UNIVERSIDADE FEDERAL DO TOCANTINS

PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL DE SISTEMAS – MESTRADO

Cícero Matheus da Silva Lacerda

Projeto de Uma Rede de Sensores Sem Fio para Monitoramento Ambiental em Cidades Inteligentes

PALMAS, 2021

Cícero Matheus da Silva Lacerda

Projeto de Uma Rede de Sensores Sem Fio para Monitoramento Ambiental em Cidades Inteligentes

Dissertação de mestrado apresentado à UFT – Universidade Federal do Tocantins – Campus Universitário de Palmas, Programa de Pós-Graduação em Modelagem Computacional de Sistemas, como requisito parcial para a obtenção do título de Mestre em Modelagem Computacional de Sistemas.

Orientador: Prof. Dr. Humberto Xavier de Araujo.

PALMAS, 2021

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da Universidade Federal do Tocantins

L131p Lacerda, Cícero Matheus da Silva.

Projeto de Uma Rede de Sensores Sem Fio para Monitoramento Ambiental em Cidades Inteligentes. / Cícero Matheus da Silva Lacerda. – Palmas, TO, 2021.

108 f.

Dissertação (Mestrado Acadêmico) - Universidade Federal do Tocantins – Câmpus Universitário de Palmas - Curso de Pós-Graduação (Mestrado) em Modelagem Computacional de Sistemas, 2021.

Orientador: Humberto Xavier de Araujo

Coorientador: Gentil Veloso Barbosa

1. Redes de Sensores Sem Fio. 2. Internet das Coisas. 3. Cidades Inteligentes. 4. Arquitetura de Software. I. Título

CDD 004

Elaborado pelo sistema de geração automática de ficha catalográfica da UFT com os dados fornecidos pelo(a) autor(a).

Cícero Matheus da Silva Lacerda

Projeto de uma Rede de Sensores Sem Fio para Monitoramento Ambiental em Cidades Inteligentes

Dissertação de mestrado apresentado à UFT – Universidade Federal do Tocantins – Campus Universitário de Palmas, Programa de Pós-Graduação em Modelagem Computacional de Sistemas, como requisito parcial para a obtenção do título de Mestre em Modelagem Computacional de Sistemas.

Trabalho aprovado em ____ de _____ de 2021:

Prof. Dr. Humberto Xavier de Araújo
Orientador

Prof. Dr. David Nadler Prata
Convidado 1

Prof. Dr. Adelício Maximiano Sobrinho
Convidado 2

Brasil
Palmas, 2020

A minha família por todo apoio e incentivo durante a jornada. A todos que contribuíram direta ou indiretamente para construção do meu conhecimento.

AGRADECIMENTOS

Aos meus pais José Aurivan Lacerda da Silva e Raimunda Nonata da Silva Lacerda pelos ensinamentos sobre como viver a vida, pelo investimento na minha educação e pelo companheirismo ao longo da minha jornada. Cito também aqui meu irmão José Aurivan Lacerda da Silva Filho pelos momentos de parceria e descontração.

Aos meus amigos, tanto dentro quanto fora do ambiente acadêmico, pelos inúmeros conselhos, vivências engraçadas e obstáculos vencidos através de um formidável trabalho em equipe. Não mencionarei nomes aqui para evitar que a preocupação com a brevidade desses agradecimentos me faça esquecer algum. Porém, aos que lerem, saberão que me refiro a vocês.

Ao meu orientador professor Dr. Humberto Xavier de Araujo por acreditar no meu potencial desde o meu segundo semestre na graduação, pelos conselhos de vida e investimento em minha formação acadêmica.

Aos demais profissionais envolvidos direta e indiretamente em minha formação acadêmica e aprendizagem, afinal, a humanidade só consegue evoluir porque o conhecimento é passado adiante ao longo das gerações.

Para todos vocês, meus sinceros agradecimentos.

RESUMO

O presente trabalho apresenta o projeto de uma rede de sensores sem fio para monitoramento ambiental em cidades inteligentes. Para tal, inicialmente, são conceituadas tanto as premissas que definem uma cidade inteligente quanto as ferramentas tecnológicas que possibilitam a comunicação remota entre máquinas. Adiante, são apresentados o microcontrolador usado para realizar o monitoramento, que é o ESP32, e a infraestrutura de TI necessária para manter o sistema funcionando. Essa infraestutura, por sua vez, conta com ferramentas como StrongSwan para VPN, Bacula para *backup*, IPTables como *firewall*, Docker para publicação e GitLab como gerenciador de repositórios e executor das *pipelines CI/CD*. Logo em seguida, através da conceituação da arquitetura limpa, é explicado como é possível construir um sistema flexível, mesmo em nível de hardware. Como resultados, o trabalho mostra as aplicações e APIs feitas em .NET 5 para acessar os dados de monitoramento armazenados em um SQL Server 2019. Posteriormente são detalhados os testes, os quais atenderam às expectativas, e, além disso, são discutidos tópicos sobre eficiência, escalabilidade, confiabilidade, resiliência e segurança do protótipo apresentado. Para finalizar, há ainda uma seção dedicada à sugestão de trabalhos futuros.

Palavras-chave: Redes de Sensores Sem Fio, Monitoramento Ambiental, Cidades Inteligentes, Internet das Coisas.

ABSTRACT

This work presents the implementation of a wireless sensor network for environmental monitoring in smart cities. To do so, initially, the premises that define a smart city and the technological tools that enable remote communication between machines are conceptualized. Besides that, the microcontroller used to perform the monitoring, the ESP32, and the IT infrastructure needed to keep the system running are presented. This infrastructure, in turn, has tools like StrongSwan for VPN, Bacula for *backup*, IPTables as *firewall*, Docker for deploying and GitLab as repository manager and runner for the *pipelines CI/CD*. Then, through the definition of clean architecture, it is explained how it is possible to build a flexible system, even at the hardware level. As result, the work shows the applications and APIs made in .NET 5 to access the monitoring data stored in a SQL Server 2019. Subsequently, the tests are detailed, which met the expectations, and, in addition, topics about efficiency, scalability, reliability, resilience and security of the presented prototype are discussed. Finally, there is also a section dedicated to suggesting future work.

Keywords: Wireless Sensor Network, Environment Monitoring, Smart Cities, Internet of Things.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Conforme abordado por (FOUNOUN; HAYAR, 2018), o incremento na qualidade de vida possibilita o aumento demográfico da população humana e, por consequência, exige o uso mais eficiente dos recursos escassos da natureza. É importante mencionar também o que foi argumentado por (MORA et al., 2018), onde é afirmado que o crescimento da internet possibilita a criação de soluções robustas e tecnológicas para os problemas contemporâneos.

Ainda em conformidade com (MORA et al., 2018) e acrescentando as conclusões de (FOUNOUN; HAYAR, 2018), é nesse contexto que ocorre a definição de cidades inteligentes: cidades que implementam serviços que funcionam a partir da internet e objetivam a maximização da satisfação da população. Naturalmente, conforme explicado por (ANTHOPOULOS, 2017), é necessário uma gama de protocolos e equipamentos para tornar isso possível.

Dentre as possíveis aplicações das cidades inteligentes propostos por (ANTHOPOULOS, 2017), aqui é focado no contexto ambiental, onde é possível ter implementações tanto no âmbito do monitoramento quanto redução de poluição (HAYAR; BETIS, 2017). Exposto isso, nesse trabalho é tratada a parte de monitoramento, em particular, as grandezas medidas são: temperatura, umidade, radiação ultravioleta, presença de incêndios e concentração de gases tóxicos e fumaça.

Restringindo geograficamente para o Brasil, há um planejamento para implantar cidades inteligentes através do programa Cidades Digitais, instituído em 2011. (DIÁRIO OFICIAL DA UNIÃO, 2011) (DIÁRIO OFICIAL DA UNIÃO, 2012). Como caso de estudo, o presente trabalho usará a rede da cidade de Porto Nacional, Tocantins, a qual é constituída por um anel de fibra óptica que conecta os principais pontos da cidade (PREFEITURA DE PORTO NACIONAL, 2017).

A infraestrutura presente na cidade mostra que o alicerce das cidades inteligentes é a internet, conforme já discutido. Portanto, os sistemas inteligentes devem implementar os protocolos já consolidados pela internet, como o TCP/IP e HTTP (COMER, 2016). Dito isto, fica aqui exposta a primeira parte do que é abordado nesse trabalho: a construção e configuração de uma infraestrutura de TI que possibilite a troca de informações através da internet.

Dentro dessa infraestrutura são empregados conceitos como arquitetura distribuída de sistemas, contêineres Docker, VPN StrongSwan, *firewall* IPTables, APIs REST com .NET 5, Bases de Dados SQL Server 2019, servidor HTTP com Nginx e de Backup com Bacula. É discutido também processos de automação de desenvolvimento e entrega de aplicações, como a ferramenta GitLab para controle de versão e *pipelines CI/CD*. Todas essas abordagens são usadas com o intuito de aprimorar o processo de desenvolvimento de sistemas.

Todavia, é importante também frisar a outra ponta, ou seja, os circuitos que vão gerar os dados que circularão pela cidade e usarão a infraestrutura construída. É nesse sentido que entram

os sistemas embarcados, apresentados por (BINDAL, 2017). O primeiro ponto nesse âmbito é o uso de sensores para aferir grandezas no mundo real e convertê-las para uma representação em forma de tensão elétrica (BOYLESTAD; NASHELSKY, 2013). A informação convertida em tensão, por sua vez, é lida por um microcontrolador (BINDAL, 2017).

Sendo o microcontrolador um computador integrado, de menor poder e mais barato (TOCCI; WIDNER; MOSS, 2011), ele é incumbido de receber os dados dos sensores e enviar para a rede usando os protocolos da internet. Dito com outras palavras, os sensores e circuitos não possuem conhecimento dos protocolos, de modo que eles só são levados em conta pelo microcontrolador na hora de enviar os dados para dentro da rede. A partir disso, pontua-se a segunda abordagem desse trabalho: construção dos circuitos de monitoramento e programação do microcontrolador.

Cada microcontrolador, em conjunto com seus sensores, forma uma célula (nome escolhido pelo autor desse trabalho) que compõe uma rede de células, onde cada uma monitora uma região geográfica. Essa associação é chamada de rede de sensores sem fio, de modo que as informações de cada microcontrolador são transmitidas pela internet para um servidor (YAVARI et al., 2019) (NETTO, 2016).

Para tal finalidade, o microcontrolador empregado no presente trabalho é o ESP32, o qual possui suporte a conexão WiFi de maneira integrada e é programado em C++ 14. Para realizar o monitoramento, foram escolhidos os sensores DHT11, para temperatura e umidade; Guva S12SD para radiação ultravioleta; MQ-135 para gases tóxicos e fumaça; e, por fim, um sensor de chama baseado em fototransistor.

Por fim, todo sistema de monitoramento precisa sumarizar e extrair informações do conjunto de dados (AKYILDIZ et al., 2002) (NETTO, 2016) (YAVARI et al., 2019). Para tal, é preciso apresentar a terceira e última abordagem desse trabalho: construção de uma aplicação web para que a população possa ver os dados do monitoramento e a implementação de um sistema para que os administradores possam ter uma visão técnica e ampla do que for coletado.

Em resumo, portanto, o presente trabalho parte de uma infraestrutura de internet na cidade de Porto Nacional, Tocantins, e aborda a implementação, configuração e programação das células de monitoramento; infraestrutura de TI, servidores e APIs para tratamento dos dados e painéis de visualização tanto para os cidadãos quanto para os administradores do sistema.

1.1 Justificativa

Sendo as cidades inteligentes uma tendência em diversos países, justifica-se esse trabalho a partir da necessidade crescente de aplicar as tecnologias computacionais no ambiente urbano. Adentrando ainda mais nas especificidades aqui pontuadas, a abordagem em três áreas (circuitos, infraestrutura de TI e interação com o usuário) permite uma visão ampla do processo

de implantação das cidades inteligentes.

1.2 Objetivos

1.2.1 Objetivo Geral

Implementar uma rede de sensores sem fio para fazer o monitoramento ambiental da cidade de Porto Nacional, Tocantins, incluindo servidores, aplicações, *hardware*, *firewalls*, VPNs e demais equipamentos de rede que viabilizam a comunicação via internet e disponibilizando interfaces de interação com o usuário para que todos possam usufruir do sistema.

1.2.2 Objetivos Específicos

- Implementar um sistema embarcado para realizar o monitoramento ambiental e enviar as informações pela internet;
- Configurar a infraestrutura de TI necessária para viabilizar o transporte de informações sobre o monitoramento;
- Implementar as APIs que possibilitam a comunicação do circuito de monitoramento com os servidores;
- Desenvolver uma aplicação web de administração para que os servidores públicos da prefeitura possam gerenciar a rede de sensores sem fio;
- Publicar uma aplicação web para que os cidadãos (também chamados de usuários comuns) possam visualizar as medidas coletadas nos monitoramentos;
- Disponibilizar uma API pública de leitura para que os cidadãos possam usar os dados para seus próprios objetivos.

1.3 Estrutura do Trabalho

O presente trabalho possui 9 capítulos, de modo que a estrutura é apresentada nos próximos parágrafos.

O **Capítulo 2** aborda a conceituação de sistemas embarcados, explanando acerca de conceitos de circuitos elétricos e arquitetura de computadores. No final do capítulo são abordados ainda o microcontrolador usado nesse trabalho, o ESP32 DevKit V1, e os detalhes de funcionamento de cada sensor.

No **Capítulo 3** são apresentados as definições dos protocolos usados nas redes de computadores. Vale dizer ainda que há aprofundamento acerca dos protocolos que são diretamente empregados aqui, como o HTTP. Além disso, são discutidas também as questões de segurança e integridade dos dados. Dessa forma, é abordado também técnicas que visam aumentar a segurança do sistema, tanto no quesito de controle de acesso quanto no contexto da integridade dos dados.

Dentro do **Capítulo 4** estão contidos o arcabouço teórico das cidades inteligentes e a implementação desse conceito no Brasil a partir do projeto Cidades Digitais. Por fim, há também uma discussão sobre internet das coisas e redes de sensores sem fio.

O **Capítulo 5** possui informações sobre a arquitetura de software distribuído e as *pipelines CI/CD*. Há também uma seção dedicada À conceituação da arquitetura limpa. A partir dos conceitos apresentados é exposto como um sistema baseado em serviços se encaixa melhor em cidades inteligentes.

Na sequência, o **Capítulo 6** aborda o funcionamento e a configuração dos diversos tipos de servidores usados no trabalho. Aqui também são expostas as configurações da *pipeline*, *backup* e do servidor *web*.

Depois, no **Capítulo 7**, são expostas as informações sobre as formas de interação com os usuários. É nessa parte, portanto, que é demonstrado o funciona da aplicação de administração para os funcionários da prefeitura e, em adição, é descrita também a aplicação para usuários comuns e a API pública de leitura.

No **Capítulo 8** são comentados os resultados e, por fim, o **Capítulo 9** apresenta as devidas conclusões.

2 MICROCONTROLADORES E SISTEMAS EMBARCADOS

2.1 Grandezas Elétricas

O estudo de circuitos elétricos começa com a definição de carga elétrica como sendo uma propriedade das partículas que compõem a matéria (HAYT; BUCK, 2013). Ainda conforme o autor, sua unidade de medida é o Coulomb (C) e, no contexto desse trabalho, basta saber que um único elétron possui carga igual a $-1,602 \cdot 10^{-19}\ C$.

Decorrente do conceito de carga elétrica, define-se a corrente elétrica como sendo o fluxo de carga por unidade de tempo. Sua unidade de medida é o Ampère (A) e matematicamente ela é dada por (1) (ALEXANDER; SADIKU, 2013):

$$i = \frac{dq}{dt} \tag{1}$$

Onde 1 Ampère corresponde a 1 Coulomb por segundo.

Seguindo a elucidação das grandezas elétricas, é definida também a tensão entre os pontos a e b. Nesse caso, a tensão é a quantidade de energia (ou trabalho) necessária para deslocar uma carga unitária entre os dois pontos. Sua unidade de medida é o Volt (V) e é definida conforme (2) (ALEXANDER; SADIKU, 2013):

$$v_{ab} = v = \frac{dw}{dq} \tag{2}$$

Onde 1 Volt é igual a 1 Joule por Coulomb. É importante ressaltar que as baterias que alimentam os circuitos são chamadas de fontes de tensão, ou seja, elas fornecem uma tensão previamente informada para o circuito. Dessa forma, onde há tensão pode haver trabalho, o que leva a movimentação de cargas elétricas e a consequente manifestação de uma corrente elétrica (ALEXANDER; SADIKU, 2013).

Para finalizar as grandezas principais, (BOYLESTAD, 2007) define potência e energia. Como a primeira é a mais corriqueira, a conceituação começará por ela, onde potência é o consumo de energia por unidade de tempo. Sua unidade de medida é o Watt (W) e é calculada de acordo com (3):

$$p = v \cdot i = \frac{dw}{dt} \tag{3}$$

Onde 1 W equivale a 1 Joule por segundo. Como (3) pode ser calculada através da tensão e da corrente, é possível extrair a fórmula da energia da própria definição de potência:

$$w = w_0 + \int_{t_0}^{t} v \cdot i \; dt \tag{4}$$

Sendo a unidade de medida o Joule (J). A título de curiosidade, as concessionárias medem a energia consumida com Watt-hora, onde $1Wh = 3600J$ (ALEXANDER; SADIKU, 2013).

2.2 Componentes Passivos

Os componentes passivos são aqueles que não injetam potência no circuito (ALEXANDER; SADIKU, 2013). Cada um desses componentes define consigo uma grandeza e possui sua própria relação entre tensão e corrente (BOYLESTAD, 2007). A Figura 1 mostra esses componentes e suas grandezas.

Figura 1 – Componentes Passivos

O primeiro deles na Figura 1 é o resistor, o qual traz a resistência R, definida em Ohms (Ω). A relação entre tensão e corrente nesse componente é dada por (5) (ALEXANDER; SADIKU, 2013):

$$v(t) = R \cdot i(t) \tag{5}$$

Vale dizer ainda que o resistor dissipa energia na forma de calor.

O elemento do meio é o capacitor, responsável por armazenar energia na forma de campo elétrico (BOYLESTAD, 2007). A grandeza atrelada a ele é a capacitância C, medida em Farads (F). A relação tensão-corrente é obtida através da seguinte equação diferencial (6) (BINDAL, 2017):

$$i(t) = C\frac{dv(t)}{dt} \tag{6}$$

Por último, há o indutor, responsável por armazenar energia em um campo magnético (ALEXANDER; SADIKU, 2013). Ele apresenta a indutância L, medida em Henrys (H). Sua fórmula também envolve uma equação diferencial e é dada por (7):

$$v(t) = L\frac{di(t)}{dt} \tag{7}$$

Em análises mais complexas, é comum agrupar as três grandezas aqui apresentadas em apenas uma chamada de impedância, representada pela letra Z (ALEXANDER; SADIKU, 2013). Dependendo do componente, a impedância é definida a partir de (8), (9) ou (10) (BINDAL, 2017):

$$Z_R = R \tag{8}$$

$$Z_C = \frac{1}{sC} \tag{9}$$

$$Z_L = sL \tag{10}$$

A equação (8) é a impedância resistiva, a (9) representa o capacitor e, por último, a (10) é para o indutor. Observe ainda que elas são definidas a partir de uma análise transitória com a Transformada de Laplace (BINDAL, 2017). Para o caso do regime estacionário, é possível usar a Transformada de Fourier fazendo $s = j\omega$, onde $j = \sqrt{-1}$ (ALEXANDER; SADIKU, 2013) Os autores ainda afirmam que a relação tensão-corrente possui a mesma fórmula para os três tipos de impedância, conforme (11):

$$V = Z \cdot I \tag{11}$$

Por questões de convenção, as letras das grandezas elétricas são usadas em maiúsculo quando o tratamento é feito do ponto de vista das impedâncias (BOYLESTAD, 2007).

2.2.1 Divisor de Tensão com Carga de Alta Impedância

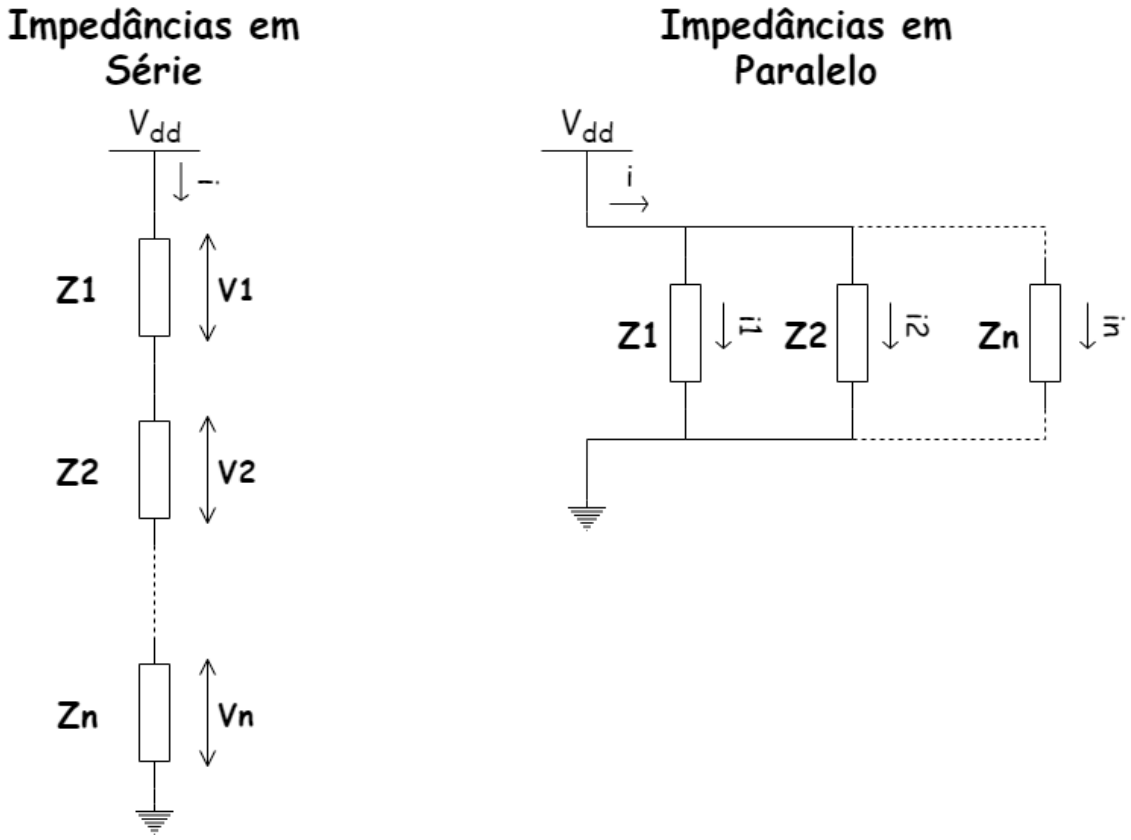
Um divisor de tensão é um circuito que pode ser formado, a priori, por qualquer um dos três componentes passivos (ALEXANDER; SADIKU, 2013). O processo é iniciado através do entendimento das possíveis associações de impedâncias, demonstradas na Figura 2.

À esquerda na Figura 2 é o esquemático da associação em série, onde as impedâncias compartilham a mesma corrente e dividem a tensão. A impedância equivalente Z_{eq} é dada por (12). Já a tensão na k-ésima impedância é calculada a partir de (13) (ALEXANDER; SADIKU, 2013).

$$Z_{eq} = \sum_{k=0}^{n} Z_k \tag{12}$$

$$V_k = V_{dd}\frac{Z_k}{Z_{eq}} \tag{13}$$

Figura 2 – Associação de Impedâncias



Fonte: Adaptado de Alexander; Sadiku, 2013.

À direita na Figura 2 é apresentada a associação em paralelo, onde agora a grandeza em comum é a tensão nas impedâncias. As equações (14) e (15) mostram, respectivamente, a impedância equivalente e a corrente no ramo k (BOYLESTAD, 2007).

$$\frac{1}{Z_{eq}} = \sum_{k=0}^{n} \frac{1}{Z_k} \tag{14}$$

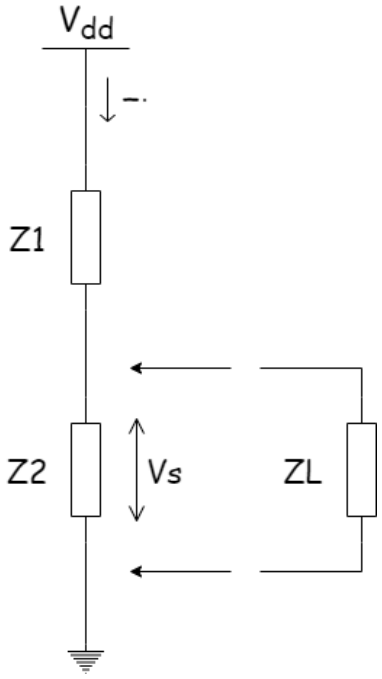
$$i_k = i\frac{Z_{eq}}{Z_k} \tag{15}$$

Apresentadas as definições acima, a Figura 3 mostra um divisor de tensão. Com base em (13), a tensão de saída V_S é dada por (16) quando o circuito está sem a carga Z_L (BOYLESTAD; NASHELSKY, 2013).

$$V_S = V_{dd}\frac{Z2}{Z1 + Z2} \tag{16}$$

Todavia, ao ser anexada a carga Z_L, há uma associação em paralelo entre Z_2 e Z_L, de modo a modificar a tensão de saída uma vez que a impedância equivalente é modificada. A impedância equivalente $Z_2^{'}$ no final do circuito da Figura 3 é dada por (17):

Figura 3 – Divisor de Tensão



Fonte: Adaptado de Boylestad; Nashelsky, 2013.

$$\frac{1}{Z_2'} = \frac{1}{Z_2} + \frac{1}{Z_L} \tag{17}$$

Isso é chamado de efeito de carga (BOYLESTAD, 2007). Entretanto, é possível contornar essa desvantagem a partir da situação limítrofe onde a carga é muito maior que Z_2. É possível ver a descrição matemática dessa situação através de (18):

$$\lim_{Z_L \to \infty} \left\{ \frac{1}{Z_2} + \frac{1}{Z_L} \right\} = \frac{1}{Z_2} \tag{18}$$

Portanto, quando Z_L é várias ordens de magnitude maior que Z_2, o efeito de carga pode ser negligenciado (BOYLESTAD; NASHELSKY, 2013). Conforme será visto adiante, o presente trabalho emprega o divisor de tensão nesse contexto.

Outra consideração importante sobre os divisores de tensão é no tocante à potência. A partir das equações (3) e (11), tem-se a relação para potência dada por (19):

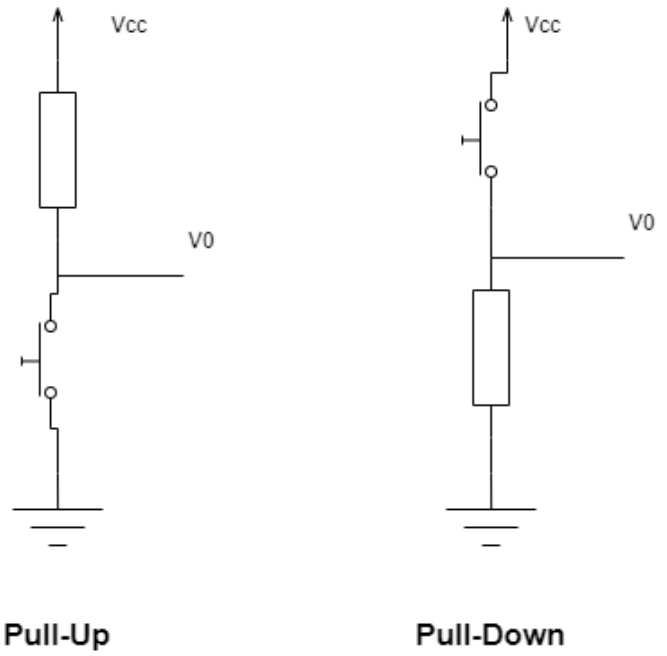
$$P = V \cdot I = \frac{V^2}{Z} \tag{19}$$

Portanto, quanto maior a magnitude das impedâncias menor será o consumo de potência do circuito (BOYLESTAD; NASHELSKY, 2013).

2.2.2 Resistores de Pull-Up e Pull-Down

É abordado por (BOYLESTAD; NASHELSKY, 2013) a necessidade de resistores de *pull-up* e *pull-down*. Para entender melhor, observe a Figura 4.

Figura 4 – Resistores de Pull-Up e Pull-Down



Fonte: Adaptado de Boylestad; Nashelsky, 2013.

Ao usar comutação em um circuito digital, é necessário que o nível lógico esteja bem definido, pois, caso o comutador fique aberto, a leitura V_0 será somente a do ruído, cuja interpretação é imprevisível (BOYLESTAD; NASHELSKY, 2013).

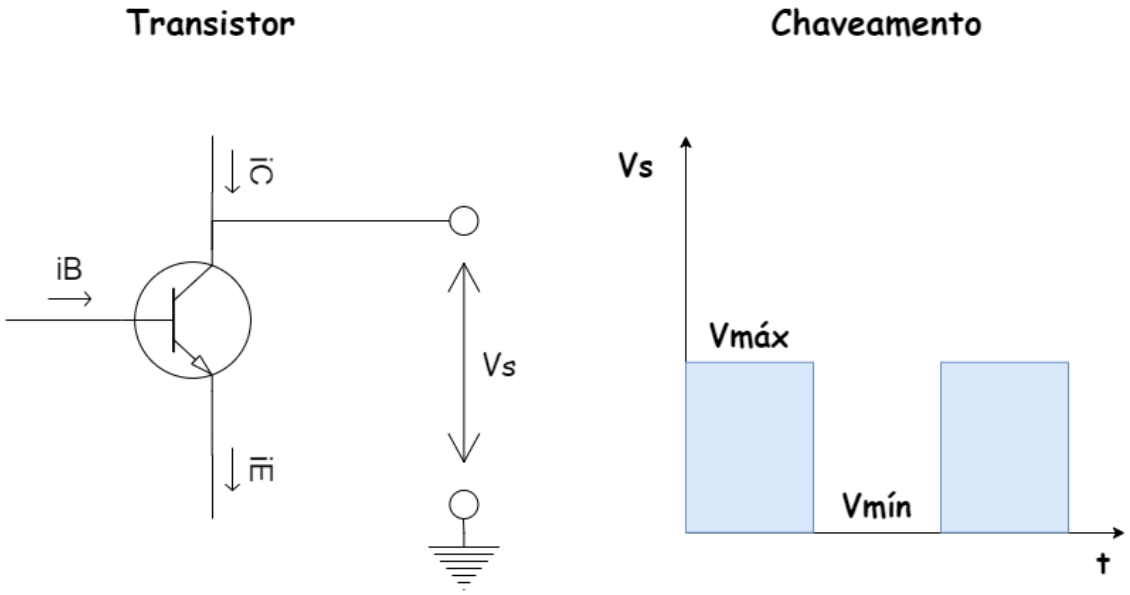
Dito isso, é aí onde os resistores devem atuar. Na configuração de *pull-up*, por exemplo, V_0 é, por padrão, igual a V_{cc}. Porém, ao pressionar o botão, ou seja, ao realizar a comutação, V_0 assume valor igual a GND. Portanto, ambos os estados lógicos estão bem definidos na comutação. O raciocínio inverso pode ser aplicado ao resistor de *pull-down*.

2.3 Semicondutores e o Sistema Binário

Conforme (SHAKELFORD, 2008), os semicondutores são materiais que apresentam condutividade intermediária entre condutores e isolantes. O autor ainda salienta que dentro da engenharia elétrica os semicondutores extrínsecos são aqueles mais utilizados. Esses materiais são obtidos através do processo de dopagem, ou seja, inserção de outros materiais na estrutura cristalina do semicondutor puro (BOYLESTAD; NASHELSKY, 2013).

O presente trabalho faz menção somente ao transistor como chave por esse componente ser a ponte entre os componentes elétricos e o sistema digital usado na computação. A Figura 5 mostra um transistor (não serão comentadas as tecnologias de transistor) e seus três terminais. A corrente na base i_B permite controlar a corrente no coletor i_C. Por fim, a corrente no emissor i_E é a soma das duas (BINDAL, 2017).

Figura 5 – Transistor como Chave



Fonte: Adaptado de Boylestad; Nashelsky, 2013.

Ao lado, na Figura 5, é visto a tensão de saída em um transistor como chave. Nesse contexto, a tensão ou é tensão máxima ou é tensão mínima. É dessa forma que são gerados os *bits* nos circuitos digitais (BOYLESTAD; NASHELSKY, 2013). Vale dizer ainda que os circuitos que permitem o funcionamento do transistor são chamados de circuitos de polarização e eles são constituídos pelos elementos passivos apresentados anteriormente (BOYLESTAD; NASHELSKY, 2013).

