadoUpdateDTO dto) {
    return service.atualizarControler(id, dto);
}

@DeleteMapping("/{ar}") no usages new *
public ResponseEntity<Void> deletarAr(@RequestParam Long id) {
    service.deletarAr(id);
    return ResponseEntity.ok().build();
}

@GetMapping("/{arsalas}") no usages new *
public ResponseEntity<List<Map<String, Object>>> buscarTodosArsSalas(@RequestParam Long salaId) {
    List<Map<String, Object>> resposta = salaService.buscarArsPorSala(salaId);
    return ResponseEntity.ok(resposta);
}

@GetMapping(value = "/codigoesp", produces = MediaType.TEXT_PLAIN_VALUE) no usages new *
public ResponseEntity<byte[]> baixarArquivoIno(@RequestParam Long id) {
    String conteudo = service.gerarCodigoArduino(id);

    byte[] conteudoBytes = conteudo.getBytes(StandardCharsets.UTF_8);

    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.TEXT_PLAIN);
    headers.setContentDisposition(ContentDisposition.attachment()
        .filename("codigo_arduino-" + id + ".ino")
        .build());

    return new ResponseEntity<>(conteudoBytes, headers, HttpStatus.OK);
}

```

Outra funcionalidade crucial é o gerenciamento dos aparelhos de ar condicionado de cada sala. Ao clicar no botão "Ares Cadastrados" associado a uma sala específica, o *front-end* renderiza uma nova tela, que deve exibir todos os aparelhos de ar condicionado vinculados àquela sala. Para popular essa tela, o *front-end* faz uma requisição GET para o *endpoint* /arsalas, passando o salaId como parâmetro. O *back-end* recebe essa requisição e, através do salaService.buscarArsPorSala, busca a sala pelo ID, acessa a lista de ControlerArCondicionado associados e mapeia esses objetos para uma *List<Map<String, Object>>*, contendo detalhes como ID, nome, estado (ligado/desligado), temperatura, modo e velocidade, além de informações como swingAtivo e marca. Essa lista é retornada ao *front-end* em formato JSON, permitindo que a interface crie e exiba os cards de cada aparelho de ar condicionado com seus respectivos estados e controles. Ao lado do botão de exclusão de cada aparelho de ar condicionado, o *front-end*, novamente com base no perfil administrativo do usuário, exibe um ícone de exclamação em círculo. Este ícone, quando clicado, dispara uma requisição GET para o *endpoint* /codigoesp, passando o id do aparelho de ar. O *back-end*, no método baixarArquivoIno, gera o conteúdo do arquivo .ino (código Arduino) para aquele dispositivo, encapsula-o em um *ResponseEntity<byte[]>* com o *MediaType.TEXT_PLAIN* e *ContentDisposition* configurados para *download*, e o envia de volta ao *front-end*. O navegador do usuário, então, inicia o *download* do arquivo

codigo_arduino_[id].ino, oferecendo uma funcionalidade administrativa essencial para o setup e manutenção dos dispositivos físicos. Dentro da tela de detalhes de um aparelho de ar condicionado, o usuário pode realizar alterações. Quando o botão "Salvar Alterações" é clicado, o *front-end* constrói uma requisição PUT para o endpoint `/[id]` (onde `id` é o ID do aparelho de ar), enviando um DTO no corpo da requisição com os novos valores (temperatura, modo, velocidade, estado de ligado/desligado, etc.). O *back-end* recebe essa requisição e, dentro do método `atualizarControle`, primeiro busca o `ControleArCondicionado` pelo `id`. Crucialmente, antes de aplicar as atualizações, o *back-end* executa uma robusta verificação de autorização: ele verifica o perfil do usuário logado (se é administrativo) e se há uma reserva ativa para a sala do ar condicionado. Se houver uma reserva, ele também verifica se o usuário logado é o responsável por essa reserva. A atualização só é permitida se o usuário for um administrador, ou se não houver reserva ativa, ou se o usuário for o responsável pela reserva. Se a validação for bem-sucedida, o *back-end* atualiza os campos do controle de ar condicionado com os dados fornecidos no DTO, registra a `ultimaAtualizacao`, salva as mudanças no repositório e retorna um `ResponseEntity.ok` com uma mensagem de sucesso e o objeto atualizado. Caso contrário, se a permissão for negada, o *back-end* retorna um `HttpStatus.FORBIDDEN (403)`, com uma mensagem indicando a falta de permissão, garantindo a segurança da operação.

Figura 28 - Endpoint que cadastra os ares-condicionados.

```
@PostMapping("/cadastrar") no usages 1 salsilver *
public ResponseEntity<String> cadastrarAparelho(@RequestBody CadastroAparelhoDTO dto) {
    ControleArCondicionado salvo = service.cadastrarAparelho(dto);
    return ResponseEntity.status(HttpStatus.CREATED).build();
}
```

Por fim, a aplicação também permite o cadastro de novos aparelhos de ar condicionado. Esta funcionalidade, geralmente restrita a administradores, é iniciada pelo *front-end* que, ao clicar em "Cadastrar Aparelho", apresenta um formulário para coletar os dados do novo aparelho. Ao submeter o formulário, o *front-end* envia uma requisição POST para o endpoint `/cadastrar`. No *back-end*, o `service.cadastrarAparelho` é invocado. Novamente, uma rigorosa validação de permissão ocorre: o *back-end* verifica se o usuário logado é do tipo administrativo. Se não for, uma exceção é lançada, impedindo a operação. Caso contrário, um novo objeto `ControleArCondicionado` é instanciado com os dados fornecidos, valores

Figura 29 - Endpoints para gerenciar as reservas

