UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

Smart-Air-Web: Uma solução *full-stack* para o controle distribuído de sistemas de ar-condicionado

Felipe de Oliveira

São Carlos – SP

Smart-Air-Web: Uma solução *full-stack* para o controle distribuído de sistemas de ar-condicionado

Felipe de Oliveira

Orientador: **Prof. Dr. Júlio Cezar Estrella**

Monografia final de conclusão de curso apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como requisito parcial para obtenção do título de Engenheiro de Engenharia de Computação.
Área de Concentração: Sistemas Distribuidos

USP – São Carlos
Novembro de 2021

Oliveira, Felipe de
Smart-Air-Web: Uma solução *full-stack* para o controle distribuído de sistemas de ar-condicionado / Felipe de Oliveira. - São Carlos - SP, 2021.
48 p.; 29,7 cm.

Orientador: Júlio Cezar Estrella.
Monografia (Graduação) - Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos - SP, 2021.

1. IoT. 2. Ar-Condicionado. 3. Aplicação Web. I. Estrella, Júlio Cezar. II. Instituto de Ciências Matemáticas e de Computação (ICMC/USP). III. Título.

Este trabalho é dedicado aos meus pais Solange e Douglas,a minha irma Amanda e a minha namorada Carolina

Sem eles eu nunca teria chegado até aqui.

AGRADECIMENTOS

Agradeço a Deus, pela minha vida, a oportunidade de cursar a universidade dos meus sonhos e por me dar forças para atravessar todos os momentos de dificuldades enfrentados ao longo da vida na graduação.

Aos meus familiares que me apoiaram incondicionalmente durante a jornada na universidade. Em especial, minha irmã Amanda Seixas Oliveira, meus pais Solange de Souza Lima Oliveira e Douglas José de Oliveira que não pouparam esforços de trabalho e incentivos para que um pudesse cumprir esta jornada. Ao meu tio Roberto Lima *(in memorian)* que sempre me apoiou.

À minha namorada, Carolina de Castilho Paneque Garcia, que com sua graça e sua doçura, sempre me apoiou para vencer obstáculos difíceis durante o curso da graduação e concedeu companhia para bons momentos de distração durante a vida universitária.

Ao meu amigo Pedro Cornachioni, com quem moro desde a chegada à São Carlos, que concedeu cumplicidade em momentos ruins e bons, dentro e fora da universidade. E aos meus amigos Ivan Sousa, Lucas Nobrega, João Pedro Torrezan e Gabriel Ribeiro pela amizade e cumplicidade criada durante a convivência no curso juntos.

Aos professores, pelo compartilhamento de conhecimentos e pelas correções que permitiram minha evolução durante o meu processo de formação profissional. Em especial, agradeço ao Prof. Dr. Júlio Cezar Estrella pela disponibilidade da orientação e pelo acesso ao Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC) e seus recursos computacionais para que este trabalho fosse realizado. Agradeço também ao Cairo Neves por ter me auxiliado com o desenvolvimento deste trabalho.

“Corte sua própria lenha. Assim, ela aquecerá você duas vezes.”
(Henry Ford)

RESUMO

OLIVEIRA, F.. ***Smart-Air*-Web: Uma solução *full-stack* para o controle distribuído de sistemas de ar-condicionado**. 2021. 48 f. Monografia (Graduação) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

A inclusão de dispositivos inteligentes para controlar aparelhos de ar-condicionado (AC) antigos remotamente, através da rede Wi-Fi, concede uma alternativa barata para gerenciamento de sistema de Internet das Coisas (IoT). O trabalho de automação dos AC antigos é extenso, partindo da confecção de uma arquitetura de hardware que aciona comandos nos AC via sinal infravermelho, bem como a necessidade de uma aplicação para controle desses aparelhos. Uma solução para gestão remota dos sistemas de controle de refrigeração é uma aplicação Web intuitiva que permite aos usuários enviar comandos aos AC à distância pela Internet, por meio de diferentes dispositivos computacionais. Esta monografia está vinculada a um projeto, que converte AC convencionais em dispositivos *smart*. Portanto, este trabalho tem como objetivo complementar o sistema de controle dos AC, fornecendo a arquitetura e uma aplicação Web para controle desses dispositivos compatível com *smartphones*, *tablets*, computadores pessoais, contribuindo para minimizar o desafio da heterogeneidade de dispositivos de controle IoT ao propor um *Front-end* leve, de fácil navegabilidade e consistência operacional. Como resultado do trabalho desenvolvido, são apresentados os testes de controle do AC via internet e experiência com usuário do laboratório.

Palavras-chave: IoT, Ar-Condicionado, Aplicação Web.

ABSTRACT

OLIVEIRA, F.. ***Smart-Air*-Web: Uma solução *full-stack* para o controle distribuído de sistemas de ar-condicionado**. 2021. 48 f. Monografia (Graduação) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

The inclusion of smart features to remotely manage older air conditioning (AC) units via Wi-Fi provides an inexpensive alternative to control. Extensive work is necessary to automate older appliances, beginning with the creation of the hardware architecture to control the AC units via infrared signals and going on to the need for a specific control application. One solution for the remote management of refrigeration control systems is an intuitive Web application that allows users to control the refrigeration devices over the internet and from a distance. This paper is linked to a project that converts conventional AC units into smart appliances. Thus, the purpose of this work is to complement the AC control system, supplying it with an architecture and a Web application to control the appliances which is compatible with smartphones, tablets, and desktop computers, resolving the challenge of IoT control for heterogeneous devices and permitting a light front-end interface with easy navigation and operational consistency. As a result of the work done, the AC control tests via the Internet and the laboratory user experience are presented.

Key-words: IoT, Air Conditioning, Web Application.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AC ..... Ares-Condicionados
API ..... *Application Programming Interface*
CoAP .... *Constrained Application Protocol*
GUI ..... Interfaces Gráficas
HTTP .... *HyperText Transfer Protocol*
HVAC .... *Heating, Ventilation, and Air Conditioning*
IoT ..... *Internet of Things*
IU ..... Interfaces de Usuário
JSON .... *JavaScript Object Notation*
LaSDPC .. Laboratório de Sistemas Distribuídos e Programação Concorrente
M2M ..... *Machine-to-Machine*
MQTT ... *Message Queue Telemetry Transport*
MVC ..... *Model-View-Controller*
OASIS ... *Organization for the Advancement of Structured Information Standards*
QoS ..... *Quality of Service* (Qualidade de Serviço)
rem ..... *Root Element*
RNN ..... *Random Neural Network*
SAE ..... Sistemas de Automação de Edifícios
SCA ..... Sistema de Controle de AC
SSR ..... *Server Side Rendering*
URL ..... *Uniform Resource Locator*
USP ..... Universidade de São Paulo
WC ..... Web Componentes

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contextualização

Nos últimos anos, a Internet das Coisas (do inglês, *Internet of Things* (IoT)) se tornou uma das tecnologias mais importantes no cotidiano do século XXI. Diversos objetos do cotidiano - industriais, agrícolas e residenciais - podem se conectar à Internet e prover funcionamento inteligente. A combinação desses objetos conectados, possibilita projetos de Sistemas de Automação de Edifícios (SAE), que visam alcançar eficiência dos custos com energia consumida pelos espaços dos edifícios por intermédio de controle automático e remoto do ambiente interno gerenciando sistemas de aquecimento, ventilação, iluminação e ar-condicionado utilizando sensores e dispositivos de acionamento interligados em uma rede sem fio com acesso à Internet. Além da otimização do consumo de energia, os SAE trazem conforto e praticidade aos usuários humanos, ao permitir o controle remotamente via aplicações mobile i.e. Web (DRöGEHORN *et al.*, 2017)).

Contudo, o processo de implementação de sistemas IoT apresenta grandes desafios. Um dos principais é lidar com equipamentos legados, aparelhos antigos que já estão instalados no ambiente cuja substituição é custosa. A substituição desses equipamentos antigos por novos Ares-Condicionados (AC) inteligentes é caro e, em caso de instituições públicas como a Universidade de São Paulo (USP), que requer processo de licitação e orçamento compartilhado com diversas obrigações para funcionamento da universidade, levará muitos anos para ocorrer aquisições de novos equipamentos. Face a esse desafio, o projeto ao qual este trabalho está vinculado, apresenta uma solução mais barata e rápida, incluindo um sistema composto por sensores infravermelho e micro controladores para converter aparelhos de AC de legado em equipamento *smart*, possibilitando a sua interação em um ecossistema IoT. Tal ação de melhoria nesses AC, oferecem o benefício da economia de energia elétrica ao possibilitar maior controle dos dispositivos em funcionamento e abre a possibilidade de conforto aos usuários humanos controlar funções desses AC remotamente.

1.2 Motivação e Objetivos

Este trabalho tem por objetivo apoiar um projeto de IoT que converte AC de legado em dispositivos *smart*, por meio de um Sistema de Controle de AC (SCA) por sensores infravermelho

e micro controladores que possibilitam a sua interação em um ecossistema IoT. Portanto, este trabalho propõe a arquitetura de uma solução Web para controle remoto do sistema de automação, apresentando o desenvolvimento de uma aplicação *Front-end Web* que é integrado com SCA. Esta integração é realizada por um serviço de comunicação que utiliza protocolos HTTP e MQTT, a fim de possibilitar que usuários consigam acionar comandos nos AC por um *Browser* em qualquer computador pessoal ou mobile pela internet, sem a necessidade de estar presente no ambiente refrigerado pelo AC. O *Front-end* é desenvolvido utilizando conceitos de *Web components*, *context*, possibilitando escalabilidade de mais equipamentos convertidos em *smart* e transparência para equipes de desenvolvimento futuras.

