UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

FRANCIELLE DE LIMA BROZOSKI

LEONALDO BARBOSA DA SILVA JUNIOR

SISTEMA DE MONITORAMENTO REMOTO DO NÍVEL DE CAIXAS D'ÁGUA EMPREGANDO A TECNOLOGIA LORAWAN

CURITIBA

2022

FRANCIELLE DE LIMA BROZOSKI
LEONALDO BARBOSA DA SILVA JUNIOR

SISTEMA DE MONITORAMENTO REMOTO DO NÍVEL DE CAIXAS D'ÁGUA EMPREGANDO A TECNOLOGIA LORAWAN

Monitoring System of Water Tank Using LoraWAN Technology

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de bacharel em engenharia eletrônica do curso de bacharelado em engenharia eletrônica da Universidade Tecnológica Federal do Paraná.

Orientador: Profª. Drª. Simone Crocetti

Coorientador: Dr. Márcio Luiz Ferreira Miguel

CURITIBA

2022

FRANCIELLE DE LIMA BROZOSKI
LEONALDO BARBOSA DA SILVA JUNIOR

SISTEMA DE MONITORAMENTO REMOTO DO NÍVEL DE CAIXAS D'ÁGUA EMPREGANDO A TECNOLOGIA LORAWAN

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de bacharel em engenharia eletrônica do curso de bacharelado em engenharia eletrônica da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 02/agosto/2022

Simone Crocetti
Doutora
Universidade Tecnológica Federal do Paraná

Luiz Fernando Copetti
Mestre
Universidade Tecnológica Federal do Paraná

Bruno Sens Chang
Doutor
Universidade Tecnológica Federal do Paraná

Márcio Luiz Ferreira Miguel
Doutor

CURITIBA
2022

As nossos pais, avós, tios, irmãos, amores e a todos de nossas famílias que, com muito carinho e apoio, não mediram esforços para que chegássemos até esta etapa de nossas vidas.

AGRADECIMENTOS

Agradecemos de maneira especial a nossos orientadores, professora Dra. Simone Crocetti e Dr. Márcio Luiz Ferreira Miguel que não mediram esforços para chegarmos à conclusão deste trabalho, sempre com muita paciência, incentivo e agilidade para nos atender, e por fazerem papel de verdadeiros mestres neste capítulo tão decisivo de nossas vidas. Agradecemos imensamente nossos familiares, que acreditaram em nossa capacidade e desde nossa tênue infância, não deixaram de nos aconselhar e de acompanhar de perto nossa evolução como profissionais, bem como nos ensinaram valores indispensáveis para a vida, contribuindo sempre para a formação de nosso caráter. Não poderíamos deixar de agradecer, de forma geral, a nossos colegas de curso, professores e todos os servidores da UTFPR Campus Curitiba, que direta ou indiretamente entraram em nossas vidas acadêmicas, proporcionando um ambiente de constante aprendizado. Agradecemos a nossos companheiros, que decidiram abdicar de seu tempo em favor da elaboração deste trabalho e todo este projeto de vida que envolve o curso de Engenharia Eletrônica, numa renomada instituição de ensino federal, o qual demandou de nós um empenho em tempo integral ao longo dos anos de formação. Enfim, agradecemos a todas as pessoas que ficaram ao nosso lado nesta preciosa trajetória, que certamente marcaram nossa história e jamais serão esquecidas. Agradecemos a todo o esforço dedicado a nós, seja no apoio intelectual ou emocional nos desafios que esta incrível jornada nos possibilitou até aqui. Agora animados seguimos para novos desafios na esperança de ter sempre por perto tanto nossos familiares e amigos, quanto nossos mentores.

“Pesquisar é acordar para o mundo.” (Marcelo Lamy)

RESUMO

O trabalho apresenta o desenvolvimento de um dispositivo com baixo consumo de energia para medição de nível em caixas d'água, sendo composto por microcontrolador ESP32 e rádio LoRa, com alimentação feita por pilhas alcalinas e comunicação via protocolo LoRaWAN, a qual é uma alternativa para aplicações que envolvem baixo consumo de energia, longo alcance e pequenas transmissões de dados. A exibição dos resultados da medição se dá po intermédio de aplicativo para dispositivos móveis desenvolvido com a biblioteca Javascript React Native para a plataforma Android e iOS. O hardware desenvolvido é microncontrolado e os dados obtidos por meio de sensor ultrassônico, para aferição de nível dos reservatórios, pilhas com autonomia acima de dois anos e transceptor LoRa de longo alcance, para envio dos dados. São utilizados *gateways* LoRa e concentrador para a recepção e tratamento pela comunidade TTN(*The Things Network*). O *backend* foi implementado na plataforma de desenvolvimento *open-source* .NET em linguagem de programação *C# (sharp)*. A aplicação foi hospedada na nuvem no provedor de serviço da Microsoft chamado Azure. O sistema propõe uma solução de fácil instalação e baixa manutenção para que o público em geral possa consultar níveis em caixas d'água de maneira fácil e segura visto que esses reservatórios se encontram, em geral, em locais de difícil acesso.

Palavras-chave: lora; lorawan; hardware; aplicativo; usuário.

ABSTRACT

This work presents the development of a device used to measure the water level on water tanks. The device has low power consumption and is composed by ESP32 micro controller and LoRa radio powered by alkaline batteries and communication by LoRaWAN protocol that is an alternative to low power consumption and low data transfer applications. The display of measure results is given through application for mobile devices developed with the Javascript React Native library based on Android and iOS. The hardware built is micro controlled and the data obtained due to ultrasonic sensor to measure the level of the tanks. Batteries with over two-year autonomy and long-range transceiver LoRa for data sending. LoRa gateways are used and concentrator for receiving and data processing by TTN (The Things Network) community. The backend was developed on .NET which is an open-source platform using C# language. The application was hosted by Microsoft Azure, a cloud-based service. The device proposes an easy installation with low maintenance required providing the customers to consult their tank level in an easy and safe way because the tanks are usually placed on a hard-to-reach location.

Keywords: lora; lorawan; hardware; application; user.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Siglas

3D	Tridimensional
ABNT	Associação Brasileira de Normas Técnicas
API	Application Programming Interface
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check)
GPIO	General Purpose Input/Output
HTTP	Hyper Text Transfer Protocol
ID	Identity
IDE	Integrated Development Environment
IOT	Internet of Things
IP	Internet Protocol
ISM	Industrial, Scientifc and Medical
LoRa	Long Range
LPWAN	Low Power Wide Area Network
LTE	Long Term Evolution
MAC	Media Access Control
MCU	Microcontroller Unit
PCI	Placa de Circuito Impresso
RSSI	Received Signal Strength Indication
SNR	Signal to Noise Ratio
SQL	Standard Query Language
TCP	Transmission Control Protocol

URL	Uniform Resource Locator
USB	Universal Serial Bus
UTFPR	Universidade Tecnológica Federal do Paraná
VCC	Tensão em Corrente Contínua
WCDMA	Wide-Band Code-Division Multiple Access

LISTA DE SÍMBOLOS

LETRAS GREGAS

μ	prefixo micro	$[10-3]$
π	número pi	

SUMÁRIO

1 INTRODUÇÃO

A Terra possui em torno de 71% da superfície composta de água em estado líquido. No entanto, aproximadamente 97% deste total é de água salgada, imprópria para consumo, uma vez que fazem parte dos oceanos. Sendo a disponibilidade de água doce de apenas 0,3% do volume total presente no mundo. Por esse motivo, a água potável e própria para consumo é um dos recursos mais raros do planeta (SILVA, 2020).

Sabe-se que o abastecimento de água é de suma importância para a sobrevivência humana e precisa ser fornecido com qualidade e regularidade à população urbana (FARMANI; WALTERS; SAVIC, 2005). Portanto, o risco de desabastecimento de água decorrente da possibilidade de falhas nos sistemas de abastecimento e de distribuição de água, pode afetar consideravelmente seus usuários (MARINHO *et al.*, 2021).

No final de 2021, o Sistema Cantareira, principal sistema hídrico responsável por abastecer a Região Metropolitana de São Paulo, fechou o mês com 28% do volume útil de seus reservatórios e operou na faixa de restrição. Segundo o Centro Nacional de Monitoramento e Alertas de Desastres Naturais (Cemaden), o Brasil tem enfrentado períodos de seca mais longos em todas as regiões (CEMADEN, 2021).

Nota-se que, atualmente, esse cenário não ocorre apenas nas regiões Norte e Nordeste do Brasil, pois também passou a atingir as regiões Sul, Sudeste e Centro-Oeste. As consequências desse cenário são a redução nos níveis dos reservatórios do sistema hídrico e o desabastecimento residencial. Outro agravante é que a agricultura prevê aumentar em 66% o uso da água para irrigar áreas de plantio até 2040, hoje o consumo de água proveniente da agricultura já é de 49,8% no Brasil (OJC, 2021).

Com isso, muitos lares brasileiros foram afetados por períodos de racionamento. Segundo Sistema de Tecnologia e Monitoramento Ambiental do Estado do Paraná (Simepar), o Paraná, teve redução histórica no indíce de chuvas entre janeiro de 2020 e agosto de 2021, o que reduziu drasticamente o volume disponível nos reservatórios do estado (FOLHA DE LONDRINA, 2021). O rodízio no abastecimento de água em Curitiba implantado durante o período de pior estiagem da história do estado do Paraná, fez com que a população adquirisse caixas d'água maiores e aqueles que não possuíam reservatórios em suas residências, passaram a ter a preocupação em as instalar, como forma de prevenção à falta de água durante os períodos de racionamento. A procura por caixas d'água aumentou em 40% as vendas deste item em 2020 em Curitiba e região (GAZETA DO POVO, 2020).

Dada a condição de crise hídrica no Paraná e no Brasil, este trabalho foi pensado a fim de auxiliar os usuários a mensurar o volume de água disponível em caixas d'água. Em 2021 devido a forte estiagem na região sul, decretos municipais foram estabelecidos para interrupção no abastecimento por até 36h. Por isso, o presente trabalho propõem-se a ajudar na diminuição dos impactos da falta d'água de maneira a proporcionar informações precisas do volume de

água disponível no reservatório do usuário, a fim de que esse possa tomar decisões de maneira a preservar ou não o recurso até o fim de cada período de rodízio.

Dessa forma, a fim de desenvolver o projeto, visando baixo custo, baixo consumo de energia, com interface com o usuário por meio de aplicativo para smartphone utilizou-se o microcontrolador ESP32 com tranceptor de rádio LoRa do inglês, *long range*, que significa longo alcance. Esse componente de rádio frequência, foi utilizado juntamente com um sensor ultrassônico à prova d'água. Sendo assim, este projeto visou combinar fatores importantes, fácil instalação e manutenção, uma vez que em geral, os reservatórios residenciais se encontram em locais altos e de difícil acesso, como em lajes ou forros.

A proposta foi criar um produto funcional e independente, que não precise ser instalado na rede elétrica e que cuja a comunicação não se dê por recursos do próprio usuário como Wi-Fi ou Bluetooth. Assim, optou-se por utilizar comunicação via rádio LoRa, por ser uma tecnologia indicada para projetos *IoT*, do inglês, *internet of things* por ter longo alcance e baixo consumo de energia, além de não depender de outro dispositivo para funcionamento.

1.1 Objetivo geral

Desenvolver um dispositivo microcontrolado portátil, capaz de medir nível de água em reservatórios de água, do tipo caixa d'água em formato tronco de cone, normatizados conforme NBR14799 (ABNT, 2011) e NBR15682 (ABNT, 2009). Além disso, deve ser de aplicação doméstica, possuindo interface com o usuário por meio de aplicativo para smartphone Android, de fácil instalação, alimentado por pilhas alcalinas convencionais com duração de dois anos sem necessidade de troca das pilhas.

1.1.1 Objetivos específicos

Para atingir esses resultados, o trabalho foi dividido nas seguintes etapas:

- Revisar a bibliografia sobre microcontroladores da família ESP32 e comunicação *LoRaWAN*.
- Desenvolver *hardware* e *firmware* de controle do circuito.
- Design de embalagem do dispositivo *ANADevice*, com proteção contra água e poeira, segundo recomendações IP66.
- Desenvolver *backend* e banco de dados, para obtenção dos dados do dispositivo recebidos pelo *Gateway LoRaWAN* e armazenados na nuvem TTN para posterior utilização e visualização em aplicativo.
- Desenvolver aplicativo de interação com o usuário.

1.2 Justificativa

O projeto foi pensado para atender as necessidades do mercado atual, uma vez que o racionamento de água vem sendo recorrente e as pessoas ficam sem saber ao certo o quanto de água possuem para consumo. Como, em geral as caixas d'água, não estão às vistas dos usuários, seu acesso muitas vezes é difícil, então não há meios de verificar a quantidade de água de forma prática e rápida. Por isso o projeto foi desenvolvido, para facilitar a gestão do recurso hídrico por parte do usuário. Adicionalmente, contribuirá para a conscientização sobre o uso excessivo deste recurso, pois a visualização de nível por meio do aplicativo se dará de hora em hora, podendo o usuário visualizar nesse período ou não. Caso visualize, poderá perceber a diferença de nível de forma contínua devido ao uso pelos integrantes da residência em que o dispositivo estará instalado.

O produto, oriundo deste projeto de pesquisa, recebe o nome de ANADevice, a sigla ANA significa Análise de Nível de Água, sendo que o aplicativo recebe o nome de ANAApp. Este projeto se diferencia dos demais presentes no mercado, por possuir fácil instalação e fácil manutenção. Visto que a instalação não envolve uso de ferramentas, nem é necessário furar ou alterar as condições da caixa d'água, pelo contrário, basta posicionar o sensor na parte interna e central da caixa, ao passo que deve-se também posicionar os ímãs na parte superior central da caixa, assim o sensor estará fixado e pronto para coletar dados. Também é necessário posicionar a caixa do dispositivo ANADevice, a partir de seu suporte, que deverá ficar posicionado na borda da caixa d'água e presa pela tampa da mesma. Fazendo isso, o dispositivo estará instalado por completo e pronto para o funcionamento. Além disso, o ANADevice, não requer manutenção e caberá ao usuário apenas trocar as pilhas alcalinas a cada dois anos de uso.

Há produtos no mercado que se destinam ao mesmo fim, porém a instalação exige que o usuário faça conexão com a rede elétrica, uma vez que o equipamento funciona alimentado por tensões alternadas de 127V ou 220V. A dificuldade se dá pelo fato de que, em geral, as pessoas não possuem tomadas ao redor de suas caixas d'água, pois elas ficam acima da estrutura do imóvel onde residem. Além disso, para tais dispositivos vendidos atualmente, é necessário instalar um painel em um local específico, dentro da residência, para a aferição da quantidade de água disponível. Esse painel em geral possui apenas quatro leds cada um correspondendo a 25% da capacidade da caixa d'água, o que gera um erro considerável. Além de não ser portátil, o usuário precisa estar fisicamente em frente ao painel para visualizar a quantidade de água estimada (MERCADO LIVRE, 2022).

Por esse motivo, este projeto visa a facilidade na instalação, além de facilitar a aferição por parte do usuário, uma vez que poderá ser realizada de qualquer lugar, pois depende apenas de um smartphone com conexão com internet. O usuário também não precisará se preocupar com comunicação Bluetooth ou wi-fi, o projeto foi concebido visando comunicação de longo alcance por rádio, escolhida por motivo de a localização física de caixas d'água ser em geral em andares superiores aos de uso comum da residência, o que dificultaria a comunicação por Blu-

etooth ou Wi-Fi que necessitam de poucas barreiras físicas entre fonte e receptor, uma vez que possuem baixo alcance. Sendo assim, o dispositivo foi pensado para não depender da internet da residência ou qualquer outro dispositivo que o usuário precise possuir previamente, apenas será necessário um smartphone com conexão de internet para simples visualização dos dados recebidos pela rede LoRaWAN que do inglês *Long Range Wide Area Network*, que significa rede de longo alcance, a qual compartilhará os dados do dispositivo ANADevice, permitindo o acesso de qualquer lugar.

1.3 Procedimentos metodológicos

Na realização deste trabalho foram utilizados como base outros trabalhos de produção científica disponíveis, *datasheets*, catálogos, vídeos, cursos online e fóruns de programação, montagem de circuitos, manuais *makers* dentre outros materiais disponíveis na internet. Foram analisadas as propriedades de cada componente utilizado, bem como realizada revisão bibliográfica dos assuntos abordados, como geometria, comunicações digitais, programação, engenharia de produto, viabilidade econômica, circuitos elétricos, microcontroladores dentre outras referências que puderam ser encontradas dentro de nossa formação acadêmica em Engenharia Eletrônica.

A seguir, um protótipo foi projetado e montado, incluindo a fabricação artesanal, de placa de circuito impresso. Com isso, os primeiros testes foram iniciados. Juntamente com a prototipagem, foi iniciada a fase de desenvolvimento de *firmware* para controle do sistema microcontrolado, simultaneamente, foi iniciado o desenvolvimento de *software* de interação com o usuário do produto final, aplicação *backend* para coleta e transferência de dados e implementação de banco de dados em nuvem. Com isso foram sendo definidos os passos que se sucederam como testes de integração de *hardware* mais *software*, análise, processamento, ajuste e envio de dados coletados pelo circuito em questão para que se fosse possível obter boa acuracidade e confiabilidade na interface com o usuário. Em seguida foram executados testes do produto final em ambiente real, foram analisados dados de consumo e adaptabilidade em diferentes cenários.

1.3.1 Especificações do projeto

Para que fosse possível desenvolver o projeto, obtendo por resultado um produto eficiente, criou-se uma lista de requisitos funcionais e não-funcionais para a delimitação do escopo do projeto. Esses requisitos resumem as características do produto, como construção, implementação e interface com usuário.

1.3.1.1 Requisitos funcionais (RF)

RF1: A integração entre *TTN* e servidor deve ser feita através de *Webhook*.

RF2: Os dados recebidos através do *Webhook* devem ser armazenados em banco de dados.

RF3: O *backend* deve cadastrar novos dispositivos de forma automática.

RF4: O *backend* deve realizar os cálculos de volume atual e salvar no banco de dados.

RF5: O *backend* deve realizar uma consulta no banco de dados para verificar se o usuário está cadastrado através de um pedido do tipo *HttpGet* do aplicativo com o parâmetro identificador único do usuário para realizar a pesquisa.

RF6: Todo pedido processado pelo *backend* que não resulte nos dados solicitados no banco banco de dados deve retornar o código *HTTP 404* (não encontrado).

RF7: O *backend* deve cadastrar novas caixas d'água através de um pedido *HttpPost* do aplicativo com os parâmetros o ID do usuário, ID do dispositivo, o ID da caixa d'água e o nome da caixa d'água.

RF8: O *backend* deve retornar todos os dispositivos cadastrados para um usuário através de um pedido *HttpGet* do aplicativo com o parâmetro o ID do usuário.

RF9: O *backend* deve retornar a última leitura realizada por um dispositivo através de um pedido *HttpGet* do aplicativo com o parâmetro o ID da caixa d'água.

RF10: O *backend* deve retornar todas as leituras realizadas por um dispositivo através de um pedido *HttpGet* do aplicativo com o parâmetro o ID da caixa d'água.

RF11: O *backend* deve retornar todas as leituras realizadas por um dispositivo por um período de tempo, através de um pedido *HttpGet* do aplicativo com os parâmetros ID da caixa d'água e número de dias.

RF12: O *backend* deve retornar qual dispositivo o usuário deseja visualizar os dados através de um pedido *HttpGet* do aplicativo com o parâmetro ID do usuário.

RF13: O *backend* deve atualizar o dispositivo selecionado pelo usuário através de um pedido *HttpPatch* do aplicativo com o parâmetro ID do usuário e ID da caixa d'água.

RF14: O *backend* deve deletar um caixa d'água cadastrada pelo usuário através de um pedido *HttpDelete* do aplicativo com o parâmetro ID da caixa d'água.

RF15: O Aplicativo deve possuir uma tela inicial de carregamento e mostrar o logotipo ANA no centro da tela.

RF16: O Aplicativo deve possuir uma tela *Meu Nível* onde deve ser exibido:

- RF16:1. Nome da caixa d'água.
- RF16:2. Leitura atual em litros.
- RF16:3. Data no formato dia/mês/ano (dd/mm/yyyy).
- RF16:4. Hora no formato hora:minuto:segundo (hh:mm:ss).
- RF16:5. Mostrar de forma gráfica o nível em porcentagem com intervalo de 5%.

RF17: A tela *Meu Nível* deve possuir dois botões:

- RF17:1. Dispositivos: deve redirecionar o usuário para a tela "Dispositivos".
- RF17:2. Histórico: deve redirecionar o usuário para a tela "Histórico".

RF18: Toda a tela deve possuir um cabeçalho:

- RF18:1. A tela "Meu nível"deve possuir ao lado esquerdo o logotipo do projeto.
- RF18:2. Todas as outras telas, devem possuir do lado esquerdo um ícone de seta, que quando pressioando redireciona o usuário para a tela anterior.
- RF18:3. Todas as telas devem exibir ao centro o nome da tela.
- RF18:4. Todas as telas devem possuir o ícone da bateria, ao lado direito, que quando pressionado direciona o usuário para a tela "Histórico da bateria".
- RF18:5. O ícone da bateria deve variar de acordo com o seu atual nível, o intervalo deve ser de 20%.

RF19: A tela *Dispositivos* deve possuir dois botões:

- RF19:1. Novo Dispositivo: deve redirecionar o usuário para a tela "Novo Dispositivo".
- RF19:2. Apagar Dispositivo: Quando pressionado, deve apagar o dispositivo selecionado.

RF20: A tela *Dispositivos* deve possuir um campo do tipo *drop down list*, quando o usuário pressiona esse campo deve ser exibida uma lista com todas as caixas d'água registradas.

RF21: A tela *Dispositivos* quando o usuário selecionar uma caixa d'água deve exibir as seguintes informações:

- RF21:1. Dispositivo selecionado: nome do reservatório.
- RF21:2. Marca: marca do fabricante do reservatório.
- RF21:3. Volume: máximo volume do reservatório em litros.

RF21:4. Raio da base: raio da base do reservatório em metros de acordo com o catálogo do fabricante.

RF21:5. Raio do topo: raio do topo do reservatório em metros de acordo com o catálogo fabricante.

RF21:6. Raio do topo: altura do reservatório em metros de acordo com o catálogo do fabricante.

RF22: A tela *Novo dispositivo* deve permitir que o usuário registre um reservatório atrelado a um dispositivo. A tela deve conter os seguintes campos:

RF22:1. ID do dispositivo: campo do tipo *input text* para que o usuário digite o identificador único do dispositivo.

RF22:2. Nome do dispositivo: campo do tipo *input text* para que o usuário digite o nome desejado para se refetir ao reservatório.

RF22:3. Selecione a caixa d'água: campo do tipo *drop down list* para o usuário selecionar uma opção de reservatório.

RF22:4. Salvar: botão que permite ao usuário salvar as informações digitadas.

RF22:5. Salvar: o botão deve permanecer desativado, impedindo que o usuário o clique caso os campos ainda não tenham sido preenchidos.

RF23: A tela *Histórico do nível* deve permitir que o usuário consulte todas as leituras de nível realizadas pelo dispositivo no formato tabela.

RF24: A tela *Histórico do nível* deve exibir os seguintes dados:

RF24:1. Volume (%): Volume em porcentagem.

RF24:2. Volume (L): Volume em litros.

RF24:3. Data: data no formato dia/mês/ano (dd/mm/yyyy).

RF24:4. Hora: hora no formato hora:minuto:segundo (hh:mm:ss).

RF25: A tela *Histórico da bateria* deve permitir que o usuário consulte todas as leituras de nível de bateria realizadas pelo dispositivo em formato tabela.

RF26: A tela *Histórico da bateria* deve exibir os seguintes dados:

RF26:1. Bateria (%): nível da bateria em porcentagem.

RF26:2. Bateria (L): nível da bateria em volts.

RF26:3. Data: data no formato dia/mês/ano (dd/mm/yyyy).

RF26:4. Hora: hora no formato hora:minuto:segundo (hh:mm:ss).

1.3.1.2 Requisitos não funcionais

RF1: O sistema deve ser microcontrolado.

RF2: O microncontrolador deve ser o ESP WROOM 32.

RF3: O sistema deve possuir um sensor ultrassônico impermeável JSN-SR04T.

RF4: O projeto deve ser alimentado por duas pilhas alcalinas AA de 1.5 V cada.

RF5: As pilhas devem durar por pelo menos dois anos sem serem substituídas.

RF6: O consumo do sistema no modo *deepsleep* deve ser abaixo de 50μAh.

RF7: O *hardware* deve possuir um suporte para duas pilhas AA.

RF8: O *hardware* deve possuir um circuito para aferição da tensão do conjunto de pilhas.

RF9: O *hardware* deve possuir um módulo LoRa RFM95W.

RF10: O *hardware* deve possuir dois botões, sendo um para *download* do *firmware* e outro para o *reset* do microcontrolador.

RF11: Deve ser utilizada a plataforma EasyEDA para projetar a placa de circuito impresso.

RF12: A placa de circuito impresso deve ser de única face.

RF13: A placa deve possuir plano de terra.

RF14: A placa deve possuir as medidas de 7x14 cm.

RF15: A caixa deve ter especificações IP66, proteção contra água e poeira.

RF16: O cabo que conecta o sensor ultrassônico na placa pcb deve ter no mínimo 1m de comprimento.

RF17: A fixação do sensor deve ser feita através de ímãs de neodímio de 10 mm de diâmetro.

RF18: O firmware deve ser desenvolvido utilizando Arduino IDE.

RF19: O sistema precisa utilizar o banco de dados SQL server.

RF20: Deve ser utilizada a *IDE Microsoft SQL Server Management Studio 18* para o desenvolvimento do banco de dados.

RF21: O banco de dados deve ser hospedado pela empresa *ITMNetworks*

RF22: O servidor deve ser desenvolvido em *.NET C#*.

RF23: Deve ser utilizada a *IDE Visual Studio 2022* para o desenvolvimento do *backend*.

RF24: O servidor deve ser um projeto do tipo *WEB API*.

RF25: O servidor deve ser hospedado em *cloud* pelo provedor de serviços *Microsoft Azure*.

RF26: O aplicativo deve ser desenvolvido em *React Native*.

RF27: Deve ser utilizada a *IDE Studio Code* para o desenvolvimento do aplicativo.

RF28: O aplicativo deve ser compatível com as plataformas *Android e IOS*.

1.4 Estrutura do trabalho

De forma a organizar o desenvolvimento do trabalho e apresentação de resultados obtidos, o presente documento foi dividido em cinco partes: introdução, referencial teórico, desenvolvimento, resultados e conclusão. No início do trabalho são apresentados a introdução e a abordagem do tema, que corroboram o problema que o produto que originou este trabalho, se propõem a resolver, além dos objetivos e da justificativa da escolha do tema, visando o detalhamento de cada objetivo. Também são expostos os procedimentos metodológicos, incluindo os requisitos funcionais e não funcionais do projeto, bem como a forma de organização do trabalho de maneira estruturada. Na segunda parte, foi explorado o referencial teórico utilizado para atingimento dos resultados, visando a explicação clara dos aspectos teóricos adotados e o porquê deste trabalho propor uma solução diferenciada, buscando explanar o desenvolvimento do projeto como um todo. A terceira seção foi dedicada a especificar as etapas de desenvolvimento do protótipo do produto esperado como resultado do presente trabalho de pesquisa. Na quarta parte são demonstrados os testes realizados e os resultados obtidos, além de realizadas as devidas discussões sobre os resultados. Por fim, na última seção são abordados comentários pertinentes ao término do projeto e realizadas as devidas conclusões.