```

@GetMapping("/salaReserva") no usages  ⚡ solsilver
public ResponseEntity<List<ReservaSalaDTO>> buscarTodosReservasSala(@RequestParam Long salaId) {
    List<ReservaSalaDTO> resposta = salaService.buscarReservasDaSala(salaId);
    return ResponseEntity.ok(resposta);
}

@PostMapping("/reservas") no usages  ⚡ solsilver
public ResponseEntity<Void> criarReservas(@RequestBody @Valid CadastroReservaDTO dto) {
    salaService.criarReserva(dto);
    return ResponseEntity.ok().build();
}

@DeleteMapping("/reservas") no usages  ⚡ solsilver
public ResponseEntity<Void> deletarReservas(@RequestParam Long id) {
    salaService.deletarReserva(id);
    return ResponseEntity.ok().build();
}

```

padrão para ligado, temperatura, modo, velocidade e *swing* são definidos, e o aparelho é associado à sala correta. Após salvar o novo aparelho no repositório, o back-end gera um código Arduino (presumivelmente para configuração física do dispositivo) e retorna este código ao front-end com um status 201 *Created*, confirmando o sucesso do cadastro.

Adicionalmente ao gerenciamento de salas e aparelhos, a aplicação ClimateControl dispõe de uma funcionalidade de Gerenciamento de Reservas. A tela de Gerenciamento de Reservas é inicialmente renderizada a partir dos dados de salas já obtidos pela requisição paginada GET /salas, exibindo uma lista das salas e seu status de reserva geral (como "Livre"). Quando o usuário clica no botão "Ver Reservas" associado a uma sala específica nessa tela, o front-end faz uma requisição GET para o endpoint /salaReserva, passando o salaId como parâmetro. O *back-end*, através do salaService.buscarReservasDaSala, recupera todas as reservas agendadas para aquela sala, mapeando-as para uma lista de ReservaSalaDTOs que incluem detalhes como ID da reserva, ID da sala, matrícula e nome do usuário, dia da semana, horário de início e fim. Essa lista é então retornada ao *front-end* em JSON, permitindo que a interface exiba um quadro de horários detalhado para a sala selecionada, indicando os períodos livres e os já reservados, e por quem. A criação de novas reservas também demonstra uma interação complexa. Quando o usuário clica em uma célula sem reserva no quadro de horários, o *front-end* abre um modal "Criar Reserva". Se o usuário logado for um administrador, este modal oferece a opção de "Buscar professor" e até mesmo tornar a reserva "permanente". Para popular a lista de professores no campo de busca do modal, o *front-end* realiza uma requisição

GET para o *endpoint* /professores. O *back-end*, por sua vez, executa a query para buscar usuários do tipo professor, transformando o resultado em uma lista de UsuarioDTOs (contendo matrícula e nome do professor) que é enviada ao *front-end*. Uma vez preenchidos os detalhes da reserva no modal, o *front-end* envia uma requisição POST para o *endpoint* /reservas, contendo um CadastroReservaDTO com o salald, diaSemana, horarioInicio, horarioFim, a matricula do professor (se aplicável e se for um admin) e o status permanente. O *back-end*, no salaService.criarReserva, processa esta requisição. Ele primeiramente identifica o Usuario responsável pela reserva: se uma matricula for fornecida (indicando que um administrador está reservando para um professor), ele busca o professor correspondente; caso contrário, a reserva é associada ao usuarioLogado. Ele então recupera a Sala e constrói um objeto ReservaSala, persistindo-o no reservaSalaRepository. Uma vez que a reserva é salva, o *back-end* responde com um ResponseEntity.ok().build(), sinalizando sucesso ao *front-end*, que atualiza o quadro de horários. Por fim, a exclusão de reservas é outra interação crítica. Ao clicar em uma reserva existente no quadro de horários, o usuário (presumivelmente com as permissões adequadas, embora a validação final resida no *back-end*) pode iniciar a remoção. O *front-end* envia uma requisição DELETE para o *endpoint* /reservas, passando o id da reserva a ser excluída como um RequestParam. O *back-end*, no salaService.deletarReserva, simplesmente invoca o reservaSalaRepository.deleteById(id), removendo o registro da reserva do banco de dados e respondendo com ResponseEntity.ok().build(), permitindo que o *front-end* atualize o quadro de horários para refletir a reserva removida.

Figura 30 - Agenda para desligar os condicionadores de ar.

```

public class DesligaArAutomatico {
    @Autowired 3 usages
    private ReservaSalaRepository reservaRepository;
    @Autowired 1 usage
    private ControleArCondicionadoRepository controleRepository;

    @Scheduled(cron = "0 */5 * * *", zone = "America/Sao_Paulo") no usages 1 solsilver
    @Transactional
    public void desligarArSemReserva() {
        LocalDateTime agora = LocalDateTime.now();
        DayOfWeek hoje = agora.getDayOfWeek();
        LocalTime horaAtual = agora.toLocalTime().truncatedTo(ChronoUnit.MINUTES);