1.3 Organização

Esta monografia do projeto de graduação está estruturada da seguinte forma: o Capítulo 2, aborda os métodos, técnicas e tecnologias utilizadas no desenvolvimento deste projeto. Em seguida no Capítulo 3, são apresentados os trabalhos relacionados que sumarizam uma análise relevante para o desenvolvimento deste trabalho. No Capítulo 4, são apresentadas as atividades desenvolvidas neste trabalho, abordando a compreensão dos requisitos do sistema de automação dos ares-condicionados (AC), a descrição da arquitetura da aplicação Web proposta e o fluxo de desenvolvimento do protótipo *Front-end*, bem como a sua integração com o *hardware* de borda que comanda o AC. No Capítulo 5 são discutidos os resultados obtidos. Por fim, no Capítulo 6 é apresentada a conclusão do trabalho, bem como uma listagem de trabalhos futuros e a relação do projeto com o curso de graduação.

Capítulo 2

MÉTODOS, TÉCNICAS E TECNOLOGIAS UTILIZADAS

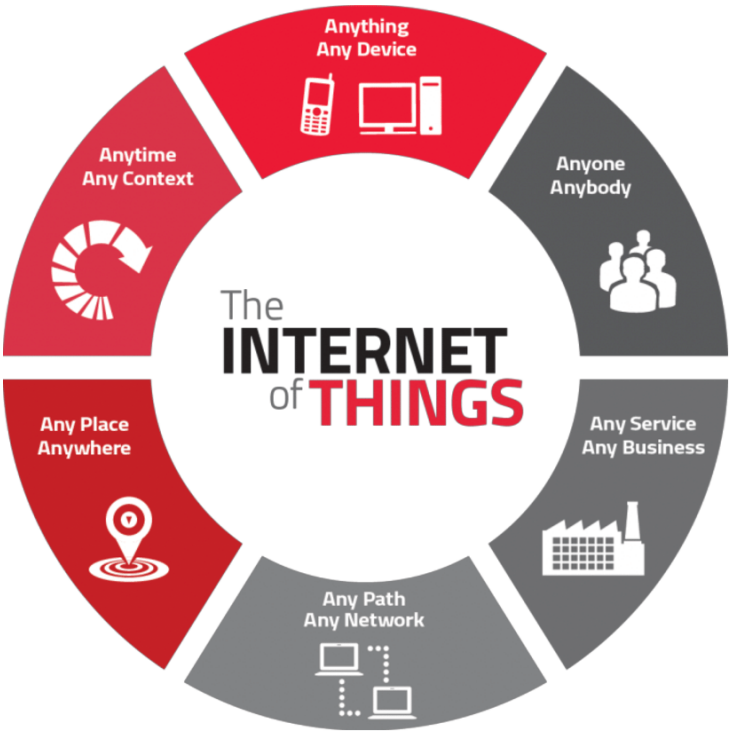
2.1 Considerações Iniciais

Nesta seção pretende-se apresentar conceitos referentes a IoT, os elementos computacionais utilizados no contexto deste trabalho, ou seja, o NodeMCU ESP32, microcontrolador utilizado para controle do ares condicionados, o *Cluster Taurus*, que provém sustentação para a comunicação entre o micro controlador e o *Front-end*. Apresenta também as linguagens de programação e ferramentas utilizadas no desenvolvimento do protótipo, bem como os protocolos utilizados na comunicação entre o dispositivo controlado e o *Front-end*.

2.2 Internet das Coisas (*Internet of Things*, IoT)

A Internet das Coisas, do inglês *do inglês, Internet of Things (IoT*), vem se popularizando com a expansão tecnológica que o mundo passa. Em (SÁTYRO *et al.*, 2018) IoT é definido pela concepção de se conectar diferentes objetos utilizados no cotidiano, como carros, eletrodomésticos e máquinas industriais à Internet, como representado na Figura 3. Assim, é possível acessá-los remotamente por computadores pessoais, *tablets*, celulares ou outros por meio da Internet. Formulada em 1999, por uma *startup* do empresário britânico Kevin Ashton, teve uma ideia que descreveu a Internet das coisas como um sistema que conectaria computadores com sensores onipresentes com uma intensa troca de dados. Em meados dos anos 2009, a quantidade de dispositivos à rede já era maior que a população mundial, consolidando também outra terminologia para IoT, a “internet de tudo” (WITKOWSKI, 2017).

Figura 1 – Definição de Internet das Coisas.

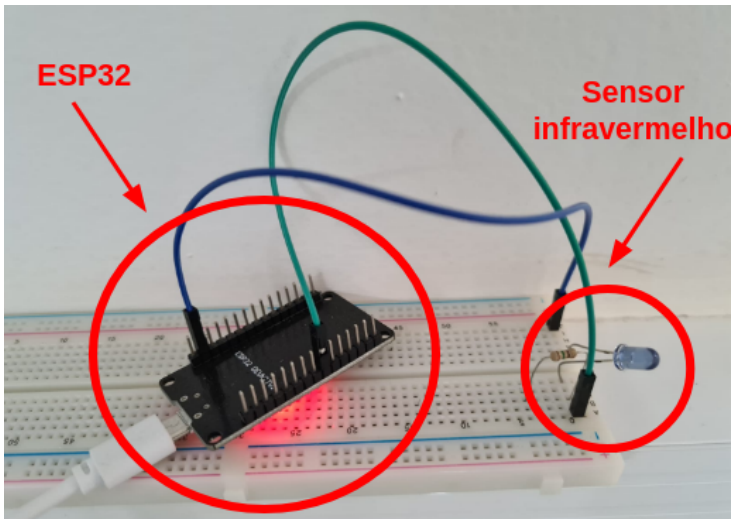


Fonte: (RAJPUT, 2020)

2.3 Microcontrolador ESP32

Os módulos microcontroladores são opções baratas na implementação de projetos de IoT. Neste trabalho em questão, é utilizado o ESP32, sistema em um chip, do inglês *System on a Chip* (SoC), em outras palavras, é um único chip que é capaz de conter memória, CPU e módulo de conectividade (Wi-Fi e *Bluetooth*) (BERTOLETI, 2019). A aplicação do ESP32 no projeto da conversão do ares-condicionados antigos em *smart* para controle de envios e recebimento de comando via sensores de infravermelhos acoplados em sua interface de comunicação foi vantajosa, pois o módulo possui dimensões pequenas para alocação embarcada em espaço reduzido, é programável em diversas linguagens de programação como C e C++, possui custo baixo e baixo consumo de potência em seu funcionamento. Na Figura 2, é apresentado o circuito do sistema IoT *Smart-Air*, utilizado neste projeto. Nele estão presentes uma ESP32, sensor infravermelho para envio do sinal ao AC e outros componentes elétricos (TRAZZI, 2021).

Figura 2 – Foto do circuito *Smart-Air* no laboratório.



Fonte: Elaborado pelo autor.

2.4 Protocolos de Comunicação

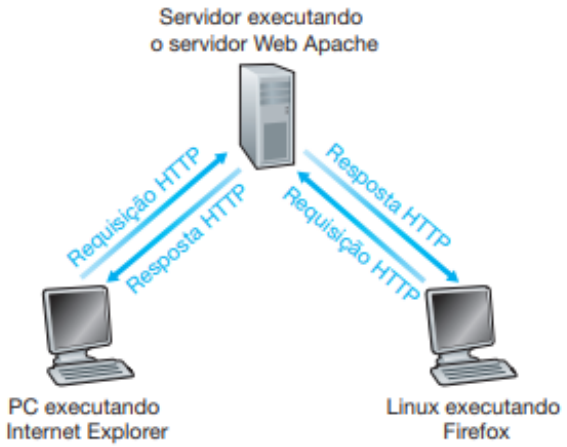
2.4.1 Protocolo de Transferência de Hipertexto (do inglês, HyperText Transfer Protocol (HTTP))

Em (KUROSE; ROSS, 2013) o protocolo *HyperText Transfer Protocol* (HTTP) é delineado como sendo uma camada de aplicação da Web, onde dois programas, o cliente e o servidor, comunicam-se entre si por meio da troca de mensagens. O protocolo estabelece a estrutura da mensagem e a maneira como ela será publicada e recebida pelo receptor. No contexto da Internet, um *browser* (cliente) troca objetos Web (páginas inteiras ou atributos) com um servidor Web e este, por sua vez, é localizado através de um endereço *Uniform Resource Locator* (URL). Quando um usuário deseja carregar uma página Web, o *browser* envia uma requisição HTTP para um servidor (via endereço URL), que recebe a mensagem e retorna a resposta com os objetos requisitados.

2.4.2 Transporte de Filas de Mensagem de Telemetria (do inglês, Message Queue Telemetry Transport (MQTT))

Criado no final da década de 90 nos laboratórios da *IBM* (YUAN, 2017), o *Message Queue Telemetry Transport* (MQTT) é um protocolo de comunicação padronizado pela *Organization for the Advancement of Structured Information Standards* (OASIS)). Possui lógica de operação simples e é muito aplicado em projetos de IoT pelo baixo consumo de potência elétrica e baixo consumo de banda. O MQTT funciona no mecanismo *publish/subscribe* com comportamento assíncrono, ou seja, é um protocolo de comunicação em que o emissor e receptor não operam sincronamente, evitando que o emissor se mantenha no aguardo da resposta de uma

Figura 3 – Fluxo de comunicação com o protocolo HTTP.

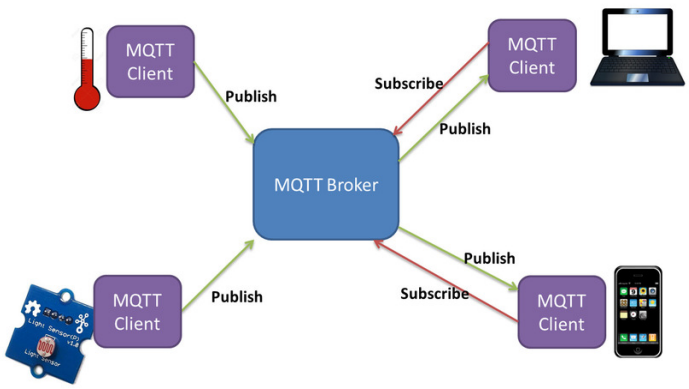


Fonte: (KUROSE; ROSS, 2013).

mensagem enviada ao receptor, favorecendo seu uso no ecossistema IoT em que em muitos casos o ambiente de operação dos sensores de borda é hostil com instabilidade na disponibilidade de Internet.