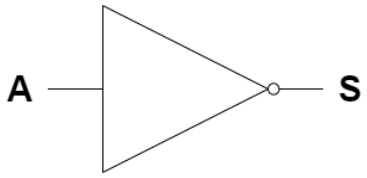
Trazendo para o sistema binário, as informações acerca dos níveis de tensão perdem força e dão lugar às representações qualitativas de $0s$ e $1s$ (TOCCI; WIDNER; MOSS, 2011). Com esse sistema, resultados lógicos podem ser representadas como verdadeiro (nível lógico 1) ou falso (nível lógico 0). Além disso, é possível escrever qualquer número decimal em sua forma binária (TOCCI; WIDNER; MOSS, 2011).

2.4 Sistemas Digitais

2.4.1 Portas Lógicas

Portas lógicas são circuitos usados para realizar operações lógicas (TOCCI; WIDNER; MOSS, 2011). A mais simples delas é a porta *NOT*, cujo símbolo é apresentado na Figura 6 e tabela-verdade na Tabela 1.

Figura 6 – Símbolo da Porta NOT



Fonte: Disponível em:
http://www.bosontreinamentos.com.br/eletronica/eletronica-digital/porta-logica-not-inversora/

Tabela 1 – Tabela-Verdade da Porta NOT

A	S
0	1
1	0

Fonte: Adaptado de Tocci; Widner; Moss, 2011.

Como é possível ver, a porta *NOT* inverte a entrada. Se ela for verdadeira, a saída será falsa. O oposto também se aplica (TOCCI; WIDNER; MOSS, 2011).

As próximas portas são exibidas na Figura 7. Já suas tabelas-verdade estão contidas na Tabela 2. Essas portas são os ingredientes básicos dos demais circuitos que compõem os sistemas computacionais e serão detalhados nas próximas seções (BINDAL, 2017).

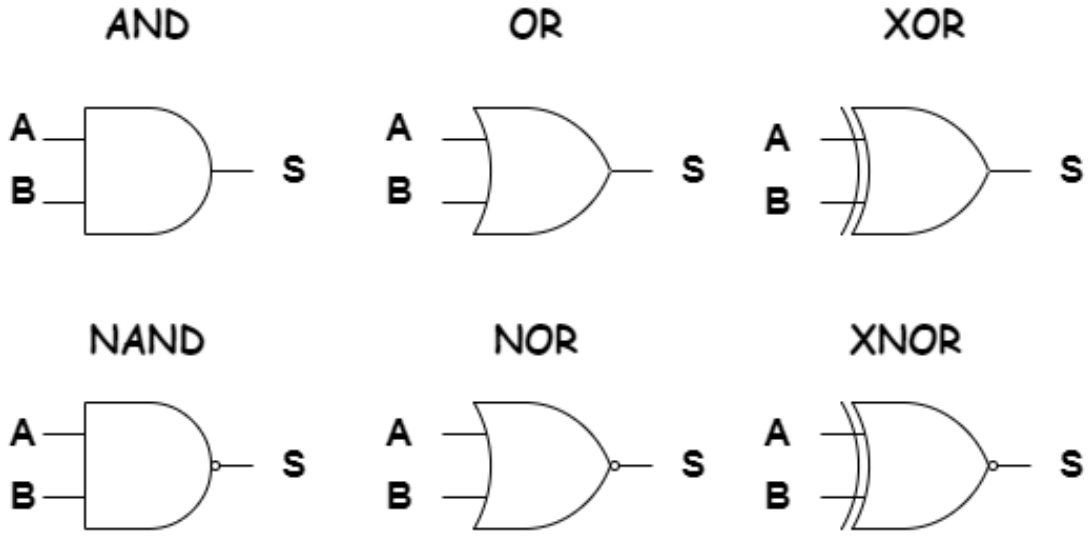
Tabela 2 – Tabelas-Verdade das Demais Portas Lógicaas

A	B	AND	OR	XOR	NAND	NOR	XNOR
0	0	0	0	0	1	1	1
0	1	0	1	1	1	0	0
1	0	0	1	1	1	0	0
1	1	1	1	0	0	0	1

Fonte: Adaptado de Tocci; Widner; Moss, 2011.

Como é possível observar na Tabela 2, a porta *AND* só possui saída verdadeira se todas as entradas forem verdadeiras. Para a porta *OR*, porém, basta apenas uma entrada em nível 1. A *XOR*, por sua vez, só é ativada se houver um número ímpar de $1s$ nas suas entradas (TOCCI; WIDNER; MOSS, 2011). As outras três podem ser entendidas como se fossem portas *NOT* ligadas à saída de uma *AND*,*OR* e *XOR*, respectivamente (BINDAL, 2017).

Figura 7 – Demais Portas Lógicas



Fonte: Adaptado de Tocci; Widner; Moss, 2011.

2.4.2 CPU

A CPU (*Central Processing Unit*, Unidade de Processamento Central) é responsável por realizar as operações lógicas e matemáticas do computador (FERNANDEZ, 2015). Na Figura 8 é possível observar como uma CPU funciona em linhas gerais. O primeiro passo é a instrução binária, oriunda de um programa, por exemplo. Ao passar por um decodificador, essa instrução é divida em duas partes: a instrução da operação a ser realizada e os parâmetros que vão para os registradores.

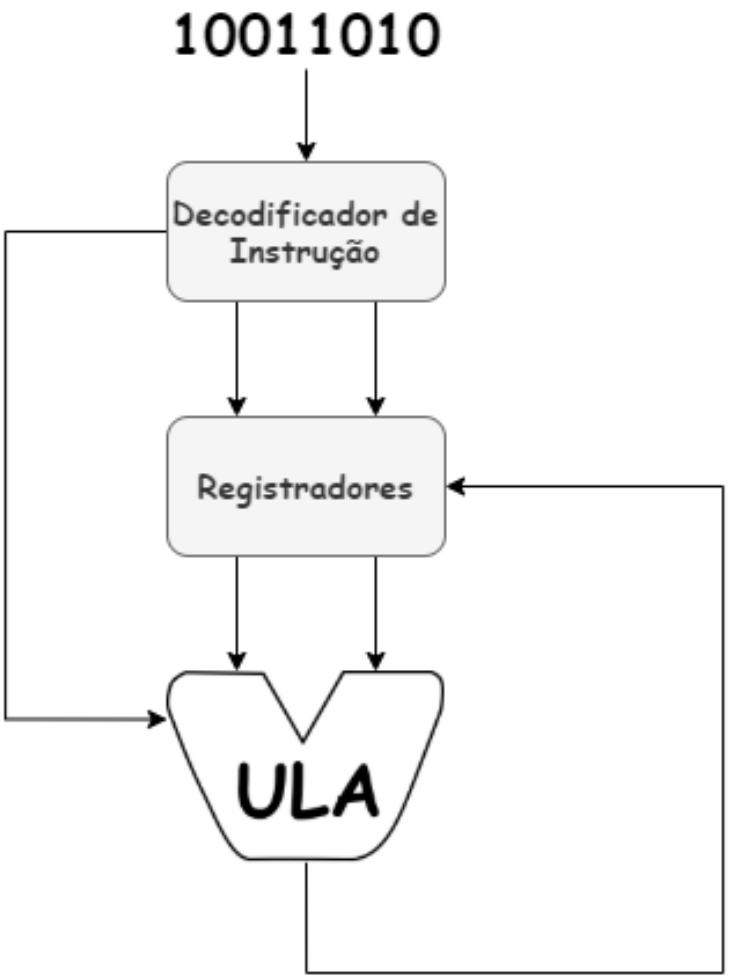
No tocante aos registradores, vale frisar que eles podem ser entendidos como memórias temporárias que irão reter os valores enquanto os demais circuitos se preparam para recebê-los. O componente que de fato realizará a operação é a ULA (Unidade Lógica-Aritmética), a qual recebe o código da operação a partir do decodificador e os parâmetros dos registradores. Por fim, o resultado da operação pode servir como entrada para a próxima, logo, a ULA também pode inserir informações nos registradores (FERNANDEZ, 2015).

Alguns detalhes ainda precisam ser adicionados: as instruções da CPU vêm de uma memória dedicada a armazenar os passos do programa que o computador executa (FERNANDEZ, 2015). Além disso, os detalhes dos protocolos de comunicação não serão abordados.

2.4.3 Memórias

As memórias são dispositivos usados para reter informações (FERNANDEZ, 2015). Antes de detalhar os tipos de memória, é preciso ficar atento ao significado de alguns termos

Figura 8 – Funcionamento de uma CPU



Fonte: Adaptado de Marcial, 2017.

(TOCCI; WIDNER; MOSS, 2011):

- Palavra: são os bits a serem armazenados na memória. É a informação propriamente dita;
- Endereços: cada palavra é armazenada em um lugar na memória. Esse lugar é chamado de endereço e cada um possui uma representação binária única;
- Capacidade: é a quantidade total de bits que pode ser armazenada na memória. Pode ser obtida multiplicando a quantidade de endereços pelo tamanho da palavra.

Vale dizer que a capacidade anunciada no mercado é na verdade a quantidade de endereços, ou seja, uma memória com "capacidade"de $4\ GB$ é na verdade uma memória com $4.294.967.296$ endereços (FERNANDEZ, 2015).

De acordo com (TOCCI; WIDNER; MOSS, 2011), as memórias podem ser voláteis ou permanentes. As voláteis tendem a ser mais rápidas, mais caras, menores e perdem os dados

após o desligamento (FERNANDEZ, 2015). As permanentes, em contrapartida, conseguem reter dados e são mais baratas, porém, tendem a ser mais lentas e maiores (TOCCI; WIDNER; MOSS, 2011). Dessa forma, as memórias que se comunicam com maior frequência com a CPU são voláteis (registradores, por exemplo), pois são mais rápidas. Consequentemente, as permanentes ficam responsáveis somente pela persistência dos dados.

Entre as memórias voláteis, além dos registradores, estão as RAMs (*Random Access Memory*, Memória de Acesso Aleatório). Essas são usadas para carregar os binários dos programas da memória permanente e enviar as instruções ao processador (FERNANDEZ, 2015). Já no campo da permanentes, se destacam a EEPROM, Flash e os discos rígidos. Para o escopo desse trabalho, é suficiente falar sobre a Flash, sendo uma memória barata e rápida, porém, permite apagar os dados somente por setores ou endereços, ou seja, não é possível apagar bit a bit.

2.4.4 Conversores Analógico-Digital

Os conversores analógico-digital são circuitos que permitem que os sistemas digitais lidem com valores oriundos do mundo analógico. Para esse trabalho, é suficiente entender apenas três conceitos.

O primeiro deles é a tensão de referência V_{ref}, a qual é definida como sendo o valor máximo de tensão que pode ser convertida para digital (BINDAL, 2017). O segundo é a quantidade de *bits* N na saída desse conversor. Com essas duas definições, é possível conceituar a resolução conforme (20) (TOCCI; WIDNER; MOSS, 2011):

$$\delta = \frac{V_{ref}}{2^N - 1} \tag{20}$$

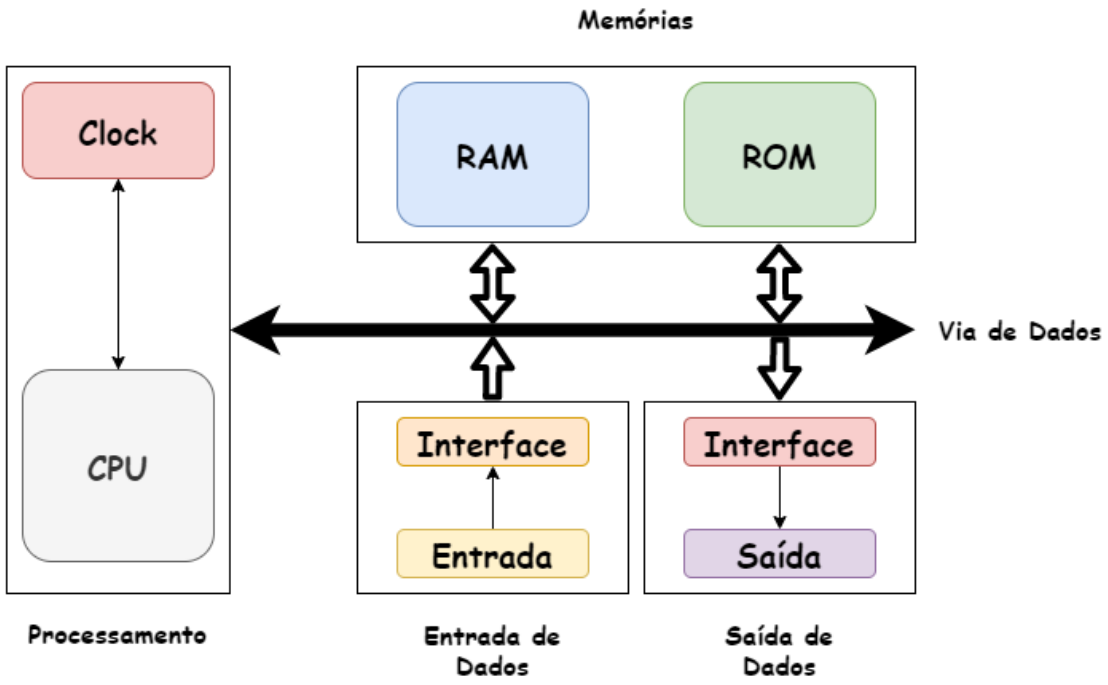
O valor δ, que é a resolução, indica o menor incremento que o conversor consegue detectar.

2.5 Microcontroladores

Os microcontroladores são circuitos integrados que embarcam os componentes mais básicos de um computador (CPU, memórias e pinos de entrada) (TOCCI; WIDNER; MOSS, 2011). Por virem em tamanho reduzido e compactado em um único *chip*, os microcontroladores são recomendados para tarefas simples em virtude de seu baixo poder computacional, porém, naturalmente, são bem mais baratos do que uma máquina comum (BINDAL, 2017). A Figura 9 mostra a arquitetura de um microcontrolador genérico.

O primeiro ponto a ser notado é a presença de uma CPU e o *clock* em um único bloco funcional, permitindo o processamento de dados sem um circuito adicional. As memórias, por sua vez, estão presentes em outro bloco funcional, de modo que a RAM envia as instruções do programa para a CPU (TOCCI; WIDNER; MOSS, 2011). A ROM, por outro lado, é usada para

Figura 9 – Arquitetura de um Microcontrolador



Fonte: Adaptado de Tocci; Widner; Moss, 2011.

guardar os binários do programa a ser executado. Além disso, há um espaço dedicado para o programador inserir informações persistentes no dispositivo (BINDAL, 2017).

Dando sequência à análise, há os blocos de entrada e saída de dados. Cada um deles possui sua respectiva interface para viabilizar a comunicação de acordo com os padrões da CPU (TOCCI; WIDNER; MOSS, 2011). Outro ponto muito importante é a chamada Via de Dados, a qual consiste em barramentos por onde trafegam todos essas informações e os pulsos de controle da CPU para os demais blocos (FERNANDEZ, 2015). É possível que microcontroladores sejam mais complexos do que o modelo genérico da Figura 9, como é o caso do que é usado no presente trabalho: o ESP32 DevKit V1.

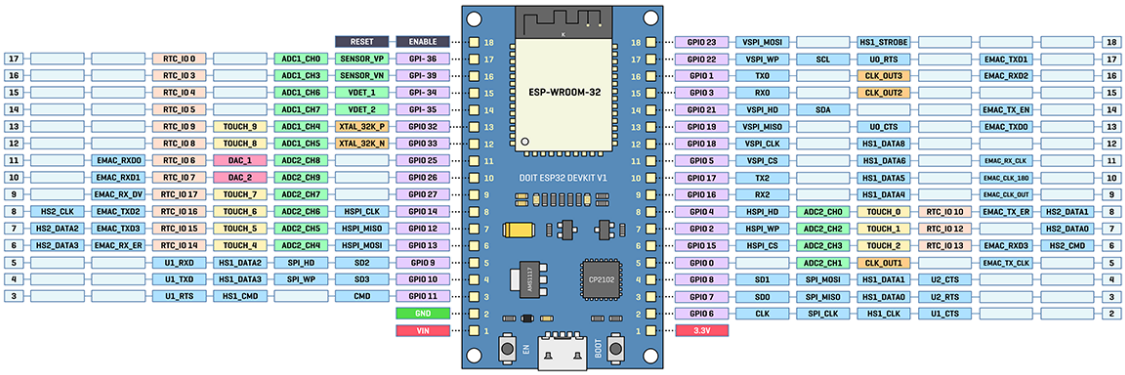
2.5.1 ESP32 DevKit V1

A Figura 10 mostra o diagrama de pinos do ESP32 DevKit V1. Por se tratar de um kit de desenvolvimento, o microcontrolador vem com alguns circuitos embutidos. Alguns serão aqui destacados (PLATFORM IO, [S.d) (ESPRESSIF SYSTEMS, 2019):

- Alguns pinos possuem um conversor analógico-digital embutido para fazer a leitura de grandezas analógicas;
- O microcontrolador possui um relógio embutido para o caso de precisar lidar com datas ou horas;

- O ESP32 DevKit V1 já vem com módulos *WiFi* e *Bluetooh* integrados, o que torna o microcontrolador indicado para IoT.

Figura 10 – Diagrama de Pinos do ESP32 DevKit V1



Fonte: Adaptado de
https://www.reddit.com/r/esp32/comments/b5mkag/a_better_pinout_diagram_for_esp32_devkit/d

É importante também tecer alguns comentários sobre as especificações do microcontrolador (PLATFORM IO, [S.d) (ESPRESSIF SYSTEMS, 2019):

- *Clock* de $240\ MHz$;
- Palavras de 32 bits;
- Memória *Flash* de $4\ KB$;
- Memória RAM de $320\ KB$;
- Conversores Analógico-Digitais de 12 bits e V_{ref} de $3,3\ V$;
- Pode ser alimentado com $3,3\ V$ ou $5\ V$;
- Tensão máxima nos pinos de entrada e saída: $3,3\ V$;
- *WiFi* na frequência de $2,4\ GHz$;
- Pode ser programado tanto em C++ quanto em *Python*.

2.6 O Sistema Embarcado Projetado

Expostos os conceitos que alicerçam o circuito de monitoramento, aqui serão feitas algumas ponderações acerca da arquitetura do sistema embarcado. O ponto a ser abordado é sobre os sensores: DHT 11 para temperatura e umidade; Guva S12SD para radiação ultravioleta; MQ-135 para fumaça e gases tóxicos e um sensor infravermelho para incêndio.

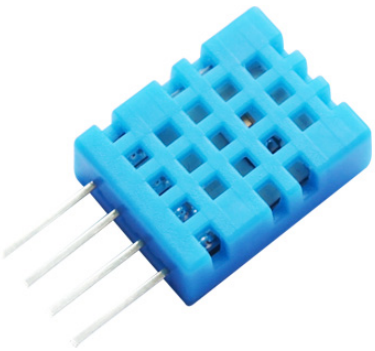
2.6.1 DHT 11

Esse sensor é fabricado pela empresa chinesa Aosong, de modo que as informações podem ser encontradas em (Aosong, [S.d]). Em resumo, eis as principais especificações do sensor:

- Tensão de Alimentação: 3.3 a 5 V_{DC};
- Faixa de Medição de Temperatura: -20 a 60 ^oC;
- Faixa de Medição de Umidade: 5% a 95%;
- Corrente em *Stand by*: $60\mu A$;
- Corrente Durante Medições: $0.3mA$;
- Necessário Resistor de *pull-up* de no Mínimo $5.1k\Omega$.

Uma última observação a ser dada é que esse sensor emite sua saída através do pino *DATA*, a qual já é em formato digital com 16 bits. A Figura 11 mostra o sensor.

Figura 11 – Sensor DHT11



Fonte:http://aosong.com/userfiles/images/product/dht11/DHT11-1.jpg

2.6.2 Guva S12SD

Principais informações (Adafruit, [S.d]):

- Tensão de Alimentação: de 2.5 a $5V$;
- Temperatura de Operação: de -30 a 85^oC;

Antes de detalhar o funcionamento desse sensor, é necessário conhecer a grandeza de saída com que ele trabalha, o índice UV (UVi). Esse índice mede a intensidade de radiação ultravioleta em um número inteiro, de modo a sinalizar as precauções necessárias, conforme listado nos intervalos abaixo (SANTOS, 2009):

- 1 e 2: Não há precauções;
- 3 a 7: Evitar exposição prolongada ao sol. Preferencialmente, usar protetor solar e procurar sombras;
- Acima de 8: Evitar qualquer exposição ao sol.

Esse sensor é composto por duas partes: um fotodiodo, responsável pela conversão dos raios ultravioletas para corrente elétrica, fabricado pela Roithner LaserTechnik (Roithner LaserTechnik, 2011), e um circuito que padroniza os níveis de tensão para microcontroladores, desenvolvido pela Adafruit (Adafruit, [S.d]). O conjunto é comercializado pela própria Adafruit e possui a relação entre índice UV e tensão de saída dada por (21) (Adafruit, [S.d]):

$$UVi = \frac{v_0}{0.1} \tag{21}$$

Onde v_0 é a tensão de saída em V. Dado que a corrente máxima no fotodiodo é de $0,4\mu A$ (Roithner LaserTechnik, 2011) e que a relação exposta em (22) entre tensão de saída e corrente no fotodiodo é válida (Adafruit, [S.d]):

$$v_o = 4.3 * i_0 \tag{22}$$

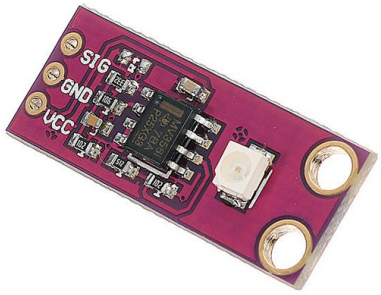
Sendo i_0 a corrente em μA, tem-se que a tensão máxima de saída é $1,72V$, correspondendo a um índice de cerca de 17. A Figura 12 mostra o sensor.

2.6.3 MQ-135

O sensor MQ-135, fabricando pela Hanwei Electronics, deve ser alimentado com $5V$ e possui o circuito equivalente mostrado na Figura 13 (Hanwei Electronics, [S.d]), onde R_s é a resistência do sensor, a qual varia de acordo com a concentração dos gases mensurados, e R_L é a resistência de carga, a qual deve ser de 10 a $47k\Omega$ (Hanwei Electronics, [S.d]). Dito isso, conhecidos a tensão de saída v_0 e o valor de R_L, é possível encontrar R_s a partir da Equação (16):

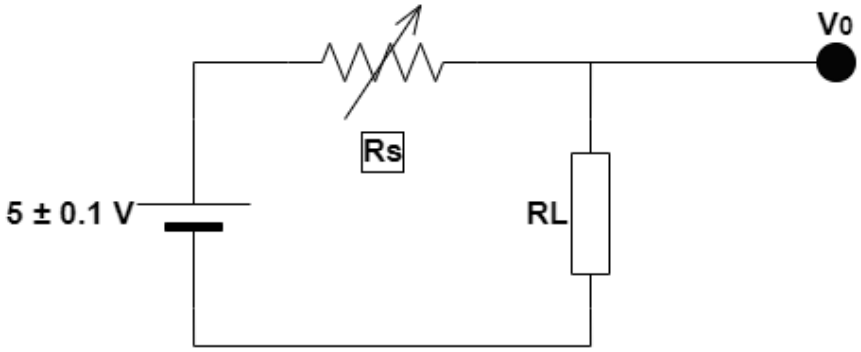
$$v_0 = 5 \cdot \frac{R_L}{R_L + R_s} \Rightarrow 5R_L = v_o R_L + v_0 R_s \Rightarrow R_s = R_L \frac{5 - v_0}{v_0} \tag{23}$$

Figura 12 – Sensor Guva S12SD



Fonte: https://uploads.filipeflop.com/2019/10/9SS34-3.jpg

Figura 13 – Circuito Equivalente do MQ-135



Fonte: Adaptado de https://www.electronicoscaldas.com/datasheet/MQ-135_Hanwei.pdf

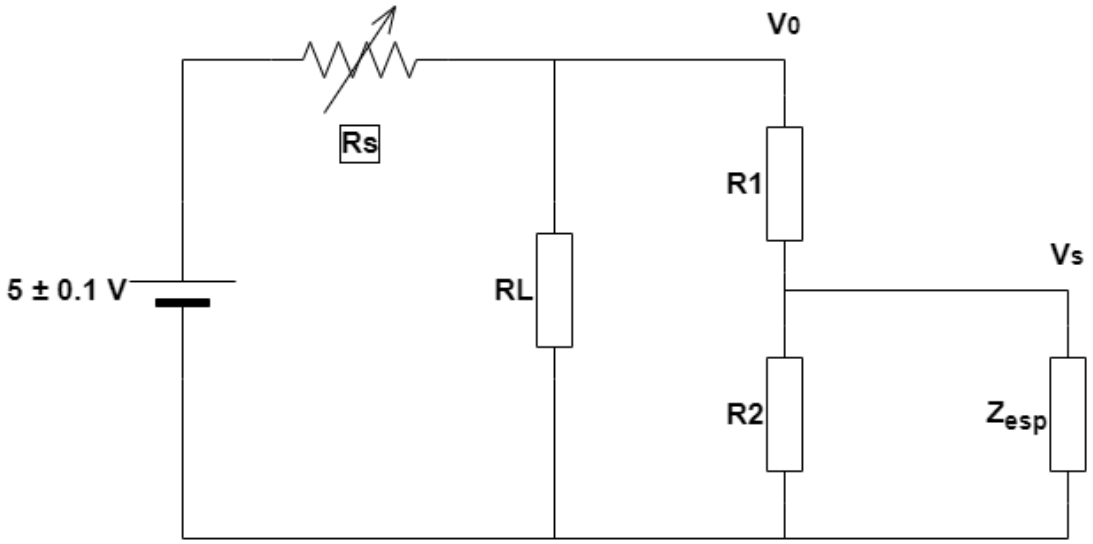
Portanto,

$$R_s = R_L \frac{5 - v_0}{v_0} \tag{24}$$

Todavia, o ESP 32, conforme já mencionado, lê até $3.3V$, portanto, será necessário aplicar um divisor de tensão entre a saída do sensor e o entrada do microcontrolador, conforme Figura 14.

Em consonância com o que já foi mencionado, as impedâncias de entrada dos microcontroladores são da faixa de Megaohms, portanto, Z_{esp} pode ser ignorado por entrar na causa limítrofe dos divisores de tensão ($Z_{esp} \to \infty$). Entretanto, os resistores R_1 e R_2 deverão ser considerados e serão da mesma ordem de R_L, pois, caso eles sejam muito maiores que R_L, eles ficarão próximos de Z_{esp}, inviabilizando a premissa anterior. Dito isso, a nova fórmula para encontrar R_s é mostrada na Equação 25:

Figura 14 – Circuito Equivalente do MQ-135 Completo



$$R_s = R_L^{'} \frac{5 - v_0}{v_0} \tag{25}$$

Onde

$$\frac{1}{R_L^{'}} = \frac{1}{R_L} + \frac{1}{R_1 + R_2} \qquad R_L^{'} \in [10, 47]\, k\Omega \tag{26}$$

A Equação (26) já deixa explícito também a necessidade de $R_L^{'}$, assim como R_L, obedecer à restrição mencionada pelo *datasheet*.

Um outro ponto a ser abordado é que o microcontrolador estará medindo v_s, dessa forma, é necessário encontrar v_0 usando o caminho inverso da Equação (16), ou seja, conforme Equação (27):

$$v_o = v_s \frac{R_1 + R_2}{R_2} \tag{27}$$

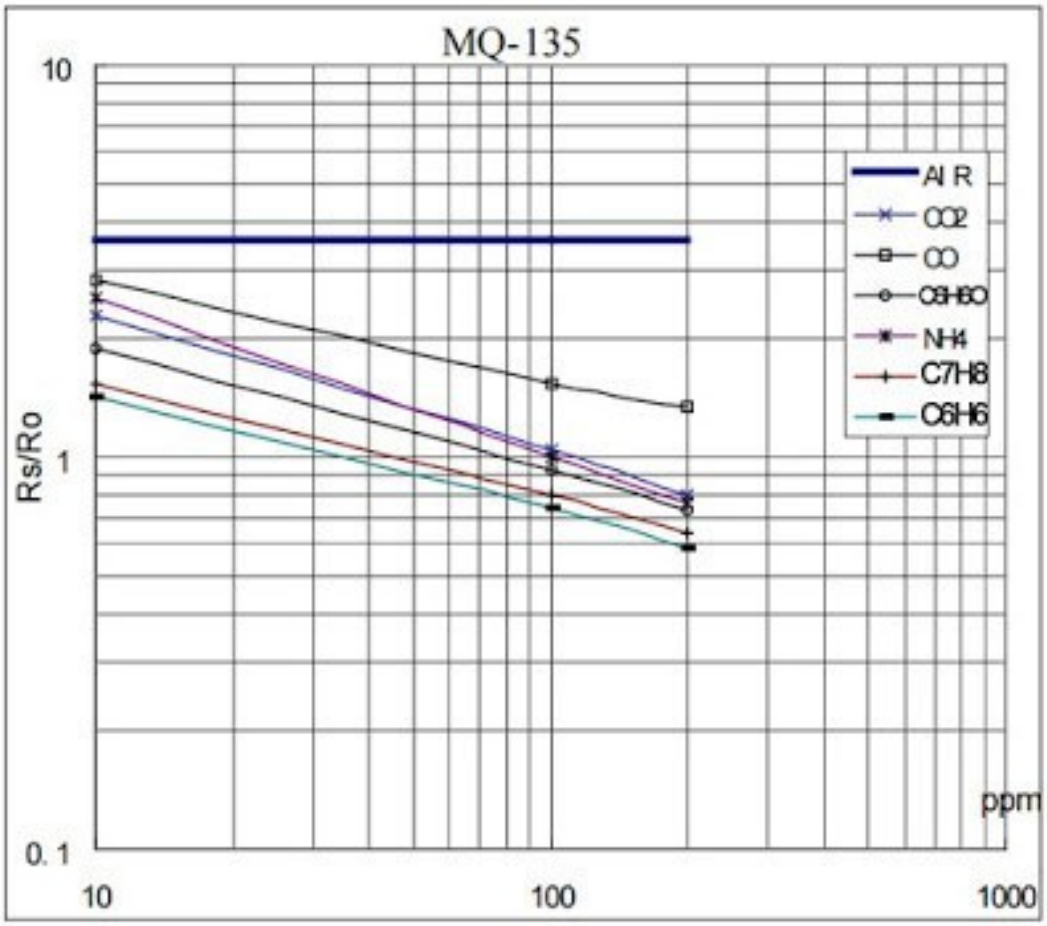
Para finalizar, uma vez que todos os cálculos serão feitos via algoritmo, não há necessidade de juntar as equações em uma única expressão.

Até aqui foram os passos necessários para encontrar R_s. A seguir, será exibido como extrair as informações de medição a partir desse valor. Para isso, salvo quando explícito ao contrário, as informações são provenientes do *datasheet* (Hanwei Electronics, [S.d]).

Em primeiro lugar, é necessário conhecer R_0, a resistência do sensor em condições pré-determinadas no *datasheet*. O gráfico da Figura 15 mostra que, para o ar limpo, $\frac{R_s}{R_0}$ é cerca de 3.8, logo, para descobrir R_0, deve-se realizar o seguinte procedimento com o sensor monitorando o ar limpo:

$$R_0 = \frac{R_s}{3.8} \tag{28}$$

Figura 15 – Curvas do MQ-135



Fonte: https://portal.vidadesilicio.com.br/sensor-de-gas-mq-135/

Feito esse passo, é possível descobrir as concentrações dos outros gases. No gráfico exibido na Figura 15, estão contidas as retas que descrevem o funcionamento do sensor.

Portanto, o ESP32 lê v_s, a partir do qual calcula-se R_s. Com o valor de R_0, por sua vez, encontra-se a medida $\frac{R_s}{R_0}$, de modo a ser possível descobrir a concentração do gás desejado a partir da reta no gráfico. No presente trabalho, a reta para o gás monóxido de carbono (CO) será usada para mensurar fumaça. Em contrapartida, o gás tolueno (C_7H_8) se encaixará nas medições de gases tóxicos.

Observe que o gráfico está em escala logarítmica, portanto, as equações são da forma:

$$\frac{R_s}{R_0} = \alpha 10^{\beta ppm} \tag{29}$$

Porém, de acordo com as Equações (30) e (31), é possível linearizar a Equação (29) aplicando logaritmo de ambos os lados:

$$log(\frac{R_s}{R_0}) = log(\alpha) + \beta \cdot ppm \tag{30}$$

$$Y = A + \beta X \tag{31}$$

Onde, $Y = log(\frac{R_s}{R_0})$, $A = log(\alpha)$ e $X = ppm$. Com isso, basta apenas olhar dois pontos em cada reta, $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$, e resolver o sistema linear resultante.