2 REFERENCIAL TEÓRICO

Nesta seção realiza-se o levantamento do referencial teórico utilizado como base para a elaboração deste trabalho. Estabele-se alguns conceitos para a construção apropriada para alcançar o correto funcionamento do produto proposto. Visa-se explicar o tipo de comunicação escolhida, bem como explicitar o estudo dos componentes escolhidos, assim como o desenvolvimento de firmware e software envolvidos, linguagens de programação escolhidas e motivos para a escolha das plataformas utilizadas, do mesmo modo que visa evidenciar dados obtidos, com a finalidade de proporcionar um bom entendimento e visão geral do projeto para o leitor.

2.1 Comunicação LoRa

2.1.1 A Rede LoRaWAN

LoRa é uma tecnologia de comunicação de dados sem fio, desenvolvida e mantida pela empresa Semtech. Essa tecnologia de comunicação tem por características possuir longo alcance, baixo consumo de energia e transmissão segura de dados (MARAIS; MALEKIAN; ABU-MAHFOUZ, 2017).

Ainda que LoRaWAN seja entendida como todo o sistema de comunicação LPWAN, (do inglês *Low Power Wide Area Network* que significa rede de baixa potência), LoRa diz respeito apenas à técnica de modulação proprietária da Semtech™. LoRaWAN é um padrão aberto que define o sistema LPWAN (MARAIS; MALEKIAN; ABU-MAHFOUZ, 2017).

Outras empresas usam camadas Media Access Control (MAC) proprietárias em um chip LoRa o que cria um *design* composto e de performance diferenciada, por isso é importante ressaltar que a rede LoRaWAN trata-se de uma plataforma aberta e contributiva, isso explica a variedade de dispositivos que utilizam a tecnologia e estão disponíveis no mercado.

A tecnologia foi projetada para resolver alguns problemas de rede para dispositivos Internet of Things (IOT), como consumo de energia dos dispositivos finais. Em geral, esses dispositivos possuem sensores que precisam enviar dados continuamente para servidores ou *gateways*, o que pode levar a uma duração reduzida de bateria. Outro problema que a tecnologia LoRa se propõem a resolver diz respeito ao alcance, pois os dispositivos IoT devem, estar localizados ao alcance da rede para que a comunicação se estabeleça de forma confiável, de um ponto de vista de disponibilidade. Para aplicações comuns como em agronomia ou em sistemas de automação residencial, a distância entre o *gateway* (receptor) e os dispositivos finais pode variar de alguns metros a vários quilômetros.

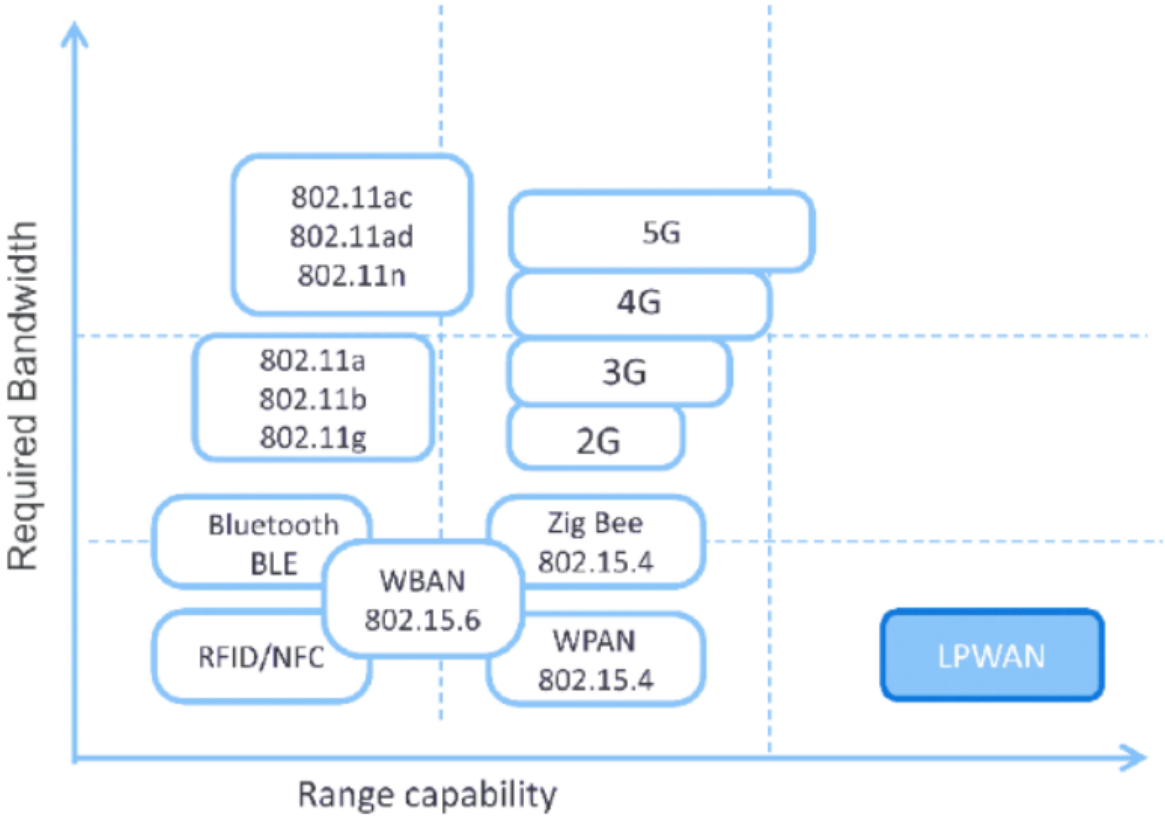
Como a tecnologia LoRa utiliza comunicação por rádio frequência de baixo consumo e longo alcance, a rede pode ser composta por uma pequena teia de *gateways* que garantem uma grande cobertura. Outra vantagem é que qualquer pessoa pode instalar um *gateway*, que

funciona como um receptor para os dispositivos finais, em qualquer lugar, facilitando assim o espalhamento dessa tecnologia e conectando dispositivos.

É importante ressaltar que a tecnologia é recomendada para aplicações em que se deseja obter poucos dados em intervalos de tempo pré-definidos. Para aplicações com vasta transmissão de dados como telemetria aeroespacial por exemplo, LoRa pode não ser a melhor escolha.

A Figura 1 mostra a comparação entre o alcance de diferentes tecnologias. Em termos de distância entre emissor e receptor, pode-se notar que a rede LoRaWAN ou Low Power Wide Area Network (LPWAN) tem maior alcance, quando comparado a algumas tecnologias muito utilizadas, como as de telefonia móvel e até Bluetooth. Contudo, o poder de transporte de dados da rede LoRaWAN é pequeno, por isso seu uso é restrito a projetos em que se deseja transmitir poucos dados, o contrário do Bluetooth por exemplo. Por isso, é importante ressaltar que cada tecnologia tem suas limitações e precisa ser empregada de acordo com os requisitos de cada projeto. Em geral as tecnologias que transportam muitos dados tem baixo alcance, já as de longo alcance tem capacidade reduzida de transmissão de dados.

Figura 1 – Comparação entre alcance e largura de banda entre tecnologias IoT



Fonte: (MCCLELLAND, 2017).

A rede LoRaWAN é dividida em camadas e possui componentes físicos e digitais. No nível mais baixo estão os dispositivos de sistemas finais, ou nós, que possuem o componente

de rádio LoRa e se comunicam com os demais pontos da rede. Sua funcionalidade, tipo de alimentação, vazão de dados e frequência de envio de dados dependerão da construção do dispositivo final e para que fim foi projetado, pois há inúmeras variações de aplicação.

Os dispositivos finais da rede LoRaWAN, utilizam o protocolo *Long Range Wide Area Network (LoRaWAN protocol)*, com isso podem transmitir e/ou receber dados de um gateway, conectados por meio da rede LoRaWAN que possui topologia do tipo *Star-of-Stars*, ou estrela de estrelas. Os *gateways* são como pontes entre os nós e a internet, eles realizam a transmissão e recepção de pacotes por meio de um servidor (GTA UFRJ, 2019).

2.1.2 Topologia de rede

As frequências de operação LoRa variam para diferentes regiões do globo, a rede utiliza frequências Industrial, Scientifc and Medical (ISM), ou médica científica industrial, que são frequências de uso não restrito. Isso quer dizer que não é necessária nenhuma licença para o uso dessas frequências, qualquer pessoa com um dispositivo LoRa pode utilizar livremente a rede, instalar *gateways* caso não hajam na região em que se deseja utilizar. Com isso facilita-se o crescimento da rede, pois o *gateway* poderá atender a inúmeros dispositivos. A Figura 2 mostra as frequências abertas mundialmente, definadas por região. São estas as frequências utilizadas pela rede LoraWAN.

Figura 2 – Frequências abertas mundialmente utilizadas

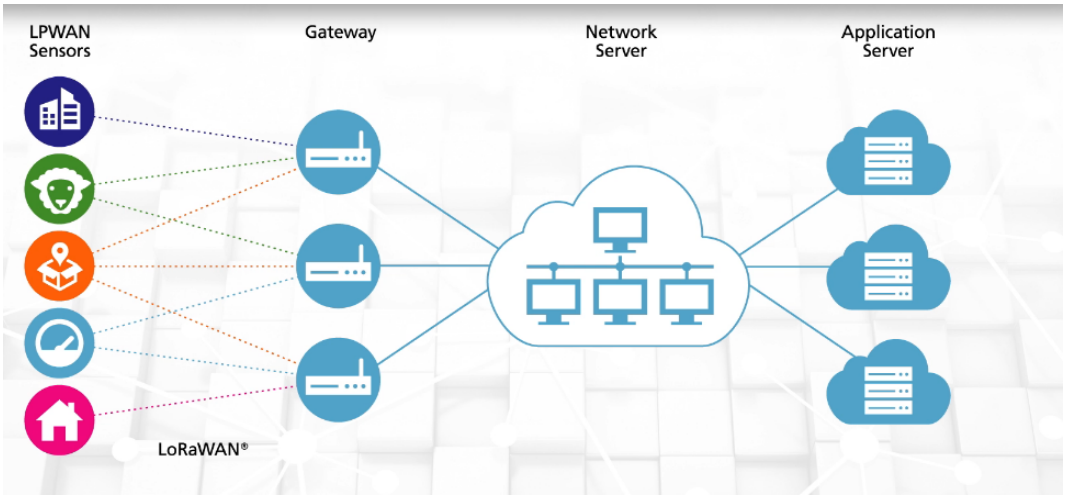
Fonte: (TECH DESIGN, 2021).

2.1.3 Modulação

A topologia utiliza essas frequências para estabelecer a comunicação dos dispositivos nós para os *gateways*, que por sua vez fazem a transmissão de dados para os servidores que utilizam os protocolos de internet Transmission Control Protocol (TCP)/Internet Protocol (IP). A

Figura 3 mostra os dispositivos *end devices*, ou dispositivos finais, à esquerda, que envolvem o uso de sensores aferindo dados pertinentes à cada situação para a qual foram projetados, sendo ainda que a segunda coluna de dispositivos representa os *gateways* da rede LoRaWAN que por sua vez recebem as informações dos *end devices* e então enviam para os servidores da TTN. Estes por fim compartilham as informações com aplicações criadas dentro ou fora da TTN. O esquema da rede é bastante simples, vale lembrar que qualquer pessoa pode instalar *gateways* se assim desejar, aumentando a capacidade da rede, pois trata-se de uma rede aberta e colaborativa.

Figura 3 – Topologia da rede LoRaWAN



Fonte: (GTA UFRJ, 2019).

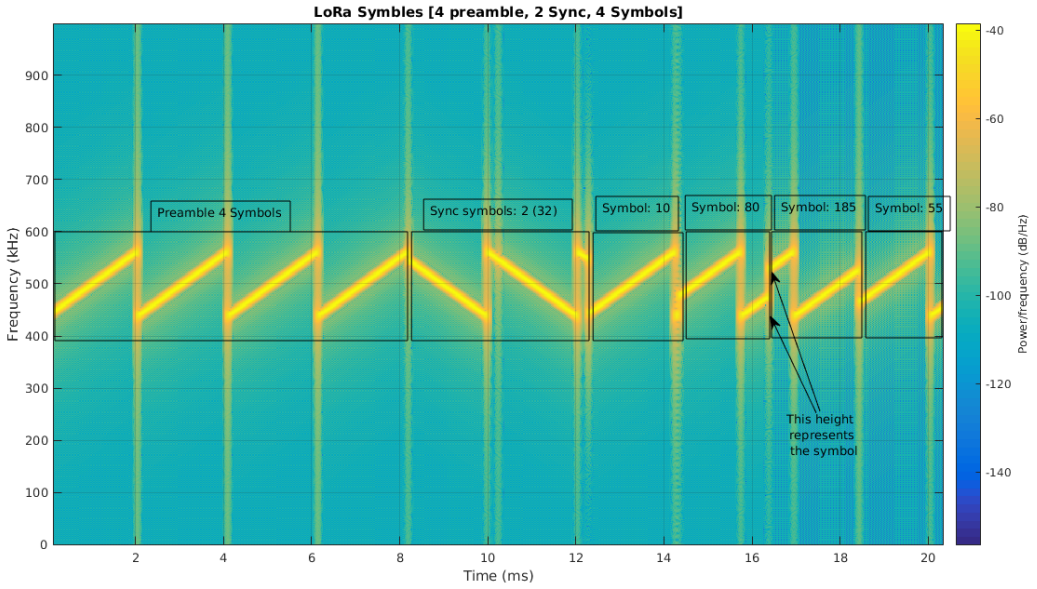
A tecnologia LoRa utiliza modulação do tipo *Chirp Spread Spectrum* (CSS), que foi desenvolvida para aplicação em sistemas de radar e para usos militares. Os sinais do tipo *Compressed High Intensity Radar Pulse* (Chirp) utilizam toda a largura de banda tendo amplitude constante, mas a frequência varia de maneira linear em determinado intervalo de tempo. Existem dois tipos de sinais, *up-chirp* e *down-chirp*, quando a frequência varia da maior para a menor dizemos *down-chirp* e se for da menor para a maior dizemos *up-chirp*.

Os sinais de frequência em relação ao tempo são do tipo rampa, que vão da frequência mínima até a máxima para o caso de *up-chirp* e o inverso para o caso de *down-chirp*. Os *chirps* que possuem deslocamento são os que transportam informação.

Na tecnologia LoRa, alguns parâmetros podem ser customizados, como o fator de espalhamento (SF), taxa de código (CR) e largura de banda (BW). Em caso de variação desses parâmetros, a taxa de *bits* efetiva da modulação será afetada, bem como sua resistência a ruído e sua decodificação.

Na Figura 4 nota-se que a duração de um *chirp* de fator de espalhamento N, será sempre equivalente ao dobro da duração de um pulso com fator de espalhamento N-1.

Figura 4 – Símbolos *LoRaWAN* análise de sinal



Fonte: (GHOSLYA, 2017).

Este tipo de modulação possui algumas vantagens como: baixa potência de transmissão, robustez à degradação de canal e resistência ao efeito Doppler, sendo também resistente ao multipercurso.

A relação entre a taxa de bits desejada de dados, a taxa de símbolos e a taxa de *chirp* da modulação LoRaWAN pode ser representada pela equação:

$$R_b = SF \cdot \frac{1}{\frac{2^{SF}}{BW}}, \tag{1}$$

Onde:

R_b = Taxa de bits [bits/s]

SF = Fator de espalhamento

BW = Largura de banda [Hz]

O período dos símbolos Ts pode ser definido por:

$$T_s = \frac{2^{SF}}{BW}, \tag{2}$$

Onde:

T_s = Periodo do símbolo [s]

SF = Fator de espalhamento

BW = Largura de banda [Hz]

Sendo que podemos definir a partir da segunda equação:

$$R_b = \frac{BW}{2^{SF}}, \tag{3}$$

Onde:
R_b = Taxa de bits [bits/s]
R_s = Taxa de símbolo [simbolos/s]
SF = Fator de espalhamento
BW = Largura de banda [Hz]

Com o auxílio das equações definidas, pode-se encontrar a taxa de *chirps* Rc:

$$R_c = R_s \dot{2}^{SF}, \tag{4}$$

Onde:
R_c = Taxa de *chirps*[simbolos/s]
R_s = Taxa de símbolo [simbolos/s]
SF = Fator de espalhamento

Substituindo a equação (3) em (4) pode-se constatar que a taxa de *chirp* da modulação LoRa depende apenas da largura de banda. A taxa de *chirp* é igual a largura de banda (um *chirp* por segundo por Hertz de largura da banda).

Esta modulação inclui também um código de correção de erro com tamanho variável, o que melhora a robustez do sinal transmitido.

Assim pode-se definir a taxa nominal de *bits* Rb como:

$$R_b = SF \frac{\frac{\dot{4}}{4+CR}}{\frac{2^S F}{BW}}, \tag{5}$$

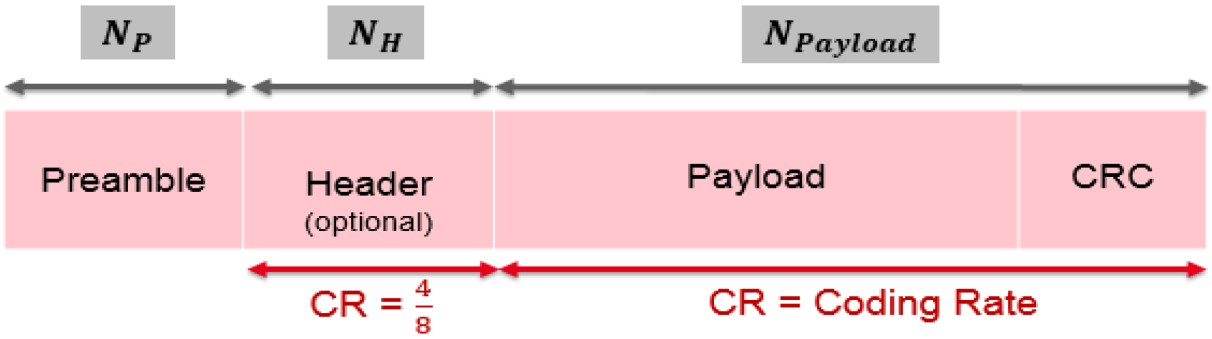
Onde:
SF = Fator de espalhamento
BW = Largura de banda [Hz]
CR = Taxa de codificação

2.1.4 Protocolo LoRaWAN

Com base na modulação LoRa, definiu-se um protocolo para operação dessa rede, chamado *LoRaWAN (Long Range Wide Area Network)*, para que se fosse possível codificar e decodificar as mensagens enviadas e recebidas pela rede (GTA UFRJ, 2019).

Existem dois tipos de pacote que são utilizados no protocolo LoRaWAN, o implícito e o explícito. O explícito trata-se do pacote com cabeçalho que contém informações como o número de *bytes*, taxa de codificação e o Cyclic Redundancy Check) (CRC) usado no pacote.

Figura 5 – Estrutura do pacote



Fonte: (BOUGUERA *et al.*, 2018).

Com base na Figura 5, acima, pode-se observar as partes do pacote. O *preamble*, ou preâmbulo, é usado para detectar o início da comunicação e da sincronização, que por padrão possui 12 símbolos. Já a *header*, ou cabeçalho, diz respeito à parte do pacote que pode ser configurada de maneira explícita ou implícita. Sendo o modo explícito o padrão normal de operação. Nela encontram-se informações como: tamanho do pacote e taxa de codificação. Tratando-se de *payload*, diz-se à parte do pacote que possui os dados de interesse, por exemplo dados de leitura do sensor. Por fim o CRC refere-se a um código de 16 bits, usado para verificar erros, utilizado somente na transmissão *uplink*.

2.1.5 Protocolo Cayenne LPP

Este protocolo foi desenvolvido para se trabalhar com a rede LoRaWAN de maneira mais fácil. O padrão *Cayenne Low Power Payload* (LPP), ou carga útil de baixa potência, tem por objetivo identificar sensores e seus dados, a fim de facilitar a leitura e a aquisição dos pacotes recebidos pelos *gateways* LoRA. Com sua utilização, é possível identificar se o *gateway* está recebendo uma leitura do tipo digital ou analógica, permitindo também que o dispositivo envie dados de múltiplos sensores ao mesmo tempo, respeitando o tamanho do pacote permitido pelo protocolo LoRaWAN.

Cada dado de um sensor é enviado utilizando 3 *bytes*, sendo que o primeiro byte se refere ao canal, o segundo ao tipo de dado e, finalmente, o terceiro ao dado em si. Com essas

Tabela 1 – Estrutura de payload Cayenne LPP

1 Byte	1 Byte	N Bytes	1 Byte	1 Byte	...
Dado1 Canal	Dado1 Tipo	Dado1	Dado2 Canal	Dado2 Tipo	...

Fonte: (Adaptado de My Devices, 2020).

informações o protocolo consegue entregar os dados já formatados e identificados. Sendo que, o usuário consegue verificar de maneira simples se o dado trata de medida analógica ou digital, de qual sensor foi gerado e seu respectivo valor medido.

Tabela 2 – Tipos de dados do protocolo Cayenne LPP

Tipo	IPSO	LPP	Hex	Tam. dado	Resolução do dado
Entrada Digital	3200	0	0	1	1
Saída Digital	3201	1	1	1	1
Entrada Analógica	3202	2	2	2	0.001
Saída Analógica	3203	3	3	2	0.001
Sensor de Luminosidade	3301	101	65	2	1
Sensor de Presença	3302	102	66	1	1
Sensor de Temperatura	3303	103	67	2	0.1℃
Sensor de Umidade	3304	104	68	1	0.5%
Acelerômetro	3313	113	71	6	0.001 G
Barômetro	3315	115	73	2	0.1 hPa
Giroscópio	3334	134	86	6	0.01 °/s
Localização GPS	3336	136	88	9	Latitude: 0.0001°
Localização GPS	3336	136	88	9	Longitude: 0.0001°
Localização GPS	3336	136	88	9	Altitude: 0.01 m

Fonte: (Adaptado de: My Devices, 2020).

O payload precisa ser enviado em formato hexadecimal, respeitando os *bytes* referentes ao canal, tipo de dado e conteúdo do dado. Conforme exemplificado na tabela 3.

Tabela 3 – Exemplo de dado enviado usando o protocolo Cayenne LPP

Payload (Hex)	03 67 01 10 05 67 00 FF	
Canal do dado	Tipo	Valor
03 => 3	67 => Temperatura	0110 = 272 => 27.2℃
05 => 5	67 => Temperatura	00FF = 255 => 25.5℃

Fonte: (Adaptado de: My Devices, 2020).

Para o desenvolvimento da aplicação a que se destina o presente trabalho, optou-se por utilizar este protocolo, uma vez que facilita a visualização dos dados e realiza as devidas conversões de valores, uma vez que o dispositivo gera dado do tipo genérico para distância medida do nível de água no reservatório e também dado do tipo tensão, para medida de nível de bateria.

2.1.6 *The Things Network*

A rede *The Things Network*, (TTN) ou a Rede das Coisas, é um dos provedores de serviços de registro de dispositivo LoRaWAN existentes no mercado. É baseada em servidores em nuvem e é responsável pela conexão entre os *gateways* da rede LoRaWAN ao redor do planeta.

Algumas cidades no mundo já possuem cobertura LoRAWAN completa, devido à quantidade de *gateways* instalados, como Amsterdan e algumas cidades da Austrália (NETO, 2017).

O diferencial desta rede se dá por se tratar de uma rede aberta e colaborativa, em que qualquer indivíduo pode construir seu próprio *gateway* e aumentar a cobertura da rede de forma a permitir que mais *end devices* sejam conectados.

A rede possui alguns elementos, como os já mencionados *end devices*, ou *end points* chamados também de nós na TTN. Esses são os dispositivos que captam informações, por meio de sensores e que através de um módulo de rádio LoRa, enviam esses dados para os *gateways* que se encontram ao alcance dos mesmos. Estes por sua vez, captam, processam e concentram as informações recebidas pelos *end devices*. Esses *gateways* são responsáveis pela comunicação dos nós com a internet. São eles os responsáveis pela cobertura da rede, e precisam estar conectados à internet, já os nós não, esse é um dos fatores que proporciona uma grande facilidade para a criação de dispositivos IoT, mesmo em ambientes remotos.

2.1.7 *Gateway LoRaWAN*

Os *gateways* formam a ponte entre os dispositivos nós e a TTN. Todos os *gateways* ao alcance de um dispositivo, receberão seus dados e os mesmos serão encaminhados à TTN (TTN, 2021).

As mensagens que são recebidas pelos dispositivos *gateways* têm sua integridade verificada por meio da verificação cíclica de redundância CRC, o que impede que pacotes inválidos sejam recebidos.

Alguns dados são anexados às mensagens de *uplink*, como *time stamp* e *Received signal strength indication* (Received Signal Strength Indication (RSSI)), que são respectivamente, dado, horário do recebimento e potência de sinal visto pelo *end device*. Assim, os pacotes, podem ser interpretados pelo *gateway*. A escolha do *gateway* a realizar o processamento do pacote dependerá do RSSI, ou potência do sinal, assim como ocorre com outras tecnologias de comunicações sem fio, como por exemplo *Wideband Code Division Multiple Access* (Wide-Band Code-Division Multiple Access (WCDMA)), ou Acesso Múlltiplo por Divisão de Código de Banda Larga que trata-se da tecnologia 3G e *Long Term Evolution* (Long Term Evolution (LTE)), ou evolução em longo prazo, que é a tecnologia 4G, quando performam *inter* ou *intra-rat*, que significa que trocam de célula quando um dispositivo está em movimento e consequentemente o dispositivo escolhe a célula que possui maior potência de sinal, ou seja, maior RSSI. Dessa forma, o *gateway* com melhor alcance em relação ao *end device* realiza a transmissão do pacote (TTN, 2021).