        List<ReservaSala> reservasQueTerminaram = reservaRepository.findByDiaSemanaAndHorarioFim(hoje, horaAtual);

        for (ReservaSala reserva : reservasQueTerminaram) {
            Sala sala = reserva.getSala();
            boolean haProximaReserva = reservaRepository.existsBySalaAndDiaSemanaAndHorarioInicio(
                sala, hoje, horaAtual
            );

            if (!haProximaReserva) {
                for (ControleArCondicionado ac : sala.getAresCondicionados()) {
                    if (ac.isLigado()) {
                        ac.setLigado(false);
                        ac.setUltimaAtualizacao(LocalDateTime.now());
                        controleRepository.save(ac);
                    }
                }
            }

            if (!reserva.isPermanente()) {
                Sala salaReserva = reserva.getSala();
                salaReserva.getReservas().remove(reserva);
                reservaRepository.delete(reserva);
            }
        }
    }
}

```

Além das interações diretas iniciadas pelo usuário através do *front-end*, o sistema ClimateControl incorpora um nível de automação no *back-end* para otimizar o gerenciamento dos aparelhos de ar condicionado em relação às reservas. Esta automação é implementada por meio de agendadores (*schedules*) que operam de forma independente da interface do usuário, mas cujo impacto é refletido na informação exibida pelo *front-end*. Um desses agendadores é responsável por desligar o ar-condicionado automaticamente em salas quando uma reserva termina e não há uma próxima reserva consecutiva. Esse processo é executado periodicamente (a cada 5 minutos, conforme a configuração `@Scheduled`), onde o *back-end* verifica as reservas que acabaram naquele exato momento. Para cada reserva finalizada, ele consulta se existe uma próxima reserva para a mesma sala no horário subsequente. Se não houver, todos os aparelhos de ar condicionado ligados naquela sala são desligados, e seu status de `ultimaAtualizacao` é registrado. Paralelamente, se a

reserva finalizada não for do tipo "permanente", ela é removida do sistema de reservas, liberando o slot para futuras alocações.

Figura 31 - Agenda para ligar os ares-condicionados.

```
@Service no usages 1 salsilver
public class LigarArAutomatico {
    @Autowired 1 usage
    private ReservaSalaRepository reservaRepository;
    @Autowired 1 usage
    private ControleArCondicionadoRepository controleRepository;

    @Scheduled(cron = "0 */5 * * * *", zone = "America/Sao_Paulo") no usages 1 salsilver
    public void ligarArAntesDaReserva() {
        LocalDateTime agora = LocalDateTime.now();
        DayOfWeek hoje = agora.getDayOfWeek();
        LocalTime cincoMinutosDepois = agora.plusMinutes(1).toLocalTime().truncatedTo(ChronoUnit.MINUTES);

        List<ReservaSala> reservas = reservaRepository.findByDiaSemanaAndHorarioInicio(hoje, cincoMinutosDepois);

        for (ReservaSala reserva : reservas) {
            for (ControleArCondicionado ac : reserva.getSala().getAresCondicionados()) {
                if (!ac.isLigado()) {
                    ac.setLigado(true);
                    ac.setUltimaAtualizacao(LocalDateTime.now());
                    controleRepository.save(ac);
                }
            }
        }
    }
}
```

Complementarmente, outro agendador atua para ligar o ar-condicionado automaticamente antes do início de uma nova reserva. Também executado periodicamente, ele identifica as reservas que estão prestes a começar (por exemplo, dentro dos próximos 5 minutos). Para cada uma dessas reservas, ele percorre os aparelhos de ar condicionado da sala correspondente e, se estiverem desligados, os liga, atualizando seu estado e o timestamp de ultimaAtualizacao. Esses agendadores garantem que a eficiência energética e o conforto térmico sejam mantidos de forma proativa, refletindo na interface do *front-end* o estado atualizado dos aparelhos, mesmo sem uma ação direta do usuário para ligá-los ou desligá-los.

Figura 32 - Endpoints da aba de configurações.

```
@PostMapping("/cadastrar") no usages 1 salsilver
public ResponseEntity<> cadastrar(@RequestBody Usuario usuario) {
    usuario.setSenha(encoder.encode(usuario.getSenha()));
    Usuario salvo = usuarioRepository.save(usuario);
    return ResponseEntity.ok(salvo);
}