Finalizando a análise, a Equação (29) mostra como obter $\frac{R_s}{R_0}$, porém, para essa aplicação, o foco é justamente a função inversa:

$$ppm = \frac{log(\frac{R_s/R_0}{\alpha})}{\beta} \tag{32}$$

Portanto, para CO, tem-se os seguintes pontos e função:

$$P_1 = (10, log(2.9)) \qquad P_2 = (100, log(1.6)) \tag{33}$$

$$ppm = \frac{log(\frac{R_s/R_0}{3.095})}{-0.00286} \tag{34}$$

O C_7H_8, por usa vez, possui a seguinte relação:

$$P_1 = (10, log(1.5)) \qquad P_2 = (100, log(0.8)) \tag{35}$$

$$ppm = \frac{log(\frac{R_s/R_0}{1.608})}{-0.003} \tag{36}$$

Por último, a Figura 16 mostra uma foto do sensor usado.

Figura 16 – Sensor MQ-135

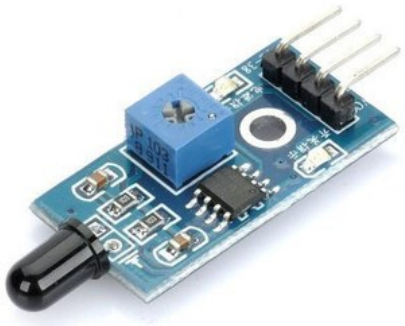
Fonte: https://www.filipeflop.com/produto/sensor-de-gas-mq-135-para-gases-toxicos/

2.6.4 Infravermelho

O sensor infravermelho para incêndio é o mais simples, pois emite um sinal digital que é verdadeiro se houver presença de fogo e falso caso contrário.

Sendo formado basicamente por um fotodiodo e um amplificador operacional, sua tensão de alimentação é de 3.3 a $5V$. Por fim, a Figura 17 mostra esse sensor.

Figura 17 – Sensor Infravermelho



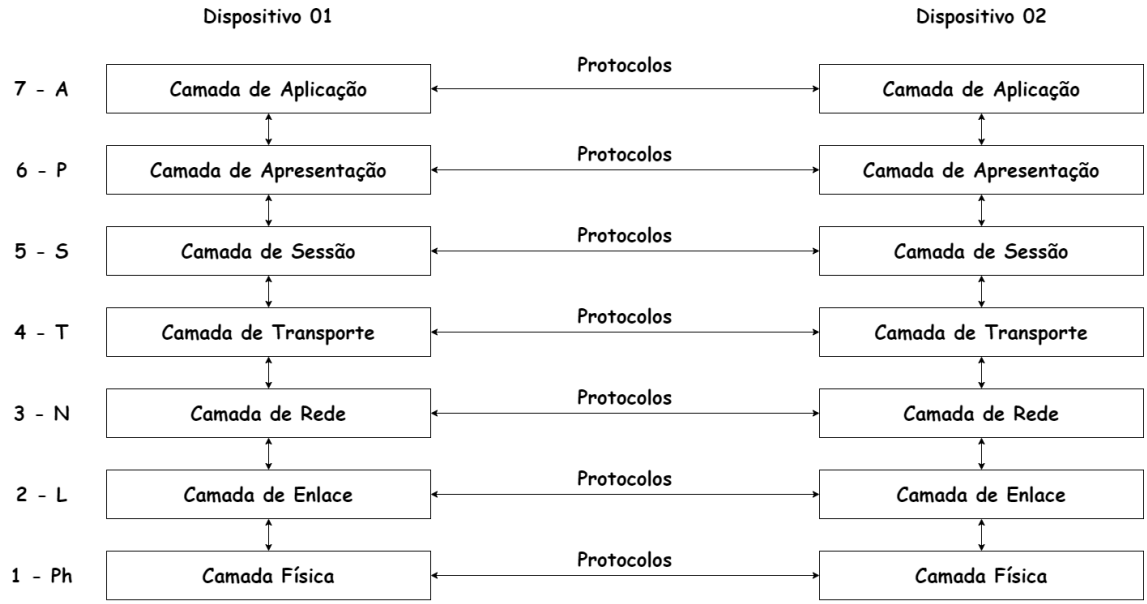
Fonte: https://uploads.filipeflop.com/2017/07/9SS21-2-4-pinos.jpg

3 REDES DE COMPUTADORES E SEGURANÇA

Conforme (SINGH; SINGH, 2015), a comunicação entre dispositivos é imprescindível para o bem-estar da humanidade. Dito isto, para tal, é necessário que certos padrões sejam obedecidos. O primeiro deles a ser aqui abordado é o padrão OSI (*Open System Interconnect*), o qual é um modelo abstrato de como os protocolos de comunicação devem ser organizados (ROCHOL, 2012).

De acordo com (LI et al., 2011), o modelo OSI é composto por sete camadas, de modo a padronizar a comunicação entre diversos dispositivos. Repetindo o proposto por (ROCHOL, 2012), cada camada só consegue se comunicar com a imediatamente superior ou com a que está logo abaixo. Além disso, entre dois dispositivos que obedecem ao padrão OSI, a comunicação ocorre apenas entre camadas do mesmo nível. A Figura 18 ilustra esse conceito.

Figura 18 – Camadas do Padrão OSI



Adaptado de Li et al, 2011

Na Figura 18 é possível ver as sete camadas do padrão do OSI. As definições de cada camada, em consonância com (LI et al., 2011), (ALURA, 2018) e (ROCHOL, 2012), são as seguintes:

- Camada Física: É o nível mais baixo e aborda a concretização da rede, ou seja, cabos, conectores, transporte de bits, entre outras coisas;
- Camada de Enlace: É a camada que coordena o fluxo de dados. Nessa camada os destinatários são estabelecidos e erros são verificados para que possam ser corrigidos antes de a informação subir de camada. Aqui se encontram as subcamadas MAC e LLC, discutidas posteriormente;

- Camada de Rede: Essa camada prioriza a ordem de envios de informação e estabelece qual a melhor rota para enviá-la de um dispositivo para o outro. Aqui se encontram os protocolos IP e ICMP, por exemplo.
- Camada de Transporte: Camada responsável pelo transporte da informação. Nela estão contidos os protocolos TCP e UDP;
- Camada de Sessão: Para acontecer o transporte, essa camada precisa sincronizar os dispositivos. Somente após esse processo é que a informação é enviada;
- Camada de Apresentação: Essa camada é responsável por tratar os dados da camada anterior e mandar para a próxima. Aqui estão serviços de criptografia e conversão de bits em caracteres, por exemplo;
- Camada de Aplicação: É a camada de consumo de dados. É onde ocorre a interação com o usuário. Em termos de procolo, essa camada engloba o HTTP, DNS e FTP.

Na Figura 18 é possível ver uma identificação numérica à esquerda de cada camada. Essa identificação numérica representa o nível atrelado a elas. Além disso, cada um desses níveis possui uma sigla, também exposta à esquerda.

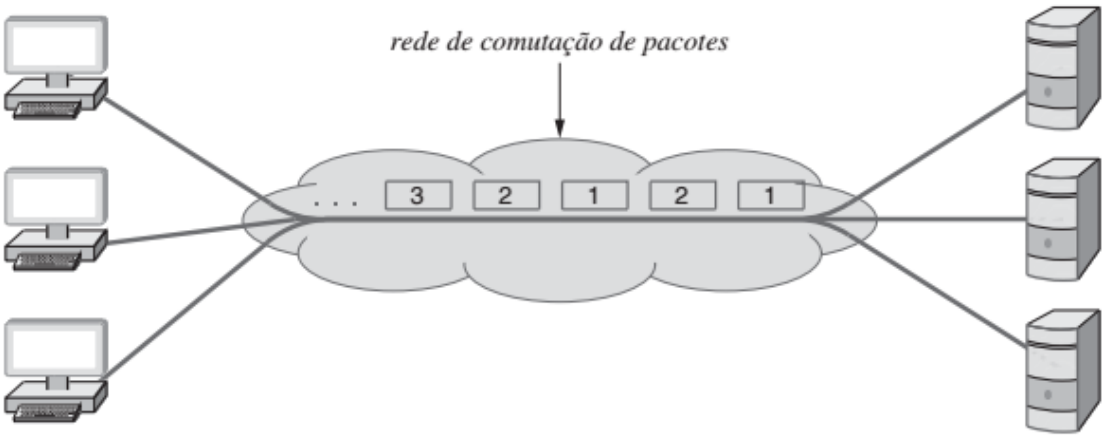
Ainda dentro da Figura 18, as camadas se comunicam entre si através de protocolos. Segundo (LI et al., 2011), é possível abstrair essas camadas e afirmar que, de um dispositivo para o outro, apenas camadas de mesmo nível podem se comunicar. Porém, (ROCHOL, 2012) alerta que, do ponto de vista puramente físico, antes de chegar na camada destino, a informação passa pela camada Física, pois é lá onde de fato existe conexão.

3.1 Comunicação por Pacotes

Naturalmente, uma comunicação entre dois dispositivos precisa de um elo físico entre eles. Porém, dada a necessidade de otimização de recursos e espaços, técnicas de multiplexação podem ser aplicadas com o intuito de usar o mesmo canal para vários enlaces (TOCCI; WIDMER; MOSS, 2018).

Nesse contexto, (COMER, 2016) explica a serventia da comunicação por pacotes, ilustrada na Figura 19. Cada remetente envia um pacote por vez para o canal. Esse pacote é composto por identificações do destinatário e remetente, bem como por partes da informação. Ainda, (COMER, 2016) salienta que, se uma máquina não tiver pacote para a enviar, a vez é passada para a próxima de modo a não deixar a rede ociosa.

Figura 19 – Comutação por Pacotes



Fonte: Comer, 2016

3.2 Padrão IEEE 802

Segundo (COMER, 2016), o padrão IEEE 802 surgiu para padronizar a redes de computadores e ele contempla as especificações da camada 01 e da camada 02. As demais camadas são abordadas por outras instituições.

Nesse padrão, o IEEE divide a camada 02 em duas subcamadas: a LLC (*Logical Link Control*) e MAC (*Media Access Control*). Enquanto a primeira padroniza endereços para cada máquina na rede, a segunda aborda o processo de compartilhamento de meio físico pelos envolvidos (COMER, 2016). O autor ainda salienta que essas subcamadas não são etapas da comunicação, são apenas abordagens com focos distintos. Em outras palavras, a subcamada LLC identifica unicamente cada participante e a subcamada MAC cuida da demultiplexação.

Em consonância com (COMER, 2016), o endereço único do IEEE recebe o nome de endereço MAC (embora seja uma atribuição da subcamada LLC), sendo composto por 48 bits. Como cada endereço é único, cada interface de rede tem o seu. A atribuição de endereços se dá da seguinte forma: os primeiros 24 bits representam o fornecedor da interface de rede; já os outros 24, por sua vez, é o número dado pelo fornecedor ao próprio hardware. Ainda segundo o autor, é devido ao endereço MAC que o processo de demultiplexação entrega a mensagem ao destinatário correto.

3.3 Endereços IP

Segundo (COMER, 2016), uma vez que endereços MAC têm suas definições arbitrárias em relação à tecnologia que representa, tem-se a necessidade de usar endereços uniformemente padronizados para identificar cada máquina na rede independentemente de seu endereço MAC.

Esse endereço é chamado de IP (*Internet Protocol*).

Alinhado com (COMER, 2016), a versão do procolo de IP atual, o IPv4, possui endereços de 32 bits. Porém, é alertado por (XU; LIU, 2020) que esse formato de endereçamento IP não tem uma quantidade de endereços disponíveis que conseguirá atender a todos os dispositivos no futuro, logo, pesquisadores trabalham em uma nova versão, a IPv6, com 128 bits. Todavia, como ainda hoje a versão 4 é a padrão, o presente trabalho emprega o IPv4, o qual será chamado apenas de IP.

Um endereço IP, segundo (JAYANTHI; RABARA, 2010), pode ser dividido em duas partes: o prefixo, que é uma identificação única da rede a qual o computador está conectado, e o sufixo, que representa a identificação única da máquina dentro daquela rede. Um corolário a ser extraído é: redes distintas podem usar os mesmos sufixos, pois seus prefixos serão diferentes e, por consequência, os IPs de cada membro das redes serão únicos.

Um ponto elucidado por (TANEMBAUM, 2011) é quantidade de bits destinada a cada parte do IP. Se houver muitos bits para o prefixos, tem-se um grande número de redes, porém cada uma com poucos participantes. O contrário também é verdadeiro, ou seja, muitos bits para o sufixo deixaria poucas redes com muitas máquinas. A resposta apresentada pelo autor é deixar essa definição arbitrária, conforme será visto a seguir.

Para customizar o tamanho de prefixo, tanto (TANEMBAUM, 2011) quanto (COMER, 2016) introduzem uma nova informação: a máscara de sub-rede ou máscara de endereço. Essa máscara é composta por 32 bits, de modo que, o que for prefixo recebe bit 1. Por outro lado, o que for sufixo receberá bit 0.

3.3.1 Representação de IPs para Humanos

Seres humanos, naturalmente, precisam entrar em contato com endereços IPs. Para tal, segue-se o uso da notação decimal pontilhada. Parafraseando (COMER, 2016), essa notação consiste em agrupar os bits em octetos e escrever a representação decimal de cada um separadas por pontos. Um exemplo pode ser visto em (37).

$$10000001\ 00110100\ 00000110\ 00000000 \rightarrow 129.52.6.0 \tag{37}$$

Para incorporar a máscara de sub-rede, (COMER, 2016) apresenta a notação CIDR (*Classless Inter-Domain Routing*). Essa notação acrescenta uma barra e o número de bits usados pelo prefixo. Como exemplificação, considere a notação decimal pontilhada para a máscara de endereço em (38). A notação CIDR do IP mostrado em (37) com a máscara mostrada em (38) é apresentado em (39).

$$11111111\ 11111111\ 11111111\ 00000000 \rightarrow 255.255.255.0 \tag{38}$$

$$129.52.6.0/24 \tag{39}$$

3.3.2 Endereços Especiais

Existem alguns endereços IPs especiais, conforme argumentado por (COMER, 2016). O primeiro deles é o endereço de rede, o qual é o primeiro endereço da rede, ou seja, aquele cujo todos os bits do sufixo são "0". Como exemplo, tem-se $192.168.100.0/24$, isto é, todos os bits do último octeto do IP são "0". O endereço de rede, como o próprio nome sugere, é usado para identificar a rede.

O segundo endereço especial é o de *broadcast*, representado quando todos os bits do sufixo são "1". Voltando à rede anterior, o IP de *broadcast* seria $192.168.100.255/24$. Esse IP é usado para enviar uma cópia da informação para todos os integrantes da rede.

Há ainda um terceiro IP especial, porém, esse não é da rede, e sim da máquina (COMER, 2016). Ainda segundo o autor, o prefixo reservado para isso é 127/8, porém, a convenção é usar $127.0.0.1$ para referenciar a própria máquina. Esse IP é chamado de *loopback*.

3.4 Conexão Entre Redes

Segundo (COMER, 2016), um roteador é um dispositivo capaz de conectar redes distintas, de modo que ambas possam trocar informações. Em sequência, o autor ainda afirma que a internet é definida como várias redes interconectadas. A partir disso, (BRITO, 2017) menciona o problema das limitações de endereços IP. Para solucionar tal problema foi criado o protocolo NAT (*Network Address Translation*, Tradução de Endereços de Rede em tradução livre).

O NAT foi proposto por (REKHTER et al., 1996) e seu conceito começa dividindo os endereços IPs em públicos e privados. Os IPs públicos são únicos e distribuídos por autoridades credenciadas, conforme exposto por (COMER, 2016). Os IPs privados, por sua vez, são destinados à redes particulares e podem ser definidos arbitrariamente, contanto que não coincidam com um público. De início, somente os IPs públicos são permitidos trafegar na internet (REKHTER et al., 1996). Para IPs privados terem conexão, portanto, é necessário o NAT (BRITO, 2017).

Dito isso, o NAT consiste em traduzir um IP privado para um IP público (BRITO, 2017). Para entender o conceito, considere o exemplo: um roteador tem duas placas de rede, uma com um IP público (comprado de uma autoridade) e outra conectada a uma rede privada, com IPs arbitrários. Ao pedir conexão com a internet, a máquina com IP privado manda a requisição para o roteador, onde esse repassa essa requisição para a internet, porém, troca o IP privado da máquina original pelo público que ele tem. Esse processo de troca é a tradução (REKHTER et al.,

1996). Na informação de retorno, o roteador entende que se trata de uma resposta e encaminha a informação para a máquina que a pediu inicialmente.

Exposto esse conceito, é possível extrair algumas deduções. A primeira delas é que os IPs privados, embora arbitrários, não devem coincidir com os IPs públicos, estes já reservados para serem vendidos pelas autoridades. Outro corolário é: duas redes distintas podem usar os mesmos IPs privados, pois as mesmas não vão se comunicar sem passar pelo NAT.

3.5 Exemplo

A Figura 20 ilustra um exemplo geral acerca de máscara de sub-rede e NAT. Nela, é possível ver a máscara $255.255.255.240$ escrita em binário, o que leva a duas redes distintas: $10.0.0.0/28$ e $10.0.0.16/28$. Como é possível ver nos endereços IPs de rede em binário, a parte verde representa os sufixos, que podem variar livremente dentro da rede. O restante dos bits são os prefixos, que só mudam quando a rede for diferente. Em particular, o bit em azul mostra justamente o que mudou de uma rede para a outra.

É possível ver ainda que as duas redes estão ligadas a um roteador, o qual também está ligado à internet. Para que as máquinas consigam acesso, é necessário que o roteador aplique o protocolo NAT e traduza a requisição, de modo que os demais computadores na internet vejam o pedido oriundo da rede interna como tendo sido feito pelo IP $203.0.113.2$. Na volta, o roteador entende que se trata de uma resposta e encaminha para o devido computador na rede interna.

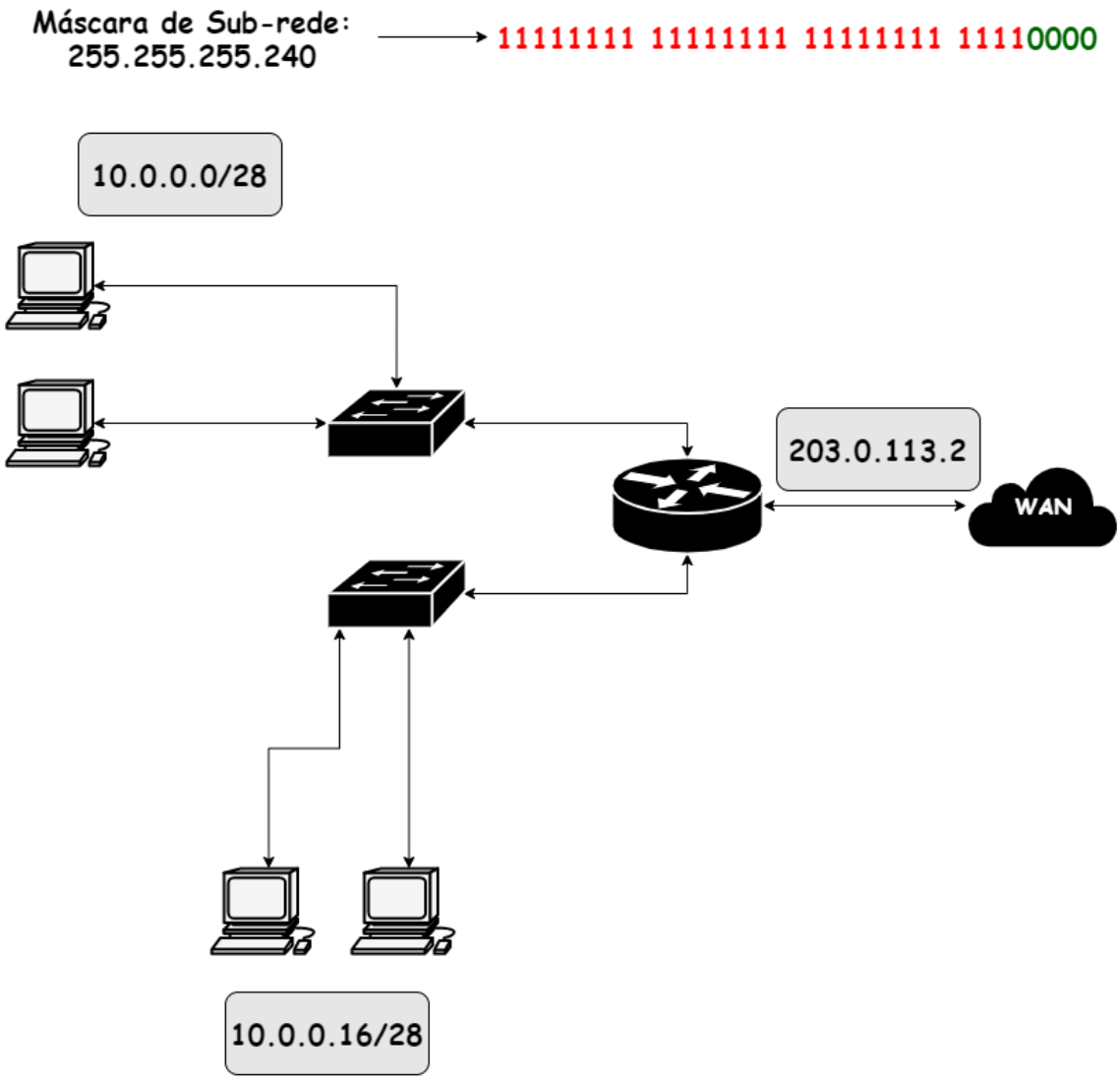
3.6 TCP e UDP

É alertado por (COMER, 2016) que o protocolo IP consegue identificar somente máquinas na rede, ou seja, ele não interpreta os bits da informação em si. Em outras palavras, se dois aplicativos estiverem usando a conexão simultaneamente, o protocolo IP não conseguirá distinguir a qual aplicativo a informação pertence. Para sanar esse problema, (COMER, 2016) afirma que foram criados dois protocolos na camada 04 para fazer essa distinção: os protocolos TCP (*Transmission Control Protocol*)e UDP (*User Datagram Protocol*).

Ainda conforme o autor, o protocolo UDP foi feito para ser mais rápido, sendo indicado para vídeo e voz. Para conseguir tal velocidade, esse protocolo dispensa verificações de correção, ordem e integridade de pacotes. Como resultado, (COMER, 2016) afirma que o UDP é usado apenas em situações onde a velocidade é primordial e a perda de alguns pacotes não causa estragos significativos. Outros dois exemplos são dados por (BRITO, 2017): jogos e DNS.

Por outro lado, o protocolo TCP foi feito para dar maior confiabilidade. Para tal, ele primeiro estabelece uma conexão antes de enviar os dados (chamada de *handshake*). Após isso, os dados são enviados e há garantia de que os pacotes chegarão na ordem e íntegros (COMER,

Figura 20 – Exemplo de Máscara de Sub-redes e NAT



10.0.0.0/28 : 00001010 00000000 00000000 00000000

10.0.0.16/28 : 00001010 00000000 00000000 00010000

2016). Por ser mais confiável, apesar de mais lento, o protocolo TCP é usado na maioria das aplicações (BRITO, 2017).

3.7 HTTP

O HTTP (*HyperText Transfer Protocol*) é o protocolo principal para transferência de dados entre um cliente e um servidor web. Segundo (COMER, 2016), esse protocolo possui as seguintes características:

- Funciona através de palavras-chave;
- É possível enviar e baixar dados;
- Transfere arquivos binários.

Uma vez que os últimos dois itens são autoexplicativos, a explicação a seguir esmiúça a comunicação por palavras-chave. Primeiramente, a conexão entre cliente e servidor é estabelecida (*handshake*). Após isso, a requisição é feita a partir dos verbos HTTP (COMER, 2016) (RESCA, 2019). Os principais são listados na Tabela 3:

Tabela 3 – Principais verbos HTTP

Verbo	Finalidade
GET	O servidor envia dados para o cliente
POST	O cliente envia um novo dado para ser inserido no servidor
PUT	O cliente atualiza dados já existentes no servidor
DELETE	O cliente deleta dados já existentes no servidor
HEAD	O servidor envia seu status para o cliente

Fonte: Adaptado de Comer, 2016

Para entender a estrutura de uma requisição HTTP, considere o seguinte exemplo oriundo de (COMER, 2016):

GET /item versão CRLF

A primeira parte, naturalmente, é o verbo HTTP. o trecho */item* representa o tipo de dado a ser acessado (ou recurso, como será explicado nos próximos capítulos). Em seguida, a versão do HTTP é especificada e, por fim, o CRLF indica qual caractere será usado para indicar quebra de linha.

Sobre a resposta, é importante frisar a primeira linha dela, a qual contém um código de status. Esse código de status indica se a requisição foi bem sucedida ou não. Os principais estão listados na Tabela 4 (COMER, 2016), (RESCA, 2019) e (BRITO, 2017):

Tabela 4 – Principais Códigos HTTP

Código	Descrição
200	Sucesso
400	Solicitação inválida
404	Não encontrado
500	Erro interno na aplicação ou no servidor

Fonte: Adaptado de Comer, 2016

3.8 Criptografia com Chaves

Em consonância com (BROWN; STALLINGS, 2017), a criptografia é usada para proteger dados, de modo que só é possível entender a informação através de cifras distribuídas para indivíduos selecionados. Dito de outra forma, a criptografia é uma maneira de garantir que somente as pessoas certas conseguirão acessar a informação que está sendo enviada.

As tecnologias de criptografia usadas nesse trabalho se baseiam em chaves, portanto, o detalhamento será dado nesse contexto. Primeiramente, é possível entender as chaves como cadeias de bits que possibilitam tanto a codificação da mensagem quanto sua interpretação (COMER, 2016). O segundo ponto a ser entendido sobre elas é que existe a criptografia simétrica e a assimétrica, onde, na primeira, os envolvidos recebem a mesma chave e, na última, são distribuídas chaves diferentes (BROWN; STALLINGS, 2017).

Na modalidade simétrica são distribuídas chaves privadas para os envolvidos, onde essas são as responsáveis tanto pela codificação quanto pela decodificação. O nome privado é dado fazendo alusão à necessidade de não serem compartilhadas com mais ninguém (BROWN; STALLINGS, 2017). No contexto assimétrico, há chaves públicas e privadas, onde, nesse caso, a pública pode ser distribuída. A comunicação com os dois tipos se dá da seguinte forma (COMER, 2016): a chave pública só consegue descriptografar mensagens codificadas pela privada e, por sua vez, a chave privada é usada para revelar mensagens criptografadas pela contraparte pública.

3.9 SSH

De acordo (BRITO, 2017), tradicionalmente, servidores Linux são instalados sem interface gráfica, ou seja, a configuração é feita via linha de comando ou CLI (*Command Line Interface*, Interface de Linha de Comando). Isso acontece, ainda segundo o autor, devido a alguns fatores. Dentre eles:

- Universalidade de comandos. O Linux possui várias interfaces gráficas ou GUI (*Graphical User Interface*, Interface Gráfica do Usuário), porém, os comandos textuais são idênticos ou muito parecidos;

- Capacidade de automação. Conhecidos os comandos, é possível escrever *scripts* e automatizar o processo;
- Economia de recursos. Servidores apenas com CLI são mais econômicos e fornecem os mesmos resultados quando comparados à contraparte gráfica.

Além dos citados acima, existe a possibilidade de se conectar remotamente ao servidor por linha de comando através do protocolo SSH (*Secure Shell*, Shell Seguro) (NOAL, 2015). Vale dizer que o protocolo SSH criptografa a conexão, de modo a torná-la segura.

O trabalho de (YLONEN, 2019) argumenta que é possível incrementar a segurança na comunicação SSH através de chaves de usuário. Segundo a mesma fonte, cada usuário possui duas chaves, uma privada, a qual ele deve manter em segredo consigo, e uma pública, que deve ser distribuída para todos os servidores que esse usuário quer ter acesso.

Vale salientar que essa metodologia é mais segura porque, além de as chaves serem bem mais compridas do que as senhas comumente usadas, as pessoas tendem a ser descuidadas com senhas (MORENO, 2019).

Para entender a autenticação usando chaves, é importante assimilar primeiro que, no enlace SSH, a chave privada é usada para decriptar mensagens. A pública, por sua vez, é usada para encriptar (YLONEN, 2019). Dito isso, a Figura 21 representa as etapas do processo.

A Figura 21 informa, inicialmente, que o usuário mantém as duas chaves, enquanto o servidor fica somente com a pública (YLONEN, 2019). Na próxima etapa, ao requisitar a conexão, o cliente SSH (a mando do usuário) informa ao servidor qual a chave pública dele, de modo que o servidor encripta uma mensagem com essa chave e manda (DIGITAL OCEAN, 2014). Após esse processo, o cliente decripta a mensagem com a chave privada e confirma a autenticação (SSH Communications Security, Inc, [S.d]).

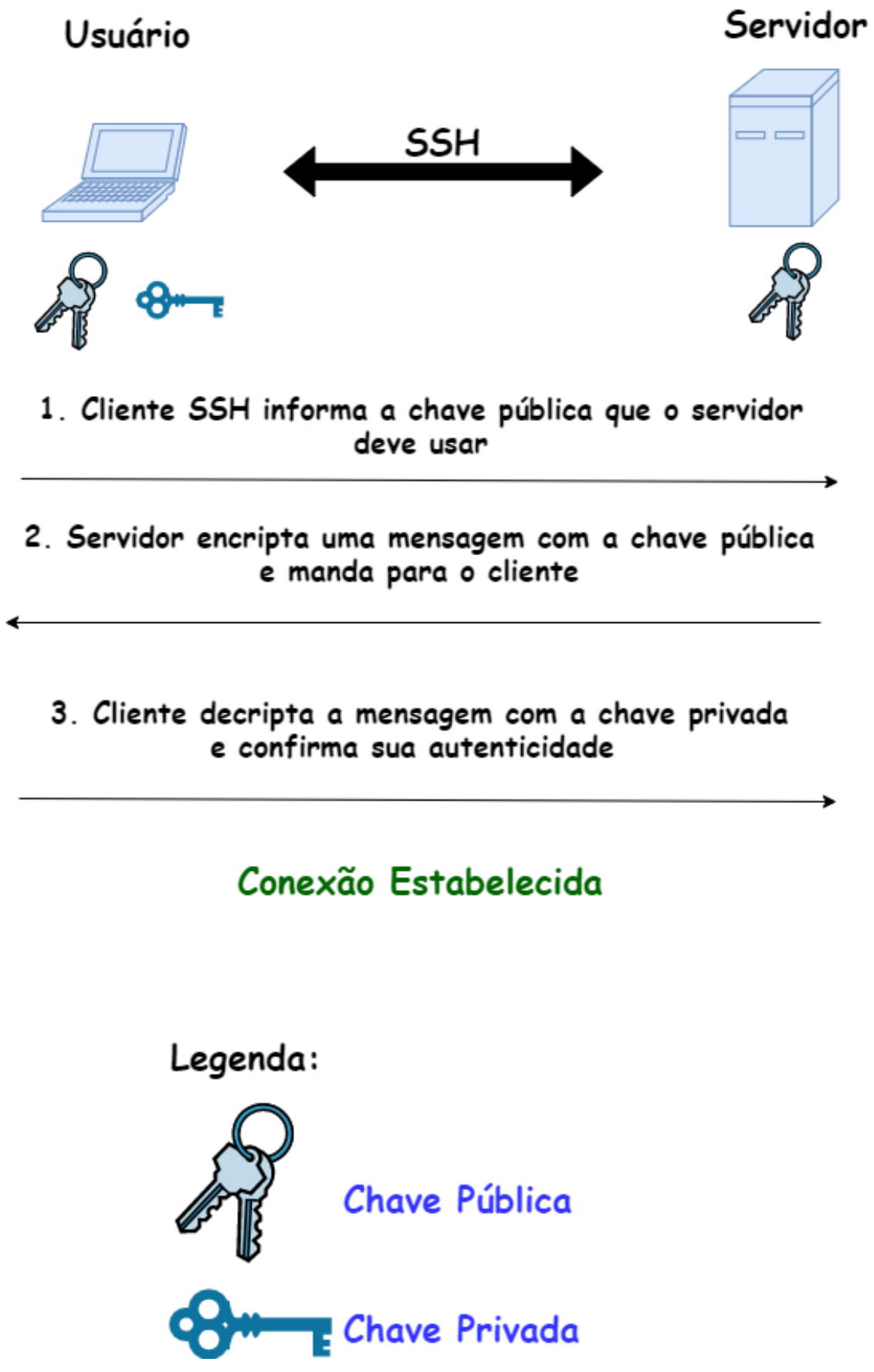
3.10 Certificados e HTTPS

Segundo (COMER, 2016), é possível criptografar requisições HTTP, formando o HTTPS, através de chaves públicas e privadas. Com isso, há proteção de dados, como informações de *login* ou outros assuntos sensíveis.

Nesse contexto, ocorre o chamado Problema da Distribuição de Chaves, afinal, ainda conforme o mesmo autor, é impossível os administradores de um servidor distribuírem chaves públicas para todos os potenciais usuários. Desse forma, foram criados os certificados.

Os certificados são responsáveis por gerarem chaves públicas toda vez que houver um requisição ao servidor, dessa forma, todos os usuários terão uma única chave pública gerada durante a comunicação (COMER, 2016). Além disso, os certificados também fornecem informações que identificam o servidor em questão (ARIANE, 2020).