As mensagens enviadas com protocolo MQTT são compostas de dois elementos, a mensagem que pode ser um *JavaScript Object Notation* (JSON)) e o tópico que é a identidade dos dispositivos de destinação das mensagens. O *Broker* é o sistema de gerenciamento das mensagens transportadas pelo protocolo MQTT, recebendo as mensagens geradas pelo *publishers* e encaminhando-as a um cliente que consumirá a mensagem, *subscribers*. O *Broker* pode intermediar mensagens de um computador, componente eletrônico e/ou microcontrolador, identificar o tópico de destino e orientar o fluxo da mensagem até o receptor (*subscriber*), que também pode ser um dispositivo, possibilitando assim, a interação *Machine-to-Machine* (M2M). Na Figura 4 há uma ilustração do fluxo de comunicação MQTT.

Figura 4 – Fluxo de comunicação com o protocolo MQTT.



Fonte: (KODALI, 2016).

2.5 Recursos e técnicas utilizadas para desenvolvimento do *Front-end Web*

2.5.1 Biblioteca React Js

React Js é um biblioteca do *JavaScript* para construção de Interfaces de Usuário (IU). Foi desenvolvida pelo *Facebook* em 2013, e é atualmente uma *lib open-source*, utilizada mundialmente em aplicações de grandes empresas como *Instagram*, *AirBnB*, *Netflix* e *Yahoo* (GEEKHUNTER, 2019). Esse sucesso de adoção do React é justificado pela qualidade e eficiência das Interfaces Gráficas (GUI) geradas a partir do *framework*, sendo categorizada como V pela *Model-View-Controller* (MVC) (HOSTINGER, 2021).

A utilização do React também é vantajosa pela praticidade de aprendizado, visto que ele utiliza a extensão .JSX, uma sintaxe expandida do *JavaScript* que permite combinar HTML com *JavaScript*, bem como, a atualização e renderização exclusiva dos componentes modificados durante o desenvolvimento das aplicações.

O desenvolvimento de grandes sites com equipes numerosas de desenvolvedores e a manutenção dessas aplicações baseadas em React são facilitadas pelo suporte ao conceito de Web Componentes (WC). A criação de WC introduz uma forma moderna de encapsulamento de frações das aplicações com a possibilidade de fluxo de dados entre componentes pela implementação de *Contexts*. Com o React também é possível desenvolver aplicativos *mobile* e, assim, favorecer o desenvolvimento de sistemas multiplataformas (SOURCE, 2021).

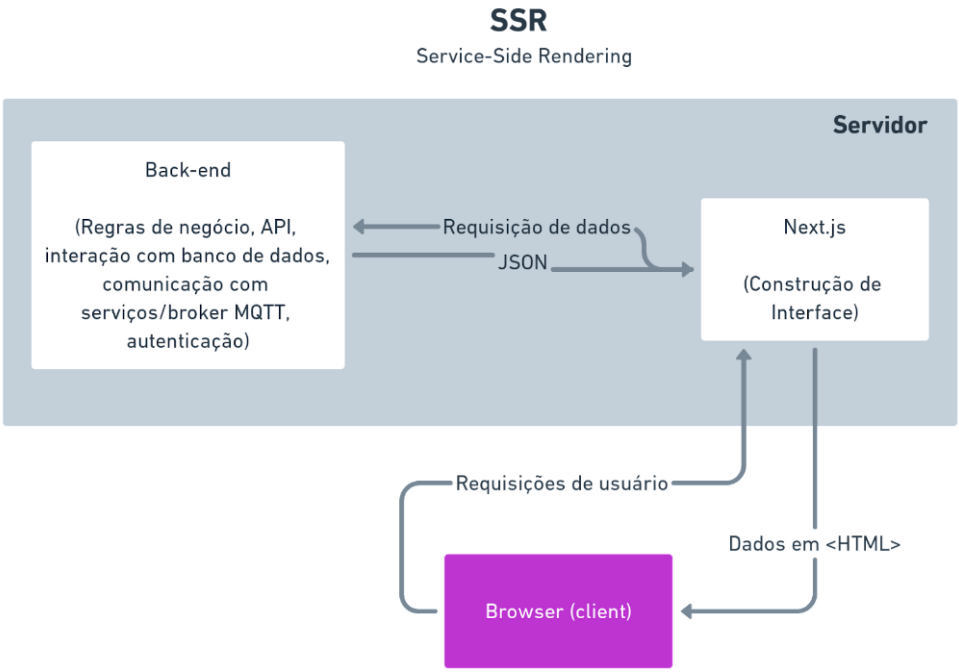
2.5.2 Framework Next.Js

No mundo do desenvolvimento *Front-end* busca-se cada vez mais construir interfaces com *design* de alta qualidade e eficiência de desempenho. *Next.Js* é definido em (GEEKHUNTER, 2021) como um *framework* para React que acrescenta diversas funcionalidades para desenvolvimento de IU. As aplicações Web tradicionais utilizam a arquitetura *client-side* que carrega todo o *JavaScript* e o *bundle* antes da apresentação completa de uma página, o que gera lentidão e desconforto aos usuários durante uma interação humano-computador (ALURA, 2021).

Como uma opção mais eficiente em tempo de carregamento e baixo consumo de largura de banda, o *Next.Js* apresenta o conceito *Server Side Rendering* (SSR). O esquema apresentado na Figura 5, demonstra o seu funcionamento, onde as renderizações de interfaces são feitas pelo lado do servidor, ou seja, quando um cliente solicita ao *Back-end* uma página, o *Next.Js* solicita os dados ao *Back-end*, que são retornados JSON e, então, o *Next.Js* renderiza os dados, retornando-os em formatos HTML para o cliente. Assim, com a renderização no servidor, diminui-se o tempo de carregamento da aplicação no *Browser* (cliente), uma vez que o custo computacional está relacionado à parte do próprio servidor, consumindo menos recursos dos usuários. Além disso, o *Next.Js* implementa uma série de outros recursos, como: suporte a sistema

de rotação, implementação de aplicações *Back-end*, possui CSS modularizado e *Fast Refresh*, que atualiza componentes específicos de uma página Web sem a necessidade de atualizá-la por inteiro, mantendo assim os valores das variáveis (VERCEL, 2021).

Figura 5 – Fluxo de requisições *Server-Side Rendering* (SSR).



Fonte: Elaborado pelo autor.

2.5.3 TypeScript

O *Typescript* foi criado em 2012 pela *Microsoft* com a missão de acrescentar recursos ao *JavaScript*. *Typescript* é definido em (GEEKHUNTER, 2021) como uma linguagem *open-source* que complementa recursos adicionais ao *JavaScript*. Suas principais vantagens de uso são a tipagem estática, ou seja, aquele em que o tipo das variáveis são visíveis no código e orientação a objetos - o *JavaScript* nativo, permite somente programação estruturada, oferecendo assim mais recursos para aplicações mais completas e melhora no processo de desenvolvimento em códigos fontes extensos com muitos desenvolvedores trabalhando em conjunto(ABREU, 2017).

A biblioteca React, baseada também em *JavaScript*, pode ser utilizada em conjunto com *Typescript* durante o desenvolvimento de aplicações Web. Ao criar projetos React, basta incluir o *Typescript* nas configurações e utilizar as extensões de arquivos .ts ou .tsx.

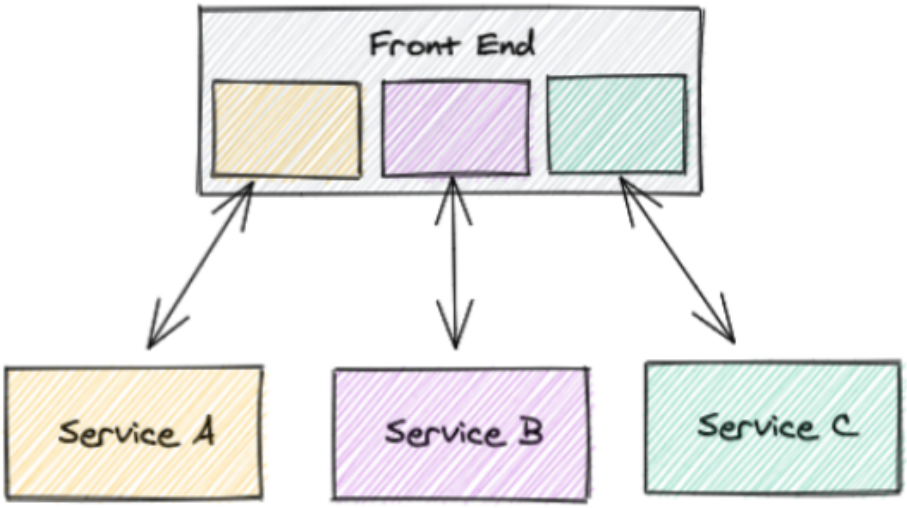
2.5.4 Micro Front-end

Introduzido pela primeira vez em 2016 no *Technology Radar* da *ThoughtWorks* [1], a técnica micro *Front-ends* consiste na utilização da arquitetura de microsserviços que combinados

[1] ThoughtWorks

compõem o *Front-end*, conforme ilustado em Figura 6. Com ela é possível que o código de cada serviço seja independente e auto-contido, ou seja, privado de variáveis globais e compartilhamento de estado, possibilitando a utilização de recursos do *browser* ao invés de API customizadas. Os times de desenvolvimento são capazes de atuar mutuamente em um projeto desenvolvendo serviços independentes e consistência de UI/UX ao permitir aos usuários uma experiência transparente entre as micro aplicações (MOVILE, 2021).