@PostMapping("/salas") no usages 1 salsilver
public ResponseEntity<Sala> criarSala(@Valid SalaCreatedTO dto) {
    Sala novaSala = salaService.criarSala(dto);
    return ResponseEntity.status(HttpStatus.CREATED).body(novaSala);
}
```

Finalmente, a aplicação ClimateControl possui uma aba de "Configurações", dedicada a funcionalidades administrativas essenciais, como a adição de novas salas e a criação de usuários no sistema. Para criar uma nova sala, o *front-end* envia uma requisição POST para o *endpoint* /salas, contendo um SalaCreateDTO com o nome e a localização da nova sala. No back-end, o salaService.criarSala é invocado. Este método executa uma validação rigorosa para garantir que apenas usuários com o perfil administrativo possam realizar esta operação. Se a permissão for validada, a nova Sala é construída com as informações fornecidas e persistida no salaRepository, e o back-end retorna um ResponseEntity com o status *CREATED* (201) e o objeto da sala recém-criada. Para a criação de um novo usuário, o *front-end* na aba de configurações coleta os dados necessários (nome, matrícula, tipo de usuário, senha, etc.) e envia uma requisição POST para o *endpoint* /cadastrar (referente ao usuário, não ao aparelho de ar). O *back-end* recebe o objeto Usuario, e antes de persistir, a senha fornecida é codificada (encoder.encode()) para garantir a segurança. O usuarioRepository.save() então armazena o novo usuário no banco de dados, e o backend responde com ResponseEntity.ok(), confirmando o cadastro bem-sucedido e possivelmente retornando o objeto do usuário salvo.

Este extenso exemplo da funcionalidade de Gerenciamento de Salas, Aparelhos de Ar-Condicionado, Reservas e Configurações, complementado pela automação inteligente dos schedules no *back-end*, ilustra perfeitamente a interação prática e contínua entre *front-end* e *back-end* em um sistema complexo. O *front-end* atua como a camada de apresentação e interatividade, traduzindo as intenções do usuário em requisições bem definidas e adaptando a visualização. O *back-end*, por sua vez, é o cérebro da aplicação, responsável por processar a lógica de negócio complexa, interagir com o banco de dados, enriquecer os dados antes da entrega e aplicar as regras de segurança e autorização, garantindo que as informações sejam manipuladas corretamente e que apenas usuários com as permissões adequadas possam executar operações críticas. A fluidez e a robustez dessa comunicação via APIs RESTful são essenciais para a construção de aplicações web modernas e eficientes, garantindo um sistema coeso e funcional.

1.18 TESTES.

Esta seção detalha os procedimentos de testes e validação aplicados ao sistema desenvolvido, visando garantir a conformidade com os requisitos funcionais e não funcionais, bem como a robustez da solução proposta. Os testes foram divididos em etapas que abrangeram desde a validação das APIs (*endpoints*), a integração do *frontend* com o *backend*, até os testes de funcionalidade do controlador ESP32 em ambiente laboratorial.

3.5.1 METODOLOGIA DE TESTES

A metodologia de testes adotada envolveu uma combinação de abordagens. Os testes de unidade e integração relacionados aos *endpoints* do *back-end* foram realizados primariamente através da interface *Swagger*, permitindo a verificação individual de cada requisição e sua respectiva resposta. A integração entre o *front-end* (React) e o *back-end* (Spring Boot Web) foi validada por meio da execução conjunta de ambas as aplicações em ambiente de desenvolvimento, simulando o fluxo completo de dados e interações do usuário. Para o controlador ESP32, os testes foram conduzidos em laboratório, com o sistema web e o controlador operando simultaneamente para simular o ambiente de uso real.

3.5.2 TESTES DA CAMADA DE INTEGRAÇÃO (API E FRONTEND)

Os testes da camada de integração focaram na comunicação entre os componentes do sistema.

Testes de *Endpoints* (Swagger): Cada *endpoint* da API RESTful (desenvolvida com Spring Boot Web) foi individualmente testado via Swagger. Foram verificadas as operações de criação, leitura, atualização e exclusão (CRUD) para entidades como salas, usuários e reservas, garantindo que as requisições HTTP retornavam as respostas esperadas e que o banco de dados MySQL estava sendo atualizado corretamente.

Testes de Integração Frontend-Backend: A execução conjunta do *front-end* (React) e do *back-end* (Spring Boot Web) permitiu verificar a fluidez da comunicação. Testou-se a exibição de todas as salas e ares-condicionados cadastrados na tela

principal, a funcionalidade de exclusão de salas e ares-condicionados, e a capacidade de *download* do código do Arduino para o ESP32 diretamente pela interface. Na aba de configurações, foi verificada a correta funcionalidade de cadastro de novas salas e usuários. Na aba de reservas, assegurou-se que as salas apareciam para permitir o cadastro e gerenciamento de reservas de uso do ar-condicionado.

Testes de Permissões de Usuário: As permissões de cada tipo de usuário foram exaustivamente testadas, verificando se o acesso a determinadas funcionalidades e informações era restrito conforme o perfil (administrador vs. usuário comum), garantindo a segurança e a integridade do sistema.

3.5.3 TESTES DO CONTROLADOR ESP32 E SOLUÇÃO IR ADICIONAL

Os testes com o controlador ESP32 em laboratório revelaram um desafio significativo: o ar-condicionado disponível no ambiente de testes não era compatível com a biblioteca padrão `#include <IRremoteESP8266.h>`, pois não havia suporte pré-existente para sua marca e modelo específicos.

Para contornar essa limitação e garantir a funcionalidade do sistema, foi desenvolvida e implementada uma solução de *hardware* e *software* para captura de códigos infravermelhos. Um circuito receptor IR foi conectado ao ESP32 no pino 4, e um *sketch* de Arduino foi utilizado para capturar e decodificar os sinais emitidos pelo controle remoto original do ar-condicionado. O código implementado foi:

Figura 33 - Código para capturar o infravermelho do controle.

```

#include <Arduino.h>
#include <IRrecv.h>
#include <IRremoteESP8266.h>
#include <IRutils.h>

const uint16_t kRecvPin = 4; // Use o pino onde o receptor IR está conectado

IRrecv irrecv(kRecvPin, 1024, 15, true); // buffer grande, marcação precisa
decode_results results;

void setup() {
  Serial.begin(115200);
  while (!Serial) // Para placas com USB nativo como Leonardo
    delay(50);
  irrecv.enableIRIn(); // Inicia receptor
  Serial.println("Pronto para receber sinais IR");
}

void loop() {
  if (irrecv.decode(&results)) {
    // Mostra tudo: protocolo, bit count, dados RAW e analisados
    Serial.println(resultToHumanReadableBasic(&results));
    Serial.println(resultToSourceCode(&results));
    Serial.println("");

    yield(); // Para evitar WDT em ESPs
    irrecv.resume(); // Pronto para o próximo
  }
}

```

Após a captura e catalogação dos códigos IR para as diversas funções do ar-condicionado (ligar/desligar, modos, temperaturas, velocidades), estes foram identificados sob a marca "GOODWEATHER" (nome atribuído à marca não suportada). A funcionalidade `irsend.sendGoodweather(codigo, 48);` foi então adicionada ao *firmware* do ESP32 para permitir o envio desses códigos específicos:

Figura 34 - Código para usar o goodweather infravermelho.