Figura 21 – Conexão SSH com Par de Chaves



Por fim, quando há a presença de certificados na comunicação HTTP, é entendido que há o uso do protocolo TLS (*Transport Layer Security*), de modo que essa associação é representada pelo protocolo HTTPS (ARIANE, 2020).

3.11 Firewalls

De acordo com (BRITO, 2017), um *Firewall* é um programa cujo objetivo é filtrar conexões, tanto as que entram (*input*), as que saem (*output*) e as que atravessam (*forward*). No presente trabalho é usado o *Firewall* que já vem embutido no Linux, o NetFilter, cujo utilitário de configuração é o IPTables (BRITO, 2017). No cotidiano, é corriqueiro chamar o conjunto NetFilter/IPTables somente por IPTables, logo, essa convenção será adotada aqui também.

A Figura 22 mostra o fluxograma do IPTables usado nesse trabalho. Vale salientar que o *Firewall* possui outras configurações mais complexas. Segundo (BRITO, 2017), o funcionamento do IPTables pode ser entendido com base em *chains* e tabelas. As *chains* representam momentos pelos quais a informação passa:

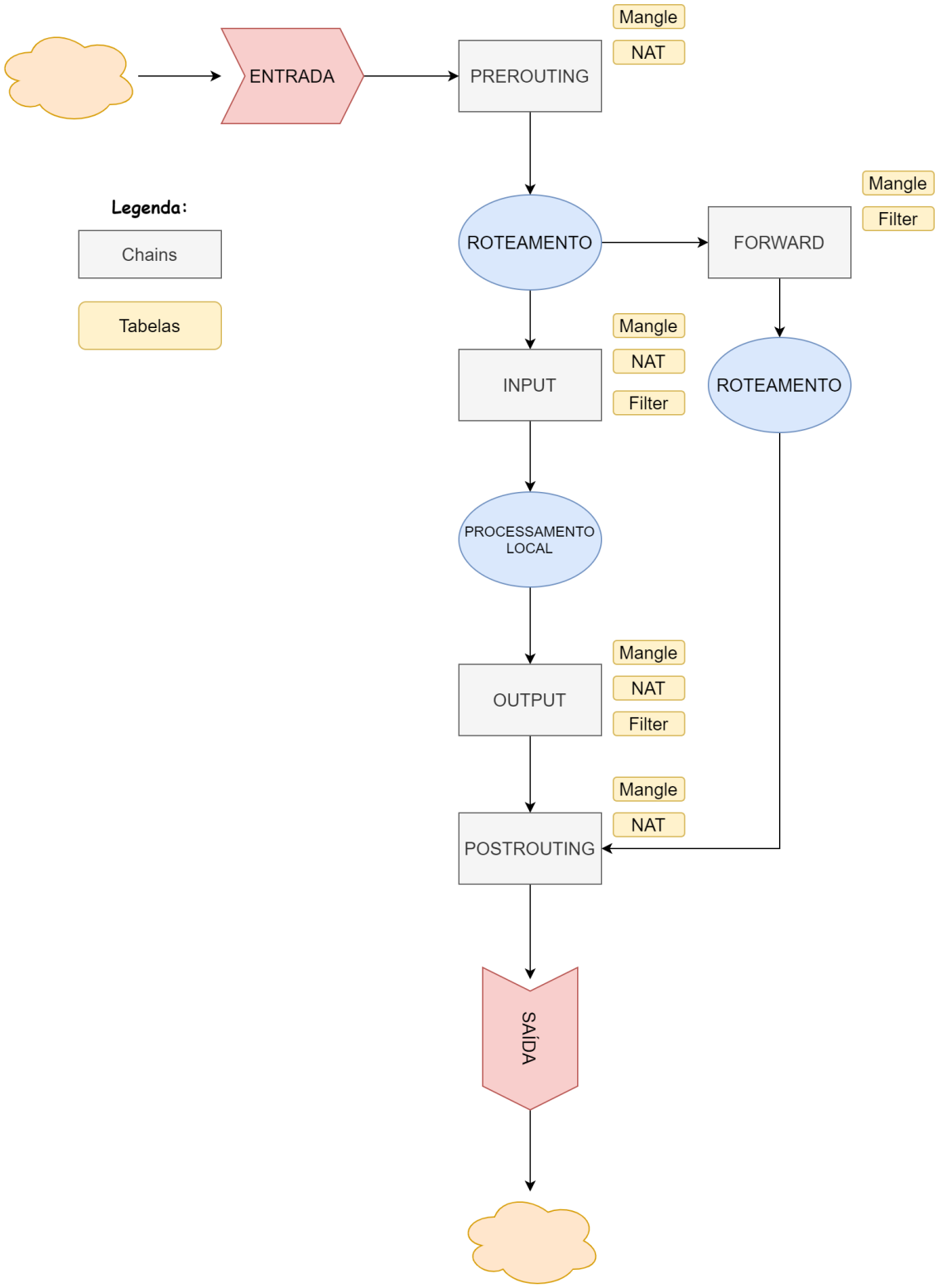
- *Prerouting*: A informação aqui é processada antes dos processos de roteamento;
- *Input*: Aqui a informação é trada no momento em que chega ao *Firewall* e com destino ao *Firewall*;
- *Output*: Nessa etapa, uma informação gerada pelo próprio *Firewall* é processada antes de sair;
- *Forward*: Aqui ocorre o processamento das informações que apenas passam pelo *Firewall*, ou seja, ele não é a origem dos dados e tampouco o destino;
- *Postrouting*: Após estabelecer a rota da informação, o IPTables pode realizar uma última etapa de processamento antes de os dados saírem do *Firewall*.

Conforme dito anteriormente, as *chains* são somente momentos no tempo em que a informação pode ser processada. Quem de fato trata os dados são as tabelas (BRITO, 2017):

- NAT: Como o próprio nome sugere, essa tabela é responsável por traduzir IPs. Além disso, é possível usá-la para fazer redirecionamentos;
- *Filter*: Tabela para filtrar informações. É aqui onde o *Firewall* bloqueia conexões, por exemplo;
- *Mangle*: Aqui é possível manipular outros parâmetros presentes na informação.

A Tabela 5 mostra exemplos de como estão dispostas algumas das principais configurações do IPTables.

Figura 22 – Fluxograma do IPTables



Fonte: Adaptado de https://user-images.githubusercontent.com/35385632/41823115-20c71772-77c8-11e8-823f-937b1b30b72c.png

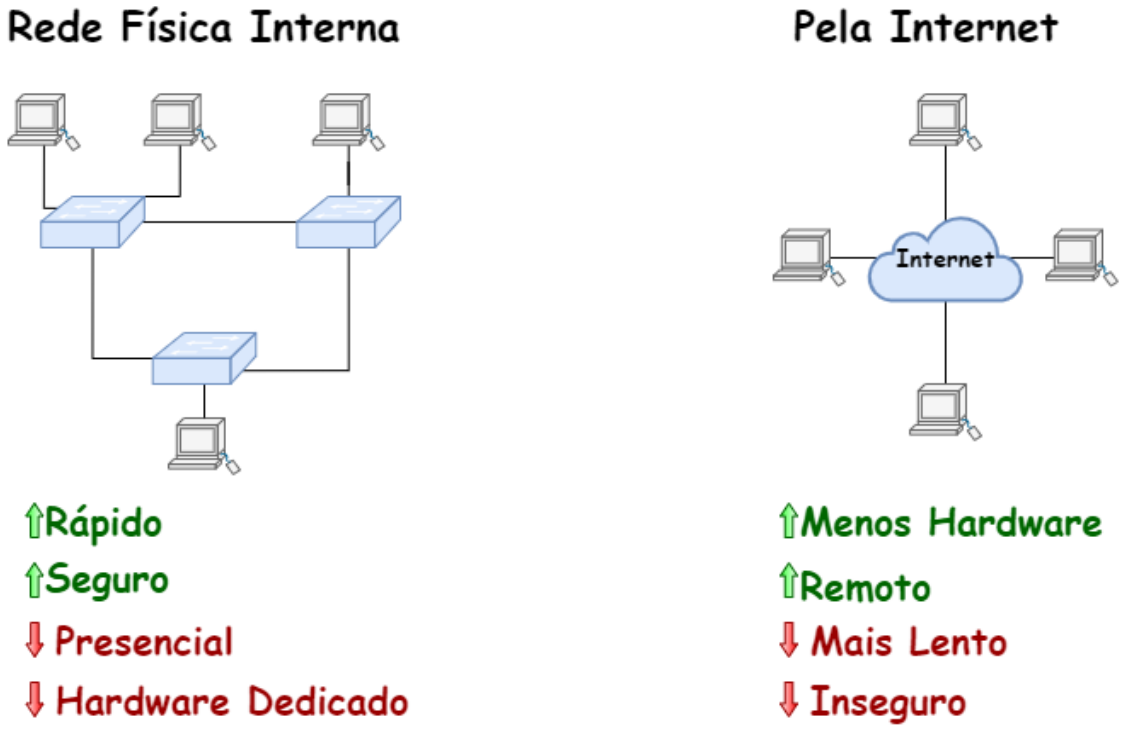
Tabela 5 – Exemplos de Alocação de Chains e Tabelas no IPTables

Função	*Chain*	Tabela	Explicação
Redirecionar para a rede interna	Prerouting	NAT	A conexão é destinada ao Firewall e ele redireciona
Tradução para IP público	Postrouting	NAT	O IPTables traduz o IP de origem da requisição
Impedir ping da rede externa	Input	Filter	Bloqueio de conexões com base no protocolo
Liberar internet para a rede interna	Forward	Filter	Permite a passagem

3.12 VPNs

Em conformidade com o exposto por (COMER, 2016), as redes privadas virtuais (VPNs - *Virtual Private Networks*) permitem o transporte de informações confidenciais através da internet. Para entender a importância das VPNs, considere, primeiro, a situação abordada na Figura 23.

Figura 23 – Comparativo de Comunicação sem VPN



Fonte: Adaptado de Comer, 2016.

A Figura 23 mostra as duas formas pelas quais é possível transferir informações entre computadores quando não há uma VPN. A primeira delas é usando uma rede interna, sendo essa

abordagem mais rápida e segura (COMER, 2016). Todavia, esse método requer a presença do usuário nas instalações da rede e necessita de hardware de rede dedicado para isso.

A outra possibilidade é através da internet. conseguindo, portanto, acesso remoto e não havendo a necessidade de hardware dedicado. Em contrapartida, no entanto, é uma abordagem mais lenta e insegura, pois os intermediários da comunicação podem interceptar os pacotes e analisar (COMER, 2016).

Como meio termo dessas duas abordagens, surgiu as VPNs, as quais conseguem criptografar as informações para que as mesmas sejam enviadas para os receptores através da internet de forma segura. Em conformidade com (BRITO, 2017), as VPNs são comumente usadas para trabalho remoto, pois, em termos práticos, é como se os computadores estivessem na mesma rede física.

3.12.1 VPN Client-to-Site

É necessário dizer que o método de conexão pode ser de dois tipos (BRITO, 2017): *Client-to-Site* e *Site-to-Site*. Nesse trabalho é usada a primeira abordagem, a qual consite em um servidor VPN no qual os usuário se conectam a ele usando credenciais (BRITO, 2017). Quando a conexão é estabelecida, esse servidor se torna o novo *gateway* do cliente, onde tudo que for enviado para ele é criptografado em conformidade com o detalhado a seguir.

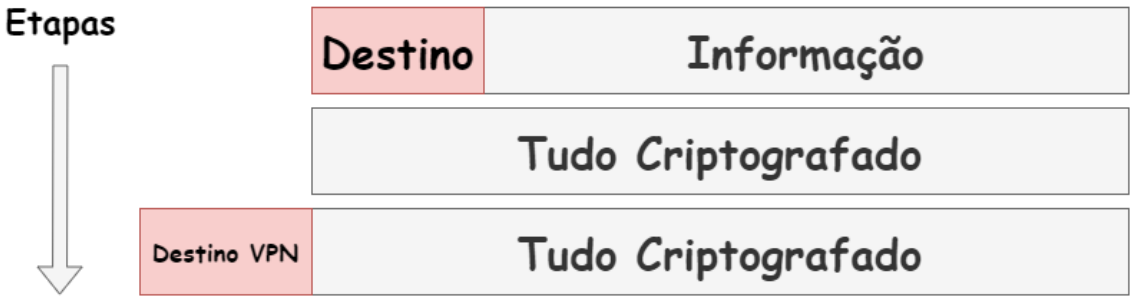
Nesse trabalho é usado o StrongSwan, uma ferramenta *Open Source* para servir VPNs. Em conformidade com (DIGITAL OCEAN, 2019), para tal, é necessário ter um certificado para a VPN, uma chave privada para a encriptação dos dados e uma faixa de IPs privados para ser atribuída aos clientes que se conectarem. Como o StrongSwan possui ferramentas que facilitam essas etapas, o detalhamento não será feito aqui.

3.12.2 Tunelamento IP-em-IP

Mencionados os tópicos acima, é importante entender como a VPN consegue criptografar a conexão. É alertado por (BRITO, 2017) que essa camada de segurança pode ser adicionada tanto em nível de aplicação quanto em nível de rede. Como esse trabalho usa a implementação em nível de rede, é essa modalidade que será abordada nessa seção.

Para tal feito é usado o tunelamento IP-em-IP, onde toda a informação (incluindo os endereços de remetente e destinatário) é criptografada com a chave gerada na configuração. Depois, o endereço público do servidor da VPN é adicionado e tudo é enviado para ele (COMER, 2016). Chegando lá, a informação é descriptografada e encaminhada para o destinatário verdadeiro. A Figura 24 exibe esse processo.

Figura 24 – Etapas do Processo de Tunelamento IP-em-IP



Fonte: Adaptado de Comer, 2016.

4 CIDADES INTELIGENTES

Com o crescimento exponencial da população nas cidades, se tornou gritante a necessidade de empregar soluções tecnológicas que objetivam o uso eficiente de recursos escassos (FOUNOUN; HAYAR, 2018). Ainda em conformidade com os autores, é nesse contexto que a necessidade de cidades inteligentes se torna acachapante.

Dito isso, é definido o conceito de inteligência no âmbito das cidades como sendo a capacidade de usar recursos escassos para maximizar a satisfação dos habitantes (FOUNOUN; HAYAR, 2018). Sendo assim, segundo argumentado por (MORA et al., 2018), cidades inteligentes são aquelas baseadas em serviços de sensoriamento e automação de modo a melhorar a qualidade de vida dos cidadãos e reduzir custos operacionais. É ressaltado ainda que um dos alicerces desses serviços é a internet e, portanto, a Internet das Coisas (discutido nos próximos tópicos) possui um papel crucial.

Com intuito de abordar de forma mais precisa esse conceito, (FOUNOUN; HAYAR, 2018) esmiúçam os seguintes assuntos que integram uma cidade inteligente, chamados de seis inteligências:

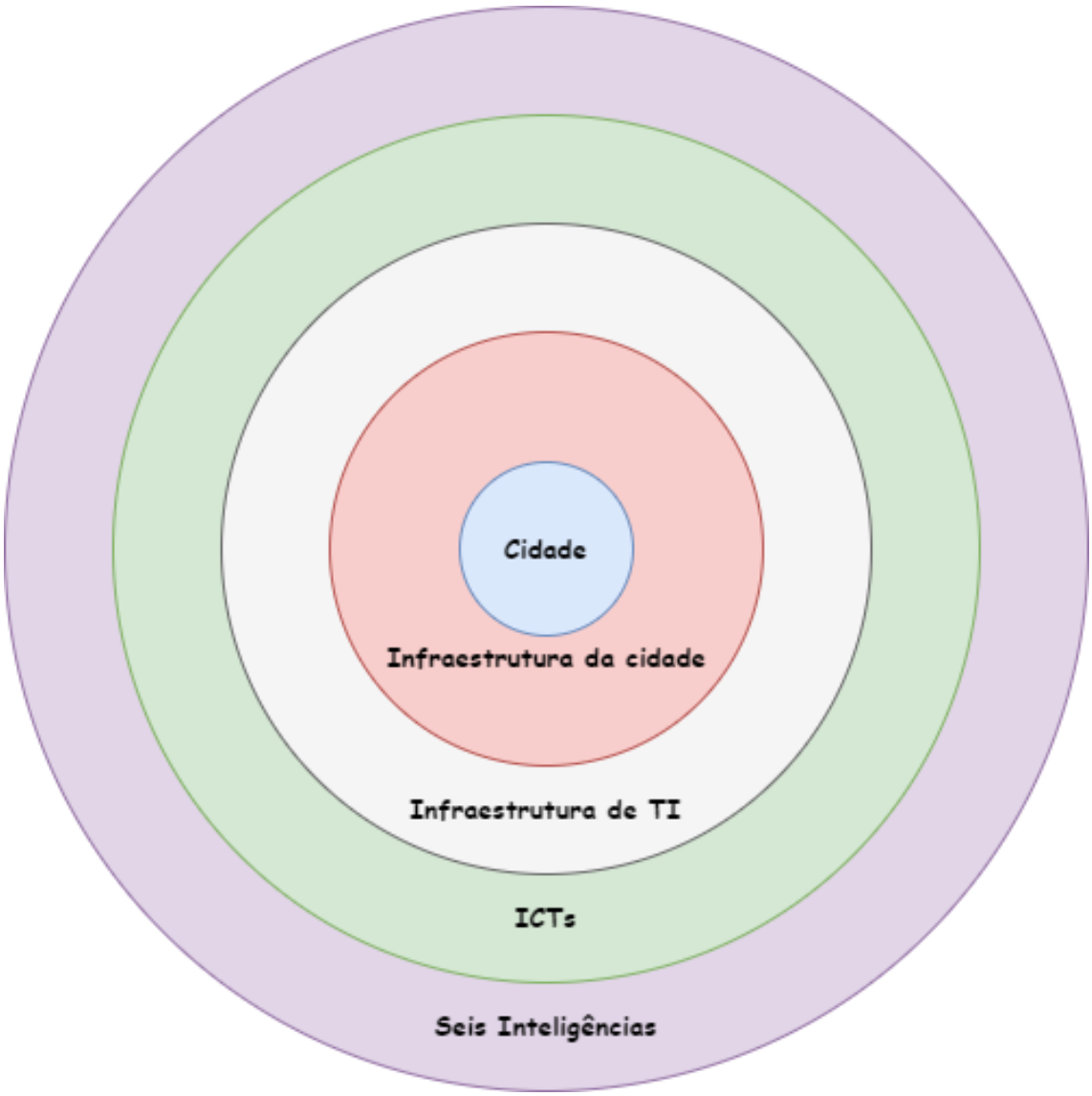
- Vivência Inteligente: Esse conceito engloba qualidade na saúde,segurança individual, facilitações educacionais e culturais e acesso a casas de boa qualidade;
- Economia Inteligente: Tópico ligado à liberdade de empreender e incentivo ao espírito de produtividade. Além disso, aqui é ressaltado a importância de negócios cujo alicerce é a internet e é aconselhado a promoção de produtos locais;
- Pessoas Inteligentes: Esse viés da cidade inteligente está ligado à cultura de aprendizado constante por parte dos cidadãos. Além disso, enaltece a importância de respeitar a diversidade;
- Governança Inteligente: Ponto ligado à necessidade de transparência governamental e a inserção de mecanismos para que a população participe das decisões tomadas pela classe política;
- Mobilidade Inteligente: Sob esse ótica, o foco é garantir transportes públicos seguros e sustentáveis e incentivar políticas de acessibilidade;
- Ambiente Inteligente: Essa pauta está ligada à proteção ambiental e controle da poluição. Em adição, ela abre a discussão sobre aproveitar de forma sustentável os recursos naturais.

Sobre os tópicos acima, (ANTHOPOULOS, 2017) faz uma consideração: eles são sustentados por uma camada chamada de Comunicação Inteligente, a qual compõe os serviços de comunicação que viabilizam a troca de dados entre os demais sistemas inteligentes da cidade. O

autor chama esses serviços de ICTs (*Information and Communicaton Technologies*, Tecnologias de Informação e Comunicação).

A Figura 25 sumariza os conceitos vistos até aqui. Para conceber uma cidade inteligente, é preciso, primeiro, haver a infraestrutura da cidade (água, energia, construções, etc.) (FOUNOUN; HAYAR, 2018). Em sequência, vem a implementação da infraestrutura computacional, ou infraestrutura de TI, a qual é responsável pelos equipamentos usados pelas ICTs (ANTHOPOULOS, 2017). Em seguida, há a criação das próprias ICTs para posteriormente virem as seis inteligências.

Figura 25 – Hierarquia de uma Cidade Inteligente



Fonte: Adaptado de Anthopoulos, 2017.

Adentrando ainda mais nas especificações das cidades inteligentes, (ANTHOPOULOS, 2017) apresenta algumas modalidades de serviços englobados pelas seis inteligências. Essas categorias são chamadas de SGs (*Service Groups*, Grupos de Serviço) e estão listadas a seguir:

- SG 1 - Serviços de E-Governo: Emprego de ferramentas de administração e transações digitais;
- SG 2 - Serviços de E-Democracia: Possibilidade de votos, plebiscitos e enquetes em meios virtuais;
- SG 3 - Serviços de E-Comércio: Ferramentas para sustentar negócios digitais;
- SG 4 - Serviços de E-Saúde: Disponibilidade de ferramentas digitais para cuidados básicos de saúde, incluindo consultas remotas;
- SG 5 - Serviços de E-Segurança: Aprimoramento da segurança pública através de tecnologias digitais;
- SG 6 - Serviços Ambientais: Sistemas voltados para proteção ambiental e controle de poluição;
- SG 7 - Tráfego Inteligente: Automação do controle de tráfego e transporte público otimizado;
- SG 8 - Serviços de Telecomunicações: Disponibilização de serviços de telecomunicações para a sociedade;
- SG 9 - Serviços de E-Educação: Incentivo e viabilização da educação a distância.

4.1 Arquitetura de uma Cidade Inteligente

Até o presente momento as classificações das cidades inteligentes foram conceituadas de forma abstrata. Após as devidas definições, (ANTHOPOULOS, 2017) continua seu trabalho montando uma arquitetura genérica para uma cidade inteligente, relacionando todas as definições discutidas até aqui com os entes viventes na localidade em questão. A Figura 26 mostra essa relação.

De acordo com a Figura 26, a base é composta pelos recursos naturais, os quais, conforme (HAYAR; BETIS, 2017), devem ser usados de forma sustentável, ou seja, de modo a evitar sua escassez no futuro. Os outros dois degraus superiores correspondem ao que foi definido anteriormente, ou seja, a infraestrutura física básica são serviços como eletricidade e a infraestrutura de TI é o que possibilita a implantação de ICTs.

As novas definições começam a partir dos Serviços Inteligentes, pois aqui é onde são concretizados os SGs, provenientes das seis inteligências. Segundo (ANTHOPOULOS, 2017), é nesse degrau que se encontra serviços como *smart grids, smart buildings* e *smart waste treatments*.

Figura 26 – Arquitetura de uma Cidade Inteligente

Fonte: Adaptado de Anthopoulos, 2017.

Por último está o que (ANTHOPOULOS, 2017) chama de *Soft Infrastructure*, que seria uma Infraestrutura Leve em tradução literal. Nesse ponto estão inseridos os trabalhadores que, embora não tenham conhecimento para construir sistemas inteligentes, são capacitados para usarem esses sistemas e aplicarem em seu domínio de atuação. Dito de forma mais direta, A infraestrutura leve são as pessoas cujo cotidiano é facilitado pelos sistemas inteligentes.

4.2 Desafios para Implantar uma Cidade Inteligente

A partir das definições anteriores, um corolário importante extraído por (TOLCHA et al., 2018) é a natureza distribuída inerente aos sistemas de cidades inteligentes. Dito de outro modo, são diversas aplicações descentralizadas que geram dados de diversos tipos. Os autores, então, destacam a necessidade de se estabelecer padrões de comunicação para que uma aplicação possa se comunicar com outra.

Com o mesmo alinhamento, o primeiro desafio apontado por (CEDILLO-ELIAS et al., 2018) é o de reforçar a infraestrutura de tecnologia da informação (TI) para que o governo consiga trabalhar de modo compatível com o de uma cidade inteligente. Isso inclui, segundo os autores, mecanismos que possibilitem a comunicação de vários sistemas distribuídos e multi-arquitetura.

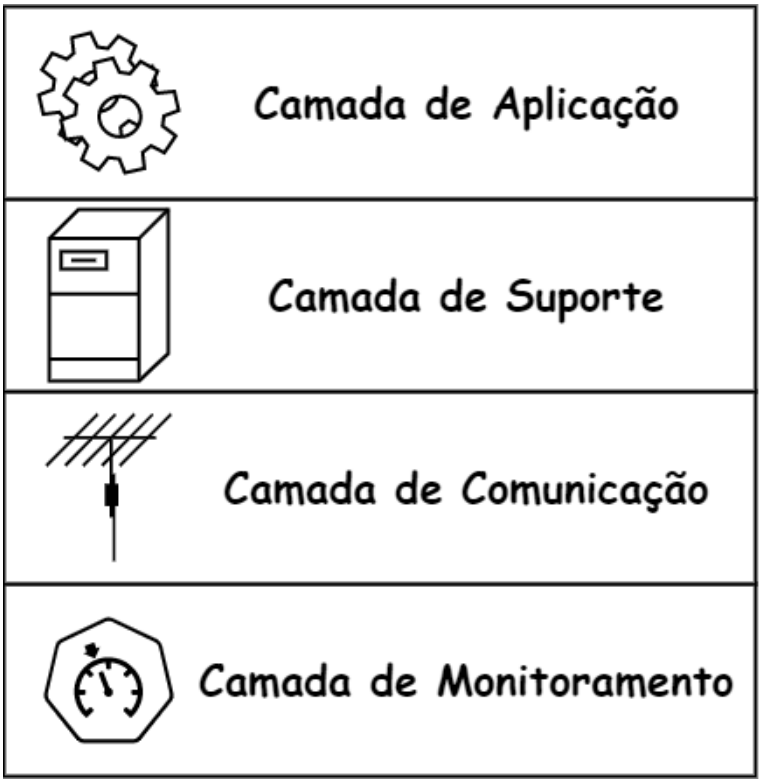
É citado ainda por (CEDILLO-ELIAS et al., 2018) a necessidade de se pensar na segurança, enaltecendo o assunto da privacidade e confidencialidade dos dados, principalmente em situações como o pagamento de impostos via aplicativo. É alertado também, por (LIANG; SHETTY; TOSH, 2018), o potencial de danos causados por dados não validados: uma vez que sistemas distribuídos trabalham com diversas fontes de dados, se um deles não estiver validado pode comprometer os demais gerados a partir dele.

Além disso, outros desafios também estão presentes: falta de investimentos, visão política para implementação e entendimento das tecnologias (KUMAR; GOEL; MALLICK, 2018). Apesar dos obstáculos, (TOLCHA et al., 2018) ressalta a importância da continuidade de pesquisas e a adoção de padrões abertos, ou seja, padrões cuja a implementação é consensual e são permitidas alterações da comunidade científica e de desenvolvedores.

4.3 Internet das Coisas

Alinhado com (SINGH; SINGH, 2015), o conceito de Internet das Coisas (*Internet of Things* ou IoT) consiste na possibilidade de conectar objetos do dia a dia à internet para tornar o cotidiano mais confortável e/ou produtivo. Nesse contexto, conforme exposto por (XING, 2020), é possível dividir um sistema para IoT em quatro camadas: aplicação; suporte; comunicação e monitoramento. Essa divisão pode ser vista na Figura 27.

Figura 27 – Camadas de um Sistema para IoT



Fonte: Adaptado de Xing, 2020

Ainda conforme o mesmo autor, a camada de aplicação é onde ocorre a interação com o mundo humano, podendo ser, portanto, um aplicativo ou um sistema de mensagens, por exemplo. É informado ainda por (BIRON; FOLLETT, 2016) que essa primeira camada consegue se comunicar com a de baixo através de APIs (*Application Programming Interface*, Interface de Programação de Aplicações). Esse assunto é mais bem abordado no Capítulo 5.

Mais abaixo é a camada de suporte, responsável pelos sistemas que hospedam e se comunicam com a aplicação. É na camada de suporte que ficam os servidores. Como já foi explicado anteriormente no Capítulo 3, a troca de mensagens com o nível inferior se dá através de protocolos de internet.

A próxima camada é a de comunicação, sendo composta pela rede que conecta os envolvidos. Por fim, na camada de monitoramento é onde o sistema coleta os dados do ambiente. Vale dizer que a comunicação entre essas duas últimas camadas se dá no nível eletrônico, conforme explicado no Capítulo 2.

No tocante à camada de monitoramento, o campo de IoT está presente nos medidores inteligentes (BETTS, 2016). Ainda em conformidade com o mesmo autor, os medidores inteligentes consistem em sensores ligados a um concentrador, o qual é capaz de se comunicar com uma central de dados através de uma rede interna ou mesmo pela internet.

4.3.1 Desafios

O primeiro desafio apresentado por (AHLGREN; HIDELL; NGAI, 2016) é em virtude da preocupação com a eficiência energética do dispositivo. Uma vez que os equipamentos podem funcionar a partir de baterias ou painéis fotovoltaicos, é importante que o consumo de energia seja o menor possível, incluindo os protocolos de comunicação.

É nesse ponto também que (AHLGREN; HIDELL; NGAI, 2016) vai de encontro com o proposto em (TOLCHA et al., 2018), isto é, a necessidade de padrões abertos para otimizar a eficiência das aplicações. Conforme abordado por ambos os trabalhos, os padrões abertos permitem que diversos sistemas sejam integrados e que possam trocar informações entre si, ampliando as possibilidades de atuação ou mesmo permitindo soluções customizadas para algum contexto mais restritivo.

4.4 Redes de Sensores Sem Fio

De acordo com (YAVARI et al., 2019) e (XING, 2020), a camada de monitoramento pode ser composta tanto por um sensor como por um grupo deles. Em verdade, (NETTO, 2016) evidencia a possibilidade de usar uma rede sem fio composta por sensores para monitorar pequenas áreas geográficas. Essas redes são chamadas de redes de sensores sem fio (RSSF).

Em consonância com (NETTO, 2016) e (AKYILDIZ et al., 2002), o emprego de RSSFs permite o monitoramento preciso em regiões geográficas cujas proporções são pequenas quando comparadas com as resoluções dos satélites usados com as mesmas finalidades. Ainda conforme (NETTO, 2016), a RSSF deve ser pensada de modo a consumir a menor quantidade possível de energia.

É importante abordar também o conceito de saltos em redes. Para tal, define-se, para o escopo desse trabalho, que o grupo de sensores usado nas medições será chamado doravante de célula. Posto isso, o salto em redes diz respeito ao caminho que a informação percorre até chegar ao servidor. Em conformidade com (NETTO, 2016), a comunicação é *single hop* se cada célula enviar a informação diretamente para o servidor. Caso a informação passe por outras células até chegar ao servidor, o enlace é chamado de *multihop*. O presente trabalho usa *single hop* de modo a aumentar a resiliência da rede isolando cada célula.

4.4.1 Resiliência em Redes

De acordo com (ALBERT; BARABáSI, 2002), o conceito de resiliência em redes consiste no quão resistente à falhas é a estrutura de comunicação. Em outras palavras, quanto mais resiliente for a rede, menos uma única falha impactará no funcionamento geral do sistema. Os autores (MORA et al., 2018) ainda argumentam que, no caso de cidades inteligentes, os serviços tendem a ser tão importantes que resiliência se torna uma política de segurança.

Em consonância com os autores citados, um dos fatores que aumenta a resiliência é a redundância, a qual consiste em caminhos alternativos para a informação percorrer. Outra forma de aumentar essa métrica é usando comunicação *single hop*, pois assim, caso uma célula falhe, as demais continuarão a funcionar normalmente, pois todas se conectam diretamente com o servidor (NETTO, 2016).

