UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

FLÁVIA VALÉRIA DE OLIVEIRA CAVALCANTE

SISTEMA DE COLETA DE DADOS DE CONSUMO DE ÁGUA DE UNIDADES HABITACIONAIS EM CONDOMÍNIOS

CAMPO MOURÃO

2023

FLÁVIA VALÉRIA DE OLIVEIRA CAVALCANTE

SISTEMA DE COLETA DE DADOS DE CONSUMO DE ÁGUA DE UNIDADES HABITACIONAIS EM CONDOMÍNIOS

Units water consumption data collection system housing in condominiums

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Eletrônica do Curso de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Eduardo Giometi Bertogna

CAMPO MOURÃO

2023

FLÁVIA VALÉRIA DE OLIVEIRA CAVALCANTE

SISTEMA DE COLETA DE DADOS DE CONSUMO DE ÁGUA DE UNIDADES HABITACIONAIS EM CONDOMÍNIOS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Eletrônica do Curso de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 24/11/2023

Eduardo Giometi Bertogna
Doutor
Universidade Tecnológica Federal do Paraná

Henrique Cunha Carvalho
Doutor
Universidade Tecnológica Federal do Paraná

Fábio Augusto Amaral
Bacharel
Universidade Tecnológica Federal do Paraná

CAMPO MOURÃO

2023

AGRADECIMENTOS

Agradeço aos meus pais por todo o apoio e condições que me deram durante a minha vida, em especial nesta fase da graduação. Sem eles, eu não teria chegado onde cheguei, nem seria a metade da pessoa que sou! Sou grata a UTFPR, por ser essa instituição incrível, que apoia e incentiva o crescimento e desenvolvimento de seus alunos. Deixo o meu muito obrigada também aos professores do departamento de Engenharia Eletrônica por estarem sempre dispostos a compartilhar seus conhecimentos e nos apoiar no que precisar!

RESUMO

A medição do consumo de água na maioria dos condomínios ainda é feita de forma coletiva, com um único hidrômetro que mede o consumo do condomínio inteiro, gerando uma única cobrança de água que é dividida igualmente entre os condôminos de acordo com o critério utilizado pelo condomínio, não levando em conta o consumo individual de cada unidade habitacional e fazendo com que os moradores não fiquem cientes do seu consumo real, além de não pagarem diretamente por ele. A solução para isso é a medição individualizada, onde são instalados hidrômetros no ramal de cada unidade residencial para medir o seu consumo e, assim, racionalizar o uso da água e fazer a cobrança proporcional a este consumo. O objetivo deste trabalho é desenvolver um sistema digital e autônomo, que seja capaz de medir, de forma individualizada, o consumo de água. Nesse sistema, a medição do consumo é feita por um sensor de efeito Hall, controlado por um microcontrolador STM32, que envia, com o auxílio de um Arduino Nano e de um módulo NRF24L01,os dados obtidos do consumo, para um Raspberry Pi. Nele, os dados são armazenados em um banco de dados, conectado a uma rede interna do condomínio, que pode ser acessada através de uma página Web. Apesar do sistema de medição de consumo ainda possuir muito potencial de melhoria e aperfeiçoamento de suas funções, não se pode negar a evolução que o projeto teve e que conseguiu cumprir com o que foi proposto.

Palavras-chave: medição individualizada; sensor de fluxo; nrf24l01+; raspberry pi; stm32f103c8t6.

ABSTRACT

The measurement of water consumption in residential condominiums has been done collectively, with a single water meter that measures the consumption of the entire condominium, generating a single water charge that is divided equally among all residents of the condominium, without taking into account the individual consumption of each residential unit and causing residents not to be aware of their real consumption, in addition to not paying directly for it. The solution to this is individual measurement, where water meters are installed in the branch of each residential unit to measure their consumption and, thus, rationalize the use of water and charge proportionally to this consumption. The objective of this work was to develop a digital and autonomous system, which was capable of individually measuring water consumption. In this system, consumption is measured by a Hall effect sensor, controlled by an STM32 microcontroller, which sends, with the help of an Arduino Nano and an NRF24L01 module, the consumption data obtained to a Raspberry Pi. There, they are stored in a database, connected to an internal network of the condominium, which can be accessed through a web page. Although the consumption measurement system still has a lot of potential for improving and perfecting its functions, it is not can deny the evolution that the project has had so far and that it has managed to fulfill exactly what was proposed.

Keywords: individualized measurement; flow sensor; nrf24l01+; raspberry pi; stm32f103c8t6.

LISTA DE FIGURAS

Figura 1 – Exemplo de hidrômetro Multijato 13
Figura 2 – Mostrador de um hidrômetro digital com ponteiros 13
Figura 3 – Medições Individualizada em condomínio horizontal. 15
Figura 4 – Medições Individualizadas em condomínio vertical. 16
Figura 5 – Sensor de fluxo de água YF-S201. 19
Figura 6 – Sensor de fluxo de água DN20 de Cobre 20
Figura 7 – Módulo STM32F103C8T6 21
Figura 8 – Arduino Nano 22
Figura 9 – Módulos de RF 23
Figura 10 – Raspberry Pi 3 24
Figura 11 – Diagrama de Blocos: Funcionamento Geral do Sistema 26
Figura 12 – Diagrama de blocos: Medição Consumo 27
Figura 13 – Diagrama de blocos: Armazenamento de dados 27
Figura 14 – Diagrama de blocos: Conexão Raspberry e NRF24L01+ 28
Figura 15 – Diagrama de Blocos: Integração RaspberryPi-Banco de dados 29
Figura 16 – Diagrama de blocos: Conexão STM32 e NRF24L01+ 29
Figura 17 – Diagrama de blocos: Funcionamento Geral 31
Figura 18 – Diagrama Detalhado do Sistema 32
Figura 19 – Fluxograma Firmware - Raspberry Pi 3 33
Figura 20 – Fluxograma Firmware - Arduino Nano 34
Figura 21 – Fluxograma Firmware - STM32F103C8T6 35
Figura 22 – Configurações do NRF24L01 36
Figura 23 – Diagrama de Blocos: Funcionamento Geral do Sistema com Arduino Nano 37
Figura 24 – Página Web de Interface Usuário-Dados 38
Figura 25 – Esquemático do Circuito Sensor 39
Figura 26 – Esquemático do Circuito Raspberry Pi 39
Figura 27 – Circuito do Sistema em placa de circuito impresso - Frente 40
Figura 28 – Circuito do Sistema em placa de circuito impresso - Verso 41
Figura 29 – Protótipo do Sistema de Medição 41
Figura 30 – Circuito Raspberry Pi + NRF 42

Figura 31 – Teste do sensor em ambiente simulado 42
Figura 32 – Tela do STM32 - mensagem de depuração 43
Figura 33 – Tela do Arduino - mensagem de depuração 43
Figura 34 – Tela do RaspberryPi - mensagem de depuração 44

SUMÁRIO

1 INTRODUÇÃO 9

1.1 Objetivos 10

1.1.1 Objetivo geral 10

1.1.2 Objetivos específicos 10

1.2 Justificativa 10

2 FUNDAMENTAÇÃO TEÓRICA 12

2.1 Hidrômetro 12

2.2 Medição em condomínios 14

2.3 Medição individualizada 15

2.3.1 Tipos de leituras 17

2.3.1.1 Leitura direta 17

2.3.1.2 Leitura via telemetria 17

2.3.1.3 Geração de pulso e transmissão via cabo 17

2.3.2 Formas de cobrança 17

2.3.3 Lei federal 18

2.4 Sensores de fluxo de água 19

2.4.1 YF-S201 ½" - Plástico 19

2.4.2 DN20 3/4" - COBRE 20

2.5 Microcontrolador STM32 20

2.6 Arduino Nano 22

2.6.1 ARDUINO IDE 22

2.7 Módulo NRF24L01+ 23

2.8 Raspberry Pi 3 24

2.9 Banco de dados SQLite 25

3 METODOLOGIA 26

3.1 Planejamento 26

3.2 Parte 1: medição do consumo de água 26

3.3 Parte 2: armazenamento de dados 27

3.4 Parte 3: conexão Raspberry Pi e NRF24L01+ 28

3.5 Parte 4: integração Raspberry Pi e banco de dados 28

3.6 Parte 5: conexão do STM32 com NRF24L01+ 29
3.7 Parte 6: conectando todas as partes do sistema 30
3.8 Parte 7: testando na aplicação 30
4 RESULTADOS E DISCUSSÕES 31
4.1 Potenciais Melhorias 45
5 CONCLUSÃO 47
REFERÊNCIAS 48
APÊNDICE A CÓDIGO DO FIRMWARE DO STM32F103C8T6 52
APÊNDICE B CÓDIGO DO FIRMWARE DO ARDUINO NANO 69
APÊNDICE C CÓDIGO DO FIRMWARE DO RASPBERRY PI 75
APÊNDICE D CÓDIGO DO BANCO DE DADOS 80
APÊNDICE E CÓDIGO DA PÁGINA WEB (INTERFACE COM O USUÁRIO) PARA CONSULTA AO BANCO DE DADOS 82
APÊNDICE F AMBIENTES DE DESENVOLVIMENTO UTILIZADOS NO PROJETO 90

1 INTRODUÇÃO

A água é um recurso natural renovável determinante para a existência de vida no planeta e essencial para o desenvolvimento socioeconômico das nações. Apesar de renovável, tem se tornado escasso, devido a grande demanda e pelo mau uso, principalmente nos centros urbanos (Oliveira, 1999).

Nos sistemas prediais, por exemplo, são comuns desperdícios de água ocasionados por diversos fatores como vazamentos em tubulações, concepções de projetos inadequadas e, principalmente, negligência dos usuários (Oliveira, 1999).

O método tradicional de medição do consumo de água, utilizado atualmente pelas concessionárias, em sistemas prediais não incentiva a redução do consumo de água e gera o desperdício (Coelho, 2001).

Durante muito tempo a medição de consumo de água em condomínios vem sendo feita de forma coletiva, com um único hidrômetro que mede o consumo do condomínio inteiro. Assim, é gerada uma única cobrança de água que é rateada pelos moradores do condomínio. Esse rateio, seja por fração ideal ou por unidade residencial ou utilizando esses dois modos, previstos no Código Civil, não leva em conta o consumo individual de cada unidade residencial (Lima, 2016; CondominioSC, 2018).

Dessa forma, mesmo que o usuário seja consciente com o seu consumo e busque economizar, isso não se reflete em sua conta de água, logo acaba desmotivando o usuário a continuar economizando, já que este pagará o mesmo valor pela água que outro usuário que não economiza (Coelho, 2001).

A solução para isso é a medição individualizada, que consiste na instalação de hidrômetros no ramal de cada unidade residencial para medir o seu consumo e, assim, racionalizar o uso da água e fazer a cobrança proporcional a este consumo (Coelho, 2004).

Já existem no mercado empresas especializadas neste ramo, que vendem hidrômetros digitais para medição individualizada em condomínios e fornecem serviços de leitura e controle destes gastos. A medição é feita por transmissão de radiofrequência, o que permite que se faça a leitura do consumo dos equipamentos de maneira remota, sem a necessidade de entrada nos apartamentos para acessar os dados de consumo (Techem, 2019).

Com a contratação de uma empresa deste tipo, não há a necessidade de algum funcionário do condomínio ter de operar o sistema, já que a empresa é responsável por todo o processo, desde a instalação, leitura e manutenção. A leitura é realizada mensalmente por um leiturista da empresa (Techem, 2019).

Só que os serviços dessas empresas entram nos valores mensais do condomínio, como despesas ordinárias, e se tornam um gasto adicional.

Além disso, a instalação de hidrômetros individuais requer investimentos altos, em torno de R$ 400 a R$ 700, por unidade. Em prédios antigos, o custo pode chegar a R$ 6 mil por apartamento (Nakamura, 2016).

O ideal seria fazer a medição individualizada de uma forma mais barata e simples, utilizando, por exemplo, um sensor eletrônico de fluxo de água para substituir os hidrômetros individuais. Acoplado a um microcontrolador, que faria a interpretação dos sinais de pulso correspondentes a vazão medida, esse sensor poderia ser lido de forma automática, sem a necessidade de ter um funcionário para fazer isso, e seus dados enviados remotamente, para uma central.

1.1 Objetivos

1.1.1 Objetivo geral

Mostrar que é possível, através de um sistema relativamente simples e autônomo, fazer um monitoramento confiável do consumo individualizado de água, em cada unidade residencial, e armazenar os dados de forma simples e segura, para que possam ser usados no rateio da conta de água. Mostrando, assim, que a medição individualizada em condomínios pode ser uma solução barata e viável para uma cobrança mais justa e um uso mais consciente da água.

1.1.2 Objetivos específicos

Desenvolver um sistema digital e autônomo, que seja capaz de medir, de forma individualizada, o consumo de água em unidades habitacionais (apartamentos ou casas), utilizando:

- Sensor de fluxo de água controlado por microcontrolador;
- Comunicação rádio frequência;
- Armazenamento em banco de dados;
- Interface com o usuário para consulta dos dados.

1.2 Justificativa

O sistema de medição de água tradicionalmente utilizado em condomínios residenciais nem sempre é social e financeiramente justo, pois na maioria dos casos não leva em conta o consumo individual de cada unidade residencial (apartamento ou casa). Todos os gastos do condomínio, residenciais e das áreas comuns, são incorporados em uma única medição e é gerada uma única cobrança de água que é rateada entre os condôminos com base na área da unidade residencial, por unidade residencial, de forma híbrida (misturando essas duas formas citadas anteriormente) ou pelo número de habitantes de cada unidade. Ou seja, não se faz, separadamente, a medição dos gastos de água individuais, e os moradores não ficam cientes

do seu consumo real, além de não pagarem diretamente por ele (Coelho, 2001; Lima, 2016; CondominioSC, 2018).

Este sistema também não leva em conta se há o esforço em economizar água em algumas unidades, enquanto que em outras não. Por isso, esse sistema, em vez de incentivar o consumo consciente de água, acaba gerando o desperdício, já que mesmo que o usuário seja consciente e busque economizar, isso não se reflete em sua conta de água, logo ele acaba se sentindo desmotivado a continuar economizando e injustiçado, por pagar o mesmo valor de quem não economiza (Coelho, 2001).

O sistema sugerido neste trabalho visa acabar com essa injustiça e com o uso irresponsável da água nos condomínios, implementando a Medição Individualizada do consumo de água, de forma prática e barata.

Com ele, além de contas mais justas para os usuários e diminuição do uso da água, resultaria também em uma economia considerável no consumo de energia elétrica, devido à diminuição do tempo de uso de chuveiros, torneiras e aquecedores de água elétricos, e no consumo de gás encanado, no caso de residências que fazem uso do gás para aquecer a água de chuveiros e torneiras. Isso geraria uma redução considerável do valor das contas de energia elétrica e gás, além da diminuição na demanda a ser atendida pelas concessionárias (Coelho, 2001; Lima, 2016).

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Hidrômetro

O hidrômetro é um aparelho de precisão que mede e registra o consumo de água em residências e estabelecimentos comerciais (Consultores, 2018; Minucci; Lima, 2013).

O hidrômetro faz a medição constante do volume de água que passa por uma tubulação, para poder quantificar o volume de água que é distribuído para o estabelecimento e, consequentemente, o volume de água consumido por este, para que o volume distribuído possa ser tarifado (Passos; Quadros; Aveiro, 2015).

Os hidrômetros são divididos, segundo o princípio de funcionamento, em dois principais tipos: os Volumétricos e os Taquimétricos (SAMAE, 2019).

Os Volumétricos possuem câmaras de volume (êmbolo), de capacidade conhecida, em seu interior que se enchem e se esvaziam conforme a água passa, transportando um determinado volume de água para a saída do medidor. Esse transporte ocorre devido a uma diferença de pressão antes e depois do hidrômetro, que provoca o movimento giratório do êmbolo. Este movimento é utilizado pelo mecanismo de medição para mensurar o consumo de água. Por possuírem um custo elevado e serem muito sensíveis a impurezas da água que possam se alojar em suas câmaras, eles não são utilizados no Brasil na medição convencional de água (SAMAE, 2019; Minucci; Lima, 2013).

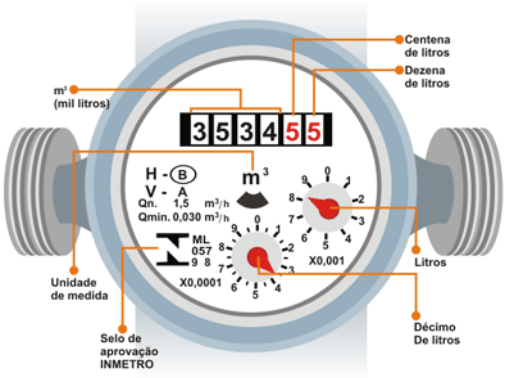
Os Taquimétricos são classificados em Monojato e Multijato. Nos Monojato, a turbina ou hélice, dentro dele, será aciona pela velocidade de um único jato de água, que passa por ele, que será medido. Já nos Multijato, como visto na Figura 1, a turbina é acionada por diversos jatos que passam sobre ela. A capacidade de vazão desses hidrômetros varia a vazão nominal entre 0, 5 e 25 m^3/h. O hidrômetro mais tradicional no Brasil é o Multijato (Passos; Quadros; Aveiro, 2015 apud ABNT, 2007).

Figura 1 – Exemplo de hidrômetro Multijato

Fonte: Hidrogerais (2019).

A medição do volume de água gasto no estabelecimento é feita através da leitura mensal do mostrador do hidrômetro. Os hidrômetros podem possuir mostradores de consumo de diferentes tipos, digital, digital com ponteiros ou de ponteiros, mas os hidrômetros multijato costumam ter mostradores do tipo digital com ponteiros (Minucci; Lima, 2013).

Figura 2 – Mostrador de um hidrômetro digital com ponteiros



Fonte: Consultores (2018).

O mostrador do hidrômetro da Figura 2 é do tipo digital com ponteiros. A leitura deste tipo é feita de forma a se ler somente os 4 primeiros números, da esquerda para a direita, que aparecem em preto nos quadrados, referentes aos m³ (Minucci; Lima, 2013).

A leitura dos hidrômetros, efetuada pelas concessionárias de água, é totalmente visual e realizada por um funcionário da concessionária, o “leiturista”, que fica responsável por percorrer todos os estabelecimentos consumidores de água, registrando os seus respectivos consumos (Minucci; Lima, 2013).

2.2 Medição em condomínios

Durante muito tempo a medição de consumo de água em condomínios vem sendo feita de forma coletiva, ou seja, um único hidrômetro geral é instalado na entrada da tubulação de água da concessionária no edifício, que medirá o abastecimento que suprirá todas as unidades do condomínio, desde as residências até as áreas comuns. Assim, é gerada uma única cobrança de água que é dividida entre os condôminos seguindo uma regra pré-estabelecida em convenção, no condomínio (Lima, 2016; CondominioSC, 2018).

Essa divisão ou rateio da cobrança de água é geralmente feita conforme sugerido pelo Código Civil, podendo ser feita por fração ideal, onde se faz o cálculo com base na área da unidade residencial e os donos de unidades maiores recebem uma parcela maior do rateio que os outros, por unidade, onde a cobrança é dividida igualmente pelo número de unidades residenciais, ou de forma híbrida, misturando essas duas formas citadas anteriormente a critério de cada condomínio. Uma outra forma de rateio seria ainda pelo número de habitantes de cada unidade residencial, mas é inviável para o condomínio manter um controle sobre isto (CondominioSC, 2018).

Como se pode perceber, esse método de medição acaba não levando em conta o consumo individual de cada unidade. Os gastos de água individuais são simplesmente incorporados à taxa de condomínio e os moradores não ficam cientes do seu consumo real, além de não pagarem diretamente por ele (Lima, 2016).

Segundo Coelho e MAYNARD (1999), as principais desvantagens desse sistema de medição coletiva (conhecida também como medição global) são:

- Tirar o direito dos usuários de pagarem uma conta proporcional ao seu consumo;
- Não levar em conta o número de pessoas que moram em cada unidade ao fazer o rateio da conta;
- Injustiça com os usuários que economizam água, pois pagam o mesmo valor dos que esbanjam;
- Se algum usuário ficar inadimplente (não pagar a taxa de condomínio), os que pagaram devidamente também correm o risco de terem a ligação de água cortada.

Desse modo, pode-se afirmar que esse sistema de medição tradicional é injusto socialmente, além de não incentivar a redução do consumo de água e gerar o desperdício, já que mesmo que o usuário seja consciente com o seu consumo e busque economizar, isso não se reflete em sua conta de água, logo ele acaba se sentindo desmotivado a continuar economizando (Coelho, 2001).

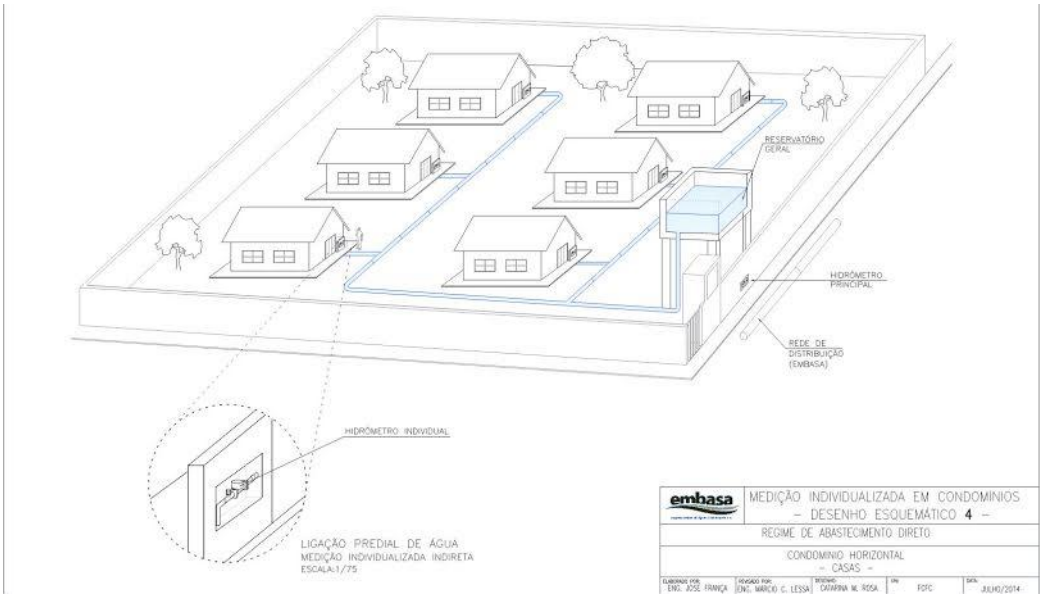
2.3 Medição individualizada

A medição individualizada consiste na instalação de hidrômetros no ramal de cada unidade residencial, de forma que todo o seu consumo seja medido, para que se possa racionalizar o uso da água e fazer uma cobrança proporcional ao consumo individual (Coelho, 2004).