Figura 6 – Arquitetura Micro *Front-end*.



Fonte: (EBERHARDT, 2021)

2.6 *Cluster Taurus*

O *Cluster Taurus* [1] apresentado na Figura 7 é a infraestrutura presente no LaSDPC que hospeda os serviços em nuvem utilizados nos trabalhos de IoT do laboratório. Para este projeto, ele hospeda o *Broker Aedes* situado no endereço andromeda.lasdpc.icmc.usp.br responsável por suportar o fluxo de mensagens entre a ESP32 e o serviço de postagem de mensagens ao MQTT pelo lado do *Front-end* por meio de acesso à Internet e comunicação pelo protocolo HTTP. Este serviço em nuvem foi benéfico para o desenvolvimento do *Front-end* e posterior testes de comunicação com um computador pessoal distante do laboratório, ambiente de desenvolvimento do protótipo Web.

[1] http://infra.lasdpc.icmc.usp.br

Figura 7 – Foto do *Cluster Tauros* no LaSDPC.

Fonte: Elaborado pelo autor.

2.7 Considerações Finais

Este capítulo abordou conceitos utilizados em projetos de automação predial, tais como a definição de IoT, Microcontrolador ESP32, os protocolos de comunicação HTTP e MQTT. Também foram abordados os recursos para desenvolvimento do *Front-end* protótipo, *React Js*, *Next.Js* e *Typescript*. Por fim, é apresentado o *Cluster Taurus*, infraestrutura que hospeda serviços utilizados nos projetos IoT do LaSDPC. A apresentação desta seção é importante, pois introduz o entendimento de conceitos utilizados no tópico de desenvolvimento presente no Capítulo 4.

Capítulo 3

TRABALHOS RELACIONADOS

3.1 Considerações Iniciais

Nesta seção 3.2 são apresentados trabalhos científicos relacionados a esta monografia. Nestes artigos são feitas análises sobre as características, métodos e desafios dos sistemas de IoT atuais. Por fim, são apresentados em cada parágrafo conclusões que fomentam o desenvolvimento deste trabalho.

3.2 Trabalhos Relacionados

Em (BADER *et al.*, 2016) são discutidos os desafios associados à capacitação dos dispositivos IoT em processar análises no *Front-end*. São tratados os sistemas de IoT atuais baseados em *back-end*, as informações coletadas entre dispositivos vizinhos são direcionados a uma central de *back-end* para que, a partir do processamento do dado recebido, o sistema realize a tomada de decisão e, só assim, envie um comando para executar uma reação necessária. A arquitetura *back-end* gera inúmeros desafios de heterogeneidade, custo de latência, largura de banda, aplicações pouco eficientes e desafio da *Quality of Service* (Qualidade de Serviço) (QoS). Em um caminho contrário à arquitetura tradicional, o paradigma *Front-end* é apresentado pelo autor por meio de um novo *design* da estrutura IoT baseada nas tomadas de decisões no *Front-end*, que considera três perspectivas: o aplicativo, a conectividade e a colaboração. O paradigma *Front-end* habilita os dispositivos para terem inteligência de tomada de decisão *in-situ* baseado nas informações coletadas dos sensores próximos e conectados diretamente entre si, possibilita aplicações otimizadas, garante sistemas críticos que demandam latência baixa e alta largura de banda operando dentro da expectativa dos usuários. Para isso, são citados dois exemplos: *CROWD MANAGEMENT*: Conglomerado de câmeras que fornece uma resolução espacial ampla que identifica a multidão em HD fazendo distinção de gênero, idade e realização da análise em tempo real dessas informações. *CYBER-PHYSICAL SYSTEMS*: *Streaming* de vídeo ao vivo que possibilita equipes de segurança em campo de batalha terem maior campo de visão, visto o compartilhamento entre os próprios dispositivos do *cluster* e a central de controle simultâneo.

Em (ADIONO *et al.*, 2018) é apresentado um aplicativo para sistema operacional *mobile Android*, que visa, por meio de uma arquitetura IoT, acessar e controlar eletrodomésticos de

uma casa inteligente, minimizando o consumo de energia elétrica. Por meio de um sistema dividido em dois ambientes (interno/externo), o aplicativo consulta *status* e altera-os com base em programação pré-determinada. No ambiente interno é apresentado um subsistema em nós com *host* e roteadores para acesso à Internet. Já no ambiente externo, tem-se um servidor em nuvem e um aplicativo cliente, que acessa o *back-end* em nuvem. Toda a comunicação entre os ambientes é baseada na criptografia RSA e AES. O fluxo de dados do sistema é baseado no tipo de dados JSON, que é manipulado pelo aplicativo utilizando o protocolo de comunicação AMQP e o intermediário de mensagem *RabbitMQ* para interagir com o *back-end* em nuvem do sistema. Com o aplicativo *mobile*, o sistema permite um controle remoto, semelhante ao trabalho proposto nesta monografia.

Em (VERMA *et al.*, 2019) é feito um estudo do cenário atual da tecnologia IoT em edifícios inteligentes. São introduzidas as camadas de um sistema de IoT, passando pelo protocolo de comunicação LoRa que tem vantagens de longo alcance, baixo consumo de energia, comunicação bidirecional, segurança, padronização, alta capacidade de transmissão e baixo custo. A arquitetura analisada utiliza um controlador inteligente, *Random Neural Network* (RNN) para gerenciar a operação de *Heating, Ventilation, and Air Conditioning* (HVAC). O controle inteligente feito pelo RNN gera uma economia no consumo de energia do sistema de ventilação de 27,12% menor, se comparado aos controladores baseados em regras comuns. Há a observação de preferência pela escolha do sistema de autenticação baseada em *Kerberos* para oferecer segurança em uma vasta gama de dispositivos físicos que podem ser conectados no sistema IoT. É apresentada a perspectiva de uso do protocolo de comunicação *ZigBee* que apresenta baixo consumo de potência, porém com baixa taxa de transferência. Enquanto o Wi-Fi, protocolo mais conhecido e utilizado nos sistemas de IoT atualmente, este oferece maior taxa de transferência e alcance, porém tem como desvantagem um maior consumo de potência.

Em (DRöGEHORN *et al.*, 2017) é apresentado o desenvolvimento de um protótipo Web para *Front-end* de um sistema de automação residencial, que visa melhorar a gestão do consumo de energia residencial utilizando uma interface amigável e usualmente útil. Para isso, foi abordada uma análise qualitativa de dados para se chegar em requisitos funcionais que resultam em uma IU interativa e responsiva para o sistema de automação residencial. No desenvolvimento deste protótipo *Front-end* que interage com sistema heterogêneo IoT são combinadas as linguagens *JavaScript, CSS* e *HTML*. Por fim, é realizada uma pesquisa de satisfação de usabilidade da interface de automação residencial projetada para entender como os diferentes usuários, em idade e níveis de conhecimento técnico, se sentiram na interação com a interface do sistema. O resultado compilado mostra um alto grau de satisfação, e a interface se mostrou de fácil aprendizado e coberta de funcionalidades úteis.

Em (HEJAZI *et al.*, 2018) é apresentado uma visão geral sobre IoT, partindo para uma discussão sobre os protocolos adotados em algumas plataformas de gerenciamento mais conhecidas, disponíveis no mercado. Nele são explicados os componentes de uma plataforma IoT, desde

uma perspectiva dos dispositivos de sensoriamento que captam as condições físicas do ambiente até o controle de equipamentos. Apresenta-se como foco principal da análise os protocolos de comunicação *Constrained Application Protocol* (CoAP) e *Message Queue Telemetry Transport* (MQTT), discutindo a holística de cada um. Conclui-se que a escolha de uso desses protocolos dependem dos objetivos do sistema IoT, em que a perspectiva MQTT é baseada em TCP e favorece aplicações que utilizam WAN para sistemas remotos, enquanto a perspectiva CoAP é compatível com o protocolo HTTP e utiliza o protocolo UDP, útil em serviços baseados em Web que requerem transmissão em alta velocidade. Ao todo foram comparadas 20 plataformas de IoT, levando em consideração gerenciamento de dispositivos, recurso de integração, protocolos de segurança, protocolo para coleta de dados, tipo de análise e suporte de visualização para gerenciamento do sistema. O resultado do estudo, mostra grande heterogeneidade dos componentes de fabricante para fabricante, mas mostra que 95% desses sistemas utilizam integração *API REST*, concluindo-se que há uma forte tendência das arquiteturas IoT se aproximarem do uso de *Web Services*.

3.3 Considerações Finais

Este capítulo apresentou uma série de trabalhos relacionados com a proposta desenvolvida nesta monografia. Com base nos trabalhos estudados, esta monografia propõe um *Front-end Web* de controle, em meio aos desafios da heterogeneidade dos sistemas IoT e as vantagens de controle remoto e eficiente para os usuários dos sistemas de ar-condicionado do laboratório. Os trabalhos analisados nesta seção transmitem uma ideia de processo continuo de melhoria dos sistemas de IoT, o que permite trabalhos de evolução do *Smart-Air*.

Capítulo 4

DESENVOLVIMENTO

4.1 Considerações Iniciais

Este capítulo tem o objetivo de descrever a arquitetura do sistema de controle dos AC e apresentar o processo de desenvolvimento do protótipo *Front-end* Web. No capítulo é apresentado a fase inicial de compreensão do projeto *Smart-Air*, ao qual este trabalho o complementa, e as etapas de análise de requisitos, modelagem do *Front-end*, implementação do protótipo e integração com o *Broker Aedes*.

