UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”

FACULDADE DE CIÊNCIAS - CAMPUS BAURU

DEPARTAMENTO DE COMPUTAÇÃO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MATHEUS SINTO NOVAES

HORTA VERTICAL COM SISTEMA DE MONITORAMENTO IOT

Bauru

2022

MATHEUS SINTO NOVAES

HORTA VERTICAL COM SISTEMA DE MONITORAMENTO IOT

Trabalho apresentado no curso de Bacharelado em Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho”

Orientador: Prof. Dr. João Eduardo Machado Perea Martins

Bauru

2022

N935h

Novaes, Matheus Sinto
Horta Vertical com Sistema de Monitoramento IoT / Matheus Sinto Novaes. -- Bauru, 2022
92 p.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Faculdade de Ciências, Bauru
Orientador: Prof. Dr. João Eduardo Machado Perea Martins

1. Internet das Coisas. 2. Telemetria. 3. API. 4. Hidroponia. 5. Horta Vertical. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Ciências, Bauru. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Matheus Sinto Novaes

Horta Vertical com Sistema de Monitoramento IoT

Trabalho de Conclusão de Curso de Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. João Eduardo Machado Perea Martins

Orientador
Departamento de Computação
Faculdade de Ciências
Universidade Estadual Paulista “Júlio de Mesquita Filho”

Profª. Drª. Simone das Graças Domingues Prado

Departamento de Computação
Faculdade de Ciências
Universidade Estadual Paulista “Júlio de Mesquita Filho”

Prof. Dr. Rogério Zanarde Barbosa

Faculdade de Ensino Superior Formação Integral

Bauru, 25 de janeiro de 2022

AGRADECIMENTOS

Agradeço primeiramente à minha família, pelo apoio durante meus anos de estudo, pelo incentivo, amor e companhia nos anos de minha vida.

Agradeço aos meus amigos que só conheci por conta deste curso, pelas noites de diversão e pelos dias de estudo, pelo apoio e incentivo mútuo, pela amizade, companheirismo e companhia em todos os momentos.

Agradeço aos professores, não só deste curso, mas também aqueles que contribuíram para a minha formação, acadêmica e pessoal, em algum momento: sem vocês eu não estaria onde estou.

Por fim, agradeço meu orientador pela disponibilidade e pela ajuda durante o desenvolvimento deste projeto.

RESUMO

Com o crescente aumento demográfico e consequente necessidade alimentar, deve-se procurar métodos eficientes de se produzir comida. Sistemas hidropônicos são alternativas mais eficientes para plantações e a verticalização é uma solução para produzir mais em menos espaço. Para este tipo de sistema funcionar é importante acompanhar alguns parâmetros. Este trabalho propõe um sistema de horta vertical hidropônica com telemetria, onde os dados são expostos através de uma API e persistidos num banco de dados. Os dados medidos são referentes a variáveis da água e do ambiente, que são pH, eletrocondutividade, temperatura, iluminação e nível; possibilitando assim sua análise. O sistema conta com iluminação artificial e motor, que podem ser ativados pela API.

Palavras-chave: Internet das Coisas, Horta Vertical, API, telemetria

ABSTRACT

With the increasing demographic growth and consequente food necessities, urges the need to find efficient methods for producing food. Hydroponic systems are efficient alternatives to plantations and the verticalization is a nice solution to produce more in less space. So this kind of system can work, it's important to keep track of some data. This essay propose a vertical hydroponic system with telemetry to an indoor garden, where the data are exposed through an API and persisted in a database. The measured data are regarding water and ambient parameters: pH, electroconductivity, temperature, illumination and water level, enabling its analysis. The system counts with artificial illumination and water pump that can be activated through the API.

Key-words: Internet of Things, Vertical Garden, API, telemetry

SUMÁRIO

1 INTRODUÇÃO

Com o crescimento populacional, cresce o consumo de alimentos e, consequentemente, o uso de terras, o que impacta em graves problemas climáticos. Felizmente para a agricultura, é possível otimizar o uso de terreno e recursos através da verticalização.

A verticalização se dá ao empilhar o plantio, assim, num determinado terreno, pode-se multiplicar a produção em quantas vezes for possível. Para otimizar a produção, são comumente utilizadas técnicas de hidroponia, que consiste em fazer o plantio na água, e não no solo; nesta água são diluídos os nutrientes necessários para o crescimento ideal da planta.

Ao retirar a necessidade do uso de solo, é possível fazer o plantio em ambiente fechado, onde é possível controlar a iluminação, sem problemas de dias nublados e não haver luz o suficiente para o desenvolvimento da planta; se proteger do clima, sem problemas com estiagem, calor extremo ou chuvas fortes e manter longe também pragas, eliminando os custos com agrotóxicos e produzindo um vegetal muito mais saudável.

Por ser feita num ambiente fechado, é possível aproximar a agricultura de seus consumidores, retirando-a do contexto rural e trazendo-a ao meio urbano, facilitando a logística e diminuindo os custos.

Quando feito em grandes escalas, chama-se Fazenda Verticail. Um exemplo de sucesso é a *PinkFarms*, que fica em São Paulo, cujo nome foi inspirado pela iluminação artificial utilizada, como mostra a figura 1.

Em menor escala, chama-se Horta Vertical, que pode ser feita em casa ou num apartamento, dentro ou num quintal.

Figura 1: *PinkFarms*, um exemplo de fazenda vertical na cidade de São Paulo

Fonte: ISTOÉ DINHEIRO, 2020

1.1 Objetivos

1.1.1 Objetivos gerais

Este trabalho tem como objetivo construir uma horta vertical hidropônica de dois andares, com iluminação artificial e sistema de telemetria, para medir valores de pH, eletrocondutividade, iluminação, temperatura e nível da água.

Os dados medidos devem ser persistidos num banco de dados e expostos através de uma API. Por ela também é possível controlar iluminação e outros periféricos, como o motor aquático.

1.1.2 Visão geral do trabalho

Foi montada uma horta hidropônica com canos PVC onde existem dois andares, cada um com 3 buracos para plantio. Para a hidroponia, é necessário um reservatório de água, que é um balde de 8 litros; nele fica a água já com nutrientes. Dentro do balde fica um motor que leva a água até o andar mais alto da horta; a água desce do andar mais alto até o andar mais baixo, e dele devolve a água ao balde.

No balde ficam os sensores de pH, eletrocondutividade, temperatura e nível. O sensor de pH foi calibrado.. A iluminação é feita por duas fitas de LED, cada uma responsável por um andar da horta.

Ao lado do balde fica o circuito com as placas Arduino Uno e ESP8266. O Arduino é responsável pela leitura dos sensores, pois o ESP sozinho não tem portas

analógicas o suficiente. Ao ler os sensores, o Arduino envia os dados num JSON para o ESP via comunicação serial. O ESP é responsável por se conectar à internet, ao *broker* de MQTT, controlar a iluminação e o motor, receber os dados do Arduino e publicá-los no *broker*.

O *broker* utilizado é o da HiveMQ, de forma gratuita e na nuvem. Ele faz uso do protocolo MQTT para receber e enviar publicações nos tópicos. Tais tópicos são separados por objeto: /debug, /led, /motor, /data; o primeiro é responsável por checar que o ESP está conectado à internet ou broker sem problemas, mandando uma mensagem a cada 10 segundos; o segundo é responsável por ligar ou desligar a iluminação de cada andar; o terceiro liga ou desliga o motor e o último recebe um JSON com os dados dos sensores.

Além da horta, conectado ao *broker* também está a API, construída em ASP.NET. Ela é responsável por pegar os dados do *broker*, gravar no banco de dados e expor os dados através de endpoints, também sendo uma interface de controle para a iluminação e motor.

O Banco de Dados foi feito no sistema gerenciador PostgreSQL, com uma tabela que armazena os dados lidos pelos sensores e a hora em que foram gravados, no formato de tempo UNIX; e outra que armazena o histórico de culturas plantadas na horta, guardando seu nome, descrição e data de início e fim.

A API expõe os dados de tais tabelas através de *endpoints* HTTP de GET, POST, PUT ou DELETE como data/getall, data/getrange, cultivation/get, cultivation/add, cultivation/update, cultivation/delete e etc..

Pela API também é possível controlar a iluminação pelo *endpoint led* e o motor pelo *endpoint motor*, em tais *endpoints* o estado desejado é passado pela URL (exemplo: *motor/off*). Esses *endpoints* publicam nos tópicos do *broker* MQTT.

1.2 Revisão bibliográfica

Motivados pela mesma preocupação em torno da escassez de recursos como água e terras para a produção de alimentos, os autores do artigo "*Vertical Farming Monitoring System Using the Internet of Things"* desenvolveram um sistema de telemetria para uma fazenda vertical utilizando um microcontrolador BeagleBone Black para captar os dados e enviar à plataforma IoT *Thingspeak* para análise e visualização. (CHIN; AUDAH, 2017)

O sistema de fazenda utilizado não é hidropônico, assim, alguns dos parâmetros medidos e controlados são referentes ao solo, tais como umidade. Outros são iluminação e temperatura. Os atuadores utilizados são os leds e um motor para a água.

O sistema é capaz de automatizar a rega e a quantidade de iluminação, assim como prover dados sobre as plantas para posterior análise. Citam ainda que poderia ser melhorado adicionando sensores de pH, carbono e qualidade do ar.

No trabalho de conclusão de curso “Agricultura 4.0: Protótipo de um *Internet of Things”* na cultura da *Lactuca sativa* (alface), a autora Marilia Neumann Couto é motivada pelos avanços tecnológicos no setor da agricultura e cita a agricultura de precisão, onde são utilizados drones e IoT para monitoramento dos plantios, otimizando processos e extraindo dados relevantes quanto a eles. (NEUMANN COUTO, 2019)

No seu trabalho foi desenvolvido um protótipo com Arduino UNO e um ESP8266 junto a um aplicativo Android para monitorar o plantio de alface num sistema hidropônico.

Em seu sistema, a autora mediu os valores do pH utilizando o sensor pH4502c, a temperatura e a umidade do ar utilizando um DHT22 e a temperatura da água utilizando um DS18B20. Para isso, ela utilizou a estufa de um produtor rural, com o objetivo de acompanhar o crescimento das plantas.

O circuito foi montado na propriedade do produtor, próximo ao tanque de água. O ESP8266 foi responsável por publicar os dados no aplicativo Android, onde era possível ver gráficos dos parâmetros. Ele foi desenvolvido na plataforma *Blynk.com* que permite o desenvolvimento simplificado de um aplicativo para fins de análise de dados coletados por microcontroladores, fornecendo até um servidor e uma biblioteca para o desenvolvimento.

Ao final de 30 dias da implementação na propriedade a autora entrevistou o produtor de alfaces para saber em como a aplicação havia o ajudado, ao qual respondeu que acessava o aplicativo de duas a três vezes ao dia para monitorar os parâmetros, entendendo melhor como se comportava a adubação da água e identificando a utilização da tecnologia como uma vantagem competitiva e sugerindo que a eletrocondutividade da água fosse também monitorada numa possível melhoria.

No artigo “Sistema para Monitoramento de Parâmetros Físicos e Químicos de Corpos D’Água”, os quatro autores são motivados pelo crescente uso de tecnologias e IoT no mundo, assim, desenvolveram uma solução que monitora parâmetros da água como o pH, temperatura, turbidez e total de sólidos dissolvidos, numa arquitetura dinâmica possível de ser adaptada a sistemas que precisem monitorar tais parâmetros. Os dados coletados pelo sistema são publicados e armazenados num site público. (VIDAL MACHADO et al., 2020)

Foram utilizados sensores digitais e um ESP8266 para o desenvolvimento do trabalho. O site foi desenvolvido em Ruby e disponibilizado através da plataforma Heroku. Os objetivos foram atingidos ao final do projeto.

Em seu trabalho de conclusão de curso, Bruno Vieira Rosa se preocupa com o aumento da demanda de alimentos e a necessidade de produzi-los de forma mais sustentável, seguindo o sistema hidropônico como uma boa solução, uma vez que este reduz em até 90% o uso da água na plantação de hortaliças e vegetais. (VIEIRA ROSA; DE OLIVEIRA, 2019)

Assim, ele propõe o "*HydroSys*, uma Solução de Baixo Custo para a Visualização de Dados Aplicada à Sistemas Hidropônicos". O sistema utiliza *hardware* barato para aferir os valores de pH, temperatura e umidade de sistemas hidropônicos em tempo real e os expor *online* e acessá-los através de dispositivos móveis.

Para isso foi utilizado um microcontrolador ESP8266 e os sensores de pH PH4502C, de temperatura e umidade do ar DHT18B20 e o de temperatura da água DS18B20. Os dados foram enviados para uma API desenvolvida em Node.js e os dados persistidos no banco de dados não relacional *MongoDB*. O cliente Android foi desenvolvido com a plataforma Blynk e a *web* em VueJS.

Ao total, o circuito completo custou em torno de 150 reais. O ESP se comunica diretamente com a API, fazendo POSTs a cada leitura de dados. A API é restrita aos clientes desenvolvidos para o trabalho, assim, uma futura melhoria identificada foi a de permitir o seu uso por terceiros para análise de dados.

Por último, no artigo "*An IoT Platform for Urban Farming", os* autores são motivados pela escassez de recursos e aumento da necessidade de alimentos pela população, propondo o uso da agricultura de precisão para otimização do uso de recursos. Neste sistema são medidos o pH, total de sólidos dissolvidos, potencial de redução de oxigênio e temperatura, além do monitoramento do crescimento ser feito com uma câmera. O sistema utiliza o MQTT para se comunicar com uma plataforma IoT na nuvem. (TAN et al., 2020)

O sistema foi instalado num ambiente hidropônico NFT (*Nutrient Film Technique*). O microcontrolador utilizado foi o Arduino UNO e um Raspberry Pi. O Raspberry se comunica com o sistema IoT na nuvem da *Amazon Web Services*, enviando os dados dos sensores e para serem persistidos e as imagens para serem processadas. Além disso, há uma bomba d'água ligada ou desligada pelo Raspberry. O sistema tem clientes *frontend web* e *mobile* que mostram os dados coletados.

Neste sistema há a intenção de futuramente incluir um sistema de predição de colheita com inteligência artificial, além de melhorias no algoritmo de predição de imagem, para incluir plantas de folhas não verdes.

1.3 Organização da monografia

A monografia deverá discorrer sobre cada etapa do desenvolvimento:

- Capítulo 2: **Hidroponia,** descreve sobre técnicas de hidroponia e a montagem da estrutura física da horta;
- Capítulo 3: **Circuito IoT,** introduz conceitos sobre IoT, descreve as tecnologias utilizadas no circuito montado e detalha a montagem do circuito;
- Capítulo 4: **Calibragem dos sensores**, explica rapidamente sobre cada parâmetro medido, como é feito o sensor responsável por sua medição e como foi o processo de calibração do sensor;

- Capítulo 5: **API**, discorre sobre as tecnologias utilizadas no processo e documenta de forma detalhada o código da aplicação e como utilizá-la.
- Capítulo 7: **Conclusão**, discorre sobre o que foi aprendido no trabalho e melhorias possíveis.

2 HIDROPONIA

Segundo o dicionário online Dicio, hidroponia trata-se da “técnica de cultivar plantas em meio aquoso, com auxílio de um suporte que as sustente, providenciando os nutrientes necessários para o seu desenvolvimento”. (DICIO, 2022)

Ou seja, uma técnica de cultivo de plantas que independe do solo, utilizando em seu lugar, um meio aquoso enriquecido com os nutrientes necessários para a planta.

Além da hidroponia, existem outras técnicas de cultivo de plantas sem o uso de solo, tais como a aeroponia e aquaponia. A primeira trata-se, no fundo, de uma técnica hidropônica, onde são utilizados aspersores para manter somente as raízes úmidas; a água é enriquecida da mesma forma que na hidroponia. Esta é uma técnica bastante econômica e se adapta melhor para espécies específicas. A segunda atende a duas necessidades diferentes, onde as plantas crescem na mesma água onde são criados peixes; a água é enriquecida com os resíduos orgânicos dos peixes, que são utilizados para nutrir as plantas; o sistema é altamente sustentável e diminui drasticamente o uso de água para a criação de peixe e hortaliças.

O uso de hidroponia data das antigas civilizações, onde combinavam o uso de água abundante com a natureza, como os Jardins Suspensos da Babilônia e os jardins flutuantes dos Astecas. Passou a ser estudada em cerca de 1930, quando é mais popularizada; durante a Segunda Guerra Mundial o exército dos Estados Unidos utilizou a técnica em ilhas do Pacífico para alimentar suas tropas. Desde os anos 80 a técnica é utilizada comercialmente e vem ganhando mais força a cada ano.

Para o futuro, o que espera a hidroponia são a possibilidade de utilizá-la no espaço, em áreas desertas ou urbanas.

No livro Guia completo para cultivar plantas na hidroponia, o autor cita diversos benefícios da técnica, tais como a possibilidade de cultivar em áreas com solo infertil ou contaminado; eliminação do trabalho de irrigar, lavrar, usar agrotóxicos e outras técnicas necessárias ao solo; maximização da colheita e da área utilizada; economia de água e nutrientes e redução de poluição; evita doenças de solo e facilita a erradicação e tratamento de outras doenças; possibilidade de maior controle do ambiente como luz, temperatura, umidade e etc.; o pH da solução pode ser controlado para otimizar a absorção de nutrientes pelas plantas; não há perda de nutrientes por lixiviação; o sistema é leve, limpo e pode ser mecanizado e mantido dentro de uma casa, pátio, estufa ou prédios. (BENTON JONES, 2014)

O autor também traz algumas desvantagens da técnica, tais como a necessidade de que quem lide com ela saiba como plantas crescem e os princípios de nutrição; algumas doenças e nematóides podem se espalhar com facilidade

dentro de um tanque fechado; algumas plantas ainda necessitam de pesquisa para saber como se desenvolvem num sistema hidropônico; a reação das plantas à boa e má nutrição são muito rápidas, portanto há a necessidade de observação constante da cultura.

2.1 Como crescem as plantas

No século XVI, van Helmond experimentava para saber mais sobre o crescimento das plantas. Plantou um salgueiro num tubo de solo cuidadosamente pesado, onde ele apenas regava a árvore. Ao fim do experimento, notou que o salgueiro cresceu de 2 para 76 quilos, mas a terra do vaso perdeu míseros 50 gramas. Como adicionou apenas água ao solo, concluiu que o crescimento foi produzido pela água (BENTON JONES, 2014).

Mais tarde no mesmo século, John Woodward plantou hortelãs em diferentes soluções de água com terra, e notou que as plantas cresciam mais conforme a quantidade de terra adicionada à água aumentava (BENTON JONES, 2014) .

Com tais conclusões e o avanço na química e biologia, questiona-se quais são os elementos necessários para o crescimento das plantas e por onde as plantas os captam. Assim, Benton Jones discorre em seu livro sobre como crescem as plantas.

Com a fotossíntese, há a conversão de luz em energia química para transformar dióxido de carbono (CO_2) e água (H_2O) em glicose ($C_6H_{12}O_6$) e oxigênio (O_2), onde a glicose é utilizada pela planta e o oxigênio liberado. A água é captada pelas raízes e o dióxido de carbono pelos estômatos nas folhas. Assim, sabe-se que a planta necessita de pelo menos carbono (C), oxigênio (O) e hidrogênio (H).

Mas não só disso vive a planta. A partir da conclusão de que as plantas obtêm carbono do ar e minerais pelas raízes, os esforços das pesquisas foram direcionados a descobrir quais os minerais presentes no solo e que são absorvidos.

Hoje em dia sabe-se que para a planta crescer há a necessidade de potássio (K), magnésio (Mg), cálcio (Ca), ferro (Fe), fósforo (P), enxofre (S), carbono (C), nitrogênio (N), hidrogênio (H), oxigênio (O) e etc.. Para absorver alguns dos nutrientes, a planta utiliza suas folhas, mas a maioria dos minerais não está no ar.

As raízes têm duas funções principais: sustentação e absorção de nutrientes. A raiz tem diversos tipos, variando de planta para planta e seu meio de crescimento. Na hidroponia, é comum que as raízes cresçam muito mais que no solo e seu crescimento é fortemente atrelado à quantidade de glicose suprida e a necessidade e capacidade da planta em encontrar seus nutrientes e água no meio de crescimento.

Para captar melhor alguns nutrientes, algumas raízes desenvolvem capilares. Quando os nutrientes estão em abundância no meio, não há a necessidade, assim, em meio hidropônico, é incomum que capilares existam.

Como mencionado anteriormente, é comum que as raízes cresçam muito no meio hidropônico, assim, é importante se atentar a isso, já que o excesso de raízes pode influenciar no meio de crescimento, diminuindo o oxigênio na água, a captação de nutrientes e até o fluxo de água, prejudicando as plantas e o sistema inteiro.

Os nutrientes absorvidos pelas plantas vão definir a sua saúde e seu formato, a falta de nutrientes no meio faz a planta perder seu formato e saúde. A planta murcha quando sua raiz não consegue captar água o suficiente para a manter cheia. Plantas do campo são mais resistentes à falta de água, enquanto plantas de ambientes controlados são mais sensíveis.

A água é puxada pelo xilema (tecido responsável por levar água e minerais das raízes às folhas da planta) e perdida na transpiração e fotossíntese feita pelas folhas. Assim, se há aumento de temperatura, a planta não quer que a energia solar seja desperdiçada com aquecimento e sim aproveitada para a fotossíntese, o que aumenta a transpiração da planta, para resfriar suas folhas. Isso pode fazer com que ela murche quando não houver mais tanta água disponível no meio.

Ao contrário, quando há declínio na temperatura, há também declínio no consumo de água pela planta. Também há queda no consumo de água quando o meio das raízes não tem oxigênio o suficiente ou aumento de íons. A temperatura demasiado elevada ou baixa pode afetar o crescimento da raiz, sendo o ideal em torno dos 20ºC.

A oxigenação da água é também afetada pela temperatura, sendo possível dissolver mais oxigênio na água em temperaturas mais baixas. A falta de oxigênio também prejudica o crescimento das raízes.

As raízes são capazes de alterar seu ambiente também, aumentando o pH ao liberar íons de hidrogênio (H^+). Esta habilidade pode ser prejudicada em meios hidropônicos, onde o pH é sempre checado e controlado. Assim, o pH do meio também é importante na saúde da planta.

2.2 A solução nutritiva

Segundo J. Benton Jones, em seu livro *Complete Guide for Growing Plants Hydroponically*, é difícil saber com o que alimentar e em que quantidade disponibilizar à planta. Embora existam os nutrientes essenciais às plantas, em geral, cada tipo tem suas especificidades. (BENTON JONES, 2014)

Não há receita de solução, é necessário testar algumas possibilidades em cada sistema para saber. Embora algumas soluções pareçam boas, já que algumas plantas são bem adaptativas e conseguem crescer mesmo com deficiência de algum nutriente ou saturação de outro, elas podem não ser perfeitas, impedindo que a planta cresça em seu potencial genético.

Embora não haja um caminho certo, existem diversas pistas de como preparar a melhor solução. É necessário pensar em qual técnica de hidroponia se

utiliza (seção. 2.4), frequência e proporção da dosagem da solução e os requisitos de nutrição da planta.

É preciso prestar atenção à qualidade da água onde as plantas ficam, pois podem haver componentes orgânicos e inorgânicos que afetam positiva ou negativamente a planta.

Alguns parâmetros podem ser observados para garantir uma qualidade mínima da água (sem nutrientes dissolvidos), como por exemplo a eletrocondutividade deve ser menor que 0.75 dS/m, os sólidos dissolvidos totais não devem passar de 480 mg/L e etc..