```
if (marca == "GOODWEATHER") {
    if (!codigoRaw.isNull() && codigoRaw.size() > 0) {
        String codigoStr = codigoRaw[0];
        Serial.println(codigoStr);
        uint64_t codigo = strtoull(codigoStr.c_str(), NULL, 16);
        irsend.sendGoodweather(codigo, 48);
    }
}
```

Para automatizar a geração desses códigos específicos no *back-end*, foi desenvolvido um serviço em Java no Spring Boot:

Figura 35 - Estrutura para gerar o infravermelho do controle.

```
@Service
public class Zhjt03Service {

    private static final String PREFIX = "D5";
    private static final String CODE_OFF = "D56214000000";
    private static final String CODE_SWING = "D56216040000";

    private static final Map<String, Integer> modoBase = Map.of(
        k1: "FRIJO", v1: 0x20,
        k2: "QUENTE", v2: 0x80,
        k3: "VENTILACAO", v3: 0x60,
        k4: "DESUMIDIFICAR", v4: 0x40,
        k5: "AUTO", v5: 0x00
    );

    private static final Map<String, Integer> velocidadeFanMap = Map.of(
        k1: "AUTO", v1: 0x1A,
        k2: "BAIXA", v2: 0x7A,
        k3: "MEDIA", v3: 0x5A,
        k4: "ALTA", v4: 0x3A
    );

    /**
     * Gera o(s) código(s) IR baseado em modo, temperatura, velocidade, swing e status ligado.
     *
     * @param modo      Modo de operação: COOL, HEAT, FAN, DRY, AUTO
     * @param temperatura Temperatura entre 16 e 30
     * @param velocidade Velocidade: AUTO, BAIXO, MEDIO, ALTO
     * @param swing      true = ativa swing (gera código extra)
     * @param ligado     true = ligado, false = desligado
     * @return Lista de strings com os códigos IR em hexadecimal
     */
}
```

Figura 36 - Função para gerar o infravermelho do controle.

```

public List<String> gerarCodigoIr(String modo, int temperatura, String velocidade, boolean swing, boolean ligado) {
    List<String> codigos = new ArrayList<>();

    if (!ligado) {
        codigos.add("0x" + CODE_OFF);
        return codigos;
    }

    if (temperatura < 16 || temperatura > 30) {
        throw new IllegalArgumentException("Temperatura deve estar entre 16 e 30.");
    }

    Integer baseModo = modoBase.get(modo.toUpperCase());
    if (baseModo == null) {
        throw new IllegalArgumentException("Modo inválido. Use: COOL, HEAT, FAN, DRY, AUTO.");
    }

    Integer byteVelocidade = velocidadeFanMap.get(velocidade.toUpperCase());
    if (byteVelocidade == null) {
        throw new IllegalArgumentException("Velocidade inválida. Use: AUTO, BAIXO, MEDIO, ALTO.");
    }

    int byteModoTemp = baseModo + (temperatura - 16); // byte 2
    String codigo = String.format("0x%s%02X%02X050000", PREFIX, byteModoTemp, byteVelocidade);
    codigos.add(codigo);