4.2 O Projeto

A proposta deste trabalho surgiu pela necessidade do projeto IoT denominado *Smart-Air* (TRAZZI, 2021), ao qual este trabalho está vinculado, ter uma arquitetura de aplicação para controle do AC à distância via Internet documentada e um protótipo *Front-end* eficiente em diversos modelos de computadores (*tablets, smartphone, desktop*) com interface amigável para acessibilidade de usuários de diferentes faixas etárias. Até o início deste projeto, o acionamento de comandos no AC convertidos em *smart* era feito por meio do Node-RED[2], uma ferramenta de programação com interface de blocos que possibilita conectar-se a *hardwares* com o protocolo MQTT e acionar manualmente o conceito *publish/subscribe*. Diferentemente do modelo utilizando Node-RED, a arquitetura deste projeto permite acrescentar diversos dispositivos *smart* sem gerar grande complexidade visual e, assim, habilita a escalabilidade futura do processo de automação dos AC a todo o campus universitário da USP em São Carlos-SP, onde o sistema *Smart-Air* está sendo desenvolvido e testado em laboratório.

Como explicado em mais detalhes na sequência, o desenvolvimento da aplicação Web de controle do AC é realizado baseado em *React Js* para a composição da aplicação com *Web components*. Também são utilizados *Typescript* para acrescentar facilidade ao desenvolvimento e proporcionar mais transparência na compreensão do código fonte do *Front-end* programado aos futuros desenvolvedores que irão evoluí-lo e *Next.Js* para abordagem da arquitetura SSR, visando apresentar um *Front-end* eficiente. O acionamento dos AC será feito por meio da interface e percorrerá a estrutura arquitetada na subseção 4.3.1.

[2] Node-RED

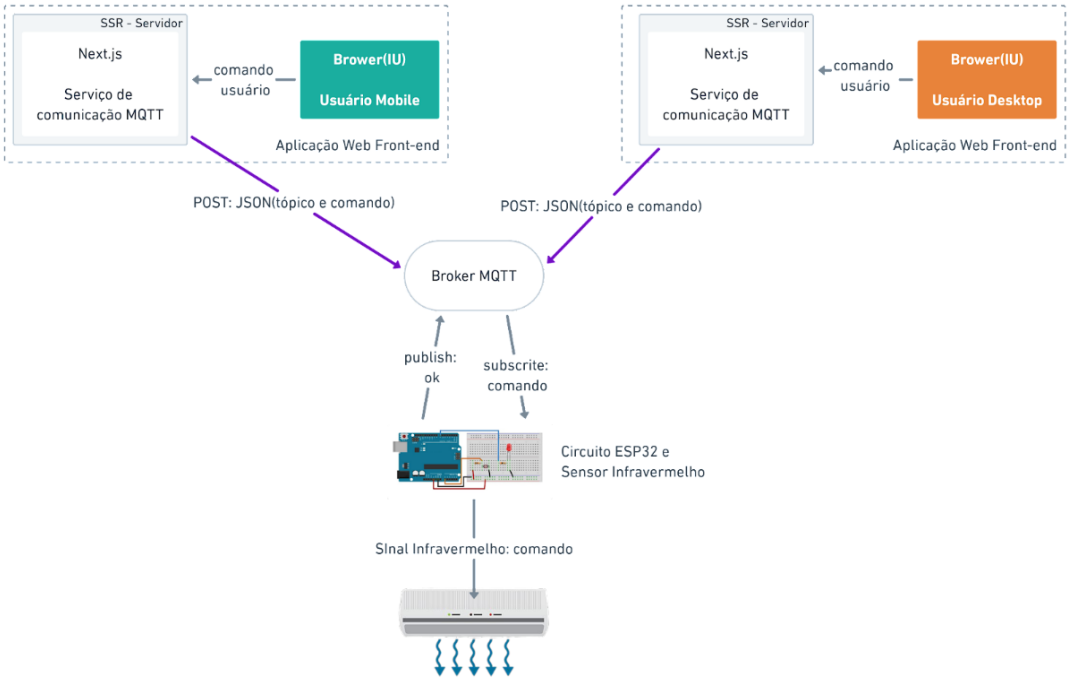
4.3 Atividades Realizadas

4.3.1 Modelagem da arquitetura para controle dos AC pela aplicação Web

Elaborada para possibilitar testes de acionamento dos comandos pela aplicação desenvolvida neste trabalho, a arquitetura apresentada nesta seção emprega o uso de micro serviço *Front-end* para integração da camada de aplicação Web com a camada do sistema *Smart-Air*. Tal serviço que é independente e implementado o componente do *Front-end* que renderiza as páginas pelo lado servidor (*Next.Js*), fez uso do protocolo HTTP para possibilitar a comunicação com o *Broker* MQTT, hospedado no *Cluster Taurus*.

A aplicação Web foi projetada para possibilitar o acionamento de botões implementados dentro de *Web components*, na qual cada componente corresponde à uma funcionalidade específica dos AC. Portanto, o fluxo do acionamento dos AC pela aplicação Web é representado na Figura 8. Na primeira etapa deste fluxo o usuário aciona os botões das funções na interface do *Smart-Air Web*, enviando o comando ao *Broker* MQTT via método POST, em que a mensagem enviada é um JSON que contém o tópico (endereço do equipamento *smart*) e os comandos que se deseja executar no AC. O *Broker*, por sua vez, receberá o *payload* da mensagem e irá reconhecer o tópico recebido e, assim, direciona os comandos à ESP32 que os processará e irá enviar um comando via sinal infravermelho para o AC alterar o seu estado de funcionamento.

Figura 8 – Esquema do fluxo de comunicação do protótipo *Smart-Air Web* até o aparelho ar-condicionado.



Fonte: Elaborado pelo autor.

4.3.2 Requisitos da aplicação Web

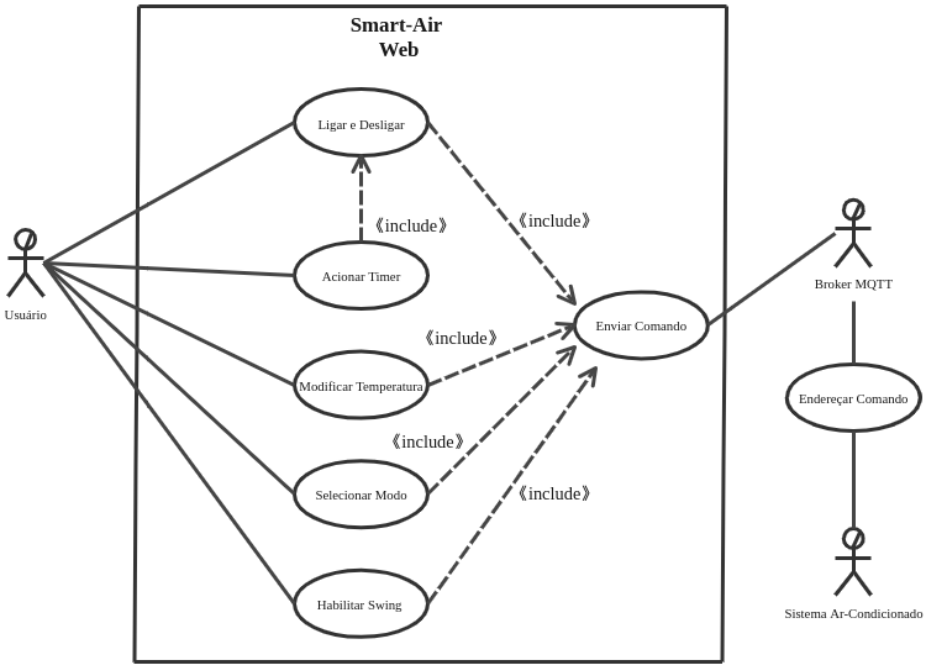
Antes de dar início à implementação do protótipo *Smart-Air Web*, foi realizada a engenharia de *software*, conforme proposto por Wazlawick (2019), para compreender os requisitos funcionais e não-funcionais que a aplicação requer para controle de um AC. Nesta fase foi feita uma inspeção do projeto *Smart-Air*, bem como uma breve investigação de funcionalidades presentes no controle remoto do AC *Fujitsu*, observado na Figura 9, em que os componentes de comando do protótipo foram inspirados.

Figura 9 – Foto do controle remoto do ar-condicionado testado no projeto *Smart-Air*.

Fonte: Elaborado pelo autor.

O diagrama de casos de uso da Figura 10, demonstra a ideia obtida dos requisitos de funcionalidade levantados para o sistema de controle operar sobre a arquitetura proposta na subseção 4.3.1. Nele, o Ator Usuário poderá interagir com as funcionalidades dos comandos similares ao do controle remoto do AC, sobre o sistema *Smart-Air Web* e o Ator *Broker* receberá o comando e o endereça ao Ator Sistema Ar-Condicionado.

Figura 10 – Diagrama de Casos de uso do *Smart-Air Web*.



Fonte: Elaborado pelo autor.