O pH também deve ser checado, mas é mais complicado de se medir, já que a água pode absorver CO_2, deixando maior quantidade de íons H^+ na solução. Deve-se filtrar para garantir que não haja sólidos nem organismos.

A maioria das soluções hidropônicas contém alguns elementos básicos: nitrato de cálcio, nitrato de potássio, fosfato de potássio diidrogenado e sulfato de magnésio.

No livro *Complete Guide for Growing Plants Hydroponically* de J. Benton Jones Jr. são passadas algumas soluções nutritivas básicas para a hidroponia no capítulo sobre a solução nutritiva; algumas delas estão descritas no quadro 1:

Quadro 1: soluções padrão para a hidroponia

Solução padrão	Quantidade (ml/L)
Solução 1	
fosfato de potasio diidrogenado (KH_2PO_4)	1
nitrato de potássio (KNO_3)	5
nitrato de cálcio ($Ca(NO_3)_2 4H_2O$)	5
sulfato de magnésio ($MgSO_4 7H_2O$)	2
Solução 2	
fosfato de amonio diidrogenado ($NH_4H_2PO_4$)	1
nitrato de potássio (KNO_3)	6
nitrato de cálcio ($Ca(NO_3)_2 4H_2O$)	4
sulfato de magnésio ($MgSO_4 7H_2O$)	2
Solução padrão de micronutriente	
ácido bórico (H_3BO_3)	2.86

cloreto de manganês ($MnCl_24H_2O$)	1.81
sulfato de zinco ($ZnSO_45H_2O$)	0.22
sulfato de cobre ($CuSO_45H_2O$)	0.08
ácido molibdênio ($H_2MoO_4H_2O$)	0.02

Fonte: (BENTON JONES, 2014)

As quantidades na solução podem se adaptar às necessidades do sistema.

Depois de pronta, a solução deve ser monitorada e controlada. Existem dois tipos de gerenciamento da solução nutritiva: aberto e fechado. O primeiro designa o sistema cuja solução passa uma única vez pelas raízes; o segundo, aquele em que a solução é recirculada após passar pelas raízes.

Depois de feita a solução, alguns parâmetros são medidos, como o pH, eletrocondutividade e sólidos dissolvidos totais (TDS). O primeiro deve ser mantido num nível ideal para que as plantas não sejam prejudicadas, o segundo e o terceiro determinam a quantidade de nutriente ainda presente na solução.

Para o crescimento ideal das plantas, o pH deve estar dentro da faixa de 5 a 7,5, mas pode-se estreitar a faixa para ficar entre 5,8 e 6,5. Quando fora da faixa, é necessário adicionar uma substância ácida ou alcalina para abaixar ou aumentar o valor do parâmetro. (BENTON JONES, 2014)

Pode-se utilizar uma solução de soda cáustica (NaOH) ou potassa cáustica (KOH), como soluções alcalinas para aumentar o pH; e ácidos como o hidroclórico (HCl) ou o sulfúrico (H_2SO_4), pode-se utilizar outros como o nítrico (HNO_3), mas ele adiciona à solução anios de nitrato (NO_3^-).

A mudança de pH pode vir de diversas fontes: trocas gasosas da água, fonte de nitrogênio da planta (NO_3^- ou NH_4^+), falta de nutrientes, espécie da planta e estágio de seu crescimento (lembrando que as raízes liberam íons H^+, algumas plantas podem liberar mais que outras, ou liberar mais em determinada fase de seu crescimento).

A temperatura da solução também é importante de ser observada, já que não devem ser muito menores que a temperatura do ar, já que diferenças demasiadamente altas podem causar *stress* na planta. Temperaturas demasiado baixas podem retardar o crescimento das plantas, gerar frutos pobres e uma planta de baixa qualidade. Temperaturas muito altas podem desoxigenar a água e causar problemas às plantas.

Os valores ideais variam de planta para planta, algumas conseguem se adaptar melhor ao clima de determinados lugares que outras. Para controlar a temperatura é necessário isolar o sistema do clima e utilizar de aquecedores e resfriadores para manter uma determinada temperatura ideal, porém isso é demasiado custoso e complexo. Assim, basta usar plantas adaptadas ao clima da região em que o sistema está implantado, pois estas não hão de ter problemas.

A eletrocondutividade pode afetar o crescimento também. Alguns dos valores em soluções nutritivas são baixos: menores que 3 dS/m. É preciso tomar cuidado com a evaporação da água da solução, pois aumenta-se a porcentagem de sais dissolvidos, e se essa água não for reposta, a eletrocondutividade pode aumentar drasticamente.

É importante prestar atenção na oxigenação da solução: utilizar borbulhadores de oxigênio podem ajudar, mas a absorção do gás pela água pode variar com a temperatura.

Ainda segundo o autor, é muito complicado ou quase impossível manter uma constância precisa de nutrientes dissolvidos na solução. Assim que cada vez que nutrientes são adicionados novamente, estes se misturam aos restos presentes na solução. A quantidade de vezes em que os nutrientes devem ser repostos varia de acordo com a quantidade de plantas e as espécies plantadas. (BENTON JONES, 2014)

2.3 Meio de enraizamento

Existem diversos sistemas hidropônicos (seção 2.4) onde cada um tem um tipo diferente de distribuição de água e nutrientes. Para cada um, deve-se pensar em como a planta será sustentada e em que meio suas raízes irão crescer.

Cada meio de enraizamento é mais adequado a um tipo de sistema, assim, serão listados apenas aqueles que servem para o sistema *NFT* (*Nutrient Film Technique* ou Técnica do Filme de Nutriente), também descrito na seção 2.4.

Para a técnica *NFT* é necessário:

- isolar cada planta, assim, uma planta não compartilha o mesmo torrão que outra, sendo o torrão o pedaço de substrato em que a planta está enraizada;
- que o substrato não se dissolva na água ou que desmonte, para isso podem ser utilizados “copos” que segurem o torrão;
- que a raiz toque no filme de água.

Assim, alguns bons substratos são a lã de rocha, fibra de coco, argila expandida e cascalho. Outros exemplos podem se soltar do torrão com facilidade.

A lã de rocha é um material fibroso produzido pela mistura de rochas vulcânicas, calcário e coca. São produzidas folhas dessa fibra e com estas são feitas mantas de 5 a 10 cm de altura. As mantas são postas na área onde a água irá passar e acima delas, são feitos cubos de plantio. É um meio muito utilizado para tomates e pepinos, tem uma ótima capacidade de retenção de água e é um ótimo substrato, além de ser neutro e não fazer trocas de cátions. Pode ser reutilizado e tem ótima absorção de água e aeração. Exemplos podem ser vistos nas figuras 2 e 3.

Figura 2: Manta de lã de rocha vista de perto

Fonte: EDUCALINGO (2021)

Figura 3: Manta de lã de rocha para plantio

Fonte: AMAZON (2021)

A fibra de coco é uma substituição orgânica para a lã de rocha, é feita do material fibroso da casca e do interior do côco e, por ser orgânica, seu descarte após o uso é mais ecológico. O material é usado para formar mantas, tal como a lã de rocha e compartilha dos mesmos benefícios desta. Um exemplo pode ser visto na figura 4.

A manta de côco deve ser descartada após o uso, pois as raízes nascem no meio da fibra, o que pode interferir na solução nutritiva de um novo cultivo, inserindo material orgânico indesejado.

Figura 4: Manta de fibra de coco

Fonte: PLANTEI (2021)

A argila expandida são pequenas esferas feitas de argila, bastante leves, com boa drenagem e acumulação de água e nutrientes. É facilmente reutilizada, basta esterilizá-las. Muito comum em sistemas de hidroponia que utilizam vasos ou copos. Exemplo na figura 5.

Figura 5: Argila expandida

Fonte: TRAY TECNOLOGIA (2021)

Os cascalhos são pedaços de pedra, com boa drenagem e baixa absorção de água. São pesados, o que pode apresentar uma vantagem ou desvantagem. Para ser utilizados, é necessário esterilizar. Exemplo na figura 6.

Figura 6: cascalhos

Fonte: OZINGA (2021)

Cada meio de enraizamento tem alguma vantagem ou desvantagem ou formas diferentes de se utilizar, além de preços e pesos diferentes. Assim, é necessário analisar qual o melhor para o sistema utilizado.

Cada um tem elementos essenciais às plantas, em diferentes proporções, mas a quantidade existente é demasiado difícil de aferir. Assim, os elementos do meio de enraizamento interagem com a raiz da planta e a solução, num geral a associação pode ser benéfica, mas pode também prejudicar o crescimento das plantas.

Além disso, o substrato pode acumular nutrientes nele, causando prejuízos significativos à planta. Por isso é importante sempre medir o pH e a eletrocondutividade da solução. Em alguns casos, quando há transgressão da faixa ideal de um desses parâmetros, pode ser necessário lixiviar o substrato, para retirar os nutrientes em excesso.

Os parâmetros ideais variam de acordo com o substrato e a planta, mas pode-se manter os citados na seção 2.2.

2.4 Sistemas de hidroponia

Para que as plantas cresçam, é necessário que a água da hidroponia e o meio de enraizamento sejam juntados de alguma forma. Para solucionar esse problema existem diversos sistemas, assim, no sexto capítulo de seu guia para hidroponia, Benton Jones descreve em detalhes alguns dos principais. (BENTON JONES, 2014)

O primeiro e mais simples é o *deep water*, onde o meio de enraizamento é a própria água. As plantas são colocadas diretamente num reservatório com solução nutritiva. Neste caso a solução deve ser aerada e trocada periodicamente, por isso pode ser demasiado custoso para aplicações comerciais.

Outra técnica é a do filme de nutriente, ou *NFT* (*Nutrient Film Technique)*, onde as plantas crescem num fluxo de solução nutritiva. A solução é bombeada nos meios de crescimento, passando pelas raízes das plantas, podendo ou não haver substrato como meio de enraizamento. A solução sai de um reservatório e retorna a ele depois de passar pelas raízes, assim, é possível economizar sem a necessidade de trocar a solução periodicamente, mas apenas medindo seus parâmetros e ajustando-os.

O fluxo de água recomendado é de aproximadamente 60l/h, a planta deve ser posta de forma que a raiz toque a água mas fique exposta no ar dentro do cano. Pode-se usar materiais baratos, é necessário tomar cuidado para não deixar a luz entrar no sistema, já que isso possibilita o crescimento de algas, assim o material deve ser opaco.

Neste sistema as raízes tendem a crescer bastante, o que pode causar falta de oxigenação no sistema, fazendo com que raízes morram rapidamente. Além disso, o controle de doenças é complexo, já que pode passar de planta a planta facilmente.

Existe também a aeroponia, onde as raízes das plantas ficam em contato com a solução nutritiva que é constantemente borrifada, criando um ambiente úmido. Neste tipo de sistema há bastante aeração e economia de água e nutriente, porém é mais difícil acertar uma solução ideal.

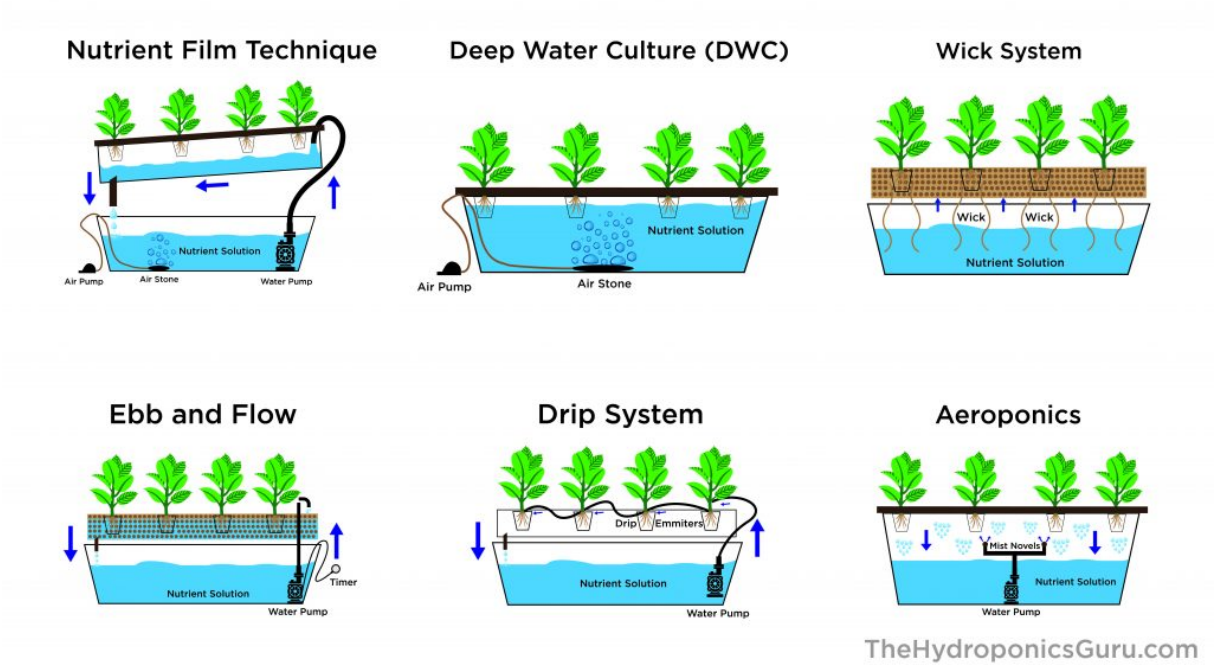
Outro método é o *flood and drain* ou *ebb and flow*, que consiste em inundar uma caixa, onde as plantas estão enraizadas num substrato inorgânico, e depois drenar a solução periodicamente. Este sistema não é mais utilizado comercialmente há alguns anos pois é muito suscetível a doenças nas raízes, o uso de água e nutrientes é ineficiente e há a necessidade de repor o meio de enraizamento periodicamente.

Há também o método de *drip,* onde a solução nutritiva é distribuída através de gotejamento. As plantas são mantidas em saquinhos de substrato inorgânico, geralmente com furos para escoar o excesso de água, onde recebem a solução nutritiva. O excesso é escoado de volta ao reservatório.

Por último, existem os sistemas de pavio (*wick*), onde a planta é colocada numa cama de substrato, que é ligado a pavios que tocam o substrato. A solução nutritiva “sobe” pelos pavios e chega às raízes. São chamados passivos pois não têm partes que se movem, não utilizam eletricidade nem gravidade para funcionar. Apesar de baratos e simples, são ineficientes na distribuição de água e nutrientes.

Os sistemas citados acima são ilustrados na figura 7.

Figura 7: Sistemas de hidroponia



Fonte: THE HYDROPONICS GURU (2021)

Além dos citados anteriormente, existem outros sistemas como o de coluna ou o de subirigação, que propõe um modelo vertical de plantação. O sistema *NFT* pode ser verticalizado facilmente, fazendo com que o fluxo passe por mais de um andar de plantio.

2.5 A aplicação

Para a aplicação da técnica de hidroponia foi escolhido o sistema *NFT*, montado com dois canos PVC, formando dois andares, onde cada um tem três buracos para o plantio, como mostra a figura 8.

Figura 8: Estrutura da horta montada

Fonte: Elaborado pelo autor

Para o tanque com a solução nutritiva foi usado um balde de oito litros. Dentro dele, um motor aquático com capacidade de fluxo de 800L/h visto na figura 9, leva a água para o andar mais alto, o andar mais baixo despeja a água de volta ao tanque, como mostra a figura 8.

Figura 9: Motor de fluxo 800L/h

Fonte: ALIEXPRESS (2021)

Para meio de enraizamento das plantas, foi utilizado um copo plástico com furos que permitam a entrada de água e argila expandida para dar sustentação à planta e filtrar a água, como mostra a figura 10.

Figura 10: Copo plástico com furos, argila expandida e muda de jibóia

Fonte: Elaborado pelo Autor

3 CIRCUITO IOT

Segundo o livro Princípios da Internet das Coisas, editado por Rajkumar Buyya e Amir Vahid, Internet das Coisas (IoT - *Internet of Things*) trata-se de um paradigma para conectar “coisas” à *internet.* Estas “coisas” são quaisquer objetos capazes de comunicar pela rede, como dispositivos inteligentes (como um *smartphone*), sensores, atuadores, entre outros. (BUYYA; VAHID DASTJERDI, 2016)

Este paradigma abre a possibilidade de novas interações entre humanos e seus objetos, possibilitando a concretização de cidades inteligentes e melhora na qualidade de vida e utilização de recursos.

Os itens que constroem uma solução IoT são sensores, invocação de serviços remotos, redes de comunicação e processamento de eventos de acordo com o contexto. Para que uma solução funcione, é necessário pensar numa arquitetura.

Existem basicamente duas: SOA e API. A primeira é a *service oriented architecture*, onde tenta se garantir a interoperabilidade de diversos dispositivos, onde um sistema complexo é quebrado em diversos sub-módulos individuais e desacoplados, pensando em sensores e dispositivos, *backend* e *frontend*; a segunda utiliza de API para se comunicar com os dispositivos, através dos endpoints, passando ou recebendo deles arquivos de dados leves (como um JSON), a API roda num *backend*, onde os dados recebidos são armazenados, o uso de API para IoT atrai consumidores pois são simples de utilizar e focam nas funcionalidades do produto ao invés de uma apresentação, além de ser mais barata.

De forma geral, uma aplicação IoT consta com a camada de dispositivos e a camada de *backend,* onde a primeira gera dados através de sensoriamento ou tem dispositivos controláveis; e a segunda recebe e lida com os dados do sensoriamento e do usuário (no caso de controle remoto de dispositivos).

Para isto, é necessária a troca de mensagens entre os dispositivos, o *backend* e o usuário, assim, é interessante uniformizar o formato de se enviar e receber uma mensagem, utilizando assim um protocolo de comunicação.

Tais protocolos estão em nosso dia a dia quando se acessa um *site* na *internet* - ao entrar num link fazemos uma requisição GET do HTTP, ao enviar algo para um link, por exemplo, um formulário, fazemos uma requisição POST do HTTP.

O HTTP é o protocolo mais utilizado para lidar com o acesso a *sites* e também em APIs. Para o IoT existem vários protocolos, sendo cada um melhor para algum tipo de aplicação.

O mais comum, por ser leve e rápido, permitir o desacoplamento de dispositivos e fácil integração com outras aplicações é o MQTT.

3.1 MQTT

MQTT é um protocolo de transporte de mensagem entre um cliente e um servidor, utilizando o padrão de publicar e inscrever (*publish/subscribe)*. Foi projetado para ser leve e simples de se utilizar e implementar, tornando-o perfeito para o uso em IoT, onde é ideal que se utilize a menor quantidade de código (pouca memória e armazenamento dos microcontroladores).

O protocolo foi desenvolvido em 1999 na IBM, onde necessitavam de um protocolo que utilizasse o mínimo de bateria e funcionasse com a menor largura de banda. Definiram então os requisitos do protocolo: fácil implementação; entrega de dados de acordo com a qualidade do serviço escolhida; leve e eficiente em diversas larguras de banda; agnóstico em dados e consciência de sessão contínua.

MQTT não é um acrônimo como muitos pensam, é apenas o nome do protocolo. Antigamente era acrônimo para *MQ Telemetry Transport*, onde MQ se refere aos produtos da série MQ da IBM. Muitos pensam que se trata de um protocolo de fila de mensagens (*Message Queue)*, mas, apesar de haver a possibilidade de enfileirar mensagens, o MQTT não é uma solução de fila de mensagens.

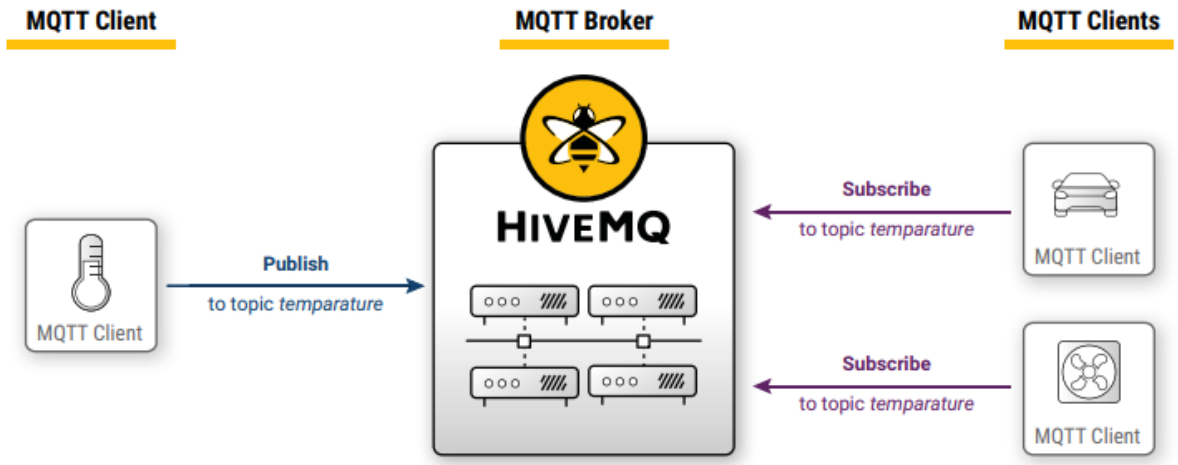
Hoje em dia o MQTT é livre de *royalties* e aberto para que qualquer pessoa desenvolva sua própria solução. A partir daí surgiram diversas implementações de servidores conhecidas, como a HiveMQ, Mosquito, AWS IoT e etc..

A comunicação entre o servidor e o cliente no MQTT se dá por meio do padrão *pub/sub* (*publish/subscribe*). O servidor é geralmente chamado de *broker*. O cliente lida diretamente com um *endpoint* do servidor, onde ele pode publicar ou se inscrever num determinado tópico. Isso dissocia os clientes que enviam mensagens dos que as recebem, intermediando a conexão entre eles com o servidor, cujo trabalho é filtrar as mensagens recebidas e distribuí-las corretamente aos inscritos. O fluxo de mensagens é exemplificado na figura 11.

A dissociação dos *publishers* e dos *subscribers* se dá em três dimensões: espaço, onde os clientes não precisam saber um do outro (sem troca de endereços IP); tempo, onde os clientes não precisam rodar ao mesmo tempo; sincronização, onde operações em ambos não precisam ser interrompidas durante a publicação ou recepção.

As operações no broker podem ser facilmente paralelizadas e as mensagens podem ser processadas como eventos, o que aumenta a escalabilidade do protocolo.

Figura 11: exemplo de fluxo *pub/sub*



Fonte: HIVEMQ (2021

O *broker* filtra as mensagens para os inscritos, podendo fazê-lo de três formas: assunto, onde as mensagens são filtradas pelo assunto ou tópico que a mensagem faz parte, assim, um cliente se inscreve num tópico de interesse e só recebe as mensagens publicadas em tal tópico. Um exemplo de tópico é "/casa/temperatura"; conteúdo: as mensagens são filtradas de acordo com seus conteúdos, utilizando filtros de linguagem; tipo: quando utilizadas linguagens orientadas a objeto, pode-se filtrar por tipo ou classe da mensagem. Exemplo, um cliente pode ouvir apenas às classes *Exception".*

Retomando, os clientes MQTT são aqueles que publicam e se inscrevem em tópicos, pode ser qualquer dispositivo com acesso à internet que rode uma biblioteca MQTT e se conecte a um *broker* MQTT. O *broker* MQTT é o ponto central do protocolo, dependendo da implementação, pode lidar com milhares de conexões por vez.