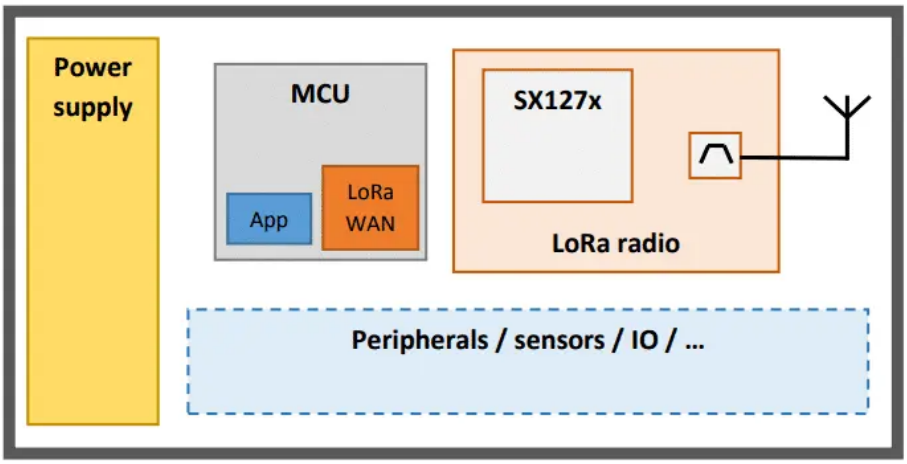
Há no mercado *gateways* disponíveis. As diferenças entre os modelos comercializados, se devem ao número de canais que possuem paralelamente. A configuração mínima é de oito canais, chegando até sessenta e quatro canais. Um *gateway* pode ser obtido através da junção de um Raspberry Pi com um módulo LoRa, por exemplo. O *software* para configuração do *ga-*

teway pode ser obtido de maneira gratuita pela plataforma da Semtech, também há no mercado *kits* para desenvolvimento de *gateways*.

2.1.8 *End devices*

A arquitetura de um *end device* da rede *LoRaWAN* é apresentada na Figura 6:

Figura 6 – Arquitetura geral de dispositivo LoRA baseado em chipsets SX127x



Fonte: (NETO, 2017).

Um *end device* é todo e qualquer dispositivo que utiliza a tecnologia LoRa e se comunica com a rede LoRAWAN. Em geral são construídos com o propósito de coletar dados pertinentes à sua aplicação, para que o usuário possa obter informações relacionadas a determinado ambiente e possa tomar decisões com base nesses dados. Em geral possuem um microcontrolador, um módulo de rádio LoRa e periféricos como sensores. Para um bom funcionamento esperado de um *end device* a Semtech definiu algumas configurações mínimas esperadas para um Microcontroller Unit (MCU), conforme mostradas na Tabela 4.

Tabela 4 – Configuração mínima para *end devices* Semtech

Parâmetro	Configuração mínima	Recomendado
RAM MCU	8 KB	16 KB
Flash MCU	128 KB	256 KB
AES 128 bits	Descrição AES no software	Elemento de segurança
Rádio DIOs conectados	DIO0, DIO1, DIO2	DIO0, DIO1, DIO2, DIO3
SPI (4 conexões:SCK,MOSI,MISO,NSS)	Mandatório	Mandatório
RTC (32.768 kHz XTAL)	Recomendado	Mandatório para devices classe B
IEEE 64-bit Identificador único	Mandatório	Mandatório

Fonte: (SEMTECH, 2017).

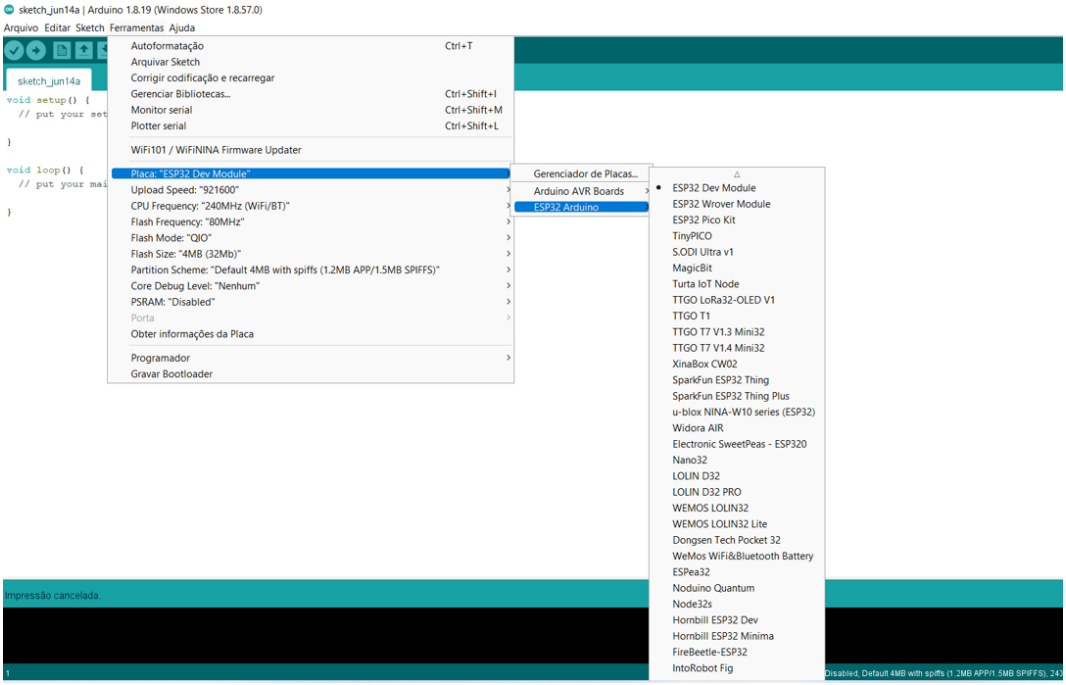
2.1.9 Microcontrolador ESP32

O mercado da Internet das Coisas ou *IoT (Internet of Things)* está em alta demanda e expansão (RAI; REHMAN, 2019). Por isso há cada vez mais opções de microcontroladores e *kits* de desenvolvimento disponíveis, tanto para uso acadêmico, como para aplicações em geral, como automações residenciais por exemplo.

Os microcontroladores podem ser usados em diversas aplicações, sendo a família dos ESP32 muito utilizada para desenvolvimento de dispositivos de *IoT*. Os microcontroladores ESP32 são os sucessores do ESP8266 (BABIUCH; FOLTÝNEK; SMUTNÝ, 2019). Há diversos *kits* de desenvolvimento e placas de circuito impresso, contendo diversos módulos integrados ao microcontrolador ESP32. Para desenvolver projetos basta optar pelo *kit* ou placa que melhor se adequa ao projeto. Alguns já incluem o rádio LoRa por exemplo e o próprio ESP32 possui conectividade Wi-Fi e Bluetooth. Além de ser facilmente encontrado, o microcontrolador ESP32 tem adaptabilidade com alguns outros sistemas como a IDE do Arduino, que é baseada na Linguagem C e torna seu uso simplificado. A utilização da IDE Arduino é adequada para uso em aplicações com o ESP32 pelo fato de possuir bibliotecas que se adequam perfeitamente às funções do ESP32.

Existem diversos tipos de microcontroladores da mesma família, pois vários fabricantes implementam o módulo do ESP32 a placas mais complexas que incluem outros módulos para uso geral. Porém, vale lembrar que quanto mais puro for o módulo do ESP32, menor será o consumo de energia do dispositivo desenvolvido, pois um simples módulo para conectividade Universal Serial Bus (USB) conectado ao ESP32 será responsável por aumentar o consumo em 5 mAh. Por esse motivo, vale analisar todas as características, principalmente as que podem influenciar o consumo do projeto, antes de se optar por um *kit* de desenvolvimento da família ESP32 caso um dos requisitos for o baixo consumo de energia. A Figura 7, extraída da IDE Arduino, mostra uma parte, da lista de placas ESP32 possíveis de serem utilizadas com a IDE uma vez incluída a biblioteca ESP32.

Figura 7 – IDE Arduino - Lista de placas ESP32



Fonte: Autoria própria, 2022.

2.1.10 *Deep sleep*

Como há diversas opções de placas no mercado, é necessário avaliar características importantes, em especial o consumo. Os microcontroladores da família ESP32 possuem uma função chamada *deep sleep*, que é responsável por desligar vários componentes do circuito do microcontrolador, quando ativada esta função via código, ou via interrupção, como o acionamento por botão, por exemplo.

O microcontrolador ESP32 possui alguns modos de operação diferentes, são eles: *Active Mode, Modem Sleep, Light Sleep, Deep Sleep e Hibernation Mode*, que são respectivamente os modos: ativo, modem dormindo, sono leve, sono profundo e hibernação. (DO BIT AO BYTE, 2021). Quando o ESP32 trabalha em modo ativo ou *active mode*, a energia flui livremente e o consumo será o máximo definido pelo fabricante da placa utilizada. Já no modo de modem dormindo, ou *modem sleep* o componente de rádio frequência é desligado, o que auxilia na economia da bateria. No modo de sono leve ou *light sleep*, algumas funções são modificadas: partes do circuito têm clock reduzido e a Central Processing Unit (CPU) é pausada. Por outro lado, o modo sono profundo ou *deep sleep* desligará periféricos, rádio, núcleos e grande parte da RAM. (DO BIT AO BYTE, 2021). Quando em modo *deep sleep* há considerável redução de consumo, sendo esse o modo mais indicado para utilizar em dispositivos IoT de baixo consumo.

Como há grande variedade de placas contendo o microcontrolador ESP32, também há grande variação de consumo dessas placas, mesmo para modo *deep sleep*, pois essa configuração vale para o ESP32. Em alguns casos, outros componentes da placa não são desligados, o que pode interferir no consumo total do dispositivo que se deseja construir. A tabela 5, extraída de (DIY I0T, 2020) mostra algumas opções de placas e uma comparação entre os consumos das mesmas.

Tabela 5 – Comparação de consumo entre placas ESP32

Placa	**Referência [mA]**	**Light Sleep [mA]**	**Deep Sleep [mA]**	**Hibernation [mA]**
ESP32 DevKitC	51	10	9	9
Ai-Thinker Node MCU-32S	55	15.05	6.18	6.18
Adafruit HUZZAH32	47	8.43	6.81	6.80
Sparkfun ESP32 Thing	41	5.67	4.43	4.43
FireBeetle ESP32	39	1.94	0.011	0.008
WiPy 3.0	192	-	0.015	-

Fonte: (Adaptado de: DiY i0T, 2020).

2.1.11 Sensor ultrassônico

Para obtenção das medidas de distância de um dispositivo em relação à distância de um objeto ou conteúdo líquido, uma das melhores opções é o sensor ultrassônico à prova d'água JSN-SR04T, por ter acabamento impermeável e módulo separado do sensor, além de possuir

um cabo longo, o que permite que a placa fique fora do alcance da água e que somente o sensor fique próximo a superfície da mesma. A Figura 8 ilustra o modelo do sensor utilizado.

Figura 8 – Sensor ultrassônico a prova d'água

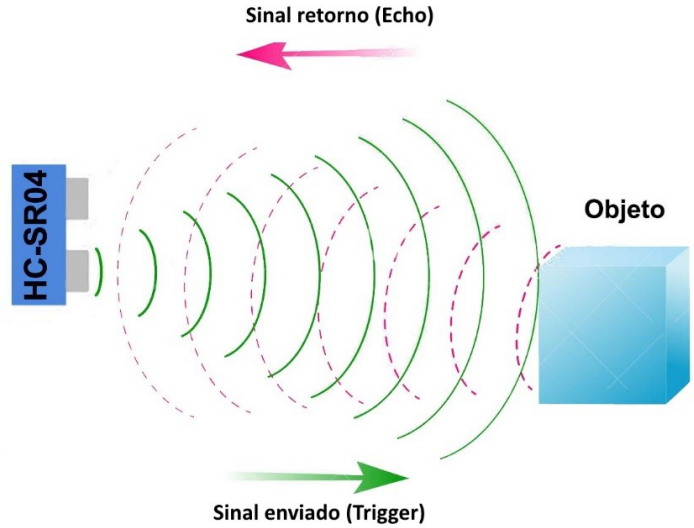
Fonte: (USINA INFO, 2022).

O JSN-SR04T pode ser alimentado por tensão de 3V a 5V, o que auxilia também no dimensionamento da bateria, para projetos alimentados por pilhas ou baterias de lítio recarregáveis, e voltados ao baixo consumo de energia. Esse sensor pode gerar medidas que vão de 20 cm a 600 cm, sendo que sua precisão é de +1/-1 cm. Este sensor, se comporta de maneira eficiente em dispositivos em que se deseja adquirir distâncias em líquidos incolores, como é o caso da água. Por utilizar o som, a onda vai até a superfície da água e é refletida novamente até o sensor, em vez de penetrar na água, ele é capaz de detectar a barreira física que é a própria superfície da água.

A Figura 9 demonstra o funcionamento do sensor HC-SR04 que possui as mesmas características do JSN-SR04T, porém não se indica para uso envolvendo água, por não possuir a proteção à prova d'água. No entanto, a demonstração do funcionamento se adequa, pois ambos possuem o mesmo princípio de funcionamento. O funcionamento de um sensor ultrassônico é baseado na emissão de onda sonora de alta frequência, e na medida do tempo que leva para a recepção do eco, produzido por esta onda quando se encontra com uma barreira capaz de refletir o som. Os sensores ultrassônicos funcionam medindo o tempo de propagação do eco, ou seja, o intervalo de tempo entre a onda sonora emitida e o eco recebido de volta. Os sensores emitem os pulsos ciclicamente. Quando um objeto reflete esses pulsos, o eco resultante é recebido e convertido em sinal elétrico. A detecção do eco incidente, depende da intensidade

e da distância entre o objeto e o sensor. A construção do sensor faz com que o feixe ultrassônico seja emitido em formato de cone e somente objetos dentro do raio do cone possam ser detectados. Os objetos a serem detectados podem ser sólidos, líquidos, granulares ou até pó. O material pode ser transparente ou colorido, de qualquer formato, e com superfície polida ou fosca. (SENSE SENSORS & INSTRUMENTS, 2014).

Figura 9 – Funcionamento do sensor ultrassônico

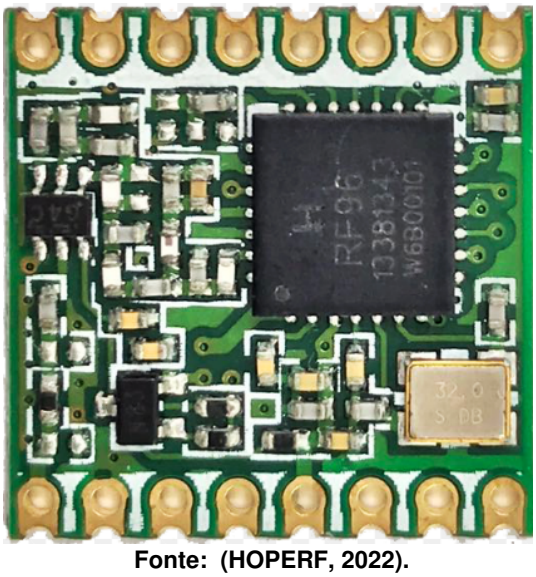


Fonte: (FLIP FLOP, 2013).

2.1.12 Rádio LoRa

O rádio LoRa ou transceptor RFM95W é um módulo que conecta uma antena a um microcontrolador e é responsável pela comunicação LoRaWAN.

Figura 10 – Transceptor RFM95W



Fonte: (HOPERF, 2022).

Segundo (CUNHA; VALIM, 2020), este módulo transceptor LoRa é mais adequado para aplicações de longa distância do que a utilização de um *kit* de desenvolvimento SX1276 que contém o módulo RFM95W integrado ao microcontrolador ESP32. A utilização de um SX1276 se torna mais fácil, pois o projeto com essa placa elimina alguns fios, ou trilhas, já que o módulo faz parte da placa. Entretanto, em termos de efetividade para longas distâncias, a utilização do módulo transceptor separado do ESP32 é mais indicada.

2.1.13 .NET C Sharp

O *backend* da aplicação deste trabalho foi desenvolvido na framework open source para criação e execução de aplicativos da Microsoft chamada .NET framework, que dá suporte ao desenvolvimento de diferentes linguagems de progração como *C#*, *F#*, ou *Visual Basic*. Optou-se por utilizar a linguagem *C#* pelo escopo do projeto, e familiaridade dos integrantes do trabalho com a linguagem. (MICROSOFT, 2022). *C#* é uma linguagem de programação, orientada a objeto. Os aplicativos desenvolvidos nessa linguagem são executados no .NET. Essa linguagem tem como base a linguagem *C*. (MICROSOFT, 2022).

2.1.14 Banco de dados

Banco de dados é qualquer coleção de informações inter-relacionadas segundo a Microsoft (2022). Uma lista telefônica salva em um caderno de papel pode ser considerada um banco de dados analógico, ou uma lista de compras, uma lista de tarefas, mas para a ciência da computação o banco de dados se define como uma coleção de informações salvas em um sistema de computador, ou seja, as mesmas listas de compras, telefônica ou tarefas salvas em um computador podem ser considerados banco de dados.

2.1.15 Webhook

Webhook é um recurso que permite que uma ferramenta ou aplicação se comunique e troque dados com outra, fornecendo dados em tempo real quando ocorrer determinado evento (PLUGA, 2018).

Webhooks servem para estabelecer comunicação entre dois *softwares* que naturalmente não se comunicariam. Para dispositivos *IoT* que utilizam a rede TTN (*The Things Network*), por exemplo, faz-se necessário implementar um ou mais *webhooks* que a própria rede TTN oferece, para transferir dados para um banco de dados, ou aplicação a cada momento que um dado é registrado por um *gateway* conectado à rede.

3 DESENVOLVIMENTO

Neste capíulo está descrito detalhadamente o desenvolvimento do protótipo físico, o dispositivo *ANADevice*, desde a sua fabrição até os testes de integração, bem como o desenvolvimento do *firmware* instalado no dispositivo, desenvolvimento do aplicativo *ANAApp*, para *smartphone*, também estão descritos os detalhes da construção do *backend* responsável pelo processamento das requisições do usuário e processamento dos dados recebidos dos dispositivos ANADevices, instalados nos reservatórios, bem como o banco de dados que fará o armazenamento de todos os dados da aplicação e os servidores usados para o hospedagem da aplicação. O escopo deste trabalho não contempla o desenvolvimento do *gateway* que faz parte da camada física de comunição Long Range (LoRa), para o presente projeto, foi utilizado *gateway* previamente instalado.

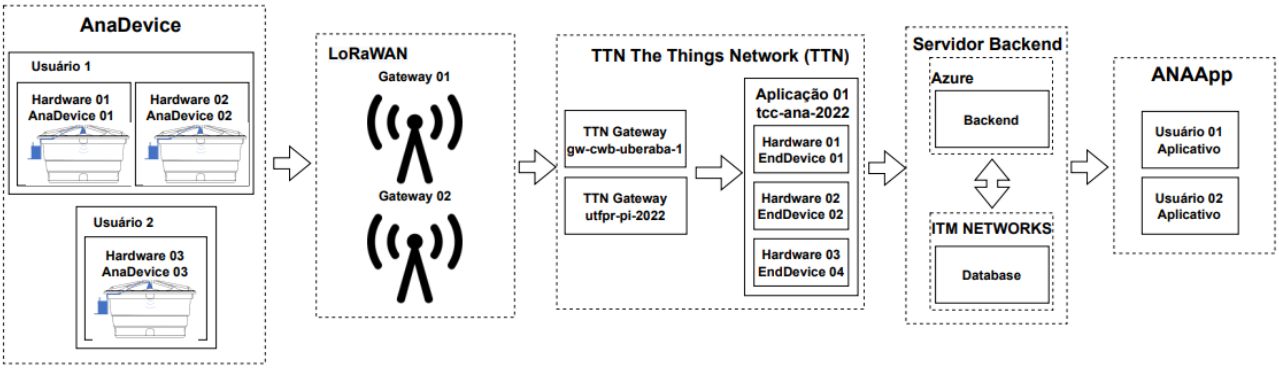
3.1 Arquitetura do sistema

A arquitetura do sistema é composta por cinco elementos principais:

- ***ANADevice***: é o *hardware* instalado na caixa d'água do usuário para fazer a leitura e envio das informações de nível de bateria para a plataforma TTN.
- ***LoRaWAN gateway***: o dispositivo físico instalado em ponto estratégico para cobrir a maior área. Gateways são os responsáveis por receber os dados dos *ANADevice* e enviá-los via LoRaWAN. Para este projeto, os *gateways* foram integrados à plataforma TTN.
- ***TTN***: possui os *gateways* e aplicações cadastradas para receber os dados e enviá-los para o servidor do *backend*.
- ***Servidor backend***: responsável pelo processamento dos dados recebidos da *TTN*, armazenar em banco de dados, e enviar para o *ANAApp* do usuário.
- ***ANAApp***: responsável por exibir os dados lidos do *ANADevice* para o usuário.

A Figura 11 ilustra a arquitetura do sistema.

Figura 11 – Arquitetura do sistema



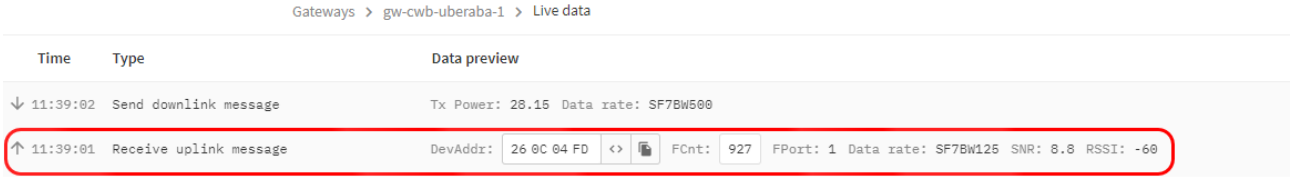
Fonte: (Autoria própria, 2022).

O usuário pode possuir mais de um *hardware*, chamado de *ANADevice* instalado em diferentes caixas d'água, cada *ANADevice* possui seu *end device* correspondente cadastrado na aplicação dentro da plataforma *TTN*. O *ANADevice* instalado, realizará as medições e as enviará os dados através da tecnologia LoRa: modulação de espectro, baixas frequências, longo alcance e transmissão de pequenos pacotes de dados. Após a transmissão, faz-se necessário que um equipamento receba esses dados, os *gateways* representados na Figura 11 como o bloco "*TTN Gateway*".

A comunidade *TTN* já possui alguns *gateways* instalados pelos usuários, aberto para uso geral. Caso não exista um próximo a sua localidade então é necessário desenvolver o *hardware* e instalar um novo na rede. Os gateways recebem os dados em uma frequência específica de acordo com as normas de cada país e região. No caso do Brasil a frequência utilizada é na 903.9MHz, portanto o *ANADevice* deve transmistir nessa frequência para realizar a efetiva comunicação. Para os testes deste projeto utilizou-se o *gateway gw-cwb-uberaba-1* localizado nas proximidades de Curitiba, Paraná, mas o *ANADevice* poderá se comunicar com qualquer *gateway*, desde que sejam feitas as configurações necessárias.

A Figura 12 ilustra uma mensagem de um dispositivo que chegou ao gateway.

Figura 12 – Mensagem gateway



Fonte: (Autoria própria, 2022).

Na Figura 12, verificam-se algumas características da mensagem como o horário do recebimento da mensagem, qual o endereço do dispositivo que mandou essa mensagem, e informações sobre características de comunicações, como o *timestamp*, RSSI (*Received signal strength indication*, do inglês indicador de força do sinal recebido) e Signal to Noise Ratio (SNR)

(*Signal-to-noise ratio*, do inglês Relação sinal-ruído). Ao verificar os detalhes da mensagem é possível ver qual o dado que foi recebido, porém como o *gateway* é apenas o receptor de todas as mensagens enviadas por *end devices*, não possuindo as ferramentas necessárias para descriptografá-las, preservando assim a confidencialidade das informações transmitidas, assim é possível ver apenas um conjunto de caracteres codificados "fFkUWixx9RA", ilustrado na Figura 13.

Figura 13 – Mensagem gateway detalhes - Parte 1

Event details


```
{
  "name": "gs.up.receive",
  "time": "2022-06-17T14:44:41.371151747Z",
  "identifiers": [
    {
      "gateway_ids": {
        "gateway_id": "gw-cwb-uberaba-1",
        "eui": "B827EBFFFF5C1DBF"
      }
    }
  ],
  "data": {
    "@type": "type.googleapis.com/ttn.lorawan.v3.GatewayUplinkM
    "message": {
      "raw_payload": "QP0EDCaApAMBfFkUWixx9RABZ08T",
      "payload": {
        "m_hdr": {
          "m_type": "UNCONFIRMED_UP"
        },
        "mic": "AWdPEw==",
        "mac_payload": {
          "f_hdr": {
            "dev_addr": "260C04FD",
            "f_ctrl": {
              "adr": true
            },
            "f_cnt": 932
          },
          "f_port": 1,
          "frm_payload": "fFkUWixx9RA="
        }
      },
      "settings": {
        "data_rate": {
          "lora": {
            "bandwidth": 125000,
            "spreading_factor": 7
          }
```


Fonte: (Autoria própria, 2022).

Para descriptografar as mensagens é necessário um segundo elemento chave na comunicação: a aplicação TTN representada pelo bloco "Aplicação 01" na Figura 11.

A aplicação dentro da plataforma da *TTN* é responsável por fazer o gerenciamento dos dispositivos, descriptografia dos dados e integração com o *backend* desenvolvido. Para que o dispositivo possa se comunicacar, o protocolo LoRaWAN exige que os dispositivos realizem autenticações de segurança, com uso de credenciais disponibilizadas pelos *end devices* da aplicação. Cada *hardware* em campo deve possuir seu correspondente *end device* na aplicação.

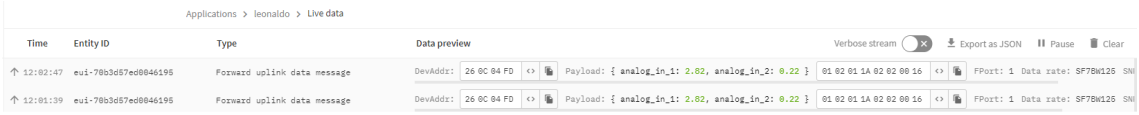
O *end device* está representado na Figura 11 como "Hardware 0X EndDevice 0x". Ele é o elemento que a rede LoRaWAN irá utilizar para fazer a autenticação, ou seja, receber as

mensagens apenas de dispositivos registrados na rede, todo *end device* possui as seguintes credenciais:

- ***Device address*** : um endereço de 32 bits para identificar o *end device*.
- ***Application Session Key***: utilizada pelo hardware e pela TTN para criptografar e decriptografar os dados transmitidos para garantir a confidencialidade das mensagens.
- ***Network Session Key*** : também utilizada pelo hardware e pela TTN para criptografar e decriptografar os dados transmitidos.

Quando um novo *ANADevice* é instalado cria-se um novo *end device* na aplicação da *TTN* e suas credenciais são armazenadas no *firmware* do *ANADevice*, assim os dados serão transmitidos para o *gateway* e então enviados para a aplicação da *TTN* que possui as chaves para descriptografá-los e enviá-los para o servidor de *backend* representado na Figura 11 como *Servidor Backend*. A Figura 14 mostra o dado transmitido e descriptografado *"analog_in_1: 2.82 e analog_in_2: 0.22"*.