    /*if (swing) {
        codigos.add("0x" + CODE_SWING);
    }*/

    return codigos;
}

```

Esta abordagem customizada permitiu que o sistema funcionasse perfeitamente com o ar-condicionado do laboratório, superando a limitação inicial de compatibilidade da biblioteca. A funcionalidade de ligar e desligar o ar-condicionado automaticamente, com base em agendamentos ou condições pré-definidas (como as implementadas nos serviços `ligarAntesDeReserva` e `desligarArSemReserva` conforme o código do *backend*), também foi testada e validada com sucesso, garantindo a robustez do sistema em cenários de automação.

3.5.4 CONCLUSÃO DOS TESTES

Os testes realizados em todas as camadas do sistema – *front-end*, *back-end* e *hardware* (ESP32) – demonstram que a solução desenvolvida é robusta, funcional e atende aos requisitos propostos. A superação do desafio de compatibilidade do controle infravermelho, através da implementação de uma solução customizada de captura e geração de códigos, ressalta a capacidade do sistema de se adaptar a diferentes cenários e equipamentos. A validação das permissões de usuário e das

funcionalidades de automação, incluindo o controle automático do ar-condicionado baseado em reservas, reforça a segurança e a eficácia da aplicação para o controle e monitoramento de ar-condicionado em ambientes reais.

CAPÍTULO IV – RESULTADOS E DISCUSSÃO.

A solução visou permitir o monitoramento, controle à distância e análise em tempo real dos parâmetros ambientais, com foco primordial na otimização do uso energético e no conforto do usuário. Adicionalmente, buscou-se analisar a viabilidade, os custos, a escalabilidade e a experiência do usuário na implementação em um ambiente real. Os resultados obtidos confirmam que o sistema desenvolvido demonstrou funcionalidade e desempenho satisfatórios, operando conforme o esperado.

4.1 DESENVOLVIMENTO DO *FIRMWARE* PARA O MICROCONTROLADOR ESP32

O *firmware* para o microcontrolador ESP32 foi desenvolvido com sucesso, estabelecendo uma conexão estável à rede Wi-Fi local. A integração de uma biblioteca de infravermelho permitiu a replicação eficaz dos códigos de controle de diversos modelos de ar-condicionado, assegurando a compatibilidade com múltiplos dispositivos. A comunicação via HTTP com o *back-end* foi implementada de forma robusta, garantindo tanto o envio confiável de dados de telemetria – como temperatura ambiente, umidade relativa e o estado operacional do ar-condicionado – quanto o recebimento e processamento de comandos de controle. Adicionalmente, esforços foram dedicados à otimização do consumo de energia do ESP32, visando a viabilidade de sua operação contínua e eficiente.

A escolha do ESP32 mostrou-se estratégica, principalmente devido à sua capacidade Wi-Fi integrada e seu custo-benefício, o que facilitou a criação de um dispositivo IoT compacto e de alto desempenho. A validação da robustez da comunicação infravermelha foi confirmada por meio de testes repetitivos, evidenciando a precisão dos comandos enviados ao ar-condicionado. A adoção do protocolo HTTP foi crucial, proporcionando uma comunicação padronizada e eficiente, essencial para a arquitetura IoT e para a futura escalabilidade do sistema.

4.2 CRIAÇÃO DA INTERFACE WEB RESPONSIVA

Uma interface web intuitiva e responsiva foi desenvolvida, empregando *React* no *front-end*. Esta interface demonstrou total adaptabilidade a diferentes tamanhos de tela, como *smartphones*, *tablets* e *desktops*, conforme validado por testes em

múltiplos dispositivos. Funcionalidades essenciais de controle, incluindo ligar/desligar, ajuste de temperatura, seleção de modo de operação e velocidade do ventilador, foram implementadas com sucesso, respondendo prontamente aos comandos do usuário. Um sistema de autenticação foi integrado para assegurar o acesso seguro à plataforma.

A responsividade da interface é um pilar fundamental para a usabilidade do sistema em um ambiente de automação residencial, permitindo acesso de diversos dispositivos. A priorização da simplicidade e clareza no design da interface visou garantir que usuários, mesmo sem conhecimento técnico aprofundado, pudessem operar o sistema com facilidade. A capacidade de controle completo do ar-condicionado via web válida a proposta de acesso remoto, superando a limitação de alcance dos controles infravermelhos tradicionais.

4.3 INTEGRAÇÃO DO ESP32 COM JAVA COM SPRING BOOT WEB

A integração do ESP32 com o *back-end* desenvolvido em Java com Spring Boot Web estabeleceu este último como a principal plataforma de gerenciamento de dados e orquestração de comunicações. O Spring Boot foi configurado para receber as requisições HTTP originadas do ESP32 e da interface web, processar os dados, aplicar regras de negócio e encaminhar os comandos de controle correspondentes. Endpoints RESTful foram criados para gerenciar as operações de leitura e escrita de dados, bem como a lógica de automação. Essa integração demonstrou notável estabilidade e baixa latência, assegurando que os comandos fossem processados e executados em tempo hábil.

O Spring Boot Web provou ser uma ferramenta excepcionalmente eficaz para o desenvolvimento rápido de APIs robustas e escaláveis, graças à sua arquitetura baseada em microsserviços e à vasta biblioteca de funcionalidades. Sua flexibilidade para integração com o protocolo HTTP e a capacidade de criar lógicas personalizadas foram elementos cruciais para a inteligência e a automação do sistema. Além disso, a centralização da lógica de negócio no *back-end* Java permitiu desafogar o ESP32, otimizando o microcontrolador para se concentrar nas tarefas de baixo nível, como a comunicação infravermelha e a conectividade Wi-Fi.

4.4 IMPLEMENTAÇÃO DO BANCO DE DADOS TEMPORAL (MYSQL)

