

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DA PARAÍBA - CAMPUS CAMPINA GRANDE
CURSO SUPERIOR DE TECNOLOGIA EM TELEMÁTICA

HENRIQUE MARTINS MIRANDA

**SISTEMA DE MONITORAMENTO DE QUALIDADE DE
IMUNOBIOLÓGICOS NA CADEIA DE DISTRIBUIÇÃO E
ARMAZENAMENTO**

Campina Grande

2021

Henrique Martins Miranda

**Sistema de Monitoramento de Qualidade de
Imunobiológicos na Cadeia de Distribuição e
Armazenamento**

Monografia apresentada à Coordenação do
Curso de Telemática do IFPB - Campus Cam-
pina Grande, como requisito parcial para con-
clusão do curso de Tecnologia em Telemática.

Orientador: Paulo Ribeiro Lins Junior

Campina Grande
2021

Henrique Martins Miranda

Sistema de Monitoramento de Qualidade de Imunobiológicos na Cadeia de Distribuição e Armazenamento

Monografia apresentada à Coordenação do
Curso de Telemática do IFPB - Campus Cam-
pina Grande, como requisito parcial para con-
clusão do curso de Tecnologia em Telemática.

Paulo Ribeiro Lins Junior
Orientador

Professor
Membro da Banca

Professor
Membro da Banca

Campina Grande
2021

Dedico esse trabalho às perdas causadas pela pandemia da COVID-19.

Agradecimentos

Agradeço a minha família, amigos e minha namorada, Jennifer Regina, que me ajudaram e deram apoio na construção deste trabalho. Agradeço também a meu amigo Gledson Santos, que participou ativamente no desenvolvimento dos protótipos citados, aos participantes dos laboratórios de pesquisa Assert e GComPI, que fornecerem local e ferramentas para construção dos protótipos e tiram dúvidas de assuntos que não domino, e por fim, agradeço a todos os professores que tive ao longo do curso.

Resumo

O armazenamento é um dos elementos mais importantes da cadeia de distribuição de vacinas, principalmente pela sensibilidade delas às variações de temperatura, que podem ocasionar diminuição da sua eficácia. Considerando isso, existe uma necessidade de manter constante monitoramento dessa variável, a fim de garantir que o produto final venha a manter suas características originais e a eficiência esperada. Esse trabalho apresenta uma solução baseada em Internet das Coisas, usando comunicação sem fio de baixa potência, para monitorar a qualidade de vacinas, por meio das medidas de temperatura e umidade nos locais de armazenamento. Os dados coletados são organizados em um banco de dados, podendo ser acessados por sistemas decisórios com a finalidade de avaliar a qualidade da vacina antes de sua aplicação, evitando transtornos em decorrência de problemas de armazenamento.

Palavras-chaves: Monitoramento. Vacinas. IoT. LoRa.

Abstract

Storage is one of the most important components of the vaccine distribution chain, mainly due to its sensitivity to temperature variations, which can cause a decrease in its effectiveness. Considering this, there is a need to keep constant monitoring of this variable, to guarantee that the final product will maintain its original characteristics and the expected efficiency. This work presents a solution based on the Internet of Things, using low power wireless communication, to monitor the quality of vaccines, by measuring temperature and humidity in storage locations. The collected data are organized in a database, which can be accessed by decision systems to assess the quality of the vaccine before its application, avoiding problems due to storage problems.

Key-words: Monitoring. Vacaciones. IoT. LoRa.

Listas de ilustrações

Figura 1 – Representação da relação entre os dispositivos LoRa (Adaptada de [1]).	19
Figura 2 – Dimensões em milímetros do sensor DHT-22 (Retirado de [2]).	20
Figura 3 – Representação do modelo cliente/servidor (Autoral).	21
Figura 4 – Ciclo de eventos no Node (Autoral).	22
Figura 5 – Organização dos dados no InfluxDB (Autoral).	23
Figura 6 – Comparação entre o modelo de virtualização e modelo de containers (Adaptada de [3]).	24
Figura 7 – Representação da conexão entre os componentes principais do <i>end node</i> (autorl).	27
Figura 8 – Foto do primeiro protótipo (autorl).	29
Figura 9 – Foto do segundo protótipo (autorl).	30
Figura 10 – Esquemático do segundo protótipo (autorl).	30
Figura 11 – Descarga da bateria 18650 conforme a temperatura do ambiente (Adap- tada de [4]).	32
Figura 12 – Foto do ESP32 LoRa da Heltec Automation (autorl).	33
Figura 13 – Página de configuração do <i>gateway</i> (autorl).	33
Figura 14 – Medidas dentro do banco de dados InfluxDB (autorl).	36
Figura 15 – Modelo entidade relacionamento do PostgreSQL (autorl).	37
Figura 16 – Desenho das telas de entrar e cadastrar na aplicação (autorl).	38
Figura 17 – Distância aproximadas entre os dispositivos (Adaptada do Google Maps).	39
Figura 18 – Percentual de entrega dos pacotes em barras e indicador de intensidade do sinal recebido em linha (Autorl).	40
Figura 19 – Gráfico de consumo dos protótipos do <i>end node</i> (Autorl).	41
Figura 20 – Comparação do custo mensal de hospedagem do servidor (Autorl). . .	43
Figura 21 – Tela de carregamento, dashboard e perfil, respectivamente (Autorl). .	51
Figura 22 – Dispositivos do usuário cadastrados (Autorl).	51
Figura 23 – Dados referente a um dispositivo (Autorl).	52
Figura 24 – Registro de um <i>gateway</i> (Autorl).	52
Figura 25 – Registro de um <i>end-node</i> (Autorl).	52

Lista de tabelas

Tabela 1 – tensão de trabalho dos componentes principais do protótipo (autoral).	31
Tabela 2 – Custos referente as peças do segundo protótipo do <i>end node</i> (autoral).	42

Lista de abreviaturas e siglas

°C	Graus Celsius
APP	Application (Aplicativo móvel)
dBm	Decibel-milliwatt
DDD	Domain-Driven Design (Design Orientado por Domínio)
E/S	Entrada e Saída de dados
GPRS	General Packet Radio Services (Serviços Gerais de Pacote por Rádio)
HTTP	Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto)
IDE	Integrated Development Environment (Ambiente de Desenvolvimento Integrado)
IFPB	Instituto Federal da Paraíba
IoT	Internet of Things (Internet das Coisas)
IP	Internet Protocol (Protocolo de Internet)
JSON	JavaScript Object Notation
kOhms	KiloOhms
LCD	Liquid Cristal Display (Tela de Cristal Líquido)
LoRa	Long Range (Longo Alcance)
mA	MiliAmpére
mAh	Miliampére-hora
Mhz	Megahertz
ms	Milisegundos
PDR	Packet Delivery Ratio (Taxa de Entrega de Pacotes)
pF	Picofarad
PNI	Programa Nacional de Imunizações
R\$	Reais

RH	Resistência a Umidade
RSSI	Received Signal Strength Indication (Indicação de Força do Sinal Recebido)
s	Segundos
SMS	Short Message Service (Serviço de Mensagens Curtas)
SNR	Signal-to-Noise Ratio (Relação Sinal-Ruído)
SO	Sistema Operacional
TDD	Test-Driven Development (Desenvolvimento Orientado por Teste)
TI	Tecnologia em Informação
US\$	Dólares estadounidenses
UBS	Unidade Básicas de Saúde
Wi-Fi	Wireless Fidelity (Fidelidade sem Fio)

Sumário

1	INTRODUÇÃO	13
1.1	O Programa Nacional de Imunizações	13
1.2	Rede de Frio	14
1.3	Justificativa e Relevância do Trabalho	14
1.4	Objetivos	15
1.4.1	Objetivo Geral	15
1.4.2	Objetivos Específicos	16
1.5	Trabalhos Relacionados	16
2	FUNDAMENTAÇÃO TEÓRICA	18
2.1	Internet das Coisas	18
2.2	Sensor DHT-22	20
2.3	Servidor	20
2.3.1	Node.js	21
2.3.2	InfluxDB	22
2.3.3	Docker	23
2.3.4	Computação em Nuvem	24
2.4	Aplicativo Móvel	24
2.4.1	React Native	24
3	METODOLOGIA	26
3.1	Estrutura dos Pacotes	26
3.2	End Node	26
3.2.1	Primeiro Protótipo	28
3.2.2	Segundo Protótipo	29
3.2.3	Escolha da Bateria	31
3.3	Gateway	32
3.4	Servidor	34
3.4.1	Padrões de Projetos	34
3.4.2	Banco de Dados	35
3.4.3	Containers	37
3.4.4	Deploy	37
3.5	Aplicativo móvel	38
4	RESULTADOS	39
4.1	Análise de Transmissão dos Pacotes	39

4.2	Análise do Consumo Energético	40
4.3	Análise do Custo	42
5	CONSIDERAÇÕES FINAIS	45
5.1	Sugestões para Trabalhos Futuros	45
	REFERÊNCIAS	47
	APÊNDICES	50
	APÊNDICE A – DESENHO DAS TELAS DO APLICADO MÓVEL	51

1 Introdução

A saúde é um fator de suma importância para todos os seres vivos, ele é um problema científico, tecnológico, político, prático e filosófico que refere-se a um estado completo de bem estar físico, emocional, social, intelectual e espiritual [5].

Segundo o artigo 196 [6] da Constituição Federal Brasileira a saúde é um direito de todos e dever do Estado garantir medidas políticas sociais e econômicas que visam à diminuição do risco de doenças e de outros agravamentos e ao acesso universal e imparcial às ações e serviços para a sua promoção, proteção e recuperação.

Para garantirmos nossa saúde, precisamos cuidar do nosso corpo e mente. Para isto, uma ferramenta que podemos contar são os imunobiológicos, como as vacinas e os soros. Diferente de remédios que ajudam no tratamento de pessoas doentes, os imunobiológicos são uma preparação biológica que fornece imunidade total ou parcial de uma determinada doença autoimune para um indivíduo saudável. As vacinas e os soros se diferem pela sua forma de imunização, as vacinas fornecem uma imunização ativa, estimulando o nosso organismo na produção de anticorpos, os soros fornecem uma imunização passiva, provendo os anticorpos para o nosso organismo que foram produzidos em outros organismo [7].

Contudo, os imunobiológicos requerem um cuidado elevado para manter a qualidade e sua eficiência, um dos fatores é que são produtos termolábeis, ou seja, se deterioram após determinado tempo expostos a variações de temperaturas e umidade, portanto, é imprescindível assegurar que seu ambiente de armazenagem mantenha uma temperatura e umidade constante [8] para garantir uma longevidade maior para o produto. Para este propósito, existe a Rede de Frio, um processo desenvolvido pelo Programa Nacional de Imunizações, PNI, de conservação, armazenamento e transporte dos medicamentos, objetivando as condições adequadas dos mesmos, mantendo suas características iniciais [8].

No ano de 2014, foi relatado no estudo [9] que a qualidade de conservação das vacinas não eram adequadas em boa parte dos municípios da macrorregião Oeste de Minas Gerais, alguns dos motivos citados foram a má gestão dos refrigeradores, falhas no monitoramento da temperatura e insuficiência de recursos humanos.

1.1 O Programa Nacional de Imunizações

Com o sucesso da Campanha de Erradicação da Varíola, CEV, iniciada em 1965, tendo seu fim em 1973 [10], amplificou dentro do Ministério da Saúde maiores investimentos no controle de doenças autoimunes, dando um impulso na criação do PNI [11]. O PNI foi fundado com objetivo de controlar e erradicar as doenças imunopreveníveis, através de

ações metalizadas de vacinação da população. Em 1980 foi realizada a primeira campanha de vacinação da poliomielite e desde então foram realizadas diversas campanhas, tais como a da rubéola, sarampo, tuberculose, febre amarela [11, 8] e atualmente contra a COVID-19.

De acordo com a Lei n.º 6.259 de 30 de outubro de 1975, regularizada pelo Decreto nº 78.231 em 1976, certificar o PNI, sobre a responsabilidade do Ministério da Saúde e define as seguintes competências [8]:

- implantar e implementar as ações do Programa, relacionadas com as vacinações de caráter obrigatório;
- estabelecer critérios e prestar apoio técnico e financeiro à elaboração, implantação e implementação dos programas de vacinação a cargo das secretarias de saúde das unidades federadas;
- estabelecer normas básicas para a execução das vacinações;
- supervisionar, controlar e avaliar a execução das vacinações no território nacional, principalmente o desempenho dos órgãos das Secretarias de Saúde, encarregados dos programas de vacinação.

1.2 Rede de Frio

A Rede de Frio, também chamada de Cadeia de Frio é um processo definido pelo PNI designado a auxiliar os profissionais da área da saúde, responsáveis pela imunização no Brasil, para que possa assim, garantir a efetividade e durabilidade dos imunobiológicos e medicamentos termolábeis.

No Manual de Rede de Frio [8] são definidos os requisitos dos ambientes de armazenagem para garantir a efetividade dos produtos, desde os laboratórios produtores às instâncias locais, passando pela instância nacional, estadual, e no transporte entre eles. Para as Câmaras frigoríficas, a temperatura de operação é entre -20°C a +2°C, variando conforme o material armazenado. Para a maioria dos imunobiológicos o recomendado é de +2°C e +8°C para ter um melhor controle da sua validade, havendo algumas exceções, como por exemplo, as vacinas Pfizer-BioNTech e Moderna, produzidas para combater o COVID-19, que precisam ser armazenadas entre -80°C a -60°C e -25°C e -15°C, respectivamente [12].