Este tipo de medição considera o consumo de cada unidade separadamente, fazendo com que cada usuário pague o que consumiu mais a sua parte no gasto de água da área comum do condomínio (Lima, 2016).

Pode-se observar nas Figuras 3 e 4 os pontos da instalação de água onde são feitas as medições individualizadas, em condomínios horizontais e verticais.

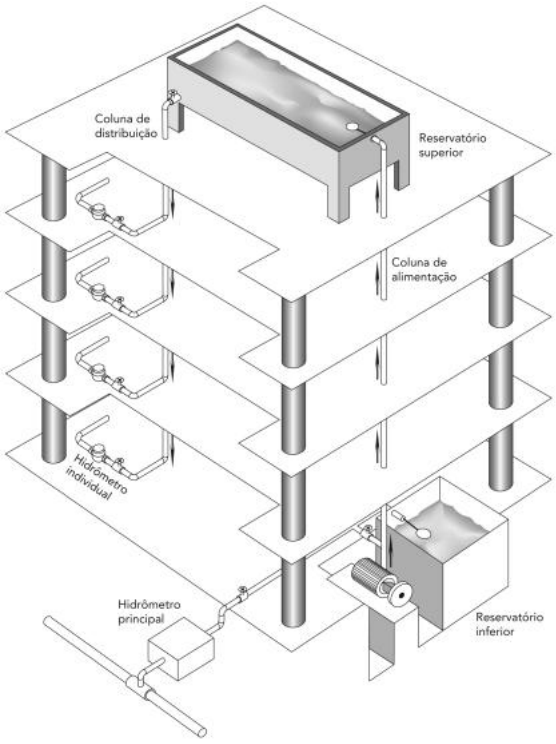
Figura 3 – Medições Individualizada em condomínio horizontal.



Fonte: Embasa (2019).

A medição individualizada é uma metodologia muito importante para a redução do desperdício domiciliar em condomínios, destinada a induzir o usuário a ter uma postura de uso racional da água, pois, como as ações de consumo dos usuários passam a influenciar diretamente sobre os gastos deles, eles passam a ter um incentivo maior ao uso mais consciente e racional da água (Coelho, 2001; Lima, 2016).

Figura 4 – Medições Individualizadas em condomínio vertical.



Fonte: Carvalho Junior (2013).

Outra vantagem da medição individualizada é a redução do consumo de energia elétrica, pela diminuição do volume de água que tem que ser bombeado para os reservatórios (superiores ou inferiores) e pela diminuição do tempo gasto no banho com chuveiro elétrico. Além de gerar uma identificação melhor de vazamentos ou problemas em geral no sistema hidráulico-sanitário do condomínio e consequentemente melhor manutenção (Coelho, 2001; Lima, 2016).

Apesar disto, em casos de condomínios verticais antigos a medição individualizada pode não ser viável. Isso ocorre quando o número de ramais de cada unidade é muito grande e as prumadas são independentes, ou seja, a alimentação de cada cômodo, em um mesmo apartamento, é feita por sistemas distintos partindo da caixa d'água do edifício, logo não existe um sistema de abastecimento único para cada unidade e isolado do resto do prédio. Nesses casos, será necessário que o condomínio contrate uma empresa especializada para avaliar o custo de readequação das instalações, necessária para receber a medição individualizada, e verificar sua viabilidade (Nakamura, 2016).

Em condomínios novos a individualização é mais simples, pois os condomínios novos já estão sendo projetados para a isto, só dependendo da colocação dos equipamentos (Nakamura, 2016).

2.3.1 Tipos de leituras

2.3.1.1 Leitura direta

A leitura é feita visualmente por uma pessoa e registrada em um terminal portátil (Coelho; MAYNARD, 1999).

2.3.1.2 Leitura via telemetria

A Telemetria, ou medição remota, baseia-se na ideia de transmitir e receber dados através de comunicação sem fio. Ela possibilita a leitura de diversos medidores em um curto período de tempo, sem a necessidade de ter uma pessoa percorrendo as áreas comuns do condomínio para realizar as leituras (Minucci; Lima, 2013; Lima, 2016).

Uma das formas mais usadas para fazer a leitura remota é utilizar transmissão via radiofrequência. Nesse caso, um circuito acoplado ao hidrômetro, faz a "leitura" do consumo de água, utilizando sensores, e transmite os dados desse consumo, por meio de sinais de radiofrequência, a uma central dentro do condomínio, que é responsável pelo recebimento e armazenamento das informações obtidas, ou seja, os dados de consumo de todos os hidrômetros do condômino. É possível também utilizar um *software* para processar os dados da central e enviar para a concessionária ou ao computador da administração do condomínio, para que a cobrança possa ser feita (Lima, 2016).

2.3.1.3 Geração de pulso e transmissão via cabo

Neste caso, o controle é feito através de um hidrômetro com saída pulsada onde, a cada litro de água consumido, o sistema envia um impulso elétrico a central. A transmissão é feita via cabo, o que permite maiores distâncias entre os medidores e a central, quando comparado com o sistema via radiofrequência (Lima, 2016).

2.3.2 Formas de cobrança

Segundo Coelho e MAYNARD (1999, p.149), com a medição individualizada, a cobrança pela concessionária pode ser feita de duas formas:

- Utilizando os hidrômetros individuais apenas para rateio do condomínio;
- A concessionária adapta o seu sistema de faturamento de forma a emitir contas de água individualizadas para cada unidade (residência) dentro do condomínio.

Esta última tem a vantagem de que cada unidade fica encarregada de efetuar o pagamento de seu consumo junto à concessionária, sendo responsável pelo eventual corte que possa ter caso fique inadimplente, mas também não é afetado pela inadimplência de outros usuários. A desvantagem é que a medição do consumo da área comum vai continuar sendo feita separadamente nas duas formas e rateada entre os usuários, só que, nesta última, a concessionária terá um trabalho maior na hora de realizar as cobranças, o que não é muito viável para ela (Coelho; MAYNARD, 1999).

Do ponto de vista dos administradores do condomínio, o sistema de medição individualizada é uma ferramenta fundamental para uma gestão mais simples e justa, pois além de induzirem os usuários ao uso consciente da água, ainda resolvem parcialmente a inadimplência nas taxas de condomínio (LIMA, 2016).

Do ponto de vista das concessionárias esse sistema normalmente resulta na diminuição da inadimplência e, devido à redução de consumo, na maior capacidade do sistema de abastecimento de água em suprir a demanda (Lima, 2016).

2.3.3 Lei federal

Há alguns anos, alguns estados brasileiros já vinham criando leis estaduais que obrigassem as construtoras civis a construírem os seus condomínios já com o sistema de medições individualizada inserido, visando à economia de água. Porém, o número de estados, com essa preocupação com o consumo excessivo da água, ainda é a minoria no país. A novidade é que, em 12 de julho de 2021, entrou em vigor uma nova lei que deve começar a mudar esse quadro.

Aprovada em julho de 2016, a Lei Federal 13.312 determina que o uso de medidores individuais de água seja obrigatório em todos os imóveis entregues a partir de 2021 (Brasil, 2016).

Segundo Brasil (2016, p.1), a Lei Federal 13.312 diz:

> Art. 1º Esta Lei torna obrigatória a medição individualizada do consumo hídrico nas novas edificações condominiais.
>
> Art. 2º O art. 29 da Lei nº 11.445, de 5 de janeiro de 2007, passa a vigorar acrescido do seguinte § 3º: [...]
>
> [. . .] §3º As novas edificações condominiais adotarão padrões de sustentabilidade ambiental que incluam, entre outros procedimentos, a medição individualizada do consumo hídrico por unidade imobiliária."
>
> Art. 3º Esta Lei entra em vigor após decorridos cinco anos de sua publicação oficial.

Com isso, as construtoras passam a ser obrigadas adotarem a medição individualizada em seus novos empreendimento e se adaptarem a nova realidade.

2.4 Sensores de fluxo de água

2.4.1 YF-S201 ½” - Plástico

O sensor de fluxo/vazão de água YF-S201, mostrado na Figura 5, consiste em um corpo de válvula de plástico, um rotor de água e um sensor de efeito *hall*. Ele funciona de modo que quando a água flui através do rotor, dentro do sensor, o rotor gira. Sua velocidade muda com taxas diferentes de fluxo. O sensor de efeito *hall*, que está em uma parte onde não há contato com a água, produz o sinal de pulso correspondente a quantidade de voltas que o rotor dá (SEA, 2023).

Figura 5 – Sensor de fluxo de água YF-S201.

Fonte: Digital (2018).

Este sensor possui um diâmetro de $1/2$” e opera numa faixa de vazão de 1 a 30 L/min, permitindo uma pressão $\leq 1{,}75$ MPa, e com um erro de $\pm 10\%$. Com uma corrente máxima de trabalho de 15mA, ele possui três fios para a conexão: Vermelho - Tensão de alimentação (DC 4,5 V a 24 V), preto - GND (0 V) e amarelo - Sinal de saída (onda quadrada) (SEA, 2023).

Sua saída fornece pulsos de acordo com o fluxo da água, onde para cada pulso corresponde a aproximadamente 2,25 mL. A fórmula básica para o cálculo do fluxo é:

$$Fluxo(L/min) = \frac{FrequenciaDosPulsos(Hz)}{7{,}5}$$

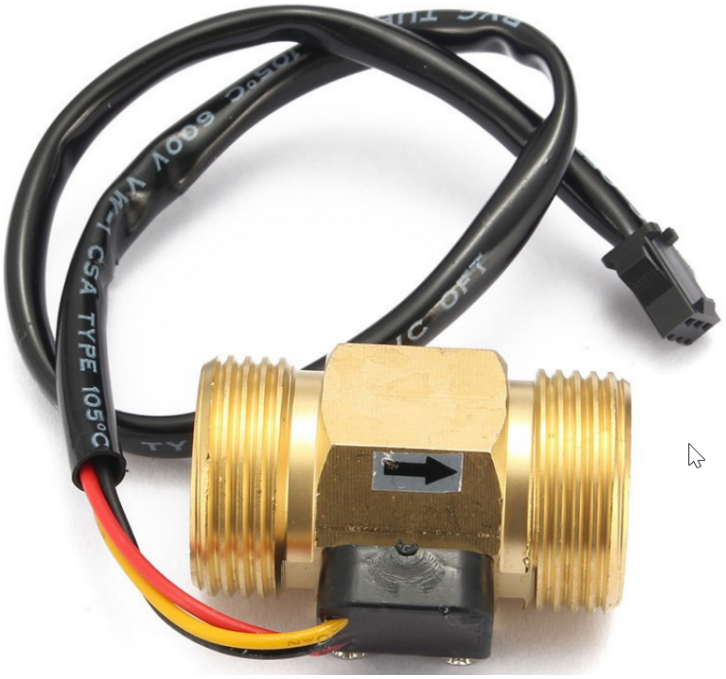
(SEA, 2023).

2.4.2 DN20 3/4” - COBRE

Este sensor, mostrado na Figura 6, opera seguindo o mesmo princípio do anterior, possuindo um sistema integrado, sensor de efeito *hall*, que gera um pulso elétrico com cada revolução. Só que é feito de material de cobre de alta qualidade, resistente e durável, podendo ser usado inclusive em sistemas de aquecimento de água, por aguentar altas temperaturas da água (De -25 ºC a 80 ºC).

Possui um diâmetro de 3/4” e opera numa faixa de vazão de 2 a 45 L/min, permitindo uma pressão $\leq 1{,}75$ MPa. Sua tensão de alimentação é de DC 4,5 V a 18 V, com uma corrente máxima de trabalho de 10 mA (Haihuilai, 2023).

Figura 6 – Sensor de fluxo de água DN20 de Cobre



Fonte: Haihuilai (2023).

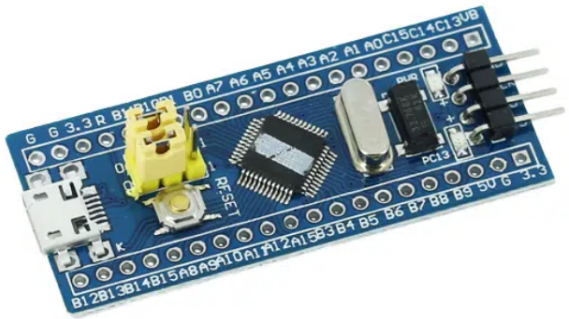
Estes são exemplos de sensores eletrônicos de fluxo de água que podem ser usados em sistemas de medição individualizada de consumo de água, substituindo os hidrômetros individuais que seriam instalados em cada apartamento.

2.5 Microcontrolador STM32

Um microcontrolador é um Circuito Integrado (CI) que tem a capacidade de efetuar processos lógicos com extrema rapidez e precisão. Ele funciona como o “cérebro” do sistema e sua grande vantagem é a sua possibilidade de ser programado para se executar a tarefa desejada, o que o torna adaptável, e que possibilita seu ajuste de acordo com a tarefa que deverá executar (Souza, 2000).

A família de microcontroladores STM32F1, criada pela STMicroelectronics, é baseada no processador ARM Cortex-M3 e oferece uma estrutura para criação de uma grande variedade de sistemas embarcados, desde simples até sistemas complexos de tempo real. Esta família de componentes inclui dezenas de configurações diferentes, provendo amplas opções de tamanho de memória, periféricos, performance e consumo de energia. Os componentes são baratos e justificam seu uso em aplicações de baixo volume de produção. Na realidade, os componentes são comparáveis em valores financeiros aos componentes da família AVR, usados nas placas de desenvolvimentos Arduino, e que ainda apresentam melhoras significativas de desempenho e periféricos mais potentes. Além disso, os periféricos utilizados são compartilhados por outros membros da família (por exemplo, os módulos USART são compatíveis com todos componentes da STM32F1) e são suportados por um único *firmware* (Brown, 2012).

Figura 7 – Módulo STM32F103C8T6



Fonte: Flop (2019).

A Figura 7 mostra uma Placa de Desenvolvimento Blue Pill (STM32F103C8T6). Ela é uma placa com o microcontrolador STM32 que possui um ótimo poder de processamento. Além do processador ARM Cortex-M3, tem uma frequência de até 72 MHz, o que a torna uma placa com alto desempenho. Utiliza-se um Módulo Gravador (ST-LINK) compatível com a placa STM32 e um *Software* de Desenvolvimento para gravá-la e depurar o programa instalado nela (Flop, 2019).

Algumas das Especificações da Placa de desenvolvimento STM32F103C8T6 são (Flop, 2019):

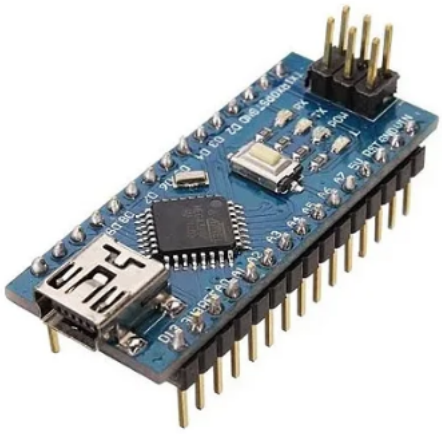
- Núcleo ARM Cortex-M3
- Memória flash: 64 k
- SRAM: 20 k
- Tensão de operação: 5 V (USB) ou 3,3V (pino VCC)
- Sleep, Stop e Standby Mode
- 37 pinos GPIO

- Consumo de corrente: 300 mA
- Conexão micro USB

2.6 Arduino Nano

O Arduino Nano, visto na Figura 8, é uma placa de desenvolvimento, integrada pelo microcontrolador ATmega328p, um conversor USB- UART (Tx e Rx) e componentes de alimentação e temporização. É uma das versões mais compactas dentre as placas da série original da Arduino, sendo ideal para conexão direta com protoboards e placas de circuito impresso (Eletrogate, 2023).

Figura 8 – Arduino Nano



Fonte: Eletrogate (2023).

Ele é compatível com uma infinidade de componentes, sensores e módulos presentes no mercado. Além disso, conta com conversor integrado que permite se comunicar com o computador diretamente pela porta USB. São programadas em linguagem C/C++ com alguns comandos próprios e seu ambiente de desenvolvimento, o Arduino IDE, já conta com todas as ferramentas e recursos necessários para a programação (Eletrogate, 2023).

2.6.1 ARDUINO IDE

É um ambiente de desenvolvimento pensado para que o usuário tenha tudo o que precisa para programar sua placa baseada nessa plataforma, escrevendo seus códigos satisfatoriamente, de forma rápida e eficiente (Quintino, 2023).

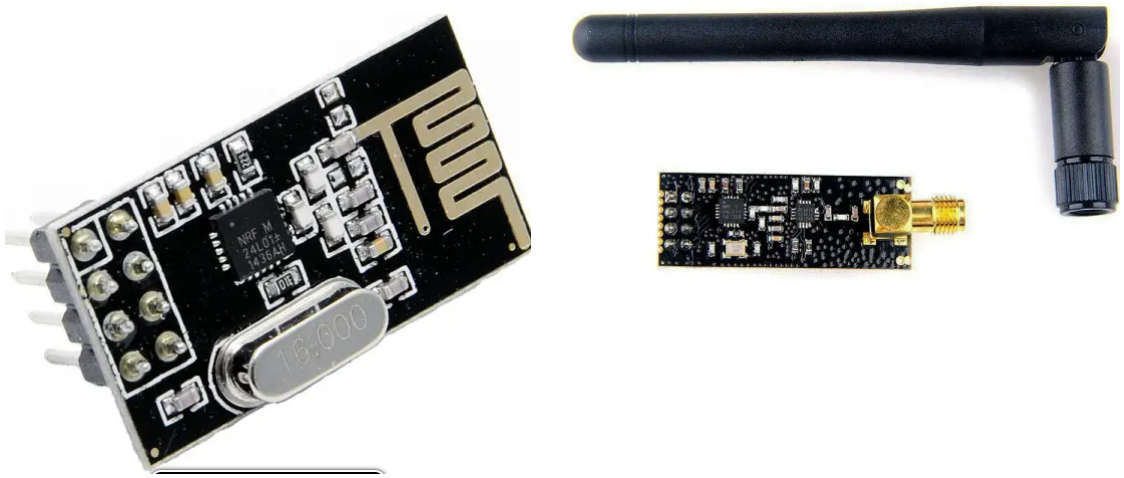
Além de ter um *layout* bastante completo e de fácil navegação, o Arduino IDE ainda permite a gravação de outras placas de desenvolvimento que não sejam da família Arduino, como o STM32 por exemplo.

2.7 Módulo NRF24L01+

O módulo de rede sem fio NRF24L01 é uma ótima opção de comunicação *Wirelless* entre dispositivos como Arduino, PIC, Raspberry, Beagle Bone, dentre outros. Seu alcance pode chegar a 1km em campo aberto. Ideal para quem deseja implementar um sistema de comunicação sem fio com baixo custo, enviando e recebendo informações à distância de sensores, microcontroladores, máquinas e equipamentos eletrônicos em geral (Eletru's, 2019).

Figura 9 – Módulos de RF

(a) NRF24L01+ **(b) NRF24L01+ PA + LNA + antena**



Fonte: Embetronicx (2022).

Na Figura 9a pode-se ver o módulo NRF24L01+ e na Figura 9b o mesmo circuito, só que numa versão com a inclusão de amplificadores de potência (PA) e amplificadores de baixo ruído (LNA), o que melhora significativamente o alcance e a capacidade de penetração do sinal de RF.

Este módulo, mostrado na Figura 9b, acompanha uma antena de 2,4 G (2 dB), com taxa de transmissão de 250 Kbps ao ar livre, a distância de comunicação pode chegar até 1 km (Nordic, 2019).

O módulo NRF24L01+ foi projetado para operar na faixa de frequência Industrial Scientific and Medical (ISM) mundial em 2,400 – 2,4835 GHz. Para projetar um sistema de rádio com ele, precisa-se somente de um microcontrolador e alguns componentes passivos externos. Pode-se operar e configurar esse módulo através de uma Interface Serial Periférica (SPI), pois seu mapa de registros, acessível através do SPI, contém todos os registros de configuração no módulo e é acessível em todos os modos de operação do chip (Nordic, 2019).

O mecanismo de protocolo de banda básica incorporado é baseado em comunicação de pacote e suporta vários modos de operação manual para operação avançada de protocolo autônomo. As FIFOs(*First In First Out*) garantem um fluxo de dados suave entre o *frontend* de rádio e o MCU (*Micro Controller Unit*) do sistema. O *frontend* de rádio usa a modulação GFSK(*Gaussian Frequency Shift Keying*). Ele tem parâmetros configuráveis pelo usuário como canal

de frequência, potência de saída e taxa de dados. O módulo suporta uma taxa de dados de 250 kbps, 1 Mbps e 2Mbps. A alta taxa de dados combinada com dois modos de economia de energia tornam o módulo muito adequado para projetos de baixo consumo de energia (Nordic, 2019).

2.8 Raspberry Pi 3

Diferentemente das placas apresentadas anteriormente, o Raspberry Pi, que pode ser visto na Figura 10, não é um microcontrolador mas é um computador completo em uma única placa. Ele conta com os principais membros de um computador de mesa, como memórias, SSD, processador e chips de interface, sendo assim capaz de armazenar e processar um sistema operacional, executando tarefas que estão muito além da capacidade dos microcontroladores (Blog_Eletrogate, 2022).