As funcionalidades apresentadas são compatíveis com uma alta gama de modelos e marcas de aparelhos de ar-condicionado. Portanto, a seguir são apresentadas as funcionalidades de controle implementadas no projeto:

- ***Power* (ligar e desligar)**: Funcionalidade liga e desliga o AC deve ser capaz de acionar o envio da mensagem ao dispositivo de *hardware* ESP32 que aciona o comando no AC, passando como dado o valor "0" para desligar e "1" para ligar;

- ***Timer***: A funcionalidade *Timer* deve ser capaz de possibilitar o ajuste de tempo para que seja enviada a mensagem de desligar ao dispositivo de *hardware* ESP32 que aciona o comando no AC. Ao programar o tempo máximo no relógio para desligamento do AC a contagem regressiva é acionada ao clicar no botão principal. Assim, logo quando terminar a contagem no instante 00:00, é acionada a funcionalidade desligar do componente *Power*, enviando a mensagem com o dado "0". Ao clicar no botão ou o temporizador completar a contagem, o *display* do componente *ON/OFF* deve atualizar, apresentando *OFF* para o caso de desligamento e *ON* para o caso de acionamento da funcionalidade. No término da contagem o componente *Power* também deverá ter o *display* atualizado para o *status OFF*, desligado;

- ***Temperature***: A funcionalidade *Temperature* deve permitir ajustar a temperatura momentânea no AC. Ao clicar no botão *down* ou *up*, deverá ser enviada uma mensagem

ao dispositivo de *hardware* ESP32 com o valor do *display* do componente na interface. Assim, será passado como dado o respectivo valor numérico da temperatura apresentada no *display*. Este valor deve estar contido no intervalo de 16 ºC à 30 ºC;

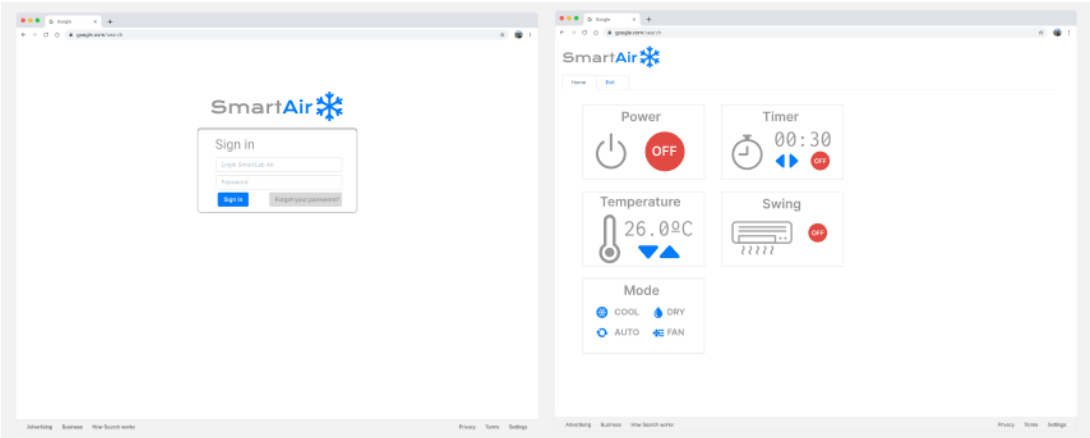
- ***Mode***: A funcionalidade *Mode* deve permitir selecionar um modo de operação momentânea do AC. Ao selecionar o modo será enviado ao AC um valor representativo de cada um dos modos implementados, portanto para o *COOL* será enviado o valor "0", *DRY* enviará o valor "1", *AUTO* enviará o valor "2" e *FAN* enviará o valor "3";

- ***Swing***: A funcionalidade *Swing* deverá ser capaz de habilitar ou desabilitar a função *swing* do AC, ao receber a interação de clique do usuário. Portanto, quando o usuário clicar no botão será enviada uma mensagem ao dispositivo de *hardware* ESP32, contendo o dado "0" para desligar a funcionalidade ou "1" para ligar a funcionalidade. Ao clicar no botão, o *display ON/OFF* deve atualizar, apresentando *OFF* para o caso de desligamento e *ON* para o caso de acionamento da funcionalidade.

Ademais, tem-se a preocupação dos requisitos não-funcionais que requerem uma aplicação eficiente para carregamento em diferente dispositivos, estável capaz de enviar as mensagens até chegar ao AC, deve ter usabilidade fluida e a manutenção em um componente não pode afetar o funcionamento de outro, exceto quando feito no componente *Power* que há vínculo de uso no componente *Timer*.

Essas funcionalidades descritas acima, foram desenhadas em um *mockup* das telas de *login* e controle de AC, apresentado na Figura 11, como proposta inicial de *layout* do *Smart-Air Web*. Esse recurso é importante, pois orienta a programação da interface com as características de *background* que serão aplicadas no css do projeto.

Figura 11 – *Mockup* do *Smart-Air Web*.



Fonte: Elaborado pelo autor.

4.3.3 Implementação

Dando início a implementação, foi criado um ambiente de desenvolvimento utilizando o *Visual Studio IDE* [3] para organização e edição dos códigos fontes, bem como um repositório para o *Smart-Air Web*[4] no *GitLab*[5] para versionamento e depósito desse códigos. Também foi necessário instalar o gerenciador de pacote *Yarn*[6], que proporciona um *workspace* para compilação da aplicação durante o desenvolvimento e, com as múltiplas portas, permite *HOST* para simular o servidor de consumo dos dados de autenticação em um *Application Programming Interface* (API) local.

Com o ambiente de desenvolvimento preparado, foi criado um projeto utilizando o *Framework Next.Js* junto à biblioteca *React Js* e o *Typescript*, visando aplicar boas práticas de desenvolvimento a partir das tipagens explícitas de funções e atributos. Esses recursos foram utilizados devido a fácil aprendizagem, disponibilidade de artifícios para desenvolvimentos ágeis, suporte longínquo, ampla comunidade para troca de conhecimento, fácil manutenção e possibilidade de escalabilidade do tamanho da aplicação. Outro motivo para uso do *Next.Js* no projeto é a construção do protótipo com a arquitetura SSR, trazendo mais eficiência no carregamento de páginas HTML, CSS e JS, uma melhor experiência do usuário se comparada a arquiteturas de aplicações Web tradicionais.

As funcionalidades anteriormente descritas na subseção 4.3.2, são implementadas com o React uma a uma em estruturas individuais *Web components*. Portanto, cada componente é desenvolvido em um arquivo de extensão .tsx individual. Isso permite que durante a programação de cada *Web component* a aplicação como o todo não seja interrompida, possibilitando também atualizações correntes em seu css e funções programadas, baseadas em *Typescript*. Esse modelo de desenvolvimento permite escalar novas funcionalidades sem comprometer a experiência de transparência do usuário. Também foi necessário utilizar *Context*, método de desenvolvimento em React para que dois ou mais *Web components*, possam se comunicar entre si. Essa técnica foi utilizada para que o temporizador do *Web component*, possa acionar a função *Power* (desligar) implementada no *Web component Power*, ao final da contagem regressiva do tempo programado pelo usuário.

Os demais componentes preservaram a propriedade de encapsulamento, comunicando-se somente com o serviço micro *Front-end* que envia as mensagens do acionamento de cada *Web component* para o *Broker MQTT Aedes*[6], que está hospedado no *Cluster Taurus*, acessado pela URL http://andromeda.lasdpc.icmc.usp.br, através do protocolo HTTP. O serviço conecta o *Front-end* com o *Broker* enviando alguns parâmetros em forma de JSON. São os parâmetros "*topic*" que indicam qual dispositivo IoT conectado ao *Broker* receberá a mensagem de comando, "*message*"

[3] Microsoft Visual Studio
[4] Repositório Smart-Air Web
[5] GitLab
[6] Yarn
[6] Aedes Broker MQTT

é o comando explicito das ações que serão enviadas pelo dispositivo de controle ESP32 ao ar condicionado, "*retain*" como "*true*" permite o *Broker* receber uma *flag* do dispositivo IoT se a mensagem foi recebida com sucesso e o campo "qos" que define a *Quality of Service* (QoS) igual a "2", ou seja, dentro do protocolo MQTT, o QoS indica que o dispositivo *subscribe* receberá a mensagem somente uma única vez, o que impede duplicação instantânea de mensagens. Na Figura 12 é possível ver um exemplo da declaração do parâmetro JSON passado ao *Broker*, onde "obj" é um elemento JSON com comandos múltiplos ao AC.

Figura 12 – Declaração do parâmetro para envio de mensagens ao *Broker* MQTT.

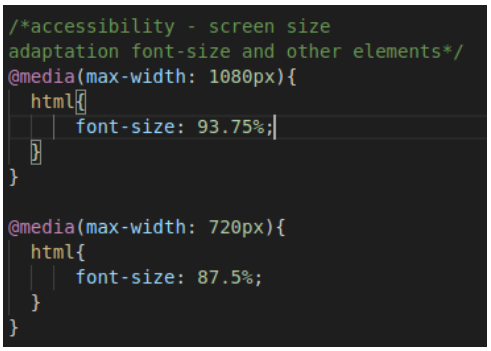
```
const params = JSON.stringify({
    "topic": 'test/topic',
    "message": obj,
    "retain": true,
    "qos": 2
  });
```

Fonte: Elaborado pelo autor.

O *Front-end* desenvolvido neste trabalho tem por premissa ser responsivo com boas práticas de acessibilidade da interface orquestradas pelo *The World Wide Web Consortium (W3C)*[8]. Portanto, buscou-se, na implementação da aplicação, customizar o CSS da interface aplicando iniciativas como contraste de tons diferentes de cores entre botões e alerta de status dos componentes (*ON/OFF*), botões ilustrados e com legendas para melhor compreensão de seus propósitos, aplicação de fontes de tamanho M ou L, ajuste de dimensão do contêiner e dos Web componentes da página declarados em *Root Element* (rem). Essa declaração permite aos elementos ajustarem seu tamanho conforme resolução da tela do usuário, além da propriedade @media para aplicar ajuste ao html da página Web, conforme a resolução da tela que a acessa. Na Figura 13 a seguir é demonstrado um exemplo utilizado no css do *Smart-Air Web*, no qual aplica um tamanho de 93,75% para elementos acessados em tela com resolução *width* até 1080px, convencional em computadores pessoais, e o ajuste de tamanho 87,5% para resolução *width* de até 720px, favorecendo uma melhor visualização em *tablets* e *smartphones*.

[8] World Wide Web Consortium

Figura 13 – Propriedade do CSS para ajuste do tamanho de elementos do HTML, a partir da resolução de tela dos usuários.



Fonte: Elaborado pelo autor.