1.3 Justificativa e Relevância do Trabalho

Atualmente, com o início da distribuição das vacinas contra o COVID-19 em todo o mundo, uma das dificuldades enfrentadas é o controle de qualidade no armazenamento

tanto em transporte [13], quanto no local da aplicação justamente por serem produtos sensíveis à temperatura e necessitarem de muita cautela. Em contrapartida, por ser um produto com uma demanda elevada, a sua oferta deve ser rápida para que haja imunização em massa da população, encaminhando-se para o fim da pandemia.

Esses desafios enfrentados na distribuição das vacinas do COVID-19 não são as únicas dificuldades para a garantia da qualidade dos imunobiológicos. Enfermeiras de Minas Gerais, com o objetivo de inteirar-se acerca do sistema de manutenção dos produtos, realizaram um estudo sobre a conservação de vacinas em Unidades Básicas de Saúde, UBSs. Nessa pesquisa foram relatadas diversas irregularidades no armazenamento dos materiais termolábeis sem o comprimento das normas da PNI, como por exemplo, a presença de vacinas que deveriam ter sido descartadas por terem atingido seu tempo máximo de diluição, ainda presentes nos refrigeradores. Cerca de 52% dos imunobiológicos armazenados nos refrigeradores eram acomodados erroneamente e 36% dos refrigeradores observados contavam com objetos em portas, como fracos vazios.

Analizando as temperaturas dos ambientes de armazenagem, foi observado que 4% das unidades não realizavam o registro da temperatura dos refrigeradores, 88% dos refrigeradores usavam termômetros analógicos de baixa confiabilidade e cerca de 12% estavam com temperatura abaixo da faixa recomendada, chegando a 0°C. Outros estudos realizado [9, 14, 15] também testemunharam essa irregularidade na temperatura em seus respectivos locais ao redor do mundo, valendo salientar o estudo realizado na Bolívia [14], que teve resultados ainda piores, sendo registado uma temperatura mínima de -7.2°C e uma máxima de 22.7°C.

Pensando nesse cenário, esse trabalho apresenta uma alternativa para melhorar a forma de monitoramento de temperatura e umidade realizada em produtos imunobiológicos por laboratórios, unidades públicas de saúde e afins, no intuito de auxiliá-los a manter os materiais em suas melhores condições.

1.4 Objetivos

1.4.1 Objetivo Geral

Construir uma solução baseada em conceitos de Internet das Coisas visando o monitoramento de temperatura e umidade de imunobiológicos para auxiliar funcionários da saúde, garantindo melhores condições para a vacinação da população frente a incidência de doenças.

1.4.2 Objetivos Específicos

- Construir um protótipo inicial para coleta da temperatura e umidade nos ambientes de armazenagens dos imunobiológicos.
- Implementar um servidor para a armazenagem dos dados coletados e posteriormente fornecer históricos das temperaturas e umidade ao aplicativo móvel.
- Desenvolver um aplicativo móvel para fornecer uma interface amigável para os usuários.
- Realizar testes e análises dos dados transmitidos a fim de garantir a confiabilidade das temperaturas e umidade coletadas.

1.5 Trabalhos Relacionados

Estudantes do Instituto Federal da Bahia desenvolveram uma pesquisa [16] sobre diferentes métodos de construção de um sensor de temperatura de baixo custo voltado para o monitoramento de vacinas, pois a temperatura é uma das variantes de grande importância no processo de conservação de materiais imunobiológicos.

No estudo [17], foi desenvolvido um produto de monitoramento de temperaturas de caixas térmicas usadas no transporte de materiais termolábeis, com o objetivo de alertar os transportadores de quando a condição de armazenagem estiver fora da faixa ideal, dispondo de sinais auditivos e visíveis. No seu protótipo foi utilizado uma tela LCD para visualizar a temperatura atual, um led vermelho e outro verde para informar visivelmente sobre a condição do ambiente, um sensor de temperatura DS18B20, para coletar os dados necessários e um Arduino Uno R3, como controlador dos equipamentos.

Em [18], um estudo similar ao citado posteriormente, foi construído um dispositivo de baixo custo para o monitoramento de temperaturas voltado para geladeiras domésticas que armazenam vacinas em municípios do Brasil de pequeno porte, pois possuem baixa quantidade de insumo que consequentemente, seus ambientes de armazenagem de materiais termolábeis são precários, possuindo poucas câmeras específicas para armazenagem deste tipo de material, que acabam utilizando geladeiras domésticas. Seu protótipo é composto por um Arduino conectado a uma célula de Peltier, refrigerado por ventoinhas acoplado em uma caixa de isopor, os dados da temperatura são coletados pelo Arduino e enviado para um computador onde pode ser visualizado por um web site e armazenado em uma planilha excel.

Na dissertação de Mestrado de Lopes Neto [19], foi desenvolvido uma plataforma em Arduino para transmissão de dados de temperatura do interior dos ambientes gerais

de conservação de materiais sensíveis à temperatura, transmitido via SMS e GPRS para uma central onde é armazenado os dados para consultas posteriores.

2 Fundamentação Teórica

Neste capítulo serão abordados os principais conceitos e tecnologias utilizadas no desenvolvimento deste projeto, do hardware à aplicação móvel, iniciando pelo conceito principal, a Internet das Coisas.

2.1 Internet das Coisas

Com o intuito de ampliar a internet atual, interligando os objetos do nosso cotidiano, animais e humanos em uma única rede, foi criado a Internet das Coisas, IoT, também conhecida como internet de todas as coisas. Para tal, os objetos viram objetos inteligentes, possuindo capacidade de comunicação associados com sensores que fornecem dados para outros dispositivos. Estes aparelhos, conectados a IoT, se comunicam via internet, trocando dados em tempo real, transmitindo informações acerca do ambiente que estão inseridos e/ou até mesmo dos seus respectivos estados.

Deste modo, é adicionada uma nova gama de possibilidades, trazendo grandes benefícios para ambientes domésticos, mas principalmente para a área industrial. No primeiro caso, aplicações, tais como: aprendizagem reforçada, monitoramento e vigilância inteligentes e vida assistida, têm despontado entre aquelas que mais chamam atenção, tanto dos usuários, como das empresas de desenvolvimento de soluções tecnológicas. No segundo caso, IoT se apresenta como um diferencial competitivo importante em campos, com a automação e manufatura industrial, logística, gestão de processos de negócio, entre outros.

Do ponto de vista da produtividade, IoT apresenta-se como um importante meio pelo qual pode-se desenvolver aplicações sofisticadas que podem integrar o mundo real e o mundo virtual. Em empresas de manufatura, produtos conectados permitem a existência de um ambiente de serviços e produtos no qual a manutenção pode ser realizada com base na necessidade real, em vez de uma suposição estatística. Adicionalmente, produtos e máquinas conectados podem receber atualizações de software quando disponíveis, para garantir que estejam sempre funcionando em eficiência ótima.

Entretanto, a IoT enfrenta grandes desafios na sua implantação, tanto na transmissão de dados quanto na autonomia energética. No ambiente residencial, paredes e dispositivos eletrônicos se tornam obstáculos para os dispositivos da IoT, na área industrial, esses empecilhos são mais presentes, devido a grande concentração de equipamentos metálicos. Outros problemas, fora os causados pelo ambiente, são devido a grande concentração de tais dispositivos transmitindo simultaneamente, que podem acabar gerando interferência.

Ademais, há um grande número de dispositivos que não podem ficar conectados na energia elétrica por cabos, por razões variadas, e, graças a sua quantidade, é preciso que durem por um grande período de tempo com a mesma fonte de energia para tornarem-se viáveis, pois, gerenciar as baterias usadas por inúmeros dispositivos em uma grande frequência, acaba sendo custoso. Tendo esses pontos em mente, algumas tecnologias foram criadas para tal finalidade, entre elas uma das que mais se destacam é a LoRa.

A tecnologia Long Range, LoRa, é uma forma de comunicação sem fio, semelhante ao Wi-Fi e ao Bluetooth, que permite um longo alcance de comunicação com baixo custo. O raio de comunicação sem fio utilizando o LoRa, dependendo do dispositivo selecionado, pode alcançar quilômetros de distância.

Embora o LoRa tenha sido fundamentalmente desenvolvido pela Semtech Corporation, seu padrão imposto pelo LoRa permitiu que muitas empresas a utilizassem para diversos projetos com um custo benefício satisfatório, aumentando o ecossistema e ganhando um envolvimento significativamente maior, uma variedade superior de produtos e um acréscimo geral no uso e aceitação.

O LoRa em si diz respeito à camada física, sendo a camada lógica chamada de LoRaWAN, um protocolo usado pelo LoRa para comunicação entre pontos de conexões de um nós finais (*end nodes*) para envios de informações diretamente a um concentrador (*gateway*), que centraliza as informações e envia a um determinado sistema [20]. Os *end nodes* são dispositivos responsáveis por coletar os dados necessários e transmiti-los a um determinado *gateway*, o *gateway* por sua vez, é responsável por receber as informações de múltiplos *end nodes* e repassá-las a um determinado sistema, por exemplo um servidor onde os dados serão armazenados, como podemos ver na figura 1.

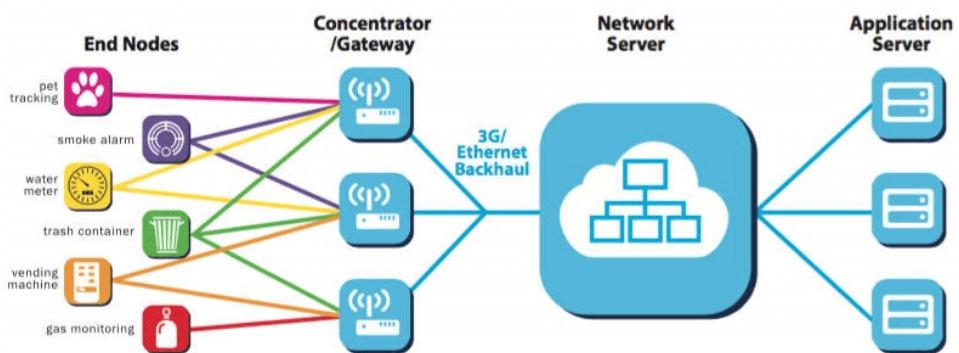


Figura 1 – Representação da relação entre os dispositivos LoRa (Adaptada de [1]).

Não apenas para o LoRa, mas para as comunicações sem fios em geral, existem parâmetros que ajudam a analisar a qualidade da transmissão dos dados, dentre eles o indicador de força do sinal recebido, RSSI, a relação sinal-ruído, SNR e a taxa de

entrega de pacotes, PDR, este última, não apenas nas comunicações sem fios, mas nas comunicações em gerais.

2.2 Sensor DHT-22

O sensor DHT-22 é um sensor de temperatura e umidade da família de sensores DHT comumente utilizado em aplicações de IoT por ser um sensor pequeno (como podemos ver na figura 2) e possuir um baixo custo. Ele opera na faixa de 3.3 a 6 Volts e consegue capturar dados de temperatura entre -40°C a 80°C e umidade entre 0% a 100% RH, com uma acurácia de 0.5°C e 2% RH, respectivamente [2].

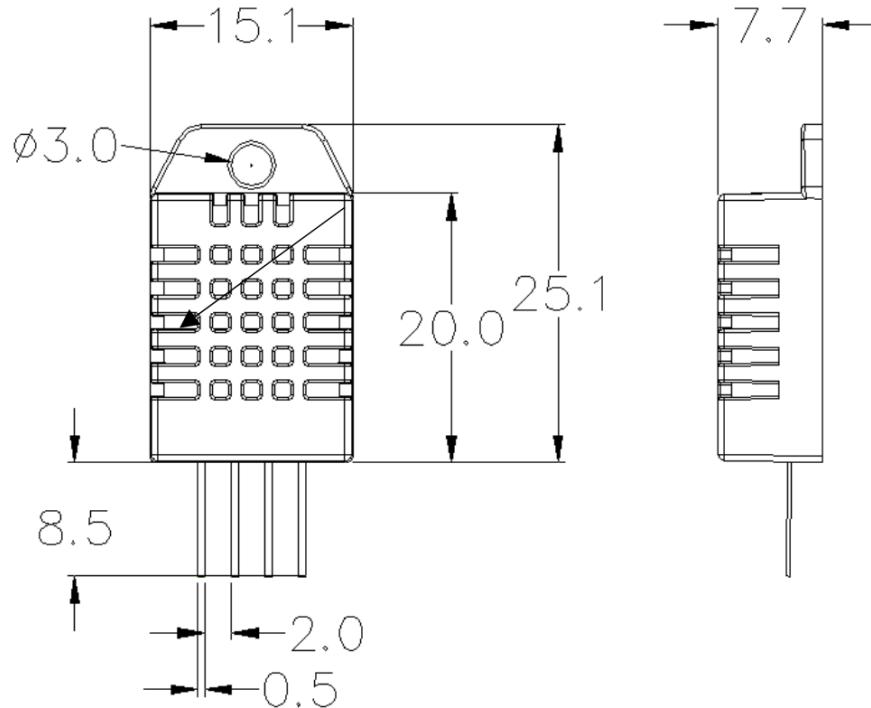


Figura 2 – Dimensões em milímetros do sensor DHT-22 (Retirado de [2]).

2.3 Servidor

Um servidor é um sistema de computação centralizada, responsável por fornecer serviços em uma rede de computadores. Esses serviços variam conforme a necessidade do sistema, podendo ser um controlador de domínio, provedor de arquivos, de impressão, de e-mails entre outros. No modelo cliente/servidor (figura 3), por sua vez, o servidor é responsável por receber as requisições de clientes, provendo informações e dados de acordo com a demanda.