Uma conexão MQTT é sempre entre um cliente e o *broker*. Para iniciar a conexão, o cliente envia uma mensagem de *CONNECT* ao endereço público do *broker,* que a responde com um *CONNACK* e um código de *status*. A conexão se mantém aberta até que o cliente envie um comando de desconexão ou conexão quebre por alguma falha.

O *CONNECT* tem alguns parâmetros que podem ser utilizados, como *clientId* (identificador do cliente), *username* e *password* (quando há a necessidade de autenticação do cliente), *lastWill* (define-se a mensagem enviada quando o cliente se desconecta, o tópico onde será enviada e a Qualidade do Serviço) e etc..

Assim que se conecta, o cliente pode enviar mensagens *PUBLISH*, que conterá os parâmetros *topicName* (o nome do tópico onde a mensagem será publicada), QoS (*Quality of Service*, ou qualidade do serviço, é um inteiro entre 0 e

2, indicando o nível de qualidade), *payload* (o conteúdo da mensagem em si) e outros.

Publicar uma mensagem não faz sentido se ninguém a receber, assim é possível se inscrever num determinado tópico, através do *SUBSCRIBE.* Ao enviar esta mensagem, o *broker* responde com um *SUBACK* e a partir daí começa a enviar o conteúdo publicado no tópico. É possível também se desinscrever de um tópico, com um *UNSUBSCRIBE,* com o qual o *broker* irá responder com um *UNSUBACK*.

3.2 A arquitetura da aplicação

O sistema tecnológico da horta tem três camadas intercomunicáveis: a de sensores e atuadores, a de comunicação e a de aplicação, como mostra o diagrama na figura 12.

Figura 12: Diagrama da arquitetura das camadas da aplicação

Fonte: Elaborado pelo Autor

A camada de sensores e atuadores concentra o circuito de telemetria e os atuadores. São os dispositivos responsáveis por gerar dados ou serem controlados.

O circuito de telemetria consiste no microcontrolador Arduino e os sensores de pH, eletrocondutividade (ec), temperatura e nível. O microcontrolador lê os dados de cada sensor a cada 5 segundos e ao final gera um JSON contendo o valor lido para cada um. Um exemplo de JSON segue abaixo:

```
{
  "ph": 6.5,
  "ec": 5,
  "temperature": 28,
  "level": 80
}
```

Cada parâmetro tem sua unidade, que será especificada no capítulo 4.

Após gerar o JSON, o Arduino passa seus dados ao ESP8266 via comunicação serial, que consiste em enviar um bit de cada vez, sequencialmente, através do barramento. Este passo é necessário pois o Arduino não tem acesso à

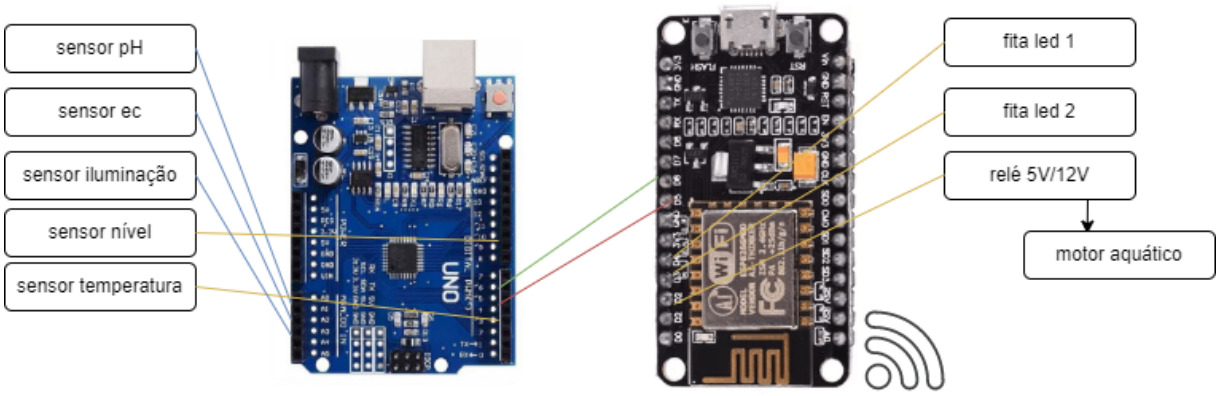
internet e o ESP8266 não tem portas analógicas o suficiente para todos os sensores.

O ESP8266 é o responsável pelo circuito de atuadores, então em suas portas digitais estão ligadas as fitas de *led* responsáveis pela iluminação e o motor aquático.

É ele também o responsável pela comunicação com a Camada de Comunicação. Ao receber os dados do Arduino, o ESP fará uma publicação no tópico "*/telemetry*", ele também está inscrito nos tópicos "*/led*" e "*/motor*", assim qualquer publicação neles refletirá no microcontrolador setando o estado dos atuadores de mesmo nome.

Uma síntese do circuito de telemetria e atuadores pode ser vista na figura 13.

Figura 13: Síntese do circuito de telemetria e atuadores.



Fonte: Elaborado pelo Autor

Na Camada de Comunicação está o *broker* MQTT. O *broker* escolhido foi uma implementação da *HiveMQ*, que oferece a facilidade de um broker seguro e já publicado na nuvem. Para se conectar a ele basta ter seu endereço e credenciais, geradas na sua criação.

Por último, na Camada de Aplicação, fica a API. Ela é responsável por persistir os dados gerados pela horta, expô-los aos seus clientes e receber dos clientes o estado desejado aos atuadores. Ela está inscrita no tópico "*/telemetry",* persistindo os dados recebidos por ele, e publica nos tópicos "*/led*" e "*/motor*", enviando até os atuadores os estados desejados.

3.3 Circuito de telemetria

O código referente a este capítulo está presente no APÊNDICE 1.

Trabalhar com sensores traz ao código uma gama de bibliotecas específicas que facilitam a programação e leitura dos dispositivos. Os sensores utilizados são

digitais ou analógicos e alguns têm seu próprio circuito e cuidados necessários na montagem e calibração.

O circuito inclui sete sensores: um sensor de pH com sonda PH4502c, um sensor de eletrocondutividade *TDS Meter V1.0*, um sensor de temperatura de dois cabos à prova d'água DS18B20*,* um fotoresistor para medir a iluminação e três sensores de nível d'água FD10. Cada sensor é ilustrado nas imagens abaixo.

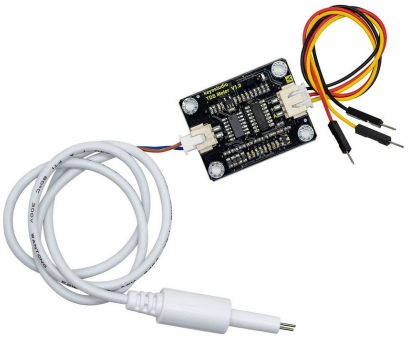
O sensor PH4502C, visto na figura 14, é o responsável por medir o pH. Ele não tem um *datasheet,* portanto os dados sabidos estão em *sites* que o vendem ou fóruns. Segundo o *site AutoCore Robótica* a sua faixa de medição varia de 0 a 14 pH, com um erro alcalino de 0.2 pH; sua temperatura de operação é de 10 a 50ºC, corrente de trabalho de 5 a 10 mA e vida útil de 3 anos. (AUTOCORE ROBÓTICA, 2022)

Figura 14: Sensor de pH com sonda PH4502C

Fonte: AUTOCORE ROBÓTICA (2022)

O sensor *TDS Meter V1.0*, visto na figura 15, é o responsável por medir a eletrocondutividade da água. Suas especificações podem ser encontradas no *site* do fabricante, *DFRobot:* o sensor tem foi feito para medir o os sólidos dissolvidos totais, um parâmetro que pode ser chegado através da eletrocondutividade, sendo assim, o *site* indica sua precisão em partículas por milhão (ppm) e não em siemens. Sua faixa de medição é de 0 a 1000 ppm, com precisão de 10% a 25ºC, sua voltagem de entrada é de 3.3 a 5.5V e trabalha na corrente de 3 a 6 mA, sua voltagem de saída é de 0 a 2.3V. (DFROBOT, 2021)

Figura 15: Sensor de eletrocondutividade *TDS Meter V1.0*



Fonte: BANGGOOD.COM (2021)

O sensor DS18B20, visto na figura 16, é o responsável por medir a temperatura da água. Em seu *datasheet* da *HK Shan Hai Group* é indicada uma faixa de trabalho de -55 e 125ºC, com uma precisão de 0,5ºC. O sensor é digital e trabalha numa voltagem de 3 a 5V. (HK SHAN HAI GROUP LIMITED, [s.d.])

Figura 16: Sensor de temperatura de dois cabos à prova d'água DS18B20

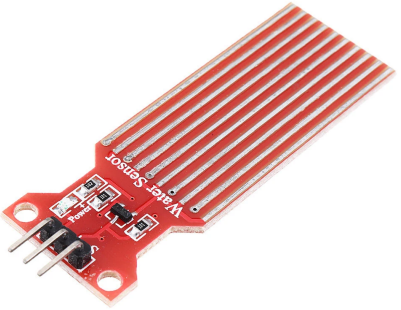
Fonte: BANGGOOD.COM (2021)

Para a iluminação é usado um fotoresistor, visto na figura 17, e para o nível são usados três FD10, visto na figura 18, colocados no balde de solução nutritiva.

Figura 17: Fotoresistor

Fonte: BANGGOOD.COM (2021)

Figura 18: sensor de nível d'água FD10



Fonte: BANGGOOD.COM (2021)

O código é dividido em seis partes: inclusão de bibliotecas, definições de portas, declaração de variáveis globais, *setup*, funções de leitura e *loop*.

O microcontrolador utilizado para o circuito é um Arduino Uno, assim, utiliza-se a linguagem C++ para programá-lo. O Arduino tem uma lógica de funcionamento específica dividida em dois módulos chamados de *setup* e *loop*, onde o primeiro é uma função que roda uma única vez, toda vez que o microcontrolador é ligado, ou seja, onde o programa começa, e tem a função de configurar as portas do Arduino; o segundo é uma função que irá rodar em *loop* infinito após o *setup*, é onde fica a lógica necessária para a leitura e escrita nas portas.

O objetivo do código é ler todos os sensores a cada 1 segundo e enviar o resultado para o ESP8266 via comunicação serial.

O primeiro passo a ser resolvido é a comunicação com o ESP: de que forma os dados serão organizados e enviados. Os dados coletados em cada *loop* são reunidos num único arquivo JSON, onde cada chave representa o parâmetro medido e o valor, o valor medido na iteração.

Para isso, foi utilizada a biblioteca ArduinoJson.h, suas funções foram utilizadas somente no *loop*, onde a cada iteração é declarado um novo arquivo JSON estático e os valores são preenchidos, a começar pela temperatura, pH,

eletrocondutividade, iluminação e nível. Para fins de desenvolvimento, o arquivo é serializado para o monitor serial do Arduino, onde o desenvolvedor possa acompanhar os valores lidos imediatamente durante o desenvolvimento.

Para enviar ao ESP8266 via comunicação serial, foi utilizada a biblioteca SoftwareSerial.h. As portas escolhidas para a comunicação são definidas na seção de variáveis globais do código: 5 e 6, onde a primeira funciona como Rx (receptora) e a segunda como Tx (transmissora). No *setup*, o objeto responsável pela comunicação serial é inicializado, com a velocidade de 9600 bits por segundo. No fim do *loop* checa-se se o ESP está disponível para receber, então o arquivo JSON é serializado.

Para fazer com que o *loop* ocorra apenas a cada 1 segundo é utilizado um *delay*, pois esta função "pausa" o programa por um tempo determinado.

Feito a base do código, é hora de se preocupar com os sensores. O primeiro a ser configurado é o de temperatura pois seu valor é utilizado na medição do pH e da eletrocondutividade. O sensor DS18B20 comprado foi o de 2 cabos, diferente do mais comumente comercializado de 3. A diferença consiste em que no de 3 cabos, a saída de dados está num cabo diferente do de alimentação (5V ou VCC) e no de 2 é feita uma ligação parasita entre eles.

O sensor é um transistor envelopado para facilitar a medição. Na ligação com o Arduino é utilizado um resistor de 4.7KΩ para conectar o fio de VCC e dados aos 5V do circuito. Entre o resistor e o cabo do sensor, é feita a ligação de dados com o Arduino.

Para ler a temperatura no código, foram utilizadas as bibliotecas OneWire.h e DallasTemperature.h. Suas classes são utilizadas na seção de variáveis globais, onde a da OneWire recebe a porta onde o sensor está ligado (3) e a da DallasTemperature recebe o objeto da OneWire:

```
//temperature sensor variables
OneWire oneWire(TEMPERATURE_PORT);
DallasTemperature sensors(&oneWire);
```

No *setup*, a OneWire é configurada para ler o sensor em modo parasita, escrevendo na porta o valor 0x44 e deixando energia ligada no cabo. Depois o DallasTemperature deve ser inicializado, através da função *begin*.

Depois é definida a função que será chamada no *loop* para ler o valor de temperatura, onde é utilizada função *requestTemperatures* da DallasTemperature para requisitar a leitura do sensor e, em seguida, no retorno, é utilizada a função *getTempCByIndex(0)* da mesma biblioteca, onde a temperatura do sensor é requisitada em graus Celsius; a biblioteca entende que podem haver diversos sensores de temperatura que se deseja ler, como só há um, ele será o primeiro, por isso o valor 0 é passado na função.

Em seguida é feita a medição do pH. Explicações mais detalhadas sobre o parâmetro medido e o funcionamento do sensor são encontrados na seção 4.1, mas explicando rapidamente, o sensor é composto por duas partes: o circuito e a sonda.

O circuito é responsável por captar o valor medido pela sonda e passar para o Arduino. A sonda é responsável por medir o pH, ou seja, é ela quem será mergulhada no líquido que se deseja medir o valor.

Ela é composta por dois eletrodos principais: o de vidro e o principal, ambos irão medir e o resultado será a diferença de voltagem entre eles. A sonda é um sensor passivo, assim, ela produz uma voltagem linearmente proporcional ao pH medido. O ideal é que num pH 7, seja produzido 0V, em valores menores que 7, a voltagem seja positiva e em valores maiores, negativa. A sonda tem alta impedância para evitar sobrecarga no circuito, assim as voltagens produzidas para fora dos 0V são na ordem de milivolts.

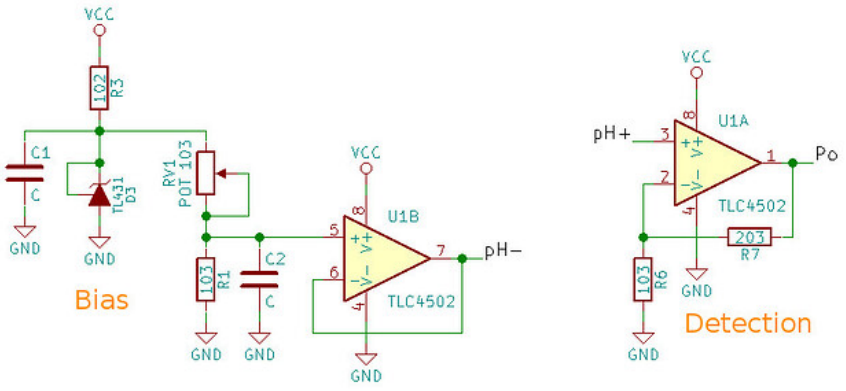
O sensor *pH4502c* tem pouca documentação *online*, assim, para entender o funcionamento dele em específico e ter *insights* de código, foi utilizado um artigo de duas partes no site *e-tinkers* chamado *Measure pH with a low-cost Arduino pH sensor board* (medindo pH com um sensor barato para Arduino).(CHEUNG, 2019)

Neste artigo, o autor diz ter encontrado um esquema do circuito num fórum. O circuito pode ser dividido em três: medição, detecção de limites e temperatura.

O circuito de detecção de limite serve para detectar quando o pH passa de um determinado limite para então acender um led vermelho, é melhor utilizar uma solução em *software* que em *hardware* para este fim, portanto, não será utilizado. A temperatura já é medida pelo DS18B20, assim, o circuito de temperatura também não será utilizado.

O circuito de medição utiliza o amplificador operacional de alta impedância TLC4502 para dividir voltagem e amplificar o ganho unitário (ou seja, não amplifica, tem ganho = 1). O Circuito pode ser visto na figura 19. Como o valor da medição varia entre negativo e positivo e o amplificador opera somente em valores positivos, metade ele é usado para dividir a voltagem, fornecendo uma referência de 2,5V (ao invés de 0V). A saída Po fornece a saída analógica da medição baseada na referência de 2,5V, assim, quando o pH for 0 o valor no Po é 0V e quando for 14, 5V. Um potenciômetro (RV1) é utilizado para calibração desta referência.

Figura 19: Circuito de medição do PH4502C



Fonte: FORUM ARDUINO (2017)

São necessários pelo menos dois pontos de calibração, uma vez que o pH é relativamente linear. A partir disso pode-se definir a inclinação da linha do pH. A inclinação da linha é determinada pela fórmula abaixo.

$m = \Delta pH/\Delta V_{pH} = (pH_{max} - pH_{min}) / (V_{pHmax} - V_{pHmin})$

onde pH é o valor de pH medido e V_{pH} é a voltagem para o valor medido. Mais detalhes sobre a calibração são encontrados na seção 4.1.

Com o valor da inclinação definido, a fórmula abaixo é aplicada para definir o valor do pH, dada a voltagem.

$pH = pH_{max} - (V_{max} - Po) * m$

onde pH_{max} é o ponto de pH mais alto da calibração, V_{pHmax} a sua voltagem, Po o valor retornado pelo sensor e *m* a inclinação da reta.

A medição do pH pode apresentar ruídos, assim, é necessário fazer várias medições seguidas e tirar a sua média. Depois de alguns testes, foi escolhido fazer 20 medições a cada 100 milissegundos para definir a voltagem lida. Depois disso, o valor do pH é retornado pela fórmula dada acima. Mais detalhes sobre isto podem ser encontrados na seção 4.1.

O próximo sensor que deve ser utilizado é o de eletrocondutividade. Para sua leitura utiliza-se a biblioteca GravityTDS.h, que requisita a leitura da voltagem do sensor e a temperatura. A biblioteca é inicializada no *setup*, através da função *begin.*

Em seguida, na função de leitura da eletrocondutividade, a porta analógica do sensor é lida e transformada em voltagem. A temperatura é recebida como parâmetro da função. Por último, é retornado o valor da eletrocondutividade através da função *getEcValue* da GravityTDS. O valor é retornado em mS/cm. Mais detalhes sobre a calibração do sensor e o seu funcionamento estão na seção 4.2.

Para medir a iluminação foi utilizado um fotoresistor que aumenta ou diminui sua resistência inversamente à quantidade de luz recebida. O elemento foi ligado ao Arduino com um resistor de 10KΩ.

Na função de leitura da iluminação o primeiro passo é a leitura de 20 amostras da porta analógica, para evitar ruídos, depois é tirada a média. O valor retornado fica entre 0 e 1024 e é mapeado para um valor entre 1 e 100, sendo 1 o valor mais escuro e 100 o mais claro possível.

Por último, é feita a checagem de nível e para isso são utilizados três sensores de nível d'água *Fd10*. Estes são sensores digitais, que indicam se ele está ou não em contato com água; Assim, o balde é dividido em 3 níveis: baixo, médio e alto. A função responsável por dizer lê cada sensor e checa o nível através de três condicionais:

- se todos estiverem em contato com a água, então o nível é cheio;
- se somente o sensor mais alto (l3) não estiver em contato, o nível é médio;
- se apenas o sensor mais baixo (l1) está em contato com a água, então o nível é baixo;

Se nenhuma das condições for satisfeita, então é retornado o valor -1, indicando que algo está errado ou o nível está abaixo do necessário.

3.4 ESP8266

O código referente a este capítulo está presente no apêndice 2.

Uma das principais preocupações ao programar para um microcontrolador é deixar o código leve e rápido. O funcionamento dos códigos geralmente são bem simples: existe uma parte responsável por fazer *setups,* e que roda uma única vez antes de tudo; e um *loop*, que irá rodar enquanto o microcontrolador estiver ligado, aqui serão feitas as lógicas que devem rodar a todo momento (leitura de sensores, publicação de dados e etc.).

Além destas duas partes, pode-se separar algumas lógicas em funções, a fim de organizar melhor o código e evitar repetições. O código fonte no microcontrolador ESP8266 se divide em quatro partes: Inclusões, Variáveis Globais, Funções e Principal.

Na primeira parte, inclusões, são adicionadas as bibliotecas utilizadas durante o código. Para o ESP são utilizadas 7 delas: Arduino.h, LittleFS.h, ESP8266WiFi.h, ArduinoJson.h, MQTT.h, string.h e SoftwareSerial.h.

Como os microcontroladores não foram programados pela IDE do Arduino, e sim pela IDE PlatformIO - que tem maior compatibilidade com diferentes placas, facilitando o trabalho de desenvolvimento na hora de gravar e *debugar* o código - inclui-se a biblioteca Arduino.h para usufruir de suas facilidades na hora de ler e escrever nas portas. Na IDE do Arduino a biblioteca é incluída automaticamente no código.

Neste código existem algumas credenciais que não devem estar escritas e nem disponíveis publicamente no repositório. Assim, é necessário separá-las num arquivo que o código irá ler. O tipo escolhido para o arquivo é o JSON por ser leve e simples de se trabalhar. Para gravar o arquivo, é necessário que haja um pequeno sistema de arquivos no ESP, e o responsável por sua criação é o código. Assim, é necessário utilizar a biblioteca LittleFS.h.

O ESP é o responsável por se conectar ao *broker*, e para isso é necessário conexão à internet. Nele vem um módulo WiFi embutido, e para utilizá-lo é necessária a biblioteca ESP8266WiFi.h.

Através da comunicação serial, ele recebe as informações dos sensores vinda do Arduino em formato JSON, além de ler as credenciais gravadas num arquivo JSON. Para isso, utiliza-se a biblioteca ArduinoJSON.h.

Após estabelecer conexão à internet, o ESP é responsável por se conectar ao *broker*, se inscrever nos tópicos de led e motor e publicar os dados. Isso é feito com a biblioteca MQTT.h.

Para lidar com cadeias de caracteres de forma mais simples, utiliza-se a biblioteca string.h. Para receber os dados do Arduino via comunicação serial, utiliza-se a SoftwareSerial.h.

Na parte de variáveis globais, são definidas as variáveis necessárias para o funcionamento da aplicação. São globais pois são inicializadas no *setup* e utilizadas no *loop*, assim, não é possível defini-las dentro de um só escopo.

As primeiras duas são *doc* e *obj*, que são um documento JSON estático e um objeto JSON, respectivamente. Elas serão utilizadas na leitura dos dados vindos via comunicação serial.

Depois são definidos vetores de caracteres para as credenciais do WiFi utilizado pelo ESP (*wifiSsid* e *wifiPassword,* que são o nome do WiFi para se conectar e sua senha) e as credenciais para se conectar no *broker* (*brokerUrl, brokerUsername* e *brokerPassword*, que são autoexplicativos).

Em seguida são declaradas as variáveis *net* e *client*, utilizadas para a conexão MQTT, sendo a *net* utilizada apenas na inicialização do *client*. O *client* é o objeto que representa o cliente MQTT responsável pelo *publish* e *subscribe*.

Por último, é declarado o SoftwareSerial *s*, com conexão nas portas D6 e D5 do ESP8266, sendo a primeira o Rx (receptora) e a segunda o Tx (transmissora); e o *unsigned long* (inteiro de 32 bits sem sinal) *lastMsg*, que marca os milissegundos desde a última mensagem recebida pelo Arduino.

