PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

LEONARDO ZANETTI MACHADO

Proposta de solução para o problema de conectividade no ambiente rural a partir de ensaio com a tecnologia LoRa

CAMPINAS

2021

pontifícia universidade católica de campinas

centro de ciências exatas, ambientais e de tecnologias

faculdade de engenharia elétrica

leonardo zanetti machado

Proposta de solução para o problema de conectividade no ambiente rural a partir de ensaio com a tecnologia LoRa

Trabalho de Conclusão de Curso apresentado como exigência para graduação no curso de engenharia elétrica, do Centro de Ciências Exatas, Ambientais e de Tecnologias, da Pontifícia Universidade Católica de Campinas.

Orientador: Prof. Dr. Frank Herman Behrens

PUC - CAMPINAS

2021

PONTIFícia universidade católica de campinas

CENTRO DE CIÊNCIAS EXATAS, AMBIENTAIS E DE TECNOLOGIAS

FACULDADE DE ENGENHARIA ELÉTRICA

LEONARDO ZANETTI MACHADO

Proposta de solução para o problema de conectividade no ambiente rural a partir de ensaio com a tecnologia LoRa

Dissertação defendida e aprovada em <colocar data> pela comissão examinadora:

Prof. Dr. Frank Herman Behrens

Orientador e presidente da comissão examinadora.

Pontifícia Universidade Católica de Campinas

<Adicionar Examinador 1 aqui>

<Adicionar Examinador 2 aqui>

Campinas

2021

Dedico este trabalho de conclusão de curso aos meus pais por terem moldado o ser humano que me tornei a exemplo de pessoas tão corretas como eles. Dedico em especial ao meu pai, por ser uma das pessoas mais incríveis que já conheci e por ser a minha grande inspiração.

AGRADECIMENTOS

**A Deus,**

Por ter sempre me guiado ao longo de toda a minha vida, iluminando meus caminhos, me dando força e coragem para seguir em frente mesmo durante os momentos mais difíceis.

**Ao meu pai (*em memória)*,**

Por sempre me inspirar a bondade e a benevolência, a simplicidade e a atenção ao que realmente importa; por incentivar sempre o meu interesse pela ciência e me apoiar em todos os projetos realizados. Por todos os momentos e conversas que tivemos juntos, por me fazer gostar de café, futebol, música boa e carros antigos.

**À minha mãe,**

Por todo o suporte, carinho, afeto e por ter me ensinado a ser uma pessoa forte, batalhadora, corajosa e que sempre enfrenta os desafios, por maiores que sejam, com a cabeça erguida. Por sempre me incentivar a correr atrás dos meus sonhos e me prover os pilares da educação para a minha formação.

**Aos primos, tios, avós e padrinhos,**

Por todo o carinho e suporte, principalmente nos momentos mais difíceis, por todos os ensinamentos que ajudaram a moldar a pessoa que me tornei.

**À minha namorada,**

Por todo o suporte afetivo e incentivo durante a faculdade, por sempre estar comigo e me apoiar nas minhas decisões e por todos os momentos bons compartilhados em todos esses anos juntos.

**Ao Professor Frank Herman Behrens,**

Por todo o conhecimento compartilhado durante os anos da faculdade em suas disciplinas e toda a orientação durante a realização do trabalho, além de todas as ideias agregadas ao projeto.

**Ao Professor Antonio Demanboro,**

Por ter sido meu orientador no projeto de Iniciação Científica, responsável por despertar meu interesse na área de comunicação sem fio e radiofrequência.

**Aos meus melhores amigos Matheus Bueno, Júlia Grannier e Gabriel Rosa,**

Por sempre estarem comigo, por todo o apoio, conversas e momentos juntos.

**Aos meus colegas da faculdade,**

Por todos os momentos e conhecimentos compartilhados.

**Ao Centro de Pesquisa e Desenvolvimento (CPQD),**

Pela oportunidade de estágio, proporcionando o maior desenvolvimento técnico da minha carreira e por todas as pessoas que foram colocadas em meu caminho.

“A vida é aquilo que acontece enquanto você está ocupado fazendo outros planos.”

John Lennon (1940 – 1980)

resumo

O trabalho realizado tem como finalidade a elaboração de um projeto para a conclusão do curso de bacharelado em Engenharia Elétrica, baseado nas tecnologias de sensores sem fio acopladas ao âmbito do agronegócio. A partir do uso de comunicação via radiofrequência, a solução proposta tem como objetivo promover um sistema de monitoramento para pequenos e médios produtores, proprietários de granjas e armentos, através de uma plataforma de gerenciamento de dados, exibindo ao usuário as diversas variáveis críticas que precisam ser analisadas para uma maior eficiência no cultivo dos produtos a partir da união entre agricultura de precisão, sistemas embarcados e a ferramenta Django.

Palavras-chave: LoRa, LPWAN, Agronegócio, IoT, Django.

abstract

The work carried out has as purpose the elaboration of a project for the conclusion of the bachelor's degree in Electrical Engineering, based on wireless sensor technologies coupled to the agribusiness scope. From the use of radio frequency communication, the proposed solution aims to promote a monitoring system for small and medium producers, owners of farms through a data management platform, showing to the user the various critical variables that need to be analysed for greater efficiency in the cultivation of products through de combination of precision agriculture, embed systems and the Django framework.

Keywords: LoRa, LPWAN, Agrobusiness, IoT, Django.

lista de figuras

Figura 1 - Valor bruto da Produção Agropecuária. 16

Figura 2 - Imagem do “DSKY”. 18

Figura 3 - LGM-30 Minuteman. 19

Figura 4 - Arduino. 21

Figura 5 - ESP-32. 21

Figura 6 - ESP-32 com conectividade de rádio. 22

Figura 7 – Data-bus de 8 bits. 24

Figura 8 – Interface serial. 25

Figura 9 – Processo de modulação em AM. 26

Figura 10 – Processo de modulação em FM. 27

Figura 11 – Processo de modulação em ASK. 28

Figura 12 – Processo de modulação em FSK. 29

Figura 13 – Modulação a partir da CSS. 30

Figura 14 – Gráfico temporal de fator de espalhamento. 30

Figura 15 – Ilustração da técnica de Polling. 32

Figura 16 – Ilustração da técnica de Round Robin. 33

Figura 17 – Topologia broadcast. 34

Figura 18 – Topologia multicast. 35

Figura 19 – Topologia unicast. 35

Figura 20 – Arquitetura do projeto. 37

Figura 21 – Sensor DHT22. 39

Figura 22 – Sensor de umidade do solo. 39

Figura 23 – Arduino IDE. 40

Figura 24 – ESP-32 LoRa e DHT22 em operação. 41

Figura 25 – Código para leitura dos valores. 41

Figura 26 – Empacotamento dos valores de temperatura e umidade. 43

Figura 27 – Parte do código executada em looping. 44

Figura 28 – Módulo de decodificação dos dados. 44

Figura 29 – Leitura da entrada de valores com Python. 45

Figura 30 – Representação do código de uma página web. 47

Figura 31 – Função de inserção dos primeiros dados. 48

Figura 32 – Função de atualização incremental dos dados. 48

Figura 33 – Gráfico para valores de temperatura. 49

Figura 34 – Gráfico para valores de umidade. 49

lista de gráficos

lista de tabelas

Tabela 1 – Faixa de frequências LoRa de acordo com as principais regiões. 31

Tabela 2 – Especificações do da placa ESP-32 LoRa. 38

Tabela 3 – Especificações do transceptor LoRa. 38

Tabela 4 – Especificações do sensor DHT22. 39

Tabela 5 – Especificações do sensor de umidade do solo. 39

Tabela 6 – Número de componentes utilizados. 40

lista de abreviações

|  |  |
| --- | --- |
| IoT | *Internet of Things* |
| WAN | *Wide Area Network* |
| CSS | *Chirp Spread Spectrum* |
| SF | *Spreading Factor* |
| UR | *Umidade Relativa* |
| IDE | *Integrated Development Envionment* |
| COM | *Component Object Model* |
| UTF | *Unicode Transformation Format* |
| HTML | *HyperText Markup Language* |
| JSON | *JavaScript Object Notation* |
| TX | *Transmissão* |
| RX | *Recepção* |

sumário

1. INTRODUÇÃO
   1. Agricultura no Brasil

A agricultura é definida pelo conjunto de técnicas de cultivo da terra para a obtenção de produtos e garantia da subsistência alimentar do ser humano e, também, a exploração de matérias-primas para a produção de combustíveis, medicamentos, ferramentas e roupas. Historicamente, a atividade agrícola teve início há mais de 12 mil anos, durante o período neolítico com a formação das primeiras civilizações. Além do plantio de sementes para o consumo, o homem passa a explorar a domesticação e criação de animais, surgindo então, a pecuária. Com o desenvolvimento das práticas e costumes, surge, também, a consolidação do escambo envolvendo produtos cultivados e até mesmo a troca de animais por outras mercadorias (ainda sem o uso da moeda), o que indica o começo de um valor econômico relacionado às atividades rurais.

No Brasil, as principais atividades agrárias se iniciam no século XVI, com as Capitanias Hereditárias, que foram um sistema administrativo implementado pela Coroa portuguesa no Brasil nos anos 1500. Esse sistema foi caracterizado pela divisão do território brasileiro pertencente a Portugal, definido no Tratado de Tordesilhas, que tinha como finalidade a exploração territorial e econômica de cada região, utilizando, principalmente, práticas agrícolas de monocultura com mão-de-obra escrava e grandes latifúndios.

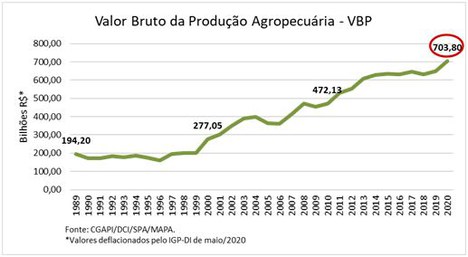
A partir do desdobramento do sistema implementado, a produção passa a ter grandes avanços e novas opções. Juntamente com a exploração de minérios, durante o século XVIII, novos tipos de sementes e outros tipos de vegetais são adicionados no leque do cultivo. Nesse momento, já existe um importante interesse comercial relacionado às atividades agrícolas, indicando sua potencialidade econômica para o país que, durante o século XIX, tem a cafeicultura como principal fonte de insumos para exportação.

Essa linha do tempo conclui-se atualmente mostrando o grande e importante papel da agricultura na economia do Brasil, que possui uma vasta extensão territorial com solos muito férteis e que possibilitam diversos tipos de cultivo, aumentando ainda mais as oportunidades econômicas.

Nos dias atuais, o Brasil é caracterizado por uma grande diversidade de produtos cultivados nas diferentes regiões. Por exemplo, na região Sul, que possui descendência de imigrantes europeus, cultiva-se principalmente sementes como soja e milho assim como cana-de-açúcar e algodão. O setor pecuário, por sua vez, é predominantemente preenchido pela cultura de aves e suínos.

Segundo a Embrapa, até o mês de setembro do ano de 2020, o Brasil produziu 124,845 milhões de toneladas de soja, em uma área plantada de 36,950 milhões de hectares. Assim como indicado pelos números expressivos, o seguinte gráfico ilustra o processo de crescimento no valor de produção agropecuária no Brasil desde 1989 até o ano de 2020.