4.4.2 Modelo Aberto para Monitoramento Ambiental

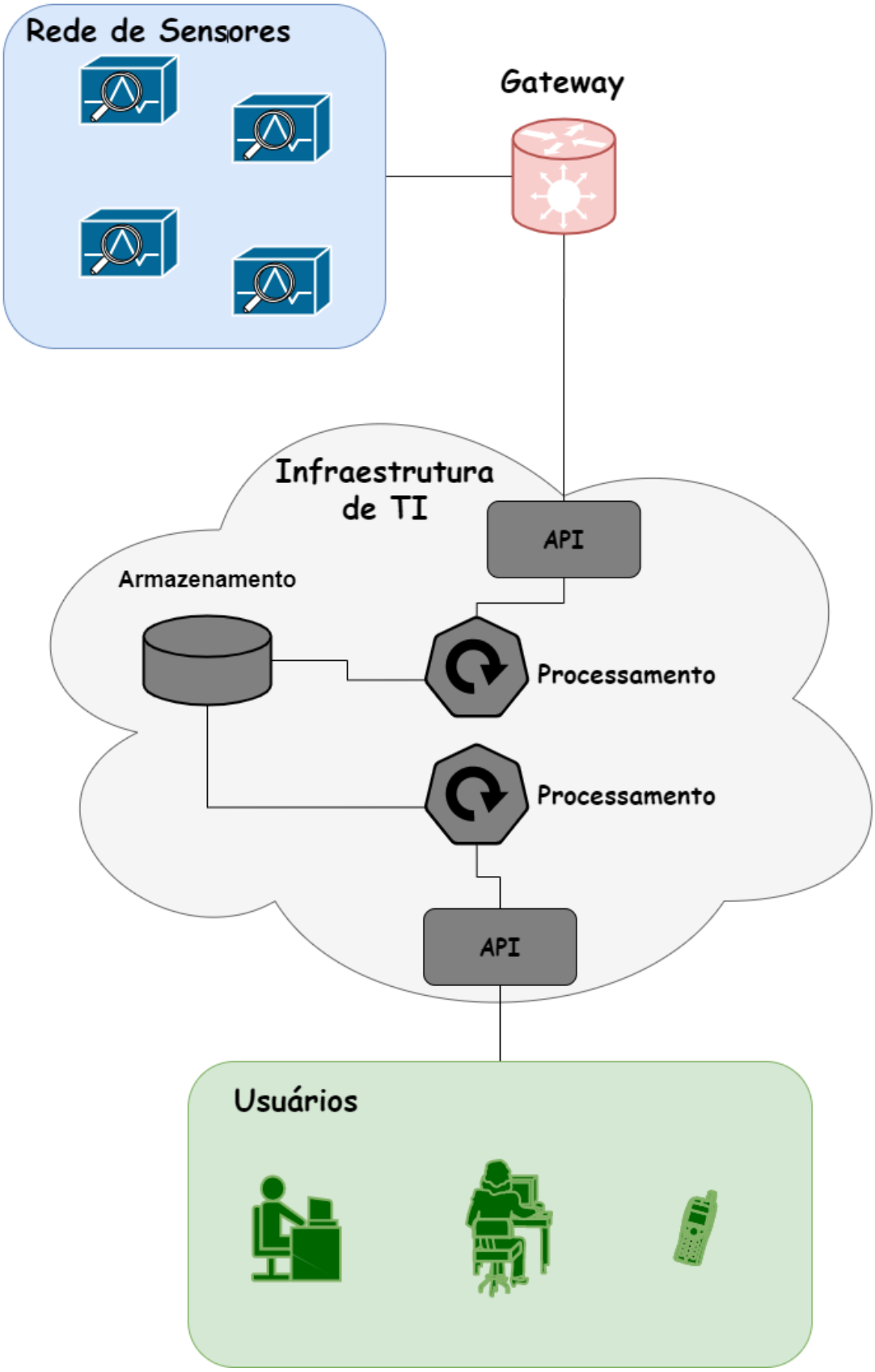
No trabalho de (AHLGREN; HIDELL; NGAI, 2016) é apresentado o projeto GreenIoT, o qual consiste em uma padronização nos sistemas de monitoramento. Os participantes testaram uma implementação que segue esse modelo na cidade de Uppsala. A Figura 28 ilustra o padrão adotado no presente trabalho, o qual é baseado no exibido por (AHLGREN; HIDELL; NGAI, 2016).

Nela, é possível ver o início do monitoramento através da rede de sensores. Para esse modelo, tanto faz ser com ou sem fio, embora o presente trabalho implemente a configuração *wireless*. Naturalmente, a comunicação entre a infraestrutura de TI e a rede de sensores se dá através de um *gateway* (ou mesmo vários).

Dentro da infraestrutura de TI, os dados enviados do monitoramento são recebidos por uma API, a qual se comunica através do protocolo HTTP (mais detalhes no capítulo 5). Logo adiante há uma etapa de processamento desses dados. Aqui, vale salientar que a própria API pode realizar essa função.

Dando continuidade ao processo, o estágio do armazenamento compreende toda a persistência de dados, ou seja, bancos de dados e/ou armazenamento de arquivos. Na outra ponta

Figura 28 – Modelo de Sistema para Monitoramento Ambiental



Fonte: Adaptado de Ahlgren; Hidell; Ngai, 2016.

há uma outra API, responsável por interagir com o usuário. Essa API acessa o armazenamento e pode processar novamente os dados antes de mandar para os destinatários.

Para finalizar, a própria natureza distribuída dos sistemas de IoT permite diversas interfaces para interagir com as pessoas. O ponto a ser destacado aqui é que tudo é feito a partir de uma API dedicada aos usuários, ou seja, em termos de aplicação, as células permanecem isoladas daqueles que precisam acessar seus dados.

Naturalmente, conforme sugerido por (TOLCHA et al., 2018) e (AHLGREN; HIDELL; NGAI, 2016), esse modelo não é rígido e tampouco é fechado. Dito isso, é importante elucidar que outros sistemas podem, por exemplo, usar a API que originalmente seria usada pelos usuários e assim inserirem uma nova etapa de automação ou tratamento de dados. Essa flexibilidade não só permite a adaptação do modelo para diversas circunstâncias como possibilita o crescimento de toda a rede de informações a partir da inserção de novos equipamentos configurados para consumir essas APIs.

4.5 Projeto Cidades Digitais

No Brasil, a implementação de cidades inteligentes se dá por meio do projeto Cidades Digitais. A implantação e manutenção das Cidades Digitais do Governo Federal, através do Ministério das Comunicações, foi instituído pela Portaria nº 376, de 19 de agosto de 2011, com o objetivo de: 1) constituir redes locais de comunicação nos municípios brasileiros; 2) promover a produção e oferta de conteúdos e serviços digitais; e 3) facilitar a apropriação de tecnologias da informação e da comunicação pela gestão pública local e pela população, de maneira coordenada e integrada entre esferas dos poderes públicos e da sociedade (DIÁRIO OFICIAL DA UNIÃO, 2011) e (DIÁRIO OFICIAL DA UNIÃO, 2012).

As cidades que recebem recurso para implantação da estrutura oferecida pelo projeto são selecionadas por meio de edital. Em 2012, o Ministério das Comunicações abriu a primeira seleção para o projeto-piloto e 80 municípios foram contemplados. Em 2013, o projeto Cidades Digitais foi incluído no Programa de Aceleração do Crescimento (PAC) do Governo Federal, oportunidade que beneficiou naquele ano, 262 municípios com população de até 50 mil habitantes (MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÃO E COMUNICAÇÕES, [S.d]).

4.5.1 Caso de Estudo: Porto Nacional - Tocantins

A cidade de Porto Nacional, Tocantins, aderiu ao projeto em 15 de dezembro de 2017 com um investimento de R$ 2.142.918,78 do Ministério das Comunicações (PREFEITURA DE PORTO NACIONAL, 2017). Segundo (MINISTÉRIO DO PLANEJAMENTO, [S.d]), Porto Nacional é uma das quatro cidades tocantinenses que aderiram ao projeto.

Em linhas gerais, o projeto contempla a cobertura da cidade de Porto Nacional com uma rede de fibra óptica e pontos de WiFi para acesso público, doravante PAP, em suas praças. Atualmente as fibras estão instaladas conforme Figura 29, havendo três PAPs.

Figura 29 – Rede Fibra Óptica em Porto Nacional

Fonte: Fornecido pela Prefeitura de Porto Nacional

De acordo com o que é visto na imagem, a rota laranja é chamada de anel e corresponde à rede principal de fibra óptica. É através desse anel que há redundância no circuito. Em verde estão os *drops*, que são ramificações para conectar pontos na rede (não estão sendo exibidos para não poluir a imagem). Por último, os três PAPs da cidade estão marcados com estrela e estão distribuídos conforme Tabela 6. Neste trabalho, a conexão da rede de sensores sem fio com a internet se dará através dos PAPs.

Tabela 6 – PAPs de Porto Nacional

Identificação	Localização
PAP 01	Praça da Saúde
PAP 02	Praça Dr. Euvaldo
PAP 03	Praça da Avenida Beira Rio

Fonte: Dados fornecidos pela Prefeitura de Porto Nacional

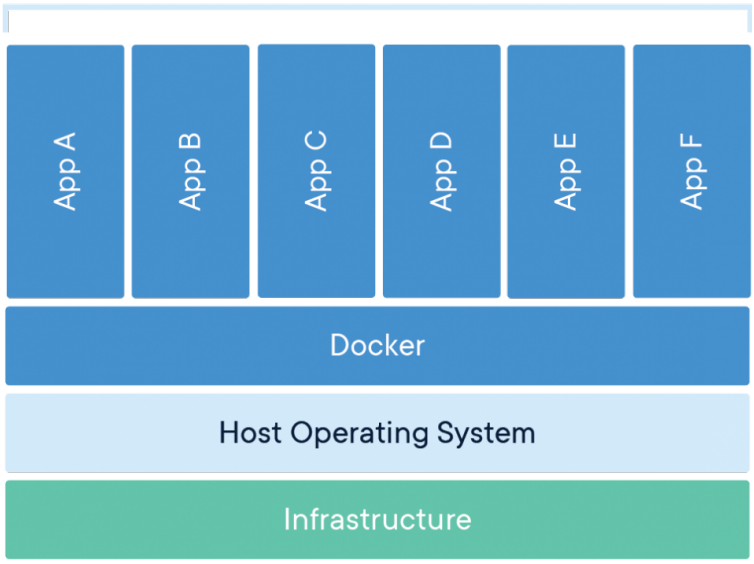
5 ARQUITETURA DE SOFTWARE DISTRIBUÍDO

5.1 Docker

Segundo a própria empresa (DOCKER, [S.d]), os contêineres Docker são uma tecnologia de virtualização, logo, os mesmos são empregados para isolar virtualmente aplicações em um mesmo *host*.

A Figura 30 mostra o conceito por trás do Docker. Observe que as aplicações são virtualmente isoladas, de modo que a Docker Engine consegue usar o sistema operacional do *host*. Em outras palavras, não é necessário instalar um sistema operacional para cada aplicação, como aconteceria caso houvesse o uso de máquinas virtuais.

Figura 30 – Funcionamento do Docker



Fonte: https://www.docker.com/resources/what-container

Através dessa simplificação, Docker consegue ser leve e funciona em qualquer sistema operacional que tenha a Docker Engine instalada (DOCKER, [S.d]). Por fim, é importante avisar que Docker não substitui totalmente as máquinas virtuais, tendo estas ainda suas aplicações e vantagens específicas. Antes de seguir, vale dizer que as próximas informações, salvo quando dito ao contrário, podem ser encontradas na documentação da própria ferramenta (DOCKER, [S.d]).

5.1.1 Redes em Docker

As redes em Docker conseguem abstrair a comunicação dos contêineres. Em outras palavras, contêineres podem interagir com outros contêineres ou mesmo com máquinas físicas.

Vale dizer ainda que a comunicação é possível mesmo entre *hosts* distintos. A arquiteturas dessas redes é baseada em *drivers*, logo, é customizável. Dentre os principais, estão:

- *Bridge* : As redes com *bridge driver* geralmente são usadas em aplicações completas e isoladas (chamadas de *standalone*) que precisam se comunicar com outros contêineres para aprimorar suas funcionalidades. Exemplo: um *site* que precisa se comunicar com um bando de dados;
- *Host*: Nessa configuração o contêiner compartilha a mesma rede do *host*, dessa forma, o mesmo consegue se comunicar inclusive com as outras máquinas da rede;
- *Overlay*: Esse *driver* é comumente usado em *clusters*, pois assim permite a comunicação entre contêineres localizados em diferentes *hosts*;
- *Macvlan*: É usado em situações onde cada contêiner precisa ter um endereço MAC. Esse *driver* consegue suprir essa demanda

Como no presente trabalho é usado o *bridge driver*, o mesmo é detalhado a seguir.

5.1.1.1 Bridge Driver

Conforme já foi explicado, essa configuração facilita a troca de informação entre contêineres. Dito isso, é preciso aprofundar em dois assuntos: *bridges* definidos pelo usuário e DNS interno.

O primeiro ponto informa que o administrador pode criar várias redes distintas usando *bridge driver*, porém, elas estarão isoladas entre si, ou seja, contêineres de duas redes distintas não se comunicam. No tocante ao DNS interno, é possível realizar a conexão entre contêineres apenas através do nome atribuído, sendo desnecessário o conhecimento dos IPs dentro da rede interna do Docker.

Para finalizar, é importante ainda mencionar dois detalhes: o DNS interno também obedece ao isolamento do *bridge driver* definido pelo usuário e, além disso, a rede *bridge* padrão do Docker não contém DNS interno.

5.1.2 Volumes e Docker Compose

Os contêineres não persistem dados, ou seja, tudo que acontece dentro dele é perdido quando o mesmo é destruído (em linguagem técnica, os contêineres são *stateless*). Todavia, a persistência de dados é necessária, seja para arquivos de *log*, procedimentos de *backup* ou carregar código-fonte. Para atender tal demanda, existem os volumes.

Com base no exposto anteriormente, define-se volume como sendo uma ligação entre um diretório no *host* e um diretório dentro do contêiner. Tudo que é feito dentro do contêiner nessa localização é refletida no *host*, de modo que os dados ali contidos não fazem parte do ciclo de vida dos contêineres.

Por fim, acerca do Docker Compose, ele é uma ferramenta para orquestração de vários contêineres em um mesmo *host*. Através do arquivo *docker-compose.yml*, o administrador consegue configurar todo o procedimento de criação de contêineres para subir determinada aplicação. Uma vez que o presente trabalho usa abordagem de separar cada funcionalidade do sistema em um único contêiner, essa ferramenta é utilizada.

5.2 Integração Contínua e Entrega Contínua

Em conformidade com o abordado por (RED HAT INC, [S.d]), os processos de integração e entrega contínuas tem como objetivo aumentar o produtividade da equipe de desenvolvimento de *software* a partir da automação de diversas etapas do desenvolvimento da aplicação. O conjunto desses processos também é chamado de *Pipeline CI/CD*, onde CI (*Continuous Integration*) corresponde à integração contínua e CD (*Continuous Deployment*) é a entrega contínua.

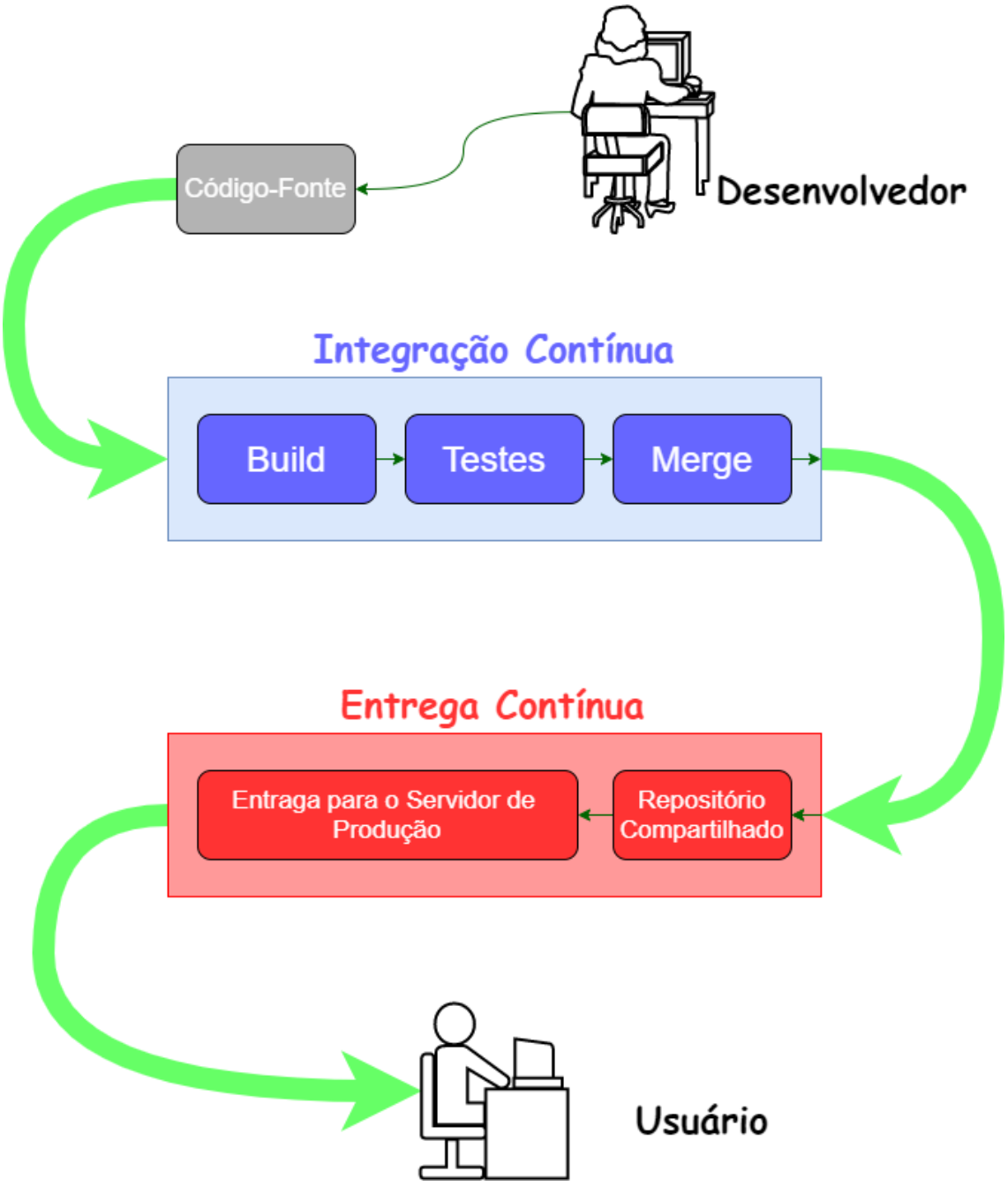
É importante discutir a diferença entre integração contínua e entrega contínua. Segundo (RED HAT INC, [S.d]), integração contínua diz respeito ao procedimento de inserção de atualizações no código original. Em outras palavras, a integração contínua é responsável por entregar cada atualização que um desenvolvedor faz aos outros da equipe de forma automática. A entrega contínua, por sua vez, é o processo automático de disponibilização das atualizações para o usuário final. A Figura 31 detalha esses conceitos.

De acordo com a Figura 31, o processo se inicia com o desenvolvedor escrevendo o código-fonte. É a partir dessa etapa que começa a automação do processo, iniciando com o passo chamado *build*, o qual, em uma tradução livre, significa construção. Segundo (NOGUEIRA et al., 2018), nesse estágio o código-fonte é compilado (quando se aplica) e as dependências do projeto são resolvidas.

A etapa seguinte é a de testagem, onde são executados os *scripts* de testes com o intuito de confirmar que a aplicação não está defeituosa (GALLABA, 2019). Na linguagem da área, uma aplicação defeituosa é chamada de "aplicação quebrada". Ainda segundo o mesmo trabalho, a próxima etapa da integração contínua é chamada de *Merge* (mistura, em tradução livre), a qual consiste em juntar todas as alterações dos outros desenvolvedores após passados os estágios anteriores.

Entrando no setor da entrega contínua, o repositório compartilhado é onde a equipe tem acesso ao código com todas as atualizações. Geralmente é nessa etapa que os testes de usabilidade são realizados (GALLABA, 2019). A última etapa é a entrega de fato para o usuário

Figura 31 – *Pipeline* CI/CD



Fonte: Adaptado de Red Hat, Inc

final, chamada também de *deployment* (RED HAT INC, [S.d]).

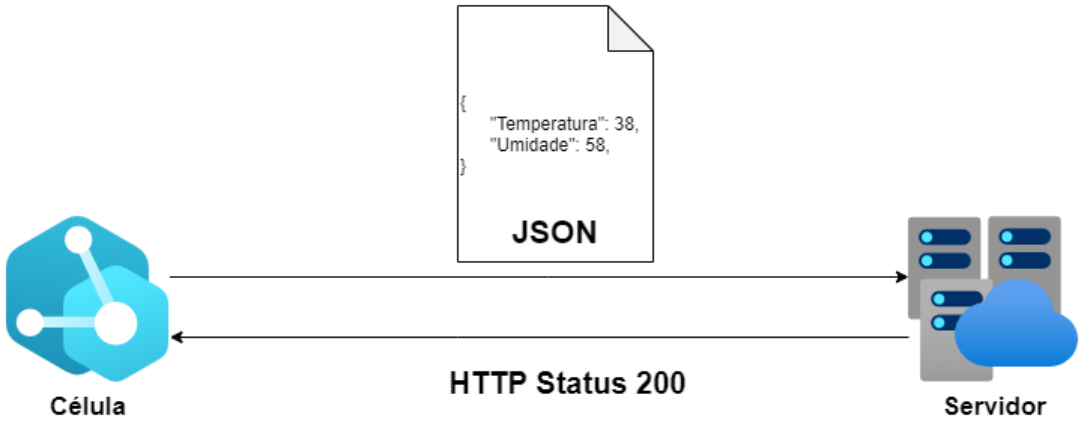
Falta dizer que a *pipeline CI/CD* não é rígida, de modo que cada equipe tem a sua. Além disso, as ferramentas existentes no mercado suportam essa devida personalização. Dentre as existentes, o presente trabalho usa o Gitlab CI, onde a *pipeline CI/CD* é especificada através do arquivo *.gitlab-ci.yml*, que deve ficar no diretório raiz do repositório. A configuração da *pipeline* será abordada nas seções sobre a construção dos servidores.

5.3 Padrão RESTful

REST (*Representational State Transfer*) é um modelo de arquitetura focado em estabelecer restrições ao acesso de recursos (RESCA, 2019). Ainda conforme o mesmo autor, os recursos são abstrações de algum tipo de informação. Vale salientar ainda outro diferencial do modelo REST: não há sessões para guardar dados entre requisições, ou seja, o servidor recebe um pedido, atende e finaliza a conexão (RESCA, 2019).

Para melhor entender o conceito de REST, considere o seguinte exemplo: uma célula pretende enviar os dados dos sensores para o servidor. Em primeiro lugar, existe no servidor um recurso capaz de lidar com os dados oriundos dos sensores. Esse recurso tem restrições acerca de como esses dados devem ser enviados e só os aceita se estiverem em conformidade com as restrições. Obedecidas as exigências, a célula enviará os dados para o recurso (localizando-o através de uma URI), o servidor voltará uma resposta e encerrará a conexão. Caso a célula deseje usar algum outro recurso (ou o mesmo), deverá solicitar outra conexão ao servidor. A Figura 32 ilustra um exemplo.

Figura 32 – Exemplo de padrão REST.



Na Figura 32 é possível ver que a comunicação se dá em texto puro. A célula envia um texto que obedece à critérios determinados pelo servidor, esse emite uma resposta e encerra a conexão. Observe que o texto enviado no exemplo segue o padrão JSON, o qual é usado para

agrupar informações em chaves e seus respectivos valores. Por fim, vale dizer que o servidor pode retornar uma resposta em JSON além de enviar o código de *status* HTTP. Após isso, a conexão é encerrada e o servidor não guarda nenhuma informação da requisição.

O padrão REST permite aplicações distribuídas, de modo que diversos dispositivos podem se comunicar por diversas formas e interfaces, bastando apenas que obedeçam as regras de envio (RESCA, 2019).

Um sistema cuja arquitetura obedece ao padrão REST é chamado de RESTful (RESCA, 2019). Portanto, o servidor implementa uma arquitetura RESTful para receber os dados. Ainda conforme o mesmo autor, quando um conjunto de recursos é criado para trabalhar em um contexto específico, ele é chamado de API (*Application Programming Interface*). Portanto, o servidor contém uma API para se comunicar com a célula.

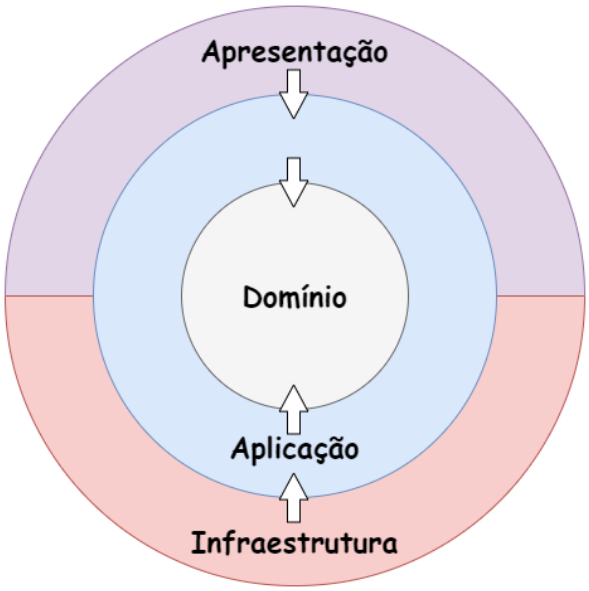
5.4 Arquitetura Limpa

Uma vez que os sistemas passam por mudanças frequentes, é de extrema necessidade que sejam implementados já levando em conta essa volatilidade (MARTIN, 2020). Apesar de toda a descrição de infraestrutura e lógica que é apresentada nesse trabalho, o sistema aqui detalhado é flexível a ponto de mudar drasticamente no futuro, caso necessário. Para tal feito, foram usados os conceitos de arquitetura limpa e código limpo, definidos por Robert C. Martin em suas obras homônimas (MARTIN, 2020) e (MARTIN, 2009).

A Figura 33 mostra o diagrama de uma arquitetura limpa genérica, ou seja, a organização de um sistema qualquer que segue os conceitos que serão descritos a seguir. Conforme é possível observar, a aplicação é dividida em pelo menos quatro camadas:

- Domínio: aqui se encontram as entidades e as regras de mais alto nível do sistema, ou seja, aquelas atividades que dão sentido à existência da aplicação. Por exemplo, é nessa camada que se encontram as classes de monitoramento e regras de emissão de alertas;
- Aplicação: nessa camada são definidas as regras da aplicação, ou seja, como os dados irão trafegar no sistema e como serão acessadas as regras de negócio da camada anterior. Geralmente, a camada de aplicação é responsável pela implementações de padrões como o de repositório-serviço (usado nesse trabalho) ou o CQRS (*Command and Query Responsability Segregation*);
- Infraestrutura: a responsabilidade dessa camada é a comunicação com outros sistemas externos. É aqui também onde se encontram as classes que lidarão diretamente com o banco de dados;
- Apresentação: essa é a camada de interação com usuários, podendo ser uma aplicação web. uma API, um aplicativo mobile ou alguma outra forma de emitir informações.

Figura 33 – Diagrama de uma Arquitetura Limpa Genérica



Fonte: Adaptado de Martin, 2020.

As setas na Figura 33 estão indicando o sentido de dependência. Dessa forma, as camadas só podem depender daquelas que estão mais próximas do centro do círculo, nunca ao contrário. Isso acontece por causa do conceito de proteção contra mudanças, exposto por (MARTIN, 2020).

Para entender essa proteção, observe que a camada de aplicação não depende da camada de apresentação. Dessa forma, qualquer mudança na interação com o usuário não afetará as regras de aplicação e, tampouco, as regras de negócio. Porém, uma mudança na camada de aplicação pode impactar a camada de apresentação.

Esse efeito é desejável porque a interação com o usuário está em constante alteração, porém, as regras mais internas não mudam com tanta frequência. Em contrapartida, se a camada de aplicação mudar, a de apresentação muito provavelmente terá que ser modificada porque as regras de tráfego de informações sofreram alterações. O mesmo raciocínio se aplica em relação à camada de infraestrutura.

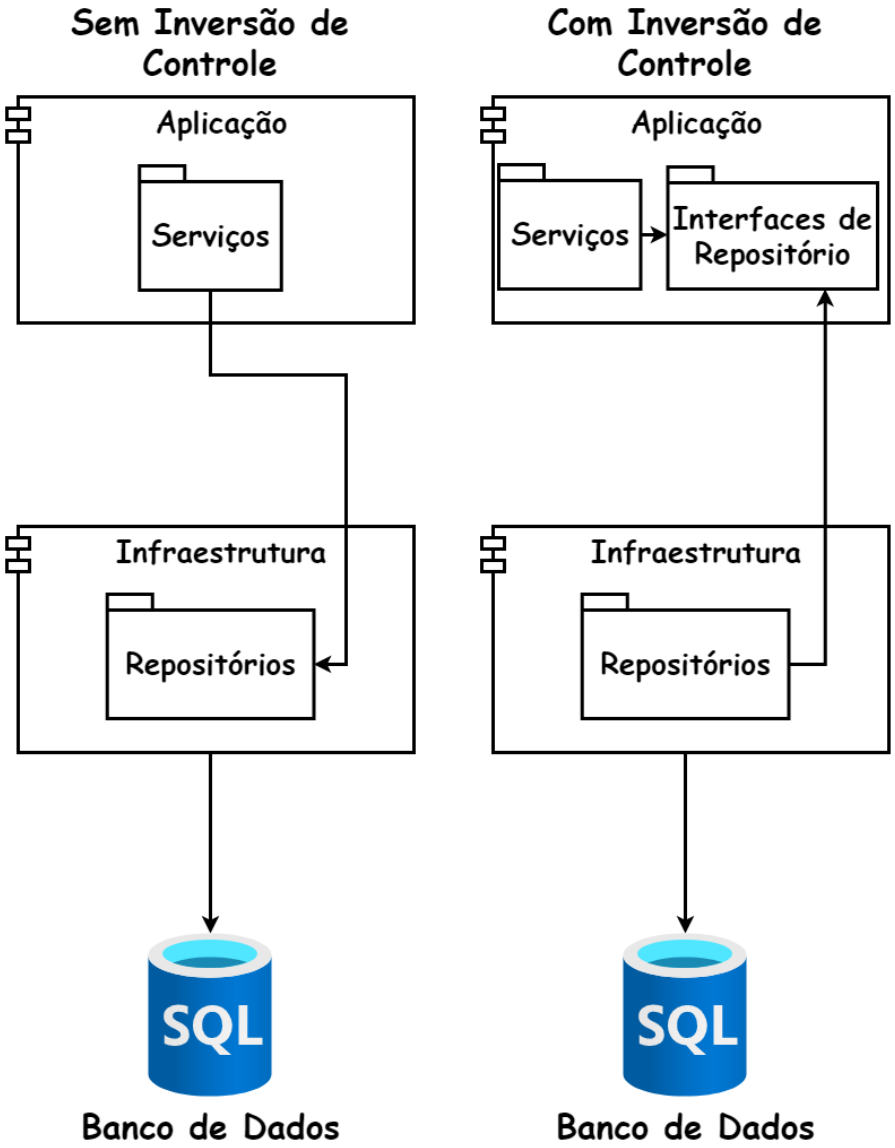
Como última análise, a camada de domínio é a mais protegida de todas, pois as regras de negócio são o núcleo do sistema (MARTIN, 2020). Por exemplo, se no futuro o projeto aqui descrito mudar do padrão repositório-serviço para CQRS, tal troca não deve afetar as regras de alto nível, pois elas ainda continuarão existindo da forma que foram definidas. Porém, se as regras de domínio forem modificadas, então é natural esperar que todo o resto do sistema seja adaptado.

Com base no que foi discutido até aqui, é possível que surjam dúvidas sobre como a camada de aplicação vai chamar as classes na camada de infraestrutura para fazer o armazenamento no banco de dados, pois, conforme foi explicado, a dependência ocorre rumo ao centro do círculo da arquitetura limpa genérica. A próxima seção é dedicada a sanar esse tipo de questionamento.

5.4.1 Princípio da Inversão de Controle

A Figura 34 mostra uma comparação entre a aplicação e a não aplicação do Princípio da Inversão de Controle (IoC - *Inversion of Control*). A forma de implementação que gera o questionamento supracitado é a da esquerda, onde, para acessar um repositório, um serviço na camada de aplicação dependeria da camada de infraestrutura.

Figura 34 – Princípio da Inversão de Controle nas Camadas de Aplicação e Infraestrutura



Uma maneira de resolver esse problema é fazendo uso de interfaces, um recurso presente em algumas linguagens orientadas a objetos como Java e C#. Uma interface é como se fosse um contrato, onde as classes que a implementam devem possuir os métodos previamente estabelecidos (MARTIN, 2009). Definindo o contrato de repositório na camada de aplicação, inverte-se a dependência porque agora a classe que implementa o repositório precisa depender desse contrato,

portanto, no exemplo da direita, aplicando o princípio IoC, a camada de infraestrutura depende da camada de aplicação. E esse raciocínio pode ser estendido para os demais casos.

O uso de interfaces contempla também um princípio do código limpo exposto em (MARTIN, 2009), que é o de blindar o seu código em relação à sistemas de terceiros. Ao usar interfaces, toda a lógica de acesso externo fica contida em classes próprias para isso e todo o sistema opera somente através dos contratos, desse forma, caso haja alguma mudança nas dependências externas, basta modificar a classe responsável, mantendo os métodos definidos na interface.