Figura 10 – Raspberry Pi 3

Fonte: Blog_Eletrogate (2022).

Algumas especificações do Raspberry Pi são:

- CPU 64 bits com quatro núcleos;
- 1 GB de memária RAM embutida;
- Bluetooth e LAN wireless embutidos;
- 40 pinos GPIO;
- 4 portas USB e 1 HDMI;
- Entrada para câmera, display e cartão SD (Blog_Eletrogate, 2022).

Apesar de aceitar outros sistemas operacionais, o Raspberry Pi foi projetado para o Linux, usando uma versão do Debian: o Raspbian (Donat, 2019).

Uma característica poderosa do Raspberry Pi é a linha de pinos GPIO. Qualquer um dos pinos GPIO pode ser designado (em *software*) como um pino de entrada ou saída e usado

para uma ampla gama de propósitos. Além disso, dois pinos de 5V e dois pinos de 3,3V estão presentes na placa, bem como um número de pinos de GND (0V) (RaspberryPi, 2023).

Além de dispositivos simples de entrada e saída, os pinos GPIO podem ser usados com uma variedade de funções alternativas, como por exemplo:

- SPI: MOSI (GPIO10 e 20); MISO (GPIO9 e 19); SCLK (GPIO11 e 21); CE0 (GPIO8 e 18), CE1 (GPIO7 e 17)
- I2C: Dados: (GPIO2); Relógio (GPIO3); Dados da EEPROM: (GPIO0); Relógio EEPROM (GPIO1)
- Serial: TX (GPIO14); RX (GPIO15) (RaspberryPi, 2023).

O Raspberry Pi tem a vantagem de já vir com bibliotecas pré-instaladas que permitem acessar os pinos usando Python, C ou C++, além de bibliotecas adicionais disponíveis e um fórum ativo de desenvolvedores (Donat, 2019).

2.9 Banco de dados SQLite

Um banco de dados é uma coleção organizada de informações (ou dados) estruturadas armazenadas eletronicamente em um sistema de computador. Os dados normalmente são modelados em linhas e colunas em uma série de tabelas para tornar o processamento e a consulta de dados mais eficientes (Oracle, 2023).

O SQL (*Structured Query Language*) é a linguagem de programação usada por quase todos os bancos de dados para consultar, manipular e definir dados e fornecer controle de acesso a eles (Oracle, 2023).

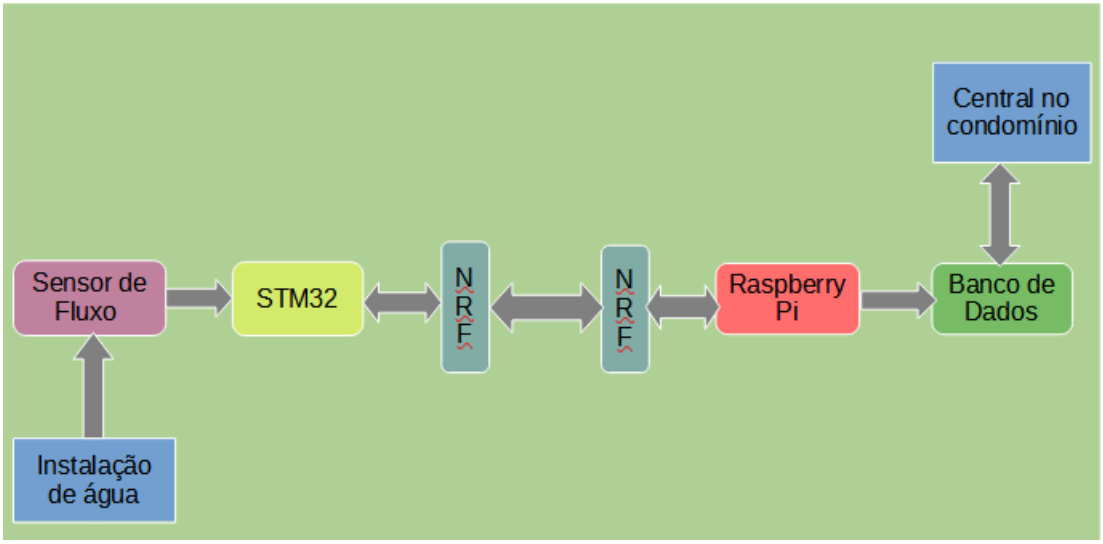
SQLite é uma biblioteca de software que implementa um mecanismo de banco de dados SQL autônomo. Ele possui algumas características que o tornam uma boa ferramenta para uso, tais como:

- Não requer um processo de servidor ou sistema separado para operar (sem servidor),
- É muito pequeno e leve,
- Oferece suporte à maioria dos recursos de linguagem de consulta encontrados no padrão SQL92 (SQL2),
- Está disponível em UNIX (Linux, Mac OS-X, Android, iOS) e Windows (Win32, WinCE, WinRT) (TutorialSpoint, 2023).

3 METODOLOGIA

Para uma melhor organização, primeiramente será feito o planejamento do trabalho e, pensando nele como um todo e que para cada unidade residencial, o sistema proposto deve ter uma estrutura como a vista na Figura 11.

Figura 11 – Diagrama de Blocos: Funcionamento Geral do Sistema



Fonte: Autoria própria (2023).

Em seguida, será feita a divisão do mesmo em partes funcionais, de forma a isolar a resolução dos pequenos “problemas”, diminuindo a complexidade total do trabalho, e permitir, através da soma dessas soluções, que alcance a uma melhor solução geral.

3.1 Planejamento

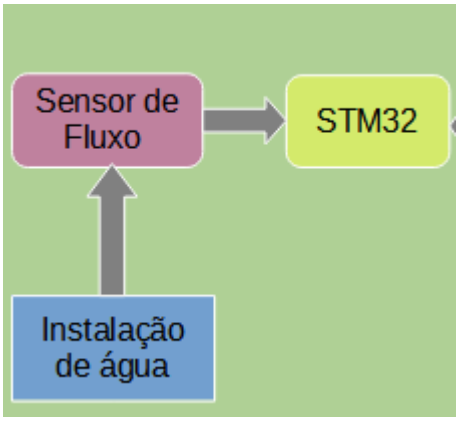
Inicialmente serão definidos todos os componentes necessários a confecção do projeto, inclusive os componentes mais simples, como resistores e capacitores, e ferramentas que serão utilizados na abordagem dos problemas e na montagem do circuito.

Em seguida, serão comprados todos os componentes e eventuais materiais necessários para a confecção do projeto.

3.2 Parte 1: medição do consumo de água

Nesta etapa, será desenvolvida a programação, em linguagem C, de um microcontrolador, contido na Placa de Desenvolvimento BluePill STM32F103C8T6, utilizando a interface de desenvolvimento Arduino IDE. Esse microcontrolador será programado para ler os sinais gerados pelo sensor de fluxo, referente ao consumo de água, interpretá-los de forma correta e armazená-los provisoriamente, como vemos na Figura 12, abaixo.

Figura 12 – Diagrama de blocos: Medição Consumo



Fonte: Autoria própria (2023).

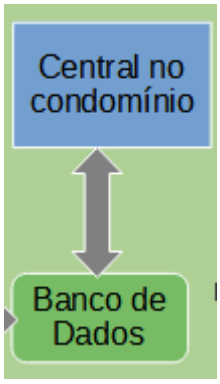
Para desenvolver esta etapa, será necessário primeiramente fazer uma ampla pesquisa sobre o funcionamento do sensor de vazão, sua conexão com o STM32, a interpretação dos dados pelo STM32, entre outros.

Conforme for sendo desenvolvido o *firmware*, serão feitos pequenos testes em matriz de contato (*protoboard*), com auxílio de equipamentos com multímetro e osciloscópio, visualizando assim o progresso da solução.

3.3 Parte 2: armazenamento de dados

Nesta etapa será feita a criação do banco de dados, como podemos ver na Figura 13, assim como a criação dos métodos de inserção, consulta e controle de dados. O banco de dados deverá conter tabelas com os dados das unidades residenciais e seus respectivos consumos de água, além de dados de data e hora da medição de consumo.

Figura 13 – Diagrama de blocos: Armazenamento de dados



Fonte: Autoria própria (2023).

O banco de dados será criado e movimentado usando a linguagem padrão SQL, através da interface do próprio Raspberry Pi, usando a biblioteca SQLite.

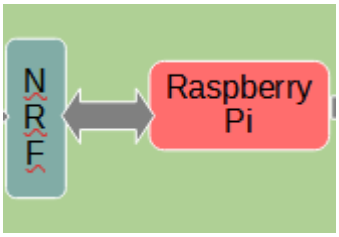
Após pesquisa detalhada sobre os detalhes do funcionamento e trabalho com bancos de dados e SQL, será desenvolvida a programação e testes do mesmo.

Será desenvolvida também a interface onde o usuário da Central, poderá acessar os dados de consumo dos apartamentos. Essa interface também será programada através do Raspberry Pi, só que em linguagem Python e HTML, pois elas tornam possível a criação de uma página web, hospedada na rede interna do condomínio, que fará a interligação entre o usuário e o banco de dados.

3.4 Parte 3: conexão Raspberry Pi e NRF24L01+

O objetivo desta etapa é desenvolver o *firmware* que fará a conexão entre o Raspberry Pi e o módulo de rádio frequência NRF24L01+, como vemos na Figura 14, pois este módulo será responsável depois por interligar a comunicação entre o Raspberry Pi e o STM.

Figura 14 – Diagrama de blocos: Conexão Raspberry e NRF24L01+



Fonte: Autoria própria (2023).

Além de uma pesquisa detalhada sobre comunicação RF e modo de funcionamento e configuração do módulo NRF, deverá ser desenvolvida a programação no Raspberry Pi, em linguagem Python3, utilizando um editor simples de texto como interface e compilando o código direto na linha de comando do Raspberry Pi.

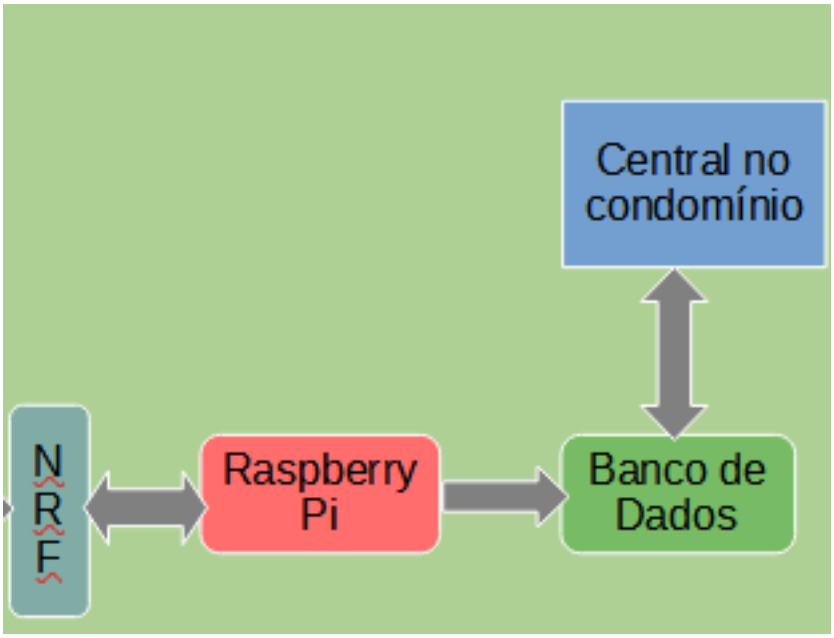
Deve-se ficar atento à complexidade de se fazer uma comunicação via rádio frequência, devido a inúmeros fatores que podem atrapalhar essa conexão, com ruído externo, interferência eletromagnética, endereço e canal de comunicação, faixa de frequência, entre outros.

Com o *firmware* feito, deverá conectar fisicamente o Raspberry Pi e o módulo NRF, numa matriz de contato, testar se sua conexão está funcionando e, após isso, testar se através do módulo o Raspberry Pi consegue manter uma comunicação sem fio com um microcontrolador, como um Arduino Nano, por exemplo.

3.5 Parte 4: integração Raspberry Pi e banco de dados

Aqui deve ser desenvolvido o *firmware* da integração entre o Raspberry Pi e o banco de dados. Isso significa, basicamente, juntar o resultado das partes 2 e 3 e garantir que a integração aconteça corretamente. A Figura 15 representa a junção dessas partes.

Figura 15 – Diagrama de Blocos: Integração RaspberryPi-Banco de dados



Fonte: Autoria própria (2023).

Após o desenvolvimento do *firmware*, deve-se testar o código direto no *hardware*, com o RaspberryPi ligado ao módulo NRF e acessando e modificando internamente o banco de dados.

3.6 Parte 5: conexão do STM32 com NRF24L01+

Nesta parte, será feito o desenvolvimento do *firmware* que fará a comunicação entre o STM32 e o NRF, como mostra a Figura 16. Permitindo assim, que o STM32 possa enviar os dados medidos de consumo para o Raspberry Pi armazenar na Central.

Figura 16 – Diagrama de blocos: Conexão STM32 e NRF24L01+

Fonte: Autoria própria (2023).

Aqui, deve-se primeiramente pesquisar como o STM32 implementa a comunicação via SPI, visto que o NRF só utiliza esta interface para se comunicar. Deve-se pesquisar também quais bibliotecas estão disponíveis para este uso, bem como a documentação das mesmas, e seus exemplos de aplicação.

Após isso, será desenvolvida a programação em linguagem C, utilizando o Arduino IDE e a HAL STM32duino, para permitir o suporte ao STM32 na plataforma.

Conforme for avançando no desenvolvimento do código, deve-se ir testando no *hardware* e verificando se seu funcionamento está de acordo com o esperado.

3.7 Parte 6: conectando todas as partes do sistema

Nesta parte será feita a conexão de todas as partes anteriores, considerando que cada uma teve a melhor solução encontrada e está funcionando corretamente, interligando os *firmwares* e *hardwares* e resolvendo eventuais problemas que possam acontecer nesta junção.

Após isso, todos os componentes do sistema serão montados e conectados numa matriz de contato, simulando o sistema de forma simples e serão testados para garantir que estão funcionando corretamente juntos.

Esses testes deverão ser feitos analisando as mensagens de depuração do sistema, recebidas através das portas USBs do STM32 e do Arduino Nano e da linha de comando do Raspberry Pi.

3.8 Parte 7: testando na aplicação

Após a constatação de que tanto a programação feita para o firmware como a integração deste com o Hardware do circuito estão funcionando corretamente, o *layout* da Placa de Circuito Impresso poderá ser desenhado utilizando um *software* especializado. Assim como o projeto do acondicionamento da placa em um case externo, onde o circuito ficará protegido depois de pronto.

Com o desenho do *layout* da Placa pronto, esta será confeccionada, e os componentes soldados no circuito, sendo então, feitos novamente os mesmos testes, para constatar se o projeto realmente está funcionando como esperado. Caso não esteja funcionando, reparos precisarão ser feitos na Placa de Circuito Impresso.

Após estar tudo funcionando, o sistema deverá ser testado em um ambiente protótipo, ligando o sensor de fluxo a uma torneira e ao resto no sistema, monitorando a medição do consumo e verificando se o funcionamento está conforme esperado.

Para concluir, estudar a possibilidade de instalar o sistema numa unidade residencial real e coletar os dados, verificando assim que o objetivo foi atingido.

4 RESULTADOS E DISCUSSÕES

Ao longo deste trabalho, foram desenvolvidas as partes do sistema de medição de consumo de água, separadamente, e unidas as soluções de cada parte, criando um protótipo do sistema proposto.

O fato de ter-se definido, primeiramente, como seriam as interfaces entre as partes do sistema, permitiu que se particionasse o sistema de forma segura, sem que isso prejudicasse a integração final.

Precisou-se também fazer algumas mudanças no projeto e na metodologia, devido a obstáculos encontrados no caminho, falta de ferramentas adequadas de trabalho, limitações de *hardware* e *software* e necessidade de simplificação do processo de desenvolvimento, devido à complexidade do sistema.

A Figura 17 é uma representação simplificada do resultado do sistema, que mostra uma visão geral do seu funcionamento.

Figura 17 – Diagrama de blocos: Funcionamento Geral

Fonte: Autoria própria (2023).

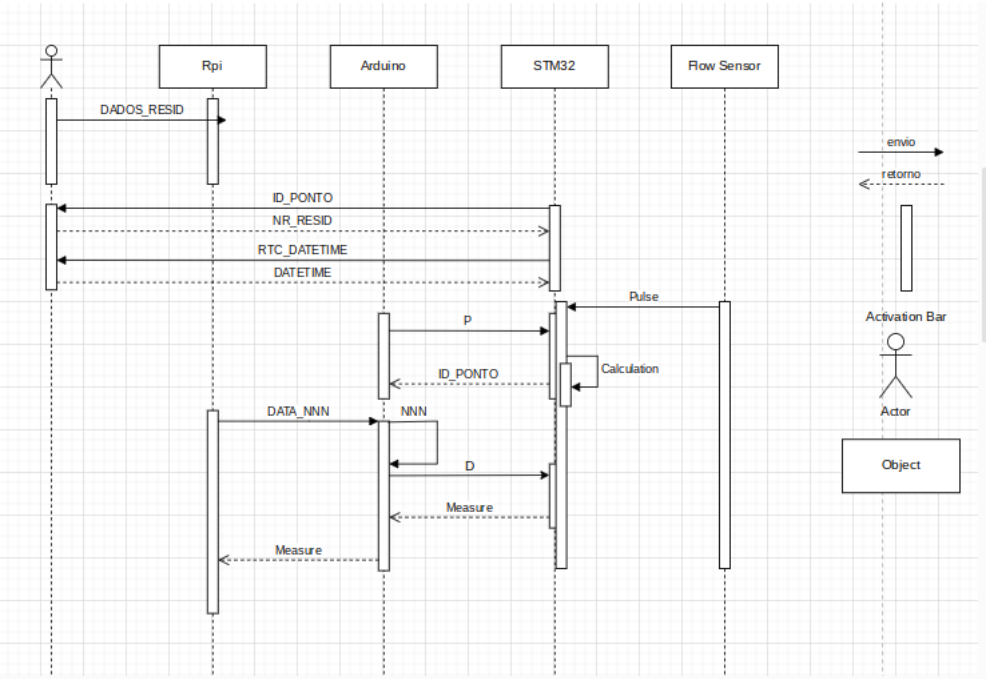
Para se aprofundar nos detalhes do funcionamento desse sistema, principalmente da comunicação entre os dispositivos, pode-se ver o diagrama da Figura 18, que mostra o resultado obtido do funcionamento do sistema, numa visão de uma única unidade residencial.

Neste diagrama está expresso uma necessidade que surgiu durante o projeto: toda vez que um técnico for instalar o sistema numa unidade residencial, momento da instalação, será necessário fazer algumas configurações:

- No Raspberry Pi: registrar os dados da unid. residencial, no banco de dados.
- No STM32: registrar o número da residência (ID_ponto) e ajustar data e hora do RTC (*Real Time Clock*).

A configuração de data e hora do RTC do STM32 também deve ser feita sempre que o circuito do STM for desenergizado. Isso se deve, porque ele não está usando uma bateria

Figura 18 – Diagrama Detalhado do Sistema



Fonte: Autoria própria (2023).

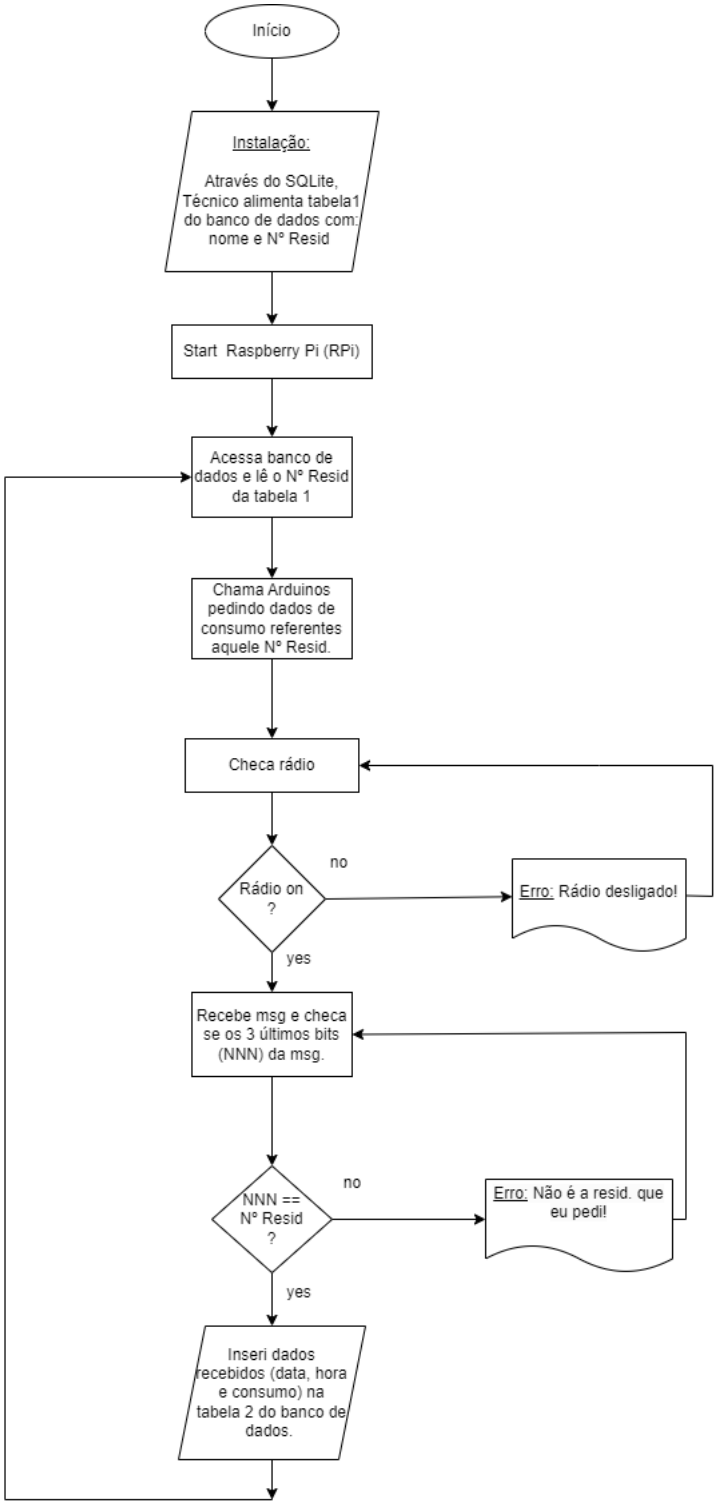
externa para manter o RTC interno funcionando, após o STM ser desligado. Isso acaba se tornando até um potencial de melhoria, pois no dia-a-dia da aplicação pode gerar transtornos não só para o técnico, como para o morador da residência, que pode perder a referência das medições caso falte energia elétrica, ou dê algum problema de queima do equipamento de medição, por exemplo.