Figura 1 - Valor bruto da Produção Agropecuária.



Fonte: EMBRAPA, 2020.

A partir da análise do gráfico presente na imagem, pode-se observar que é notável a participação do setor agrônomo como grande fonte de renda para o país e também o coloca entre as principais nações exportadoras de produtos agrícolas.

Devido à grande importância do setor agropecuário no Brasil, juntamente com a transformação das técnicas e modernização das atividades, o campo também sofreu mudanças positivas em relação aos métodos de cultivo e às ferramentas utilizadas são cada vez mais tecnológicas para que seja atingida maior eficiência na produção.

* 1. Revoluções tecnológicas

A história é marcada por diversas revoluções e, entre as mais importantes, estão as Revolução Industriais. A primeira, iniciada entre os séculos XVIII e XIX, foi marcada pela invenção das máquinas a vapor e, com isso, a utilização das vias férreas cujas máquinas eram movidas a esse tipo de energia recém-descoberto. Em seguida, já no século XX, tem-se a segunda revolução industrial, caracterizada pela implementação do modelo de linhas de produção em massa desenvolvido por Henry Ford.

Na segunda metade do século XIX, é iniciada a terceira revolução industrial, representada pelo surgimento da internet, possibilitando a conexão entre dispositivos, através do tráfego de dados. Em seguida, é estabelecida a 4ª e atual revolução industrial, que tem a premissa de representar uma nova tendência no cenário industrial com a introdução de dispositivos mais inteligentes e que podem ser interconectados, incluindo aparelhos que, essencialmente, não foram produzidos para isso. A esse cenário, dá-se o nome de “Internet das Coisas” (*Internet of Things*).

* 1. Sistemas Embarcados

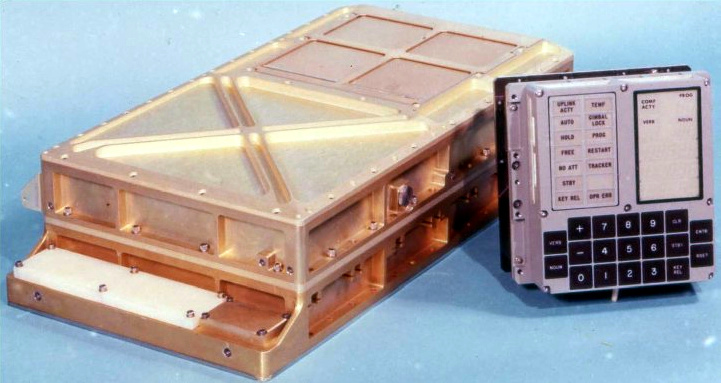
Paralelamente às revoluções industriais presenciadas pela humanidade, desde o surgimento dos primeiros computadores, acompanha-se o desdobramento de uma evolução tecnológica extremamente veloz, contando com dispositivos cada vez mais potentes e versáteis. Entretanto, no início da descoberta desses instrumentos, em 1940, os computadores digitais ainda tinham um grande tamanho, portanto não poderiam ser transportados com facilidade e nem utilizados de maneira remota. Nesse contexto, destaca-se a importância dos sistemas de processamento encapsulado, que promovem mobilidade e praticidade aos processos computacionais.

* + 1. Apollo Guidance Computer

É reconhecido que, em 1965, ao desenvolver um sistema de guia para o projeto Apollo, Charles Stark Draper, no MIT, acaba criando o primeiro sistema embarcado, que se tratava de um guia de movimentação, orientação e controle (computador de bordo), operando em tempo real, o qual recebeu o nome de *Apollo Guidance Computer*. Para que tivesse seu tamanho e peso reduzidos, além de ser móvel, o dispositivo contava com o uso de circuitos integrados, que são circuitos eletrônicos de tamanho reduzido, compostos por materiais elétricos (principalmente transistores) encapsulados, sendo um dos primeiros a fazer uso de tal tecnologia ainda recente na época.

O sistema contava com portas lógicas e suas combinações, como os *flip-flops*, que eram responsáveis por realizar as operações lógicas e, assim, possibilitar o controle de partes do módulo lunar via interface de usuário, a qual acredita-se que seja uma das primeiras da história, a partir de um teclado e um *display*, denominado *“DSKY”* (Fig. 2).

Figura 2 - Imagem do “DSKY”.



Fonte: Wikipedia, 2018.

* + 1. LGM-30 Minuteman

O míssil *LGM-30 Minuteman* (Fig. 3) é um projeto balístico nuclear norte-americano e foi lançado em 1961, marcando a data em que os sistemas embarcados começam a ser maciçamente produzidos. Além do processamento modular, o sistema contava com um disco rígido para o armazenamento das informações. Com a evolução e descobertas acerca dos operadores lógicos, também foram implementadas novas alternativas de portas lógicas como o NAND, o que corroborou para o seu barateamento, diminuindo os custos de produção dos sistemas embarcados até chegar no estado em que estão atualmente.

Figura 3 - LGM-30 Minuteman.



Fonte: MISSILETHREAT, 2018.

Com o aumento da produção e utilização dos sistemas embarcados, seu preço vem diminuindo desde a década de 60 e seu poder de processamento aumentando expressivamente. Para regulamentar o processo de produção e uso dos dispositivos, em 1978, é lançada pela associação americana *National Electrical Manufacturers Association* (criada em 1926) uma norma referente aos controladores programáveis, que trata de uma padronização para o uso de produtos elétricos e eletrônicos.

Com a difusão dos sistemas embarcados, os componentes eletrônicos externos, antes utilizados de forma modular e separada, foram incorporados pelos processadores dos sistemas, trazendo assim, maior redução em seu tamanho e uma maior capacidade de processamento para a realização de um número maior de tarefas simultaneamente.

Além do tamanho reduzido e grande poder, com a evolução da eletrônica e o surgimento dos sistemas digitais, alguns componentes mais caros puderam ser substituídos por circuitos integrados que têm a capacidade de desempenhar a mesma função, o que contribuiu para uma redução ainda maior em seu preço, tornando-os cada vez mais acessíveis.

Atualmente, os sistemas embarcados são parte fundamental do panorama tecnológico, que contribuem para a dição do rumo em que os avanços tecnológicos caminham. Através deles, é possível entender como acontece a 4ª Revolução Industrial, também intitulada de surgimento da Indústria 4.0.

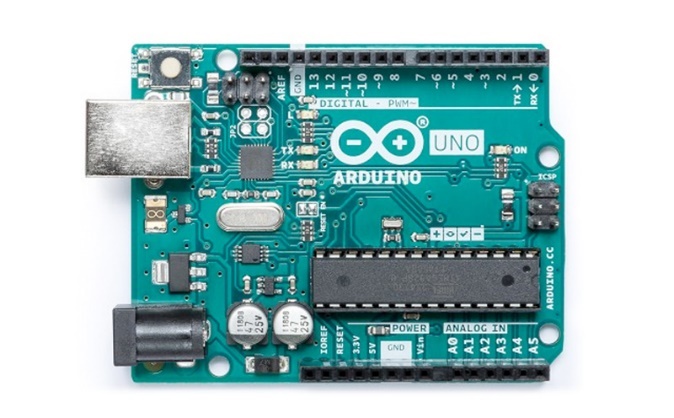
* 1. Internet das Coisas

A partir do avanço da microeletrônica, do surgimento dos diversos tipos de sensores, do desenvolvimento de protocolos de comunicação de dados *online* e *offline*, bem como da evolução dos sistemas, é possível entender o significado do termo “Internet das Coisas” (*Internet of Things*), utilizado para caracterizar o atual momento tecnológico.

“É a rede de objetos físicos, sistemas, plataformas e aplicativos com tecnologia embarcada para comunicar, sentir ou interagir com ambientes internos e externos. Permite que as “coisas” interajam umas com outras e que tomada de decisões sejam feitas. A internet das coisas é a base da Indústria 4.0 (HERMANN, PENTEK, OTTO, 2015).”

Os sistemas embarcados têm protagonismo nessa área, porque possibilitam, a partir de suas interfaces, conectar dispositivos como sensores e atuadores a objetos e aparelhos que, inicialmente, não possuem conexão, para inúmeras finalidades, como o monitoramento de temperatura, pressão, umidade do ar e até mesmo o controle de motores através da internet. Além da versatilidade, os dispositivos são caracterizados pelo tamanho reduzido e por seu alto poder de processamento de dados. Um exemplo comum e bastante popular é o Arduino, uma placa de desenvolvimento para aplicações em IoT.

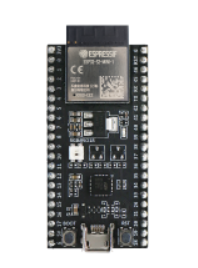
Figura 4 - Arduino.



Fonte: GOOGLE, 2021.

Além dos mais simples, existem aparelhos voltados à Internet das Coisas que possuem especificações ainda mais robustas, como é o caso dos modelos produzidos pela empresa *Espressif Systems*, que não só conta com o processamento, mas também a possibilidade de tráfego de dados via internet com o modelo ESP-32 (Fig. 5) e, até mesmo, radiofrequência, com o modelo ESP-32 LoRa (Fig. 6), desenvolvido por empresas como a *Heltec* a partir do modelo inicial da *Espressif*.

Figura 5 - ESP-32.



Fonte: ESPRESSIF, 2021.

Figura 6 - ESP-32 com conectividade de rádio.



Fonte: MERCADO LIVRE, 2021.

A partir da utilização de dispositivos como esses, é possível a criação de redes de comunicação entre unidades de processamento que monitoram diversos tipos de ambientes.

1. FUNDAMENTAÇÃO TEÓRICA

Através da introdução, pode-se entender o escopo geral do trabalho, apresentando os principais pontos de forma abrangente. Assim, para que o objetivo do projeto possa ser compreendido, faz-se necessário o entendimento específico de cada uma de suas partes.

* 1. O que é a agricultura de precisão

Por definição, o termo “Agricultura de precisão” remete à “utilização de aparelhagem tecnológica avançada para avaliar e acompanhar de maneira mais precisa as condições das áreas de atividades agronômicas baseada no princípio da variabilidade do solo e clima” (WIKIPEDIA, 2020). Entre outras palavras, está ligada ao monitoramento refinado das diversas variáveis críticas e exploração de dados relacionadas ao cultivo dos produtos rurais e, para isso, utiliza-se de diversas opções presentes no mercado.

De acordo com o “Manual de Segurança e Qualidade para a Cultura da Soja”, disponibilizado pela Embrapa, a soja, principal semente cultivada no Brasil, requer valores específicos relacionados ao clima para que se tenha o rendimento esperado em sua colheita. Por exemplo, “a faixa de temperatura ideal para o desenvolvimento da soja está entre 20ºC e 30ºC. Sob temperaturas menores ou iguais a 10ºC ou temperaturas acima de 40ºC, a soja sofre redução de crescimento ou distúrbios na floração e diminuição na capacidade de retenção de vagens” (EMBRAPA, 2005).