Em suma, o padrão arquitetural exposto na Figura 33, com o auxílio do princípio IoC, traz os seguintes benefícios (MARTIN, 2020):

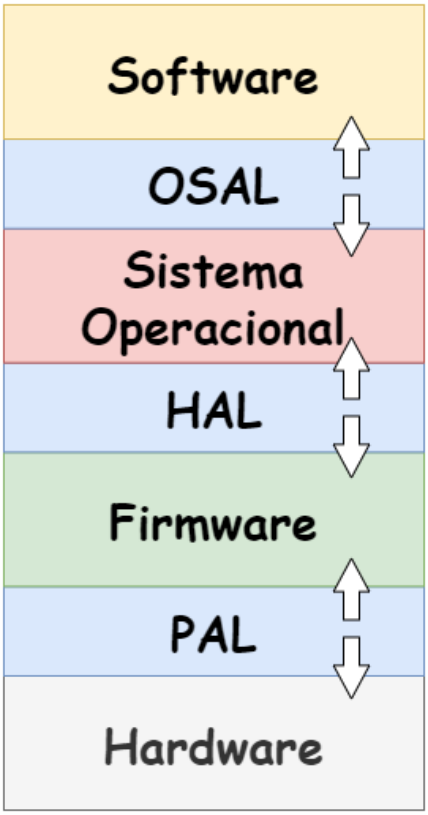
- Independência de *frameworks*: ao separar as regras mais internas em camadas específicas, a arquitetura limpa pode ser usada com qualquer *framework*, podendo, inclusive, haver mudanças ao longo do tempo uma vez que as camadas internas estão devidamente protegidas;
- Testável: a divisão em camadas e o uso de interfaces permite o emprego de testes automatizados, porém, embora tenham sido usados no projeto, discorrer sobre testes automatizados foge do escopo desse trabalho;
- Independência da Interação com o Usuário: a responsabilidade de interagir com o usuário fica em uma das camadas externas, que é a de apresentação;
- Independência do Banco de Dados: uma vez que o conhecimento do banco de dados fica restrito à camada de infraestrutura, ao mudar o local de armazenamento, o restante do sistema permanecerá indiferente;
- Independência de Sistemas Externos: a camada de infraestrutura também isola demais serviços externos, bastando apenas que o restante do sistema utilize as interfaces definidas na camada de aplicação.

Com algumas adaptações, é possível também usar a arquitetura limpa em sistemas embarcados, conforme abordado na próxima seção.

5.5 Arquitetura Limpa em Sistemas Embarcados

A arquitetura limpa se aplica a todos os tipos de sistemas, justamente por ser composta por princípios e permitir customizações (MARTIN, 2020). Essa aplicabilidade holística contempla, portanto, programação de nível mais baixo, como é o caso de *firmwares* e sistemas operacionais.

Figura 35 – Arquitetura Limpa Genérica do *Hardware* ao *Software*



Fonte: Adaptado de Martin, 2020.

Dito isso, a Figura 35 mostra como é recomendado organizar um sistema, partindo do *hardware* até o *software*.

Inicialmente, tem-se o *hardware*, que é parte física do sistema e, também, aquela inflexível. Dito de outra forma, só é possível modificar o hardware construindo outro circuito. Para blindar as camadas superiores dessa característica, existe a PAL (*Processor Abstraction Layer* - Camada de Abstração do Processador) (MARTIN, 2020).

A PAL permite que instruções como acessar um dado registrador ou ler uma determinada porta seja feita sem que as demais camadas tenham código específico para aquele processador ou microcontrolador. Trazendo para o contexto do presente trabalho, é graças a PAL que é possível migrar do ESP32 para Arduino sem alterar diretamente o *firmware*.

Subindo na figura, há a HAL (*Hardware Abstraction Layer* - Camada de Abstração do Hardware). A HAL consegue isolar as informações do *firmware* das camadas superiores, dessa forma, atualizações no *firmware* podem ser feitas sem impactar o código de nível mais alto. A HAL existe principalmente na confecção de sistemas operacionais (MARTIN, 2020).

A última camada de abstração é a OSAL (*Operating System Abstraction Layer* - Camada de Abstração do Sistema Operacional). Ao isolar o código específico de sistema operacional

em um camada, é possível fazer um *software* multiplataforma, conforme ocorre com o .NET (MARTIN, 2020).

Nessa visão mais holística também há uma orientação acerca das dependências, porém, ele ocorre justamente nas abstrações. Para exemplificar, será explicado usando a PAL, todavia, o raciocínio pode ser expandido para as demais camadas.

A PAL depende tanto do *firmware* quanto do *hardware*. A parte de dependência do *firmware* acontece porque é nele que serão definidas as interfaces de comunicação. É através dessa inversão de controle que o *firmware* é blindado de modificações no *hardware*. A título de exemplo, pode-se criar uma interface que contenha um método para ler a temperatura, porém, o *firmware* não conhece a implementação que de fato faz a leitura.

Uma vez que a PAL precisa se comunicar com o hardware, é ela que é a responsável pela implementação das interfaces do *firmware*, portante, é esperado que o código da PAL tenha instruções específicas para o hardware em questão.

Um outro ponto a ser destacado é a testabilidade que o sistema embarcado ganha ao ser programado usando a arquitetura limpa. Graças às camadas de abstração, é possível levar o código sob teste para qualquer plataforma, garantindo, por exemplo, testes automatizados no computador de desenvolvimento antes de fazer o *upload* do *firmware* para o *hardware* (MARTIN, 2020).

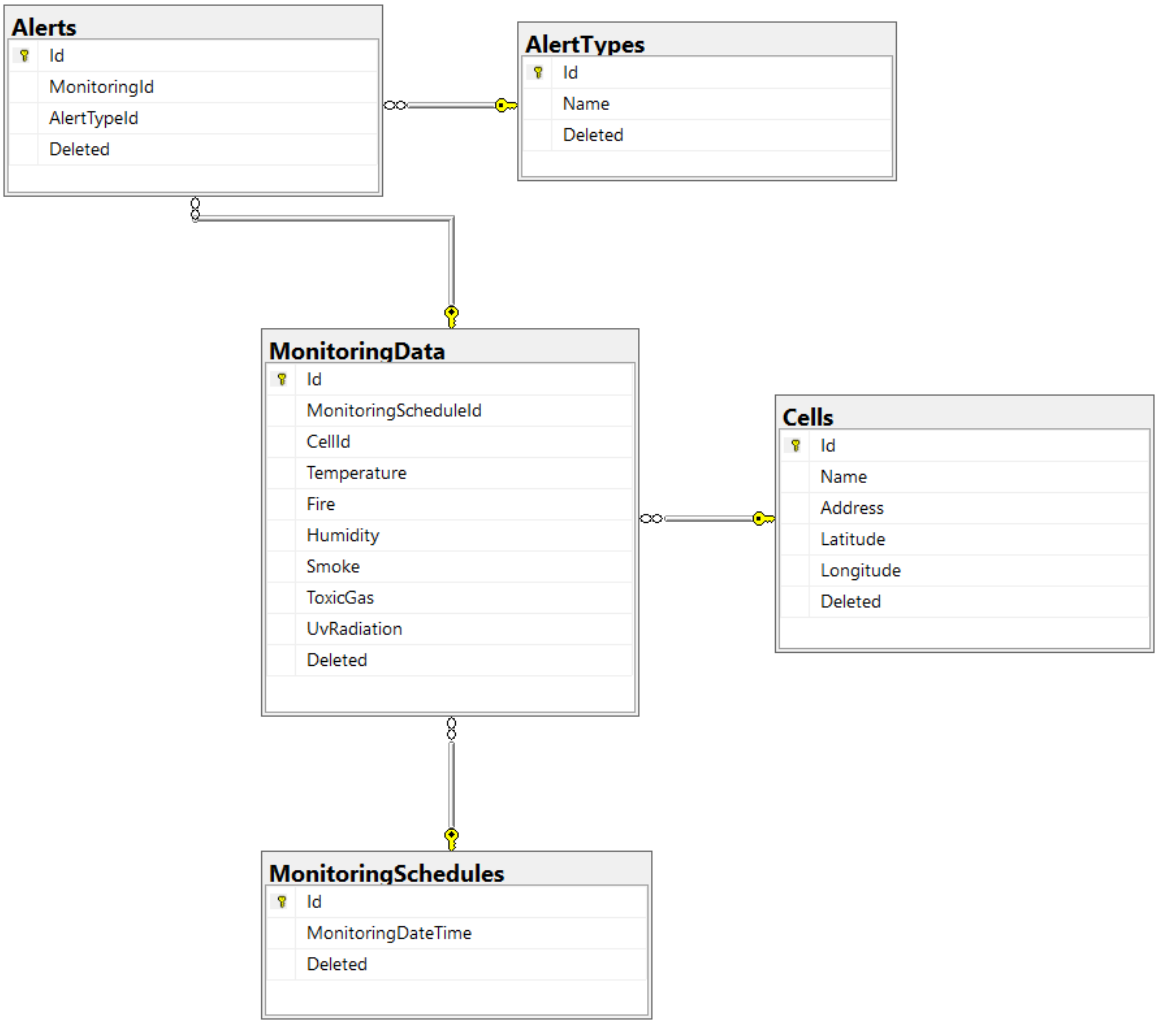
Para finalizar a discussão sobre arquitetura limpa, é importante mencionar que os conceitos aqui expostos são princípios simples e amplos, logo, são adaptáveis de acordo com a realidade de cada sistema. Graças a essa versatilidade e simplicidade, os ensinamentos da arquitetura limpa podem ser aplicados em qualquer sistema e em qualquer contexto (MARTIN, 2020).

6 MATERIAIS E MÉTODOS - SERVIDORES E INFRAESTRUTURA

6.1 Banco de Dados

O Sistema de Gerenciamento de Banco de Dados (SGBD) usado no presente trabalho é o SQL Server 2019 Express, da Microsoft, com um banco de dados modelado conforme a Figura 36.

Figura 36 – Modelagem do Banco de Dados



Salienta-se que as tabelas relacionadas ao *login* foram omitidas porque, conforme será explicado na próxima seção, foi usado a biblioteca ASP.NET Core Identity 2.2.0, a qual já possui seu funcionamento documentado em (MICROSOFT, 2020a).

Exposta a Figura 36, é importante entender o relacionamento das tabelas:

- ***MonitoringData***: Essa tabela armazena os dados dos monitoramentos. Além disso, faz a associação do monitoramento com uma célula e do monitoramento com um agendamento.

- ***Cells***: Guarda os dados de cada célula, ou seja, nome, endereço e localização;
- ***MonitoringSchedules***: Uma tabela para armazenar os agendamentos de medições. Como uma forma de obter sincronia, as células obedecem a esse registro para enviar seus respectivos monitoramentos;
- ***Alerts***: Tabela que relaciona cada monitoramento com os tipos de alerta emitidos de acordo com os valores das medições;
- ***AlertTypes***: Contém os tipos de alertas que podem ser emitidos de acordo com os valores dos sensores.

A Tabela 7 mostra os valores da tabela ***AlertTypes***. Por fim, é importante dizer que a estrutura do banco de dados (nome das tabelas e colunas) está em inglês para garantir compreensão da arquitetura internacionalmente. Já o conteúdo está em português por se tratar de uma aplicação no Brasil, de modo que os registros podem ser facilmente alterados para outros idiomas.

Tabela 7 – Valores da Tabela *AlertTypes*

Id	Name	Deleted
1	Temperatura Elevada	0
2	Baixa Umidade	0
3	Fogo Detectado	0
4	Radiação UV Elevada	0
5	Fumaça Detectada	0
6	Gases Tóxicos Detectados	0

Um último ponto a ser mencionado acerca desse tópico diz respeito à coluna ***Deleted*** em todas as tabelas da Figura 36. Ela está presente em virtude do conceito de *Soft Delete*, apresentado por (RESCA, 2019). Nessa proposta, os dados não são deletados para evitar inconsistências no banco de dados, porém, caso a coluna ***Deleted*** tenha valor 1, esse registro será ignorado pela aplicação.

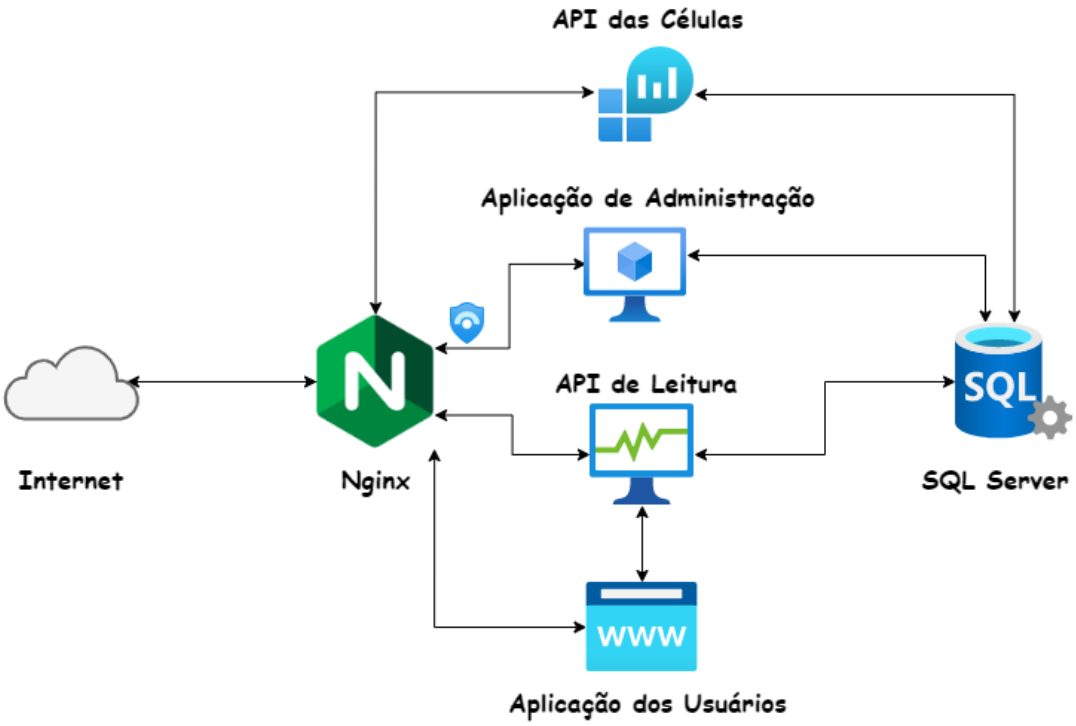
6.2 APIs e Aplicações

6.2.1 Nginx

O Nginx é um *software* de código aberto que pode atuar como servidor web, balanceador de cargas, *proxy* reverso e outras coisas (NGINX, [S.d]). Conforme (DEJONGHE, 2019), umas das aplicações do Nginx é em sistemas conteinerizados, afinal, o próprio *software* pode ser inserido dentro de um contêiner. A seguir serão exibidas as características de cada API e Aplicação e, ao final, será exposto a configuração do Nginx.

A Figura 37 mostra a organização do servidor. Como é possível ver, todas as requisições passam pelo Nginx. De cima para baixo, a primeira API é das células, usada para enviar as medições para o banco de dados. Como é usada apenas pelas células, não há uma aplicação com interface gráfica associada a ela.

Figura 37 – Arquitetura do Servidor



Abaixo está uma aplicação para administração das células, usada pelos administradores da RSSF. Ela é uma aplicação web comum (modelo MVC) e o escudo desenhado nela significa que só é possível acessá-la através da VPN, portanto, essa aplicação não fica disponível para acesso externo. Além disso, por seguir o modelo MVC, a aplicação se comunica diretamente com o banco de dados.

Em seguida, há a API de leitura. Essa API é responsável por ler os dados de banco de dados e expô-los de forma tratada e compreensível para seres humanos. Porém, por ser uma API, ela sozinha não tem interface gráfica, precisando de uma aplicação (*web* ou *mobile*) para poder exibir essas informações.

Por fim, a aplicação dos usuários é a que mostra as medições para o mundo externo. Observe que, como ela se comunica com a API de leitura, não há necessidade de essa aplicação acessar o banco de dados, dessa forma, é possível criar diversas aplicações de leitura para diferentes dispositivos, bastando apenas que elas acessem a API de leitura.

Todas as aplicações e APIs foram feitas em .NET 5, um ecossistema multiplataforma e código aberto da Microsoft focado em desenvolvimento de sistemas *Desktop, Web, Mobile, Jogos, Machine Learning* e *Cloud* lançado em novembro de 2020 (MICROSOFT, 2020c). As

APIs e a aplicação de administração foram feitas usando *framework* ASP.NET 5, já a aplicação de leitura, por sua vez, foi criada com o *framework* Blazor 3.2.

6.2.2 Pipeline CI/CD

Antes de inicar os comentários acerca do assunto, é importante frisar que, salvo quando dito ao contrário, as informações dessa seção podem ser encontradas na documentação da ferramenta GitLab (GITLAB DOCS, [S.d]).

No Gitlab, as *pipelines* são divididas em *jobs* e estágios. Os estágios são agrupamentos de *jobs* e são executados sequencialmente. Já os *jobs*, por sua vez, são a mínima unidade de execução, ou seja, são os comandos que serão executados para de fato construir e publicar as aplicações. Vale dizer ainda que os *jobs* de um mesmo estágio são executados paralelamente, portanto, se houver uma relação de dependência entre *jobs*, eles devem ficar em estágios separados.

Outra informação relevante a ser comentada é que os *jobs* normalmente são executados em contêineres docker, porém, é possível designar uma máquina específica para executar os comandos. Essa possibilidade foi usada nessa *pipeline* e será devidamente comentada no momento oportuno. É possível ainda configurar serviços, chamados *services*, que são outras imagens docker anexadas ao contêiner do *job*.

A *pipeline CI/CD* é configurada através do arquivo *.gitlab-ci.yml*, o qual é exibido a seguir.

```
image: mcr.microsoft.com/dotnet/sdk:5.0

stages:
  - tests
  - publish
  - deploy

test_docker:
  image: docker
  stage: tests
  script:
    - docker login $CI_REGISTRY -u $User -p $Password
  services:
    - docker:dind

unit_tests:
  stage: tests
  script:
```

```
    - dotnet test -c Release

publish:
  stage: publish
  before_script:
    - $Env:PATH += ":" + "/bin/dotnet"
  script:
    - docker login $CI_REGISTRY -u $User -p $Password
    - . ./ci.ps1
    - Publish
    - Dockerize
  only:
    - master
  tags:
    - publish-srv
```

A primeira instrução informa a imagem docker padrão a ser usada em todas as etapas, ou seja, caso alguma etapa não informe explicitamente a imagem, essa padrão é a que será utilizada. Conforme pode ser visto, a imagem padrão é a do SDK (*Software Development Kit* - Kit de Desenvolvimento de *Software*) do .NET 5.0.

A próxima instrução é referente aos *stages*, que representam as etapas da *pipeline*. Conforme é mostrado, o processo é dividido em três: testes (*tests*), publicação (*publish*) e entrega (*deploy*).

Iniciando os testes, há dois *jobs*: o *test_docker* e o *unit_tests*. O primeiro deles testa as credenciais para enviar as imagens docker geradas pela *pipeline*. Aqui, é usada a imagem oficial do docker e é empregado o serviço DinD (*Docker in Docker*), o qual permite acessar o próprio docker dentro de um contêiner. O outro *job* realiza os testes unitários do projeto para assegurar a ausência de *bugs* dentro dos cenários considerados pelo desenvolvedor.

A próxima etapa, a publicação, é executada em uma máquina específica. Naturalmente, dependendo das especificações necessárias, as imagens docker não serão suficientes, mesmo quando estendida com os serviços. Para isso, o GitLab permite que *jobs* sejam executados em máquinas escolhidas pelo próprio desenvolvedor, onde essas máquinas recebem um agente chamado *GitLab Runner*, responsável por executar a *pipeline* na máquina em questão. Cada máquina recebe um nome, de modo que é possível atribuir *jobs* a uma dada máquina através da instrução *tags*, que recebe a identificação da máquina em questão.

A máquina escolhida para a publicação recebeu o nome de *publish-srv*. O que torna essa máquina tão específica é a combinação do .NET 5.0, PowerShell Core 7.1 e Debian 10.7. É justamente por esse motivo que os scripts, para esse *job*, possuem instruções em PowerShell.

Essa escolha se deve ao fato de o PowerShell proporcionar maior poder na linha comando, permitindo automatizar tarefas mais facilmente em relação ao Bash, linha de comando padrão do Linux.

As etapas desse *job* são as seguintes: ele faz *login* no registro docker do GitLab, executa as instruções no arquivo *ci.ps1* e chama as funções *Publish* e *Dockerize*. Por fim, a seção *only* do *job* informa que essa etapa deve ser executada somente na *branch master*.

6.2.3 PowerShell Core

O PowerShell Core é uma estrutura multiplataforma para gerenciamento e automação de tarefas. Essa ferramenta é constituída de um utilitário de linha de comando (*shell*) e uma liguagem de *scripts* (MICROSOFT, 2020b).

A seguir é exibido o script responsável pelo *job* de publicação da *pipeline* de entrega contínua.

```
$projectsToPublish=@(
        "WSN.Admin"
        "WSN.Cells"
        "WSN.User.API"
        "WSN.User.SPA"
)

function GetPublishDir($project) {
    return "publish/" + $project
}

function GetProjectPath($project) {
    return "src/" + $project + "/"
}

function GetProjectFullPath($project) {
    return $(GetProjectPath $project) +  $project + ".csproj"
}

function GetDockerImageName($project) {
    return $project.ToLower().Replace(".","-")
}

function GetFullDockerImageName($containerRegistry, $project,
```

```
    $tag) {
      return  $containerRegistry + ":" + $(GetDockerImageName
         $project)
}

function GetDockerFilePath($project) {
    return $(GetProjectPath $project) + "Dockerfile"
}


function Publish() {
    foreach($project in $projectsToPublish) {
        dotnet publish --configuration Release -o $(
           GetPublishDir $project) $(GetProjectFullPath
           $project)
    }
}

function Dockerize() {
    foreach($project in $projectsToPublish) {
        $imageName = $(GetFullDockerImageName "registry.gitlab.
           com/formacao/mestrado/wsn-cidade-digital" $project)

           docker build -f  $(GetDockerFilePath($project)) -t
              $imageName .
           docker push $imageName
    }
}
```

Uma vez que os comandos para gerar os binários e a imagem docker são similares, mudando apenas o nome e diretório dos projetos, esse *script* foi criado para automatizar esse processo, sendo assim, evita-se ficar repetindo comandos no arquivo *.gitlab-ci.yml*.

6.3 Teoria de Backup

Segundo (FARIA, 2017), *backup* consiste em manter redundância de dados para posterior recuperação em caso de perda das informações originais. Ainda em conformidade com essa fonte, os procedimentos de cópia de segurança são feitos geralmente à noite, pois assim o *backup* não interfere nos demais procedimentos do horário comercial.

Uma vez que esse trabalho emprega a ferramenta *open source* Bacula, as especificações serão acerca dele. Dito isso, o primeiro passo é conhecer alguns termos técnicos (FARIA, 2017):

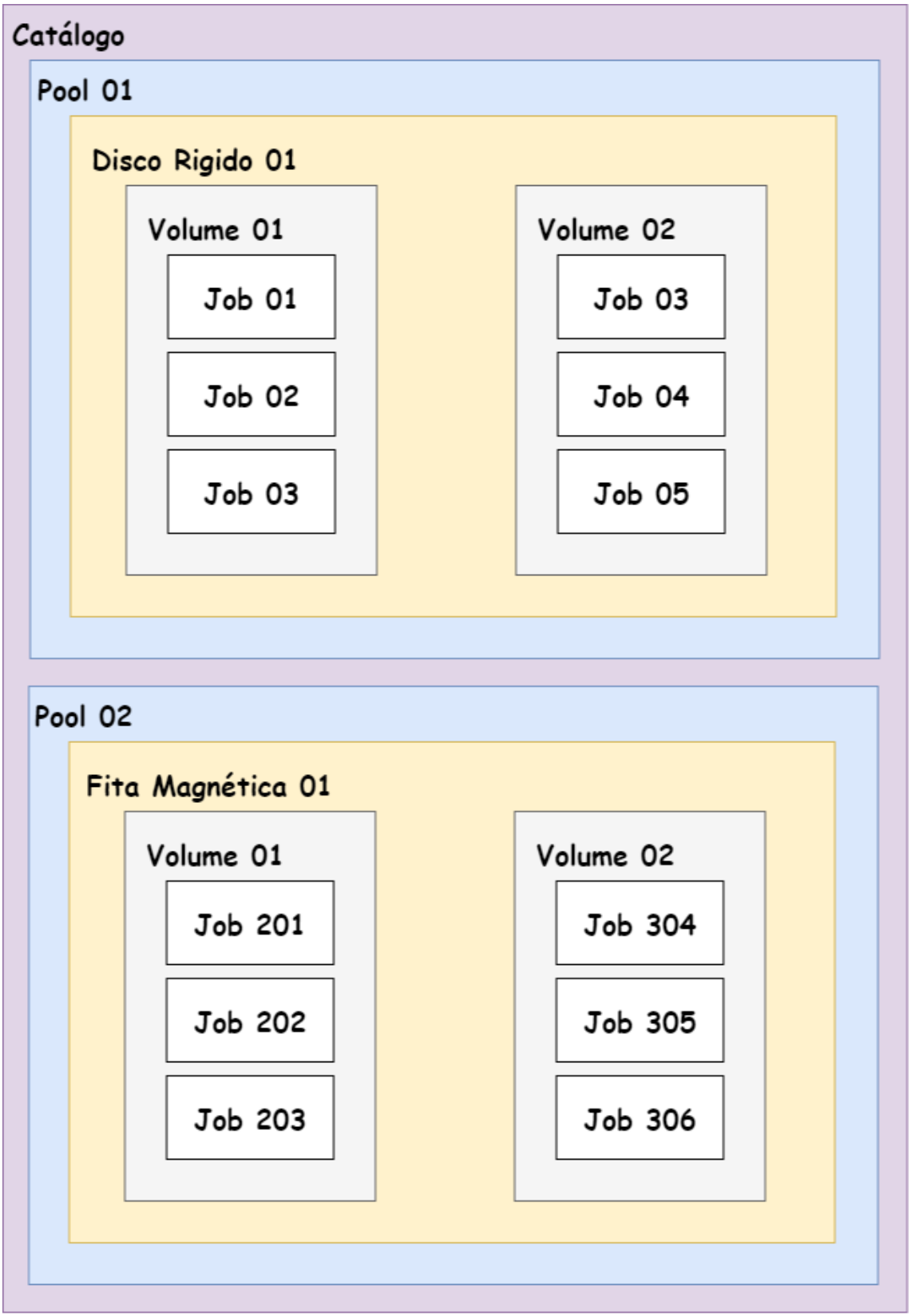
- Retenção: É o tempo que o *backup* permanece guardado. Dado que o Bacula faz *backups* automáticos, é imprescindível que os mais antigos sejam apagados (ou transferidos) para dar lugar aos mais novos e manter sustentabilidade no local de armazenamento;
- Expiração: Estado após o prazo de retenção. Aqui, os *backups* podem ser substituídos por novos se não houver mais espaço;
- *Job*: É a operação de *backup* em si. O *Job* pode ser tanto uma cópia de segurança, como uma restauração ou migração (mover o *backup* de uma máquina para outra);
- Volume: Os volumes são arquivos binários que contêm o *backup*. Em outras palavras, os *jobs* não fazem simples cópia de arquivos, eles compactam tudo em um volume, o qual pode ter seu tamanho limitado. Vale dizer que a reciclagem é feita no volume todo, portanto, o Bacula espera a expiração de todos os *backups* guardados nesse arquivo;
- *Pool*: Conjunto de volumes com características semelhantes. O administrador do Bacula consegue no máximo ligar um *job* aos *pools*, ou seja, o tratamento dos volumes é feito somente pelo software, não há intervenção do usuário;
- *FileSet*: Lista de diretórios para fazer *backup*;
- Catálogo: Banco de dados que armazena índices e informações sobre os *backups*. É a partir do catálogo que é possível navegar entre os arquivos de cada *job*, por exemplo.

A Figura 38 mostra um exemplo de organização dessas definições. O primeiro fato a ser observado é que um mesmo *job* pode ocupar dois volumes distintos ou ficar contido em apenas um único volume. Isso é feito automaticamente pelo Bacula (FARIA, 2017). O segundo ponto a ser observado é que os volumes não precisam coincidir com o armazenamento físico, ou seja, o mesmo disco rígido, por exemplo, pode ter vários volumes. É necessário dizer que o Bacula trabalha com vários dispositivos de armazenamento e, por fim, o catálogo armazena penas informações dos *backups*, e não o conteúdo (FARIA, 2017).

6.3.1 Tipos de Backup

É importante conhecer os tipos de *backups*, pois cada um traz características distintas que podem maximizar a eficiência da política de *Backup* (FARIA, 2017).

Figura 38 – Conceios Básicos do Bacula



Fonte: Adaptado de Faria, 2017

6.3.1.1 Backup Full

Backup Full ou Total copia todos os arquivos do *FileSet*. Por ser mais demorado e oneroso, esse *backup* geralmente é feito no final de semana, pois, teoricamente, haverá menor carga no sistema. Por padrão, além do agendamento, o Bacula realiza esse tipo de *backup* quando não houver nenhum outro anterior ou quando houver atualização no *FileSet*.

6.3.1.2 Backup Diferencial

Faz *backup* apenas dos arquivos modificados em relação ao último *Backup Full*. Dito isso, conclui-se que essa modalidade é mais rápida que a anterior. Para fazer a restauração de um *Backup* Diferencial, portanto, o que ocorre é a leitura do último *Backup Full* e do último Diferencial. Em virtude dessa grande dependência do *Backup* Total, é recomendável que esse seja feito com certa periodicidade.

6.3.1.3 Backup Incremental

O *Backup* Incremental copia apenas os arquivos modificados em relação ao último *backup* realizado, independente do nível. Dessa forma, a restauração deve passar por todos de nível Incremental até chegar no último Diferencial, o qual chama o último Total. Vale dizer que essa modalidade de *backup* é a mais rápida, porém, a menos confiável também.

6.3.2 Configuração do Bacula

A configuração do Bacula é relativamente simples, pois, apesar do tamanho do projeto, apenas o banco de dados precisa ter cópia de segurança.

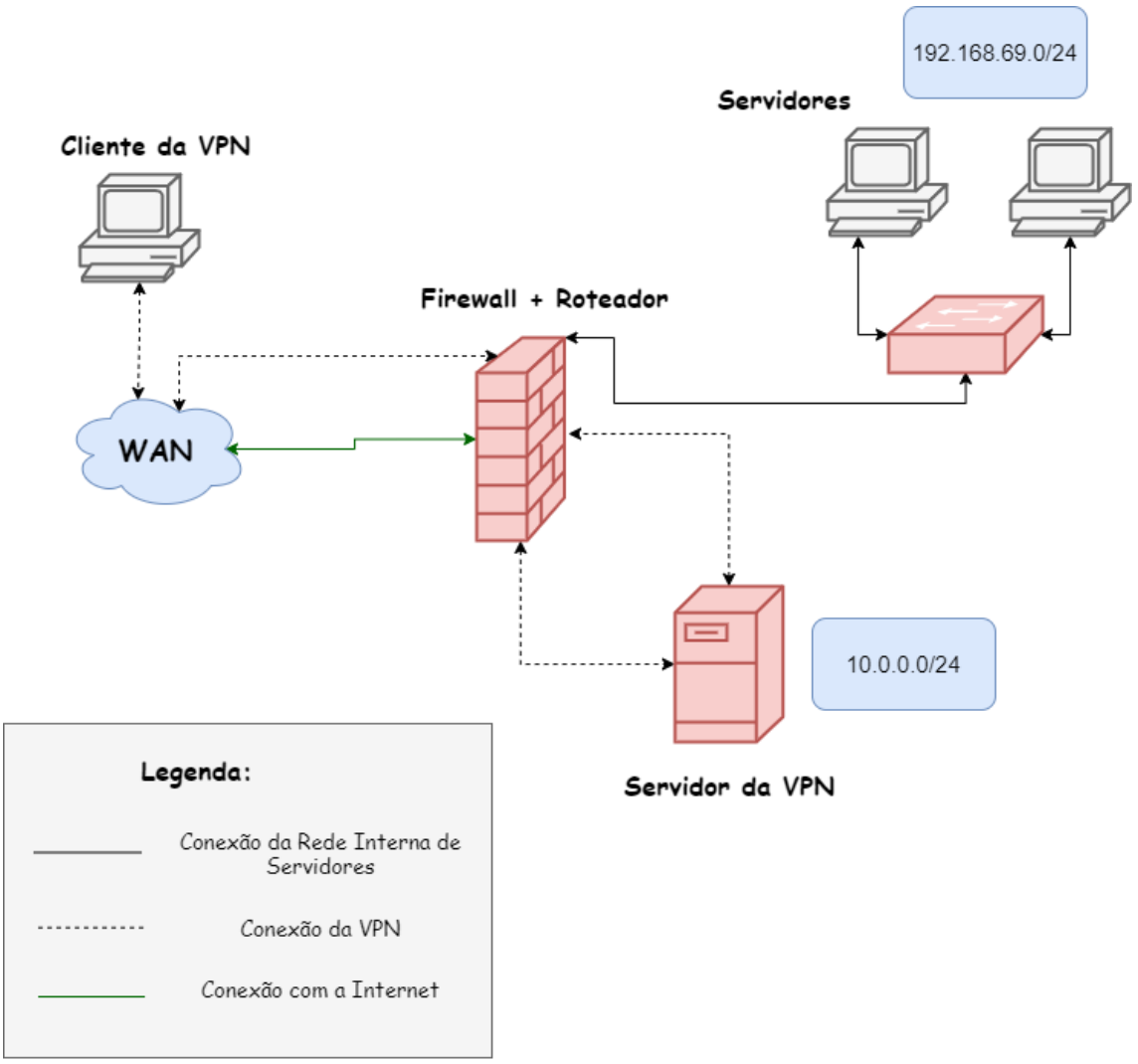
O primeiro passo é estabelecer que o armazenamento será feito no disco rígido da máquina com o Bacula instalado. Feito isso, o *FileSet* a ser copiado é o diretório */var/opt/mssql* dentro do contêiner, o qual pode ser exposto ao mundo externo a partir de um volume, conforme explicado anteriormente.