4.4 Considerações Finais

Este capítulo descreveu em detalhes o processo de desenvolvimento da arquitetura da aplicação Web com o seu fluxo de dados entre o protótipo e o dispositivo IoT que controla o AC, bem como detalha as funcionalidades adotadas nessa solução. Foi especificado as etapas desde a criação do projeto *Front-end* até as técnicas para responsividade de acessibilidade na interface.

Capítulo 5

RESULTADOS

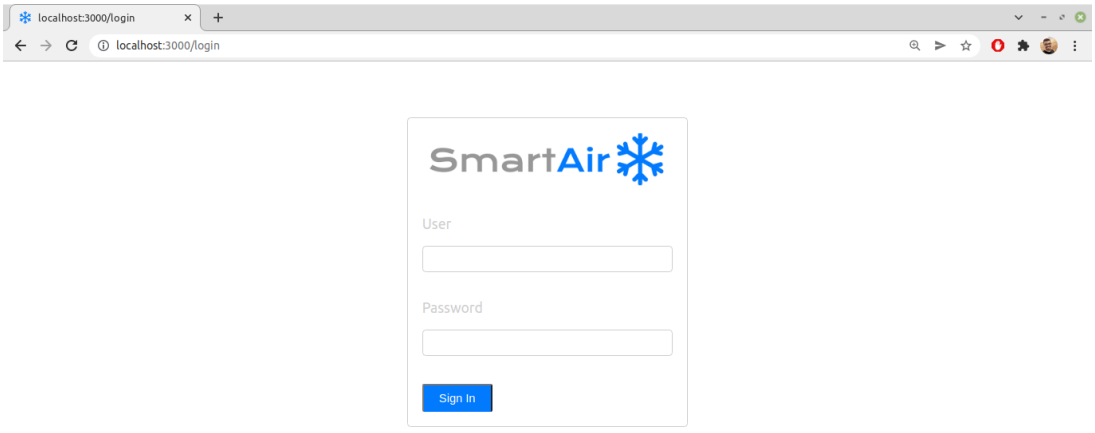
5.1 Considerações Iniciais

Este seção visa apresentar os resultados provenientes do trabalho desenvolvido no Capítulo 4.

5.2 Protótipo do *Front-end Smart-Air Web*

Um dos principais objetivos deste trabalho, consistia em produzir um protótipo *Front-end* do *Smart-Air* com uma interface amigável e desempenho adequado para controle de AC à distância, via Internet. Portanto, as Figura 14 e Figura 15 apresentam, respectivamente, as interface de *login* e controle dos AC da aplicação Web em um monitor de resolução 1440 x 900.

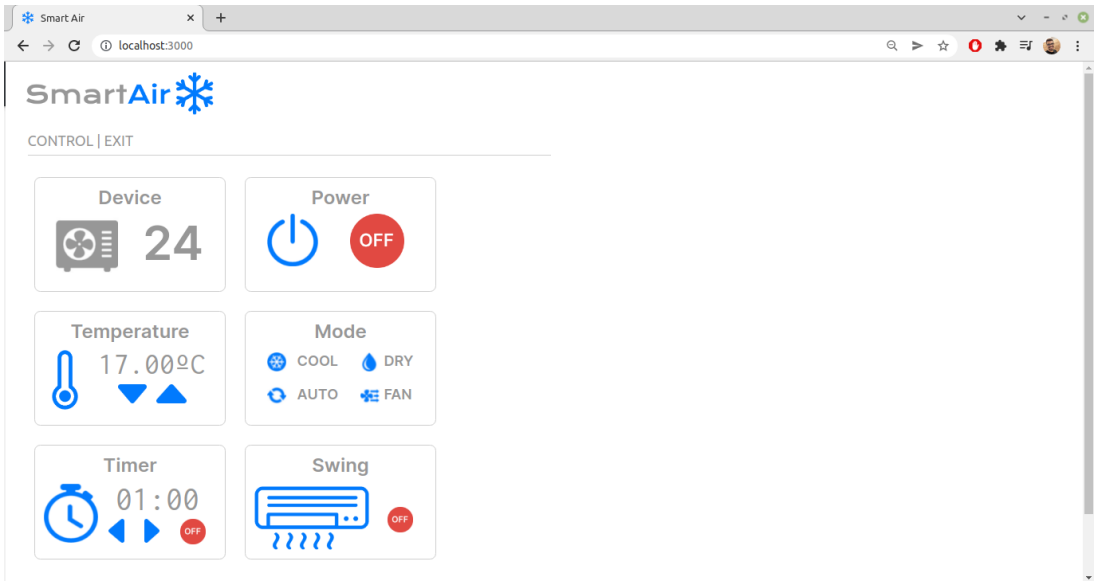
Figura 14 – Interface de login do *Smart-Air Web* no monitor de um *desktop*.



Fonte: Elaborado pelo autor.

A compatibilidade com outros dispositivos computacionais (*smartphones* e *tablets*) também foi a missão deste projeto, com técnicas de auto ajuste de tamanho dos componentes, conforme a resolução da tela dos usuários. Sendo assim, o acesso ao protótipo *Smart-Air Web* em dispositivos *mobile* é apresentado na Figura 16. Neste teste de renderização do *Front-end*

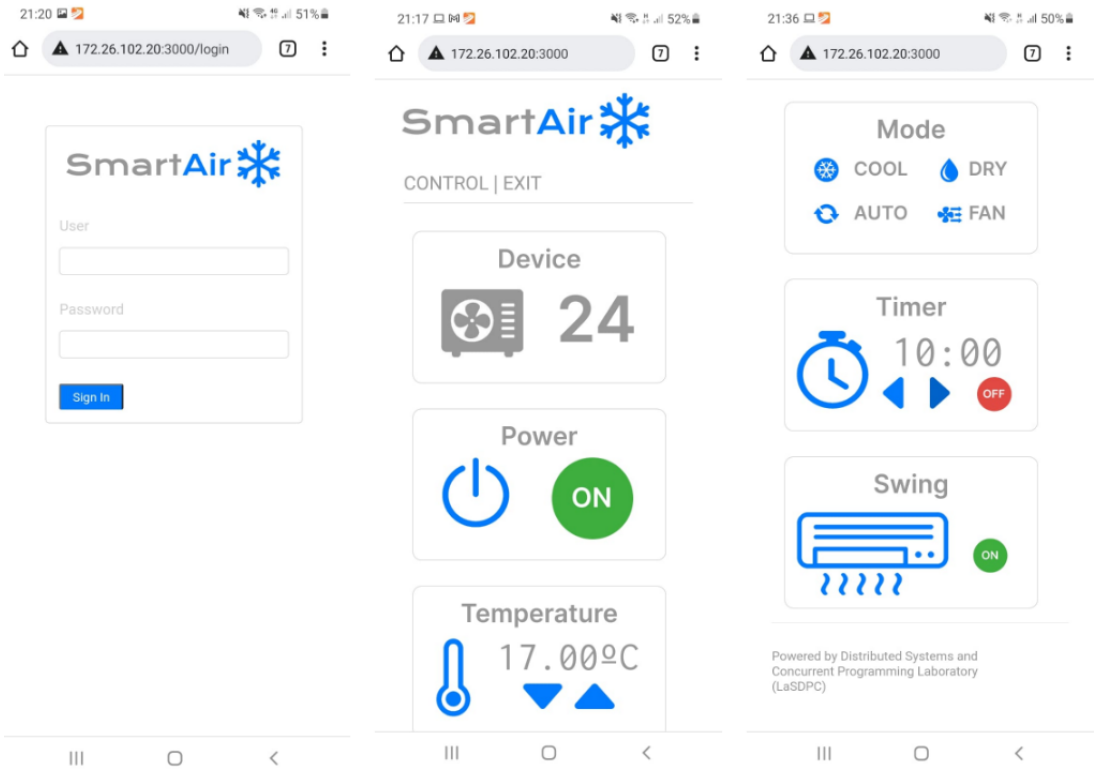
Figura 15 – Interface de controle do *Smart-Air Web* no monitor de um *desktop*.



Fonte: Elaborado pelo autor.

em um *smartphone* com resolução de tela 2400 x 1080, pode-se observar o auto ajuste dos Web componentes das funções de controle do AC.

Figura 16 – Interfaces de *login* e controle do *Smart-Air Web* na tela de um *smartphone*.

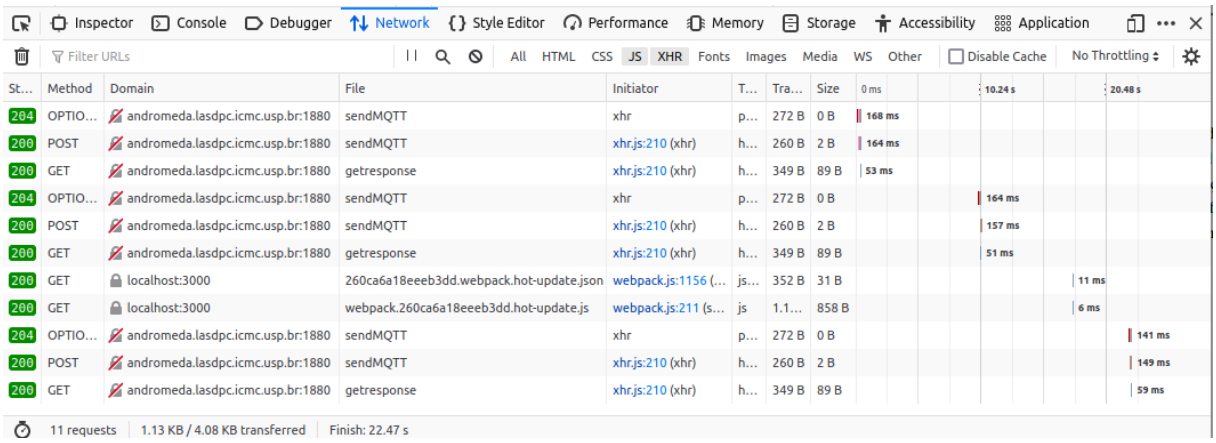


Fonte: Elaborado pelo autor.