Um banco de dados MySQL foi instalado e configurado, atuando como o repositório central para os dados de telemetria coletados pelo ESP32, incluindo temperatura, umidade e o estado operacional do ar-condicionado. Esquemas de dados foram cuidadosamente definidos para otimizar o armazenamento e a consulta de séries temporais, garantindo eficiência. A ingestão de dados em tempo real foi exaustivamente testada e validada, com todos os dados sendo armazenados com seus respectivos *timestamps* precisos. Consultas básicas e avançadas foram realizadas com sucesso, demonstrando a plena capacidade de recuperar dados históricos para análises futuras e identificação de padrões.

A escolha do MySQL como banco de dados relacional foi justificada por sua robustez, ampla adoção e capacidade de lidar com volumes consideráveis de dados, além de sua integração facilitada com o Spring Boot. A disponibilidade de dados históricos é um recurso indispensável para a análise de padrões de uso, a subsequente otimização energética e a identificação precoce de quaisquer anomalias no comportamento do sistema. A integração simplificada com o *back-end* Java facilitou a inserção automática e confiável dos dados, tornando o processo de armazenamento robusto.

4.5 CONFIGURAÇÃO DO FRONT COM REACT PARA VISUALIZAÇÃO EM TEMPO REAL

Dashboards interativos foram desenvolvidos no *front-end* utilizando React, estabelecendo conexão com o *back-end* Java para obtenção dos dados. Adicionalmente, foram configurados alertas básicos para notificar o usuário sobre condições específicas, como a temperatura excedendo um limite predefinido. A visualização dos dados se mostrou fluida e em tempo real, fornecendo uma visão abrangente e clara do ambiente monitorado e do funcionamento do ar-condicionado.

O *React* complementa o *back-end* de forma altamente eficaz, transformando dados brutos em informações visuais compreensíveis e acionáveis. Os dashboards desenvolvidos podem servir como uma ferramenta de diagnóstico valiosa, permitindo

a identificação rápida de problemas ou a análise de tendências de uso ao longo do tempo.

4.6 AVALIAÇÃO DO FUNCIONAMENTO DO SISTEMA EM AMBIENTE REAL

O sistema foi implementado e operou continuamente em um ambiente real por um período de 4 horas. A estabilidade da comunicação entre os componentes (ESP32-Backend Java-Front-end *React*) foi consistentemente alta, com um número mínimo de falhas registradas, resultando em um *uptime* de aproximadamente 100%.

Um dos principais benefícios verificados foi o potencial impacto significativo na economia de energia, atribuído à capacidade do usuário de programar horários de desligamento ou de ajustar a temperatura remotamente, evitando o desperdício de energia com o ar-condicionado operando desnecessariamente. Para quantificar essa economia, considerou-se um ar-condicionado de 12.000 BTUs, cuja potência aproximada é de 3,52 kW ($12.000 \text{ BTUs} \times 0,000293071 \text{ kW/BTU}$). A análise comparativa do consumo energético em dois cenários distintos, com base em uma tarifa de energia de R\$ 0,711/kWh, revelou o seguinte:

Figura 37- Ar condicionado utilizado nos testes



- Cenário 1: Operação Contínua (24 horas/dia): O consumo diário seria de 84,48 kWh ($3,52 \text{ kW} \times 24 \text{ h}$), resultando em um custo diário de R\$ 60,04.

- Cenário 2: Operação Otimizada (9,32 horas/dia, conforme períodos de uso inteligente/programado): O consumo diário cairia para aproximadamente 32,81 kWh ($3,52 \text{ kW} \times 9,32 \text{ h}$), com um custo diário de R\$ 23,33.

Esta comparação demonstra uma diferença diária de consumo de 51,67 kWh, que se traduz em uma economia diária de R\$ 36,71. Projetando essa economia para períodos maiores, estima-se uma economia mensal de aproximadamente R\$ 1.101,30 (30 dias) e uma economia anual potencial de R\$ 13.399,15. Tais números evidenciam o valor agregado do sistema no que tange à otimização do uso energético e à redução de custos para o usuário.

A escalabilidade do sistema foi implicitamente validada pela sua arquitetura modular, que permite a adição de múltiplos dispositivos ESP32 para controlar diversos ares-condicionados ou outros aparelhos eletrônicos, expandindo o escopo da automação. Os custos de implementação foram considerados baixos, visto o uso predominante de *hardware* e *software* de código aberto ou com versões gratuitas, tornando a solução altamente acessível.

O custos do hardware completo levando em consideração o Modulo WiFi ESP32 Bluetooth 30 pinos, Modulo Emissor Infravermelho KY-005, caixa e cabo usb somam um valor de R\$86,80 o que representa um valor promissor de custo em relação a economia simulada de energia.

Figura 38- Protótipo do hardware completo montado.



A performance satisfatória do sistema em ambiente real valida sua proposta de valor para a automação residencial, demonstrando sua confiabilidade e eficácia no uso diário. A experiência do usuário, um fator crucial em soluções de IoT, foi considerada satisfatória, indicando que a interface e a resposta do sistema são

adequadas para o público-alvo. A capacidade de gerenciar o consumo de energia de forma inteligente representa um benefício tangível para o usuário, alinhando-se aos princípios da automação residencial e da sustentabilidade. A arquitetura baseada em tecnologias abertas e modulares garante a adaptabilidade e a expansão futura do sistema, permitindo a inclusão de novos dispositivos ou funcionalidades.

CAPÍTULO VI – CONCLUSÃO.