Outro grande integrante do grupo de maiores produtos presentes na economia brasileira é o café que, por exemplo, passa por um processo de secagem antes de ser torrado e comercializado. Nesse processo, a umidade do café deve atingir valores entre 11% e 12% de acordo com a região em que é cultivado, a fim de que esteja preparado para a torra e, posteriormente, para sua comercialização. Com isso, entende-se que é necessário um controle preciso e refinado sobre a umidade do local em que o café passa por esse processo, para que a qualidade seja atingida de acordo com o esperado pelo produtor.

Além das condições climáticas, há a preocupação em relação ao tipo de solo, sua fertilidade e, também, o manejo correto do cultivo, juntamente com outros fatores de extrema importância, os quais necessitam ser avaliados pelo produtor para que, por sua vez, tome as decisões adequadas a fim de que obtenha o maior rendimento possível.

Para isso, são utilizadas tecnologias recentes, contando com sensores analógicos e digitais, microcontroladores capazes de processar os dados de monitoramento e, até mesmo, transceptores que são responsáveis pelo envio dos dados processados de maneira sem fio, gerando uma versatilidade muito grande para a aplicação.

* 1. Comunicação de dados

A comunicação de dados representa o transporte de valores entre dispositivos eletrônicos, permitindo que a informação seja veiculada de um ponto de origem a um ponto de destino. Esse processo pode acontecer tanto em meios confinados, como ambientes cabeados (computadores, por exemplo) nos quais os dados são trafegados a partir da corrente elétrica, quanto em meios não-confinados, como o ar, no qual os dados são trafegados através de ondas eletromagnéticas, portanto sem a necessidade de conexão física (cabeamento).

* + 1. Comunicação em meios confinados

A tecnologia dos sistemas embarcados demanda que os circuitos interligados possam trocar informações entre si através de canais confinados ou barramentos e para que isso seja alcançado de forma íntegra e eficiente, é necessária a definição de um conjunto de regras e procedimentos que controlem todo esse processo, que são os chamados protocolos de comunicação.

Em meios confinados, os protocolos de comunicação podem ser organizados em 2 categorias: comunicação paralela e comunicação serial.

* + - 1. Comunicação Paralela

Esse tipo de comunicação é realizado através da transmissão simultânea de maneira rápida, a partir de “ondas” de *bits*. Entretanto, são necessárias múltiplas conexões no barramento com 8, 16, ou mais fios conectados, formando os chamados *“data-buses*”, que carregam todo o conteúdo da informação de uma vez.

Figura 7 – Data-bus de 8 bits.

Uma imagem contendo Gráfico

Descrição gerada automaticamente

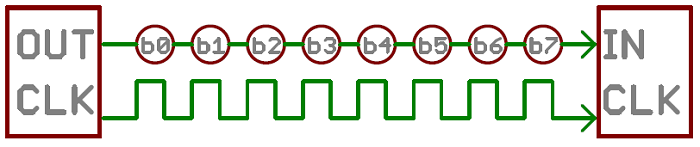
Fonte: SPARKFUN.

A Figura 13 mostra um *data-bus* de 8 *bits*, que envia 8 informações a cada pulso de *clock*, utilizando 8 fios para a transmissão dos dados e 1 fio para a contagem de tempo.

* + - 1. Comunicação Serial

Esse tipo de comunicação, diferentemente da anterior, utiliza a transmissão de informações através do envio sequencial de dados, um por vez. Ao mesmo tempo em que o resultado obtido é uma comunicação mais lenta em relação à paralela, usa-se muito menos fios nas conexões. A Figura 14 mostra a interface de uma comunicação serial, transmitindo um *bit* a cada pulso de *clock*, utilizando apenas 2 fios.

Figura 8 – Interface serial.



Fonte: SPARKFUN.

* + 1. Comunicação em meios não confinados

Para que os dados possam ser transportados de maneira remota, sem fio, é necessário que eles passem por um processo chamado de “modulação”, que codifica os valores binários para a transmissão de forma que possam ser decodificados em sua recepção.

A modulação ocorre na modificação de parâmetros de uma forma de onda eletromagnética e pode ser feita de forma analógica, para a transmissão de sinais analógicos, como a voz (exemplo: telefone antigo) e pode ser feita de forma digital, para a transmissão de sinais digitais, os *bits* (exemplo: celulares). Assim que o espectro de dados é definido, utiliza-se uma onda chamada de Portadora que tem a função de adequar o sinal ao espectro de frequência na qual será transmitido.

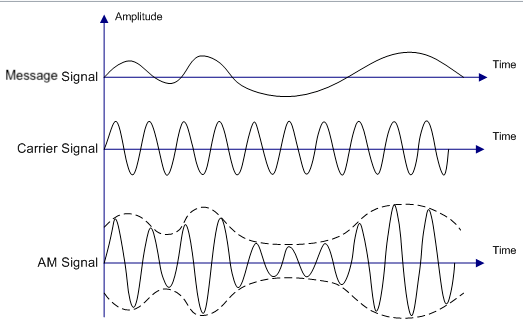
* + - 1. Modulações Analógicas

As modulações analógicas foram as primeiras a ser utilizadas na transmissão de dados, de acordo com os recursos que eram disponíveis. Dois exemplos muito utilizados são a modulação AM e FM.

* + - * 1. Modulação AM (Amplitude Modulation)

A modulação AM é feita a partir da alteração da amplitude de um sinal eletromagnético de acordo com o sinal que carrega os valores que serão transmitidos (*Message Signal*), gerando padrões que podem ser identificados na recepção e “demodulados”. Então, o sinal da onda portadora (*Carrier Signal*) é encapsulado a partir do sinal da mensagem e transmitida. A Figura 7 ilustra o processo de modulação.

Figura 9 – Processo de modulação em AM.



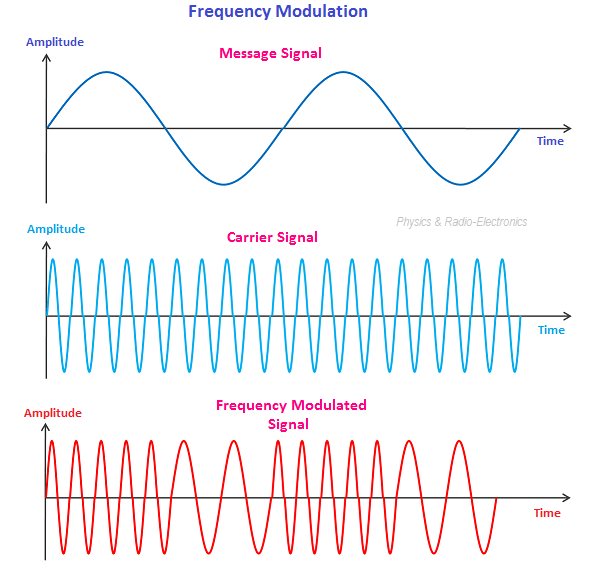
Fonte: WIKIPÉDIA, 2021 (adaptado).

Por se tratar de uma modificação na amplitude do sinal, a modulação em AM tem uma limitação muito grande, pois é bastante suscetível a interferências (ruídos).

* + - * 1. Modulação FM (Frequency Modulation)

A modulação FM, por sua vez, é realizada na modificação da frequência do sinal de acordo com a mensagem a ser transmitida. Com a variação dos valores de amplitude do sinal, a frequência é aumentada ou diminuída, como mostra a Figura 8.

Figura 10 – Processo de modulação em FM.



Fonte: PHYSICS AND RADIO ELECTRONICS, 2021.

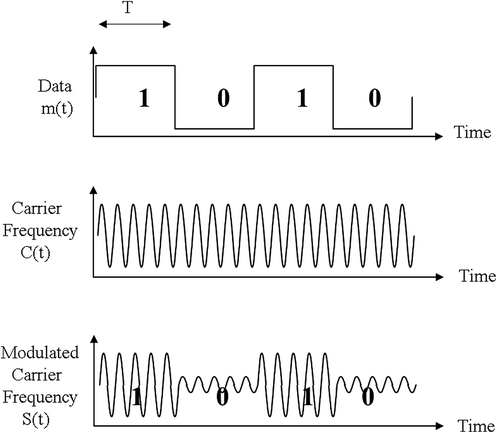
* + - 1. Modulações Digitais

Com o desenvolvimento tecnológico, novas possibilidades surgem e passam a ser utilizadas as modulações digitais que tratam *bits*, no lugar de valores analógicos, também modificando parâmetros de ondas eletromagnéticas. Dois exemplos são ASK e FSK, porém existem outros métodos e, até mesmo, combinações de métodos, como o QAM, que utilizam a união entre modulações em amplitude e fase.

* + - * 1. Modulação ASK (Amplitude Shift-Keying)

A modulação digital de amplitude trata de um chaveamento (por isso a origem do nome “*shift*”) entre valores binários (0 e 1), variando a amplitude do sinal da portadora (*Carrier*) de acordo com os valores binários presentes no sinal da mensagem (*Data*) (Fig. 9).

Figura 11 – Processo de modulação em ASK.

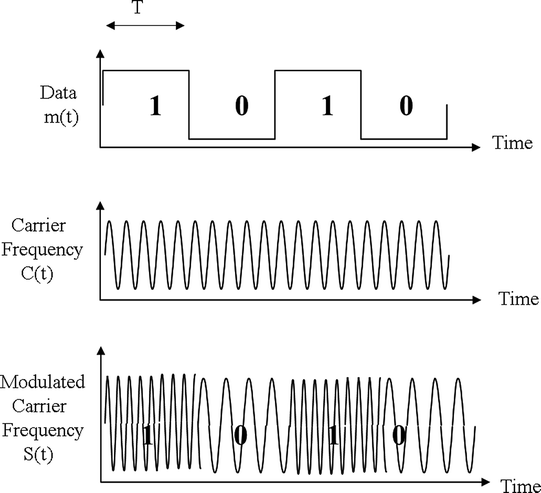


Fonte: SPRINGERLINK, 2021.

* + - * 1. Modulação FSK (Frequency Shift-Keying)

No caso da modulação em frequência, em vez do chaveamento de amplitude, ocorre o chaveamento de frequência de acordo com os *bits* presentes no sinal da mensagem que é transmitida (Fig. 10).

Figura 12 – Processo de modulação em FSK.



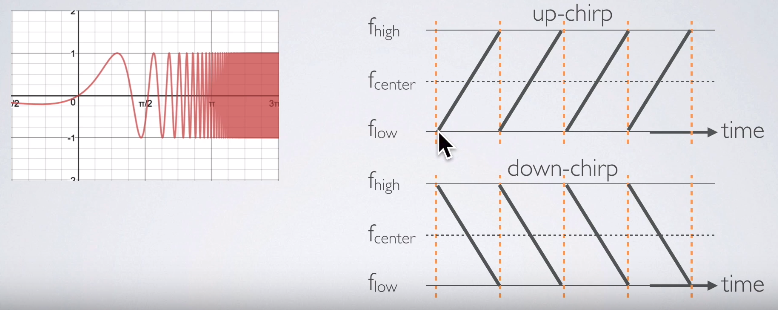
Fonte: SPRINGERLINK, 2021.

* + - 1. Chirp Spread Spectrum e LoRa

A partir das técnicas de modulação digital, surgem métodos similares como o CSS (*Chirp Spread Spectrum*), que é uma forma semelhante ao chaveamento de frequências, porém a partir de saltos de frequência.