Na parte de funções, são definidas as funções que serão utilizadas pelo setup e pelo loop, de forma a organizar melhor o código. São cinco: *loadConfig, setupWifi, reconnect, messageReceived* e *deserializeData.*

A função *loadConfig* faz uso da biblioteca LittleFS.h para abrir o arquivo de configurações *secrets.json.* Neste arquivo estão armazenadas as credenciais necessárias para as conexões que o ESP fará. Esta função lê seus valores e os atribui às variáveis globais correspondentes. Abaixo, um exemplo do arquivo de credenciais:

```
{
 "wifi_ssid": "Nome da rede",
 "wifi_password": "senha_da_rede",
 "broker_url":"link.para.o.broker",
 "broker_username": "nome_de_usuario_do_broker",
 "broker_password": "senha_do_broker"
}
```

Se houver erro na deserialização do arquivo ou na atribuição das variáveis, é retornado o valor booleano *false*, se tudo ocorrer corretamente, retorna *true*.

A *setupWifi* é responsável por fazer a conexão WiFi, utilizando as variáveis globais *wifiSsid* e *wifiPassword,* setadas na função citada anteriormente.

A função *reconnecct* é responsável por se reconectar ao *broker* se, em algum momento, a conexão for quebrada. Assim, enquanto o cliente não for conectado, ela

tenta conexão, utilizando as credenciais do *broker*. Se houver sucesso na conexão, inscreve o ESP nos tópicos desejados (*led* e *motor*).

messageReceived é um *callback* para quando chega alguma mensagem dos tópicos em que o ESP é inscrito. Por isso são passados os argumentos *topic* e *payload,* indicando o tópico de onde a mensagem vem e o conteúdo da mensagem. De acordo com estes parâmetros, uma decisão é tomada no código, como por exemplo, se o tópico for *led* e o conteúdo 1, a iluminação do primeiro andar será ligada; se for 0, desligada; se for 3, a do segundo andar será ligada e se for 2, desligada.

Por último, a *deserializeData* é responsável por deserializar o JSON recebido via comunicação serial do Arduino, publicando no tópico *data* os dados recebidos.

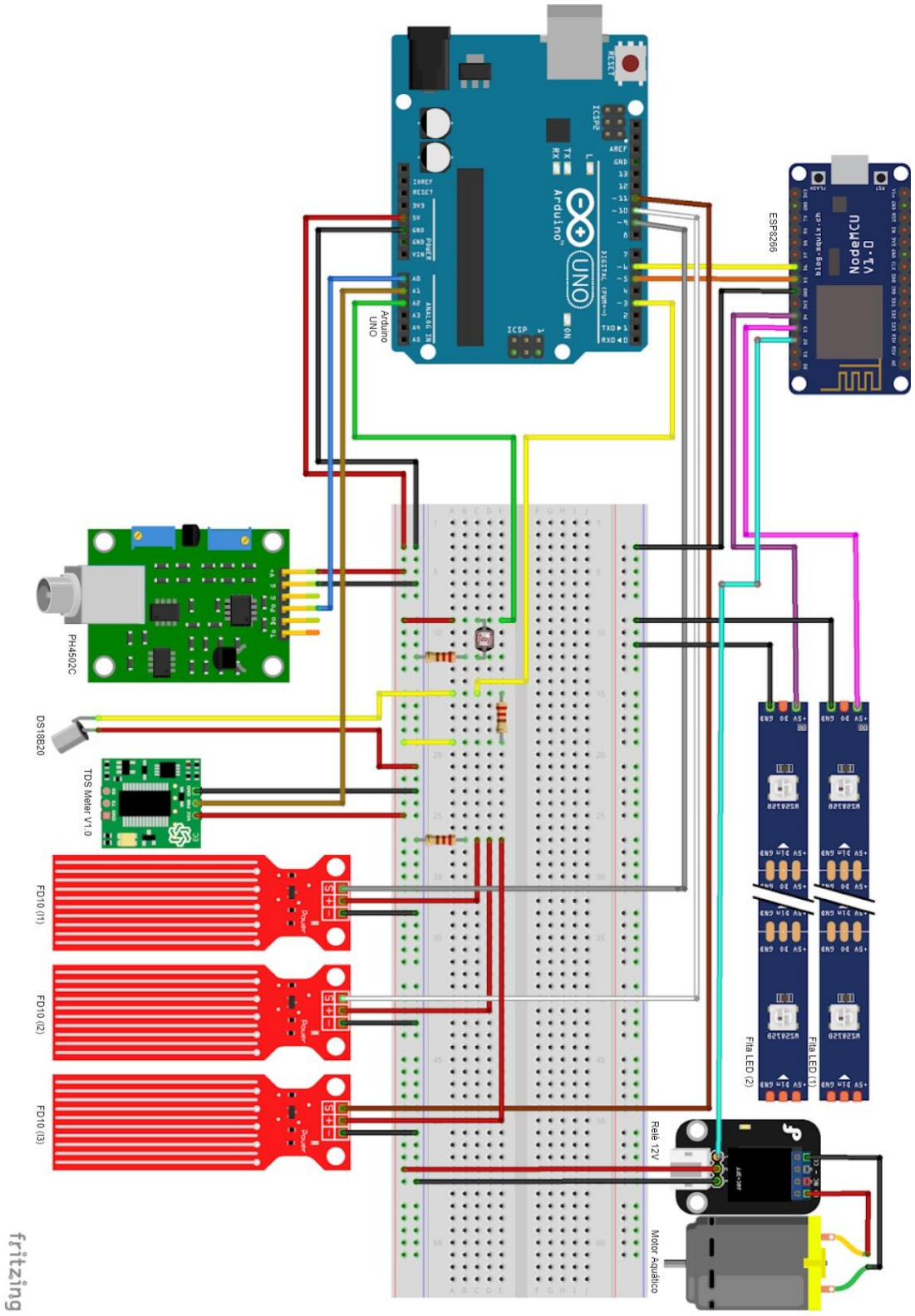
Depois de tudo isso, vem a parte principal, onde ficam as funções de *setup* e *loop*. A primeira inicializa a comunicação serial (variável *s)*, monta o sistema de arquivos (com o LittleFS.h), utliza a função *loadConfig* para carregar as informações necessárias para as próximas etapas do código. Em seguida, é utlizada a função *setupWifi*, para se conectar à rede. Depois são setados os pinos que o ESP utilizará, sendo dois para iluminação e um para o motor. Por último é configurado o *client* MQTT, passando seu endereço, sua porta e a função de *callback messageReceived* para quando alguma mensagem chegar.

No *loop* checa se o *client* permanece conectado ao broker, se não estiver, utiliza-se a função *reconnect*. Como o comando de conectar o *client* não foi dado no *setup*, na primeira vez que o *loop* roda ele está obrigatoriamente desligado, assim, utiliza-se a função *reconnect* para se conectar, evitando duplicação de código. Depois de conectado, o cliente é posto em *loop* para chegar mensagens vindas do *broker.*

Em seguida é atribuída uma variável que marca o milissegundo em que o código está rodando, chamada *now*. Ela servirá para marcar quantos segundos se passaram na execução, pois a cada 5 segundos, os dados vindos do Arduino serão deserializados, com a função *deserializeData*.

A ligação de todos os componentes pode ser vista na figura 20.

Figura 20: Ligação dos componentes



Fonte: Elaborado pelo Autor

4 CALIBRAGEM DOS SENSORES

Seguindo a definição de Ramos da Fonseca, um sensor é tudo aquilo que “percebe” uma grandeza física ou química e a indica de alguma forma, ou seja, é um dispositivo que tem algum parâmetro interno que varia em função de uma grandeza física: um termômetro de mercúrio por exemplo percebe a temperatura e a indica através da expansão do mercúrio. (RAMOS DA FONSECA, 2006)

Um dispositivo de sensoriamento pode operar como um transdutor, ou seja, é capaz de transformar um tipo de energia em outro, onde há uma grandeza de entrada - no caso do termômetro, a temperatura - e uma de saída - o nível indicado pelo mercúrio após sua expansão - relacionados de alguma forma.

Para integrar sistemas inteligentes, os sensores precisam transformar a grandeza de entrada em sinal elétrico que possa ser lido e interpretado. O sinal elétrico pode ser digital ou analógico.

Uma saída digital é binária, ou seja, seu valor pode ser 1 ou 0. Geralmente é utilizada para representar estados de sim ou não, aberto ou fechado, acima ou abaixo, ligado ou desligado, verdadeiro ou falso. Por exemplo, na porta das geladeiras há um sensor que identifica se a porta está aberta ou fechada para que a luz esteja ligada ou desligada.

Uma saída analógica é a representação de uma faixa de valores que uma grandeza pode assumir. Por exemplo, o pH pode ter valor entre 0 e 14, um sensor de pH assumirá uma determinada voltagem que indica o pH lido dentro desta faixa. Em casos de uma grandeza que não tem uma faixa específica, por exemplo, a temperatura que pode variar entre infinito positivo ou negativo, é escolhida uma faixa que o sensor é capaz de medir, por exemplo, um termômetro medirá a faixa de temperatura entre 5 e 55 graus. O resultado é uma voltagem entre 0 e a voltagem do circuito, num geral 5V. O dispositivo que lê irá transformar isto num valor entre 0 e 1024 normalmente, podendo o valor máximo ser maior para medições mais precisas.

Nos casos de sensores digitais, saber se o resultado corresponde ao valor medido é bastante simples, já que só existem duas possibilidades; porém, no caso dos analógicos, é necessário garantir de alguma forma que o valor medido corresponda à realidade.

Para isso é feita a calibração, ou seja, checar quais os valores retornados pelo sensor quando ele lê uma grandeza cujo valor já é sabido de antemão. O processo de calibração ajustará a relação do valor de saída ao valor de entrada.

Assim, é importante saber o funcionamento do sensor utilizado e da grandeza medida, para que a relação da saída possa ser devidamente feita.

4.1 pH

O pH é a escala de concentração em mols por litro de íons livres de H^+ e OH^-, responsáveis por tornar uma solução ácida ou básica. Esta escala varia entre 0 e 14, sendo 0 uma solução completamente ácida (1 mol/litro de H^+) e 14 uma solução completamente básica (1 mol/litro de OH^-). Uma solução neutra é representada na escala com o valor 7, onde as concentrações de íons livres são iguais (meio mol/litro de cada).

Para medir o pH existem três métodos principais: o visual, o fotométrico e o potenciométrico.

No visual o pH é indicado através da comparação de cores num papel sensível ao pH, como mostra a figura 21. Este método é bastante comum e barato, mas não é exatamente preciso, já que na escala de 0 a 14 os valores são tidos como inteiros.

No fotométrico é utilizado alguma solução capaz de mudar de cor ao ser misturada na solução a ser medida, como por exemplo o suco de repolho roxo, que é mostrado na figura 22.

Figura 21: papel sensível ao pH, método de medição visual

Fonte: INDIAMART.COM (2021)

Figura 22: suco de repolho roxo em diferentes soluções, método fotométrico

Fonte: TARNOWSKI (2017)

Por último, o potenciométrico é uma medição eletroquímica da força eletromotriz gerada por uma reação química. Mais especificamente é medido o potencial galvânico de um par de eletrodos.

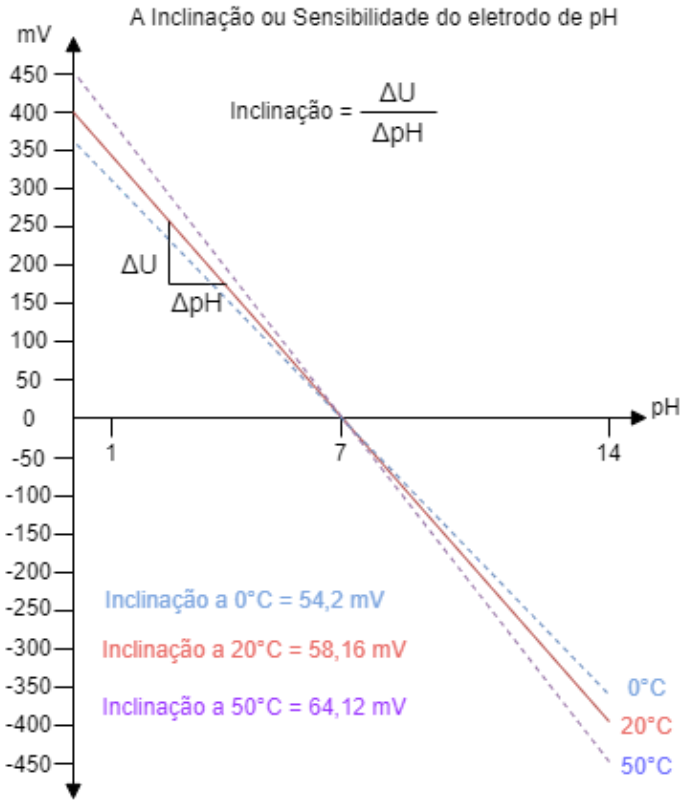
Este é o único que pode ser utilizado em processos de controle e seu funcionamento é baseado na equação de NERNST, descrita abaixo.

$$E = \frac{R.T}{n.F} . log \frac{C1}{C2}$$

Onde E é a diferença de potencial em mV; R é a constante universal dos gases perfeitos (8,31439 J/(mol . K)); F é a constante de Faraday (96495,7 C/mol); T é a temperatura em Kelvin; n a carga do íon medido (no caso de H, n = 1); C1 é a concentração de íons H ativos na solução C1 e C2 o mesmo, mas para a solução C2.

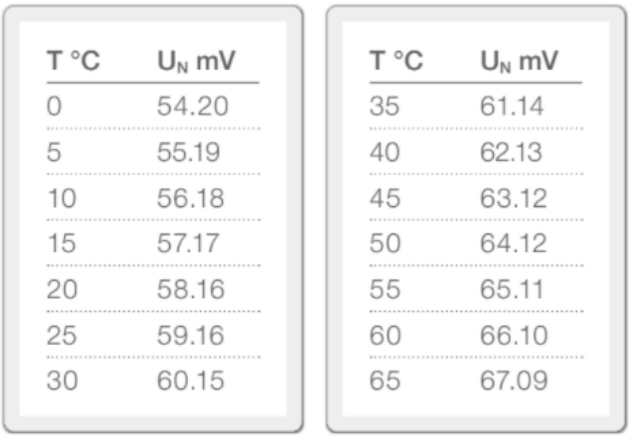
O pH varia de acordo com a temperatura, assim, a calibração deixa de ser tão simples. Não é possível isolar a variável que corresponde à concentração de íons H^+ na fórmula de NERNST, assim, o ideal é calibrar para um intervalo de temperaturas. A mudança de voltagem de acordo com a temperatura interfere somente na inclinação da reta de pH. Um exemplo disso pode ser visto na figura 23 e 24 abaixo.

Figura 23: Gráfico de medições de pH com variação de temperatura



Fonte: Reprodução de K. SPRINGER (p. 22, 2014)

Figura 24: Tabela de Potencial de NERNST de acordo com a temperatura para duas soluções de concentração 10:1



T °C	U_N mV
0	54.20
5	55.19
10	56.18
15	57.17
20	58.16
25	59.16
30	60.15

T °C	U_N mV
35	61.14
40	62.13
45	63.12
50	64.12
55	65.11
60	66.10
65	67.09

Fonte: K. SPRINGER (p. 11, 2014)

Assim, é possível escolher duas temperaturas (dois pontos) e medir o pH para descobrir a inclinação da reta em cada uma, depois, de acordo com a temperatura medida, descobrir a inclinação da reta do pH.

No sistema potenciométrico existem dois eletrodos: o de medição e o de referência. O primeiro é sensível à atividade de íons e o segundo fica imerso num eletrólito que irá gerar a diferença de potencial galvânica. Assim, a equação de NERNST demonstra a relação entre a solução medida e o eletrólito (C1 e C2).

O sistema de medição é composto por eletrodos e um voltímetro de alta impedância. O eletrodo de medição é normalmente composto por um fio de prata (Ag) revestido de cloreto de prata (AgCl), dentro de uma membrana de vidro com uma uma solução tampão neutra interna (normalmente cloreto de potássio [KCl]), que serve como eletrodo condutivo.

É importante que o vidro da membrana tenha as melhores condições para melhores medições. Ao entrar em contato com a solução a ser medida, é formada uma fina camada de gel entre o vidro e a solução, a grossura da camada depende da qualidade do vidro.

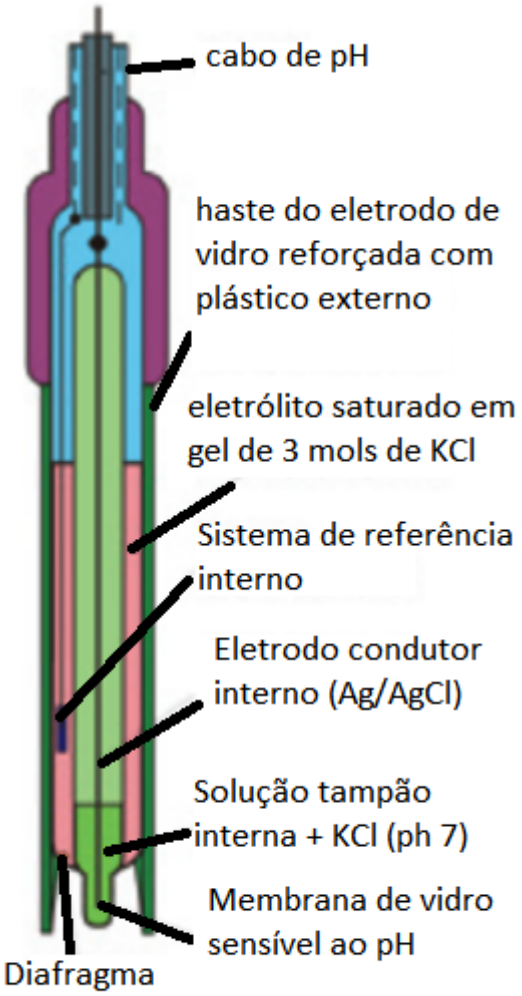
Dentro da membrana também é formada uma camada de gel, assim, ambas trocam íons H^+ de acordo com a concentração nelas. Se a concentração for igual, há uma diferença de potencial igual a 0 mV; se há diferença na concentração das soluções, então haverá diferença de potencial proporcional.

O eletrodo de referência é responsável pela precisão da medição: ele produz um potencial elétrico previsível e fica imerso num eletrólito definido. O fio que o compõe é geralmente de prata e cloreto de prata, imerso num eletrólito de 3 mols de cloreto de potássio.

O eletrólito deve estar em contato com o meio medido, então é usado um diafragma que o permita se difundir ou vazar no meio, geralmente feito de cerâmica porosa.

Para a confecção do dispositivo final, os eletrodos são combinados numa sonda, como mostrado na figura 25.

Figura 25: esquema de uma sonda de medição de pH



Fonte: K. SPRINGER (p. 16, 2014)

Em 1980 foi criado o eletrodo de polímero, e agora o diafragma de cerâmica é substituído por um pequeno buraco na superfície. Muito tempo imerso pode fazer com que o eletrólito se contamine com a solução medida, interferindo nas medições.

Quando próximas de pH superior a 10 ou inferior a 2, as medições deixam de ser tão precisas.

Para a calibração são necessárias soluções tampão: uma solução de um ácido fraco com sua base conjugada, ou uma base fraca com seu ácido conjugado, a fim de obter uma solução resistente às mudanças de pH quando ácidos ou bases são adicionados à ela. Nestas soluções a atividade de hidrogênio é estável em diversas quantidades de solubilidade.

A calibração deve ocorrer no multímetro, ajustando-o para compensar as imperfeições do eletrodo, pois estas não podem ser mudadas.

Num microprocessador é possível compensar os valores medidos através de *software*. Para calibrar são necessárias 3 soluções, uma com o menor valor desejado, outra com o maior valor desejado e a terceira com um valor entre estas. A maior e a menor irão definir a faixa de medição do sensor, a terceira serve para verificar se a calibragem está correta.

O processo consiste em medir a solução de menor valor, ajustar o multímetro para o valor desejado, medir a de maior e ajustar, verificar se a medição para o menor não foi mudada e ajustar e depois verificar se a solução de verificação condiz com o valor calibrado. Entre cada medição é importante limpar a sonda em água destilada e secá-la para não contaminar as soluções tampão.

Problemas na calibração podem acontecer por conta de contaminação das soluções, do eletrólito de referência, eletrodo não hidratado, diferença de temperatura entre o eletrodo e a solução, interferências na conexão e etc..

É importante armazenar a sonda numa solução de 3KCL para manter o eletrólito na mesma concentração.

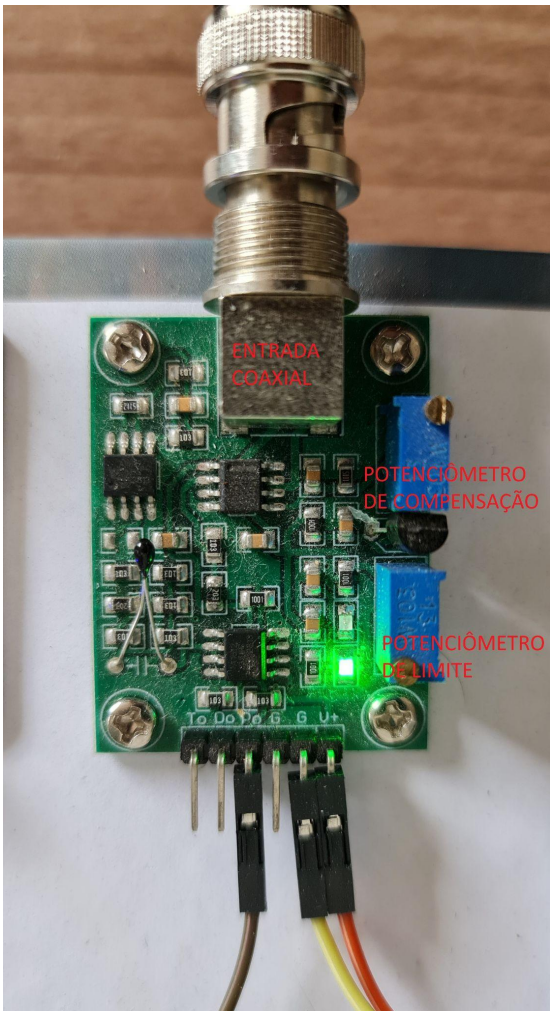
O sensor escolhido para o projeto foi o pH4502c. Ele é composto por uma placa onde está o circuito de medição e uma sonda com cabo coaxial. Na placa existem dois potenciômetros: o de limite e o de compensação. O primeiro é referente ao circuito de detecção de limite do pH, onde é determinado um valor de pH que será comparado e, se for maior, acenderá um LED; o segundo é referente à compensação na saída, assim, mexer neste potenciômetro aumenta ou diminui a voltagem de saída.

A placa exerce três funções, portanto tem três circuitos inclusos: temperatura, limite e pH. Para cada uma há um pino saída: To, Do, Po. O To é referente à saída do termistor da placa, como a temperatura é medida por outro sensor, este não será utilizado; Do é uma saída digital binária que indica se o pH está acima do limite escolhido no potenciômetro de limite; Po é a saída de pH. Para alimentar a placa existem 3 pinos de energia: V+ e dois G; V+ é a entrada de 5V e G o terra. Existem dois terras pois o primeiro (ao lado do V+) é referente ao circuito de medição de pH e o segundo aos circuitos de temperatura e limite. O esquema da placa pode ser visto na figura 26.

Segundo o artigo *Measure pH with a low-cost Arduino pH sensor board* encontrado no site e-tinkers e outras fontes, a primeira coisa a se fazer para a sua calibração é curto-circuitar a entrada coaxial e ajustar o potenciômetro de compensação. A entrada curto-circuitada pode ser vista na figura 27.