O funcionamento do sistema se dá de forma que a toda a comunicação entre as partes do sistema é “puxada” pelo Raspberry Pi. Ele inicia o fluxo do programa solicitando ao Arduino os dados de consumo referente a uma unidade residencial específica, com um número de identificação único. Esse número é uma das muitas chaves específicas que agem como sinalização na comunicação de todo o sistema. Elas garantem que a informação transmitida e/ ou recebida seja a mais precisa possível.

Ao receber a solicitação, o Arduino compara se a chave de identificação corresponde ao identificador do “seu” STM32, ou não. Se corresponder, ele solicita o pacote de dados para o STM32 e envia ao Raspberry Pi, para que ele guarde os dados no banco de dados. Enquanto isso, o STM32 fica medindo os dados do sensor de fluxo e armazenado eles numa fila.

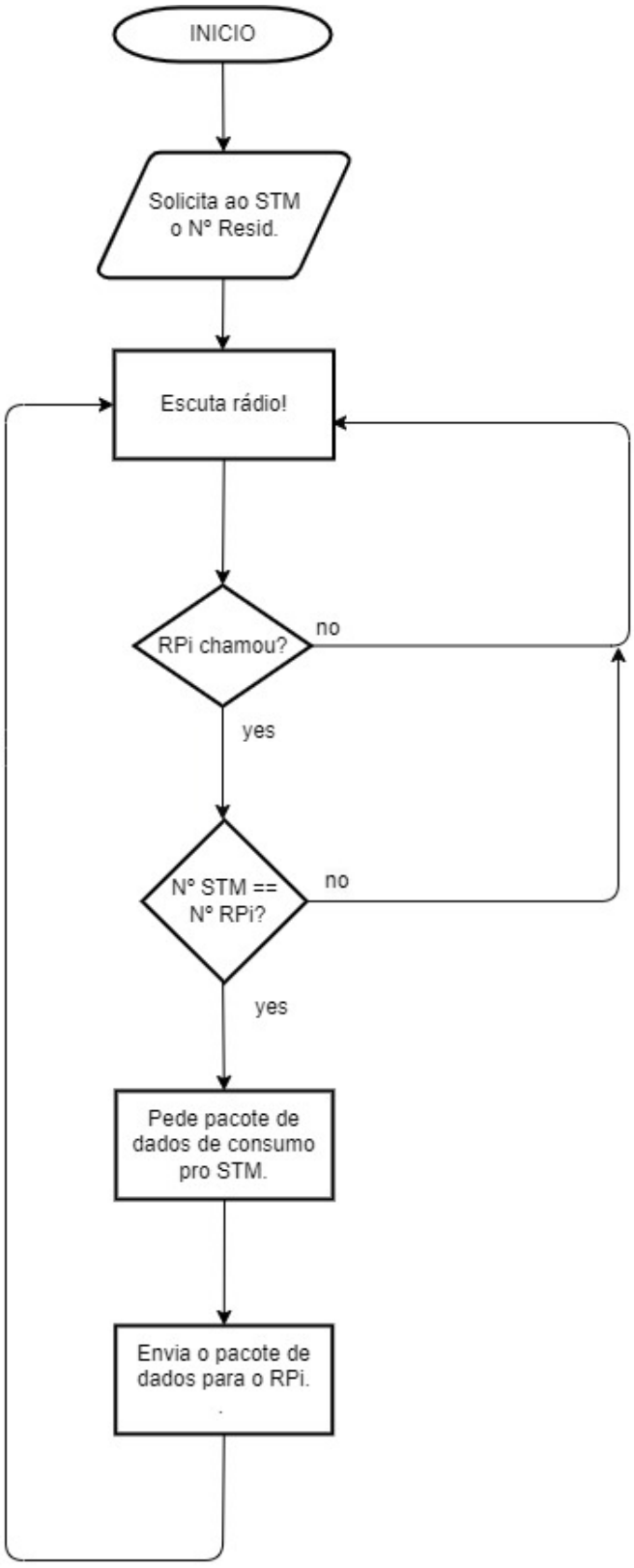
Todo esse funcionamento está bem detalhado nos fluxogramas do Raspberry Pi, Arduino e STM32, vistos nas Figuras 19, 20 e 21, respectivamente.

Figura 19 – Fluxograma Firmware - Raspberry Pi 3



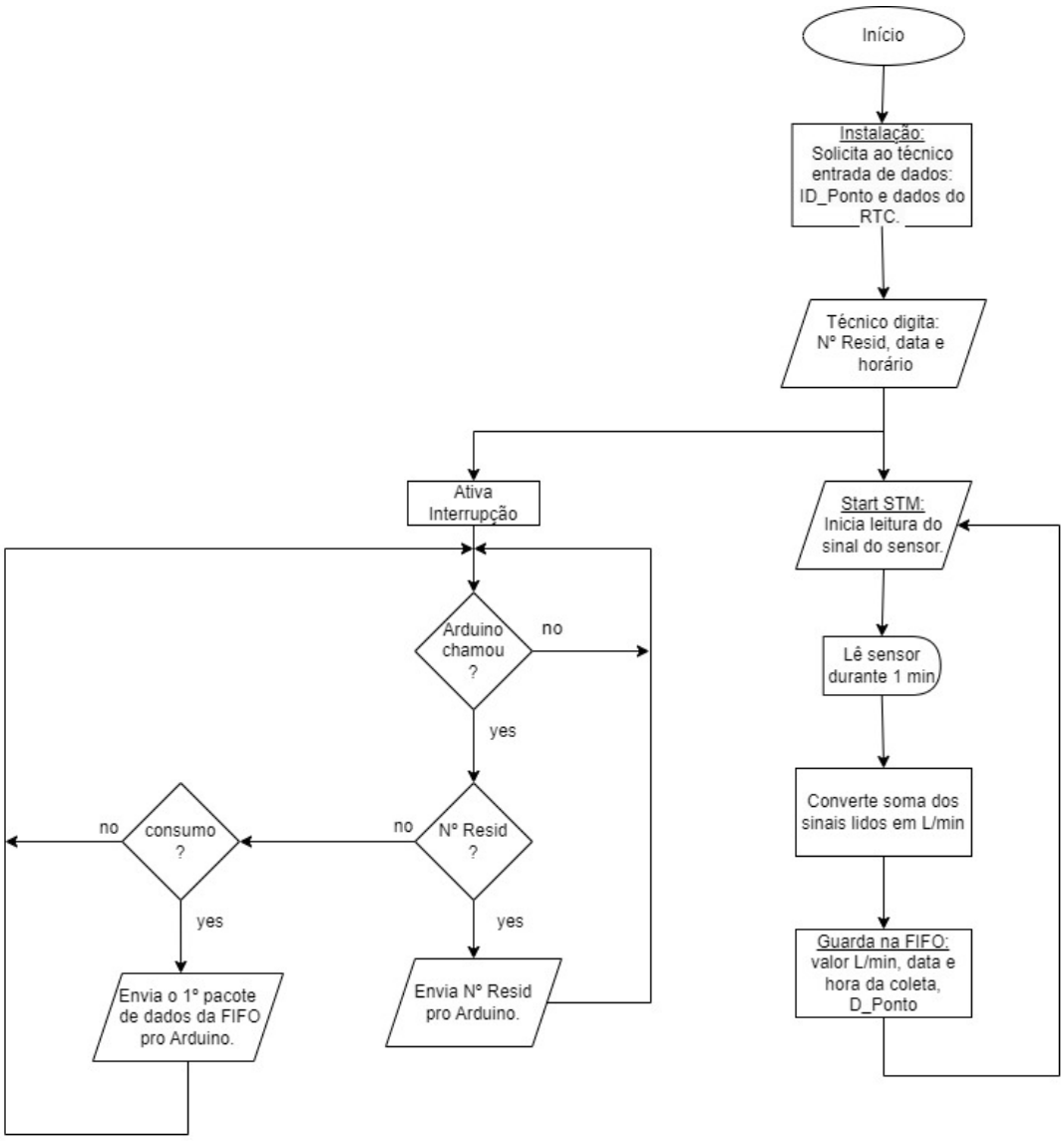
Fonte: Autoria própria (2023).

Figura 20 – Fluxograma Firmware - Arduino Nano



Fonte: Autoria própria (2023).

Figura 21 – Fluxograma Firmware - STM32F103C8T6



Fonte: Autoria própria (2023).

Os códigos dos *firmwares* do STM32, Arduino e Raspberry Pi, que foram desenvolvidos durante o trabalho, podem ser vistos nos Apêndices A, B e C, respectivamente. Assim como os ambientes de desenvolvimento utilizados, que podem ser vistos no Apêndice F.

Em relação ao Raspberry Pi, o que gerou mais dúvidas e representou um nível maior de dificuldade, além de levar um tempo maior de desenvolvimento, foi a comunicação via rádio frequência.

A conexão do Raspberry Pi com o módulo NRF24L01+ é cheia de particularidades, configurações (das portas GPIO, da taxa de dados, do canal, etc) que precisam ser feitas para que ele funcione, além de interferências eletromagnética vinda de outros aparelhos que funcionam na mesma frequência, problemas de falta de documentação e constantes erros em bibliotecas disponíveis.

Depois de muita pesquisa e inúmeras tentativas, finalmente foram corrigidos todos os erros e chegou-se a uma configuração ideal para a aplicação do sistema, como pode ser visto no trecho de código da Figura 22.

Figura 22 – Configurações do NRF24L01

```
def setup():
     #consultar tabela Unidade_Residencial e jogar valores em uma lista

    acessaTabelaResid()
    GPIO.setmode(GPIO.BCM)                  # set the gpio mode
    radio.begin(0, 15)                      #start the radio and set the ce,csn pin ce= GPIO08, csn= GPIO25
    radio.setRetries(7,4)
    radio.setPayloadSize(32)                #set the payload size as 32 bytes
    radio.setChannel(0x76)                  # set the channel as 76 hex
    radio.setDataRate(NRF24.BR_1MBPS)       #set radio data rate
    radio.setPALevel(NRF24.PA_LOW)          #set PA level
    radio.setAutoAck(True)                  # set acknowledgement as true
    radio.openWritingPipe(pipes[0])
    radio.openReadingPipe(1,pipes[1])
```

Fonte: Autoria própria (2023).

O interessante é que o desenvolvimento da conexão entre o STM32 e o NRF, para que o STM32 pudesse se comunicar com o Raspberry Pi via rádio frequência, foi igualmente complexo e demorado, pois surgiram problemas relacionados aos dois, tais como:

- Interface de desenvolvimento da STMicro (fabricante do STM32) apresenta *software* com dependência estrita ao Windows e adaptações do mesmo para outras plataformas apresentam muita variabilidade, pouca maturidade e falta de documentação. Isso levou à escolha da plataforma Arduino IDE como ferramenta de suporte ao desenvolvimento.

- Problema com ST-LINK V2, de procedência duvidosa, que não dispõe de todos os recursos do equivalente oficial da STMicro, como o suporte de *hardware* ao *debug*. Exige programa utilitário com requisitos que são difíceis de tratar (como a configuração de UDEVs no Linux) e a alternativa de *upload* via placa serial carece de documentação de uso.

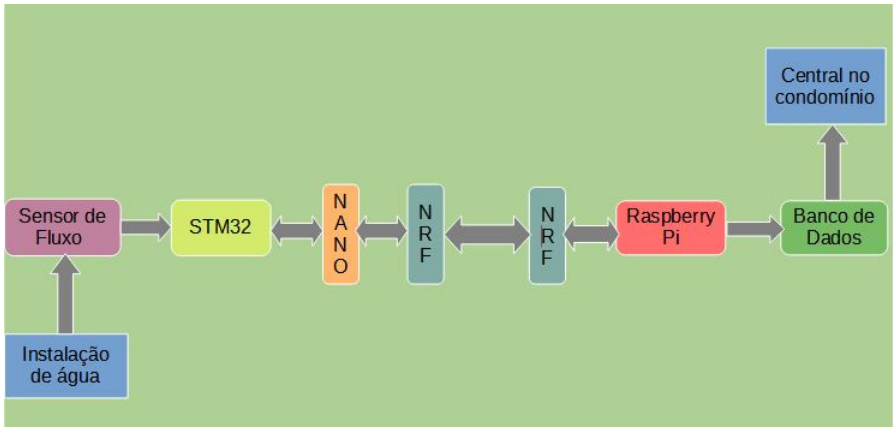
- Mesmo na plataforma Arduino IDE, encontrou-se problemas quanto ao uso efetivo dos recursos do processador, por parte de pacote de suporte ao STM32, levando a inúmeros problemas de dependência de biblioteca; foi necessário então, trocar o pacote para o disponibilizado pela própria STMicro (o STM32duino), para que tais problemas fosem solucionados. Vale ressaltar que tal package não é disponibilizado nativamente pelo Arduino IDE.

- Módulos adquiridos como sendo nRF24L01+, mas que vinham com o nRF24L01 (sem o +), o que impossibilitava algumas opções de configuração, como o uso de uma frequência mais baixa na comunicação por exemplo, gerando incerteza com relação ao funcionamento dos módulos e uma grande perda de tempo de desenvolvimento.

- Adicionalmente, os nRF24L01+ apresentam sensibilidade grande a variação de tensão de alimentação e apresenta problemas não documentados oficialmente, exigindo a busca ostensiva por respostas.

Assim, devido principalmente a grande quantidade de horas de trabalho em excesso, geradas por estes problemas, optou-se por mudar o projeto e colocar um Arduino Nano ligado a porta Serial do STM32, para que ele se conectasse ao NRF no lugar do STM32. No diagrama da Figura 23, pode-se ver o resultado do sistema com a adição do Arduino Nano.

Figura 23 – Diagrama de Blocos: Funcionamento Geral do Sistema com Arduino Nano



Fonte: Autoria própria (2023).

Uma surpresa positiva veio com o uso de módulos RF-Nano, onde a integração do chip nRF24L01+ foi bem feita e não representou nenhum grande problema para o estabelecimento das conexões. No final, optou-se pelo Arduino Nano “puro”, com módulo nRF24L01+ separado, porque o RF-Nano disponível tem um alcance do link de RF menor e não possui suporte para antena externa.

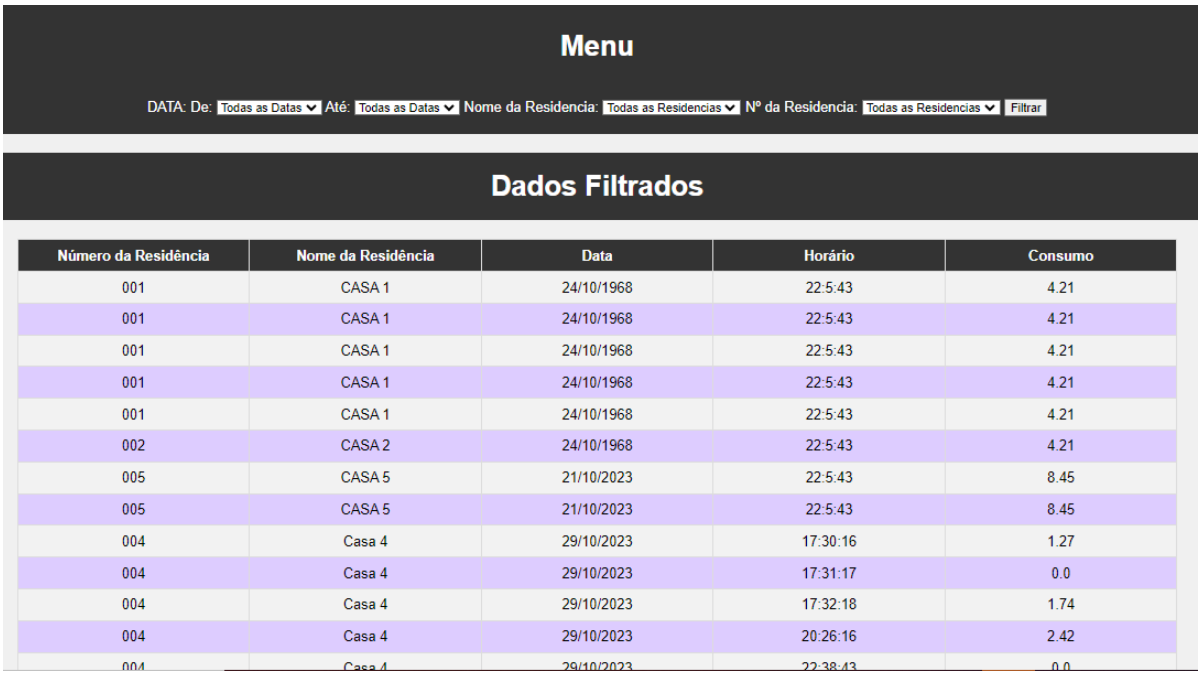
Com relação ao processo de captura de dados do sensor, a placa BluePill STM32F103C8T6 atua fazendo a contagem de pulsos do sensor, através de interrupção, e a captura do valor do contador em processo de pooling interno no *loop* do *firmware*. Houve tentativa de implementação completa via interrupção mas sem sucesso, tanto usando os *timers* internos quanto usando alarmes do RTC. O que se pode apreender de alguns fóruns é que há uma desconfiança que alguns chips estão sendo colocados no mercado como íntegros mas tem problemas no RTC e *timers*.

A criação do banco de dados, assim como de suas tabelas e métodos de inserção, consulta e controle de dados, foi feita com facilidade, através a biblioteca SQLite e usando o próprio Raspberry Pi como interface de desenvolvimento. Pode-se visualizar o banco de dados criado e seus detalhes, consultando o Apêndice D.

Junto do banco, foi criada também uma interface *Web* simples, conectada a uma rede interna ao condomínio, onde o usuário poderá consultar os dados do banco por um computador da Central e extrair informações relevantes de consumo de água, que poderão ser usadas

posteriormente. Os códigos, em Python3 e HTML, podem ser visualizados nos Apêndice E e a face da página, vista na Figura 24.

Figura 24 – Página Web de Interface Usuário-Dados



Número da Residência	Nome da Residência	Data	Horário	Consumo
001	CASA 1	24/10/1968	22:5:43	4.21
001	CASA 1	24/10/1968	22:5:43	4.21
001	CASA 1	24/10/1968	22:5:43	4.21
001	CASA 1	24/10/1968	22:5:43	4.21
001	CASA 1	24/10/1968	22:5:43	4.21
002	CASA 2	24/10/1968	22:5:43	4.21
005	CASA 5	21/10/2023	22:5:43	8.45
005	CASA 5	21/10/2023	22:5:43	8.45
004	Casa 4	29/10/2023	17:30:16	1.27
004	Casa 4	29/10/2023	17:31:17	0.0
004	Casa 4	29/10/2023	17:32:18	1.74
004	Casa 4	29/10/2023	20:26:16	2.42
004	Casa 4	29/10/2023	22:38:43	0.0

Fonte: Autoria própria (2023).

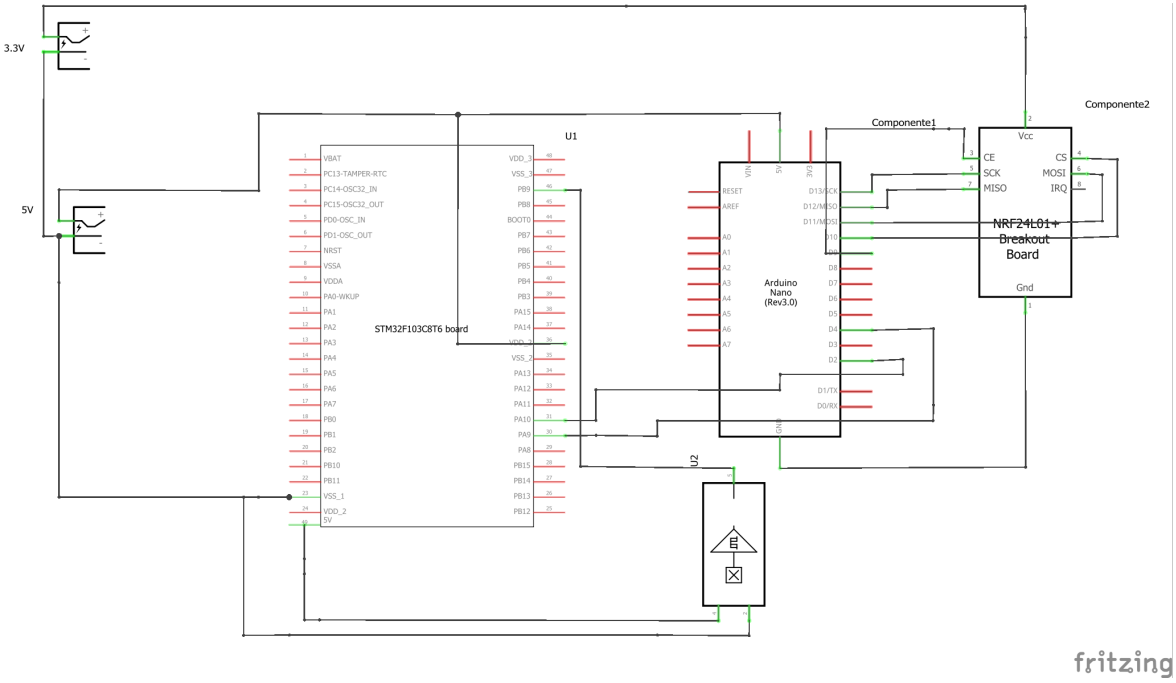
A página *Web* ainda precisa de algumas melhorias, como a função de exportar os dados para um arquivo de planilha eletrônica, por exemplo, que seria muito útil para a administração do condomínio poder tratar os dados, melhorar as opções de filtros e estender o acesso aos dados para os condôminos, entre outros.