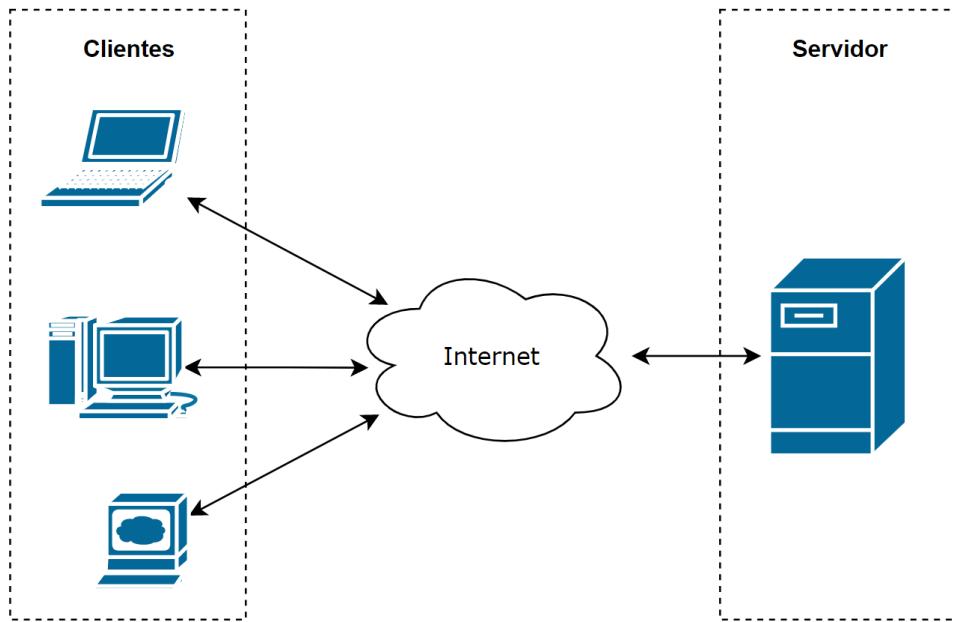


Figura 3 – Representação do modelo cliente/servidor (Autoral).

2.3.1 Node.js

Em 2009, Ryan Dahl criou o Node.js ou simplesmente Node, um ambiente de código aberto de execução Javascript para ser utilizado por servidores (*server-side*) baseado no interpretador V8 da Google. O objetivo do Node é fornecer uma forma de criar serviços com alta capacidade de escala, consumindo pouca memória e de fácil aprendizado. Apesar da linguagem de programação Javascript fornecer uma performance inferior às linguagens já existentes para esse objetivo, seu uso oferece dois grande benefícios, ser uma linguagem de fácil aprendizado e de grande uso por ser a linguagem padrão dos navegadores, e consequentemente do desenvolvimento web [21].

A forma de execução no Node é por eventos assíncronos, diferente da maioria dos outros ambientes modernos que utilizam *threads* do sistema operacional, SO. O Node usa apenas uma thread assíncrona principal que executa operações de entrada e saída de dados, E/S, chamada de *Event Loop*, em português, ciclo de eventos [22]. O *Event Loop* funciona em um ciclo, escutando uma lista de chamadas, onde são armazenadas as requisições recebidas, e as direcionando cada uma para uma *thread* a parte, que executará a função recebida, como podemos ver na figura 4. A priori, o Node possui quatro *threads* trabalhadoras, responsáveis por executar as funções. Entretanto, é possível configurar a quantidade de *threads* usadas, dependendo da máquina, onde está sendo efetuado a instância do Node.

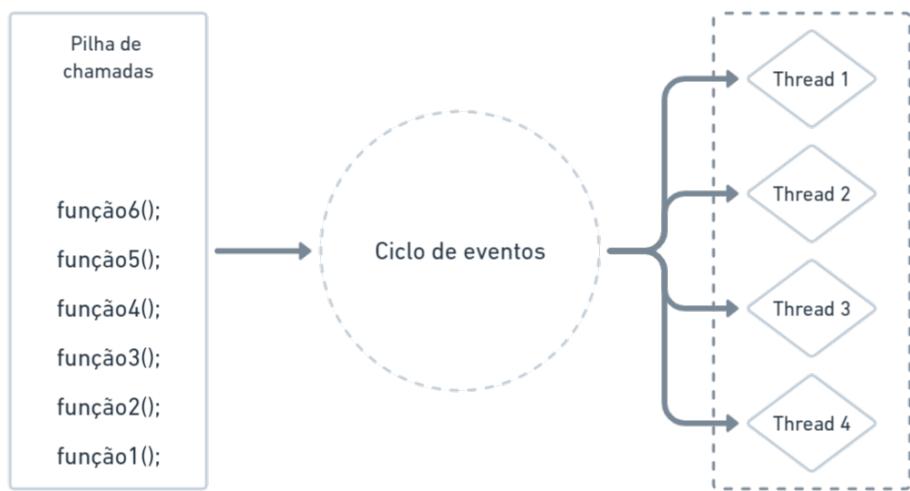


Figura 4 – Ciclo de eventos no Node (Aitoral).

2.3.2 InfluxDB

O InfluxDB é um banco de dados que armazena séries temporais (*data series database*), ou seja, sua chave é o tempo e sua forma de armazenagem de dados é em ordem cronológica. Ele foi projetado para lidar com grandes cargas de escrita e consulta, perfeito para armazenar dados em tempo real, como monitoramento de DevOps, métricas de aplicativos, big data e dados de sensores da IoT [23]. De forma geral, séries temporais acabam se tornando gráficos em função do tempo em um determinado período, por exemplo a temperatura de um freezer ao decorrer do dia, podendo assim ver facilmente a máxima, a mínima e suas variações. Esses dados podem ser também coletados e feito uma análise mais complexa usando qualquer ferramenta estatística, dependendo da sua necessidade.

O InfluxDB é composto por *databases* (banacos de dados), *measurements* (medidas), *fields* (campos) e *tags*. Podemos representar essa estrutura como conjuntos como podemos ver na figura 5. No InfluxDB é possível ter inúmeros *databases*, onde cada *database* contém suas *measurements* que são tabelas de dados correspondentes a algum dado em específico. Por exemplo, se tivermos 2 sensores que coletam dados diferentes, cada sensor viraria um *measurement*, e cada *measurement* é composto de dois tipos de atributos, os *fields*, onde ficam os dados da sua medida, e as *tags*, que são campos de dados que diferem *fields* por serem campos indexáveis, feitos exclusivamente para realizar buscas, tal como, é comum adicionar uma *tag* que seja um identificador do dispositivo que coletou esse medida.

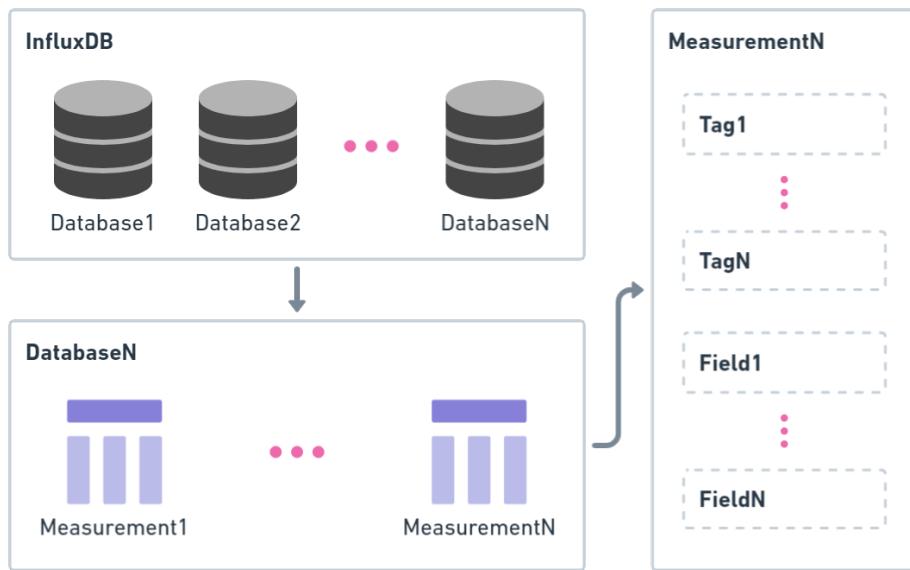


Figura 5 – Organização dos dados no InfluxDB (Autoral).

2.3.3 Docker

Docker é uma plataforma *open source*, desenvolvida utilizando a linguagem de programação GO pela Google, com o objetivo de criar facilmente ambientes isolados (containers) com um alto desempenho e portáteis, sendo uma opção em relação às virtualizações. Desta maneira, é possível, por exemplo, criar inúmeras aplicações usando as mesmas tecnologias, cada uma em um *container* diferente e nenhuma vai interferir na outra, e todas na mesma máquina. Se for preciso replicar em outra máquina, é possível criar uma imagem do container e instalar o mesmo ambiente nesta outra máquina.

Em comparação com as virtualizações, os containers não precisam de um sistema operacional, apenas o essencial para executar determinada função. Dessa forma, os containers conseguem ter um controle maior, consomem menos recursos, ganham uma maior flexibilidade e manutenibilidade. Podemos ver a comparação entre virtualização e containers na figura abaixo.

Apesar dessa facilidade que containers trazem consigo, gerenciar vários ao mesmo tempo pode ser bem trabalhoso, para isto, foram criados os orquestradores de containers, se baseando em orquestradores de orquestras sinfônicas, que rege o comportamento da sua banda durante uma determinada apresentação. Tais orquestradores auxiliam na organização dos containers, informando que devem iniciar primeiro, quais são as dependências de cada um, interligam suas redes se preciso, entre outras funcionalidades. Para o Docker, um dos orquestradores mais utilizados é o Docker Compose.

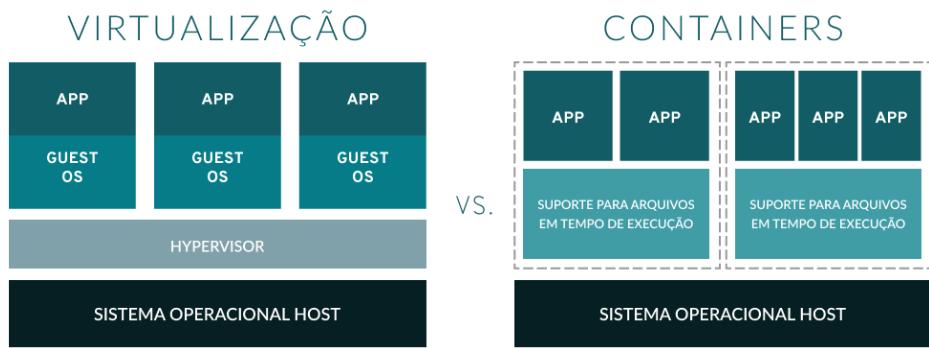


Figura 6 – Comparação entre o modelo de virtualização e modelo de containers (Adaptada de [3]).

2.3.4 Computação em Nuvem

Há uma crescente demanda por serviços de Tecnologia em Informação, TI, oferecidos através da Internet, incluindo armazenamento de dados, hospedagem de sites, softwares, banco de dados, rede entre outros. Este fornecimento de serviços de computação, via Internet, é chamado de computação em nuvem. Tal crescimento vem das diversas vantagens oferecidas pela computação em nuvem, como uma disponibilidade maior, recursos flexíveis, normalmente se pagando apenas o usado, trazendo assim, uma redução no custo, possibilidade de escalonamento de acordo com as necessidades da empresa, não precisar gastar nem se preocupar com a infraestrutura entre outras [24].

2.4 Aplicativo Móvel

Aplicativo móvel, APP, é um *software* desenvolvido especificamente para dispositivos móveis, como telefones celulares, tablets, relógios inteligentes entre outros. Cada SO utiliza uma determinada linguagem de programação para desenvolvedores poderem criar os apps. Com uma grande diversidade de sistemas operacionais, com linguagens de programação distintas, transformam o desenvolvimento de aplicativos multiplataformas mais trabalhoso e repetitivo, pois tem que ser programada uma versão diferente para cada plataforma. Tendo esse problema em mente, várias formas de programação híbridas foram criadas, com o princípio de poder programar utilizando apenas um código para várias plataformas distintas, dentre elas, as duas que mais se destacam atualmente são o Flutter e o React Native.

2.4.1 React Native

O React Native é um *framework* para desenvolvimento nativo voltado para dispositivos para os dois principais SO móveis atualmente, Android e iOS. Ele dispõe de

um código aberto, mantido pelo Facebook e sua comunidade. É um *framework* bastante popular, utilizado por grandes empresas globais atualmente, como Netflix, Uber, a própria Facebook e empresas brasileiras como o EBANX e a Globo [25].

O React Native utiliza a linguagem JavaScript como principal e a renderiza para código nativo, para isso é adicionado uma camada em JavaScript, chamada de *Bridge* que se comunica com o sistema operacional, mandando comandos para renderizar os componentes nativos [26].

3 Metodologia

O sistema proposto neste projeto, se refere a uma plataforma que fornece tanto os dispositivos para a captação dos dados dos imunobiológicos, quanto ao sistema em nuvem para armazenagem dos dados e de um aplicativo móvel para a gerência e análise das informações, seguindo a mesma estrutura representada na figura 1 na sessão 2.1, de modo a fornecer aos usuários, um produto de fácil uso para monitorização de temperatura e umidade, possibilitando o histórico dos dados coletados em um determinado cliente.

Podemos então, separar este sistema em quatro partes, cada uma delas com uma determinada função, elas são, *end node*, *gateway*, servidor e aplicativo móvel. Mas antes, para uma visão geral, vamos ver como foi estruturado o envio de pacotes entre cada etapa.