Tecnicamente, o CSS funciona a partir de saltos (“*chirp”*, em inglês) de frequência (Fig. 11) entre uma faixa de valores, podendo ser *up-chirp*, caso o salto seja para uma frequência superior ou *down-chirp*, caso o salto seja para uma frequência inferior. Cada um dos saltos representa um símbolo que será decodificado e cada símbolo, por sua vez, pode representar um ou mais *bits* de dados.

Figura 13 – Modulação a partir da CSS.



Fonte: GTA, UFRJ.

O número de símbolos gerados pode variar de acordo com o fator de espalhamento “SF”, que representa o número de *bits* os quais serão utilizados na modulação. Esse está diretamente ligado a distância que um dispositivo pode se comunicar com outro, visto que, quanto maior o fator de espalhamento, maior a distância capaz de percorrer. Entretanto, quanto maior o fator de espalhamento, maior o tempo de transmissão, diminuindo assim, sua taxa de dados. A figura 12 ilustra o comportamento dos diferentes valores de fator de espalhamentos em um gráfico temporal.

Figura 14 – Gráfico temporal de fator de espalhamento.



Fonte: RESEARCHGATE, 2021.

Percebe-se que a imagem mostra que o menor valor possível de fator de espalhamento (7) gera uma comunicação bem mais rápida, enquanto o maior valor representa, também, uma maior latência de transmissão.

Uma das ferramentas derivadas da tecnologia CSS e que vem se tornando muito popular no cenário da IoT é a recém-apresentada LoRa (*Long Range*, “grande distância”, em inglês), que trata de uma técnica de modulação para redes de áreas extensas (WAN) utilizando frequências de espectro não-licenciados, variando de acordo com o continente em que é aplicada. A tabela 1 mostra a faixa de frequência utilizada pela tecnologia LoRa nas principais regiões em que é utilizada.

Tabela 1 – Faixa de frequências LoRa de acordo com as principais regiões.

|  |  |
| --- | --- |
| Região | Frequência (MHz) |
| Europa | 433 ou 863-870 |
| Estados Unidos | 902-928 |
| China | 470-510 ou 779-787 |
| Austrália | 915-928 |
| Índia | 865-867 |
| Ásia | 433 |
| América do Norte | 915 |

Fonte: MOKOSMART.

N Brasil, a faixa de frequência mais comum para a utilização do LoRa é entre 868 MHz e 915 MHz.

A partir da utilização da tecnologia LoRa, é possível conectar dispositivos que estão localizados em pontos distantes, o que traz proximidade entre a tecnologia e nichos específicos, como a agricultura de precisão.

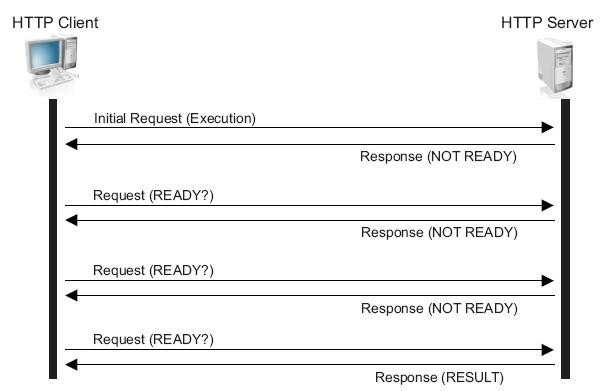
* 1. Tecnologia Master/Slave

Essa tecnologia é definida por um modelo de comunicação com uma hierarquia básica na qual um dispositivo (*Master*) tem o controle unidirecional sobre todos os outros (*Slaves*), que são os subordinados. Portanto, pode ser aplicada em sistemas nos quais é necessária a requisição de pelo dispositivo controlador aos dispositivos subordinados através, por exemplo, de iterações ou escalonamento.

* + 1. Técnica de Polling

Essa técnica é baseada em uma estrutura na qual um registrador é dedicado a armazenar o status de um processo e o sistema verifica periodicamente qual é esse status. Para a hierarquia de *Master/Slave*, seria como se o dispositivo controlador enviasse requisições e checasse de tempos em tempos se a resposta dos dispositivos subordinados foi recebida, assim como ilustrado pela figura 13.

Figura 15 – Ilustração da técnica de Polling.



Fonte: RESEARCHGATE.

Em um sistema com mais de um dispositivo subordinado, isso causaria um atraso muito grande na execução de processos, visto que o *Master* só enviaria a requisição ao próximo *Slave*, caso verificasse que a resposta do *Slave* anterior foi recebida.

* + 1. Técnica Round Robin

A técnica de *Round Robin* é desenvolvida a partir de um algoritmo escalonador de processos, os quais são organizados em filas e tem um tempo definido para serem executados, chamado de *“Time-Slice”*. Assim que cada fatia de tempo é encerrada, o próximo processo da fila é iniciado e assim sucessivamente. A figura a seguir mostra um diagrama representativo dessa técnica.

Figura 16 – Ilustração da técnica de Round Robin.



Fonte: EMBARCADOS.

Em uma topologia de rede na qual é empregado esquema de *Master/Slave*, o *Round Robin* pode ser aplicado em uma situação em que o dispositivo controlador envia requisições de forma escalonada entre os subordinados a cada fatia de tempo e durante o *Time-Slice*, o *Master* fica disponível para receber a resposta A partir do momento que ela se encerra, mesmo caso não receba resposta alguma, a requisição é enviada ao próximo subordinado. Assim que todos são acionados, o processo retorna ao primeiro *Slave*.

Embora a ideia por trás da técnica seja majoritariamente aplicada para a comunicação entre dispositivos eletrônicos em meio confinado, como circuitos interligados, ela pode ser transposta a outros modelos e arquiteturas, como a interação entre rádios em uma rede, de acordo com a sua topologia. Dessa forma, um dos dispositivos atua como *Master* enviando requisições e monitorando respostas dos rádios que fazem o papel de *Slaves*.

* 1. Topologias de Rede

No âmbito das telecomunicações, a forma com a qual os dispositivos são interligados em uma rede e como trocam dados e informações entre si recebe o nome de topologia. Existem inúmeros tipos de topologia de rede, mas levando em consideração o foco do projeto, as mais importantes são *Broadcast*¸*Multicast* e *Unicast*.

* + 1. Topologia *Broadcast*

A topologia *broadcast* é dada pela existência de um dispositivo central que consegue enviar a todos os outros receptores informações simultaneamente através de rádio difusores, por exemplo. Sinais midiáticos como televisão, rádio etc. são enviados utilizando essa técnica, pois a mesma informação precisa ser enviada da emissora a todos os receptores simultaneamente. A figura a seguir ilustra a topologia de *broadcast*, na qual o dispositivo central (em vermelho) envia o mesmo sinal aos receptores (em verde) ao mesmo tempo.

Figura 17 – Topologia broadcast.

Uma imagem contendo relógio

Descrição gerada automaticamente

Fonte: WIKIPEDIA.

* + 1. Topologia *Multicast*

No caso da topologia *multicast*, o funcionamento do tráfego de informações é muito semelhante ao *Broadcast*, simultaneamente, com a exceção de que os dados são enviados apenas a um grupo seletivo de receptores, ponto a ponto. Isso pode ser observado, por exemplo, em transmissões de dados com base na rede em que os dispositivos estão conectados, portanto, só receberão as informações, aqueles que estiverem na mesma rede. A figura 16 ilustra o funcionamento da topologia *multicast*.

Figura 18 – Topologia multicast.

Padrão do plano de fundo

Descrição gerada automaticamente

Fonte: WIKIPEDIA.

* + 1. Topologia *Unicast*

A última topologia de rede citada se refere à transmissão de informações endereçadas um único destino, também de forma ponto a ponto. A figura a seguir ilustra a comunicação *unicast*, em que o ponto em vermelho envia os dados que são endereçados somente ao ponto em verde.

Figura 19 – Topologia unicast.

Desenho de um cachorro

Descrição gerada automaticamente com confiança baixa

Fonte: WIKIPEDIA.

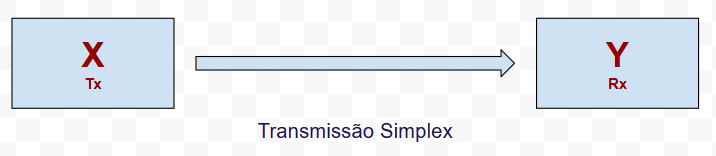
* 1. sistemas de comunicação

Os sistemas de comunicação estão inseridos nas topologias de rede de troca de informações unidirecional ou bidirecional. Dessa forma, são classificados em 3 categorias: *simplex*, *half-duplex* e *full-duplex*.

* + 1. *Simplex*

O sistema chamado de *simplex* consiste na troca de dados, unidirecional entre dois pontos, mas em que há apenas um transmissor e um receptor, ou seja, os dados só são trocados em um sentido. Um exemplo de transmissão *simples* é a TV, sistema no qual a emissora apenas envia em *broadcast* e os receptores apenas recebem. A figura a seguir ilustra um sistema em comunicação *simplex*.

Figura 20 - Comunicação *simplex*.

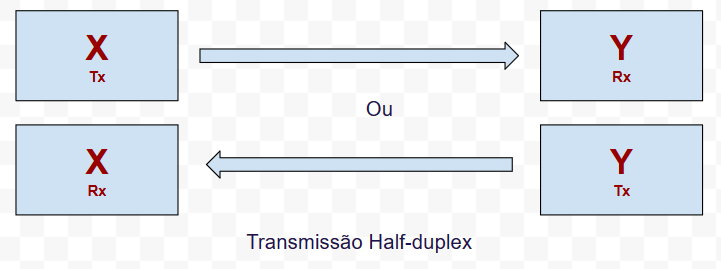


Fonte: BOSON TREINAMENTOS.

* + 1. *HALF-DUPLEX*

No caso da transmissão *half-duplex*, o sistema em questão é bidirecional, ou seja, os dados são transmitidos nos dois sentidos. Entretanto, a troca de informações entre os pontos não ocorre simultaneamente, implicando que, cada ponto espere sua vez de enviar os pacotes. Rádios do tipo *walkie-talkie* são exemplos desse sistema de comunicação mostrado na figura abaixo.

Figura 21 - Comunicação *half-duplex*.

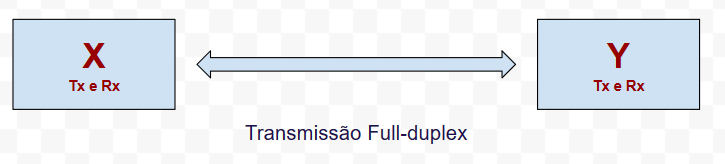


Fonte: BOSON TREINAMENTOS.

* + 1. *FULL-DUPLEX*

A terceira categoria, chamada de *full-duplex*, descreve sistemas de comunicação nos quais a transmissão e recepção das informações acontece de forma bidirecional e simultânea, possibilitando que sejam transmitidas e enviadas informações ao mesmo tempo. Aparelhos celulares são exemplos de sistemas de comunicação *full-duplex*, já que, com eles, há a possibilitade de transmissão e recepção de dados simultaneamente. A figura abaixo ilustra o tipo de comunicação *full-duplex*.