A interface do Raspberry Pi com o Banco de dados funcionou conforme o esperado. A escolha do SQLite se deu pelas suas características de performance e baixa exigência de recursos computacionais (*footprint*). Além disso, sua integração com o Python vem permanecendo estável há muito tempo.

Com a integração final de todas as partes do sistema alguns problemas apareceram, notadamente o aparecimento de caracteres espúrios que não fazem parte do acordo de interface celebrado no início do desenvolvimento. Parte do problema estava relacionado ao uso do *buffer* de transmissão do NRF que impõe o envio de 32 bytes sob risco de uma transmissão indevida complemente o *buffer* e gere um pacote inválido no processo de comunicação. Ajustando-se ao comprimento padrão do *buffer* os efeitos colaterais sumiram.

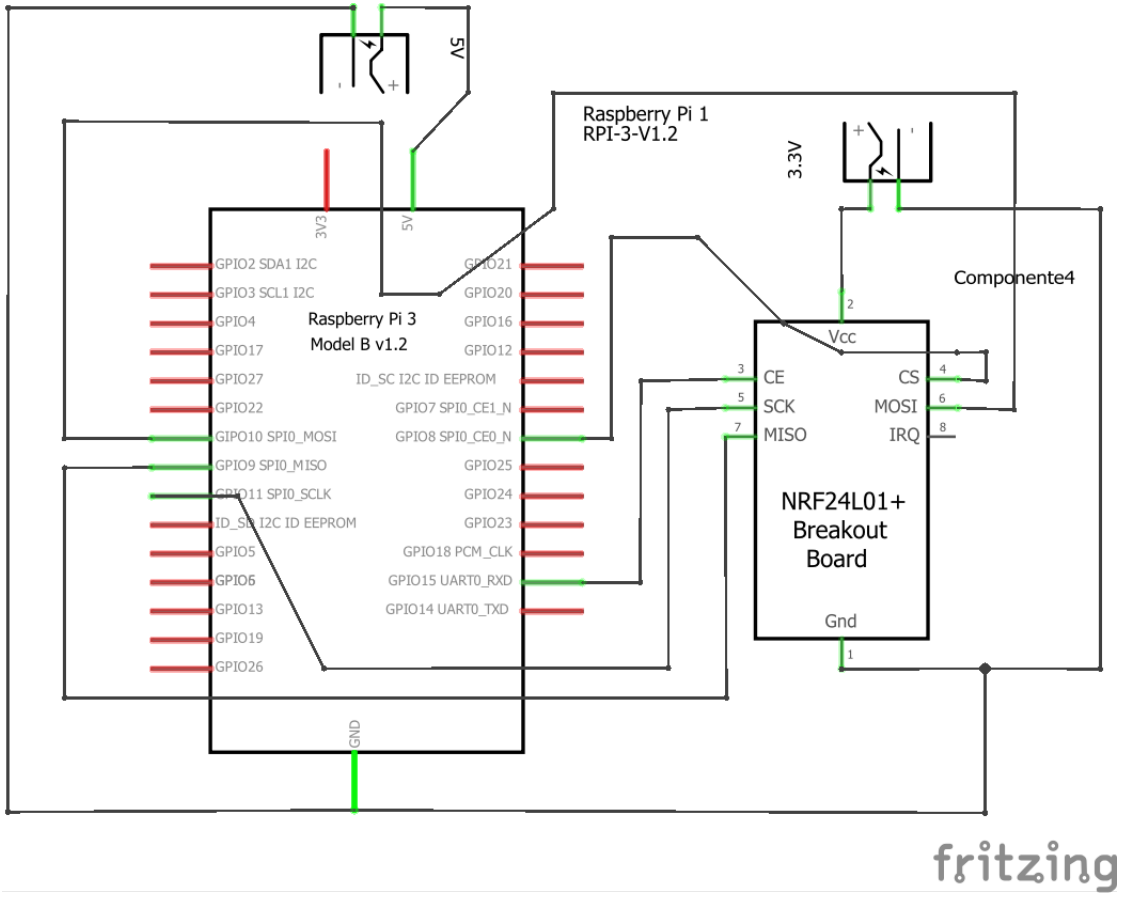
Durante o desenvolvimento dos *firmwares*, foi-se carregando os códigos no circuito, montado na matriz de contato, e testando as saídas e reações do mesmo. A configuração ideal do *hardware* do sistema que ficou definida, por fim, pode ser vista nos circuitos das Figuras 25 e 26.

Figura 25 – Esquemático do Circuito Sensor



Fonte: Autoria própria (2023).

Figura 26 – Esquemático do Circuito Raspberry Pi



Fonte: Autoria própria (2023).

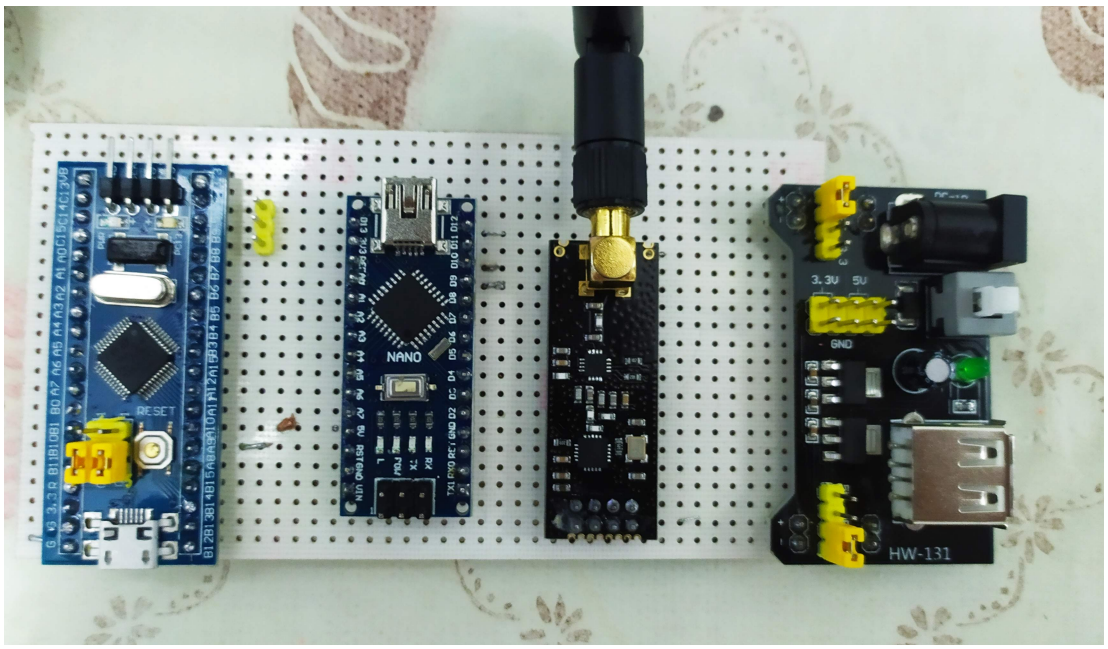
Com todas as partes do sistema funcionando, pôde-se montar um protótipo do sistema completo e testar numa situação que simulasse a instalação do dispositivo numa residência real.

Assim, os componentes do sistema foram soldados numa placa de circuito impresso, e o sensor foi conectado a uma mangueira e ela conectada a uma torneira, por onde sairia a água que passaria por dentro do sensor. Além disso, as Placas do circuito foram conectadas a um computador, via USB ou Wi-Fi, para monitoramento e análise das mensagens de depuração do sistema.

Com isso, foi aberta a torneira e visto que o sensor realmente estava medindo o fluxo da água e enviando dados para o resto do sistema.

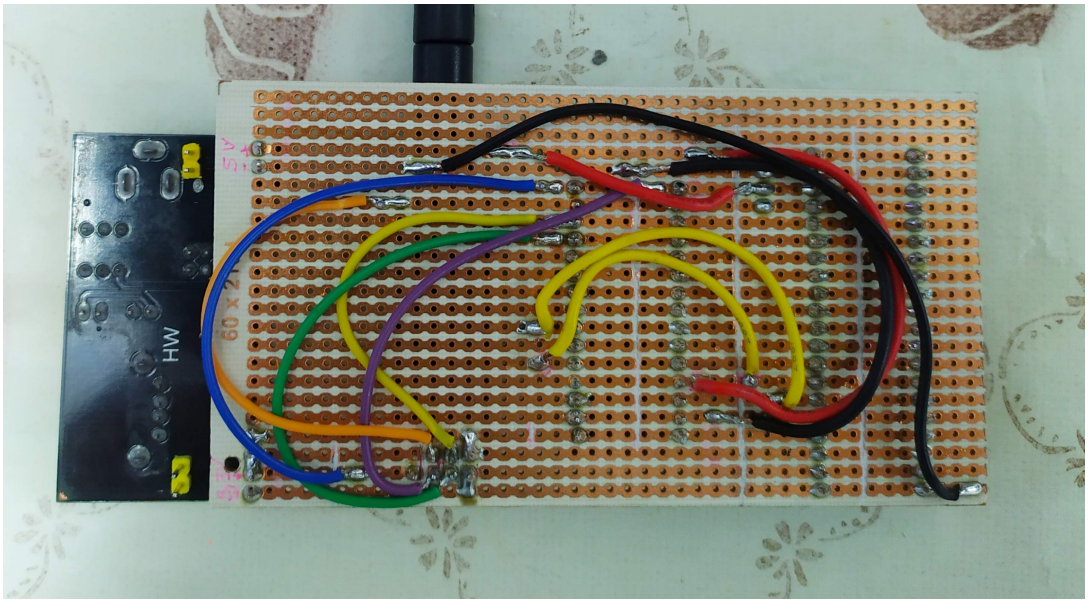
As Figuras 27, 28, 29, 30 e 31 mostram como ficou o protótipo e como foi montado o sistema para medição da vazão. Nas Figuras 32, 33 e 34 pode-se ver a tela com as mensagens de depuração do Raspberry Pi, Arduino e NRF24L01.

Figura 27 – Circuito do Sistema em placa de circuito impresso - Frente



Fonte: Autoria própria (2023).

Figura 28 – Circuito do Sistema em placa de circuito impresso - Verso

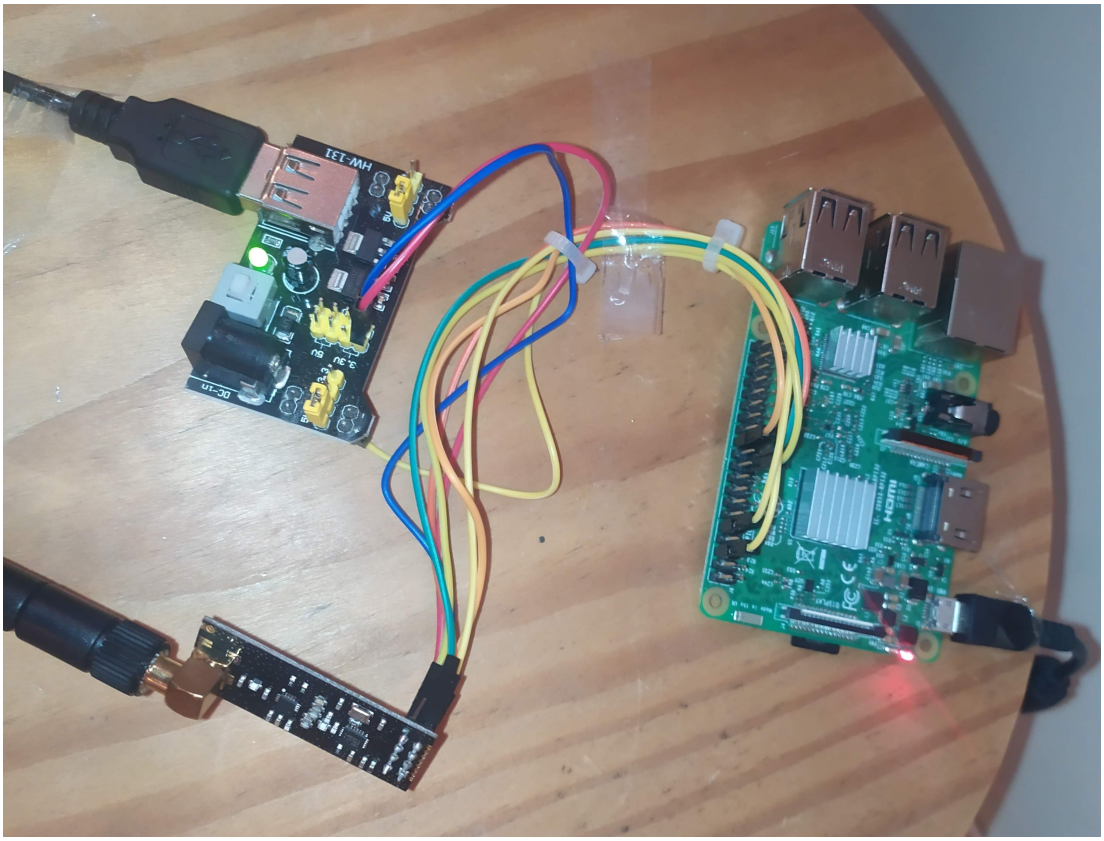


Fonte: Autoria própria (2023).

Figura 29 – Protótipo do Sistema de Medição

Fonte: Autoria própria (2023).

Figura 30 – Circuito Raspberry Pi + NRF

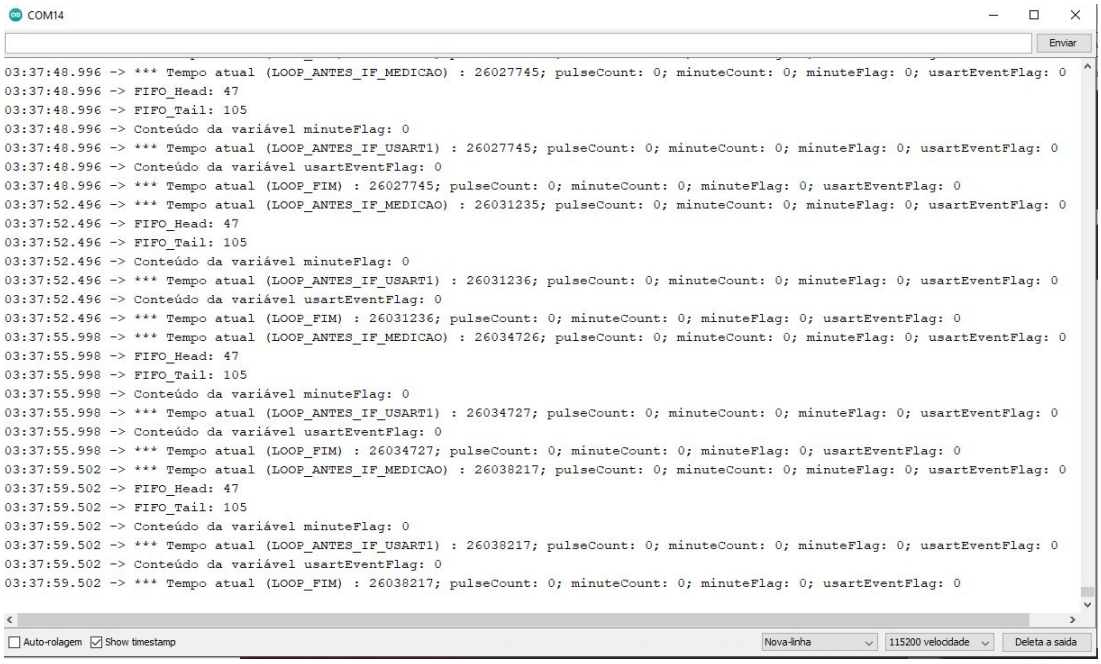


Fonte: Autoria própria (2023).

Figura 31 – Teste do sensor em ambiente simulado

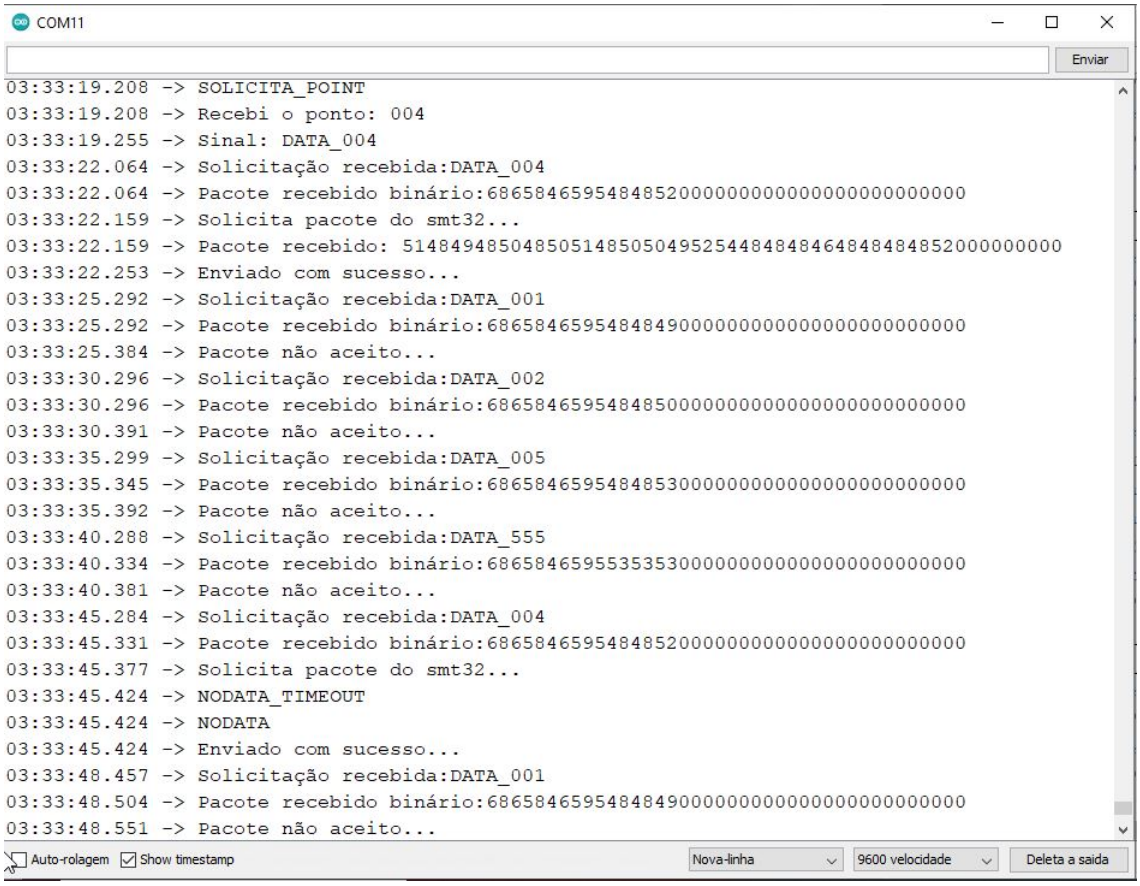
Fonte: Autoria própria (2023).

Figura 32 – Tela do STM32 - mensagem de depuração



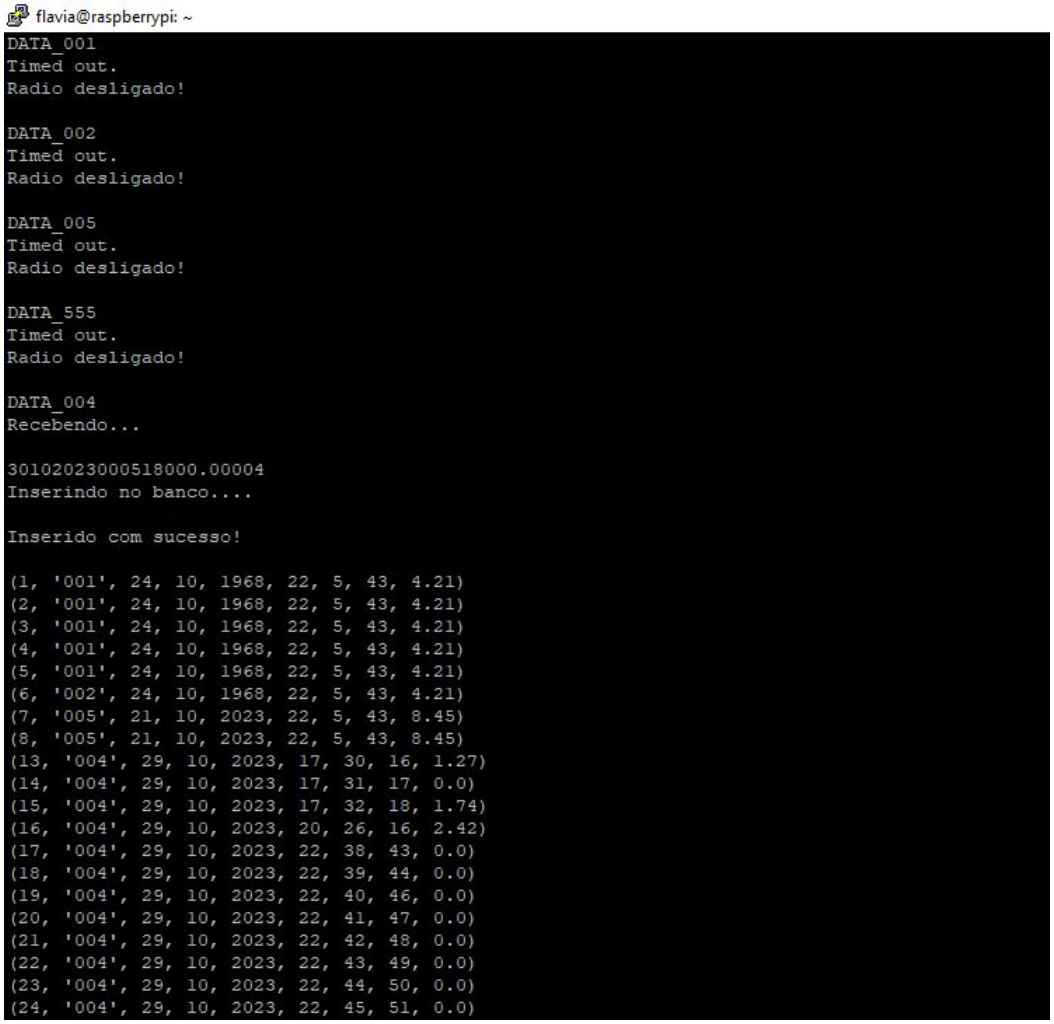
Fonte: Autoria própria (2023).