3.1 Estrutura dos Pacotes

Definir um padrão de transmissão dos pacotes é de suma importância para facilitar a identificação dos dados e ajuda no reconhecimento de possíveis erros que possam ocorrer na transmissão dos dados.

Dentre a comunicação do *gateway*, servidor e o aplicativo móvel, que utiliza o protocolo HTTP, existem vários padrões para transmitir dados já definidos e consolidados pela comunidade, um dos mais utilizados é o *JavaScript Object Notation*, JSON, um formato leve e de código aberto, se baseando na estrutura de objetos do JavaScript.

Entretanto, o JSON é considerado um formato leve para o HTTP e seu ecossistema, para o LoRa, que tem um limite bem menor de dados para ser transmitido, é preciso utilizar outro padrão. Para isto, optamos pelo simples, transmitir os dados separando-os pelo caractere ponto e vírgula, na seguinte ordem: temperatura, umidade, identificador do *end node* e o contador de pacotes transmitidos.

3.2 End Node

O *end node* é o hardware que fica dentro dos ambientes de armazenagem coletando os dados de temperatura e umidade e enviado para o *gateway*. O objetivo é que o *end node* seja *plug-and-play*, que precisaria apenas colocá-lo dentro do refrigerador juntos com os imunobiológicos sem precisar modificar o ambiente, como por exemplo, adicionar um cabo para suprir sua demanda energética. Para o melhor entendimento podemos separá-lo em três partes, o sensor, o transceptor LoRa e o microcontrolador, como pode ser visto na figura 7.

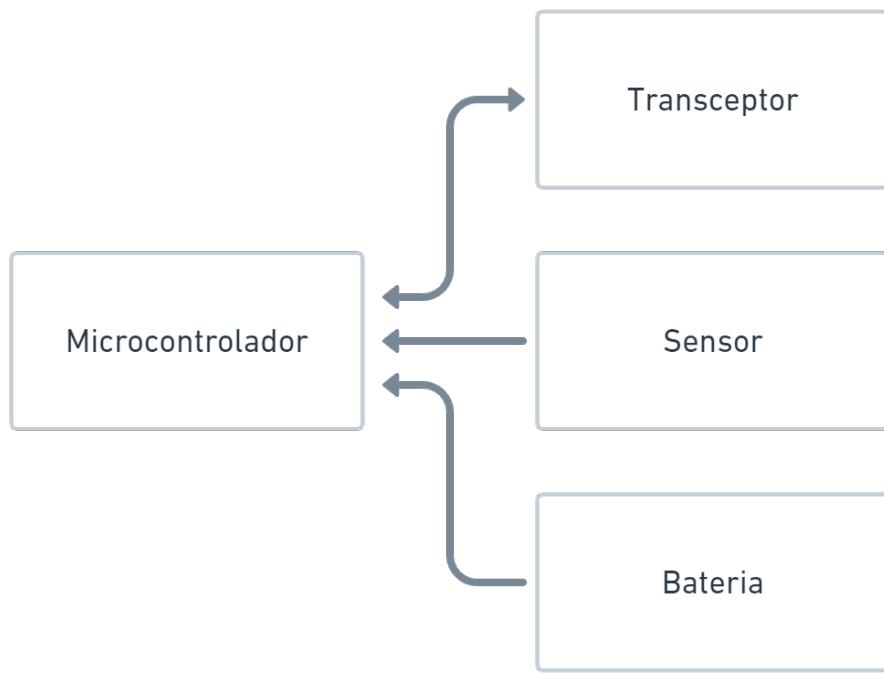


Figura 7 – Representação da conexão entre os componentes principais do *end node* (autônoma).

Dentre os diversos sensores disponíveis no mercado para monitoramento de temperatura e umidade, optou-se por escolher um sensor da família DHT, pois é bastante difundida na comunidade, tendo uma boa relação custo benefício. Entre esses sensores, aderimos por usar o DHT22, que consegue captar a temperatura e umidade do ambiente dentro das faixas necessárias para o projeto, com uma boa precisão.

Para o microcontrolador, foi escolhido usar um ATmega328, pois é o utilizado na maioria dos Arduino, o que nos trás dois grandes benefícios, poder utilizar um arduino como protótipo inicial, por ser fácil de encontrar, de programar e para um protótipo secundário, é possível construir uma placa própria utilizando apenas o ATmega328 e componentes essenciais para seu funcionamento, e por utilizar o mesmo microcontrolador do Arduino é possível utilizar o mesmo código, bibliotecas e o mesmo Ambiente de desenvolvimento integrado, IDE, utilizado no primeiro protótipo.

Entre os modelos de transceptores LoRa, o eleito foi o RFM95W, por ser o modelo mais simples e com menos custo e por ser o utilizado dentro do *shield* Arduino, um componente modular feito para adicionar novas funcionalidades a um Arduino de forma fácil e prática, facilitando assim, o desenvolvimento do primeiro protótipo.

Em relação à alimentação do protótipo, é um ponto mais complicado de se analisar e ter uma escolha realmente boa para o caso, mesmo fazendo uma análise aprofundada, apenas teremos uma aproximação da sua eficiência, tal processo é melhor detalhado na

seção 3.2.3.

3.2.1 Primeiro Protótipo

O primeiro protótipo foi feio com o objetivo de validar os usos das tecnologias, seu alcance, e sua imunidade em relação a interferências ao longo do trajeto. O consumo energético foi deixado de lado neste protótipo, pois foi utilizado o Arduino e ele possui muitos componentes desnecessários para esta aplicação, que acabam aumentando o gasto energético do dispositivo.

Ele coleta os dados utilizando o sensor DHT-22, formata os dados em uma string e o envia através do LoRa, após isto, ele entra em modo de sono profundo (*deep sleep*), desligando o máximo das suas funcionalidades, deixando apenas o mínimo para poder voltar ao funcionamento normal, isto resulta em uma economia do consumo energético por um determinado tempo, ao acordar, ele realiza a medição novamente, e fica nesse ciclo.



Figura 8 – Foto do primeiro protótipo (autorral).

3.2.2 Segundo Protótipo

O objetivo na construção deste protótipo é construir um dispositivo com um tamanho reduzido e melhorar sua eficiência energética, visando o seu funcionamento por meses com apenas uma bateria de 2200 mAh, melhor detalhada na seção 3.2.3. Para conseguir alcançar tais metas, é preciso remover as partes desnecessárias para o funcionamento da aplicação pelo Arduino. Empregando apenas os componentes essenciais, microcontrolador Atmega328, transceptor LoRa RFM95W e o sensor DHT-22, junto com alguns componentes de suporte para fornecer o funcionamento correto, um capacitor de cerâmica de 100 pF, dois resistores de 10 kOhms e um circuito para fornecer o clock para o microcontrolador composto por um cristal oscilador de 16 mhz e dois capacitores de cerâmica de 22 pF. Tal circuito, como pode ser visto na figura 9 e seu esquemático na figura 10, foi montado em uma protoboard junto com um conversor USB para serial, para

ser possível ver os logs do dispositivo pelo computador.

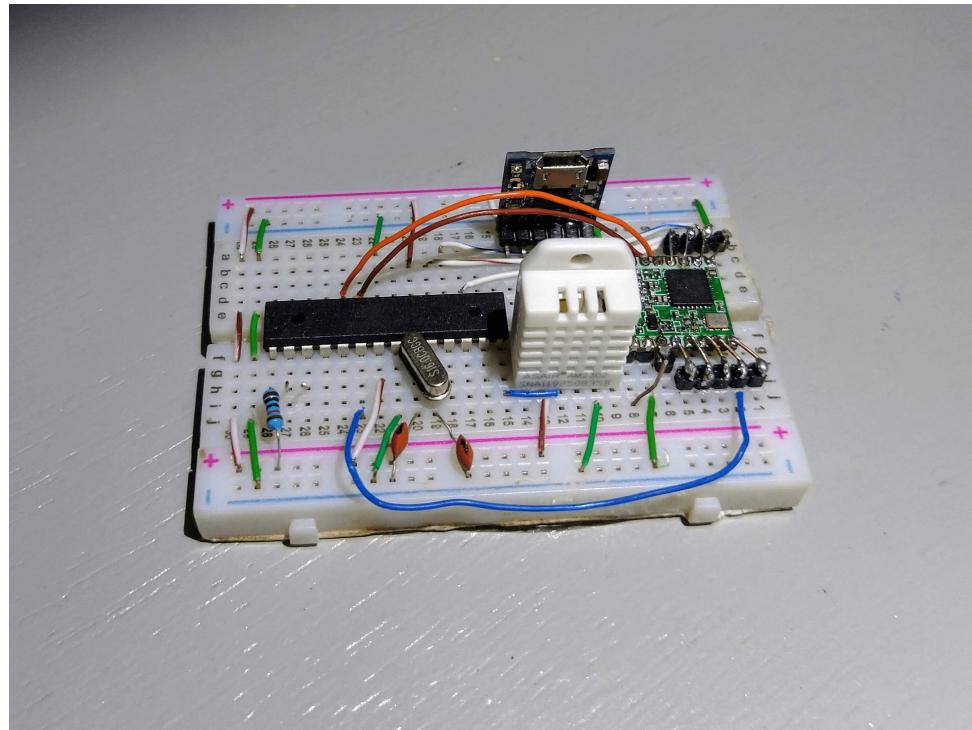


Figura 9 – Foto do segundo protótipo (autoral).

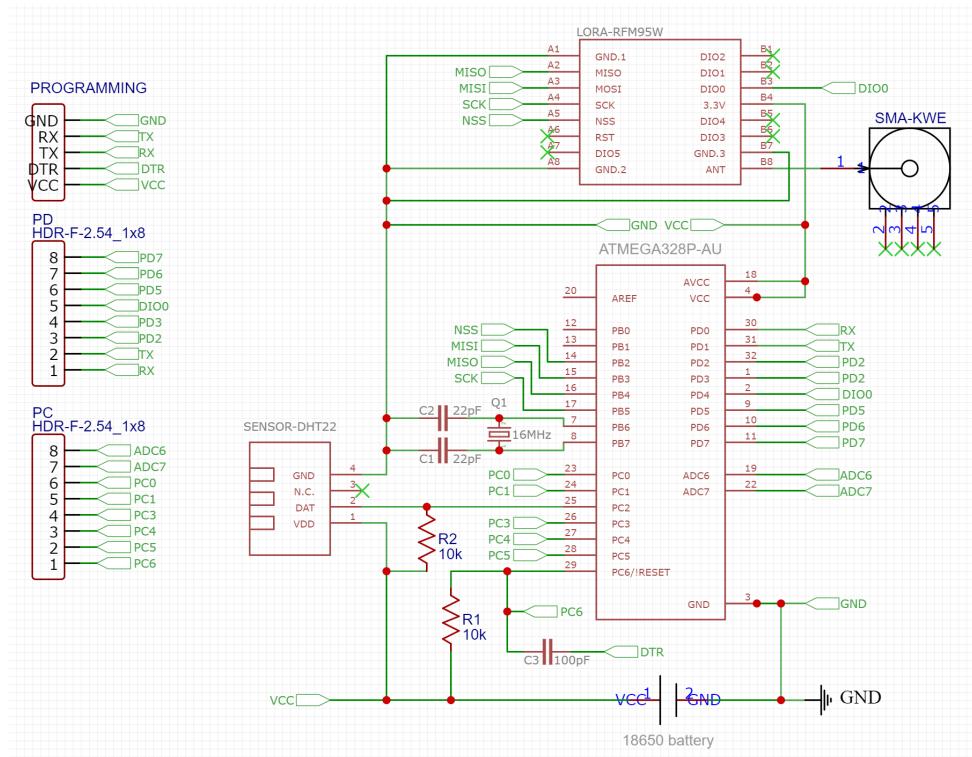


Figura 10 – Esquemático do segundo protótipo (autoral).

3.2.3 Escolha da Bateria

A parte crucial a ser analisada é o consumo de energia, isso é importante para a escolha correta da bateria, que levará a uma autonomia maior de operação que pode ser meses até anos dependendo da necessidade do produto. É preciso ter em mente as tensões de trabalho dos componentes principais da aplicação, para assim ser possível analisar qual deve ser o valor ideal de trabalho do dispositivo, é possível ver tal informação na tabela 1, mostrando a tensão mínima e máxima do microcontrolador, transceptor e sensor. Com isto, podemos ver que a tensão de limite superior do dispositivo é referente ao do transceptor LoRa, que funciona até 3,7 Volts, e a tensão de limite inferior, é a do sensor DHT-22, que equivale a 3,3 Volts, se utilizar uma tensão fora deixa faixa, o dispositivo não funcionará corretamente.

A ideia inicial antes da análise era de usar duas pilhas AA convencionais, entretanto após a análise, tal opção torna-se inviável, pois a tensão máxima de duas baterias AA equivale a 3 Volts, o que é menos do que o mínimo necessário para o sensor DHT-22 funcionar.

Componente	Tensão de trabalho
ATmega328	2,7V até 5,5V
LoRa RFM95W	1,8V até 3,7V
DHT-22	3,3V até 5,5V

Tabela 1 – tensão de trabalho dos componentes principais do protótipo (autoral).

Depois de algumas pesquisas, nos deparamos com uma bateria genérica de referência 18650, que tem o mesmo tamanho de uma AA, é fácil de encontrar em lojas de eletrônica, tem uma capacidade que varia entre 2200 mAh e 3400 mAh conforme o fabricante, e o mais importante, tem a tensão nominal de 3.7 Volts, o máximo que nosso circuito suporta. Sabendo que os ambientes de armazenagens de vacinas ficaram em torno de -5°C a 8°C, na figura 11 podemos ver a linha de descarga da bateria conforme a temperatura de funcionamento, para simplificar os cálculos podemos usar a linha em azul de 0°C. nossa tensão de corte é de 3.3 Volts, analisando o gráfico vemos que a bateria em um ambiente a 0°C a capacidade da bateria cai para aproximadamente 90%, que equivale há 1980 mAh do modelo de 2200 mAh.