Figura 22 – Comunicação *full-duplex*.



Fonte: BOSON TREINAMENTOS.

* 1. *Python*, Django e *javascript*

*Python* é uma linguagem de programação de alto nível e orientada a objetos, lançada em 1991. Com a combinação de sintaxes claras e concisas, além das várias bibliotecas existentes, se tornou uma linguagem muito poderosa e que prioriza a legibilidade do código juntamente com a velocidade de interpretação.

O Django, por sua vez, trata-se de uma ferramenta (ou framework) de desenvolvimento web em python criada para a elaboração de aplicações práticas com alta velocidade e eficiência e pode ser dividido em 2 partes: *back-end* e *front-end*.

O *back-end* se refere à capacidade de processamento dos dados internos na aplicação. É nele que os dados são recebidos, tratados e devolvidos, o que acaba abrindo um leque de possibilidades muito expressivo. Outra grande vantagem é a presença de um banco de dados nativo, que pode armazenar informações de usuários ou quaisquer outros valores dependendo da aplicação.

Já o *front-end* é caracterizado pela parte visível e acessível ao usuário da aplicação, visto que ele comporta páginas *HTML* (com partes programadas em linguagem *JavaScript*) como sites e permite a sua customização incluindo os próprios dados obtidos pelo *back-end*.

*JavaScript* é uma linguagem de programação, também orientada a objetos, mas com foco em desenvolvimento web, utilizada para desempenhar diversas funções, como manipular elementos de uma página e transportar dados do *back* ao *front-end*.

De forma geral, a aplicação do Django proporciona um dinamismo muito grande, visto que os dados que são obtidos através da programação de código *python* em seu “núcleo”, podem ser tratados e exibidos de forma simples, rápida e muito eficiente em um servidor local ou, até mesmo, domínios públicos. Além disso, ele possui ferramentas de segurança cibernética muito poderosas e tem uma escalabilidade muito grande, podendo ir de um pequeno projeto até parte de grandes aplicações do mercado, como *Spotify*, *Instagram*, *Mozilla*, entre outros.

1. PROPOSTA

Atualmente, ainda que a evolução tecnológica tenha passado por diversos desdobramentos e tenham surgido novas propostas de comunicação de dados, a Internet das Coisas no ambiente rural ainda se encontra dando seus primeiros passos.

Visando incorporar conectividade à agricultura de precisão, o presente projeto tem como proposta apresentar uma aplicação, que promove ao pequeno e médio produtor agropecuário, o protótipo (utilizando um leque de sensores que pode ser expandido) de um sistema responsivo de monitoramento e gerenciamento de valores de temperatura, umidade do ar e umidade do solo, com a finalidade de aumentar a eficiência do cultivo de seus produtos.

Portanto, a arquitetura proposta consiste em rádios interligados em topologia *unicast* com escalonamento de requisições enviadas pelo *master* aos *slaves* conectados a ele, utilizando a técnica de *Round Robin*.

Além da rede de sensores sem fio, a aplicação conta com um servidor local com a finalidade de exibição dos dados obtidos. Para isso, utilizar-se-á a ferramenta de desenvolvimento *web* Django, programada em linguagem *python*, juntamente com páginas *HTML* para a inserção dos *dashboards*. Visando a privacidade e personalização, é disponibilizado também um sistema de *login* com usuário e senha, para que as informações possam ser mantidas de forma segura.

A figura a seguir ilustra o fluxograma de proposta do projeto.

Figura 23 – Arquitetura do projeto.

Uma imagem contendo Diagrama

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

1. RECURSOS
   1. Recursos de *Hardware*
      1. ESP-32 LoRa

Para o processamento remoto dos dados obtidos pelos sensores e envio deles até a central para sua exibição, foi utilizada a placa ESP-32 (Fig. 14), da *Heltec Automation*, que conta com comunicação LoRa através de um transceptor (modelo SX1276) ligado à placa, Wi-Fi e Bluetooth, além de um display OLED que possibilita a leitura de informações úteis do processo de medição, transmissão e recepção dos dados.

Nas tabelas 3 e 4 é possível visualizar as especificações técnicas do módulo e do transceptor LoRa.

Tabela 2 – Especificações do da placa ESP-32 LoRa.

|  |  |
| --- | --- |
| Tensão de Operação | 2,2 V a 3,6 V |
| *Clock* do processador | 240 MHz |
| Memória interna | 502 KB |
| Corrente de consumo típica | 80 mA |
| Corrente de consumo máxima | 500 mA |

Fonte: FILIPEFLOP, MERCADO LIVRE.

Tabela 3 – Especificações do transceptor LoRa.

|  |  |
| --- | --- |
| Frequência de operação | 915 MHz |
| Sensibilidade | -139 dBm (máximo) |
| Tensão de alimentação | 1,8 V a 3,7 V |
| Corrente de consumo (recepção) | 10,8 mA |
| Corrente de consumo (transmissão) | 120 mA |
| Potência máxima | 20 dBm |
| Modos de modulação | LoRa, FSK, GFSK e OOK |
| Temperatura de trabalho | -40ºC a 85ºC |

Fonte: MERCADOLIVRE, GITHUB.

* + 1. DHT 22

Para realizar as medições de temperatura do ambiente, o sensor escolhido foi o DHT22, capaz de medir tanto a temperatura como a umidade relativa do ar. É a versão mais precisa da linha DHT, sucessor do DHT11. A tabela 5 apresenta as características do sensor seguida pela figura 14, que mostra seu encapsulamento.

Tabela 4 – Especificações do sensor DHT22.

|  |  |
| --- | --- |
| Tensão de operação | 3-5VDC (5,5VDC máximo) |
| Faixa de medição de umidade | 0 a 100% UR |
| Faixa de medição de Temperatura | -40º a +80ºC |
| Corrente | 2,5mA máx. durante uso |
| Precisão de medição da umidade | ± 2,0% UR |
| Precisão de medição da temperatura | ± 0,5 ºC |

Fonte: FILIPEFLOP.

Figura 24 – Sensor DHT22.

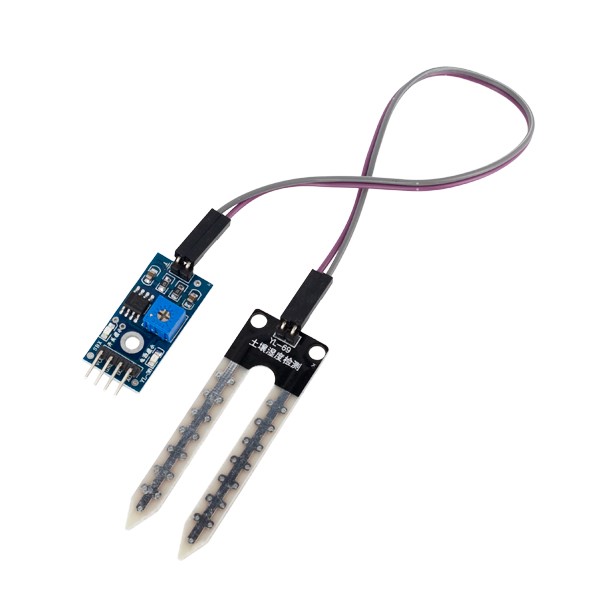


Fonte: FILIPEFLOP, 2021.

* + 1. Sensor de Umidade do Solo

Para realizar as medições da umidade do solo, foi escolhido o sensor analógico, mostrado na figura 16.

Figura 25 – Sensor de umidade do solo.



A tabela 6 apresenta as características eletrônicas e de medição do sensor.

Tabela 5 – Especificações do sensor de umidade do solo.

|  |  |
| --- | --- |
| Tensão de operação | 3,3V até 5V |
| Precisão de medida | Regulável com o potenciômetro |

Fonte: FILIPEFLOP, 2021.

A tabela 7 lista os componentes utilizados no projeto e suas quantidades.

Tabela 6 – Número de componentes utilizados.

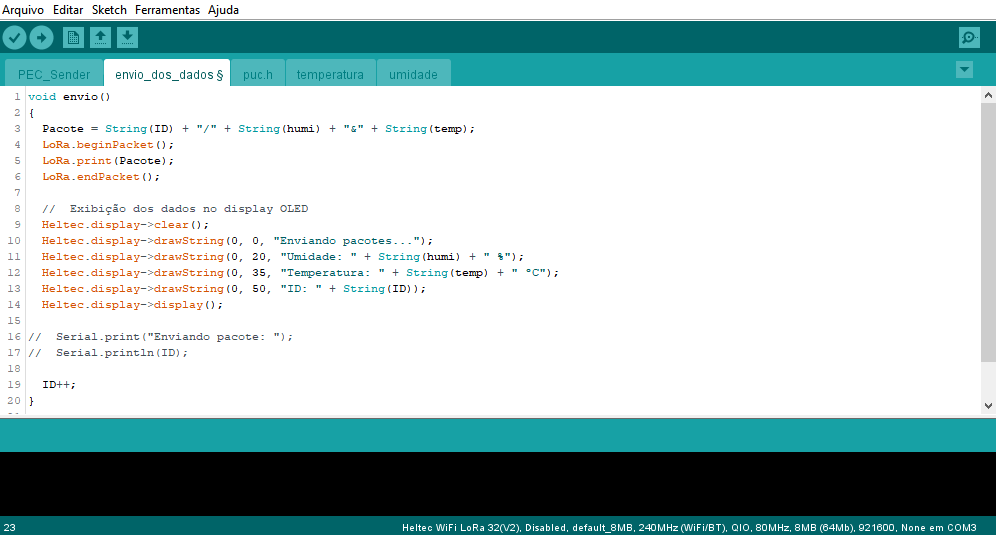
|  |  |
| --- | --- |
| Componentes | Quantidades |
| Placa ESP32 LoRa | 3 |
| Antenas | 3 |
| Sensor DHT22 | 1 |
| Sensor de Umidade do Solo | 1 |
| Resistor 10kΩ | 1 |
| Protoboard | 2 |
| Computador | 1 |

Fonte: AUTORIA PRÓPRIA.

* 1. Recursos de *Software*

Para o desenvolvimento do *software* do projeto, é necessária uma interface para que seja possível programar as funções das placas em que estão localizadas os microcontroladores e sensores. Isso pode ser feito através de um ambiente integrado de desenvolvimento chamado “Arduino IDE” (figura abaixo), que reconhece quando a placa está conectada ao computador e permite que seja programada a partir de linhas de código baseadas em C++.

Figura 26 – Arduino IDE.



Fonte: AUTORIA PRÓPRIA, 2021.

Além da interface para a programação das placas, é necessário um ambiente de desenvolvimento para a exibição dos dados, que será feita através de um *dashboard*. Para isso, foi escolhida a ferramenta Django na versão 3.2.3, que trata de uma forma comum e atual para a combinação entre funções de linguagem Python e JavaScript, que possibilita a elaboração de páginas *web*, contendo os dados obtidos.

1. METODOLOGIA
   1. Programação dos dispositivos de *hardware*

A metodologia utilizada na elaboração do projeto consiste em três grandes partes: coleta dos dados pelos sensores, transmissão dos dados através do protocolo LoRa e desenvolvimento da plataforma de gerência e monitoramento. Para a parte do *hardware* responsável pela medição e envio/recebimento de dados, a arquitetura é particionada em 3 subdivisões: nó (ou *Slave*) 1, que será responsável pelos valores de umidade e temperatura do ar, nó (ou *Slave*) 2, que será responsável pelos valores de umidade do solo, e *Master*.