Em relação à periodicidade, os *backups* incrementais são realizados nos dias úteis e no sábado. Aos domingos, ocorre um *backup* diferencial. Por fim, no primeiro domingo de cada mês é feito um *backup full*. Por fim, vale dizer que esse processo é realizado durante o período da madrugada, de modo a não impactar significativamente o servidor da aplicação.

6.4 Arquitetura da Rede de Servidores

A Figura 39 mostra a rede usada para comportar o sistema de monitoramento. Ela é composta basicamente por três partes: o *Firewall*, os servidores e a VPN.

Figura 39 – Arquitetura da Rede



Inicialmente considere o *Firewall*. Ele atua também como roteador em virtude do tamanho da rede. Como configuração primária, ele possui um NAT para que as máquinas internas consigam se conectar à internet, porém, ele não aceita nenhuma conexão da internet para os servidores.

As configurações de redirecionamento do *Firewall* estão sumarizadas na Tabela 8. Apenas como última informação, o servidor Docker possui o IP $192.168.69.2/24$, que é o IP para o qual as conexão são redirecionadas.

A outra etapa é configurar a VPN, a qual usa a porta 4500. Ao se conectar à VPN, as requisições do cliente, agora criptografadas, são destinadas ao servidor da VPN, e este as encaminha para o *Firewall* para realizar o devido encaminhamento. Em termos práticos, é como se o cliente estivesse conectado internamente ao *Firewall*. O intuito dessa VPN é configurar os

Tabela 8 – Configurações de Redirecionamento para o Servidor Docker

Porta	Aplicação
1433	SQL Server 2019 Express
5000	API para Visitantes
5001	API das Células
8080	Aplicação de Administração
80	Aplicação para Visitantes

demais servidores, pois assim é possível ter uma conexão com as máquinas de dentro da rede de forma segura mesmo estando do lado de fora.

6.5 Arquitetura da Célula

A lista abaixo apresenta os componentes usados para montar o protótipo da célula:

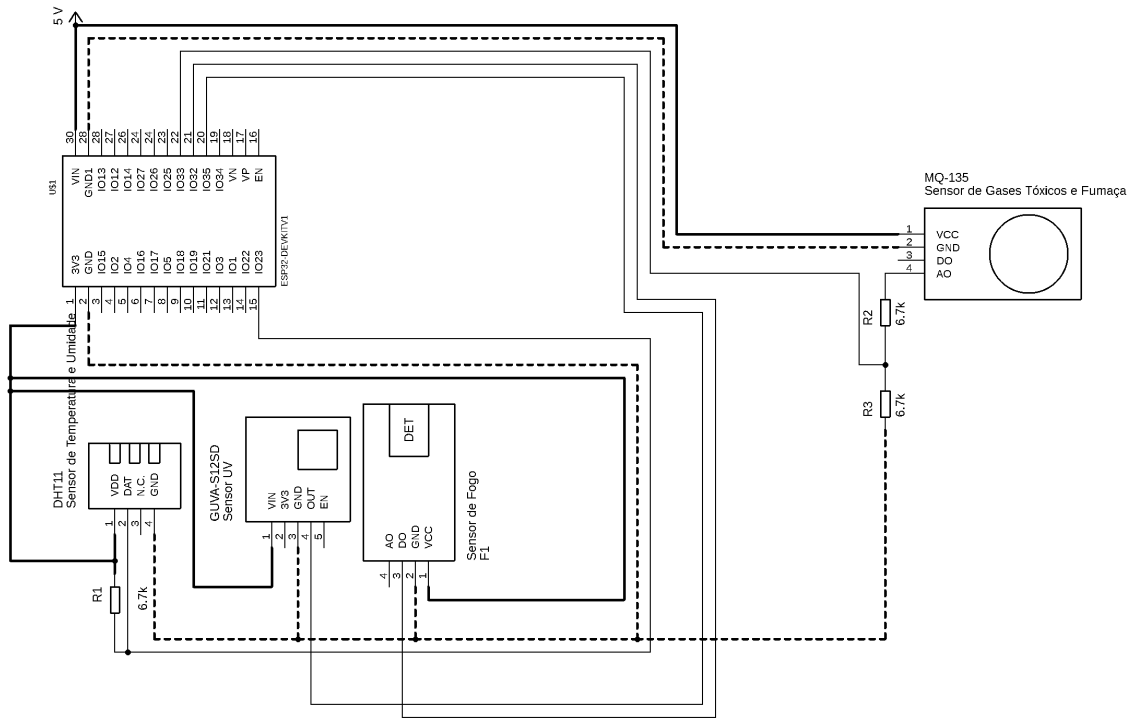
- ESP 32 DevKit V1;
- Sensor DHT 11 (temperatura e umidade);
- Sensor Guva S12SD (radiação ultravioleta);
- Sensor MQ-135 (fumaça e gases tóxicos);
- Sensor Infravermelho (incêndio);
- 03 resistores de precisão de $6,7\ k\Omega$;
- Fonte chaveada DC de $9\ V$ e $2\ A$;
- Circuito regulador de tensão para $3,3\ V$ e $5\ V$.

Os sensores usados para a confecção do protótipo são de versões comerciais, portanto, os mesmos já vêm embarcados juntos com os circuitos necessários para a comunicação, bastando apenas adicionar alguns resistores de *pull-up* e divisores de tensão para calibrar as saídas. A Figura 40 ilustra o esquemático.

Na Figura 40, as linhas de maior espessura são as linhas de alimentação +Vcc, as tracejadas representam as conexões dos terminais GND. As linhas mais finas e contínuas são as conexões de dados. Por fim, os resistores usados são de precisão com resistência de 6,7 kΩ. Conforme é possível observar, só há três resistores: um de *pull-up* para o DHT 11 e os outros dois para formarem um divisor de tensão para o MQ-135.

No tocante à alimentação, o ESP 32 e o MQ-135 são alimentados com 5 V e os demais componentes com 3,3 V. Essa distinção acontece porque o ESP 32 é alimentado com 5 V, porém

Figura 40 – Esquemático da Célula



lê dados com nível máximo de 3,3 V. Uma vez que o MQ-135 usado só funciona em 5 V, é necessário usar um divisor de tensão para mantê-la em conformidade. Por último, destaca-se que o DHT 11 usa comunicação serial, o de infravermelho é um sensor digital (alertando se há fogo ou não) e os demais são analógicos.

6.5.1 Lógica de Funcionamento

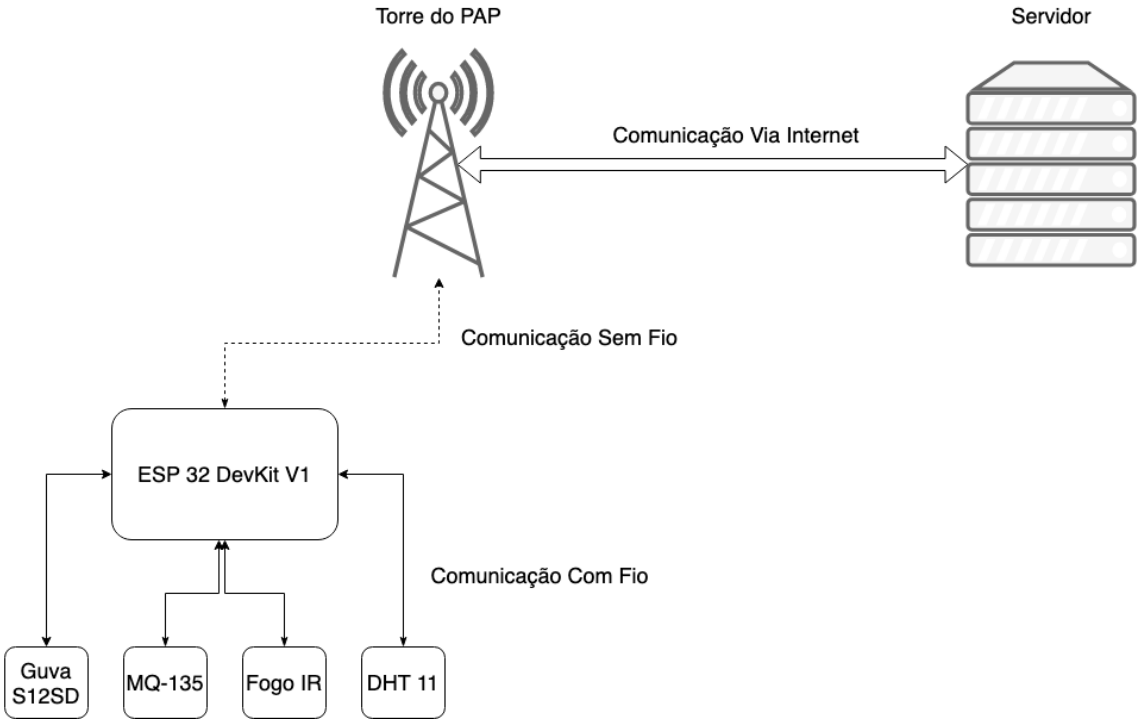
A lógica de funcionamento da célula é simples: ela checa se existe algum monitoramento agendado. Se existir, fará 20 leituras de cada sensor, calculará a média de cada grandeza e enviará ao servidor para que o mesmo realize o processamento e armazenamento das informações. Por fim, como camada de segurança, as células precisam realizar *login* para poder se comunicarem com o servidor.

6.6 Arquitetura da Rede de Sensores

Conforme foi explicado anteriormente, cada PAP possui pontos de acesso *WiFi*, onde há um canal privado para que o nó com sensores, chamado de célula, possa se conectar à internet. Cada célula é composta por um módulo ESP 32 DevKit V1 e os sensores DHT 11 (temperatura e umidade), Guva S12SD (ultravioleta), MQ-135 (gases tóxicos e fumaça) e infravermelho (incêndio). Dito isto, ressalta-se que conjunto é apto a se conectar à rede sem fio privada

disponível no PAP e então consegue estabelecer enlaces com o servidor. A Figura 41 ilustra essa arquitetura.

Figura 41 – Arquitetura da RSSF



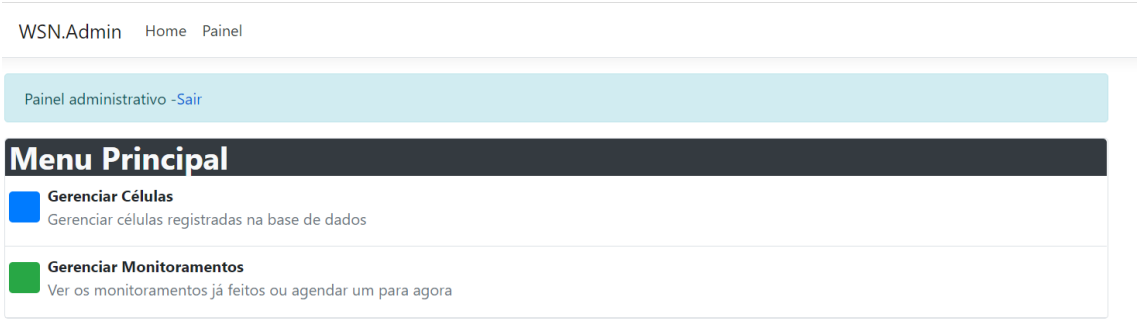
Como é possível observar, o ESP 32 atua como um concentrador dos dados dos sensores e então os manda para o servidor. Essa abordagem é conhecida como *single hop*, isto é, o microcontrolador tem conexão direta com o servidor.

7 MATERIAIS E MÉTODOS - APLICAÇÕES E TESTES

7.1 Aplicação para Administração

A aplicação para administração só pode ser acessada através da VPN, portanto somente os administradores da RSSF possuem acesso a essa parte. Ao inserir com sucesso as credenciais, a primeira tela a ser exibida é a mostrada na Figura 42.

Figura 42 – Painel de Administração



A primeira opção é para gerenciar as células, podendo adicionar, alterar, criar ou mesmo deletá-las. A segunda, por sua vez, permite ver os monitoramentos agendados ou criar um agendamento para o instante atual. Vale recordar que o próprio sistema já agenda monitoramentos periodicamente, portanto, a opção de adicionar pelo painel deve ser usada somente em casos excepcionais.

7.1.1 Gerenciar Células

A Figura 43 mostra a listagem das células cadastradas no sistema, incluindo células virtuais para realizarem testes de carga nos servidores. A tabela exibida na figura em questão possui quatro colunas, uma com o nome atribuído à célula, outra com seu endereço, uma indicado o status da célula e a última com três possíveis ações. Por fim, na parte inferior, há um botão para adicionar uma nova célula.

A Primeira opção detalha o cadastro da célula, conforme Figura 44. Na Figura são exibidas duas informação além do nome e do endereço já presentes na tabela de listagem: a latitude e a longitude. Esses dois campos identificam a localização da célula de forma precisa, enquanto o endereço informa onde a célula está de forma amigável aos humanos.

A próxima opção na Figura 43 é a de deletar uma célula. Aqui, é importante salientar que deletar significa apenas conferir o status de inativo a célula, de modo que ela não possa mais enviar medições. Isso acontece porque, se fosse realmente deletar a célula da base de dados,

Figura 43 – Listagem das Células Cadastradas

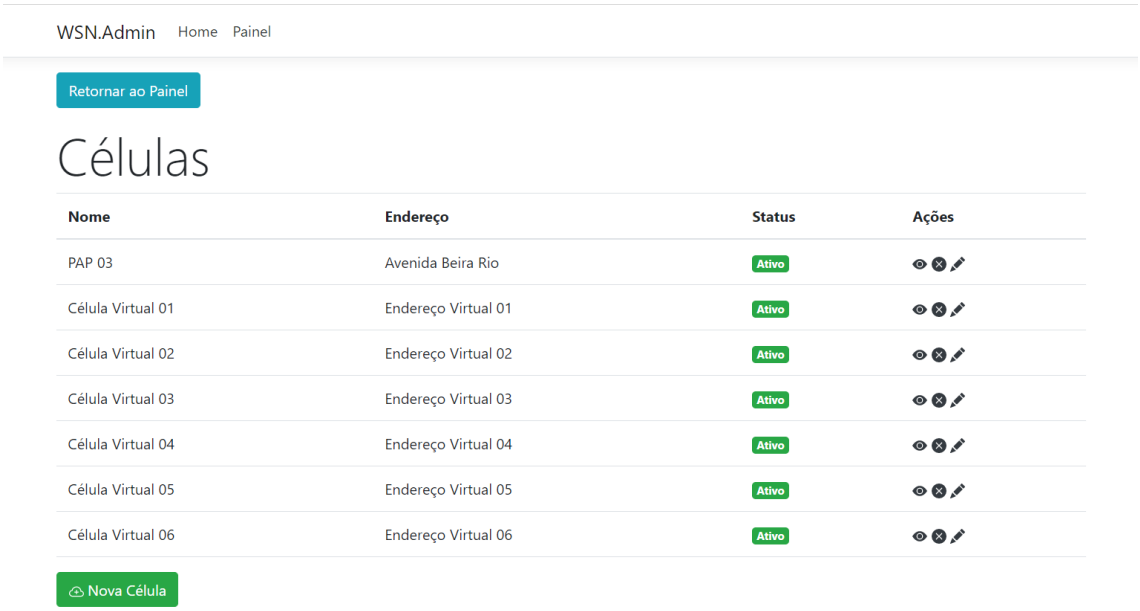
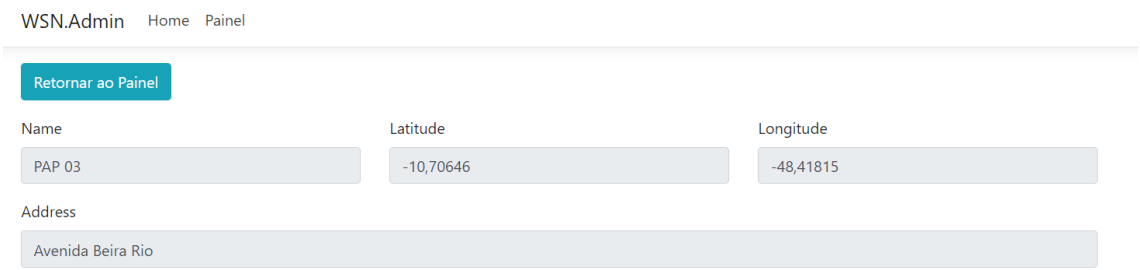


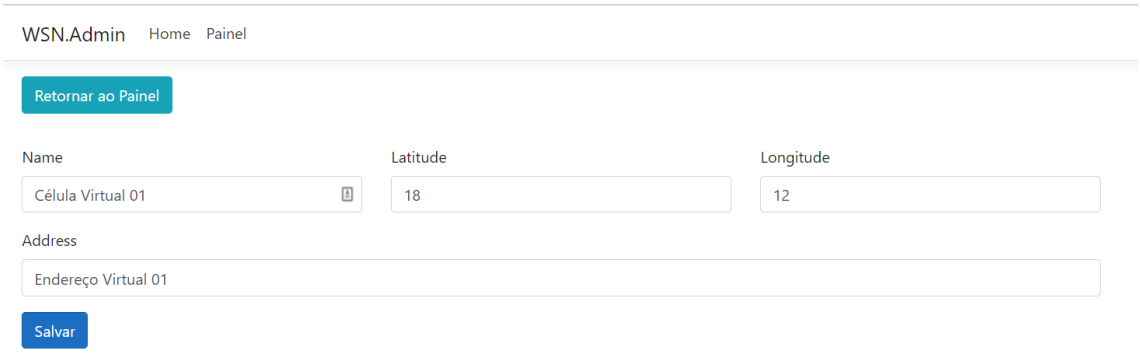
Figura 44 – Exibição dos Detalhes da Célula



precisaria apagar todos os monitoramentos feitos por ela, algo que não é desejável para esse contexto.

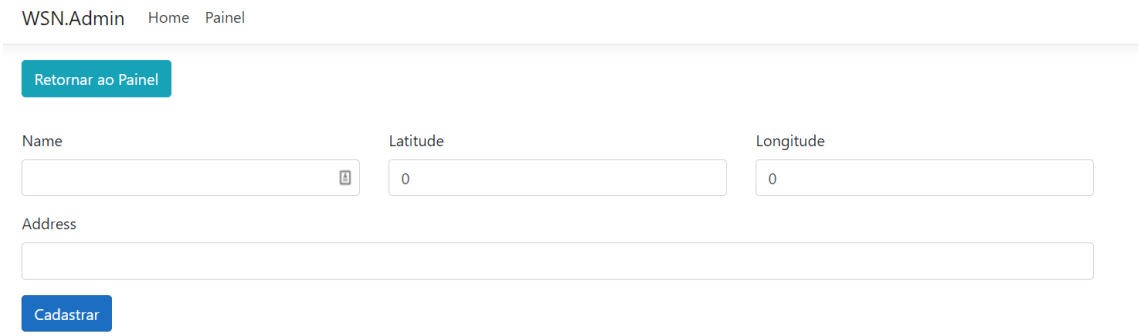
Finalizando as opções de listagem, a última é para editar uma célula. Conforme Figura 45. A estrutura do formulário é idêntica ao de exibição de detalhes, sendo a única diferença que os campos são editáveis.

Figura 45 – Formulário para Edição da Célula



Por fim, o formulário para adicionar célula é exibido na Figura 46, possuindo as mesmas grandezas a serem inseridas que os demais.

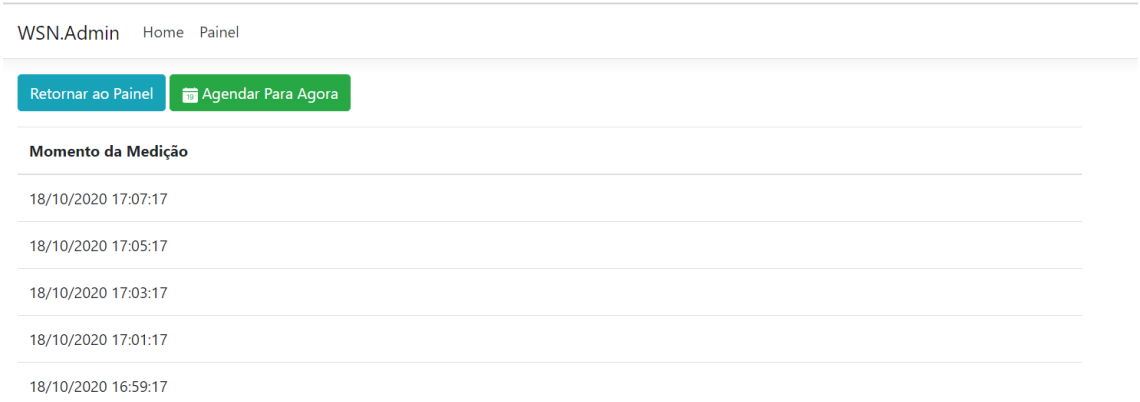
Figura 46 – Formulário para Adição de Célula



7.1.2 Gerenciar Monitoramentos

A Figura 47 exibe a aparência dessa parte do painel de administração. Nela, há uma listagem com os últimos monitoramentos agendados e, além disso, na parte superior, há um botão para criar um agendamento para o momento atual.

Figura 47 – Painel de Gerenciamento de Monitoramentos



7.2 Aplicação para Visitantes

A aplicação para visitantes possui acesso externo liberado, portanto, qualquer pessoa pode acessar os dados. Vale dizer que essa aplicação faz somente operações de leitura, portanto, o usuário não consegue persistir informações no banco de dados. Conforme Figura 48, a aplicação possui um menu lateral que dá acesso às três partes do sistema: Visão Geral, Ver Célula e Dados Brutos.

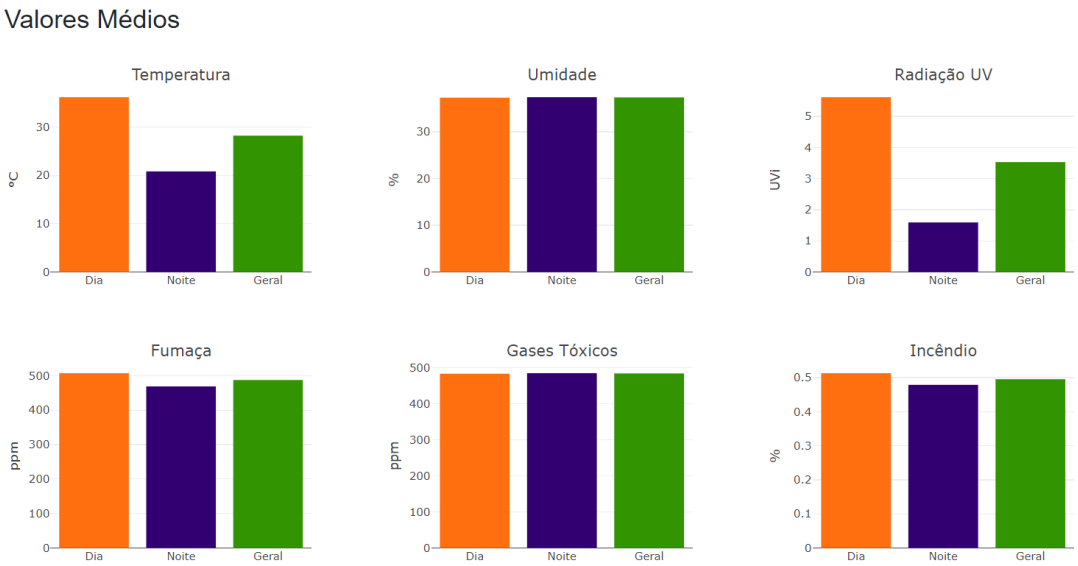
Figura 48 – Menu Lateral da Aplicação para Acesso Externo

7.2.1 Visão Geral

A seção de Visão Geral possui duas divisões mostradas na mesma tela: Valores Médios e Últimos Alertas. A seção de valores médios, conforme o nome sugere, mostra a média de todas as medições de todas as células. Os Últimos Alertas, por sua vez, mostram os cinco alertas mais recentes emitidos por qualquer uma das células.

As Figuras 49 e 50 mostram as informações que são exibidas na tela. Conforme é possível ver na Figura 49, há seis gráficos representando cada uma das seis grandezas: Temperatura (graus Celsius), Umidade (porcentagem), Radiação UV (UVi), concentração de fumaça e gases tóxicos (ppm) e incêndio (porcentagem de ocorrências em relação ao número de medições). Por fim, cada gráfico possui três barras, uma para a média dos valores diurnos, outra para os noturnos e a terceira engloba os dois turnos.

Figura 49 – Valores Médios das Medições



Por fim, a Figura 50 apresenta uma tabela com os cinco alertas mais recentes. Na primeira coluna, a célula é identificada pelo seu endereço, pois assim fica mais fácil saber onde foi emitido

o alerta. A segunda coluna, portanto, informa uma descrição amigável sobre o alerta emitido.

Figura 50 – Últimos Alertas

7.2.2 Ver Célula

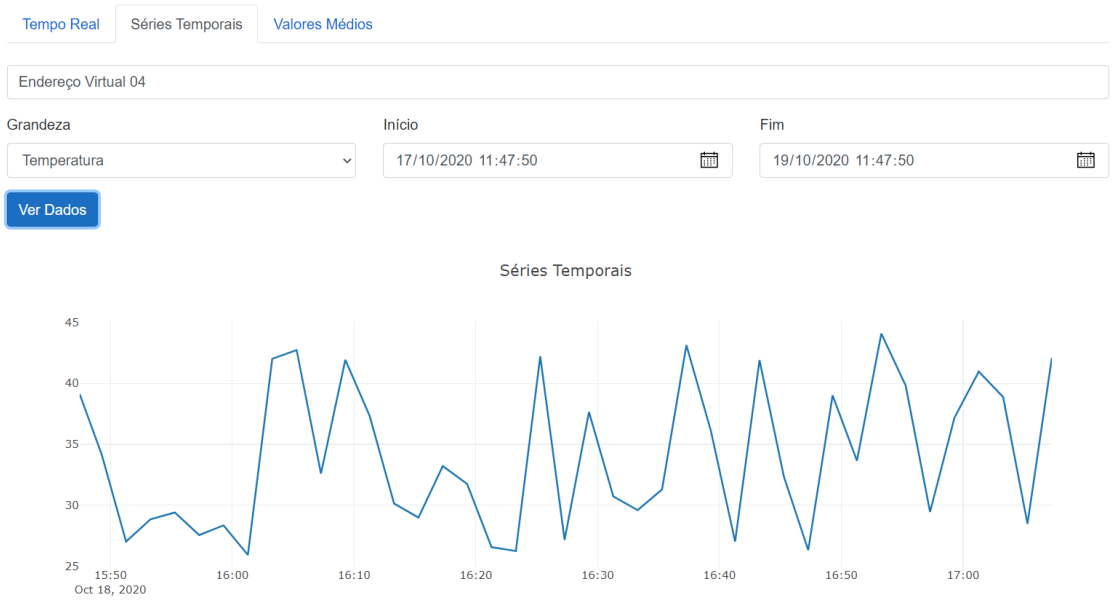
Essa seção apresenta os dados individuais de cada célula e possui três abas: Tempo Real, Séries Temporais e Valores Médios. A Figura 51 mostra tanto essas abas quanto as informações contidas na primeira delas.

Figura 51 – Dados em Tempo Real por Célula

Novamente, há seis gráficos, cada um para cada grandeza. Porém, dessa vez, eles são exibidos na forma de velocímetros, divididos em duas regiões: uma verde, indicado que a medição está abaixo do limiar de alerta, e outra vermelha, indicando que foi emitido um alerta para aquela célula em relação à respectiva grandeza. Por fim, a faixa preta apenas mostra a divisão entre zonas, estando localizada precisamente no local do limiar.

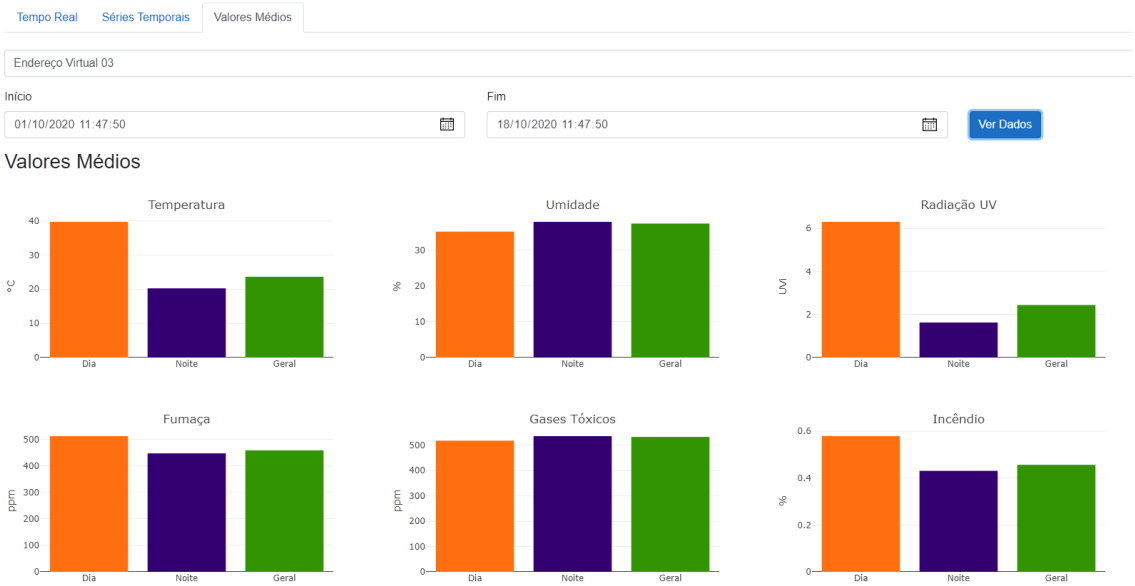
Seguindo adiante nas abas, a Figura 52 mostra a tela Séries Temporais, a qual plota medições ao longo de uma faixa de tempo específica. Aqui também é possível escolher tanto a célula quanto a grandeza a ser exibida.

Figura 52 – Séries Temporais por Célula



A última aba, Valores Médios, mostra as médias das medições de cada grandeza dentro da faixa temporal especificada. A Figura 53 mostra os gráficos, que são similares ao da primeira seção da aplicação.

Figura 53 – Valores Médios por Célula



7.2.3 Dados Brutos

Essa seção é dedicada a exporta os dados das medições para um arquivo *.csv*. Para isso, a Figura 54 mostra o formulário a ser preenchido, havendo a possibilidade, inclusive, de escolher várias células de uma única vez. A Figura 55, por sua vez, mostra o arquivo gerado.

Figura 54 – Formulário para Gerar Arquivo *.csv*

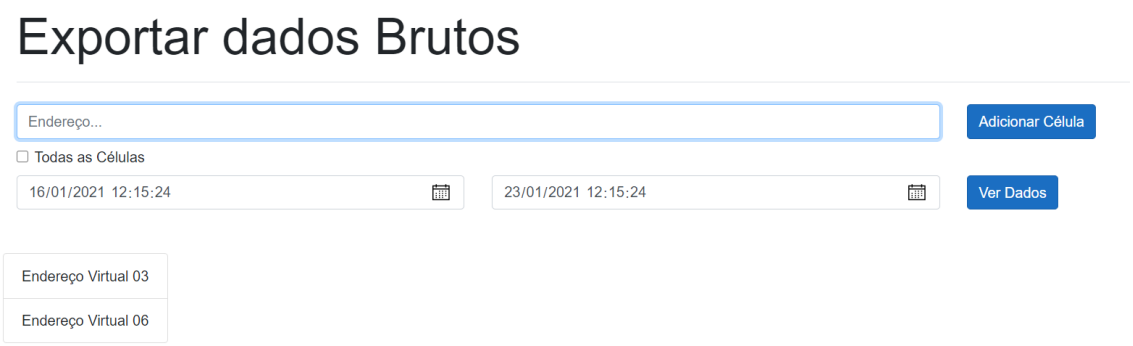


Figura 55 – Arquivo *.csv* Gerado

Temperature	Fire	Humidity	Smoke	Toxic Gas	Uv Radiation	Name	Address	Latitude	Longitude	Monitoring Moment
31.75	0.77	39.58	843	609	4.83	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
35.56	0.4	26.31	109	234	5.34	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
48.79	0.51	33.16	225	224	7.25	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
39.8	0.58	35.67	987	261	7.13	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
45.96	0.49	29.71	191	821	6.73	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
45.2	0.79	27.46	317	13	5.25	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
42.13	0.82	34.86	887	653	4.11	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
19.47	0.01	27.8	505	235	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
23.49	0.07	33.02	376	171	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
19.6	0.35	47.69	973	591	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
21.66	0.89	42.92	639	427	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
27.72	0.24	42.14	487	995	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
23.53	0.92	39.56	330	700	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
19.45	0.76	46.04	201	740	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
28.76	0.18	38.28	72	899	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
23.99	0.29	28.45	81	366	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
20.91	0.26	36.63	840	308	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
27.17	0.62	46.24	784	573	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
25.5	0.72	26.42	706	914	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
17.84	0.29	38.18	284	204	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
22.97	0.07	42.08	596	431	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
29.25	0.17	40.39	98	81	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
17.36	0.56	46.43	54	272	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
28.93	0.59	29.51	473	900	0	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
21.98	0.71	46.9	425	742	2.15	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
22.28	0.64	36.08	510	416	2.21	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
25.21	0.45	29.56	973	716	1.33	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
16.87	0.32	30.81	478	755	1.57	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
21.18	0.25	32.08	679	225	1.38	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
18.6	0.76	35.42	685	123	2.14	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
20.91	0.92	45.49	893	998	2.13	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
15.42	0.05	33.95	405	154	2.06	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
22.92	0.09	30.68	738	662	2.1	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
23.84	0.61	41.29	287	869	2.27	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00
17.64	0.89	38.24	969	117	2.05	PAP 03	Avenida Beira Rio	-10.71	-48.42	01/01/0001 00:00:00

Conforme é possível observar, o arquivo gerado possui, além das seis grandezas, as seguintes informações: nome, endereço, latitude, longitude e o momento em que a leitura foi realizada. Com esse arquivo, portanto, é possível que o usuário trate os dados da forma que precisar, podendo ser tanto para uso próprio como para alguma pesquisa ou mesmo empreendimento.