Figura 33 – Tela do Arduino - mensagem de depuração



Fonte: Autoria própria (2023).

Figura 34 – Tela do RaspberryPi - mensagem de depuração



Fonte: Autoria própria (2023).

Num segundo teste, buscou-se descobrir se o sensor estava medindo o volume de água corretamente. Para isso, precisava-se determinar a real vazão que passava pelo sensor.

Nesse processo de medição, foram utilizados recipientes de plástico com escalas de volume de baixo nível de precisão. Foi usado também um cronômetro e a “precisão” do log do STM32.

Para que o erro fosse minimizado, procurou-se calibrar alguns recipientes a partir de outro padrão, graduado para medida de volume. A medição de vazão, portanto, envolveu a coleta de água por um dado tempo especificado, a determinação do volume coletado nesse tempo e o cálculo da vazão. Com tal procedimento de medição, prosseguiu-se para determinar a vazão máxima da torneira, para servir como limitador do que se teria disponível.

Para a medição da vazão comparativa do sensor, estabeleceu-se o protocolo:

- começa-se a coleta da água que passa pelo sensor quando o *firmware* coloca uma nova medição na fila FIFO interna, usando como evento a sinalização no log do STM32;
- encerra-se a coleta quando nova medição é colocada na fila;

- efetua-se a medição do volume de água coletada;
- calcula-se a vazão, considerando que a base de tempo interna do STM32 é de 1 minuto para coleta, empacotamento e inserção na fila.

Após coletar os dados, medindo durante 1 min, 5 vezes seguidas, observou-se que há uma tendência a uma diferença constante entre os valores medidos através do esquema caseiro e os medidos pelo sensor, indicando eventualmente ser necessário um ajuste no fator de conversão, como se pode observar na Tabela 1.

Tabela 1 – Valores Medidos

N° Medições	Medições (L/min)		
	Sensor	Manual	
		Real	Corrigida
1	12,54	7,95	'---
2	12,14	10,88	10,88
3	11,39	9,7	9,7
4	11,78	10,16	10,16
5	14,41	12,75	12,75
Média:	12,45	10,29	10,87
Desvio Padrão:	1,17	1,75	1,34
Coef. De Variação:	9,40%	17,01%	12,33%

Como podemos observar na medição manual 1, houve uma discrepância grande entre o valor encontrado nesta e o encontrado nas outras medições, provavelmente ocasionado por algum erro no procedimento de medição. Para as outras medições, as distâncias entre a medição manual e a medição pelo sensor, se mantêm com baixa variação, ou seja, aproximadamente constante.

Conforme a coluna da direta da medição manual, ao descartar a primeira medição, a média manual se aproxima da média do sensor. O que se pode concluir é que isso sugere que o fator de ajuste do sensor não está correto, necessitando providenciar sua calibração.

4.1 Potenciais Melhorias

Durante o desenvolvimento do projeto foram surgindo ideias de melhorias que podem ser aplicadas futuramente. São elas:

- A colocação de uma bateria externa para alimentar o RTC do STM32, pois se este ficar sem energia elétrica, desligará e desconfigurará o RTC, afetando a referência utilizada no momento da coleta dos dados;

- A criação de um aplicativo vinculado a página *WEB*, para que todos os moradores do condomínio possam acompanhar o seu consumo de água, pessoalmente;
- Na página Web, criar uma função que exporte os dados para um arquivo do tipo planilha eletrônica, melhorar as opções de filtros e criar a opção que gere um relatório que já apresente o rateio da conta de água, feito conforme consumo individual.

5 CONCLUSÃO

Apesar de todas as dificuldades encontradas durante o desenvolvimento deste projeto, viu-se que é possível desenvolver e implementar um sistema, que coleta dados de consumo de água, relativamente simples e autônomo, tornando a medição individualizada uma opção viável, segura e mais atrativa para os condomínios.

Observou-se, ao final do projeto que o sistema funcionou corretamente, medindo o volume, coletando os dados e transmitindo-os, conforme foi proposto no início. Porém, a questão da precisão da medição ainda é um fator importante que precisa ser reavaliado, com tempo e atenção, para que seja feita a correta configuração do sensor e garantir que este esteja medindo e convertendo corretamente.

O sistema de medição de consumo ainda possui muito potencial de melhoria e aperfeiçoamento de suas funções, para atender cada vez mais às necessidades dos moradores de condomínio. Apesar disso, não se pode negar a evolução que o projeto teve até aqui e que conseguiu cumprir o que foi proposto.

REFERÊNCIAS

ABNT. **NBR 15538**: Hidrômetros para água fria - ensaios para avaliação de desempenho de hidrômetros em alta e baixa vazões em hidrômetro até 25 m3h de vazão nominal para água fria. Rio de Janeiro, 2007. 24 p.

BLOG_ELETROGATE. **Primeiros Passos com o Raspberry Pi 3 e OS Raspbian.** 2022. Disponível em: https://blog.eletrogate.com/primeiros-passos-com-o-raspberry-pi-3-e-os-raspbian/. Acesso em: 27 out. 2023.

BRASIL. Lei nº 13312, de 12 de julho de 2016. Brasília, jul. 2016.

BROWN, G. **Descobrindo o Microcontrolador STM32**. [*s.n.*], 2012. Disponível em: https://img.filipeflop.com/files/download/Descobrindo\%20o\%20STM32.pdf. Acesso em: 12 abr. 2019.

CARVALHO JUNIOR, R. d. **Instalações hidráulicas e o projeto de arquitetura**. 7. ed. São Paulo: Blucher, 2013.

COELHO, A. C. **Manual de economia de água**: conservação de água. Olinda: Ed. do Autor, 2001.

COELHO, A. C. **Medição de água individualizada**: manual do condomínio. Olinda: Luci Artes Gráficas, 2004.

COELHO, A. C.; MAYNARD, J. C. d. B. **Medição Individualizada de água em Apartamentos.** Recife: Ed. dos Autores, 1999.

CONDOMINIOSC. **Finanças. Taxa de condomínio**: Fração ideal ou por unidade? 2018. Disponível em: http://www.condominiosc.com.br/jornal-dos-condominios/financas/2101-taxa-de-condominio-fracao-ideal-ou-por-unidade. Acesso em: 8 jun. 2019.

CONSULTORES, E. **Você sabe como funciona o hidrômetro de água?** 2018. Disponível em: https://www.eosconsultores.com.br/funcionamento-do-hidrometro-de-agua/. Acesso em: 12 abr. 2019.

DIGITAL, I. **Sensor de Fluxo e Vazão de água 1/2 – YF-S201**. 2018. Disponível em: http://www.institutodigital.com.br/pd-24b6c2-sensor-de-fluxo-e-vazao-de-agua-1-2-yf-s201.html. Acesso em: 25 jun. 2019.

DONAT, W. **Programação do Raspberry Pi com Python.(L. A. Kinoshita, Trad.)**. [*S.l.*]: Apress; Novatec, 2019.

ELETROGATE. **Arduino Nano**. 2023. Disponível em: https://www.eletrogate.com/arduino-nano. Acesso em: 25 out. 2023.

ELETRU'S. **Modelo NRF24L012, 4GHZ C ANTENA**. 2019. Disponível em: https://www.eletruscomp.com.br/produto/modulo-nrf24l01-2-4ghz-c-antena/. Acesso em: 17 abr. 2019.

EMBASA. **Guia para Implantação de Sistemas de Medição Individualizada**. 2019. Disponível em: https://atendimentovirtual.embasa.ba.gov.br/documents/1708453/1711061/Guia-Implantacao_Sistemas_Medio_Individualizada.pdf. Acesso em: 5 dez. 2023.

EMBETRONICX. **How wireless nRF24L01+ module works?** 2022. Disponível em: https://embetronicx.com/tutorials/tech_devices/wireless-nrf24l01-module-working/. Acesso em: 27 out. 2023.

FLOP, F. **Placa de Desenvolvimento STM32 F103C8T6**. 2019. Disponível em: https://www.filipeflop.com/produto/placa-de-desenvolvimento-stm32-para-arduino/#tab-description. Acesso em: 8 abr. 2019.

HAIHUILAI. **Sensor de fluxo 3/4"DN20 Cobre**. 2023. Disponível em: https://www.aliexpress.us/item/3256804231489746.html. Acesso em: 3 dez. 2023.

HIDROGERAIS. **Hidrômetros multijato**. 2019. Disponível em: http://www.hidrogerais.com.br/hidrometros-multijato.html. Acesso em: 12 abr. 2019.

LIMA, B. C. Sistema de medição individualizada de água: Estudo de caso de edifÍcio comercial em sÃo paulo. **REEC- Revista Eletrônica de Engenharia Civil**, v. 11, n. 3, p. 56–66, 2016. Disponível em: http://www.reec.com.br. Acesso em: 5 abr. 2019.

MINUCCI, A. d. S.; LIMA, F. A. d. S. **AUTOMATIZAÇÃO NO PROCESSO DE LEITURA DE HIDRÔMETROS**. Ponta Grossa: [*s.n.*], 2013. Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa. Disponível em: http://repositorio.roca.utfpr.edu.br/jspui/handle/1/9713. Acesso em: 1 abr. 2019.

NAKAMURA, J. **Hidrometro Individual em apê ajuda na economia, mas exige investimento alto**. 2016. Disponível em: https://universa.uol.com.br/noticias/redacao/2016/07/15/hidrometro-individual-em-ape-induz-a-economia-mas-exige-investimento-alto.htm?cmpid=copiaecola. Acesso em: 20 maio 2019.

NORDIC. **NRF24L01+ Single Chip 2.4GHz Transceiver Product Specification v1.0**. [*S.l.*], 2019. Disponível em: https://img.filipeflop.com/files/download/Datasheet_nRF24L01P_1_0.pdf. Acesso em: 17 abr. 2019.

OLIVEIRA, L. H. d. **Metodologia para implementação de programa de uso racional da água em edifícios**. 1999. Tese (Doutorado) — Doutorado em Engenharia, Universidade de São Paulo, São Paulo, 1999. Disponível em: http://www.teses.usp.br/teses/disponiveis/3/3146/tde-16042018-084622/publico/LuciaHelenadeOliveira_T.pdf. Acesso em: 8 jun. 2019.

ORACLE. **O que é um banco de dados?** 2023. Disponível em: https://www.oracle.com/br/database/what-is-database/. Acesso em: 26 out. 2023.

PASSOS, I. d.; QUADROS, M. A. C. d.; AVEIRO, A. G. d. **Sistema de Telemetria de Hidrômetro Residencial**. Curitiba: [*s.n.*], 2015. Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa. Disponível em: http://repositorio.roca.utfpr.edu.br/jspui/handle/1/3235. Acesso em: 1 abr. 2019.

QUINTINO, E. d. C. **O que é IDE Arduino?** 2023. Disponível em: https://www.makerhero.com/blog/o-que-e-ide-arduino/. Acesso em: 25 out. 2023.

RASPBERRYPI. **Hardware Raspberry Pi**. 2023. Disponível em: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html. Acesso em: 27 out. 2023.

SAMAE. **Apostila ajustador de hidrômetro**. 2019. Disponível em: http://www.samaecaxias.com.br/Concurso/DownloadArquivoConcurso/Apostila%20Ajustador%20de%20Hidr%C3%B4metros.pdf. Acesso em: 11 abr. 2019.

SEA. **MODEL YF-S201**. [*S.l.*], 2023. Disponível em: http://www.mantech.co.za/Datasheets/Products/YF-S201_SEA.pdf. Acesso em: 25 out. 2023.

SOUZA, D. J. d. **Desbravando o PIC**: Baseado no microcontrolador pic 16f84. São Paulo: Érica, 2000.

TECHEM. **Perguntas frequentes**. 2019. Disponível em: https://www.techem.com.br/perguntas-frequentes/para-os-que-ainda-nao-sao-clientes.html. Acesso em: 20 maio 2019.

TUTORIALSPOINT. **SQLite**: Visão geral. 2023. Disponível em: http://www.tutorialspoint.com/sqlite/sqlite_overview.htm. Acesso em: 25 out. 2023.