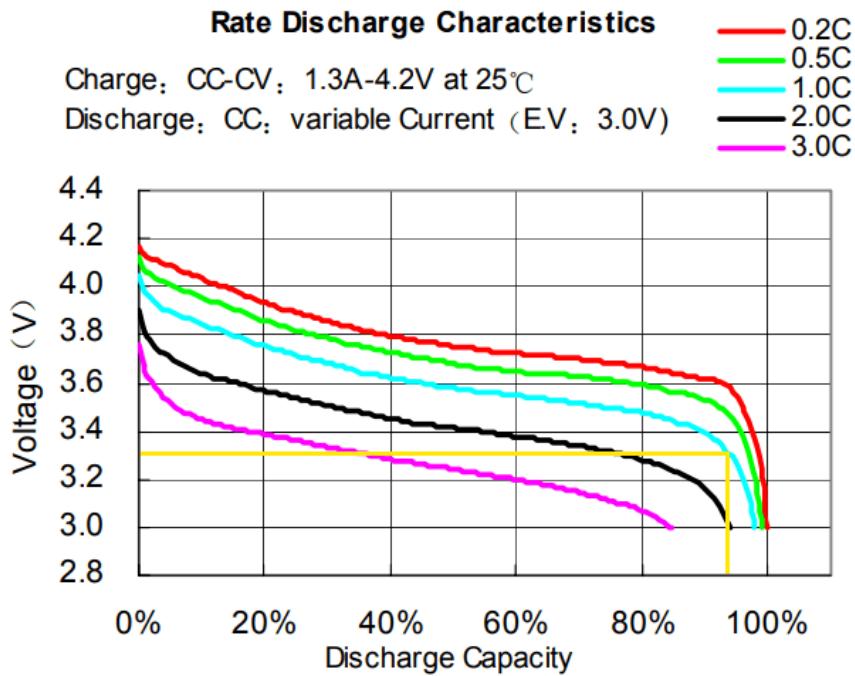


Figura 11 – Descarga da bateria 18650 conforme a temperatura do ambiente (Adaptada de [4]).

3.3 Gateway

O *gateway* é o dispositivo que fica entre os *end nodes* e o servidor, e em cada um desses lados, é utilizado uma tecnologia de comunicação diferente, para receber os dados enviados pelos *end nodes*, se utiliza o LoRa, já para enviar estes dados para o servidor, utiliza-se o HTTP via WiFi. Por isso, é preciso que o *gateway* possua essas duas tecnologias nele. Felizmente, a Heltec Automation, uma empresa que desenvolve produtos voltada para a IoT utilizando o LoRa, possui um microcontrolador ESP32 que contém tanto o transceptor LoRa, quanto o WiFi, como podemos ver na figura 12. Para programá-lo, foi utilizado a linguagem Arduino, compartilhando assim, a mesma biblioteca entre o *end node* e o *gateway*, para o manuseio do transceptor LoRa.

Seu funcionamento segue a seguinte linha, fica aguardando os pacotes dos *end nodes*. Ao receber um, é realizado uma análise de erro simples, onde é verificado se a formatação do pacote está como esperado, se for detectado algum erro de formatação é enviado para o servidor os valores do RSSI, SNR, o contador do pacote e um atributo booleano marcado como falso, indicando que veio com erro para a seguinte rota, */packages/:endnodeId*, onde *:endnodeId* é substituído pelo identificador do *end node* recebido pelo *gateway*. Se nenhum erro foi detectado, é enviado para a rota do servidor */sensors/:endnodeId*, os mesmo dados de RSSI, SNR e contador do pacote com o acréscimo dos dados de temperatura e umidade.



Figura 12 – Foto do ESP32 LoRa da Heltec Automation (autoral).

A configuração do *gateway*, seu identificador criado pelo servidor e o nome e a senha da rede sem fio que vai ficar conectado, é feita por um ponto de acesso, onde ao liga-lo, ele fornece uma rede sem fio onde é possível acessar seu IP via navegador de Internet, e se pode realizar as configurações necessárias.

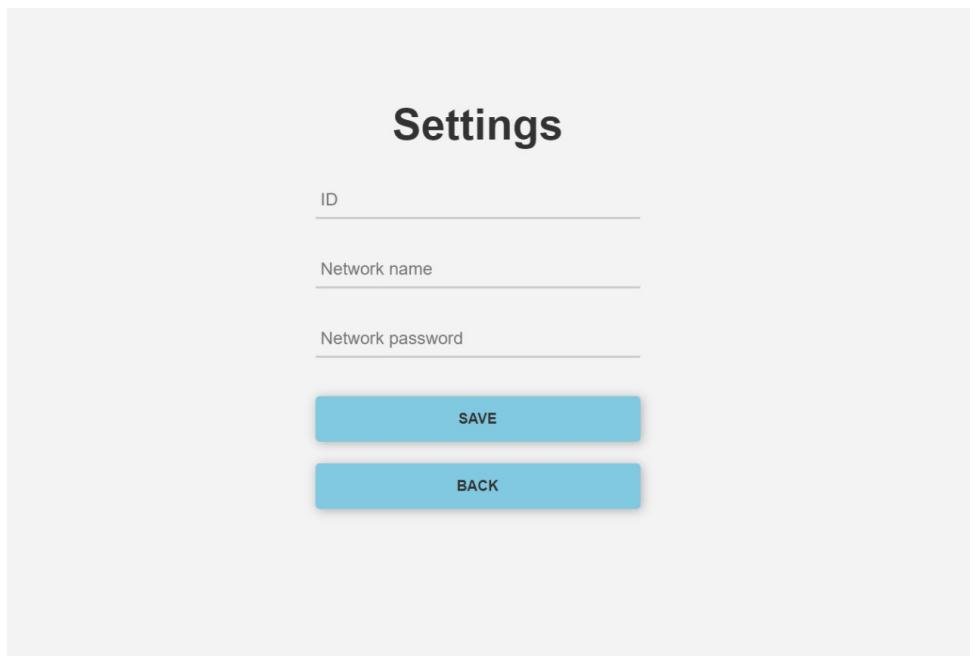


Figura 13 – Página de configuração do *gateway* (autoral).

3.4 Servidor

Para este projeto, o servidor tem as seguintes responsabilidades: armazenar os dados coletados pelos *end nodes*, juntamente com os dados do instituto que está usando a aplicação e dos seus respectivos *end nodes* e *gateways* adquiridos; fornecer rotas para troca de informações com o aplicativo e o *gateway*, para que possam realizar suas devidas funções.

3.4.1 Padrões de Projetos

Foi escolhido para desenvolver o servidor, a utilização do Node.js e a linguagem Typescript, se baseando em alguns padrões de *softwares*, como o Domain-Driven Design (DDD), Test-Driven Development (TDD) e o SOLID visando assim, a construção de um servidor robusto, escalável e de fácil manutenção.

Vale salientar que, esses padrões de *softwares*, são metodologias de desenvolvimento, e devem ser aplicados de acordo com a demanda da aplicação em construção. Elas são boas práticas, entretanto, dependendo do projeto, alguns pontos podem acabar deixando o processo lento, e dando pouco benefício, e por isso, nesse projeto, não foi seguido fielmente esses padrões, apenas foi baseado.

Antes de tudo, é preciso definir as regras de negócio básicas do servidor, elas servem para garantir que a aplicação atenda as necessidades esperadas. Para este serviço deve ser possível o usuário se cadastrar, para assim realizar seu *login*, e ter acesso às funcionalidades de cadastrar, editar e remover seus *end nodes* e seus *gateways*, além de poder visualizar os dados coletados.

Durante o desenvolvimento de um *software*, é essencial que a entrega do *software* funcione corretamente, com qualidade e de acordo com as regras de negócio. Para garantir tais exigências, é importante que se realize testes, tendo em vista identificar possíveis erros antes de chegar aos usuários. Entretanto, realizar testes corretamente é uma tarefa complicada, por isto, existem diversas metodologias que visam facilitar e simplificar tal desenvolvimento. O TDD é uma dessas metodologias, ela defende o desenvolvimento do teste antes das funcionalidades, garantindo a cobertura completa do código pelos testes.

Junto com o TDD, foi utilizado também o DDD, uma filosofia para auxiliar os desenvolvedores na construção de aplicações complexas de *software*, ela é referência na organização do código, separando por domínios, de forma isolada. Para poder implementar bem o DDD é preciso definir quais são os domínios da aplicação, analisando as regras de negócio, foi separado em três domínios: *end nodes*, *gateways* e usuários.

É preciso também, separar a camada de domínio da aplicação da camada de infraestrutura, a camada de infraestrutura é responsável pelas tecnologias utilizadas para

realizar determinada função, por exemplo, a camada de domínio sabe que precisa armazenar os dados dos sensores, mas não é precisar saber onde e nem como, isso é de responsabilidade da camada de infraestrutura.

Para ter uma melhor separação das responsabilidades entre as camadas, é comum utilizar o DDD junto com princípios do SOLID. O SOLID é um acrônimo de 5 princípios da programação orientada a objetos que ajudam o programador a escrever códigos mais limpos, com alta manutenibilidade, separando as responsabilidades e diminuindo acoplamentos.

3.4.2 Banco de Dados

É preciso armazenar os dados coletados pelos sensores, para futuras consultas, armazenar os dados de transmissão dos pacotes, para analisar a qualidade da transmissão, e é preciso também, salvar os dados dos usuários e dos seus respectivos produtos cadastrados na plataforma. Tais dados têm suas diferenças, e por isto foi escolhido utilizar banco de dados diferentes para cada um deles.

Começando pelos dados coletados pelos sensores e transmitido pelos *end nodes*, esses dados são de medidas, coletados com um período curto, e consequentemente, possuem um grande volume. Por isto, foi escolhido utilizar o InfluxDB para armazenar tais dados.

Foram criadas duas *measurements* dentro do banco, uma referente aos dados dos sensores e outra aos dados do pacote transmitido, os dois possuindo a mesma *tag*, o identificador do *end node*, chamado de *endnodeId*. Dentro da medida Sensor, foi adicionado dois campos referente a temperatura e umidade do ambiente. Para a medida Package, foi adicionado os campos *success*, *rssi* e *snr*, atributos necessários para saber se o pacote foi entregue com sucesso, a dificuldade de transmitir esse pacote e a relação sinal-ruído. Podemos então ver uma representação do banco na figura 14.

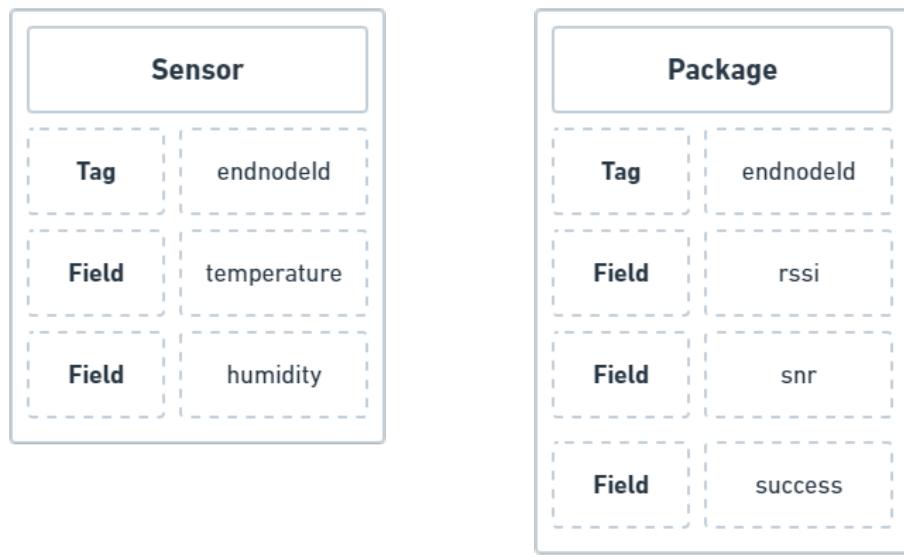


Figura 14 – Medidas dentro do banco de dados InfluxDB (autoral).

Diferentes dos dados que vem dos hardwares, as informações dos usuários e seus dispositivos, não possuem essa demanda alta de escrita e leitura, e mais importante, contém relações entre eles, o que torna banco de dados relacionais uma boa opção para armazenar tais dados. Entre tantos bancos de dados relacionais, foi optado por usar o PostgreSQL por motivos de afinidade.

Cada usuário pode possuir inúmeros *gateways* e *end nodes* cadastrados, e cada *end node* precisa ser vinculado com apenas um *gateway*. Essas relações entre entre as entidades no postgres representada na figura 15, foi pensada olhando tanto para os cenários de pequenos postos de saúde, que tem apenas um local para armazenar seus imunobiológicos, quando para grande hospitais e laboratórios, que pode ter varias salas, com dada sala contendo uma ou mais equipamentos frigoríficos.