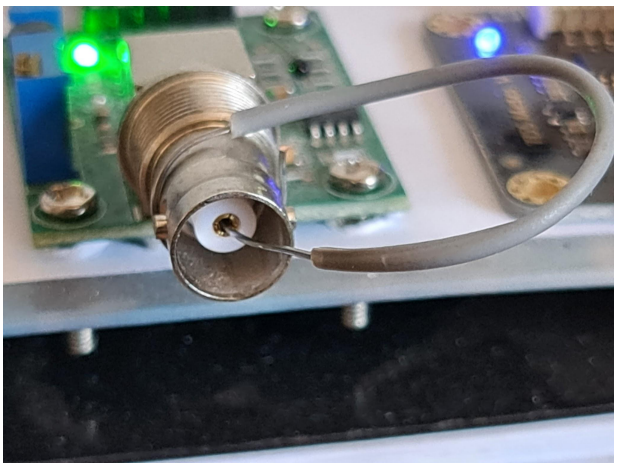
O curto circuito é entendido como um pH neutro, portanto o potenciômetro deve ser ajustado até que a saída Po tenha 2,5V.

Figura 26: Esquema da placa do sensor *ph4502c*



Fonte: Elaborado pelo Autor

Figura 27: Entrada coaxial curto-circuitada



Fonte: Elaborado pelo Autor

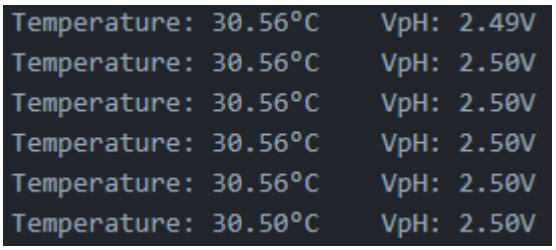
Para a calibração foi utilizado o código no apêndice 3. Nele é medida a temperatura com o DS18B20 e a voltagem de saída em Po. Os resultados do

primeiro passo (ajuste em curto-circuito) podem ser vistos abaixo nas figuras 28 e 29.

Figura 28: Resultados medidos em curto-circuito antes de ajuste no potenciômetro

Fonte: Elaborado pelo Autor

Figura 29: Resultados após ajuste no potenciômetro



Fonte: Elaborado pelo Autor

Depois disso são utilizadas as soluções tampão: foram utilizados três soluções de diferentes valores de pH: 4.01, 6.86 e 9.18 em 25ºC. As soluções vêm em pó e devem ser misturadas em 250ml de água deionizada até que o pó seja completamente dissolvido. As soluções podem ser vistas na figura 30.

Para cada solução, há uma tabela de seus valores em determinadas temperaturas no verso da embalagem, como mostra a figura 31. Esta tabela é utilizada na calibração para comparar a temperatura e o valor que deve ser lido.

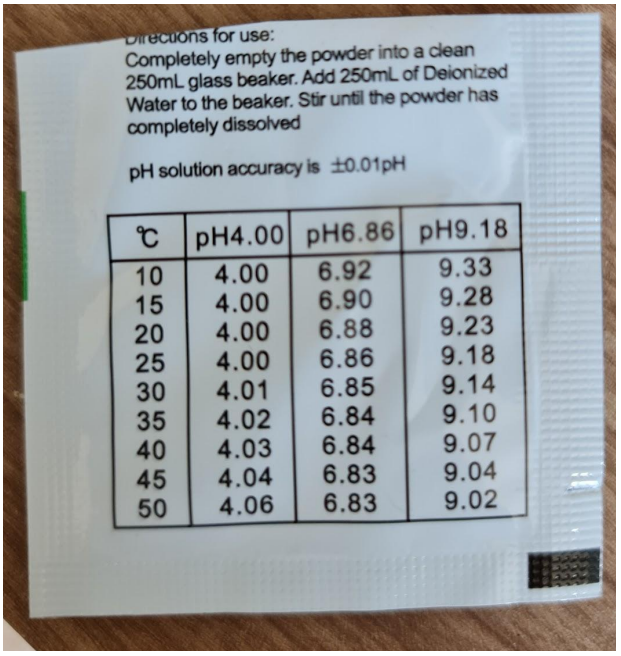
Sabe-se que a relação entre voltagem e pH é linear, onde pH mais baixos têm voltagens mais altas que pH mais altos, assim, a relação pode ser representada por uma reta decrescente.

Note que a temperatura é importante pois pode influenciar o pH e a voltagem da medição, como já citado na seção 3.3.

Figura 30: Soluções tampão prontas

Fonte: Elaborado pelo Autor

Figura 31: Tabela de variação de pH e temperatura


Directions for use:
Completely empty the powder into a clean 250mL glass beaker. Add 250mL of Deionized Water to the beaker. Stir until the powder has completely dissolved

pH solution accuracy is ±0.01pH

℃	pH4.00	pH6.86	pH9.18
10	4.00	6.92	9.33
15	4.00	6.90	9.28
20	4.00	6.88	9.23
25	4.00	6.86	9.18
30	4.01	6.85	9.14
35	4.02	6.84	9.10
40	4.03	6.84	9.07
45	4.04	6.83	9.04
50	4.06	6.83	9.02

Fonte: Elaborado pelo Autor

Sabe-se que a relação entre voltagem e pH é linear, onde pH mais baixos têm voltagens mais altas que pH mais altos, assim, a relação pode ser representada por uma reta decrescente, de forma que a calibração é feita para dois valores diferentes de pH, um menor e um maior, de onde será tirada a reta que relaciona a voltagem medida com o pH medido.

Mas como a temperatura afeta a inclinação desta reta, é necessário medir em temperaturas diferentes para tirar uma relação da inclinação da reta. Assim, as soluções foram primeiramente geladas a 10ºC e medidas e depois aquecidas pelo ambiente até uma temperatura de aproximadamente 25ºC. Em seguida foram aquecidas até aproximadamente 36ºC e resfriadas pelo ambiente até aproximadamente 30ºC.

Os resultados obtidos foram armazenados em arquivos de texto e depois convertidos em um arquivo csv (*comma separated values*, valores separados por vírgula) e depois trabalhados num *notebook* python para determinar a relação entre os valores lidos.

O código pode ser encontrado no apêndice 4. Nele são importadas as bibliotecas necessárias para ler o arquivo csv, construir os gráficos e fazer as operações matemáticas necessárias.

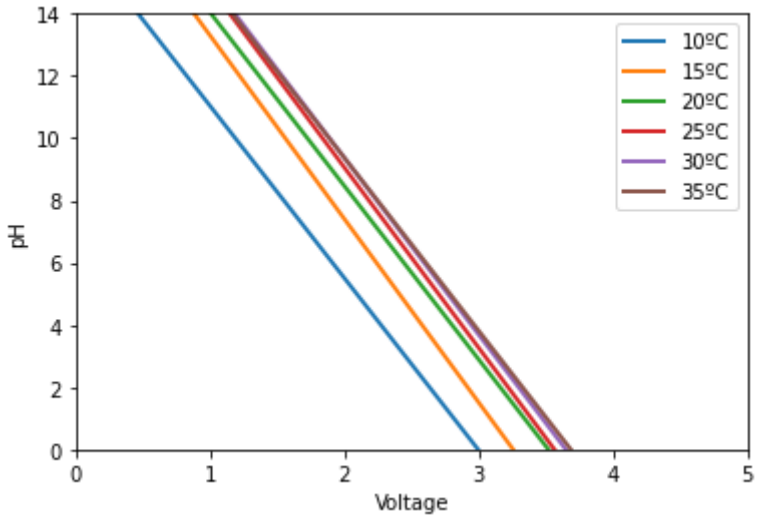
Em seguida, os arquivos são abertos e lidos. Os valores são categorizados por temperatura: foram escolhidas somente as temperaturas cujo pH está na tabela da figura 30, numa faixa de +-0.5º, então para 10ºC, são utilizadas as temperaturas entre 9,5ºC e 10,5ºC, e assim vai até os 35ºC. Temperaturas maiores que esta não foram registradas durante a medição.

Os valores da voltagem são armazenados numa matriz referente a cada solução em que o pH foi lido (ph 4 e pH 9). Em seguida, para cada linha da matriz é calculada a média dos valores, assim, ao final, cada linha da matriz vira um único valor representante da voltagem para a temperatura.

As médias são utilizadas junto com os valores de temperatura e valores de pH lido para calcular o coeficiente angular para cada temperatura. Cada coeficiente angular (m) é armazenado num vetor.

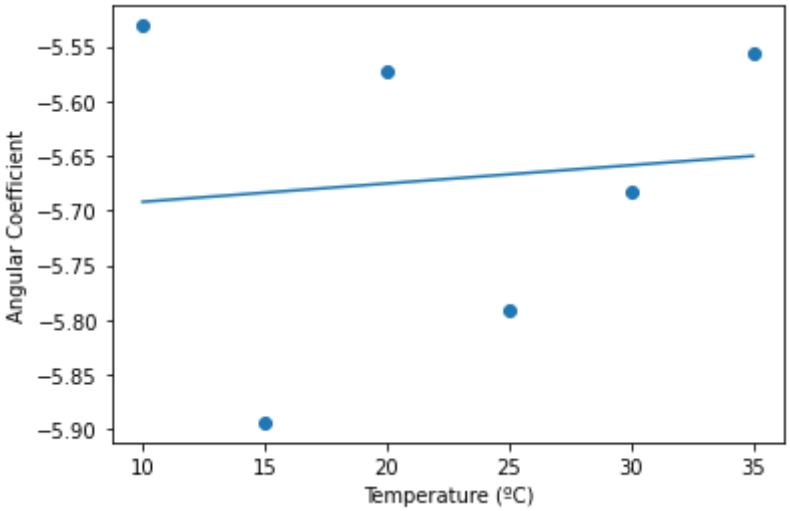
Ao final são *plotados* dois gráficos: a linha que relaciona o pH e a voltagem para cada temperatura, para comparar a inclinação de cada uma; e a regressão linear para o coeficiente angular de cada temperatura. Ambos são mostrados abaixo nas figuras 32 e 33.

Figura 32: Gráfico voltagem x pH para cada temperatura



Fonte: Elaborado pelo Autor

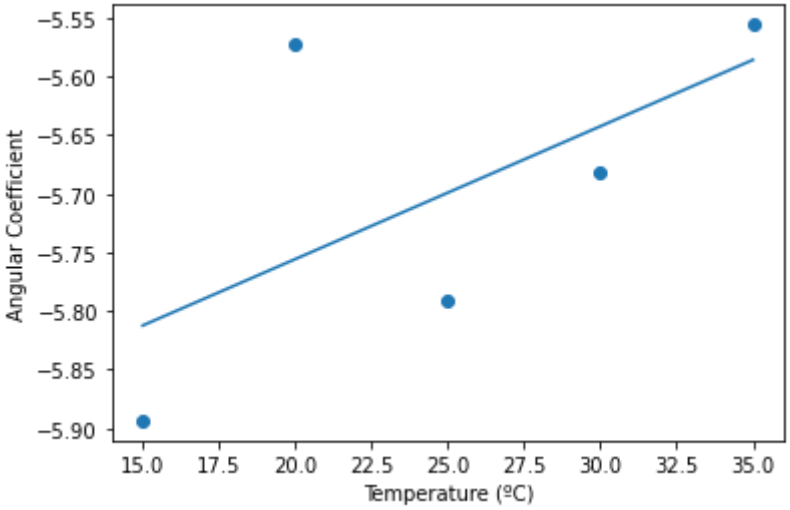
Figura 33: Gráfico pontos temperatura x coeficiente angular e regressão linear para pontos do gráfico



Fonte: Elaborado pelo Autor

A aproximação da reta para os coeficientes angulares é bastante insatisfatória, mas uma função que passe por todos os pontos é demasiado complexa para resolver o problema, assim, o ponto para a temperatura de 10ºC é excluído da aproximação, pois a água do reservatório não será refrigerada até tal temperatura, levando em conta a temperatura média do ano de 2021 para a cidade de Bauru, tornando a reta um pouco melhor mas ainda não perfeita, como pode ser vista na figura 34 abaixo.

Figura 34: Gráfico pontos temperatura x coeficiente angular e regressão linear para pontos do gráfico sem pontos dos 10ºC



Fonte: Elaborado pelo Autor

Sendo assim, o coeficiente angular da reta do pH será calculada pela função

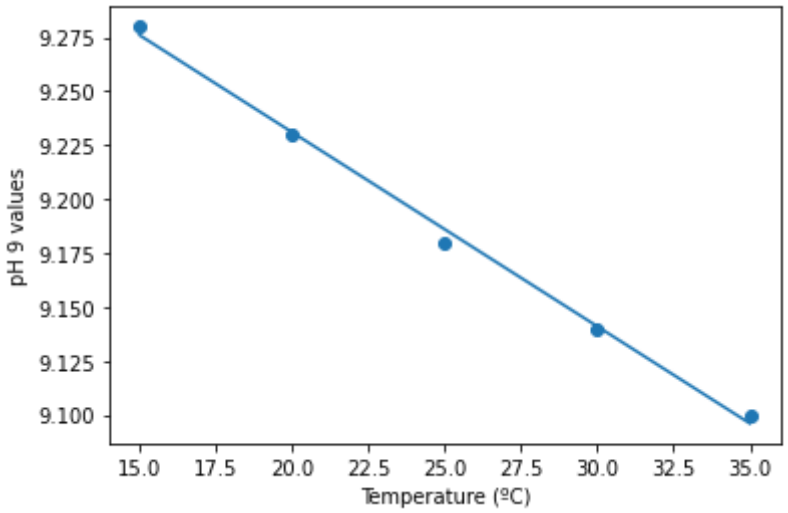
$$m = T * 0.01133237 - 5.98241695$$

em seguida, o pH_{max} será calculado pela função

$$pH_{max} = T * (-0.009) + 9.411$$

representada no gráfico da figura 35 abaixo. Para esta regressão linear foram utilizados os dados da coluna pH 9.18 da tabela na figura 35.

Figura 35: Regressão linear para valores de pH 9 e temperatura

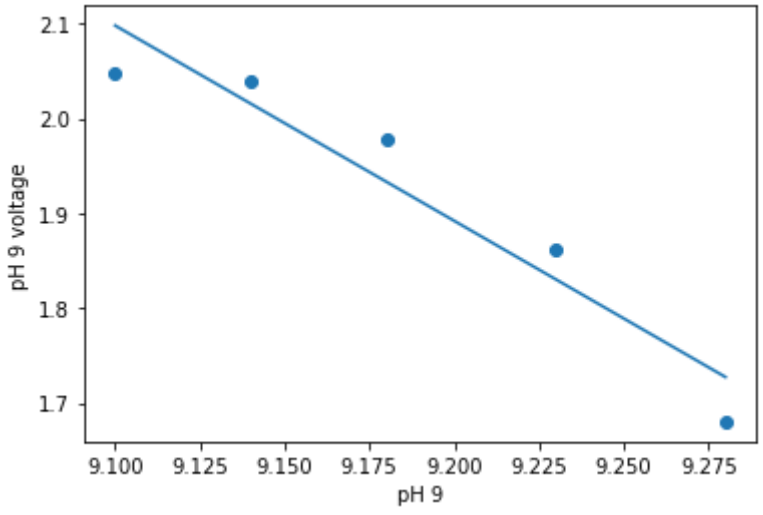


Fonte: Elaborado pelo Autor

Para calcular o V_{max} foi feito de cima, porém com os dados médios da voltagem lida para cada pH_{max}, como pode ser observado na figura 36. A função utilizada para seu cálculo é a

$$V_{max} = pH_{max} * (-2.05946685) + 20.83907748$$

Figura 36: Regressão linear para valores de voltagem lidos no pH 9 e temperatura



Fonte: Elaborado pelo Autor

por fim, o pH pode ser calculado pela fórmula abaixo.

$$pH = pH_{max} - (V_{max} - Po) * m$$

Incluídas no código fonte, é hora de checar a precisão da calibração. Para isso foram utilizadas as soluções tampão nas temperaturas tabeladas. As medições foram feitas para a solução de verificação de pH 6.86 nas temperaturas de 15, 20, 25, 30 e 35ºC. Os resultados podem ser vistos na tabela 1 abaixo, comparando o valor lido com o esperado.

Tabela 1: Medições para primeiros cálculos

Temperatura (ºC)	pH Esperado	média do pH Lido	diferença
15	6.90	6.20	0.7
20	6.88	6.11	0.77
25	6.86	5.98	0.88
30	6.85	6.33	0.52
35	6.84	6.67	0.17

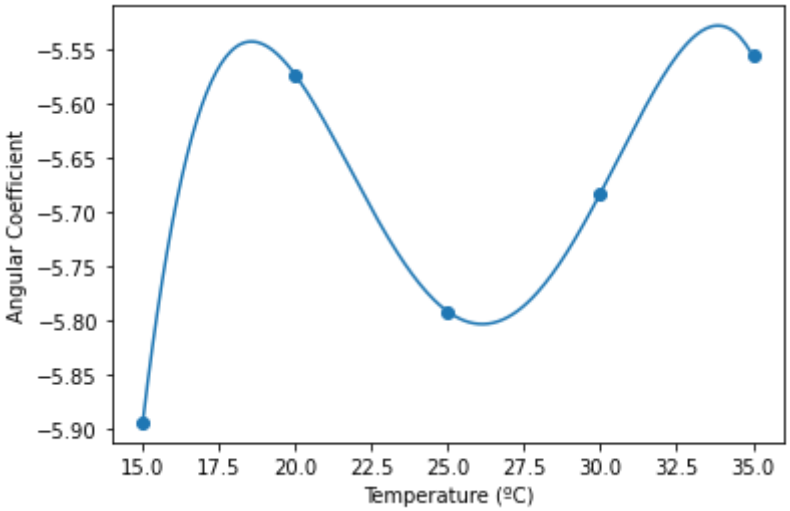
Fonte: Elaborado pelo Autor

Com grandes diferenças assim, é melhor ajustar a curva aos pontos de forma mais precisa: para as curvas de temperatura x coeficiente angular e pH_{max} x V_{pHmax} será utilizada a técnica dos mínimos quadrados para gerar uma função polinomial que satisfaça todos os pontos.

Foi utilizada a biblioteca *numpy* e a função *polyfit*, que executa o método dos mínimos quadrados, para retornar a função polinomial. Para a curva de temperatura x coeficiente angular foi necessário um polinômio de grau 4, a curva pode ser vista no gráfico na figura 37 e a função encontrada foi a seguinte:

$-7.847e\text{-}05\ T^4 + 0.008218\ T^3 - 0.3136\ T^2 + 5.154\ T - 36.41$

Figura 37: Ajuste de curva aos pontos temperatura x coeficiente através da técnica dos mínimos quadrados

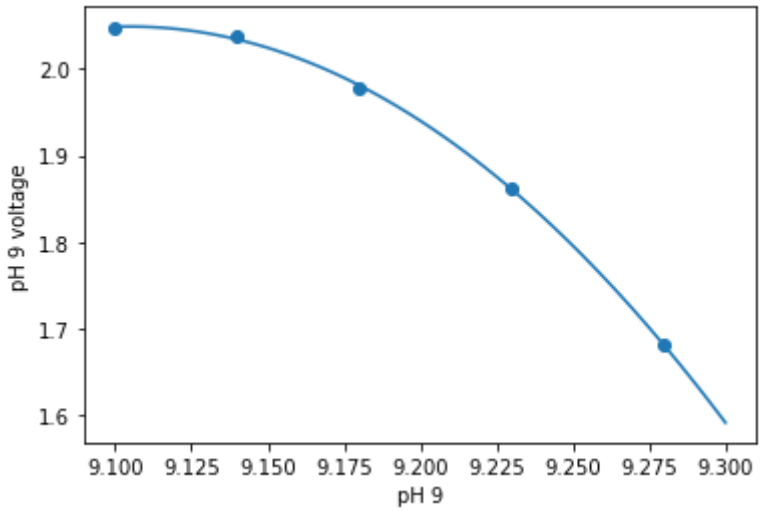


Fonte: Elaborado pelo Autor

Para a curva do pH_{max} x V_{pHmax} foi necessária uma função quadrada, descrita abaixo e com gráfico representado na figura 38.

$$-12.02\ pH_{max}^2 + 218.8\ pH_{max} - 994.2$$

Figura 38: Ajuste de curva aos pontos pH_{max} x V_{pHmax} através da técnica dos mínimos quadrados



Fonte: Elaborado pelo Autor

Depois disso, o programa foi atualizado para fazer os cálculos. A nova fórmula foi testada na solução de pH 6.88 e os resultados são vistos na tabela 2.

Tabela 2: Medições para os novos cálculos

Temperatura (ºC)	pH Esperado	média do pH Lido	diferença
15	6.90	6.13	0.24
20	6.88	6.68	0.2
25	6.86	6.86	0.0
30	6.85	6.97	-0.12
35	6.84	6.96	-0.12

Fonte: Elaborado pelo Autor

Com resultados mais uma vez insatisfatórios, foi escolhido calibrar para uma faixa de pH menor, uma vez que a água do reservatório deve estar sempre entre 5 e 7.5, podendo estar numa faixa mais estreita de 5.8 e 6.5, como já citado na seção 2.2.

Além disso, revendo diversos materiais sobre a calibração, a temperatura não é levada em conta, apesar de causar diferenças, a aplicação não exige tamanha precisão. Sendo assim, as soluções são medidas na temperatura de 25ºC.

No são coletadas 20 amostras de pH a cada 100 milisegundos, depois a média de todas é tirada, para evitar ruídos do sensor. Assim, a média de voltagem para os novos pH limite são as vistas na tabela 3.

Tabela 3: Valores de pH e sua voltagem para nova calibração.

pH	4.01	6.86
Voltagem	2.91	2.40

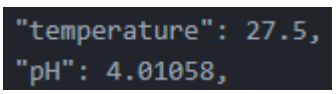
Fonte: Elaborado pelo Autor

Sendo assim, o pH_{max} agora é 6.88; a V_{pHmax} é 2.40 e o coeficiente angular da reta é (4 - 6.880) / (2.91 - 2.4) = -5.6078431373. Sendo assim, a nova função de cálculo do pH é:

$$pH = 6.86 - (2.40 - Po) * (-5.6078431373)$$

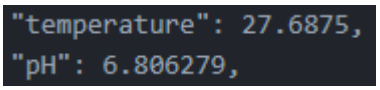
Os resultados obtidos após são os vistos nas figuras 39 e 40 abaixo.

Figura 39: Medição na solução de pH 4.00



Fonte: Elaborado pelo Autor

Figura 40: Medição na solução de pH 6.88



Fonte: Elaborado pelo Autor

A precisão necessária para o trabalho é de 0.1, assim, a diferença vista entre pH medido e esperado é desprezível, uma vez que as diferenças caem na precisão de 0.01.

Com o tempo, o sensor deve perder sua precisão e ficar novamente descalibrado, assim, é necessário refazer a calibração de tempos em tempos, especialmente por que a sonda ficará em contato contínuo com a solução, é necessário tomar cuidados como devolvê-la à solução de 3 KCl periodicamente, ou ela será irreversivelmente descalibrada.

4.2 Eletrocondutividade

Quando numa solução, metal ou gás é medida a capacidade de transmissão elétrica, estamos falando da eletrocondutividade. Numa solução, a corrente é fornecida por cátions e ânions e num metal por elétrons.

Se um material ou substância tem alta condutividade, sua resistência é baixa; se tiver baixa condutividade, sua resistência é alta. Assim, a condutividade é definida como o inverso da resistência, sendo medida em Siemens.