APÊNDICE A – Código do firmware do STM32F103C8T6

```
//Código do firmware do STM32F103C8T6
//stm322.ino

#include <STM32RTC.h>
#include <FlashStorage_STM32.h>
#include <Arduino.h>
#include <cstring> // Para a função memcpy
#include <stdint.h>
#include <string.h>
#include "stm32_v1.h"

/* Change this value to set alarm match offset in millisecond */
//static uint32_t atime = 1000;

// Variável para armazenar o último tempo em
// que o processo foi acionado
unsigned long previousMillis;
// Intervalo desejado em milissegundos
// (1000 = 1 segundo, por exemplo)
const long interval = 60000;

// Contagem de pulsos do sensor
volatile int pulseCount = 0;
// Contagem de pulsos por minuto do sensor
volatile int minuteCount = 0;
// Sinalizador de pulsos por minuto disponivel
volatile bool minuteFlag = false;

//variavel armazena média de consumo a cada 1 min
float consumoPorMin;

// Sinalizador de que tem mensagem pela USART1
volatile bool usartEventFlag = false;
const char sinalNanoD = 'D';
const char sinalNanoP = 'P';
//volatile char sinalRecebNano;
char sinalRecebNanoNV;

struct Msg mensagem;
struct Ponto id_ponto;
char point[4]; //"NNN"

// Controle do ritmo de log no código
const int log_count = 1000000;
int _log = 0;
int i = 0;
```

```
//EEPROM
const int WRITTEN_SIGNATURE = 0xBEEFDEED;
const int EMPTY_SIGNATURE   = 0x00000000;
const int START_ADDRESS     = 0;
//EEPROM address to start reading from
const int eeAddress = START_ADDRESS + sizeof(WRITTEN_SIGNATURE);
// Indica se foi feita a gravação da EEPROM com o id do ponto

int signature;

// Funções principais
void setup() {
    delay(5000);
    // Inicializa a Serial USB
    Serial.begin(115200);
    // Inicializa a comunicação serial USART1
    // nos pinos TX/RX (PA9 / PA10)
    Serial1.begin(9600);

    monitor("SETUP_ANTES_FLASH_WRITE");

    // Trata o ID do ponto e gravação na Flash (EEPROM):
    // Permite invalidar gravação do id do ponto
    clearIdPonto();

    // Verifica se o ID do ponto de medição existe;
    Serial.println("Validando a gravação do ID do Ponto...");
    validaIdPonto();

    monitor("SETUP_ANTES_AJUSTE_RTC");

    // Inicializa RTC
    //rtc.setPrediv(-1, -1);
    rtc.begin();
    //Ajusta RTC
    Serial.println("Ajuste dos valores iniciais do RTC...");
    ajustaRTC();

    monitor("SETUP_ANTES_INTERRUP");

    // Interrupções
    // Interrupção para contar pulsos de subida no pino PB9
    pinMode(SENSORPIN,INPUT);
    digitalWrite(SENSORPIN, HIGH); // Optional Internal Pull-Up
    //Configura o pino PB9 (interrupção 0)
    //RISING :acionar a interrupção quando o
    // estado do pino for de LOW para HIGH apenas
```

```
    attachInterrupt(digitalPinToInterrupt(SENSORPIN),pulse,RISING);

    if (rtc.isConfigured()) {
        Serial.println("RTC configurado!");
    } else {
        Serial.println("RTC NÃO CONFIGURADO!");
    }

    if (rtc.isAlarmEnabled()) {
        Serial.println("Alarme A do RTC configurado!");
    } else {
        Serial.println("Alarme A do RTC NÃO CONFIGURADO!");
    }

    monitor("SETUP_FIM");

    delay(5000);

    previousMillis = millis();
}

void loop() {

    consumoPorMin = 0.0;

    //Dispara processo de resposta a eventos da USART1
    if (Serial1.available()) {
        sinalRecebNanoNV = Serial1.read();
        usartEventFlag = true;
    }

    // Dispara processo periódico a cada intervalo (60 segundos)
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= interval) {
      // Atualiza o último tempo em que o processo foi acionado
        previousMillis = currentMillis;

      // Executa o processo periódico
      // Captura o valor do contador de pulsos
        minuteCount = pulseCount;
        // Zera o contador de pulsos para o próximo minuto
        pulseCount = 0;
        minuteFlag = true; // Seta flag
    }

    if ((i % log_count) == 0) {
        _log = 1;
```

```
     i   = 0;
} else {
    _log = 0;
}

if (_log) {
    monitor("LOOP_ANTES_IF_MEDICAO");
}

//Imprime index da FIFO
if (_log) {
    Serial.print("FIFO_Head: ");
    Serial.println(head);
    Serial.print("FIFO_Tail: ");
    Serial.println(tail);

    Serial.print("Conteúdo da variável minuteFlag: ");
    Serial.println(minuteFlag);
}

if(minuteFlag){
    Serial.print("Iniciando tratamento da medição");
    Serial.print("e colocação na FIFO...");

    monitor("IF_MINUTE_FLAG");

    Serial.print("Conteúdo da variável minuteCount: ");
    Serial.println(minuteCount);

    // Sensor YF-S201 - Calcula o consumo por minuto [L/min]
    consumoPorMin = minuteCount/450.0;

    // Mostra string formatada
    char buffer[7]; // Buffer para armazenar a string formatada
    // Converte o float para uma string com a máscara
    // de formatação desejada
    snprintf( buffer
            , sizeof(buffer)
            , "%03d.%02d"
            , (int)consumoPorMin
            , int(consumoPorMin * 100) % 100
            );
    // Imprime a string formatada
    Serial.print("Consumo por minuto formatado: ");
    Serial.println(buffer);

    // Preenche a struct com os valores obtidos do RTC,
```

```
    // consumo e id do ponto
    preencherDadosRTC(&mensagem.dateTime);
    // Ajusta o consumo por minuto em um formato "%03d.%02d"
    //snprintf( reinterpret_cast<char*>(mensagem.consumo)
    //        , sizeof(mensagem.consumo)
    //        , "%03d.%02d"
    //        , (int)consumoPorMin
    //        , (int)(consumoPorMin * 100) % 100
    //        );
    memcpy(mensagem.consumo, buffer, 6);
    // Recupera o valor do identificador de
    // configuração do ponto
    EEPROM.get(eeAddress, mensagem.idPonto);

    // Tamanho da estutura de medição
    size_t tamanho = sizeof(mensagem);
    Serial.print("Tamanho do pacote de dados da medição: ");
    Serial.println(tamanho);

    // Apresentar estrutura Msg formatada
    printFormattedStruct(mensagem);

    // Inserir medição na FIFO
    enqueue(mensagem);

    // Aguarda nova medição
    minuteFlag = false;
}

if (_log) {
    monitor("LOOP_ANTES_IF_USART1");
}

if (_log) {
    Serial.print("Conteúdo da variável usartEventFlag: ");
    Serial.println(usartEventFlag);
}

// Buffer temporário não volátil
//char buffer[3];

if (usartEventFlag) {
    Serial.println("Solicitação de pacote recebida...");
    //Serial.print("Conteúdo do array sinalRecebNano: ");
    //Serial.println(buffer);
    Serial.print("Conteúdo da variável sinalRecebNano: ");
    Serial.println(sinalRecebNanoNV);
```

```
        if (sinalRecebNanoNV == sinalNanoD) {
            Serial.println("Recebido solicitação de pacote de medição.");
            if (!isEmpty()) {
                Serial.println("Há pacote de medição disponível na FIFO.");
                // Envia uma medição da fila para o Nano
                envia_Msg(dequeue(), "Serial1");
            } else {
                Serial.println("Não há pacote de medição na FIFO.");
                // Sinaliza para o Nano que não há medição a transferir
                Serial1.print("NODATA");
            }
        } else if (sinalRecebNanoNV == sinalNanoP) {
            Serial.println("Recebido solicitação do ID do ponto.");
            EEPROM.get(eeAddress, id_ponto);
            envia_Ponto(&id_ponto, "Serial1");
        } else {
            Serial.print("Solicitação de pacote recebida ");
            Serial.println("está com sinalização errada.");
        }
        // Seta flag para false
        usartEventFlag = false;
    }

    if (_log) {
        monitor("LOOP_FIM");
    }

    i++;
}

// ===== Funções do EEPROM =====
// Invalida a gravação do ID do ponto
void clearIdPonto() {
    Serial.readStringUntil('\n'); //limpa buffer da Serial
    Serial.print("Quer invalidar a gravação anterior ");
    Serial.print("do ID do Ponto (N/S) ?: ");
    while (Serial.available() < 1) {
        delay(10);
    }
    char resposta = Serial.read();
    Serial.println(resposta);
    if (resposta == 'S') {
        EEPROM.put(START_ADDRESS, EMPTY_SIGNATURE);
        if (!EEPROM.getCommitASAP()) {
           Serial.println("CommitASAP não rodou... Precisa fazer o commit()");
           EEPROM.commit();
```

```
        }
    } else {
        Serial.println("Gravação anterior do ID do Ponto mantida...");
    }
}
// Valida a gravação do ID do ponto
void validaIdPonto() {
    Serial.print("EEPROM length: ");
    Serial.println(EEPROM.length());
    // Lê assinatura
    EEPROM.get(START_ADDRESS, signature);

    if (signature != WRITTEN_SIGNATURE) {
        EEPROM.put(START_ADDRESS, WRITTEN_SIGNATURE);
        putID(eeAddress);
        if (!EEPROM.getCommitASAP()) {
            Serial.println("CommitASAP não rodou... Precisa fazer o commit()");
            EEPROM.commit();
        }
        Serial.println("Identificador gravado na memória Flash!");
    } else {
        EEPROM.get(eeAddress, id_ponto);
        Serial.print("Identificador encontrado: ");
        envia_Ponto(&id_ponto, "Serial");
    }
}
// Grava o ID do ponto a partir da entrada USB
void putID(int address) {
    Serial.readStringUntil('\n'); //limpa buffer da Serial
    Serial.print("Entre o id do ponto de medição com três algarismos (NNN): ");
    // Mantém espera até que o dado seja fornecido
    while (Serial.available() < 3) {
        delay(10);
    }
    // Preenche estrutura
    for (int i=0;i<3;i++) {
        // Lê um caractere
        char resposta = Serial.read();
        point[i] = resposta;
        id_ponto.ponto[i] = resposta;
    }
    // Adiciona o caractere nulo ao final das strings
    point[3] = '\0';
    id_ponto.ponto[3] = '\0';
    Serial.println(point);
    // Grava estrutura na flash
    EEPROM.put(address, id_ponto);
```

```
}

// ===== Funções do RTC =====
// Ajusta o relógio do RTC a partir da entrada USB
void ajustaRTC() {
    //limpa buffer da Serial
    Serial.readStringUntil('\n');
    // Buffer para armazenar a entrada do usuário como
    // uma string (2 char + terminator)
    char buffer[3];
    // Variável para armazenar o valor convertido de ASCII para inteiro
    uint8_t value;

  // Ajusta a data
  // Dia
    Serial.print("Entre com o Dia (DD):");
    while (Serial.available() < 2) {
        delay(10);
    }
    // Lê os dois caracteres da entrada do usuário como uma string
    Serial.readBytes(buffer, 2);
    //limpa buffer da Serial
    Serial.readStringUntil('\n');
    // Converte o buffer em um número inteiro (uint8_t)
    uint8_t day = atoi(buffer);
    // Imprime o valor na serial usb
    Serial.println(day);
    // Seta o valor do campo do RTC
    rtc.setDay(day);
    // Limpa o buffer para a próxima entrada
    memset(buffer, 0, sizeof(buffer));
  // Mes
    Serial.print("Entre com o Mês (MM):");
    while (Serial.available() < 2) {
        delay(10);
    }
    // Lê os dois caracteres da entrada do usuário como uma string
    Serial.readBytes(buffer, 2);
    //limpa buffer da Serial
    Serial.readStringUntil('\n');
    // Converte o buffer em um número inteiro (uint8_t)
    uint8_t month = atoi(buffer);
    // Imprime o valor na serial usb
    Serial.println(month);
    // Seta o valor do campo do RTC
    rtc.setMonth(month);
    // Limpa o buffer para a próxima entrada
```

```
  memset(buffer, 0, sizeof(buffer));
// Ano
  // O RTC do STM32 somente aceita ano com dois digitos
  Serial.print("Entre com o Ano (YY):");
  while (Serial.available() < 2) {
      delay(10);
  }
  // Lê os dois caracteres da entrada do usuário como uma string
  Serial.readBytes(buffer, 2);
  //limpa buffer da Serial
  Serial.readStringUntil('\n');
  // Converte o buffer em um número inteiro (uint8_t)
  uint8_t year = atoi(buffer);
  // Imprime o valor na serial usb
  Serial.println(year);
  // Seta o valor do campo do RTC
  rtc.setYear(year);
  // Limpa o buffer para a próxima entrada
  memset(buffer, 0, sizeof(buffer));

// Ajusta o tempo
// horas
  Serial.print("Entre com as horas (HH):");
  while (Serial.available() < 2) {
      delay(10);
  }
  // Lê os dois caracteres da entrada do usuário como uma string
  Serial.readBytes(buffer, 2);
  //limpa buffer da Serial
  Serial.readStringUntil('\n');
  // Converte o buffer em um número inteiro (uint8_t)
  uint8_t hours = atoi(buffer);
  // Imprime o valor na serial usb
  Serial.println(hours);
  // Seta o valor do campo do RTC
  rtc.setHours(hours);
  // Limpa o buffer para a próxima entrada
  memset(buffer, 0, sizeof(buffer));
// minutos
  Serial.print("Entre com os minutos (MM):");
  while (Serial.available() < 2) {
      delay(10);
  }
  // Lê os dois caracteres da entrada do usuário como uma string
  Serial.readBytes(buffer, 2);
  //limpa buffer da Serial
  Serial.readStringUntil('\n');
```

```
    // Converte o buffer em um número inteiro (uint8_t)
    uint8_t minutes = atoi(buffer);
    // Imprime o valor na serial usb
    Serial.println(minutes);
    // Seta o valor do campo do RTC
    rtc.setMinutes(minutes);
    // Limpa o buffer para a próxima entrada
    memset(buffer, 0, sizeof(buffer));
  // segundos
    Serial.print("Entre com os segundos (SS):");
    while (Serial.available() < 2) {
        delay(10);
    }
    // Lê os dois caracteres da entrada do usuário como uma string
    Serial.readBytes(buffer, 2);
    //limpa buffer da Serial
    Serial.readStringUntil('\n');
    // Converte o buffer em um número inteiro (uint8_t)
    uint8_t seconds = atoi(buffer);
    // Imprime o valor na serial usb
    Serial.println(seconds);
    // Seta o valor do campo do RTC
    rtc.setSeconds(seconds);
    // Limpa o buffer para a próxima entrada
    memset(buffer, 0, sizeof(buffer));
}

// Função para preencher os dados do RTC na estrutura RTCData
void preencherDadosRTC(struct RTCData *dadosRTC) {
    //Serial.readStringUntil('\n'); //limpa buffer da Serial
    Serial.println("Preenchendo data/hora do RTC na mensagem...");

    // STM32F1 suporta ano no RTC somente no formato YY
    const char seculo[3] = "20";
    char buffer[3];
    uint8_t u_buffer[2];
    uint8_t a_buffer[4];

    //Segundos
    snprintf(buffer, sizeof(buffer), "%02d", rtc.getSeconds());
    u_buffer[0] = buffer[0];
    u_buffer[1] = buffer[1];
    //dadosRTC->segundo = u_buffer;
    memcpy(dadosRTC->segundo, u_buffer, sizeof(u_buffer));
    // Minutos
    snprintf(buffer, sizeof(buffer), "%02d", rtc.getMinutes());
    u_buffer[0] = buffer[0];
```

```
    u_buffer[1] = buffer[1];
    //dadosRTC->minuto = u_buffer;
    memcpy(dadosRTC->minuto, u_buffer, sizeof(u_buffer));
    // Horas
    snprintf(buffer, sizeof(buffer), "%02d", rtc.getHours());
    u_buffer[0] = buffer[0];
    u_buffer[1] = buffer[1];
    //dadosRTC->hora = u_buffer;
    memcpy(dadosRTC->hora, u_buffer, sizeof(u_buffer));
    // Dia
    snprintf(buffer, sizeof(buffer), "%02d", rtc.getDay());
    u_buffer[0] = buffer[0];
    u_buffer[1] = buffer[1];
    //dadosRTC->dia = u_buffer;
    memcpy(dadosRTC->dia, u_buffer, sizeof(u_buffer));
    // Mes
    snprintf(buffer, sizeof(buffer), "%02d", rtc.getMonth());
    u_buffer[0] = buffer[0];
    u_buffer[1] = buffer[1];
    //dadosRTC->mes = u_buffer;
    memcpy(dadosRTC->mes, u_buffer, sizeof(u_buffer));
    // Ano
    snprintf(buffer, sizeof(buffer), "%02d", rtc.getYear());
    a_buffer[0] = seculo[0];
    a_buffer[1] = seculo[1];
    a_buffer[2] = buffer[0];
    a_buffer[3] = buffer[1];
    //dadosRTC->ano = a_buffer;
    memcpy(dadosRTC->ano, a_buffer, sizeof(a_buffer));
}

// ===== Funções de Notificação =====
// Envia o ID do ponto para uma serial
void envia_Ponto(struct Ponto* idponto, char destino[8]) {
    // Obtém um ponteiro para a estrutura
    uint8_t* ptr = reinterpret_cast<uint8_t*>(idponto);
    int structSize = sizeof(struct Ponto);
    Serial.print("structSize: ");
    Serial.println(structSize);
    // Obtém os valores do ponto
    if (strcmp(destino, "Serial") == 0) {
        for (int i=0; i < structSize; ++i) {
            // Envia o byte atual pela porta Serial
            Serial.write(*ptr);
            // Move para o próximo byte na estrutura
            ++ptr;
        }
```

```
        Serial.println();
    } else if (strcmp(destino, "Serial1") == 0) {
        for (int i=0; i < structSize; ++i) {
            // Envia o byte atual pela porta Serial1
            Serial1.write(*ptr);
            Serial.write(*ptr);
            // Move para o próximo byte na estrutura
            ++ptr;
        }
        Serial.println();
        Serial.println("Ponto enviado para Nano.");
    } else {
        Serial.println("Erro no envio do ponto para uma serial...");
    }
}

// Envia a medição (entrada da FIFO) para uma serial
void envia_Msg(struct Msg mens, char destino[8]) {
    //Serial.readStringUntil('\n'); //limpa buffer da Serial
    // Obtém um ponteiro para a estrutura
    uint8_t* ptr = reinterpret_cast<uint8_t*>(&mens);
    int structSize = sizeof(struct Msg);

    if (strcmp(destino, "Serial") == 0) {
        for (int i=0; i < structSize; ++i) {
            // Envia o byte atual pela porta Serial
            Serial.write(*ptr);
            // Move para o próximo byte na estrutura
            ++ptr;
        }
        Serial.println();
    } else if (strcmp(destino, "Serial1") == 0) {
        for (int i=0; i < structSize; ++i) {
            // Envia o byte atual pela porta Serial1
            Serial1.write(*ptr);
            // Move para o próximo byte na estrutura
            ++ptr;
        }
        Serial.println("Dado enviado para Nano.");
    } else {
        Serial.println("Erro no envio da mensagem para uma serial...");
    }
}

// ============ ISR ============
// Handler para interrupção de contagem de pulsos
void pulse(){
```

```
    pulseCount ++;
}

// ==== Métodos da FIFO ====
// Adiciona dados à fila
void enqueue(Msg data) {
    fifo[head] = data;
    // Avança a cabeça e trata o wrap-around
    head = (head + 1) % bufferSize;
    Serial.println("Colocado um elemento na FIFO...");
}

// Remove dados da fila
Msg dequeue() {
    Msg data = fifo[tail];
    // Avança a cauda e trata o wrap-around
    tail = (tail + 1) % bufferSize;
    Serial.println("Retirado um elemento na FIFO...");
  return data;
}

// Verifica se a fila está vazia
bool isEmpty() {
    return head == tail;
}

// ===== Funções Auxiliares =====
// Monitor de tempo
void monitor(char local[32]) {
    Serial.print("*** Tempo atual (");
    Serial.print(local);
    Serial.print(") : ");
    Serial.print(millis());
    Serial.print("; pulseCount: ");
    Serial.print(pulseCount);
    Serial.print("; minuteCount: ");
    Serial.print(minuteCount);
    Serial.print("; minuteFlag: ");
    Serial.print(minuteFlag);
    Serial.print("; usartEventFlag: ");
    Serial.println(usartEventFlag);
}

void printFormattedStruct(struct Msg mens) {
    // Formata e imprime a estrutura Msg
    // no formato "DDMMAAAAHHMMSSddd.ddNNN"
    Serial.write(mens.dateTime.dia[0]);
```

```
    Serial.write(mens.dateTime.dia[1]);
    Serial.write(mens.dateTime.mes[0]);
    Serial.write(mens.dateTime.mes[1]);
    Serial.write(mens.dateTime.ano[0]);
    Serial.write(mens.dateTime.ano[1]);
    Serial.write(mens.dateTime.ano[2]);
    Serial.write(mens.dateTime.ano[3]);
    Serial.write(mens.dateTime.hora[0]);
    Serial.write(mens.dateTime.hora[1]);
    Serial.write(mens.dateTime.minuto[0]);
    Serial.write(mens.dateTime.minuto[1]);
    Serial.write(mens.dateTime.segundo[0]);
    Serial.write(mens.dateTime.segundo[1]);
    Serial.write(mens.consumo[0]);
    Serial.write(mens.consumo[1]);
    Serial.write(mens.consumo[2]);
    //Serial.print(".");
    Serial.write(mensagem.consumo[3]);
    Serial.write(mensagem.consumo[4]);
    Serial.write(mensagem.consumo[5]);
    Serial.write(mensagem.idPonto.ponto);
    imprimirDummyComoPontos(mensagem.dummy, sizeof(mensagem.dummy));
}

void imprimirDummyComoPontos(uint8_t dummy[], size_t tamanho) {
    for (size_t i = 0; i < tamanho; ++i) {
        if (dummy[i] == '\0') {
            // Se o elemento for '\0', imprima um ponto
            Serial.print('.');
        } else {
            // Caso contrário, imprima o valor original
            Serial.print(dummy[i]);
        }
    }
    Serial.println(); // Nova linha no final
}
```

```
//Arquivo de header do firmware do STM32F103C8T6
// stm32_v1.h

#ifndef _STM32_SENSOR_
#define _STM32_SENSOR_

//Sensores
//=====================================================================
//                 Faixa Vazão    Relação
// Modelo   Diâm.     (L/min)       F (Hz)                Unid   Precisão
// -------  -----  -----------  ----------------------  -----  --------
// YF-B10    1"      2 a 50      F =  11 * Q            L/Min    ±5%
// YF-B10    1"      2 a 50      F = (6 * Q - 8) * Q    L/Min    ±5%
// YF-B1    3/4"     2 a 45      F =  11 * Q            L/Min    ±3%
// YF-S201  1/2"     1 a 30      F = 7.5 * Q            L/Min)   ±5%
// ---------------------------------------------------------------------
// Formato medição consumo: XXX.YY
//=====================================================================
// Calcula o consumo por minuto, fazendo a correção
// Fator de Ajuste = Fator de Correção do Sensor * Tempo de Integração
//      450.0      =            7.5              *       60.0

#define SENSORPIN PB9
//#define RX_PIN PA10

// Structs
struct Ponto {
  char ponto[4]; // 4 bytes
};

// RTC do STM32F103C8T6 somente suporta ano no formato YY
struct RTCData {
    uint8_t dia[2];     // 2 bytes
    uint8_t mes[2];     // 2 bytes
    uint8_t ano[4];     // 4 bytes
    uint8_t hora[2];    // 2 bytes
    uint8_t minuto[2];  // 2 bytes
    uint8_t segundo[2]; // 2 bytes
};

struct Msg {
    struct  RTCData dateTime;
    uint8_t consumo[6];   // 6 bytes
    struct  Ponto idPonto; // 3 bytes
    uint8_t dummy[8] = {'\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0'};
};
```

```
// FIFO: declaração
// Tamanho do buffer da fila
const int bufferSize = 128;
// Buffer para armazenar dados na fila
struct Msg fifo[bufferSize];
// Índice da cabeça da fila (onde os dados são adicionados)
int head = 0;
// Índice da cauda da fila (onde os dados são lidos)
int tail = 0;

// Inicializa objetos
// https://github.com/stm32duino/STM32RTC/blob
//          /main/examples/SimpleRTC/SimpleRTC.ino
STM32RTC& rtc = STM32RTC::getInstance();

//Interfaces das funções
void ajustaRTC();
void preencherDadosRTC(struct RTCData *dadosRTC);

void clearIdPonto();
void validaIdPonto();
void putID(int address);

void pulse();

void envia_Ponto(Ponto idponto, char destino[8]);
void envia_Msg(struct Msg mens, char destino[8]);

void enqueue(struct Msg data);
struct Msg dequeue();
bool isEmpty();

void monitor(char local[32]);
void printMsgFormatado(struct Msg mens);
void imprimirDummyComoPontos(uint8_t dummy[], size_t tamanho);

#endif /* _STM32_SENSOR_ */
```

APÊNDICE B – Código do firmware do Arduino Nano

```
//Código do firmware do ARDUINO NANO
//nano.ino
#include <SoftwareSerial.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <string.h>

// Arduino Nano
// Serial: 0 (RX) and 1 (TX).
// Used to receive (RX) and transmit (TX) TTL serial data.
// These pins are connected to the corresponding pins
// of the FTDI USB-to-TTL Serial chip.
SoftwareSerial mySerial(4, 2);

// Instancia o rádio
RF24 radio(9, 10); // CE, CSN

// Define os endereços de comunicação
//const uint64_t rpipe = 0xE0E0F1F1E0LL;
//const uint64_t wpipe = 0xF1F1F0F0E0LL;
const uint64_t enderecoMestre = 0xE0E0F1F1E0LL;
const uint64_t enderecoEscravo = 0xF1F1F0F0E0LL;

// Valiáveis globais
uint8_t dataPacket[32];
const char text[5] = "DATA_";

char point[4];//"NNN"
char sinal[9]; //"DATA_NNN"

//const char noData[] = "NODATA"; // 6 bytes

// Ao invés de enviar o sinal "DATA" será enviado "D"
// para acionar somente um vez, e não 5,
// a IRQ da USART1 no STM32
char  dado_stm_sinal = 'D';
char ponto_stm_sinal = 'P';

// Controle do ritmo de log no código
const int log_count_d = 10000;
// 1seg [100 delay(10)]
const int log_count_p = 1000;
int _log = 0;
int i = 0;

// Limite de Timeout de espera pelo STM32
```

```
int timeout_dado = 10000;// 10 segundos

// Funções acessórias
void getId() {
    int i = 0;
    Serial.println("SOLICITA_POINT");
    // Envia sinal para serial USART1
    mySerial.print(ponto_stm_sinal);

    while (mySerial.available() < 4) {
    //while (!(mySerial.available() > 3)) {
        i++;
        if ((i % log_count_p) == 0) {
            Serial.println("Ainda aguardando o id do ponto...");
            i = 0;
        }
        delay(10);
    }

    // Lê os 4 bytes individualmente e armazena em point
    //point = mySerial.read();
    for (int i = 0; i < 4; i++) {
        point[i] = mySerial.read();
    }
    //point[3] = '\0';

    // Envia uma linha de debug para a USB
    Serial.print("Recebi o ponto: ");
    //Serial.println(point);
    for (int i = 0; i < sizeof(point); i++) {
        Serial.print(point[i]);
    }
    Serial.println();

    // Concatena text com point na variável sinal
    strcpy(sinal, text);
    strcat(sinal, point);
    // Envia uma linha de debug para a USB
    Serial.print("Sinal: ");
    //Serial.println(sinal);
    for (int i = 0; i < sizeof(sinal); i++) {
        Serial.print(sinal[i]);
    }
    Serial.println();
}

void getData() {
```

```
Serial.println("Solicita pacote do smt32...");
// Momento de acionamento da comunicação com o STM32
int tempo_inicial, tempo_final = millis();
int i = 0;
// Envia sinal para serial USART1
mySerial.print(dado_stm_sinal);
while ((mySerial.available() < 32) &&
       ((tempo_final - tempo_inicial) < timeout_dado)) {
    i++;
    if ((i % log_count_d) == 0) {
        Serial.println("Ainda aguardando o dado...");
        i = 0;
    }
    delay(10);
    tempo_final = millis();
}
if ((tempo_final - tempo_inicial) >= timeout_dado) {
    // Copia os dados de noData para dataPacket
    // -1 para excluir o caractere nulo '\0'
    memcpy( dataPacket
          , "NODATA"
          , sizeof("NODATA") - 1
          );
    // Preenche o restante de dataPacket com zeros
    memset( dataPacket + sizeof("NODATA") - 1
          , 0
          , sizeof(dataPacket) - sizeof("NODATA") + 1
          );
    // Loga na serial USB
    Serial.println("NODATA_TIMEOUT");
}
if (mySerial.available() < 32) {
    // Copia os dados de noData para dataPacket
    // -1 para excluir o caractere nulo '\0'
    memcpy(dataPacket, "NODATA", sizeof("NODATA") - 1);
    // Preenche o restante de dataPacket com zeros
    memset( dataPacket + sizeof("NODATA") - 1
          , 0
          , sizeof(dataPacket) - sizeof("NODATA") + 1
          );
    // Loga na serial USB
    Serial.println("NODATA");
} else {
    // Lê os 32 bytes individualmente e armazena em dataPacket
    // dataPacket = mySerial.read();
    for (int i = 0; i < 32; i++) {
        dataPacket[i] = mySerial.read();
```

```
        }
        Serial.print("Pacote recebido: ");
        // Imprime na serial o pacote recebido
        //Serial.println(dataPacket);
        for (int i = 0; i < sizeof(dataPacket); i++) {
            Serial.print(dataPacket[i]);
        }
        Serial.println();
    }
}

// Funções principais
void setup() {
    radio.begin();
    Serial.begin(9600); // Saida pela USB
    mySerial.begin(9600); // Saída pelos pinos TX e RX
    // Atraso no setup para possibilitar o acompanhamento
    delay(5000); // 2 seg
    // Obtem o ID do STM32
    getId();
    // Ajustes do nRF
    radio.setChannel(0x76);
    radio.setRetries(7,4);
    radio.setDataRate(RF24_1MBPS);
    radio.setPALevel(RF24_PA_HIGH);
    //radio.openWritingPipe(wpipe);
    //radio.openReadingPipe(1,rpipe);
    //radio.openWritingPipe(rpipe);
    //radio.openReadingPipe(1,wpipe);
    //radio.openWritingPipe(wpipe);
    //radio.openWritingPipe(wpipe);
    //radio.openReadingPipe(1,rpipe);
    radio.openWritingPipe(enderecoEscravo);
    radio.openReadingPipe(1,enderecoMestre);
    radio.setPayloadSize(32);
    radio.startListening();
    radio.printDetails();
}

void loop() {
    // Aguarda RPi enviar algo
    if(radio.available()) {
        // Obtemos os dados
        radio.read(dataPacket,radio.getDynamicPayloadSize());
        Serial.print("Solicitação recebida:");
        Serial.println((char *)dataPacket);
        Serial.print("Pacote recebido binário:");
```

```
        for (int i = 0; i < sizeof(dataPacket); ++i) {
            Serial.print(dataPacket[i]);
        }
        Serial.println();

        // Parando a recepção
        radio.stopListening();

        // Solicita pacote ao STM32 se o sinal for igual para este ponto
        if (strncmp(sinal, dataPacket, 8) == 0) {
            //cmp("DATA_NNN", "DATA_NNN**************...")
            // Solicita pacote ao stm32 (aguarda até receber o pacote...)
            getData();
            // Aguarda que o RPi esteja pronto para receber radio.available()
            delay(50);
            // Transmitindo para o RPi
            if (radio.write(dataPacket, sizeof(dataPacket))) {
                Serial.println("Enviado com sucesso...");
            } else {
                Serial.println("Erro na transmissao...\n");
            }
            // Pequeno delay para transmitir
            delay(10);
        } else {
            Serial.println("Pacote não aceito...");
        }

        // Ligando a recepção
        radio.startListening();
    }
    delay(10);
}
```