* + 1. Master

O código para a programação do *Master* é o mais extenso, visto que é o responsável por gerenciar o fluxo, tanto de saída como de entrada dos dados. As duas principais funções realizadas por esse nó são o envio de requisições aos *slaves* pedindo pelos dados e, em seguida sua amostragem através do Monitor Serial (ferramenta presente na interface do Arduino para a visualização de dados obtidos pela placa). As figuras a seguir ilustram todas as instruções que o nó realiza.

Figura 27 – Inclusão da biblioteca necessária*.*



Fonte: AUTORIA PRÓPRIA.

Nas duas primeiras linhas do código é incluída a biblioteca necessária para o funcionamento correto do código. Ela é responsável por realizar o gerenciamento de todos os componentes da placa esp32 LoRa bem como habilitar as funções relacionadas ao transceptor de rádio LoRa.

Figura 28 - Declaração de variáveis.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

Após a inclusão da biblioteca, é feita a definição da frequência de comunicação escolhida (915 MHz), a definição do número de nós (2) para a contagem e a declaração das variáveis (do tipo *unsigned long*) que serão utilizadas no código para medir o tempo entre cada requisição feita pelo *Master*. A variável “*received”* é uma cadeia de caracteres responsável por armazenar os dados dos pacotes recebidos de cada *slave*.

A próxima etapa do código é programar as instruções que serão realizadas apenas uma vez, assim que o nó é energizado. Para isso, existe o bloco de funções chamado “*setup”*, no qual estão contidas as funções “*Heltec.begin()”*, incluindo as configurações de funcionamento do da placa, como os parâmetros “*DisplayEnable*” (para a habilitação do *display* OLED da placa), “*Heltec.Lora Disable”* (para a reinicialização do transceptor), “*PABOOST* *Enable*” (para a habilitação do configurador de potência do chip transmissor), “*BAND*” (para a definição da frequência pré-definida na seção de declarações) e “*Serial Enable*” (para a habilitação da comunicação serial da placa com o computador). Por fim, é ativado o recebimento de pacotes com a função “*LoRa.receive();”*. A figura a seguir ilustra o bloco de funções “*setup*”. Em seguida é declarada a variável “i”, responsável por identificar qual dos nós receberá a requisição enviada pelo *Master*.

Figura 29 - Bloco de funções "setup".

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

Após a inicialização, o bloco “*loop*” é responsável por executar as funções presentes repetidamente, em ciclos. Ao final da execução do da última instrução, é retornada à primeira e assim por diante. A figura a seguir apresenta todas as funcionalidades que serão executadas continuadamente.

Figura 30 - Bloco loop do código do Master.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

* + - 1. Função millis()

A função *millis()*, pertencente à própria categoria de funções base da placa Arduino, tem como objetivo retornar o número de milissegundos passados desde o início do programa atual, ou seja, da função *loop*, até o momento em que é chamada no código. Nesse contexto, ela é responsável pela contagem de tempo entre o envio de requisições do *Master* aos *Slaves*, evitando que haja um excesso de dados enviados e recebidos. Além disso, é possível a implementação de *timeouts*, ou seja, intervalos de tempo máximo entre a espera da resposta dos *Slaves*. Caso o tempo de espera ultrapasse o valor de *timeout*, a requisição é enviada ao próximo *Slave*.

Essa funcionalidade se mostra muito mais útil em comparação à função *delay()*, que é muito mais comum, porque a segunda função interrompe todos os processos realizados durante o período de tempo determinado, enquanto a primeira é utilizada em “segundo plano”, permitindo que outras funções sejam realizadas durante a contagem de tempo.

* + - 1. Envio da requisição ao *Slave*

No começo do laço, a função *millis()* é chamada para que seja iniciada a contagem de tempo durante a realização das outras tarefas e o valor obtido é armazenado na variável “*currentTime*”. Em seguida, é realizada a checagem do valor da variável “i”, para que o *Master* saiba o destinatário da requisição. Caso o valor armazenado nela seja maior que a constante “NODES” (que indica o número de nós presentes no sistema e que receberão requisições), é retornado a ela mesma o valor “1”, significando o final das iterações e recomeçando o processo, pelo primeiro nó.

Depois, é checado se o valor obtido da função *millis()* é superior ao intervalo de tempo constante pré-definido (5 segundos). Caso seja confirmado, o processo de empacotamento da requisição é iniciado através da função “*LoRa.beginPacket*” e, em seguida, é definida a potência de transmissão, a partir dos valores de ganho (14 dB) e do configurador de potência do transmissor (*PABOOST*). Logo após, é inserido o valor de “i”, que significa a mensagem de requisição que será enviada ao nó correspondente. Então, é finalizado o empacotamento com a função “*LoRa.endPacket*”.

Feito isso, o valor armazenado na variável “*currentTime*” é substituído pelo valor contido em “*lastSendTime*” para que ela seja sempre atualizada com a contagem atual de tempo.

* + - 1. Recebimento de pacotes

Após a checagem, é feita a tentativa de recebimento do pacote contendo os dados enviados pelo *slave* correspondente, a partir da função “*LoRa.parsePacket()*”, que é responsável por verificar se há algum dado para disponível para ser lido, retornando um valor inteiro correspondente ao tamanho do pacote disponível e armazenando-o na variável “*packetSize*”. Caso não exista nenhum dado, o valor retornado será 0. Então, é checado o valor contido nessa variável e, se for maior que 0, é iniciado o processo de decodificação das informações recebidas. Isso é possível através da função “*LoRa.readString()*”, que lê os caracteres do pacote recebido (enquanto ele estiver disponível, o que é checado pela função *“LoRa.available()*”, dentro de um laço do tipo *while*, ou seja, “enquanto o pacote estiver disponível”) e os armazena em uma variável do tipo *string*, que é uma cadeia de caracteres.

Em seguida, são enviados à porta *Serial* o pacote recebido e a intensidade do sinal da comunicação, indicada pelo valor retornado da função “*LoRa.packetRssi()*”, separados por um caractere identificador, nesse caso, a barra “/”, para que sejam identificados e separados no pós processamento.

Por fim é feita a checagem do valor de *timeout* do algoritmo, a partir da comparação do valor de tempo retornado pela função *millis()* em relação ao tempo pré-definido (nesse caso de 5 segundos). Caso a diferença seja maior que o valor de *timeout*, é entendido que o tempo de espera do *Master* foi ultrapassado e ele deve enviar a requisição ao próximo *Slave*. Isso pode significar que o *Slave* foi desconectado ou apresenta algum problema de comunicação.

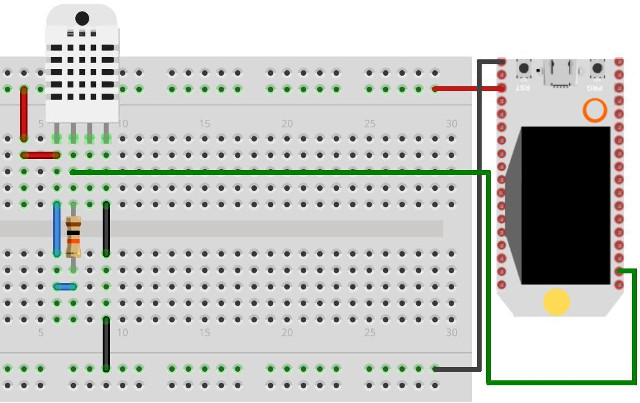
Para que a iteração seja continuada, o valor de “i” é incrementado, portanto, a próxima requisição terá o valor do pacote contendo informações para o próximo nó.

Uma das partes cruciais do projeto é a impressão dos valores recebidos no Monitor Serial através da função “*Serial.println()*”, que possibilitará o pós processamento dos dados através do código *python*.

* + 1. Nó 1 - Temperatura e Umidade

A partir da programação do sensor DHT22 será realizada a medida da temperatura e umidade relativa do ar. Também é adicionado ao esquemático do circuito um resistor de 10kΩ, que tem a função de assegurar que os valores obtidos não sejam corrompidos por quaisquer interferências eletrônicas. As conexões em vermelho simbolizam a ligação para a saída de alimentação, as ligações em preto simbolizam a ligação para o GND e a ligação em verde a saída de dados.

Figura 31 – Conexão entre ESP-32 LoRa e DHT22.



Fonte: AUTORIA PRÓPRIA.

Para a programação da placa, é utilizado o código contendo todas as instruções a serem realizadas pelos componentes. Assim como realizado com o *Master*, a primeira etapa da codificação é a inclusão das bibliotecas necessárias para o funcionamento da placa e, nesse caso, também do sensor.

Figura 32 – Cabeçalho do código para o nó 1.

Relógio digital com números

Descrição gerada automaticamente com confiança baixa

Fonte: AUTORIA PRÓPRIA.

Além da “*heltec.h*”, responsável pelo controle da placa e controle do transceptor, é necessária a declaração da biblioteca “*DHT.h*”, que é responsável por gerenciar todas as funções relacionadas ao sensor DHT22, como a sua inicialização e o tratamento dos dados para que possam ser processados.

Posteriormente, é feita a definição de todos os parâmetros que serão referenciados, assim como mostra a figura a seguir.

Figura 33 - Declaração de variáveis do nó 1.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

O primeiro, “*BAND*”, assim como em todas as placas, representa a frequência de comunicação entre os rádios (915 MHz). Em seguida, é definido o pino pelo qual serão recebidos os dados de temperatura e umidade do sensor (21) e, depois, é definido o modelo do sensor DHT (22). Então, essas informações são indicadas à biblioteca *DHT.h*, para que o sensor seja inicializado corretamente.

Logo após, são declaradas as variáveis que irão armazenar valores importantes dentro do código, como a variável “*received*”, responsável por armazenar os caracteres recebidos da requisição em uma cadeia (*String*), “*packet*”, que irá guardar os valores de resposta a serem enviados ao *Master*, “*request*”, que indica qual é o identificador da requisição recebida, para que o nó apenas envie a resposta caso a requisição seja para ele e as variáveis “*temperature*” e “*humidity*”, que são responsáveis por armazenar os dados de leitura do sensor.

O próximo passo é a configuração da placa a partir da função “*setup()*”, chamada da mesma forma realizada para o dispositivo *Master* (figura 26).

O laço de repetição (*loop*) do nó 1 contêm as funcionalidades de recebimento da requisição e envio da resposta, caso seja sua vez durante o processo de iteração. A figura a seguir apresenta as tarefas a serem realizadas por esse nó.

Figura 34 - Bloco loop do código do nó 1.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

A primeira parte do bloco é relacionada à identificação da presença de pacotes na recepção, porém, nesse caso, o pacote esperado é a requisição enviada pelo *Master*. Caso o tamanho de “*packetSize*” seja maior que 0, a variável “*received*” é preenchida com os caracteres do pacote. Se a informação contida nele for igual à variável “*request*”, significa que é a vez desse nó enviar os dados. Para a sinalização desse evento, é mostrado no Monitor Serial o pacote que foi decodificado na recepção.