Para medir a condutividade elétrica é necessário aplicar uma determinada voltagem no material a ser medido e mensurar a eletrocondutividade de alguma forma. Isto é feito através de dois eletrodos: um cátodo (negativo) e um ânodo (positivo). Os cátions são atraídos pelo cátodo e os ânions pelo ânodo. Um par de eletrodos tem uma distância L entre os eletrodos e uma área A de contato com o condutor. A partir disso, é construída uma relação entre eletrocondutividade (G) e parâmetros do eletrodo e condutor, vista abaixo:

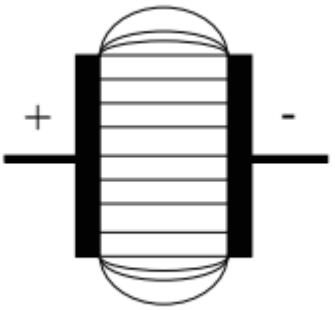
$$G = \gamma \cdot \frac{A}{L} = \frac{1}{\rho} \cdot \frac{A}{L}$$

onde *G* é a eletrocondutividade em Siemens (S), *A* a área de contato do eletrodo com o condutor, *L* a distância entre os eletrodos, γ a condutividade específica do condutor em S/m e ρ a resistência específica do condutor em Ωm.

Geralmente os eletrodos são feitos de grafite ou aço inoxidável, materiais adequados para a polarização necessária à medição.

A razão entre a área e a distância dos eletrodos é chamada constante do eletrodo. Ela não é descoberta simplesmente medindo a área e a distância entre os eletrodos pois a área é maior que o esperado normalmente: o fluxo de energia entre as pontas se alonga nas bordas, fazendo com que a área geométrica a ser medida seja irregular e mutante. Um exemplo é mostrado na figura 41, as linhas do fluxo entre os eletrodos são retas no meio, mas nas bordas formam arcos.

Figura 41: Linhas do campo de energia entre os eletrodos



Fonte: XYLEM ANALYTICS GERMANY SALES GMBH & CO. KG.(2019)

Para determinar a constante do sensor é necessário usar uma solução de calibração com uma condutividade conhecida.

A eletrocondutividade é um parâmetro bastante dependente da temperatura. No livro *Conductivity Handbook* da *Xylem* esta dependência é ilustrada com o exemplo de uma solução de 0.01 mol de cloreto de potássio que a 20ºC apresenta condutividade de 1278 µS/cm e a 25ºC, 1413 µS/cm. (XYLEM ANALYTICS GERMANY SALES GMBH & CO. KG., 2019)

Para lidar com isso é necessário compensar a temperatura: pode-se utilizar uma função linear, não-linear ou nenhuma compensação. A variação é diferente para cada líquido, sendo uma compensação mais adequada que outra para cada caso, por exemplo, uma solução salina apresenta variação mais linear que soluções aquosas.

Normalmente a constante do eletrodo já vem determinada pelo fabricante, assim, o eletrodo está calibrado com uma tolerância de aproximadamente 2%. Para maiores precisões, o eletrodo pode ser calibrado em soluções de condutividade conhecida, como uma solução de KCl. É importante durante o processo de calibração tomar o cuidado de não contaminar a solução, o que irá alterar sua condutividade e torna o processo inconfiável.

A calibração deve ser feita para checar e não ajustar: a constante do sensor é um parâmetro mecânico e só irá mudar em caso de contaminação do eletrodo, assim, é recomendado limpá-lo até voltar aos parâmetros corretos e não ajustá-lo ao erro.

As soluções de calibração são bastante específicas, podendo ter suas características alteradas por evaporação, diluição ou contaminação. A solução usada para uma calibração deve ser descartada, o que não for usado deve ser guardado em recipiente tampado e não deve ser usado após a data de validade.

É importante tomar o cuidado de limpar o sensor periodicamente para evitar contaminação iônica. Para isso pode-se usar água quente com detergente ou álcool e uma escova ou pincel para esfregar.

Uma vez conhecida a constante do eletrodo, basta usá-lo para medir a resistência da solução. A partir daí, é utilizada a fórmula para determinar a eletrocondutividade.

A partir dela é possível determinar outros parâmetros da solução medida, tais como a salinidade, total de sólidos dissolvidos (TDS) e resistência específica.

A salinidade é um parâmetro que identifica quantos gramas de sal está dissolvido numa determinada massa de água (g/kg). Para chegar nele é comparada a concentração de oxigênio e eletrocondutividade em tabelas, como a *International Oceonographic Table.*

O TDS (*Total Dissolved Solids)* é a quantidade de sólidos dissolvidos numa solução. O método verdadeiro de medir isto é evaporando a água e pesando os sólidos restantes, porém, é possível estimar quantas partículas por milhão (ppm) está dissolvido numa solução se soubermos a eletrocondutividade: hidrogênio e oxigênio (H_2O) praticamente não carregam eletricidade (a eletrocondutividade da água puríssima é muito próxima de zero), mas metais, minerais e sais sim. Sendo assim, é possível converter a eletrocondutividade para TDS. Apesar de não ser uma conversão exata, é uma boa estimativa.

Para o trabalho em questão, foi decidido não calibrar o sensor e nem checar sua constante, uma vez que a fabricante fornece uma biblioteca específica para seu uso e o custo da solução de calibração é um pouco alto. Nela estão presentes funções de cálculo de eletrocondutividade já com compensação de temperatura.

5 API

Uma Interface de Programação de Aplicação (do inglês Application Programming Interface) trata-se de “um conjunto de definições e protocolos usados no desenvolvimento e na integração de *softwares*” segundo o *site* RedHat. (REDHAT, 2017)

Ou seja, permite que uma solução utilize serviços externos sem a necessidade de saber como estes foram implementados, diminuindo o tempo e simplificando o desenvolvimento.

Existem diversas APIs, gratuitas ou pagas, que resolvem problemas comuns ou específicos, assim, muitas vezes uma solução faz uso delas para evitar gastar tempo e esforço programando algo que já está pronto e consolidado.

Uma API recebe uma solicitação, esta solicitação deve ser documentada pelo criador da API para que os usuários possam estruturá-la corretamente. A forma como a solicitação é estruturada determina a forma que a API responde à solicitação.

Elas podem ser publicadas em três formas: privada, de parceiros e pública. Na primeira ela é feita para uso interno, geralmente de empresas; na segunda ela é compartilhada com parceiros de negócios e na última ela é disponibilizada para todos.

As APIs geralmente são remotas, ou seja, seus recursos estão fora do computador de quem a solicita, e geralmente estão disponíveis através da *internet,* através do protocolo HTTP.

O HTTP (protocolo de transferência de hipertexto, ou *Hypertext Transfer Protocol*) é um protocolo para a troca de dados entre um cliente e um servidor. Para obter, enviar, deletar ou atualizar um dado são utilizados verbos nas requisições: GET, POST, DELETE e PUT. Para cada retorno de requisição há um código associado, podendo ele ser OK (código 200), *Not Found* (código 404), *Bad Request* (código 400) e etc..

Uma requisição HTTP é uma URL, onde há um domínio, um *endpoint* (caminho para o recurso), uma *query* para passar parâmetros para o recurso. Numa requisição, além da URL é possível passar um corpo, incluindo mais dados e outros parâmetros.

No projeto da Horta, a API entra para fazer a persistência de dados coletados pelos sensores da horta: os dados são enviados para o *broker*, a API se inscreve nos tópicos relevantes e, toda vez que um dado chega ele é armazenado num banco de dados.

Os dados armazenados devem ser disponibilizados para o cliente, assim, ele irá requisitar à API os dados coletados através de métodos HTTP. Um cliente é qualquer programa que consuma a API, podendo ser ele uma aplicação de análise de dados, uma aplicação *frontend* e etc..

A API é deve ser um componente opcional para a estrutura geral, podendo ou não ser utilizada, por isso o sistema de telemetria se comunica com um *broker* MQTT, desacoplando o uso da API e permitindo usar clientes MQTT diretos (existem diversos *softwares* que se comunicam diretamente com um *broker*). A intenção é que a estrutura de visualização de dados e persistência possa ser escolhida pelo usuário.

5.1 O Banco de dados

O *script* SQL utilizado para a criação das tabelas pode ser encontrado no apêndice 5.

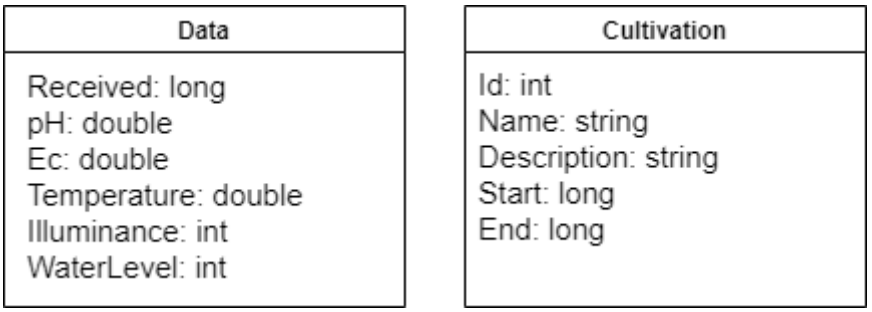
Para a persistência dos dados coletados pelos sensores da Horta, foi escolhido o gerenciador de banco de dados PostgreSQL, uma solução gratuita de código aberto para tabelas relacionais.

Os dados são armazenados em duas tabelas, sem relação no banco, mas relacionadas por código: *Data* (dados) e *Cultivation* (cultura).

A tabela de dados armazena os dados que chegam no *broker*, os dados lidos pelos sensores. Para cada parâmetro lido há uma coluna: pH, eletrocondutividade (ec), temperatura (*temperature*), iluminação (*illuminance*) e nível d'água (*water level*). É anotado o momento em que um dado chega do *broker* e gravado na coluna *received* (recebido) no formato de data Unix, por ser mais fácil de lidar entre o banco e o código da API (trata-se de um inteiro longo) e ser mais leve para armazenar que uma string que representa a data no formato ISO 8601, por exemplo.

A tabela de culturas tem como objetivo armazenar um período de tempo (colunas *start* e *end*) e lhe dar um nome e uma descrição (*name* e *description*), por exemplo, se o usuário plantou alfaces, armazenar período em que eles cresceram facilita o acesso aos dados específicos desta cultura. Um esquema das tabelas pode ser visto na figura 42.

Figura 42: Esquema das tabelas do banco de dados



Fonte: Elaborado pelo Autor

5.2 Desenvolvimento em ASP.NET

Os códigos referentes a esta seção podem ser encontrados no repositório remoto “HortaAPI”[1] no site *GitHub*.

A API é programada na linguagem C# com o *framework* ASP.NET pela facilidade em se desenvolver uma aplicação com o *framework*, bom desempenho da linguagem e afinidade do autor.

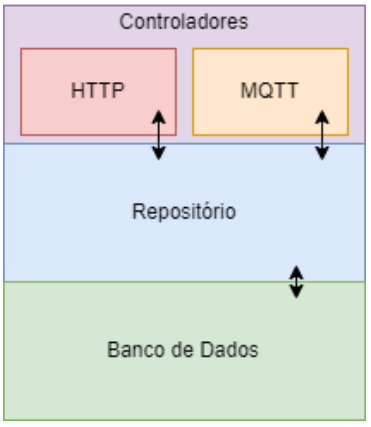
A versão utilizada foi a 5, pois era a última lançada no momento do desenvolvimento da aplicação (alguns meses depois foi lançada a 6, mas escolhi não atualizar ainda).

O objetivo da aplicação é se conectar ao banco de dados para gravar e ler; se conectar ao *broker* para receber os dados a serem gravados e expor os dados do banco de dados em *endpoints* acessíveis por requisições HTTP.

Para isso foi escolhida a arquitetura de repositório, onde as tabelas e as ações no banco de dados são abstraídas em código, criando uma camada de abstração que lida com o banco de dados, assim, a camada de *endpoints* não precisa lidar diretamente com ele.

Sendo assim, a aplicação pode ser dividida em três camadas: banco de dados, repositório e controladores. O banco armazena os dados em suas tabelas; o repositório é responsável por se conectar ao banco e performar as operações de CRUD (*Create, Read, Update and Delete* ou Criação, Leitura, Atualização e Remoção) nos dados; os controladores são divididos em dois tipos: HTTP e MQTT; o primeiro é responsável por expor os dados persistidos no banco de dados através de *endpoints* acessíveis via HTTP, utilizando o repositório para se comunicar com o banco; o segundo é responsável por receber os dados do *broker*, utilizar o repositório para persisti-los no banco e expor *endpoints* que irão publicar no *broker* para controle dos atuadores (como a iluminação e o motor). A arquitetura da aplicação pode ser vista na figura 43.

Figura 43: arquitetura da API.



Fonte: Elaborado pelo Autor

[1] Link para o repositório do projeto: https://github.com/matheoxz/HortaAPI

5.2.1 Repositório

Os arquivos referentes à camada do repositório são os presentes nas pastas *Model* e *Repository*.

Na pasta *Model* existem duas classes: *CultivationModel* e *DataModel*. Estas classes são modelos de dados, ou seja, não devem ter métodos, apenas propriedades. Cada uma é referente à tabela de mesmo nome no banco de dados, representando uma abstração destas em código C#.

Na pasta Repository existem três interfaces e três classes: cada classe implementa uma interface. Uma interface é um contrato para uma classe: nela são definidas as propriedades e métodos que uma classe deve implementar, sendo assim, uma interface não tem lógica dentro de si, nenhum método é implementado, apenas “assinado”.

As interfaces não são obrigatórias num geral, mas para facilitar a manutenção do código no futuro e estar na linha do bom desenvolvimento, foi escolhido utilizar a injeção (ou inversão) de dependências, sendo assim, a implementação de interfaces para estas classes é obrigatória.

A injeção de dependências trata-se de separar a construção do uso: é aplicada a inversão de controle ao gerenciamento de dependências. Veja, uma classe depende de outra para suas implementações, num cenário comum a responsável pela instanciação da classe a que se depende é da própria classe dependente; na inversão de controle, a instância da classe a que se depende é passada para um objeto dedicado a instanciar classes.

O objeto responsável pela injeção de dependência no ASP.NET faz uso das interfaces para identificar uma classe, assim, dada uma classe que depende de outras, todas as suas dependências são pedidas no construtor e, quando o *container* de serviços a instanciar, irá também fornecer uma instância das classe pedidas.

Para que o *container* de serviços identifique uma interface e uma classe que a implemente, é necessário registrá-las: no arquivo *Startup* na função *ConfigureServices* as interfaces são registradas com as classes que as implementam através do *AddSingleton*, que significa que uma classe será instanciada uma única vez, assim, quando outra classe pedir uma instância dela, será passada a única já feita.

As três classes para o repositório são *DbContext*, *DataRepository* e *CultivationRepository*.

A primeira é bem simples e tem como única função retornar uma conexão com o banco de dados. Para isto, é passada uma *string* de conexão, que informa a localização e o nome do banco, além de outros parâmetros para a conexão. Esta *string* é considerada sensível, sendo assim, deve ser armazenada num local seguro.

O *Visual Studio* (ambiente de desenvolvimento utilizado) oferece um gerenciador de segredos, que é um arquivo criptografado e secreto onde tais dados podem ser armazenados. Sendo assim, a *string* de conexão é armazenada lá. Para

acessá-la, é necessário pedi-la pelo construtor e na classe de *Startup* passá-la no registro da injeção de dependência. Para isso, é usado o objeto *Configuration*, que acessa o arquivo e retorna o valor pedido.

Uma abordagem comum é a de passar direto à classe o objeto *Configuration* via injeção de dependência (pedir no construtor um *IConfiguration*), mas isto dá à classe mais informações que o necessário.

A classe *DataRepository* é responsável por lidar com a tabela de dados no banco de dados. Para isto, são definidos quatro métodos: *Add*, *GetAll*, *GetRange* e *GetByCultivation*. O primeiro recebe um *DataModel* para gravar no banco; o segundo retorna tudo o que já foi gravado na tabela em ordem cronológica; o terceiro recebe uma data de início e uma de fim para retornar os dados referentes a esta faixa de tempo; o último recebe o nome de uma cultura (*cultivation*) e retorna os dados referentes ao tempo de início e fim dela.

A classe *CultivationRepository* é responsável por lidar com a tabela de culturas no banco de dados. Para isto, são definidos cinco métodos: *Add, GetAll, GetByName, Update* e *Delete*. A primeira é responsável por adicionar um *CultivationModel* ao banco de dados; a segunda por retornar todas as culturas gravadas no banco; a terceira por retornar uma única cultura, dado seu nome; a quarta atualiza os dados de uma cultura e a última deleta.

Para a realização das funções, em ambas é necessária a conexão com o banco de dados, assim, ambas pedem pelo construtor um *IDbContext.* Dentro de cada método a conexão é aberta, as operações SQL necessárias são feitas e então ela é fechada. É importante garantir que em caso de qualquer exceção a conexão seja fechada, pois uma conexão aberta não permite que outras sejam abertas, bloqueando o programa.

5.2.2 Controlador MQTT

O controlador MQTT tem como função fazer as comunicações que utilizam o protocolo MQTT, sendo assim, ele se inscreve em tópicos e recebe suas mensagens, assim como envia mensagens a tópicos.

O controlador pede em seu construtor uma instância do objeto *MqttClient* da biblioteca *M2Mqtt,* um *IDataRepository* e um *IConfiguration*. Para ser instanciado na injeção de dependência, o primeiro deve ser registrado na classe *Startup* pois a biblioteca não tem suporte para isto; mesmo não tendo uma interface, é possível registrá-lo; na sua instanciação é pedida a url e a porta do *broker*, que são informações sensíveis, sendo assim, são armazenadas no gerenciador de segredos do *Visual Studio*. O segundo é utilizado para armazenar os dados recebidos no banco de dados e o último é para estabelecer a conexão com o *broker* passando os dados de *login* (nome de usuário e senha).

Dentro do construtor a conexão com o broker é estabelecida, as inscrições nos tópicos são feitas e é associada uma função de *callback* para quando uma mensagem é recebida.

Esta função de *callback* trata-se da *messageReceived* que é privada. Nela é checado o tópico de onde a mensagem veio e de acordo com a origem, há um destino para a mensagem. No caso só há o tópico *data* que retorna os dados do ESP8266 em forma de JSON. Estes dados são então *deserializados* para um *DataModel* e em seguida é utilizado o repositório para gravá-los.

Depois disso são definidos *endpoints* HTTP para controle dos atuadores via MQTT: o cliente faz uma requisição HTTP para a API e ela faz uma publicação MQTT no *broker*. Os dois *endpoints* são para o motor e para o led. No primeiro é passado o estado em *string* que se deseja pôr o motor (*on* ou *off*), e assim, será feita uma publicação no tópico MQTT referente ao motor de acordo com o estado desejado. No segundo é passado o andar que se deseja ligar o led (1 ou 2) e o estado (*on* ou *off*), em seguida a publicação será feita no tópico referente.

Por último há o *endpoint* necessário para iniciar a classe: o *start*. Este *endpoint* apenas retorna um OK (código 200) com a mensagem de que o serviço foi inicializado. Ele é necessário pois um controlador só é instanciado pelo gerenciador de dependências quando requisitado, assim, é necessário inicializá-lo para que comece a ouvir as mensagens publicadas no broker.

5.2.3 Controladores HTTP

Os controladores HTTP são a interface com a qual um cliente irá interagir para requisitar os dados desejados. Assim, eles nada mais são do que uma abstração dos repositórios.

Existem duas classes referentes à eles na pasta *Controllers*: *DataController* e *CultivationController*. Cada um é referente a um repositório, que é referente a uma tabela do banco de dados.

Sendo assim, cada *endpoint* presente neles é referente a uma função presente em seu repositório, retornando os dados pedidos com um OK (código 200) ou o código HTTP adequado em caso de erro.

5.3 Os *endpoints*

Este capítulo tem o propósito de servir como documentação da interface de uso da API. A tabela 4 cumpre tal propósito.

Tabela 4: documentação da API

verbo	*endpoint*	função
GET	/mqtt/motor/{state}	Liga o motor quando *state* = *on* e desliga quando *state* = *off.*
GET	/mqtt/led/{level}/{state}	Liga ou desliga a iluminação do andar especificado, sendo o primeiro andar denotado pelo número 1, o segundo pelo 2

		e assim por diante. A variável *level* é um inteiro referente ao andar e *state* uma *string* que pode ser *on* ou *off*.
GET	/mqtt/start	Inicia o serviço de MQTT da API. Deve ser a primeira coisa a ser chamada ao iniciar a API ou os dados não serão persistidos.
GET	/data/getall	Retorna todos os dados persistidos no banco de dados
GET	/data/getrange	Retorna os dados entre a data do *start* e de *end*. *Start* e *End* são *DateTimeOffset*s passados via *query* da URL.
GET	/data/getcultivation	Retorna os dados referentes à cultura passada. O nome da cultura é passado pela *query* da URL através do parâmetro *cultivationName*.
POST	/data/add	Grava no banco de dados o *DataModel* passado pelo corpo da requisição no parâmetro data.
GET	/cultivation/getall	Retorna todas as culturas gravadas no banco de dados.
GET	/cultivation/get	Retorna uma cultura de acordo com o nome passado pelo parâmetro *name* na *query* da requisição.
POST	/cultivation/add	Grava no banco de dados o *CultivationModel* passado pelo corpo da requisição no parâmetro data.
PUT	/cultivation/update	Atualiza o registro no banco de dados da cultura de nome passado no parâmetro *name* pela *query* da URL com os dados passados pelo corpo da requisição
DELETE	/cultivation/delete	Exclui o registro de cultura com o nome passado pelo parâmetro *name* na *query* da requisição.

Fonte: Elaborado pelo Autor

7 CONCLUSÃO

A Horta é feita por vários componentes, tanto físicos quanto digitais. Para construí-la foi necessário um aglomerado de conhecimentos que ultrapassam a área da computação.

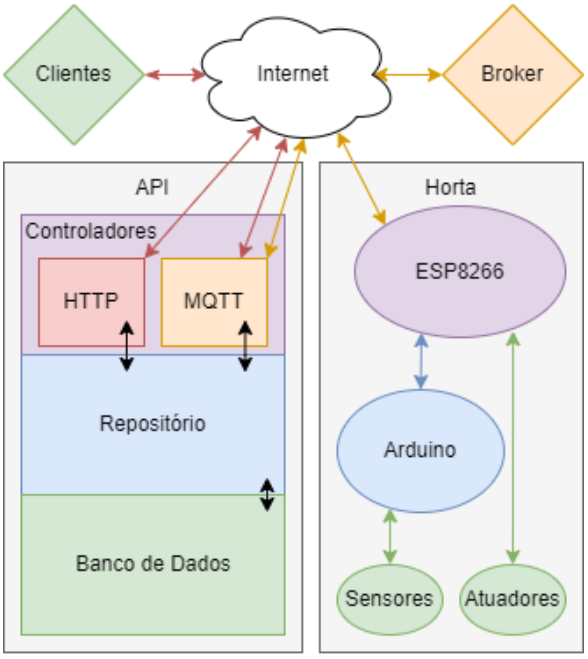
A começar pela botânica, necessária para entender sobre as plantas, qual a importância de determinados nutrientes em seu crescimento, como a hidroponia funciona e porque ela é uma boa solução para crescer alimentos num ambiente controlado.

Em seguida, a química foi importante para entender os parâmetros da água, como medi-los, como calibrar seus sensores e porque eles são importantes.