Figura 14 – Mensagem gateway detalhes - Parte 2

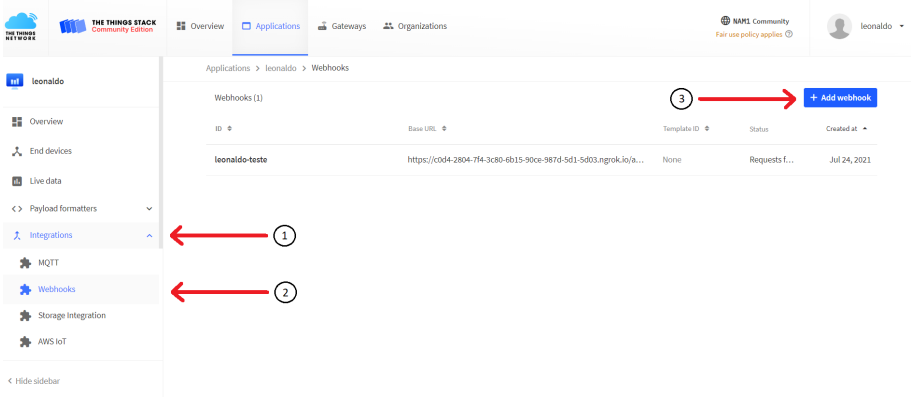


Fonte: (Autoria própria, 2022).

Para que seja possível realizar a integração da rede *TTN* como uma aplicação externa, utilizou-se de uma técnica conhecida como *"webhook"*, abordado no item 2.1.15, que é suportado pela plataforma. O *webhook* resumidamente é um dispositivo de notificação. Para toda nova mensagem que chega à aplicação, será disparado um evento, ou seja, será notificada uma aplicação externa. Para receber essas notificações, foi desenvolvido um servidor *backend*.

Na plataforma *TTN* é possível criar *webhooks* dentro da aplicação, basta clicar sobre *"Integrations"* ao lado esquerdo, em seguida *"Webhooks"* e por fim pressionar o botão *"+ Add Webhook"*. As setas em vermelho e os números indicam a sequências dos passos a serem seguidos na criação do webhook na Figura 15.

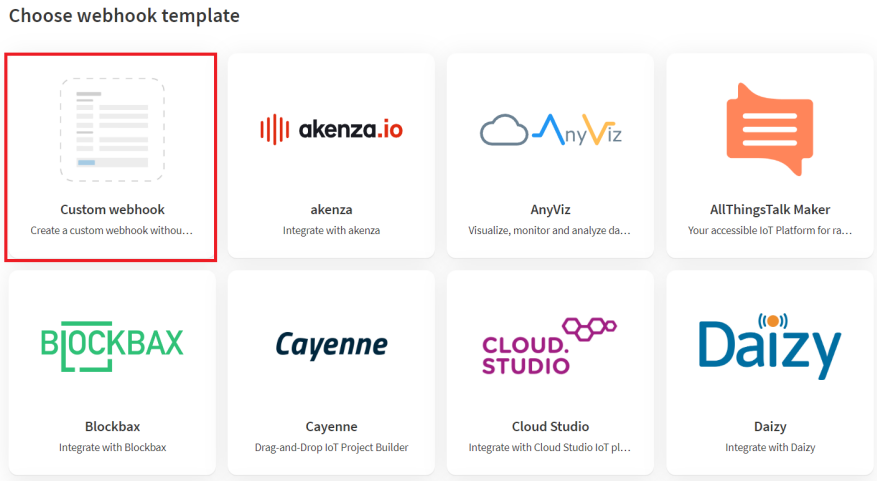
Figura 15 – Criando um webhook



Fonte: (Autoria própria, 2022).

A plataforma apresenta algumas opções de *weebhook* pré-configurados para fazerem integrações com outros sistemas. Contudo, neste projeto, criou-se um *webhook* do tipo customizado, clicando na opção *"Custom webhook"* mostrado na Figura 16

Figura 16 – Weebhook customizado



Fonte: (Autoria própria, 2022).

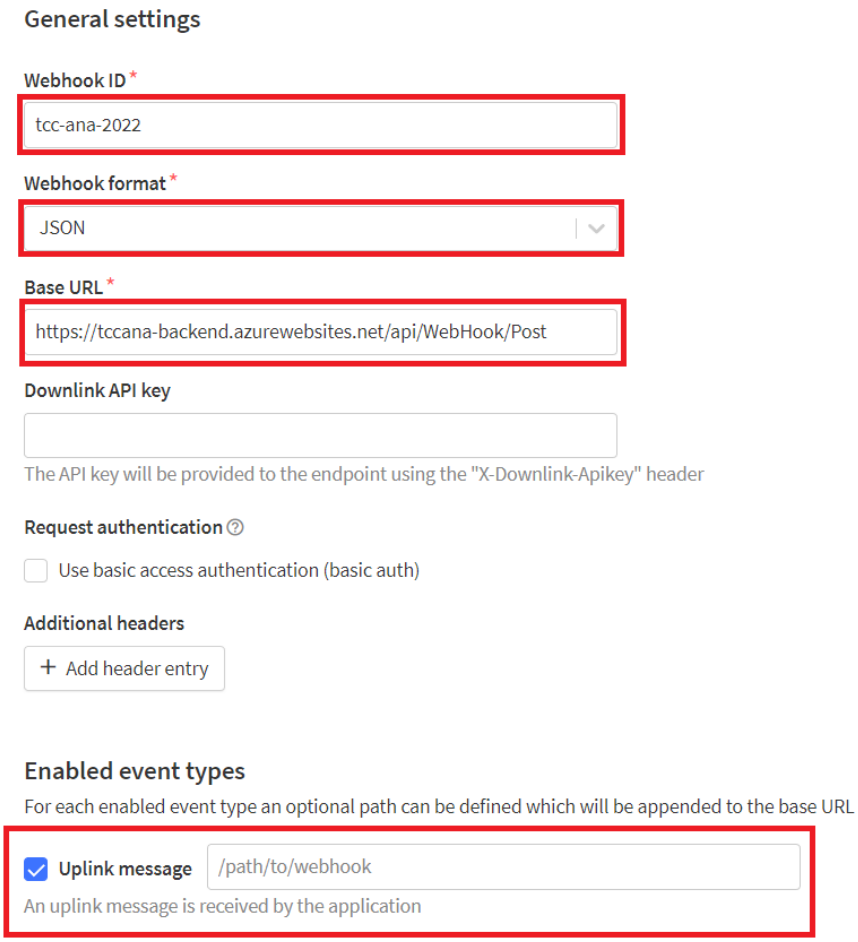
Após clicar na opção *"Custom webhook"* as seguintes configurações gerais são apresentadas:

- ***Webhook ID***: o nome escolhido pelo usuário para o *webhook*. Definiu-se "tcc-ana-2022".
- ***Webhook format***: qual o formato do dado que o *weebhook* enviará. Há duas opções: *"Json"* e *"Protocol buffers"*. Optou-se pelo formato *"Json"*.
- ***Base URL***: para qual endereço serão enviado os dados. Colocou-se o endereço *web* onde está hospedado o servidor *backend*: "https://tccana-backend.azurewebsites.net/api/WebHook/Post".

- ***Enable event types***: qual o tipo de evento será enviado. O projeto apenas envia dados do *ANADevice* para a *TTN* essa comunicação é chamada de *"Uplink"*. Selecinou-se apenas essa opção.

A Figura 17 mostra como ficaram definidas todas essas configurações para o *webhook* e em vermelho quais configurações foram alteradas:

Figura 17 – Configurações *webhook*



Fonte: (Autoria própria, 2022).

O servidor *backend* é responsável por receber os dados da rede *TTN* através de *webhook* processá-los e armazená-los no banco de dados. O servidor também deve ser responsável por processar os pedidos dos usuários via *ANAApp*, por exemplo: qual o nível atual para determinado *hardware* e usuário. O *backend* foi hospedado na plataforma *cloud Azure*, assim ele fica disponível em um endereço *url*. O servidor é apenas uma unidade de processamento de pedidos, ele não possui nenhuma interface com o usuário.

O banco de dados é representado pelo bloco *Database*, que é reponsável por armazenar os dados recebidos pelos dispositivos assim como os dados dos usuários. O servidor do banco de dados foi criado na plataforma da empresa *ITM NETWORKS* que é um provedor privado de banco de dados.

Finalmente, como último bloco, temos o aplicativo chamado de *ANAApp*: uma interface interativa para o usuário. No *ANAApp* o usuário consegue selecionar sobre qual dispositivo ele gostaria de ver os dados da última leitura, assim como histórico ou mesmo plotar um gráfico selecionando o período da coleta dos dados.

3.2 Hardware

Para o desenvolvimento do circuito foram avaliadas algumas possibilidades de microcontroladores. A família de microcontroladores ESP32 foi considerada a mais apropriada por terem inovações recentes e também por terem sido desenvolvidos módulos para o ESP32 que já contemplavam o módulo de rádio LoRa e também o circuito interno controlador de bateria com saídas específicas para ligação de bateria. Isso pouparia o excesso de ligações, bem como o espaço físico utilizado para o circuito. Um dos principais objetivos era desenvolver um conjunto pequeno, de fácil utilização para o usuário e que pudesse ser colocado num pequeno invólucro possibilitando também, o baixo consumo de energia e baixo custo, pois o módulo ESP32 com rádio LoRa e circuito para bateria integrados tem custo menor do que os mesmos componentes adquiridos de forma independente. Os custos médios dos componentes estão indicados na tabela abaixo:

Tabela 6 – Custos componentes projeto *ANADevice*

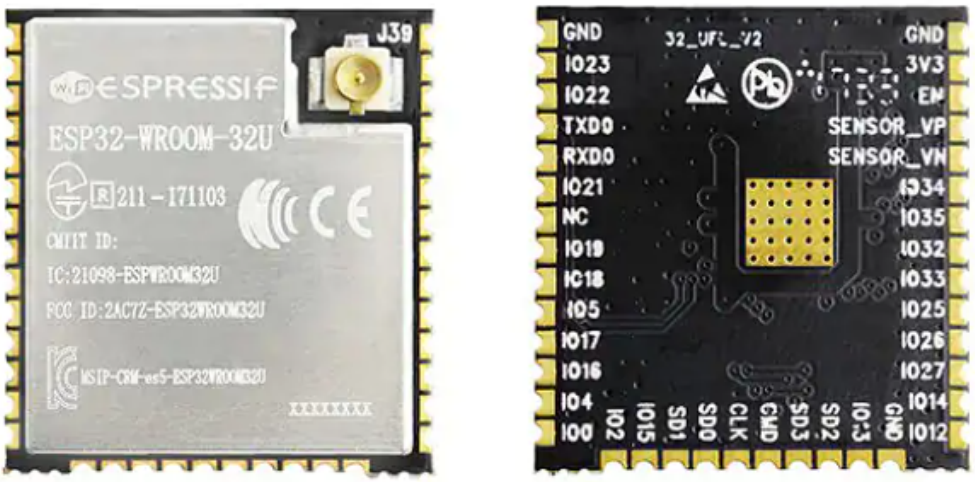
Componente	Custo
LILYGO ESP32 SX1276	R$ 68,50
LoRa RFM95W	R$ 60,80
ESP WROOM 32	R$ 40,76
Sensor JSN-SR04T	R$ 57,00

Fonte: (Autoria própria, 2022).

No entanto, devido a testes de consumo, observou-se que o microcontrolador escolhido ESP32 LILYGO TTGO sx1276, embora tenha menor custo do que a utilização de um módulo ESP WROOM 32 mais a junção do rádio LoRa RFM95W, por ter módulo USB integrado, rádio LoRa, módulo carregador de bateria, dentre outros, possui um consumo relativamente alto para o projeto, em torno de 20 mAh mesmo operando em modo *deep sleep*, portanto julgou-se inadequado para o projeto.

Buscou-se observar aspectos que melhorariam o desempenho da bateria e notou-se que quanto menos componentes o circuito da placa ESP32 possuir, de maneira integrada, melhor para o consumo. Com isso, escolheu-se o microcontrolador ESP32 WROOM *module*. A Figura 18, representa o ESP32 escolhido para o projeto: o ESP32 WROOM.

Figura 18 – Módulo ESP32 WROOM



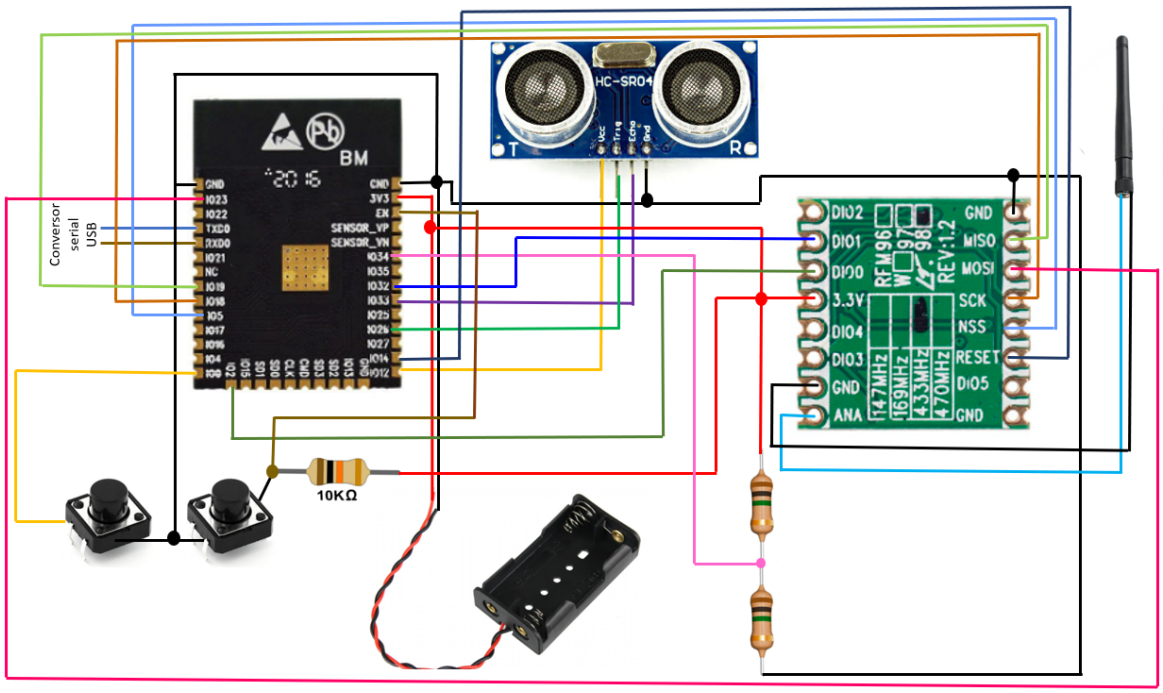
Fonte: (ESPRESSIF, 2022).

Como a placa escolhida não possui módulos integrados, o consumo é praticamente o teórico informado pelo fabricante que é de 10 a 150uA em modo *deep sleep* (SYSTEMS, 2022). Com a alteração do microcontrolador, o projeto ficou adequado, de modo a cumprir um dos requisitos fundamentais, que consistem na duração da bateria por pelo menos dois anos.

3.3 Construção de placa de circuito impresso

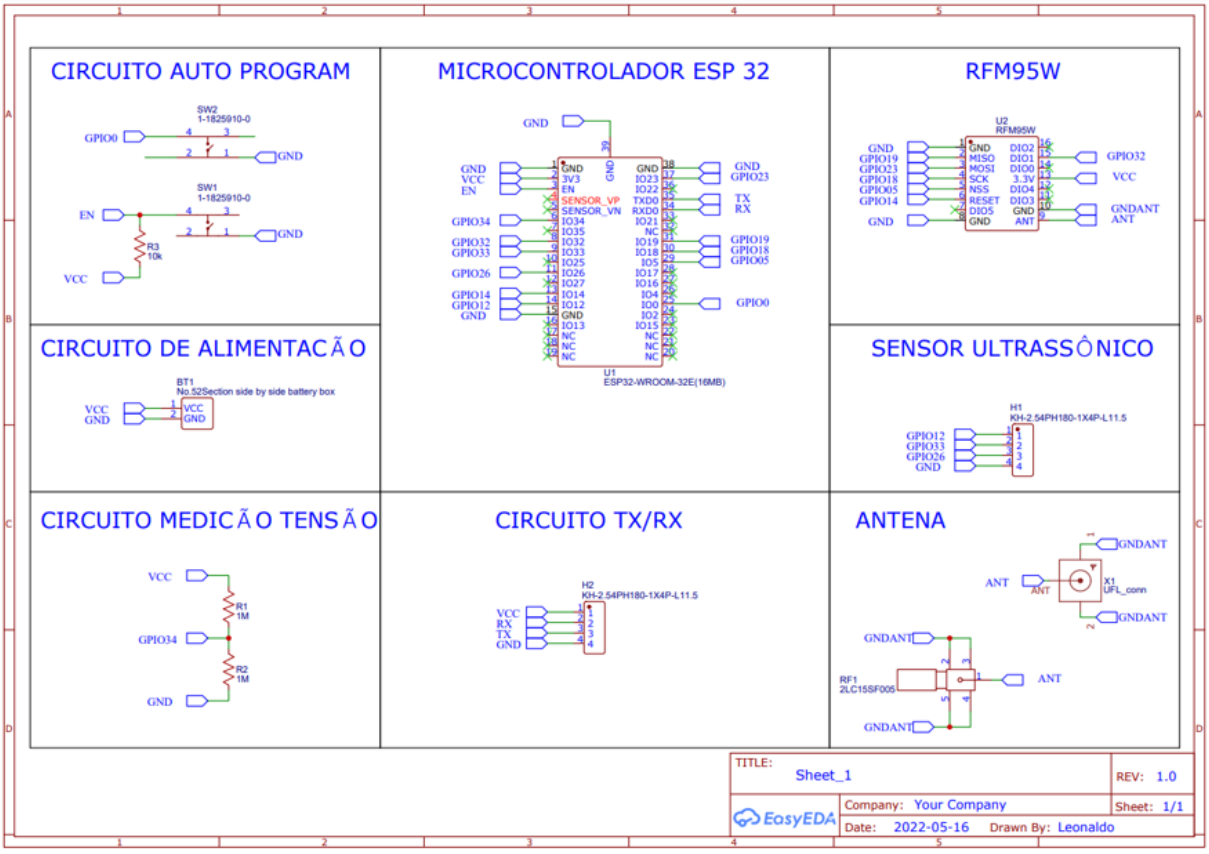
Para acomodar o circuito de maneira apropriada foi projetada e fabricada uma placa de circuito impresso, em laboratório nas dependências da Universidade Tecnológica Federal do Paraná (UTFPR). O projeto da placa de circuito impresso fabricada, seguiu o diagrama pictórico da Figura 19 e consequentemente, construiu-se o esquema elétrico conforme a Figura 20.

Figura 19 – Diagrama pictórico do circuito



Fonte: (Autoria própria, 2022).

Figura 20 – Esquema elétrico do circuito



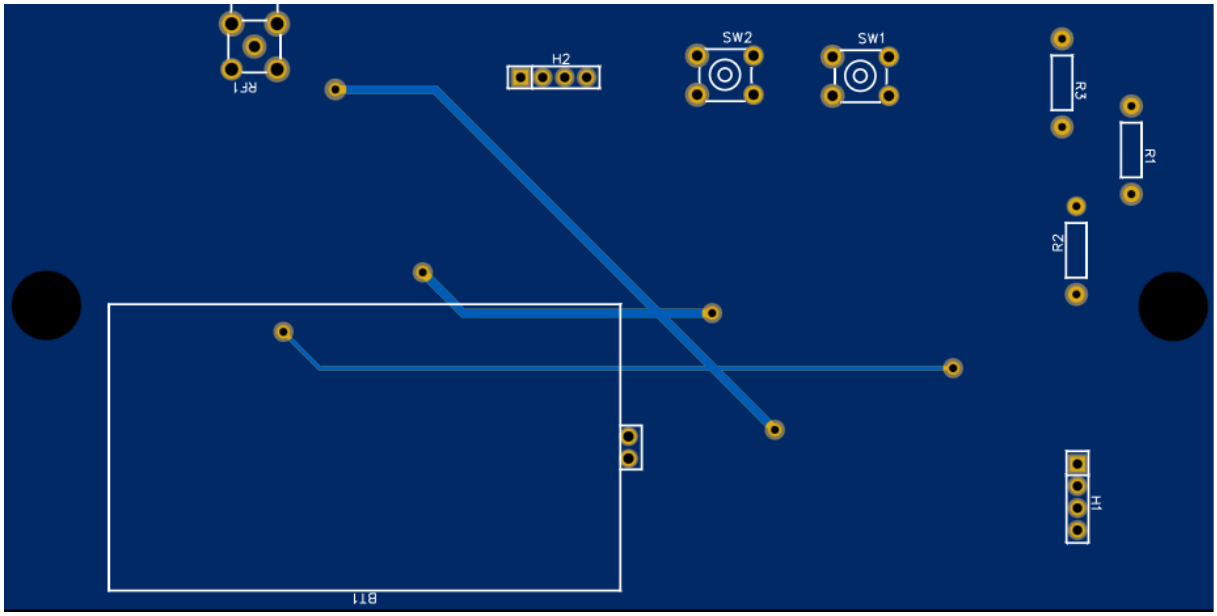
Fonte: (Autoria própria, 2022).

A placa de fenolite e cobre utilizada, passou por processos químicos sustentáveis sem agredir o meio ambiente, pois todo o resíduo gerado foi neutralizado durante processo. O material corrosivo mais comumumente utilizado no passado era o percloreto de ferro, porém os custos para regeneração são elevados e seu descarte contamina o meio ambiente. Por isso, surgiram outras formas de realizar o processo de corrosão de placas de circuito impresso (TEC-CI, 2022).

Os diagramas das placas de circuito impresso, estão demonstrados nas figuras 20 e 21.

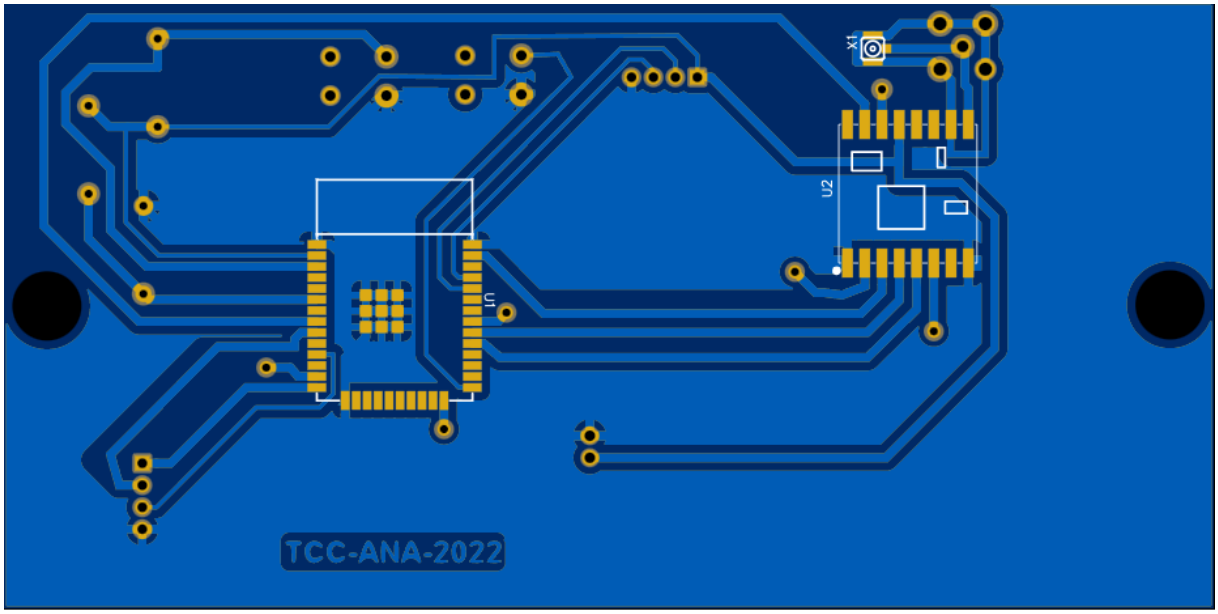
Primeiramente, fez-se a gravação do desenho das trilhas do circuito na superfície da placa. Depois a placa foi mergulhada em uma solução de NaOH (hidróxido de sódio) à temperatura de 60º C, posteriormente passou pela solução de Na2CO3 (carbonato de sódio). Em seguida foi aplicada tinta fotosenssível a fim de que se cobrisse toda a superfície da placa. Posteriormente, para retirada da tinta nas partes que interessavam, ou seja, as trilhas do circuito, foi utilizada novamente a solução de Na2CO3 (carbonato de sódio) e para corrosão do cobre em excesso foi utilizado HCl (ácido clorídrico) de baixo volume, que não oferece danos quando em contato com a pele. Por fim, lavou-se a placa em uma solução que continha ácido clorídrico e água a fim de neutralizá-la para posterior utilização. Essas etapas fizeram parte da preparação da placa de circuito impresso utilizando compostos químicos que foram neutralizados para que

Figura 21 – Projeto da placa parte superior



Fonte: (Autoria própria, 2022).

Figura 22 – Protótipo da placa parte inferior



Fonte: (Autoria própria, 2022).

não gerassem resíduos tóxicos quando descartados no meio ambiente. A Figura 23 demonstra parte do processo de fabricação da placa.

Figura 23 – Processo de fabricação da PCI

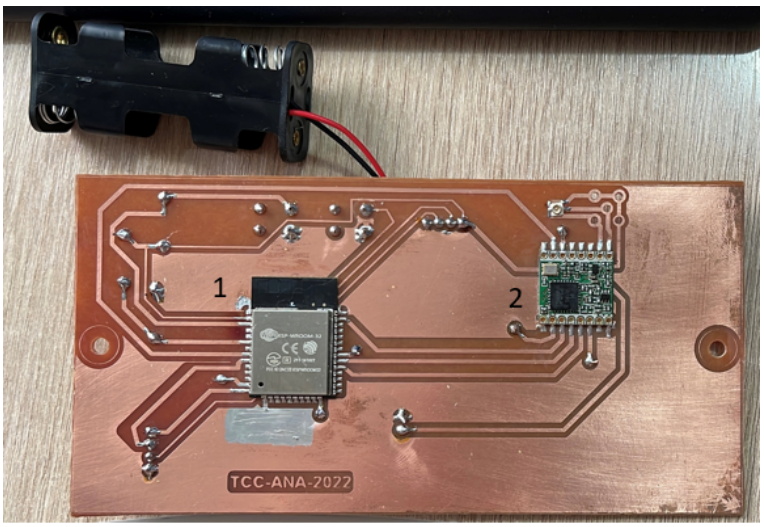
Fonte: Autoria própria, 2022.

Em seguida, uma camada de verniz foi aplicada, pois protege o circuito e derrete ao entrar em contato com o estanho líquido e quente, proporcionando uma boa aderência de solda. Após a secagem, a placa passou pelos processos de corte e perfuração com broca, utilizando uma furadeira de baixa potência.

Esse processo produziu duas placas de maneira sustentável, a fim de acomodar o circuito do protótipo de maneira adequada. O processo foi importante, pois levou em consideração os impactos que o resíduo de cobre e outras substâncias podem causar se depositadas no meio ambiente.