Em seguida é acionada a função “*readData()*”, apresentada na figura a seguir.

Figura 35 - Função readData().

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

Essa seção é responsável por recolher os dados de temperatura e umidade do ar medidos pelo sensor DHT22 e armazená-los nas variáveis correspondentes (*temperature* e *humidity*) através das funções “*dht.readTemperature()*”, que faz a leitura da temperatura e “*dht.readHumidity()*”, que faz a leitura da umidade e estão presentes na biblioteca “*DHT.h*”, incluída no cabeçalho do código. Logo após, é montado o pacote que será enviado como resposta à requisição do *Master* e ele é composto por cadeias de caracteres concatenadas estrategicamente para que seja de fácil leitura e processamento.

Primeiramente é incluída no pacote a variável “*request*”, representando ao *Master* de qual nó é a informação e, em seguida, são incluídas medidas realizadas pelo sensor. Todas as informações são separadas pelo caractere “/”, que tem a função de sinalizar ao destinatário as posições em que a cadeia de informações deve ser separada, possibilitando assim, o envio de múltiplas informações em um único pacote. A seguir, também é mostrado no Monitor Serial as medições que foram empacotadas.

Finalmente, o *loop* é finalizado com o envio do pacote através das funções “*LoRa.beginPacket()*”, “*LoRa.setTxPower()*”, “*LoRa.print()*” e “*LoRa.endPacket()*”.

* + 1. Nó 2 – Umidade do Solo

A partir da programação do higrômetro será realizada a medida da temperatura e umidade do solo. A figura a seguir ilustra a montagem da placa ESP-32 LoRa conectada aos componentes.

Figura 36 - Conexão entre ESP-32 LoRa e higrômetro.

Uma imagem contendo circuito, computador

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

O cabeçalho da programação do segundo nó é exatamente igual ao código do *Master*, visto que esse sensor, por se tratar de leituras analógicas, não requer nenhum tipo de biblioteca auxiliar. Entretanto, as variáveis utilizadas são diferentes, como mostra a figura a seguir.

Figura 37 - Declaração de variáveis do nó 2.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

Para que a placa possa identificar os dados do sensor, é necessário que seja indicado o pino pelo qual serão transmitidas as informações. Para isso, é passada a variável “*sensorPin*” do tipo inteiro, contendo o número do pino (13) ao qual será conectado o higrômetro. Assim como no nó 2, são declaradas as variáveis “*received*”, “*packet*” e “*request*” que, nesse caso, é igual a 2 (número do nó). Em seguida é declarada a variável “*moisture*”, do tipo “*float*” (para a exibição de algarismos não inteiros), que irá armazenar o valor lido pelo sensor.

O próximo passo, assim como nos outros dispositivos, é a configuração da placa a partir da função “setup()”, chamada da mesma forma realizada para o dispositivo Master (figura 26).

O processo de recebimento da requisição é exatamente igual ao do primeiro nó, com a checagem do pacote de requisição e o envio dos dados (figura 31), com a diferença somente na função “*readData()*”, que é explicada na figura a seguir.

Figura 38 - Função *readData()* para o higrômetro.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

Como se trata de um sensor analógico, a leitura dos dados do sensor é feita a partir da função “*analogRead()*” (levando a variável *sensorPin* como parâmetro para indicar o pino do qual serão recebidas as informações). Entretanto, com a leitura analógica, o valor retornado pelo sensor varia entre 0 e 1023 e, para que seja possível extrair dele valores que possam ser mais bem interpretados, é necessária a conversão a partir da seguinte fórmula:

Assim, o valor retornado será a porcentagem da umidade do solo.

Em seguida, é feito o armazenamento desse valor na variável “*moisture*” e, sem seguida o empacotamento, junto ao dado de identificação do nó, com a variável “*request*”. Logo após, é mostrado no Monitor Serial o pacote gerado e, em seguida, são enviadas as informações ao *Master*.

* 1. Programação do *software*

A programação do *software* da aplicação se concentra na elaboração do núcleo para o processamento dos dados obtidos através do Monitor Serial e, também, a codificação das páginas do servidor *web* para a realização do *login* do usuário e exibição dos gráficos no *dashboard*.

Todo código contendo as funções realizadas pelo “*core*” ou “núcleo” da aplicação estão contidas em um arquivo específico, em formato interpretável pela versão do *python* instalada no computados, chamado “*views.py*”. Nele são feitas importações de bibliotecas importantes para o funcionamento de todas as partes, como a renderização das páginas *HTML* e a geração de respostas a elas (*render*, *redirect* e *HttpResponse*), a autenticação dos usuários para o sistema de *login* e *logout* (*auth*, *authenticate*, *login*, *logout* e *login\_required*), a formatação de dados (*json*) e, por fim, o controle da porta Serial (*serial*). A figura a seguir ilustra a seção de importações do código, presente no arquivo em questão.

Figura 39 - Importações do código python.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

* + 1. Pós processamento dos dados

Para que as informações mostradas enviadas pelo *Master* ao computador sejam processadas e analisadas, é necessário que o código *python* da aplicação tenha acesso à porta Serial, o que é realizado através da função “*Serial()*”, presente na biblioteca importada.

Figura 40 - Acesso à porta Serial.



Fonte: AUTORIA PRÓPRIA.

A partir da figura acima, pode-se observar que é feita a indicação da porta que será acessada (COM3) e pela qual serão transmitidas as informações ao computador bem como a taxa de dados (*baudrate*), que é igual à taxa definida na programação do código do *Master*. Por fim, é adicionado um *timeout* de 1 segundo por questões de segurança, com a finalidade de evitar travamentos no procedimento de acesso à porta e perda dos dados.

O próximo passo é a leitura dos dados e interpretação deles, mostrados na figura a seguir.

Figura 41 - Leitura e interpretação dos dados.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

A função “*getData()*” é responsável por todo esse processo, começando pela leitura da linha exibida na porta Serial, através da função “*readline()*” aplicada à “COM3”.

Como trata-se de uma comunicação de dados Serial, é necessária a tradução dos caracteres lidos, já que eles estão escritos em valores binários. Para isso, é utilizada a função “*decode*” com o parâmetro “utf-8”, que é o padrão mais utilizado de decodificação de caracteres. Na sequência, esses valores são transformados em uma cadeia de caracteres com a função “*str()*”, para que possam ser interpretados mais facilmente. Dessa forma, o valor obtido e armazenado na variável “*values*” é exatamente o pacote recebido pelo *Master*, no seguinte formato:

Figura 42 - Exemplo de formato de dados.



Fonte: AUTORIA PRÓPRIA.

A partir da figura acima, pode-se perceber que os dados codificados correspondem exatamente ao formato esperado, contendo os valores (nesse exemplo, para o nó 1) de identificação (“1”), temperatura (“19.10”), umidade (“84.00”) e intensidade do sinal ou *rssi* (“-2”), todos separados pelo caractere “/” para que sejam identificados pelo código com a função “*split*”, que faz a separação de todos os valores e o armazenamento deles em um vetor (ou *array*) de 4 posições (0, 1, 2 e 3).

Após a separação dos valores, deve-se obter o seguinte resultado:

Figura 43 - Dados formatados.



Fonte: AUTORIA PRÓPRIA.

Em seguida, é feita a checagem do valor desse vetor, para que não haja a possibilidade de um vetor vazio, violando assim a integridade dos dados. Para isso é utilizada a verificação do tamanho desse vetor a partir da função “*len()*”, que retorna os *bytes* contidos nele. Então, é verificado se esse número é diferente de 0, ou “*None*”, a partir do comparador lógico “!=”.

Caso a comparação seja verdadeira, a função é seguida e é feita outra verificação: o valor do caractere da primeira posição do vetor (posição 0). Dessa forma, o código consegue identificar de qual nó é essa resposta, visto que o primeiro digito é o identificador dele.

Se o valor lido é igual a “1”, trata-se de uma resposta do primeiro *slave* (temperatura e umidade do ar). Logo, é feita a distribuição dos valores de temperatura (sendo a posição número 1 do vetor), umidade (sendo a posição número 2 do vetor) e intensidade do sinal da comunicação (sendo a posição número 3 do vetor) no vetor “*data*” com as suas respectivas variáveis.

Em seguida, é feito o tratamento desses dados, transformando-os em formato *json*, para que possam ser enviados à página *web*, através de uma função chamada de “requisição AJAX”, representada pelo identificador “getJSON()”, presente no código *JavaScript*.

Finalmente, os dados podem ser enviados para a página *web,* na qual serão distribuídos em seus respectivos gráficos, através do retorno “*HttpResponse()*”, contendo as informações formatadas da variável “*data*”.

* + 1. renderização das páginas *web*

O processo de exibição das páginas escritas em *HTML* que irão conter os dados também é realizado a partir do código em *python*, como mostra a figura a seguir.

Figura 44 - Renderização das páginas *web*.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

A partir da primeira função, chamada de “*user\_login*” é retornada a página de autenticação, na qual o usuário insere seus dados e é levado à página de gráficos, definida pela segunda função (*home*). A única diferença é em relação ao parâmetro “*login\_required*”, que impede que a página seja carregada sem que o usuário esteja devidamente autenticado.

* + - 1. página de *login*

Figura 45 - Página de *login* do sistema.

Tela de computador com texto preto sobre gramado

Descrição gerada automaticamente com confiança baixa

Fonte: AUTORIA PRÓPRIA.

A figura acima apresenta a página de login renderizada pela função “*user\_login()*” para que o usuário insira seus dados e seja autenticado. A figura abaixo mostra a página de *dashboard*, renderizada através da função “*home()*”, após a autenticação.

Figura 46 - Página de Dashboard.

<INSERIR PÁGINA DE DASHBOARD>

Fonte: AUTORIA PRÓPRIA.

* + 1. processo de *login* e *logout* de usuários

Um usuário é cadastrado com suas informações na base de dados pelo administrador do sistema e, assim, pode ser autenticado através das funções de *login* e *logout*, mostradas a seguir.

Figura 47 - Função de login no *sistema*.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

Assim que o usuário preenche os campos com suas credenciais (*username* e *password*), a função “*signIn()*” é chamada, pegando os valores submetidos e tentando realizar a autenticação, através da função “*authenticate()*”. Em seguida, é feita a verificação do valor de usuário, para que não seja nulo, a partir da comparação do valor dele com o parâmetro “*None*”. Se a comparação for verdadeira, é realizado o login e o usuário é redirecionado à página “*home*”, contendo os dados exibidos no *dashboard*. Caso contrário, é recarregada a página de login com a mensagem indicando a ele que o processo não foi completado, solicitando uma nova tentativa.

A função de *logout* é mais simples, já que é necessário apenas o encerramento da sessão do usuário, realizado através da função “*user\_logout()*”. A única tarefa a ser realizada por essa função é executar o procedimento “*auth.logout()*”, presente na própria biblioteca e, em seguida, direcionar o usuário à tela de login novamente, como mostrado a seguir.