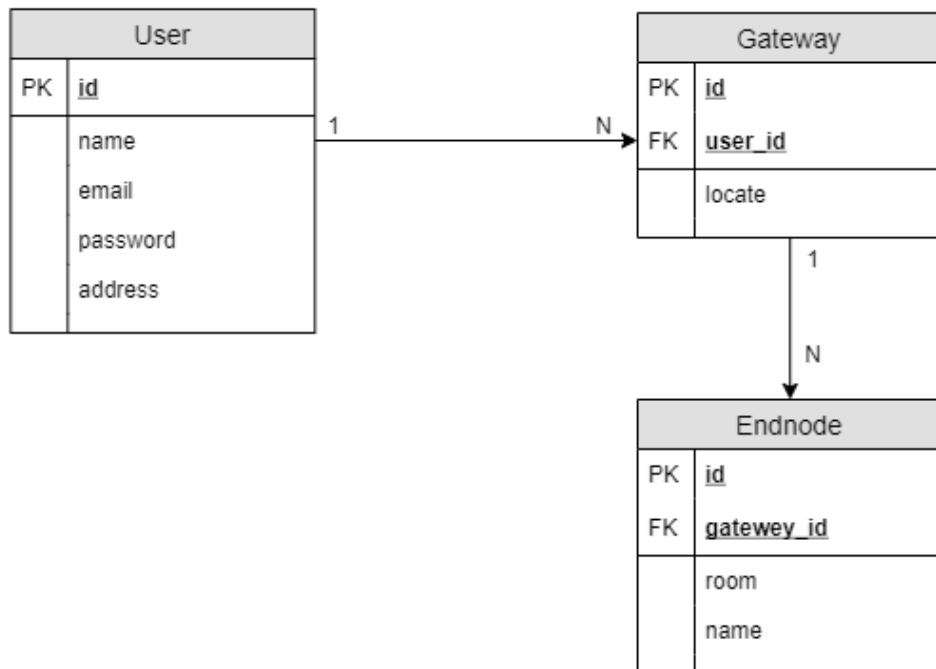


Figura 15 – Modelo entidade relacionamento do PostgreSQL (autoral).

3.4.3 Containers

Para facilitar o desenvolvimento do servidor, seu deploy e diminuir possíveis problemas por ter várias aplicações rodando na mesma máquina, foi escolhido por conter tanto os bancos de dados quanto a aplicação em si, utilizando o Docker e para facilitar o gerenciamento dos três containers (PostgreSQL, InfluxDB e a aplicação em Node.js), interligando as redes entre eles, foi utilizado o Docker Compose.

Para os bancos de dados, não foi preciso criar as imagens dos containers, dado que são aplicações muito utilizadas e a própria comunidade já oferece algumas opções dependendo da sua finalidade. Entretanto, para a aplicação em si, foi necessário a criação de uma nova imagem, devido algumas configurações extras exigidas pelas tecnologias utilizadas pela aplicação. Tal imagem foi feita baseando-se na versão alpine do node, uma versão mais leve, com apenas as funcionalidades principais.

3.4.4 Deploy

Entre as opções de servidores na nuvem, foi escolhido por usar a Microsoft Azure para realizar o deploy do servidor. Sua escolha se deu pois a Microsoft oferece uma assinatura para estudantes, o que não teve nenhum custo adicional no desenvolvimento deste protótipo e na realização dos testes.

3.5 Aplicativo móvel

À necessidade de ter uma aplicação para que os usuários possam gerenciar seus *end nodes* e seus *gateways*, e poder acessar o histórico das medidas coletadas, tal aplicação poderia ser uma página web, entretanto, tendo em vista o crescimento do uso dos smartphones, tivemos como preferência a criação de uma aplicação móvel, mas especificamente para o sistema operacional Android, devido ao sistema iOS da Apple, restringir seu desenvolvimento para aparelhos da marca, incluindo a necessidade de um computador do mesmo para poder emular um dispositivo móvel com seu sistema operacional.

Tendo em mente as regras de negócios destrinchadas na seção 3.4.1, a primeira coisa feita foi o desenho das telas utilizando a ferramenta figma, específica para tal tarefa. Ao finalizar os desenhos das telas, partimos para a programação.

Dentre as diversas formas de construir um aplicativo deste porte, a ferramenta escolhida para designar esta função foi o React Native, por motivos de utilizar a mesma linguagem de programação do servidor, typescript, e ter a possibilidade de criar para sistemas iOS futuramente usufruindo do mesmo código.

Pensando na construção de uma aplicação com possibilidades altas de crescimento, foi decidido documentar os componentes que compõem o programa, utilizando a biblioteca Storybook. Na figura 16 juntamente com o apêndice A pode ser visto os desenhos das telas referente ao aplicativo.

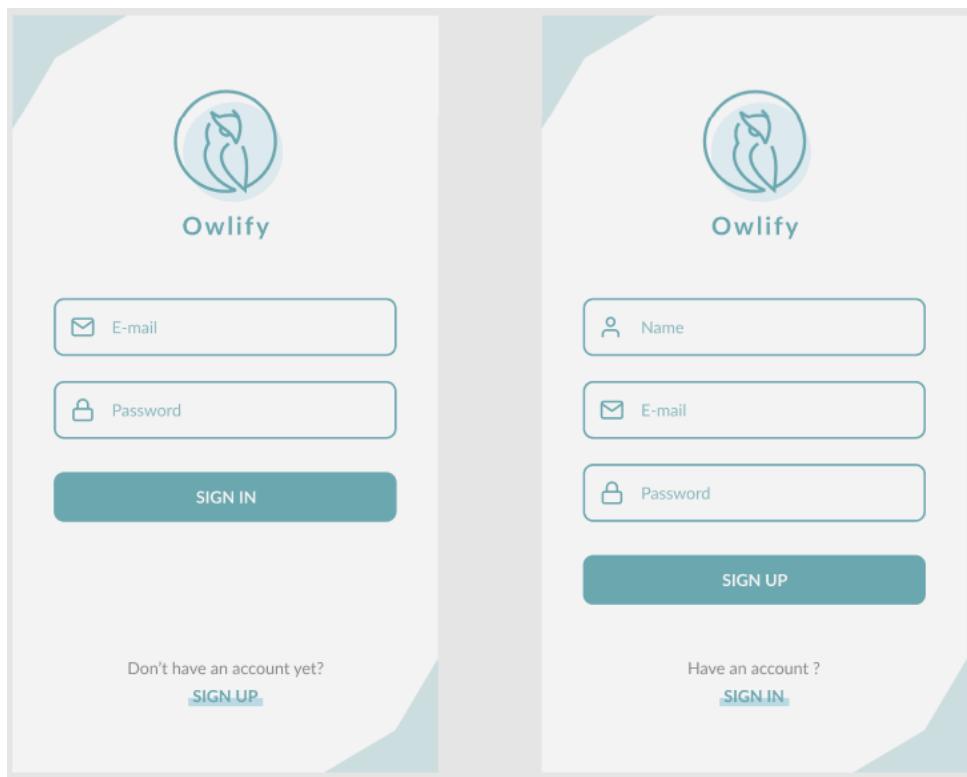


Figura 16 – Desenho das telas de entrar e cadastrar na aplicação (autoral).

4 Resultados

Para sabermos se o protótipo atingiu nossos objetivos é preciso analisar três pontos importantes: sua capacidade de transmissão em longo alcance juntamente com a eficiência de lidar com altas interferências ao longo do trajeto, seu consumo energético e seu custo de produção. Todo o código fonte desenvolvido neste projeto pode ser acesso em [27].

4.1 Análise de Transmissão dos Pacotes

Foi executado um teste de transmissão de pacotes com o objetivo de verificar a capacidade de transmissão em um longo alcance em um ambiente com altas interferências. O experimento foi realizado no bloco dos professores do IFPB - Campus Campina Grande, onde o *end node* ficou posicionado no laboratório GComPi, localizado no subsolo, enquanto o *gateway* ficou em um espaço diferente, no Laboratório Assert, no térreo do prédio, com cerca de sessenta metros de distância entre os dispositivos, e contanto com obstáculos variados à comunicação, como paredes densas, laje, objetos metálicos e equipamentos eletrônicos. Na figura 17 podemos ver a distância entre o *gateway* e o *end node*, respectivamente, A e B.

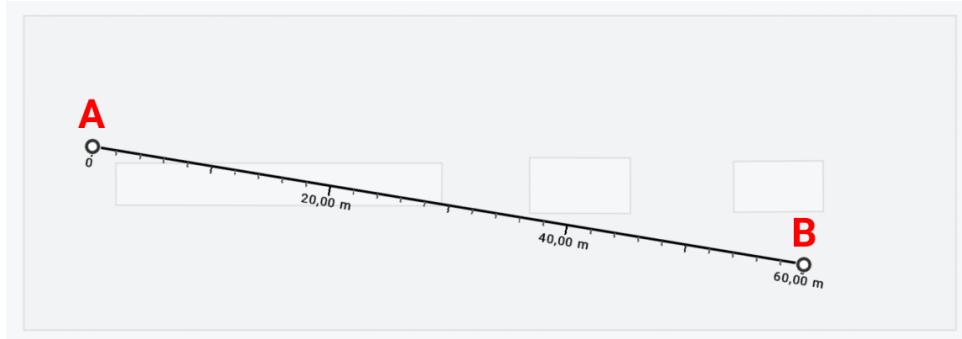


Figura 17 – Distância aproximadas entre os dispositivos (Adaptada do Google Maps).

Os dispositivos ficaram funcionando por cinco dias, 24 horas por dia, realizando envios contínuos de pacotes em intervalos de cinco minutos e o código da análise poder ser acessado em [28]. O gráfico na figura 18 apresenta a taxa de entrega de pacote, PDR, por horas, das coletas realizadas no dia 21 de abril de 2020. A média geral da PDR foi de 78,82%, um valor agradável, no entanto, em alguns momentos a PDR ficou em torno de 20%, o que pode ter sido ocasionado por modificações no perfil de multi-percurso do ambiente, obstáculos temporários ou presença de fontes de interferência. Considerando a taxa de transmissão de pacote utilizada, com PDR de 20% ainda é possível obter uma nova informação a cada 25 minutos, em média, o que é suficiente para a realização do monitoramento de temperatura e umidade, que são grandezas que variam lentamente.

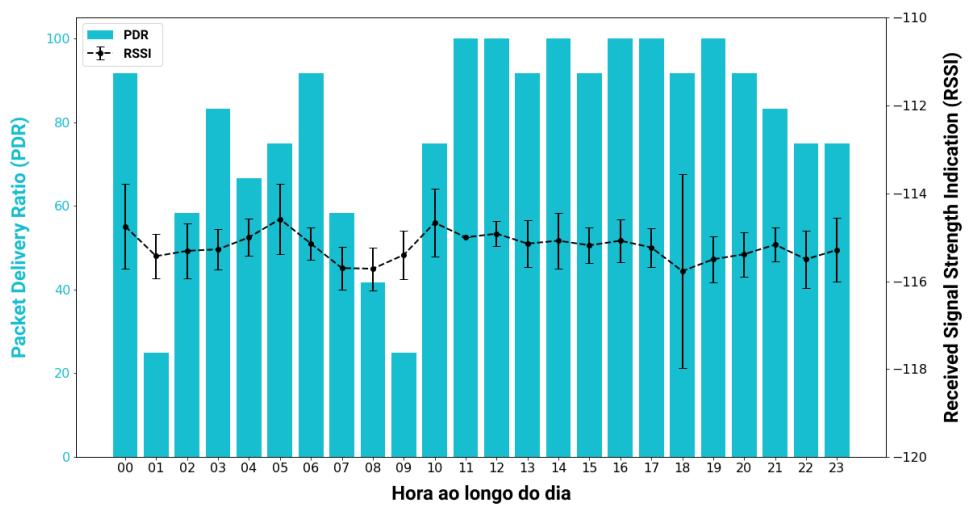


Figura 18 – Percentual de entrega dos pacotes em barras e indicador de intensidade do sinal recebido em linha (Autoral).

4.2 Análise do Consumo Energético

É possível estimar o tempo de duração do *end node* com a bateria escolhida, dividindo a capacidade da bateria pelo consumo de energia médio do dispositivo, e para saber o consumo médio do *node*, precisamos saber o valor do consumo em dois pontos, quando estiver realizando a transmissão de um pacote, quando estiver em período de espera para a próxima transmissão e o tempo de duração de cada período.

Os dados foram medidos utilizando um multímetro e um osciloscópio, para o primeiro protótipo, entretanto, para o segundo protótipo, devido a dificuldade de acesso aos equipamentos por causa da pandemia do COVID-19, não foi possível realizar a medição de forma manual, e por isso foi utilizado os dados retirados dos *datasheet* dos componentes [2, 29, 30]. Para a medição com osciloscópio, foi necessário o uso de um resistor de derivação, também conhecido como resistor *shunt*, de 0,1 Ohm, medindo a queda de tensão diferencial através deste resistor.

O período de coleta que consideramos ser o suficiente para gerar uma boa monitoração é de uma hora e o LoRa demora em torno de 42 ms para realizar uma transmissão de um pacote. Podemos ver na figura 19, o gráfico mostrando os valores do período de transmissão dos seguintes protótipos para comparação: primeiro protótipo, do segundo protótipo utilizando a função padrão do Arduino, *delay*, e o segundo protótipo em modo *deep sleep* quando estiver em espera, este modo desativa o máximo de funcionalidade do microcontrolador deixa apenas o essencial para voltar ao funcionamento depois de um determinado período. Podemos ver, o consumo em espera e quando está realizando uma transmissão, respectivamente, para o primeiro protótipo, temos 50 ms e 150 ms, já para o

segundo protótipo utilizando *odelay*, temos 18 ms e 118 ms e para o segundo protótipo em modo *deep sleep*, temos os menos valores, com 0.06 ms e 118 ms.