Finalmente o circuito foi montado na placa. Assim foi possível desenvolver os detalhes do invólucro para o conjunto, bem como realizar testes de integração entre hardware e firmware. Na Figura 24, que refere-se à vista frontal da placa montada para o dispositivo, o número (1) corresponde ao microcontrolador ESP32 WROOM e o número (2) representa o módulo do rádio LoRa. Também é possível observar o suporte para pilhas comuns, alcalinas do tipo AA, além das trilhas que compõe o circuito.

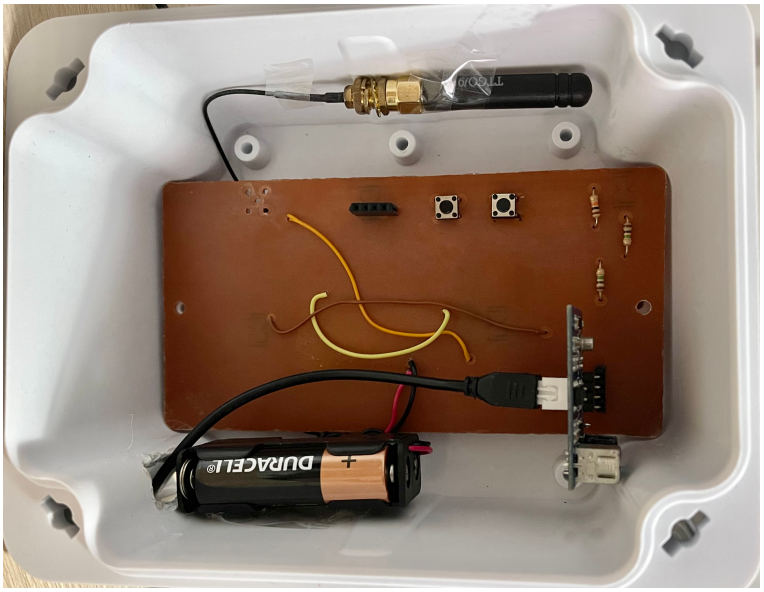
Figura 24 – Circuito final do dispositivo visão frontal



Fonte: (Autoria própria, 2022).

Como nem todas as trilhas necessárias, puderam ser adicionadas à face frontal da placa, alguns jumpers precisaram ser implementados na parte posterior do circuito, conforme demonstrado na Figura 25.

Figura 25 – Circuito final do dispositivo visão traseira



Fonte: (Autoria própria, 2022).

Na Figura 25, é possível notar os demais componentes presentes no circuito, como botões de *enable* e *reset*, uma vez que o microcontrolador ESP32 WROOM não possui módulo de carregamento por USB, visto que esse módulo é responsável por consumo de bateria em torno de 5 mAh, valor bem acima do requisito *RF6* que determina que o consumo total do sistema em *deepsleep* deve ser abaixo de 50 μAh.

Além disso, na Figura 25 estão presentes as pilhas AA que alimentam o circuito, bem como na parte superior pode-se notar a presença da antena do rádio LoRa e ainda o módulo do sensor ultrassônico que está localizado na parte inferior direita da imagem. O invólucro escolhido para armazenar o circuito também está presente na Figura 25.

3.4 Sensor ultrassônico à prova d'água JSN-SR04T

Para a obtenção das medidas de distância do dispositivo em relação à altura da água presente na caixa d'água, utilizou-se o sensor ultrassônico à prova d'água JSN-SR04T. Essa escolha possibilitou o desenvolvimento de um recipiente simples para a placa e demais componentes, de modo que apenas o sensor ficasse próximo à superfície da água, sendo que o invólucro contendo a placa com demais componentes ficasse posicionado no exterior do reservatório de água.

O JSN-SR04T pode ser alimentado por tensão de 3V a 5V, assim foi possível a utilização de duas pilhas alcalinas AA, totalizando 3 V de tensão.

3.5 Microcontrolador ESP32

Primeiramente foi definido o microcontrolador ESP32 LILYGO TTGO sx1276, para o desenvolvimento deste projeto. Esse microcontrolador foi escolhido, por possuir módulo LoRa integrado, e simplificada configuração, economizando algumas trilhas no projeto da placa, bem como espaço útil da mesma.

No entanto, após alguns testes com esse *kit* de desenvolvimento, observou-se alto consumo de energia, em torno de 40 mA em modo *deep sleep*, o que consumia inteiramente as pilhas em poucos dias. Esse fato comprometeu um dos requisitos primordiais do projeto, o que se refere ao baixo consumo, pois a ideia é de que o usuário não precise se preocupar em substituir as pilhas que alimentam o dispositivo em períodos menores do que 2 anos.

Portanto, após vastas pesquisas sobre os módulos ESP32 mais eficientes e com menor consumo, concluiu-se que, quanto mais puro for o módulo ESP32, ou seja quanto menos componentes forem agragados ao microcontrolador, mais econômico torna-se o circuito. Dessa maneira, decidiu-se implementar um módulo ESP32 WROM tendo o transceptor Lora que ser adicionado separadamente.

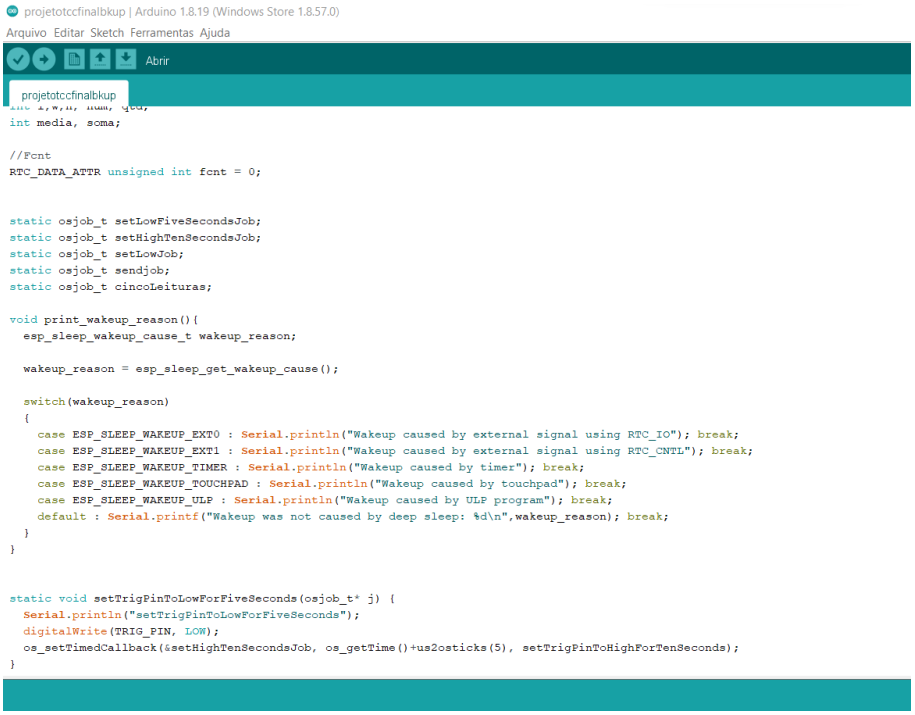
Os *kits* de desenvolvimento que possuem o microcontrolador ESP32 e dispõem de mais recursos, tendem a possuir maior consumo de energia.

3.6 Firmware

O firmware foi pensado de modo a se integrar com o hardware, ou seja, a escolha do microcontrolador influenciou de maneira significativa o desenvolvimento do firmware, pois dependendo de qual fosse a placa microcontroladora, a linguagem de programação poderia ser de baixo ou alto nível, bem como impactaria na definição de quais bibliotecas seriam usadas.

Por esse motivo optou-se pela linguagem C com Integrated Development Environment (IDE) Arduino por possuir bibliotecas específicas para leitura de sensores de maneira analógica e digital. O foco foi construir um firmware simples com um circuito com todos os módulos já integrados a fim de descomplicar a montagem e desenvolvimento do programa. A complexidade de programação ficou a cargo do *backend* e do desenvolvimento do *ANAApp* A Figura 26 representa parte do código extraído do *firmware* desenvolvido.

Figura 26 – Trecho de código firmware



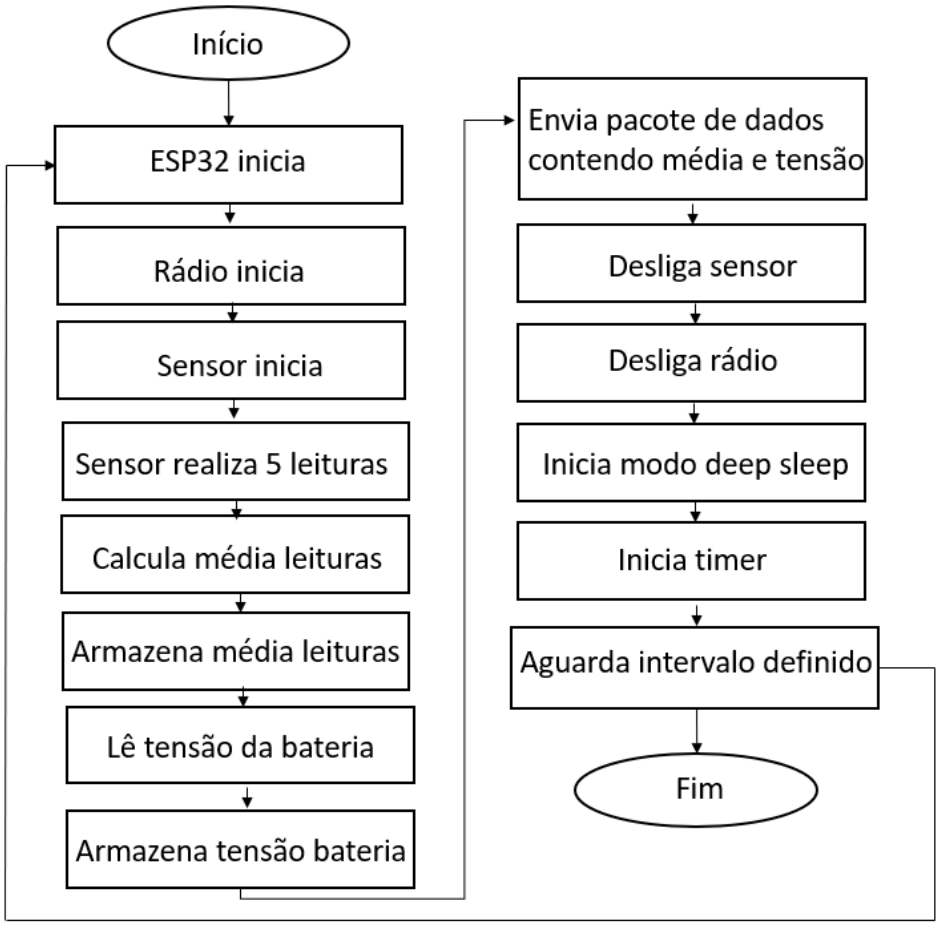
Fonte: (Autoria própria, 2022).

O código inicia o módulo de rádio LoRa que faz a comunicação com um *gateway* que está localizado nas proximidades do *ANADevice*, o *firmware* também inicia o sensor ultrassônico, que em seguida realiza cinco leituras da altura da coluna d'água consecutivas, em seguida os envia para a rede TTN e assim que o pacote é enviado, entra em modo *deep sleep*, desligando todos os componentes do circuito, para preservar a bateria, prolongando a vida útil da mesma. O processo completo do código leva dois segundos para ser finalizado e ocorre a cada

uma hora. O processo para cálculo do volume do reservatório é realizado no *backend*, sendo que no *firmware* são coletadas apenas medidas de distância entre o sensor e a superfície da água.

Um dos motivos para a escolha do módulo ESP32 deve-se ao fato de economizar bateria na função *deep sleep*, a qual é comum em microcontroladores da família ESP. Essa função é empregada em momentos que o microcontrolador está em espera, ou seja, sem transmitir ou receber dados. Por se tratar de um dispositivo que visa enviar dados de leitura com periodicidade de uma hora, uma maneira de atingir maior vida útil da bateria, era deixar o circuito em modo de baixo consumo ou *ultra low power* e através de funções da biblioteca do microcontrolador, fazê-lo despertar em momentos de transmissão operando entre intervalos pré-definidos. A Figura 27 ilustra o funcionamento do *firmware*.

Figura 27 – Diagrama firmware

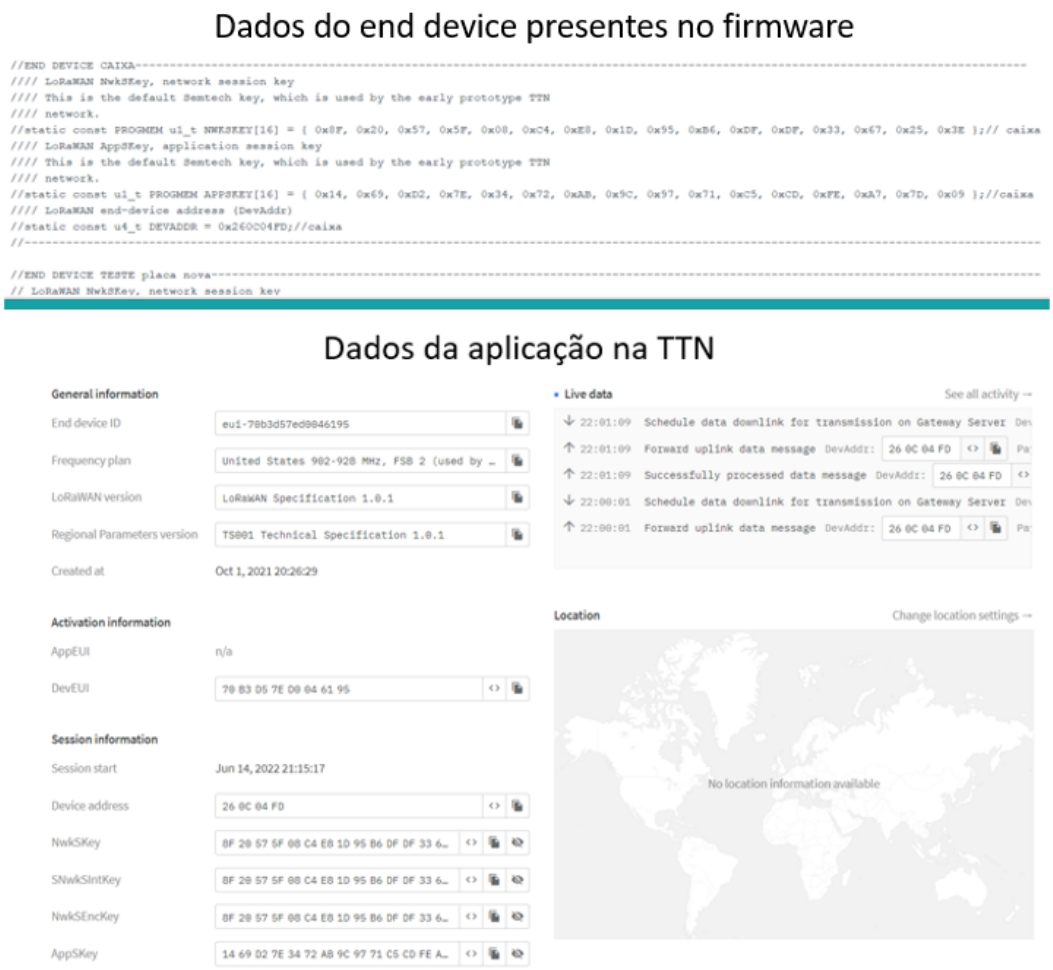


Fonte: (Autoria própria, 2022).

3.7 TTN The Things Network

Para este projeto, utilizou-se a rede TTN *The Things Network*, por ser aberta e gratuita ao uso para dispositivos que utilizam a comunicação pelo protocolo LoRaWAN. Para que os dados sejam visualizados na rede TTN, primeiramente é fundamental que haja um cadastro na mesma, além disso também é necessário que haja um *gateway* instalado e operante nas redondezas do *end device*. Também é necessário criar uma aplicação na TTN, sendo que os dados dentro do *firmware*, precisarão ser compatíveis com os dados presentes na aplicação. Com isso, tendo o *end device*, ou nó conectado e operando, os dados enviados pelo sensor são recebidos na aplicação da TTN. A Figura 28 demonstra a aplicação criada na TTN e retrata os dados sendo recebidos por essa aplicação.

Figura 28 – Dados *ANADevice firmware* e TTN

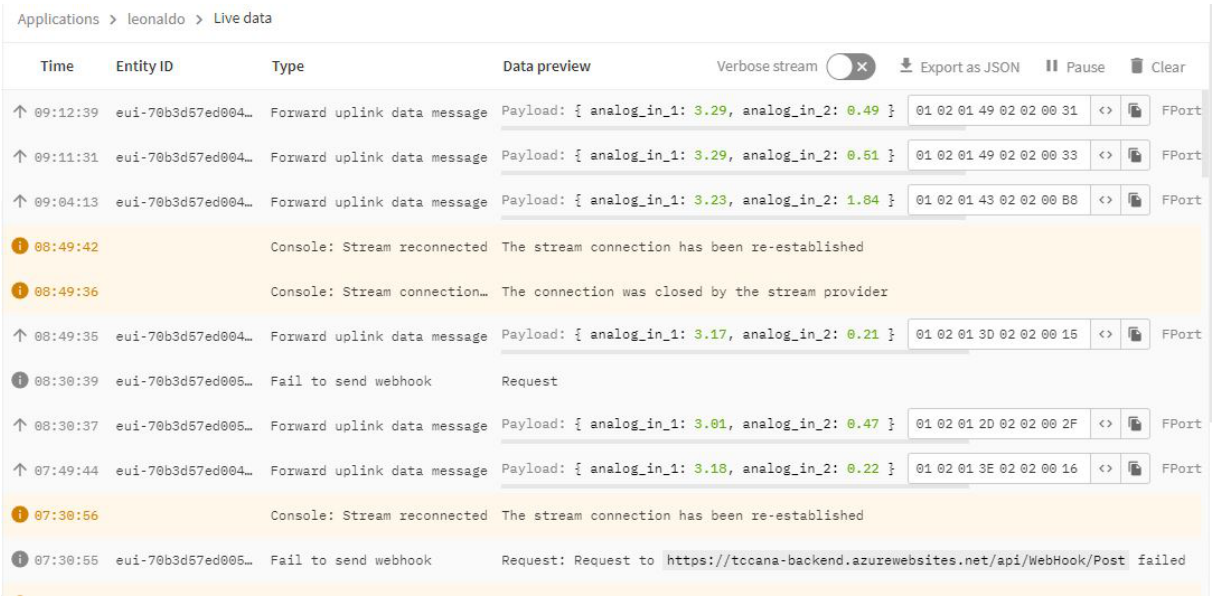


Fonte: (Autoria própria, 2022).

Conforme configurado através do *firmware*, o ANADevice envia os dados coletados pelo sensor ao *gateway*, que consequentemente mostrará na aplicação na TTN. Conforme demonstram as figuras 27 e 28, sendo que os dados são fornecidos e mostrados como dados analógicos, por consequência do uso de protocolo *Cayenne LPP*, que também é configurado pela própria TTN que já possui integração com esse protocolo. Desse modo, na TTN são recebidos sempre dois dados, um referente à leitura da bateria que é obtida através de um dos pinos do ESP32 e o outro dado é referente à altura do nível de água, obtido através do sensor ultrassônico.

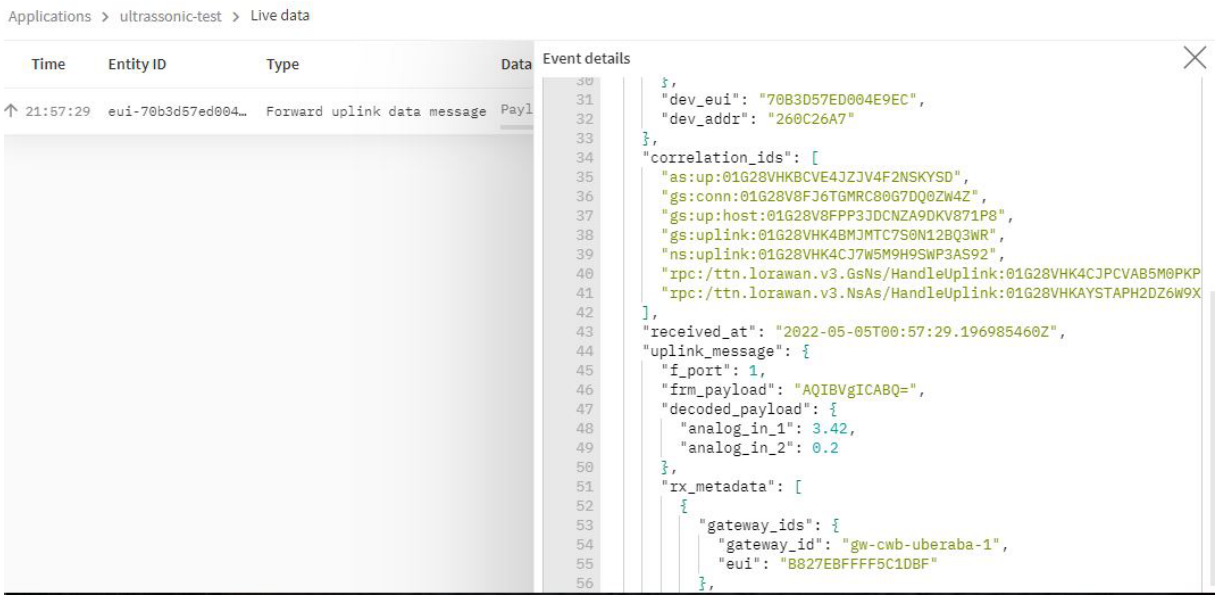
Através do *webhook* configurado na aplicação criada na TTN, os dados são enviados para a nuvem da Microsoft Azure e tratados pelo *backend* para que sejam mostrados no ANA-App.

Figura 29 – Dados recebidos na TTN



Fonte: (Autoria própria, 2022).

Figura 30 – Dados analógicos recebidos na TTN



Fonte: (Autoria própria, 2022).

3.8 Desenvolvimento da aplicação

Na sequência é apresentado o desenvolvimento do servidor *backend*, o banco de dados e o ANAApp para dispositivos móveis.

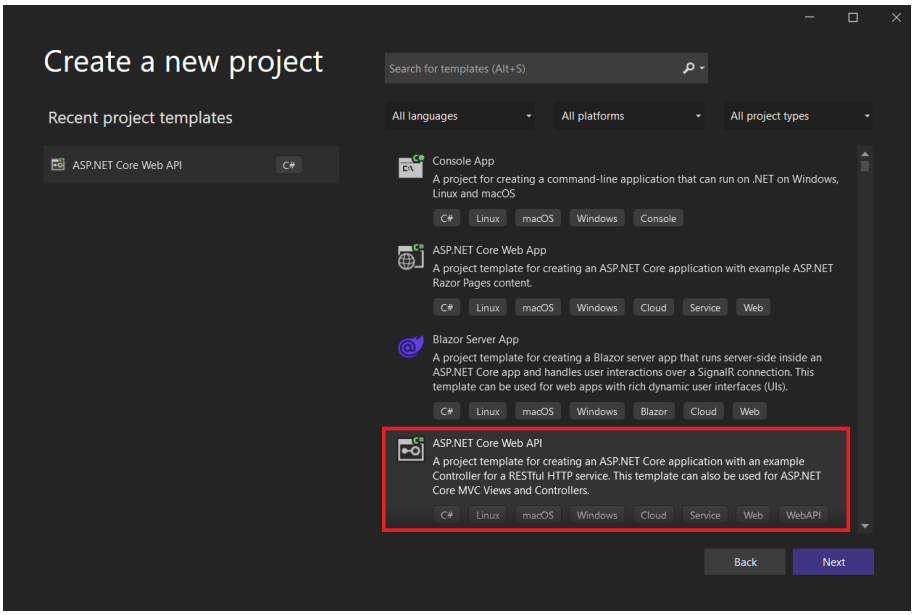
3.8.1 Servidor *backend*

Para que os usuários finais possam receber os dados obtidos pelos dispositivos instalados na caixa d'água fez-se necessário o desenvolvimento de um servidor *backend* para a integração entre dispositivo, plataforma TTN e ANAApp. A integração entre *backend* e TTN foi realizada através de *webhooks*, nos quais cada leitura do dispositivo é enviada à TTN e dispara um evento do tipo *HTTP POST* para o *endpoint* do servidor *backend* com os dados dessa leitura.

O servidor *backend* é responsável por receber pacotes da rede TTN, processar, armazenar em banco de dados e também receber e enviar mensagens para o ANAApp. A forma como se realiza essa comunicação é por meio de *APIs RESTful*.

A própria plataforma de desenvolvimento Microsoft Visual Studio possui alguns templates pré-configurados que podem ser usados para acelerar o desenvolvimento de aplicações. A Figura 31 mostra algumas dessas opções, como: *Console App*, *ASP.NET Core Web App*, *Blazor Server App* e em vermelho *ASP.NET Core Web Application Programming Interface (API)* que foi utilizado para a criação do servidor *backend*.

Figura 31 – ASP.NET Core Web API



Fonte: (Autoria própria, 2022).

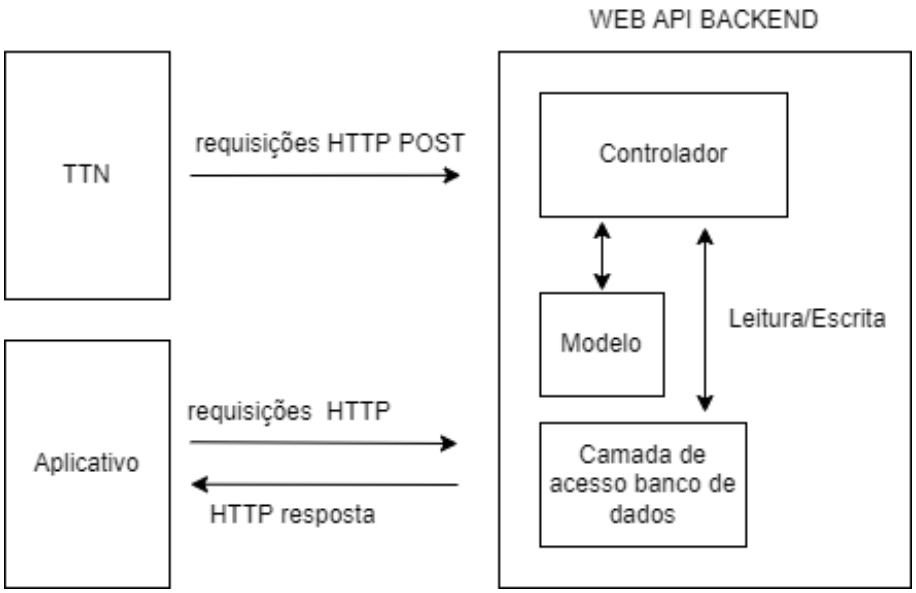
Um projeto *ASP.NET Core Web API* possui três módulos principais:

- ***Controlador***: responsável por todos os métodos de entrada das requisições para o servidor, como *"HTTP GET, POST, DELETE, PATCH"*. Criou-se dois controladores para essa aplicação, um para processar os pedidos do dispositivos e outro para processar os pedidos do usuário, são eles, respectivamentes *"UserUiController"* e *"WebHookController"*.