Dentro da computação, o conhecimento em microcontroladores para conseguir utilizar os sensores e os atuadores; em *Internet* das Coisas para conectar os microcontroladores à rede mundial e acessar os dados gerados pelos sensores de forma inteligente; em desenvolvimento *web* para programar uma API que persiste os dados gerados e em engenharia de *software* para que o desenvolvimento seja organizado.

Por fim, tem-se a arquitetura na figura 44. Pode-se dividi-la em duas partes: a Horta e a API. A Horta é o aparato físico, a estrutura onde se encontram as plantas, a água, o motor, os leds e os sensores. Ela se conecta à *internet* e utiliza o protocolo MQTT para publicar os dados gerados num *broker* na nuvem. A API é o *software* que dá acesso aos dados gerados e publicados pela Horta, persistindo-os em um banco de dados e expondo-os via HTTP.

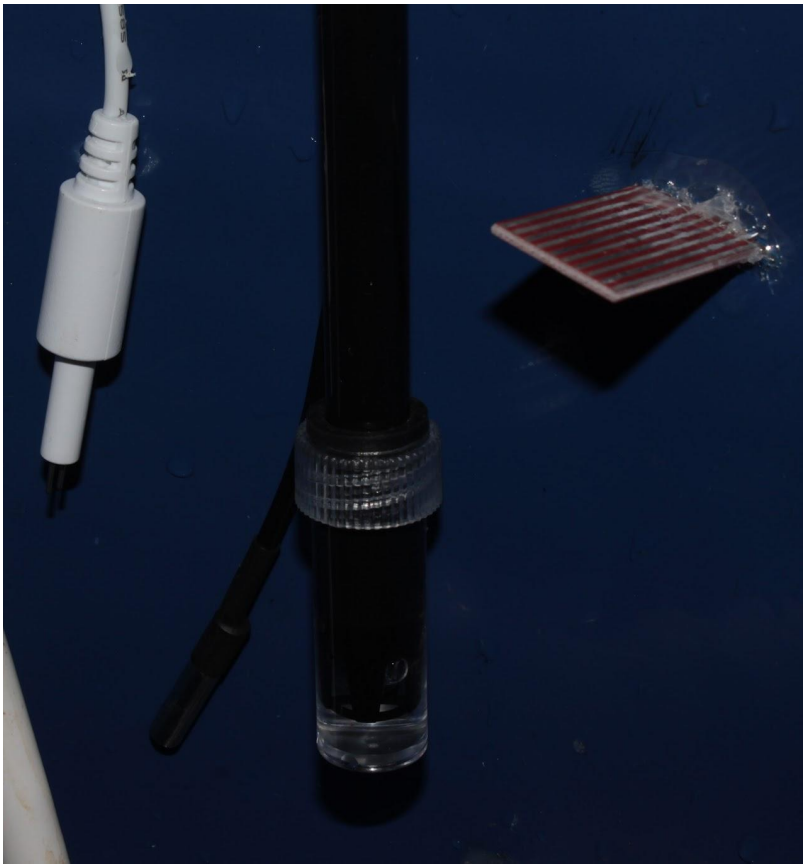
Figura 44: esquema da arquitetura final do projeto



Fonte: Elaborado pelo Autor

Todos os sensores foram postos na estrutura e o circuito foi montado dentro de uma caixa de plástico, como mostram as figuras 45, 46, 47 e 48.

Figura 45: Da esquerda para a direita: sonda de eletrocondutividade, sensor de temperatura, sonda de pH e sensor de nível mais alto

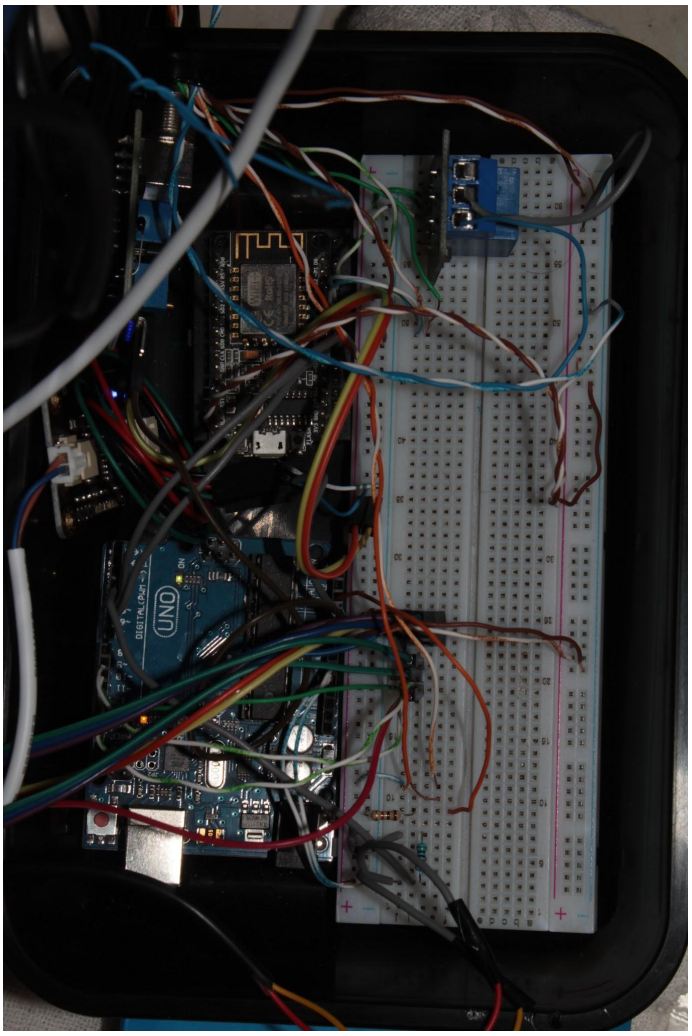


Fonte: Elaborado pelo Autor

Figura 46: Sensor de iluminação no último andar da estrutura

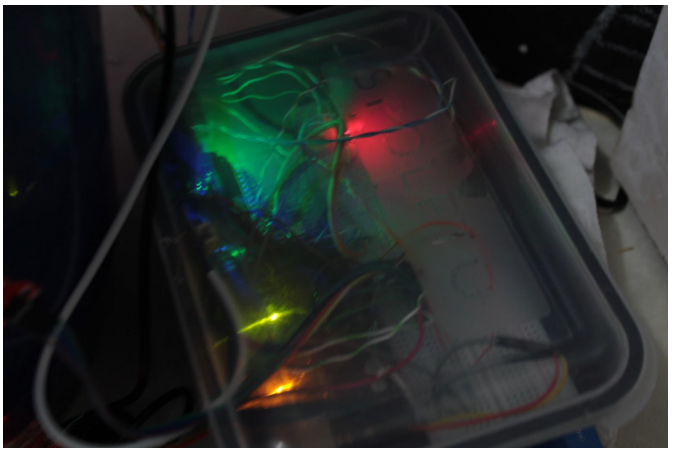
Fonte: Elaborado pelo Autor

Figura 47: Montagem do circuito dentro da caixa de plástico



Fonte: Elaborado pelo Autor

Figura 48: Circuito dentro da caixa fechada

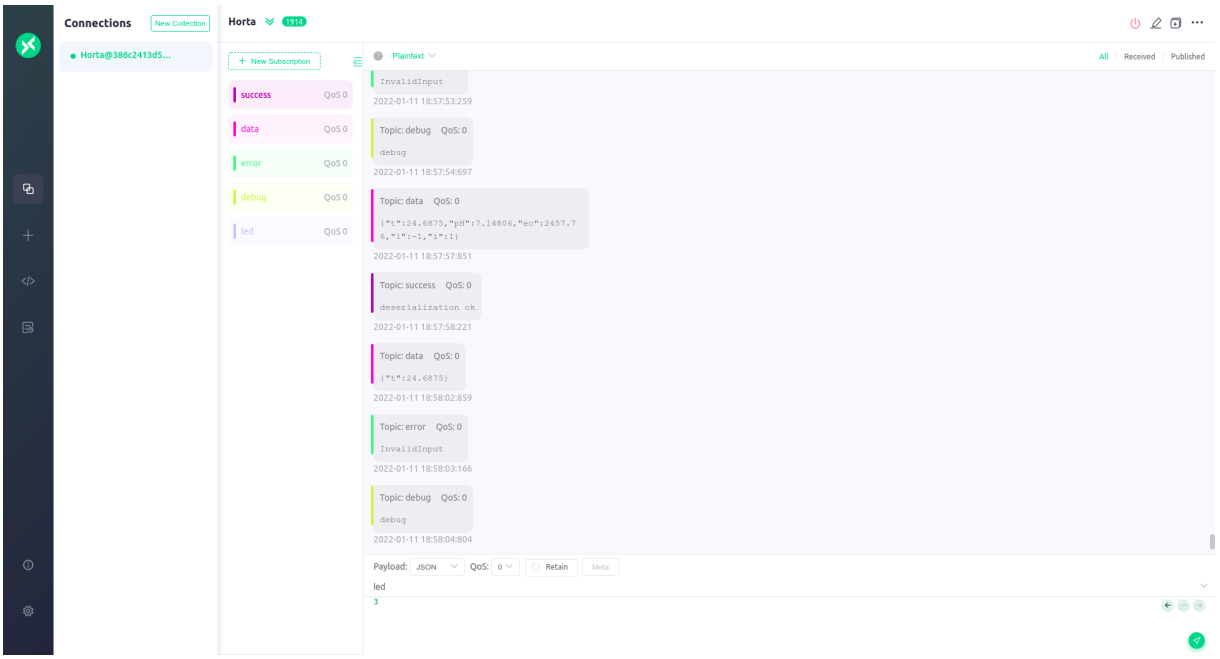


Fonte: Elaborado pelo Autor

A montagem da estrutura não é ótima, mas funciona: o circuito pode ser impresso numa placa e os componentes soldados, fios melhores podem ser usados e alguns detalhes podem ser melhorados.

Quando ligado à tomada, o Arduino lê os sensores, passa ao ESP e este publica no *broker*, na figura 49 são vistas as mensagens recebidas no *broker* (o *software* utilizado foi o MQTTX para Ubuntu).

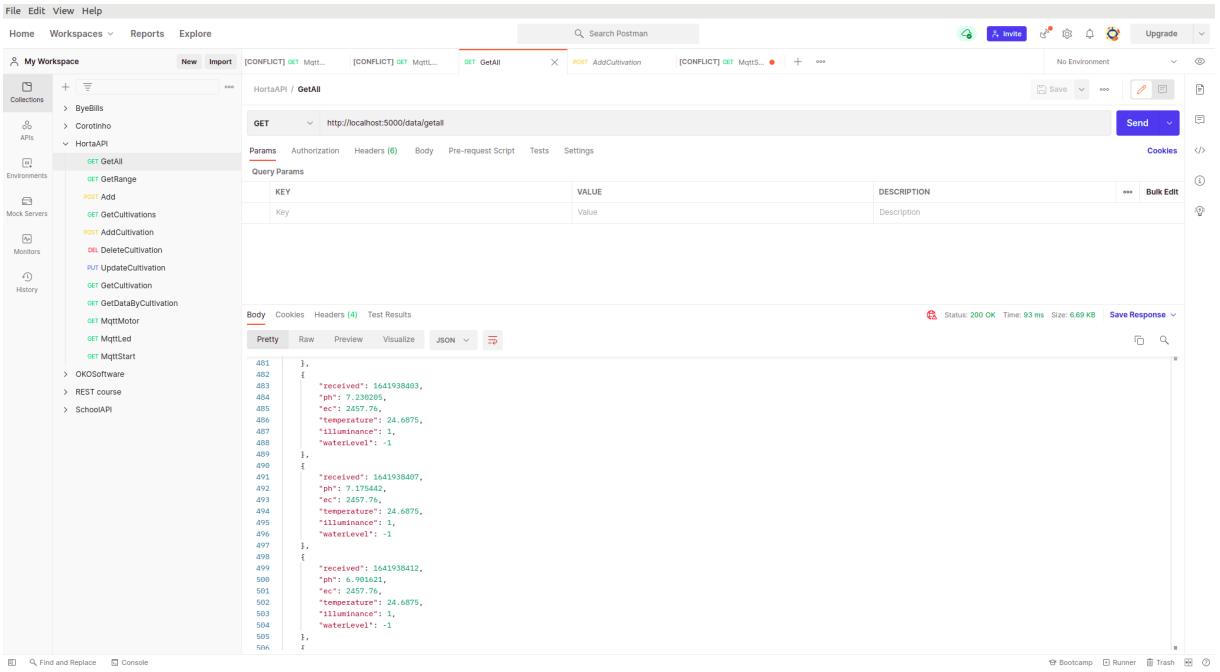
Figura 49: Mensagens recebidas da horta via MQTT no *software* MQTTX



Fonte: Elaborado pelo Autor

A API lê os dados, os armazena no banco de dados e os disponibiliza através de seus *endpoints*. Na figura 50 pode-se ver o resultado de um GET no *endpoint data/getall* feito *pelo software* Postman.

Figura 50: parte do resultado do GET no *endpoint data/getall* da API, no *software* Postman



Fonte: Elaborado pelo Autor

Apesar de funcionar, alguns pontos podem ser melhorados no projeto, como adição de mais sensores e atuadores, tais como temperatura e umidade fora da água, turbidez da água, ventiladores, câmeras e etc..

Para usar da API é um pouco complicado, então seria interessante a construção de um cliente *frontend* que abstraia os *endpoints* da API em botões e gráficos, dando ao usuário uma interface intuitiva. Quanto à forma de rodar da API, esta poderia ser conteinerizada, facilitando o seu uso num computador pessoal ou na nuvem.

Como a horta deve fazer parte de um ecossistema IoT de uma casa ou lugar, o próprio ESP poderia servir uma API mais simplificada, com menos funcionalidades, que persistisse os dados num banco de dados na nuvem, que é uma opção mais simples.

REFERÊNCIAS

ISTOÉ DINHEIRO. **Primeira fazenda vertical de São Paulo teve de se reinventar por pandemia**. Disponível em: <https://www.istoedinheiro.com.br/primeira-fazenda-vertical-de-sao-paulo-obrigada-a-se-reinventar-por-causa-da-pandemia/>. Acesso em: 9 jan. 2022.

CHIN, Y. S.; AUDAH, L. Vertical Farming Monitoring System Using the Internet of Things (IoT). 2017.

NEUMANN COUTO, M. **AGRICULTURA 4.0: PROTÓTIPO DE UM INTERNET OF THINGS (IoT) NA CULTURA DA LACTUCA SATIVA**. Trabalho de Conclusão de Curso—UTFPR: [s.n.].

TAN, E.-K. et al. **An IoT Platform for Urban Farming**. Disponível em: <https://ieeexplore.ieee.org/document/9163781>. Acesso em: 13 ago. 2020.

VIDAL MACHADO, C. et al. **SISTEMA PARA MONITORAMENTO DE PARÂMETROS FÍSICOS E QUÍMICOS DE CORPOS D'ÁGUA**. Artigo—Latin Science: [s.n.].

VIEIRA ROSA, B.; FRANCISCO DE OLIVEIRA, R. **HydroSys - Solução de Baixo Custo para a Visualização de Dados Aplicada à Sistemas Hidropônicos**. Trabalho de Conclusão de Curso—Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - IFSP: [s.n.].

DICIO. **Hidroponia**. Disponível em: <https://www.dicio.com.br/hidroponia/>. Acesso em: 25 jan. 2022.

BENTON JONES, J. **Complete guide for growing plants hydroponically**. Boca Raton, Florida: Crc Press, 2014.

EDUCALINGO. **ROCK WOOL - Definition and synonyms of rock wool in the English dictionary**. Disponível em: <https://educalingo.com/en/dic-en/rock-wool>. Acesso em: 11 jan. 2022.

AMAZON. **Rockwool Slabs: Patio, Lawn & Garden**. Disponível em: <https://www.amazon.com/Rockwool-Slabs-wide-long-tall/dp/B003B94D50>. Acesso em: 11 jan. 2022.

JARDINEIROS.NET. **Argila Expandida - 12 litros**. Disponível em: <https://www.jardineiros.net/argilaexpandida12l>. Acesso em: 11 jan. 2022.

OZINGA. **Pea Gravel | Landscaping Gravel | Drainage Gravel**. Disponível em: <https://ozinga.com/product/pea-gravel/>. Acesso em: 11 jan. 2022.

TRAY TECNOLOGIA. **Manta Calandrada de Fibra de Coco 1m x 1m x 4mm COQUIM | Loja Plantei**. Disponível em: <https://www.plantei.com.br/manta-calandrada-de-fibra-de-coco-1m-x-1m-x-4mm-coquim>. Acesso em: 11 jan. 2022.

THE HYDROPONICS GURU. **Types of Hydroponics Systems - The Hydroponics Guru**. Disponível em: <https://thehydroponicsguru.com/types-of-hydroponics-systems/>.

ALIEXPRESS. **Bomba d'água, 800l/h, 5m, solar, 12v, submersível, circulação do motor, 19w**. Disponível em: <https://pt.aliexpress.com/i/32840967200.html>. Acesso em: 11 jan. 2022.

BUYYA, R.; VAHID DASTJERDI, A. **Internet of Things : principles and paradigms**. Amsterdam ; Boston ; Heidelberg: Morgan Kaufmann, 2016.

HIVEMQ. **MQTT & MQTT 5 Essentials**. [s.l.] HiveMQ, [s.d.].

RAMOS DA FONSECA, F. **Sensores**. [s.l: s.n.].

DFROBOT. **Gravity Analog TDS Sensor Meter for Arduino**. Disponível em: <https://wiki.dfrobot.com/Gravity__Analog_TDS_Sensor___Meter_For_Arduino_SKU__SEN0244>. Acesso em: 2 nov. 2021.

HK SHAN HAI GROUP LIMITED. **DS18B20 Waterproof Temperature Sensor Cable Product Description**. [s.l: s.n.]. Disponível em:

<https://www.terraelectronica.ru/pdf/show?pdf_file=%2Fz%2FDatasheet%2F1%2F1420644897.pdf>. Acesso em: 25 jan. 2022.

AUTOCORE ROBÓTICA. **Sonda Eletrodo Sensor de PH com Módulo PH4502C**. Disponível em: <https://www.autocorerobotica.com.br/sonda-eletrodo-sensor-de-ph-com-modulo-PH4502C>. Acesso em: 11 jan. 2022.

BANGGOOD.COM. **TDS analógico Sensor Kit de placa de módulo de condutividade Sensor em água**. Disponível em: <https://br.banggood.com/Analog-TDS-Sensor-Water-Conductivity-Sensor-Module-Board-Kit-p-1681060.html>. Acesso em: 11 jan. 2022.

BANGGOOD.COM. **2 Fios Cabos Temperatura Digital Sensor Sonda DS18B20 Termômetro Térmica**. Disponível em: <https://br.banggood.com/2-Wires-Cables-Digital-Temperature-Sensor-Probe-DS18B20-Thermometer-Thermal-p-986819.html>. Acesso em: 11 jan. 2022.

BANGGOOD.COM. **20 pcs Resistor Dependente de Luz LDR 5 MM Elemento Fotoresistor Fotoelétrico Interruptor Foto Detector 5516**. Disponível em: <https://br.banggood.com/20pcs-Light-Dependent-Resistor-LDR-5MM-Photoresistor-Photoelectric-Switch-Element-Photo-Detector-5516-p-1693475.html>. Acesso em: 11 jan. 2022.

BANGGOOD.COM. **3pcs DC 3V-5V 20mA Nível de água da chuva Sensor Altura de profundidade da superfície líquida de detecção de módulo para Geekcreit para Arduino - produtos que funcionam com placas oficiais Arduino**. Disponível em: <https://br.banggood.com/3pcs-DC-3V-5V-20mA-Rain-Water-Level-Sensor-Module-Detection-Liquid-Surface-Depth-Height-For-Geekcreit-for-Arduino-products-that-work-with-official-Arduino-boards-p-1633679.html>. Acesso em: 11 jan. 2022.

K. SPRINGER, E. **pH Measurement Guide**. [s.l.] Hamilton, 2014.

CHEUNG, H. **Measure pH with a low-cost Arduino pH sensor board**. Disponível em: <https://www.e-tinkers.com/2019/11/measure-ph-with-a-low-cost-arduino-ph-sensor-board/>. Acesso em: 23 nov. 2021.

CHEUNG, H. **A review on Seeed Studio pH and EC Sensor Kits Part 2**. Disponível em: <https://www.e-tinkers.com/2020/07/a-review-on-seeed-studio-ph-and-ec-sensor-kits-part-2/>. Acesso em: 24 nov. 2021.

TARNOWSKI, K. DOS S. **Indicador ácido-base de repolho roxo**. Disponível em: <https://quimicaempratica.com/2017/07/06/indicador-acido-base-de-repolho-roxo/>.

INDIAMART.COM. **Ph Test Strips Indicator 1 14 Paper Litmus Paper**. Disponível em: <https://www.indiamart.com/proddetail/ph-test-strips-indicator-1-14-paper-litmus-paper-18977725891.html>. Acesso em: 11 jan. 2022.

FORUM ARDUINO. **pH sensor detection circuit design**. Disponível em: <https://forum.arduino.cc/t/ph-sensor-detection-circuit-design/478631>. Acesso em: 11 jan. 2022.

XYLEM ANALYTICS GERMANY SALES GMBH & CO. KG. **Conductivity Handbook**. [s.l: s.n.].

HMDIGITAL. **The Relationship Between TDS and Electrical Conductivity**. Disponível em: <http://hmdigital.com/knowledge_base/the-relationship-between-tds-and-electrical-conductivity/>. Acesso em: 25 out. 2021.

REDHAT. **O que é API?** Disponível em: <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>.

RICK-ANDERSON. **Injeção de dependência no ASP.NET Core**. Disponível em: <https://docs.microsoft.com/pt-br/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0>. Acesso em: 18 dez. 2021.

XUKYO. **Luminosity measurement with a photoresistor • AranaCorp**. Disponível em: <https://www.aranacorp.com/en/luminosity-measurement-with-a-photoresistor/>. Acesso em: 24 nov. 2021.