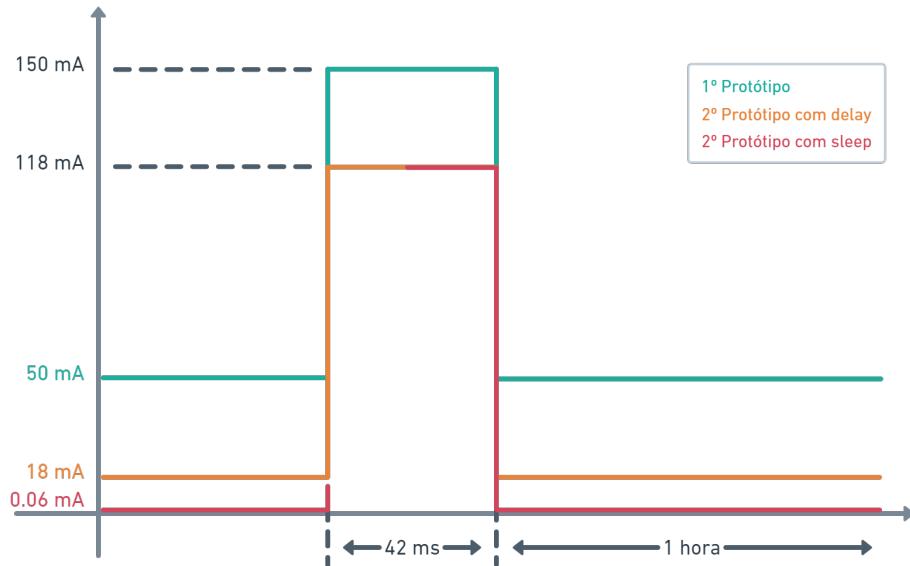


Figura 19 – Gráfico de consumo dos protótipos do *end node* (Aitoral).

Com esses dados, é possível calcular a corrente média somando os valores de medida multiplicados pelas suas porcentagens referentes ao tempo total. O tempo total do período equivale a 3600,042 s, que corresponde à soma do tempo em transmissão, 42 ms, com o tempo em espera, 3600 s. Temos então 56.0010904 mA para o primeiro protótipo, 18.00116 mA para o segundo utilizando *delay* e 0,06136 mA para o segundo em modo *deep sleep*, como é possível ver nas equações 4.1, 4.2 e 4.3, respectivamente.

$$(0,9999884 * 56mA) + (0,0000116 * 150mA) = 56,0010904mA \quad (4.1)$$

$$(0,9999884 * 18mA) + (0,0000116 * 118mA) = 18,00116mA \quad (4.2)$$

$$(0,9999884 * 0,06mA) + (0,0000116 * 118mA) = 0,06136mA \quad (4.3)$$

Agora, com esses dados, podemos calcular a duração teórica dos protótipos para uma bateria, no nosso caso a 18650, com 2200 mAh, que é aproximadamente 39 horas para o primeiro protótipo, 122 horas para o segundo protótipo utilizando *delay* e 35.849 horas para o segundo protótipo utilizando *deep sleep*, como mostra nas equações 4.4, 4.5 e 4.6.

$$\frac{2200mAh}{56,0010904mA} = 39,2849494h \quad (4.4)$$

$$\frac{2200mAh}{18,00116mA} = 122,214346h \quad (4.5)$$

$$\frac{2200mAh}{0,06136mA} = 35.849h \quad (4.6)$$

4.3 Análise do Custo

Todas as compras dos equipamentos foram realizadas no Brasil e em baixa quantidade, o que acaba deixando o protótipo mais caro comparado a compra realizada na China ou comprando em grande escala, que acaba tendo uma redução considerável do seu custo. Tendo isto em mente, podemos analisar a tabela 2, representando o custo (sem consideração do frete), da cada peça encontrada no Brasil e na China para montar o segundo protótipo do *end node* apresentado na sessão 3.2.2. Os dados da tabela foram preenchidos com os menores valores encontrados no dia 11 de abril de 2021 considerando os revendedores mais confiáveis. Temos então, um total de R\$ 97,66 comprando localmente e R\$ 48,40 importando, que dá em torno de 49,56% a menos comprando na China, tornando assim um melhor custo benefício. Já para o *gateway*, o preço no Brasil fica em torno de R\$ 180,00 e importando pode ser encontrado por menos de R\$ 100,00.

Considerando que, ao utilizar um produto deste porte, não será mais necessário a compra de câmeras conservadoras com monitoração, que pode chegar a custar R\$ 10.000,00, podendo assim, economizar até R\$ 7.000,00 por refrigerador, tornando o nosso produto viável financeiramente para as empresas.

Componente	Preço no Brasil	Preço na China
Um ATmega328	R\$ 19,90	R\$ 9,12
Um LoRa RFM95W 915 Mhz	R\$ 56,00	R\$ 20,99
Um Bateria 18650 3000 mAh	R\$ 19,90	R\$ 17,65
Um Cristal Oscilador 1 6mhz	R\$ 01,47	R\$ 00,46
Um Capacitor de cerâmica 100pF	R\$ 00,05	R\$ 00,04
Dois Capacitor de cerâmica 22pF	R\$ 00,22	R\$ 00,08
Dois Resistores de 10 kOhms	R\$ 00,12	R\$ 00,06
Total	R\$ 97,66	R\$ 48,40

Tabela 2 – Custos referente as peças do segundo protótipo do *end node* (autor).

Temos então os preços dos protótipos da parte física, os hardwares, é preciso agora, calcular o custo da publicação do aplicativo na *Play Store*, loja de aplicativos da Google, e o custo de hospedagem do servidor. Para a publicação do aplicativo é mais simples, a Google cobra apenas um valor fixo para poder realizar qualquer publicação na loja de US\$25,00, em torno de R\$ 131,56 na cotação do dia 17 de maio de 2021.

Por fim, temos o custo da hospedagem do servidor, essa é um valor mensal que varia conforme a demanda de processamento, armazenamento e transferência de dados, e também é diferente para cada provedor. Dentre os diversos provedores, os mais conhecidos são: Amazon Web Services, Google Cloud, Microsoft Azure e DigitalOcean.

A DigitalOcean oferece em seu site, uma calculadora que compara o custo em dólares estadounidenses por mês, entre os provedores citados, conforme a configuração do servidor desejada¹. Inicialmente, a melhor configuração é a básica, pois a demanda de serviço é baixa, com o tempo, basta aumentar de acordo com o crescimento da demanda. Tendo isto em mente, levei em consideração a configuração básica e a de propósitos gerais fornecida pela DigitalOcean, os dados podem ser vistos na figura 20.

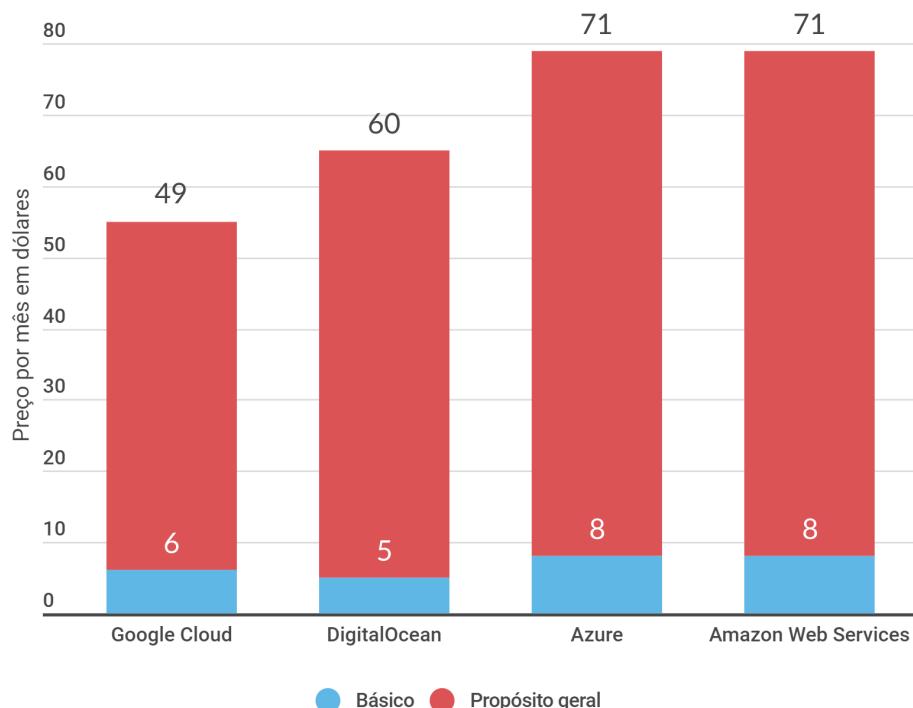


Figura 20 – Comparação do custo mensal de hospedagem do servidor (Autoral).

Na configuração básica, a DigitalOcean é a opção mais barata, custando US\$5,00, seguido pela Google Cloud por US\$6,00. Já na configuração de propósitos gerais, temos uma inversão, a mais barata fica com a Google Cloud por US\$49,00 seguida pela DigitalOcean

¹ <https://www.digitalocean.com/pricing/calculator/>

por US\$60. Entretanto, conforme aumenta a demanda, acaba com a DigitalOcean ficando na frente, além dela prover uma boa usabilidade.

5 Considerações Finais

O presente trabalho apresentou um sistema de monitoração de produtos imunobiológicos, visando o auxílio dos profissionais da saúde em manter a qualidade dos materiais. Tal solução é composta por um aplicativo móvel para o cliente realizar a monitoração, um servidor em nuvem, responsável por concentrar os dados coletados dos usuários e dos hardwares, que captam as temperaturas e umidades e transmitem para o servidor.

Foi comprovado na sessão 4.1 a efetividade dos hardwares de transmitir os dados coletados em uma distância considerável, cerca de 60 metros, em um ambiente com inúmeros obstáculos causando bastante interferência, proporcionado uma taxa de entrega de pacotes de 78,82%, um valor alto para o ambiente proposto.

Em relação ao custo total de confecção do produto, comprando localmente, ficou em torno de R\$ 180,00 para o *gateway* e R\$ 97,66 para cada *end node*, a quantidade de *end nodes* varia conforme a quantidade de ambientes de armazenagem do usuário, uma para cada câmera de conservação de termolábeis. Em relação ao *gateway*, será necessário adquirir outra unidade apenas em caso de algum *end node* ficar localizado em uma distância grande ao ponto do receptor não conseguir receber os pacotes, já que nas condições estabelecidas, utilizando o LoRaWAN, cada *gateway* suporta, teoricamente, cerca de 62 mil *end node* [20]. Considerando a possibilidade de realizar a comprar dos componentes importando da China, o preço dos produtos pode chegar a 50% a menos em comparação aos preços locais, além de, pensando na produção dos dispositivos em massa, esse preço tende a cair, entretanto, não foi realizado um estudo para saber o quanto de economia teria. Já para a publicação do aplicativo temos um custo fixo de \$25,00 e para o servidor, iniciando com a configuração básica, temos \$5,00 por mês, podendo crescer conforme o aumento da demanda.

5.1 Sugestões para Trabalhos Futuros

Há diversas melhorias que podem ser feitas neste projeto. O LoRa possui um ajuste da potência de transmissão que varia entre +7 dBm a +20 dBm, quanto maior a potência, maior o alcance de transmissão, entretanto, o consumo também cresce. É possível realizar um teste entre esses valores em busca de um melhor balanço entre economia energética e alcance de transmissão, seria interessante também adicionar uma configuração no dispositivo onde o usuário possa escolher qual potência usar, conforme for a sua demanda.

Outra sugestão seria adicionar uma nova funcionalidade ao *end nodes* para detectarem

que estão em um ambiente fora do recomendado, ou se teve uma variação considerável em um curto período de tempo da temperatura do ambiente, para que possam emitir um alerta, onde o aplicativo recebe, em tempo real, tal informação.

Olhando para o consumo energético, apesar de termos alcançado um bom resultado, ainda é possível diminuir tal consumo utilizando um *clock* menor, o componente responsável por isso é o cristal oscilador, que neste projeto foi utilizando um de 16 mHz, entretanto, esta aplicação não necessita dessa *clock* alto, podendo assim utilizar um cristal oscilador de valor menor, inclusive, o próprio Atmega328, possui um *clock* interno de 1 mHz, que pode ser utilizado no lugar do cristal oscilador, o que traz dois benefícios, uma diminuição dos componentes para o circuito e uma economia de energia.

Um teste interessante a ser realizado, é uma análise em relação a quantidade de dispositivos conectados simultaneamente a um *gateway*, entretanto, necessitaria de muitos dispositivos para realizar tal teste, o que dificultaria na sua realização, considerando que, teoricamente, um *gateway* suporta 1,5 milhões de pacotes por dia, e cada *end node* transmite um pacote por hora, totalizando 24 pacotes por dia.

Este trabalho teve um foco maior na construção do *end node*, o que deixou o *gateway* com um grande margem de melhoria, como a construção de um protótipo com apenas os componentes necessárias para seu funcionamento, que resultaria em um custo benefício maior e uma economia energética melhor.

Por fim, o servidor foi construído de uma forma modular, que facilita a expansão de novas funcionalidades em trabalhos futuros e, pensando na usabilidade do sistema como um todo, tentaria melhorar a usabilidade do aplicativo móvel, em torno do gerenciamento dos dispositivos cadastrados, na atribuição dos identificados aos equipamentos, entre outros.