- ***Modelo***: responsável por definir a "forma"dos objetos e para facilitar na hora de receber e enviar dados nos pedidos *HTTP*. Por exemplo, o modelo *"EndDeviceEvent"* é um objeto que possui todas as variáveis que caracterizam um evento recebido, dentre essas variáveis temos: *"ReceivedAt"* que diz o horário que a mensagem foi recebida, *"AnalogIn1"* o dado lido pelo dispositivo, *"DevAddr"* o endereço do device entre outras.
- ***Camada de acesso ao banco de dados***: responsável por criar conexão com o banco de dados, ou seja, quando o servidor necessita realizar uma leitura ou escrita no banco de dados, esse pedido é realizado na camada de acesso ao banco.

A Figura 32 ilustra esses módulos e a arquitetura básica do *backend*:

Figura 32 – Arquitetura do *backend*

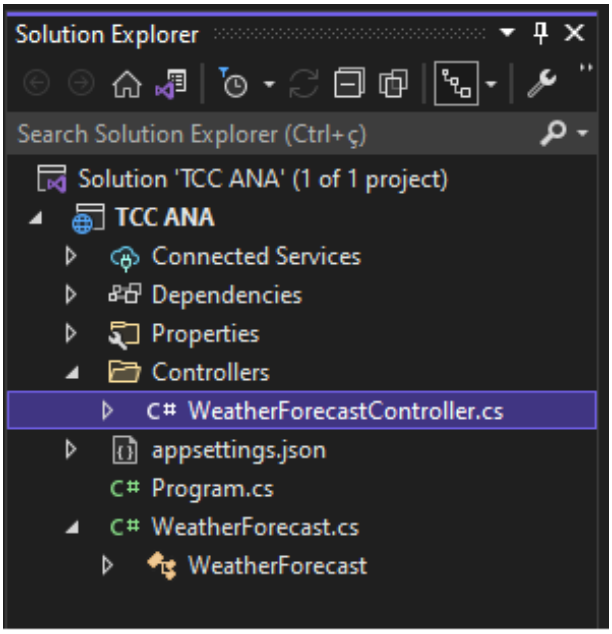


Fonte: (Autoria própria, 2022).

A estrutura básica de arquivos deste projeto possui uma pasta chamada *Controllers*, onde são criados arquivos de como serão definidas as requisições *http* ao servidor, um arquivo *Programa.cs*, no qual são configurados códigos de inicialização da aplicação e serviços, e também é criado um arquivo chamado *WeatherForecast.cs* para exemplificar como criar modelos de objetos para a aplicação.

A Figura 33 ilustra a árvore de arquivos, que é a estrura de pastas e arquivos gerados automaticamente quando um projeto *ASP.NET Core Web API* é criado:

Figura 33 – Estrutura da aplicação



Fonte: (Autoria própria, 2022).

Dentro da pasta controllers existe um arquivo chamado *WeatherForecastController.cs* que é um exemplo de *controller*. Ao abrir este arquivo encontra-se o seguinte código ilustrado na Figura 34:

Figura 34 – Arquivo "WeatherForecastController"

```
using Microsoft.AspNetCore.Mvc;

namespace TCC_ANA.Controllers
{
    [ApiController]
    [Route("[controller]")]
    3 references
    public class WeatherForecastController : ControllerBase
    {
        private static readonly string[] Summaries = new[]
        {
        "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
    };

        private readonly ILogger<WeatherForecastController> _logger;

        0 references
        public WeatherForecastController(ILogger<WeatherForecastController> logger)
        {
            _logger = logger;
        }

        [HttpGet(Name = "GetWeatherForecast")]
        0 references
        public IEnumerable<WeatherForecast> Get()
        {
            return Enumerable.Range(1, 5).Select(index => new WeatherForecast
            {
                Date = DateTime.Now.AddDays(index),
                TemperatureC = Random.Shared.Next(-20, 55),
                Summary = Summaries[Random.Shared.Next(Summaries.Length)]
            })
            .ToArray();
        }
    }
}
```

Fonte: (Autoria própria, 2022).

Primeiramente, renomeou-se o arquivo *"WeatherForecastController.cs"* para *"WebHook-Controller.cs"*, pois esse controlador será responsável pelo processamento dos pedidos recebidos da rede *TTN*. Removeu-se a varável *Summaries* pois não foi utilizada. Alterou-se o decorador *Route* de *[Route("[controller]")]* para *[Route("api/[controller]/[action]")]* o qual define que o padrão de rota do *endpoint* das requisições dentro do controlador será *"api/WebHook/[action]"*. Removeu-se todo o método *Get* e criou-se ou novo método que simplesmente retorna a *string* "olá mundo", como resultado para o primeiro controlador o código detalhado está demonstrado na Figura 35:

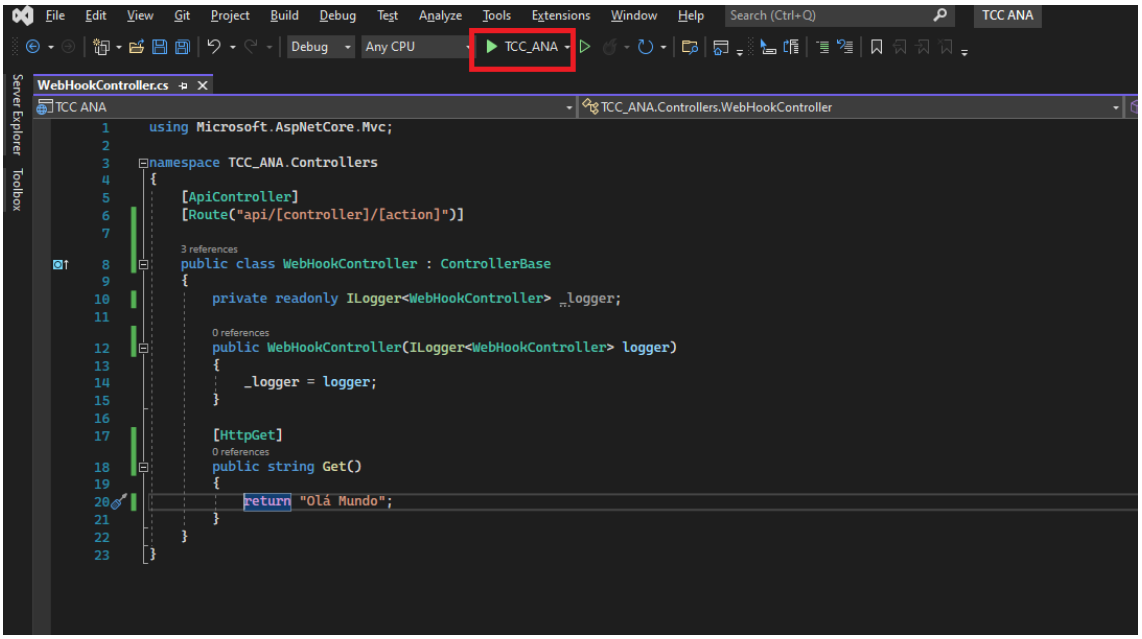
Figura 35 – Arquivo "WebHookController"

```
using Microsoft.AspNetCore.Mvc;

namespace TCC_ANA.Controllers
{
    [ApiController]
    [Route("api/[controller]/[action]")]

    3 references
    public class WebHookController : ControllerBase
    {
        private readonly ILogger<WebHookController> _logger;

        0 references
        public WebHookController(ILogger<WebHookController> logger)
        {
            _logger = logger;
        }

        [HttpGet]
        0 references
        public string Get()
        {
            return "Olá Mundo";
        }
    }
}
```

Fonte: (Autoria própria, 2022).

Realizaram-se testes executando a aplicação localmente, a *IDE* permite que se execute o servidor localmente, para isso basta pressionar a tecla de atalho *F5* no teclado, ou então, clicar sobre o ícone verde com o símbolo de *start* como nome do projeto ao lado, como mostrado no retângulo em vermelho na Figura 36.

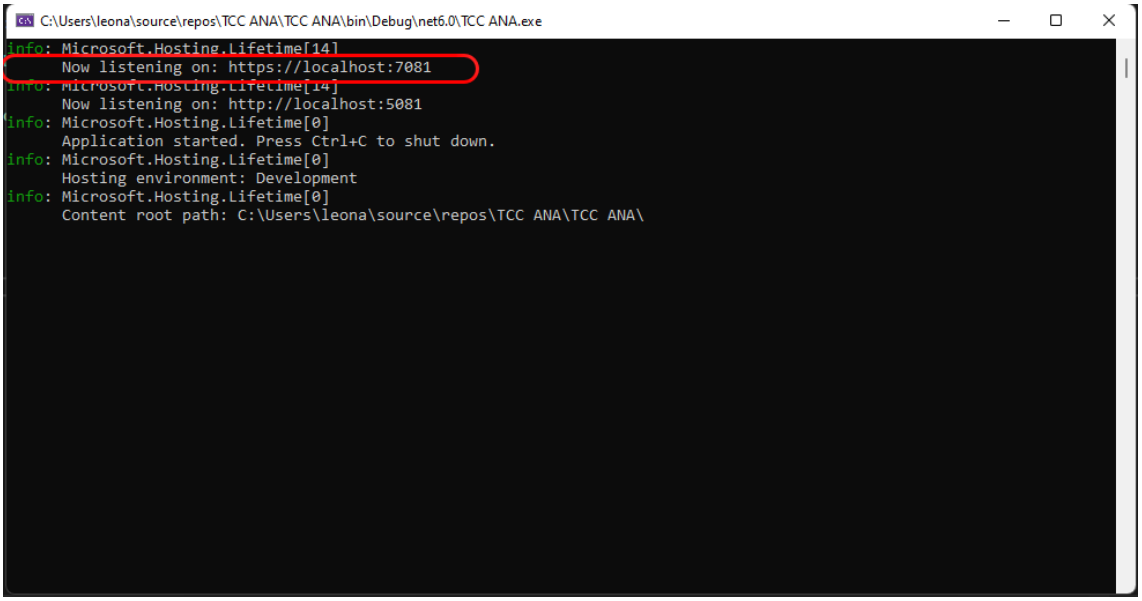
Figura 36 – Execução local da aplicação



Fonte: (Autoria própria, 2022).

Após executar a aplicação é aberta uma janela, chamada de *console*, na qual é possível verificar onde a aplicação está executando localmente, destacada na elipse em vermelho na Figura 37.

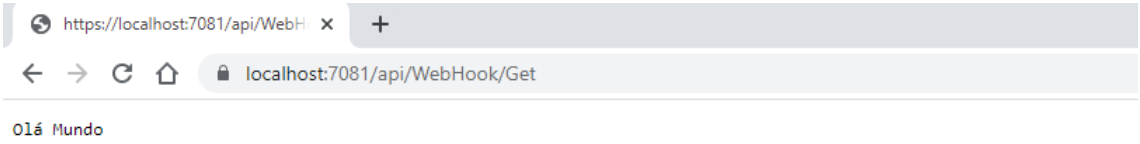
Figura 37 – Janela console



Fonte: (Autoria própria, 2022).

A aplicação está executando localmente em *"https//:localhost:8081"*. Definiu-se que a rota padrão para o controlador *WebHookController* é *"/api/WebHook/[action]"*. Nesse caso, o *action* é *get* pois acima do método temos o decorador *[HttpGet]*, portanto para enviarmos uma requisição *http* para a aplicação que está executando localmente precisamos enviar um pedido do tipo *get* para o *endpoint "https://localhost:7081/api/WebHook/Get"*. Pedidos do tipo *get* podem ser feitos direto do navegador apenas abrindo o a Uniform Resource Locator (URL), como mostrado na Figura 38.

Figura 38 – Resultado da requisição



Fonte: (Autoria própria, 2022).

O resultado desse pedido foi a *string* "Olá Mundo", pois foi o conjunto de caracteres adicionado como retorno do método. Isso foi uma exemplificação de como criar um projeto do tipo *ASP.NET Core Web API*, criar um controlador para lidar com requisições *http*, criar um pedido do tipo *Get*, executar localmente e visualizar os resultados. Os próximos passos são a criação de novos métodos para lidar com os pedidos da *TTN* que envia os dados através de uma requisição do tipo *POST* e os pedidos dos usuários. O código final do servidor do *backend* pode ser encontrado no link https://github.com/LeonaldoJunior/TCC-ANA-2022-Backend.git.

3.8.2 Leitura de evento

Os eventos enviados da plataforma *TTN* ao servidor possuem todas as informações referentes a LoRaWAN, a aplicação TTN, o dispositivo e as leituras realizadas pelo ANADevice. Nem todos os dados recebidos são relevantes para o projeto, portanto fez-se necessário um processamento para que sejam filtradas e salvas apenas as informações pertinentes ao projeto no banco de dados, dentre elas temos:

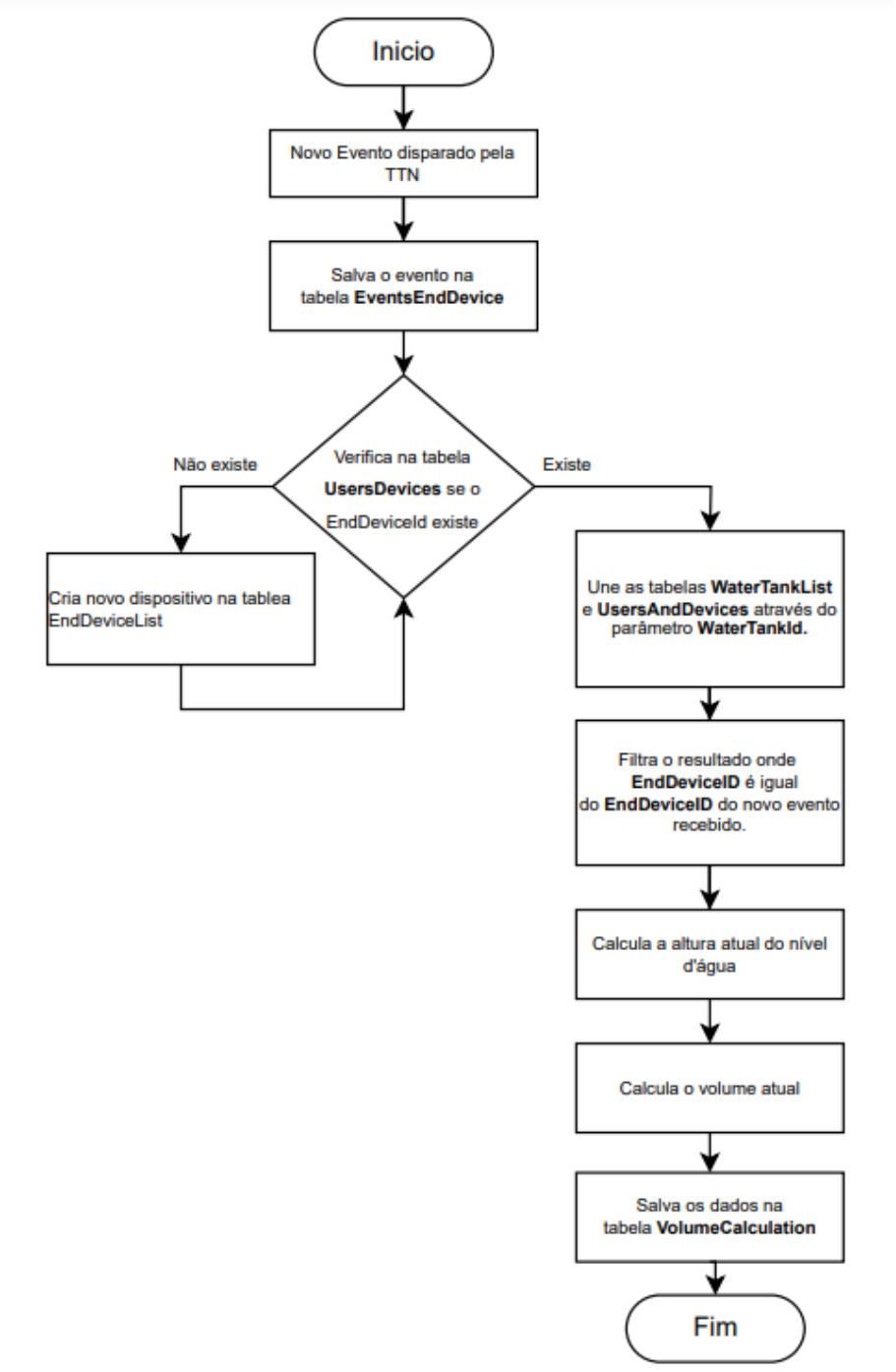
- ***EndDeviceId***: possui 20 caracteres e é identificador único de cada dispositivo.
- ***ReceivedAt***: o horário em que o evento foi recebido.
- ***AnalogIn1***: nível da bateria em volts.

- ***AnalogIn2***: leitura da distância entre o sensor e o nível de água em centímetros.

Criou-se então um método do tipo *POST*, pois é o tipo de requisição realizada pela *TTN* para o envio dos dados, dentro do controlador *WebHookController*. Esse método é responsável pelo processamento e armazenamento de novo registro no banco de dados na tabela *EventsEndDevice*. A lógica, ilustrada na Figura 39, para esse método foi primeiramente verificar se o evento recebido é de um novo dispositivo ou de um já existente, para isso verifica se o *EndDeviceId* recebido no evento existe na tabela *EndDeviceList*. No caso de um novo dispositivo o servidor cria um novo registro na tabela *EndDeviceList*, ou seja, os administrados do sistema não precisam cadastrar dispositivos novos, isso é feito de forma automática. No caso do evento com dispositivo ja existente o servidor apenas ignora e segue o fluxo normal do proceso.

O próximo passo é realizar o cálculo de volume atual do reservatório e nível de bateria. Para isso é necessário informações como: a leitura do nível, que encontra-se no evento recebido, da altura da caixa d'água, do raio da base; e do raio do topo, armazenada na tabela *WaterTankList*, porém essa tabela armazena as informações de todos os reservatório cadastrados, é necessário saber qual é o reservatório que o usuário cadastrou através do *ANAApp* para o *EndDeviceId*. Essas informações estão armazenadas na tabela *UsersAndDevices*. Para o cálculo do volume, primeiro calcula-se a altura em que o nível dentro do reservatório encontra-se. Com a altura do nível calculada, encontra-se o nível do reservatório. O diagrama da Figura 39 ilustra o processo de o servidor lidar com os eventos recebidos de leitura do dispositivo.

Figura 39 – Fluxograma leitura de eventos pelo servidor *backend*



Fonte: (Autoria própria, 2022).

3.8.3 Pedidos do *ANAApp*

O servidor também processa os pedidos realizados pelo *ANAApp*, tais pedidos podem ser:

- ***HttpGet GetUserbyId***: método *get*, retorna usuário do banco de dados.

- ***HttpGet GetAllWaterTank***: método *get*, retorna lista de caixas d'água do banco de dados.
- ***HttpPost NewUserDevice***: método *post*, cria um novo registro na tabela *UsersAndDevices* no banco de dados.
- ***HttpPost NewWaterTank***: método *post*, cria um novo registro na tabela *WaterTankList* no banco de dados.
- ***HttpGet GetAllDevicesByUserId***: método *get*, retorna todos os devices cadastrados de um usuário.
- ***HttpGet GetVolumeCalculationByUsersAndDevicesId***: método *get*, retorna o cálculo do volume atual de um dispositivo.
- ***HttpGet GetVolumeCalculationByUsersAndDevicesIdList***: método *get*, retorna a lista do cálculo do volume atual de um dispositivo.
- ***HttpGet GetVolumeCalculationByUsersAndDevicesIdAndDate***: método *get*, retorna a lista do cálculo do volume atual de um dispositivo filtrado por data.
- ***HttpGet GetSelectedDeviceByUserId***: método *get*, retorna o dispositivo selecionado de um usuário.
- ***HttpPatch PatchUsersAndDevicesById***: método *patch*, atualiza um registro na tabela *UsersAndDevices*.
- ***HttpDelete DelDeviceByDeviceId***: método *delete*, apaga um registro na tabela *UsersAndDevices*.

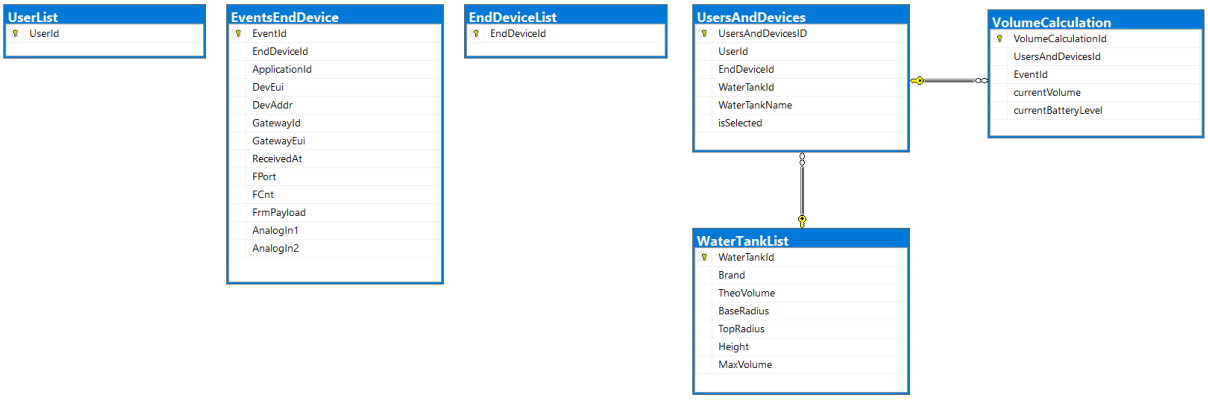
3.8.4 Banco de dados

Os dados dos dispositivos, usuários e informações de fabricantes precisavam ser guardados de forma relacional e em uma estrutura com comandos simples e específicos para gerenciar, armazenar, atualizar ou mesmo deletar registros. Portanto optou-se pela utilização de banco de dados do tipo *Standard Query Language (SQL) Server*. Para tanto, criou-se seis diferentes tabelas, mostradas na Figura 40.

- *EventsEndDevice*: todo novo evento transmitido pelos dispositivos é processado pelo *backend* e salvo nessa tabela.
- *WaterTankList*: catálogo de caixas d'água com informações dos fabricantes.
- *EndDeviceList*: lista de dispositivos cadastrados na rede *TTN*.

- *UserList*: lista de usuários cadastrados.
- *UsersAndDevices*: dados do usuário, dispositivo e reservatório.
- *VolumeCalculation*: dados dos cálculos de volume e bateria.

Figura 40 – Diagrama de banco de dados do servidor de aplicação



Fonte: (Autoria própria, 2022).

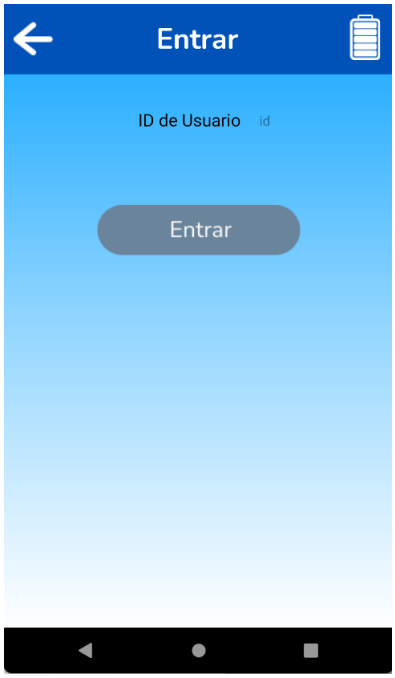
3.8.5 ANNAApp

Para que fosse possível o monitoramento em tempo real do nível pelo usuário foi desenvolvido o *ANAApp* um aplicativo para dispositivos móveis. O requisito do projeto *RF26* diz respeito ao *ANAApp* ser multiplaforma, o que significa, compatibilidade para sistemas operacionais do tipo *Android* e *IOS*. Portanto, optou-se pela utilização da *framework React Native* baseada em linguagem *javascript*, que possibilita o desenvolvimento de aplicativos multiplataforma.

3.8.5.1 Tela de login

A primeira vez que o *ANAApp* é aberto pelo usuário, este precisa fornecer um identificador único válido para que possa receber os dados do servidor. Esse identificador é salvo na tabela do banco de dados *UserList*, novos usuários são cadastrados nessa tabela pelos desenvolvedores do sistema. O identificador único é um código de cinco dígitos podendo conter letras maísculas, minúsculas, números e caracteres especiais. Escolheu-se essa maneira para autenticar usuários pois o sistema não requer maior nível de segurança visto que os dados compartilhados entre servidor e *ANAApp* não possuem nenhuma informação sensível. A Figura 41 mostra a tela onde os usuários realizam o login:

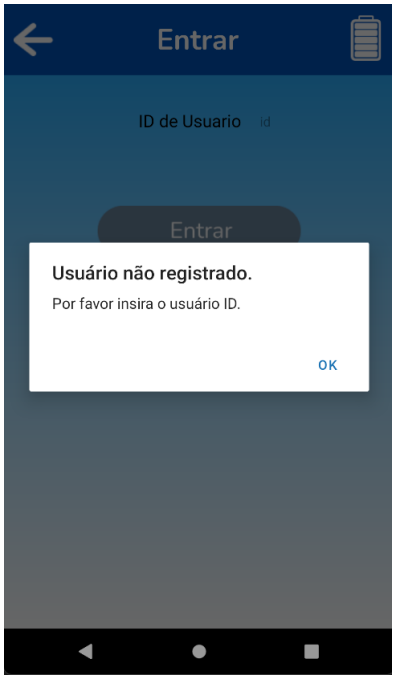
Figura 41 – Tela de login



Fonte: (Autoria própria, 2022).

Quando o usuário ainda não realizou a autenticação, ele é redirecionado para a tela de login e é exibida a mensagem que está na Figura 42:

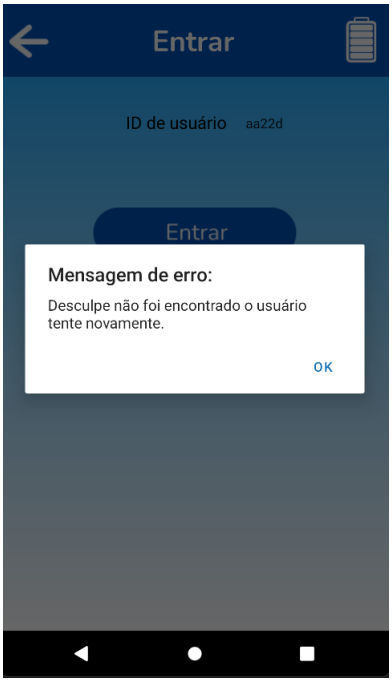
Figura 42 – Tela de login mensagem



Fonte: (Autoria própria, 2022).

Após o usuário inserir seu código e pressionar o botão "Entrar", o *ANAApp* faz um pedido *HttpGet GetUserbyId* para o servidor *backend* passando como parâmetro o código digitado. Caso o servidor não encontre o código digitado na tebela *UserList* é exibida a mensagem conforme mostrado na Figura 43.