Conforme observado em Figura 17, também foi realizado um teste do tráfego de comu-

nicação do protótipo com o sistema IoT de controle do AC. Nele foi obtido êxito no envio de comandos ao servidor na nuvem endereçado em http://andromeda.lasdpc.icmc.usp.br:1880/ via método POST do serviço HTTP. O recebimento da mensagem no circuito da ESP32 é observado por um retorno de confirmação, recebido via método GET na aplicação endereçada em http://localhost:3000/.

Figura 17 – *Snapshot* do trafego de rede no *Browser* do usuário.



Fonte: Elaborado pelo autor.

5.3 Depoimento de uso do usuário

O *Smart-Air Web* foi testado por um integrante do laboratório, aqui denominado IL1, através do acesso ao *Front-end*, endereçado na rede interna da USP, edoruam: http://172.26.102.20:3000/ e seu *feedback* sobre o protótipo foi o seguinte:

> Após o uso da interface desenvolvida, algumas características chamam a atenção, como a intuitividade da aplicação e também a rápida propagação do comando ao ar condicionado, o qual responde praticamente instantaneamente ao comando enviado. Vale acrescentar que a interface acrescenta muita comodidade e praticidade ao uso do laboratório, já que em muitas ocasiões o controle não se encontra um local de acesso, já a interface pode ser acessada em qualquer lugar e instantaneamente. (IL1)

Com essa avaliação positiva do usuário, pode-se ter a percepção da facilidade em utilizar a aplicação, bem como a sua relevância de uso no cotidiano do laboratório. Pesquisas de usabilidade deverão ser realizadas nas próximas etapas do projeto.

5.4 Considerações Finais

Neste capítulo, foi possível observar as interfaces do protótipo *Smart-Air Web* em funcionamento nos *browser* de um dispositivo *mobile* e um *notebook*. Também foi possível observar o

tráfego de comunicação da aplicação com o AC acionando funcionalidades remotamente. Por fim, observou-se o *feedback* positivo da experiência de uso do protótipo por um usuário no laboratório, onde o AC encontra-se instalado.

Capítulo 6

CONCLUSÃO

6.1 Contribuições

O trabalho buscou completar a necessidade do projeto *Smart-Air* (TRAZZI, 2021) ter uma aplicação própria para controle dos AC. O modelo da implementação do *Smart-Air Web*, permitiu que se obtenha e responsividade com as metas de sistemas distribuídos, principalmente pela premissas de transparência de distribuição ao usuário e escalabilidade do sistema, pela flexibilidade de se trabalhar com *Web components*. A aplicação demonstrou-se uma ferramenta eficiente, visual agradável e de acordo com as boas práticas de acessibilidade de interface. A arquitetura inicial, apresentada para funcionamento desta primeira versão do *Smart-Air Web*, tem compatibilidade para acoplamento de outros elementos e, também, a possibilidade de expansão de controle a outros dispositivos *smart*.

6.2 Trabalhos Futuros

Durante o desenvolvimento deste projeto, foram observadas importantes oportunidades para trabalhos futuros, apontando-se as seguintes:

- Hospedagem do protótipo em *container Docker* na nuvem para possibilitar um ambiente de desenvolvimento capaz de aplicar *releases* mais facilmente;
- Implementação de um *back-end* orientado a micro-serviços, hospedados em *containers Docker* na nuvem. A adição deste item na arquitetura do *Smart-Air*, possibilitam otimizar regras de negócios, bem como implementar recursos mais sofisticados à gestão do sistema, tal como *machine learning* coletando dados e gerando insights de consumo, tempo de utilização, relatórios importantes para gestão financeira das contas de energia e apoio à decisão de substituir um equipamento AC por um mais novo;
- Aplicar testes de *snapshots* da interface *React* através de bibliotecas, como *Jest*[9]. Também elaborar testes de carga com o *framework Selenium*[10] no *Front-end* e aplicar testes de volume com a ferramenta *JMeter*[11] no *back-end* a ser desenvolvido;

[9] Biblioteca Jest
[10] Selenium
[11] Apache JMeter

- Gerar uma galeria de *Web components* de funcionalidade para controle de AC compatíveis com outras marcas de equipamento, bem como, espelhar o *template* do *Smart-Air Web* para aplicações que controle outros dispositivos, como: televisores, ventiladores e lâmpadas.

REFERÊNCIAS

ABREU, L. Typescript: O javascript moderno para criação de aplicações. **São Paulo, SP: FCA**, 2017. Citado na página 26.

ADIONO, T.; HARIMURTI, S.; MANANGKALANGI, B. A.; ADIJARTO, W. Design of smart home mobile application with high security and automatic features. In: **2018 3rd International Conference on Intelligent Green Building and Smart Grid (IGBSG)**. [S.l.: s.n.], 2018. p. 1–4. Citado na página 29.

ALURA. **NextJS: por que usar?** 2021. Internet. Disponível em: <https://www.alura.com.br/artigos/next-js-vantagens>. Acesso em: 03 ago 2021. Citado na página 25.

BADER, A.; GHAZZAI, H.; KADRI, A.; ALOUINI, M.-S. Front-end intelligence for large-scale application-oriented internet-of-things. **IEEE Access**, v. 4, p. 3257–3272, 2016. Citado na página 29.

BERTOLETI, P. **Projetos com ESP32 e LoRa**. [S.l.]: Instituto NCB, 2019. Citado na página 22.

DRöGEHORN, O.; LESLIE, M.; PITTUMBUR, M.; PORRAS, J. Front-end development for home automation systems-a design approach using javascript frameworks. In: . [S.l.: s.n.], 2017. Citado 2 vezes nas páginas 19 e 30.

EBERHARDT, C. **You probably don't need a micro-frontend**. 2021. Internet. Disponível em: <https://blog.scottlogic.com/2021/02/17/probably-dont-need-microfrontends.html>. Acesso em: 23 oct 2021. Citado na página 27.

GEEKHUNTER. **Um guia para usar React JS**. 2019. Internet. Disponível em: <https://blog.geekhunter.com.br/um-guia-para-usar-react-js/#O_que_e_realmente_o_React_JS>. Acesso em: 22 jul 2021. Citado na página 25.

____. **Entendendo Next.js e aplicando suas funcionalidades**. 2021. Internet. Disponível em: <https://blog.geekhunter.com.br/o-que-e-next-js/#O_que_e_o_Nextjs>. Acesso em: 03 ago 2021. Citado 2 vezes nas páginas 25 e 26.

HEJAZI, H.; RAJAB, H.; CINKLER, T.; LENGYEL, L. Survey of platforms for massive iot. In: **2018 IEEE International Conference on Future IoT Technologies (Future IoT)**. [S.l.: s.n.], 2018. p. 1–8. Citado na página 30.

HOSTINGER. **O Que é React e Como Funciona?** 2021. Internet. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-react-javascript>. Acesso em: 22 jul 2021. Citado na página 25.

KODALI, R. An implementation of mqtt using cc3200. In: . [S.l.: s.n.], 2016. p. 582–587. Citado na página 24.

KUROSE, J. F.; ROSS, K. W. Redes de computadores e a internet. **São Paulo: Person**, v. 28, 2013. Citado 2 vezes nas páginas 23 e 24.

MOVILE. **Arquitetura micro frontend**. 2021. Internet. Disponível em: <https://movile.blog/arquitetura-micro-frontend/>. Acesso em: 22 oct 2021. Citado na página 27.

RAJPUT, S. **Top 15 Standard IoT Protocols in Germany Berlin, Munich, Frankfurt**. 2020. Internet. Disponível em: <https://www.omni-academy.com/top-standard-iot-protocols-germany/>. Acesso em: 17 nov 2021. Citado na página 22.

SÁTYRO, W. C.; SILVA, M. T. da; BONILLA, S. H.; GONÇALVES, R. F.; SACOMANO, J. B. **Indústria 4.0: Conceitos e fundamentos**. [S.l.]: Blucher, 2018. Citado na página 21.

SOURCE, F. O. **O Que é React e Como Funciona?** 2021. Internet. Disponível em: <https://reactjs.org/>. Acesso em: 23 jul 2021. Citado na página 25.

TRAZZI, B. M. **Smart-Air - Um mecanismo inteligente para o controle de sistemas de ar-condicionado**. 51 f. Monografia (Trabalho de Conclusão de Curso) — ICMC-USP, Universidade de São Paulo, São Carlos - SP, 2021. Citado 3 vezes nas páginas 22, 33 e 45.

VERCEL. **The React Framework for Production**. 2021. Internet. Disponível em: <https://nextjs.org/>. Acesso em: 03 ago 2021. Citado na página 26.

VERMA, A.; PRAKASH, S.; SRIVASTAVA, V.; KUMAR, A.; MUKHOPADHYAY, S. C. Sensing, controlling, and iot infrastructure in smart building: A review. **IEEE Sensors Journal**, v. 19, n. 20, p. 9036–9046, 2019. Citado na página 30.

WAZLAWICK, R. **Engenharia de software: conceitos e práticas**. [S.l.]: Elsevier Editora Ltda., 2019. Citado na página 35.

WITKOWSKI, K. Internet of things, big data, industry 4.0 – innovative solutions in logistics and supply chains management. **Procedia Engineering**, v. 182, p. 763–769, 2017. ISSN 1877-7058. 7th International Conference on Engineering, Project, and Production Management. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877705817313346>. Citado na página 21.

YUAN, M. **Conhecendo o MQTT**. 2017. Internet. Disponível em: <https://developer.ibm.com/br/articles/iot-mqtt-why-good-for-iot/>. Acesso em: 17 jun 2021. Citado na página 23.