Referências

- 1 3GLTEINFO. *LoRa Architecture*. 2020. Disponível em: <<https://www.3glteinfo.com/lora/lora-architecture>>. Citado 2 vezes nas páginas 7 e 19.
- 2 ELECTRONICS, A. *DHT22 Datasheet*. 2020. Disponível em: <<https://pdf1.alldatasheet.com/datasheet-pdf/view/1132459/ETC2/DHT22.html>>. Citado 3 vezes nas páginas 7, 20 e 40.
- 3 HAT, R. *Containers x máquinas virtuais*. 2020. Disponível em: <<https://www.redhat.com/pt-br/topics/containers/containers-vs-vms>>. Citado 2 vezes nas páginas 7 e 24.
- 4 EEMB. *Lithium-ion Battery 18650 Datasheet*. 2010. Disponível em: <<https://www.ineltro.ch/media/downloads/SAAItem/45/45958/36e3e7f3-2049-4adb-a2a7-79c654d92915.pdf>>. Citado 2 vezes nas páginas 7 e 32.
- 5 FILHO, N. d. A. *O que é saúde?* [S.l.]: Editora Fiocruz, 2011. Citado na página 13.
- 6 MOURA, E. S. D. O direito à saúde na constituição federal de 1988. *Âmbito Jurídico, XVI*, v. 114, 2013. Citado na página 13.
- 7 SOMA, C. *Como funciona o tratamento com imunobiológicos?* 2018. Disponível em: <<https://clinicasoma.com.br/como-funciona-o-tratamento-com-imunobiologicos>>. Citado na página 13.
- 8 SAÚDE, M. da. *Manual de Rede de Frio*. 2001. Disponível em: <http://bvsms.saude.gov.br/bvs/publicacoes/manual_rede_frio.pdf>. Citado 2 vezes nas páginas 13 e 14.
- 9 OLIVEIRA, V. C. d. et al. Avaliação da qualidade de conservação de vacinas na atenção primária à saúde. *Ciência & Saúde Coletiva*, SciELO Public Health, v. 19, p. 3889–3898, 2014. Citado 2 vezes nas páginas 13 e 15.
- 10 MUNIZ, É. S. *Memórias da erradicação da varíola*. [S.l.]: SciELO Brasil, 2011. Citado na página 13.
- 11 TEMPORÃO, J. G. O programa nacional de imunizações (pni): origens e desenvolvimento. *História, ciências, saúde-manguinhos*, SciELO Brasil, v. 10, p. 601–617, 2003. Citado 2 vezes nas páginas 13 e 14.
- 12 NIFORATOS, J. D. Common questions about the pfizer-biontech and moderna covid-19 vaccines. Citado na página 14.
- 13 BAE, J. et al. Challenges in equitable covid-19 vaccine distribution: A roadmap for digital technology solutions. Citado na página 15.
- 14 NELSON, C. et al. Monitoring temperatures in the vaccine cold chain in bolivia. *Vaccine*, Elsevier, v. 25, n. 3, p. 433–437, 2007. Citado na página 15.

- 15 FALCÓN, V. C. et al. A vaccine cold chain temperature monitoring study in the united mexican states. *Vaccine*, Elsevier, v. 38, n. 33, p. 5202–5211, 2020. Citado na página 15.
- 16 CRUZ, R. F.; CORREA, C. S.; SILVA, W. L. Desenvolvimento de um sensor de temperatura de baixo custo aplicado ao controle da qualidade de vacinas. Citado na página 16.
- 17 LIMA, M. S. et al. Controle de temperatura com arduino. *Revista Mythos*, v. 12, n. 2, p. 48–55, 2019. Citado na página 16.
- 18 KERSBAUM, M. et al. Monitoramento de temperatura para câmara de vacina. *Anais da Mostra Nacional de Iniciação Científica e Tecnológica Interdisciplinar (MICTI)-e-ISSN 2316-7165*, v. 1, n. 12, 2019. Citado na página 16.
- 19 NETO, F. D. N. L. Monitoramento remoto de temperatura para a armazenagem de materiais biológicos e vacinas, aplicado em pesquisa clínica. 2019. Citado na página 16.
- 20 ALLIANCE, L. *What is LoRaWAN® Specification*. 2020. Disponível em: <<https://lora-alliance.org/about-lorawan>>. Citado 2 vezes nas páginas 19 e 45.
- 21 TILKOV, S.; VINOSKI, S. Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, IEEE, v. 14, n. 6, p. 80–83, 2010. Citado na página 21.
- 22 FOUNDATION, O. *Documentação oficial do Node.js*. 2019. Disponível em: <<https://nodejs.org/pt-br/about/>>. Citado na página 21.
- 23 GIACOBBE, M. et al. An implementation of influxdb for monitoring and analytics in distributed iot environments. In: SPRINGER. *International conference on the Sciences of Electronics, Technologies of Information and Telecommunications*. [S.l.], 2018. p. 155–162. Citado na página 22.
- 24 SOUSA, F. R.; MOREIRA, L. O.; MACHADO, J. C. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. *II Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI)*, p. 150–175, 2009. Citado na página 24.
- 25 BRASIL, R. *Empresas que utilizam React no Brasil*. 2020. Disponível em: <<https://github.com/react-brasil/empresas-que-usam-react-no-brasil>>. Citado na página 25.
- 26 FACEBOOK. *Documentação oficial do React native*. 2020. Disponível em: <<https://reactnative.dev/>>. Citado na página 25.
- 27 AUTORAL. *Código fonte do sistema desenvolvido neste trabalho*. 2020. Disponível em: <<https://github.com/henry-ns/owlify>>. Citado na página 39.
- 28 AUTORAL. *Código da análise do teste de transmissão*. 2020. Disponível em: <<https://github.com/GComPI-IFPB/lora-transmitter-test>>. Citado na página 39.
- 29 ATMEL. *ATmega328P Datasheet*. 2020. Disponível em: <https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf>. Citado na página 40.

- 30 ELECTRONIC, H. *LoRa RFM95*. 2020. Disponível em: <https://cdn.sparkfun.com/assets/learn_tutorials/8/0/4/RFM95_96_97_98W.pdf>. Citado na página 40.

Apêndices

APÊNDICE A – Desenho das Telas do Aplicado Móvel

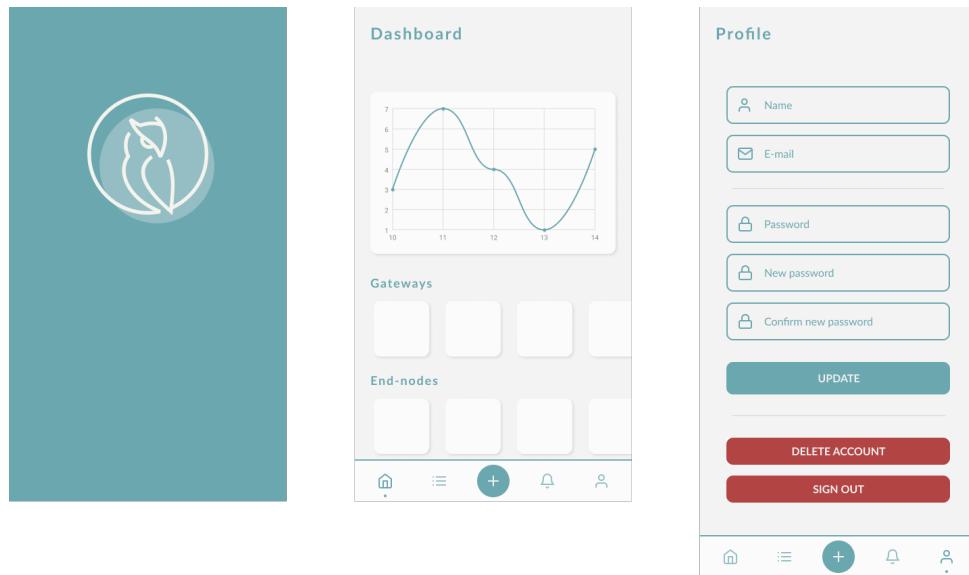


Figura 21 – Tela de carregamento, dashboard e perfil, respectivamente (Autoral).

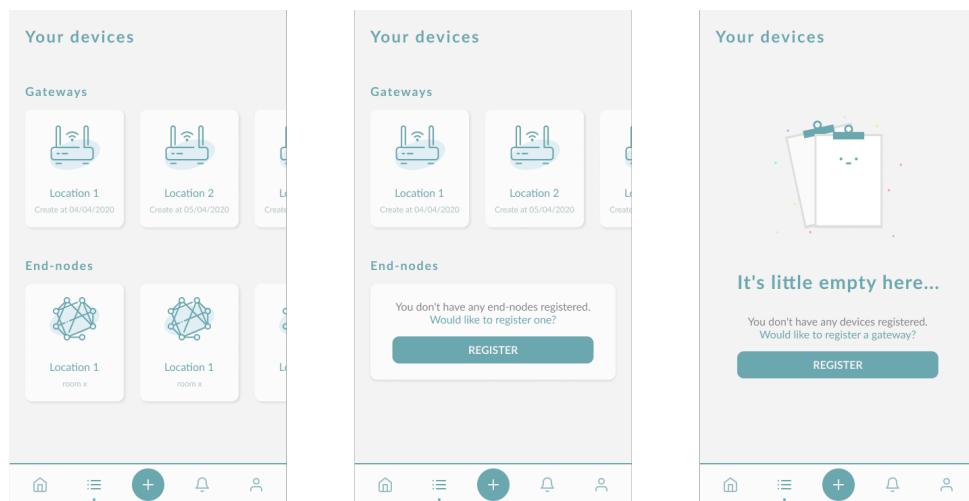


Figura 22 – Dispositivos do usuário cadastrados (Autoral).

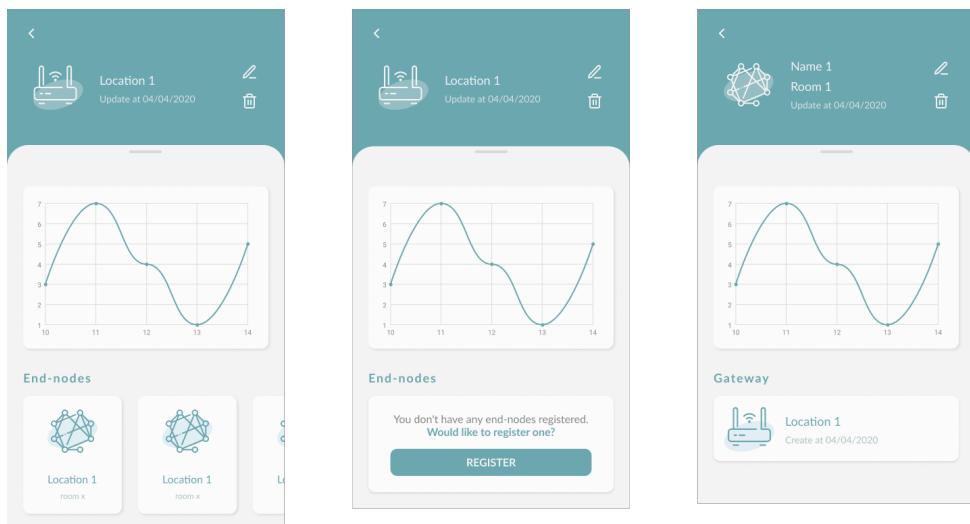
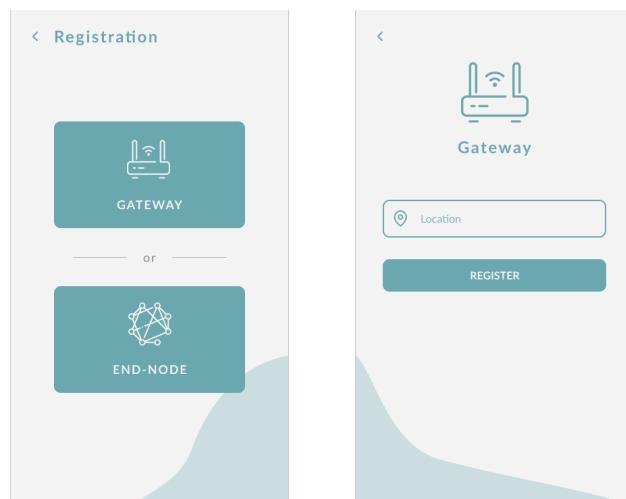
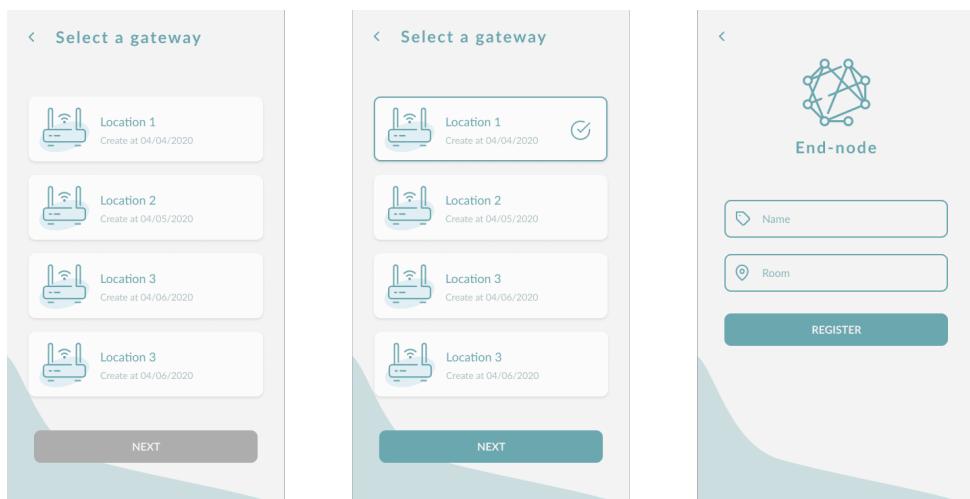


Figura 23 – Dados referente a um dispositivo (Autoral).

Figura 24 – Registro de um *gateway* (Autoral).Figura 25 – Registo de um *end-node* (Autoral).