APÊNDICE 1: Código Arduino

```
/*********************************Includes*************************************/
//Arduino Library
#include <Arduino.h>
//Serial communication library
#include <SoftwareSerial.h>
// library to encapsulate data into JSON format
#include <ArduinoJson.h>
//temperature sensor libraries
#include <OneWire.h>
#include <DallasTemperature.h>

#include <GravityTDS.h>

/*********************************Defines**************************************/
#define PH_PORT A0
#define EC_PORT A1
#define TEMPERATURE_PORT 3
#define ILLUMINANCE_PORT A2
#define LEVEL_PORT_1 9
#define LEVEL_PORT_2 10
#define LEVEL_PORT_3 11

/*****************************Global Variables*********************************/
// Using pin 5 as Rx and pin 6 as Tx
SoftwareSerial nodemcu(5,6);

//temperature sensor variables
OneWire oneWire(TEMPERATURE_PORT);
DallasTemperature sensors(&oneWire);

//GravityTDS
GravityTDS gravityTDS;

//serialized data
char serializedData[500];

/*********************************Setup****************************************/
void setup() {
  Serial.begin(9600);
```

```
  // setup of Serial communication
  nodemcu.begin(9600);
  delay(1000);

  //temperature
  oneWire.write(0x44, 1); //set parasiteMode
  sensors.begin(); //start temperature

  //pH
  pinMode(PH_PORT, INPUT);

  //ec
  gravityTDS.setPin(EC_PORT);
  gravityTDS.setAref(5.0);
  gravityTDS.setAdcRange(1024);
  gravityTDS.begin();

  //lux port
  pinMode(ILLUMINANCE_PORT, INPUT);

  //level ports
  pinMode(LEVEL_PORT_1, INPUT);
  pinMode(LEVEL_PORT_2, INPUT);
  pinMode(LEVEL_PORT_3, INPUT);
}
/**************************Reading Functions*********************************/
//Read pH value
double getPh(){
  //reading analog port
  int adcValue;
  for(int i = 0; i < 20; i++){
    adcValue += analogRead(PH_PORT);
    delay(100);
  }
  adcValue /= 20;
  //converting byte to voltage
  double phVoltage = (float)adcValue * 5.0 / 1024.0;

  //calculate pH -> phMax - (VphMax - phVoltage) * m
  return 6.86 - (2.40 - phVoltage) * (-5.6078431373) ;
}
```

```
float getEc(float temperature){
    gravityTDS.setTemperature(temperature);    // set the temperature and execute
temperature compensation
  gravityTDS.update();  //sample and calculate
  float tdsValue = gravityTDS.getEcValue();
  return tdsValue;
}

float getTemperature(){
  //request temperature from sensor
  sensors.requestTemperatures();
  //convert to Celsius
  return sensors.getTempCByIndex(0);
}

float getIlluminance(){
  float mean = 0;
  for(int i = 0; i < 20; i++){
    mean += analogRead(ILLUMINANCE_PORT);
  }
  return map(mean/20, 0, 1024, 1, 100);
}

int getLevel(){
  //reading digital ports
  int l1 = digitalRead(LEVEL_PORT_1);
  int l2 = digitalRead(LEVEL_PORT_2);
  int l3 = digitalRead(LEVEL_PORT_3);

  //checking level
  if(l1 && l2 && l3) return 100;
  else if(l1 && l2 && !l3) return 50;
  else if(l1 && !l2 && !l3) return 0;
  return -1;
}
/*********************************Loop*************************************/
void loop() {
  //creating the JSON document
  StaticJsonDocument<500> doc;
  // reading each sensor and storing the values into the JSON document
  doc["t"] = getTemperature();
  doc["pH"] = getPh();
```

```
  doc["ec"] = getEc(doc["t"]);
  doc["l"] = getLevel();
  doc["i"] = getIlluminance();

  //serialize JSON with read data
  //serializeJsonPretty(doc, Serial);

  //send the JSON document to the NodeMCU
  serializeJson(doc, nodemcu);

  delay(500);
}
/*******************************************************************************/
```

APÊNDICE 2: Código ESP8266

```
/************************INCLUDES************************/
#include <Arduino.h>
#include <LittleFS.h>
#include <ESP8266WiFi.h>
#include <ArduinoJson.h>
#include <MQTT.h>
#include <string.h>
#include <SoftwareSerial.h>
/********************************************************/

/*********************GLOBAL VARIABLES*******************/
//config files
StaticJsonDocument<400> doc;
JsonObject obj;

//Wifi configuration
char wifiSsid[20];
char wifiPassword[30];

//MQTT broker and credentials configuration
char brokerUrl[256];
char brokerUsername[20];
char brokerPassword[20];

//PubSub client
WiFiClientSecure net;
MQTTClient client;

//Serial communication, (Rx,Tx)
SoftwareSerial s(D6, D5);
unsigned long lastMsg = 0, lastMsg10 = 0;
StaticJsonDocument<500> msg;
char data[500];

/********************************************************/

/************************FUNCTIONS***********************/

// load the configuration file data
```

```
bool loadConfig()
{
  //saveConfig();
  //opens the configuration file in read mode
  File configFile = LittleFS.open("/secrets.json", "r");

  //check the existence of the configuration file
  if (!configFile)
  {
    Serial.println("failed to open configuration file");
    return false;
  }
  //allocate a buffer to store the contents of the file and
  //after use it with ArduinoJson deserialize function
  size_t size = configFile.size();
  std::unique_ptr<char[]> buf(new char[size]);

  //Read the file in Json format
  configFile.readBytes(buf.get(), size);

  auto error = deserializeJson(doc, buf.get());
  if (error)
  {
    Serial.println("Failed to parse configuration file");
    return false;
  }

  obj = doc.as<JsonObject>();

  //assign the results to the global variables

  strcpy(brokerUsername, obj["broker_username"]);
  strcpy(brokerPassword, obj["broker_password"]);

  strcpy(wifiSsid, obj["wifi_ssid"]);
  strcpy(wifiPassword, obj["wifi_password"]);

  strcpy(brokerUrl, obj["broker_url"]);

  return true;
}
```

```
// setup the WiFi and connect to it
void setupWiFi()
{
  Serial.print("Connecting to ");
  Serial.println(wifiSsid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(wifiSsid, wifiPassword);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

//Reconnect to broker
void reconnect()
{
  // Loop until we're reconnected
  while (!client.connected())
  {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "Horta-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    net.setInsecure();

    if (client.connect(clientId.c_str(), brokerUsername, brokerPassword))
    {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("outTopic", "hello world");
      // ... and resubscribe
```

```
      client.subscribe("led");
      client.subscribe("motor");
    }
    else
    {

      Serial.print("failed, rc=");
      Serial.print(client.lastError());

      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void messageReceived(String &topic, String &payload)
{
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("]: ");
  Serial.println(payload);

  //control illumination
  if(topic.compareTo("led") == 0){
    Serial.print("led: ");
    //First floor control
    if((char)payload[0]=='0'){
      digitalWrite(D4, LOW);
      Serial.println("1, off ");
    }
    if((char)payload[0]=='1'){
      digitalWrite(D4, HIGH);
      Serial.println("1, on ");
    }

    //second floor control
    if((char)payload[0]=='2'){
      digitalWrite(D3, LOW);
      Serial.println("2, off ");
    }
    if((char)payload[0]=='3'){
```

```
      digitalWrite(D3, HIGH);
      Serial.println("2, on ");
    }
  }

  //control motor
  if(topic.compareTo("motor")==0){
    Serial.print("motor: ");
    if((char)payload[0]=='0'){
      digitalWrite(D2, LOW);
      Serial.println("off ");
    }
    if((char)payload[0]=='1'){
      digitalWrite(D2, HIGH);
      Serial.println("on ");
    }

  }

}

 void deserializeData(){
  if (s.available())
  {
    DeserializationError err = deserializeJson(msg, s);
    serializeJson(msg, data);
    client.publish("data", data);
    if (err == DeserializationError::Ok)
    {
      client.publish("success", "deserialization ok");
    }
    else
    {
      client.publish("error", err.c_str());

      // Flush all bytes in the "link" serial port buffer
      while (s.available() > 0)
        s.read();
    }
  }
}
```

```
/*********************************************************/

/*************************MAIN*************************/
void setup()
{
  //start serial monitor
  Serial.begin(115200);

  //start serial communication to Arduino board;
  s.begin(9600);

  //mount the file system to access the configuration file
  Serial.println("Mounting file system...");
  if (!LittleFS.begin())
  {
    Serial.println("failed to mount file system");
  }

  //load configuration file
  if (!loadConfig())
  {
    Serial.println("Failed to load configuration file");
  }
  else
  {
    Serial.println("Configuration loaded");
  }

  //connect to WiFi
  setupWiFi();

  //set the pinouts
  pinMode(D4, OUTPUT); //first floor illumination
  pinMode(D3, OUTPUT); //second floor illumination
  pinMode(D2, OUTPUT); //motor

  //connect to MQTT broker
  client.begin(brokerUrl, 8883, net);
  //set callback funtion to when it receives a message
  client.onMessage(messageReceived);

  delay(500);
```

```
}

void loop()
{

  if (!client.connected())
  {
    reconnect();
  }
  client.loop();

  unsigned long now = millis();

  if (now - lastMsg > 5000)
  {
    lastMsg = now;
    deserializeData();
  }


  if (now - lastMsg10 > 10000)
  {
    lastMsg10 = now;

    //Serial.println("Publish message: debug");

    client.publish("debug", "debug");
  }
}

/*******************************************************/
```

APÊNDICE 3: Calibração pH

```
//Arduino Library
#include <Arduino.h>
//temperature sensor libraries
#include <OneWire.h>
#include <DallasTemperature.h>

#define PH_PORT A0
#define TEMPERATURE_PORT 3

//temperature sensor variables
OneWire oneWire(TEMPERATURE_PORT);
DallasTemperature sensors(&oneWire);
float temperature;

//pH variables
int analogValue;
float VpH;

void setup() {
  Serial.begin(9600);
  //temperature
  oneWire.write(0x44, 1); //set parasiteMode
  sensors.begin(); //start temperature

  //pH
  pinMode(PH_PORT, INPUT);
}

float getTemperature(){
  //request temperature from sensor
  sensors.requestTemperatures();
  //convert to Celsius
  return sensors.getTempCByIndex(0);
}

void loop(){
  temperature = getTemperature();
  Serial.print(temperature);
  Serial.print(",");
```

```
analogValue = analogRead(PH_PORT);
VpH = (float)analogValue * 5.0 / 1024.0;
Serial.print(VpH);
Serial.print("\n");

delay(500);
}
```

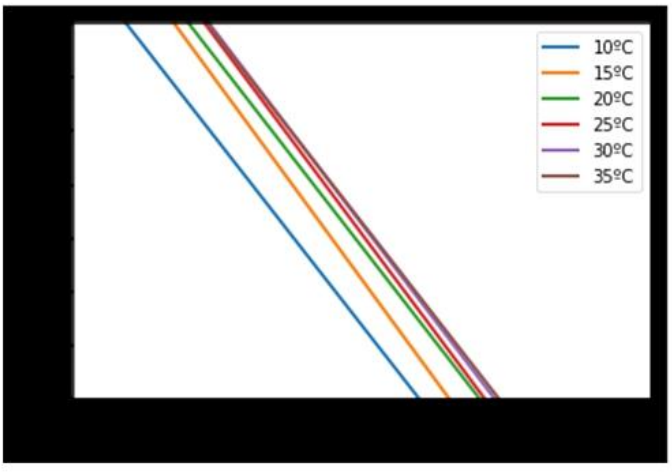
APÊNDICE 4: Análise dos dados de calibração

```
import csv
import matplotlib.pyplot as plt
import numpy as np


ph4_file = open('ph4.csv')
ph4 = csv.reader(ph4_file)

ph9_file = open('ph9.csv')
ph9 = csv.reader(ph9_file)


ph4_array = [[],[],[],[],[],[]]
ph9_array = [[],[],[],[],[],[]]


for ph in ph4:
  try:
    temperature = float(ph[0])
    if(temperature >= 9.5 and temperature<=10.5):
      ph4_array[0].append(float(ph[1]))
    elif(temperature >= 14.5 and temperature<=15.5):
      ph4_array[1].append(float(ph[1]))
    elif(temperature >= 19.5 and temperature<=20.5):
      ph4_array[2].append(float(ph[1]))
    elif(temperature >= 24.5 and temperature<=25.5):
      ph4_array[3].append(float(ph[1]))
    elif(temperature >= 29.5 and temperature<=30.5):
      ph4_array[4].append(float(ph[1]))
    elif(temperature >= 34.5 and temperature<=35.5):
      ph4_array[5].append(float(ph[1]))
  except:
    continue


for ph in ph9:
  try:
    temperature = float(ph[0])
    if(temperature >= 9.5 and temperature<=10.5):
      ph9_array[0].append(float(ph[1]))
    elif(temperature >= 14.5 and temperature<=15.5):
      ph9_array[1].append(float(ph[1]))
    elif(temperature >= 19.5 and temperature<=20.5):
      ph9_array[2].append(float(ph[1]))
    elif(temperature >= 24.5 and temperature<=25.5):
      ph9_array[3].append(float(ph[1]))
    elif(temperature >= 29.5 and temperature<=30.5):
      ph9_array[4].append(float(ph[1]))
    elif(temperature >= 34.5 and temperature<=35.5):
      ph9_array[5].append(float(ph[1]))
  except:
    continue


v4_means = []
```

```
v9_means = []
temperatures = [10,15,20,25,30,35]
pH4_temp = [4.0, 4.0, 4.0, 4.0, 4.01, 4.02]
pH9_temp = [9.33, 9.28, 9.23, 9.18, 9.14, 9.10]


for phs in ph4_array:
  v4_means.append(sum(phs)/len(phs))

for phs in ph9_array:
  v9_means.append(sum(phs)/len(phs))


ph_m = []
for ph4, ph9, v4, v9 in zip(pH4_temp, pH9_temp, v4_means, v9_means):
  ph_m.append((ph9 - ph4)/(v9 - v4))


print(ph_m)
```

```
[-5.53101342253932, -5.893592577257041, -5.572879281359909, -5.79128768506710
```

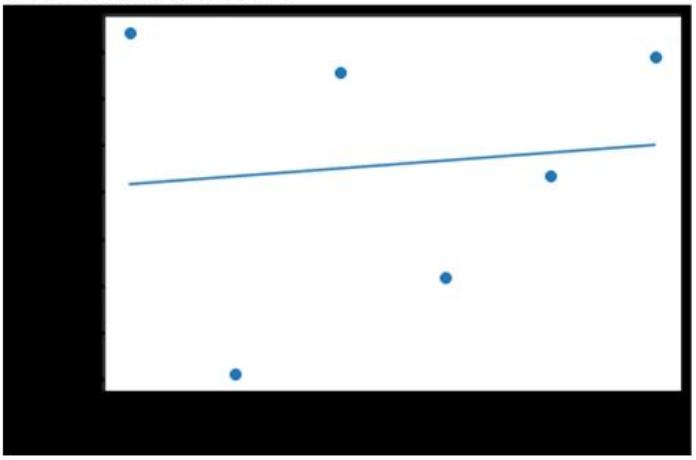
```
po = np.linspace(0,5,500)
fig, ax = plt.subplots()
for m, phmax, vmax, temp in zip(ph_m, pH9_temp, v9_means, temperatures):
  ph = phmax - (vmax - po) * m
  ax.plot(po, ph,  linewidth=2.0, label=(str(temp)+ "ºC"))

ax.set(xlim=(0, 5), xlabel="Voltage",
       ylim=(0, 14), ylabel = "pH")
plt.legend()
plt.show()
```




```
from sklearn.linear_model import LinearRegression
from scipy.optimize import curve_fit


fig3, cx = plt.subplots()
```

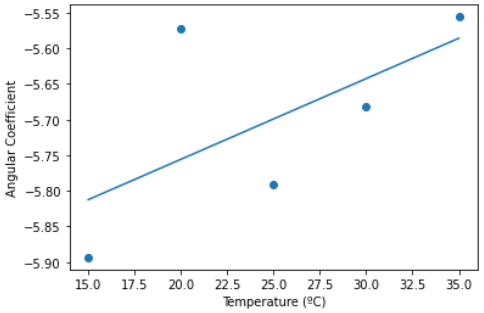
```
temperatures_reshaped = np.array(temperatures).reshape((-1,1))
model = LinearRegression().fit(temperatures_reshaped, np.array(ph_m))
temp = np.linspace(10, 35, 250)
print(model.coef_)
print(model.intercept_)
print(model.score(temperatures_reshaped, np.array(ph_m)))
v = temp * model.coef_ + model.intercept_
cx.scatter(np.array(temperatures), np.array(ph_m))
cx.plot(temp, v)
cx.set(xlabel="Temperature (ºC)",
      ylabel = "Angular Coefficient")
plt.show()
```

```
[0.00167295]
-5.70873328475953
0.011479368434183068
```



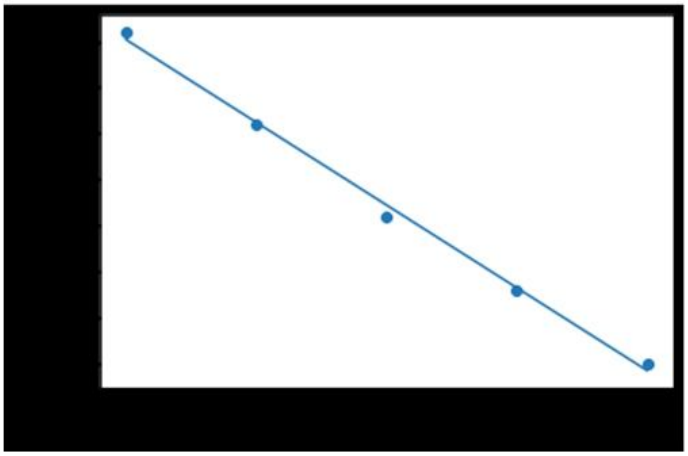
```
fig3, cx = plt.subplots()
temperatures = temperatures[1:6]
ph_m = ph_m[1:6]
temperatures_reshaped = np.array(temperatures).reshape((-1,1))
model = LinearRegression().fit(temperatures_reshaped, np.array(ph_m))
temp = np.linspace(15, 35, 250)
print(model.coef_)
print(model.intercept_)
print(model.score(temperatures_reshaped, np.array(ph_m)))
v = temp * model.coef_ + model.intercept_
cx.scatter(np.array(temperatures), np.array(ph_m))
cx.plot(temp, v)
cx.set(xlabel="Temperature (ºC)",
      ylabel = "Angular Coefficient")
plt.show()
```

```
[0.01133237]
-5.982416950485635
0.38626031841290054
```




```
temperatures = [15,20,25,30,35]
pH4_temp = [4.0, 4.0, 4.0, 4.01, 4.02]
pH9_temp = [9.28, 9.23, 9.18, 9.14, 9.10]
v9_means = v9_means[1:6]

fig3, cx = plt.subplots()
temperatures_reshaped = np.array(temperatures).reshape((-1,1))
model = LinearRegression().fit(temperatures_reshaped, np.array(pH9_temp))
temp = np.linspace(15, 35, 250)
print(model.coef_)
print(model.intercept_)
print(model.score(temperatures_reshaped, np.array(pH9_temp)))
v = temp * model.coef_ + model.intercept_
cx.scatter(np.array(temperatures), np.array(pH9_temp))
cx.plot(temp, v)
cx.set(xlabel="Temperature (ºC)",
       ylabel = "pH 9 values")
plt.show()
```

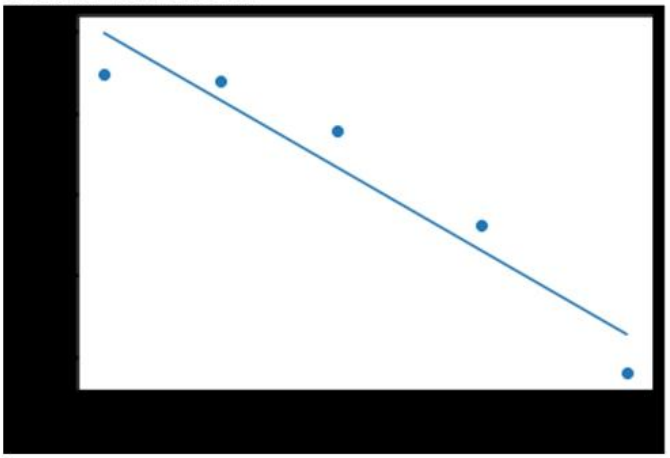
```
[-0.009]
9.411
0.9965551181102366
```



```
fig3, cx = plt.subplots()
pH9_temp_reshaped = np.array(pH9_temp).reshape((-1,1))
model = LinearRegression().fit(pH9_temp_reshaped, np.array(v9_means))
temp = np.linspace(15, 35, 250)
print(model.coef_)
print(model.intercept_)
print(model.score(pH9_temp_reshaped, np.array(v9_means)))
v = pH9_temp * model.coef_ + model.intercept_
```

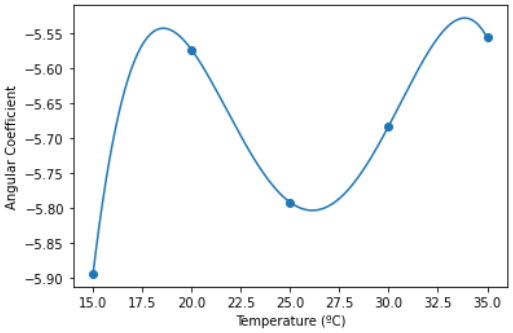
```
cx.scatter(np.array(pH9_temp), np.array(v9_means))
cx.plot(pH9_temp, v)
cx.set(xlabel="pH 9",
       ylabel = "pH 9 voltage")
plt.show()
```

```
[-2.05946685]
20.839077484482623
0.9127447524430432
```



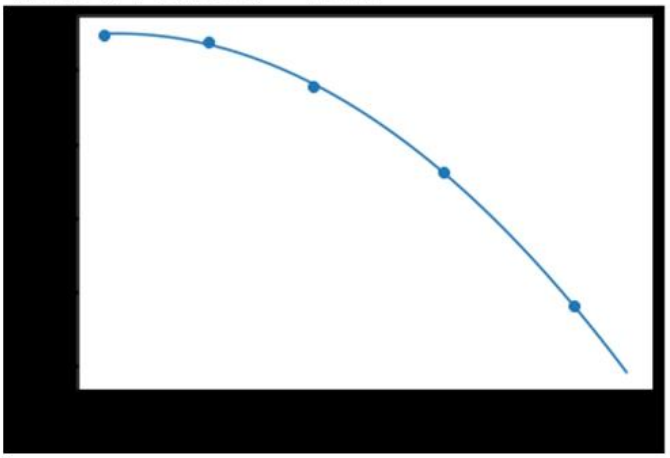
```
fig3, cx = plt.subplots()

temperatures_reshaped = np.array(temperatures).reshape((-1,1))

z = np.polyfit(temperatures, ph_m, 4)
f = np.poly1d(z)
temp = np.linspace(15, 35, 250)
v = f(temp)

print(z)
print(f)
print(f(25))

cx.scatter(np.array(temperatures), np.array(ph_m))
cx.plot(temp, v)
cx.set(xlabel="Temperature (ºC)",
       ylabel = "Angular Coefficient")
plt.show()
```

```
[-7.84680734e-05  8.21773450e-03 -3.13561426e-01  5.15387726e+00
 -3.64128384e+01]
           4           3          2
-7.847e-05 x + 0.008218 x - 0.3136 x + 5.154 x - 36.41
-5.791287685067189
```




```
fig3, cx = plt.subplots()

pH9_temp_reshaped = np.array(pH9_temp).reshape((-1,1))

z = np.polyfit(pH9_temp, v9_means, 2)
f = np.poly1d(z)
x = np.linspace(9.1, 9.3, 100)
v = f(x)

cx.scatter(np.array(pH9_temp), np.array(v9_means))
cx.plot(x, v)

print(z)
print(f)

cx.set(xlabel="pH 9",
       ylabel = "pH 9 voltage")
plt.show()
```

```
[ -12.0180885   218.84459199 -994.21925907]
          2
-12.02 x + 218.8 x - 994.2
```



```
print(v9_means)
```

```
[1.680172413793102, 1.8611480362537804, 1.9772727272727273, 2.0385, 2.0469819
```

APÊNDICE 5: Script SQL para a criação do Banco de Dados

```
-- Create a new database called 'hortaiot'
-- Connect to the 'master' database to run this snippet
USE master
GO
-- Create the new database if it does not exist already
IF NOT EXISTS (
   SELECT name
      FROM sys.databases
      WHERE name = N'hortaiot'
)
CREATE DATABASE hortaiot
GO

\c hortaiot;

-- Create a new table called 'measurement_data' in schema 'hortaiot'

-- Create the table in the specified schema
CREATE TABLE measurement_data
(
   received BIGINT NOT NULL PRIMARY KEY, -- primary key column
   ph double precision,
   ec double precision,
   temperature double precision,
   illuminance integer,
   water_level integer
);

CREATE TABLE plant_cultivation
(
   id serial PRIMARY KEY, -- primary key column
   name VARCHAR(128) NOT NULL UNIQUE,
   description VARCHAR(512),
   start bigint,
   finish bigint
);
```