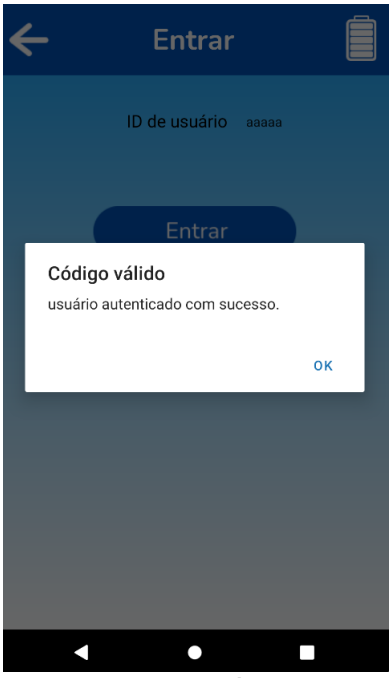
Figura 43 – Tela de login mensagem de erro



Fonte: (Autoria própria, 2022).

Caso o código seja válido, então é exibida a mensagem de sucesso, ver Figura 44, e o usuário é redirecionado para a tela principal após dois segundos. Essa informação é salva localmente no *ANAApp*, assim não é necessário autenticar na próxima vez que o *ANAApp* for aberto.

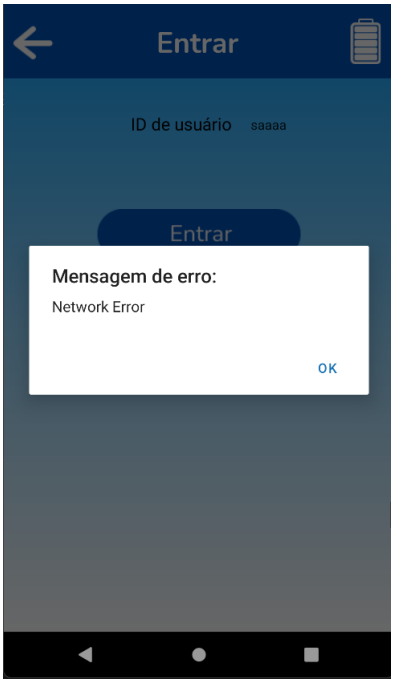
Figura 44 – Tela de login mensagem de sucesso



Fonte: (Autoria própria, 2022).

Quando algum erro inesperado ocorre no pedido para o servidor de *backend* a mensagem de erro é exibida e na segunda linha o tipo de erro, conforme a Figura 45.

Figura 45 – Tela de login mensagem de erro inesperado



Fonte: (Autoria própria, 2022).

3.8.5.2 Tela novo dispositivo

Após o usuário ter sido autenticado com sucesso, é necessário registrar seu dispositivo. Em resumo, o usuário precisa informar qual o reservatório em que o dispositivo será instalado. Quando a tela "Novo Dispositivo"é aberta então é feito o pedido *HttpGet GetAllWaterTank* ao *backend* que retorna a lista de caixas d'água existentes na tabela do banco de dados *WaterTankList*. A tela "Novo Dispositivo"possui três campos e um botão, são eles:

- ***ID do dispositivo***: Esse campo é do tipo entrada de texto, onde o usuário precisa digitar o identificador único do dispositivo, que é um conjunto de 20 caracteres podendo ser letras maísculas ou minúsculas, números e caracteres especiais. Esse Identity (ID) é fornecido junto com o dispositivo. Exemplo: "eui-70b3d57ed0051a28".
- ***Nome do dispositivo***: Campo do tipo entrada de texto, o usuário escolhe um nome para ser usado como referência.
- ***Selecione a caixa d'água***: Campo do tipo *drop down list* em que o usuário precisa selecionar a caixa d'água na qual o dispositivo será instalado. Esse passo é de extrema importância pois interferirá diretamente nos cálculos de volume do reservatório.
- ***Botão Salvar***: Após os usuário completar os três campos, ele precisa pressionar o botão "Salvar"para enviar essas informações para o servidor *backend*. Caso o usuário não complete algum desses campos, o botão fica desabilitado impossibilitando que o usuário cadastre um novo dispositivo.

A Figura 46 ilustra a tela "Novo Dispositivo".

Figura 46 – Tela novo dispositivo

Fonte: (Autoria própria, 2022).

Quando o usuário clica sobre o campo "Selecione a caixa d'água"é exibida uma lista com as principais marcas e modelos, conforme mostra a Figura 47.

Figura 47 – Lista de caixas d'água

Fonte: (Autoria própria, 2022).

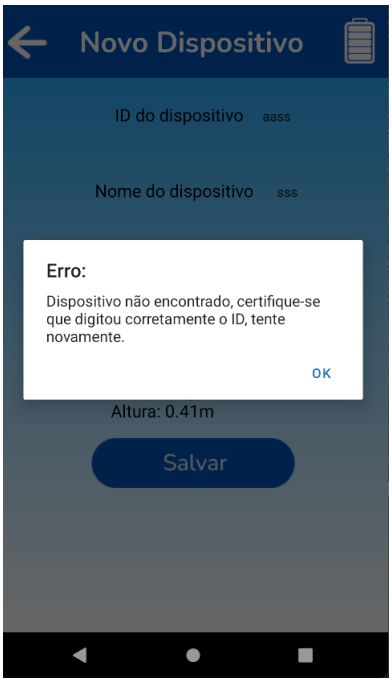
Quando o usuário clica sobre uma das caixas na lista são exibidas suas principais características, conforme mostra a Figura 48.

Figura 48 – Caixa d'água selecionada

Fonte: (Autoria própria, 2022).

Caso o usuário digite um ID do dispositivo que não seja válido é exibida a mensagem de erro da Figura 49.

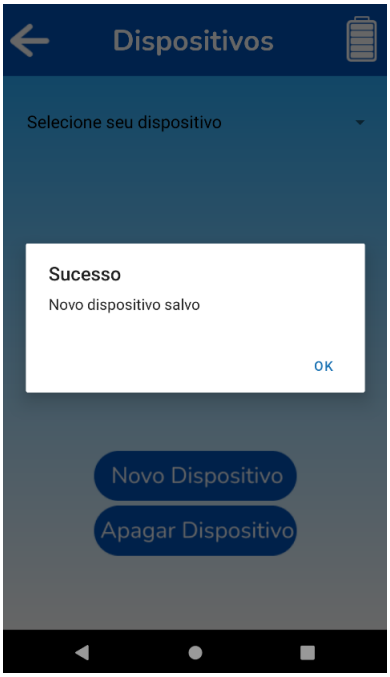
Figura 49 – ID inválido



Fonte: (Autoria própria, 2022).

Já quando o usuário preenche todos os campos e digita um ID do dispositivo válido, após clicar no botão "Salvar"é realizado o pedido *Hyper Text Transfer Protocol (HTTP)Post NewUser-Device*, que responde com a mensagem da Figura 50.

Figura 50 – Novo dispositivo salvo



Fonte: (Autoria própria, 2022).

Depois do cadastro bem sucedido de um novo dispositivo o usuário é redirecionado para a tela "Dispositivos".

3.8.5.3 Tela dispositivos

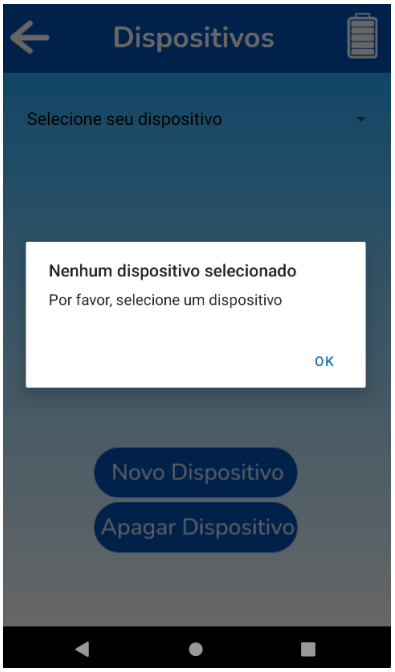
O usuário pode ter mais de um dispositivo, então a tela "Dispositivos"permite que seja selecionada sobre qual caixa d'água o usuário gostaria de ver os dados de leitura.

Essa tela possui um campos e dois botões, são eles:

- ***Selecione seu dispositivo***: campo do tipo *drop down list* em que o usuário precisa selecionar o dispositivo sobre o qual receberá os dados.
- ***Botão Novo Dispositivo***: quando pressionado o usuário é redirecionado para a tela "Novo Dispositivo"para registrar outro dispositivo.
- ***Botão Apagar Dispositivo***: quando pressionado, apaga o dispositivo selecionado.

Quando o usuário ainda não selecionou nenhum dispositivo a seguinte mensagem é mostrada, conforme a Figura 51.

Figura 51 – Nenhum dispositivo selecionado



Fonte: (Autoria própria, 2022).

Após a tela da Figura 51 ser aberta, é enviado ao servidor o pedido *HttpGet GetAllDevicesByUserId*, que retorna todos os dispositivos cadastrados pelo usuário salvos na tabela do banco de dados *UsersAndDevices*. Clicando sobre o campo ao lado de "Selecione seu dispositivo", são exibidos esses dispositivos, como mostra a Figura 52.

Figura 52 – Nenhum dispositivo selecionado

Fonte: (Autoria própria, 2022).

Quando é selecionada uma das opções exibidas, então o servidor recebe o pedido *HttpPatch PatchUsersAndDevicesById* que atualizará o registro da tabela *UsersAndDevices* para

indicar sobre qual dispositivo o usuário gostaria de receber os dados. Após ser processado o pedido, a tela da Figura 53 é exibida com as características do dispositivo selecionado, são elas: dispositivo selecionado, marca, volume, raio da base, raio do topo, altura.

Figura 53 – Dispositivo selecionado

Fonte: (Autoria própria, 2022).

3.8.5.4 Tela meu nível

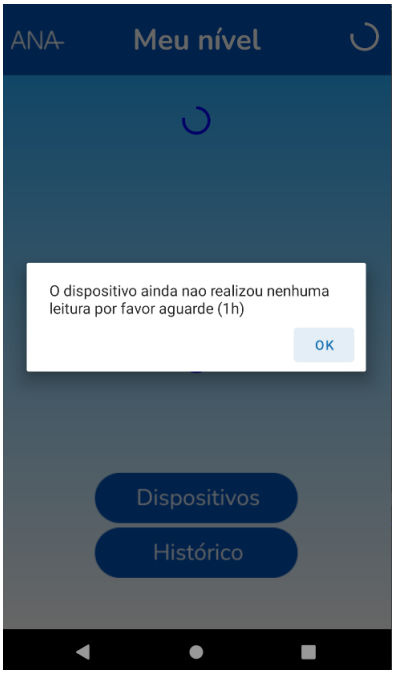
Essa é a principal tela do *app*, pois é nela que as informações do nível do reservatório são exibidas.

Essa tela possui dois botões:

- ***Dispositivos***: quando pressionado o usuário é redirecionado para a tela "Dispositivos" na qual pode selecionar o dispositivo que gostaria de receber os dados.
- ***Histórico***: quando pressionado o usuário é redirecionado para a tela "Histórico" na qual pode ver os últimos dados recebidos.

Quando o usuário seleciona um dispositivo que ainda não realizou nenhuma leitura a mensagem da Figura 54 é exibida.

Figura 54 – Meu nível sem leitura



Fonte: (Autoria própria, 2022).

Após o dispositivo ter realizado alguma leitura então são exibidos estes dados, ilustrados na Figura 55.

Figura 55 – Meu nível leitura

Fonte: (Autoria própria, 2022).

A primeira linha após o título da tela é exibido o nome do dispositivo escolhido pelo usuário, na linha seguinte "Leitura atual" e logo abaixo o volume atual do reservatório em litros. A última linha exibe a data e a hora em que foi realizado a coleta dos dados pelo *hardware*.

Ao centro da tela é exibido uma imagem que representa a porcentagem aproximada proporcional ao volume do reservatório, podendo variar de 0% para o menor volume a 100% o maior volume. Por exemplo, caso o reservatório seja uma caixa d'água com capacidade de 150 L "0% "indica 0 L e 100% 150 L.

3.8.5.5 Tela histórico

A tela histórico permite que o usuário consulte as leituras realizadas pelo dispositivo, as quais são exibidas em ordem decrescente, ou seja, o dado mais recente encontra-se no topo, enquanto os dados mais antigos no final da lista.

A disposição dos dados é apresentada na forma de tabela, na qual no cabeçalho temo-se, ver Figura 56:

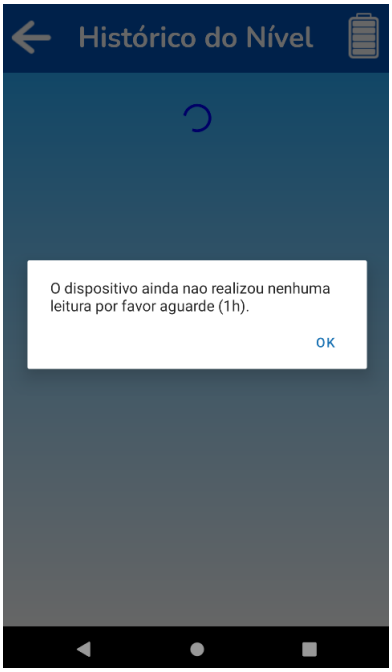
- ***Volume (%)***: Volume da leitura em porcentagem.
- ***Volume (L)***: Volume da leitura em Litros.
- ***Data***: Data em que a leitura foi realizada no formato dia/mês/ano (dd/mm/yyyy).
- ***Hora***: Hora em que a leitura foi realizada no formato hora:minuto:segundo (hh:mm:ss).

Figura 56 – Histório do nível

Fonte: (Autoria própria, 2022).

Quando ainda não existem dados no histórico a mensagem exibida é a que está apresentada na Figura 57.

Figura 57 – História do nível sem dados - Parte 1



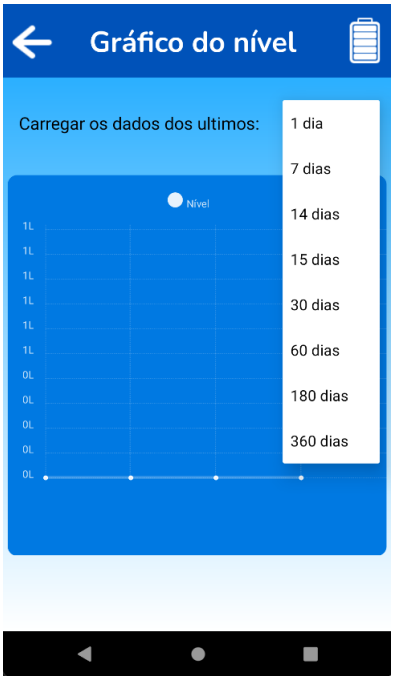
Fonte: (Autoria própria, 2022).

A tela da Figura 57 ainda possui o botão "Gráfico", que quando pressionado redireciona o usuário para a tela de gráfico.

3.8.5.6 Tela gráfico

A tela gráfico permite que o usuário visualize o histórico de nível graficamente. Os usuário precisam selecionar a partir de quando eles gostariam de visualizar os dados em dias, que pode ser no mínimo a partir do último dia ou no máximo dos últimos 360, conforme é ilustrado na Figura 58.

Figura 58 – Histório do nível sem dados - Parte 2



Fonte: (Autoria própria, 2022).

3.8.5.7 Tela histórico da bateria

Todas as telas possuem no canto superior direito um ícone que indica o seu atual nível, caso o usuário pressione esse ícone ele é direcionado para a tela de histórico do nível de bateria. Os dados são exibidos em ordem descrecente, portanto o dado mais recente encontra-se no topo, enquanto os dados mais antigos ao final da lista.

A disposição dos dados é apresentada na forma de tabela, na qual no cabeçalho tem-se, (a tela está ilustrada na Figura 59):

- ***Bateria(%)***: Nível da bateria em porcentagem.
- ***Bateria(V)***: Nível da bateria em volts.
- ***Data***: Data em que a leitura foi realizada no formato dia/mês/ano (dd/mm/yyyy).
- ***Hora***: Hora em que a leitura foi realizada no formato hora:minuto:segundo (hh:mm:ss).

Figura 59 – Histório da bateria

Fonte: (Autoria própria, 2022).

3.8.6 Hospedagem

A aplicação foi hospedada em *cloud* através do provedor da *Microsoft* chamado de *Azure*. Optou-se por esse provedor em virtude de os estudantes terem familiaridade com o ambiente e pela disponibilidade a licença para estudantes por um período de um ano. Criou-se então uma subscrição do tipo estudante, que disponibiliza o recurso *Azure App services* destinado a hospedagem de aplicativos *web* com limite de até 10 aplicações e 1GB de armazenamento.

Através da IDE da *Microsoft Visual Studio* foi possível realizar a autenticação com a *Azure* e utilizar ferramentas para fazer a implantação do servidor do *backend*.

O banco de dados não é disponibilizado pela inscrição de estudante do *Azure*, então optou-se por fazer a hospedagem pela empresa "ITMNetworks", que disponibiliza servidores *SQL* através de uma inscrição mensal no valor de R$28,99. As tabelas foram criadas através de comando *SQL*, e a integração com o servidor foi realizada com o *connection string* um código único disponibilizado pelo provedor para realizar a autenticação e autorização para acesso ao banco de dados.

3.8.7 Repositório

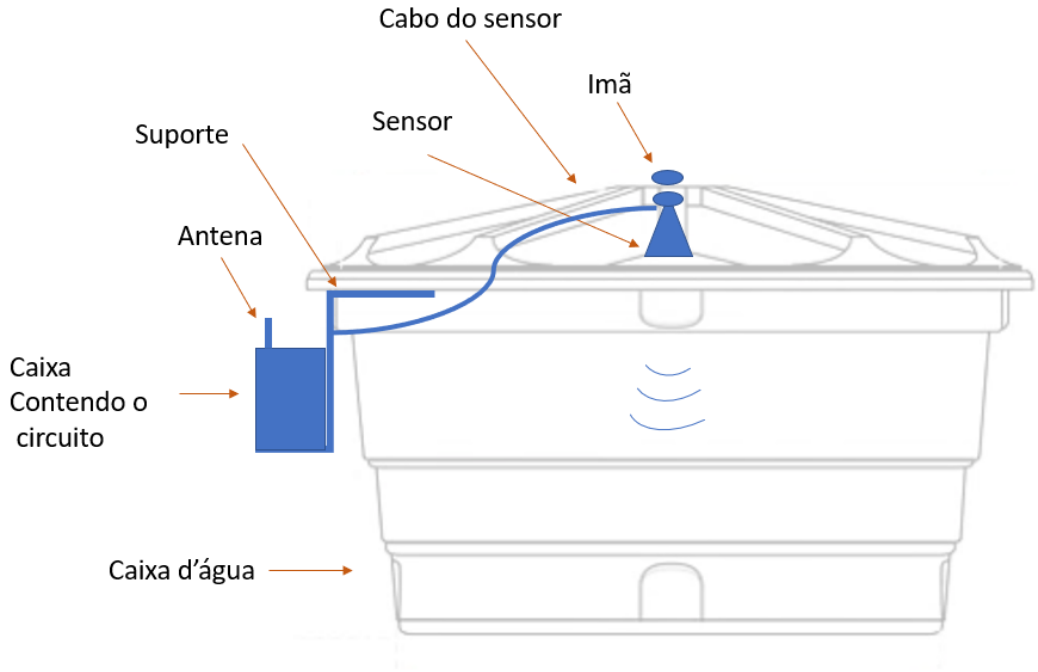
Todas as aplicações: *ANAApp*, *backend* e *firmware* foram armazenadas na plataforma de versionamento remoto *Github*, para facilitar o trabalho conjunto, e rastrear as alterações realizadas no código fonte, são elas:

- ***Firmware***: https://github.com/LeonaldoJunior/TCC-ANA-2022-Fw.
- ***ANAApp***: https://github.com/LeonaldoJunior/TCC-ANA-2022-App.
- ***Backend***: https://github.com/LeonaldoJunior/TCC-ANA-2022-Backend.

3.9 Design do protótipo

Para acomodar o dispositivo de forma adequada, o esquema da Figura 60 foi projetado.

Figura 60 – Esquema de fixação do dispositivo



Fonte: (Autoria própria, 2022).

Na Figura 60, percebe-se que a estrutura projetada é bastante simples. Trata-se de uma caixa plástica com característica IP66 que garante proteção contra água e poeira. Dentro está acondicionada a placa de controle do dispositivo. Essa caixa está sendo segurada por um suporte do tipo mão francesa, responsável por um contrabalanço entre a caixa plástica e a tampa da caixa d'água. Também pode-se notar que a fixação do sensor foi feita através de ímãs de neodímio (ver Figura 62), tendo em vista que o uso de qualquer tipo de cola em contato com a água seria inviável. Para este projeto, procurou-se realizar uma instalação simples e limpa para o usuário, sem qualquer prejuízo ou risco à saúde.

Para posicionar o sensor de maneira adequada, optou-se por utilizar um cone fabricado em impressora Tridimensional (3D) conforme projeto previamente disponibilizado na rede, (THINGIVERSE, 2014). Esse cone foi utilizado para reduzir o efeito das bordas, pois caso o

sensor ficasse muito próximo à borda da caixa, ou se a caixa fosse pequena, as laterais da mesma podem ser detectadas pelo sensor, ao invés da superfície da água. Com a adoção do cone fabricado em 3D, observou-se um comportamento mais eficiente no que diz respeito à detecção do líquido ao invés de detecção de bordas. Além disso, a utilização desse cone influenciou na instalação do dispositivo, tornando-a mais fácil, devido ao melhor posicionamento do sensor com a utilização do cone. A Figura 61 ilustra o cone desenvolvido.

Figura 61 – Cone do sensor

Fonte: (Autoria própria, 2022).

Figura 62 – Fixação do sensor na caixa d'água

Fonte: (Autoria própria, 2022).

Foi analisada outra posição para o sensor, porém de acordo com as normas que determinam o volume das caixas d'água para os fabricantes, observou-se que a melhor posição para o sensor, a fim de se obter o cálculo de volume de água o mais próximo da realidade possível, seria posicioná-lo no ponto central e mais alto da tampa. Caso o sensor fosse posicionado mais próximo à borda da caixa, a fixação por ímãs não seria necessária, uma vez que a própria mão francesa faria esta sustentação.

Contudo, notou-se que o distanciamento do sensor, colocando-o no ponto mais alto possível, na parte interna da caixa, seria primordial uma vez que a capacidade de leitura do mesmo, inicia em 20 cm e de acordo com a norma NBR1479 (ABNT, 2002), que determina especificações de caixas d'água, existem 6 cm não aproveitáveis na altura das caixas, devido a presença de cano na parte superior das caixas. Com isso, foi possível obter maior precisão nos dados coletados via sensor.

A Figura 63 mostra a versão final do design do protótipo. De acordo com os testes o produto cumpre o requisito de facilidade na instalação e portanto foi aprovado conforme projeto prévio.

Figura 63 – Design do protótipo versão final

Fonte: (Autoria própria, 2022).

3.10 Cálculo de volume

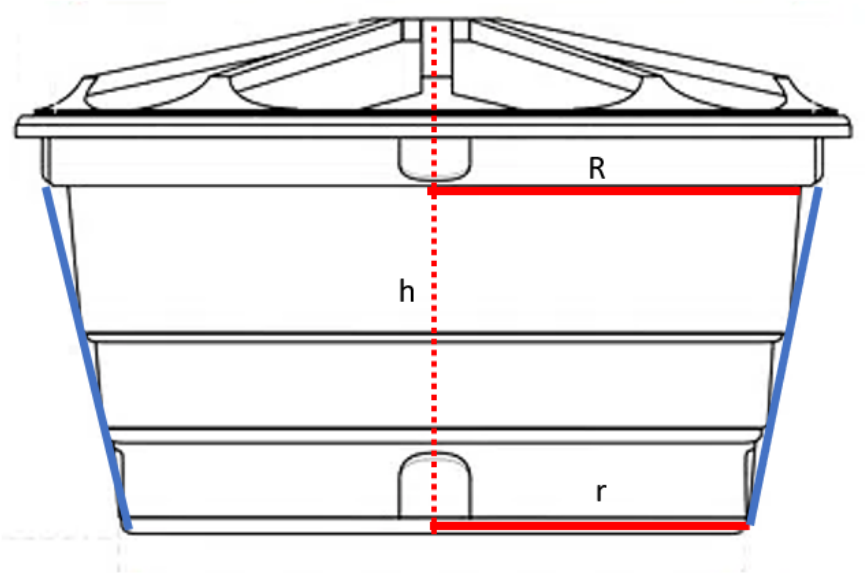
O cálculo do volume da água presente na caixa d'água foi aproximado pela equação do volume do tronco de cone.

$$V = \frac{\pi * h}{3} * (R^2 + Rr + r^2), \tag{6}$$

Onde R é o raio da base maior, r o raio da base menor e h a altura da caixa d'água, conforme definido pelo catálogo do fabricante. Pode-se observar pela Figura 63, que existem três troncos de cone no formato da caixa estudado, porém para se calcular o volume de cada um dos três troncos de cone, seria necessário possuir os raios das bases maior e menor dos três troncos de cone separadamente para obter o cálculo mais próximo do real. No entanto, os catálogos dos fabricantes contemplam apenas dois raios, referentes à base menor, ou seja, o fundo da caixa e o raio da base maior, que é o topo da caixa. Dessa forma, o tronco de cone aproximado para o cálculo foi o observado na Figura 64 com as bordas em azul. Nota-se que o volume dessa forma será maior do que o real, contudo, a norma Associação Brasileira de Normas Técnicas (ABNT) a NBR1479 (ABNT, 2002) define que o volume nominal pode ser maior ou menor do que o volume real em até 6% além de definir que a altura do nível de água deve ser limitado a até 6 cm distante da altua máxima da caixa, em virtude da posição do furo para o cano de entrada de água.

Como o sensor utilizado é capaz de medir a partir de 20 cm e a altura da tampa da caixa é de 14 cm, valendo-se do fato que a água estará 6 cm abaixo da altura máxima da caixa, decidiu-se posicionar o sensor na parte central do topo da tampa da caixa, pois com isso, somando-se 14 cm com 6 cm da norma, tem-se 20 cm da resolução do sensor, ou seja, toda vez que o sensor enviar medidas iguais a 20 cm é considarada caixa cheia ou seja, volume igual a 100%. O cálculo do volume do tronco de cone está implementado no *backend*. O volume máximo determinado e implementado é igual ao volume nominal do fabricante, mesmo para casos em que o cálculo aproximado resulte em mais litros do que o volume nominal.

Figura 64 – Cálculo do volume da caixa



Fonte: (Autoria própria, 2022).

3.11 Erro estimado no cálculo do volume

Para a aproximação do volume nominal de cada reservatório, utilizou-se a equação (6), que se refere ao volume de tronco de cone. No entanto, valendo-se apenas das medidas nominais dos reservatórios, para cada tamanho de reservatório o erro encontrado era diferente, pois os volumes nominais não correspondem ao volume efetivo de cada reservatório. Por isso, utilizou-se a norma ABNT correspondente, para o correto cálculo de erro (ABNT, 2011). Portanto, se a norma não fosse levada em consideração, o erro aproximado seria de até 26%, no caso de aproximação pela equação do volume do tronco de cone, sem levar em conta as tolerâncias específicas para cada reservatório.