Ao longo da pesquisa e desenvolvimento, todos os objetivos específicos propostos foram satisfatoriamente alcançados. O *firmware* para o microcontrolador ESP32 foi desenvolvido com sucesso, habilitando a comunicação Wi-Fi e o envio preciso de comandos infravermelhos. A interface web, construída com React, demonstrou ser responsiva e intuitiva, proporcionando uma experiência de usuário fluida em diferentes dispositivos. A integração do ESP32 com o *back-end* em Java e Spring Boot Web se mostrou robusta, gerenciando eficientemente o fluxo de dados e orquestrando as comunicações. O banco de dados MySQL foi implementado com êxito para o armazenamento e consulta de dados históricos de operação e ambiente, enquanto o *frontend* em React foi configurado para uma visualização clara e em tempo real através de *dashboards* interativos.

A avaliação do sistema em um ambiente real comprovou sua estabilidade, com comunicação consistente e alta usabilidade da interface. Um dos resultados mais significativos foi a demonstração do expressivo potencial de economia de energia. Tomando como base um ar-condicionado de 12.000 BTUs e uma tarifa de R\$ 0,711/kWh (valor ilustrativo, sujeito às variações da concessionária local em Itumbiara, Goiás), demonstrou-se que a gestão inteligente do uso, reduzindo o tempo de operação de 24 horas para 9,32 horas diárias, pode gerar uma economia diária de aproximadamente R\$ 3671,00. Projetada mensalmente, essa economia atinge cerca de R\$ 1.101,30, e anualmente, a estimativa supera os R\$ 13.339,15. Tais cifras sublinham o impacto financeiro positivo da solução, além dos benefícios de sustentabilidade e conforto. A viabilidade, baixo custo de implementação e a escalabilidade da arquitetura modular, baseada em tecnologias abertas, foram igualmente validadas. Além disso, os custos de materiais se mostram vantajosos quando comparados a economia de energia simulada.

Em síntese, este trabalho resultou em um sistema funcional e eficaz que válida a aplicação da IoT para a automação residencial, oferecendo uma solução inteligente e acessível para o controle de ar-condicionado. A pesquisa contribui para o avanço das soluções de *Smart Home*, evidenciando como a integração de hardware e software pode otimizar o consumo de recursos e melhorar a qualidade de vida.

REFERÊNCIAS

AMERICAN SOCIETY OF HEATING, REFRIGERATING AND AIR-CONDITIONING ENGINEERS (ASHRAE). **HVAC Systems and Equipment Handbook**. 4. ed. Atlanta: ASHRAE, 2017.

ARDUINO. **Documentação Oficial da IDE Arduino**. [S.l.], [s.d.]. Disponível em: <https://www.arduino.cc>. Acesso em: 22 jun. 2026.

ASHTON, K. **Internet of Things: A Brief History**. [S.l.], 2009. Disponível em: <https://example.com>. Acesso em: 22 jun. 2026.

ATZORI, L.; IERA, A.; MORABITO, G. **The Internet of Things: A Survey**. **Computer Networks**, v. 54, n. 15, p. 2787-2805, 2010.

BOLTON, W. **Instrumentation and Control Systems**. 3. ed. Oxford: Elsevier, 2015.

CARRIER, W. **The History of Air Conditioning**. [S.l.]: Carrier Corp., 2009.

DBeaver. **Documentação Oficial**. [S.l.], [s.d.]. Disponível em: <https://dbeaver.io>. Acesso em: 22 jun. 2026.

DATE, C. J. **An Introduction to Database Systems**. 8. ed. Boston: Addison-Wesley, 2003.

DUBOIS, P. **MySQL Cookbook**. 3. ed. Sebastopol: O'Reilly, 2014.

ESPRESSIF SYSTEMS. **ESP32 Technical Reference Manual**. [S.l.], [s.d.]. Disponível em: <https://www.espressif.com>. Acesso em: 22 jun. 2026.

FIELDING, R. T.; TAYLOR, R. N. **Principled Design of the Modern Web Architecture**. **ACM Transactions on Internet Technology**, v. 2, n. 2, p. 115-150, 2002.

FLANAGAN, D. **JavaScript: The Definitive Guide**. 6. ed. Sebastopol: O'Reilly, 2011.

GELMAN, A. **SCADA Systems: Fundamentals and Applications**. New York: Springer, 2002.

GIT. **Documentação Oficial**. [S.l.], [s.d.]. Disponível em: <https://git-scm.com>. Acesso em: 22 jun. 2026.

GITHUB. **GitHub Guides**. [S.l.], [s.d.]. Disponível em: <https://github.com>. Acesso em: 22 jun. 2026.

GONÇALVES, R.; SILVA, L. **Redes de Comunicação para IoT**. São Paulo: Editora Tech, 2020.

GOSLING, J. et al. **The Java Language Specification**. 5. ed. Boston: Addison-Wesley, 2014.

GROOVER, M. P. **Automation, Production Systems, and Computer-Integrated Manufacturing**. 4. ed. London: Pearson, 2014.

INTERGOVERNMENTAL PANEL ON CLIMATE CHANGE (IPCC). **Climate Change 2021: The Physical Science Basis**. [S.l.], 2021.

JETBRAINS. **IntelliJ IDEA Documentation**. [S.l.], [s.d.]. Disponível em: <https://www.jetbrains.com/idea>. Acesso em: 22 jun. 2026.

LIGHT, R. A. **MQTT: A Protocol for the Internet of Things**. IEEE Internet of Things Journal, v. 4, n. 3, p. 1-12, 2017.

LOMBOK. **Project Lombok Documentation**. [S.l.], [s.d.]. Disponível em: <https://projectlombok.org>. Acesso em: 22 jun. 2026.

MAZIDI, M. A.; MAZIDI, J. G.; MCKINLAY, R. D. **The 8051 Microcontroller and Embedded Systems**. 2. ed. London: Pearson, 2007.

MEDEIROS, J. et al. ***Eficiência Energética em Sistemas de Climatização***. São Paulo: Editora Sustentável, 2020.

MICROSOFT. ***Visual Studio Code Documentation***. [S.l.], [s.d.]. Disponível em: <https://code.visualstudio.com>. Acesso em: 22 jun. 2026.

OGATA, K. ***Engenharia de Controle Moderno***. 5. ed. São Paulo: Pearson, 2010.

OLIVEIRA, P.; NASCIMENTO, R. ***Sustentabilidade e Automação Residencial***. Rio de Janeiro: Editora Verde, 2019.

SANTANA, C., ANDRADE, L., MELLO, B., SAMPAIO, J., BATISTA, E., PRAZERES, C. (2019). Teoria e Prática de Microserviços Reativos: Um Estudo de Caso na Internet das Coisas. 10.5753/sbc.481.6.03.

SANTOS, M.; OLIVEIRA, T. ***Controle Remoto por Infravermelho: Princípios e Aplicações***. Belo Horizonte: Editora Connect, 2018.

SILVA, A. et al. ***IoT e Automação Predial***. Porto Alegre: Editora Smart, 2020.

SPRING. ***Spring Boot Documentation***. [S.l.], [s.d.]. Disponível em: <https://spring.io/projects/spring-boot>. Acesso em: 22 jun. 2026.

ZHANG, Y.; WANG, L.; LI, X. ***Predictive Maintenance for HVAC Systems***. IEEE Transactions on Industrial Informatics, v. 15, n. 1, p. 1-10, 2019.