APÊNDICE C – Código do Firmware do Raspberry Pi

```
# Código do Firmware do Raspberry Pi
#rasp.py

# NRF Transmitter Side Code (Raspberry Pi):

import sqlite3
import RPi.GPIO as GPIO
import time
import spidev
import sys
from lib_nrf24 import NRF24  #import NRF24 library

#conectando a um banco de dados existente
banco = sqlite3.connect('banco_dados.db')

## fix python3 printDetails() bug  in  lib_nrf24.py library
## remove extra new line  when is not needed
class bypass_stdout(object):
    def __init__(self,V):
        sysout = sys.stdout
        self.line = V

    def write(self, message):
        self.line+= message


    def flush(self):
        pass


# set the pipe address. this address shoeld be entered on the receiver alo
pipes = [[0xE0, 0xE0, 0xF1, 0xF1, 0xE0], [0xF1, 0xF1, 0xF0, 0xF0, 0xE0]]

radio = NRF24(GPIO, spidev.SpiDev())         #use the gpio pins

# lista que vai receber os números das residencias,
# contido na tabela do banco de dados
listaResid = []


#consultar tabela Unidade_Residencial e jogar valores em uma lista
def acessaTabelaResid():
    #conectando a um banco de dados existente
    # banco = sqlite3.connect('banco_dados.db')
    cursor = banco.execute("SELECT NR_RESID from UNID_RESID")
    for row in cursor:
    listaResid.append(row[0])
```

```
    print (listaResid)

    #banco.close()


#Pegar rcv_mesg, separar em parte e salvar no banco de dados
def inseriNoBanco(dados):
    print ("Inserindo no banco....\n")
    dia = dados[0:2]
    mes = dados[2:4]
    ano = dados[4:8]
    hora = dados[8:10]
    min = dados[10:12]
    seg = dados[12:14]
    consumo = dados[14:20]
    NR = dados[20:23]
    #banco = sqlite3.connect('banco_dados.db')
    cursor = banco.cursor()
    #cursor.execute("BEGIN")
    cursor.execute("INSERT INTO DADOS_CONSUMO
                           (
                             NR_RESID
                           , DIA
                           , MES
                           , ANO
                           , HORA
                           , MINUTO
                           , SEGUNDO
                           , CONSUMO
                           )
                    VALUES (?, ?, ?, ?, ?, ?, ?, ?)",
                    (NR, dia, mes, ano, hora, min, seg, consumo))
    banco.commit()
    print ("Inserido com sucesso!\n")
    cursor.execute("SELECT * FROM DADOS_CONSUMO")
    for row in cursor:
        print (row[0:9])


def setup():
     #consultar tabela Unidade_Residencial e jogar valores em uma lista

    acessaTabelaResid()
    # set the gpio mode
    GPIO.setmode(GPIO.BCM)
    #start the radio and set the ce,csn pin ce= GPIO08, csn= GPIO25
    radio.begin(0, 15)
```

```
    radio.setRetries(7,4)
    #set the payload size as 32 bytes
    radio.setPayloadSize(32)
    # set the channel as 76 hex
    radio.setChannel(0x76)
    #set radio data rate
    radio.setDataRate(NRF24.BR_1MBPS)
    #set PA level
    radio.setPALevel(NRF24.PA_LOW)
    # set acknowledgement as true
    radio.setAutoAck(True)
    radio.openWritingPipe(pipes[0])
    radio.openReadingPipe(1,pipes[1])

    ### fix print details in python3
    if sys.version_info[0] > 2:
        # need to deal with new line
        old_stdout =  sys.stdout
        v = []
        sys.stdout = bypass_stdout(V=v)
        # print basic detals of radio
        radio.printDetails()
        sys.stdout = old_stdout
        print(''.join(v)
                .replace('=\n','= ')
                .replace('\n0x',' 0x')
                .replace('\n\n','\n')
                .replace('= ','= '))
    else:
        radio.printDetails()
    ##########################
    radio.stopListening()


# Rasp vai enviar um sinal no formato "DATA_NNN",
# onde NNN é o id do ponto
def main(args):
    setup()

    while True :
        for unid in listaResid:
            texto = "DATA_" + unid
            print (texto)
            sendMessage = bytes(texto, 'utf-8')
            sendMessage += b''*(32-len(sendMessage))
            #start the time for checking delivery time
            start = time.time()
```

```
            #close radio
            radio.stopListening()
            # just write the message to radio
            radio.write(sendMessage)
            #print a message after succesfull send
            #print("Sent the message: {}".format(sendMessage))

            #Start listening the radio
            radio.startListening()

            while not radio.available():
               time.sleep(1/100)
               if time.time() - start > 2:
                  # print errror message if radio disconnected
                  # or not functioning anymore
                  print("Timed out.")
                  break

            if radio.available():
               print("Recebendo...\n")
               rcv_mesg = []
               #just write the message to radio
               radio.read(rcv_mesg)
               rcv_mesg = bytearray(rcv_mesg).decode('utf-8')
               print (rcv_mesg)
               #print(rcv_mesg[20:23])
               n = rcv_mesg[20:23]
               #print (n)
               if n == unid:
                   inseriNoBanco(rcv_mesg)
               else:
                   print("não é a residencia que eu pedi!\n")
            else:
               print("Radio desligado!\n")

            #give delay of 3 second
            time.sleep(3)

    return 0

if __name__ == '__main__':
    import sys
    sys.exit(main(sys.argv))
    banco.close()
```

APÊNDICE D – Código do Banco de Dados

```
//Código do Banco de Dados
//banco_dados.db

CREATE TABLE UNID_RESID (
      ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL
    , NR_RESID TEXT NOT NULL CHECK(typeof("NR_RESID") = "text"
                                   AND length("NR_RESID") = 3)
    , NM_RESID TEXT NOT NULL CHECK(typeof("NM_RESID") = "text"
                                   AND length("NM_RESID") <= 50)
);

CREATE TABLE sqlite_sequence(name,seq);

CREATE TABLE DADOS_CONSUMO(
      ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL
    , NR_RESID TEXT     NOT NULL CHECK(typeof("NR_RESID") = "text"
                                       AND length("NR_RESID") = 3)
    , DIA INTEGER       NOT NULL
    , MES INTEGER       NOT NULL
    , ANO INTEGER       NOT NULL
    , HORA INTEGER      NOT NULL
    , MINUTO INTEGER    NOT NULL
    , SEGUNDO INTEGER   NOT NULL
    , CONSUMO REAL      NOT NULL
);
CREATE VIEW V_CONSUMO AS
SELECT DC.ID
     , DC.NR_RESID                                          AS NUM_RESID
     , UR.NM_RESID                                          AS NOME_RESID
     , (DC.DIA || '/' || DC.MES || '/' || DC.ANO)          AS DATA
     , (DC.HORA || ':' || DC.MINUTO || ':' || DC.SEGUNDO) AS HORARIO
     , DC.CONSUMO
FROM DADOS_CONSUMO DC LEFT OUTER JOIN UNID_RESID UR
         ON (DC.NR_RESID = UR.NR_RESID);
```

APÊNDICE E – Código da Página WEB (interface com o usuário) para consulta ao banco de dados

```
#Código da Página WEB (interface com o usuário) para consulta ao banco de dados
#app2.py
from flask import Flask, render_template
import sqlite3

app2 = Flask(__name__)
# Configuração para servir arquivos estáticos (CSS)
app2.static_folder = 'static'
# Rota para exibir os dados do banco de dados
@app2.route('/menu')
def mostrar_dados():
    # Conectar ao banco de dados
    conn = sqlite3.connect('..//banco_dados.db')
    cursor = conn.cursor()

    # Executar uma consulta SQL
   #cursor.execute('''SELECT NR_RESID
   #                        , DIA
   #                        , MES
   #                        , ANO
   #                        , HORA
   #                        , MINUTO
   #                        , SEGUNDO
   #                        , CONSUMO
   #                     FROM DADOS_CONSUMO'''
   #               )
    cursor.execute('''SELECT NUM_RESID
                           , NOME_RESID
                           , DATA
                           , HORARIO
                           , CONSUMO
                        FROM V_CONSUMO'''
                   )
    resultados = cursor.fetchall()

    # Consulta para obter as DATAS distintas
    cursor.execute('''SELECT DISTINCT DATA
                        FROM V_CONSUMO''')
    data = cursor.fetchall()

    # Consulta para obter as Nomes de Resid distintas
    cursor.execute('''SELECT DISTINCT NOME_RESID
                        FROM V_CONSUMO''')
    nome = cursor.fetchall()

    # Consulta para obter as Numeros de Resid distintas
    cursor.execute("SELECT DISTINCT NUM_RESID FROM V_CONSUMO")
```

```
    num = cursor.fetchall()

    # Fechar a conexao com o banco de dados
    conn.close()

    # Renderizar um modelo HTML para exibir os dados
    return render_template( 'menu.html'
                          , dados=resultados
                          , DATA=data
                          , NUM_RESID=num
                          , NOME_RESID=nome
                          )

# Rota para processar os dados do formulario de filtragem
@app2.route('/filtrar', methods=['GET'])
def filtrar_dados():
    # Conectar ao banco de dados
    conn = sqlite3.connect('..//banco_dados.db')
    cursor = conn.cursor()
    # Obter o valor selecionado no formulario
    data = request.args.get('DATA')
    # Obter o valor selecionado no formulario
    nome = request.args.get('NOME_RESID')
    # Obter o valor selecionado no formulario
    num = request.args.get('NUM_RESID')

    # Construir a consulta SQL com base no criterio de filtragem
    if data:
        cursor.execute('''SELECT NUM_RESID
                               , NOME_RESID
                               , DATA
                               , HORARIO
                               , CONSUMO
                            FROM V_CONSUMO
                           WHERE DATA = ?'''
                       , (data,)
                       )
    else:
        cursor.execute('''SELECT NUM_RESID
                               , NOME_RESID
                               , DATA
                               , HORARIO
                               , CONSUMO
                            FROM V_CONSUMO'''
                       )

    if nome:
```

```
            cursor.execute('''SELECT NUM_RESID
                                   , NOME_RESID
                                   , DATA
                                   , HORARIO
                                   , CONSUMO
                                FROM V_CONSUMO
                               WHERE NOME_RESID = ?'''
                           , (nome,)
                           )
        else:
            cursor.execute('''SELECT NUM_RESID
                                   , NOME_RESID
                                   , DATA
                                   , HORARIO
                                   , CONSUMO
                                FROM V_CONSUMO'''
                           )

        if num:
            cursor.execute('''SELECT NUM_RESID
                                   , NOME_RESID
                                   , DATA
                                   , HORARIO
                                   , CONSUMO
                                FROM V_CONSUMO
                               WHERE NUM_RESID = ?'''
                           , (num,)
                           )
        else:
            cursor.execute('''SELECT NUM_RESID
                                   , NOME_RESID
                                   , DATA
                                   , HORARIO
                                   , CONSUMO
                                FROM V_CONSUMO'''
                           )

        resultado=cursor.fetchall()

        # Fechar a conexao com o banco de dados
        conn.close()

        # Renderizar a pagina HTML com os dados filtrados
        return render_template('dados_filtrados.html', dados=resultado)

if __name__ == '__main__':
    app2.run(host='0.0.0.0', port=5000)
```

```
<!--Código HTML da página Web  -->
<!--menu.html →

<!DOCTYPE html>
<html>
<head>
    <title>Relatório</title>
    <link rel="stylesheet" type="text/css"
          href="{{ url_for('static', filename='style.css') }}">
      <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f0f0f0;
            margin: 0;
            padding: 0;
        }
        h1 {
            background-color: #333;
            color: #fff;
            text-align: center;
            padding: 20px;
        }
        table {
            width: 80%;
            margin: 20px auto;
            border-collapse: collapse;
            background-color: #fff;
        }
        th, td {
            border: 1px solid #ddd;
            padding: 8px;
            text-align: center;
        }
        th {
            background-color: #333;
            color: #fff;
        }
        tr:nth-child(even) {
            background-color: #f2f2f2;
        }
        a {
            display: block;
            text-align: center;
            background-color: #333;
            color: #fff;
            padding: 10px;
            text-decoration: none;
```

```
        }
      </style>
</head>
<body>
  <!-- Parte Superior da Página (Filtros e Menus) -->
  <div class="top-section">
    <h1>Menu</h1>
    <form method="GET" action="/filtrar">
       <label for="DATA">DATA: De:</label>
       <select name="DATA" id="DATA">
           <option value="">Todas as Datas</option>
           {% for data in DATA %}
               <option value="{{ DATA[0] }}">{{ data[0] }}</option>
           {% endfor %}
           <!-- Adicione mais opçoes conforme necessario -->
       </select>
       <label for="DATA">Até:</label>
       <select name="DATA" id="DATA">
           <option value="">Todas as Datas</option>
           {% for data in DATA %}
               <option value="{{ DATA[0] }}">{{ data[0] }}</option>
           {% endfor %}
           <!-- Adicione mais opçoes conforme necessario -->
       </select>
       <label for="NOME_RESID">  Nome da Residencia:  </label>
       <select name="NOME_RESID" id="NOME_RESID">
           <option value="">Todas as Residencias</option>
           {% for nome in NOME_RESID %}
               <option value="{{ NOME_RESID[] }}">{{ nome[] }}</option>
           {% endfor %}
           <!-- Adicione mais opçoes conforme necessario -->
       </select>
       <label for="NUM_RESID">  Nº da Residencia:  </label>
       <select name="NUM_RESID" id="NUM_RESID">
           <option value="">Todas as Residencias</option>
           {% for num in NUM_RESID %}
               <option value="{{ NUM_RESID[] }}">{{ num[] }}</option>
           {% endfor %}
           <!-- Adicione mais opçoes conforme necessario -->
       </select>
       <input type="submit" value="Filtrar">
    </form>
  </div>
  <!-- Parte Inferior da Página (Dados Filtrados) -->
  <div class="bottom-section">
    <h1>Dados Filtrados</h1>
    <table border="1">
```

```
        <tr align="center" bgcolor="#9966ff">
            <th style="width:200px">Número da Residência</th>
            <th style="width:200px">Nome da Residência</th>
            <th style="width:200px">Data</th>
            <th style="width:200px">Horário</th>
            <th style="width:200px">Consumo</th>
        </tr>
        {% for linha in dados %}
        <tr align="center"  bgcolor="#ddccff">
            <td>{{linha[0]}}</td>
            <td>{{linha[1]}}</td>
            <td>{{linha[2]}}</td>
            <td>{{linha[3]}}</td>
            <td>{{linha[4]}}</td>
        </tr>
        {% endfor %}
    </table>
    <a href="/filtrar">Filtrar Dados</a>
  </div>
</body>
</html>

<!--Código HTML dos dados filtrados  -->
<!--dados_filtrados.html-->

<!DOCTYPE html>
<html>
<head>
    <title>Dados Filtrados</title>
      <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f0f0f0;
            margin: 0;
            padding: 0;
        }
        h1 {
            background-color: #333;
            color: #fff;
            text-align: center;
            padding: 20px;
        }
        table {
            width: 80%;
            margin: 20px auto;
            border-collapse: collapse;
            background-color: #fff;
```

```
        }
        th, td {
            border: 1px solid #ddd;
            padding: 8px;
            text-align: center;
        }
        th {
            background-color: #333;
            color: #fff;
        }
        tr:nth-child(even) {
            background-color: #f2f2f2;
        }
        a {
            display: block;
            text-align: center;
            background-color: #333;
            color: #fff;
            padding: 10px;
            text-decoration: none;
        }
      </style>
</head>
<body>
    <h1>Dados Filtrados</h1>
    <table border="1">
        <tr align="center" bgcolor="#9966ff">
            <th style="width:200px">Número da Residência</th>
            <th style="width:200px">Nome da Residência</th>
            <th style="width:200px">Data</th>
            <th style="width:200px">Horário</th>
            <th style="width:200px">Consumo</th>
        </tr>
        {% for linha in dados %}
        <tr align="center"  bgcolor="#ddccff">
            <td>{{linha[0]}}</td>
            <td>{{linha[1]}}</td>
            <td>{{linha[2]}}</td>
            <td>{{linha[3]}}</td>
            <td>{{linha[4]}}</td>
        </tr>
        {% endfor %}
    </table>
</body>
</html>
```

APÊNDICE F – Ambientes de Desenvolvimento utilizados no projeto

Seguem os Ambientes de Desenvolvimento utilizados no projeto.

```
Arduino IDE
    Arduino IDE v1.8.19 instalado como portable

STM32
    Packages: [
        Fornecedor": "STMicroelectronics",
        websiteURL": "https://github.com/stm32duino",
        maintainer": "STMicroelectronics",
        email": "stmduino@st.com",
        help online": "http://www.stm32duino.com/"
        platforms": [
            name": "STM32 MCU based boards",
            architecture": "stm32",
            version": "2.0.0",
            category": "Contributed",
            url": "https://github.com/stm32duino/Arduino_Core_STM32/
                        releases/download/2.0.0/STM32-2.0.0.tar.bz2",
            archiveFileName": "STM32-2.0.0.tar.bz2",

    Bibliotecas:

        name=STM32duino RTC
        version=1.4.0
        author=STMicroelectronics
        maintainer=stm32duino
        sentence=Allows to use the RTC functionalities of STM32 based boards.
        paragraph=With this library you can use the RTC peripheral in
                    order to program actions related to date and time.
        category=Timing
        url=https://github.com/stm32duino/STM32RTC.git
        architectures=stm32

        name=FlashStorage_STM32
        version=1.2.0
        author=Khoi Hoang
        maintainer=Khoi Hoang <khoih.prog@gmail.com>
        license=MIT
        sentence=The FlashStorage_STM32 library aims to provide a
                    convenient way to store and retrieve user data using
                    the non-volatile flash memory of STM32F/L/H/G/WB/MP1.
                    It is using the buffered read and write to minimize
                    the access to Flash. It now supports writing and
                    reading the whole object, not just byte-and-byte.
                    New STM32 core v2.0.0+ is also supported now.
        paragraph=Useful if the EEPROM is not available or too small.
```

```
                    Currently, STM32F/L/H/G/WB/MP1 are supported.
        url=https://github.com/khoih-prog/FlashStorage_STM32
        architectures=stm32,ststm32
        category=Data Storage
        includes=FlashStorage_STM32.h,FlashStorage_STM32.hpp
```

Arduino Nano:

Biblioteca:

```
        name=RF24
        version=1.4.8
        author=TMRh20
        maintainer=TMRh20,Avamander
        sentence=Radio driver, OSI layer 2 library for nrf24L01(+) modules.
        paragraph=Core library for nRF24L01(+) communication.
                    Simple to use for beginners, but offers advanced
                    configuration options. Many examples are included
                    to demonstrate various modes of communication.
        category=Communication
        url=https://nRF24.github.io/RF24/
        architectures=*
```

RPi

Sistema Operacional:

```
        @raspberrypi:~ $ uname -a
        #1613 SMP Thu Jan 5 11:59:48 GMT 2023 armv7l GNU/Linux
        Linux raspberrypi 5.15.84-v7+

        @raspberrypi:~ $ hostnamectl
           Static hostname: raspberrypi
                 Icon name: computer
                Machine ID: 9fa3aa9cd7c341a688bccc7bcb9fdd34
                   Boot ID: 2cd174d4c65144fb9338afe2da93caf1
          Operating System: Raspbian GNU/Linux 11 (bullseye)
                    Kernel: Linux 5.15.84-v7+
              Architecture: arm
        @raspberrypi:~ $ cat /etc/os-release
        PRETTY_NAME="Raspbian GNU/Linux 11 (bullseye)"
        NAME="Raspbian GNU/Linux"
        VERSION_ID="11"
        VERSION="11 (bullseye)"
        VERSION_CODENAME=bullseye
        ID=raspbian
        ID_LIKE=debian
        HOME_URL="http://www.raspbian.org/"
```

```
        SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
        BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
        @raspberrypi:~ $ sudo lsb_release -a
        No LSB modules are available.
        Distributor ID: Raspbian
        Description:    Raspbian GNU/Linux 11 (bullseye)
        Release:        11
        Codename:       bullseye
```

Python 3.9.2

```
# This file lib_nrf24.py is a slightly tweaked version of Barraca's "pynrf24".
lib_nrf24
```

SQLite

Sqlite 3: nativo do Python

Web

Python 3.9

```
Name: Flask
Version: 3.0.0
Summary: A simple framework for building complex web applications.
Maintainer-email: Pallets <contact@palletsprojects.com>
Requires-Python: >=3.8
Requires-Dist: Werkzeug>=3.0.0
Requires-Dist: Jinja2>=3.1.2
Requires-Dist: itsdangerous>=2.1.2
Requires-Dist: click>=8.1.3
Requires-Dist: blinker>=1.6.2
Requires-Dist: importlib-metadata>=3.6.0; python_version < '3.10'
Requires-Dist: asgiref>=3.2 ; extra == "async"
Requires-Dist: python-dotenv ; extra == "dotenv"
```