Para compor a tabela 7, foram utilizadas as medidas das principais caixas d'água, modelo tronco de cone, do fabricante FORTLEV, utilizando as aproximações normatizadas, com isso, tem-se que o maior erro aproximado igual a 6,92%,

Tabela 7 – Erro estimado no cálculo do volume

Fabricante	Volume nominal (l)	Raio da base menor (m)	Raio da base maior (m)	Altura (m)	Volume máximo permitido(l)	ERRO
FORTLEV	100	0,27	0,365	0,41	105,29	5,29%
FORTLEV	150	0,31	0,435	0,43	152,08	1,39%
FORTLEV	250	0,38	0,515	0,5	263,44	5,38%
FORTLEV	310	0,38	0,515	0,54	288,53	6,92%
FORTLEV	500	0,48	0,61	0,58	464,68	7,06%
FORTLEV	750	0,5	0,675	0,73	710,03	5,33%
FORTLEV	1000	0,58	0,755	0,76	957,28	4,27%
FORTLEV	1500	0,72	0,875	0,83	1494,20	0,39%
FORTLEV	2000	0,78	0,94	0,9	1900,07	5,00%

Fonte: (Autoria própria, 2022).

Com isso, pode-se observar que o resultado final de volume estimado não compromete a utilização do ANA Device, pois conta-se com estimativa de volume tendo em vista as diferenças presentes nas normas ABNT 15682 (ABNT, 2002) e ABNT 14799 (ABNT, 2011) com erro máximo dentro do esperado.

4 RESULTADOS

4.1 Testes de consumo

O protótipo é alimentado por duas pilhas alcalinas do tipo AA, ligadas em série, sendo que cada uma possui 1,5 V totalizando 3 V. O módulo do microcontrolador opera entre 3 V e 5 V sem qualquer prejuízo, o mesmo ocorre com o módulo do sensor ultrassônico.

Para a ligação do sensor, utilizou-se um pino *General Purpose Input/Output* (General Purpose Input/Output (GPIO)), pois quando o sensor é alimentado diretamente pelo pino Tensão em Corrente Contínua (VCC) do módulo ESP32. Mesmo em estado *deep sleep*, o sensor não era desligado completamente, consumindo em torno de 10 mA, o que drenava a bateria em pouco tempo. Portanto, o modo mais comum de ligação dos pinos do sensor foi desconsiderado. Adotou-se então a alimentação através de pino GPIO, pois sabe-se que esse tipo de pino tem corrente cortada e igual a zero enquanto o ESP32 está em estado *deep sleep*. Para alcançar este resultado, vários testes foram feitos e adotou-se então a seguinte ligação; VCC do sensor ligado ao pino GPIO 12 do ESP32, *trigger* do sensor ligado ao pino GPIO 26 do ESP32, *echo* do sensor ligado ao pino GPIO 33 do ESP32 e por fim o pinho *GND (ground)*, ou terra do sensor ligado ao pino correspondente ao *GND* do microcontrolador ESP32.

Dessa forma, foi possível observar que enquanto transmite dados, em estado ativo o conjunto contendo o microcontrolador, rádio LoRa e sensor, são responsáveis pelo consumo de 58 mA durante o intervalo de 2 segundos (tempo de transmissão). Já para o restante do tempo, o conjunto consome 32 μA. Considerando que os dados são enviados em intervalos de uma hora, ou seja, são enviadas 24 medidas por dia, totalizando 48 segundos ao dia em que o circuito opera em estado ativo, consumindo 58 mA. Pode-se concluir que o dispositivo consome 16 uAh. Considerando que uma pilha alcalina possui em torno de 2100 mAh e que no estado *deep sleep* o conjunto consome apenas 32 μA durante 3552 segundos pelo intervalo de uma hora, portanto o circuito consome em torno de 0,03625 mAh.

$$C = \frac{Cb[mAh]}{i[mA]}, \tag{7}$$

Na equação (6), C é a quantidade em horas que durará a bateria, Cb é a capacidade da bateria em mAh e i é a corrente consumida em determinado tempo.

$$C = \frac{2100[mAh]}{0{,}032[mA]} = 65.625[h], \tag{8}$$

Para o caso do dispositivo testado, temos que Cb é igual a 2100 mAh, e i é igual a 32 μA Com isso, é possível concluir que o conjunto alimentado por duas pilhas alcalinas AA, tem duração de bateria teórica calculada de 65.625 horas, o que resulta em torno de 7 anos de duração da bateria. Contudo, na prática, como ainda temos os picos de tranmissão que consomem 58

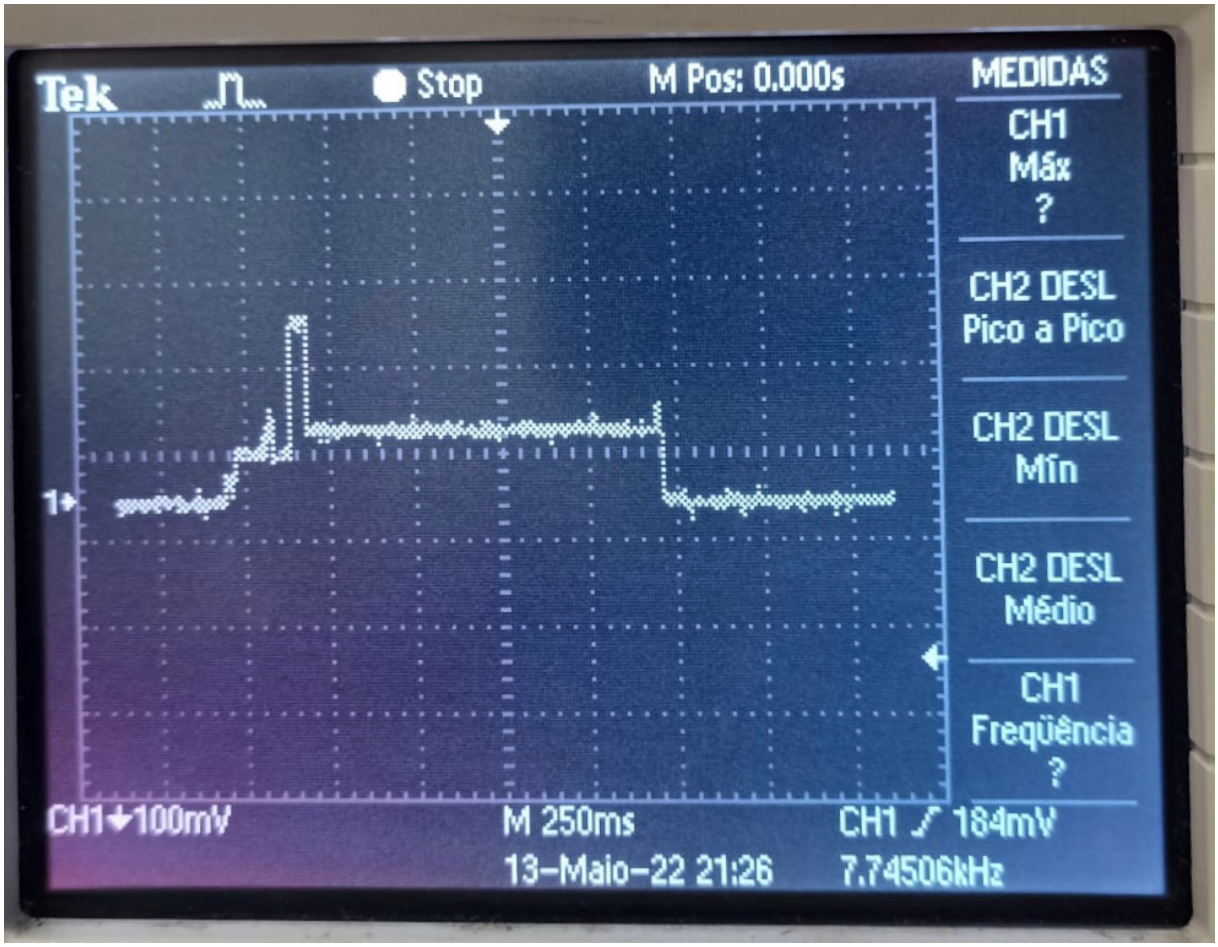
mA, precisamos incluir este fator no cálculo do consumo. Porém como este pico ocorre durante dois segundos a cada hora do dia, ao decorrer de um dia terá um consumo total de 58 mA por apenas 48 segundos, com isso sabe-se que para consumir o total de 58 mA durante uma hora completa, será necessário 75 dias, se dividirmos então 2100 mAh por 58 mA, teremos uma duração de 36,2 horas, porém como esse consumo de 58 mA leva 75 dias para ocorrer, multiplicamos então 36,2 dias por 75 e chegamos a 2715,51 dias aproximadamente

$$C = \frac{2100[mAh]}{58[mA]} = 36{,}2[h], \tag{9}$$

Este resultado também demonstra que o dispositivo tem capacidade de duração teórica maior que 7 anos. No entanto, sabe-se que na prática o dispositivo deixará de funcionar bem antes que a bateria de esgote, então o tempo na prática, será drasticamente reduzido, pois o cálculo considera o consumo da carga até zerar a bateria, o que na prática não acontece. Contudo, foi comprovado que o dispositivo projetado, atende o requisito de duração da bateria por pelo menos dois anos.

A Figura 65 mostra o osciloscópio, demonstrando um período de transmissão completo, que representa a curva do consumo de corrente durante a operação, o qual chega a 58 mA durante a transmissão e em modo *deep sleep* cai para 32 μ A.

Figura 65 – Osciloscópio - Intervalo de transmissão



Fonte: (Autoria própria, 2022).

4.2 Testes de integração

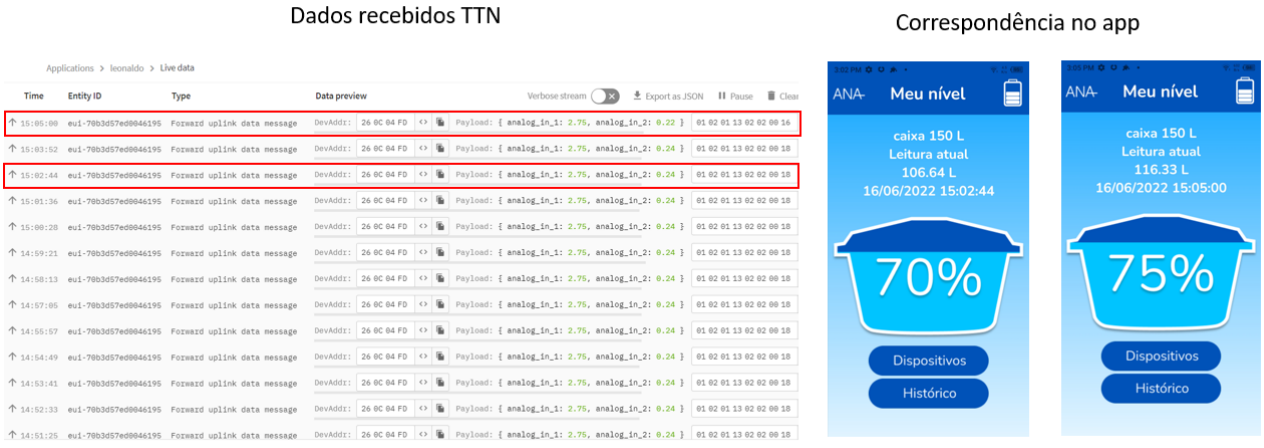
Quanto aos testes de integração entre *hardware* e *software*, foi necessário apenas colocar o dispositivo físico em operação com os dados referentes a esse *end device* estando presentes no *firmware*, condizentes com os dados criados na TTN dentro da aplicacão. Com isso, observou-se os dados reais chegando na TTN e em tempo real sendo enviados ao aplicativo, para que o usuário possa visualizar. Como pode-se observar na Figura 66, o dado *analog_in_1* é igual a 2,76 que corresponde à leitura da bateria no momento em que o pacote foi enviado, já o dado *analog_in_2*, diz respeito à leitura da distância entre o sensor e a superfície da água, que nesse caso era de 0,25 metros.

A Figura 67 mostra, à esquerda os dados recebidos pela TTN, provenientes do *end device* e à direita é possível notar a correspondência de horário e valor em litros e porcentagem. Em um dos momentos foi lido ao equivalente a 70% e, em seguida, a 75%, para uma caixa do fabricante Fortlev de 150 litros que possui altura máxima de 43 cm.

Figura 66 – Dados TTN tempo real

Fonte: (Autoria própria, 2022).

Figura 67 – Dados TTN e App



Fonte: (Autoria própria, 2022).

5 CONCLUSÃO

O presente projeto atendeu satisfatoriamente todos os requisitos esperados, tanto os funcionais quanto os não funcionais. Foram vários desafios a serem superados antes de se chegar ao modelo do protótipo final. No início, o microcontrolador ESP32 LILYGO TTGO sx1276 com rádio LoRa integrado, se mostrou uma boa opção por possuir o rádio integrado evitando mais conexões e trilhas no circuito. No entanto, o módulo apresentava consumo superior ao esperado mesmo em modo *deep sleep*, o que prejudicava a vida útil da bateria. O sistema, inicialmente, utilizando o módulo ESP32 drenava uma bateria recarregável de lítio de 2900 mAh em apenas três dias, pois o consumo era constante e igual a 40 mAh. O desafio então era encontrar um microcontrolador que atendesse ao requisito principal relacionado ao consumo, ser um circuito *ultra low power*, com duração da bateria assegurada por pelo menos dois anos.

Em seguida, foram avaliadas várias opções de microcontroladores da família ESP32 e pôde-se perceber que quanto mais enxuto o módulo, menor é o consumo de operação. O consumo para o ESP32, esperado pelo fabricante Espressif, é coerente com o projeto, de acordo com o *datasheet*.Contudo, quando se adquirem placas ou *kits* de desenvolvimento que possuem mais módulos integrados, o consumo do ESP32 previsto no *datasheet*, não pode ser garantido. Por exemplo, somente um módulo com conector USB já é responsável por consumir 5 mAh do total do microcontrolador, mesmo em estado *deep sleep*.

Por isso, o projeto foi refeito considerando o uso do microcontrolador ESP WROOM 32 que tem o consumo médio de 58 mA. Apenas no momento da transmissão de dados e em modo *deep sleep* tem o consumo do conjunto inteiro reduzido a apenas 32 μAh, o que garante com segurança o requisito de durabilidade da bateria. Por esse motivo, o tipo de alimentação também foi substituído, como esse microcontrolador não contava com um módulo carregador de bateria integrado, resolveu-se então utilizar pilhas comuns, alcalinas do tipo AA, pois o usuário, nesse caso, precisa apenas substituir as pilhas ao invés de recarregá-las, o que poderia ser um desafio, visto o local de instalação das caixas d'água residenciais.

Outro desafio foi acomodar o sensor de modo a não perder a precisão de leitura, uma vez que o sensor começa a medir a partir da distância mínima de 20 cm. Com o sensor preso à tampa da caixa d'água, ele fica a uma altura de aproximadamente 12 cm de altura em relação à altura máxima da caixa. Além disso, a norma ABNT que regulamenta recipientes do tipo caixa d'água, prevê que o nível de água precisa ser 6 cm menor que a altura total da caixa. Nesse caso, com esse posicionamento do sensor, garante-se que o mesmo ficará na distância adequada, uma vez que as medidas iniciam com 20 cm. Ou seja, quando a leitura do sensor receber apenas 20 cm, significa caixa completamente cheia. Como sugestão a trabalhos futuros, orienta-se utilizar um sensor com capacidade de medir a partir de distâncias menores, o sensor ultrassônico comum HC-SR04 por exemplo, pode obter medidas a partir de 4 cm, no entanto não tem proteção contra água. Caso seja possível a implementação de outro sensor, o posici-

onamento do mesmo na caixa pode ser facilitado, poderá desse modo ser instalado próximo à borda, em vez de ser posicionado no ponto mais alto da tampa e fixado através de imãs.

Para o desenvolvimento da aplicação os maiores desafios foram a integração entre TTN com o servidor *backend* e a hospedagem do banco de dados. A TTN possui diferentes formas de integração, dentre elas optou-se por utilizar *weebhooks* customizados, ou seja, ainda que a TTN disponibilize modelos de *webhooks* integrados com outras plataformas, para esse projeto foi construído do zero. Para o servidor de *backend* receber os dados do *webhook* da TTN, a maior dificuldade foi criar um objeto exatamente igual ao enviado pela TTN, ou seja, dizer para o servidor exatamente o que ele receberia, todas as variáveis com o exato nome e tipo, sendo que caso houvesse qualquer diferença, esta acarretaria em um erro no sistema, caracterizando o não funcionamento da integração entre TTN e *backend*. A respeito da integração entre o servidor *backend* e o banco de dados, ocorreu de forma mais simplificada, sem haver mais dificuldades, isso porque a *framework .NET* possui as ferramentas necessárias para tal desenvolvimento e implementação.

Por último, houve a integração do aplicativo com o servidor *backend*, pois o ANAApp exibe ao usuário os dados armazenados no banco de dados, porém esse acesso não ocorre de forma direta. Em outras palavras, o aplicativo não faz o pedido dos dados diretamente ao banco de dados, o intermediário dessa comunicação foi o servidor *backend*, sendo necessário o desenvolvimento de requisições desses dados, de modo que cada requisição fosse única, pois possuía diferentes parâmetros, além de que, realizava diferentes operações e respondia com diferentes dados. Sendo assim, criou-se mais de 10 requisições.

A hospedagem do servidor *backend* foi realizada no provedor de nuvem Azure sem grandes desafios, pois o provedor disponibilizou licença de estudante para hospedagem de servidores e mais US$ 100,00 (R$542,99 cotação 06/07/2022), para uso na plataforma, porém essa licença não contemplava a hospedagem do banco de dados. Tentou-se ainda hospedar o banco de dados na Azure com o valor disponibilizado de US$ 100,00, porém não foi suficiente visto que um servidor SQL tinha um custo mensal de aproximadamente R$ 2000,00 (US$368,33 cotação 06/07/2022). A solução foi encontrar um novo provedor com um custo menor, optou-se pelo provedor ITMNetworks com custo mensal de R$28,99, que foi suficiente para o cumprimento dos requisitos do projeto.

REFERÊNCIAS

ABNT. **NBR 1479: Reservatório poliolefínico para água potável - Requisistos**. [*S.l.*]: Associação Brasileira de Normas Técnicas, 2002.

ABNT. **NBR 15682: Tanque estacionário rotomoldado em polietileno (PE) para acondicionamento de águas - Requisitos e métodos de ensaio**. [*S.l.*]: Associação Brasileira de Normas Técnicas, 2009.

ABNT. **NBR 14799: Reservatório poliolefínico para água potável - Requisistos**. [*S.l.*]: Associação Brasileira de Normas Técnicas, 2011.

BABIUCH, M.; FOLTÝNEK, P.; SMUTNÝ, P. Using the esp32 microcontroller for data processing. *In*: IEEE. **2019 20th International Carpathian Control Conference (ICCC)**. [*S.l.*], 2019. p. 1–6.

BOUGUERA, T. *et al.* Energy consumption model for sensor nodes based on lora and lorawan. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 18, n. 7, p. 2104, 2018.

CEMADEN. **MONITORAMENTO DE SECAS E IMPACTOS NO BRASIL**. 2021. Disponível em: http://www2.cemaden.gov.br/wp-content/uploads/2021/11/Boletim_BRASIL_102021.pdf. Acesso em: 28 jul. 2022.

CUNHA, L. J. da; VALIM, P. R. O. Rede de sensores sem fio para monitoramento de variáveis de ambiente. **Anais do Computer on the Beach**, v. 11, n. 1, p. 007–009, 2020.

DIY I0T. **Guide to Reduce ESP32 Power Consumption by 95%**. Diy I0T, 2020. Disponível em: https://diyi0t.com/reduce-the-esp32-power-consumption/. Acesso em: 3 abr. 2022.

DO BIT AO BYTE. **4 Sleep Modes com ESP32**. Do bit ao Byte, 2021. Disponível em: https://www.dobitaobyte.com.br/4-sleep-modes-com-esp32/. Acesso em: 3 abr. 2022.

ESPRESSIF. **ESP-WROOM-32**. Espressif, 2022. Disponível em: https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32/hw-reference/modules-and-boards.html. Acesso em: 10 abr. 2022.

FARMANI, R.; WALTERS, G. A.; SAVIC, D. A. Trade-off between total cost and reliability for anytown water distribution network. **Journal of water resources planning and management**, v. 131, n. 3, p. 161–171, 2005.

FLIP FLOP. **Como Conectar o Sensor Ultrassônico HC-SR04 ao Arduino**. Flip Flop, 2013. Disponível em: https://www.filipeflop.com/blog/sensor-ultrassonico-hc-sr04-ao-arduino/. Acesso em: 7 abr. 2022.

FOLHA DE LONDRINA. **MAIOR CRISE HÍDRICA DA HISTÓRIA DO PARANÁ LANÇA DESAFIOS EM MÚLTIPLAS FRENTES**. 2021. Disponível em: https://www.folhadelondrina.com.br/reportagem/maior-crise-hidrica-da-historia-do-parana-lanca-desafios-em-multiplas-frentes-3100476e.html. Acesso em: 14 jun. 2022.

GAZETA DO POVO. **Torneira seca a cada 36 horas faz disparar venda de caixas d'água em Curitiba**. 2020. Disponível em: https://www.gazetadopovo.com.br/parana/rodizio-dispara-venda-caixas-de-agua-curitiba/. Acesso em: 20 mar. 2022.

GHOSLYA, S. **All About LoRa and LoRaWAN**. 2017. Disponível em: <http://www.sghoslya.com/. Acesso em: 25 mar. 2022.

GTA UFRJ. **LoRa Arquitetura - Topologia de Rede**. 2019. Disponível em: https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/lora/arquitetura.html. Acesso em: 20 mar. 2022.

HOPERF. **RFM95W LoRa Module**. Hope RF, 2022. Disponível em: https://www.hoperf.com/modules/lora/RFM95.html. Acesso em: 7 abr. 2022.

MARAIS, J. M.; MALEKIAN, R.; ABU-MAHFOUZ, A. M. Lora and lorawan testbeds: A review. **2017 IEEE Africon**, IEEE, p. 1496–1501, 2017.

MARINHO, S. D. A. M. *et al.* Interfaces entre a produção do espaço urbano e o risco de desabastecimento de água. **Engenharia Sanitaria e Ambiental**, SciELO Brasil, v. 26, p. 417–427, 2021.

MCCLELLAND, C. **LPWAN-The Benefits of LPWAN Technology vs. other IoT Connectivity Options**. 2017. Acesso em: 10 jun. 2022.

MERCADO LIVRE. **Nivel Caixa D Agua - Sensor Nivel Caixa D Agua - Cisterna**. 2022. Disponível em: https://produto.mercadolivre.com.br/MLB-819077851-nivel-caixa-d-agua-sensor-nivel-caixa-d-agua-cisterna-_JM. Acesso em: 20 abr. 2022.

MICROSOFT. **Visão geral do .NET Framework**. 2022. Disponível em: https://docs.microsoft.com/pt-br/dotnet/framework/get-started/overview. Acesso em: 8 abr. 2022.

NETO, E. D. S. **Criando end-devices LoRa: arquitetura e especificações**. 2017. Disponível em: https://www.embarcados.com.br/end-devices-lora-arquitetura/. Acesso em: 30 mar. 2022.

OJC. **Seca histórica provoca desabastecimento de água em estados do sul, sudeste e centro-oeste, em um cenário de aumento da área irrigável pela agropecuária tradicional**. 2021. Disponível em: https://www.justicaeco.com.br. Acesso em: 28 jul. 2022.

PLUGA. **O que são webhooks e como usar um agora mesmo, em menos de 2 minutos**. Pluga, 2018. Disponível em: https://pluga.co/blog/webhook/. Acesso em: 10 abr. 2022.

RAI, P.; REHMAN, M. Esp32 based smart surveillance system. *In*: IEEE. **2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)**. [*S.l.*], 2019. p. 1–3.

SEMTECH. **MCU Requirements for LoraWAN™**. 2017.

SENSE SENSORS & INSTRUMENTS. **Sensores Ultrassônicos**. 2014. Disponível em: https://www.sense.com.br/arquivos/produtos/arq1/Sensores_Ultrass\%C3\%B4nicos_Sense_Folheto_Rev_\%20J.pdf. Acesso em: 18 abr. 2022.

SILVA, C. B. C. **A teoria das práticas e o consumo residencial de água: um estudo nos lares paulistanos**. abr 2020. 79 p. Tese (Doutorado) — FUNDAÇÃO GETULIO VARGAS ESCOLA DE ADMINISTRAÇÃO DE EMPRESAS DE SÃO PAULO, SÃO PAULO, abr 2020.

SYSTEMS, E. Esp32 series. **Datasheet**, v. 3.9, n. 1, p. 68, 2022.

TEC-CI. **Corrosão de placas de circuito impresso**. 2022. Disponível em: https://tec-ci.com.br/blog/circuito-impresso/corrosao-circuito-impresso/. Acesso em: 17 jan. 2022.

TECH DESIGN. **Quick Guide to Understand LoRa and LoRa Modules**. Tech Design Blog, 2021. Disponível em: https://blog.techdesign.com/quick-guide-to-understand-lora-and-lora-modules/. Acesso em: 22 mar. 2022.

THINGIVERSE. **SR04T - Ultrasonic distance sensor cone**. 2014. Disponível em: https://www.sense.com.br/arquivos/produtos/arq1/Sensores_Ultrass\%C3\%B4nicos_Sense_Folheto_Rev_\%20J.pdf. Acesso em: 22 abr. 2021.

TTN. **Building a global open LoRaWAN® network**. 2021. Disponível em: https://www.thethingsnetwork.org/. Acesso em: 28 mar. 2022.

USINA INFO. **Sensor Ultrassônico JSN-SR04T**. Usina Info, 2022. Disponível em: https://www.usinainfo.com.br/sensor-ultrassonico/sensor-ultrassonico-jsn-sr04t-a-prova-d-agua-modulo-para-arduino-4704.html. Acesso em: 4 abr. 2022.

APÊNDICE A – Servidor *backend*

A.1 Repositório remoto do servidor backend

O código final do servidor *backend* pode ser encontrado no link: https://github.com/LeonaldoJunior/TCC-ANA-2022-Backend.

APÊNDICE B – Aplicativo ANAApp

B.1 Repositório remoto do aplicativo ANAApp

O código final do aplicativo ANAApp pode ser encontrado no link: https://github.com/LeonaldoJunior/TCC-ANA-2022-App.

APÊNDICE C – *Firmware* do ANADevice

C.1 Repositório remoto do *firmware* ANADevice

O código final do *firmware* ANADevice pode ser encontrado no link: https://github.com/LeonaldoJunior/TCC-ANA-2022-Fw.