7.3 Protótipo de Testes

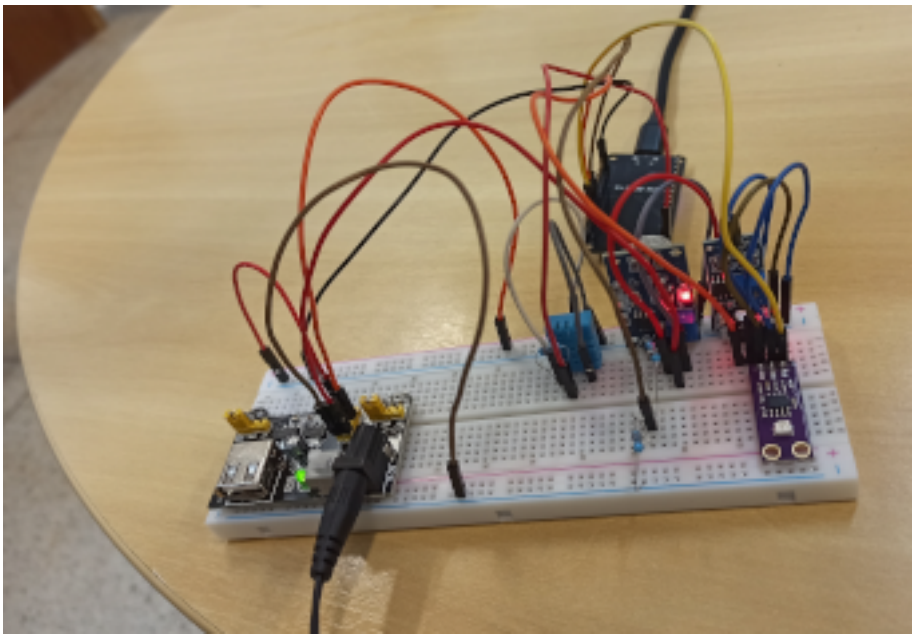
Para a realização dos testes e, portanto, confecção do protótipo, os materiais. com os respectivos preços, estão listados na Tabela 9.

Tabela 9 – Materiais e Preços do Protótipo

Material	Valor
1x Sensor Guva S12SD	R$ 33,90
1x Sensor MQ-135	R$ 21,90
1x Sensor de Chama IR	R$ 9,90
1x Kit Protoboard Intermediate	R$ 59,90
1x Módulo ESP32	R$ 69,90
1x Cabo Micro USB 2m	R$ 18,90
1x Sensor DHT11	R$ 13,90
Subtotal	R$ 228,30
Frete	R$ 66,55
Total	R$ 294,85

Os primeiros testes com o protótipo foram feitos em locais fechados e controlados, os quais contavam com a presença de outros sensores de referência. Os dois locais escolhidos foram a Prefeitura Municipal de Porto Nacional e a Universidade Federal do Tocantins (UFT). A Figura 56 ilustra um dos testes realizados na UFT.

Figura 56 – Teste Inicial na UFT



Após essa etapa, a célula foi levada para testes em local aberto dentro da UFT. Por último, a performance do equipamento foi monitorada no pátio da prefeitura. Vale salientar ainda que foi testado a reação da célula ao desligar a conexão com a rede momentaneamente. A Figura 57 mostra o teste no pátio, enquanto a Figura 58 mostra as mensagens de *debugging* de uma célula.

Figura 57 – Teste no Pátio da Prefeitura

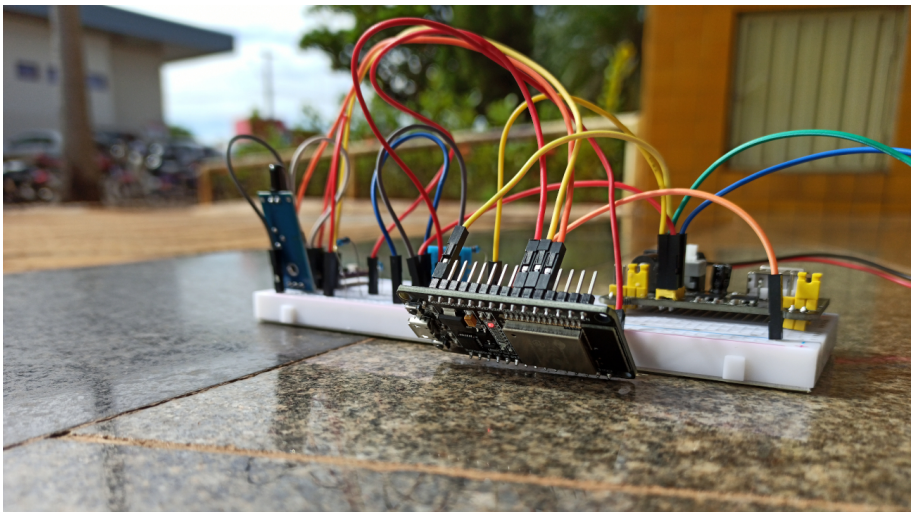


Figura 58 – Monitor de Debugging

COM3

```
Amostra coletada com sucesso: 9/20.
Amostra coletada com sucesso: 10/20.
Amostra coletada com sucesso: 11/20.
Amostra coletada com sucesso: 12/20.
Amostra coletada com sucesso: 13/20.
Amostra coletada com sucesso: 14/20.
Amostra coletada com sucesso: 15/20.
Amostra coletada com sucesso: 16/20.
Amostra coletada com sucesso: 17/20.
Amostra coletada com sucesso: 18/20.
Amostra coletada com sucesso: 19/20.
Amostra coletada com sucesso: 20/20.
Temperatura: 29.22 °C
Umidade: 64.35 %
Radiação UV: Índice 0
Fumaça: 232
Gás Tóxico: 229
Fogo: 0
Nova requisição atendida com sucesso.
Última requisição foi atendida.
Última requisição foi atendida.
```

8 RESULTADOS E DISCUSSÕES

Conforme foi possível observar anteriormente, com a aplicação para administração os servidores públicos responsáveis pelo sistema poderão gerenciar as células e agendar monitoramentos avulsos. A aplicação para visitantes, por sua vez, permite que os cidadãos acompanhem as medidas conforme sua necessidade. Por fim, a API pública concede os dados brutos para casos de usos mais específicos.

Em relação aos testes, a célula se comportou dentro do esperado nas situações, conseguindo enviar as leituras corretamente para o servidor e, no caso da conexão desligada intencionalmente, ela conseguiu se reconectar. Dito isto, é necessário tecer alguns comentários sobre o sistema.

8.1 Eficiência

Em termos de consumo de dados, o padrão REST possibilita que haja eficiência através de suas restrições. Uma vez que cada recurso recebe apenas dados em conformidade com as regras programadas, evita-se o desperdício com informações desnecessárias.

O consumo de energia elétrica é algo a ser comentado também, pois o fato de haver um sensor operando em 5 V acarreta no uso de componentes adicionais que aumentam a energia necessária. Portanto, a partir desta premissa, é interessante buscar alternativas para que o sistema trabalhe em níveis regulares de tensão.

Outro ponto a ser abordado aqui é o fato de o ESP 32 DevKit V1 ser um kit de desenvolvimento e, portanto, carrega consigo componentes que não são necessários em ambiente de produção. Apenas como exemplo, o conversor de interface serial para USB é desnecessário uma vez que o microcontrolador esteja programado. O ideal, nesse cenário, é embarcar apenas o módulo responsável pela comunicação sem fio e ter um circuito de programação separado.

8.2 Escalabilidade

A capacidade de um sistema crescer é uma métrica importante. Do ponto de vista de *software*, uma arquitetura RESTful permite a separação de funções em serviços, facilitando a confecção de sistemas distribuídos. Em outras palavras, o modelo de contêineres facilita o crescimento de uma rede de sensores sem fio.

Do ponto de vista do *hardware*, o tópico dos componentes desnecessários dificulta a escalabilidade em virtude do preço. Nesse caso, reitera-se o que foi dito anteriormente: é importante enxugar o sistema quando o mesmo for para um ambiente de produção.

8.3 Resiliência

A resiliência de um sistema informa o quanto ele é tolerante a falhas. Do ponto de vista de *software*, a falha de um sensor não compromete o restante da rede por causa da comunicação *single hop*. Nesse sentido, como o servidor é um nó central, é necessário uma política de *backup* e continuidade de serviço para ele.

No tocante ao *hardware*, faz-se necessário elaborar um invólucro protetor para mitigar os efeitos das intempéries climáticas. Outro ponto importante é assegurar a proteção elétrica das células oferecidas equipando as fontes de alimentação com circuitos protetores.

8.4 Confiabilidade

A parte de *software* não faz análise de dados procurando por valores fora do padrão, logo, a confiabilidade dos dados lidos fica a mercê da própria célula. Dito isto, a célula realiza várias medições e envia para o servidor uma média dos valores. Essa medida consegue suavizar eventuais discrepâncias em virtude de ruídos ou perturbações. Contudo, o sistema ainda não é capaz de verificar se um sensor está com defeito.

8.5 Segurança

O *hardware* está isento da responsabilidade com segurança, ficando essa responsabilidade a cargo do *software*. Como medidas iniciais, a API conta com um sistema de login, onde as credenciais estão gravadas no microcontrolador da célula. Além disso, o servidor conta com um *firewall* para filtrar as conexões e evitar ataques externos. Por fim, vale dizer que esse protótipo conta com um certificado de segurança válido, possibilitando, portanto, o envio de dados criptografados.

8.6 Interação com o Usuário

Naturalmente, não é esperado que o usuário interaja com o *hardware* do sistema, logo, esse tópico também é de responsabilidade integral do *software*. Nesse sentido, o padrão REST também é vantajoso, pois oferece a possibilidade de sistemas distribuídos. Dito de outra forma, os serviços para enviar avisos ou interfaces de acesso ficam separados da API, consumindo-a da mesma forma que a célula faz. Essa separação permite construir aplicativos (para qualquer dispositivo ou online) sem precisar alterar a API.

8.7 Sugestões de Trabalhos Futuros

Em virtude das ponderações anteriores, é possível fazer algumas sugestões sobre trabalhos futuros com o intuito de aprimorar o projeto.

O primeiro ponto é sobre a confecção da placa de circuito impresso. Uma vez construída essa placa, e por consequência, retirados os circuitos secundários que compõem o kit, os gargalos nos quesitos eficiência e escalabilidade serão removidos.

Há ainda um segundo ponto dentro da escalabilidade, que é migrar a arquitetura do sistema para microsserviços. Embora não tenha sido formalmente definido no presente trabalho, esse padrão é indicado para sistemas distribuídos para internet das coisas, como é o caso desse projeto. Fazendo isso, há ganho tanto na expansão no sistema quanto no provisionamento de novas máquinas para compor um cluster.

Adentrando na resiliência, além de construir o sistema de modo é ser mais resistente a erro, é importante ser rapidamente notificado quando uma falha ocorrer. No caso das células, é possível usar ferramentas de monitoramento de rede como o Zabbix. Já as APIs, por exemplo, poderiam ser monitoradas usando um padrão conhecido como *Health Check*, que também não foi definido nesse trabalho.

No quesito confiabilidade, seria interessante usar técnicas de *Machine Learning* para detectar erros, tanto no circuito quanto nas medições. A título de exemplificação, uma temperatura de 0 ^{o}C em Porto Nacional é um forte indicativo de que o sensor em questão está com defeito.

Acrescenta-se também uma sugestão no quesito segurança. É uma atitude prudente, além da necessidade de login das células, filtrar o acesso à API de monitoramento por endereço MAC. Uma vez que é possível acessar o IPTables com .NET, regras no *firewall* podem ser adicionadas de acordo com a inserção de novas células.

Por fim, salienta-se que a interação com o usuário pode ser melhorada. Uma vez que algumas prefeituras possuem seus próprios aplicativos, a API pública pode ser usada para incorporar os dados das medições nesse aplicativo, onde, dessa forma, os cidadão poderão checar as informações quando e onde quiserem.

9 CONCLUSÃO

O presente trabalho abordou a implementação de uma rede de sensores sem fio para monitoramento ambiental seguindo três eixos: os servidores, os quais englobam a funcionalidade da rede, a infraestrutura, composta pelos ICTs, células e os equipamentos da Cidade Digital, e as interfaces de interação com o usuário.

Acerca das células, é necessário destacar que as modificações sugeridas não inviabilizam o sistema caso não sejam implementadas, porém, conforme já foi dito, são alterações que otimizam o custo-benefício do projeto, fazendo uso melhor, portanto, da verba proveniente dos impostos.

Continuando as pontuações sobre a infraestrutura, é importante frisar que o uso da rede proveniente do programa Cidades Digitais consegue padronizar a implantação da RSSF, afinal, os equipamentos similares permitem reaproveitar a maior parte do sistema de monitoramento caso haja a decisão de levá-lo para outras cidades. Além disso, o uso da Cidade Digital vai de encontro com os objetivos do programa, pois, umas das pautas é justamente auxiliar a gestão pública em prol da sociedade. Dito de outro modo, há uma relação mutualista entre o programa Cidades Digitais e a rede proposta nesse trabalho, maximizando, portanto, o impacto que ambas as iniciativas podem trazer para população.

Um outro ponto a ser destacado ainda acerca do compartilhamento de infraestrutura citado acima é sobre o desacoplamento, algo importante para sistemas. Apesar de o presente trabalho ter sido formulado já pensando no apoio do programa Cidades Digitais, a construção do *hardware* e do *software* foi feita de modo independente, pois, caso seja necessário mudar a infraestrutura no futuro, o custo de modificação do projeto será reduzido porque essa alteração não se propagará por todo sistema e, dessa forma, não onerará todas as partes implementadas. Logo, afirma-se que esse desacoplamento contribui para a eficiência de custos atrelados à RSSF, pois a torna mais flexível.

Passando para o eixo dos servidores, é importante mencionar as vantagens trazidas pelo uso de contêineres Docker. Conforme foi exibido, os contêineres criam uma camada de isolamento entre os serviços, possibilitando que sejam tratados de forma individual e, dessa forma, sem impactar todo o sistema. Além disso, o uso do Docker cria a possibilidade de empregar *clusters*, e, com isso, possibilita que os servidores sejam entregues em computadores simples e, quando for necessário, novas máquinas podem simplesmente ser adicionadas para aumentar o poder computacional. Essa característica reduz o custo inicial de implementação do projeto, pois, é possível começar com uma pequena parte e depois ir expandindo conforme as condições forem mais favoráveis.

Outro tópico que vale a pena ser discutido é sobre os servidores de apoio para gerenciamento da RSSF: o de *backup* e a VPN. Embora esses servidores não incrementem as

funcionalidades da rede de sensores sem fio, eles proporcionam maior segurança, com redundância nos dados e criptografia, e reduzem custos com manutenção, afinal, os administradores podem operar remotamente em vez de se locomoverem para um local específico.

Trazendo o assunto para as APIs, em particular, sobre a encarregada de receber os dados células, é interessante mencionar que o fato de ela já receber os valores das grandezas em vez do nível de tensão protege a API das modificações na célula, afinal, caso as mudanças propostas no circuito sejam de fato efetivadas, o sistema permanecerá indiferente a essa alteração e, novamente, isso o torna mais maleável e eficiente.

Ainda no âmbito das APIs, é necessário pontuar os padrões abertos comentados nos capítulos desse trabalho. O uso de tecnologias abertas como um todo trazem uma série de vantagens: segurança, custos diluídos, flexibilidade e possibilidade de serem auditadas. Ao usar um padrão aberto já proposto no meio científico, a RSSF possui seu alicerce feito de forma já testada e aprovada por outros profissionais.

A presença de desacoplamento, distribuição e independência, naturalmente, trazem consigo complexidade para o processo de entrega do sistema. É por causa desse aspecto que foi criada, como contrapeso, uma *pipeline CI/CD*. Além do ganho direto que a automação do processo de entrega traz, há a vantagem indireta de que, caso os serviços não estejam em conformidade com as etapas de construção, a *pipeline* não será concluída, alertando a equipe que algo está errado antes mesmo de a RSSF ir para produção.

Sumarizando essas conclusões parciais e levando em consideração os resultados apresentados, portanto, é possível afirmar que o sistema cumpre o que foi proposto: realizar monitoramento ambiental tendo com base a infraestrutura do programa Cidades Digitais. Além disso, foram levadas em conta diversas questões de desacoplamento e independência para tornar o projeto o mais flexível possível e, portanto, adaptável para outros escopos além do originalmente pensando pelo autor.

REFERÊNCIAS

Adafruit. *Analog UV Light Sensor Breakout - GUVA-S12SD*. [S.d]. Https://www.adafruit.com/product/1918. Acesso em: 03 janeiro 2021. Citado 2 vezes nas páginas 29 e 30.

AHLGREN, B.; HIDELL, M.; NGAI, E. Internet of things for smart cities: Interoperability and open data. IEEE Computer Society, 2016. Citado 3 vezes nas páginas 57, 58 e 60.

AKYILDIZ, I. F. et al. Wireless sensor networks: a survey. *ScienceDirect*, 2002. Citado 2 vezes nas páginas 13 e 57.

ALBERT, R.; BARABáSI, A.-L. Statistical mechanics of complex networks. *Reviews of Modern Physics, v.24*, 2002. Citado na página 58.

ALEXANDER, C.; SADIKU, M. *Fundamentos de Circuitos Elétricos*. 5. ed. [S.l.]: AMGH, 2013. ISBN 9788580551730. Citado 3 vezes nas páginas 16, 17 e 18.

ALURA. *O Modelo OSI e Suas Camadas*. 2018. Https://www.alura.com.br/artigos/conhecendo-o-modelo-osi. Acesso em: 09 maio 2020. Citado na página 36.

ANTHOPOULOS, L. *Understanding Smart Cities: A Tool for Smart Government or an Industrial Trick?* [S.l.: s.n.], 2017. v. 22. Citado 5 vezes nas páginas 12, 52, 53, 54 e 55.

Aosong. *Módulo de sensor capacitivo de temperatura e umidade DHT11*. [S.d]. Http://aosong.com/en/products-21.html. Acesso em: 03 janeiro 2021. Citado na página 29.

ARIANE, G. *O que é SSL / TLS e HTTPS?* 2020. Https://www.hostinger.com.br/tutoriais/o-que-e-ssl-tls-https/. Acesso em 09 janeiro 2021. Citado 2 vezes nas páginas 45 e 47.

BETTS, R. *Architecting for the Internet of Things*: Making the most of the convergence of big data, fast data, and cloud. 2. ed. [S.l.]: O'Reilly, 2016. ISBN 9781491965412. Citado na página 57.

BINDAL, A. *Electronics for Embedded Systems*. [S.l.]: Springer, 2017. ISBN 9783319394374. Citado 7 vezes nas páginas 13, 17, 18, 22, 23, 26 e 27.

BIRON, J.; FOLLETT, J. *Foundational Elements of an IoT Solution*: The edge, the cloud, and application development. [S.l.]: O'Reilly, 2016. ISBN 9781491950975. Citado na página 56.

BOYLESTAD, R. L. *Introductory Circuit Analysis*. 11. ed. [S.l.]: Pearson, 2007. ISBN 0131730444. Citado 5 vezes nas páginas 16, 17, 18, 19 e 20.

BOYLESTAD, R. L.; NASHELSKY, L. *Electronic Devices and Circuit Theory*. 11. ed. [S.l.]: Pearson, 2013. ISBN 9788564574212. Citado 5 vezes nas páginas 13, 19, 20, 21 e 22.

BRITO, S. H. B. *Serviços de Rede em Servidores Linux*. São Paulo, SP: Novatec, 2017. ISBN: 978-85-7522-620-9. Citado 6 vezes nas páginas 40, 41, 43, 44, 47 e 50.

BROWN, L.; STALLINGS, W. *Seguraça de Computadores: Teoria e Prática*. 2. ed. [S.l.]: Pearson, 2017. ISBN 9780134794105. Citado na página 44.

CEDILLO-ELIAS, E. J. et al. Smart government infrastructure based in sdn networks: the case of guadalajara metropolitan area. IEEE, 2018. Acesso em: 24 maio 2020. Citado na página 55.

COMER, D. E. *Redes de Computadores e Internet*. 6. ed. Porto Alegre, RS: Bookman, 2016. ISBN: 978-85-8260-373-4. Citado 11 vezes nas páginas 12, 37, 38, 39, 40, 41, 43, 44, 45, 49 e 50.

DEJONGHE, D. *Nginx Cookbook*. 2. ed. [S.l.]: O'Reilly, 2019. Citado na página 74.

DIGITAL OCEAN. *SSH Essentials: Working with SSH Servers, Clients, and Keys*. 2014. Https://www.digitalocean.com/community/tutorials/ssh-essentials-working-with-ssh-servers-clients-and-keys. Acesso em: 18 maio 2020. Citado na página 45.

DIGITAL OCEAN. *Como configurar um Servidor VPN IKEv2 com o StrongSwan no Ubuntu 18.04*. 2019. Https://www.digitalocean.com/community/tutorials/how-to-set-up-an-ikev2-vpn-server-with-strongswan-on-ubuntu-18-04-2-pt. Acesso em: 23 maio 2020. Citado na página 50.

DIÁRIO OFICIAL DA UNIÃO. Portaria n 376 de 19 de agosto de 2011. *Imprensa Nacional*, 2011. Citado 2 vezes nas páginas 12 e 60.

DIÁRIO OFICIAL DA UNIÃO. Portaria n 186 de 28 de março de 2012. *Imprensa Nacional*, 2012. Citado 2 vezes nas páginas 12 e 60.

DOCKER. *Why Docker?* [S.d]. Https://www.docker.com/resources/what-container. Acesso em: 25 fev. 2020. Citado na página 62.

ESPRESSIF SYSTEMS. *ESP32 Technical Reference Manual*. 2019. Https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf. Acesso em: 28 maio 2020. Citado 2 vezes nas páginas 27 e 28.

FARIA, H. *Bacula: O Software Livre de Backup*. 3. ed. São Paulo - SP: Brasport, 2017. ISBN 9788574528526. Citado 2 vezes nas páginas 79 e 80.

FERNANDEZ, M. P. *Arquitetura de Computadores*. 3. ed. [S.l.]: UECE, 2015. ISBN 9788578264123. Citado 4 vezes nas páginas 24, 25, 26 e 27.

FOUNOUN, A.; HAYAR, A. Evaluation of the concept of the smart city through local regulation and the importance of local initiative. IEEE, 2018. Acesso em: 24 maior 2020. Citado 3 vezes nas páginas 12, 52 e 53.

GALLABA, K. *Improving the Robustness and Efficiency of Continuous Integration and Deployment*. 2019. Https://ieeexplore-ieee-org.ez6.periodicos.capes.gov.br/stamp/stamp.jsp?tp=&arnumber=8919213. Acesso em: 17 maio 2020. Citado na página 64.

GITLAB DOCS. [S.d]. Https://docs.gitlab.com/ee/ci/introduction/. Acesso em: 17 maio 2020. Citado na página 76.

Hanwei Electronics. *MQ-135 Gas Sensor*. [S.d]. Https://www.electronicoscaldas.com/datasheet/MQ-135_Hanwei.pdf. Acesso em: 04 janeiro 2021. Citado 2 vezes nas páginas 30 e 32.

HAYAR, A.; BETIS, G. Frugal social sustainable collaborative smart city casablanca paving the way towards building new concept for “future smart cities by and for all”. IEEE, 2017. Acesso em: 25 maio 2020. Citado 2 vezes nas páginas 12 e 54.

HAYT, W. H.; BUCK, J. A. *Eletromagnetismo*. 8. ed. [S.l.]: AMGH, 2013. ISBN 9788580551549. Citado na página 16.

JAYANTHI, J. G.; RABARA, S. A. Ipv4 addressing architecture in ipv6 network. *IEEE*, 2010. Acesso em: 08 maio 2020. Citado na página 39.

KUMAR, N. M.; GOEL, S.; MALLICK, P. K. Smart cities in india: Features, policies, current status, and challenges. IEEE International Conference on Technologies for Smart-City Energy Security and Power, 2018. Acesso em: 25 maio 2020. Citado na página 56.

LI, Y. et al. *IEEE*, 2011. Acesso em: 09 maio 2020. Citado 2 vezes nas páginas 36 e 37.

LIANG, X.; SHETTY, S.; TOSH, D. Exploring the attack surfaces in blockchain enabled smart cities. IEEE, 2018. Acesso em: 25 maio 2020. Citado na página 55.

MARTIN, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. [S.l.]: Pearson Education, 2009. Citado 3 vezes nas páginas 67, 69 e 70.

MARTIN, R. C. *Arquitetura Limpa: O Guia do Artesão para Estrutura e Design de Software*. [S.l.]: Alta Books, 2020. Citado 5 vezes nas páginas 67, 68, 70, 71 e 72.

MICROSOFT. *Introdução ao Identity no ASP.NET Core*. 2020. Https://docs.microsoft.com/pt-br/aspnet/core/security/authentication/identity?view=aspnetcore-5.0z&tabs=visual-studio. Acesso em 08 janeiro 2021. Citado na página 73.

MICROSOFT. *O que é o PowerShell?* 2020. Https://docs.microsoft.com/pt-br/powershell/scripting/overview?view=powershell-7.1. Acesso em 25 janeiro 2021. Citado na página 78.

MICROSOFT. *What’s new in .NET 5*. 2020. Https://docs.microsoft.com/pt-br/dotnet/core/dotnet-five. Acesso em 14 janeiro 2021. Citado na página 75.

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÃO E COMUNICAÇÕES. *Cidades Digitais*. [S.d]. Https://www.mctic.gov.br/mctic/opencms/indicadores/detalhe/Cidades-Digitais-Lista-de-Cidades-Atendidas-2.html. Acesso em: 24 fev. 2020. Citado na página 60.

MINISTÉRIO DO PLANEJAMENTO. *Cidades Digitais*. [S.d]. Http://pac.gov.br/infraestrutura-social-e-urbana/cidades-digitais. Acesso em: 25 fev. 2020. Citado na página 60.

MORA, O. B. et al. A use case in cybersecurity based in blockchain to deal with the security and privacy of citizens and smart cities cyberinfrastructures. IEEE, 2018. Acesso em: 24 maio 2020. Citado 3 vezes nas páginas 12, 52 e 58.

MORENO, D. *Introdução ao Pentest*. 2. ed. São Paulo - SP: Novatec, 2019. ISBN 9788575228074. Citado na página 45.

NETTO, G. T. *Redes de Sensores Sem Fio: Revisão*. 2016. Http://netto.ufpel.edu.br/phd/lib/exe/fetch.php?media=survey.pdf. Acesso em: 25 fev. 2020. Citado 3 vezes nas páginas 13, 57 e 58.

NGINX. *What is NGINX?* [S.d]. Https://www.nginx.com/resources/glossary/nginx/. Acesso em 08 janeiro 2021. Citado na página 74.

NOAL, L. A. J. *Linux para Linuxers: Do Desktop ao Datacenter*. São Paulo - SP: [s.n.], 2015. ISBN 9788575224724. Citado na página 45.

NOGUEIRA, A. F. et al. 2018. Https://ieeexplore-ieee-org.ez6.periodicos.capes.gov.br/stamp/stamp.jsp?tp=&arnumber=8590203. Acesso em: 17 maio 2020. Citado na página 64.

PLATFORM IO. *DOIT ESP 32 DevKit V1*. [S.d. Https://docs.platformio.org/en/latest/boards/espressif32/esp32doit-devkit-v1.html. Acesso em: 28 maio 2020. Citado 2 vezes nas páginas 27 e 28.

PREFEITURA DE PORTO NACIONAL. *Porto Nacional Se Torna Cidade Digital Com O Maior Investimento Feito Pelo Ministério*. 2017. Http://www.portonacional.to.gov.br/index.php/noticias/sec-de-comunicacao/ 993-porto-nacional-se-torna-cidade-digital-com-o-maior-investimento-feito-pelo-ministerio. Acesso em: 25 fev. 2020. Citado 2 vezes nas páginas 12 e 60.

RED HAT INC. *Integração e Entrega Contínuas: Pipeline CI/CD*. [S.d]. Https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd. Acesso em: 17 maio 2020. Citado 2 vezes nas páginas 64 e 66.

REKHTER, Y. et al. Address allocation for private internets. *IETF*, 1996. Acesso em 10 maio 2020. Citado 2 vezes nas páginas 40 e 41.

RESCA, S. *Hands-On RESTful Web Services with ASP.NET Core 3...* [S.l.]: Packt Publishing, 2019. ISBN: 978-1-78953-761-1. Citado 4 vezes nas páginas 43, 66, 67 e 74.

ROCHOL, J. *Comunicação de Dados*. Porto Alegre, RS: Bookman, 2012. (22). ISBN: 978-85-407-0037-6. Citado 2 vezes nas páginas 36 e 37.

Roithner LaserTechnik. *Guva-S12SD*. 2011. Https://cdn-shop.adafruit.com/datasheets/1918guva.pdf. Acesso em: 03 janeiro 2021. Citado na página 30.

SANTOS, J. C. dos. *Radiação Ultravioleta: Estudo Sobre Índices de Radiação...* [S.l.]: Universidade Estadual de Santa Cruz, 2009. Http://www.uesc.br/centros/ctr/modulos/didatico/palestras/seminarios/sct_2009/sem_dia1_joao.pdf. Acesso em: 03 janeiro 2021. Citado na página 30.

SHAKELFORD, J. *Ciência dos Materiais*. 6. ed. [S.l.]: Pearson, 2008. ISBN 9788576051602. Citado na página 21.

SINGH, S.; SINGH, N. Internet of things(iot): Security challenges, business opportunities & reference architecture for e-commerce. *IEEE*, 2015. Acesso em: 08 maio 2020. Citado 2 vezes nas páginas 36 e 56.

SSH Communications Security, Inc. *Public Key authentication for SSH*. [S.d]. Https://www.ssh.com/ssh/public-key-authentication. Acesso em: 18 maio 2020. Citado na página 45.

TANEMBAUM, A. S. *Redes de Computadores*. 5. ed. São Paulo, SP: Pearson Prentice Hall, 2011. ISBN: 978-85-7605-924-0. Citado na página 39.

TOCCI, R.; WIDMER, N.; MOSS, G. *Sistemas Digitais: Princípios e Aplicações*. 12. ed. [S.l.]: Pearson, 2018. ISBN: 978-65-5011-052-9. Citado na página 37.

TOCCI, R. J.; WIDNER, N. S.; MOSS, G. L. *Sistemas Digitais*: Princípios e aplicações. 11. ed. [S.l.]: Pearson, 2011. ISBN 9788576059226. Citado 6 vezes nas páginas 13, 22, 23, 25, 26 e 27.

TOLCHA, Y. K. et al. Oliot-opencity: Open standard interoperable smart city platform. IEEE, 2018. Acesso em: 24 maio 2020. Citado 4 vezes nas páginas 55, 56, 57 e 60.

XING, L. Reliability in internet of things: Current status and future perspectives. *IEEE Xplore*, 2020. Acesso em: 08 maio 2020. Citado 2 vezes nas páginas 56 e 57.

XU, J.; LIU, Y. Research on ipv6 network construction and application in higher vocational colleges. *IEEE 4th Information Technology,Networking,Electronic and Automation Control Conference*, 2020. Acesso em: 08 maio 2020. Citado na página 39.

YAVARI, A. et al. Internet of things-based hydrocarbon sensing for real-time environmental monitoring. *IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019. Acesso em: 08 maio 2020. Citado 2 vezes nas páginas 13 e 57.

YLONEN, T. Ssh key management challenges and requirements. *IEEE*, 2019. Acesso em: 18 maio 2020. Citado na página 45.