Figura 48 - Função de logout do sistema.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

* + 1. exibição dos dados no dashboard

A exibição dos dados no *dashboard* da página é realizada pela codificação em linguagem *JavaScript*, a partir da requisição de informações a cada intervalo de tempo definido. Os gráficos que irão receber os dados do *back-end* são posicionados para serem manipulados através de um *plug-in* da própria linguagem chamado “*Chart.*js”, especializado em gráficos e instrumentos de amostragem. A escrita em *HTML* é realizada por meio de “*tags*”, que simbolizam elementos da página, os quais serão interpretados pelo navegados para que possam ser exibidos na tela. A figura a seguir mostra a seção do código em que serão carregados os gráficos.

Figura 49 - Localização dos gráficos no código HTML.Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

As *tags* “*<div></div>*” são utilizadas para dividir a página *web* em blocos, possibilitando assim, uma melhor organização dos elementos. As *tags* “*<h2></h2>*” e “*<span></span>*” representam elementos de texto a serem carregados na página. Nesse caso, as *tags* utilizadas para que o *plug-in* seja conectado ao código *JavaScript* serão definidas como “*<canvas></canvas>*”. Dessa forma, o sistema pode localizar os gráficos, separados por identificadores de cada valor (por exemplo, no caso da temperatura: *id=”Temperature”)* presentes nessas *tags*, para que os dados possam ser distribuídos de forma correta.

Passando para a programação em linguagem *JavaScript*, todas as tarefas são englobadas em um bloco para serem realizadas assim que a página for carregada. Para isso é utilizado o objeto “*window*” e sua propriedade “*onload*”.

Em seguida, primeiramente são declaradas 3 variáveis importantes, assim como mostrado na figura a seguir.

Figura 50 - Declaração de variáveis do *front-end*.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

A primeira variável, “*updateInterval*”, é responsável por armazenar o intervalo de tempo a partir do qual as informações serão atualizadas nos gráficos. Nesse caso, de forma síncrona com a disponibilização dos dados pelo *Master*, o valor é de 5000, expresso em milissegundos, totalizando 5 segundos de duração. A variável “*numberElements*” tem a função de armazenar o valor do número máximo de informações (pontos) que serão exibidas no gráfico. Assim que esse número é ultrapassado, o valor mais antigo é descartado e um novo valor é adicionado. Para isso, a variável “*updateCount*” atualiza a contagem do número de pontos assim que é inserido um novo dado.

Em seguida, são armazenados em variáveis os identificadores de cada gráfico, para que o programa consiga localizá-los e realize a atualização dos valores. Esse procedimento é mostrado na figura abaixo.

Figura 51 - Localização dos gráficos.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

Dessa forma, é possível encontrar dentro dos elementos da página todos os identificadores (*tag* “*id*”) e encaminhá-los ao *plug-in*.

Depois, é definida a variável “*commonOptions*” que fica responsável por armazenar dados de preferência visual, como espaçamento e posicionamento de dados e legendas dos gráficos e a formatação dos eixos X e Y.

Figura 52 - Variável *commonOptions*.

Tela preta com letras brancas

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

A figura acima mostra a configuração de aspectos visuais do gráfico, como o espaçamento, a configuração do gráfico como responsiva, para que ele possa ser atualizado, o tipo de escala dos eixos e o formato de exibição dos pontos que representam o tempo. Por fim, são desabilitadas as legendas do gráfico (*legend* e *tooltip*), visto que há *tags* designadas especificamente para isso no corpo da página.

O próximo passo é instanciar os gráficos que serão exibidos (temperatura, umidade do ar e umidade do solo) e, para isso é utilizada a seguinte variável.

Figura 53 - Instância para a temperatura.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

Assim que criada, é chamada a função “*Chart()*”, que será responsável por criar o gráfico, passando a ela o parâmetro “*Temperature*”, já identificado no código (figura 48). Em seguida, é definido o tipo de gráfico como “*line*”, para que sejam exibidos pontos em linhas e aspectos como o rótulo, que indicará a temperatura (*label*), o valor de início (*data = 0*), o preenchimento (*fill* *= false*), a cor da borda do gráfico, dada em valor hexadecimal (*borderColor*) e sua espessura (*borderWidth*).

Em seguida, são adicionados ao gráfico de forma modular os valores de preferência padronizados pela variável “*commonOptions*” (figura 49), a partir da função “*Object.assign()*” e, depois, parâmetros relativos ao título do gráfico, como a exibição (*display*), o texto que será exibido (*text*) e o tamanho de sua fonte (*fontSize*).

Para os outros 2 gráficos, o procedimento será exatamente o mesmo, apenas alterando os nomes das variáveis de identificação e as cores de exibição para diferenciação.

A atualização dos gráficos em tempo real é realizada a partir da função “*updateData()*”, mostrada abaixo.

Figura 54 - Função *updateData().*

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

A primeira tarefa realizada é a chamada da função “*getData*”, presente no *back-end* da aplicação, através do mecanismo de requisição AJAX chamado “*getJSON*”, que tem como função buscar elementos com formato *json* no código *python* enviado à página *web*. Em seguida, a função passa os argumentos retornados à outra função do código, chamada “*addData()*”, mostrada abaixo e depois realiza a temporização entre uma requisição e outra, de acordo com a variável “*updateInterval*”, declarada de forma síncrona com as ações do *Master*.

Figura 55 - Função *addData()*.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

Primeiramente, é checado se o valor recebido pela função não é nulo. Caso não seja, a função passa por outra verificação, que é a identificação do nó responsável por aquela informação. Se a variável “*id*” for igual a “1”, as informações são passadas aos gráficos de temperatura e umidade. As funções “*TemperatureInstance.data.labels.push()*” e “*HumidityInstance.data.labels.push()*” são responsáveis por adicionar aos eixos X de cada um o valor referente à data que corresponde ao ponto gerado. Já as funções “*TemperatureInstance.data.datasets.forEach()”* e “*Humidity.data.datasets.forEach()*” são chamadas para adicionar aos gráficos os valores contidos nas variáveis correspondentes, no caso “temperatura” e “umidade”.

Em seguida, é feita a checagem do número de pontos presentes em cada gráfico. Caso seja maior que o número armazenado na variável “*updateCount*”, o valor mais antigo é descartado e o novo é adicionado. Caso não seja maior, é feito o incremento da variável “*updateCount*” e os valores são adicionados.

Para o segundo nó, o procedimento é exatamente igual, mudando apenas os nomes das variáveis identificadoras.

1. RESULTADOS

Os resultados do trabalho de conclusão se concentram em proporcionar uma avaliação da aplicação em relação às funcionalidades discutidas ao longo do documento, contendo 3 principais ensaios: troca de dados entre 2 pontos com topologia *unicast* e comunicação *simplex* unidirecional, comunicação entre *Master* e apenas 1 *Slave* com topologia *unicast* e comunicação *half-duplex* bidirecional e, por último, a aplicação proposta do projeto, ou seja, a comunicação entre *Master* e 2 nós (*Slaves*), com topologia *unicast* e comunicação *half-duplex* bidirecional, realizando o pós-processamento dos dados e os exibindo em uma página *web* (servidor local) a partir de gráficos em um *dashboard*.

* 1. Comunicação ponto a ponto entre 2 nós

O ensaio da comunicação ponto a ponto (ou P2P – *point to* point) entre os 2 dispositivos foi realizado a partir da utilização de 2 placas ESP32 LoRa, em que uma delas foi ligada ao sensor DHT22 de temperatura e umidade do ar. A codificação foi realizada para que o fossem feitas as medições dos valores e, a cada segundo, fosse montado o pacote com os dados e transmitidos à outra placa, sem a utilização do método *Master-Slave* e, nem mesmo, técnicas como o *Round Robin*.

A figura a seguir ilustra a arquitetura do ensaio realizado.

Figura 56 - Ensaio de comunicação ponto a ponto.

Uma imagem contendo Diagrama

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

Dessa forma, assim que os dados foram medidos pelo sensor, foi realizado seu empacotamento e envio, seguindo o padrão de formatação (valores de temperatura e umidade separados pelo caractere “/”, porém sem o número identificado, devido ao tipo de comunicação), assim como mostrado na figura a seguir, a partir do acompanhamento do processo através do Monitor Serial.

Figura 57 – Envio dos dados pelo nó sensor.

<substituir imagem>

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

O empacotamento dos dados e o envio dos pacotes foi realizado, em média, a cada 1 segundo. Dessa forma, as respostas exibidas do outro lado da comunicação também são intercaladas a partir desse intervalo, como mostra a figura abaixo.

Figura 58 - Recepção dos dados pelo nó receptor.

<substituir imagem>

Interface gráfica do usuário, Aplicativo, Tabela

Descrição gerada automaticamenteFonte: AUTORIA PRÓPRIA.

A partir da figura acima, pode-se observar que os dados foram recebidos como esperado: separados estrategicamente dentro do intervalo de tempo especificado.

* 1. Comunicação master-slave com 1 nó

O próximo ensaio realizado foi montado a partir da arquitetura baseada na técnica de *Master/Slave*, porém com apenas 1 nó receptor (ligado ao sensor de temperatura e umidade do ar DHT22), com a finalidade de testar a estabilidade e comportamento da técnica. Para a topologia de rede, foi escolhido o método *unicast*, porém, nesse caso, com comunicação bidirecional em *half-duplex*, ou seja, os nós apenas recebem ou transmitem.

Para isso, foi adicionado o esquema de requisição (*vide item 5.1.1*) a partir do envio do caractere “g” de “*get*”, do inglês “obter”, para a obtenção da resposta contendo os dados medidos pelo sensor. A próxima figura apresenta a ilustração da arquitetura proposta para o segundo ensaio.

Figura 59 - Primeiro ensaio de comunicação *Master-Slave*.

Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

A partir da exibição das tarefas realizadas pelo programa no Monitor Serial do dispositivo *Master*, é possível observar que ele realiza o envio do pacote de requisição contendo o caractere e, depois, recebe a resposta, contendo os valores esperados. A figura abaixo mostra a captura de tela das informações exibidas pelo *Master*.

Figura 60 - Monitor Serial do *Master*.

<substituir imagem>

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

Em seguida, é ilustrado com a figura 61 o comportamento do nó ligado ao sensor de temperatura e umidade.

Figura 61 - Monitor Serial do *Slave*.

<substituir imagem>

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

A partir da análise da figura acima, é possível observar que o nó recebe o pacote com a requisição “g”, empacota os dados medidos pelo sensor DHT22 e os envia como resposta.

* 1. Comunicação entre master e 2 nós

Por fim, o último ensaio realizado foi a aplicação da arquitetura proposta para o projeto final, contendo um dispositivo que realiza o papel de *Master*, enviando requisições e 2 dispositivos realizando a função de *Slaves* (nós sensores), respondendo com os dados medidos pelos sensores DHT22 (nó 1, medindo temperatura e umidade do ar) e Higrômetro (nó 2, medindo a umidade do solo).

Para isso, foi montada a arquitetura proposta no item 3, apresentando uma topologia de rede *unicast* com comunicação bidirecional *half-duplex* e utilização da técnica *Round Robin* para a realização das iterações entre os nós durante um intervalo de tempo e com o *timeout* aplicado. A figura abaixo ilustra a arquitetura proposta para o último ensaio.

Figura 62 - Ensaio final com comunicação *Master-Slave*.

Diagrama

Descrição gerada automaticamente

Fonte: AUTORIA PRÓPRIA.

1. CONCLUSÃO
2. REFERÊNCIAS